DATABASE SECURITY LAYER

Elder Costa
João Pedro Lorenzo de Siqueira

Monografia apresentada ao Curso de Graduação em Sistemas de Informação, do Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, CEFET/RJ Campus Nova Friburgo, como parte dos requisitos necessários à obtenção do certificado de Bacharel em Sistemas de Informação.

Orientador(a): Nilson Mori Lazarin
Coorientador(a): Carlos Eduardo Pantoja

Rio de Janeiro
Janeiro de 2022

DATABASE SECURITY LAYER

Monografia apresentada ao Curso de Graduação em Sistemas de Informação, do Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, CEFET/RJ Campus Nova Friburgo, como parte dos requisitos necessários à obtenção do certificado de Bacharel em Sistemas de Informação.

Elder Costa

João Pedro Lorenzo de Siqueira

Banca Examinadora:

NILSON MORI LAZARIN:72809353115

Assinado de forma digital por NILSON MORI LAZARIN:72809353115
DN: cn=NILSON MORI LAZARIN:72809353115, ou=CEFET-RJ - Centro Fed. Educ. Tecnol. Celso S. Fonseca, o=ICPEdu, c=BR
Dados: 2022.01.13 18:28:27 -03'00'

Presidente, Professor Me. Nilson Mori Lazarin (CEFET/RJ) (orientador)

gov.br Documento assinado digitalmente
Carlos Eduardo Pantoja
Data: 18/01/2022 12:09:28-0300
Verifique em https://verificador.iti.br

Professor Dr. Carlos Eduardo Pantoja (CEFET/RJ) (coorientador)

BRUNO POLICARPO TOLEDO FREITAS:00484594028

Assinado de forma digital por BRUNO POLICARPO TOLEDO FREITAS:00484594028
Dados: 2022.01.17 11:07:27 -03'00'

Professor Me. Bruno Policarpo Toledo Freitas (CEFET/RJ)

HELGA DOLORICO BALBI:09599358783

Assinado de forma digital por HELGA DOLORICO BALBI:09599358783
Dados: 2022.01.13 20:18:19 -03'00'

Professora Dra. Helga Dolorico Balbi (CEFET/RJ)

LUIS CLAUDIO BATISTA DA SILVA:07180635707

Assinado de forma digital por LUIS CLAUDIO BATISTA DA SILVA:07180635707
Dados: 2022.01.17 22:18:07 -03'00'

Professor Dr. Luis Claudio Batista da Silva (CEFET/RJ)

Rio de Janeiro

Janeiro de 2022

CEFET/RJ – Sistema de Bibliotecas / Biblioteca Uned Nova Friburgo

C837d Costa, Elder

Database security layer. / Elder Costa, João Pedro Lorenzo Siqueira. – Rio de Janeiro, RJ: 2022.

viii, 52f.: il. (algumas color.) , tabs.

Trabalho de Conclusão de Curso (Curso de Bacharelado em Sistemas de Informação) - Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, 2022.

Bibliografia : f. 51-52

Orientador: Nilson Mori Lazarin

Coorientador: Carlos Eduardo Pantoja

1. Computadores – Medidas de segurança. 2. Redes de computadores – Medidas de segurança. 3. Criptografia de dados (Computação). 4. Proteção de dados. I. Siqueira, João Pedro Lorenzo. II. Lazarin, Nilson Mori (Orientador). III. Pantoja, Carlos Eduardo (Coorientador). IV. Título.

CDD 005.8

Elaborada pela bibliotecária Pâmella Priscilla Negrão Braga CRB-7/6062

RESUMO

DATABASE SECURITY LAYER

Com o vigor das novas legislações que discorrem sobre como os dados de usuários devem ser coletados, tratados, armazenados e protegidos, se faz cada vez mais necessário conceber e desenvolver projetos levando em consideração a segurança da informação que trafegará nos sistemas.

Diante do fato de que funções criptográficas disponíveis nos sistemas de gestão de base de dados são limitadas a alguns tipos de dados, este trabalho propõe um complemento para adição de segurança dos dados, através de uma biblioteca para quebra de integridade, atuando em um dos pilares da segurança de informação não comumente utilizados, a integridade.

Palavras-chave: Criptografia; Segurança; Integridade

ABSTRACT

DATABASE SECURITY LAYER

With the enforce of new legislation that discuss how user data must be collected, processed, stored and protected, it is increasingly necessary to design and develop projects taking into account the security of information that will travel in the systems.

Given the fact that cryptographic functions available in database management systems are limited to some types of data, this work proposes a complement to add data security, through an integrity-breaking library, acting one not commonly used pillars of information security, integrity.

Keywords: Cryptography; Security; Integrity

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

Introdução

O mundo está cada vez mais tecnológico. As pessoas, cada vez mais conectadas. Os dados pessoais trafegam o tempo inteiro pela rede. Assim, em um cenário que se aproxima de uma digitalização completa, a segurança da informação torna-se cada vez mais indispensável.

Recentes escândalos envolvendo ataques cibernéticos, como vazamentos de dados de mais de 540 milhões de usuários do Facebook (G1, 2019), mega vazamentos de dados de mais de 223 milhões de brasileiros (G1, 2021) e vazamentos que mostram o faturamento de *streamers* do *Twich* (ESPN, 2021), têm se tornado cada vez mais recorrentes e mostram que a segurança da informação vem se desdobrando para acompanhar o exponencial crescimento da tecnologia e digitalização.

O grande número de usuários na internet, aliado aos recorrentes escândalos citados fez, inclusive, com que os governos ao redor do mundo reformulassem, aprovassem e colocassem em vigor novas leis que tratam desse assunto.

No caso do Brasil, a Lei Geral de Proteção de Dados (LGPD) - LEI Nº 13.709, DE 14 DE AGOSTO DE 2018 - é a lei que está em vigor desde 18 de setembro de 2020 e discorre sobre como os dados de usuários devem ser coletados, tratados, armazenados e protegidos. Como por exemplo, o Artigo 46 que diz:

> Art. 46. “Os agentes de tratamento devem adotar medidas de segurança, técnicas e administrativas aptas a proteger os dados pessoais de acessos não autorizados e de situações acidentais ou ilícitas de destruição, perda, alteração, comunicação ou qualquer forma de tratamento inadequado ou ilícito.” (BRASIL, 2018)

Além disso, ela também define punições e trata algumas sanções administrativas em caso de não cumprimento do que está escrito. Como, por exemplo, o Artigo 52 que diz:

> Art. 52. “Os agentes de tratamento de dados, em razão das infrações cometidas às normas previstas nesta Lei, ficam sujeitos a sanções administrativas aplicáveis pela autoridade nacional.” (BRASIL, 2018)

Essas novas legislações exercem pressão sobre as empresas para desenvolverem soluções de segurança e proteção aos dados que trafegam em seus sistemas e estão suscetíveis aos mais variáveis ataques à segurança existentes na atualidade.

Isso porque um erro, ou descuido na implementação de segurança da aplicação

que está sob responsabilidade da empresa pode determinar seu futuro. Com as novas sanções administrativas, multas e mais definições advindas da nova LGPD, pode ser que muitas empresas não sobreviveriam em caso de um ataque virtual aos dados que estão sob sua posse.

1.1 Problema

Os principais sistemas de gestão de base de dados (SGBD) no mercado, tais como MariaDB[1], MySql[2], SQLServer[3] e Oracle, fornecem funções de criptografia nativa, como, por exemplo, o MD5[4] e AES_ENCRYPT[5]. Entretanto, todos esses grandes SGBD'S possuem algum fator limitante para prover a segurança da informação dos dados armazenados neles, seja por conta das próprias funções de criptografia que estão restritas a certos tipos de dados, como no caso do MySql que não abrange tipos de *date*, *int* e *decimal*.

Existem também serviços que não atuam em dados que estejam em produção, como é o caso do Oracle que oferece o serviço de Data Masking (ORACLE, 2021). Isso é uma desvantagem, visto que a base principal, a de produção, continua sem proteção aos dados sensíveis em caso de invasão.

A partir de uma análise dos mecanismos de bancos de dados existentes no mercado, verificou-se que é comum que o tipo de proteção aos dados provido por eles se baseie em criptografia, que consiste em embaralhar e esconder informações que não se deseja que sejam expostas para quem não tem o acesso as tais.

Segundo (TANENBAUM, 2003), essa técnica tem como objetivo receber uma entrada chamada de "texto em claro", o qual é transformado por uma função parametrizada por uma chave, gerando uma saída chamada de "texto cifrado".

Diante dessa constatação, observou-se que essa técnica que utiliza funções criptográficas predefinidas nos SGBDs, tem como retorno apenas dados no formato

---

[1] https://mariadb.com/kb/en/encryption-hashing-and-compression-functions/. Acesso em: 27 dez. 2021

[2] https://dev.mysql.com/doc/refman/8.0/en/encryption-functions.html. Acesso em: 27 dez. 2021

[3] https://docs.microsoft.com/pt-br/sql/t-sql/functions/encryptbykey-transact-sql. Acesso em: 27 dez. 2021

[4] https://mariadb.com/kb/en/md5/. Acesso em: 27 dez. 2021

[5] https://mariadb.com/kb/en/aes_encrypt/. Acesso em: 27 dez. 2021

string de forma cifrada, atuando no pilar da confidencialidade da segurança da informação. Dessa forma, ela não provê segurança para outros tipos de dados, os quais podem conter informações sensíveis de um usuário ou empresa, como uma data de nascimento, ou saldo em caixa de um cliente, por exemplo.

Assim, mediante a enorme evolução da tecnologia e com o consequente aumento de armazenamento de dados nas redes, se faz necessário o presente estudo e a implementação de uma solução alternativa que contorne e atue nessas brechas existentes ao que tange segurança da informação, de tal modo que, independentemente do tipo dos dados e em um ambiente produtivo, as informações estarão seguras em caso de invasão.

1.2 Motivação

Segurança da informação será sempre um tema em alta e recorrente em fóruns de discussão. À medida que se cria soluções em segurança, novas maneiras e tentativas para quebrá-las são desenvolvidas. É assim que a segurança evolui. Por este motivo, empresas ofertam prêmio para quem conseguir burlar seus sistemas de segurança (BREWSTER, 2019).

Devido a imensa e crescente carga cibernética do mundo atual, em um cenário onde nossa capacidade de colher, processar e distribuir informações torna-se cada vez maior, e as demandas por formas de processamento de informações ainda mais sofisticadas, é iminente a necessidade do aperfeiçoamento das técnicas de segurança da informação para tipos específicos de dados em um banco de dados.

A nova Lei Geral de Proteção de Dados fomentou essas discussões, definiu conceitos que precisavam de definição e fez com que empresas pensassem em maneiras de proteção dos dados que tomam conta e estimulou estudos sobre essa temática.

1.3 Contribuição

Mediante a perspectiva subscrita e a necessidade de novas alternativas para proteger os tipos dados que ainda não são tratados de forma adequada pelos sistemas gerenciadores de bancos de dados, o presente estudo propõe algo diferente do que é

utilizado atualmente.

O trabalho apresenta um modelo baseado em combinações de algoritmos existentes, como o AES (*Advanced Encryption Standard*) Key Expansion e a Cifra de Feistel, de forma a quebrar a integridade do dado, ao invés de atacar a confidencialidade.

O valor de seu conteúdo pode se provar interessante através deste trabalho com sua prova de conceito. Isso porque existe a possibilidade de uma quebra de paradigma ao utilizar outro pilar da segurança da informação como uma nova ferramenta, com uma abordagem diferente para proteção destes tipos de dados, além de servir como um ponto de partida para uma infinidade de novas discussões e propostas utilizando-se dessa abordagem.

1.4 Estrutura do trabalho

No Capítulo 2, é apresentada o Referencial Teórico sobre alguns conceitos e técnicas necessárias para o entendimento do modelo proposto.

No Capítulo 3, são abordados trabalhos relacionados.

No Capítulo 4, é apresentada a proposta deste trabalho: Database Security Layer.

No Capítulo 5, é apresentada a implementação e estrutura da proposta para a prova de conceito.

No Capítulo 6, é apresentada a prova de conceito.

No Capítulo 7, são feitas as considerações finais.

2 Referencial Teórico

Neste capítulo, será abordado um pouco mais sobre conceitos, técnicas relacionadas à segurança da informação e o fluxo de dados de uma aplicação ao sistema gerenciador de banco de dados para entendimento do modelo proposto adiante.

2.1 Segurança da Informação

Conforme o crescimento exponencial do uso de tecnologia no dia a dia e suas aplicações em tarefas cada vez mais privadas, como o uso de meios de pagamentos digitais, é necessário entender o que faz um dado ou informação estar realmente segura. "*Segurança de sistemas de informação é a coleção de atividades que protegem o sistema de informação e os dados armazenados nele*" (KIM e SOLOMON, 2014, p. 6).

Segundo (KIM e SOLOMON, 2014), para a informação ser considerada realmente segura ela deve satisfazer três princípios fundamentais:

1. Disponibilidade: A informação deve estar acessível por usuários autorizados sempre que for solicitada;
2. Integridade: Somente usuários autorizados podem alterar a informação; e
3. Confidencialidade: Somente usuários autorizados podem visualizar a informação.

2.2 Criptografia

A criptografia é um dos métodos mais utilizados atualmente para prover segurança a dados em sistemas de informação. De acordo com (TANENBAUM, 2003) essa técnica consiste em um processo de cifragem dos dados, os quais são transformados caractere por caractere ou bit por bit sem a necessidade de o produto ter uma coesão linguística.

Segundo (KIM e SOLOMON, 2014) um criptossistema é um algoritmo utilizado para poder cifrar (transformar o texto em claro em texto cifrado) e/ou decifrar (transformar o texto cifrado em texto em claro).

Para a utilização de um criptossistema, é necessária a definição de uma chave. A partir da combinação de texto em claro, chave e algoritmo, obtém-se uma saída que será diferente para cada chave utilizada.

O produto principal da criptografia é conhecido como texto cifrado, o qual pode ser transmitido por meios considerados inseguros pois, caso essa informação seja roubada, ela não necessariamente conseguirá ser visualizada pelo usuário não autorizado.

Nos tópicos a seguir desta seção, será apresentado algumas funções criptográficas para ajudar a definir o modelo proposto adiante.

