UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

ARTHUR BEMFICA SACHET

ARMAZENAMENTO DE ARQUIVOS EM UMA REDE P2P UTILIZANDO *BLOCKCHAIN*

CAXIAS DO SUL

2019

ARTHUR BEMFICA SACHET

ARMAZENAMENTO DE ARQUIVOS EM UMA REDE P2P UTILIZANDO *BLOCKCHAIN*

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Prof. Dr. André Luis Martinotto

CAXIAS DO SUL

2019

ARTHUR BEMFICA SACHET

ARMAZENAMENTO DE ARQUIVOS EM UMA REDE P2P UTILIZANDO *BLOCKCHAIN*

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado em 28/11/2019

BANCA EXAMINADORA

Prof. Dr. André Luis Martinotto
Universidade de Caxias do Sul - UCS

Prof. Dra. Maria de Fatima Webber do Prado Lima
Universidade de Caxias do Sul - UCS

Prof. Dra. Helena Graziottin Ribeiro
Universidade de Caxias do Sul - UCS

RESUMO

A *blockchain* simula um livro-razão, utilizado pela contabilidade de um empresa para registrar as transações realizadas por essa. O objetivo dessa tecnologia é criar consenso, confiança e segurança em trocas que envolvem duas partes sem a necessidade do intermédio de terceiros. O número de propostas de aplicações, utilizando a tecnologia de *blockchain*, vêm aumentado devido ao sucesso das criptomoedas, como por exemplo, o *bitcoin* criado por NAKAMOTO, 2008. Dentro dessas aplicações, uma que vem recebendo destaque é o armazenamento de arquivos descentralizados.

Este trabalho apresenta como principal objetivo desenvolver uma solução que possibilite o armazenamento de arquivos em uma rede P2P utilizando uma *blockchain* para o gerenciamento das transações. A solução baseia-se na integração das bibliotecas *CryptoJS*, *InterPlanetary File System* e a plataforma de *blockchain* da *Stellar*. Nesta os arquivos são subdivididos em partes de 256 *Kbytes* e a biblioteca *CryptoJS* é utilizada para a criptografar as partes no *upload* e descriptografar na recuperação dessas partes. A ferramenta IPFS é utilizada para o armazenamento das partes dos arquivos em uma rede P2P, sendo que a localização das partes é realizada através de *hashs* gerados pelo IPFS. Por fim, os *hashs* gerados pelo IPFS são armazenados na *blockchain* da *Stellar*.

Palavras-chaves: *Blockchain*, Armazenamento Descentralizado, IPFS, P2P, PoW, PoS, PoET, PBFT, FBA, Stellar

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

P2P	*Peer to Peer*
PoW	*Proof of Work*
PoS	*Proof of Stake*
PoET	*Proof of Elapsed Time*
TEE	*Trusted Execution Environment*
SGX	*Intel's Software Guard Extensions*
PBFT	*Practical Byzantine Fault Tolerance*
FBA	*Federated Byzantine Agreement*
IPFS	*InterPlanetary File System*
IPNS	*InterPlanetary Naming System*
DHT	*Distributed Hash Table*
DAG	*Directed Acyclic Graph*
TCP	*Transmission Control Protocol*
SCTP	*Stream control transmission protocol*
SSH	*Secure Shell*
NAT	*Network Address Translation*
SDK	*Software development kit*
MD5	*Message-Digest algorithm 5*

SUMÁRIO

1 INTRODUÇÃO

A criação da criptomoeda *bitcoin* por (NAKAMOTO, 2008) e de outras criptomoedas provocou um aumento no interesse na tecnologia de *blockchain*, surgindo várias propostas de aplicações que usam essa tecnologia. Dentro dessas, uma que vem recebendo destaque é o armazenamento de arquivos descentralizados (ZYSKIND; NATHAN et al., 2015), (WILKINSON; LOWRY; BOSHEVSKI, 2014). De fato, diversas soluções para armazenamento de arquivos descentralizados que utilizam o conceito de *blockchains* foram desenvolvidas, sendo que dentre elas pode-se destacar o projeto *Filecoin* (BENET; GRECO, 2017).

A tecnologia de *blockchain* permite a troca de dados entre duas partes sem a necessidade do intermédio de terceiros. Essa tecnologia consiste basicamente em uma lista de blocos, onde cada bloco contém: a sua data de criação, um conjunto de dados, um *hash* e o *hash* do bloco anterior, criando assim uma cadeia (BASHIR, 2017). Essa cadeia fica hospedada em uma rede descentralizada de computadores, onde cada um dos nós contém uma cópia do *blockchain*. Quando um nó adiciona um novo bloco na cadeia ele precisa ser replicado em todos os outros nós para manter a consistência, não existindo a opção de remoção ou atualização de um bloco. Já na recuperação de um dado é verificado se cada nó possui uma cadeia equivalente e válida, garantindo assim a confiabilidade da informação. O *blockchain* precisa ser hospedado em uma rede descentralizada, para que um único nó não seja o responsável por assegurar a veracidade da informação.

Algumas operações são necessárias para o armazenamento de arquivos em uma estrutura que utiliza a tecnologia de *blockchains*. Primeiramente, o usuário que deseja adicionar um arquivo na rede realiza a divisão desse arquivo em partes menores, sendo que cada parte é criptografada e em seguida é gerado o seu *hash*. Para garantir uma alta disponibilidade cada parte do arquivo é replicada e distribuída entre os diferentes nós da rede. Para cada parte do arquivo é criado um novo bloco na *blockchain*, que contém alguns dados relevantes sobre a parte do arquivo, como por exemplo, a localização e o *hash* dessa. Após o processo o bloco é adicionado na *blockchain*, que é sincronizada em toda a rede para manter a consistência.

O armazenamento de arquivos em uma rede descentralizada com *blockchain* apresenta um menor custo se comparado com as nuvens de arquivos atuais, pois não há necessidade de manter servidores com uma grande capacidade de armazenamento (WILKINSON; LOWRY; BOSHEVSKI, 2014). Outro benefício desse tipo de sistema sobre as nuvens convencionais é que, devido a divisão e a criptografia dos dados o acesso ilegal aos dados é uma tarefa complicada e custosa. E, por fim, como a rede é descentralizada, e são feitas cópias dos arquivos, mesmo que alguns nós estejam *offline* ainda é possível

recuperá-los (WILKINSON; LOWRY; BOSHEVSKI, 2014).

Dentro deste contexto, neste trabalho será realizada a criação de uma estrutura para o armazenamento de arquivos distribuídos e de informações de suas transações em uma *blockchain*. O sistema de arquivos distribuído deverá permitir a inclusão e recuperação de arquivos, sendo que os arquivos deverão ser divididos e criptografados antes de serem armazenados.

1.1 OBJETIVOS

Desenvolver um sistema para o armazenamento distribuído de arquivos, onde as informações das transações serão gerenciadas por uma *blockchain*. Com base no objetivo geral, foram elaborados os seguintes objetivos específicos:

1. Divisão do arquivo e criptografia das partes antes do compartilhamento na rede;
2. Utilização do sistema *InterPlanetary File System* (IPFS) para armazenar e recuperar arquivos em uma rede distribuída;
3. Utilização da plataforma de *blockchain Stellar* para o armazenamento dos endereços dos arquivos retornados pelo IPFS;

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está organizado da seguinte forma:

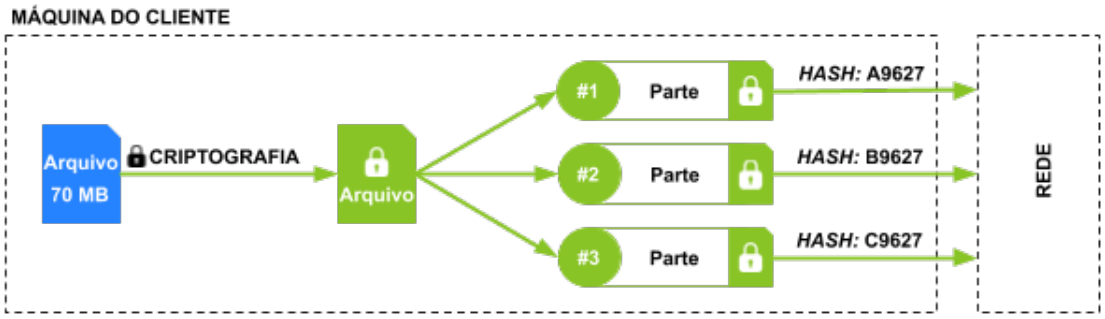
- No Capítulo 2 é apresentado o funcionamento de um sistema de armazenamento descentralizado e uma *blockchain*. Para maiores informações sobre estes assuntos, sugere-se a leitura de (CHRISTIDIS; DEVETSIKIOTIS, 2016) e (WILKINSON; LOWRY; BOSHEVSKI, 2014).
- No Capítulo 3 é apresentada a estrutura da solução utilizada para o desenvolvimento deste trabalho.
- No Capítulo 4 são apresentadas as principais etapas do desenvolvimento da aplicação para o armazenamento de arquivos e os resultados dos testes realizados sobre a implementação.
- Por fim, no Capítulo 5 são apresentadas as considerações finais do trabalho e sugestões de trabalhos futuros.

2 ARMAZENAMENTO DESCENTRALIZADO

Os projetos *Sia* (VORICK; CHAMPINE, 2014), *Storj* (WILKINSON; LOWRY; BOSHEVSKI, 2014) e *Filecoin* (BENET; GRECO, 2017) são exemplos de sistemas de arquivos descentralizados que utilizam *blockchain* para o armazenamento das informações das operações realizadas. Esses sistemas permitem que os usuários disponibilizem uma parte do disco rígido de seus computadores para que outros usuários armazenem os seus arquivos. Esses sistemas de arquivos descentralizados operam de maneira similar e contam com suas próprias criptomoedas, as quais são utilizadas para pagar a quantidade de armazenamento utilizada.

Esses sistemas efetuam a divisão do arquivo em partes menores, processo conhecido como *sharding*, ainda no computador solicitante. Em seguida, é calculado o *hash* do arquivo, que é utilizado tanto como um identificador do arquivo bem como para verificar uma adulteração no mesmo. Caso o arquivo tenha sido modificado após seu envio o *hash* será diferente. O arquivo então é duplicado e enviado para nós distintos, aumentando assim a redundância e a disponibilidade do sistema (WILKINSON; LOWRY; BOSHEVSKI, 2014). Nesses sistemas o arquivo é criptografado antes de ser enviado para a rede (WILKINSON; LOWRY; BOSHEVSKI, 2014). Na Figura 1 tem-se uma ilustração desse processo.

Figura 1: Visualização do processo de *sharding*.



Fonte: WILKINSON et al. (2016), adaptado

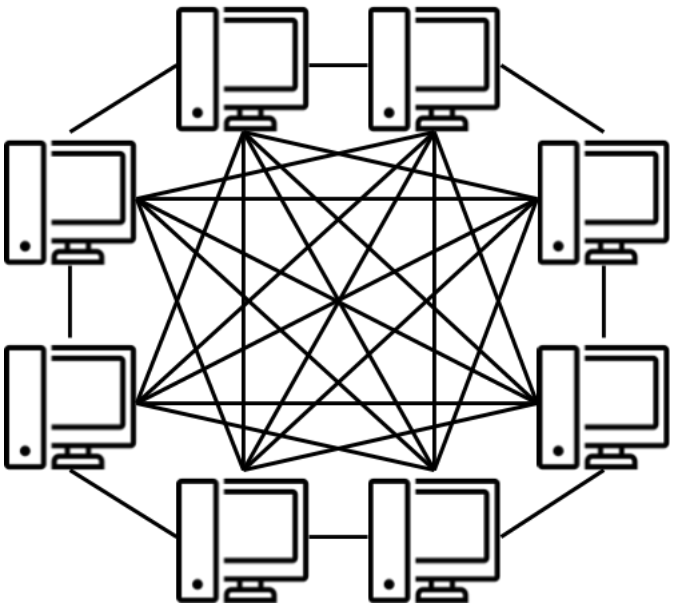
Destaca-se que o armazenamento do arquivo na *blockchain* não é viável devido à necessidade de armazenar uma grande quantia de dados, que é conhecido como *blockchain bloat*. Esse problema é resolvido armazenando na *blockchain* apenas os dados relevantes da transação, como por exemplo: o identificador do nó que armazena cada parte do arquivo, o *hash* do arquivo salvo, o tamanho do arquivo, o valor cobrado, etc (WILKINSON; LOWRY; BOSHEVSKI, 2014).

