Ghazi *et al. EURASIP Journal on Information Security* (2016) 2016:16
DOI 10.1186/s13635-016-0040-5

EURASIP Journal on Information Security

RESEARCH Open Access

DB-SECaaS: a cloud-based protection system for document-oriented NoSQL databases

Yumna Ghazi[1], Rahat Masood[2*], Abid Rauf[1], Muhammad Awais Shibli[3] and Osman Hassan[1]

Abstract

The trend of cloud databases is leaning towards Not Only SQL (NoSQL) databases as they provide better support for scalable storage and quick retrieval of exponentially voluminous data. One of the more prominent types of NoSQL databases is document-based storage, which is being increasingly used in the dynamic cloud paradigm. However, there are inherent security issues in cloud, including remote data residency along with the non-existent control of owners over their own data. In addition to that, the inherent security features of most document-based NoSQL databases lack granular access control and robust confidentiality mechanisms. There is also a distinct lack of a comprehensive solution that effectively caters to all the security requirements of a document-oriented database in cloud. In order to overcome these issues, we propose a database security-as-a-service (DB-SECaaS) system over document-oriented database hosted in cloud, which provides authentication, fine-grained authorization, and encryption of the database objects, while ensuring that access to the data is granted only to authorized users on a need-to-know basis. The paper shows that the DB-SECaaS system strongly enhances the security of document-oriented databases on cloud, and it is thus expected to facilitate the industry to reap the benefits of NoSQL without worrying over security issues. In order to certify the abovementioned security enhancements, provided by DB-SECaaS, the paper also provides a formal analysis of DB-SECaaS using the Scyther model checker. As a proof of concept, the core functionalities of the protocol, i.e., authorization, authentication, and encryption, are formally modeled in Scyther to formally verify that the proposed framework mitigates privacy and security concerns.

Keywords: Cloud database, Document-oriented NoSQL, Security-as-a-service, eXtensible access control markup language (XACML), Database security

1 Introduction

Not Only SQL (NoSQL) databases provide a comprehensive solution for a wide range of database issues, characterized by basically available, soft state, eventually consistent (BASE) in lieu of relational models of atomicity, consistency, isolation, and durability (ACID) [1]. Compared to relational database management system (RDBMS), NoSQL is found to have an accelerated rate of data processing. It also provides a relatively inexpensive way for enterprises to efficiently manage large volumes of data [2]. Among the many types of NoSQL databases, an important and widely used type is document-oriented storage, wherein an object model is stored as a document. Being schema-free, it allows great flexibility for updating data, without the need for any significant restructuring. The complexity of the documents varies based on the user's preference. Also, the independence of documents from one another improves performance and decreases concurrency side effects [3]. CouchDB [4], RavenDB [5], and MongoDB [6] are some of the most popular document-based NoSQL databases.

With the evolution of distributed computing into the popular cloud paradigm, the aforementioned benefits of document storage have come to the forefront, considering its support for distributed storage and scalability at will. Cloud exploits this advantage by providing a more cost-efficient, outsourced database management solution that allows enterprises to easily store, manage, and pro-

*Correspondence: rahat.masood@student.unsw.edu.au
[2]University of New South Wales (UNSW), NSW, Sydney, Australia
Full list of author information is available at the end of the article

cess large volumes of data. Recently, many solutions, such as Mongolab [7], RavenHQ [8], and Cloudant [9] have been developed that commercially provide document-oriented storage in cloud. However, while considerable efforts are being made to enhance the performance and provide flexible scalability options, the more pressing issues of security in document-oriented databases seems to be largely absent from the research landscape (see Section 5). At best, most document databases provide role-based access control, which is coarse-grained, where users having the permission to read from or write to a database can apply the operation on the entire database. Even though this model of access control ensures authorized access, it leaves the database vulnerable to attacks from malicious insiders, which, according to a report by McAfee [10], poses a considerable threat. In addition, encryption of data, both in transit and at rest, is provided, using SSL for the former and third-party services for the latter (see Table 4). Moreover, there are no holistic solutions specifically tailored for document-oriented databases that fulfill all the security requirements. The recent breaches in cloud databases are a testament to their vulnerabilities [11–13].

To overcome these limitations, we have developed a comprehensive database security-as-a-service (DB-SECaaS) system, on top of a document-oriented database in cloud, which would provide strong authentication, fine-grained authorization, and data encryption to ensure maximum security for the document-oriented database layer lying underneath. The system has a service-oriented architecture, which allows deploying individual components for performing specific functions as separate services. This choice of architecture introduces abstraction to the services; therefore, making it easy for different document-oriented databases to seamlessly integrate the DB-SECaaS without having to customize and embed it into their source code. The "as-a-service" concept provides an economic advantage for the businesses, since they would be able to outsource the security of their document-oriented database without having to buy the requisite hardware or hire experts to operate them. Therefore, our DB-SECaaS system would be equally effective for any document-oriented database because of its utilization regardless of the individual underlying architectures.

Our system provides strong authentication using the Federal Information Processing Standards 196 (FIPS 196) [14] challenge-response protocol, which is designed to mitigate replay attacks. We also add an additional layer of security using security assertion markup language (SAML) authentication assertions to validate the identity of the user and the various services participating in the system [15]. We offer a fine-grained authorization at the granularity of attribute and document level of the database, based on eXtensible access control markup language (XACML) 3.0 [16], which is an access control policy creation language. The usage of SAML and XACML enhances the interoperability of the system, since they are both well-known standards. The fine-grained XACML policies restrict the access level of the users to ensure that a user can only perform certain operations on a certain data, if and only if she is allowed to do so. In addition, we also offer encryption to strengthen the protection detail for the stored data.

The paper also presents a formal analysis and verification on handover procedures of DB-SECaaS using the Scyther model checker [17] to extract and debug the main security flaws and threats that might exist in such procedures. Scyther allows the analysis of many potential attacks, such as man-in-the-middle attack, replay attack, message tampering, and information leakage (identity), which can be launched on protocols. We have analyzed these attacks using various Scyther attributes like Alive, Niagree, Nisynch, and secret.

The rest of the paper is organized as follows: Section 2 explains the detailed architecture of the proposed DB-SECaaS system. Section 3 describes the workflow and the intercommunication of all the services that make up the DB-SECaaS system. We evaluate the security of the DB-SECaaS system in Section 4. In Section 5, we provide an overview of the related work while highlighting our contributions and finally, Section 6 concludes the paper.

2 The DB-SECaaS system

As mentioned previously, the proposed system is a comprehensive DB-SECaaS system for hosting document-based NoSQL in cloud. It offers all the primarily required security services for the underlying database, specifically strong authentication, fine-grained authorization, data encryption, and data integrity. The service-oriented architecture ensures that the system provides effective security mechanisms to any document-based database. Our system constitutes multiple independent services that handle important security features. Each of them handles an important security feature and communicates with other services, working together to provide holistic security to the document-oriented database.

For the sake of simplicity, we do not attach the requisite aaS for as-a-service every time they are mentioned. In order to provide these services, we utilize renowned standards, namely SAML [15], XACML [16], and FIPS 196 [14]. The three main services at play are authentication service, fine-grained authorization service, and encryption service, which further depend on services that complete their functionality. In the following subsections, we describe these services that make up the DB-SECaaS. Figure 1 illustrates the architecture for building the system in the context of cloud services.

