Simão de Sousa Dolores

Licenciatura em ciência e engenharia informática

Plataforma blockchain para a guarda genérica de dados

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador: Eng. Ricardo Pinto, Diretor do departamento de investigação, Agap2IT

Co-orientador: Prof. Dr. Miguel Monteiro, Professor do departamento de informática, FCT-UNL

Júri:

Presidente: Prof. Dra. Fernanda Barbosa

Vogais: Prof. Dr. Nuno Marques

Eng. Ricardo Pinto

Janeiro, 2020

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Simão de Sousa Dolores

Licenciatura em ciência e engenharia informática

Plataforma blockchain para a guarda genérica de dados

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador: Eng. Ricardo Pinto, Diretor do departamento de investigação, Agap2IT

Co-orientadores: Prof. Dr. Miguel Monteiro, Professor do departamento de informática, FCT-UNL

Júri:

Presidente: Prof. Dra. Fernanda Barbosa

Vogais: Prof. Dr. Nuno Marques
Eng. Ricardo Pinto

Janeiro, 2020

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Dedicatória

Agradeço aos meus pais pelo apoio incondicional em todos os momentos difíceis durante a minha trajetória académica. Sem eles a realização desta dissertação não seria possível.

Um agradecimento também aos meus colegas e ao meu orientador Eng. Ricardo Pinto por me terem apoiado durante a realização desta dissertação.

Resumo

Esta dissertação tem como finalidade a investigação de uma plataforma para a guarda e proteção de dados de forma a garantir a confidencialidade, integridade e a disponibilidade dos dados (tríade CIA).

Esta dissertação enquadra-se dentro de um projeto de dimensão maior, que utiliza a tecnologia blockchain para a guarda de bases de dados. A tecnologia blockchain tem várias propriedades interessantes, sendo que uma delas é permitir a descentralização de sistemas, isto é, não existe um único ponto de falha, providenciando assim um sistema com robusto e com elevada disponibilidade. Esta tecnologia tem sido maioritariamente aplicada na construção de sistemas financeiros distribuídos, porém, no caso onde esta dissertação se enquadra, pretende-se aplicá-la à guarda de bases de dados, onde todo o conteúdo guardado consiste no histórico de todas as operações CRUD, tanto sobre a estrutura de bases de dados, tal como sobre os dados em si.

Considerando que os dados estarão distribuídos por vários sistemas não confiáveis, o foco desta dissertação é cifrar as bases de dados guardadas em blockchain nesses sistemas de forma a que estas mantenham a confidencialidade dos seus dados, impedindo assim a sua consulta indevida. A encriptação de dados impede a sua respetiva pesquisa e ordenação, pelo que esta dissertação também estuda potenciais estratégias para manter um equilíbrio entre a confidencialidade dos dados e a capacidade de os pesquisar e ordenar.

Existem poucas implementações de bases de dados cifradas, e todas estas apresentam diferentes vantagens e desvantagens. Pretendemos nesta dissertação explorar este estado de arte, e daí desenvolver um mecanismo de persistência seguro enquadrado com o sistema *blockchain* envolvente, e que maximize o potencial de pesquisa de dados relevando o mínimo sobre os mesmos.

Palavras chave: Blockchain, Base de dados, Tríade CIA, Transações, Segurança.

Abstract

This dissertation's goal is to investigate a platform to store and protect data in a way that it ensures confidentiality, integrity and the availability of the data (CIA).

This dissertation fits in a larger project that utilizes blockchain technology to store multiple databases. The blockchain technology has various interesting properties, one of them being the fact that it allows for decentralized systems, which means, there is not a single point of failure, providing then a robust system with high availability. This technology has been normally applied to constructing distributed financial systems, however, in the event that this dissertation fits into, the aim is to apply it to store databases, where all of its contents consists of the history of all CRUD operations, both the structure of the database and the data itself.

Considering that the data will be distributed amongst various un-trustworthy systems, the aim of this dissertation is to encode the stored databases in blockchain in these systems so that they can maintain the confidentiality of their data, stopping unwanted access. The encryption of data stops its own search and sorting, in which this dissertation also studies potential strategies to maintain the balance between confidentiality of data and the capacity to search and sort them.

There exist few implementations of encrypted data bases and all of them show different advantages and disadvantages. In this dissertation we want to explore these art states and from there develop a safe persistence mechanism that fits within the blockchain system, and that it maximizes the potential of the search of data while revealing the bare minimum about them.

Keywords: Blockchain, database, CIA, Transactions, Security.

Conteúdo

Lista de Figuras

Lista de Tabelas

Siglas/Abreviaturas

GDPR	General Data Protection Regulation
OPE	Order Preserving Encryption
ORE	Order Revealing Encryption
CIA	Confidentiality, Integrity, Availability
ECB	Electronic Codebook
CBC	Cipher Block Chaining
CTR	Counter
AES	Advanced Encryption Standard
DES	Data Encryption Standard
RSA	Rivest–Shamir–Adleman
ECC	Eliptic Curve Cryptography
DSS	Digital Signature Standard
DET	Deterministic
RND	Random
NIST	National Institute of Standards and Technology

1 Introdução

A proteção e a privacidade de dados pessoais são temas da maior importância. Na atualidade, muita informação importante sobre as populações encontra-se distribuída por diversas organizações, como, por exemplo, a saúde, as forças de segurança, o exército, entre outras. A revelação desta informação para o público pode ser prejudicial para os seus proprietários. Um sistema seguro não só deve proteger a confidencialidade dos dados, mas também garantir que a mesma não é alterada, e deve estar sempre disponível quando requisitada.

Os sistemas atuais encontram-se de tal maneira interligados, que a informação das sociedades se encontra distribuída e dispersa ao longo de variadíssimos sistemas compostos pelos mais diversos componentes e implementados pelas mais diversas tecnologias. Numa situação destas, a informação tende a ficar dispersa, que nem água dento de uma canalização sem fim. Porém, contrariamente à água, a fuga de informação pode ser extraordinariamente prejudicial. Deve, portanto, procurar-se desenvolver sistemas seguros, que garantam a confidencialidade dos dados, tal como a sua integridade e disponibilidade. Esta dissertação foca-se neste tema.

De seguida, é feita uma introdução à segurança das bases de dados e à importância de os proteger. Tendo em conta que o foco desta dissertação está contido dentro de um sistema maior, o qual segue um paradigma *blockchain*, é também feita uma introdução a

este tema, sendo identificados os seus processos e o seu potencial para diferentes aplicações. Terminaremos com uma explicação de como estes dois tópicos irão correlacionar-se para a resolução do problema apresentado.

1.1 Segurança de Sistemas Informáticos

O valor e a quantidade da informação dos sistemas têm vindo aumentar, mas ao mesmo tempo esta informação tem ficado cada vez mais vulnerável à variedade de ameaças existentes, como o acesso e uso não autorizado, alteração e destruição dos dados [1].

As bases de dados atuais contêm informação importante e confidencial que os utilizadores não querem expor ao resto do mundo, como, por exemplo, a sua morada, as transações bancárias, número de telefone, entre outras. Esta informação encontra-se distribuída e replicada em diversos sistemas diferentes. Visto que as bases de dados são um recurso principal para a guarda e obtenção de informação, devemos aplicar políticas e procedimentos para a sua segurança e integridade. Estas, focam-se essencialmente em garantir a guarda dos dados tal como a sua transmissão segura. Esta segurança é fornecida através de técnicas criptográficas.

Existem várias ameaças às bases de dados. Uma delas, por exemplo, consiste na perda de disponibilidade do sistema, resultando no impedimento de os utilizadores conseguirem aceder à sua informação. [2].

Outra ameaça é, por exemplo, o abuso de poder, o administrador do sistema pode ter interesse em alterar dados em seu benefício, ou em recolher informação para atividades indevidas como *data-mining* ou a partilha com outros *websites*.

Data-mining é o processo de fazer *queries* para extrair informação de uma grande quantidade de informação numa base de dados. Este processo pode causar problemas de segurança, expondo informação privada do utilizador. Por exemplo, nos Estados Unidos, um empregado da empresa Target recebeu uma reclamação de um cliente devido a

terem enviado cupões de roupa para bebés à sua filha adolescente. Veio-se a descobrir que a filha deste cliente estava grávida. O sistema Target conseguiu obter esta informação a partir de *data-mining* [3].

Para impedir o acesso indevido à informação, são necessárias várias táticas e práticas, que requerem conhecimento e vontade, necessitam de tempo e dedicação, impõem um acompanhamento contínuo, e constituem um custo. Por todas estas razões, as organizações tendem a desleixar-se no cumprimento das mesmas. A situação encontra-se de tal forma grave que desde o ano de 2005 houve quase 12 mil milhões de registos comprometidos [4], e estes valores tem vindo a aumentar de ano para ano. Por exemplo, em 2017 foi um ano onde os setores da saúde e de serviços de seguro médico foram especialmente atacados.

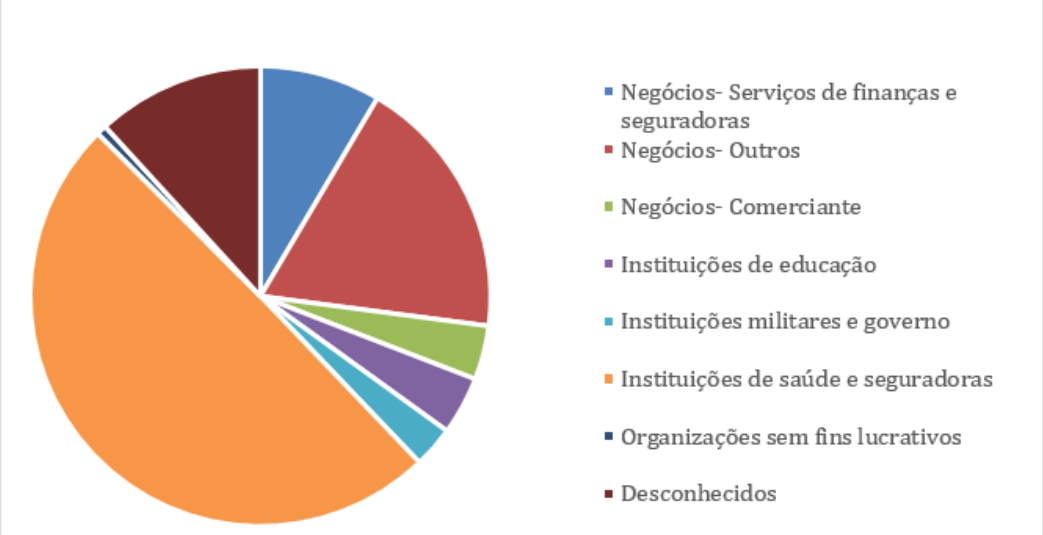

Figura 1.1- Percentagem de registos comprometidos por setor em 2017, obtidos em [4]

É importante referir que esta informação é referente a apenas casos que foram tornados públicos. Ficando de fora os casos que não foram comunicados.

Devido a estes problemas, no início de 2018 foram implementadas na Europa leis para a proteção de dados (GDPR) [5], que proporcionam aos cidadãos Europeus um maior controlo sobre como os seus dados estão a ser processados.

Nestas condições o utilizador tem direito a [6]:

1. Receber informação de como os seus dados estão a ser processados;
2. Ter acesso aos seus dados pessoais;
3. Requerer que um provedor de serviço transmita os seus dados para outro provedor;
4. Ser esquecido;
5. Ser necessário o seu consentimento para o processamento dos seus dados pessoais;
6. Ter os seus dados sensíveis seguros com políticas e práticas adequadas para o efeito;
7. Ser informado caso haja perda ou uso indevido aos seus dados.

Devido a todos os motivos supra mencionados, este tema é um tema complexo e em constante mudança. Se até empresas de grande dimensão sofrem ataques bem-sucedidos, é compreensível que as outras empresas achem este tema assustador e complexo. Para tal, devem-se aplicar *standards* de segurança que facilitem a proteção dos dados, os quais devem ser logo pensados desde a sua conceção [1]. Com a capacidade de processamento dos computadores a aumentar de ano para ano, as chaves dos algoritmos de encriptação são mais vulneráveis a ataques. Existem técnicas que facilitam a descoberta da chave secreta para revelar o valor cifrado. Este processo tem de ser adaptável, o que é seguro no dia de hoje pode já não ser seguro no dia de amanhã.

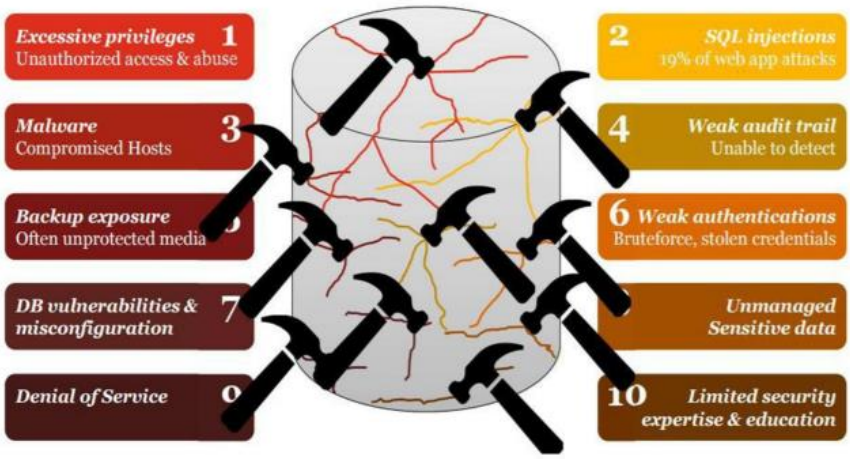

Figura 1.2- Ameaças a base de dados obtido em [1]

Na Figura 1.2- Ameaças a base de dados obtido em Figura 1.2, retirado do artigo em [1], podemos observar os ataques mais comuns às bases de dados, onde o segundo ataque mais comum é *SQL injections*. Existe uma variedade de bibliotecas que nos protegem contra *SQL injections*.

Infelizmente, existe muita ignorância e negligência neste tema. A abordagem típica das organizações consiste em restringir o acesso aos canais de comunicação através de uma *intranet* e um serviço de diretório. Isto é, fortificar as instalações com uma barreira que separa a infraestrutura do mundo exterior. O problema surge quando um atacante obtém acesso interno à rede.

Consultoras depararam com vários sistemas informáticos sem segurança ou com uma falsa noção de segurança. São encontradas regularmente *password*s sem estarem cifradas, ou *password*s que se encontram cifradas de tal maneira que é possível a uma pessoa com acesso privilegiado de decifrar as mesmas.

Qualquer sistema que se pretenda seguro, terá que ter como foco a criptografia, a qual consiste em técnicas que procuram esconder informação secreta, permitindo que a mesma seja revelada apenas a quem tiver a devida autorização. Estas técnicas existem

desde a antiguidade, estando normalmente associadas a atividades militares e diplomáticas. As mesmas surgiram da necessidade de comunicar em segredo ao longo de grandes distâncias e através de mensageiros. Nos tempos modernos, esta necessidade veio a crescer significativamente, principalmente com o surgimento da *internet*. Para tal foram desenvolvidas diferentes técnicas criptográficas, como a criptografia simétrica, a criptografia assimétrica, as assinaturas digitais, as funções *hash*, entre outras [7].

Em suma, a criptografia consiste na transformação de dados legíveis em dados ilegíveis, de forma a que estes fiquem seguros. Dados cifrados só poderão ser recuperados recorrendo ao uso de uma chave ou um segredo.

Contudo, dados cifrados ficam ilegíveis impossibilitando a sua pesquisa e ordenação. Esta falta levou ao desenvolvimento de várias técnicas no sentido de contornar estas limitações, como a OPE (*Order-preserving encryption*), a ORE (*Order-revealing encryption*), e a criptografia homomórfica.

Portanto para termos uma base de dados segura, temos três principais objetivos: (1) garantir a confidencialidade dos dados, permitindo que os mesmos só possam ser lidos e processados por utilizadores ou entidades autorizadas. (2) garantir a integridade destes dados, impondo que a informação só possa ser alterada por utilizadores autorizados, e que as suas alterações sejam rastreáveis. (3) assegurar a disponibilidade dos dados, isto é, que o acesso aos mesmos não seja comprometido. Estes três objetivos são conhecidos no mundo da segurança como tríade CIA (“Confidenciality”, “Integrity” e “Availability”).

1.2 Enquadramento com Sistemas *Blockchain*

Durante a última década, surgiu uma tecnologia inovadora denominada por *blockchain*. A tecnologia *blockchain* é, essencialmente, uma base de dados de registos distribuídos remotamente. Estes registos são transações ou eventos digitais que são executados e distribuídos por todos os seus participantes. Cada transação é verificada por

consenso da maioria dos participantes do sistema. Transações que entram na *blockchain* não podem ser eliminadas. Nestes termos, a *blockchain* contém todas as transações que foram aceites pelos participantes.

A *Bitcoin* foi a primeira tecnologia a apresentar o uso do paradigma blockchain, usada para a construção de um sistema financeiro descentralizado (peer-to-peer). A *Bitcoin* foi construída com o intuito de não ser uma moeda centralizada. Este *blockchain* sobreviveu ao teste do tempo. Com milhares de milhões de dólares em *Bitcoin*s guardados dentro do seu *Blockchain*, onde o acesso ao mesmo é público e acessível para qualquer um, não houve até hoje nenhum ataque que conseguisse comprometer o sistema.

A tecnologia *blockchain* consiste num conjunto de blocos que estão ligados entre si, onde cada bloco contém um conjunto de transações que são verificadas pelos *miners*. *Miners* são as entidades da rede que tratam de produzir e validar os blocos. Cada bloco contém o *hash* (resumo) do bloco anterior, ligando assim os dois blocos, garantindo que para alterar um bloco teríamos de alterar todos os blocos subsequentes. Em resumo, os utilizadores da tecnologia *blockchain* vão enviar as suas transações para um dos *miners*, o *miner* irá validar a transação e inclui-la num bloco. E de seguida vai enviar o bloco para o resto dos *miners* para serem validados.

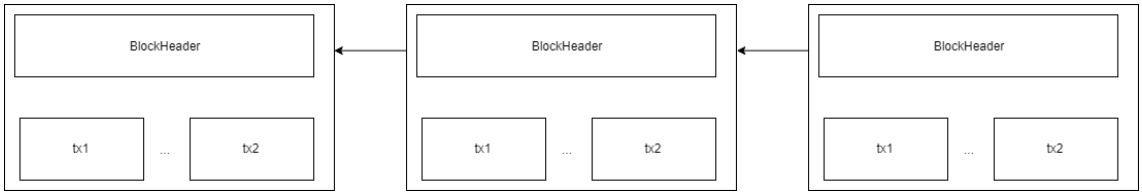

Figura 1.3- Estrutura simples da blockchain

Depois do sucesso da *Bitcoin*, surgiram outras implementações que utilizam esta tecnologia, por exemplo, a moeda Monero, que permite fazer transações sem serem visíveis o recipiente e o remetente.

Apesar da aplicação inicial da tecnologia *blockchain* ter sido para fins financeiros, nada impede outro tipo de aplicações, como para cadeias de abastecimento, seguradoras, registos médicos, entre muitos outros [8].

A *Bitcoin* utiliza *proof-of-work* para a validação de blocos e por motivos de segurança. P*roof-of-work* providencia uma propriedade bastante importante para a segurança da *blockchain*, esta propriedade denomina-se por imutabilidade [9]. Este método faz com que seja preciso uma capacidade exorbitante de computação, onde um atacante precisa de ter mais de 51% do poder de computação da rede para poder ter uma influência nefasta na produção de blocos.

P*roof-of-work* pode ser pensado como um *puzzle* matemático, que depende da informação de cada bloco, onde se alterarmos um bloco será necessário recalcular o *puzzle*. E visto que os blocos contêm um resumo do bloco anterior, seria necessário recalcular o *puzzle* de todos os blocos sucessivos caso se quisesse alterar um bloco anterior.

Este método tem um elevado consumo de energia, tendo sido estimado que, no mínimo, a rede de *miners* gasta cerca de 23.38 TWh. Este valor foi calculado com base no *hashrate* e da energia mínima para obter esse *hashrate* [10].

Outras criptomoedas utilizam outros protocolos para fazer a verificação dos blocos sem utilizar o mesmo método que a *Bitcoin*. Por exemplo a EOS.IO, utiliza o método *delegated proof-of-stake* que consiste em eleger um conjunto de 21 *miner*s que, em rede, determinarão a sequência correta de blocos, sem recorrer ao método *proof-of-work*.