2.2.1 AES

O desenvolvimento e popularização do AES se deu devido o anúncio do NIST (*National Institute of Standards and Technology*) em janeiro de 1997 informado a necessidade de um algoritmo de criptografia sucessor ao DES (*Data Encryption Standard*) que suportava chaves de apenas 56 bits e havia se tornado ultrapassado. Neste mesmo ano, foi lançado um concurso para o algoritmo sucessor que deveria atender alguns requisitos como: direitos autorais livres; divulgação pública; maior desempenho em relação ao 3DES (*Triple Data Encryption Standard*); cifrar blocos de 128bits com chaves de 128, 192 e 256 bits; e possibilidade de implementação em software e hardware (NECHVATAL, BARKER, *et al.*, 2001).

Após a análise de diversos candidatos, o vencedor foi o algoritmo de Rijndael, criado por Vicent Rijmen e Joan Dael, que suportava blocos tanto de 128 bits quanto de 160, 192, 224 e 256 bits. Esse algoritmo foi normatizado e passou a ser implementado somente para blocos de 128 bits e se tornou mundialmente utilizado como AES (NECHVATAL, BARKER, *et al.*, 2001).

Conforme a FIPS PUB 197, o AES é um algoritmo criptográfico utilizado para proteger dados eletrônicos. Esse algoritmo é classificado como uma cifra simétrica de bloco, e é capaz de utilizar chaves criptográficas de 128, 192 e 256 bits para cifrar e decifrar blocos de 128bits (NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, 2001).

2.2.1.1 AES Key Expansion

O algoritmo do AES para realizar o processo de cifragem ou decifragem recebe uma chave a qual define o número de rodadas que será executada sobre o bloco de texto inserido. Para que ocorra esses processos, a chave passada é expandida para

suportar o número de rodadas de acordo com ela, conforme apresentado na Tabela 1.

Tabela 1 - Chaves AES

	AES-128	AES-192	AES-256
Tamanho da chave	128	192	256
Tamanho do bloco	128	128	128
Número de rodadas (rounds)	10	12	14
Tamanho da chave rodada	128	128	128
Tamanho da chave expandida	176	208	240

Segundo (STALLINGS, 2014), a chave(K_n) recebida é copiada para as quatro primeiras *words*(W_n) conforme apresentado no algoritmo geral da Figura 1(a), e as chaves expandidas são preenchidas com quatro words de cada vez. Cada *word* incluída *w[i]* depende da *word* anterior, *w[i-1]*, e da *word* quatro posições atrás, *w[i-4]*. Para cada *word* no array onde a posição não é um múltiplo de 4 é feita apenas uma operação de XOR. Caso contrário, é utilizada uma função mais complexa.

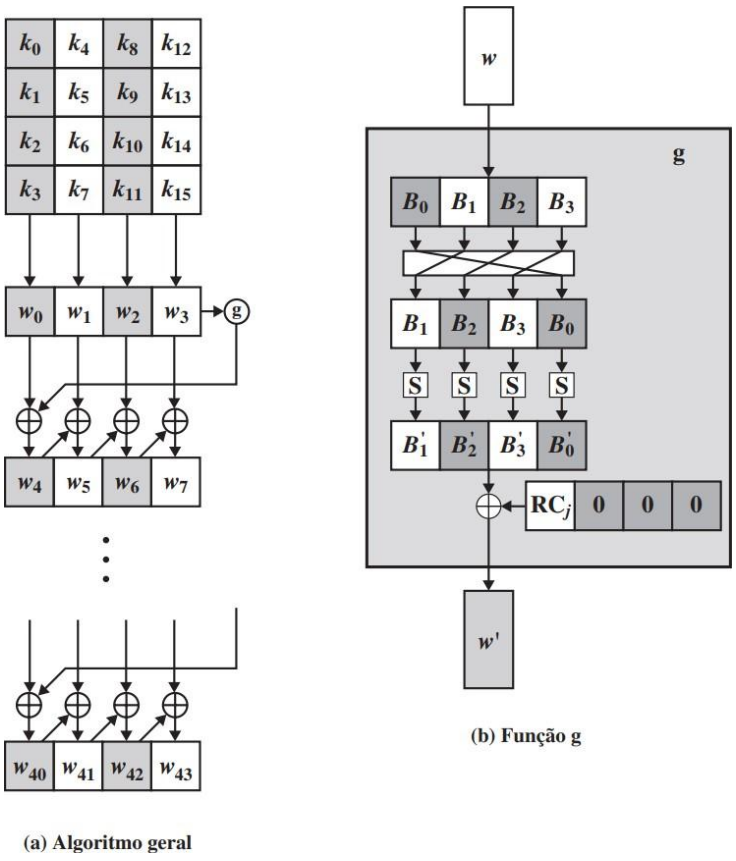

Figura 1 - Expansão de chaves do AES (STALLINGS, 2014)

A função g ilustrada na Figura 1(b), apresenta três processos:

1. *RotWord*: Deslocamento circular byte a byte para a esquerda em uma *word*. Conforme apresentada na Figura 1(b) *word* [B0, B1, B2, B3] é rotacionada para [B1, B2, B3, B0].
2. *SubWord*: É realizada uma substituição byte a byte da word utilizando a *S-box* (tabela de permutação do AES).
3. O resultado das etapas 1 e 2 passa por uma operação lógica *XOR* com a constate de rodada, Rcon.

A constante de rodada é definida como uma word onde os bytes mais a direta são 0, portanto a operação se resume a um XOR com o byte mais à esquerda da word.

O Rcon é formado da seguinte maneira:

Rcon[j] = (RC[j], 0, 0, 0), sendo Rcon[1] = 1 e Rcon[j] = 2 * Rcon[j-1], com multiplicação definida sobre o corpo GF(2^8) e valores expressos em hexadecimal conforme a Tabela 2.

Tabela 2 - Valores do Rcon (STALLINGS, 2014)

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01000000	02000000	04000000	08000000	10000000	20000000	40000000	80000000	1b000000	36000000

2.2.2 Funções Hash

De acordo com (KIM e SOLOMON, 2014), funções *hash* ajudam a detectar falsificações pois calculam uma soma de verificação e depois a combinam com uma função criptográfica. Os *digests* produzidos normalmente têm um tamanho fixo baseado no algoritmo utilizado.

Soma de verificação é um cálculo unidirecional que gera um resultado facilmente verificável ao se passar os dados novamente pela função. Portanto, é um meio de transmitir dados e verificar se eles chegaram corretamente ao destino ou se necessitam serem transmitidos novamente.

Apesar deste meio de conferência dos dados, as somas de verificação são funções extremamente simples e podem ser forçadas para que aparentem estar corretas em uma mensagem trocada ou que alguém modifique a mensagem e ela ainda

seja válida. Deste modo, observa-se que a soma de verificação não garante a segurança, mas sim a confiabilidade.

> Um hash é uma soma de verificação projetada de modo que ninguém possa forjar uma mensagem de forma que resulte no mesmo hash de uma mensagem legítima. Hashes normalmente possuem tamanho fixo. O resultado é um valor de hash. Em geral, valores de hash são maiores que valores de soma de verificação. Hashes atuam como a impressão digital dos dados. Você pode disponibilizar hashes como referência para que destinatários possam ver se a informação mudou. (KIM e SOLOMON, 2014)

A Figura 2 apresenta o funcionamento de uma função de *hash* no qual pequenas alterações no texto inserido (*input*) geram resultados (*digest*) completamente diferentes.

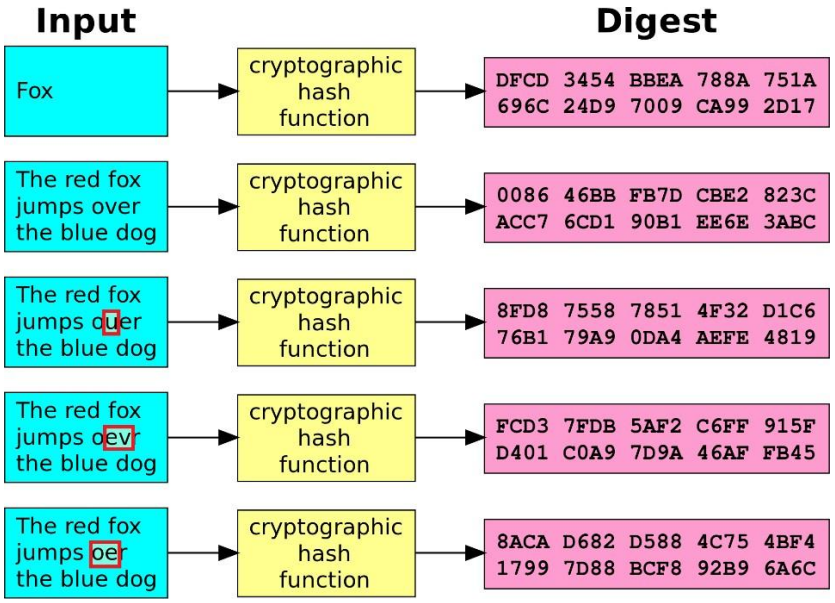

Figura 2 - Função de Hash. Adaptado de (WIKIPÉDIA, 2019)

2.2.3 Cifra de Feistel

Horst Feistel foi um criptógrafo que trabalhou no desenvolvimento de cifras da IBM, e em 1973 propôs um modelo de forma a se aproximar da cifra de blocos ideal através de um conceito de cifra de produto. Esse modelo tem como método a utilização de duas ou mais cifras simples em sequência de forma que o resultado seja mais forte que qualquer uma das cifras utilizada separadamente (WIKIPÉDIA, 2021).

A Figura 3 apresenta a estrutura proposta por Horst Feistel, atualmente muito utilizada para construção de algoritmos de criptografia.

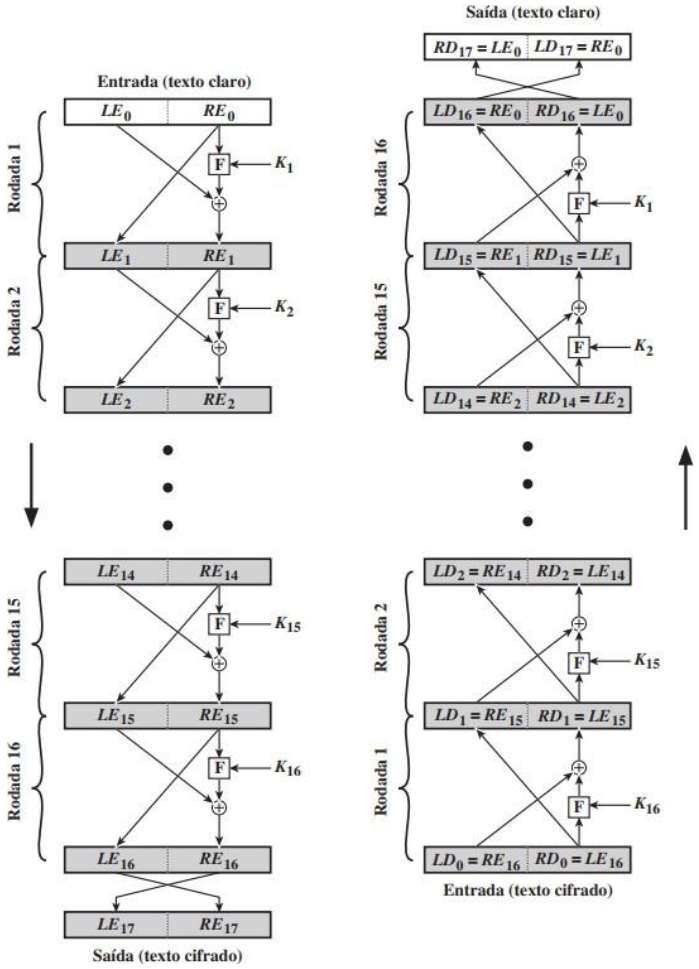

Figura 3 - Cifra de Feistel (STALLINGS, 2014)

De acordo com (STALLINGS, 2014), a Cifra de Feistel é um algoritmo de encriptação/decriptação, o qual recebe como entrada uma chave e um bloco de bits chamado de texto em claro.

Esse bloco e dividido em duas metades, LE_0 e RE_0. Essas metades passam por rodadas de processamento e depois se combinam para produzir o texto cifrado. Para cada rodada de processamento, são utilizados como entrada os blocos produzidos na rodada anterior e uma subchave derivada de chave inicial.

Todas as rodadas seguem o mesmo padrão, no qual é realizado uma substituição na metade esquerda através de uma operação de XOR, com o resultado da operação da metade direta associado a uma função predefinida utilizando a chave da rodada.

Após todas as operações descritas, é feita uma permutação entre os blocos onde o lado direto passa a ser esquerdo e vice-versa, formando assim uma rede de substituição-permutação.

2.3 Mapeamento Objeto-Relacional (ORM)

O mapeamento objeto-relacional é uma técnica de desenvolvimento de *software* utilizada na programação orientada a objeto visando facilitar a persistência em bancos de dados relacionais. Ao empregar essa técnica, o programador utiliza uma interface do mapeador para persistir os dados no SGBD, sem a necessidade de interagir com comandos SQL.

Com a utilização do ORM as tabelas do banco de dados são representadas de acordo com as classes, e os registros de cada tabela são as instâncias das classes correspondentes.

> Um caminho melhor é isolar completamente o Modelo de Domínio (..) do banco de dados, tornando sua camada de indireção inteiramente responsável pelo mapeamento entre os objetos do domínio e as tabelas do banco de dados. Este Mapeador de Dados (...) lida com toda a carga e armazenamento entre o banco de dados e o Modelo do Domínio (...) e permite a ambos variar independentemente. É a mais complicada das arquiteturas de mapeamento de banco de dados, mas seu benefício é o completo isolamento das duas camadas (FOWLER, 2007, p. 55).

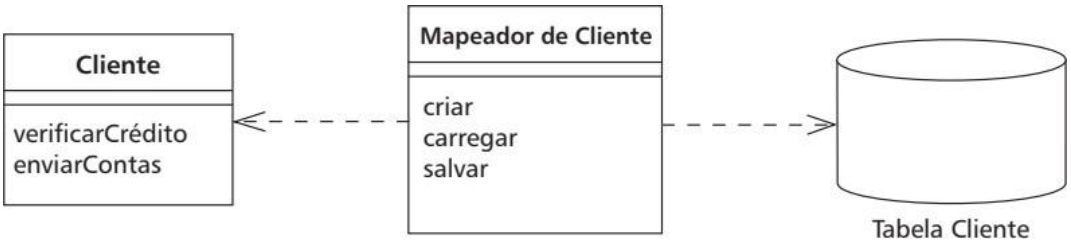

Figura 4 - Mapeamento objeto-relacional (FOWLER, 2007).