2.1 *BLOCKCHAIN*

A *blockchain* é uma tecnologia que simula um livro-razão, que é utilizado na contabilidade das empresas para registrar as transações realizadas por essas. Porém, a *blockchain* é mantida de forma pública de forma que todos possuem acesso às transações (PECK, 2017). Além disso, a *blockchain* têm como objetivo criar um consenso, confiança e segurança em trocas que envolvem duas partes sem a necessidade do intermédio de terceiros (NAKAMOTO, 2008).

A rede utilizada para a implementação de uma *blockchain* é do tipo P2P (*peer-to-peer*) (NAKAMOTO, 2008). Uma arquitetura P2P proporciona uma rede descentralizada, onde não existe noção de cliente ou servidor e sim pares de nós que realizam ao mesmo tempo as tarefas de cliente e de servidor (SCHOLLMEIER, 2001). Esses nós são chamados de *Servent*, palavra essa que é originada da primeira sílada de *server* (*"Serv-"*) e da segunda sílaba de *client* (*-ent"*) (SCHOLLMEIER, 2001). Além disso, em uma rede P2P todos os nós estão interconectados, permitindo o acesso de qualquer nó para qualquer nó (Figura 2). No caso de uma rede P2P de *blockchain*, cada um dos nós contém uma cópia da cadeia de blocos.

Figura 2: Representação de uma rede P2P.

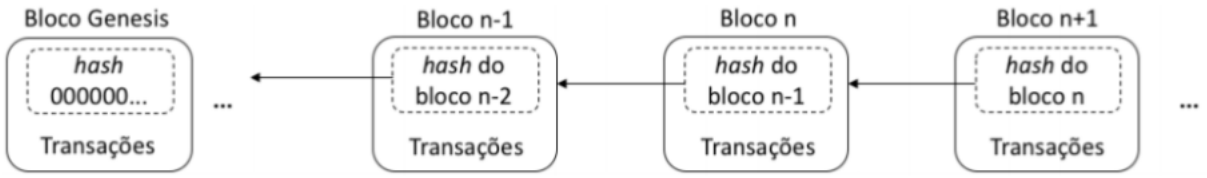


Fonte: o autor (2019)

Uma *blockchain* é formada por blocos ligados uns aos outros criando uma cadeia (Figura 3). Os blocos contidos em uma *blockchain* incluem um *hash*, um *hash* do bloco anterior, uma data e dados relacionados a transações (BASHIR, 2017). Os dados das transações podem variar de acordo com a aplicação. O *hash* do bloco é calculado a partir dos dados presentes no bloco. Caso ocorra uma alteração no bloco será gerado um novo

hash, ocasionando assim a quebra da cadeia, pois o bloco seguinte não terá a referência do bloco anterior. Essa quebra acontece também se algum bloco é excluído. Desta forma, a única operação permitida em uma *blockchain* é a inserção. O primeiro bloco da *blockchain*, chamado de gênesis, é um bloco especial pois ele não contém o *hash* do bloco anterior (SINGH; SINGH, 2016). Esse é criado manualmente assim que a *blockchain* é instanciada.

Figura 3: Representação da cadeia de blocos.



Fonte: OLIVEIRA et al. (2018)

As funções *hash* utilizadas na *blockchain* são de sentido único, ou seja, são funções que possuem como entrada um dado x de qualquer tamanho e produzem como saída um valor y de tamanho fixo e pequeno (MERKLE, 1990). Um exemplo de uma função de sentido único é a função SHA-256 que, a partir de um conjunto de dados de qualquer tamanho, gera uma *string* de 256 *bits* (GILBERT; HANDSCHUH, 2003).

Uma outra propriedade da função de sentido único é o alto custo computacional para encontrar um dado x que gere o mesmo *hash* de um dado z, isto é $F(x) = F(z)$ e $x != z$ (NAOR; YUNG, 1989). Essa propriedade é importante para que não sejam criados dois *hashs* iguais a partir de dados diferentes. Isso evita a alteração dos dados de blocos já inseridos na cadeia e que um *hash* de bloco anterior referencie dois blocos diferentes (quebra da cadeia). Existem inúmeras funções *hash* de sentido único, sendo que entre essas pode-se destacar as funções que são usadas na *blockchain* do *bitcoin*: a SHA256, que é usada para a criação do *hash* do bloco; e a RIPEMD160, que é usada na criação de endereços de *bitcoin* (NAIK; COURTOIS, 2013).

As transações que são registradas na *blockchain* podem ser de qualquer tipo, dependendo apenas do objetivo da aplicação utilizada. Em uma transação monetária por exemplo, o registro salvo no bloco possui dados como: quem está pagando, quem está recebendo, o valor, a data, etc.. Já em um sistema de arquivos distribuído o bloco possui informações relativas ao nó que armazena o arquivo, o *hash* do arquivo, o tamanho do arquivo, o valor cobrado para o armazenamento do arquivo, etc. (WILKINSON; LOWRY; BOSHEVSKI, 2014).

Com o objetivo de serem incluídas na cadeia, uma transação precisa ser validada pela rede e então será incluída em um bloco que, posteriormente, será adicionado na *blockchain*. A validação da transação é feita a partir de regras definidas pelo sistema, sendo que cada

blockchain apresenta suas próprias regras de validação (CHRISTIDIS; DEVETSIKIOTIS, 2016). Para as transações serem validadas e incluídas na cadeia se faz necessário a utilização de um mecanismo de consenso distribuído. A escolha do mecanismo de consenso depende de características como, por exemplo, se a cadeia é pública ou privada bem como o tipo de ataque que a cadeia quer evitar (CHRISTIDIS; DEVETSIKIOTIS, 2016). Nas próximas seções serão apresentados os mecanismos de consenso que são utilizadas nas principais *blockchains* existentes. Posteriormente, será apresentado o mecanismo de consenso que será utilizado no desenvolvimento deste trabalho.

2.2 MECANISMOS DE CONSENSO

O mecanismo de consenso é um algoritmo implementado com o objetivo de impedir que as *blockchains* divirjam e criem bifurcações. Ou seja, o mecanismo de consenso evita que os nós da rede contenham *blockchains* diferentes entre si e são usados para a validação dos dados antes da inserção na *blockchain* (CHRISTIDIS; DEVETSIKIOTIS, 2016).

O mecanismo de consenso deve apresentar três propriedades: *safety*, *liveness* e *fault tolerance*. A propriedade *safety*, define que todos os nós devem produzir o mesmo resultado para uma transação, e que esse resultado deve ser válido de acordo com o mecanismo de consenso. O *liveness*, determina que todos os nós não falhos, que participam do consenso, produzam um valor. Já a *fault tolerance* estabelece que o mecanismo deve se recuperar no caso da falha de um dos nós (BALIGA, 2017).

Segundo FISCHER; LYNCH; PATERSON, o mecanismo de consenso não garante que as três propriedades sejam atendidas de forma simultânea. Considerando-se que a propriedade de *fault tolerance* é essencial em um sistema distribuído, o mecanismo de consenso tende a optar como uma segunda propriedade, entre a *safety* e a *liveness* (BALIGA, 2017). A escolha da segunda propriedade depende das características as quais o mecanismo de consenso busca atender.

O mecanismo de *fault tolerance* procura resolver dois tipos de falhas. O primeiro deles é o *fault-stop*, onde um nó deixa de participar do mecanismo de consenso, por uma falha de *software* ou *hardware*. O segundo deles é chamada de *Byzantine faults*, sendo conhecido como *Byzantine General's Problem* (BALIGA, 2017). Esse nome é oriundo do problema dos generais bizantinos, onde várias divisões do exército bizantino cercam uma cidade inimiga, sendo que cada divisão é comandada pelo seu general. Os generais precisam concordar em atacar ou fugir, comunicando-se apenas por mensageiros. Entretanto, existe a possibilidade de alguns generais serem traidores tentando impedir os generais honestos de chegarem a um acordo (LAMPORT; SHOSTAK; PEASE, 1982). No caso de um sistema distribuído os nós correspondem aos generais e o mecanismo de consenso, deve ser resistente à falhas de nós, particionamento da rede, atraso de mensagens, mensagens fora

de ordem e mensagens corrompidas. Além disso, o mecanismo de consenso deve lidar com nós maliciosos.

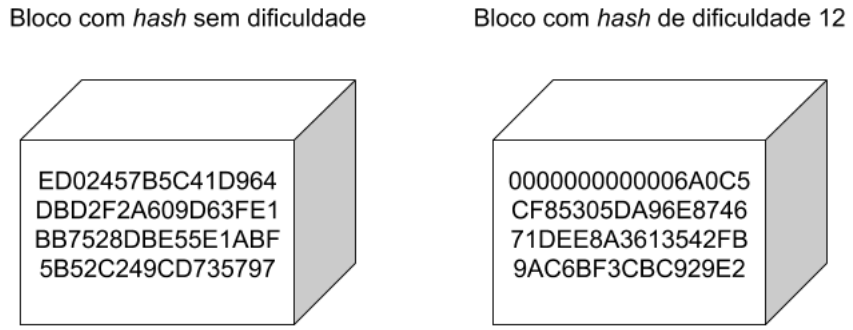
Existem vários modelos de consenso para *blockchain*, sendo que os mais conhecidos são: *proof of work*, *proof of stake*, *proof of elapsed time*, *byzantine fault tolerance* e *federated byzantine agreement* (BALIGA, 2017). Para o desenvolvimento da proposta de solução deste trabalho, a *blockchain* deve conter um mecanismo de consenso que atenda aos requisitos de velocidade de transação alta, finalização das transações imediata e alta escalabilidade da rede.

2.2.1 *Proof of Work*

O *proof of work* (PoW) é o modelo de consenso utilizado pela *blockchain* da *bitcoin* (NAKAMOTO, 2008). Esse modelo é computacionalmente custoso no que diz respeito a criação de um novo bloco.

Na criação do novo bloco, o *hash* obtido a partir do bloco deve obedecer um parâmetro chamado de *difficulty*. Essa *difficulty* representa uma quantia de zeros, em sequência, que o *hash* deve conter no seu início (Figura 4) (CHRISTIDIS; DEVETSIKIOTIS, 2016). A *difficulty* é um valor variável que leva em consideração a quantidade de blocos gerados por hora na rede. Assim, se muitos blocos foram criados a *difficulty* aumenta (NAKAMOTO, 2008). Desta forma, o trabalho necessário para calcular o *hash* aumenta exponencialmente com o aumento do número de blocos (MACDONALD; LIU-THORROLD; JULIEN, 2017).

Figura 4: Exemplo de *hash* sem *difficulty* e *hash* com *difficulty*.

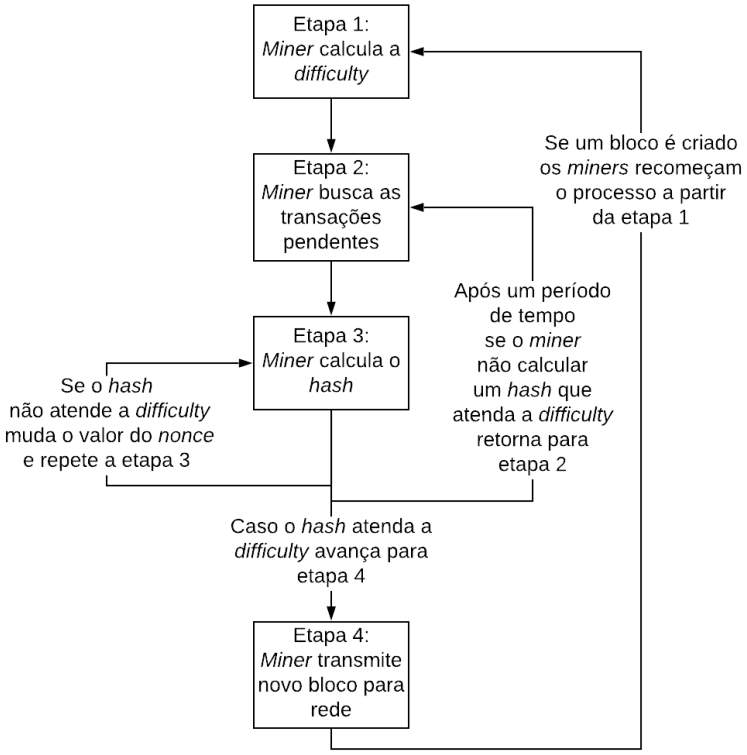


Fonte: o autor (2019)

O modelo PoW implementa a *difficulty* para que os nós não possam criar inúmeros blocos em um período curto de tempo. Caso contrário, seria possível criar blocos com transações fraudadas que seriam incluídas na *blockchain*. Assim, no modelo PoW a criação do novo bloco é uma competição de poder computacional entre os nós, onde os nós com maior poder computacional têm uma probabilidade maior de calcular um *hash* que satisfaça

a *difficulty*. O processo de criação é conhecido como *mining*, sendo que os nós que atuam nesse processo são chamados de *miners*. Como pode ser observado na Figura 5, o processo de *mining*, é dividido em quatro etapas.

