Integrity: An Object-relational Framework for Data Security

Elder Costa, João Pedro Lorenzo De Siqueira, Carlos Eduardo Pantoja\textsuperscript{a} and Nilson Mori Lazarin\textsuperscript{b}

Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (Cefet/RJ), Rio de Janeiro, RJ, Brazil

Keywords: Software Engineering, Cryptography, Database.

Abstract: Considering the recent laws that discuss how user data must be collected, treated, stored, and protected, designing and developing projects considering the information security in application systems is necessary. Given that cryptographic functions available in database management systems are limited to some data types, this work proposes an object-relational framework to add data security using a process to mask data in the persistence layer of a layered application.

1 INTRODUCTION

Recent scandals involving cyberattacks, such as data leaks of more than 540 million Facebook users and Twitch streamers’ billing, have become increasingly recurrent and show that information security plays a significant role in protecting sensitive information from your users.

The growing number of internet users, the services digitization, and the need to ensure the security of users’ sensitive data have led governments worldwide to reformulate, approve and enact new laws that address these issues. The General Data Protection Regulation (GDPR), the California Consumer Privacy Act of 2018 (CCPA), and the Personal Information Protection and Electronic Documents Act (PIPEDA) are some examples of current laws in Europe and North America. These legislations pressure companies to develop security and protection solutions for the data that travels through their systems and are susceptible to the most varied types of attacks.

Regarding data storage, some Database Management Systems (DBMS) such as MariaDB/MySQL, PostgreSQL, Oracle or SQLServer, provide native encryption functions, such as AES\_ENCRYPT, pgcrypto, DBMS\_CRYPTO or ENCRYPTBYKEY. However, all these DBMS have some limiting factor in providing the information security of the stored data due to the cryptography functions that are restricted to certain types of data, not covering types such as date, int or decimal. Thus, these systems do not provide security for data that may contain sensitive information about a user or company, such as date of birth or a customer’s cash balance, for example. There are also limitations in services that do not work on data that are in production like (Zhang et al., 2018), which offers the data masking.

This work aims to present Integrity, an object-relational mapping framework for information systems development that presents a model based on combinations of existing cryptographic algorithms to mask the data. Integrity gets application data and performs data masking on selected columns using a predefined application-level and design-time key defined by the developer team. In this way, the data is stored masked in the relational database. The data will only make sense when the inverse process is carried out at the application level by Integrity. With this other data types like date can be encrypted.

The Integrity framework was developed using the C# programming language and tested through a proof of concept to demonstrate the assertiveness of its main features. The main contribution of this work is an object-relational framework to encrypt varchar data and mask date data in relational databases.

This work is structured as follows: Section 2 explains the basic concepts and algorithms needed to understand how Integrity works. In Section 3, some related works are discussed. In Section 4, the framework and its implementation are presented. In Section 5, a case study is presented, showing how the framework works, and, finally, in Section 6, the conclusions are presented.
2 THEORECTICAL REFERENCE

Information Systems Security is the set of actions that protect information systems and their stored data meeting three fundamental principles: Confidentiality, where only authorized users can access the stored information; Integrity, in which only authorized users can change the information stored; Availability, in which the information must be available whenever necessary (Kim and Solomon, 2014).

Encryption is one of the most used methods to provide Confidentiality to data in information systems. This technique consists of processing unencrypted data, operating at the bit level, combined with a key, producing an output called ciphertext without linguistic cohesion. This ciphertext can be transmitted or stored in insecure channels or media because if this information is stolen, it cannot be viewed by an unauthorized user (Tanenbaum, 2003)(Kim and Solomon, 2014).

2.1 AES Key Expansion

Advanced Encryption Standard (AES) is a symmetric iterated block cipher in which the processed plaintext block size is 128 bits, and the cryptographic key can vary from 128, 192, or 256 bits. The algorithm’s internal operations are performed on a byte array called state that has four rows and four columns. The given key defines the number of iterations to be performed on the state array. A subkey, generated through an expansion process, is used at each iteration of AES (Dworkin et al., 2001).