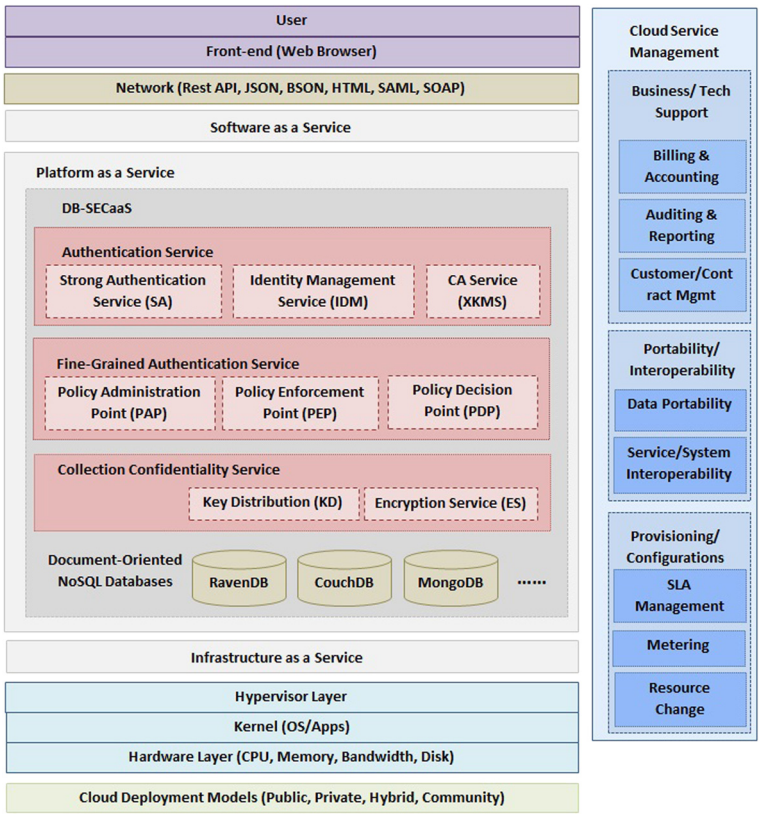

Fig. 1 The proposed architecture of the DB-SECaaS system over a document-oriented database hosted in cloud

2.1 Authentication service

The authentication service is responsible for verifying the identities of the database users as well as the inter-system requesting parties. It is composed of three services: *strong authentication as a service, identity management as a service, and certificate authority as a service.*

1. *Strong authentication (SA) as a service*: The SA service performs authentication using the FIPS 196 mutual authentication protocol. Its extended security functions include verification of certificates by the CA service and verification of identities by the IDM service. SA service is also used to issue and store SAML authentication tickets to other services and users.
2. *Identity management (IDM) as a service*: The IDM service manages the identity credentials of the database users, and once their identity is verified, it issues SAML authentication assertions to confirm their authenticity.
3. *Certificate authority (CA) as a service*: The CA[1] service, with the support of XML key management specification (XKMS), provides digital signature services and issues certificates, which consist of a public key and the necessary credentials required to verify the database users and the services, for authentication purposes. The CA service checks the authenticity and also manages the keys and certificates thus, establishing a reliable and safe networking environment. The XKMS is used to integrate and allow the easy management of the CA service, thereby reducing the complexity of managing CA at the end users and other DB-SEC system services. The integration of XKMS with CA provides a simple XML-based protocol for processing key/certificate information by eliminating the need for user applications and other services to understand the CA syntax and semantics.

The aforementioned authentication services are used not only for verifying the users but also for other DB-SECaaS system services. The entities can mutually or unilaterally authenticate each other before exchanging the messages or share the data.

2.2 Fine-grained authorization service

The fine-grained authorization service is responsible for protecting the data from unauthorized disclosure. It is

primarily composed of *policy administration point as a service, policy decision point as a service, and policy enforcement point as a service*.

1. *Policy administration point (PAP) as a service*: The PAP service allows users to create XACML policies that are compliant with the fine-grained access control (FGAC). System administrators can create fine-grained policies through the PAP service and modify them any time, as per their requirements. These policies are stored in a *policy repository on cloud*.
2. *Policy enforcement point (PEP) as a service*: The PEP service acts as an intermediary between the policy decision point (PDP) service and the client application by capturing the client's request for accessing documents or collections in the document-oriented database and converting it into an SAML-wrapped XACML authorization decision query and sending it over to the PDP service. It is also responsible for interpreting the authorization decision that it receives from the PDP service and converting it into the native form that the client application can understand.
3. *Policy decision point as a service*: The PDP service is responsible for evaluating the authorization decision request sent by the PEP service. Once it receives the request from the PEP service, it fetches policies from the *policy repository* and finds the policies applicable to the request. It then determines whether to permit or deny the user access to the database. If the permission is granted, it puts forth a SAML attribute request to the key distribution (KD) service to retrieve the requested resource. If the permission is denied, the PDP service returns a response to the PEP service, which is then forwarded to the client.

2.3 Collection confidentiality service

This encryption service is responsible for performing the encryption and decryption of the data stored in the collection on the request of a privileged user. When the data owner initiates the request to store the data in collection, it first passes through the encryption service where the data is encrypted through the collection key. By default, every collection is encrypted with a unique key, i.e., all the data which is to be stored in the same collection is encrypted by using the same key. However, if security is an utmost concern, then fine-grained encryption can also be provided where each column of the table has a unique key. This kind of encryption increases the security at the cost of performance of the system. After encryption, the service stores the encrypted data in the document-oriented NoSQL database. There are two sub-modules, i.e., *key distribution service and encryption service*, at the core of the encryption service.

1. *Key distribution as a service*: The KD service operates with symmetric encryption and shares secret symmetric keys for the encryption and decryption of the data stored in document-based NoSQL. KD service separately manages the keys for each user and for each resource. A complete mapping between keys, users, and document-oriented database resources is provided to avoid conflicts and searching overhead. This service is also responsible for creating, managing, and distributing the keys across different services within the proposed system. PDP service sends an *attribute query request* (discussed in the next section) to the KD service for the retrieval of keys from the KD database. KD service validates the request through CA (FIPS 196 protocol) and checks for keys corresponding to the attributes mentioned in the request. The keys are passed to the encryption service for further processing of encryption/decryption operation.
2. *Encryption as a service*: This service handles the encryption and decryption of the data using the advanced encryption standard (AES) algorithm, after getting keys from the KD service. Its working process is the same as traditional encryption-decryption models; it is responsible for handling the designated operations based on user queries to document-oriented database.

2.4 Document-oriented NoSQL database

This can be any document-oriented NoSQL database in cloud that will make use of all the security services that our system provides, for example, MongoDB, CouchDB, and RavenDB.

3 Execution flow of the DB-SECaaS

In this section, we will provide a detailed description of the workflow of our system, which includes the three basic security features that our system provides, i.e., *authentication, authorization, and encryption*. The system works in four phases, namely policy creation phase, authentication phase, fine-grained authorization phase, and encryption phase.

3.1 Policy creation phase

In this phase, the system administrator creates fine-grained policies that are meant to effectively filter out unauthorized users from getting access to the document-oriented database. The steps comprising this phase are described below:

1. Administrators define fine-grained policies through the PAP service and can modify and create new policies at any time.

2. Policies are pushed into a policy repository, which the PDP service uses to evaluate authorization decisions.

3.2 Authentication phase

This phase deals with the authentication of the user and DB-SEC system services, which is handled by the SA service, the IDM, and the CA service, compliant with the FIPS 196 mutual authentication protocol. Figure 2 shows the workflow of the user authentication phase. The same procedure has also been followed for services authentication with each other.