3 Trabalhos Relacionados

Nesta seção, serão apresentados trabalhos que serviram como base para elaboração do modelo proposto de quebra de integridade.

3.1 TEAL: Transparent Encryption for the Database Abstraction Layer

(LOREY, BUCHMANN e BÖHM, 2016) desenvolveram um método chamado TEAL (*Transparent Encryption for the database Abstraction Layer*), uma forma de aplicar criptografia na camada de abstração do Banco de Dados (DBAL, do inglês *Database Abstraction Layer*). Dessa forma, a segurança dos dados é ainda mais garantida pois, especialmente quando se trata de terceirização, não há necessidade de o provedor do serviço de armazenamento obter as chaves de criptografia dos dados. A camada de abstração de Banco de Dados é uma maneira de fazer uma "tradução" dos diferentes comandos específicos dos Sistemas Gerenciadores de Bancos de Dados (SGBD) existentes no mercado.

No trabalho citado, foi utilizado o Laravel, um *framework* para essa função que possui integração com o PHP. Este trabalho, relaciona-se com o presente por utilizar de meios diferentes para alcançar um mesmo objetivo. Aqui, é demonstrado um *framework* de criptografia, porém, na atual abordagem será diferente, visto que será trabalhada com o conceito de integridade e não confidencialidade para alguns tipos de dados.

3.2 DBCrypto: Database Encryption System using Query Level Approach

Da mesma forma, (DESHPANDE, PATIL, *et al.*, 2012) desenvolveram, em seu artigo, um *middleware* (programa que atua entre o usuário e o SGBD) chamado DBCrypto. Sua função é fazer a criptografia dos dados reescrevendo as queries vindas do usuário e usando a cifra de Vigenère, uma forma de encriptação simples de soma de números.

As chaves ficam salvas no próprio programa e, portanto, quem tiver acesso aos dados no banco, não poderá saber seus reais valores. Existe um funcionamento diferente para cada tipo de query (*CREATE, INSERT, SELECT, UPDATE*, ...), e a cada chamada do usuário, a query é alterada para se adaptar ao banco encriptado.

A semelhança entre o projeto descrito e o presente encontra-se no fato de ambos

se utilizarem de códigos modificadores de queries para armazenar os dados dentro do banco. Entretanto, o funcionamento do sistema proposto é diferente, visto que não será feita uma busca no banco encriptado a cada chamada, e sim, para alguns tipos de dados. Ao quebrar a integridade desse dado é possível fazer comparações com ele não integro acelerando o processo de consulta. Além do mais, eles se utilizam de um método criptográfico que já foi quebrado e caiu em desuso há muito tempo, deixando em dúvida a efetiva da segurança aplicada.

3.3 LGPD Compliance: A security persistence data layer

LGPD Compliance é um trabalho que foi desenvolvido na disciplina de segurança e auditoria de sistemas, durante o semestre letivo na graduação de Sistemas de Informação, no CEFET/RJ campus Nova Friburgo.

O artigo publicado por (PITTA, COSTA, *et al.*, 2020), tem uma proposta muito semelhante ao modelo apresentado neste trabalho, mas, nesta publicação, os autores tinham como principal objetivo os sistemas legados, evitando a refatoração para implementação de segurança.

De acordo com o que foi apresentado pelos autores, a técnica consistia na utilização de uma biblioteca para realizar um *dump* da base original e alterar a estrutura das tabelas, alterando os tipos dos dados armazenados e posteriormente aplicando o AES para persistência em disco. Essa técnica também previa a utilização de tabelas em memória primaria com quebra de integridade de dados para acelerar as consultas.

Diferente do que foi proposto neste artigo, a quebra de integridade será feita com o retorno do dado no seu tipo original, diferente do que ocorre ao utilizar o AES, pois o retorno é sempre do tipo *string*, persistindo todos em disco sem a necessidade de conversão, permitindo eles serem lidos, porém com a informação “não integra”.

4 A Proposta

A Integridade refere-se à consistência e precisão do estado ou a informação pretendida. Qualquer modificação não autorizada nos dados é uma violação da integridade de dados (HINTZBERGEN, HINTZBERGEN, *et al.*, 2018).

Este trabalho apresenta uma alternativa para cifrar diferentes tipos de dados, os quais, os principais SGBD's do mercado não suportam através de suas funções de criptografia. Localizado na camada de persistência de dados, o modelo proposto atua sobre um pilar da segurança da informação não convencionalmente explorado, a integridade.

Ao abordar esse conceito, desde a sua concepção, a proposta não tem por objetivo impedir que os dados sejam vazados, mas sim, em caso de ataque, que estejam expostos de uma maneira não integra. Isso para que não façam sentido para o atacante e ainda, que não seja possível utilizá-los de nenhuma maneira.

Um exemplo disso seriam os diversos ataques colocados na introdução do texto; fosse o uso da abordagem descrita, um vazamento com informações confidenciais de usuários tais como CPF, data de nascimento, RG, conta bancária, CEP, número de cartão de crédito, não causaria transtorno algum, visto que, com a quebra da integridade do dado inserido, ao invés de estar salvo na base invadida o CPF "087.523.312-35" algo como "BALzCsLteGH7T9XLyhFx5MPAdQ==" seria encontrado pelo atacante. Ou até mesmo uma data de cadastro que, em oposição a uma data '1993-03-23', estaria presente '2107-03-21'. Como na figura 5 abaixo:

dataNascimento	dataCadastro	conta	cpf
2021-01-12	2107-03-21 00:09:20	agBqI73teGFKnYeNs/QTfCo=	BALzCsLteGH7T9XLyhFx5MPAdQ==

Figura 5 – Exemplo tabela ofuscada

Após a Lei Geral de Proteção de Dados (Lei Nº 13.709) entrar em vigor no dia 18 de setembro de 2020, a qual rotulou alguns dados como sensíveis, foi necessária a adequação de muitos sistemas para cumprir com a proteção imposta pela legislação.

Diante dessa necessidade e a verificação das ferramentas e técnicas utilizadas para garantir a segurança desses dados, foi concluído que, para dados que não estejam no formato de *string,* os SGBDs não são capazes de prover segurança em seu formato original, conforme apresentado nas Figuras 6 (insert em uma tabela com campo do tipo string com AES no MySQL) e 7 (insert em uma tabela com campo tipo date com AES).

Figura 6 - Insert do tipo STRING com AES

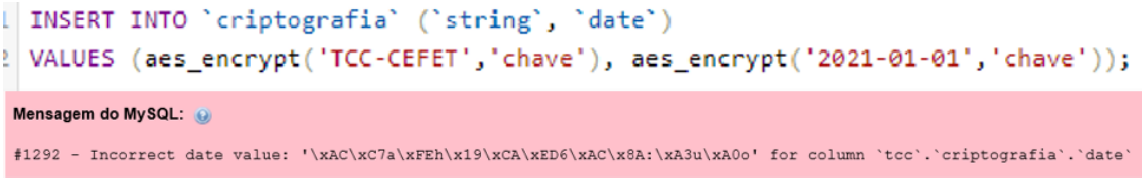

Figura 7 - Erro retornado de insert do tipo DATE com AES

Após essa constatação e análise dos pilares que compõem a segurança da informação, foi elaborada uma solução para os dados que não poderiam utilizar as funções de criptografia disponíveis nos SGBD`s.

Com a finalidade de não trabalhar com conversões de tipos de dados para não modificar a estrutura do banco, não foi possível utilizar as funções criptográficas existentes para alguns tipos. Foi pensada uma solução de forma que, caso ocorra algum tipo de vazamento, o atacante disponha de dados não íntegros, apesar de estarem em seu formato original.

4.1 Concepção

Nesta seção, serão apresentados as ideias e concepções que serviram como base para elaboração do modelo proposto.

4.1.1 Corpo Finito

A proposta é baseada em um relógio, onde se limita o tamanho do corpo finito, ou seja, é possível “saltar” com o dado além do limite preestabelecido no SGBD. A ideia assemelha-se a, justamente, a um relógio que, com seu giro, quando passa do limite final, volta ao início (Figura 8).

Como exemplo, o campo do tipo date no MySQL suporta apenas datas com limites entre 0001-01-01 a 9999-12-31, ou um range de 3652060 dias. Com o

funcionamento da cifra e algoritmo proposto, uma data “2021-10-01” poderia ultrapassar esse limite e estourar o limite final, causando erro ao ser inserido no banco de dados. Para contornar isso, a biblioteca utilizaria da operação de MOD, o que limitaria o tamanho do campo para entregar uma nova data com sua integridade quebrada, porém dentro dos limites predefinidos, sem gerar erro de estouro do campo ao salvá-la.

Figura 8 - Concepção

4.1.2 Quebra da Integridade

Para que a quebra da integridade funcione corretamente, exige-se uma chave em hexadecimal por contemplar caracteres não imprimíveis no teclado assim aumentando a segurança, seguindo a tabela ASCII, e uma semente que será utilizada para geração de entropia. Essa semente será formada a partir do objeto em uso na implementação do desenvolvedor, criada através do nome da tabela, uma mistura das propriedades da entidade, o que justifica a definição e implementação do conceito de mapeamento objeto relacional, e a chave de entropia configurada.

Todas essas etapas e configuração servem para aumentar ainda mais a segurança da aplicação e dificultar a quebra do algoritmo por análise de frequência, além de tornar possível a restauração da integridade do dado.

A chave de cifra passa pelo algoritmo do *AES KEY EXPANSION* para ser expandida em 10 novas chaves de 128 bits.

A semente passa pelo algoritmo *SHA 256* para ser aumentada a 256 bits e para que seja possível dividi-la em duas partes de 128 bits, Ri e Li.

Após a divisão da semente em duas partes e com a chave expandida em 10 novas chaves, o algoritmo passará por um loop de K=10 vezes, utilizando o modelo de

CIFRA DE FEISTEL, com operações de XOR entre os lados e as chaves até que, ao final, um token de 256 bits seja gerado, de acordo com o seguinte processo:

K1 (Primeira chave de 128 bits) entra e sofre XOR com R1 (Primeiro lado direito da semente de 128 bits). O resultado dessa operação passa a se tornar o novo L2 (Segundo lado esquerdo de 128 bits) e é submetido a uma nova operação XOR com o L1 (Primeiro lado esquerdo da semente de 128 bits) gerando um bloco de 128 bits que passa a ser o R2 (Segundo lado direito de 128 bits), e assim continua até o fim das 10 chaves criadas a partir do AES KEY EXPANSION.

O modelo abaixo (Figura 9) demonstra, simplificadamente, esse processo para geração do token de 256 bits.

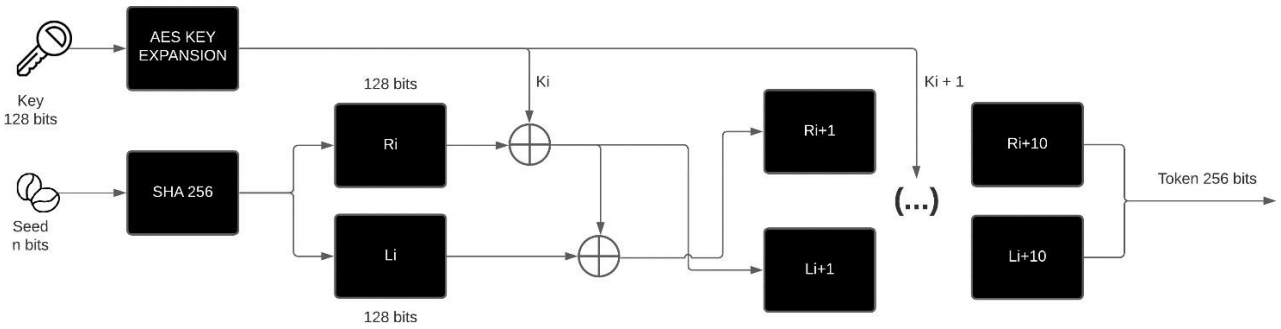

Figura 9 - Processo de geração do token na integrity a partir da chave e semente

O token gerado é transformado em um inteiro e utilizado em conjunto com o dado em claro e o Mod N, utilizado para “saltar” o dado devido ao seu corpo finito, a fim de evitar estouro do limite do campo, para ser gerado o dado ofuscado seguindo a equação 1.

$$Dado_{ofuscado} = (Dado_{em\ claro} + Token)\ mod\ N \quad (1)$$

O N é o limite do campo de acordo com o tipo do dado no SGBD escolhido, sendo ele o responsável de “roletar” o dado ao estouro do limite do campo. Sempre que uma mesma entrada (chave e semente iguais) entrarem no processo de *tokenização* do algoritmo, haverá uma mesma saída. Dessa forma, o processo de restauração do dado dá-se na alteração da fórmula, ao diminuir o Dado Ofuscado pelo Token, conforme a equação 2.

$$Dado_{em\ claro} = (Dado_{ofuscado} - Token)\ mod\ N \quad (2)$$

5 Implementação

Com a finalidade de provar o abordado na seção anterior e, pelo fato de trabalhar-se com o conceito de quebra de integridade, criou-se a biblioteca chamada **Integrity**.

A Integrity foi concebida como uma extensão da camada de persistência de uma aplicação e implementada como uma biblioteca, um pacote que o desenvolvedor faz referência ao seu *namespace* e utiliza suas classes e métodos.

Ela é baseada em três características principais:

- Mapeamento objeto relacional: deve ser capaz de representar tabelas de bancos de dados em objetos da aplicação, isso para que seja possível a geração de entropia, através de uma mistura dos atributos definidos na classe com a chave configurada no início da aplicação.
- Persistência de dados: deve ser capaz de, através do mapeamento objeto relacional, criar comandos e *queries* para persistência na base de dados.
- Segurança e quebra da integridade: com a criação de comandos, através da persistência de dados e com a entropia gerada através do mapeamento relacional, ela deve ser capaz de quebrar e restaurar a integridade dos dados através do algoritmo apresentado anteriormente.

