A Middleware Proposal Towards The Compliance of Small Business Databases with the LGPD

Yasmin Maria Zerbielli*, Vinicius Begnini Felicio*, Cristian Solutchak*, William Kunzler*, Gustavo Schwitzki Peretti*, Glória de S. P. Ozório†, Walter Priesnitz Filho† and Heitor Scalco Neto*
*Instituto Federal Catarinense, Concórdia, Santa Catarina, Brazil
Corresponding e-mail: zyasminmaria@gmail.com
†Universidade Federal de Santa Maria, Santa Maria, Rio Grande do Sul, Brazil
Corresponding e-mail: walter@redes.ufsm.br

***Abstract*—This proposal introduces a middleware development solution aimed at enhancing data security measures for small databases, thereby improving data protection compliance with the Brazilian General Data Protection Law (LGPD). The middleware uses MySQL Proxy to intercept traffic between the application and the database. From the data obtained with MySQL Proxy, the middleware extracts the characteristics of the query and applies AES encryption. This proposal aims to make this adjustment process transparent to the user without needing changes in the database or application. The obtained results validate the proposal for applying symmetric cryptography for the data previously stored in the database and new MySQL operations.**

***Keywords*—Software Architecture; LGPD; Cryptography.**

I. Introduction

From the application of the Brazilian General Law for the Protection of Personal Data (LGPD[1]), it is observed that several institutions face the challenge of adapting their systems to guarantee the confidentiality, availability, and integrity of data. In this sense, the LGPD represents a substantial change in the way companies and organizations handle their customers' or users' information, requiring the application of stricter measures to prevent data leaks and protect the privacy of individuals [1].

However, many companies, especially small ones, need help implementing effective data security measures to prevent data leakage. Such difficulties occur due to budget constraints and a need for specialized technical knowledge. In this context, it is crucial to find accessible and practical solutions that allow small companies to protect their users' data, seeking to comply with the LGPD.

Based on this context, together with a MySQL Proxy, a script was developed in Lua[2]. The middleware uses the language Lua due to its effectiveness in performance and agility in development. Its portability allows adapting to different operating systems with few code changes. Furthermore, it is crucial to highlight LuaJIT[3], a *just-in-time* (JIT) implementation of the language, widely recognized for its substantial performance improvements in comparison with the default implementation [2].

This work presents an approach that integrates a MySQL Proxy, a script in Lua with a symmetric cryptography algorithm, to create a middleware[4]. The proposed resource aims to provide companies with the possibility of data protection via encryption without the need to change or even replace the business management system.

MySQL Proxy acts as an intermediary between the database and the application, providing an additional layer of security by protecting access to data. From this middleware, it is possible to manipulate database queries/operations, maintaining an encrypted database according to the sensitivity of the data contained in each table.

Symmetric encryption algorithms change sensitive information into an encrypted format for unauthorized users, thus ensuring that the data remains scrambled even in the event of leaks. This work presents a data protection solution aimed at small companies/systems. The successful integration of middleware, using MySQL Proxy with the script for handling SQL commands developed in Lua, brought significant results in terms of security.

This work is organized as follows: Sections II and III present general concepts about LGPD, symmetric cryptography, and the AES algorithm (Advanced Encryption Standard). Section IV presents MySQL Proxy concepts. In Section V, the materials and methods used for the development of this work are discussed. Finally, in Section VI, the conclusions reached so far are presented, as well as the next steps and expected results for the future.

[1]In Portuguese: Lei Geral de Proteção de Dados Pessoais
[2]https://www.lua.org/about.html
[3]https://luajit.org/luajit.html
[4]Available at: http://github.com/labsep/privacy-shield-mysql-proxy

II. Brazilian General Data Protection Law - LGPD

Approved on August 14, 2018, the Brazilian General Law for the Protection of Personal Data (LGPD) aims to regulate the processing and collection of personal data, including through digital means. Although the LGPD has been approved, specific regulations and detailed guidelines from the National Data Protection Authority (ANPD[5]) remain experimental. The law aims to protect fundamental rights, such as freedom, privacy, and free development of the natural or legal person, public or private. Concerning companies, the LGPD aims at greater reliability in data management, facilitating the analysis and control of personal and business data to minimize the impacts caused by data leaks. It is worth mentioning that the law applies to data management for economic purposes. In the event of non-compliance with the law, data processing agents are subject to some administrative sanctions [3].