Figura 5: Diagrama do processo de mineração



Fonte: o autor (2019)

Na primeira etapa os *miners* calculam o parâmetro *difficulty*. Na segunda etapa os *miners* buscam as transações pendentes da fila. Essas transações são todas as transações realizadas após a criação do último bloco. O nó então calcula o *hash* do novo bloco, levando em conta os dados das transações, o *hash* do bloco anterior e um valor randômico chamado de *nonce*. Se o *hash* cumprir com a *difficulty* o nó pode transmitir o novo bloco para a rede. Caso contrário, ele atribui um novo valor para o *nonce* e repete o cálculo do *hash*. A alteração do valor do *nonce* têm como finalidade alterar o *hash* gerado no cálculo. Se após um período de tempo o nó não conseguir calcular o *hash* ele repete o processo a partir da etapa dois. Caso algum *miner* gere o bloco, todos os *miners* voltam a primeira etapa (MINGXIAO et al., 2017).

Após, o nó calcular o novo *hash* ele passa pela prova de trabalho e pode adicionar o novo bloco em sua *blockchain* e transmitir o bloco para os outros nós (NAKAMOTO, 2008). A partir da função de *hash*, os outros nós validam o bloco enviado, adicionando o mesmo, se esse gerar o mesmo *hash*. No caso da *blockchain* da *bitcoin*, o nó que gerou

o bloco ganha uma determinada quantia de novas criptomoedas pelo trabalho realizado, produzindo assim um incentivo para os nós contribuírem (NAKAMOTO, 2008).

Por se tratar de um mecanismo de consenso probabilístico, já que depende da probabilidade de um nó calcular o *hash* antes dos outros, é possível que alguns nós criem um bloco quase que simultaneamente (OLIVEIRA et al., 2018). Caso isso ocorra os outros nós da rede aderem na sua *blockchain* o primeiro bloco recebido e descartam o outro, causando uma bifurcação.

Para ilustrar uma bifurcação, considere um nó A que calcula o *hash* e transmite o bloco para a rede. Ao mesmo tempo, o nó B também cria o bloco e o transmite na rede. Devido a latência da transmissão, os nós que recebem primeiramente o bloco originado de A o adicionam em sua *blockchain* e descartam o originado de B. O inverso acontece com os nós que recebem o bloco de B antes, causando a bifurcação pois alguns nós têm o bloco originado de A e os outros nós o bloco originado de B.

A bifurcação geralmente é resolvida na criação do próximo bloco, uma vez que a probabilidade de que os dois nós criem simultaneamente o novo bloco é baixa. Após a criação do próximo bloco os nós aderem a cadeia mais longa, ou seja a cadeia que contém a maior carga de trabalho. Por isso esse mecanismo recebe o nome de *proof of work* (CHRISTIDIS; DEVETSIKIOTIS, 2016).

Por exemplo, na criação do próximo bloco a probabilidade de dois nós resolverem a prova de trabalho ao mesmo tempo é baixa. Assim, supondo que um nó C com o bloco originado de A gere o novo bloco, e transmita o novo bloco para a rede. Então os nós com o bloco originado de B não conseguiriam adicionar o novo bloco na cadeia pois o *hash* do bloco anterior não é idêntico. Esses nós constatariam que sua cadeia é mais curta e iriam aderir a cadeia mais longa, com os blocos originados de A e C, descartando sua *blockchain* atual.

O modelo *proof of work* têm a carga de trabalho como forma de segurança, uma vez que o tamanho da cadeia é proporcional a carga de trabalho investida nela. Os nós consideram a cadeia mais longa como sendo a cadeia correta. Sendo assim, para alterar um bloco já inserido na *blockchain*, um nó precisa recalcular o *proof of work* do bloco alterado e de todos os blocos seguintes da cadeia para criar uma cadeia mais longa (NAKAMOTO, 2008). De acordo com MINGXIAO et al., para realizar essa alteração, seria necessário um custo computacional equivalente a metade do custo já investido na geração dos *hashs* dos blocos já existentes na *blockchain*.

Existem três desvantagems no modelo PoW, sendo a primeira o desperdício de energia elétrica atríbuido aos inúmeros cálculos de *hash* necessários para a geração de novos blocos. A segunda desvantagem é a baixa velocidade na efetivação das transações devido ao alto custo computacional para a criação de um novo bloco. E, por fim, a última

desvantagem é a concentração do poder de cálculo dos *hashs*. De fato, alguns mineradores podem se agrupar de forma a aumentar a sua capacidade de geração do *hash*, criando assim o que se chama de *mining pool*. Esse *mining pool* pode causar uma centralização na criação dos novos blocos possibilitando fraudes como, por exemplo a alteração da cadeia (MINGXIAO et al., 2017).

2.2.2 *Proof of Stake*

O *proof of stake* (PoS) é o algoritmo de consenso utilizado pela criptomoeda *PPCoin* (KING; NADAL, 2012). Esse foi desenvolvido com o objetivo de solucionar o problema de consumo de energia elétrica necessário para gerar um bloco no modelo PoW (BALIGA, 2017). No modelo *proof of stake* a escolha do nó que irá criar um bloco é realizada considerando outros fatores, como por exemplo a riqueza (quantidade de criptomoedas), recursos computacionais, entre outros (OLIVEIRA et al., 2018). Os nós que desejam criar blocos devem bloquear seus recursos como prova de sua participação (KING; NADAL, 2012).

A chance de um nó ser selecionado é proporcional a quantia de recursos bloqueados (BALIGA, 2017). Por exemplo, se o recurso for monetário e um nó A bloqueou cem *coins* e o nó B bloqueou mil *coins*, o nó B teria dez vezes mais chances de ser o validador. Assim sendo, se um nó detém uma grande quantia de criptomoedas ele pode ser o validador na maioria das vezes ocasionando uma centralização. Para que isso não ocorra, o algoritmo de validação considera ainda o tempo que o nó detém a criptomoeda. Neste caso, a quantidade de criptomoedas é multiplicada pelo tempo que o nó detém essas criptomoedas, o que é chamado de *coin age* (KING; NADAL, 2012).

Por exemplo, se A recebe um pagamento de dez *coins* de B e mantém por noventa dias, dizemos que A acumulou novecentos *coin days*. Se A gastar os dez *coins* do pagamento, o acúmulo de *coin age* também é consumido (KING; NADAL, 2012). Utilizando o *coin age*, o nó que é selecionado para criar o bloco realiza um pagamento para si mesmo, consumindo seu *coin age* (KING; NADAL, 2012). Desta forma, mesmo que o nó possua uma grande quantia de *coins* ele não será sempre o escolhido devido ao desconto do *coin age*.

Em alguns casos dois nós podem ser escolhidos para gerar o novo bloco, criando assim uma bifurcação. Para resolver o problema da bifurcação todos os nós escolhem qual das cadeias eles vão seguir, sendo que a cadeia com a maior taxa de escolha entre todos os nós se torna a cadeia correta, enquanto a outra é descartada.

No caso do modelo PoS, o nó que cria um novo bloco valida todas as transações que contém nele, se todas forem válidas ele o adiciona na *blockchain*. Caso um nó valide uma transação fraudada ele perde parte dos recursos bloqueados. Em algumas *blockchains* de criptomoedas existe um incentivo para o nó validador, ou seja, o nó ganha uma quantia

em criptomoedas pela validação de um bloco (BALIGA, 2017).

2.2.3 *Proof of Elapsed Time*

O *proof of elapsed time* (PoET) é um algoritmo de consenso desenvolvido pela *Intel* e, posteriormente, disponibilizado para o uso da comunidade. Esse modelo foi projetado para ser executado em um ambiente de execução confiável (TEE). Mais especificamente, esse modelo foi desenvolvido para ser executado no ambiente *Intel's Software Guard Extensions* (SGX) (BALIGA, 2017).

O SGX é uma tecnologia que possibilita a criação de ambientes confiáveis que são chamados de *enclaves* (XING; SHANAHAN; LESLIE-HURD, 2016). Os *enclaves* são regiões de memória, onde os dados armazenados e os códigos encontram-se protegidos. Esses podem ser acessados somente através de funções criadas pelo desenvolvedor do *software*, sendo que essa restrição é garantida pelo processador.

O mecanismo PoET usa um modelo de eleição baseada em loteria para a escolha do nó que irá criar e transmitir o novo bloco. Para a escolha do nó, todos os nós devem estar executando em um *enclave* SGX. No processo de eleição, cada nó realiza uma requisição ao *enclave*, que retorna para o nó um cronômetro com um tempo de espera. Assim que o nó recebe a resposta, ele começa uma contagem regressiva do tempo de espera. O nó que terminar primeiro a contagem regressiva será responsável pela criação do novo bloco.

Por exemplo, se os nós A e B fazem uma requisição por um cronômetro. O nó A recebe como resposta um cronômetro com o tempo de espera de cinco minutos. Já o nó B recebe um cronômetro com o tempo de espera de dois minutos. Os nós passam para a fase de espera onde devem efetuar a contagem regressiva. Como o tempo de espera de B é menor, ele se tornará o líder, recebendo um certificado. Então o nó B cria o novo bloco e o trasmite junto com o seu certificado para a rede. O certificado permite que os outros nós validem que o nó B esperou pelo tempo determinado pelo *enclave*.

Devido a latência da rede dois nós podem acabar criando simultaneamente o novo bloco, ocasionando uma bifurcação. Essa bifurcação é resolvida nas próximas criações, pois é improvável que dois nós criem ao mesmo tempo um novo bloco. Assim, todos os nós então escolhem a cadeia de maior tamanho (BALIGA, 2017).

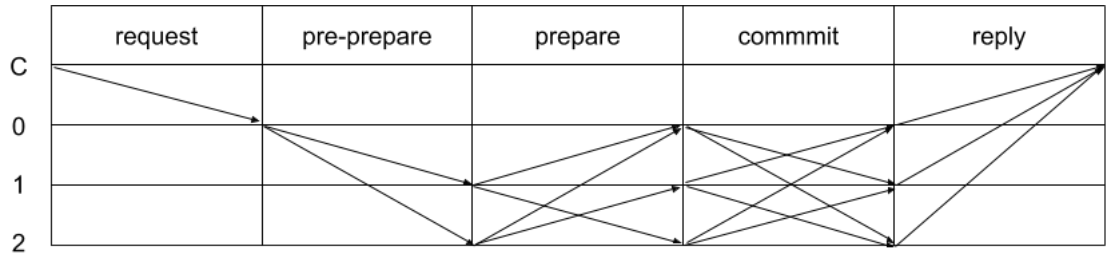
A probabilidade de um nó ser eleito é proporcional ao valor de recursos que ele possui, como por exemplo, quantidade de processadores executando um enclave. O modelo PoET não possui incentivo para o nó que cria o bloco. A principal desvantagem desse modelo é a dependência de processadores *Intel* com suporte ao *Software Guard Extensions* (MACDONALD; LIU-THORROLD; JULIEN, 2017).

2.2.4 *Practical Byzantine Fault Tolerance*

O modelo *practical byzantine fault tolerance* (PBFT) utiliza o conceito de réplicas e uma votação através das réplicas para mudanças de estado. Além disso, esse modelo utiliza criptografia para a troca de mensagens entre as réplicas e os clientes (BALIGA, 2017). Por fim, esse mecanismo necessita da definição de um nó primário.