Como primeira versão desenvolvida[6], ela foi construída na linguagem C Sharp, através da plataforma de desenvolvimento .NET Core 5.0, criada e mantida pela Microsoft e a ferramenta Visual Studio.

Além disso, o escopo da prova de conceito foi reduzido para que coubesse neste trabalho. Outras implementações estão como sugestão para trabalhos futuros na seção de conclusão. Por esse fator, o desenvolvimento contemplou apenas o SGBD do MySql e tratou somente os tipos de dado Datetime e String. O exemplo da Figura 10, demonstra o uso da biblioteca em uma classe de persistência “ClientIntegrityData” que estende a classe da Integrity “IntegrityBaseRepository”.

[6] Disponível em: https://gitlab.com/eldercosta/tcc-admin-back

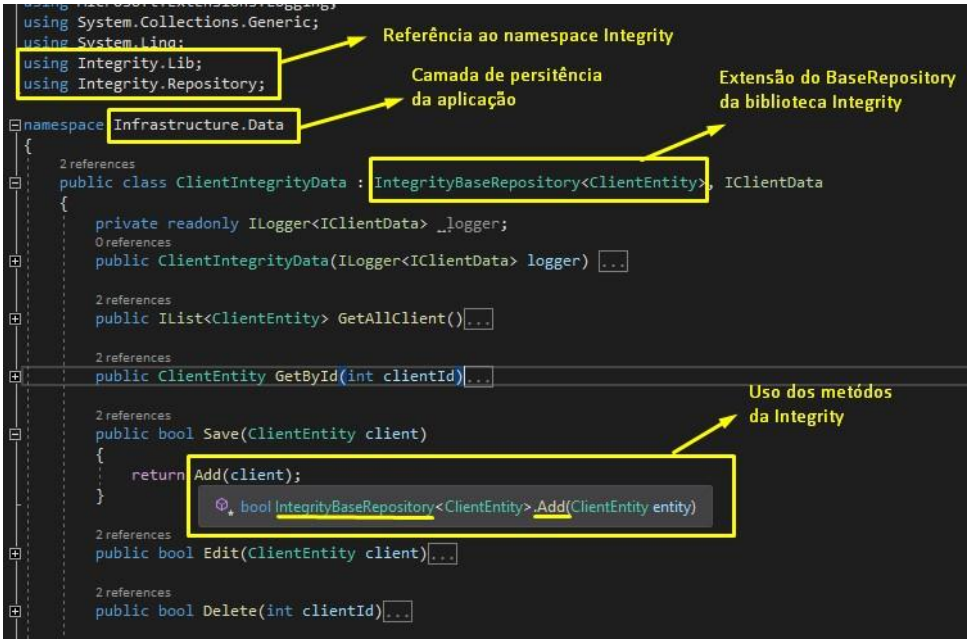

Figura 10 - ClientIntegrityData estendendo IntegrityBaseRepository

5.1 Estrutura

A Integrity está estruturada em três projetos ou bibliotecas de classe, nomenclatura para esse tipo de projeto na linguagem C Sharp, conforme apresentado na Figura 11:

- *Integrity.Core*, que contém as principais classes para o funcionamento da biblioteca.
- *Integrity.Attributes*, que contém tudo relacionado aos atributos dos objetos, para implementação do mapeamento objeto relacional da biblioteca.
- *Integrity.Lib* que contém classes úteis para implementação de persistência de dados e segurança da biblioteca.

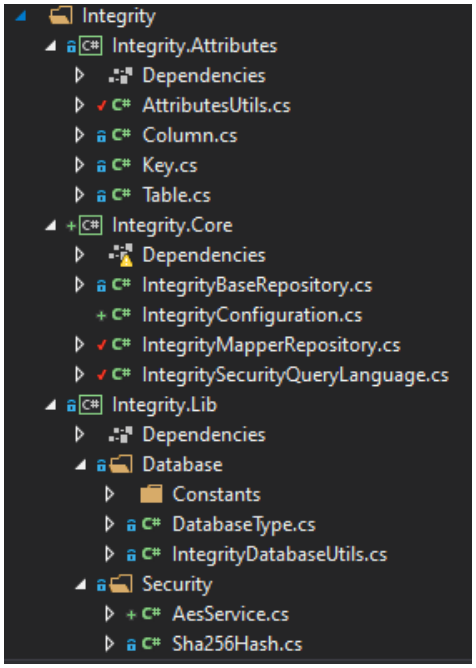

Figura 11 - Estrutura da Integrity

5.1.1 Integrity.Core

A biblioteca de classes “Integrity.Core” é o núcleo do pacote. Nele, estão presentes as principais classes, seja de configuração, mapeamento relacional, repositório base e de segurança, sendo elas: IntegrityConfiguration.cs, IntegrityBaseRepository.cs, IntegrityMapperRepository.cs e IntegritySecurityQueryLanguage.cs

- IntegrityConfiguration.cs é responsável pela configuração inicial da aplicação, através do método “ConfigureIntegrity”. Ela invoca os métodos de cada uma das outras principais bibliotecas para salvar a *string* de conexão com o banco, chave de cifra e chave de entropia.

- IntegrityBaseRepository.cs é a classe abstrata que deve ser estendida pelas classes da camada de persistência da aplicação e fornece os métodos desenvolvidos para o funcionamento da integrity para persistência e leitura dos dados.

- IntegrityMapperRepository.cs é a classe responsável pelo mapeamento objeto relacional da biblioteca. Ela é responsável por criar os comandos de insert, delete, update, select. Além disso, ao usar os métodos da classe IntegritySecurityQueryLanguage, ela quem define os parâmetros que serão salvos na base.

- IntegritySecurityQueryLanguage.cs é a classe que possui as definições e algoritmo de cifragem da biblioteca e dessa abordagem. Ela fornece os métodos para ofuscagem e desofuscagem dos dados.

5.1.2 Integrity.Attributes

A biblioteca de classes *"Integrity.Attributes"* possui tudo relacionado aos atributos dos objetos, para implementação do mapeamento objeto relacional da biblioteca e contém as seguintes classes: AttributesUtils.cs, Column.cs, Key.cs e Table.cs.

- AttributesUtils.cs é a classe que contém utilidades para manipulação dos atributos definidos nas entidades, como criação de dicionário de parâmetros e retorno de lista de atributos a partir da propriedade dos objetos.
- Column.Cs é o atributo de coluna que captura e relaciona o nome da coluna no banco de dados à propriedade do objeto.
- Key.cs é o atributo que define qual determinada propriedade é uma chave da tabela.
- Table.cs é o atributo que define o nome da tabela que faz referência ao objeto em uso na aplicação

Todos os atributos listados nesse projeto são de uso obrigatório para o funcionamento da biblioteca. A partir deles, é possível criar as queries e comandos do banco de dados. Eles são definidos nas classes *entity* da aplicação.

5.1.3 Integrity.Lib

A biblioteca de classes *"Integrity.Lib"* contém classes úteis para implementação de persistência de dados e segurança da biblioteca, como as que seguem: IntegrityDatabaseUtils.cs, MysqlConstants.cs, DatabaseType.cs, AesService.cs e Sha256Hash.cs.

- IntegrityDatabaseUtils.cs possui utilidades para salvamento da *string* de conexão da base de dados que será utilizada pela aplicação através da biblioteca integrity e criação de conexão.
- MysqlConstants.cs possui constantes definidas pelo próprio SGBD para uso do "salto" que o dado deve dar de acordo com o limite do campo tratado, como é o

caso do tipo de dado tratado nessa abordagem.

- AesService.cs é a controladora do algoritmo do AES, baseada na Aes128Ctr.cs[7]. Ela quem fornece os métodos como KeyExpansion que é usado para expandir uma chave de 128 bits em 10 novas chaves de 128bits necessária na geração do token.

5.2 Diagramas

A seguir, nas figuras 12 e 13 são apresentados dois diagramas de sequência que representam a sequência da geração do dado ofuscado e a recuperação e transformação do dado em claro. Na Figura 14 é apresentado um diagrama de classe que corresponde ao mapeamento da estrutura e modelo das classes da implementação da solução.

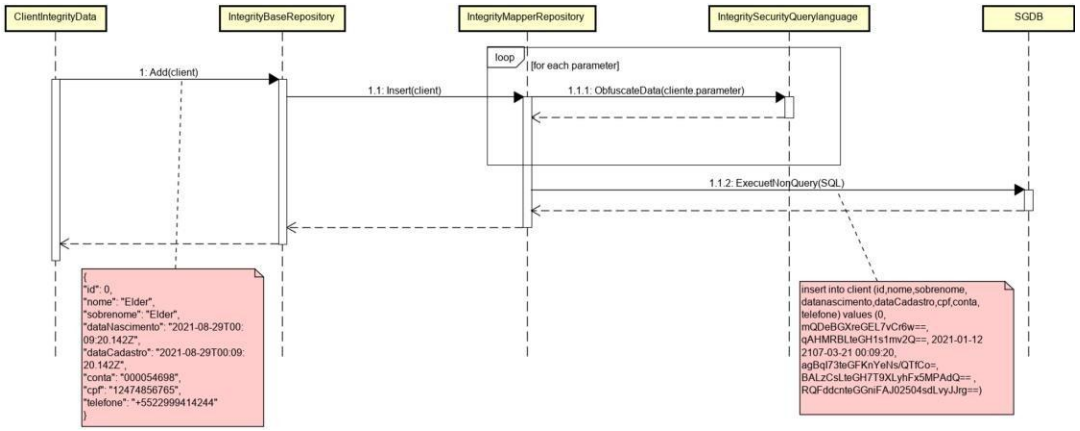

Figura 12 - Diagrama de sequência de ofuscamento do dado

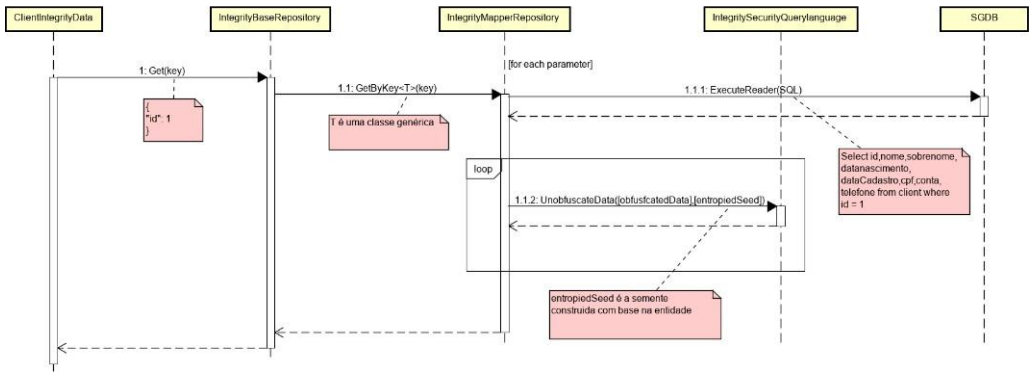

Figura 13 - diagrama de sequência de desofuscar o dado

[7] https://gist.github.com/DamirLisak/5aa61535eba729bf76e4a5511bb01729

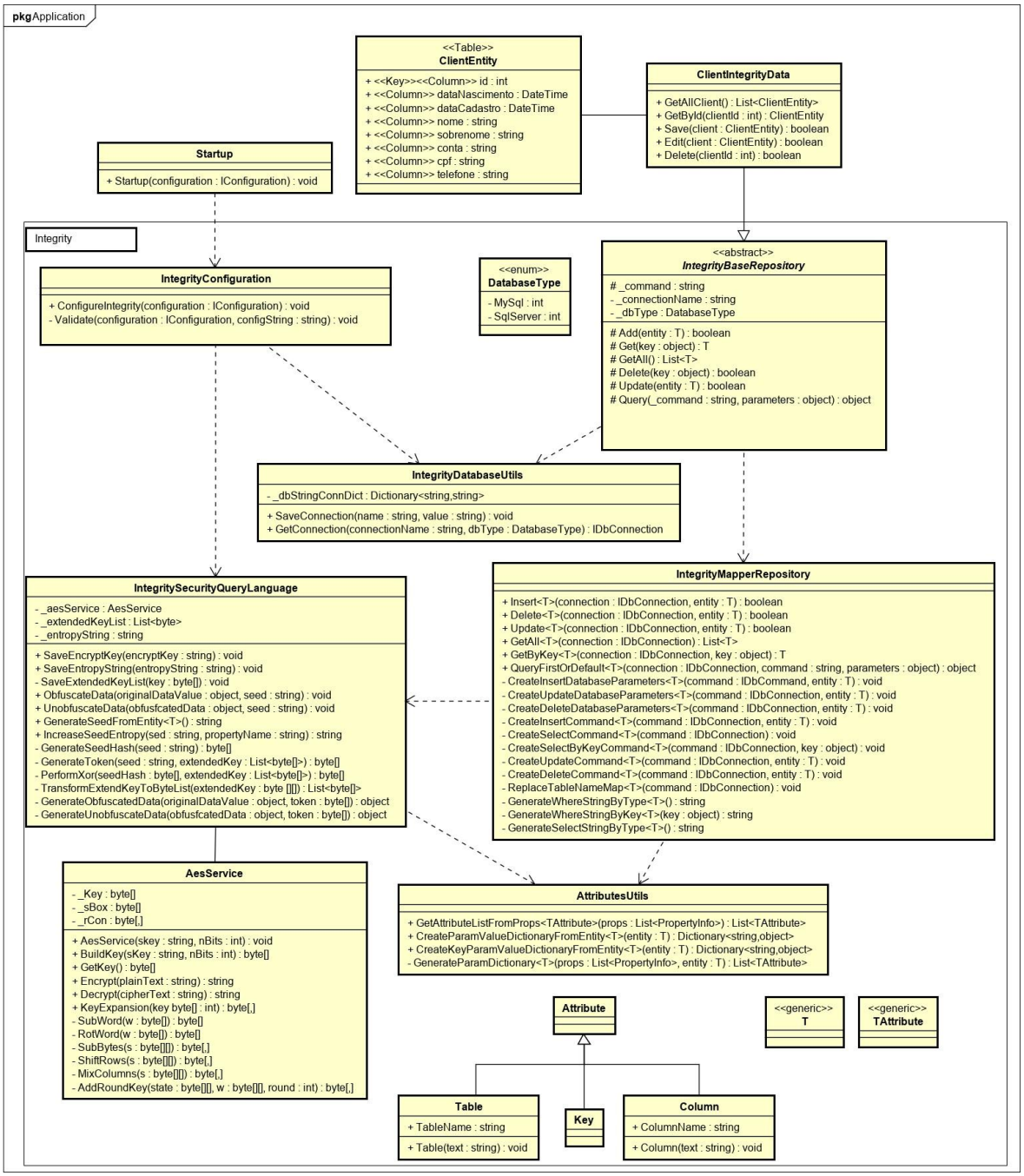

Figura 14 - Diagrama de classes

6 Prova de Conceito

Para que possa ser utilizada, o desenvolvedor necessita acrescentar as informações da Integrity em um json de appsettings contendo as definições de ConnectionString, EncryptKey e EntropySeedString conforme mostrado na Figura 15:

```
},
"Integrity": {
    "ConnectionString": "Server=localhost;Port=3306;Database=integrity;Uid=integrity_user;Connection LifeTime=0;Pooling=true;Pwd=querocifrar123;",
    "EncryptKey": "ENCRYPTED KEY INTEGRITY ';A",
    "EntropySeedString": "ENTROPY SEED"
}
```