1. The user of the DB-SECaaS system sends a login request to the SA service as the first message in the FIPS 196 strong authentication protocol.
    (a) The SA service checks the validity of the user with the IDM service and
    (b) Sends the response back to the user as per the FIPS 196 message.
2. The user submits the certificate to the SA service, which then verifies it using certificate chain validation and the OCSP protocol.
    (a) The certificate request and response mechanism follows the XKMS protocol to perform the basic public key infrastructure (PKI) operations at the CA service. In other words, user application (web browser) sends XKMS messages over simple object access protocol (SOAP) to perform the operations.
    (b) If both verifications are successful, SA service will execute the FIPS 196 compliant challenge/response protocol with the user.[2]
3. After the successful execution of challenge-response protocol and user verification, the SA service issues the SAML ticket, which is sent back to the user (client application) along with the SAML authentication response *<saml: Response>*, consequently asserting the user's identification. The *<saml: Response>* contains, among other tags, a SAML assertion *<saml: Assertion>* and authentication statement *<saml:AuthnStatement>*, which in turn contains the evidence of authentication and the user and issuer's attributes.
4. The user stores the SAML ticket locally in a file on a disk. This SAML ticket is also stored by the SA service for verification at the next steps. The final results of the authentication procedure are that the user has the SAML ticket on his/her disk and the SA service has the copy of the ticket issued to the user. The authentication workflow mentioned above, is also used by other DB-SEC services of the system to authenticate one another. For instance, in order to send the authorization request and response, the PEP and PDP services should first mutually authenticate each other using the authentication mechanism defined above. Similarly, the KD and encryption services should first authenticate before sending the database keys. In other words, all the services of DB-SECaaS system must be authenticated and a SAML ticket is assigned to each service for a specific lifetime (more than the lifetime of the user SAML ticket), during which the services can communicate with each other for operations execution.

3.3 Authorization phase

Once the user has been authenticated, she can make a request to access the collection, document, or an attribute

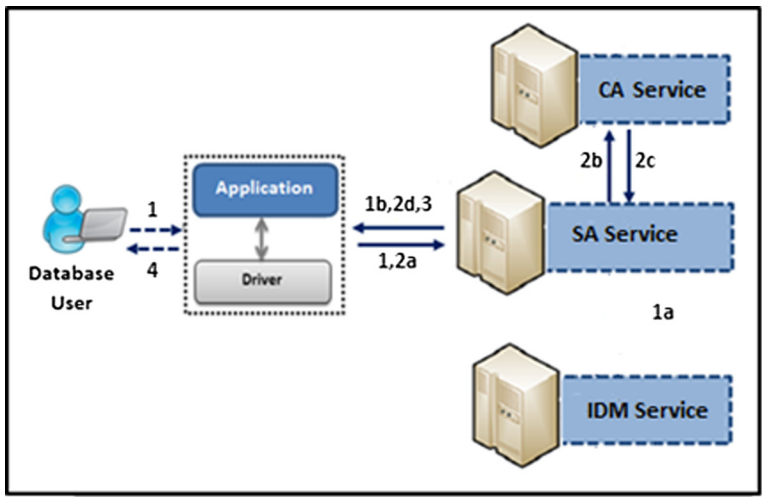

Fig. 2 Detailed workflow of the authentication phase

of a document, where the fine-grained authorization service steps are in. Figure 3 depicts the internal working of the fine-grained authorization service.

5. (a) User requests data from the document-oriented database and, along with the request, sends the SAML authentication ticket, which is intercepted by the PEP service.
   (b) PEP server passes the SAML ticket to the SA service, which checks the ticket and confirms its validity.
   (c) The SA service returns its decision back to the PEP.
6. The PEP service passes a *XACML-SAML authorization decision query* *<xacml-samlp:XACMLAuthzDecisionQuery>* to the PDP service for approval. The request contains the user ID, action required, and indication of all document-oriented database resources needed to fulfill the request. The PDP makes the decision of whether the user should be allowed access or not.
7. In order to evaluate the user's request, the PDP service fetches the XACML policies from the *policy repository*. The PDP service will use the XACML subject, resource, and action attributes within the authorization query to decide which policy is applicable and whether the subject can perform the specified action on the requested resource (see Fig. 4).
8. Information and policy in hand, the PDP service is able to render a decision.
   (a) If the request is denied, the PDP service sends the "Deny" decision to the PEP service. In case the access is granted, the PDP service generates and sends the SAML authorization ticket and response in the form of *<samlp:Response>* (see Fig. 5) to the PEP, which then sends it to the user application.
   (b) In addition, the PDP service also simultaneously sends a *SAML attribute query* *<samlp:AttributeQuery>* to the document database to retrieve the data (attributes values, documents, collection) for the requested data, but since the database is encrypted, the request will be intercepted by the KD service.
9. The SAML authorization ticket received from PEP is stored at the user's local disk as well as at PDP service to allow the user to access the same resource again within the interval specified in the ticket.
   These SAML tickets, generated at steps 6 and 8 of authentication and authorization phases, represent single sign-on (SSO) into the network or cloud. The ticket has a default lifetime of 8 hrs. The advantage of SSO is that if a user tries to access the same data from the document-oriented database within a specific interval, mentioned in the ticket, then she does not require to follow the same authentication and authorization steps again. Rather, the user application only needs to represent the SAML authentication and authorization ticket to the PEP, which is forwarded to the PDP and SA services for

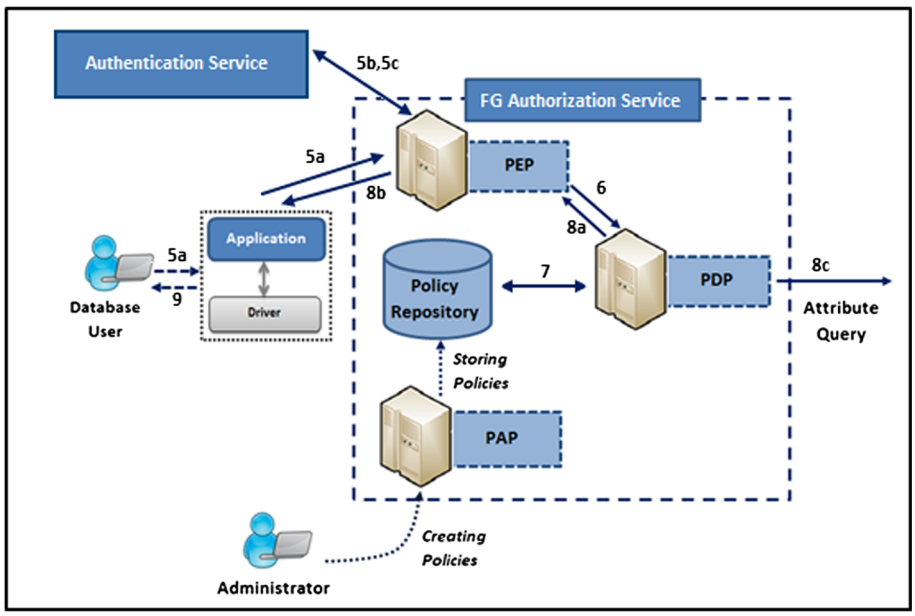

Fig. 3 The detailed workflow of the authorization phase (The *dotted arrows* depict the policy creation phase)

```
<xacml-samlp:XACMLAuthzDecisionQuery...>

    <xacml-context:Request>

      <xacml-context:Subject>

         <xacml-context:Attribute...>
           <xacml-context:AttributeValue>
             User
           </xacml-context:AttributeValue>
         </xacml-context:Attribute>
       </xacml-context:Subject>
       <xacml-context:Resource>
         <xacml-context:Attribute...>
           <xacml-context:AttributeValue>
             Document
           </xacml-context:AttributeValue>
         </xacml-context:Attribute>
       </xacml-context:Resource>
       <xacml-context:Action>
         <xacml-context:Attribute...>
           <xacml-context:AttributeValue>
             Read
           </xacml-context:AttributeValue>
         </xacml-context:Attribute>
       </xacml-context:Action>

    </xacml-context:Request>

    <saml:Evidence>

      <saml:Assertion>

        <saml:Issuer>Client</saml:Issuer>
        <saml:AuthnStatement AuthnInstant="2001-12-17T09:20:47.0Z">
        </saml:AuthnStatement>

      </saml:Assertion>

    </saml:Evidence>

</xacml-samlp:XACMLAuthzDecisionQuery>
```

Fig. 4 XACML-SAML authorization decision query

onward decision. This procedure of SSO and re-usability of SAML ticket saves the time and decreases the overhead of calling same operations repeatedly.