During the expansion process, the received key is copied to the first four words as shown in the general algorithm of Figure 1, and the expanded keys are filled with four words of each turn. Each word included $W_i$ depends on the previous word, $W_{i-1}$, and the word four positions behind, $W_{i-4}$. For each word in the array where the position is not a multiple of 4, only one XOR operation is performed, otherwise it is submitted to a function. The function illustrated in Figure 1, has three processes, according to (Stallings, 2015):

- RotWord: Byte-by-byte circular shift to the left.
- SubWord: A byte-by-byte substitution of the word is performed using the AES substitution table, available in (Dworkin et al., 2001).
- RCon: The result of steps 1 and 2 goes through a logical operation XOR with the round constant, represented in hexadecimal, where: $RC_0 = [01]$; $RC_2 = [02]$; $RC_3 = [04]$; $RC_4 = [08]$; $RC_5 = [10]$; $RC_6 = [20]$; $RC_7 = [40]$; $RC_8 = [80]$; $RC_9 = [1b]$; $RC_{10} = [36]$.

Figure 1: AES Key Expansion, adapted of (Stallings, 2015).

2.2 Feistel Networks

According to (Stallings, 2015), the Feistel Cipher is an encryption/decryption algorithm, which takes as input a key and a block of bits called plaintext. This block is divided into two halves, $L_0$ and $R_0$. These halves go through rounds of processing and then combine to produce the ciphertext. The blocks produced in the previous round and a subkey derived from the initial key are used as input for each processing round. All rounds follow the same pattern, where a replacement is performed on the left half using an XOR operation with the right half of the operation associated with a predefined function using the round key. After all the described operations, a permutation is made between the blocks where the right side becomes the left side and vice versa, thus forming a substitution-permutation network.

2.3 Secure Hash Standard

Hash functions are iterative, one-way algorithms that can process a variable-length input message to produce a fixed-size, condensed representation called digest. These algorithms work on the Integrity pillar since they determine the integrity of a message. Any change in the input message will, with high probability, result in a different digest. The SHA-256 algorithm deals with messages of up to $2^{64}$ input bits, providing a fixed output of 256 bits (Dang, 2015).
3 RELATED WORK

Transparent Encryption for the database Abstraction Layer (TEAL) (Lorey et al., 2016) says that data security is guaranteed as there is no need for the storage service provider to obtain data encryption keys. The Database abstraction layer is a way of translating different specific commands of the existing DBMS on the market.

Likewise, DBCrypto (Deshpande et al., 2012) encrypts data by rewriting queries using the Vigenère cipher, a form of simple sum-of-numbers encryption. The keys are stored in the application, and, therefore, whoever has access to the data in the database will not be able to know their real values. There is a different operation for each type of query. At each call, the query changes to adapt to the encrypted database.

In LGPD Compliance (Pitta et al., 2020), the authors’ main objective was legacy systems using a technique to perform a dump of the original base and change the structure of the tables, changing the types of stored data and later applying AES for disk persistence. This technique also employed data tables in primary memory with data obfuscation to speed up queries. They use query modifier codes to store data inside the database. However, DBCrypto searches the encrypted database only for specific data types.

In this work, we use obfuscation for date data type. Different from TEAL, our proposed approach allows comparisons with an obfuscated date without the need to decipher, consequently speeding up query processing in this case. Because it is not necessary to decipher all table data to perform the comparison with the date in plain text, like (Lorey et al., 2016).

In this work, we use Advanced Encryption Standard to cipher data type varchar different from DBCrypto our proposed approach uses a modern cryptographic algorithm. Furthermore, (Deshpande et al., 2012) uses a cryptographic method that has already been broken and has fallen into disuse for a long time, compromising the effectiveness of the security applied.