Figura 15 - Appsettings com configurações da Integrity

Definida as configurações, é necessário usar a extensão da Integrity, e invocar o método “ConfigureIntegrity” no startup da aplicação, para que a biblioteca capture essas definições e salve em memória as informações, conforme as figuras 16 e 17:

```
11  using Integrity;
12
13  namespace WebApplication1
14  {
        2 references
15      public class Startup
16      {
            0 references
17          public Startup(IConfiguration configuration)
18          {
19              Configuration = configuration;
20
21              Configuration.ConfigureIntegrity();
22          }
23
                                (extension) void IConfiguration.ConfigureIntegrity()
            2 references
```

Figura 16 – Startup - ConfigureIntegrity

```
public static void ConfigureIntegrity(this IConfiguration configuration)
{
    var conn = configuration.Validate("ConnectionString");
    var encryptKey = configuration.Validate("EncryptKey");
    var entropySeed = configuration.Validate("EntropySeedString");

    IntegrityDatabaseUtils.SaveConnection("INTEGRITY", conn);
    IntegritySecurityQueryLanguage.SaveEncryptKey(encryptKey);
    IntegritySecurityQueryLanguage.SaveEntropyString(entropySeed);
}
```

Figura 17 – IntegrityConfig - ConfigureIntegrity

Posterior à etapa de configuração, os dois últimos processos necessários para que a biblioteca funcione corretamente é criar a entidade, seguindo o modelo objeto relacional com os atributos que a Integrity expõe, estender a classe abstrata de

persistência que é fornecida pelo pacote, e usar seus métodos como mostrado na Figura 18:

```
using Integrity.Lib;
using Integrity.Repository;

namespace Infrastructure.Data
{
    2 references
    public class ClientIntegrityData : IntegrityBaseRepository<ClientEntity>, IClientData
    {
        0 references
        public ClientIntegrityData() : base("Integrity", DatabaseType.MySql)
        {
        }

        2 references
        public IList<ClientEntity> GetAllClient() => GetAll().ToList();

        2 references
        public ClientEntity GetById(int clientId) => Get(new { id = clientId });

        2 references
        public bool Save(ClientEntity client) => Add(client);

        2 references
        public bool Edit(ClientEntity client) => Update(client);

        2 references
        public bool Delete(int clientId) => Delete(new { id = clientId });
```

IEnumerable<ClientEntity> IntegrityBaseRepository<ClientEntity>.GetAll()

Figura 18 – Extensão da camada de persistência

Após toda configuração feita, objetos definidos e camada de persistência estendendo a classe da IntegrityBaseRepository, a aplicação já está apta a iniciar com o uso da biblioteca.

Como prova de conceito, foi criada uma simples Api de um módulo de cliente e métodos CRUD (*Create, Read, Update e Delete*) de inserção, deleção, atualização e seleção padrão.

Logo no início, no *startup* da aplicação, conforme fora mostrado mais acima, a api configura a integrity chamando o método ConfigureIntegrity. Ele, instantaneamente, invoca os métodos SaveConnection, para salvar a conexão da base de dados, SaveEncrypt para salvar a chave de cifra, e SaveEntropyString (Figura 20).

SaveEncrypt utiliza a chave de cifra na configuração e, automaticamente, cria a chave extendida que será utilizada no processo de tokenização conforme mostrado na Figura 19.

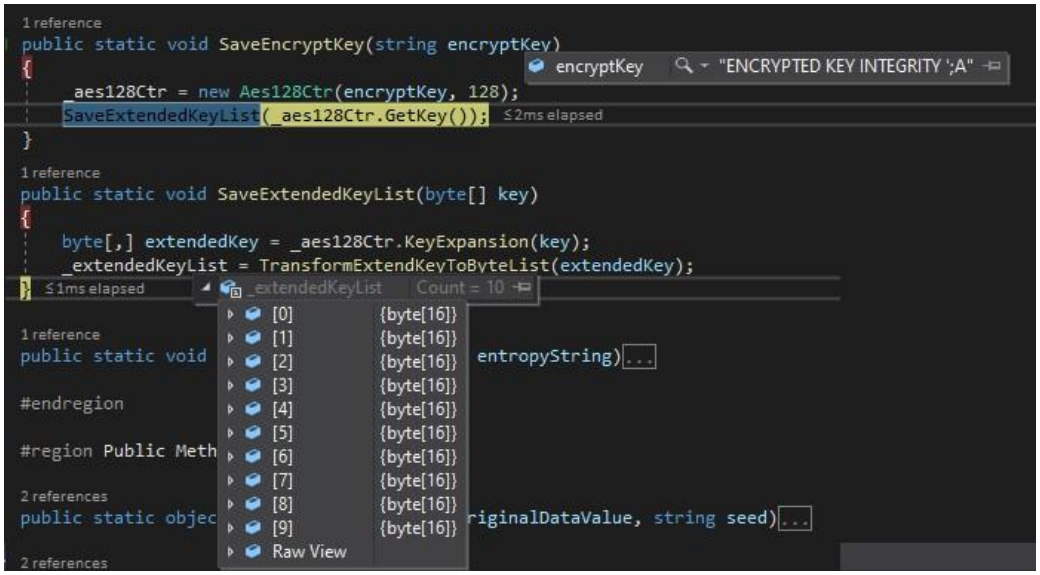

Figura 19 – Extensão da chave

SaveEntropyString utiliza a *string* da chave de entropia e a transforma em uma chave de 128 bits usando o serviço do AesService, e volta com ela para o formato *string*. Esse processo de ida e volta é realizado para garantir que, mesmo que a chave definida seja menor que 128 bits, ao criar a chave ela sempre terá o mesmo valor. Isso para que seja possível quebrar e voltar com a integridade do dado. A Figura 20 demonstra esse processo.

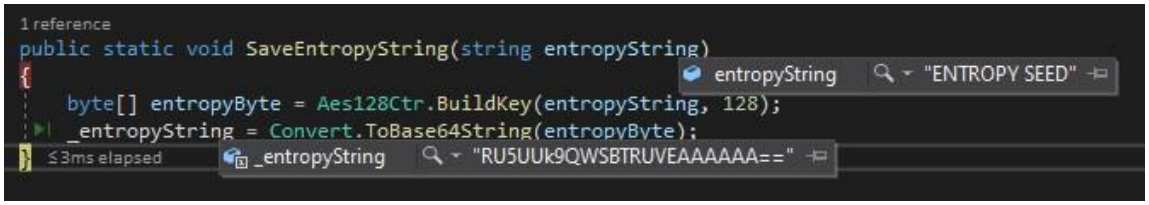

Figura 20 – Geração e salvamento da chave de entropia

Com a aplicação iniciada, é possível dar início ao salvamento dos dados refentes a um cliente qualquer na base de dados.

Para isso foi, criado uma tabela no MySql para cliente com as seguintes informações: id, nome, sobrenome, dataNascimento, dataCadastro, conta, cpf, telefone, como mostrado na Figura 21:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
nome	varchar(100)	NO		NULL	
sobrenome	varchar(100)	NO		NULL	
dataNascimento	date	NO		NULL	
dataCadastro	datetime	NO		NULL	
conta	varchar(100)	NO	UNI	NULL	
cpf	varchar(100)	NO	UNI	NULL	
telefone	varchar(100)	NO	UNI	NULL	

Figura 21 – Estrutura da tabela

O detalhe aqui e que será visto mais adiante é que excetuando-se o id e os campos date, todo os outros campos estão em formato varchar. Isso porque a biblioteca passou a atuar também com campos do tipo *string*.

O uso é basicamente o mesmo, através da função do Aes_Encrypt, mas sua abordagem é diferente do usual e será vista mais adiante.

A Figura 22 demonstra um exemplo de uma requisição post passando dados de um cliente para serem salvos na base, o que dará início a todo o funcionamento do algoritmo:

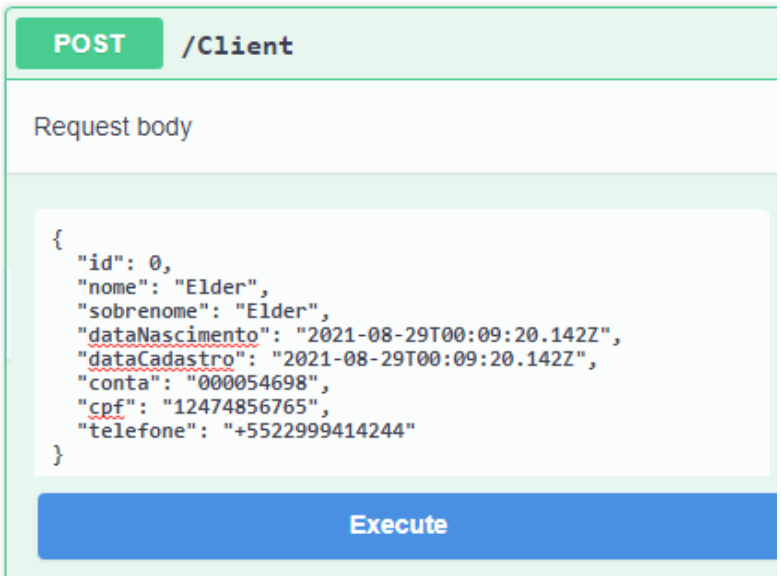

Figura 22 – Requisição POST

Perceba que o valor de dataNascimento e dataCadastro são iguais. Isso foi proposital para demonstrar um conceito que a biblioteca emprega: Entropia entre classes e dentro da classe. Ou seja, classes e propriedades diferentes fornecem

sementes diferentes. Sementes diferentes fornecem tokens distintos. Com tokens distintos, o salto que o dado deve dar é diferente. Detalhes sobre isso serão vistos mais adiante.

O objeto a ser persistido percorre a camada de aplicação, navegando da controladora de cliente (Figura 23) para a camada de negócio quando chega ao serviço de cliente (Figura 24). Até então, o objeto percorreu o fluxo natural da aplicação.

Figura 23 – Objeto na controladora de cliente

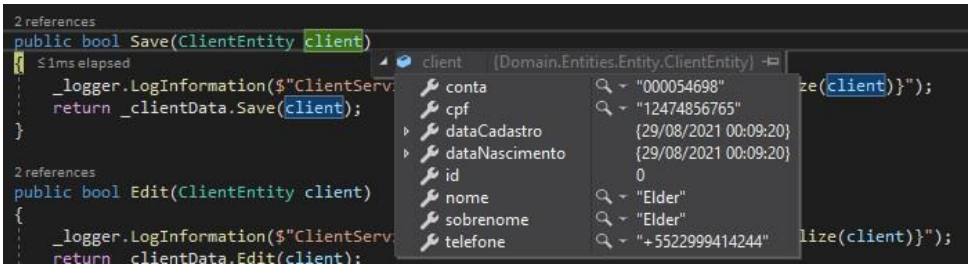

Figura 24 – Objeto no serviço de cliente

Ao chegar na camada de persistência, começa a atuação da integrity de fato.

Ao ser estendida, a IntegrityBaseRepository recebe um tipo T genérico, além de possuir, em seu construtor, a definição da conexão, contendo o nome da conexão (salva no início da aplicação) e o tipo de SGBD em uso, o que caracteriza o contexto que será usado, conforme a Figura 25.