Esta criptomoeda providencia outra funcionalidade que consiste na execução de código em blockchain, denominado por *smart contracts*. A *Bitcoin* permite também estes contractos, mas são muito limitativos.

Outra característica da EOS.IO é que os blocos são produzidos a cada meio segundo, enquanto que no caso da *Bitcoin*, a produção de blocos é a cada 10 minutos [11].

Podemos concluir que existem *blockchains* com propriedades distintas. Foi apresentada a *Monero* por apresentar propriedades de privacidade, a EOS.IO por apresentar uma variedade de funcionalidades e um desempenho bastante superior à *Bitcoin* e, por fim, a *Bitcoin* por esta ser a primeira a resolver o problema de *double-spending* e outras criptomoedas terem como base esta criptomoeda.

1.3 Motivação

As características inovadoras da tecnologia *blockchain*, o seu impacto na sociedade, e os valores monetários significativos que as criptomoedas têm vindo a demostrar, foram fatores importantes para a motivação deste estudo. Além disto, houve uma evolução significativa de outras *blockchain*s para com a *Bitcoin*, como EOS.IO ou Ethereum, os quais apresentam mais funcionalidades, e funcionalidades mais complexas.

A finalidade desta dissertação é investigar mecanismos que permitam cifrar dados de tal maneira que os mesmos se mantenham pesquisáveis, revelando o mínimo sobre os mesmos. Onde vamos querer ter várias camadas de segurança, utilizando como fundamento os *onion-routing*, para permitir ter vários níveis de segurança onde a camada mais acima é equivalente à camada mais segura e a camada mais abaixo a menos segura.

Para poder fazer *queries* sobre os dados encriptados é necessário preciso ser feito um estudo sobre as OPE (*Order preserving encription*), ORE (*Order revealing encription*), encriptação homomórfica e formas de fazer pesquisas de palavras sobre dados encriptados sem que esta revele o seu valor.

1.4 Objetivos da Dissertação

Nos dias de hoje, existem sistemas que contêm uma grande quantidade de informação valiosa. Garantir a segurança desta informação é importante, mas tem-se verificado cada vez mais furtos, como referido na secção 1.1. O que aconteceu recentemente, e foi muito falado nos meios de comunicação social, foi o furto de informação na empresa Facebook, de cerca de 50 milhões de registos [12]. Há rumores que a utilização desta informação foi utilizada para influenciar as eleições norte americanas. Com este exemplo conseguimos perceber como a manipulação desta informação pode colocar em risco o nosso dia-a-dia e mesmo o das pessoas que não utilizam o sistema.

Sabendo que a *blockchain* contém características que facilitam o cumprimento da tríade CIA, providenciando uma sólida abordagem para a guarda de dados, simplificando a segurança dos mesmos, e reduzindo o esforço externo para os manter, pretendemos utilizá-la para esse efeito.

O sistema deve:

- Promover a confidencialidade de todos os dados. Toda informação deve ser previamente cifrada do lado do cliente. Apenas pode não estar cifrada por configuração prévia do administrador.
- Fornecer mecanismos de proteção de dados que impeçam os administradores do sistema, e ao próprio sistema em si, de os conseguir aceder.
- Fornecer mecanismos de pesquisa sobre os dados cifrados revelando informação mínima sobre os mesmos.
- Manter um histórico de todas as alterações que ocorreram no sistema e deve ser impossível de remover ou alterar este histórico.

As *blockchain*s cumprem por natureza com alguns pontos identificados em cima. A *blockchain* pretendida para esta solução apresenta muitas semelhanças com os outros *blockchain* existentes.

Apesar disso a *blockchain* proposta tem as seguintes diferenças das outras:

- O conteúdo da *blockchain* será um histórico de operações SQL
- Os dados guardados estão cifrados
- Os *miner*s utilizam uma base de dados relacional para guardar os dados dos utilizadores.

Em suma, o objetivo desta dissertação é ir construindo as diversas bases de dados a partir da leitura das diversas transações que estão contidas na blockchain. De seguida é necessário garantir a confidencialidade dos dados, para tal será feita a cifra dos mesmos. Por fim, para se efetuar pesquisas sobre os dados cifrados será necessário construir infraestruturas para nos auxiliar neste processo.

2 Trabalho Relacionado

Nesta secção é apresentado de forma sumária alguns trabalhos relacionados, escolhidos como os mais relevantes e correlacionados com esta investigação. O trabalho relacionado é apresentado em três secções distintas, sendo a primeira dedicada a sistemas *blockchain*, a segunda dedicada a algoritmos criptográficos, e a terceira é dedicada a base de dados com dados cifrados e que permitam pesquisas sobre os mesmos.

2.1 Sistemas Blockchain

Como referido na secção 1.1.3, apenas foram escolhidas três *blockchain*s. A primeira é a *Bitcoin*, que requer uma explicação mais extensa e detalhada para explicar o paradigma geral dos sistemas *blockchain*. De seguida será apresentada a *Monero* que contém propriedades interessantes de privacidade. E por fim, é apresentada a EOS.IO, que em vez de ter apenas transações monetárias, também permite a execução de código. Estas ações são definidas pelos utilizadores. Podem ser, por exemplo, movimentos de um jogo.

2.1.1 Bitcoin

A *Bitcoin* foi inventada em 2008 por alguém com o pseudónimo de Satoshi Nakamoto, pela publicação de um *paper* intitulado por "*Bitcoin*: A Peer-to-Peer Electronic Cash System," [13].

No seu livro 'Mastering Bitcoin' [14], Andreas Antonopoulos faz uma análise aprofundada sobre a história da *Bitcoin*, a sua estrutura, e o seu funcionamento. Iremos recorrer várias vezes a este livro dentro desta secção.

Nakamoto combinou várias invenções antigas como *b-money* e *HashCash* para criar uma moeda virtual completamente descentralizada sem depender de uma autoridade central. Foi o primeiro a resolver o problema do *double-spending*, sem recorrer a uma autoridade centralizada.

A *Bitcoin* utiliza um sistema *peer-to-peer*, onde cada nó do sistema contém uma réplica completa do blockchain, onde estão incluídas todas as transações, semdo o seu tamanho atual maior que 100GB. Para dispositivos com baixa capacidade de armazenamento, Nakamoto criou uma funcionalidade que permite sumarizar o blockchain, utilizando *merkletree* o utilizador apenas precisa de guardar o *hash* de cada transação.

No artigo de conferência em [15], *double-spending* é definido por alguém poder emitir uma transação em paralelo e transferir o mesmo montante para destinatários diferentes. Num sistema bancário centralizado o problema pode ser resolvido, adicionando um número único a uma transação e assim o sistema consegue averiguar que a transação só é feita uma única vez. O valor total da *Bitcoin* à data da escrita desta dissertação é de 63 mil milhões de US dólares. A maior transação na rede *Bitcoin* foi de 150 milhões de US dólares.

Em 2011, Satoshi Nakamoto passou a responsabilidade do desenvolvimento do código para a comunidade.

A *Bitcoin* foi contruída com o uso de uma variedade de conceitos e tecnologias. Os utilizadores podem transferir *Bitcoin*s pela rede para fazer todo o tipo de atividades que as moedas físicas também fazem, como, por exemplo, comprar e vender mercadorias, transferir dinheiro para entidades ou organizações. *Bitcoin* é completamente virtual. As moedas estão implícitas em transações que transfere um valor de um remetente para um destinatário. Os utilizadores da *Bitcoin* utilizam chaves secretas que permitem provar que possuem *Bitcoin*s na rede da *Bitcoin*. Ao possuir estas chaves, permite o utilizador a assinar transações para desbloquear o valor e transferi-lo para outro utilizador. Estas chaves normalmente são guardadas em carteiras digitais que estão localizadas num computador ou telemóvel. Basta possuir a chave para uma determinada transação para

poder gastar o valor, tendo assim o controlo total do dinheiro nas mãos de cada utilizador.

No caso das carteiras digitais, estas geram chaves automaticamente, onde esta, irá gerar duas chaves inicialmente, onde uma será o *Bitcoin address* (chave pública) e a chave para assinar transações (chave privada). Esta assinatura prova que o montante de uma transação pertence à pessoa que a assinou. Estas chaves após o momento de criação não estão apresentadas em nenhuma rede ou registada em nenhum sistema.

Figura 2.1- Bitcoin wallet, obtido em [14]

Apenas quando houver uma transação para o *Bitcoin address*, este endereço será reconhecido pela rede *Bitcoin*. Este *Bitcoin address* será como o NIB bancário, onde poderemos entregar a qualquer pessoa ou organização para ser feito uma transferência bancaria para a nossa conta. E note que sabendo o NIB o utilizador não terá acesso ao nosso dinheiro. Do mesmo modo, para ser feita uma transferência de *bitcoin*s será necessário primeiro enviar o *Bitcoin address* ao destinatário.

Visto que o utilizador é a única pessoa que terá as chaves, a perda destas irá ser equivalente a perder as *Bitcoin*s associadas a essas chaves. Por isso os utilizadores são recomendados a replicar as suas chaves e guardá-las em sítios seguros.

*Bitcoin*s podem ser obtidas:

- Por um amigo que tenha *Bitcoin*s e pode comprar diretamente a ele.
- Utilizar *websites* para encontrar vendedores ao pé de nós.

- Vender um produto por *Bitcoin*s.
- Existe caixas de multibanco que permite fazer a conversão.

Antes de ser feita a troca por uma moeda física deve ser feito a verificação da taxa de cambio. O valor desta taxa é simplesmente calculado baseado na oferta e procura da moeda.

Como referido anteriormente, a tecnologia *blockchain* é um sistema distribuído *peer-to-peer*, a *Bitcoin* apresenta as mesmas propriedades. As *bitcoin*s são criadas pelo processo denominado por "*mining*", que envolve os "*miners*" competirem para descobrir uma solução de um problema matemático, enquanto valida as transações de *Bitcoin*. A dificuldade do problema matemático é ajustada para que a solução seja encontrada de 10 em 10 minutos. Isto permite que seja produzido um bloco de 10 em 10 minutos, e o *miner* que encontrar a solução será recompensado com *Bitcoin*s. Este processo alem de recompensar o *miner*, está a criar novas moedas.

Este problema matemático tem o nome de *proof-of-work*, onde o processo de obter a solução é difícil e validar se a solução é correta é fácil. O processo basicamente é composto por 2 passos:

1. Alterar um valor do bloco *header* arbitrariamente – nonce
2. Fazer o *hash* do bloco até obter um certo número de zeros

Supondo que queremos efetuar uma transação de *Bitcoin*s para outra pessoa, teremos de esperar em média 5 minutos para que esta transação seja confirmada. Uma transação confirmada significa que esta transação foi verificada pelos *miner*s e incluída num bloco.

No caso da *Bitcoin* os seus blocos são constituídos por uma série de transações validadas pelos *miner*s e como referido anteriormente, os blocos estão interligados em corrente, onde para alterar um bloco terá de recalcular o *proof-of-work* para todos os seus sucessores. Isto faz com que os blocos mais antigos sejam mais seguros do que os recentes.

Uma forma ingénua de tentar atacar este sistema seria alterar as transações para serem destinadas ao atacante. Esta tentativa iria ser inútil, para isto este atacante teria de obter a solução correta do *proof-of-work,* que é bastante difícil de obter. E mesmo que o atacante conseguisse atingir a solução correta, e transmitisse para a rede um bloco com transações inválidas, este bloco iria ser verificado pelos outros *miner*s e considerado invalido e descartado da *blockchain*.

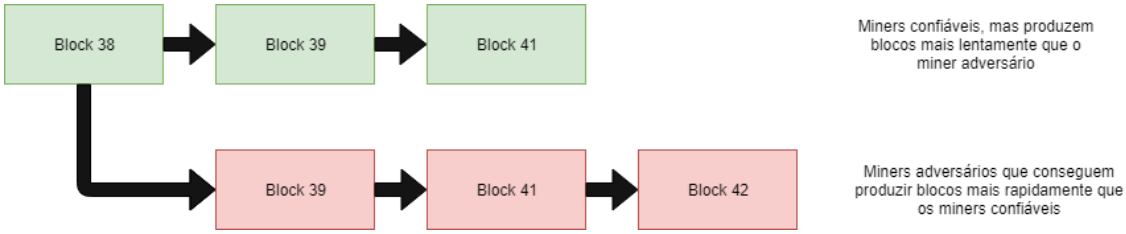

Figura 2.2- Blockchain 51% attack

Um possível ataque ao *proof-of-work* seria o ataque de 51%, onde o atacante tem 51% da capacidade de computação da *blockchain* e decidem construir uma *blockchain* em paralelo não contendo uma determinada transação que já foi validada na *blockchain*. Esta *blockchain* vai ser construída em paralelo a partir do bloco onde o atacante pretende omitir a transação e irá tentar criar uma corrente maior que a *blockchain* principal. *Blockchain* principal será substituída pela *blockchain* criada pelos atacantes caso esta seja maior que a *blockchain* principal. Isto acontece devido aos nós da rede aceitarem como a *blockchain* principal a que seja mais longa e que tenha as suas transações válidas. Com este ataque é possível cancelar a transação previamente feita e gastar o dinheiro outra vez que foi gasto previamente (*double-spending*). Para ser efetuado um ataque deste tipo seria necessário um grande investimento em *hardware*, o que provavelmente não seria um ataque lucrativo.

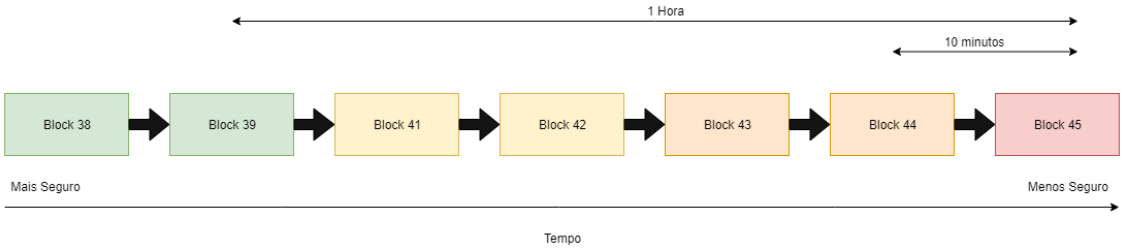

Figura 2.3- Blockchain

Devido a este problema, se quisermos ter a certeza que a nossa transação foi aceite em transações de grandes valores, só se deve aceitar a transação como confirmada ao fim de ter se criado pelo menos 6 blocos na *blockchain*. Para tal terei de esperar entre 50-60 minutos. Ao fim de 6 blocos é considerado irrevogável, porque requeria uma grande quantidade computacional para invalidar e recalcular o *proof-of-work* de 6 blocos.

Transações são maioritariamente constituídas por *inputs* e *outputs*. Cada transação é constituída por um ou mais *inputs* que é a quantidade de dinheiro que queremos gastar. Além de *input*s, as transações contêm *outputs*, onde são referidos os destinatários do nosso *input* e o montante. Além destes, também é constituída por uma taxa de transação (*transaction fee*) que é uma pequena quantia para incentivar o *miner* a validar esta transação. Cada *input* vai ter uma prova como esses *inputs* pertencem ao utilizador com a assinatura do utilizador.

No sistema *Bitcoin*, gastar *Bitcoin*s é equivalente a assinar uma transação que transfere o dinheiro de uma transação antiga para uma nova. Ou seja, para gastar estas moedas temos de obter *output*s de transações anteriores que recebemos e que ainda não foram gastos, de seguida, pô-los como *input*s assinando-os como prova que este *output* me pertence e, por fim, criar os *output*s com as quantidades e o *Bitcoin address* para respetivos utilizadores.

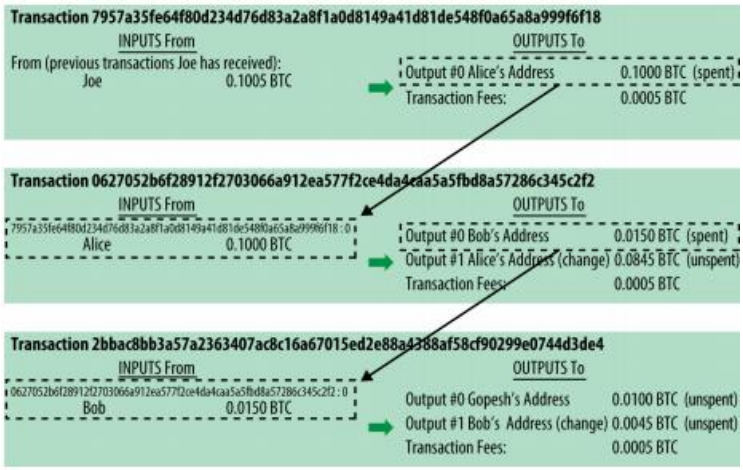

Figura 2.4-Transações Bitcoin, obtido em [14]

Como se pode observar na Figura 2.4, se tivermos um *output* destinado a nós de 10 *Bitcoin*s e queremos enviar apenas 5 *Bitcoin*s, teremos de criar o troco para nós, ou seja, terei de criar 2 *output*s um com 5 para o destinatário que pretendemos enviar 5 *Bitcoin*s e outro *output* com 5 *Bitcoin*s para nós.

Mas neste caso a nossa transação não teria taxa de transação, o que faz com que provavelmente, os *miners* não vão querer validar a nossa transação, a taxa de transação é uma maneira de incentivar o *miner* a validar a minha transação. Esta taxa é mais uma propriedade que faz com que aumente a segurança da *blockchain*, fazendo com que o *miner* receba uma recompensa caso valide a transação corretamente

Para saber qual deveria ser o valor da taxa de transação, existe *websites* que nos dizem o valor por cada *byte* que a transação tem. Quanto maior for a taxa de transação por *byte* mais rápido esta transação será incluída no bloco.

É importante ter atenção, que mesmo que uma transação não seja validada devido à taxa de transação ser baixa, esta transação não irá ficar presa. Pode se usar os mesmos *outputs* para criar a mesma transação, mas com uma maior taxa de transação. Isto é possível porque enquanto a transação não seja validada e incluída num bloco é como se nunca tivesse acontecido. Assim quando a transação com uma maior taxa de transação for inserida num bloco, a transação mais antiga será invalida e descartada pelos *miners*.

Podemos concluir que a *Bitcoin* tem alguns problemas de escalabilidade, visto que as transações só são validadas de 10 em 10 minutos. Nem todas as transações serão validadas porque os blocos têm um tamanho estável, ou seja, não aumenta o tamanho dos blocos com o número de transações à espera de serem validadas. Com isto, um aumento de números de transações para serem inseridas num bloco, aumenta proporcionalmente o tamanho de espera para que as transações sejam validadas. Concluindo que *Bitcoin* não apresenta um sistema escalável, podendo esperar 10 minutos ou mais para uma transferência ser validada e incluída num bloco.

Este sistema também não garante a privacidade de quem a usa. Todas as transações podem ser vistas por qualquer utilizador, sendo proprietário ou não e ainda é possível seguir desde a origem da moeda *Bitcoin* até aos seus destinatários.

2.1.2 Monero

A *Monero* é uma moeda digital, lançada pelo Nicolas van Saberhagen mais conhecido pela alcunha de "thankful_for_today" a 18 de abril de 2014. Inicialmente o seu nome era *BitMonero*, no entanto foi alterado quando o criador abandonou o desenvolvimento. Isto aconteceu porque a comunidade que apoiava a *Monero* quis fazer melhorias, mas o seu criador não concordou. Sete pessoas dessa comunidade tiraram-lhe o poder e ficaram no "comando". Esta moeda foi construída a partir do zero e não um clone da *Bitcoin*, o que leva à dificuldade de produzir *wallets* e outras aplicações para esta moeda.

No artigo "CryptoNote v2" de Nicolas van Saberhagen [16], Saberhagen faz uma análise da estrutura e do funcionamento desta criptomoeda. Iremos recorrer este artigo dentro desta secção para explicar os diferentes conceitos.

Cada conta possui duas chaves privadas a *spend key* e a *view key*. A *spend key* permite gastar os fundos da conta. Esta chave pode ser utilizada para originar todas as outras chaves da conta. A *view key* é utilizada para ver todas as transferências que o utili-

zador recebeu. As transações são privadas, mas opcionalmente podem tornar-se transparentes, ou seja, se o utilizador assim desejar, pode partilhar a sua *view key* para que outras pessoas possam ver as suas transferências recebidas.