Finally, this work presents a data obfuscation process for disk persistence keeping the column’s original data type, unlike LGPD Compliance. (Pitta et al., 2020) transforms all tables in the database on blob data type to storage on the hard disk, and part of the unencrypted database is mirrored in a temporary database in primary memory.

4 INTEGRITY FRAMEWORK

This work presents an alternative for protecting data types that are not supported by the native cryptographic functions of the main DBMS. Situated in the data persistence layer, at the application level, the proposed model acts on a pillar of information security not conventionally explored, Integrity. According to (Hintzbergen et al., 2018), Integrity refers to the consistency and accuracy of the intended state or information, and any unauthorized modification of data is a violation of data integrity. In this way, the proposed model acts by deliberately manipulating the integrity of the data to be stored.

A scenario where a buyer interacts by entering some personal and sensitive data on a large retailer’s purchase site is proposed to illustrate the functioning of the Integrity framework. Bob accessed a large retailer’s site to purchase a product he had been looking for in a while. Bob was redirected to the payment page upon selecting his product and placing it in the shopping cart. Bob then enters his personal and credit card details for payment. The system asks if Bob wants to keep his card details registered in the system. As Bob wants to make future purchases easily and quickly, he confirms his credit card details into the system. The system then stores the name printed on the card, the expiration date, and Bob’s secret card number. However, after a while, due to an internal security breach, the database with the card information of several users was leaked, exposing Bob’s data.

The Integrity framework can be used at the application level as an object-relational mapping framework where data is already encrypted in the database to avoid data exposure in these situations. In this way, once the data is sent from the model layer to the data persistence layer, Integrity performs an object-relational mapping, transforming the object model into a relational model preparing it for storage in a database. After this, the Data Masking process obfuscates the data using Feistel and AES Key Expansion ciphers in the case of Date fields. If the data is of String type, the data is encrypted. Next, Integrity SQL (ISQL) automatically assembles the SQL query for the data store. In case of a query to retrieve encrypted or masked information from the database, ISQL does the opposite process, taking the encrypted String data and leaving the text unencrypted.

However, in the case of the Date type, as there is a maximum representation limitation, masking can leap. Then, ISQL builds the query considering the finite body size. Therefore, when using Integrity, even if the data were exposed, it would not make any sense to whoever obtained it. Figure 2 shows the architec-
ture of the Integrity framework. For example, if the retail company system adopts Integrity when registering the name Bob and the credit card expiration date of 2022-01-01 the framework would encrypt the name and obfuscate the expiration date by taking a leap in the finite body, storing the information in the database with the values X5rMIOVrdE0BqjXKii3VvA== and 8806-12-14.

4.1 Data Masking

The Data Masking process, represented in Figure 3, receives the data to be manipulated, a 128-bit key defined by the developer in the system’s source code and an automatically generated seed. This seed is derived from the initial seed, the table structure, and the column chosen to be encrypted. Consequently, the aim is to increase entropy and make difficult an attack by frequency analysis since each table column will use a different seed in the masking process.

The Data Masking process follows the following steps:

- The encryption key \([K]\) is submitted to the AES Key Expansion routine (Dworkin et al., 2001), generating 10 128-bit subkeys \([K_1], [K_2], ..., [K_{10}]\).
- The seed is submitted to SHA-256 (Dang, 2015), generating a fixed-length 256-bit output that is later split into two 128-bit parts, \([R_i]\) and \([L_i]\).
- The two parts of the SHA-256 digest and the 10 subkeys are subjected to a 10-round Feistel Network.
- Each round, a logical operation XOR is performed between the round subkey \([K_i]\) with the 128bits \([R_i]\) right side.
- The output of this operation becomes the left side of the next round \([L_{i+1}]\) besides it is operated again (XOR) with the left side of the \([L_i]\) round, generating the right side of the next round \([R_{i+1}]\).