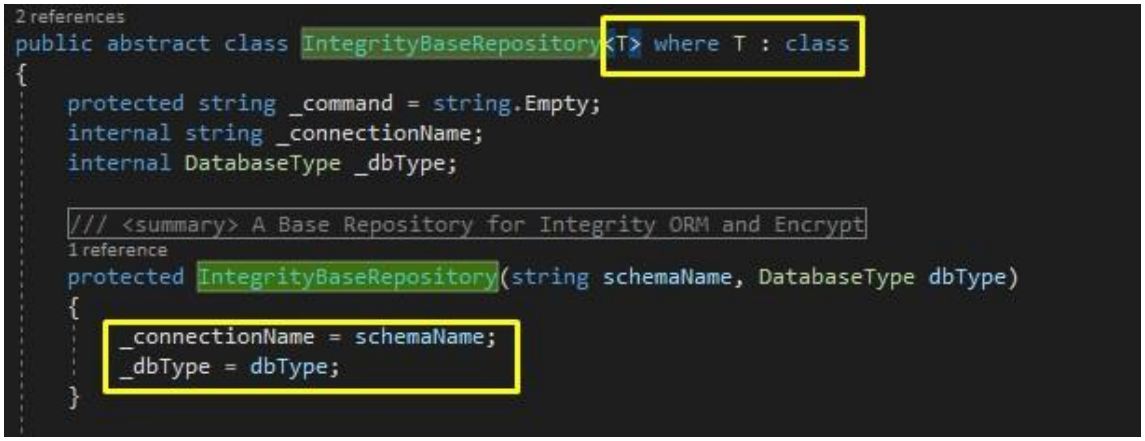

Figura 25 – IntegrityBaseRepository

No caso da prova de conceito, uma entidade do tipo ClientEntity está sendo salva a partir da camada de persistência de cliente, que se conecta ao SGBD Mysql a partir da conexão “Integrity”, conforme a Figura 26:

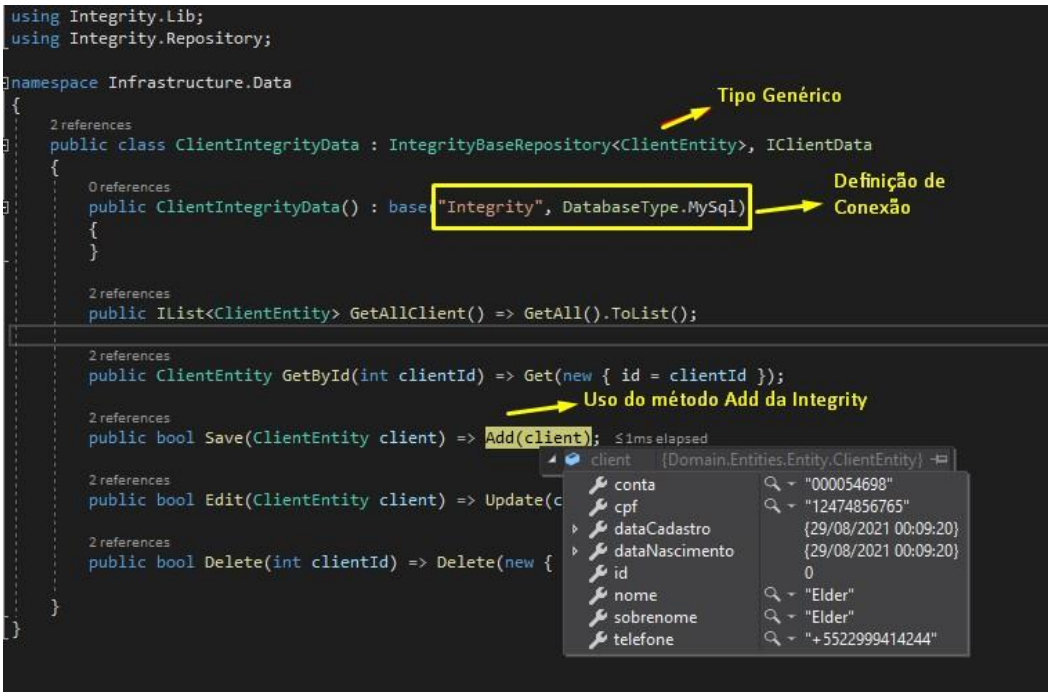

Figura 26 – Exemplo da camada de persistência de cliente salvando um objeto

Ao usar o método Add da Integrity, o objeto fica de “posse da biblioteca”. A partir desse momento, a aplicação original não necessita fazer mais nada além do envio da resposta, que a Integrity irá retornar.

Sob seu domínio, o repositório base, através do método Add, inicia uma conexão

com o banco definido pela classe pai e, com a entidade passada, inicia o processo de insert, que está na IntegrityMapperReposity, na base.

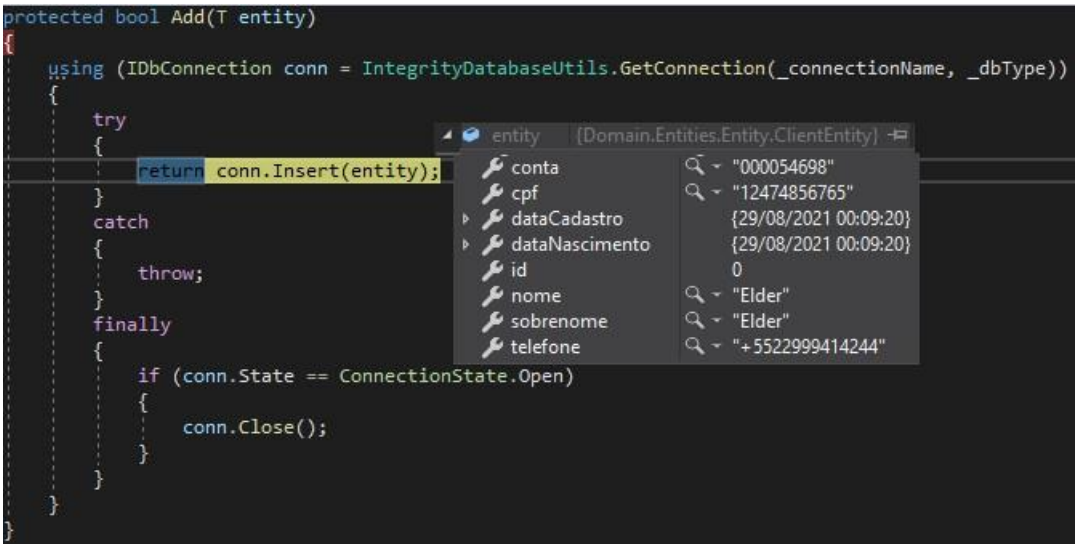

Figura 27 – Método add na IntegrityBaseRepository usando conexão anteriormente definida

O método de insert tem 3 funções básicas: abrir a conexão, criar o comando de insert e executar o comando.

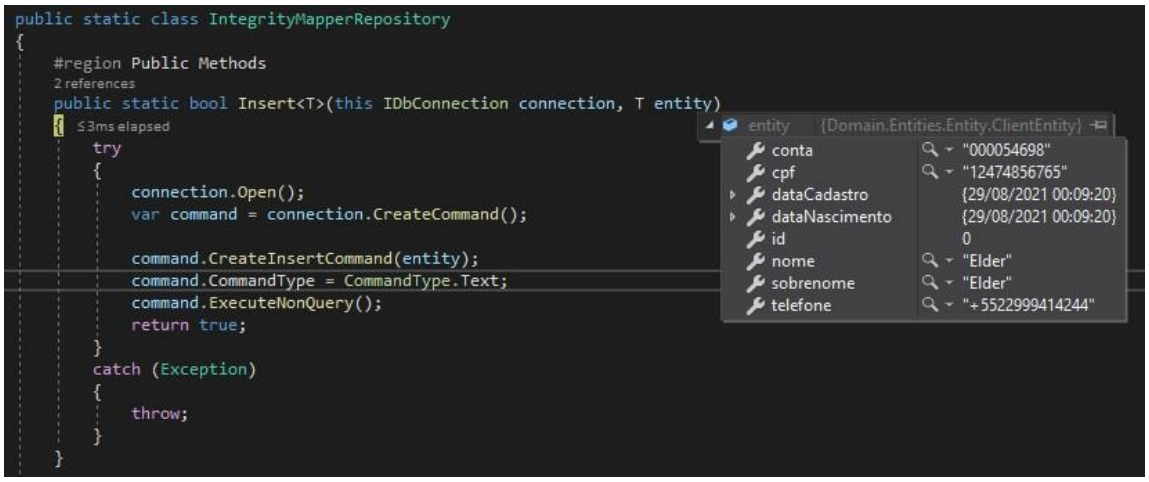

Figura 28 – Método Insert na IntegrityMapperReposity

O funcionamento do mapeamento objeto relacional da Integrity está definido na criação do comando de insert. Ao obter as propriedades do objeto genérico T, os atributos do tipo Column mapeando as propriedades do objeto a base de dados, nome da tabela, é possível gerar o comando de insert para cada tipo T passado.

A Figura 29 demonstra esse funcionamento e o comando gerado.

Figura 29 – Funcionamento do mapeamento objeto relacional e comando gerado

Gerado o comando que será executado mais adiante, cabe então ao método CreateInsertDatabaseParameters criar os parâmetros que serão persistidos na base.

Esse é o momento central do funcionamento da Integrity. Nesse método, será criado o dicionário contendo os parâmetros e valores a serem manipulados. Após isso, uma primeira semente de entropia, chamada de entropia entre classes, que será utilizada como padrão para todas as propriedades da mesma classe, é gerada a partir do método GenereateSeedFromEntity.

O GenereateSeedFromEntity cria a semente utilizando o nome da tabela com a junção das propriedades da classe e a semente de entropia definida no início da aplicação, resultando em algo parecido com o que está na Figura 30.

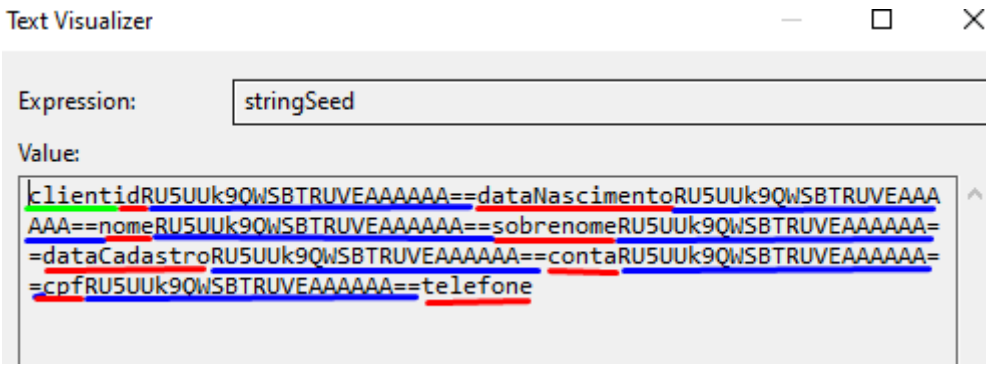

Figura 30 - StringSeed gerada a partir da entidade

É possível reparar que, o que está sublinhado em verde é o nome da tabela “cliente”, em vermelho são as propriedades do objeto, em azul a semente de entropia gerada no inicio da aplicação.

Essa primeira entropia difere entre classes porque, como ela utiliza o nome da

tabela, supondo uma classe que use uma tabela “funcionário”, uma nova chave seria gerada.

Após isso, CreateInsertDatabaseParameters executa, para cada parâmetro e valor presente no dicionário, a manipulação e adição dele ao comando que será executado.

Nesse passo, uma nova semente de entropia é gerada, adicionado ao fim da *string* da semente principal, a propriedade que está sendo adicionada ao comando, como mostra a Figura 31:

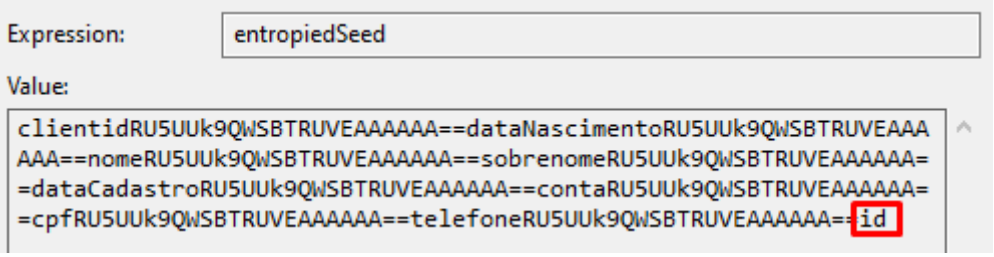

Figura 31 - EntropiedSeed gerada a partir da classe e atributo

É possivel reparar que a *string* de entropia sofreu mais uma entropia para aquela propriedade específica de id. Isso modifica completamente o salto que o dado deve dar em datas iguais em campos diferentes, por exemplo. Isso ocorre mesmo em campos do tipo *string*, pois essa mudança na semente de entropia tanto altera a chave de encriptação do algoritmo AES, como também, com uma semente diferente para cada coluna, o token gerado para saltar o dado se torna diferente.

Com a semente e o valor a ser ofuscado, o método invoca a camada de segurança IntegritySecurityQueryLanguage através do método obfuscateData.

Através da Figura 32 a seguir é possivel observar o comportamento geral do método.

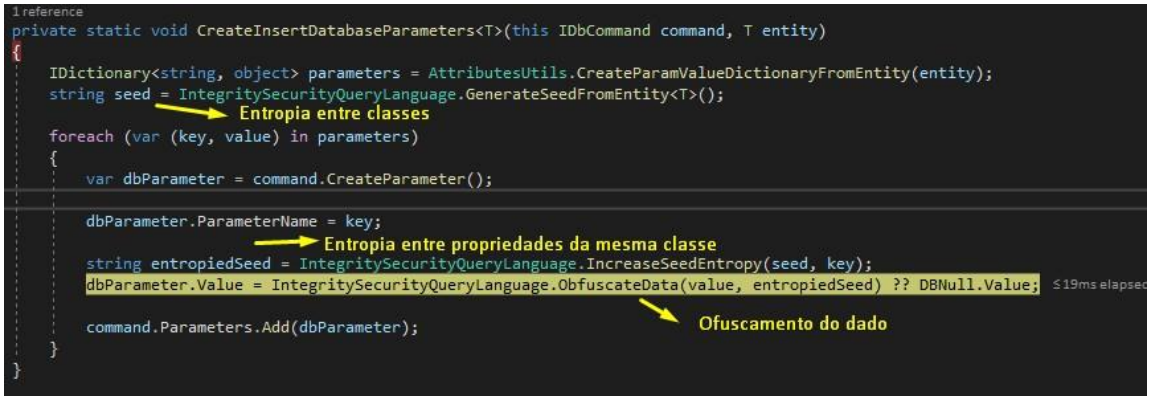

Figura 32 - CreateInsertDatabaseParameters

O método ObfuscateData é o responsável por quebrar o dado que chega íntegro para ser salvo de maneira não íntegra na base.

O primeiro passo dele, após receber o dado original e a semente, é gerar o token a partir da semente e da chave extendida no inicio da aplicação. Por fim, ele invoca o método privado GenerateObfuscatedData que será visto mais adiante para gerar o dado ofuscado de fato. A Figura 33 demonstra o método em questão.

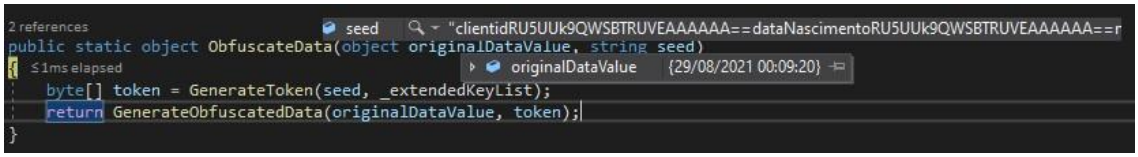

Figura 33 – ObfuscateData

Nesse caso, foi passada a *string* que sofreu entropia entre classes e entropia entre propriedades da mesma classe, e o dado em claro passado no inicio da requisição “dataNascimento: 2021-08-29 00:09:20”

A partir daí, é dado inicio ao processo de geração do token que já foi abordado ao longo do trabalho, mas para que se possa ter uma visualização da sua implementação, a Figura 34 a seguir demonstra seu funcionamento.