O PBFT apresenta três fases: *pre-prepare*, *prepare* e *commit*. Na fase *pre-prepare* um nó cliente que deseja criar um novo bloco faz uma requisição para o nó primário que então propaga uma mensagem de *pre-prepare* para todas as réplicas. As réplicas validam se a mensagem *pre-prepare* está correta e então respondem para todos os nós com uma mensagem *prepare*. Os nós que receberam a mensagem *prepare* validam e entram na fase de *commit* onde mandam uma mensagem de *commit* para todos os outros nós. Se os nós receberem as mensagens de *commit* eles respondem a requisição ao nó cliente. O nó cliente então verifica se recebeu o mesmo resultado de todas as réplicas. Em seguida, o nó cliente envia a transação junto com o resultado recebido para um nó, esse nó cria o novo bloco e o transmite para a rede. A Figura 6 demonstra a troca de mensagens, onde *C* é o nó cliente, *0* é a réplica primária e *1, 2 e 3* são as outras réplicas (CASTRO; LISKOV et al., 1999).

Figura 6: Troca de mensagens do PBFT.



Fonte: CASTRO; LISKOV et al. (1999), adaptado

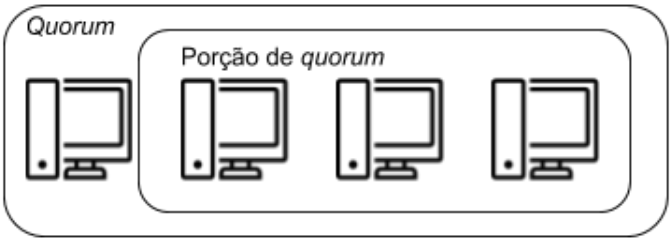
Uma das desvantagens desse modelo é que as réplicas devem ser recomendadas por uma autoridade central, tornando a rede centralizada. Além disso, esse modelo apresenta um alto número de troca de mensagens. De fato, estudos demonstraram que esse modelo apresenta um desempenho satisfatório utilizando no máximo de 20 réplicas. Se o número de réplicas for superior, o modelo apresenta uma sobrecarga de mensagens, uma vez que as réplicas sempre respondem as mensagens para todas as outras réplicas (BALIGA, 2017).

2.2.5 *Federated Byzantine Agreement*

O *federated byzantine agreement* (FBA) é um modelo de consenso alternativo ao *Practical Byzantine Fault Tolerance*. Esse utiliza o conceito de *quorum* e de porções de *quorum* (BACH; MIHALJEVIC; ZAGAR, 2018). O *quorum* é um conjunto de nós suficientes para chegar a um consenso (MAZIERES, 2015). Já a porção de *quorum* é um

subconjunto do *quorum*, onde os nós podem convencer uns aos outros sobre a decisão do consenso (BALIGA, 2017). Na Figura 7 é apresentado um *quorum* de quatro nós, dos quais três deles formam uma porção de *quorum*.

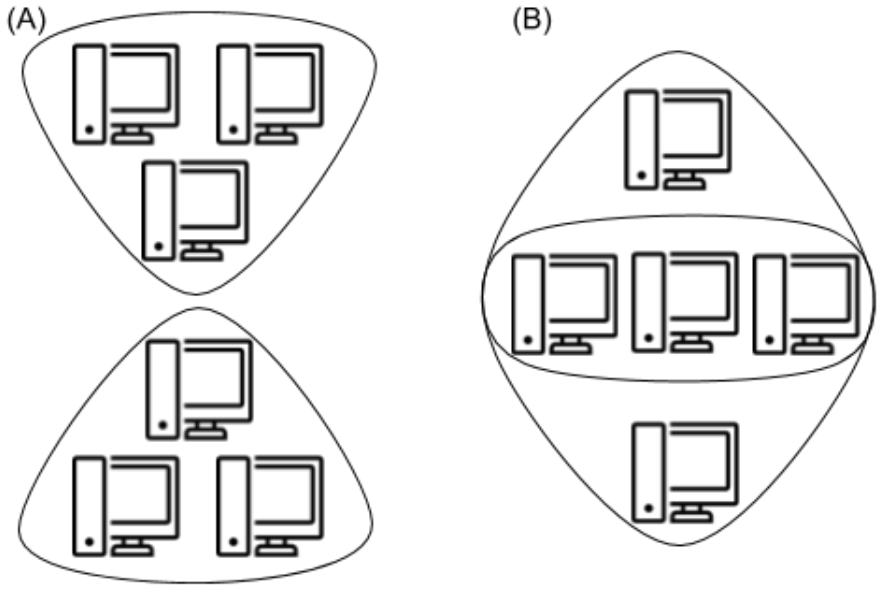
Figura 7: Porção de *quorum*.



Fonte: FOUNDATION (2015), adaptado

Para que a falha de um nó não comprometa o consenso, os nós podem aparecer em uma ou várias porções de *quorum*. Quando um nó pertence a duas ou mais porções de *quorum* ele cria uma interseção entre essas porções (Figura 8) (MAZIERES, 2015).

Figura 8: *Quorums* com porções sem intersecção (A) e com intersecção (B)

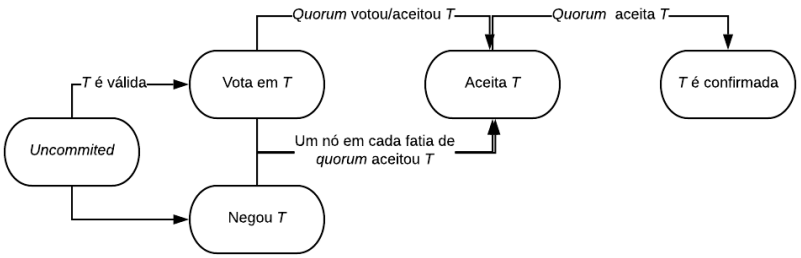


Fonte: FOUNDATION (2015), adaptado

No modelo FBA são realizadas várias rodadas de uma votação federada para chegar a um consenso onde a maioria dos nós precisam aceitar a transação para que ela seja estabelecida. A votação do consenso é dividida em três fases, que são: votação, aceitação e confirmação. A votação federada segue o diagrama na Figura 9.

Em uma rodada da votação, os nós validam a transação e votam nela apenas se ela é válida e se nunca votaram em uma transação que a contradiz. Em seguida, cada nó

Figura 9: Diagrama votação FBA.



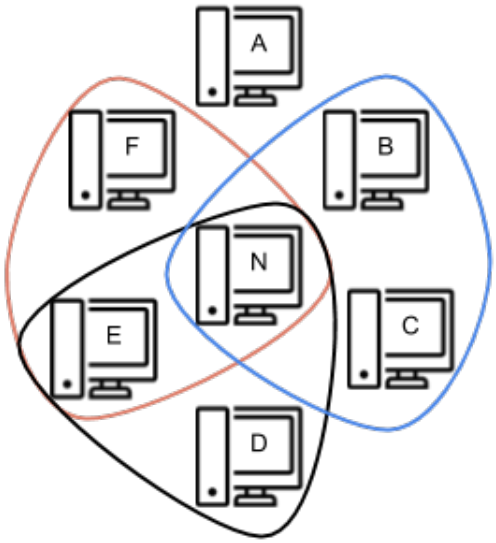
Fonte: MAZIERES (2015), adaptado

transmite uma mensagem para a rede contendo sua identidade, as porções de *quorum* ao qual faz parte e o seu voto.

Os nós, ao receberem os votos, entram na fase de aceitação, onde aceitam ou não a transação. Um nó aceita a transação, se nunca aceitou uma transação contrária e se todos os nós de seu *quorum* também tenham votado a favor ou aceitado a transação. Neste caso, o voto de aceitação é trasmitido para a rede (GLICKSTEIN, 2019).

Porém, pode ser que o *quorum* não concorde imediatamente com o seu voto. Caso isso ocorra, para avançar para a fase de aceitação, o nó busca o que se chama de *blocking set*. O *blocking set* é constituído somente por um nó de cada porção de *quorum* que o nó faz parte (Figura 10). Neste caso, o nó irá aceitar um valor diferente ao que ele votou, se todo o *blocking set* tenha aceitado esse valor (GLICKSTEIN, 2019).

Figura 10: *B-D-F* e *B-E* são exemplos de *blocking sets* de *N*.



Fonte: GLICKSTEIN (2019), adaptado

Por exemplo, na etapa de votação um nó A vota em uma transação T e transmite seu voto para os outros nós, afirmando que T é válida e promete que nunca vai votar em uma transação que contradiz T. Na etapa de aceitação o nó A recebe os votos de todos os nós que participam das suas porções de *quorum*. O nó A aceita a transação T e transmite seu voto de aceitação para a rede se nunca aceitou uma transação que contradiz T e se pelo menos um nó de cada porção de *quorum* de A aceitou a transação. Caso contrário, o nó recusa a transação transmitindo o voto de não aceitação. Esse processo é repetido e os nós podem convencer os outros nós a aceitarem ou recusarem a transação T. Deste modo, toda a rede aceitará ou recusará a transação T (FOUNDATION, 2015).

Ao receberem os votos de aceitação os nós entram na fase de confirmação. O nó só confirma uma transação se todo o seu *quorum* a aceitou. Deste modo o nó chegou ao fim de uma rodada da votação federada (GLICKSTEIN, 2019).

2.2.6 Definição do modelo de consenso

Para que o *upload* do arquivo possa ser realizado de forma rápida torna-se necessário utilizar um mecanismo de consenso com uma alta velocidade de transação. Outro fator a ser considerado, é que após o armazenamento a transação não pode ser cancelada por causa de bifurcações, ou seja, a transação deve ser finalizada de forma imediata. E, por fim, o mecanismo de consenso deve ser escalável para que o tamanho do sistema de armazenamento possa crescer com a inclusão de novos nós. O tipo de *blockchain*, o custo de participação e o modelo de confiança são características que não influenciaram na escolha do modelo de consenso, pois são características que não afetam o funcionamento do sistema de arquivos proposto. Na Tabela 1 é apresentado um resumo das características dos modelos de consenso *proof of work*, *proof of stake*, *proof of elapsed time*, *practical byzantine fault tolerance* e *federated byzantine agreement*. Os campos em cinza correspondem as cararacterísticas que atendem os requisitos da aplicação proposta.

O modelo do PoW apresenta uma baixa performance devido ao alto custo computacional para o cálculo do *hash* na criação de um bloco. Já o modelo PoET apresenta uma performance superior se comparado ao PoW, devido ao processo de eleição ser realizado de maneira imediata. Porém, esse ainda possui uma redução na performance devido ao fato dos nós terem que esperar pelo tempo do cronômetro recebido do *enclave*. Os algoritmos PoS, PBFT e FBA apresentam uma confirmação de transações mais rápida e, consequentemente, apresentam uma velocidade de transação alta (BALIGA, 2017). Neste sentido, os modelos que são mais adequados ao requisito de alta velocidade de transação são os modelos PoS, PBFT e FBA.

Nos modelos PoW, PoS e PoET a finalização das transações são probabilísticas uma vez que podem aparecer bifurcações na cadeia. No caso da existência de uma bifurcação, essa é resolvida a partir da negação de transações de um ramo da bifurcação. Assim, mesmo

Tabela 1: Comparação dos modelos de consenso

	PoW	PoS	PoET	PBFT	FBA
Velocidade das transações	Baixa	Alta	Média	Alta	Alta
Finalizações das transações	Probabilística	Probabilística	Probabilística	Imediata	Imediata
Escalabilidade da rede	Alta	Alta	Alta	Baixa	Alta
Tipo de *blockchain*	Pública	Pública ou Privada	Pública ou Privada	Privada	Pública
Custo para participar?	Sim	Sim	Não	Não	Não
Modelo de confiança	Sem confiança	Sem confiança	Sem confiança	Semi-confiança	Semi-confiança

Fonte: BALIGA (2017), adaptado

que uma transação tenha sido adicionada na *blockchain* ela não é final (BALIGA, 2017). Já nos modelos PBFT e FBA, não ocorrem bifurcações, deste modo, as transações não correm o risco de serem canceladas. Como o sistema proposto necessita de uma finalização imediata os únicos modelos adequados são o PBFT e FBA.