Cada conta também tem duas chaves públicas correspondentes à chave privada *spend key* e à chave privada *view key*. A *public view key* é utilizada para gerar um endereço único, que se denomina por *stealth address*, o qual permite utilizar a *private view key* para determinar se uma data transação é para a respetiva conta. Ademais, a *public spend key* tem a funcionalidade de verificar se as transações são válidas.

Os blocos têm tamanho dinâmico, ou seja, podem aumentar de tamanho consoante o fluxo de transações, havendo também um incentivo para os *miners* aumentarem o tamanho dos blocos de uma forma gradual. Os blocos são produzidos de 2 em 2 minutos, sendo gerados pelos *miners* de acordo com um modelo de consenso equivalente ao da *Bitcoin*, *proof-of-work*. Neste caso, a *Monero* utiliza o algoritmo *Cryptonight* para gerar os *hash* dos blocos, este algoritmo pode ser processado de forma eficiente por GPUs e CPUs. Isto acontece devido ao algoritmo *Cryptonight* requerer no mínimo 2mb de memória. Se acedermos à memória de um GPU GDDR5 é relativamente mais lento que aceder à memória de um CPU, isto faz com que CPU consiga ter um bom desempenho utilizando este algoritmo.

Semelhantemente à *Bitcoin*, os *miner*s escolhem e validam as transações inserindo-as num bloco, procedendo à execução do algoritmo o *Cryptonight* até conseguirem obter a solução correta.

O grande objetivo da *Monero* consistem em que as transações apresentem as seguintes propriedades [17]:

- Não serem rastreáveis, ou seja, que as transações não formem uma corrente de transações como na *Bitcoin*, onde começando de uma transação conseguimos rastrear até a sua origem.

- Não terem nenhuma ligação, ou seja, não ser possível associar um utilizador a uma transação.
- Serem privadas, ou seja, não ser possível determinar o conteúdo das transações.

Ring Signatures

As *ring signatures* permitem esconder a identidade da pessoa que pretende enviar os seus fundos. As transações são bastante parecidas às transações da *Bitcoin*, a *ring signature* é construída pela combinação de vários *output*s existentes (denominados por *decoys*), com o *output* verdadeiro do utilizador que pretende fazer a transação [18]. Estes *outputs* são obrigatoriamente *outputs* válidos que podem ser gastos. Com este processo, é possível ao utilizador de enviar uma transação sem comprometer a sua identidade.

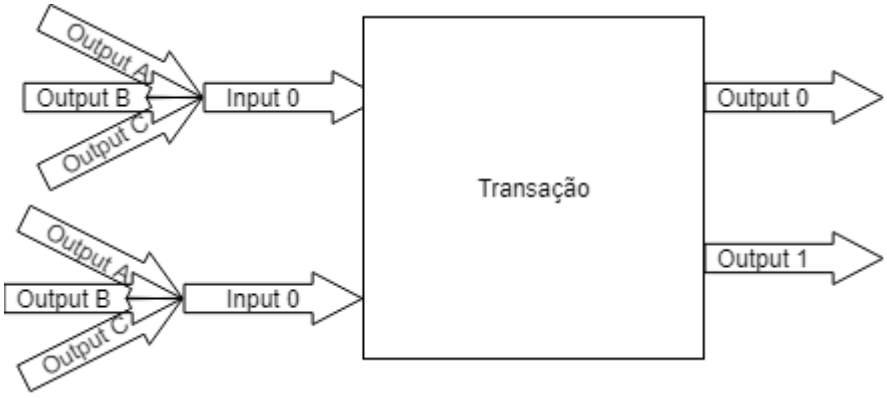

Figura 2.5 -Ring Signature

Devido ao miner não saber de quem é o output, é preciso de arranjar uma forma para o miner conseguir verificar que o output só é gasto uma única vez, para tal é utilizado uma *key image*.

A *key image* é derivada do *output* original e é calculado a partir de um algoritmo de curvas elípticas, fazendo com que não seja possível descobrir o *output* sabendo a *key image* e que o *output* só é gasto uma única vez.

Antes da implementação da *Ring Confidential Transaction* (RingCT), nas transações da *Monero* era necessário que o valor fosse dividido em vários anéis, por exemplo, se uma pessoa quisesse enviar 12.5 XMR seria necessário criar três anéis um com os respetivos valores: 10, 2 e 0.5. Esta limitação permitia aos utilizadores ver as quantidades que estão a ser transferidas.

Com a implementação da RingCT [19] torna-se possível esconder os valores que estão a ser transferidos nas transações sem impedir que seja possível fazer a verificação de se a transação é válida.

Estas transações são semelhantes às transações da *Bitcoin*, onde são constituídas por *inputs* e *output*s, mas no caso da *Bitcoin,* para uma transação ser válida, a soma de *inputs* tem de ser igual à soma de *output*s mais a taxa de transação. No caso da *Monero* é utilizado um algoritmo criptográfico, Pedersen Commitment [19], que permite que as somas sejam feitas sobre valores encriptados sem revelar o valor individual de cada *input* ou *output*. Como existem vários *inputs* e não se sabe qual é o *input* certo, a RingCT apenas necessita de que uma das combinações dos *inputs* seja igual à soma dos *output*s. Esta solução veio trazer outro problema, devido a permitir que possa haver valores negativos e como quem verifica não sabe se este valor é positivo ou negativo, isto permitiria, por exemplo, criar uma transação onde o *input* é 10 XMR e os *output*s seriam 20 XMR e -10 XMR, isto faz com que transação fosse válida porque a soma dos seus *input*s seria igual ao dos *output*s. Para prevenir este problema, implementou-se a *range proof* que permite verificar se o valor encriptado é um valor positivo. No entanto, após algum tempo substitui-se este método pelo *bulletproof,* o qual diminuiu substancialmente o tamanho das transferências, cerca de 80%, e o tempo de verificação, mas aumentou o tempo de criação da transação.

Stealth Addresses

Stealth Addresses são utilizados para esconder a identidade dos recetores, como referido anteriormente.

Sempre que existe uma transação, os remetentes são obrigados a construir um *stealth address*, isto é, um endereço que só pode ser utilizado uma única vez. Este endereço é construído a partir da *public view key* do destinatário e de um número gerado aleatoriamente [17]. Este número aleatório será utilizado também para calcular o identificador da transação.

Para estes cálculos será utilizado o algoritmo de curvas elípticas.

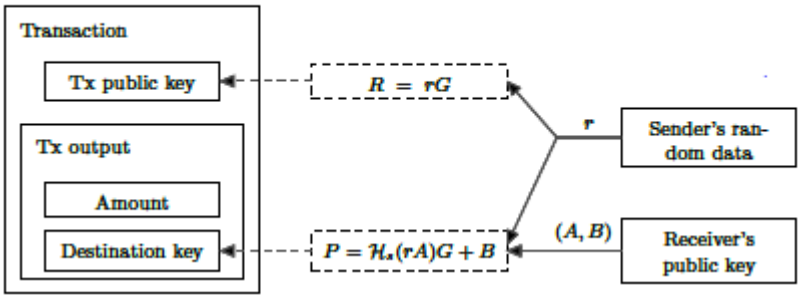

Figura 2.6 -Cálculo do Stealth Address obtido em [16]

De seguida o destinatário utilizando a sua *private view key*, em todas as transações aplicando um cálculo que se o resultado coincidir com o identificador da transação é porque a transação é para si (Figura 2.7).

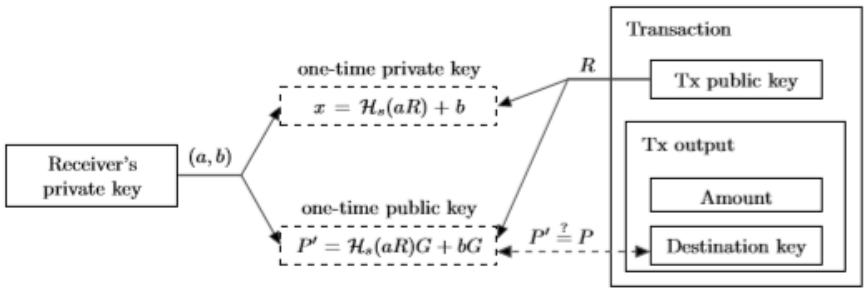

Figura 2.7 -Verificação de transações obtido em [16]

Podemos concluir que esta *blockchain* apresenta diversas propriedades para proteger a privacidade do utilizador, sendo que em cada transação não se consegue descobrir

nenhuma informação sobre o remetente e o destinatário. Também é de notar que não é possível seguir uma transação até a origem da moeda. Apesar desta *blockchain* produzir blocos de 2 em 2 minutos, esta produção é pouco escalável, visto que se tem de esperar em média 1 minuto para ver que a minha transação foi confirmada.

2.1.3 EOS.IO

A EOS.IO é uma *blockchain* bastante recente, lançada no dia 1 de junho de 2018. A EOS.IO opera como uma plataforma que executa *smart contracts* e como um sistema operativo remotamente distribuído. Os *smart contracts* definem uma interface (ações, estruturas de dados, parâmetros) e o código que implementa esta interface. A EOS.IO foi construída com intuito de ser utilizado para a implementação de aplicações de escala industrial e descentralizadas sobre esta. Este permite aos programadores de software criar aplicações onde os utilizadores da aplicação podem utilizar a sua aplicação gratuitamente. Esta apresenta uma solução para o problema de escalabilidade da *Bitcoin*, onde os blocos são criados de meio em meio segundo e ainda permite execuções em paralelo.

Nesta secção irá ser descrito alguns processos da EOS.IO a partir do único documento disponível em [11] pelos principais contribuidores deste sistema.

A EOS.IO apresenta um modelo de consensos diferente da *blockchain* e da ethereum, o modelo utilizado é o *Delegate Proof Of Stake* (DPOS) enquanto as *blockchains* anteriormente referidas utilizam o *proof-of-work*. Este algoritmo é dividido em duas partes: a eleição dos produtores dos blocos (*block producers)* e o agendamento da produção (quando é que o produtor de blocos deve produzir). Apenas os utilizadores que têm dinheiro investido têm o poder de votar para escolher quem serão os produtores os blocos. Quanto mais dinheiro investido maior valor terá o seu voto, isto faz com que a pessoa vote num produtor de blocos legitimo. Ter mais dinheiro investido, é consequentemente, quem tem mais a perder. Isto é, se a *blockchain* não está a ter um funcionamento correto, a moeda perde valor e consequentemente o investidor irá perder o seu dinheiro.

A EOS.IO tem 21 produtores de blocos, onde a cada meio segundo um produtor será escolhido para produzir um bloco, para de seguida o bloco ser incluído na *blockchain*, pelo menos 15 produtores de blocos têm de validar este bloco.

Se um produtor falhar a produção de um bloco e não produzir num período de 24 horas este será removido, até notificar a *blockchain* que pretende voltar a produzir blocos. Visto que só existe 21 produtores e os produtores são pagos em EOS, estes vão fazer de tudo para produzirem os blocos corretamente, para não perderem o seu lugar como produtor de blocos. Daniel Larimer, é o principal contribuidor para a construção da EOS.IO publicou um documento online em [20] sobre a guarda dos dados da *blockchain*, onde afirma que os produtores são responsáveis por guardar e replicar ficheiros de grande dimensão. Isto leva a que os produtores blocos sejam escolhidos com base em estarem dispersos e da sua capacidade de armazenamento. Este sistema de guarda de ficheiros é público e os utilizadores que queiram que o acesso seja privado têm de encriptar os seus ficheiros antes de serem enviados.

Além disso, os produtores de blocos têm a capacidade de congelar contas, isto é, se o produtor verificar que o seu *smart contract* tem um mau funcionamento, como consumir demasiados recursos, o produtor pode congelar a conta, deixando de executar as ações desse *smart contract.* As contas só poderão ser congeladas se 15 dos 21 produtores de blocos concordarem com essa ação.

A EOS.IO foi construída com o fundamento de ter aplicações descentralizadas com os seguintes requisitos:

- Estar disponível para milhares de pessoas
- Gratuito para os utilizadores
- Responsivo
- Ser possível atualizar estas aplicações
- Utilizar nomes em vez de chaves criptográficas

O facto de EOS.IO usar DPOS e ser uma *blockchain* cobre a maior parte dos pontos dito a cima.

Na EOS.IO existem três tipos de recursos consumidos pelas aplicações que correm nesta *blockchain*:

- *Bandwidth* e o armazenamento do registro (ROM)
- Utilização do CPU
- E armazenamento do estado (RAM)

A *blockchain* mantém um registro de todas as ações efetuadas a uma aplicação. Isto permite reconstruir o estado de todas as aplicações. O débito computacional é a quantidade de cálculos que se tem de fazer para recuperar o estado a partir do registro. Se este debito crescer demasiado, é preciso de criar *snapshots* do estado das *blockchain* e eliminar todo o histórico anterior. Se este crescer demasiado rápido, então pode ser preciso 6 meses para executar 1 ano de registros.

O armazenamento do estado é informação acedida a partir da lógica da aplicação. Esta informação deve ser apenas a informação do estado da aplicação, isto é, informação que é manipulada pela aplicação.

Os produtores de blocos publicam quanto *bandwith*, computação e estado têm. Os utilizadores podem utilizar estes recursos investindo os seus *tokens*, onde se investir 1% dos *tokens* existentes, o investidor obtém 1 % do armazenamento do estado, da *bandwith*, e da computação disponível pelos produtores de blocos.

Isto permite a quase criar uma aplicação gratuitamente, onde compramos e investimos *tokens*, para obter os recursos disponíveis. Depois quando já não quisermos a nossa aplicação a correr sobre esta *blockchain* podemos retirar os *tokens* investidos e vendê-los de volta.

A EOS.IO permite esconder as chaves criptográficas do utilizador, onde todos os utilizadores são identificados por um nome escolhido pelo criador da conta. Este nome pode ter até 12 caracteres. Para criar uma conta é necessário reservar RAM. Este processo é feito pelos criadores da aplicação, fazendo com que os utilizadores não tenham de pagar para usar a aplicação.

A EOS.IO implementa um processo de *governance* que direciona eficientemente a influencia existente dos produtores de blocos. O sistema reconhece que quem tem o poder são os utilizadores que tem *tokens* investidos, que podem delegar poder aos produtores de blocos. Com isto, é atribuído aos produtores um poder limitado que lhes permite congelar contas, atualizar aplicações e propor alterações ao protocolo existente.

Antes que qualquer mudança no *blockchain* ocorra, os produtores têm de aprovar essa alteração, mas se os produtores não aceitarem as mudanças desejadas, os utilizadores que têm *tokens* investidos podem mudar o seu voto, para que o produtor deixe de produzir blocos.

Podemos concluir que esta *blockchain* tem propriedades bastante diferentes comparado com as outras *blockchain*s apresentadas. Algumas destas propriedades são: (1) permite todo o tipo de transações e não apenas transações financeiras. (2) recuperação de conta. (3) sistema de permissões. (4) permite produção e verificação de blocos de uma forma bastante eficiente comparado com as outras *blockchain* apresentadas.

Esta *blockchain* foi construída com o intuito de facilitar aos programadores o desenvolvimento de aplicações descentralizadas com baixo custo e que consiga efetuar milhares de transações. Para tal, apenas é necessário que os programadores invistam os seus *tokens* e quando não precisar mais deste serviço é possível vendê-los de volta. Permitindo quase a hospedagem gratuita das suas aplicações. A hospedagem é quase gratuita devido ao preço destas moedas criptográficas serem instáveis, podendo descer ou aumentar de um dia para o outro.

2.2 Algoritmos Criptográficos

Nos dias de hoje existe uma grande necessidade de transmitir e guardar informação de forma secreta a partir da *internet*, como por exemplo, mensagens trocadas, transmitir informações de cartão de crédito, as nossas *password*s, entre outras. Este problema é resolvido a partir de criptografia [21]. O objetivo principal desta secção será mostrar

alguns esquemas de encriptação que podem ser relevantes para alcançar os objetivos desta dissertação.

2.2.1 Funções *hash*

O propósito das funções *hash* consiste em produzir uma "impressão digital" de um ficheiro, mensagem ou um bloco de dados. Uma função H deve ter as seguintes propriedades [22]:

- Fácil de computar
- Pode ser aplicado a um bloco de dados com um tamanho arbitrário
- Deve produzir um *output* de tamanho fixo
- Dado um output h é computacionalmente infazível encontrar x, sendo que H(x)=h
- Para qualquer bloco de dados x, é computacionalmente infazível encontrar um y≠x com H(x) = H(y)
- É computacionalmente infazível encontrar qualquer par (x,y), onde H(x)=H(y)

Existem várias funções de *hash*, como o SHA-1, o SHA-256, o SHA-384, o SHA-512, estes produzem outputs com diferentes tamanhos, 160, 256, 384, 512, respetivamente.

Estas funções são utilizadas frequentemente para guardar passwords de utilizadores, onde não deve ser possível ninguém saber esta password a não ser pelo próprio utilizador. Com as funções de *hash* podemos produzir o *hash* de uma password para ser guardado numa base de dados, isto permite a que, quem obter os *hashs* das passwords não consegue reverter o processo e obter a password.

Com as propriedades apresentadas anteriormente, também é possível garantir autenticação das mensagens, com a junção de cifras assimétricas, e a integridade das mensagens. No caso de alterar algum elemento na mensagem, consequentemente o *hash* resultante irá ser alterado, assim o utilizador pode verificar a integridade das mensagens comparado com o *hash* inicialmente produzido.

2.2.2 Cifra Simétrica

O paradigma da cifra simétrica consiste em permitir a cifragem e decifragem de conteúdos recorrendo a apenas uma chave secreta tanto para cifrar como para decifrar.

Assim, pode ser partilhada uma chave secreta entre várias entidades, e estes podem trocar mensagens entre eles, permitindo apenas às entidades que tenham a chave secreta decifrar o conteúdo das mensagens. É importante notar que quem descobrir a chave secreta consegue decifrar também o conteúdo destas mensagens.

Existe dois tipos de cifras simétricas: (1) cifra em *stream* ou (2) cifra em blocos.

A cifra em *stream* é utilizado quando queremos enviar conteúdo de forma contínua e não de uma vez só, por exemplo, pode ser utilizado para enviar vídeos, onde o vídeo é descarregado à medida que o utilizador observa o vídeo. Cifra em *stream* foi derivado de cifra em blocos, onde o conteúdo em vez de ser cifrado em blocos de tamanho fixo será cifrado de bit em bit. Ou seja, este tipo de cifra consiste em gerar bits de forma pseudoaleatória e combinar com o texto dado. Os bits gerados de forma pseudoaleatória são derivados da chave secreta.

Normalmente quando queremos enviar o conteúdo de uma vez só, utiliza-se cifra em blocos. Este tipo de cifra separa a mensagem em vários blocos de tamanho fixo, se o último bloco não for preenchido será necessário preenchê-lo.

Para a cifra em blocos existe vários modos de cifra, que são bastante importantes para a segurança das mensagens, como para o desempenho que queremos que as nossas aplicações tenham. Os modos mais utilizados são: ECB, CBC, CTR. Vou recorrer ao artigo [23], para descrever os modos referidos anteriormente.

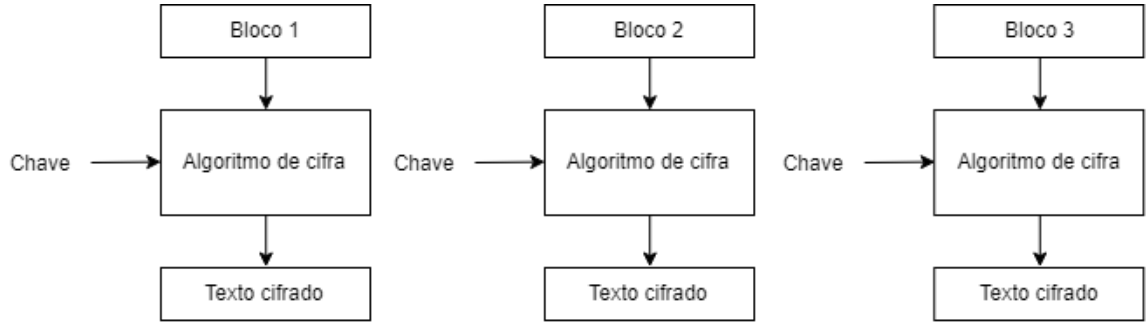

Figura 2.8- Processo ECB

O modo ECB é o modo que nos dá menos segurança, visto que dado um bloco, o resultado vai ser sempre o mesmo. E este modo é suscetível a ataques de dicionário, onde o atacante vai guardando a mensagem e o respetivo resultado da mensagem cifrada.