III. Symmetric Encryption

Symmetric encryption algorithms use a single key for the encryption and decryption process, meaning only the sender or the recipient of a message can decrypt the encrypted content [4]. Furthermore, due to the use of a single key in the encryption process, symmetric algorithms provide greater agility when compared to asymmetric algorithms, which require the use of two keys for communication. Examples of symmetric encryption algorithms include the Data Encryption Standard (DES), the Triple Data Encryption Standard (3DES), the Blowfish, the Cast-128, the Advanced Encryption Standard (AES), the Salsa20 and the RC2 [5]. As mentioned in [1], through tests involving algorithms described previously, it was concluded that the most effective symmetric algorithm is the AES. The analysis of the function calls and the necessary time to perform the encryption and decryption process revealed a superior performance of this algorithm in both aspects, even after the predefined values gradually increased. Therefore, the algorithm chosen for the experiments in this work was AES.

A. Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES), works with 128-bit data blocks and can use 128, 192 and 256-bit keys. To execute the encryption, four stages are required: AddRoundKey, SubBytes, ShiftRows and MixColumns [6]. The algorithm is known for its resistance to cryptanalysis attacks, being used in several applications. It is also notable for being selected by the National Institute of Standards and Technology (NIST) as a secure encryption standard. The AES used in the proposal is already found natively integrated in MySQL, and can be used with the functions AES_ENCRYPT and AES_DECRYPT. The default method used by MySQL is the AES-128-ECB [7].

[5]In Portuguese: Autoridade Nacional de Proteção de Dados

IV. MySQL Proxy

The MySQL Proxy is a tool that acts as an intermediary between the clients and the MySQL server. This communication occurs through a MySQL network protocol. Its function is to receive connection from clients and redirect them to the appropriate servers, helping to optimize the functioning of a database. A feature of MySQL proxy is the ability of intercepting and modifying Structured Query Language (SQL) commands sent by the clients before they are sent to the MySQL server. This allows the implementation of the additional logic, as caching, power balancing, data partitioning, encryption and decryption, among other possibilities [8].

V. Development

Recognizing the problem of compliance with LGPD in small companies, the proposed middleware intercepts the communication between the management system and the database to execute the process of encryption with the data that the system administrator deems necessary. In this section, the utilized methodology will be discussed in detail.

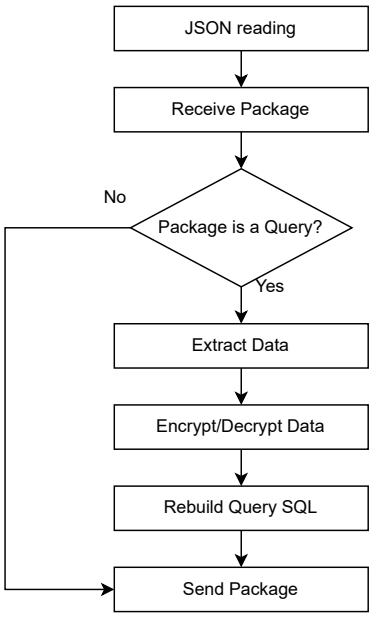

Figure 1. Middleware Operation

The operation of the developed middleware is explained from Figure 1. Initially, the system reads the encryption settings, which are stored in a JavaScript Object Notation (JSON) file automatically generated when the middleware is connected to

MySQL at first time, offering the administrator the options to choose which columns will be considered necessary to be encrypted, from an interface. An example of this file can be seen in Figure 2.

```
local DATABASE_CONFIGURATION = {
    name = "labsep",
    tables = {
        {
            name = "users",
            columns = {
                {
                    name = "name",
                    type = "VARCHAR(255)",
                    encrypt = true
                },
                {
                    name = "cpf",
                    type = "CHAR(11)",
                    encrypt = true
                },
            }
        }
    }
}
```

Figure 2. Example of generated JSON file

When the middleware is initialized, the *JSON* file is read and stored in a Lua *table*[6]. Then, packets that may include a string containing diverse data are captured. These data can range from a query to a notification informing that a database switch occurred. When a packet is captured, a verification is performed to determine if the captured packet is a query; if not, the packet is redirected directly to the server. However, if the packet is a query, another verification will be performed to certify the existance of a connection with the database.