3.4 Encryption phase

The encryption phase is shown in Fig. 6. Once the PDP sends the attribute query to the KD service to retrieve the key, the following steps take place inside the encryption service:

10. The KD service receives the SAML Attribute Query *<samlp:AttributeQuery>* (see Fig. 7) from the PDP, which requests the data object(s) that the user requested.

```
<samlp:Response...>

    <saml:Issuer...>PDP</saml:Issuer>

    <ds:Signature>...</ds:Signature>

    <saml:Assertion...>

       <saml:Statement...>
       <xacml-context:Response...>
           <xacml-context:Result...>
              <xacml-context:Decision>Deny</xacml-context:Decision>
                  <xacml-context:Status>
                     <xacml-context:StatusCode.../>
                  </xacml-context:Status>
           </xacml-context:Result>
       </xacml-context:Response>
       </saml:Statement>

    </saml:Assertion>

</samlp:Response>
```

Fig. 5 XACML-SAML authorization decision query response

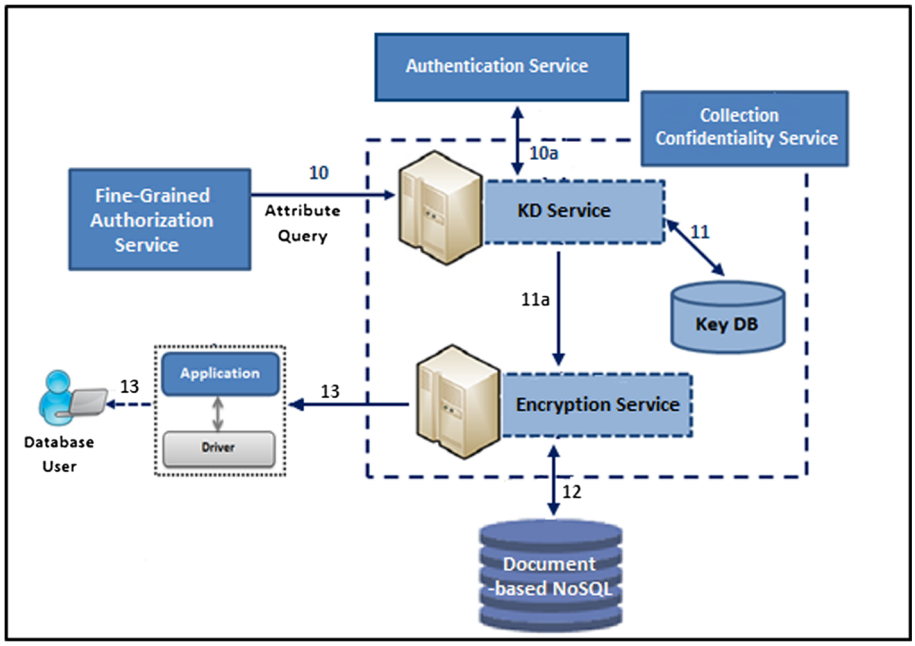

Fig. 6 The workflow of the encryption phase

(a) The KD service verifies with the CA if the request is actually from the PDP service so that it can mitigate the risk of repudiation. The CA service would extract the signature from the attribute query to verify the sender of the request.

11. Once the attribute query has been verified, the KD service communicates with the database to retrieve the desired keys.

    (a) KD service sends the key to the encryption service for decryption process.

12. *Encryption service* retrieves the encrypted data from the document-oriented database and decrypts it using keys.

13. Data is finally sent to the user, enveloped in the *SAML attribute response <samlp: Response>* (see Fig. 8).

Figure 9 shows the overview of the workflow of the system, a holistic view of the interaction of the services with each other.

4 Security evaluation

The main objective of any security feature or parameter within a system is to defend against the attacks and to patch the vulnerabilities from being exploited by the attacker. Thus, every newly developed system must be verified to ensure that it is free of security loopholes or weaknesses. We have separately evaluated the effectiveness of the major services the DB-SECaaS has to offer, i.e., fine-grained authorization service and the encryption service via qualitative measures as per the National Institute

```
<samlp:AttributeQuery...>
     <saml:Issuer>PDP</saml:Issuer>
     <ds:Signature>...</ds:Signature>
     <saml:Subject>
        <saml:NameID>User</saml:NameID>
     </saml:Subject>
     <saml:Attribute Name="DocumentData"/>
</samlp:AttributeQuery>
```

Fig. 7 SAML attribute query

```
<samlp:Response...>

        <samlp:Status>
          <samlp:StatusCode/>
        </samlp:Status>
        <saml:Assertion>
          <ds:Signature>...</ds:Signature>
          <saml:Subject>
            <saml:NameID>User</saml:NameID>
          </saml:Subject>
          <saml:AttributeStatement
      xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
            <saml:Attribute Name="Document">
              <saml:AttributeValue>DocumentData</saml:AttributeValue>
            </saml:Attribute>
          </saml:AttributeStatement>
        </saml:Assertion>
</samlp:Response>
```

Fig. 8 SAML attribute query response

of Standards and Technology (NIST) recommendations. Moreover, we have used a well-known formal verification tool, known as Scyther [18], to verify secure communication within our system.

4.1 Evaluation of the authorization service

The evaluation of the authorization service has been done with respect to the NIST-identified security metrics for access control models [19, 20]. NIST guidelines for access control evaluation metrics provide a complete ontology for access control policy. This ontology determines the relationships between access control primitives. For fine-grained authorization service, we have also designed an ontology that explains its core features and functions. The fine-grained authorization service is then evaluated using the metrics, defined by NIST, which are further divided into two categories: *administration* and *enforcement*. Table 1 explains the access control metrics supported by the system.

The NIST guidelines for access control evaluation metrics provide a complete ontology for access control policy. This ontology determines the relationships between access control primitives. For fine-grained authorization service, we have also designed an ontology that explains its core features and functions. The fine-grained authorization service is then evaluated using the metrics, defined by NIST, which are further divided into two categories: *administration* and *enforcement*. Table 1 explains the access control metrics supported by the system.

4.2 Evaluation of encryption service

Evaluation of encryption service is carried out through the given guidelines and recommendations of NIST on encryption and key management [21]. For the effective use of cryptography in databases, it is essential to properly create and manage the keys. We have used the KD service for secure key generation, storage, and distribution

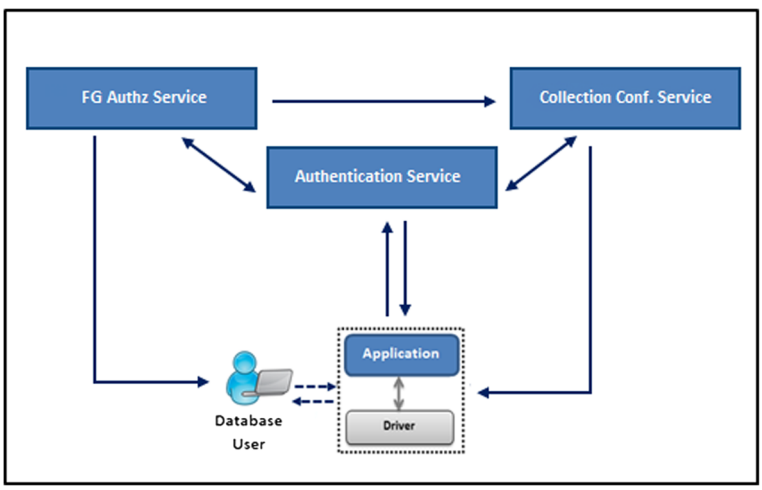

Fig. 9 Overview of the workflow of the DB-SECaaS system

Table 1 Support of NIST access control metrics by fine-grained authorization service of the proposed system

