Towards Scalable and Dynamic Data Encryption for Multi-Tenant SaaS

Ansar Rafique, Dimitri Van Landuyt, Vincent Reniers, Wouter Joosen
imec-DistriNet, KU Leuven
3001 Leuven, Belgium
{Ansar.Rafique, Dimitri.Vanlanduyt, Vincent.Reniers, Wouter.Joosen}@cs.kuleuven.be

ABSTRACT

Application-level data management middleware solutions are becoming increasingly compelling to deal with the complexity of a multi-cloud or federated cloud storage and multi-tenant storage architecture.

However, these systems typically support traditional data mapping strategies that are created under the assumption of a fixed and rigorous database schema, and mapping data objects while supporting varying data confidentiality requirements therefore leads to fragmentation of data over distributed storage nodes. This introduces performance overhead at the level of individual database transactions and negatively affects the overall scalability.

This paper discusses these challenges and highlights the potential of leveraging the data schema flexibility of NoSQL databases to accomplish dynamic and fine-grained data encryption in a more efficient and scalable manner. We illustrate these ideas in the context of an industrial multi-tenant SaaS application.

CCS Concepts

•**Information systems** → **Data federation tools; Cloud based storage; Distributed storage;** •**Security and privacy** → *Management and querying of encrypted data;*

Keywords

Cloud data storage, NoSQL, Data encryption, Untrusted clouds, Secure data management, Multi-tenant SaaS

1. INTRODUCTION

Software applications are increasingly offered as services that are deployed on cloud platforms [4, 15]. The limitations of traditional databases in terms of performance, scalability, availability, and fault-tolerance in a cloud environment have been studied extensively and are currently well known [1, 12]. This has given rise to a class of cloud-friendly databases, commonly referred to as NoSQL databases, which have become the backbone of cloud-based applications [3].

One of the key characteristics of cloud computing is that storage is outsourced to specialized cloud storage services [5]. Many organizations actively outsource their storage to cloud storage providers for economic reasons. However, at the same time organizations are reluctant to share their sensitive data with public cloud providers, due to limited trust and privacy issues [5, 9, 16]. This forms a main obstacle to wider adoption of cloud storage and the respective cloud storage providers [14, 23]. Therefore, cryptographically encrypting confidential data from within the *application level* is the most feasible cloud storage data security tactic and must be considered before adapting to external cloud providers.

In the context of a multi-tenant Software-as-a-Service (SaaS) application, confidentiality and privacy requirements may differ considerably between tenants. For example, different customer organizations (tenants) may impose different confidentiality requirements at any level of granularity (i.e. from full entity towards individual attributes). This form of customization must be supported at run time, in an efficient and a scalable manner.

In addressing these requirements, the SaaS provider commonly lacks data model flexibility to accommodate seamless changes at runtime. In addition, dealing with such requirements at the *application level* (i.e., in code) is not an adequate strategy, as it (i) substantially increases application complexity; (ii) inherently limits the requirements that can be accommodated; and (iii) hardcodes these requirements, as such hindering the ability to flexibly change the storage architecture. Moreover, as the communication channel between the application and the external cloud provider(s) cannot necessarily be trusted, dealing with confidentiality and privacy requirements only at the *database level* does not mitigate threats of snooping the application data in motion and requires trust in third-party cloud storage providers.

This paper, hence argues and motivates that the complex confidentiality and privacy requirements of multi-tenant SaaS applications must be realized by *application-level middleware*. We envision on leveraging the data model flexibility of columnar NoSQL databases –in terms of their ability to support dynamic columns– to accommodate different re-

SAC 2017,April 03-07, 2017, Marrakech, Morocco

http://dx.doi.org/10.1145/3019612.3019855

quirements of multiple tenants simultaneously.

The remainder of this paper is organized as follows: Section 2 provides the necessary background information and introduces a motivating example for the paper. Section 3 presents our analysis of the main requirements, while Section 4 outlines our ideas on addressing these requirements. Section 5 connects and contrasts our work with other related research in this area. Finally, Section 6 concludes the paper and indicates directions for the future work.

2. BACKGROUND

In the context of a multi-tenant SaaS application, different tenants commonly impose different requirements on the application[1], and these requirements also affect the data storage tier of the application. There is therefore a need to customize the multi-tenant SaaS application at run time to meet these different non-functional requirements.