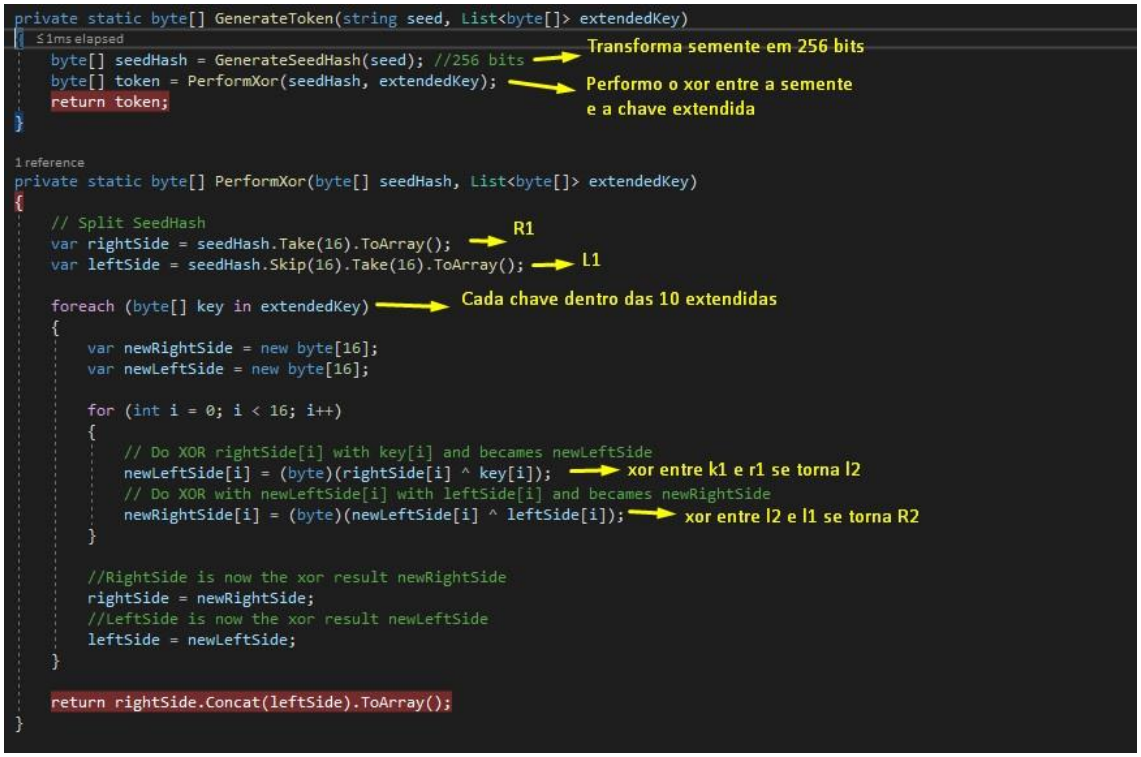

Figura 34 - Geração do token

O resultado dessa operação, como já fora discutido antes, é um token de 256 bits, ou 32bytes já que 8 bits equivalem a 1 byte.

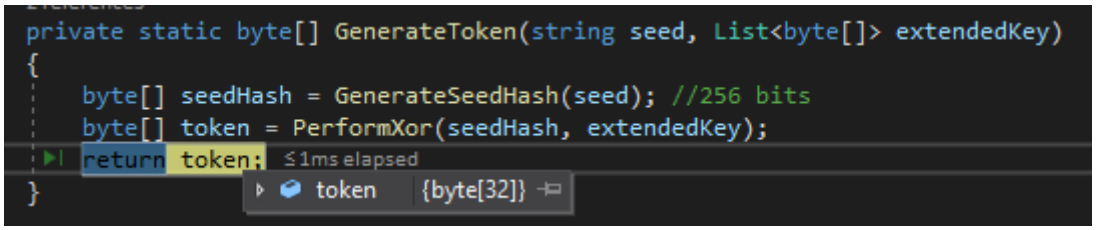

Figura 35 - Resultado token de 256 bits

Com o token gerado, a última etapa se torna a definição do tipo de dado que irá saltar ou cifrar, e o uso da fórmula do dado ofuscado apresentada anteriormente na equação 1. Tudo isso pode ser visto na implementação do metódo GenerateObfuscatedData

O primeiro passo é converter essa cadeia de byte em um inteiro que seja possível saltar o dado.

Após isso, ao definir que o tipo do objeto é um datetime, é feito um mod entre o inteiro gerado do token e o range do limite de dias do MySql para datetime. Esse resultado é somado (ou subtraído, dependendo do valor do inteiro do token) do dado

original. Isso transforma a data original em outra completamente diferente.

A Figura 36 demonstra esse processo.

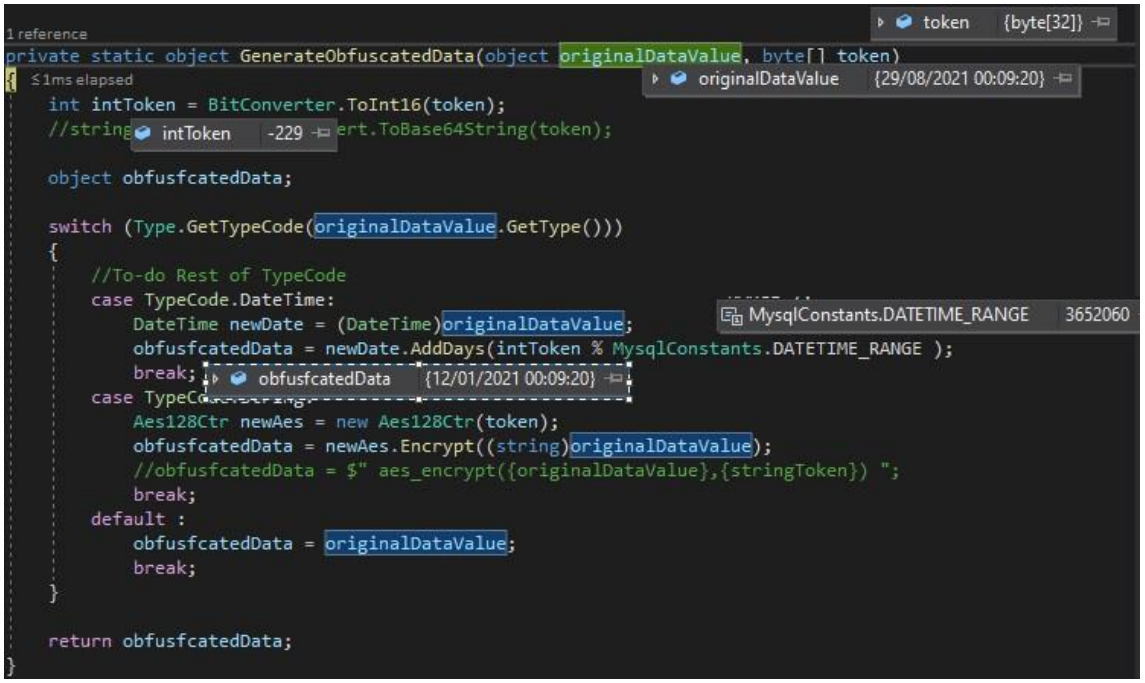

Figura 36 - Geração do dado ofuscado

Como é possível observar, dado o token resultado do processo de tokenização, o inteiro -229 foi gerado. Com a data original ‘2021-08-29’ e esse inteiro do token e a operação de mod com o limite do tamanho do campo de datetime do mysql, essa data “saltou” para ‘2021-01-12’, perdendo seu valor em caso de vazamento da informação.

Para tipos do tipo string, o processo permanece o mesmo: É gerada uma semente de entropia para a propriedade nome e, a partir dela, um token é novamente criado, completamente diferente do outro. A diferença , então, aparece no momento de gerar o dado ofuscado.

Nesse caso, é criada uma nova controladora do AES com o novo token gerado. Nesse momento, uma chave é construída especialmente para aquele campo, e é possível utilizar o próprio método de encriptação da biblioteca para cifrar o dado.

Esse é o ponto abordado mais acima a biblioteca fornecia uma complexidade maior para campos do tipo string. Pelo fato de o token sempre ser diferente entre diferentes tabelas e diferentes campos da tabela, a chave de encriptação alterna-se e torna-se mais complexo um ataque de análise de frequência. Sem contar também que “encapsula” a segurança por campo da tabela, já que, encontrar a chave de cifra para o campo nome, por exemplo, não significa que servirá para o campo sobrenome. Além do mais, isso pode vir a ser configurável em trabalhos futuros para que ainda mais

complexidade seja fornecida pela biblioteca.

As Figuras 37 e 38 demonstram o que fora abordado acima.

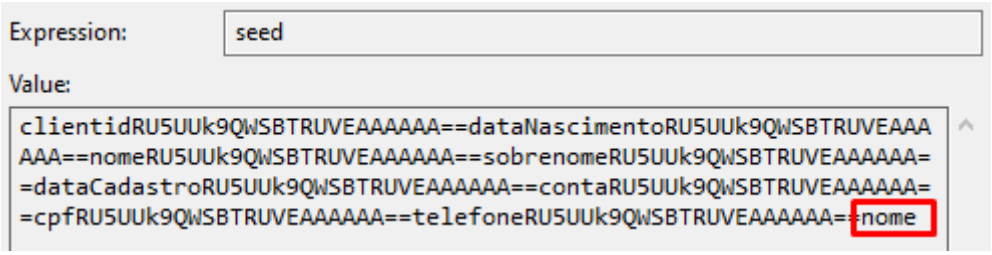

Figura 37 - Nova semente gerada com um novo campo da tabela

Uma nova semente para o campo nome é gerada, com isso um novo token é gerado.

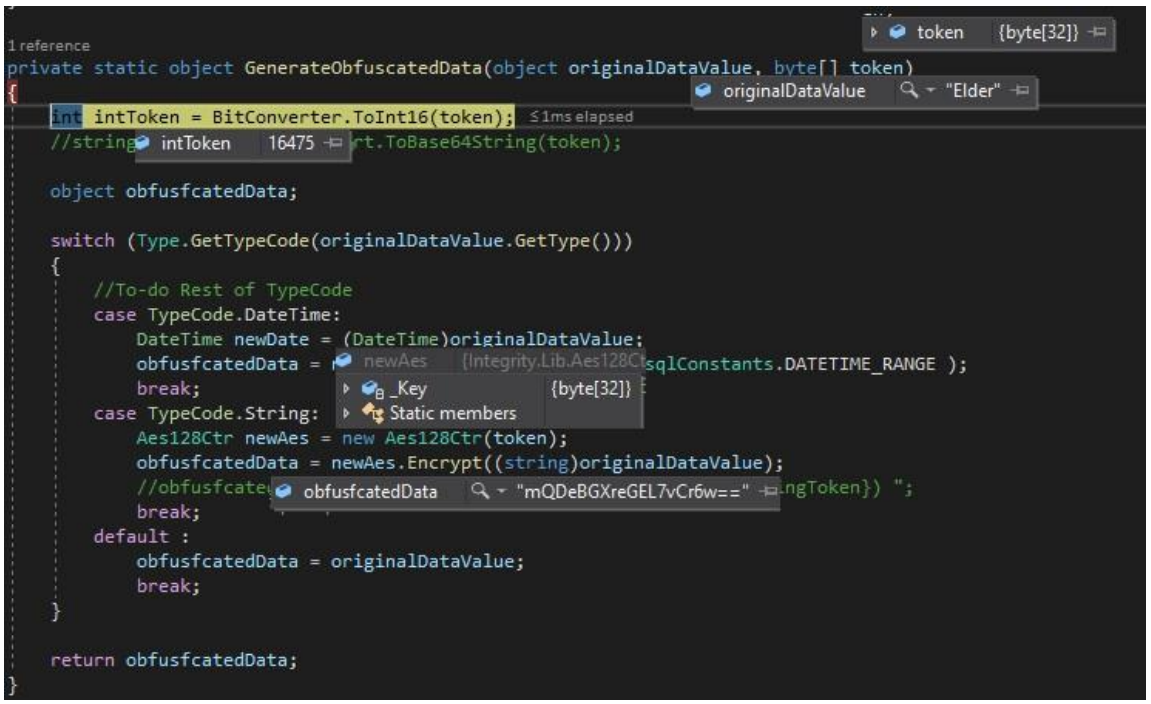

Figura 38 - Novo token gerado

É possível observar que, devido ao token ser diferente, o inteiro do token também foi alterado. Ao fim da execução, uma string cifrada é gerada e pronta para ser persistida na base.

Para provar a força da implementação, foi enviado, no campo sobrenome, também o valor “Elder”. Na Figura 39, é possível observar um dado ofuscado completamente diferente do obtido no campo nome para o mesmo valor “Elder”. Isso por conta do token que foi gerado com entropia entre as propriedades da classe.