Posteriorly, the step of selecting database encryption configuration begins. It is performed by comparing the database name described in the JSON file with the database name being used by the client. After that, the SQL query is extracted by string manipulation techniques, resulting in a new Lua table. Then, the middleware verifies if the query type is processable. If the result of this verification is positive, a second verification is performed to identify the specific query type, presenting the possibility of it being a write or read query. Depending on this result, different paths will be followed.

If a read query is identified, the middleware extracts the data from the table and selects the appropriate encryption configuration. Then the columns are decrypted and normally converted back to their original type. An example of a correspondent SQL code to this operation is presented below:

```
SELECT
 CAST(AES_DECRYPT(CPF, 'EncryptionKey') AS CHAR(11)),
 CAST(AES_DECRYPT(AGE, 'EncryptionKey') AS INT))
FROM patients
WHERE
 CAST(AES_DECRYPT(AGE, 'EncryptionKey') AS INT) >= 18
```

It can be observed that, since the query is a read query, the command will decrypt the requested data to display it to the requester.

If the identified query is a write query, the processes executed are different from the ones executed with read queries. Initially, the middleware extracts the data from the string received and selects the encryption configuration to the table in question. Due to the fact that the middleware only encrypts data categorized as CHAR, the next step consists of converting the encrypted values to this specific data type. After that, the data are encrypted. As an example of an INSERT command processed after encryption, a code example can be seen below:

```
INSERT INTO patients (CPF, AGE) VALUES (
    AES_ENCRYPT('101.202.303-90', 'EncryptionKey'),
    AES_ENCRYPT(CAST(17 as CHAR), 'EncryptionKey')
);
```

The approach for other write commands (DELETE and UPDATE) follows the same pattern as the code above.

It is worth noting that, regardless of the type of the identified query, the process converges to a single path. After the specific operations of each type of query are concluded, the middleware checks if the query contains a condition (WHERE clause). If the query does not include any condition, it is redirected to the compilation process. Otherwise, the data related to the condition are extracted, the columns are decrypted and converted to their original type. Posteriorly, the query is compiled and the old query is replaced by the new one. Finally, the packet is sent to the application.

With the process completed, the times measured in the end of operations were observed. A Dell Inspiron 7599 notebook was used, containing an Intel(R) Core(TM) i5-7300HQ processor, with a CPU @2.50GHz, with 8GB of RAM memory. The time differences between native MySQL and MySQL Proxy, with and without encryption, can be seen in Figure 3. The graphic illustrates the time execution, in milliseconds, of different operations performed in a MySQL database system. The native version of MySQL is established as a baseline for each of these operations, being considered a benchmark in terms of performance. The main objective of the graphic is to demonstrate the comparison of the perfomance of the native version, with and without encryption, against the performance

[6]Map-like data structure, known in other programming languages.