Metric	Description	Fine-grained authorization service
	Administration properties	
Privileges/capabilities discovery	Query or graphic display for discovering the subjects, objects, or capabilities from assigned privileges.	Our system provides a complete interface which is provided to authorize users (administrators, data owners) to view the database subjects (data users), objects (database columns), and capabilities (actions).
Ease of privilege assignments	Steps required to 1. Assign, change, or remove privilege from a subject. 2. Assign, change, or remove the capabilities of a subject. 3. Create, change, or remove a subject.	The fine-grained authorization service requires less steps for assigning, changing, managing, and removing privileges, capabilities, objects, and subjects Usability of a system is increased through friendly interface and less turnaround steps.
Syntactic support for specifying AC rules	Authorization system must be capable of providing logical expression for the specification of policies and rules.	FG authorization service is based on the architecture of XACML; therefore, full support is provided for complex expressions such as AND, OR, <, and >.
Policy management	Authorization system must provide the ability to resolve policy conflicts, policy revocation, and policy identification functions.	The proposed system provides policy management features to administrators via PAP.
Flexibilities of configuration into existing systems	Access control needs to be enforced by application and client/service protocol in order to provide more flexibility and security.	FG authorization service is based on application and client-service model. PAP acts like an application for administrators to create and manage policies. This will provide ease in installation and configuration.
The horizontal scope	Authorization system for unstructured databases should be supported by multiple hosts via network. Moreover, access control should be defined across database records and fields of database records.	Distribution ability in FG authorization service is provided by hosting each service of the system in different services. In addition, multiple services can be used to host replicas of the single service. Vertical scope is provided by defining policies across database records and fields.
	Enforcement properties	
Bypass	Authorization system can be designed in a way to bypass the policy rules for exceptional access control decisions.	The FG authorization service of our system does not allow request to bypass PEP or PDP to access database resources. There is no method defined that can ignore policy service in exceptional or critical situations.
Least privilege principle	An effective authorization system supports least privilege principle. For databases, least privilege needs to be defined at the cell level or column level.	FG authorization service specifies policies at cell, column, and table levels of the database. Access to every cell requires permission from policy service.
Separation of duty (SoD)	Authorization system can either implement static or dynamic separation of duties.	In order to prevent data from excessive privilege abuse, the proposed system permits authorized users to access duty-related resources. However, fine-grained authorization only provides static SoD where privileges assign to subjects need to be defined before practical execution of the system.
Conflict resolution or prevention	Authorization system must be capable to prevent and resolve policy rule conflicts.	Use of XACML ensures the prevention of policy and rule conflicts. Conflict-avoiding algorithms are provided to resolve the conflicts automatically.
Operational/situational awareness	An effective authorization system must provide situational awareness (environmental constraints and conditions).	FG authorization service has the ability to take into account environmental variables such as time, threshold values, and behavior for making access decisions. XACML provides the environmental functions and conditions to restrict access to particular domains.
Granularity of control	Authorization system must be capable to provide control up to granularity of cell or column (objects). Same data needs to be protected at different levels of granularity.	Architecture of FG authorization model is based on granularity of objects. Therefore, the proposed system also provides privacy control for the data with different classifications of the fields in database.
Expression (policy/model) properties	Authorization system needs to support existing access control standards or rule specification language.	XACML, a standard policy language for access control systems, is used for the representation of policies and rules to protect the data.

via strong algorithms. These keys are used in symmetric key algorithms for protecting data against unauthorized attempts and modifications. Table 2 presents the evaluation of our encryption service according to the guidelines and recommendations given by NIST. The evaluation of encryption service helps to approve its correct functioning for document-oriented databases.

4.3 Formal analysis

In this subsection, we provide a detailed description of our formal analysis of DB-SECaaS using Scyther, which has been shown to be a quite effective model checking tool for verification, falsification, and analysis of security protocols. Scyther can verify protocols with unbounded number of sessions, with guaranteed termination. Scyther formally analyzes security protocols under the assumption of perfect cryptography; for example, it may validate that an attacker can learn nothing from an encrypted message unless it has a key.

Table 2 Evaluation of encryption service in the system

Cryptograhic algorithm: symmetric data encryption
Security function: data encryption
Algorithm-related information: • Initialization vectors (IVs) are used to alleviate the problem of encrypting the same data with the same key. The problem of repeated block detection and substitution of individual blocks is catered by introducing variable IVs. • Shared secrets such as passwords or passphrases can also be used to generate the key. This key is then used to encrypt the data. For decryption, the same key needs to be generated via shared secret. This secret is only shared with the users which are authorized to access the data. • RNG seeds: random seed values are also used to generate different keys for the encryption. However, these randomly generated values are stored at KD service repository in order to get the same key for decryption.
Cryptoperiods: We define cryptoperiods as the amount of data that are protected by a given key. For this encryption service, we have used different keys for each column or table to avoid risks of exposure. Little data is exposed if key is compromised by any adversary. Moreover, cryptoperiods to protect the data with one key is usually longer because of overhead issues of changing keys frequently.
Cryptographic mechanism:
Key size: AES-128
Operating environment: limited access to KDMS service (only to authorized users)
Protection mechanism: • Availability: backups and replicas of KD service are created at different locations to make data readily available for cryptographic functions • Integrity: KD service is protected from unauthorized modifications using physical and cryptographic mechanisms. This service is placed at a fully secure environment with appropriate access controls and limited access. In addition, integrity of the stored information is checked through message authentication code (MAC). • Confidentiality: an encryption algorithm approved by FIPS 140-2 is used through which it is not easier to recover the key. Moreover, controlled access is provided via access control mechanisms. • Integration with other applications: KD service is hosted separately from other applications and services. This service is organized in a form of layered architecture and communicates securely via secure channel; therefore, there are fewer chances of data misuse and disruption.

Model checking is a state-space-based formal technique for modeling and analyzing hardware and software systems. It performs exhaustive verification of a system modeled as a finite state machine (FSM) in an automatic manner. The verification process consists of the following steps:

1. The system to be verified is described as a state-space model in a formal notation specified by the model checker.
2. The properties of the system that need to be verified are extracted from the working of the system and formally specified using temporal logic, like linear temporal logic (LTL) or computational tree logic (CTL).
3. The formal model of the system along with its properties are fed to the model checker to automatically verify if the system meets the required specifications.

One of the most powerful features of model checking is the provision of generating a counter-trace in case of a failing property. This counter-trace can be used to determine the cause of the error, which could be a modeling issue or a bug in the system under consideration. However, the state-space model of a complex system may become very large making the verification task quite complex computationally, i.e., a problem that is usually referred to use the state-space explosion problem. Generating a more abstract model of the system or using bounded model checking (BMC) [22] are commonly used to overcome the state-space explosion problem. BMC reduces the state-space exploration effort by restricting the search for a counterexample for a given property within k-bound levels.

In order to verify the security of DB-SECaaS, we propose to focus on three of its main security features, i.e., authentication, authorization, and encryption. The specification of the roles in all three phases is done individually for the verification of corresponding properties. These properties are formally specified in the form of Scyther claims.

We use five claims that Scyther supports namely, *Secret*, *Alive*, *Niagree*, *Weakagree*, and *Nisynch*. *Secret* makes sure that the confidentiality of our messages being sent/received is not compromised. For the DB-SECaaS system, it ascertains secure communication between the database application, SA service, and IDM service. *Alive* validates that the communication partner has sent a

response to the requesting entity and is available. It ensures that the responding entity is available when a request is sent to it, e.g., it can be used to check if the CA is alive. *Nisynch* is a synchronization claim that ensures that the protocol is in synch and that it is secure against replay attacks. It can be used to validate that the PEP and PDP services are synchronized. *Niagree* is the non-injective agreement claim that verifies source authentication, for instance, it can be used to ascertain that the CA authenticates the source effectively.

4.3.1 Model description

The Scyther model for DB-SECaaS mainly describes the behavior of the protocol in terms of its roles, i.e., either an initiator or a responder. Our system consists of multiple communicating agents, including the client, and the various component service agents of DB-SECaaS, e.g., KDS, PEPaaS, and PDPaaS. Agents execute their runs to achieve their security goals. Every role specification consists of a sequence of events describing the messages, which the agent shall send and receive, as well as certain security claims. An intruder may try to oppose these security goals. The capabilities of the intruder determine its strength in attacking a protocol run. In order to resist attacks, an agent can make use of cryptographic primitives when constructing messages. More details about the message elements that are used in our formal model can be found in [23].