This section first outlines and discusses a number of current and timely trends in the context of multi-tenant Software-as-a-Service (SaaS) applications. The presented research goals are motivated strongly from the context of these trends. Section 2.1 introduces the trend of offering storage as a service. Secondly, the data tier of multi-tenant SaaS applications is increasingly being structured as a *federated storage architecture*, which we discuss in Section 2.2. Then, Section 2.3 introduces the motivating example for the paper.

2.1 Cloud Storage

Cloud providers offer online and on-demand services that can elastically scale up (or down) dynamically as demand increases (or decreases). Cloud data storage is one of the prominent services of cloud providers, which predominantly allows data owners to store their data in the cloud.

From the point of view of the SaaS application provider, however, the selection of a single cloud storage provider in practice is a difficult decision: (i) application providers face a lack of trust in the cloud storage provider and are reluctant to share sensitive application data; (ii) they and their customer organizations (tenants) may require different service levels (guarantees as to data availability, performance, and responsiveness), and technologies for different types of data; and (iii) as market conditions change, they may opportunistically want to switch cloud storage providers but be confronted with a situation of *provider lock-in*.

2.2 Federated Storage Architecture

To address these concerns, SaaS providers are increasingly leveraging a combination of different cloud storage resources, technologies, and providers in a so-called *federated storage architecture*. A federated storage architecture combines different storage resources (private and cloud resources) and allows data storage needs of a single application provider to be attained by combining different cloud storage offerings. This however comes at the cost of increased complexity and maintenance, which is commonly addressed in the *application-level middleware* [2, 19, 14].

[1]Usually expressed in Service Level Agreements (SLAs)

2.3 Multi-Tenant Log Management Case

As a running example for this paper, we present a multi-tenant Log Management-as-a-Service (LMaaS) application [14, 15]. This SaaS application provides log management facilities to its customer organizations (tenants), for example, banks, supermarkets, hospitals, telecom operators, etc. The application focuses on storing large amounts of heterogeneous data: *raw log entries*, *archived logs*, *log meta-data*, *historical logs*, *incident reports*, and *time series data* and is successful in doing so by using a *federated storage architecture*, and by applying multi-tenancy, i.e. sharing these storage resources maximally among tenants.

In the case of the LMaaS application, log aggregation components are installed at the tenants' side, which collect and stream log events to the LMaaS application. Figure 1 illustrates three such events, each sent by a different tenant organization, which are stored in a single `Log` database table.

rows

ID	DeviceID	DeviceName	DeviceType	...	Tenant
1	401	BRI-Router-001	ciscortr	...	1
2	701	BRI-special-001	cisco-ace	...	2
3	301	CAN-PIX-FW-001	Pix7	...	3

Figure 1: `Log` table for storing events information.

The table holds a chunk of log data, identified by an *ID* attribute, which uniquely identifies each row in the `Log` table. The (*DeviceID*, *DeviceName*, *DeviceType*, and *...*) attributes hold information about the device that generated the log event. The *Tenant* attribute refers to the tenant for which the event is generated.

However, as mentioned above, different tenants may have different data storage requirements. As an example from the log management application, we contrast three tenant organizations, a financial agency (i.e. a bank), a medical institution (i.e. a hospital), and an SME, that each impose different requirements when it comes to data confidentiality: clearly, stricter regulations on data confidentiality apply for the financial and medical institution, as opposed to the SME.

To illustrate with the data presented in Figure 1: as the tenant with id 1 is a financial agency, even the meta-data about the device is considered highly sensitive. Similarly, as tenants with id 2 and id 3 both represent medical institutions, only a part of the data should be considered to be sensitive: for the tenant with id 2, only the (*DeviceID* and *DeviceName*) attributes hold sensitive information, whereas for the tenant with id 3, the (*DeviceID*, *DeviceName*, and *DeviceType*) attributes contain confidential information.

3. REQUIREMENTS

In this section, we introduce four key requirements for data encryption solutions in the specific context of multi-tenant SaaS applications, as introduced in the previous section.

R1. Encryption at Differing Levels of Granularity: From the perspective of data confidentiality, different tenants impose different, sometimes even contrasting confidentiality requirements, and this in turn may affect the level of granularity at which data encryption is

to be applied[2]. As such, specific data encryption support is required for applying encryption at differing levels of granularity at runtime for multiple tenants.

For example, as strict confidentiality requirements apply to the tenant with id 1, full entity-encryption is clearly the most suited option. Similarly, as tenants with ids 2 and 3 operate with relatively relaxed confidentiality requirements, instead of applying full entity-encryption, partial data encryption (i.e. only encrypting the sensitive information) is considered a more appropriate strategy.