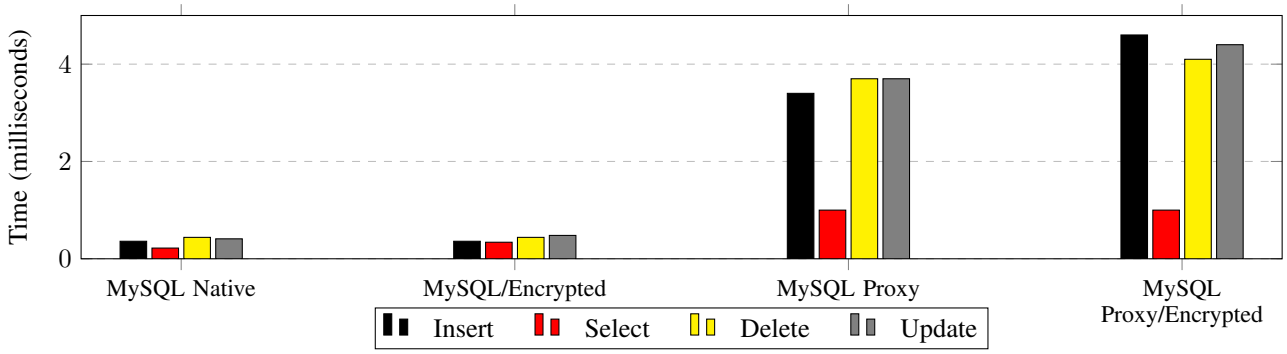

Figure 3. Comparisons between execution times

of MySQL Proxy, also with and without encryption. It should be noted that the values presented in Figure 3 come from the arithmetic averages of time obtained by repeating the tests 10 times.

After analyzing Figure 3, it was possible to observe notable time disparities between the native version of MySQL and the MySQL Proxy, especially in UPDATE, DELETE and INSERT commands. The SELECT command in MySQL Proxy, despite obtaining higher times compared to native MySQL, was the one that came closest to the reference times, both with encryption and just with MySQL Proxy. The commands that presented higher discrepancy with the reference time were UPDATE and DELETE in MySQL Proxy without the implementation of an encryption. It is believed that the high time taken by the operations of these commands occurs due to the methods of string processing implemented in the script developed in Lua. However, it is important to notice that this is an effective and cost-free solution to institutions that do not have specialized technical professionals able to adapt their systems to LGPD.

VI. Conclusions

Given what has been shown, it is possible to identify that compliance with LGPD has been a significant challenge to small businesses. The proposal presented in this article aims to assist in this process of control and processing of sensitive personal data, with the objective of intermediating a management system and one or more databases, providing the definition of which data will be encrypted, based on the choice from the administrator. However, it is possible to identify that this feature helps the company adapt to legislation, and consequently increases the privacy and necessary protection of sensitive data entered into the management system, without requiring manual modifications to the system and database. It is worth noting that to comply with the LGPD, compliance with many other actions is necessary in addition to the proposed middleware.

As a performance discrepancy between native MySQL and MySQL Proxy was observed, as it is shown in Figure 3, improvements are needed in this regard. Therefore, as proposals for future works, it is intended to utilize other programming languages, as C, C++, Go and Python, also comparing the execution times obtained in the usage of each one. As in Lua, Python also contains tools to improve its performance, as PyPy[7]. In addition to conducting performance tests in larger databases, another need for future work is the implementation of a function capable of changing the database's encryption key in the event of a supposed key leak.

References

[1] Y. Maria Zerbielli, I. Karine Maziero Marchese, M. Amélia Mafessoni Herpich, W. Priesnitz Filho, and H. Scalco Neto, "Protection of personal data in health using symmetric encryption: a comparative study between different algorithms," *Concilium*, vol. 23, no. 5, p. 199–214, mar. 2023. [Online]. Available: https://clium.org/index.php/edicoes/article/view/1064
[2] Lua, "Luajit," 2023. [Online]. Available: https://luajit.org/luajit.html
[3] Brasil, "Lei nº 13.709, de 14 de agosto de 2018. dispõe sobre o tratamento de dados pessoais [...]." Brasil, Brasília, DF, 2018. [Online]. Available: http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm
[4] W. Stallings, "Criptografia e segurança de redes princípios e práticas, ch. 6," 2006.
[5] M. Al-Shabi, "A survey on symmetric and asymmetric cryptography algorithms in information security," *International Journal of Scientific and Research Publications (IJSRP)*, vol. 9, no. 3, pp. 576–589, 2019.
[6] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback, and J. F. Dray Jr, "Advanced encryption standard (aes)," 2001.
[7] MySQL, "Mysql documentation," 2021. [Online]. Available: https://dev.mysql.com/doc/
[8] ——, "Mysql proxy documentation," 2021. [Online]. Available: https://downloads.mysql.com/docs/mysql-proxy-en.pdf

[7]https://www.pypy.org/