Escalabilidade da rede é a capacidade da rede de chegar a um consenso mesmo com o aumento no número de nós. Todos os algoritmos, exceto o PBFT, são considerados escaláveis. No caso do modelo PBFT é recomendado manter no máximo vinte nós na rede, devido a elevada quantidade de mensagens (BALIGA, 2017). Desta forma, o único modelo que não atende ao requisito de escalabilidade da rede é o PBFT.

Existem dois tipos de *blockchain*, as públicas e as privadas. A *blockchain* pública permite que qualquer usuário interaja com a rede. Já a *blockchain* privada é um ambiente fechado onde apenas alguns usuários podem participar. Como pode ser observado na Tabela 1, os modelos PoW e FBA são mecanismos de consenso utilizados em *blockchain* do tipo público. Essas podem ser usadas de forma privada, porém não são o modelo mais adequado para esse caso (BALIGA, 2017). Pelo sistema proposto neste trabalho não conter esse requisito, pode ser utilizado qualquer um dos modelos de consenso.

O PoW e PoS apresentam um custo para os nós participarem da rede. O custo do PoW é referente ao consumo de energia elétrica usada na mineração dos blocos. Já no PoS é necessário que o nó tenha um valor inicial de criptomoedas para realizar o depósito de confiança (BALIGA, 2017). Já os modelos PoET, PBFT e FBA não apresentam um custo para a participação na rede. Por não apresentar nenhum requisito sobre o custo de participação, todos os modelos são adequados quanto a essa característica.

O modelo de confiança depende se os nós que participam do consenso precisam ser

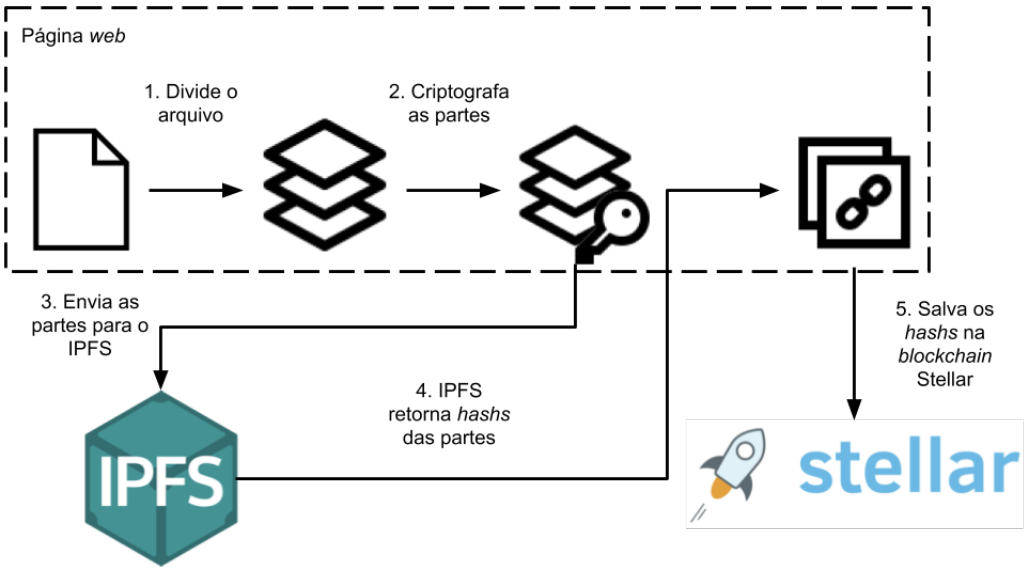
conhecidos ou confiáveis. Nos modelos PoW, PoS e PoET, os nós não precisam ser confiáveis uma vez que a confiabilidade é baseada em outros meios como, por exemplo, trabalho computacional ou depósitos seguros. Já no PBFT os nós necessitam estar registrados no sistema para participarem do consenso. Por fim, no FBA cada nó precisa garantir que incluiu nós de sua confiança na sua porção de *quorum* (BALIGA, 2017). Pelo sistema não apresentar um requisito pelo modelo de confiança, todos os modelos de consenso são adequados.

Analisando as características apresentadas e comparando com os requisitos do sistema, o modelo de consenso que cumpre os requisitos, é o modelo FBA. Deste modo, a plataforma de *blockchain* escolhida para o trabalho deve conter o mecanismo FBA como mecanismo de consenso.

3 PROPOSTA DE SOLUÇÃO

A solução desenvolvida neste trabalho é baseada na integração de ferramentas já existentes. Essa integração foi realizada utilizando a linguagem de programação *JavaScript* (NEGRINO; SMITH, 2012) através de uma página *web*. O modelo de funcionamento é similar aos sistemas *Sia* (VORICK; CHAMPINE, 2014) e *Storj* (WILKINSON; LOWRY; BOSHEVSKI, 2014). Nestes sistemas, os arquivos são divididos em partes menores que são criptografadas e armazenadas em nós que estão distribuídos com uma rede P2P. As partes possuem como identificador os seus respectivos *hashs*, os quais são salvos na *blockchain*. É através desses *hashs* que as partes dos arquivos são localizadas na rede. No processo de recuperação é realizado o processo inverso, onde primeiramente, os *hashs* das partes são buscados na *blockchain*, e são usados para localizar as partes do arquivo na rede. As partes recuperadas são descriptografadas e reunidas formando o arquivo original. Na Figura 11 tem-se um fluxograma do processo descrito.

Figura 11: Fluxo do sistema.



Fonte: o autor (2019)

3.1 DIVISÃO DO ARQUIVO E CRIPTOGRAFIA DAS PARTES

Inicialmente a divisão do arquivo havia sido definida através da utilização da biblioteca *split-file* (SPLIT-FILE, 2019), que é implementada na linguagem de programação *JavaScript*. A biblioteca permite a divisão do arquivo em partes, bem como a junção das partes gerando um novo arquivo. Porém, ela depende de acesso ao diretório de arquivos do computador e, por questão de segurança, os navegadores *web* não possuem essa permissão.

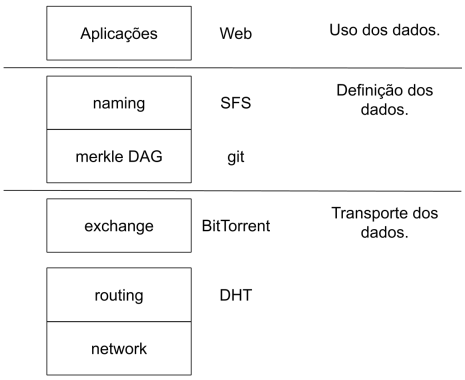
Para criptografia das partes foi utilizada a biblioteca *CryptoJS* (CRYPTOJS, 2013), que é uma biblioteca implementada na linguagem *JavaScript*. Com essa biblioteca é possível criptografar e descriptografar as partes do arquivo, utilizando diferentes algoritmos de criptografia como, por exemplo, MD5, SHA-1, SHA-256, AES, DES, etc.. O algoritmo que foi utilizado para a criptografar as partes é o AES, por se tratar de um algoritmo de chave simétrica, ou seja, a mesma chave usada para criptografar é usada para descriptografar as partes.

3.2 ARMAZENAMENTO DAS PARTES NA REDE

Para o armazenamento das partes do arquivo em uma rede distribuída foi utilizado o sistema de arquivos *InterPlanetary File System* (IPFS) (BENET, 2014). O IPFS é um sistema de arquivos implementado em uma arquitetura P2P. Esse tem como principal funcionalidade distribuir os arquivos na rede, retornando um *hash* de cada arquivo. Esse *hash* é utilizado para a localização dos arquivos na rede. O IPFS subistitui o atual modelo *location based addressing*, onde é utilizada a localização do conteúdo para recuperá-lo, pelo *content base addressing*, que permite a localização do conteúdo através do seu *hash* (KRSTONIć, 2018).

O IPFS é implementado em cinco camadas: *network*, *routing*, *exchange*, *Merkle* DAG e *naming* (Figura 12) (KRSTONIć, 2018). A camada de *network* gerencia as conexões com os outros *peers*, fornecendo recursos para a comunicação ponto a ponto entre dois nós da rede. Para isso a camada permite a utilização dos protocolos de rede como TCP (*Transmission Control Protocol*), SCTP (*Stream control transmission protocol*), SSH (*Secure Shell*), NAT (*Network Address Translation*) *Traversal*, entre outros (BENET, 2014).

Figura 12: Camadas IPFS



Fonte: KRSTONIć (2018), adaptado

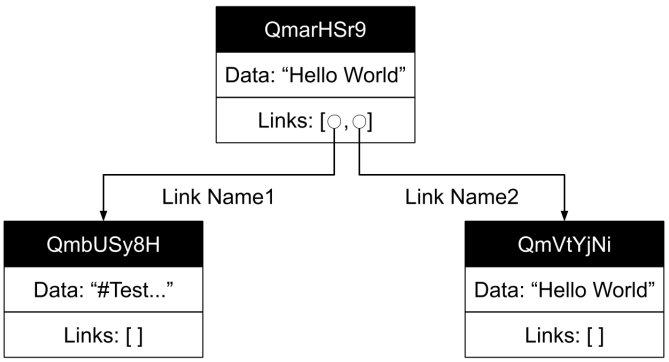
No IPFS, os nós utilizam um sistema de *routing* para encontrar o endereço dos

outros nós. A camada de *routing* utiliza uma *Distributed Hash Table* (DHT) (BENET, 2014) baseada em *S/Kademlia* (BAUMGART; MIES, 2007). A DHT armazena pares de chave e valor, onde a chave é o *hash* do conteúdo e o valor é o endereço do nó onde ele está armazenado.

A camada de *exchange* é responsável pela comunicação de blocos entre os nós, sendo que para essa comunicação é utilizado o protocolo *BitSwap* do *BitTorrent* (BENET, 2014). O *BitSwap* é um protocolo baseado em mensagens, onde o nó que deseja obter determinados blocos envia uma mensagem com a lista de blocos desejados para todos os outros nós da rede. Ao receber essa lista, os nós devem considerar o envio dos blocos, se eles os possuírem. Assim que receber os blocos desejados o nó solicitante envia uma mensagem informando que não deseja mais os blocos (KRSTONIć, 2018).

O conteúdo salvo na rede é armazenado em uma estrutura de dados chamada de *Merkle* DAG (*Directed Acyclic Graph*). Como pode ser observado na Figura 13, o conteúdo da rede é salvo em IPFS *objects*, que são estruturas que possuem dois campos: um campo que armazena os dados, de tamanho menor que 256 kB; e um *array* de IPFS *links*, que são referências para outros IPFS *objects*. Os IPFS *links*, por sua vez, são compostos por um nome, o *hash* do objeto vinculado e o tamanho do objeto vinculado. O tamanho do objeto é a soma do seu tamanho e dos objetos referenciados pelos seus *links* (LUNDKVIST; LILIC, 2016).

Figura 13: Exemplo de um IPFS *object*



Fonte: LUNDKVIST; LILIC (2016)

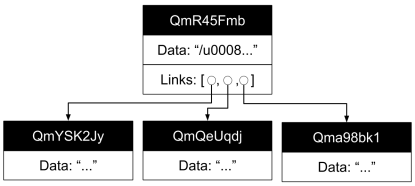
O IPFS também define um conjunto de objetos que permitem implementar um sistema de arquivos versionado utilizando IPFS *objects*. Esse conjunto é constituído por quatro tipos de objetos (*file objects*): *blob*, *list*, *tree* e *commit*. O tipo *blob* é utilizado para armazenar um arquivo de tamanho inferior a 256 kB. Esse é representado por um IPFS *object* que não contém *links* e apresenta o conteúdo do arquivo no campo *data* (BENET, 2014). Na Figura 14 é apresentado um exemplo de um arquivo texto armazenado em um *blob*.

Figura 14: Exemplo de um *blob*

Fonte: LUNDKVIST; LILIC (2016), adaptado

Já o tipo *list* é utilizado para armazenar arquivos maiores que 256 kB, que são separados em vários *blobs*. Um tipo *list* contém *links* que referenciam outros IPFS *objects*, que podem ser tanto *blobs* ou outras *list*, porém os *links* não possuem nome (Figura 15) (BENET, 2014).