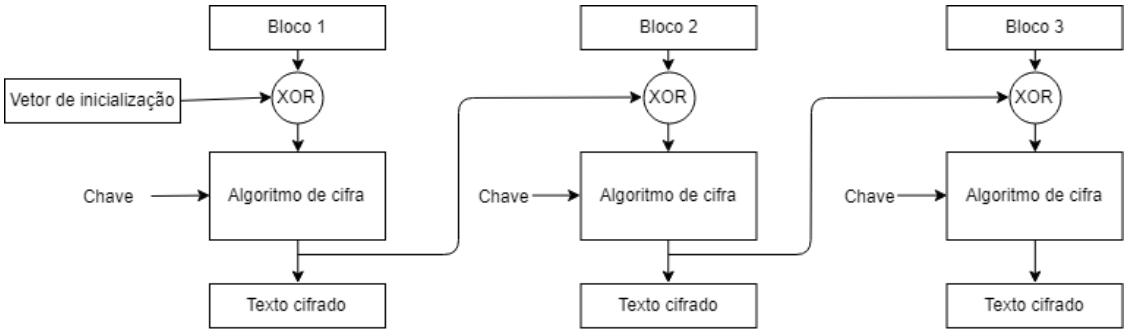

Figura 2.9- Processo CBC

No modo CBC, a cada bloco é efetuado um XOR com o resultado do bloco previamente cifrado. Terá de ser feito um XOR ao primeiro bloco com o vetor de inicialização, visto que o primeiro bloco não tem bloco anterior. Este vetor é simplesmente um vetor de bytes gerados aleatoriamente.

Todo este processo faz com que o resultado de um bloco depende do bloco anterior, apesar de isto aumentar a segurança, diminui bastante o desempenho do algoritmo.

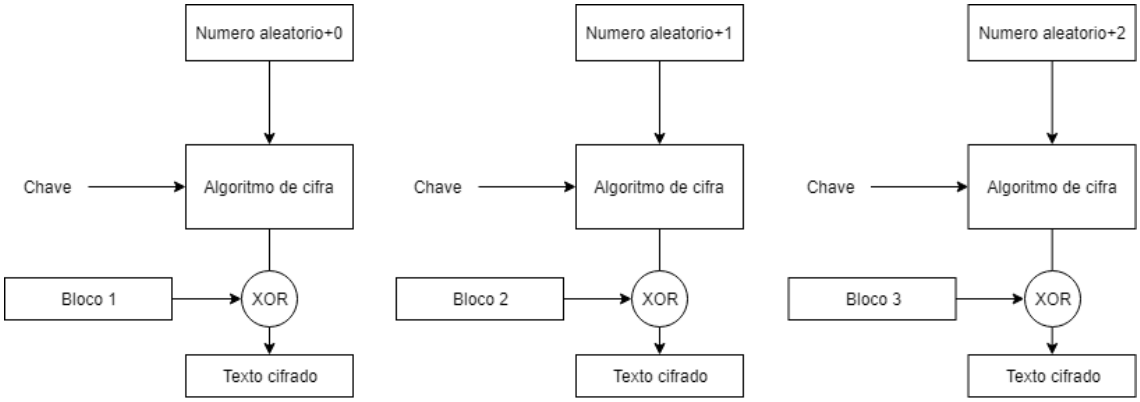

Figura 2.10- Processo CTR

Por último, CTR utiliza um número gerado aleatoriamente, onde para cada bloco é efetuado a cifra da soma do número aleatório com a posição do bloco, para posteriormente ser efetuado um XOR do bloco com o resultado da cifra. Isto faz com que este processo possa ser paralelizado, visto que para processar um bloco não depende do anterior.

Esta técnica trás a inconveniência que, por exemplo, se A e B quiserem comunicar de forma secreta com criptografia simétrica:

- A ou B selecionam uma chave fisicamente ao respetivo destinatário
- Uma entidade terceira entrega a mesma chave fisicamente a ambos

Isto faz com que esta técnica seja pouco prática para iniciar uma comunicação.

Existem vários algoritmos de cifra como o AES, DES, BLOWFISH, entre outros. Cada algoritmo utiliza chaves que têm um tamanho estipulado e dependendo disso os blocos irão ter um determinado tamanho. No caso do DES este só pode ser utilizado com chaves de 56 bits e os blocos têm de ter 64 bits. Devido ao tamanho da chave deste algoritmo, é possível achar a chave com ataques de força bruta nos tempos de hoje em dia, tornando este algoritmo bastante vulnerável. Isto leva ao problema que se aumentar demasiado a capacidade de computação, os algoritmos irão ficar obsoletos devido a terem um tamanho de chave específico.

Com isto dito é necessário verificar quais são os algoritmos que são seguros atualmente. Como já referido numa secção anterior, o que é seguro no dia de hoje pode já não ser seguro no dia de amanhã.

2.2.3 Cifra Assimétrica

Para descrição desta técnica será utilizado, o livro Cryptography and Network Security: Principles and Practice de William Stallings [22], iremos recorrer várias vezes a este livro para descrever a criptografia assimétrica nesta secção.

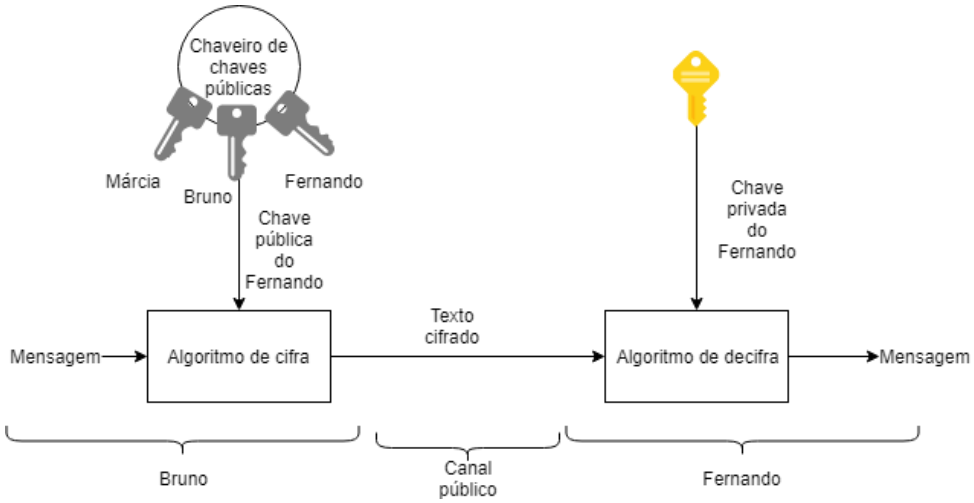

Figura 2.11- Cifra com a chave pública

A criptografia assimétrica consiste em utilizar diferentes chaves para o processo de cifra e de decifra. Uma chave denomina-se por *public key* (chave pública) e outra por *private key* (chave privada) que só deve ser conhecida pelo proprietário. Onde a chave pública, como o nome diz, pode ser partilhada com qualquer pessoa. Cada chave privada tem uma chave pública correspondente. E ao saber a chave pública não é possível obter a chave privada. Ou seja, se cifrarmos o conteúdo com a chave pública, só podemos decifrar o conteúdo com a chave privada, e vice-versa. Esta técnica de cifra pode ser utilizada para autenticação como para trocar uma mensagem secreta.

Para trocar uma mensagem de forma secreta, basta cifrar o conteúdo com a chave pública, assim apenas o utilizador que tiver a chave privada pode obter a mensagem. Esta propriedade é bastante útil, visto que podemos enviar uma mensagem para um utilizador sem ter que combinar previamente uma chave em segredo. Mas levanta o problema de não conseguirmos saber quem é que nos enviou a mensagem.

Uma forma ingénua de ser efetuada autenticação, seria enviar no corpo da mensagem em forma de texto o remetente da mensagem, mas mesmo assim, este texto não nos dá uma prova que foi essa pessoa a enviar a mensagem, pode ter sido alguém a fazer-se passar por outra entidade. Para resolver este problema, insere-se uma assinatura digital no corpo da mensagem, esta assinatura é constituída por um *hash* do conteúdo e depois é cifrado com a chave privada do remetente. Para verificar a autenticidade do conteúdo, o destinatário terá de decifrar assinatura com chave pública do remetente e executar a mesma função de *hash* no conteúdo, para depois comparar o resultado do *hash* produzido com o *hash* contido na assinatura. Visto que deciframos o conteúdo com a chave publica do remetente, o remetente será o único capaz de produzir aquele *hash*, tendo em conta que é a única pessoa que tem acesso à sua chave privada.

É de notar que se nós cifrarmos o conteúdo com a nossa chave privada, consequentemente, qualquer entidade que tenha a chave pública conseguirá decifrar o conteúdo. E se alguém cifrar o conteúdo com a nossa chave pública, apenas nós que possuímos a chave privada conseguimos decifrar este conteúdo.

Existe vários algoritmos que utiliza estas técnicas de encriptação simétrica, como o RSA, o Diffie-Hellman, o DSS, o Elliptic curve, entre outros. Cada algoritmo tem propriedades específicas que pode ser utilizado para diferentes aplicações, descritos na Tabela 2.1.

Tabela 2.1-Algoritmos de cifra assimétrica

Algoritmo	Cifrar/Decifrar	Assinatura digital	Troca de chave
RSA	Sim	Sim	Sim
Diffie-Hellman	Não	Não	Sim
DSS	Não	Sim	Não
Elliptic curve	Sim	Sim	Sim

Estes algoritmos são lentos devido a necessitarem de uma grande capacidade de computação. O algoritmo *Elliptic Curve Encryption*(ECC) é bastante mais eficiente que os outros algoritmos apresentados. Este algoritmo, utiliza chaves bastante mais pequenas que o RSA e oferece o mesmo nível de segurança. Para obtermos o mesmo nível de segurança, o ECC apenas precisa de uma chave de tamanho de 160 bits, onde RSA necessita de uma chave de 1024 bits. Isto faz com que seja bastante mais eficiente, visto que o algoritmo vai executar cálculos matemáticos sobre valores de dimensão bastante mais pequena [24].

Ou seja, este algoritmo permite-nos oferecer todas as funcionalidades das cifras assimétricas e com um desempenho bastante melhor que os outros algoritmos de cifra assimétrica. Mas é de notar que com algoritmos de cifra simétrica continuamos a obter um desempenho um pouco melhor e maior segurança.

Esta técnica apesar de resolver alguns dos problemas da cifra simétrica, como a partilha de chaves ou o tamanho dinâmico das chaves, estes têm o problema de que para ter a mesma segurança que algoritmos de cifra simétrica necessitam ter chaves bastantes maiores(excluindo o algoritmo ECC), e isto leva a que tenham de efetuar exponenciação de números de grandes dimensões, tornando estes algoritmos bastante mais lentos que os algoritmos de cifra simétrica [25].

Por estes motivos, a cifra assimétrica é utilizada essencialmente para combinar entre entidades uma chave para o algoritmo de cifras simétrico e para garantir autenticidades das mensagens com assinaturas digitais.

2.2.4 Cifragem com Preservação de Ordem

Com a necessidade de guardar os dados sensíveis em sistemas desconhecidos de uma forma segura levou igualmente à necessidade de fazer perguntas sobre estes dados cifrados, sem revelar os dados aos administradores destes sistemas. Isto levou à criação de esquemas de encriptação que utilizam OPE (*Order Preserving Encryption*).

Para um algoritmo ser considerado OPE necessita de satisfazer as seguintes condições:

- O algoritmo de geração de chaves tem de gerar uma chave aleatória k
- O algoritmo de cifra *Enc* usa uma chave k e o texto p para produzir um output $c = Enc_k(p)$
- O algoritmo de decifra *Dec* usa uma chave k e o texto cifrado c para produzir um output p
- A relação de ordem dos textos (cifrado e decifrado) é preservado. Por exemplo, $p1 \leq p2$ então $Enc(p1) \leq Enc(p2)$

Nas bases de dados, o OPE dá nos um grande poder em fazer *queries*, permitindo ao sistema de base de dados fazer *queries* sem necessitar decifrar os dados, e além disso permite fazer comparações utilizando o texto cifrado da mesma forma que comparamos o texto não cifrado. Existe alguns esquemas propostos como o de Boldyreva et al. em [26]. Este esquema revela metade dos bits mais significativos, o que faz com que o algoritmo não nos dê confidencialidade sobre os dados. Outra solução proposta pelos contribuidores da CryptDB, que nos dá mais segurança sobre os dados, apesar de apresentar um processo lento. Este mostra ter propriedades fortes de segurança, onde apenas revela a ordem dos textos cifrados.

Este esquema de encriptação é suscetível a vários tipos de ataques como *frequency analysis*, *sorting attack*, *cumulative attack*, entre outros [27]. Devido a isto só se deve ser utilizado estes esquemas em dados pouco importantes.

2.3 Sistemas de base de dados

Nesta secção será feita uma descrição de duas bases de dados com aspetos diferentes. Primeiro será feito uma descrição sobre a CryptDB que é uma base de dados cifrada, onde o principal objetivo é garantir a privacidade dos dados. E por ultimo a BigchainDB que apresenta propriedades de uma *blockchain* e de um sistema de base de dados.

2.3.1 CryptDB

Será recorrido várias vezes ao artigo [28], escrito pelos principais contribuidores deste sistema, para fazer uma descrição geral do sistema CryptDB nesta secção.

A CryptDB é um sistema que fornece confidencialidade a aplicações que utilizam bases de dados SQL. Este sistema executa *queries* SQL sobre dados cifrados utilizando diferentes esquemas de encriptação. CryptDB também tem a funcionalidade de criar outras chaves a partir da *password* do utilizador, assim apenas o utilizador que tive a *password* pode aceder aos seus respetivos dados. Como resultado o administrador da base de dados não consegue decifrar os dados, mesmo que o servidor esteja comprometido.

A CryptDB foca-se em proteger os nossos dados contra dois tipos de ameaças. A primeira consiste no administrador da base de dados tentar decifrar os dados que são privados. A segunda ameaça consiste no adversário tentar ganhar controlo do servidor *proxy* e da base de dados, neste caso a CryptDB apenas protege os utilizadores que estão offline.

A CryptDB tem três ideias para resolver os desafios previamente referidos:

- A primeira é executar *queries* SQL sobre os dados cifrados;

- A segunda é ajustar o modelo de cifra dependendo das *queries* feitas à base de dados;

- A terceira é criar todas as chaves a partir da *password* do utilizador;

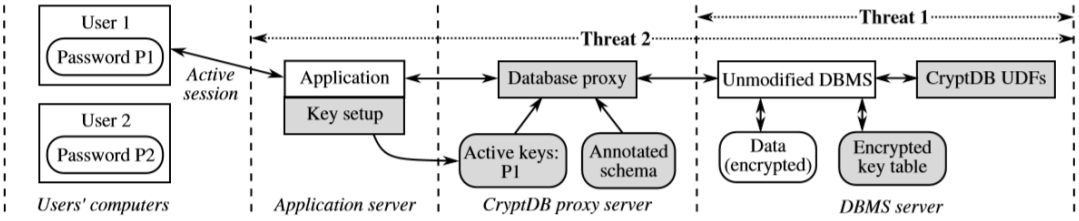

Figura 2.12 -Arquitetura da CryptDB obtido em [28]

A Figura 2.12 mostra a arquitetura e as duas ameaças consideradas. No sistema CryptDB todas as *queries* são executadas na *Database proxy,* que rescreve as *queries* para poderem ser executadas em dados cifrados. Basicamente o *proxy* tem o propósito de cifrar e decifrar todos os dados.

O sistema da base de dados nunca irá receber as chaves, assim não vai poder decifrar a informação e consequentemente o administrador do sistema não vai saber o valor decifrado.

Um adversário que compromete a base de dados pode eliminar qualquer informação da base de dados. Embora o CryptDB proteja a confidencialidade da informação, este não assegura a integridade dos resultados e nem que os resultados são os mais recentes.

Como dito anteriormente é possível fazer *queries* sobre dados cifrados sem decifrar os dados, permitindo o servidor executar certas funções sobre os dados cifrados. Por exemplo, quando executamos uma *query* onde existe um GROUP BY na coluna c, o servidor deve conseguir saber quais os dados são iguais na coluna, mas sem saber o conteúdo original. Para tal, temos de permitir que o *proxy* dê apenas esta informação ao

servidor, a CryptDB permite isto através da utilização de um esquema de encriptação dinâmico que se ajusta ao tipo de *queries* pretendidas. Contudo, se o servidor precisar de executar um GROUP BY na coluna c, o servidor não deve conseguir saber, por exemplo, a ordem dos dados da coluna c, nem deve conseguir correlacionar nenhuma outra informação das outras colunas com a coluna c.

A CryptDB oferece apenas confidencialidade aos dados e aos nomes das tabelas e colunas. Esta não esconde a estrutura das tabelas, número de linhas, o tipo de colunas, nem o tamanho aproximado dos dados.

Resumidamente, a CryptDB oferece as seguintes propriedades:

- Os dados nunca são revelados ao servidor

- A informação dada ao servidor sobre os dados depende das *queries* efetuadas:
    - Se uma coluna nunca for filtrada por um predicado, o servidor nunca saberá nada para além do seu valor cifrado.
    - Se a aplicação executar uma *query* de igualdade a uma coluna, o servidor apenas irá mostrar as repetições desta coluna
    - Se a aplicação executar uma *query* de ordem a uma coluna, o servidor apenas irá mostrar a ordem dessa mesma coluna

- O servidor não consegue obter nenhuma informação aos dados se a aplicação nunca executar uma *query* sobre estes dados.

O *proxy* da CryptDB guarda uma *master key* para poder cifrar os nomes das colunas e das tabelas. Cada coluna está encriptada com diferentes camadas de segurança.

Para processar uma *query* na CryptDB envolve 4 passos:

1. A aplicação envia uma *query* como se os dados não estivessem cifrados para o *proxy*, de seguida o *proxy* cifra cada variável da *query* e utiliza a sua *master key* para cifrar os nomes da tabelas e colunas.
2. O *proxy* verifica se a base de dados tem de ajustar a camada de segurança para poder fazer a *query*.
3. O *proxy* envia a *query* com os valores encriptados para o servidor e esta será executada como se fosse uma *query* normal
4. O servidor envia o resultado (encriptado) da *query*.

Com estes passos conseguimos permitir que a segurança das colunas é feita de uma forma dinâmica, onde o tipo de segurança vai variar com o tipo de *queries* feitas a uma coluna. Como referido anteriormente, se for efetuada uma *query*, que apenas necessita unicamente de saber igualdades entre os valores, o sistema não irá revelar nenhuma informação adicional a não ser esta.

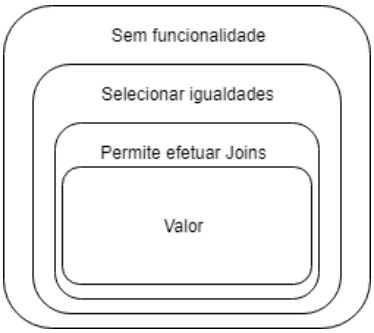

Figura 2.13- Onion encryption de uma coluna

Para tal, a CryptDB utiliza um esquema de encriptação, denominado por *onion encryption layers*. Todas as colunas serão cifradas com a chave $PRP_{MK}(table\ t, column\ c, onion\ o, layer\ l)$, onde PRP é uma permutação pseudoaleatória (Por exemplo, AES). Onde na camada superior da cebola iremos obter maior segurança, mas não iremos conseguir executar qualquer tipo de *queries* sobre estes dados. No caso das camadas inferiores vai ser possível executar *queries*, mas relembrando que,

ao fazermos *queries*, onde a base de dados necessita de saber informação adicional, estamos também a revelar esta informação ao administrador do sistema. Esta informação adicional pode ser igualdades ou ordem dos dados dos dados cifrados.

No caso da camada superior pode se usar por exemplo AES em modo CBC, ou seja, se utilizarmos o mesmo *input*, mas com diferentes vetores de inicialização e a mesma chave, consequentemente, os *output*s serão diferentes, não permitindo obter qualquer informação sobre este *input*.