The output of the Feistel Network generates a 256bit Token that is submitted to a function. For data masking of the date the token into an Integer, representing the amount of “leaps” that will be added to the data to be obfuscated unsing the following: \(\text{DataMasked} = \text{DataClear} + (\text{Token} \mod N)\).

The output will be when the same input (key and seed) enters the algorithm’s tokenization process. In this way, the data restoration process takes place by reducing the Data Obfuscated by the Token, as follows: \(\text{DataClear} = \text{DataObfuscated} - (\text{Token} \mod N)\).

For data encryption of the varchar type, the AES algorithm is used because native functions such as AES_ENCRYPT() require varbinary or blob. In this case, the 256bit token is used as the encryption pass-
word (128bit MSB) and initialization vector (128bit LSB).

4.2 Implementation

Integrity can be used to extend an application’s persistence layer, implemented as a library. It is based on three main features:

• Object-relational Mapping: represents database tables in the application as objects so that entropy generation is possible using the attributes defined in the class combined with the key also configured at the application.

• Data Persistence: must be able, through object-relational mapping, to create commands and queries for persistence in the database.

• Security and Obfuscation: it obfuscates and stores the data integrity using the algorithm presented in section 4.1, considering commands created for data persistence and the entropy generated by the object-relational mapping.

This version\(^1\) was built in the C# language using the .NET Core 5.0 development platform and the Visual Studio IDE. The development considered only the MySql DBMS and only dealt with the Datetime and varchar data types. The Integrity class diagram can be seen in Figure 4. Integrity is structured in three class libraries:

• Core, which contains the main classes for the library functioning.

• Attributes, which contain everything related to the classes’ attributes for the library’s object-relational mapping.

• Lib, which contains useful classes for implementing data persistence and security.

The Core class library is the core of the package. It contains the main classes for configuration, relational mapping, base repository, and security, namely:

• IntegrityConfiguration is responsible for the initial configuration of the application by invoking the ConfigureIntegrity method. It invokes the other major libraries’ methods to store the database Connection String, the Cipher Key, and the Entropy Key.

• IntegrityBaseRepository is the abstract class that must be extended by the application’s persistence layer classes and provides the methods developed for integrity functioning for persistence and data reading.

• IntegrityMapperRepository is the class responsible for the object-relational mapping of the library. It is responsible for creating the insert, delete, update, and select commands. In addition, when using the methods of the IntegritySecurityQueryLanguage class, it defines the parameters to be saved in the database.

• IntegritySecurityQueryLanguage is the class that has the definitions and encryption algorithm of the library and this approach. It provides the methods for obfuscating the data and recovering the original data.

The Attributes class library has everything related to object attributes for implementing the library’s object-relational mapping and contains the following classes:

• AttributesUtils is the class that contains utilities for manipulating the attributes defined in the entities, such as creating a parameter dictionary returning a list of attributes from the object’s properties.

• Column is the column attribute that captures and relates the column name in the database to the object’s properties.

• Key is the attribute that defines a specific property as a table key.

• Table is the attribute that defines the table’s name that refers to the object in use in the application.

All attributes listed are mandatory for the library to function. It is possible to create the database’s queries and commands from them. They are defined in the application’s Entity classes. The Lib class library contains useful classes for implementing data persistence and library security, as following:

• IntegrityDatabaseUtils has utilities for saving the database connection string used by the application to create a connection to the DBMS.

• MysqlConstants has constants defined by the DBMS to be considered in the data leap according to the field limit of the employed field, as is the Date case.

• AesService is the class that implements the 128-bit AES algorithm, which provides the key expansion methods used for token generation. We chose to use AES-128 CBC because the token output is 256bits divided into two parts, one for the initialization vector and one for the encryption key.

The sequence diagram interconnecting all the messages exchanges by objects in the Integrity framework can be seen in Figure 5.