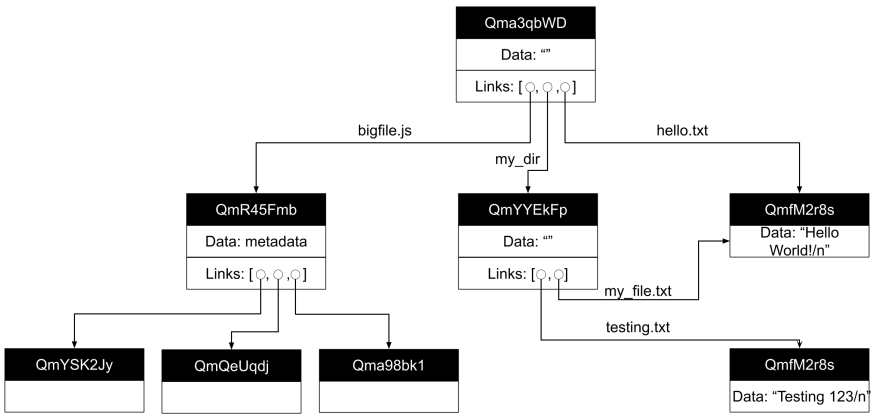
Figura 15: Exemplo de uma *list*



Fonte: LUNDKVIST; LILIC (2016), adaptado

O tipo *tree* representa um diretório, sendo similar a um *list* porém podendo referenciar outras *tree*. Além disso, ele possui nome nos *links* (BENET, 2014). A Figura 16 consiste em um exemplo de *tree*, nela é possível observar que os *links my_file.txt* e *hello.txt* estão apontando para o mesmo *blob*. Isso ocorre quando dois arquivos iguais foram adicionados na rede. O IPFS trata esse caso armazenando apenas um dos arquivos e direcionando os *links* para o seu *hash*, impedindo duplicação de dados. Como o sistema não apresenta *upload* de diretórios, apenas arquivos, o tipo *tree* não foi utilizado.

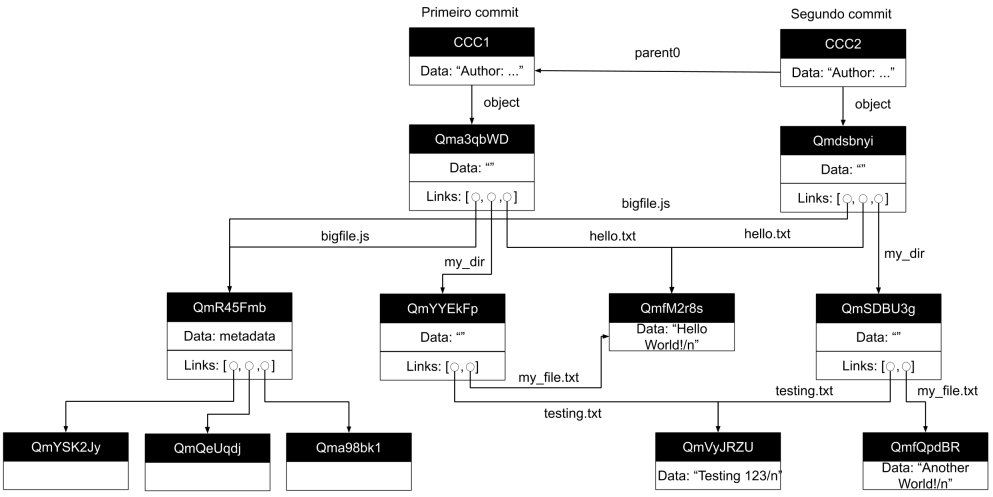
Figura 16: Exemplo de uma *tree*



Fonte: LUNDKVIST; LILIC (2016), adaptado

E, por último, o tipo *commit*, representa as versões de um objeto e pode referenciar qualquer tipo de objeto (BENET, 2014). Esse possui ainda o autor dos objetos (BENET, 2014). Os *commits* apresentam um *link* que aponta para um outro tipo de objeto que não seja um *commit*. Eles também podem possuir um ou mais *links* apontando para *commits* anteriores, criando assim o versionamento, onde o último *commit* é a versão atual do objeto e os *commits* anteriores são as versões antigas. Tendo como exemplo a *tree* da Figura 16, se o arquivo apontado pelo *link my_file.txt* for alterado, o *commit* obtido será o representado pela Figura 17 (LUNDKVIST; LILIC, 2016). Como o sistema proposto não apresenta requisito de versionamento o tipo *commit* não foi utilizado.

Figura 17: Exemplo de um *commit*



Fonte: LUNDKVIST; LILIC (2016), adaptado

Para a implementação do *upload* das partes do arquivo foi utilizada o SDK (*Software development kit*) do IPFS (IPFS. . . , 2019). Utilizando esse SDK são enviados dados contendo cada parte do arquivo, tendo como resposta o *hash* de cada parte. Esses *hashs* são armazenados na *blockchain*.

3.3 ARMAZENAMENTO DOS *HASHS* EM UMA *BLOCKCHAIN*

Para armazenar os *hashs* retornados pelo IPFS foi utilizada a plataforma de *blockchain Stellar* (STELLAR, 2014). Outras plataformas foram analisadas, como por exemplo, *Ethereum* e *Hyperledger Sawtooth*. Porém, a escolha da *Stellar* se deu devido ao fato dela implementar como mecanismo de consenso o modelo FBA, que foi o mecanismo de consenso escolhido para o desenvolvimento deste trabalho. Já a plataforma *Ethereum* implementa o modelo *proof of work* e a plataforma *Hyperledger Sawtooth* implementa o modelo *proof of elapsed time*.

O *Stellar* é uma plataforma de *blockchain*, que encontra-se disponível para a utilização da comunidade em geral, onde uma aplicação pode interagir com a rede da plataforma através de seu SDK (JS-STELLAR-SDK, 2019). A plataforma consiste em uma rede descentralizada de servidores, onde cada servidor contém uma cópia da Stellar *blockchain* que armazena as transações realizadas na rede (Figura 18) (STELLAR, 2014).

Figura 18: Representação da rede Stellar

Fonte: STELLAR (2014)

A *Stellar blockchain* é do tipo pública, ou seja, a rede é aberta a entrada de novos nós independente de quem seja. O modelo de consenso utilizado é o *Stellar Consensus Protocol* (MAZIERES, 2015), que é baseado no modelo FBA. O modelo *Stellar Consensus Protocol* têm uma velocidade de transação que varia entre três a cinco segundos e sua finalização de transação é imediata. Para cada transação realizada na rede é cobrada uma taxa (STELLAR, 2014).

Como a rede principal da *Stellar* apresenta taxas sobre as transações realizadas, optou-se pela utilização da rede de testes da *Stellar* chamada *testnet*. Para o desenvolvimento da implementação foi criada uma conta na rede de testes. Essa rede de testes possui todos os recursos da rede principal, porém tem um *reset* periódico dos dados da rede.

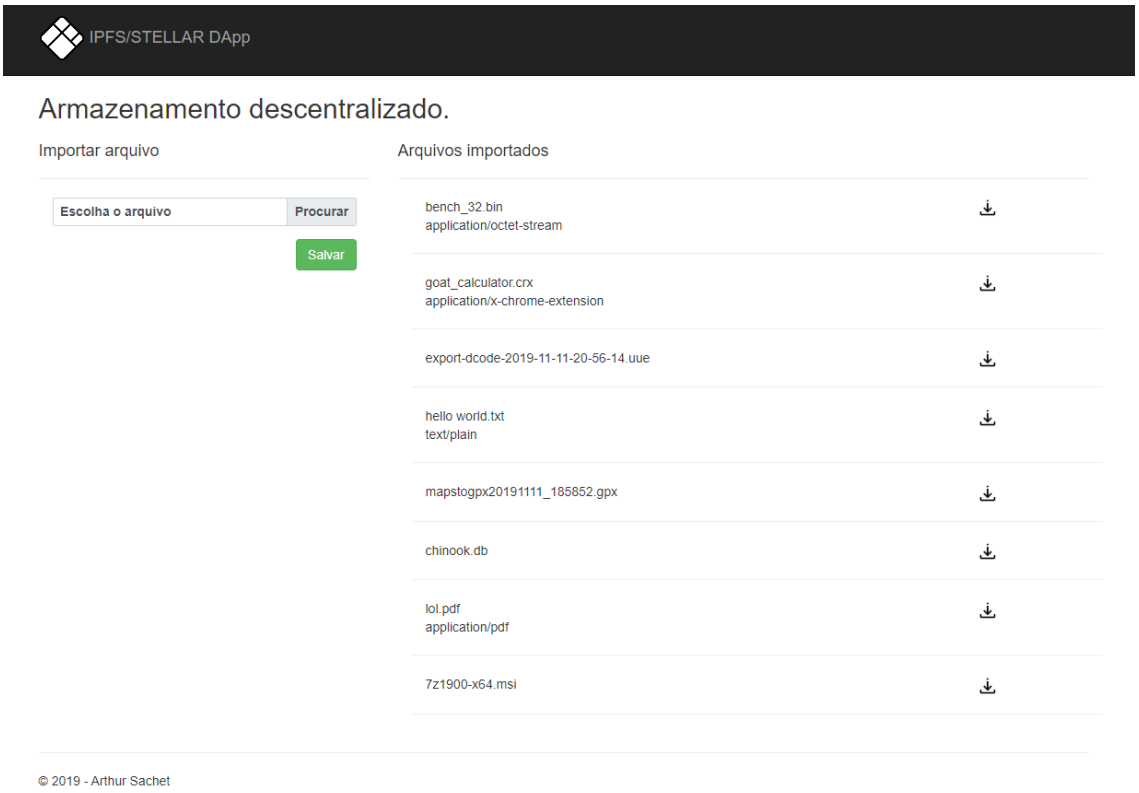
4 IMPLEMENTAÇÃO E TESTES REALIZADOS

Como já mencionado, as ferramentas utilizadas para a implementação foram o *CryptoJS*, IPFS, e a *Stellar*. A biblioteca de divisão de arquivos, *split-file*, não foi utilizada pois necessita de acesso ao diretório de arquivos do computador. Os navegadores *web*, por questão de segurança, não possuem permissão de acesso ao diretório de arquivos. Desta forma, foi implementada uma função própria para a divisão e reconstrução do arquivo. Neste capítulo será feita uma descrição das funcionalidades implementadas e os resutados dos testes realizados.

4.1 ENVIO DE UM ARQUIVO PARA A REDE

Na Figura 19 tem-se uma imagem da página *web* desenvolvida. A interface é dividida em duas partes, onde na primeira parte é possível fazer o *upload* dos arquivos. Já na segunda parte são listados os arquivos armazenados na rede, sendo possível realizar o *download* dos arquivos listados.

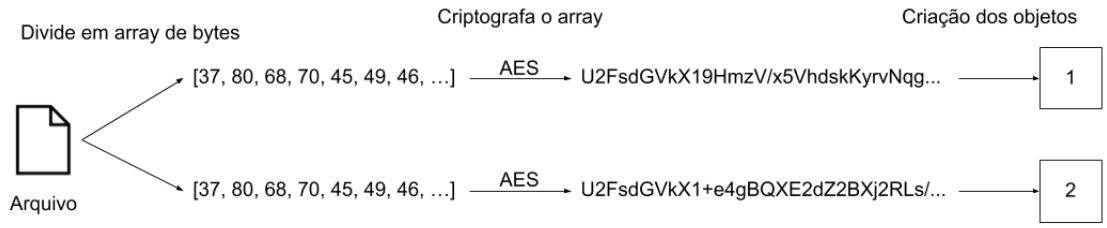
Figura 19: Interface desenvolvida para o gerenciamento dos arquivos.



Fonte: o autor (2019)

Ao inserir um arquivo no campo de *input* e pressionar o botão *Salvar*, o arquivo passa pelo fluxo apresentado na Figura 20. Primeiramente, o arquivo é armazenado em um *array* de *bytes* e então esse *array* é dividido em partes. Então para cada parte desse *array* é criado um objeto com: o nome e o formato do arquivo, o *array* de *bytes* criptografado, a posição do conteúdo e o MD5 (*Message-Digest algorithm 5*) do objeto da primeira parte.

Figura 20: Fluxograma da divisão e criptografia do arquivo.