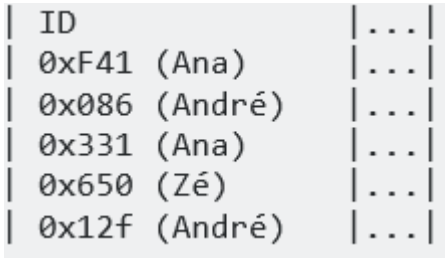

Figura 2.14- Cifra AES em modo CBC com diferentes vetores de inicialização e a mesma chave

No caso de uma camada inferior pode se utilizar na mesma o algoritmo AES para a cifra dos valores, mas sempre com o mesmo vetor de inicialização e a mesma chave, assim permitindo ao servidor fazer *queries* onde envolve igualdades de valores na mesma coluna.

Figura 2.15- Cifra AES em modo CBC com o mesmo vetor de inicialização e a mesma chave

Inicialmente os valores vão estar encriptados com várias camadas. Conforme o *proxy* vai recebendo *queries* da aplicação, o *proxy* irá dar as chaves ao sistema da base de

dados para remover as camadas necessárias, para ser posteriormente efetuada a respetiva *query*.

A CryptDB ao fazer *joins* entre tabelas, apenas irá informar à base de dados quais são os valores iguais entre as duas colunas. Para tal, irá ser utilizado um *token* para cada valor, este *token* é um *hash* do valor e é ajustável em tempo de execução. Estes *token*s inicialmente tem um valor diferente para o mesmo valor em diferentes colunas, que pode ser ajustado para mostrar as igualdades entre as colunas. Ou seja, ao fazer um *join* da coluna A com coluna B, o *proxy* vai informar a base de dados de como ajustar os *tokens* da coluna B para que se existir um valor da coluna B igual na coluna A, estes tenham um *token* igual.

Para efetuar *range queries* (…WHERE age>10), CryptDB utiliza uma *avl tree* no servidor, onde os dados cifrados estão ordenados, sem o servidor saber qual é o seu verdadeiro valor. Para a inserção de um valor nesta arvore, são necessários os seguintes passos: (1) inicialmente o servidor irá enviar o valor cifrado na raiz da arvore ao cliente. (2) O cliente irá decifrar este valor, dizendo ao servidor se este valor é maior ou menor que o valor que pretende introduzir. (3) Caso seja menor, o servidor irá devolver o nó à esquerda do valor anteriormente devolvido pelo servidor, caso contrário o nó direito. (4) O cliente irá continuar este processo de comparar e decifrar valores até o servidor devolver um nó vazio. (5) O cliente envia o nó que pretende inserir para o servidor inserir no nó vazio.

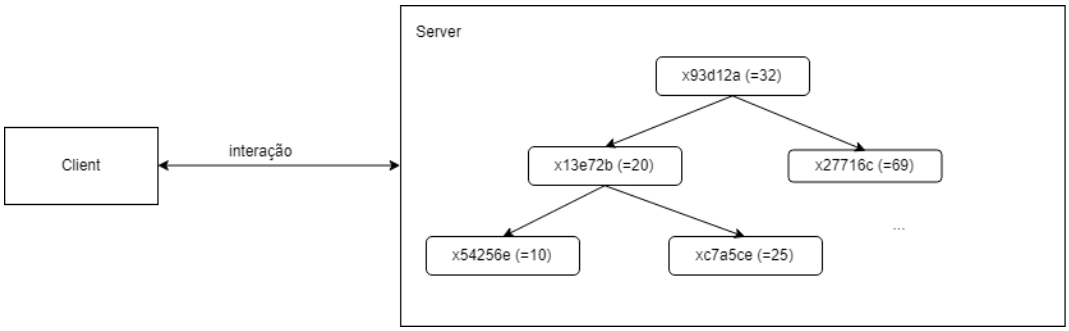

Figura 2.16 -CryptDB OPE tree

Utilizando a arvore da Figura 2.16, se o utilizador quiser inserir o valor 80:

1. O utilizador pede ao servidor a *root* da arvore

2. O servidor responde com o valor cifrado de 32
3. O utilizador decifra o valor e compara 80 com 32, visto que 80 é maior, o utilizador pede ao servidor para enviar o nó à direita de 32.
4. O utilizador vai comparar 69 com 80, o utilizador pede o nó da direita
5. O servidor devolve um nó vazio
6. O utilizador envia o nó para ser inserido nesse nó vazio

O *encoding* de um valor que é dado pela posição na arvore, por exemplo, se o seu *encoding* for três, isto significa que é o terceiro maior valor na árvore. Para facilitar o servidor irá ter uma tabela que guarda os valores cifrados e o respetivo *encoding*, assim caso o valor cifrado já esteja nesta tabela, o servidor consegue devolver rapidamente o respetivo *encoding*. E para efetuar as comparações basta comparar o *enconding* dos valores cifrados.

A CryptDB apresenta propriedades bastante importantes para a guarda de dados em sistemas considerados inseguros, onde permite-nos fazer *queries* sobre dados encriptados sem revelar nenhuma informação para além da ordem ou igualdades dos valores.

2.3.2 BigchainDB

A descrição desta da BigchainDB é feita com base no *white paper* [29], escrita pelos os contribuidores da mesma.

A BigchainDB utiliza o TinderMint que trata da parte do modelo de consensos e da rede *peer-to-peer*. O TinderMint utiliza um modelo de consensos *proof-of-stake*, o que permite ter um desempenho bastante elevado.

A BigchainDB é uma *database blockchain*, esta tem propriedades de uma *blockchain*, como por exemplo, é descentralizada, imutável, entre outras. Na BigchainDB utiliza como base de dados o MongoDB.

A arquitetura dos nós que garantem a persistência é constituída por duas bases de dados, uma que vai conter toda a informação da base de dados de cliente e outra que

contém as transações dos clientes. O numero de nós é determinado pelo cliente, quanto mais nós o cliente escolher maior será a segurança dos seus dados.

A rede BigchainDB pode ser publica, privada. Onde numa rede publica o cliente pode acede-la e criar a sua própria base de dados na rede. Enquanto na rede privada necessita de permissões para tal.

Para efetuar transações, o utilizador apenas precisa de enviar uma transação para a rede e o sistema da BigchainDB irá garantir que apenas será executada uma vez. Outra propriedade é que todas as transações estão assinadas pelo cliente, garantindo assim a autenticação das transações.

Apesar de a BigchainDB ter estas propriedades, isto não cumpre com os objetivos da aplicação que queremos ter. O principal problema é a confidencialidade dos dados. Os dados da BigchainDB não estão encriptados. Assim este sistema não consegue garantir a privacidade dos dados do utilizador. A única forma de conseguir a confidencialidade é criar uma rede privada.

2.4 Análise do Estado de Arte

Têm vindo a ser desenvolvido várias *blockchains*, umas com mais funcionalidades, outras com maior desempenho, e outras com escalabilidade flexível, mas nenhuma com o intuito de guardar dados genéricos de forma segura.

Inicialmente as *blockchains* mais conhecidas, como a *bitcoin* e a *monero*, apenas permitem efetuar transações financeiras, com o aparecimento de *blockchains,* como a *EOS.IO* e a *ethereum*, com a funcionalidade de construir *smart contracts* complexas, ou seja, podemos definir ações que podem ser executadas pelos os utilizadores da blockchain, isto dá-nos o potencial de construir aplicações sobre uma *blockchain*.

No sistema *blockchain* todos os dados são públicos, ou seja, qualquer entidade pode aceder aos dados dos blocos, isto leva-nos ao problema de ser necessário salva guardar os dados de forma segura, onde não revele os dados a pessoas que não tem permissão para tal. Existe algumas implementações de sistemas de base dados, que nos dão estas

propriedades, onde quem tem as bases de dados é considerado uma ameaça. A principal implementação é a CryptDB, outras implementações foram descontinuadas ou utilizam a CryptDB como base. A CryptDB tem propriedades bastante importantes, como por exemplo, ser possível fazer *queries* sobre os dados sem revelar informação dos dados a quem executa as *queries*. Este sistema utiliza cifra com a preservação de ordem, apesar deste utilizar um esquema lento, providencia uma segurança elevada, onde revela apenas a ordem dos textos cifrados. Existe outros algoritmos de cifra com a preservação de ordem, mas apresentam um baixo nível de segurança, visto que revelam informação sobre o dado cifrado.

Com a análise dos vários *blockchains* conseguimos concluir que esta tecnologia está bastante evoluída, onde existe *blockchains* que nos permitem criar aplicações sobre *blockchains*. Contudo, ainda existe poucas bases de dados encriptadas onde o seu conteúdo seja pesquisável e se possa fazer *queries* complexas sem revelar informação ao sistema da base dados.

Como podemos ver não existe nenhum sistema parecido com o nosso sistema, onde temos diversas *blockchains* bastante desenvolvidas, mas que não tem um sistema de base de dados que nos permita guardar qualquer tipo de dados de forma segura e temos bases cifradas que tem alguns problemas de segurança. Isto leva-nos a criar o nosso próprio sistema, onde juntamos a tecnologia *blockchain* com um sistema de base de dados cifrada.

3 Solução proposta

Na secção Trabalho Relacionado foram descritos vários sistemas que utilizam a tecnologia *blockchain*, começando com uma descrição de como a *blockchain* da Bitcoin funciona. Tal como referido anteriormente, a *blockchain* da Bitcoin permite apenas guardar transações financeiras tendo também vários problemas de escalabilidade. Atualmente já existem outras *blockchains* que resolvem estas limitações permitindo também ter *smart contracts* complexos que nos dão mais funcionalidades, como a EOS que nos permite desenvolver aplicações descentralizadas que correm simplesmente sobre a *blockchain*.

Nesta secção é apresentada uma breve descrição do projeto onde a solução proposta se enquadra, e de como serão utilizados *smart contracts* para a implementação do mesmo. Como referido anteriormente, o foco principal desta dissertação encontra-se na confidencialidade dos dados, onde apenas vamos fazer desenvolvimentos sobre as bases de dados das *sidechains*, mas para tal é necessário efetuar uma descrição geral do projeto para posteriormente explicar como podemos garantir a privacidade dos dados e o motivo de termos de cifrar os dados. Relembrando que esta dissertação apenas está relacionada com a confidencialidade e a guarda dos dados.

De seguida é feita uma proposta de como garantir a confidencialidade dos dados das bases de dados, e da forma de fazer pesquisas sobre os dados cifrados.

Por último, é apresentado um esquema de como podemos obter um sistema de permissões sobre dados cifrados, onde é necessário adicionar/remover permissões sem ser necessário revelar chaves de cifra/decifra.

3.1 Estrutura Geral

O uso de smart contracts permite construir sistemas assentes sobre *blockchain*. No projeto onde esta tese se enquadra, estes são utilizados para construir *sidechains*. Uma *sidechain* é uma *blockchain* separada que está ligada a uma *blockchain* principal. O facto destas *sidechains* serem manipuladas por *smart contracts*, permite que as mesmas utilizem funcionalidades da *blockchain* principal, como a sua cryptomoeda, mas numa *blockchain* separada. Estas *sidechains* podem ter modelos de consenso diferentes da *blockchain* principal, tal como ter os próprios produtores de blocos, entre outras possíveis alterações.

Sidechains também podem ser descartáveis, quando a informação contida numa *sidechain* já não for precisa. Esta *sidechain* poderá ser apagada pelos produtores de blocos, e estes continuarão a ter as suas cryptomoedas visto que estas moedas estão na *blockchain* principal.

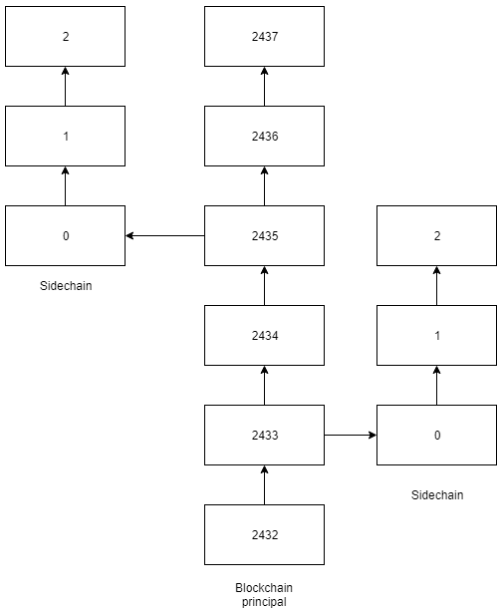

Figura 3.1- Representação gráfica da blockchain e das suas sidechains

Estas propriedades permitem-nos desenvolver um sistema onde cada produtor de blocos poderá participar na produção de várias sidechains, sendo que cada uma irá conter dados referentes às bases de dados que o cliente requisitante da *sidechain* pretende guardar.

Cada *sidechain* tem o seu *smart contract* na *blockchain* principal. Este tem todas as configurações da *sidechain*, como por exemplo, quantas cryptomoedas é que o produtor de blocos vai ter que investir, quantos produtores de blocos necessita aquela *sidechain*, entre outras. Quando o *smart contract* estiver devidamente configurado abrem-se as candidaturas, uma vez que no *smart contract* estão guardados todos os candidatos para posteriormente ser feita a seleção dos produtores de blocos.

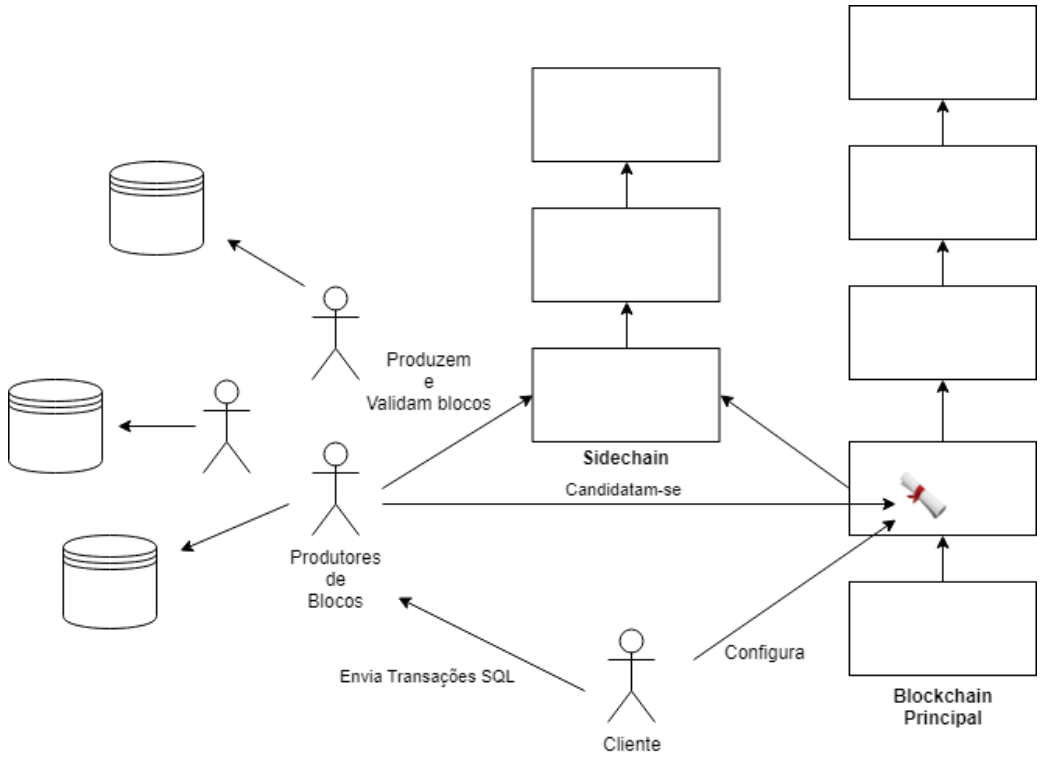

Figura 3.2- Vista geral do sistema

A Figura 3.2 apresenta uma vista geral do sistema, no qual temos um cliente que configura o seu respetivo *smart contract* e pode efetuar operações SQL. As operações SQL do cliente serão enviadas para um dos produtores de blocos para serem inseridas na *blockchain*.

No caso do produtor de blocos este pode candidatar-se aos vários *smart contracts*, e caso seja escolhido pelo *smart contract*, poderá produzir blocos e construir as respetivas bases de dados. Estes decidem participar na produção de uma sidechain, mediante um incentivo económico fornecido pelo cliente. Quanto maior e mais estável for esse incentivo, maior a probabilidade dos produtores se manterem a produzir a sidechain. Caso haja falhas no cumprimento desse incentivo, os produtores poder-se-ão sentir tentados a sair da produção da sidechain. Este incentivo também proporciona uma maior segurança à *sidechain*, visto que as vagas para produzir blocos são poucas, o produtor vai querer fazer de tudo para continuar a receber este incentivo e cumprir com as regras estipuladas pelo cliente.

Se o cliente quiser ter uma maior segurança:

- Terá de escolher vários produtores de blocos. Quantos mais produtores de blocos tiver maior será a sua segurança;

- De forma a cativar os produtores de blocos vai pagar mais por bloco;

Um dos objetivos desta dissertação é a recriação das diversas bases de dados a partir das transações contidas na *sidechain*. Para tal temos estruturas, onde são guardadas as diferentes operações SQL. Como por exemplo, um *CREATE TABLE* necessita de ter o nome da tabela, uma lista de colunas, onde cada coluna tem um nome e outras características.

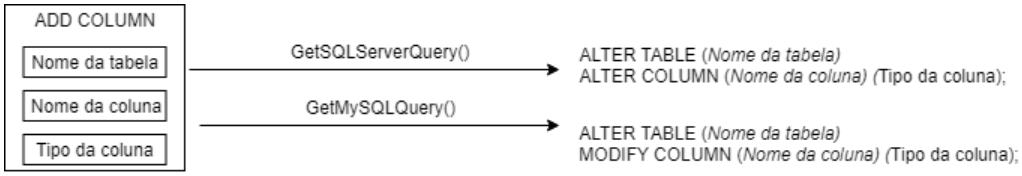

Figura 3.3- Exemplo da estrutura da operação ADD COLUMN

Utilizando estas estruturas para definir as diferentes transações permite-nos ter diferentes tipos de bases de dados, como por exemplo, SQL server, mongoDB, MySQL, entre outras.

Um sistema com estas propriedades permite que quando um cliente não precise mais das suas bases de dados, a *sidechain* possa ser apagada. Os produtores vão continuar com os seus ganhos, visto que o dinheiro será gerido na *blockchain* principal.

3.2 Criptografia e segurança

3.2.1 Abordagem utilizada

Como referido na secção 1.1, o roubo e fuga de informação acontece constantemente nos dias de hoje. Visto que estamos a utilizar a tecnologia *blockchain*, um sistema público para guardar toda a informação do utilizador necessita de garantir a privacidade dos seus clientes, mantendo a confidencialidade dos seus dados.

O foco principal desta dissertação assenta na confidencialidade dos dados, devido a grande parte da integridade destes ser resolvida pela utilização da tecnologia *blockchain*.

A principal ameaça do sistema são os próprios produtores de blocos do sistema, dado estes possuírem a base de dados do cliente e poderem tentar obter e analisar os dados privados. Por este motivo, é necessário cifrar os dados de forma a proteger os dados privados do cliente, onde apenas o cliente tem acesso à chave de cifra/decifra e

consequentemente será a única pessoa que vai poder cifrar/decifrar os dados. Porém, se cifrarmos o conteúdo com um algoritmo de encriptação simétrica, como por exemplo o AES, o conteúdo resultante não será pesquisável com *queries*, dado não ter nenhuma semelhança para com os dados originais. Uma possível solução prática seria dar a respetiva chave de encriptação aos produtores de blocos para decifrar os dados, mas assim o produtor de blocos iria obter acesso a toda a informação na base de dados.

Logo, o principal objetivo é cifrar o conteúdo de tal forma que seja possível executar *queries* sobre os dados, mas minimizando a informação revelada ao produtor de blocos. Por exemplo, se para executar uma *query*, o produtor de blocos apenas necessita de saber as igualdades, então apenas devemos revelar essa informação aos produtores de blocos. Ultimamente um dos problemas encontrados consiste no facto de que sabendo igualdades é possível revelar os dados a partir de ataques de análise de frequência. Um ataque de análise de frequência envolve comparar uma base de dados não cifrada de grande dimensão com uma base de dados cifrada. Onde utilizamos o numero de vezes que determinado valor está repetido na base de dados cifrada e comparamos com a frequência dos valores da base de dados não cifrada. Recentemente um ataque desta natureza aos dados de um hospital conseguiu revelar cerca de 95% da informação. Consequentemente, tivemos de utilizar outra estratégia, assim a solução pensada para a resolução deste problema consiste em solicitar informação extra ao administrador da base de dados, de forma a que o administrador não saiba o que está a ser pedido.