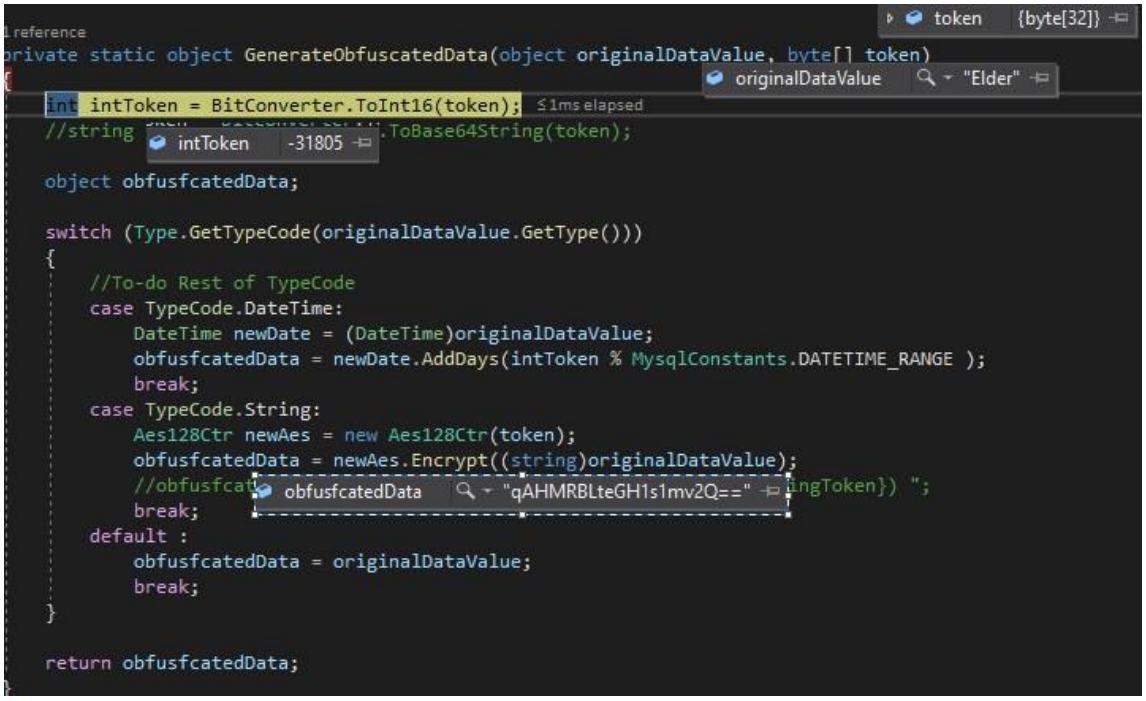

Figura 39 - Dado ofuscado diferente por conta da entropia

Ao fim da execução do post do clienteé possível observar que as informações relativas ao cliente foram registradas de forma ofuscada na base de dados, como mostra a Figura 40.

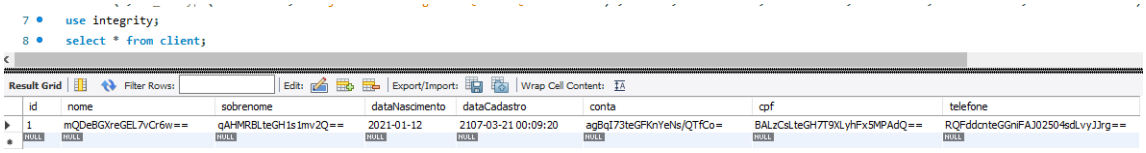

Figura 40 - Cliente ofuscado salvo na base.

Nota-se que os campos nome e sobrenome, dataNascimento e dataCadastro, apesar de guardarem a mesma informação, foram registrados na base com valores diferentes.

A volta da integridade dá-se pelo mesmo processo: geração da semente da entropia da classe, diferenciação e aumento de entropia por parametro e, geração do token a partir da semente deferenciada. A diferença dá-se na volta do campo, uma operação inversa no token é feita para casos do tipo datetime e a função Decrypt é usada para campos do tipo string, quando chamado o método GenerateUnobfuscatedata, conforme a Figura 41:

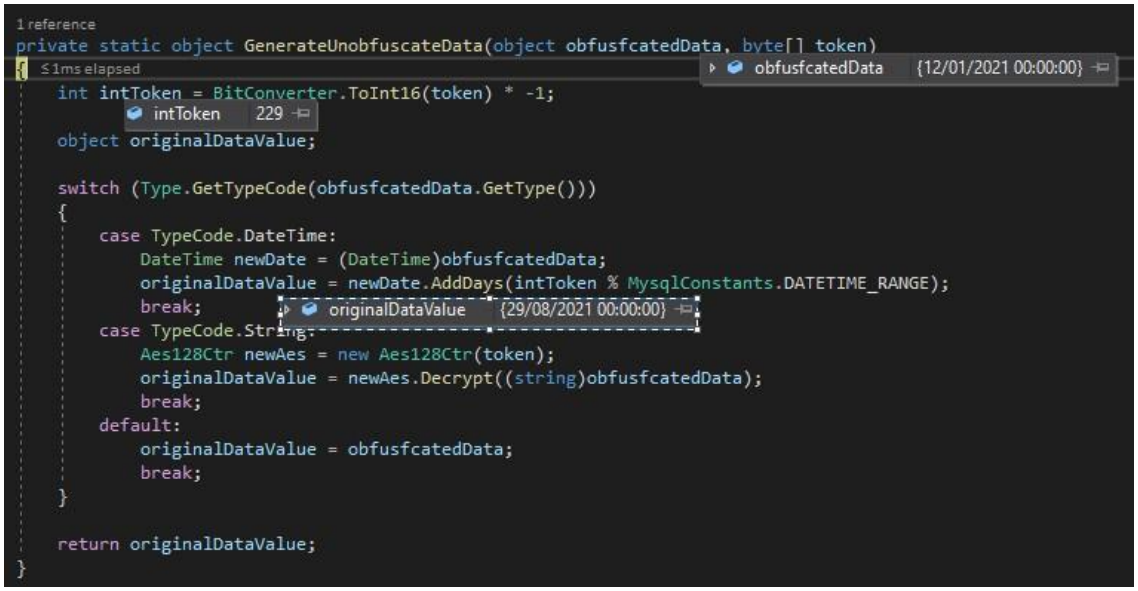

Figura 41 - Volta da integridade do dado

Por fim, o retorno da chamada get na tabela de cliente retorna os dados que seriam salvos originalmente na base, provando-se que é possivel quebrar e restaurar a integridade do dado através da biblioteca.

As figuras 42, 43 e 44 mostram a chamada post com o objeto original, os dados persistidos com sua integridade quebrada e o retorno dos dados através da biblioteca.

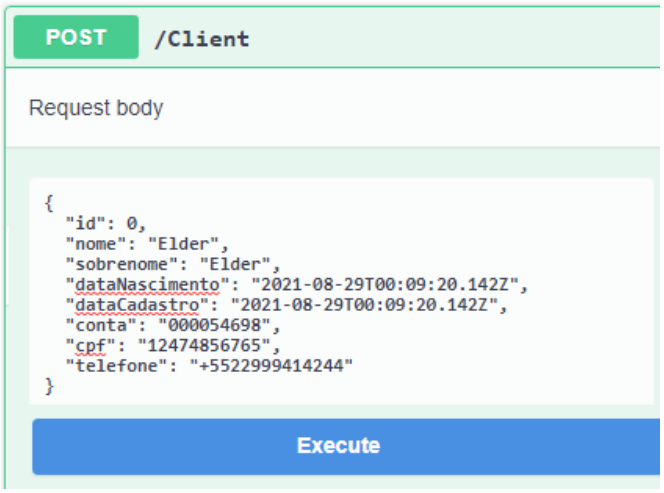

Figura 42 - Requisição post com cliente original

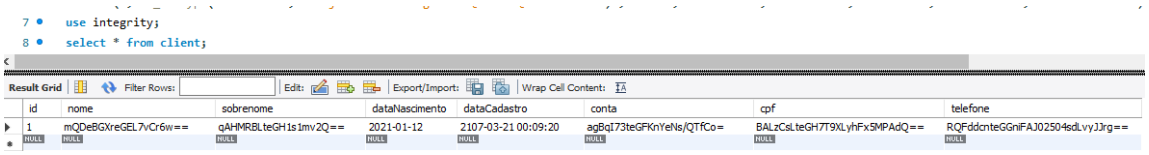

Figura 43 - Cliente salvo de maneira não integra na base

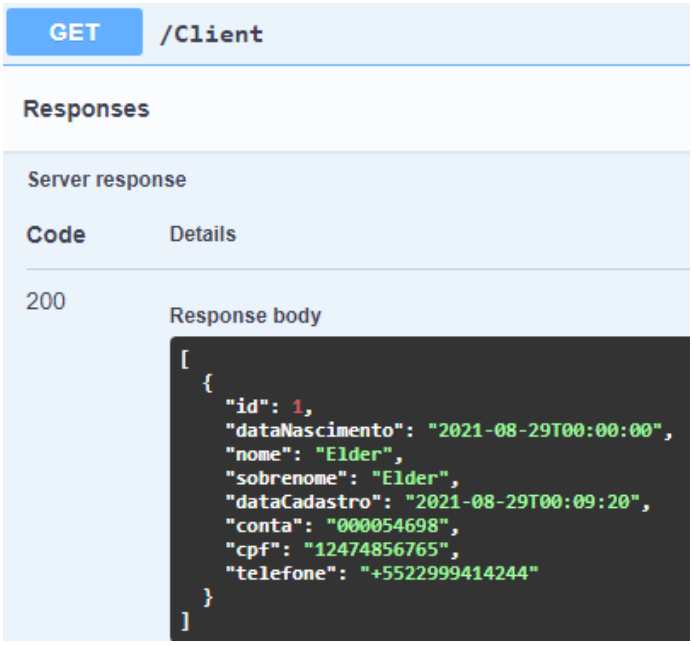

Figura 44 - Cliente recuperado de maneira integra na chamada get

7 Conclusão

A segurança da informação é um assunto que estará sempre em discussão; quanto mais aplicações, mais usuários. Quanto mais usuários, mais dados trafegados através da rede, e assim, cada vez mais necessário a proteção destes.

A LGPD revigorou as discussões e impôs regras para a coleta e utilização de dados sensíveis, obrigando empresas se adequarem sob pena de sansões financeiras e operacionais, sendo necessária a implementação de técnicas para garantir a segurança do dado armazenado.

Este trabalho apresentou uma proposta alternativa para burlar as dificuldades e limitações dos presentes nos SGBDs, adicionando uma camada de segurança e apresentando uma nova ótica para segurança de dados.

Através do exposto ao longo da proposta, observou-se que é perfeitamente possível atuar em segurança da informação abordando diferentes conceitos de seus pilares, no caso da Integrity, a integridade dos dados.

Ao persistir dados não íntegros e recuperá-los de maneira íntegra, através do algoritmo desenvolvido, provou-se que, em caso de invasão a uma base construída com o uso da biblioteca, temos uma camada de segurança adicional no caso de vazamento da informação, visto que ela não teria uso algum imediato para o atacante. Desta forma o conceito foi demonstrado de forma satisfatória através de implementação e teste da proposta.

Apesar de ser um excelente ponto de partida, a Integrity ainda é uma biblioteca em construção e, como todo projeto em sua fase inicial existem falhas e melhorias a serem feitas, e mais que isso, oportunidades para uma nova abordagem em segurança, criptografia e persistência de dados.

Como sugestão para trabalhos futuros, seria interessante abordar tipos de dados não tratados pela biblioteca, como tipos inteiros e decimal. Além disso, iniciar estudos e implementação de queries mais robustas como joins e betweens, além da implementação em outras linguagens como Java.

Referências

BRASIL. Lei nº 13.709, de 14 de agosto de 2018. **Lei Geral de Proteção de Dados Pessoais (LGPD).**, Brasília, DF, ago 2018.

BREWSTER, T. Forbes. **Forbes**, 2019. Disponivel em: <https://forbes.com.br/principal/2019/08/apple-confirma-premio-de-us-1-mi-para-quem-hackear-iphone/>. Acesso em: 20 Setembro 2021.

DESHPANDE, A. et al. DBCrypto: A Database Encryption System using Query Level Approach. **International Journal of Computer Applications**, Maio 2012. 27-32.

ESPN. ESPN. **ESPN**, 2021. Disponivel em: <https://www.espn.com.br/esports/artigo/_/id/9316521/dados-vazados-da-twitch-mostram-faturamento-de-streamers-como-gaules-ninja-e-critical-role>. Acesso em: 8 Outubro 2021.

FOWLER, M. **Padrões de Arquitetura de Aplicações Corporativas**. 1ª. ed. Porto Alegre: Bookman, 2007.

G1. G1. **G1**, 2019. Disponivel em: <https://g1.globo.com/economia/tecnologia/noticia/2019/04/04/dados-de-540-milhoes-de-usuarios-do-facebook-ficam-expostos-em-servidor.ghtml>. Acesso em: 25 Maio 2021.

G1. G1. **G1**, 2021. Disponivel em: <https://g1.globo.com/economia/tecnologia/noticia/2021/01/28/vazamento-de-dados-de-223-milhoes-de-brasileiros-o-que-se-sabe-e-o-que-falta-saber.ght>. Acesso em: 25 Maio 2021.

HINTZBERGEN, J. et al. **Fundamentos de Segurança da Informação Com base na ISO 27001 e na ISO 27002**. 1ª. ed. Rio de Janeiro: Brasport, 2018.

KIM, D.; SOLOMON, M. G. **Fundamentos de Segurança de Sistemas de Informação**. 1ª. ed. Rio de Janeiro: LTC, 2014.

LOREY, K.; BUCHMANN, E.; BÖHM, K. TEAL: Transparent Encryption for the Database Abstraction Layer. **CEUR Workshop Proceedings**, 13-17 Junho 2016. 145-152.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. NIST Technical Series Publications. **National Institute of Standards and Technology**, 2001. Disponivel em: <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf>. Acesso em: 25 Maio 2021.

NECHVATAL, J. et al. Report on the Development of the Advanced Encryption Standard (AES). **Journal of Research of the National Institute of Standards and Technology**, Gaithersburg, Maio-Junho 2001. 511-576.

ORACLE. Oracle. **Oracle**, 2021. Disponivel em: <https://www.oracle.com/br/security/database-security/data-masking/>. Acesso em: 14 Fevereiro 2021.

PITTA, P. E. B. et al. LGPD Compliance: A security persistence data layer. **Escola Regional de Redes de Computadores**, Porto Alegre, 25 Novembro 2020. 123-127.

STALLINGS, W. **Criptografia e Segurança de Redes Princípios e Práticas**. 6ª. ed. São Paulo: Pearson, 2014.

TANENBAUM, A. S. **Computer Networks**. 4ª. ed. [S.l.]: Campus, 2003.

WIKIPÉDIA. Wikipédia. **Wikipédia**, 2019. Disponivel em: <https://pt.wikipedia.org/wiki/Fun%C3%A7%C3%A3o_hash_criptogr%C3%A1fica>. Acesso em: 12 Junho 2021.

WIKIPÉDIA. Wikipédia. **Wikipédia**, 2021. Disponivel em: <https://pt.wikipedia.org/wiki/Cifra_Feistel>. Acesso em: 01 setembro 2021.