R2. Application Tier Encryption: In a federated storage architecture that may include different public cloud providers, storage requires sending the application data over uncontrolled communication channels such as the public Internet. The lack of control and the trust in these third-party cloud storage providers force dealing with these confidentiality requirements within the application tier itself.

R3. Generic, Transparent, and Reusable Solution: Dealing with the confidentiality requirements of different tenants introduces substantial application complexity. In addition, the ability to change these requirements at run time, for example to accommodate new tenant requirements or changes in the federated storage architecture is a strong motivation to externalize this solution from the main application code.

More concretely, this requirement is the need for a generic and a reusable solution, which (i) externalizes the encryption logic from the application, accomplishing a clear separation of concerns (i.e. being able to change the encryption logic without changing the application code); and (ii) provides tenant administrators and SaaS application operators with advanced configuration and management facilities, such as data storage policies, cryptographic key management infrastructure, configuration dashboards to select different algorithms for data encryption (e.g., AES, RSA, etc.).

R4. Scalable Data Encryption: Encryption impacts application performance significantly [17], and scalability is a key concern, especially in terms of the amount of tenants and storage nodes.

The next section outlines our ideas on how to address these requirements, more specifically by leveraging the data model flexibility features of columnar NoSQL databases.

4. DISCUSSION

There is an increasing research interest in solving these requirements at the level of advanced *data access middleware platforms* [8, 11]. These solutions as such address **R2** and **R3** (partially), but not **R1** and **R4**. Especially, the rise of Object-NoSQL Data Mappers (ONDMs), which apply the principles of the widely popular Object-Relational Mapping (ORM) frameworks in a NoSQL context [6, 7, 22].

[2]Note that differences in search requirements may also affect the level of granularity at which encryption is to be applied.

The data mapping strategies (i.e. how to map in-memory object to in-table rows) built into these systems however rely extensively on the assumption of fixed database tables. This naive strategy would in our example of Figure 1 leads to the definition of three different database tables, one for each tenant, and as such these platforms do not address **R4**. A negative consequence of scattering data of the same type over different database tables for example, is that the back-end NoSQL databases (which are by design distributed databases) might fragment this data over multiple database nodes, as such negatively affecting scalability and the overall performance, for example when performing queries over all `Log` entries.

To address **R1** and **R4**, we envision more efficient data mapping strategies for Object-NoSQL data mappers (ONDMs). By leveraging the flexibility of columnar NoSQL databases (i.e. there is no fixed schema according to which data object must be structured), data objects that are encrypted differently can still be stored within a single database table. This will avoid data fragmentation across multiple database rows and multiple database nodes and will allow NoSQL databases to treat this data more efficiently.

5. RELATED WORK

Recently, there has been extensive research focusing on highlighting the security issues in NoSQL databases [13, 18, 21]. To end this, several research contributions [10, 20, 22] have been made, which provide encryption support for NoSQL databases at different levels to protect outsourced data.

The existing solutions [8, 11] in the state-of-practice to support encryption at the *middleware level* either (i) offer limited support for encryption where attributes with only specific data types can be encrypted, or (ii) provide solution-specific data types to be used in the application to encrypt sensitive data. Moreover, they operate on a fixed data model and provide no flexibility to support encryption at various levels of granularity.

A number of libraries are available in different programming languages, achieving encryption support in the *application level*. For example, one of the easiest ways to encrypt sensitive data in Java is by using custom data types provided by the Java simplified encryption (Jasypt) [8]. However, these libraries have various limitations: (i) they need to be configured to specify sensitive data during deployment time; and (iii) they do not support encryption at various-levels of granularity, which can also be altered during run-time.

6. CONCLUSION

Outsourcing data to third-party cloud storage providers offer a wide array of clear benefits over hosting data in costly on premise data centers. In practice, however, data confidentiality considerations often prohibit outsourcing confidential application data to external and often untrusted storage providers.

This paper motivates that data confidentiality considerations of multi-tenant SaaS applications must be supported within the *application-level middleware* by utilizing NoSQL databases. We envision to create an efficient data mapping

strategy that leverages the flexibility of columnar NoSQL databases such as Apache Cassandra to support efficient, dynamic, and scalable data encryption that can be enacted at different levels of granularity.

This work fits into our ongoing research on application-level middleware for federated data storage architectures in support of multi-tenant SaaS applications. This is an ongoing research, future work involves the implementation of the proposed data mapping strategy and validate it in a prototype.

Acknowledgments This research is partially funded by the Research Fund KU Leuven (project GOA/14/003 - ADDIS), the SBO DeCoMAdS project, and the iMinds SeClosed project.