\(^1\)https://gitlab.com/eldercosta/integrity-csharp
5 CASE STUDY

A case study is presented based on the scenario proposed in Section 4, where Bob and Alice register their data on a retailer’s platform to show how Integrity works. The case study will be conducted from the developer’s perspective on how Integrity works on the backend. The case study will store the customer name (client_name varchar(256)), registration date (clientSince date), cardholder name (cardHolder_name varchar(256)), card number (card_number varchar(256)), and expiration date (cardExpiration date) in a database. The fields using varchar and date will be encrypted and obfuscated.

From the developer’s point of view, an initial password (EncryptKey=0123456789ABCDEF) and an initial seed (EntropySeedString=IntegrityExample) are defined, which will be used in the obfuscation and encryption of the application data, as shown in the Figure 6.

Afterward, it is created a class in the business model layer named ClientEntity with five columns, as shown in Figure 7.

Each column of the table receives a different token, generated from the EncryptKey and EntropySeedString defined in the configuration file. Then, when the data is persisted in the database, the names and dates will be different. It makes difficult frequency attacks. The token generation process is presented in Table 1.

When the system is in production, the client_name Bob and Alice will be persisted as X5rMlOVrdE0BqjXKii3VvA== and bnRP-
To calculate the leap used in the obfuscation of Date fields, the token \((eaeb8282559672c919a1933c6515c2767bf68d1b0308672f552bde49ca170)\) generated for the field clientSince was used. The Token is divided into 64-bit words. A jumpToken is obtained through the exclusive-or operation between each word of the token. The jumpToken is converted to a 64-bit unsigned Integer. The mod is calculated with \(N\), in this case, 3652060, which is the day limit of the date field in MySQL. The result \((304138)\) is the number of days to be added to the date. The process is shown in Figure 9.

For \textit{varchar} fields, the AES-128 algorithm is used in CBC operating mode. The most significant bits of the generated token are used as the key and the least significant bits are used as the initialization vector. In the case of the field client\_name, which received the token \((2cbacfc2856c26b617b2533ef91d7846b4240a3719c31de68fd87e109ca9f0)\), it uses:

- key = \([2cbacfc2856c26b617b2533ef91d7846b4240a3719c31de68fd87e109ca9f0]\)
- iv = \([b424f0a3719c31de68fd87e109ca9f0]\)

When the system retrieves information from the database, the Integrity framework does the inverse
process, considering the columns’ annotations and seeds defined in the source code. Then, the data is retrieved unencrypted or unobfuscated. Integrity was designed to make all the encryption and data obfuscation processing transparent to the system developer and user. Figure 11 shows the recovered data.

When a query with date comparison is performed, the informed date also goes through the leap process before the query is performed, so there is no need to deobfuscate all dates in the column to perform the comparison.

6 CONCLUSIONS

This work presented a framework for object-relational mapping that aims to obfuscate the information in the database, leaving the responsibility of the cryptographic process to the application layer instead of delegating this process to the database management system. Likewise, the framework aims to make the entire process transparent for the developer, who only needs to interfere in the solution’s source code to define passwords and which columns will suffer data obfuscation. In this way, developing software considering the new data protection laws is facilitated.

The data will be stored in the database in an obfuscated way. The data that Integrity has persisted will be meaningless to those who illicitly obtain it if there is an information leak. Not even the Database Administrator will have access to such information since the system development team defines the password and the name of the attributes used in the cyphering process.

As future works, Integrity will be part of the Spring framework in Java to take advantage of all the infrastructure already available and for its wide use in the market. Other types of fields will also be used in Integrity, such as the Integer since only String and Date data are currently being used in this version. We opted for working firstly in two data types to reduce the scope of this work. Though, as future works, we intend to integrate data types with a finite body well-defined, as the IEEE 754 standard data types.

The framework provides the search for exact data in the database or by date range. However, searching for data using a String snippet in an obfuscated field is an open challenge. Thus, solutions to these limitations will be studied and proposed.

REFERENCES