Para garantir os nossos objetivos, é necessário que toda a informação guardada no produtor esteja cifrada, incluindo nomes de tabelas e colunas, impedindo assim a revelação de qualquer informação sobre a estrutura da base de dados ao atacante.

Para tal, toda a estrutura da base de dados está cifrada com o algoritmo AES, devido a este algoritmo conseguir ter o melhor *tradeoff* entre segurança e desempenho. Como referido na secção 2.2.3, os algoritmos de cifra assimétrica apesar de apresentarem mais funcionalidades que os algoritmos simétricos, sendo estas dispensáveis para a resolução deste problema, são mais lentos que os algoritmos de cifra simétrica.

As chaves que utilizámos para o algoritmo AES têm um tamanho de 256 bits. Estas chaves vão ser usadas em várias cifras durante um longo tempo, por isso é necessário utilizar chaves de grande dimensão para garantir a segurança das mesmas. Outro motivo é pela NIST (National Institute of Standards and Technology) recomendar que os algoritmos simétricos utilizem chaves maiores ou iguais a 256 bits. Outro ponto importante no algoritmo de cifra simétrica que usámos é o seu modo de operação. No sistema implementado usamos o modo CBC, apesar de ser o modo mais lento, é o que garante uma maior segurança. Estes dados, apesar de estarem cifrados, também estão expostos ao público e podem ter um grande valor para o cliente, logo é necessário garantir o máximo de segurança sobre os dados da base de dados.

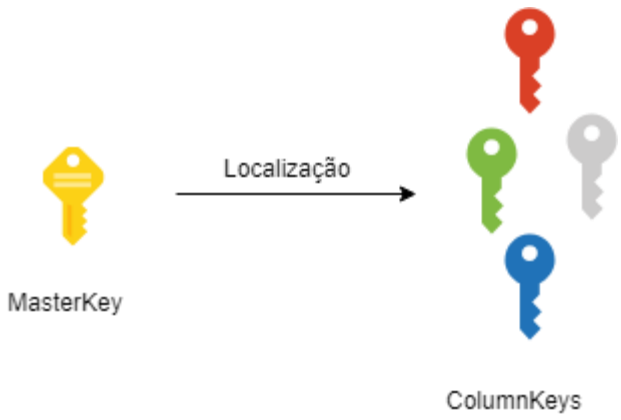

Figura 3.4- Geração de chaves

Se o sistema cifrar todos os valores com uma única chave, esta chave estaria muito exposta a ataques, e uma vez que o atacante obtivesse a chave iria ter acesso completo à base de dados. Por este motivo cada coluna vai ter a sua própria chave. Contudo, numa base de dados de grande dimensão seria necessário guardar uma quantidade grande de chaves. Para tal, a gestão de chaves é feita a partir de uma única chave, denominada por *master key*, e pela sua localização na base de dados. Por exemplo, se quisermos cifrar um valor de uma coluna utilizaremos o nome da tabela e o nome da coluna.

Todas as chaves de colunas são calculadas em tempo de execução a partir da seguinte formula:

$$AES_m(\text{sha256(Nome da tabela, Nome da coluna)}),$$

onde m é a *master key*.

Assim, se cifrarmos o mesmo valor em colunas diferentes obteremos resultados diferentes, mas isto ainda não é o pretendido. O que pretendemos é cifrar o mesmo valor numa coluna e obtermos resultados diferentes. Para tal vamos utilizar os seguintes esquemas de encriptação:

- Random (RND) – RND é um esquema de cifra probabilístico, significa que dois valores iguais tem uma probabilidade muito grande de serem mapeados para textos cifrados destintos. Apesar de ser muito seguro devido a não mostrar igualdades, não nos permite fazer nenhuma pesquisa sobre os dados. A implementação utilizada para este esquema consiste em utilizar o algoritmo AES com um vetor de inicialização gerado aleatoriamente.

- Deterministic (DET) – DET é um esquema de cifra determinístico, no qual o texto cifrado de dois valores iguais vai ser sempre igual. Este algoritmo permite verificar a igualdade entre valores. A implementação utilizada para este esquema é utilizar o algoritmo AES sempre com o mesmo vetor de inicialização.

Para utilizar o esquema RND temos de guardar os vetores de inicialização utilizados na cifra dos valores da base de dados, para posteriormente podermos decifrá-los. Existem dois locais onde se podem guardar os vetores de inicialização, o local mais seguro seria o cliente guardar estes valores, mas para tal o cliente teria de guardar para cada valor um vetor de inicialização, o que não é muito vantajoso visto que o cliente em

vez de guardar o vetor de inicialização poderia guardar o próprio valor. Uma vez que um dos objetivos consiste em que a guarda de dados seja feita por terceiros, o local ideal é guardar diretamente na base de dados.

Para cifrarmos o nome da tabela e das colunas, recorremos à cifra destas e a uma chave, por sua vez, é calculada pela sua localização e da *master key*. A implementação utilizada para o cálculo das chaves de cifra dos nomes das tabelas é feita da seguinte forma:

$$AES_m \text{ (SHA256 (Nome da base de dados))},$$

onde *m* é a *master key*.

No caso dos nomes das colunas, a implementação para o cálculo da chave é feito da seguinte forma:

$$AES_m \text{ (SHA256 (Nome da base de dados, Nome da tabela))},$$

onde *m* é a *master key*.

Tabela 3.1- Exemplo da base de dados cifrado com o esquema de cifra RND

Ivjg432h	**jg432h**	**Iv43gfefg**	**43gfefg**
0xKJX	0xFKG	0x9FJ	0x6FD
0x87F	0x4JF	0x8JFD	0xDD0
0x32D	0xS3Q	0x37C	0x21P

Desta forma conseguimos garantir, que não é possível ver dados que são iguais e que o cliente não necessite de guardar um número exorbitante de vetores de inicialização e chaves.

Tendo os dados cifrados desta forma, conseguimos garantir a segurança dos valores, mas podemos verificar que não nos permite fazer *queries* sobre os dados. A única forma prática de obter um resultado é ir pedindo valores ao produtor de blocos e decifrá-los até obtermos o resultado pretendido. O que faz com que o sistema seja bastante lento numa base de dados de grande dimensão.

Para resolver este problema tivemos de criar uma abordagem nova, designada por *bucket*. Um *bucket* é apenas um número que pode corresponder a vários valores destintos. A ideia é associar um número a diferentes valores, e diferentes valores poderão ter o mesmo número. É necessário que a função que gere o *bucket* para um valor apenas possa ser calculada pelo utilizador. Desta forma, o atacante não conseguirá saber quais são os valores que estão em determinado *bucket*. Outro requisito é que haja colisões, isto é, para um *bucket* é necessário que haja vários valores destintos, caso contrário o atacante conseguirá detetar igualdades.

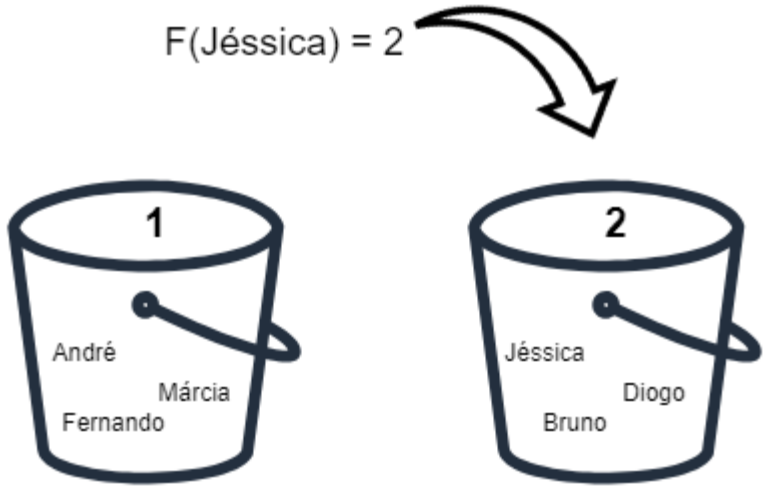

Figura 3.5- Representação gráfica de buckets para Equality Queries

Como podemos verificar na Figura 3.5, utilizámos a função F para calcular em qual *bucket* devemos inserir o valor. Mas para tal é necessário que quem esteja a desenhar a base de dados defina quantos *buckets* são necessários. O número de *buckets* recomendado para uma determinada coluna é metade do número de possíveis valores distintos que pode assumir. Assim se forem inseridas todas as possibilidades de valores, obrigatoriamente vai haver valores que vão ficar no mesmo *bucket*. No melhor dos casos vão ficar distribuídos de forma equilibrada, e cada *bucket* conterá apenas dois valores. Esta propriedade permite-nos fazer uma pesquisa sem mostrar igualdade.

Tabela 3.2- Exemplo de uma tabela na base de dados com o respetivo bucket

Names	Bucket
0x3DS2 (André)	1
0xGF24 (Ana)	2
0x4KGF (Fernando)	1

Utilizando a Tabela 3.2, se o utilizador pretender pesquisar pelo valor "Fernando", ele vai calcular primeiro o *bucket* correspondente a "Fernando" que será 1 e irá obter todos os valores com o *bucket* a 1. Ou seja, o resultado obtido será André e Fernando, como podemos verificar, o utilizador terá de filtrar os valores que não correspondem à *query* executada, neste caso vai descartar "André".

Podemos observar que, quanto mais *buckets* o utilizador escolher, maior será a probabilidade de valores que sejam iguais ficarem no mesmo *bucket*. Em resultado disto o administrador da base de dados vai poder efetuar ataques à base de dados apenas sabendo estas igualdades, conseguindo revelar o seu verdadeiro valor.

Porem, o contrario também se verifica, quanto menor for o número de *buckets*, maior será a probabilidade dos valores ficarem todos no mesmo *bucket*, resultando em pesquisas mais lentas, pois implicaria uma filtragem exaustiva desses valores.

A função implementada para calcular o respetivo *bucket* de um valor y é:

$$\text{SHA256}(AES_c(\text{y})) \bmod \text{x},$$

Onde *c* é a chave da coluna e *x* é a quantidade de *buckets* de uma determinada coluna.

Desta forma conseguimos fazer o que foi referido anteriormente, sendo que este modelo oferece-nos uma pesquisa probabilística. Ou seja, ao procurar por um valor o resultado pode conter valores errados, estes valores servem de *decoy* para o atacante, isto permite dificultar os ataques de frequência que ocorrem muitas vezes em bases de dados cifradas. O utilizador é quem define para cada coluna qual é o nível de segurança que quer dar a uma determinada coluna, mas é importante também que o cliente tenha a noção que optando por um modelo com mais segurança implica uma redução do desempenho e vice-versa.

Outro ponto importante desta proposta de solução que nos reforça a segurança, é que cada coluna está cifrada com uma chave, isto faz com que, caso uma chave seja descoberta por um atacante, apenas essa coluna será descoberta.

Este modelo apenas permite fazer *Equality Queries*, ou seja, não é possível de uma forma prática pedir ao produtor de blocos para devolver, por exemplo, todas as pessoas que têm idade acima dos 80. A única forma de fazer este tipo de *queries* com este esquema seria pedir de forma exaustiva. Onde iria pedir primeiro pelas pessoas com idade igual a 80, depois com 81 e sucessivamente. Até o utilizador esgotar todas as possiblidades. Para poder executar este tipo de *queries*, denominados por *Range Queries*, temos de ter um modelo diferente, mas para tal temos de abdicar de alguma segurança. A ideia inicial é inserir os valores em *buckets*, onde se o valor x é maior que o valor y e estão em *buckets*

diferentes, logo todos os valores que estão no mesmo *bucket* que x, incluindo x, são maiores que os valores que estão contidos no mesmo *bucket* que y, incluindo y.

Figura 3.6-Representação gráfica dos buckets para Range Queries

Este processo é menos seguro que o anterior, revela a ordem e sabendo um *bucket* é possível, com um ataque simples, saber quais são os possíveis valores que aquele *bucket* pode tomar. Supondo que temos uma tabela de utilizadores, mesmo com o seu nome cifrado, o atacante por análise pode descobrir que se trata de uma tabela de utilizadores, onde vai utilizar a aplicação que utiliza a base de dados e vai criar vários utilizadores, verificando em quais tabelas houve modificações. Se for uma aplicação com pouco uso, a tabela que sofreu modificações vai ser a tabela no qual está contida a informação dos utilizadores. De seguida o atacante pode criar várias contas com diferentes idades de forma incremental, por exemplo de 0 a 10, e vai verificando na tabela se a conta que acabou de criar alterou o *bucket*. Digamos que quando o utilizador criou a conta com idade 5 o seu *bucket* foi alterado, o atacante vai saber que o *bucket* anterior contém os valores das idades 0 a 4. E que todos os valores no *bucket* onde o 5 foi inserido são maiores ou iguais que 5.

Neste modelo é necessário que quem desenhe a base de dados decida quantos valores diferentes poderá ter um *bucket* e será o número máximo que a coluna pode obter.

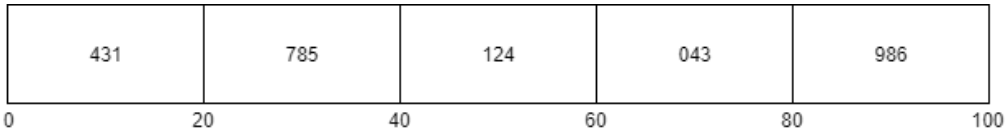

Figura 3.7- Segmentação do domínio uma coluna

Na Figura 3.7, quem construiu a base de dados decidiu que cada *bucket* pode ter 20 valores diferentes e que o valor máximo para uma coluna é 100. Assim todos os valores de 0-20 vão para o *bucket* 431, os valores de 20-40 vão para o *bucket* 785. A forma de calcular este *bucket* é a partir do limite, ou seja, se o valor estiver entre o valor 0-20 então utilizaremos o 0, caso seja 20-40 utilizaremos o 20, utilizando o mesmo processo para os outros domínios.

Onde o *bucket* é calculado da seguinte forma:

$$\mathrm{SHA256}(AES_c(\mathrm{y})),$$

Onde *c* é a chave da coluna e y é o limite inferior do *bucket*.

Se utilizarmos a Figura 3.7 e o cliente fizer uma *query*, como, por exemplo, obter todos os valores maiores que 65, o cliente vai começar por calcular os *buckets*, neste caso vai utilizar os limites 60 e 80, e obteremos 043 e 986 respetivamente. De seguida vai pedir à base de dados todos os valores que tenham o *bucket* 043 e 986 e como podemos verificar, este processo é semelhante ao anterior, daí podemos receber resultados *decoy*, como por exemplo valores entre 60-64, o que faz com que seja uma procura probabilística.

Constatamos que, dependendo do modelo utilizado, existe um *trade-off* de segurança. Somente na primeira abordagem, que apenas nos permite efetuar *equality queries* se verifica um maior nível de segurança que as *range queries.* Por isso é necessário analisar se é viável ter essa funcionalidade de efetuar range *queries* sobre uma coluna. Por estes

motivos, iremos dar a opção ao utilizador de decidir se quer que a coluna seja *NormalColumn* que permite apenas efetuar *equality* queries ou uma *RangeColumn* que permite fazer *equality queries* e *range queries*.

Ambas as colunas são cifradas com o esquema RND, devido aos motivos mencionados anteriormente, e terão uma coluna adjacente que indicará em que *bucket* vai estar inserido.

Existe também a necessidade de haver *UniqueColumns*, visto que, se tivermos uma coluna em que todos os seus valores sejam únicos, podemos cifrar estes valores com o esquema de encriptação DET. Assim conseguimos executar *queries* sem a necessidade de utilizar *buckets*, consequentemente vamos aumentar o desempenho das pesquisas na base de dados. Mas visto que utilizamos o esquema DET, e o vetor de inicialização é sempre o mesmo, não existe necessidade de estar sempre a guardar o vetor de inicialização como fazemos com o esquema RND. Os vetores de inicialização são calculados em tempo de execução, a partir de um único vetor inicialização (*masterIV*) e da sua localização, da seguinte forma:

AES_m (sha1(Nome da tabela, Nome da coluna, masterIv)),

onde *m* é a *master key*.

Assim podemos esconder o vetor de inicialização, onde apenas o cliente consegue calcular.

Tabela 3.3- Exemplo de uma tabela com cifra com esquema DET

Nomes
0xFKG (André)
0x4JF (Ana)
0xFKG (André)

Se, por exemplo, quisermos pesquisar na Tabela 3.3 por André, basta cifrar o valor André com a chave e o vetor de inicialização daquela coluna. O resultado da cifra será igual ao valor que está na tabela a representar o nome André.

Por último, com estes 3 tipos de colunas não é possível fazer *joins* entre tabelas, devido a cada coluna estar cifrada com diferentes chaves, e para tal é necessário que as colunas tenham valores em comum. Para resolver este problema as *primary keys* e *foreign keys* são obrigatoriamente GUID (*global unique identifier*), pelo que os seus valores vão estar cifrados com a chave da PrimaryColumn, utilizando o esquema DET visto que uma GUID é única e necessitamos de ver igualdades para poder efetuar um *join*.

Em suma, existem 5 tipos de colunas:

- PrimaryColumns – Os valores aceites nestas colunas são GUIDs e vão estar cifrados com a chave da respetiva coluna, utilizando o esquema DET;

- ForeignColumns – Os valores aceites nestas colunas são GUIDs. Estes têm uma referência para outra PrimaryColumn e vão estar cifrados com a mesma chave que a sua PrimaryColumn, utilizando o esquema DET;

- UniqueColumns – Colunas em que todos os seus valores são únicos e estão cifrados com o esquema DET;

- NormalColumns – Os valores destas colunas estão cifrados com o esquema RND e permitem efetuar apenas equality queries;

- RangeColumns – Estas são iguais às NormalColumns mas em vez de permitir apenas efetuar *equality queries*, permitem também efetuar *range queries*.

3.2.2 Outras abordagens

A primeira abordagem utilizada para se poder efetuar pesquisas sobre dados cifrados foi utilizar uma abordagem como a da CryptDB. A CryptDB, dependendo da *query* que se fizer a uma coluna vai retirando camadas de segurança, como explicado na secção 2.3.1. Ou seja, vamos ter os dados cifrados em três camadas, onde a primeira camada permite ver valores iguais apenas entre duas colunas, a segunda permite ver igualdades na mesma coluna, utilizando o esquema de encriptação DET e a última camada não tem qualquer funcionalidade e é considerada a mais segura devido a utilizar o esquema RND.

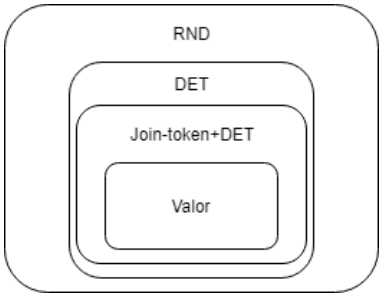

Figura 3.8- Onion Encryption

Inicialmente, com este esquema, todos os dados da base dados estão cifrados com todas as camadas, como na Figura 3.8. Se fosse necessário executar *queries* quando precisamos de igualdades, como por exemplo, pedir todos os utilizadores que tenham 20 anos, o cliente começa por dar a chave da primeira camada, neste caso a chave da camada RND e o produtor de blocos que contém a base de dados vai passar por todos os valores e decifrá-los para ficar na camada DET. Com a camada DET já é possível verificar igualdades e assim já poderemos executar a *query* que o utilizador pretende. O cliente apenas

necessita de reconstruir o *onion* até à camada DET, visto que se cifrarmos o valor 20 nesta camada o resultado será sempre igual.