The actual behavior of the entire system, consisting of the intruder and a set of agents executing a number of runs, is encoded in the traces of the system. Every claim event in a trace results in a declaration about the trace that may or may not be true. A secrecy claim event is essentially the statement that something is never known to the adversary. Authorization is the process of determining which permissions a person or system are supposed to have. Authentication is captured by the notion of synchronization, which in turn requires that the corresponding send and receive messages are executed in the expected order. Effective security must employ both strong authentication and strict authorization.

The following claims are used, where the x denotes the role for which the claim is tested and y is the message:

- *Claim (x, Secret, y):* The agent performing the role x knows that the intruder will never have knowledge of y.
- *Claim (x, Nisynch):* The agent performing the role x knows that the message it received is from an authenticated sender [17].
- *Protocol specification:* Describes the behavior of each of the roles in the protocol. Most often, a role in a security protocol is specified as a sequential list of events.
- *Agent model:* The agents execute the roles of the protocol. The agent model is based on a closed world assumption. This means that honest agents show only the behavior described in the protocol specification.
- *Communication model:* Describes how the messages are exchanged between the agents. We assume asynchronous communication with a single network buffer because this is the most general approach.
- *Threat model:* It is based on a parameter in the semantics of the model based on the Dolev and Yao's network threat model [24], where the intruder has complete control over the communication network.
- *Cryptographic primitives:* They are idealized mathematical constructs, such as encryption, using the black box approach. This means that an adversary cannot learn anything from an encrypted message except if he has the key.
- *Security requirements:* They are expressed as safety properties (i.e., something bad will never happen).

4.3.2 Formal verification results

The strength of any security application is the effectiveness with which it prevents attacks. Our evaluation through Scyther verifies the strength and validity of the DB-SECaaS for document-based databases on cloud. We have designed the DB-SECaaS system to provide protection to the underlying document-oriented NoSQL database by providing certain security features. Whether these features are effective or not depends on how successfully the system circumvents attacks, and the claims in Scyther help validating the mitigation of attacks. Table 3 provides the specific attacks our solution mitigates and the mechanisms it employs to do so. The results show that the provided security feature are quite effective, and given the fact that the system employs a secure communication protocol, like SAML, all the interactions between the system entities are also protected from external interferences. Therefore, apart from providing security services, our findings show that the DB-SECaaS system itself mitigates the security concerns which are raised in the previous sections.

More details about the results and specifications can be found in [23]. All phases of the system have been analyzed separately in order to ascertain the mitigation of all possible cryptographic attacks. Our analysis results conclude that there are no attacks that can be identified within the given bounds. There can be attacks outside the bounded state-space, but the state-space in the case of our protocol is very huge; thus, the model cannot be checked rigorously due to the infamous problem of state-space explosion in model checking. We overcame this problem by using a modular approach and thus analyzing sub-modules of the complete system at a time. Authentication, authorization,

Table 3 Attacks mitigated by the DB-SECaaS system, as a whole

Attack	Status	Entity	Mechanism
Alteration	Mitigated	KD service, encryption service	Our system uses encryption to protect the data from alteration and forgery. Only authorized users have the privilege to get the keys for decryption.
Denial of service	Partially mitigated	KD service, FG authorization service	This threat is partially mitigated using access control. Requests cannot directly communicate with the database, rather they need to be validated by authorization service to get data access. However, further study is needed to investigate DoS attack on authorization service.
Excessive privilege abuse	Mitigated	FG authorization service	Fine-grained policies in our system are defined based on "need-to-know" principle where privileges to resources are assigned from cell level to table level. Principle of least privileges is followed.
Unauthorized elevation of privileges	Mitigated	FG authorization service	Each request to database is evaluated by fine-grained authorization policies, which keep malicious insiders from escalating their privileges.
Injection attacks	Partially mitigated	FG authorization service	Injection attacks mostly occur at application layer. Even if malicious user breaches the application security, he still needs to pass the fine-grained authorization service. Injection attacks are also partially mitigated using least privilege access control model. Fine-grained access control model allow user to access data on a need-to-know basis.
Backup data exposure	Mitigated	Encryption service, KD service	The system protects backup data using encryption service. Data is stored in encrypted form; therefore, if malicious attempts are made, it is still useless.
Login attacks	Mitigated	SA service, IDM service, fine-grained authorization service, encryption service, KD service	We have combined authentication with authorization where user request is evaluated to access to data. In addition, keys are managed separately at KD service to decrypt the data. Encryption together with the authorization makes overall security more strong.

and encryption phases are verified individually with the properties, as shown in the analysis exhibits.

5 Related work

A lot of research has been carried out in the last few years to mitigate the security challenges in cloud databases, but contributions specifically dealing with NoSQL databases are few and far between. Moreover, to the best of our knowledge, none of the existing works propose a comprehensive security solution for NoSQL databases in cloud. There are a few works available regarding document-oriented NoSQL security, which serves to emphasize the immaturity of security techniques in this particular domain. Due to the lack of research in our area of focus, we have carried out an extensive survey with a three-dimensional approach: we begin with solutions that provide a particular security service for cloud databases; subsequently, we move on to solutions that increasingly incorporate more security features to protect cloud databases, like security-as-a-service (DB-SECaaS). Finally, we narrow it down by enlisting the security mechanisms that some of the more popular document-oriented databases have to offer.

5.1 Focused security solutions for cloud databases

Delettre et al. [25] have proposed a data concealment security component in order to ensure the confidentiality of the data stored in cloud. Tao et al. [26] also provide confidentiality through fully homomorphic encryption, which performs functions on encrypted data stored in the cloud via an encryption proxy. In a distinctive approach, presented in [27], Ferreti et al. get rid of the encryption proxy, making the operations on encrypted data much faster. Bracci et al. [28] propose to provide adequate encryption and key management support by using a real use case of Vitaever, i.e., a home health-care SaaS application deployed on Amazon AWS. Sanka et al. [29] propose symmetric key sharing between CSP and the user by using the Diffie-Hellman key exchange protocol. The authors make a lot of assumptions, which can be potentially harmful in the real-world scenario.

Wang et al. [30] and Tribhuwan et al. [31] make use of homomorphic tokens to achieve data correctness. The only essential security service this covers is the integrity of the data in cloud. NETDB2 Multi-Shares (NETDB2 MS) [32] ensures privacy in database-as-a-service and is based on multi-service providers and the secret sharing algorithm instead of encryption used by the existing NetDB2 service. This new model circumvents the high cost of data encryption and decryption.

All the aforementioned works are commendable in their own right; however, all of them offer focused solutions for data security in cloud environment, mainly providing one security service at a time, for instance, confidentiality.

We, however, are looking for more holistic solutions that incorporate all the basic security requirements.

5.2 Security-as-a-service solutions for cloud databases

A few fairly recent attempts have been made to present a more inclusive solution for data security in cloud. A three-dimensional approach to securing data in cloud is presented in [33], wherein data is categorized on the basis of a priority rating, which incorporates confidentiality, integrity, and availability. Sood [34] extends this system by adding data protection via the 128-bit SSL encryption and strengthening the authentication mechanism. Mohamed et al. [35] propose a similar data security model for cloud environments, having three layers, where each layer handles specific security tasks: authentication, data encryption, data integrity and user privacy, and fast data recovery. Islam et al. [36] present an agent-based framework for providing confidentiality, integrity, and authenticity to data storage in cloud.