Fonte: o autor (2019)

No Algoritmo 1 tem-se uma representação de um objeto criado, onde os campos de nome (*fileName*) e formato (*type*) são utilizados para que, ao realizar o *download* do arquivo, ele mantenha seu nome e formato original. Já o *array* de *bytes* (*content*) contém a parte do conteúdo do arquivo, esse *array* é criptografado utilizando a função AES da biblioteca *CryptoJS*. O campo da posição do conteúdo (*part*) é utilizado para a reconstrução do arquivo, sendo utilizado para a manutenção da ordem correta das partes. E por fim, é armazenado o MD5 do primeiro objeto (*firstHash*). Esse é obtido utilizando a função MD5 da biblioteca *CryptoJS* e é utilizado para identificar as partes do arquivo. Ou seja, esse *hash* é utilizado como o *hash* para a busca de todas as partes do arquivo na rede IPFS.

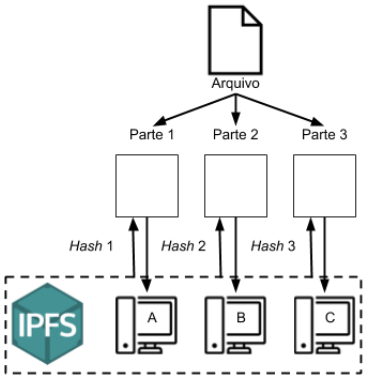
Algoritmo 1: Exemplo de objeto salvo

```
{
    content: "U2FsdGVkX1+35m7S9V3rA/avwruKmQDWE8M9s7y1AA3tNjOAFYNgeKYzJnap3SsI",
    fileName: "hello world.txt",
    firstHash: "428f09f6049b372bc37ec5505ddba7fc",
    part: 1,
    type: "text/plain"
}
```

Após, o conjunto de objetos de um arquivo é enviada para o sistema IPFS. Ao enviar os objetos para o IPFS através da função *add*, disponibilizada pelo SDK (IPFS..., 2019) do IPFS, tem-se como retorno os *hashs* para localização dos objetos na rede. Esses *hashs* são salvos na plataforma *Stellar*. Na Figura 21 tem-se a representação de um arquivo sendo incluído no IPFS.

Para o armazenamento do *hash* na plataforma *Stellar* é criada uma transação na *blockchain*, onde o conteúdo é o próprio *hash* do objeto no IPFS. Porém, a plataforma

Figura 21: Representação das partes de um arquivo incluído no IPFS



Fonte: o autor (2019)

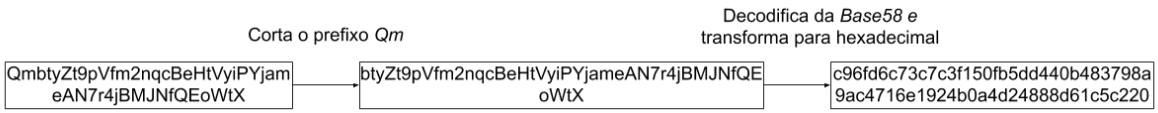
Stellar permite salvar apenas 3 tipos de formatação de texto nas suas transações. O primeiro tipo de formatação aceito é um texto comum de até 28 *bytes*, o segundo é um número inteiro de 64 bits sem sinal e o último é um *hash* de 32 *bytes* em base hexadecimal. O *hash* retornado pelo IPFS é codificado em *Base58*. No Algoritmo 2 tem-se um exemplo de um *hash* do objeto em *Base58*.

Algoritmo 2: Exemplo de *hash* do IPFS

```
{
    QmbtyZt9pVfm2nqcBeHtVyiPYjameAN7r4jBMJNfQEoWtX
}
```

Desta forma, para ser possível salvar o *hash* na *blockchain* se faz necessário uma transformação do *hash*. Como pode ser observado no Algoritmo 2 os dois primeiros caracteres são *Qm*. Esses são iguais em todos os *hashs* retornados pelo IPFS uma vez que correspondem ao algoritmo utilizado para obter o *hash*. Assim, esses dois primeiros caracteres foram removidos e os valores restantes convertidos para hexadecimal para que o *hash* possa ser salvo na transação da *Stellar*. Por exemplo, para o *hash* do Algoritmo 2 o valor convertido seria *c96fd6c73c7c3f150fb5dd440b483798a9ac4716e1924b0a4d24888d61c5c220*. Na Figura 22 tem-se o fluxo de transformação do *hash* do Algoritmo 2.

Figura 22: Fluxograma da transformação de um *hash*



Fonte: o autor (2019)

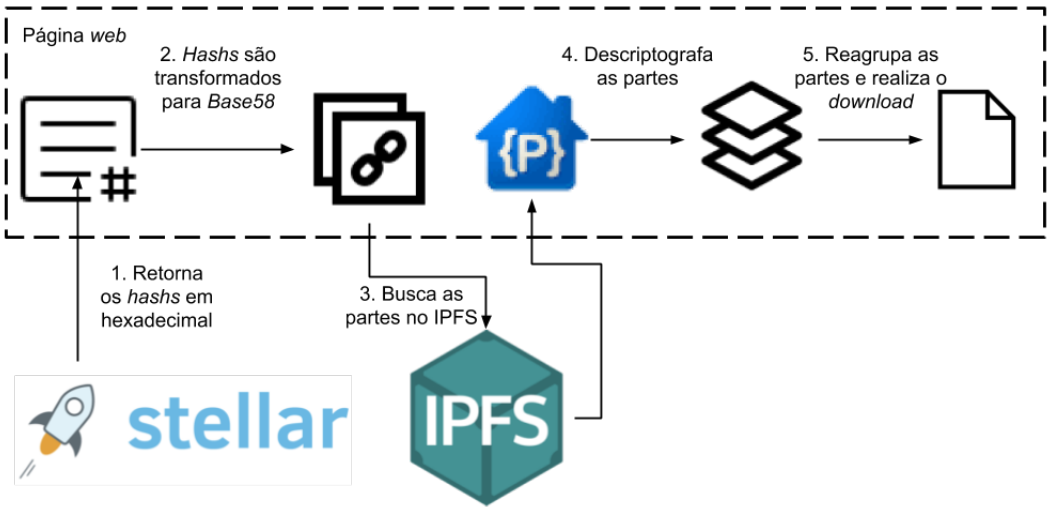
Após a conversão do *hash* é criada uma transação que é enviada para a *blockchain* de testes da *Stellar*. Para o envio foi utilizado a função *submitTransaction* disponibilizada

pelo SDK (JS-STELLAR-SDK, 2019) da *Stellar*. Ao receber o retorno de sucesso da função o processo de armazenamento do arquivo na rede é finalizado.

4.2 RECUPERAÇÃO DE UM ARQUIVO NA REDE

Para recuperar um arquivo é realizado o processo inverso (Figura 23). Através da função *transactions*, do SDK (JS-STELLAR-SDK, 2019) da *Stellar*, são recuperadas todas transações realizadas na *blockchain* de testes. Então para cada transação é recuperado o *hash* do seu conteúdo. O *hash* é convertido para decimal adicionando o prefixo *Qm* e então codificado para a *Base58*.

Figura 23: Fluxo de recuperação de um arquivo



Fonte: o autor (2019)

Com os *hashs* transformados, é realizada a chamada da função *cat* do SDK (IPFS. . . , 2019) do IPFS. Essa função recebe como parâmetro o *hash* e retorna o objeto salvo. O campo *content* do objeto é descriptografado pela função AES da biblioteca *CryptoJS*.

Ao pressionar para baixar o arquivo na interface, os objetos com o mesmo MD5 são ordenados pelo campo posição e os *arrays* de *bytes* são unidos formando o arquivo original. Após, é criado um *link* através da função *URL.createObjectURL* para que o *browser* faça o *download* do arquivo.

4.3 VALIDAÇÃO DA IMPLEMENTAÇÃO

Para a validação da implementação foi realizado o *upload* de arquivos de diferentes tipos. Para esse teste foram utilizados os arquivos mais comuns que são listados em: *<https://fileinfo.com/filetypes/common>*. Na Tabela 2 tem-se a lista dos arquivos utilizados

para o teste com seu respectivo tamanho em kB. Em todos os testes realizados o arquivo recuperado manteve-se idêntico ao arquivo original.

Tabela 2: Lista de tipos de arquivos testados

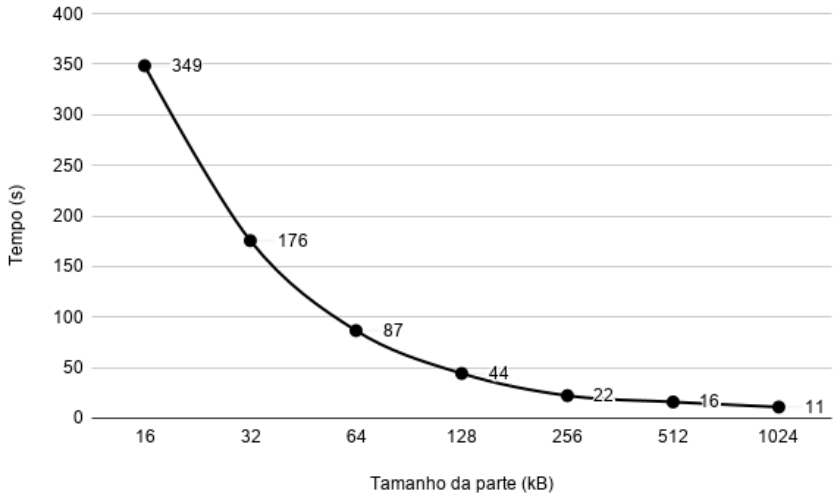
Tipo de arquivo	Tamanho (kilobytes)	Tipo de arquivo	Tamanho (kilobytes)
Plain Text File (txt)	0,011	Drawing Exchange Format File (dxf)	265
PowerPoint Open XML Presentation (pptx)	632	GPS Exchange File (gpx)	4
MP3 Audio File (mp3)	3660	JavaScript File (js)	8,62
MPEG-4 Video File (mp4)	2280	Chrome Extension (crx)	11
Wavefront 3D Object File (obj)	158	Generic Font File (fon)	34,9
Portable Network Graphic (png)	23	Dynamic Link Library (dll)	87
Scalable Vector Graphics File (svg)	13	Configuration File (cfg)	0,58
Portable Document Format File (pdf)	315	Uuencoded File (uue)	0,063
Microsoft Excel Open XML Spreadsheet (xlsx)	9,52	7-Zip Compressed File (7z)	0,153
Database File (db)	864	Binary Disc Image (bin)	4
Windows Executable File (exe)	48,8	C# Source Code File (cs)	0,977
Saved Game (sav)	3,02	Temporary File (tmp)	27,4
Windows Installer Package (msi)	2000		

4.3.1 Definição do tamanho da parte

Para a definição do tamanho mais adequado para o particionamento do arquivo na rede IPFS foram realizados testes utilizando diferentes tamanhos de partes. No teste foi utilizado um arquivo de 1 Mb. O tamanho das partes utilizadas nos testes foram definidas em potência de 2 começando por 16 *kBytes* até 1 *MByte*. Para cada tamanho de parte foram realizadas 5 medições de tempo e a partir delas feita uma média. Na Figura 24 é apresentado o gráfico de Tempo (s) x Tamanho da parte kB.

Observa-se no gráfico da Figura 24 que partes menores possuem um desempenho inferior. Isso deve-se ao fato de gerarem um número maior de partes e por consequência uma quantidade mais elevada de requisições de *upload*. Por exemplo, com o tamanho de 16 *kBytes* o arquivo é dividido em 64 partes, ocasionando essa mesma quantia de requisições de *upload*, gerando um tempo médio de aproximadamente 6 minutos. Porém, como pode ser observado no gráfico o desempenho torna-se praticamente constante a partir de partes com tamanho superior a 256 kB. Desta forma, optou-se pela utilização deste tamanho de parte, uma vez que possibilita a divisão do arquivo em uma quantidade maior de partes sem uma perda significativa de performance.

Figura 24: Gráfico da relação entre tamanho da parte e o tempo de *upload*.