7. REFERENCES

[1] D. Agrawal, S. Das, and A. El Abbadi. Big data and cloud computing: Current state and future opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 530–533, New York, NY, USA, 2011. ACM.

[2] D. Bermbach, M. Klems, S. Tai, and M. Menzel. Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 452–459, July 2011.

[3] G. DeCandia et al. Dynamo: amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[4] K. Grolinger, W. A. Higashino, A. Tiwari, and M. A. Capretz. Data management in cloud environments: Nosql and newsql data stores. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1):1, 2013.

[5] J. Hu and A. Klein. A benchmark of transparent data encryption for migration of web applications in the cloud. In *Dependable, Autonomic and Secure Computing, 2009. DASC '09. Eighth IEEE International Conference on*, pages 735–740, Dec 2009.

[6] M. Huber, M. Gabel, M. Schulze, and A. Bieber. *Cumulus4j: A Provably Secure Database Abstraction Layer*, pages 180–193. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[7] Impetus. A JPA 2.1 compliant Polyglot Object-Datastore Mapping Library for NoSQL Datastores. https://github.com/impetus-opensource/Kundera, 2016. [Last visited on December 02, 2016].

[8] Jasypt. Java Simplified Encryption. http://www.jasypt.org/, 2016. [Last visited on July 19, 2016].

[9] L. M. Kaufman. Data security in the world of cloud computing. *IEEE Security Privacy*, 7(4):61–64, July 2009.

[10] L. Liu and J. Gai. A new lightweight database encryption scheme transparent to applications. In *2008 6th IEEE International Conference on Industrial Informatics*, pages 135–140, July 2008.

[11] K. Lorey, E. Buchmann, and K. Böhm. TEAL: Transparent Encryption for the Database Abstraction Layer. In *Proceedings of the CAiSE 16 Forum at the 28th International Conference on Advanced Information Systems Engineering*, New York, NY, USA , 2016.

[12] S. Malkowski et al. Empirical analysis of database server scalability using an n-tier benchmark with read-intensive workload. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1680–1687, New York, NY, USA, 2010. ACM.

[13] L. Okman et al. Security issues in nosql databases. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 541–547. IEEE, 2011.

[14] A. Rafique, D. Van Landuyt, B. Lagaisse, and W. Joosen. Policy-driven data management middleware for multi-cloud storage in multi-tenant saas. In *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*, pages 78–84, Dec 2015.

[15] A. Rafique, D. Van Landuyt, B. Lagaisse, and W. Joosen. On the performance impact of data access middleware for nosql data stores. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2016.

[16] A. Rafique, S. Walraven, B. Lagaisse, T. Desair, and W. Joosen. Towards portability and interoperability support in middleware for hybrid clouds. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 7–12. IEEE, 2014.

[17] S. Q. Ren, S. H. Zhang, Y. Z. Chen, M. R. Felipe, Y. J. Ha, and K. M. M. Aung. Empirical study of accelerating data protection for multi-tenant storage. *Advances in Information Sciences and Service Sciences*, 5(13):19–25, 08 2013.

[18] A. Ron, A. Shulman-Peleg, and A. Puzanov. Analysis and mitigation of nosql injections. *IEEE Security Privacy*, 14(2):30–39, Mar 2016.

[19] S. Seshadri, L. Liu, B. F. Cooper, L. Chiu, K. Gupta, and P. Muench. A fault-tolerant middleware architecture for high-availability storage services. In *IEEE International Conference on Services Computing (SCC 2007)*, pages 286–293, July 2007.

[20] V. Sidorov et al. Transparent data encryption for data-in-use and data-at-rest in a cloud-based database-as-a-service solution. In *2015 IEEE World Congress on Services*, pages 221–228, June 2015.

[21] D. S. Terzi, R. Terzi, and S. Sagiroglu. A survey on security and privacy issues in big data. In *10th International Conference for Internet Technology and Secured Transactions*, pages 202–207, Dec 2015.

[22] X. Tian, B. Huang, and M. Wu. A transparent middleware for encrypting data in mongodb. In *Electronics, Computer and Applications, 2014 IEEE Workshop on*, pages 906–909, May 2014.

[23] T. Waage and L. Wiese. *Foundations and Practice of Security: 7th International Symposium, FPS 2014, Montreal, QC, Canada, November 3-5, 2014. Revised Selected Papers*, chapter Benchmarking Encrypted Data Storage in HBase and Cassandra with YCSB, pages 311–325. Cham, 2015.