Akin to what we are proposing for cloud, Yamany et al. [37] presented an intelligent service-oriented architecture security framework, which includes authentication and security service (NSS), as well as the authorization service (AS), based on the WS-* security standards. Similarly, Song et al. [38] proposed a generic framework of data-protection-as-service (DPaaS), which integrates various protection modules including access control, key management, and logging, to provide a combined multi-tier protection mechanisms for the current cloud. This, like [37], is just a concept that aims to provide integrity, privacy, confidentiality, and fine-grained access control in cloud. In particular, these methods do not target NoSQL databases. Hussain et al. [39] introduce security-as-a-service (DB-SECaaS), a user-centric architecture that provides security services for cloud computing on its different levels (SaaS, PaaS, and IaaS).

The papers, mentioned in this subsection, aim to provide multiple security services through one solution, which is essentially our goal as well. However, most of them are merely concepts and do not discuss implementation details. Also, they are not sensitive to the security requirements of document-oriented databases in cloud, which is our domain of interest.

5.3 Document-oriented database security

Based on the research we have conducted, we have found that document-oriented databases are increasingly incorporating more and more security mechanisms to ensure data security. Table 4 provides a comparison of the basic security features provided by some of the more popular document-oriented NoSQL databases, i.e., CouchDB [40], MongoDB [41], and RavenDB [42], with DB-SECaaS.

From Table 4, we can conclude that even the most popular of the document-oriented databases cannot provide comprehensive security in the most efficient manner. Authorization mechanisms need to be more fine-grained, not only based on roles but also on resources. But on the whole, they are all lacking in one or more aspects of security and this needs to be addressed if document-based databases in cloud are to be widely adopted.

5.4 Discussion

We have reviewed the above mentioned papers, most of which provide specific solutions for cloud security. Some focus on specific security services, like confidentiality or availability, while others aim to mitigate specific risks or threats that exist in the environment. None of them provide a holistic security solutions for NoSQL databases, and those that tackle this problem have certain weaknesses or have not been implemented and just propose a concept. Not to mention, none of them aims to provide security for NoSQL databases in the cloud. Another notable attribute of most solutions is the tight coupling of security with the underlying database, which restricts the solution from being applied to any other database.

In order to overcome the identified weaknesses, we have implemented a security-as-a-service system over document-oriented NoSQL in cloud, which provides authentication, fine-grained authorization, as well as encryption, using well-established industry standards. In contrast to most solutions, ours is generic; and its layered architecture support any document-oriented database.

6 Conclusions

Document-oriented NoSQL in cloud adds the advantages of cloud computing—like scalability, inexpensiveness, and pay-per-use—to the benefits of document-based databases storage capacity for large volumes of data. On the other hand, however, the security weaknesses of cloud and document-based NoSQL also add up, outweighing the advantages. The proposed solution tends to enhance the security of document-based NoSQL databases in cloud and to, consequently, increase its adoption in enterprises. With these motivations, we have developed the DB-SECaaS system that protects the underlying document-oriented database in cloud by providing authentication, fine-grained authorization, and data encryption services. In order to implement the DB-SECaaS, we used the Javax cryptographic libraries, and well-known industry standards SAML 2.0 and XACML 3.0. We separately analyzed the major functional modules, i.e., the fine-grained authorization service and the encryption service, according to the NIST-defined standards. We used Scyther to further evaluate the security of our DB-SECaaS system. We also included a list of the potential security attacks that our system helps in mitigating. In addition, we have

Table 4 Comparison of security services provided by some popular document-oriented NoSQL databases with DB-SECaaS

Security service	MongoDB security [41]	CouchDB security [40]	RavenDB security [42, 43]	DB-SECaaS
Authentication	Authentication in MongoDB can be incorporated using password-based challenge and response protocol or x.509 certificates. Additionally, MongoDB supports various other third-party authentication mechanisms to integrate with existing authentication infrastructure.	Apart from basic username-password authentication, CouchDB also provides Cookie authentication, which generates a one-time token that can be used in the next request. By default, a token is valid for 10 min.	RavenDB comes with a built-in authentication functionality, and it supports two types of authentication: Windows authentication and OAuth authentication. An appropriate mechanism is chosen by examining the incoming request headers and by default all actions except read-only are being authenticated.	DB-SECaaS provides strong authentication using the FIPS 196 challenge-response protocol, which is designed to mitigate attacks such as reply. An additional layer of security using security assertion markup language (SAML) is included to validate the identity of the user and the various services participating in the system. The three major modules of *authentication service*, i.e., IDM, CA, and SA, provide the authentication using standardized technologies and help in validating users and modules that interact with DB-SEC system.
Authorization	MongoDB allows role-based access control wherein access is granted or denied based on the roles assigned to a user. Access can also be granted based on action and resource. MongoDB provides numerous built-in roles and users can create specific roles customized to clients' requirements.	By default, everyone is given administrative privileges, which allows them to do anything with the databases. However, privileges can be customized in order to restrict the operations that they are allowed to perform.	Any anonymous user can perform read-only operations. But other functions require authentication. Admin has the privilege to carry out all operations.	DB-SECaaS offers authorization at the granularity of attribute and document level of the database (fine-grained), based on XACML 3.0. The fine-grained XACML policies restrict the access level of users to ensure that a user can only perform certain operations on a certain data, if and only if is allowed to do so.
Encryption (in transit)	MongoDB provides support for SSL to make sure that only the intended recipient receives a transaction.	CouchDB, as of version 1.1.0, comes with built in SSL support.	RavenDB allows the usage of SSL.	The data is completely wrapped into SAML queries to provide protection during transit.
Encryption (at rest)	MongoDB provides encryption of data at rest by incorporating the proprietary data encryption solution provided by Gazzang [44]. Gazzang encrypts data in real time and offers advanced key management solutions.	No support for encryption of data at rest.	RavenDB provides support for data encryption. By default, it uses the AES-128 encryption algorithm, but that can be changed if needed. It applies to all documents and to all indexes.	The two major modules KD and encryption as a service helps in performing the encryption and decryption of the data stored in collection on the request of privileged user.

also defined classes of data security for document-based databases and have found that our system provides data security of class 3. After a thorough evaluation of our DB-SECaaS system, we conclude that the mechanisms for providing major security features are effective and afford optimal protection to document-oriented NoSQL databases. Using the Scyther security verification tool, the authentication, authorization, and encryption components of the proposed framework are tested against possible attacks with promising results. The Scyther tool introduces adversaries that can compromise agents during the run of the protocol learning their keys or the random numbers generated. It can verify a protocol against adversaries that range from the usual model in symbolic analysis, the Dolev-Yao, to much more powerful adversaries. Thus, the successful verification of DB-SECaaS using Scyther definitely raises the level of trust in its security aspects.

In the future, we plan focus on the performance of DB-SECaas so that it can provide protection while minimizing its latency. We also aim to incorporate new security features into the system, including auditing, which involves the tracking of a database user's actions. This would elevate the security level of the database to class 4 (as per Table 1). Using the DB-SECaaS system, enterprises can host their critical data in document-based NoSQL

databases on the cloud securely, where the owner of the data would be able to control access to their data.

Authors' contributions

YG has penned the majority of this manuscript and has been involved in the process of the initial system design and its evolution to its current stage. RM has made key contributions in this research article, initially, by proposing the concept and then providing continuous support to the first author during all stages of the paper drafting. She was involved in the paper conception and provided major feedbacks till the final version of the paper. MAS has provided constant guidance and supervision through every iteration of this article. AR developed a formal model of the proposed protection system and analyzed it using the Scyther tool. OH supervised the work of Abid and ascertained the correctness of his formal models. Moreover, he also reviewed the text in the paper. All authors read and approved the final manuscript.

Author details

[1]School of Electrical Engineering and Computer Science (SEECS), National University of Sciences and Technology (NUST), 44000 Islamabad, Pakistan. [2]University of New South Wales (UNSW), NSW, Sydney, Australia. [3]VisionIT, Detriot, MI, USA.

Competing interests

The authors declare that they have no competing interests.