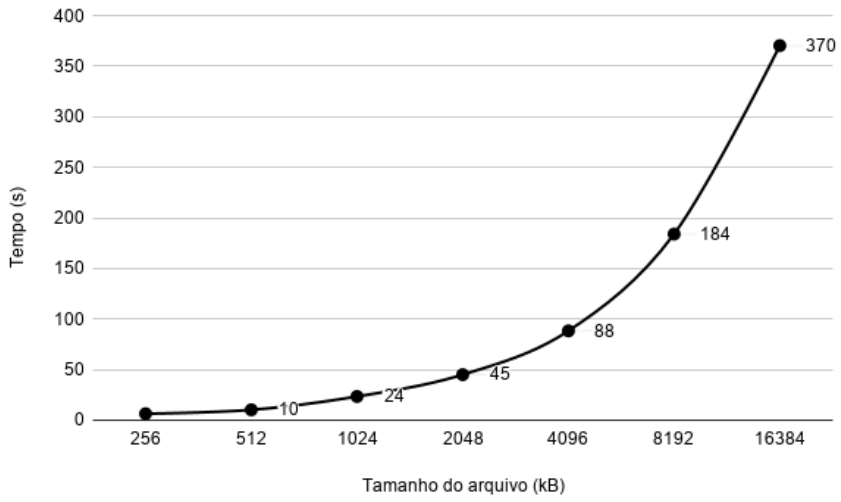


Fonte: o autor (2019)

4.3.2 Teste de performance

Posteriormente, foram realizados testes de performance considerando o *upload* de arquivos de tamanhos diferentes. Os tamanhos de arquivo foram definidos com potência de 2 utilizando arquivos de 256 kB, até 16 MB. Para cada tamanho de arquivo foram realizadas 5 medições de tempo de *upload* e feita uma média dos resultados. Na Figura 25 é apresentado o gráfico de Tempo (s) x Tamanho do arquivo (kB) com os resultados para o *upload* na rede. Como pode ser observado o tempo de *upload* aumenta em função do tamanho do arquivo, devido principalmente ao maior número de partes geradas e consequentemente um maior número de *uploads*. Por exemplo, o arquivo com tamanho de 16 MB é dividido pelo tamanho da parte (256 kB) gerando 64 partes. E assim como no teste do tamanho da parte, ocasiona 64 requisições de *upload* gerando um tempo médio de aproximadamente 6 minutos.

Figura 25: Gráfico da relação entre tamanho do arquivo e o tempo de *upload*.



Fonte: o autor (2019)

5 *CONSIDERAÇÕES FINAIS*

Neste trabalho foi apresentado o uso da tecnologia de *blockchain* com ênfase no armazenamento de arquivos descentralizados em um rede P2P. Na solução desenvolvida os arquivos são divididos e criptografados, para então serem armazenados na rede e seus *hashs* salvos em uma *blockchain*.

Inicialmente, para a divisão e reconstrução do arquivo seria utilizada a biblioteca *split-file*. Porém, devido a sua dependência de acesso ao diretório de arquivos não ser suportada pelos navegadores *web* ela não foi utilizada. Assim, para a divisão e reconstrução dos arquivos foi criada uma função própria.

Para criptografar as partes dos arquivos foi utilizada a biblioteca *CryptoJS*. A escolha dela decorreu devido ela ser desenvolvida na linguagem de programação *JavaScript*. A criptografia é realizada através do algoritmo AES da *CryptoJS*. O algoritmo AES foi utilizado por se tratar de um algoritmo de chave simétrica, ou seja, a mesma chave usada para criptografar é usada para descriptografar as partes.

Para a implementação da solução foi definido o IPFS como sistema de arquivos distribuídos. Ele foi escolhido pois atende ao requisito de armazenamento de arquivos em uma rede P2P, sendo que esses podem ser recuperados utilizando o seu *hash*. Ele implementa tipos de objetos, sendo eles, *blob*, *list*, *tree* e *commit*. Desses objetos foram utilizados o *blob* e o *list* para armazenar as partes dos arquivos.

Em relação a plataforma de *blockchain*, a escolha se deu pela *Stellar*. A plataforma *Stellar* foi escolhida por implementar modelo de consenso FBA, requisito do sistema. Essa plataforma possui uma versão para testes que foi utilizada na implementação do sistema proposto. A versão de testes possui todos os recursos da rede principal, porém tem um *reset* periódico dos dados da rede.

Após a implementação do sistema foram realizados alguns testes. O primeiro teste é o de validação do sistema. Ele foi realizado com o intuito de verificar se o sistema suporta os tipos mais comuns de arquivos. No teste foi realizado o *upload* e o *download* de arquivos de tipos diferentes. Em todos os testes realizados o arquivo recuperado manteve-se idêntico ao arquivo original

O segundo teste realizado foi o de definição do tamanho da parte. Ele foi utilizado para a definição do tamanho mais adequado para o particionamento do arquivo. Para isso, foi realizado o *upload* de um arquivo com diferentes tamanhos de partes. Após analisar o tempo de *upload* ficou definido o tamanho como 256 kB para as partes. A escolha se deu para que o arquivo possa ser dividido sem uma perda significativa de performance.

E por fim, o último teste foi o de desempenho do sistema. No teste foram realizados o *upload* de arquivos de tamanhos diferentes para analisar a performance do sistema. Nele observou-se que o tempo de *upload* aumenta em função do tamanho do arquivo, devido principalmente ao maior número de partes geradas.

A implementação deste trabalho foi baseada em sistemas de armazenamento de arquivos utilizando *blockchain* já existentes como: o *Sia* (VORICK; CHAMPINE, 2014), *Storj* (WILKINSON; LOWRY; BOSHEVSKI, 2014) e *Filecoin* (BENET; GRECO, 2017). Porém, a diferença da implementação desenvolvida é a divisão e a criptografia das partes do arquivo. Esse processo aumenta a segurança no acesso dos arquivos pois, mesmo estando em uma rede pública, para se ter acesso aos dados do arquivo é necessário obter todas as suas partes bem como conhecer a chave que foi utilizada para criptografá-las.

5.1 SUGESTÕES DE TRABALHOS FUTUROS

Como trabalhos futuros sugere-se:

- Criação de um *cluster* para o sistema IPFS ao invés de utilizar sua rede pública.
- Implementação de uma *blockchain* própria para não ser necessário utilizar a rede de testes da *Stellar*.
- Permitir que o usuário defina em quantas partes o arquivo será separado.
- Melhoria do desempenho do sistema, para que arquivos de grande tamanho possam ser incluídos na rede em uma menor quantidade de tempo.

REFERÊNCIAS

BACH, L.; MIHALJEVIC, B.; ZAGAR, M. Comparative analysis of blockchain consensus algorithms. In: IEEE. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. [S.l.], 2018. p. 1545–1550.

BALIGA, A. Understanding blockchain consensus models. In: *Persistent*. [S.l.: s.n.], 2017.

BASHIR, I. *Mastering blockchain*. [S.l.]: Packt Publishing Ltd, 2017.

BAUMGART, I.; MIES, S. S/kademlia: A practicable approach towards secure key-based routing. In: IEEE. *2007 International Conference on Parallel and Distributed Systems*. [S.l.], 2007. p. 1–8.

BENET, J. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

BENET, J.; GRECO, N. Filecoin: A decentralized storage network. *Protoc. Labs*, 2017.

CASTRO, M.; LISKOV, B. et al. Practical byzantine fault tolerance. In: *OSDI*. [S.l.: s.n.], 1999. v. 99, p. 173–186.

CHRISTIDIS, K.; DEVETSIKIOTIS, M. Blockchains and smart contracts for the internet of things. *Ieee Access*, Ieee, v. 4, p. 2292–2303, 2016.

CRYPTOJS. 2013. Disponível em: <https://code.google.com/archive/p/crypto-js/>.

FISCHER, M. J.; LYNCH, N. A.; PATERSON, M. S. *Impossibility of distributed consensus with one faulty process*. [S.l.], 1982.

FOUNDATION, S. D. *On Worldwide Consensus The future is distributed: a summary of the Stellar Consensus Protocol white paper*. 2015. Disponível em: <https://medium.com/stellar-development-foundation/on-worldwide-consensus-359e9eb3e949>.

GILBERT, H.; HANDSCHUH, H. Security analysis of sha-256 and sisters. In: SPRINGER. *International workshop on selected areas in cryptography*. [S.l.], 2003. p. 175–193.

GLICKSTEIN, B. *Understanding the Stellar Consensus Protocol*. 2019. Disponível em: <https://medium.com/interstellar/understanding-the-stellar-consensus-protocol-423409aad32e>.

IPFS HTTP Client Library. 2019. Disponível em: <https://github.com/ipfs/js-ipfs-http-client>.

JS-STELLAR-SDK. 2019. Disponível em: <https://stellar.github.io/js-stellar-sdk/index.html>.

KING, S.; NADAL, S. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, v. 19, 2012.

KRSTONIć, N. *A Closer Look at the InterPlanetary File System (IPFS)*. 2018. Disponível em: <https://medium.com/mvp-workshop/a-closer-look-to-the-inter-planetary-file-system-b3f3af31a3c7>.

LAMPORT, L.; SHOSTAK, R.; PEASE, M. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, ACM, v. 4, n. 3, p. 382–401, 1982.

LUNDKVIST, C.; LILIC, J. *An Introduction to IPFS*. 2016. Disponível em: <https://medium.com/@ConsenSys/an-introduction-to-ipfs-9bba4860abd0>.

MACDONALD, M.; LIU-THORROLD, L.; JULIEN, R. The blockchain: a comparison of platforms and their uses beyond bitcoin. *COMS4507-Adv. Computer and Network Security*, 2017.

MAZIERES, D. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation*, Citeseer, 2015.

MERKLE, R. C. A fast software one-way hash function. *Journal of Cryptology*, Springer, v. 3, n. 1, p. 43–58, 1990.

MINGXIAO, D. et al. A review on consensus algorithm of blockchain. In: IEEE. *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. [S.l.], 2017. p. 2567–2572.

NAIK, R. P.; COURTOIS, N. T. *Optimising the SHA256 Hashing Algorithm for Faster and More Efficient Bitcoin Mining*. [S.l.]: Department of Computer Science, University College London, 2013.

NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system. Working Paper, 2008.

NAOR, M.; YUNG, M. Universal one-way hash functions and their cryptographic applications. In: ACM. *Proceedings of the twenty-first annual ACM symposium on Theory of computing*. [S.l.], 1989. p. 33–43.

NEGRINO, T.; SMITH, D. *JavaScript for the World Wide Web*. [S.l.]: Peachpit Press, 2012.

OLIVEIRA, M. T. et al. Uma avaliação de desempenho de cadeias de blocos privadas permissionadas através de cargas de trabalho realísticas*. In: SBC. *SBSeg 2018*. [S.l.], 2018. p. 309–322.

PECK, M. E. Blockchains: How they work and why they'll change the world. *IEEE spectrum*, IEEE, v. 54, n. 10, p. 26–35, 2017.

SCHOLLMEIER, R. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In: IEEE. *Proceedings First International Conference on Peer-to-Peer Computing*. [S.l.], 2001. p. 101–102.

SINGH, S.; SINGH, N. Blockchain: Future of financial and cyber security. In: IEEE. *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*. [S.l.], 2016. p. 463–467.

SPLIT-FILE. 2019. Disponível em: <https://www.npmjs.com/package/split-file>.

STELLAR. 2014. Disponível em: <https://www.stellar.org>.

VORICK, D.; CHAMPINE, L. Sia: Simple decentralized storage. *White paper available at https://sia.tech/sia.pdf*, 2014.

WILKINSON, S. et al. *Storj: A peer-to-peer cloud storage network v2. 0*. [S.l.]: Citeseer Press, 2016.

WILKINSON, S.; LOWRY, J.; BOSHEVSKI, T. Metadisk a blockchain-based decentralized file storage application. *Storj Labs Inc., Technical Report, hal*, p. 1–11, 2014.

XING, B. C.; SHANAHAN, M.; LESLIE-HURD, R. Intel® software guard extensions (intel® sgx) software support for dynamic memory allocation inside an enclave. In: ACM. *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. [S.l.], 2016. p. 11.

ZYSKIND, G.; NATHAN, O. et al. Decentralizing privacy: Using blockchain to protect personal data. In: IEEE. *2015 IEEE Security and Privacy Workshops*. [S.l.], 2015. p. 180–184.