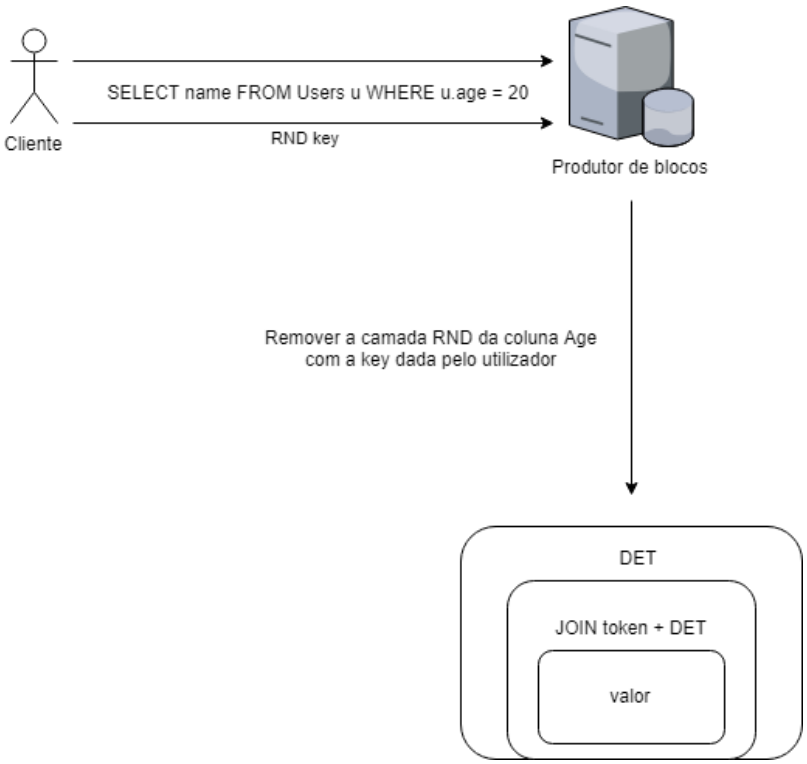

Figura 3.9-Processo de remoção de camadas

Apesar de o cliente dar a chave ao produtor de blocos, o valor vai continuar cifrado por outros esquemas de cifra.

Como poderemos verificar esta abordagem acarreta vários problemas. Um dos principais problemas é a segurança dos dados. Caso seja necessário efetuar *queries* onde necessitamos saber igualdades de uma determinada coluna, teremos de remover a camada segurança RND. Essa coluna vai expor as igualdades dos valores. Como referido anteriormente, um dos problemas de segurança nas bases de dados cifradas é que apenas mostrando igualdades é possível revelar os valores, problema é que se tivermos uma quantidade exorbitante de dados numa coluna e o cliente pretender executar *queries* sobre uma coluna, que tem todas as camadas de segurança, o produtor de blocos vai ter de

decifrar todos os valores desta coluna. Isto representa processo bastante lento, e um grande gasto computacional para o produtor de blocos. Ou seja, esta abordagem não é a ideal para resolver este problema.

Na abordagem utilizada na secção 3.2.1 só é possível fazer um *join* pelas colunas que são *primary key* e *foreign key*. Uma abordagem possível para o cliente poder fazer um *join* numa coluna qualquer, é utilizar o esquema que a CryptDB implementou.

Este esquema envolve duas funções:

- Função para calcular o *token*: dado um valor e uma chave devolve um *token*, este *token* tem as mesmas propriedades que um *hash*;

- Função de ajuste: é a função que ajusta um *token* para ficar com o valor que deveria ficar se utilizasse outra chave. Esta função recebe o *token*, e um valor para ajustar o *token*.

Ou seja, com este esquema, cada coluna vai ter uma chave diferente e cada valor vai ter um *token* correspondente. Este *token* vai ter as mesmas propriedades que um *hash* e se calcularmos o *token* de dois valores iguais utilizando a mesma chave iremos obter o mesmo resultado. Visto que cada coluna vai ter uma chave diferente, apenas vai ser possível verificar as igualdades na mesma coluna. Somente, com a função para calcular o *token*, não é possível efetuar um *join* em diferentes colunas, para tal é necessário mostrar igualdades entre duas colunas. Utilizando a função de ajuste permite-nos mostrar igualdades entre duas colunas, onde iremos ajustar os *tokens* de uma coluna para serem calculados da mesma forma que os *tokens* de outra coluna e permitindo ver quais são os valores das diferentes colunas que são iguais. Esta função de ajuste só é possível ser utilizada pelo produtor de blocos com a ajuda do cliente, permitindo ao o cliente dizer ao produtor de blocos como ajustar os *tokens*. Este facto é importante porque se o produtor de blocos pudesse ajustar os *tokens* de forma independente, faria com que ele pudesse ajustar todas as colunas de forma a conseguir ver todos os valores que fossem iguais na base de dados.

Para o cálculo destes *tokens* são necessárias duas chaves, uma denomina-se *basekey* e *columnkey*. A *columnkey* é construída com base na localização da coluna e a *basekey* é igual para todas as colunas da base de dados. Estes *tokens* são calculados com base nas fórmulas das curvas elípticas.

$$token = P\ .\ kc * AES_{kb}(SHA1(valor)),$$

onde *kc* é a *columnkey*, *kb* é a *basekey* e P é um ponto da curva alíptica utilizada.

Como cada coluna tem uma *columnkey* diferente o resultado é sempre diferente. Tendo na base de dados cada *token* para o respetivo valor, quando for necessário efetuar um *join*, o cliente vai ter de enviar a chave de ajuste ao produtor de blocos para poder ajustar. Para obtermos a chave de ajuste apenas é necessário calcular o seguinte:

$$\Delta K = Ka^{-1}\ .\ Kb,$$

onde *Ka* é a chave da coluna *a* e a *Kb* é a chave da coluna *b*.

Ou seja, se dermos este ΔK ao produtor de blocos é possível calcular os valores da mesma forma que os da coluna b, onde tendo:

$$\begin{aligned} token_{Ka}\ .\ \Delta K &= P\ .\ Ka * Ka^{-1} * Kb * AES_{kb}\big(SHA1(valor)\big) \\ &= P\ .\ Kb * AES_{kb}\big(SHA1(valor)\big) = token_{Kb} \end{aligned}$$

Como podemos verificar este esquema tem os mesmos problemas da abordagem referida anteriormente nesta secção, onde para efetuar um *join* é necessário ajustar o *token* de todos os valores de uma determinada coluna. Tal como no exemplo anterior, se tivermos uma quantidade exorbitante de valores teríamos de ajustar todos os valores dessa coluna sempre que se efetua um *join*.

3.3 Gestão de chaves

Como anteriormente referido, todas as chaves são calculadas em tempo de execução, mas uma funcionalidade que deve ser possível na base de dados é poder criar permissões para diferentes tipos de utilizadores. Contudo, existe um problema neste sistema, visto que todo o seu conteúdo está cifrado. Basicamente, se quisermos dar permissões a um determinado utilizador, esse utilizador tem que ter acesso às chaves de cifra/decifra para poder inserir e visualizar valores. Isto é um problema porque uma vez dado as chaves de cifra/decifra de uma coluna ao utilizador, esse utilizador vai ter sempre acesso aos valores, o que faz com que não seja possível remover essas permissões ao utilizador. Uma forma simples, mas inadequada para resolver este problema é voltar a cifrar o conteúdo com uma chave distinta, o que se tivermos uma quantidade elevada de registos faz com que este processo seja muito lento.

Podemos resolver este problema tendo um *proxy* que tivesse um sistema de contas com as respetivas permissões, onde o *proxy* é o único que contém a *master key* para poder gerar as restantes chaves. Inicialmente o utilizador, tem que iniciar sessão, enviando as suas credencias ao *proxy*. Se esta ação tiver sucesso, o utilizador pode enviar as suas *queries* ao *proxy* sem estarem cifradas. O *proxy* quando receber a *query* de um utilizador, vai certificar se que o utilizador tem permissões para executar a *query*.

Se tiver permissões, o *proxy* vai:

1) Extrair o conteúdo das *queries*;

2) Vai cifrar com as chaves respetivas;

3) Enviar a *query* ao produtor de blocos com o seu conteúdo cifrado. Quando receber o conteúdo devolvido pelo produtor de blocos, vai decifrá-lo e enviá-lo ao utilizador correspondente.

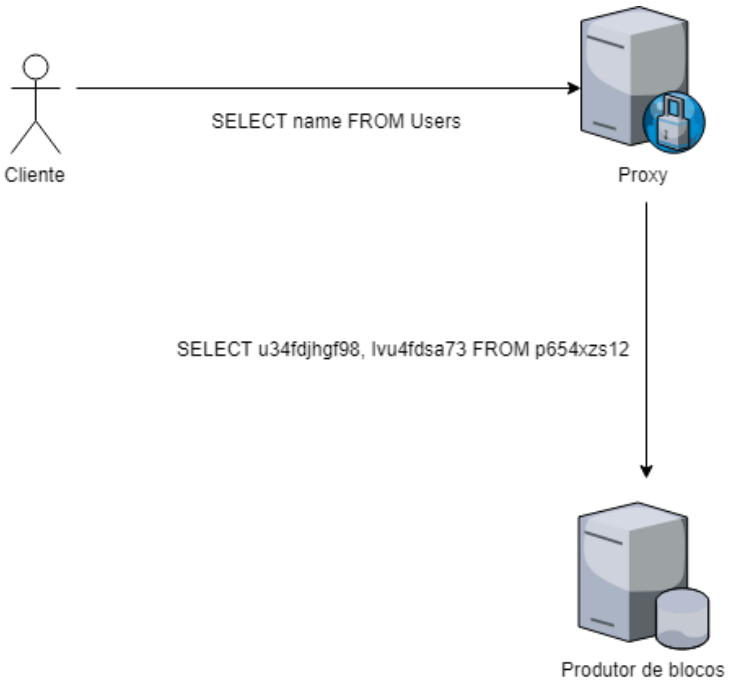

Figura 3.10- Sistema Utilizador-Proxy-Produtor de Blocos

Assim se quisermos tirar permissões de um utilizador basta remover essa permissão do *proxy*. Com esta solução poderemos identificar algumas desvantagens, como por exemplo:

- Cria um ambiente centralizado, onde em vez de o utilizador comunicar com os diferentes produtores de blocos, ele apenas vai comunicar com o seu *proxy*;

- Maior latência nas comunicações, visto que comunica primeiro com o proxy e de seguida o *proxy* vai comunicar com um dos produtores de blocos;

- O *proxy* vai ser sobrecarregado com o processo de cifra e decifra dos dados, mas também podemos ter a vantagem de o utilizador não necessitar

de ter uma grande capacidade computacional, visto que não tem de cifrar os valores.

Devido a esta estrutura apresentar algumas desvantagens, é necessário dar ao utilizador a possibilidade de escolher entre duas estruturas: 1) cliente-produtor de blocos; 2) cliente-*proxy*-produtor de blocos.

Nesta segunda abordagem, cliente-produtor de blocos, não permite ter permissões. Os utilizadores vão ter de guardar a *master key*. Nesta abordagem, o utilizador tem de ter um dispositivo com maior capacidade computacional, devido ao dispositivo ter de cifrar todos os dados antes de enviá-los. O que faz com que não seja uma solução viável para dispositivos com baixo poder computacional ou dispositivos com baterias. Mas por outro lado a capacidade computacional está distribuída pelos diferentes utilizadores e não apenas no *proxy* como na abordagem anterior.

Figura 3.11- Sistema Utilizador-Produtor de Blocos

Assim dependendo das necessidades do utilizador, este terá de escolher entre estas duas estruturas, tendo em conta as vantagens e desvantagens de cada uma.

4 Caso de estudo

Na secção anterior foram descritas as várias abordagens tomadas em consideração, onde durante o desenvolvimento de 3.2.2 Outras abordagens foram verificados vários problemas como não só de segurança, mas como desempenho.

Esta secção tem como objetivo demonstrar o que foi desenvolvido com base na solução proposta. O desenvolvimento prático foi bastante importante para o estudo de abordagens. O desenvolvimento permitiu verificar os aspetos negativos das primeiras abordagens e analisar como poderíamos manter os aspetos positivos e eliminar os aspetos negativos de forma a que consigamos ter o melhor *trade-off* entre segurança e desempenho.

Mais uma vez, reforço que apenas foram feitos desenvolvimentos sobre as bases de dados das *sidechains*. Onde o desenvolvimento apenas envolve a construção das bases de dados das sidechains de forma a manter a confidencialidade dos dados.

As tecnologias utilizadas para o desenvolvimento deste projeto foram:

- Linguagem de programação C# com .NET Core;
- Sistema de base dados: SQL Server, MongoDB, MySQL;
- Ambiente de desenvolvimento: Microsoft Visual Studio;

- Object-relational mapper (ORM): Entity Framework.

4.1 Arquitetura

O caso de estudo foi dividido em vários módulos, onde a ordem da implementação de cada módulo é importante para o desenvolvimento da solução proposta. Tendo uma arquitetura bem definida, permite-nos organizar melhor o projeto e consequentemente melhorar o desempenho no desenvolvimento do projeto.

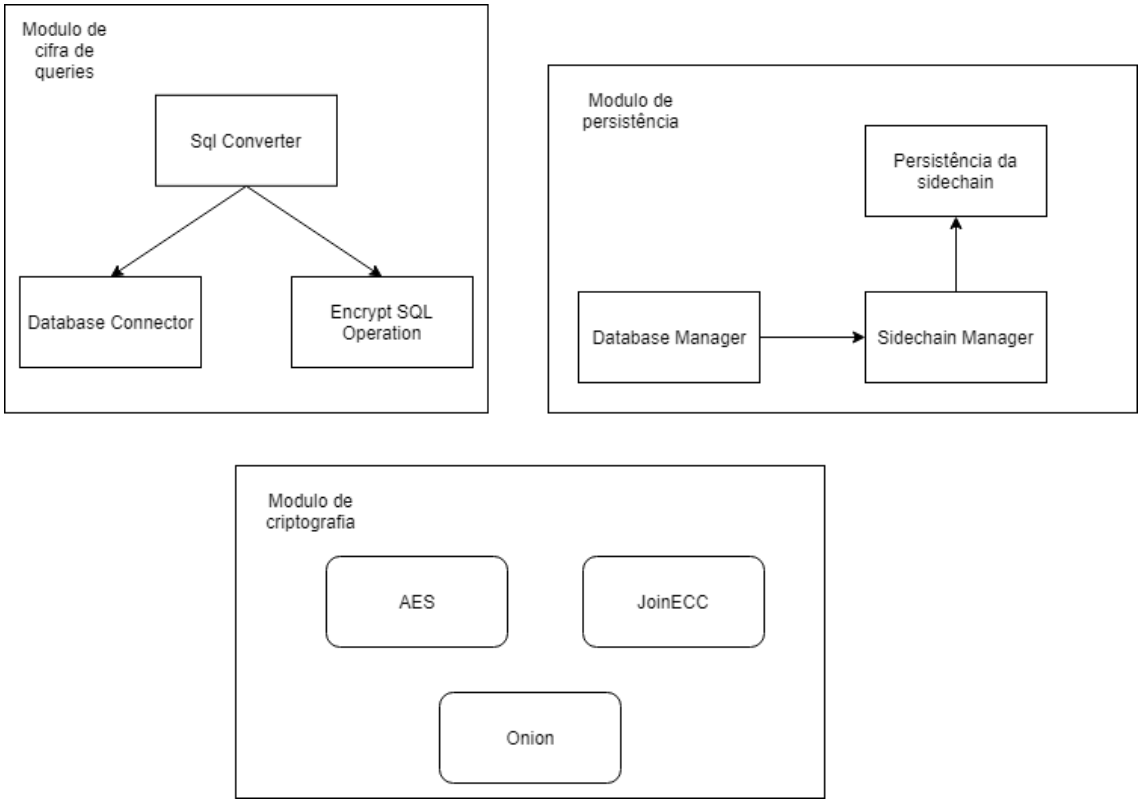

Figura 4.1- Arquitetura do Caso de estudo

Como se pode verificar na Figura 4.1 projeto foi dividido em 3 grandes módulos. Estes módulos são os seguintes:

- Modulo de persistência;
- Modulo de criptografia;
- Modulo de cifra de *queries*.

Com o modulo de persistência permite ao produtor de blocos conseguir ler as várias transações SQL contidas nas diferentes *sidechains* e reconstruir as respetivas bases de dados dos clientes. Com o modulo de criptografia e de cifra de queries, permite ao cliente enviar *queries* SQL aos produtores de blocos de forma a que os produtores de blocos não percebam os dados que estão a ser enviados e que consigam executar na base de dados.

Figura 4.2- Arquitetura do caso de estudo

Resumidamente, o sistema desenvolvido permite a um cliente construir uma *querie* SQL, onde todo o conteúdo da *querie* será cifrado, exceto as operações SQL da *querie* (por exemplo: SELECT, WHERE, etc). De seguida o produtor de blocos executa a *querie*, devolve o resultado ao cliente e o cliente conseguirá decifrar os valores recebidos.

4.2 Modulo de persistência

Este módulo foi o primeiro a ser desenvolvido, este módulo consiste na persistência do estado das diversas *sidechains* e está localizada nos produtores de blocos, onde

inicialmente o que se fez foi definir todas as transações possíveis a serem executadas na *blockchain*, neste caso as transações possíveis são operações SQL.

As operações implementadas foram:

- CreateColumn- Cria uma coluna numa tabela já existente;
- CreateTable- Cria uma nova tabela;
- DeleteColumn- Elimina uma coluna;
- DeleteRecord- Apaga um ou múltiplos valores nas colunas pretendidas;
- DeleteTable- Apaga a tabela;
- InsertRecord- Insere um ou vários valores nas colunas pretendidas;
- UpdateRecord- Atualiza um ou vários valores nas colunas pretendidas.

Foi implementado uma estrutura para cada operação e funções que devolvem uma *query* para diferentes tipos de bases de dados. Estas estruturas serão importantes para a cifra dos dados de cada operação, assim é possível obter os dados de cada *query* e proceder com a cifra dos dados, sem ter que dar parse a uma string, à procura dos campos que pretendemos cifrar. Outro facto é poder utilizar estas estruturas para poder construir a *query* para a base de dados que o produtor de blocos pretende utilizar.

O componente *SidechainManager* é responsável pela e criação e reconstrução da base de dados da sua *sidechain*. Este módulo também é responsável de criar todas as estruturas necessárias para ser possível efetuar *queries* sobre os dados cifrados.

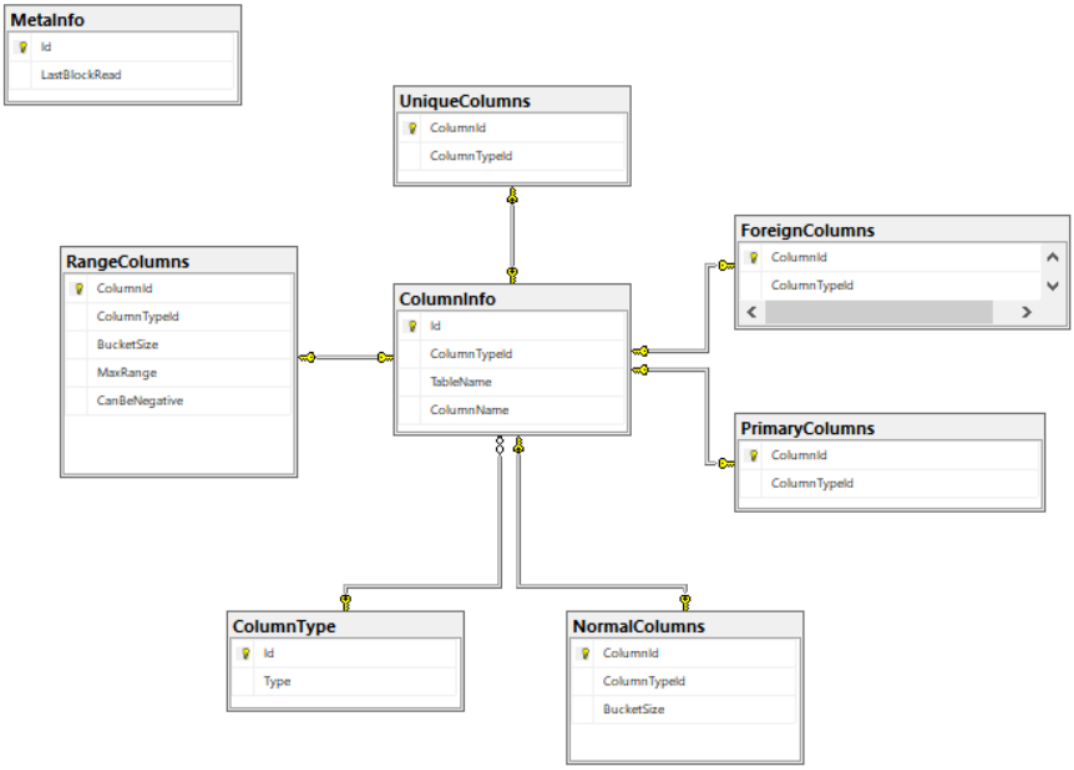

Figura 4.3- Infraestruturas da *sidechain*

Na criação de uma *sidechain* serão criadas as infraestruturas apresentadas na Figura 4.3. Como referido na secção Abordagem utilizada estas estruturas serão necessárias para ser possível efetuar *queries*.