Received: 17 October 2015 Accepted: 28 June 2016
Published online: 03 August 2016

References

1. R Cattell, Scalable SQL and NoSQL data stores. ACM SIGMOD Record. **39**(4), 12–27 (2011)
2. J Han, E Haihong, G Le, J Du, in *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference On*. Survey on NoSQL database (IEEE, Port Elizabeth, South Africa, 2011), pp. 363–366
3. B Ritchie, An introduction to document databases (2010). http://weblogs.asp.net/britchie/archive/2010/08/12/document-databases.aspx. Accessed 29 May 2016
4. Apache, CouchDB (2015). http://couchdb.apache.org/. Accessed 16 May 2016
5. H Rhinos, RavenDB (2015). https://ravendb.net/. Accessed 14 May 2016
6. 10gen., MongoDB (2015). http://www.mongodb.org/. Accessed 29 May 2016
7. MongoDB., MongoLab (2015). https://mongolab.com/welcome/. Accessed 28 May 2016
8. RavenDB, RavenHQ (2015). http://ravenhq.com/. Accessed 14 May 2016
9. IBM, Cloudant. Web page (2015). https://cloudant.com/. Accessed 28 May 2016
10. McAfee, Data loss by the numbers (2015). http://docs.media.bitpipe.com/io_10x/io_108016/item_630596/wp-data-loss-by-the-numbers.pdf. Accessed 28 May 2016
11. TrendMicro, MongoHQ data breach a cautionary tale for startups (2015). http://blog.trendmicro.com/mongohq-data-breach-cautionary-tale-startups/. Accessed 29 May 2016
12. Forbes, iCloud data breach: hacking and celebrity photos (2015). http://www.forbes.com/sites/davelewis/2014/09/02/icloud-data-breach-hacking-and-nude-celebrity-photos/. Accessed 20 May 2016
13. W Post, A Snapchat security breach affects 4.6 million users. Did Snapchat drag its feet on a fix? (2015). http://www.washingtonpost.com/blogs/the-switch/wp/2014/01/01/a-snapchat-security-breach-affects-4-6-million-users-did-snapchat-drag-its-feet-on-a-fix/. Accessed 28 May 2016
14. S Wakid, Entity Authentication Using Public Key Cryptography. (National Institute of Standards and Technology (NIST), Gaithersburg, Maryland, 1997), pp. 1–52
15. F MALER, OASIS security assertion markup language (SAML) (2013). http://www.oasis-open.org/committees/security. Accessed 28 May 2016
16. E Rissanen, Oasis eXtensible access control markup language (XACML) version 3.0. OASIS Committee Specification 1. (OASIS 2010), 1–150. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf, Accessed 24 July 2016
17. CJF Cremers, *Scyther: semantics and verification of security protocols*. (Eindhoven University of Technology, Eindhoven, Netherlands, 2006)
18. Scyther, Evaluation tool (2015). http://www.cs.ox.ac.uk/people/cas.cremers/scyther/. Accessed 1 May 2016
19. VC Hu, D Ferraiolo, DR Kuhn, *Assessment of access control systems*. (US Department of Commerce, National Institute of Standards and Technology, 2006). http://csrc.nist.gov/publications/nistir/7316/NISTIR-7316.pdf. Accessed 24 July 2016
20. VC Hu, KA Kent, *Guidelines for access control system evaluation metrics*. (Citeseer, US Department of Commerce, United States of America, 2012)
21. E Barker, W Barker, W Burr, W Polk, M Smid, in *NIST Special Publication*. Recommendation for key management-part 1: general (revised (Citeseer, US Department of Commerce, United States of America, 2006)
22. A Biere, A Cimatti, EM Clarke, O Strichman, Y Zhu, Bounded model checking. Handb. Satisfiability. **185**, 457–481 (2009)
23. A Rauf, DB-SECaaS: A cloud based protection system for document-oriented NoSQL databases (2015). http://save.seecs.nust.edu.pk/projects/DB-SECaaS/. Accessed 24 July 2016
24. D Dolev, AC Yao, On the security of public key protocols. Inf. Theory IEEE Trans. **29**(2), 198–208 (1983)
25. C Delettre, K Boudaoud, M Riveill, in *Computers and Communications (ISCC), 2011 IEEE Symposium On*. Cloud computing, security and data concealment (IEEE, Kerkyra, Greece, 2011), pp. 424–431
26. L Tao, Y Xiaojun, W Jianmin, in *Proceedings of the Fourth Asia-Pacific Symposium on Internetware*. Protecting data confidentiality in cloud systems (ACM, Qingdao, China, 2012), p. 18
27. L Ferretti, M Colajanni, M Marchetti, in *Cyberspace Safety and Security*. Supporting security and consistency for cloud database (Springer, Melbourne, Australia, 2012), pp. 179–193
28. F Bracci, A Corradi, L Foschini, in *Computers and Communications (ISCC), 2012 IEEE Symposium On*. Database security management for healthcare SaaS in the Amazon AWS cloud (IEEE, Cappadocia, Turkey, 2012), pp. 000812–000819
29. S Sanka, C Hota, M Rajarajan, in *Internet Multimedia Services Architecture and Application (IMSAA), 2010 IEEE 4th International Conference On*. Secure data access in cloud computing (IEEE, Bangalore, India, 2010), pp. 1–6
30. C Wang, Q Wang, K Ren, N Cao, W Lou, Toward secure and dependable storage services in cloud computing. Serv. Comput. IEEE Trans. **5**(2), 220–232 (2012)
31. M Tribhuwan, V Bhuyar, S Pirzade, in *Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 International Conference On*. Ensuring data storage security in cloud computing through two-way handshake based on token management (IEEE, Kottayam, India, 2010), pp. 386–389
32. MA ALzain, E Pardede, in *System Sciences (HICSS), 2011 44th Hawaii International Conference On*. Using multi shares for ensuring privacy in database-as-a-service (IEEE, Kauai, Hawaii, 2011), pp. 1–9
33. P Prasad, B Ojha, RR Shahi, R Lal, A Vaish, U Goel, in *Computer Research and Development (ICCRD), 2011 3rd International Conference On*. Three-dimensional security in cloud computing, vol. 3 (IEEE, Shanghai, China, 2011), pp. 198–201
34. SK Sood, A combined approach to ensure data security in cloud computing. J. Netw. Comput. Appl. **35**(6), 1831–1838 (2012)
35. EM Mohamed, HS Abdelkader, S El-Etriby, in *Informatics and Systems (INFOS), 2012 8th International Conference On*. Enhanced data security model for cloud computing (IEEE, Cairo, Egypt, 2012), p. 12
36. MR Islam, M Habiba, in *Computer and Information Technology (ICCIT), 2012 15th International Conference On*. Agent based framework for providing security to data storage in cloud (IEEE, Chittagong, Bangladesh, 2012), pp. 446–451
37. HFE Yamany, MA Capretz, DS Allison, Intelligent security and access control framework for service-oriented architecture. Inf. Softw. Technol. **52**(2), 220–236 (2010)
38. D Song, E Shi, I Fischer, U Shankar, Cloud data protection for the masses. Computer. **45**(1), 39–45 (2012)
39. M Hussain, HM Abdulsalam, Software quality in the clouds: a cloud-based solution. Clust. Comput. **17**(2), 389–402 (2014)
40. CouchDB, The definitive guide "Security" (2015). http://guide.couchdb.org/draft/security.html. Accessed 18 May 2016
41. MongoDB, MongoDB security guide (2015). http://docs.mongodb.org/master/MongoDB-security-guide.pdf. Accessed 28 May 2016

42. RavenDB, Bundle: encryption (2015). http://ravendb.net/docs/article-page/2.0/Csharp/server/extending/bundles/encryption. Accessed 14 May 2016
43. RavenDB, authentication & authorization (2015). http://ravendb.net/docs/article-page/2.0/Csharp/server/authentication. Accessed 14 May 2016
44. Z Gazzang (2015). http://www.gazzang.com/products/zncrypt. Accessed 20 May 2016