O componente DatabaseManager é responsável por coordenar todos os SidechainsManagers. Onde o componente DatabaseManager vai ter a sua própria base de dados onde é guardada toda a informação de todas as *sidechains* e as suas respetivas transações em formato JSON e cada SidechainManager vai apenas executar operações na base de dados da sua *sidechain*.

Assim o DatabaseManager vai ler as diversas transações da base de dados que guarda todas as transações das *sidechains*, e irá converter o JSON para a respetiva estrutura e enviar ao SidechainManager para executar na sua base de dados.

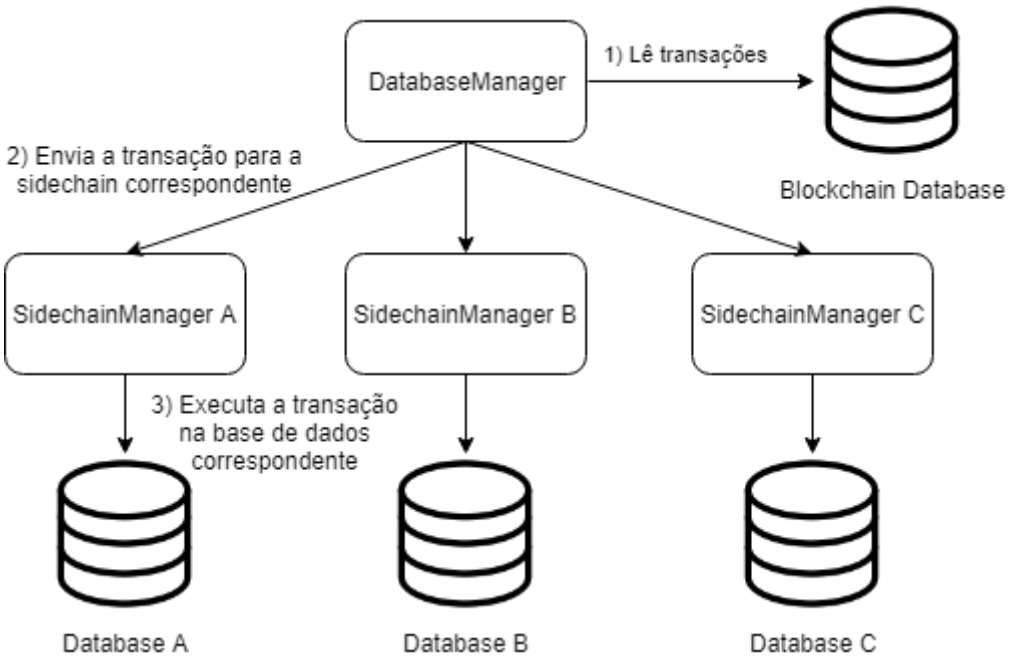

Figura 4.4- Estrutura do modulo da persistência

4.3 Modulo de criptografia

Este modulo é responsável por fornecer as diferentes ferramentas para a cifra dos dados.

Este módulo tem o componente AES que tem várias funções que recebem um *byte array* e cifram ou decifram o conteúdo com o algoritmo AES. Este apresenta também diferentes modos de cifra e requer que a dimensão das chaves utilizadas neste algoritmo seja de 256 bits.

Existe outro componente denominado por *Onion* que permite construir e retirar as diferentes camadas do *onion scheme* com as diferentes camadas referidas na secção 3.2.2

Por último, a componente JoinECC, apesar deste componente não ser utilizado para a implementação da abordagem utilizada, este foi implementado. Este componente é onde está toda a lógica das curvas elípticas, da geração dos *tokens* e do ajuste de *tokens*.

Apesar dos componentes *Onion* e JoinECC não terem sido utilizados, ambos foram implementados. Apenas foram feitos *unit tests*.

4.4 Modulo de cifra de queries

Este é o módulo de maior dimensão, onde é feita a cifra de uma *query* que o cliente pretenda efetuar e toda a comunicação entre cliente e produtor de blocos.

A componente Encrypt Sql Operation é o componente que contém uma função para todas as operações SQL possíveis, onde as operações recebem como parâmetro de entrada a operação e devolve a operação SQL cifrada. Estas funções podem devolver mais que uma operação, por exemplo, operações que vão guardar quantos *buckets* uma respetiva coluna pode ter (Figura 4.5).

Figura 4.5- Cifra de *queries*

Mas antes de poder cifrar as *queries* é necessário perguntar ao produtor de blocos qual é o estado da base de dados, por exemplo, para calcular qual é o respetivo *bucket* de um valor é necessário obter quantos *buckets* tem determinada coluna. Devido a este facto existe uma componente denominada por Database Connector que é responsável por comunicar e executar as diferentes *queries* a perguntar pelo estado da base dados.

Por último teremos o SQL Converter que vai utilizar o Database Connector para comunicar com a base de dados e o Encrypt Sql Operation para cifrar as *queries*. Este também é responsável por realizar *equality queries, range queries* e *joins*. Este processo é extenso, onde por exemplo se o cliente efetuar um WHERE na *query*, será necessário

passar essa coluna para o SELECT da *query*, visto que na abordagem utilizada para efetuar uma procura é necessário usar os *buckets*.

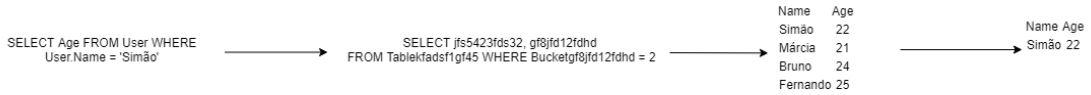

Figura 4.6- Filtragem de uma *query* WHERE

Para calcular o *bucket* necessitamos saber o tipo da tabela, por exemplo, se for uma *RangeColumn* é necessário pedir quantos *buckets* aquela coluna tem e qual é o valor máximo que aquela coluna pode ter.

Existe também a necessidade de verificar se a coluna que está no WHERE se já esta no SELECT, se estiver não é necessário adicionar.

E por último dependendo do tipo de *query* é necessário pedir, além do valor, o seu vetor de inicialização.

Este processo necessita de várias *queries* simples para perguntar qual é o estado da base de dados, como por exemplo, qual é o numero de *buckets* e qual é o valor máximo de uma coluna para tal temos uma cache para guardar estes valores, diminuindo assim o numero de comunicações e aumentando significativamente a performance do tempo de resposta da base de dados.

4.5 Resultados

Foram feitos testes com base na execução de várias operações, onde a execução envolve: (1) cifra das operações SQL, (2) envio da operação SQL para um dos produtores de blocos, (3) produtor de blocos valida e executa a operação SQL, (4) produtor de blocos envia o resultado ao cliente, (5) o cliente decifra e processa o resultado. Estes testes foram feitos onde o cliente e o produtor de blocos estão na mesma maquina para anular o tempo e falhas de comunicação.

Um problema encontrado durante estes testes foi que, para cada valor que era necessário cifrar ou decifrar, estávamos a fazer um pedido ao produtor de blocos para obter o tamanho dos buckets. Para resolver este problema em vez de fazer um pedido de cada vez, vamos pedir todos os tamanhos dos buckets necessários num só pedido. E ainda conseguimos melhorar o tempo com a utilização de uma cache para diminuir o numero de pedidos.

Foram feitos vários *unit tests* com as diferentes operações SQL, para verificar se as operações comportam-se como esperado.

Na pesquisa de valores é difícil de mostrar métricas com base nos tempos, devido na pesquisa em NormalColumns e RangeColumns ser uma procura probabilística, onde dependendo da configuração da base de dados, como por exemplo a quantidade de *buckets* que o utilizador pretenda que determinada coluna tenha, vai fazer com que tenha de filtrar mais ou menos resultados. Outro facto é que se efetuarmos uma pesquisa numa UniqueColumn, PrimaryColumns e ForeignColumns vamos obter um desempenho muito melhor, isto porque neste tipo de colunas não existe a necessidade de filtrar o resultado.

Mas um facto interessante encontrado durante estes testes é que quanto mais específica for a pesquisa em NormalColumns e RangeColumns menor será o tempo de execução da *query*. Isto é simples de demonstrar com um exemplo. Se procurar por André e estiver contido num *bucket* que contém vários nomes, o cliente tem de filtrar e decifrar todos estes nomes até achar o André. Se especificarmos mais a *query* onde procuramos também pela idade, já vamos estar a filtrar ainda mais os resultados. Onde o resultado será todos os valores que estejam no mesmo *bucket* que a idade que estamos a pesquisar e no mesmo *bucket* que André.

Por ultimo, após os produtores de blocos executarem as transações enviadas pelos clientes, as tabelas da base de dados vão ficar com o aspeto da Figura 4.7, onde todos os seus valores vão estar cifrados e acompanhados com o seu vetor de inicialização e os seu respetivo *bucket*.

ValuedIncojfn6k3wnah3wsio3joqac	IvdIncojfn6k3wnah3wsio3joqac	Bucketd...	Valuexmy7fzm696apjqag49r...	Ivxmy7fzm696apjqag49r1hwar	Bucketxmy7fzm696apjqag49r1
0xE8A347A36906D1E6D13D2DED...	0xD713A6F3D84283BF88DF...	0x07	0x1DA53708126B70EA4C...	0xFAC7FBDCE207872A95B4...	0x652F5F4978A5EEBEA18B
0xE0E876C81FF3ADA663086E27F...	0xADF21CE30114C3DF3CC...	0x02	0xED1EAB9BDD9E442C8...	0xA80F98BD5A18F3CA6245...	0x891D92EA95A4EC13A87C
0xBAACB06C06743E0044CE15AFC...	0x543EDC68CAE4FAE41F26...	0x01	0x52BBE735174DB382CB...	0xE618025E819FEE8FBB32...	0xCF1132F50B1F799F3D99I
0x546EA81A6542AA3DA497EE642...	0xDC99E1E9F64857126AC5...	0x07	0x63FED5AEE9A36A2BAC...	0xEF6260A29C0BEB0409C8...	0x77D6AD72CF1CB3D880A(
0x10F63B6B31D3817F661D448A0...	0x5772DDD6AA131CD45DD...	0x05	0x1B803759C0F4A8F7394...	0x8618D90EC069B7922F7A...	0x3FB82EC1AEC08C281467
0x2D50F6CCFEA57D925B2D1BC4...	0xBC0C9F155442A5213504...	0x05	0x897AE8FC9D10F7C806...	0xFFF5B2CF2C6273BCB595...	0x77D6AD72CF1CB3D880A(
0x25CB2AC086F3A8C88B368779F...	0x51F77ACAFA2F91FA1086...	0x01	0x9D9B311A0B0B3D68A8...	0x422D92B8A45533DB12BE...	0x503163FA50DA126316EFI
0xB9CFF075E210A4B88BBED4739...	0xF966FD51C91F420457613...	0x00	0x17E4336032ED7DD940...	0x9C9445D44E37868F445E...	0xFE2CABFE1298504F5DB8
0x1200CFEECAE5C99420558606 6...	0x4A44DD73E2885037DD6...	0x06	0x39829818 6AAE47B4520...	0x4094C77EE2871863DD6F...	0x652F5F4978A5EEBEA18B
0x5E760908DCC22DC7ECBCC947...	0x9E0BB1CDE3DC61ECE6B...	0x05	0xAFCD46045C9240E956...	0x88914D19AA110D632260...	0xDB054CC7396E13EFAC2F

Figura 4.7- Base de dados cifrada

5

5 Conclusões e Trabalho Futuro

Como foi referido várias vezes durante esta dissertação, garantir a privacidade dos dados é bastante importante nos tempos atuais. A informação é valiosa quer para o proprietário quer para as empresas. Determinou-se que existe uma grande negligencia na implementação de segurança nos sistemas. A segurança dos sistemas é efetuada principalmente com a utilização da criptografia.

Este facto levou-nos a estudar os principais algoritmos de cifra, e as diferentes técnicas de como cifrar o conteúdo. Depois de entender como estes algoritmos funcionam, procedemos ao estudo das diferentes bases de dados cifradas. A principal foi a CryptDB. Nas outras bases de dados cifradas, o seu desenvolvimento tinha sido abandonado ou tinham o mesmo processo de segurança que a CryptDB.

Para esta dissertação foi importante o estudo das diferentes *blockchains* para posteriormente ser possível efetuar a integração da base de dados cifrada com a *blockchain*. Sem este estudo, não era possível implementar as diferentes estruturas, transações e a persistência da própria *blockchain*.

A primeira abordagem foi utilizar a mesma metodologia que a CryptDB. Com o desenvolvimento desta metodologia foram encontrados vários problemas de segurança e de desempenho. Um dos problemas encontrados na CryptDB é que, se o cliente efetuar uma *equality query* sobre uma coluna, teremos de mostrar os valores iguais ao administrador da base de dados, o que permite a um atacante revelar os dados cifrados.

Consequentemente foi imperativo o estudo de uma nova abordagem onde não poderíamos mostrar valores iguais na base de dados e que permitisse de forma semelhante efetuar pesquisas sobre dados cifrados.

Esta nova abordagem tem como base uma pesquisa probabilística, onde todos os valores mesmo sendo iguais serão apresentados com valores diferentes. E sempre que for pesquisado um valor poderá devolver o valor correto ou não, enganando os atacantes que estão a visualizar o que está a acontecer na base de dados. Isto faz com que diminua o desempenho porque o cliente terá de filtrar o resultado recebido.

Na abordagem utilizada é possível efetuar *joins* (apenas sobre as chaves primárias), *equality queries* e *range queries.* Uma técnica de criptografia que se pode utilizar para poder efetuar SUM (somatório) ou AVG (média) sobre os dados é utilizar criptografia homomórfica. O problema desta técnica é que ainda apresenta um processo demorado e ainda não existem implementações para a língua de programação para o qual o projeto foi desenvolvido. No futuro esta técnica poderia ser considerada, caso surja um algoritmo novo de criptografia homomórfica com um desempenho bastante melhor.

Um problema que poderá ocorrer nesta abordagem é no cálculo do *bucket* onde irá ficar um determinado valor. Onde existe uma probabilidade de um *bucket* ficar associado a vários valores destintos e outros *buckets* sem valores. O que fará com que a pesquisa sobre os dados seja bastante lenta. Uma melhoria seria arranjar uma forma de distribuir os valores de forma equilibrada, ou seja, todos os *buckets* teriam o mesmo número de valores associados.

Em suma, o desenvolvimento desta dissertação permitiu-nos verificar o potencial de fazer uma pesquisa probabilística. Na solução desenvolvida conseguimos ter uma melhor segurança à custa do desempenho. Esta veio a corrigir alguns problemas da CryptDB, como por exemplo, um dos problemas da CryptDB é que apesar de os valores estarem cifrados, estes mostram as igualdades. O que na nossa solução conseguimos esconder as igualdades entre valores.

Outro problema comum encontrado nos esquemas de cifra para permitir efetuar *range queries*, é que a maior parte são lentos e mostram igualdades ou revelam grande parte dos bits mais significativos. O esquema utilizado para permitir efetuar *range queries* da solução proposta apenas permite a um atacante revelar a ordem de valores.

Contudo, o estudo destas abordagens permite-nos construir um sistema onde conseguimos garantir a confidencialidade dos dados, guardando-os remotamente, conseguindo efetuar pesquisas sobre os dados, revelando o mínimo de informação para efetuar as mesmas.

Bibliografia

[1] M. Malik e T. Patel, "Database Security - Attacks and Control Methods," *International Journal of Information Sciences and Techniques,* vol. 6, pp. 175-183, 2016.

[2] A. H. Almutairi e A. H. Alruwaili, "Security in Database Systems," *Global Journals Inc.,* vol. 12, nº 17, 2012.

[3] L. Xu, C. Jiang, J. Wang, J. Yuan e Y. Ren, "Information Security in Big Data: Privacy and Data Mining," *IEEE Access,* vol. 2, pp. 1-28, 1 2014.

[4] "Privacy Rights Clearinghouse," [Online]. Available: https://www.privacyrights.org/data-breaches .

[5] R. Herian, "Regulating Disruption: Blockchain, Gdpr, and Questions of Data Sovereignty," *Journal of Internet Law,* pp. 1-16, 2018.

[6] "GDPR," [Online]. Available: https://eugdpr.org/.

[7] L. Teran e A. Meier, "Encryption techniques: A theoretical overview and future proposals," em *Third International Conference on eDemocracy & eGovernment (ICEDEG),* 2016.

[8] R. Weber, "An Advisor's Introduction to Blockchain.," *Journal of Financial Service Professionals,* vol. 72, pp. 49-53, 2018.

[9] D. Fullmer e A. S. Morse, *Analysis of Difficulty Control in Bitcoin and Proof-of-Work Blockchains,* 2018, pp. 5988-5992.

[10] J. Li, N. Li, J. Peng, H. Cui e Z. Wu, "Energy consumption of cryptocurrency mining: A study of electricity consumption in mining cryptocurrencies," *Energy,* vol. 168, pp. 160-168, 2019.

[11] "EOS.IO white paper," [Online]. Available: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md.

[12] B. Haddock, "How the Facebook data breach has provided lessons for other businesses: Every business needs to guard customer data from exposure," *Wyoming Business Report,* vol. 19, p. 13, 2018.

[13] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," [Online]. Available: https://bitcoin.org/bitcoin.pdf.

[14] A. Antonopoulos, Mastering Bitcoin: Programming the Open Blockchain, O'Reilly Media, 2014.

[15] H. Hellani, A. E. Samhat, M. Chamoun, H. E. Ghor e A. Serhrouchni, "On BlockChain Technology: Overview of Bitcoin and Future Insights," em *IEEE International Multidisciplinary Conference on Engineering Technology*, 2018.

[16] N. v. Saberhagen, "CryptoNote v 2.0," [Online]. Available: https://cryptonote.org/whitepaper.pdf.

[17] S. Noether e S. Noether, "Monero is Not That Mysterious," 2014. [Online]. Available: https://ww.getmonero.org/.

[18] D. A. Wijaya, J. Liu, R. Steinfeld e D. Liu, "Monero Ring Attack: Recreating Zero Mixin Transaction Effect," Cryptology ePrint Archive, 2018.

[19] L. Wang, X. Shen, J. Li, J. Shao e Y. Yang, "Cryptographic primitives in blockchains," *Journal of Network and Computer Applications,* vol. 127, pp. 43-58, 2019.

[20] D. Larimer, "Introducing EOS.IO Storage," [Online]. Available: https://github.com/EOSIO/Documentation/blob/master/EOS.IO%20Storage.pdf.

[21] P. Singh e P. Shende, "Symmetric Key Cryptography: Current Trends," *International Journal of Computer Science and Mobile Computing,* vol. 3, nº 12, pp. 410-415, 2014.

[22] W. Stallings, Network security essentials: Applications and standards, Pearson, 2017.

[23] D. Blazhevski, A. Bozhinovski, B. Stojchevska e V. Pachovski, "Modes of operation of the AES algorithm," em *The 10th Conference for Informatics and Information Technology*, 2013.

[24] S. Sharma e V. Chopra, "Analysis of AES encryption with ECC," em *Proceedings of International Interdisciplinary Conference On Engineering Science & Management* , 2016.

[25] M. Agrawal e P. Mishra, "A Comparative Survey on Symmetric Key," *International Journal on Computer Science and Engineering,* vol. 4, nº 5, pp. 877-882, 2012.

[26] A. Boldyreva, N. Chenette e A. O'Neill, *Order-preserving encryption revisited: Improved security analysis and alternative solutions.*, P. Rogaway, Ed., Springer, Berlin, Heidelberg, 2011, p. 578–595.

[27] M. Naveed, S. Kamara e C. V. Wright, "Inference Attacks on Property-Preserving Encrypted Databases," em *ACM Conference on Computer and Communications Security*, 2015.

[28] R. A. Popa, C. M. S. Redfield, N. Zeldovich e H. Balakrishnan, "CryptDB: Protecting Confidentiality with Encrypted Query Processing," *SOSP'11 - Proceedings of the 23rd ACM Symposium on Operating Systems Principles,* pp. 85-100, 2011.

[29] "BigchainDB 2.0 The Blockchain Database," 2018. [Online]. Available: https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf.