INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DO RIO GRANDE DO SUL
CAMPUS RESTINGA
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ARMAZENAMENTO SEGURO DE ARQUIVOS ATRAVÉS DA COMBINAÇÃO DE CRIPTOGRAFIA ASSIMÉTRICA, SMART CONTRACTS E IPFS

GIORDANO GOLLIN BUFFON

PORTO ALEGRE
2018

GIORDANO GOLLIN BUFFON

ARMAZENAMENTO SEGURO DE ARQUIVOS ATRAVÉS DA COMBINAÇÃO DE CRIPTOGRAFIA ASSIMÉTRICA, SMART CONTRACTS E IPFS

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul - IFRS, Campus Restinga.

Orientador: Prof. Me. Régio Antônio Michelin

Porto Alegre

2018

GIORDANO GOLLIN BUFFON

ARMAZENAMENTO SEGURO DE ARQUIVOS ATRAVÉS DA COMBINAÇÃO DE CRIPTOGRAFIA ASSIMÉTRICA, SMART CONTRACTS E IPFS

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Sul - IFRS, Campus Restinga.

Data de Aprovação: 17/12/2018

Banca Examinadora

_____

Prof. Me. Régio Antônio Michelin - IFRS - Campus Restinga
Orientador

_____

Prof. Me. Roben Castagna Lunardi- IFRS- Campus Restinga
Membro da Banca

_____

Prof. Me. Jezer Machado de Oliveira- IFRS- Campus Restinga
Membro da Banca

Se você pensa que pode ou se pensa que não pode, de qualquer forma você está certo.

Henry Ford

RESUMO

Praticidade e segurança no armazenamento e compartilhamento de arquivos são necessidades cada vez maiores no dia-a-dia das pessoas e das organizações. As soluções de armazenamento de arquivos em nuvem possibilitam o armazenamento e compartilhamento de arquivos com um alto grau de eficiência, porém a segurança dessas informações fica dependente do provedor de nuvem e não há formas práticas para a realização de auditorias sobre a integridade dos dados. Com o intuito de suprir esta lacuna, este trabalho propõe um sistema que emprega as tecnologias de criptografia assimétrica, smart contracts e IPFS para armazenar e compartilhar arquivos garantindo a data de armazenamento, a segurança e a integridade dos dados. Os arquivos encriptados são armazenados no IPFS, um sistema de arquivos distribuídos peer-to-peer, e as hash resultantes são enviadas para a Ethereum Blockchain através dos smart contracts. Neste trabalho são apresentadas a solução proposta e avaliações sobre sua aplicação prática.

Palavras-chave: blockchain, smart contracts, ethereum, peer-to-peer, ipfs.

ABSTRACT

Practical and secure storage and file sharing are increasingly needed by people and organizations on a daily basis. Cloud storage solutions enable high-efficiency file storage and sharing, but the security of this information is dependent on the cloud provider, and there are no practical ways to perform audits of data integrity. In order to fill this gap, this paper proposes a system that uses asymmetric cryptography, smart contracts and IPFS technologies to store and share files, ensuring data storage, security and data integrity. Encrypted files are stored in IPFS, a peer-to-peer distributed file system, and the resulting hashes are sent to Ethereum Blockchain through smart contracts. In this paper the proposed solution and evaluations about its practical application are presented.

Keywords: blockchain, smart contracts, ethereum, peer-to-peer, ipfs.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A garantia de disponibilidade e a velocidade acesso à dados é de extrema importância em um mundo de grande avanço tecnológico. O número de pessoas conectadas à internet cresce a cada dia e, na mesma proporção, o tráfego de informações na rede. Em um modelo centralizado de acesso (cliente-servidor), um servidor responde a requisições de clientes de diversos pontos, muitas vezes do outro lado do mundo. Essa arquitetura torna o sistema suscetível a ataques e falhas no ponto de origem (VIEIRA, 2010). Com isso, rotas e servidores acabam ficando sobrecarregados e, muitas vezes, indisponíveis.

De outro lado, há a necessidade de segurança dos dados armazenados remotamente. Ao enviar e acessar dados sensíveis na internet, o usuário espera que esses dados sejam armazenados de forma segura, ficando inacessíveis à outros usuários que não tenham as devidas permissões.

Em bancos de dados tradicionais, o controle de acesso é feito por uma autoridade designada, que é responsável por autenticar e prover acesso de um cliente ao banco de dados. Uma vez que a segurança dessa autoridade administradora estiver em risco, todos os dados também estarão, podendo ser alterados ou excluídos (RAY, 2017).

Em ascensão desde o surgimento do Bitcoin, a tecnologia de blockchain surgiu como alternativa ao modelo centralizador de armazenamento de dados e de administração dessas informações. A principal característica da tecnologia da blockchain é a garantia da integridade e transparência dos dados. Cada usuário pode ter a certeza de que os dados estão incorrompidos e inalterados desde o momento em que foram registrados na base, bem como podem verificar todo o histórico de operações ao longo do tempo (RAY, 2017).

Tendo em vista as limitações de segurança nos modelos tradicionais de armazenamento de dados e a proposta que a tecnologia de armazenamento distribuído aliado à blockchain oferece, o presente trabalho propõe uma solução que emprega as tecnologias de criptografia assimétrica, smart contracts e IPFS para armazenar e compartilhar arquivos garantindo a data de armazenamento, a segurança e a integridade dos dados. Os arquivos encriptados são armazenados no IPFS, um sistema de arquivos distribuídos peer-to-peer, e as hashes resultantes são enviadas para a Ethereum Blockchain através dos smart contracts.

2 TRABALHOS RELACIONADOS

As pesquisas sobre as aplicações da blockchain tem se desenvolvido constantemente no meio científico nos últimos anos, devido às características dessa tecnologia e aos benefícios que ela pode proporcionar nas mais diversas áreas de atuação. Desta forma, já foram propostos sistemas para a área da saúde (COSTA et al., 2018), eleições (MCCORRY et al., 2017), IoT (CHRISTIDIS; DEVETSIKIOTIS, 2016), educação (GRÄTHER et al., 2018), entre outros. A Tabela 1 faz um comparativo de soluções existentes com o sistema proposto e, na sequência, são apresentadas estas soluções.

Tabela 1 – Comparação de soluções existentes com o sistema proposto.

Sistema	**Público alvo**	**Método de proteção dos dados**	**Permite armazenamento de dados dos usuários**	**Utiliza autoridade certificadora**
Sistema proposto neste trabalho	Qualquer usuário	Criptografia assimétrica	Sim	Não
DSHR	Setor de saúde	Criptografia baseada em atributos	Sim	Sim
Blockchains and Smart Contracts for the Internet of Things	Setor de Internet das Coisas (IoT)	Criptografia homomórfica	Não se aplica	Não
Open Vote Network	Eleições	Criptografia de curva elíptica	Não se aplica	Não
Blockchain for Education: Lifelong Learning Passport	Setor de educação	Dados abertos	Sim	Sim

Assim como este trabalho, diversas soluções propostas, que também têm a necessidade de armazenar uma grande quantidade de dados, fazem o uso do IPFS em conjunto com a blockchain. Isso porque o armazenamento de grandes quantidades de dados em blockchains é inviável devido aos altos custos de processamento, energia e, consequentemente, de criptomoedas para executar essa tarefa. Com a cotação atual do ETH (moeda da Ethereum) - aproximadamente $85.75 USD - o custo aproximado por kB armazenado é de $0.27 USD, o que equivale a R$ 1,06. Com o emprego do IPFS, é possível potencializar o uso da tecnologia da blockchain com um baixo custo.

O DSHR (COSTA et al., 2018) é um protocolo para o compartilhamento seguro de ar-

quivos de saúde, que tem como principal contribuição a garantia da integridade dos arquivos, que envolve alteração e a deleção indevida destes. Para garantir a segurança e a escalabilidade, foram empregados criptografia baseada em atributos (CBA), redes IPFS e blockchain. As principais diferenças para o sistema proposto é que o método de criptografia utilizado pelo DSHR exige a participação de autoridades certificadoras online e de atributos. No que se refere ao desempenho do armazenamento e compartilhamento de dados, há uma proximidade entre as soluções.

McCorry, Shahandashti e Hao (MCCORRY et al., 2017) propuseram o Open Vote Network: a implementação de um smart contract executado na Ethereum blockchain, que visa garantir a privacidade total do voto em eleições, sem a necessidade de qualquer autoridade confiável controladora. O Open Vote Network não objetiva o armazenamento de arquivos, apenas registra votos com o uso de criptografia de curva elíptica e contabiliza o resultado de votações, não empregando o IPFS na solução, diferentemente do sistema proposto. Entretanto, o uso de smart contracts autônomos sem a necessidade de uma autoridade controladora confiável é uma característica semelhante à proposta deste trabalho.

Christidis e Devetsikiotis (CHRISTIDIS; DEVETSIKIOTIS, 2016) analisaram como o uso de smart contracts e blockchain podem facilitar o compartilhamento de serviços e recursos entre dispositivos no setor de Internet das Coisas (IoT), além de permitir a automação verificável criptograficamente de fluxos de trabalho existentes. Também destacaram alguns problemas que o uso da blockchain pode trazer para o setor, desde a privacidade transacional até o valor esperado dos ativos digitalizados negociados na rede. Diferentemente do sistema proposto, o setor de IoT não objetiva armazenar arquivos, mas sim registrar ações e informações que permitam o rastreamento de um determinado item em uma cadeia. Como os dispositivos de IoT normalmente dispõe de pouco recurso computacional, a privacidade de informações pode ser comprometida, pois técnicas de criptografia tradicionais demandam cálculos intensivos. Como alternativa, os autores sugerem o uso da criptografia homomórfica e técnicas de intercalo de uso entre blockchains distintas.

Blockchain for Education (GRÄTHER et al., 2018) é uma plataforma que propõe o uso de blockchain para o armazenamento de certificados educacionais, de modo a garantir sua inviolabilidade e disponibilidade a longo prazo. A plataforma pode ser usada por certificadores, alunos e terceiros, como empregadores. Com a participação de autoridades certificadoras, os usuários podem atestar a autenticidade de um determinado certificado, ficando esses documentos disponíveis no sistema sem nenhum tipo de criptografia. Diferentemente do sistema

proposto, Blockchain for Education não tem o objetivo de manter os arquivos privados, pois estes contém informações públicas sobre os usuários e devem ser acessados por terceiros para cumprir o seu propósito.

3 REFERENCIAL TEÓRICO

3.1 BLOCKCHAIN

A blockchain é uma estrutura de dados distribuída que é replicada e compartilhada entre os membros de uma rede (CHRISTIDIS; DEVETSIKIOTIS, 2016). É composta por uma lista crescente de registros, chamados blocos, que são vinculados e protegidos por criptografia. Cada um desses blocos contém uma lista de transações, um registro de data e hora de assinatura e um hash que aponta para o bloco anterior (PRATHYUSHA et al., 2018), formando uma cadeia.

Para ser adicionado na cadeia, um bloco precisa ser validado mutuamente pelos nodos vizinhos (os chamados mineiros) na rede, passando por um mecanismo de consenso. Uma vez validado, ele é assinado, adicionado na cadeia e retransmitido para a rede; caso seja invalidado, é descartado (CHRISTIDIS; DEVETSIKIOTIS, 2016). Ao ser adicionado na cadeia, esse bloco torna-se imutável, não permitindo edições ou deleção de qualquer informação contida nele. Além disso, toda essa cadeia de informações é pública, o que garante transparência e confiabilidade, pois pode ser auditada a qualquer momento.

3.2 ETHEREUM

Ethereum é uma plataforma open source que foi especificamente construída para executar smart contracts utilizando a tecnologia blockchain (CHRISTIDIS; DEVETSIKIOTIS, 2016). Tem por objetivo fornecer ao desenvolvedor final um sistema integrado para construção de softwares em um paradigma de computação até então inexplorado, capaz de garantir que um acordo será executado exatamente como pretendido entre indivíduos consentidos (WOOD, 2014).

Na Ethereum, todas as transações de Ether ou de Tokens, assim como as operações computacionais realizadas por smart contracts tem um custo de *gas*, uma unidade de medição do trabalho computacional dos mineiros (WOOD, 2014). O *gas* por sua vez normalmente é pago com *Gwei*, uma fração do Ether (ETH) - a criptomoeda da Ethereum - que tem um valor monetário real.

3.3 SMART CONTRACTS

Os smart contracts (ou contratos inteligentes) são protocolos de transação informatizados que executam os termos de um contrato, como condições de pagamento, penhor, confidencialidade e cumprimento, minimizando exceções maliciosas ou acidentais e a necessidade de intermediá-

rios confiáveis (SZABO, 1994).

No contexto da Ethereum, os smart contracts são escritos como códigos de programação utilizando-se a linguagem Solidity (que é baseada em C++, Python e Javascript) e, uma vez escritos e compilados, são imutáveis. Eles operam como atores autônomos, cujo comportamento é completamente previsível. Como tal, eles podem ser confiáveis para impulsionar qualquer lógica que possa ser expressa como uma função de entradas de dados na cadeia, desde que os dados que eles precisam gerenciar estejam dentro do seu próprio alcance (CHRISTIDIS; DEVETSIKIOTIS, 2016).

3.4 IPFS

O IPFS (*InterPlanetary File System*) é um sistema de arquivos distribuídos *peer-to-peer*, que se propõe como alternativa ao paradigma centralizado cliente-servidor do modelo HTTP (BENET, 2014). No IPFS, cada navegador pode ser um nó na rede, que insere e armazena dados. Ao tentar acessar um arquivo que não está armazenado localmente, o *peer* faz a busca em seus "vizinhos" mais próximos sem, necessariamente, ter que requisitar à um *data center* muito distante. Assim, uma rede de websites completamente descentralizados tem o potencial de acelerar a transferência de arquivos e os tempos de streaming (PRATHYUSHA et al., 2018).

No IPFS é possível ler e adicionar arquivos, que são referenciados na rede por uma *hash* única calculada a partir de seu conteúdo. Essa característica garante a imutabilidade do arquivo em relação à uma determinada *hash*, visto que a mudança de um bit do seu conteúdo gera uma *hash* diferente da original, e evita a duplicidade, pois arquivos idênticos são referenciados por uma mesma *hash*.

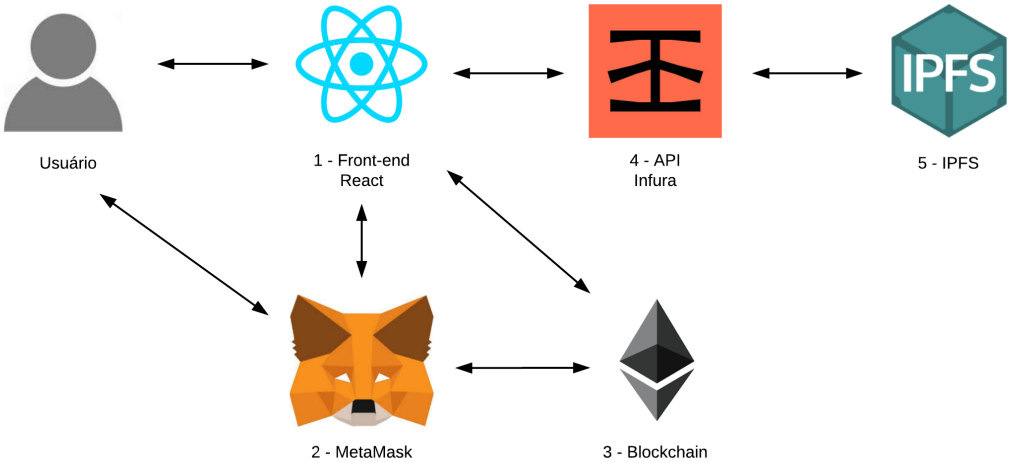
4 MODELO CONCEITUAL

Buscando garantir a disponibilidade, a segurança e a autenticidade e permitir o compartilhamento dos dados armazenados por um usuário, a proposta do sistema utiliza a combinação entre o IPFS, onde os dados são armazenados, e a blockchain através dos smart contracts, onde as hashs dos arquivos encriptados são armazenadas e gerenciadas. Esta arquitetura permite o acesso destes dados de qualquer local através da internet, uma vez que ficam distribuídos, mas somente por usuários autorizados, de acordo com o controle de acesso implementado no smart contract.

Há também a opção de compartilhamento dos dados com outros usuários que têm acesso ao sistema, todos eles criptografados com as chaves públicas de cada usuário, utilizando-se criptografia assimétrica. Após a hash do arquivo ser armazenada na blockchain, o smart contract é o responsável por permitir o compartilhamento e o gerenciamento das propriedades das hashs dos arquivos entre os usuários.

A Figura 1 apresenta a estrutura da solução proposta, mostrando os principais módulos e tecnologias empregadas para o funcionamento do sistema.

Figura 1 – Arquitetura do sistema.



A interação do usuário com o sistema se dá pelo front-end desenvolvido em React (1), onde o usuário pode fazer upload de novos arquivos, compartilhar com outros usuários e fazer download desses arquivos.

Ao adicionar ou compartilhar arquivos, o usuário precisa pagar aos mineradores da Ethereum blockchain pelas operações executadas. A Metamask (2) é uma carteira Ethereum que permite a comunicação direta entre a blockchain e o usuário para que o mesmo possa aceitar ou

recusar transações, além de fornecer informações para o front-end React (1) sobre o endereço atual da conta usada pelo usuário para operar com a blockchain.

O front-end React comunica-se com o IPFS (5) através da Infura (4), uma API que proporciona uma eficiente comunicação entre os sistemas, e com a Ethereum (3) por meio da biblioteca web3.js. Quando o usuário adiciona um novo arquivo no sistema, o front-end (1) o encripta e envia o código inelegível resultante para o IPFS (5), que retorna uma hash identificando esse código na rede. Na sequência, a hash do novo arquivo é enviada à Ethereum blockchain (4), onde ficará armazenada com segurança. Para fazer o download do arquivo, o fluxo é invertido.

5 SOLUÇÃO

A implementação do sistema foi feita com base em tecnologias atuais, conforme a definição do modelo conceitual apresentado na seção 4. Esta seção apresenta em detalhes a implementação e as principais funcionalidades propostas pelo sistema.

5.1 IMPLEMENTAÇÃO

Os dados enviados à Ethereum blockchain são visíveis para qualquer usuário, característica intrínseca da tecnologia blockchain. Deste modo, o armazenamento de dados sensíveis (no caso da proposta deste trabalho, de arquivos privados) fica prejudicado. Mesmo que fossem armazenados na blockchain apenas as hashes do IPFS correspondentes aos arquivos, qualquer usuário externo ao sistema poderia ter acesso a esses dados e chegar até os arquivos originais. Também por essa característica, qualquer algoritmo de criptografia que pudesse ser implementado em um smart contract não surtiria o efeito pretendido, pois os dados originais enviados como parâmetros ainda assim estariam expostos.

Como medida para garantir a segurança e a privacidade dos arquivos armazenados no sistema, adotou-se o algoritmo de criptografia assimétrica RSA, também conhecida como criptografia de chave pública. Este algoritmo utiliza-se de duas chaves: a chave pública e a chave privada. A chave pública é utilizada para criptografar os dados, que somente poderão ser decriptografados pela chave privada. A chave privada deve ser mantida em sigilo pelo detentor, ao contrário da chave pública, que pode ser divulgada sem prejuízos à segurança das informações. Deste modo, o mecanismo de criptografia assimétrica foi aplicado ao arquivo original no front-end React, antes mesmo do envio deste à rede IPFS. Essa abordagem proporciona maior segurança ao garantir que, mesmo que um usuário externo ao sistema tiver acesso à hash do arquivo na blockchain e buscar no IPFS o arquivo armazenado, ele não será capaz de decodificar o seu conteúdo.

5.1.1 Implementação dos smart contracts

Os smart contracts podem ser escritos em qualquer editor de texto e compilados usando terminais em máquinas locais. Como forma alternativa, há o Remix, um sistema online onde é possível criar o código e compilar os contratos, opção utilizada neste trabalho. Ao fazer implantação (*deploy*), o smart contract é adicionado à Ethereum blockchain e recebe um endereço específico na rede, através do qual é possível acessá-lo e interagir com o mesmo.

Para utilizar o smart contract em uma aplicação, é preciso ter o seu endereço e a sua ABI - Application Binary Interface. A ABI é um arquivo JSON que descreve o contrato implementado e suas funções, permitindo ao desenvolvedor contextualizar o contrato e chamar suas funções. Na presente solução proposta, essas informações são utilizadas pelo front-end React para acessar o smart contract na blockchain por meio de métodos disponibilizados pela API da web3.js, uma coleção de bibliotecas que permitem a interação com a Ethereum usando uma conexão HTTP.

5.1.2 Controle de acesso

Uma das características importantes da privacidade das informações é o controle de acesso através do smart contract, evitando que, via front-end da aplicação, um usuário não cadastrado possa acessar o sistema - embora pela blockchain seja possível visualizar os dados armazenados. Deste modo, optou-se por utilizar um meio simplista para fazer este controle: o smart contract foi implementado com uma lista de usuários já cadastrados, o que impede a adição de novos usuários ao sistema. Essa abordagem é limitante para a maior parte das aplicações reais, mas foi suficiente para o demonstrar a solução proposta.

A lista de usuários cadastrados no smart contract contém o nome, o que facilita a identificação do usuário, uma vez que o endereço da conta é muito extenso e de difícil memorização; o endereço da conta, este que é a chave de acesso às funções do smart contract: somente usuários que estejam na lista conseguem chamá-las; e a chave pública (previamente gerada) do usuário, que é utilizada para encriptar os arquivos, tanto ao adicionar um novo quanto ao compartilhar um existente com outros usuários. Deste modo, não é preciso informar a chave pública de um usuário em nenhum momento durante o uso do sistema.

5.1.3 Ropsten Testnet

Para evitar custos monetários na interação com a Ethereum blockchain durante o desenvolvimento do sistema, utilizou-se a Ropsten Ethereum. Esta rede, mais conhecida como Ropsten Testnet, é uma rede que executa o mesmo protocolo da rede principal da Ethereum (Mainnet), porém com a diferença de que nesta as criptomoedas não tem nenhum valor real. Todas as transações e operações computacionais realizadas por smart contracts também são pagas com *gas*, assim como na Mainnet, entretanto é possível requisitar Ether das Faucets - carteiras da Ethereum que repassam Ether gratuitamente mediante requisição.

5.1.4 Metamask

Todas as operações com a blockchain requerem um endereço de uma carteira na blockchain, conhecidas como *wallets*, e que são utilizadas para enviar e receber as criptomoedas. Ela também serve para checar o saldo, além de armazenar chaves públicas e privadas e interagir com a blockchain. No sistema proposto, foi utilizado a Metamask, uma carteira Ether que pode ser instalado como uma extensão nos navegadores Google Chrome e Mozilla Firefox.

A cada interação com a blockchain em que hajam custos - transações ou operações computacionais, como a adição de novos arquivos ou o compartilhamento dos existentes com outros usuários - a Metamask atua como meio para o usuário confirmar ou rejeitar os valores envolvidos na operação, podendo inclusive aumentá-los ou diminui-los. Essas ações impactam diretamente no desempenho da operação, uma vez que o valor pago aos mineradores por uma determinada operação é o "atrativo"para eles concentrarem mais ou menos esforço para concluir o trabalho de mineração e adicionar as transações na blockchain.

Além de atuar como meio para o usuário aceitar ou recusar o pagamento por transações, a Metamask também é o meio pelo qual o usuário gerencia sua carteira, podendo utilizar diversas contas e definir uma para ser utilizada no momento pelas aplicações em operações com a blockchain. Para fazer requisições, é preciso sempre enviar à blockchain o endereço da carteira que requisitante, e é através do Metamask que o front-end em React é capaz de detectar a conta que o usuário está usando no momento.

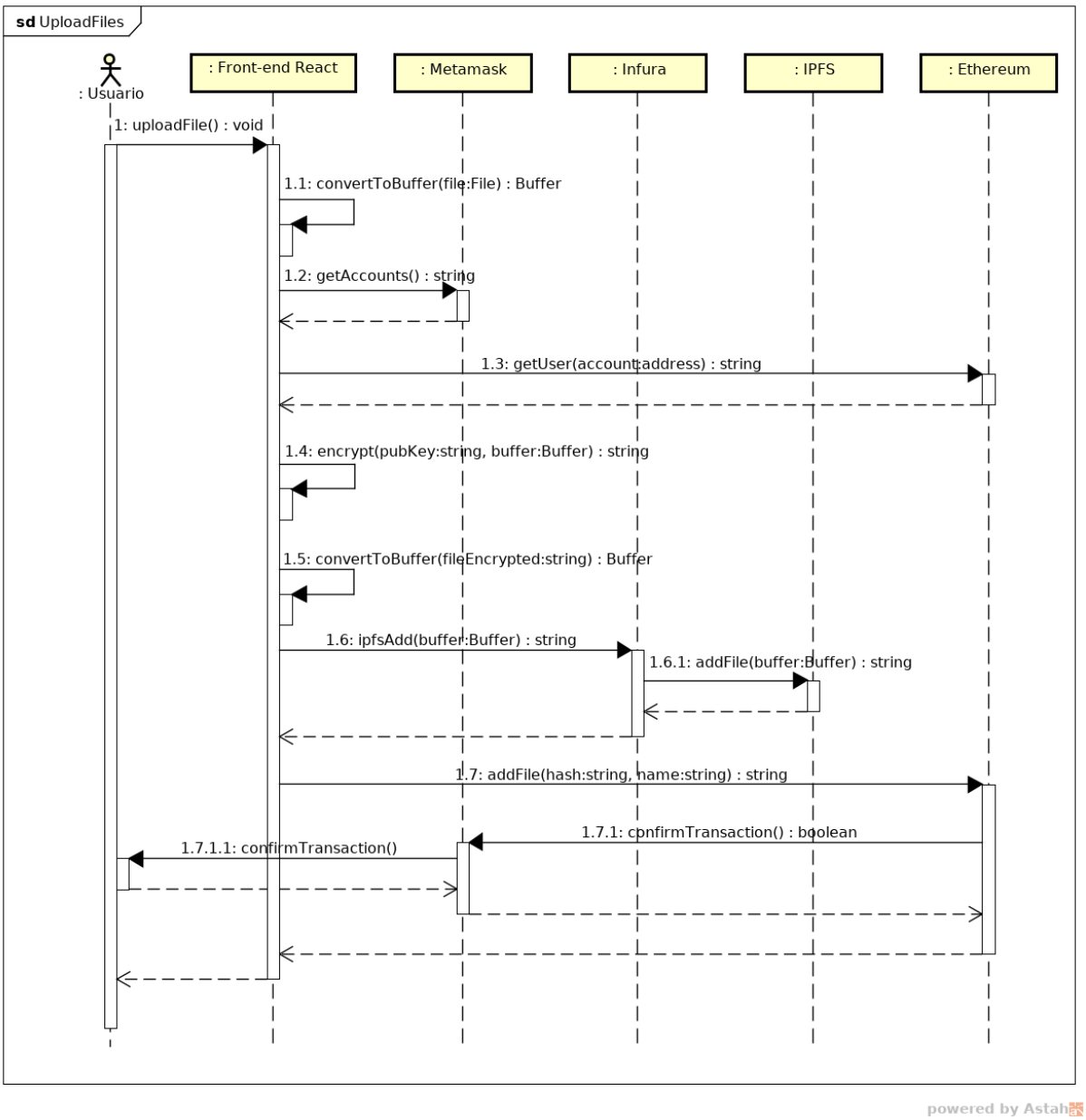
5.2 PRINCIPAIS FUNCIONALIDADES

5.2.1 Upload de arquivos

Para armazenar seus arquivos com segurança, o usuário seleciona a opção de upload no sistema e seleciona um arquivo do seu computador, conforme é apresentado em detalhes na Figura 2. Ao carregar o arquivo no navegador, o método *convertToBuffer(file)* é executado e faz a conversão do arquivo em um buffer. Este procedimento é necessário para adicionar o arquivo à rede do IPFS. Na sequência, por meio da web3.js, o método *getAccounts()* é utilizado para obter o endereço da conta em uso na Metamask, informação que servirá de parâmetro para *getUser(account)*. Este método acessa o smart contract para buscar a chave pública do usuário.

Com a chave pública do usuário, o sistema faz a criptografia do buffer do arquivo em *encrypt(pubKey, buffer)*. Em seguida, o dado criptografado é convertido novamente em um buffer em *convertToBuffer(fileEncrypted)* e enviado à rede do IPFS através da API da Infura, por

Figura 2 – Upload de arquivos - diagrama de sequência.



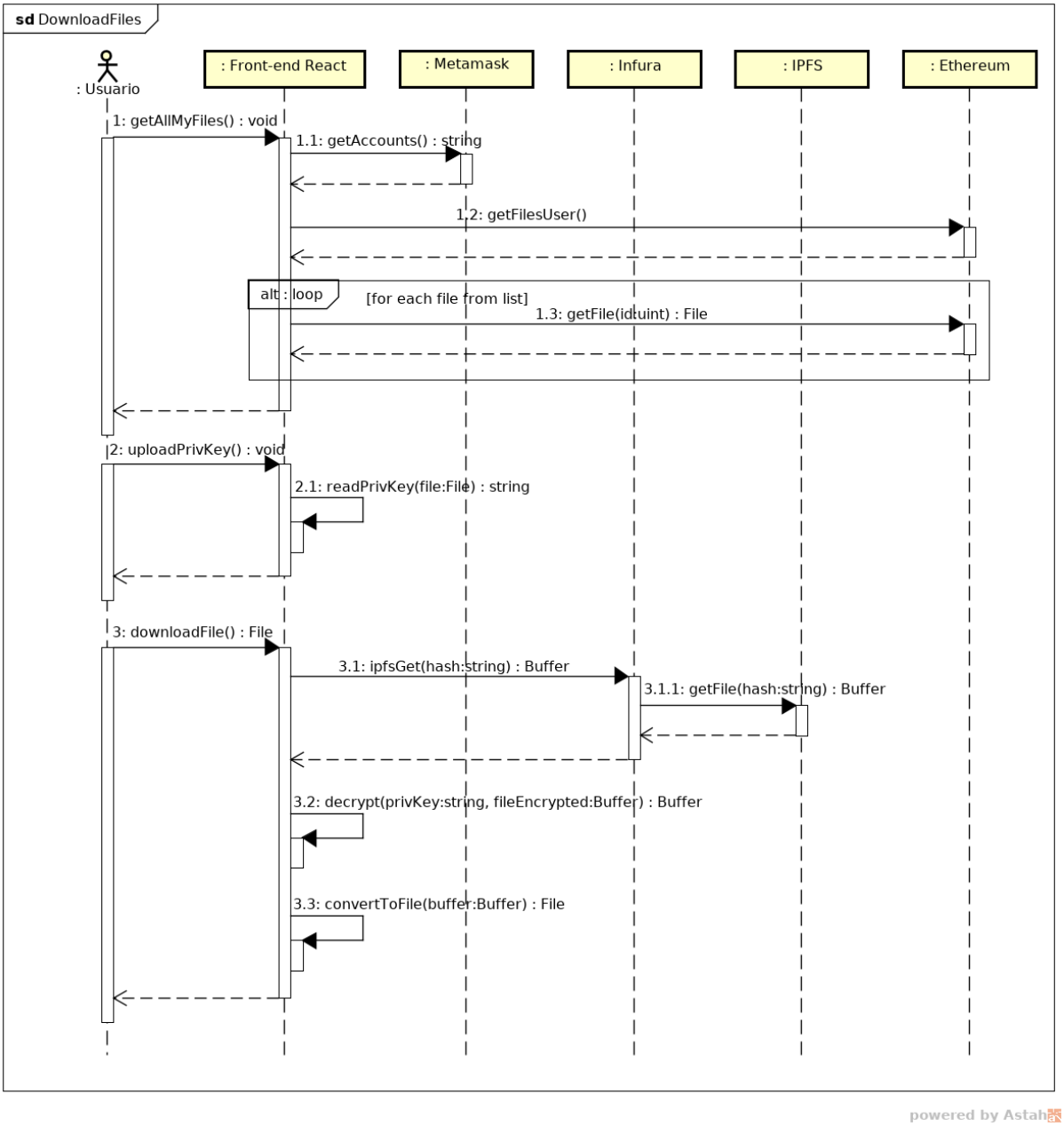
meio do *ipfsAdd(buffer)*, que estende o chamado ao IPFS com *addFile(buffer)*. Como retorno desse chamado, a Infura encaminha a hash gerada pelo IPFS ao front-end React, que executa o método *addFile(hash, name)*, enviando o arquivo para a blockchain.

Ao receber a requisição de armazenamento de informações - execução de um método do smart contract que tem um custo de processamento - a Ethereum chama *confirmTransaction()* na Metamask, que então abre um *pop-up* na tela para que o usuário possa confirmar ou recusar a transação. Se o usuário aceitar a transação, a hash e o nome do arquivo são armazenados na blockchain, ligados ao endereço da sua conta. Caso contrário, a operação é cancelada.

5.2.2 Download de arquivos

O download de arquivos é a meio pelo qual o usuário acessa o conteúdo dos seus arquivos armazenados no sistema. Esta funcionalidade está representada na Figura 3.

Figura 3 – Download de arquivos - diagrama de sequência.



Ao acessar o sistema, o método *getAllMyFiles()* é executado no front-end React, desencadeando outros métodos que serão responsáveis por apresentar ao usuário todos os arquivos aos quais ele tem acesso. O primeiro deles é o *getAccounts()*, que obtém o endereço da conta em uso na Metamask. Em seguida, *getFilesUser()* é chamado no smart contract e retorna uma

lista com os identificadores de todos os arquivos aos quais o usuário tem acesso. Com essa lista, o front-end React cria um *loop* para requisitar ao smart contract cada um deles, através do *getFile(id)*. Essa abordagem se fez necessária porque, até o presente momento, uma função em Solidity não pode retornar listas com objetos complexos, apenas tipos primitivos (mesmo assim, com algumas limitações), impossibilitando que com apenas uma requisição o usuário pudesse ter acesso a todas as informações de todos os seus arquivos.

Após ter acesso aos seus arquivos, o usuário faz upload de um arquivo *.txt* contendo a sua chave privada em *uploadPrivKey()*, a qual é lida pelo método *readPrivKey(file)*. Em seguida, o usuário seleciona o arquivo e clica na opção de download, que executa *downloadFile()* no front-end React. O método *ipfsGet(hash)* é chamado na Infura, que estende ao IPFS com *getFile(hash)* enviando como parâmetro a hash do arquivo armazenada na blockchain. Como retorno do método, o front-end recebe um buffer do arquivo encriptado. O processo segue com a decriptação do buffer em *decrypt(privKey, fileEncrypted)*, usando a chave privada fornecida pelo usuário anteriormente. Para finalizar, *convertToFile(buffer)* é utilizado para conversão do buffer em um arquivo, e o download é concluído.

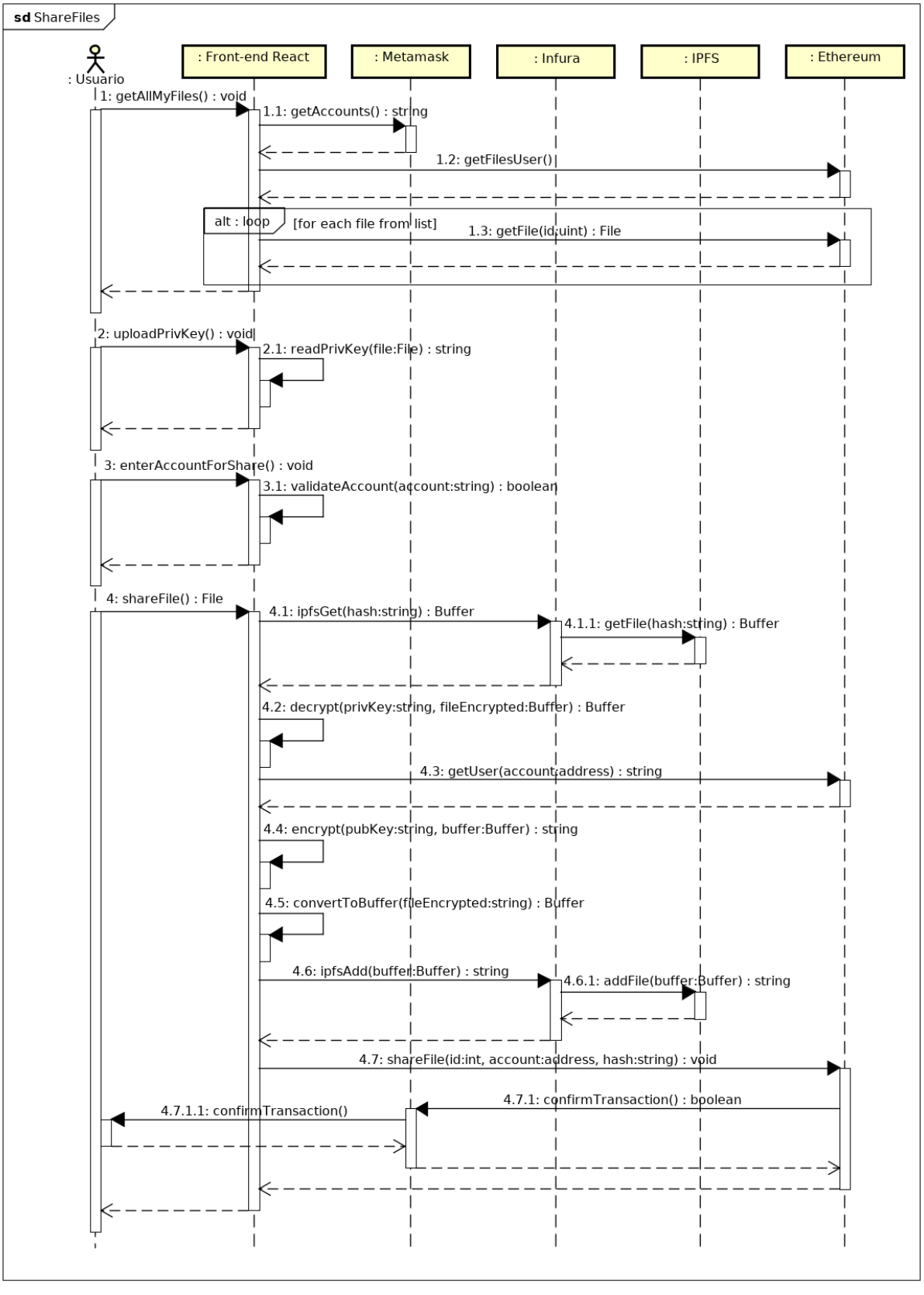
5.2.3 Compartilhamento de arquivos

Compartilhar os arquivos que o usuário tem acesso com outros usuários cadastrados é uma das principais funcionalidades do sistema. A parte inicial do processo de compartilhamento de arquivos é muito semelhante à da funcionalidade de download de arquivos, conforme o detalhamento da Figura 4. Para apresentar todos os arquivos aos quais o usuário tem acesso, o seguinte fluxo é seguido: ao acessar o sistema, o front-end React detecta o endereço da conta que o usuário está utilizando, faz um chamado ao smart contract para obter a lista de identificadores dos arquivos aos quais o usuário tem acesso e cria um *loop* para buscar cada um deles na blockchain. Em seguida, o usuário faz upload de um arquivo *.txt* contendo a sua chave privada e o front-end faz a leitura desse arquivo. Esta chave será utilizada na sequência, durante o procedimento de decriptação do arquivo.

A partir deste ponto, as funcionalidades se diferem. O usuário seleciona o arquivo e digita o endereço da conta do usuário com o qual ele deseja fazer o compartilhamento (*enterAccountForShare()*) e o sistema faz uma validação se a sequência digitada corresponde à um endereço válido da Ethereum através do *validateAccount(account)*.

Com o método *shareFile()* o front-end React dá início ao principal procedimento da funcionalidade de compartilhamento de arquivos. Este procedimento consiste no acesso ao buf-

Figura 4 – Compartilhamento de arquivos - diagrama de sequência.



fer do arquivo codificado através da Infura com o método *ipfsGet(hash)* estendido ao IPFS com *getFile(hash)*, enviando como parâmetro a hash armazenada na blockchain; decriptação do código com a chave privada do usuário com o método *decrypt(privKey, fileEncrypted)*; busca da chave pública do usuário que receberá o acesso ao arquivo na blockchain, executando *getUser(account)* no smart contract com o endereço digitado como parâmetro; encriptação do buffer do arquivo original com a chave pública do usuário que acaba de receber o acesso, em *encrypt(pubKey, buffer)*; conversão do dado criptografado em um buffer em *convertToBuffer(fileEncrypted)*; adição deste buffer no IPFS, através de *ipfsAdd(buffer)* na Infura e *addFile(buffer)* no IPFS; chamado do método *shareFile(id, account, hash)* no smart contract, que recebe como parâmetros o identificador do arquivo a ser compartilhado na blockchain, o endereço da conta do usuário que receberá o acesso e a hash da codificação do arquivo original com a sua chave pública; confirmação da transação pelo *confirmTransaction()*, chamado que parte da Ethereum para a Metamask e é estendido para o usuário através de um *pop-up* na tela para que o usuário possa confirmar ou recusar a transação.

6 AVALIAÇÃO

O sistema proposto utiliza blockchain, IPFS e criptografia para garantir a segurança. Para avaliar o impacto do tamanho dos arquivos no desempenho do sistema durante os processos de upload, download e compartilhamento de arquivos, foram executados testes utilizando-se cinco arquivos com a extensão .txt de tamanhos 1kB, 10kB, 100kB, 1MB e 10MB. No processo de busca dos arquivos do usuário na blockchain foram executados testes com 10, 50 e 100 arquivos armazenados, objetivando-se avaliar o impacto da quantidade de arquivos armazenada no tempo de carregamento inicial do sistema. Cada teste foi executado 15 vezes em um computador com processador Intel Core i5 e 3,7 GB de RAM, sistema operacional Ubuntu 18.04.1 LTS, no navegador Google Chrome e com conexão à internet com velocidade de 48,04 Mbps de download e 25,89 Mbps de upload. Os gráficos apresentados a seguir foram construídos com a média de tempo dos resultados obtidos de cada conjunto.

6.1 RESULTADOS EXPERIMENTAIS DO PROCESSO DE UPLOAD DE ARQUIVO

No processo de upload de arquivo, foram avaliados os tempos de execução das etapas de conversão do arquivo em um buffer, encriptação do buffer do arquivo original, conversão do arquivo encriptado novamente em um buffer, adição deste buffer no IPFS e envio da hash resultante à blockchain, bem como o tempo total de execução da funcionalidade de upload do arquivo.

Na etapa de conversão do arquivo original para um buffer ilustrado pela Figura 5, o arquivo de 1kB teve um tempo médio de 115 ms, o de 10kB 126 ms, o de 100kB 129 ms, o de 1MB 143 ms e o de 10MB 158 ms. Os resultados obtidos revelam que houve um aumento no tempo de conversão à medida que o tamanho do arquivo aumentou, porém não na mesma proporção.

Durante a etapa de encriptação do buffer do arquivo original, ficou evidente o impacto do tamanho do arquivo no tempo de execução da codificação, como ilustrado na Figura 6. Partiu de 82 ms no arquivo de 1kB, passando por 114 ms no de 10kB, 174 ms no de 100kB, 556 ms no de 1MB e saltou para 9.074 ms no arquivo de 10MB - um aumento de 8,52 s em relação ao arquivo de 1MB, o que representou 16,32 vezes. Diante dos números apresentados, foi possível concluir que o a etapa upload de arquivos maiores do que 1MB pode prejudicar o desempenho do sistema, com percepção direta do usuário.

O tempo transcorrido durante a etapa de conversão do arquivo encriptado em um buffer também apresentou um aumento à medida que se aumentou o tamanho do arquivo, tendência

Figura 5 – Resultado experimental do tempo de conversão do arquivo original para buffer.

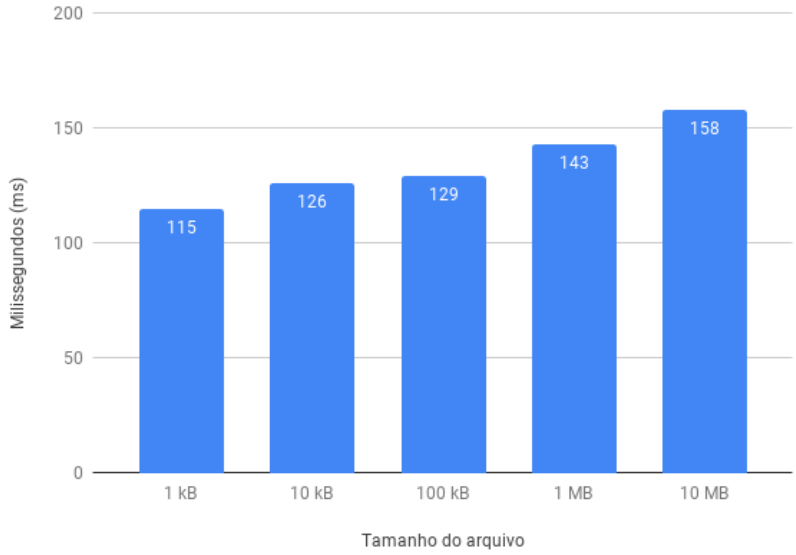
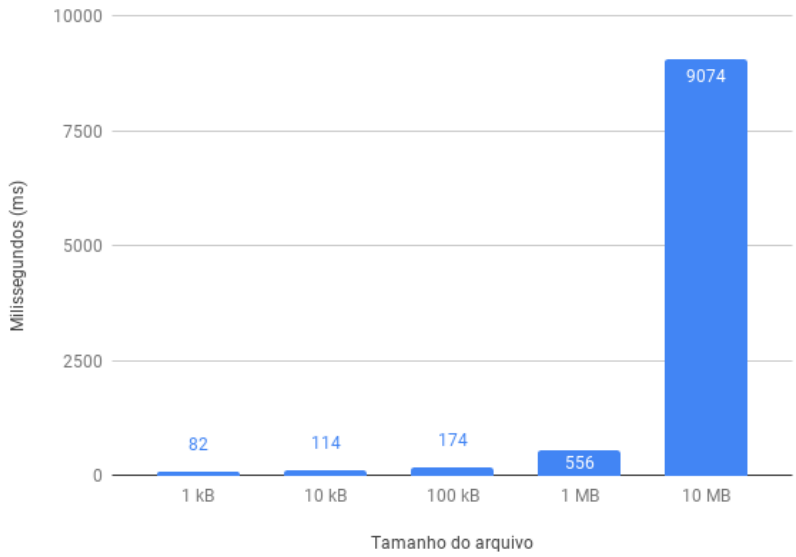


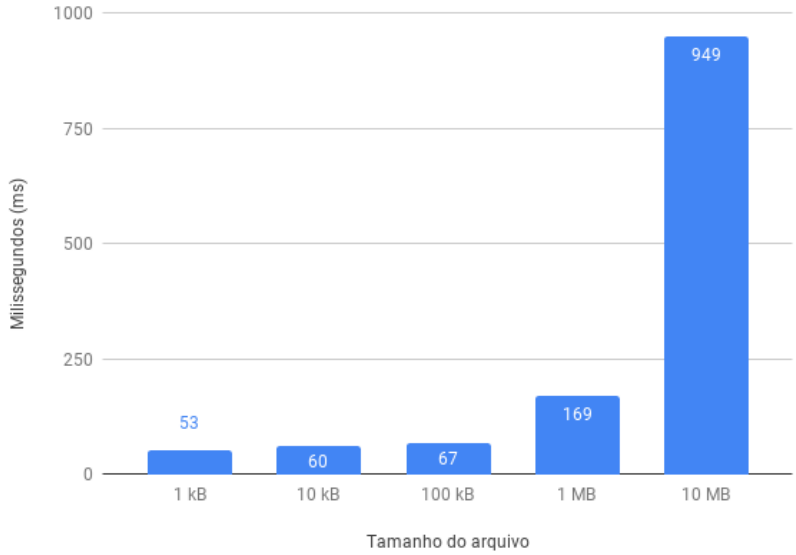
Figura 6 – Resultado experimental do tempo de encriptação do buffer.



semelhante à etapa anterior. Entretanto, a diferença entre o arquivo de 1MB e o de 10MB foi de apenas 0,78 s, um impacto muito menor no desempenho ainda quando comparado à etapa de encriptação. Os resultados obtidos foram: 53 ms de conversão para o arquivo de 1kB, 60 ms para o de 10kB, 67 ms para o de 100kB, 169 ms para o de 1MB e 949 ms para o de 10MB, conforme apresentado na Figura 7.

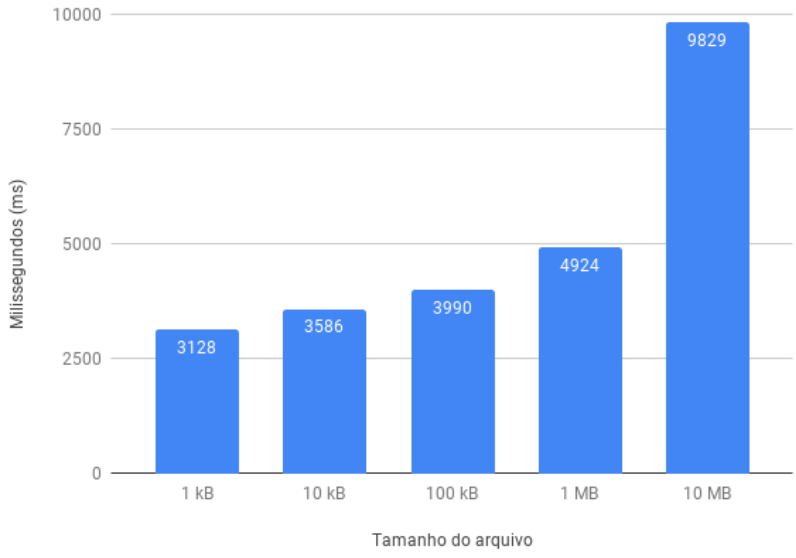
Nos testes executados durante a etapa de adição do buffer do arquivo encriptado no IPFS, apresentados na Figura 8, obtiveram-se os tempos de 3.128 ms para o arquivo de 1kB, 3.586 ms para o de de 10kB, 3.990 ms para o de 100kB, 4.924 ms para o de 1MB e 9.829 ms

Figura 7 – Resultado experimental do tempo de conversão do arquivo encriptado em um buffer.



para o de 10MB. Esta etapa depende, quase que na sua totalidade, de um serviço externo ao front-end desenvolvido. Com isso, o tempo de duração da adição pode variar de acordo com a velocidade de conexão à internet que o usuário dispõe e com a disponibilidade e velocidade de resposta da Infura e do IPFS. De qualquer forma, foi possível observar-se que houve um aumento significativo no tempo de duração da etapa quando o tamanho do arquivo foi superior a 1MB.

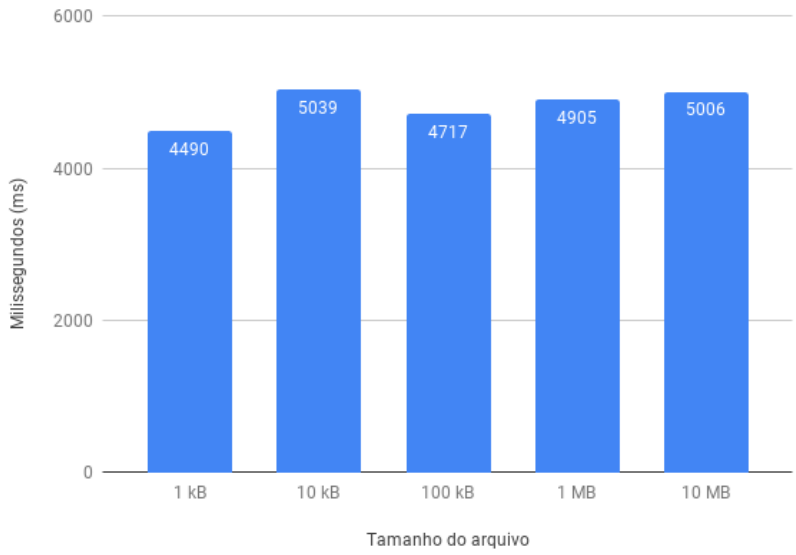
Figura 8 – Resultado experimental do tempo de adição do buffer no IPFS.



Durante a etapa de envio da hash do arquivo à blockchain, os tempos de duração men-

surados foram de 4.490 ms para o arquivo de 1kB, 5.039 ms para o de de 10kB, 4.717 ms para o de 100kB, 4.905 ms para o de 1MB e 5.006 ms para o de 10MB, conforme a Figura 9. Diferentemente das tendências dos resultados obtidos em outras etapas, os testes executados demonstraram que o tempo de envio da hash do arquivo à blockchain não tem relação direta com o tamanho do arquivo adicionado. E isto é compreensível, uma vez que, independentemente do conteúdo e do tamanho do arquivo adicionado ao IPFS, a hash resultante é sempre do mesmo tamanho. Com isso, o valor sugerido de pagamento para a mineração na blockchain também é sempre o mesmo, porém pode ser alterado pelo usuário, o que provavelmente modificará os tempos. Da mesma forma que a etapa de adição do buffer ao IPFS, o tempo de envio da hash à blockchain é dependente da performance de um sistema externo e da conexão com a rede somado a interação com o usuário - através da Metamask, com o aumento ou diminuição do valor ofertado para mineração e com a aprovação ou rejeição das transações na Ethereum blockchain.

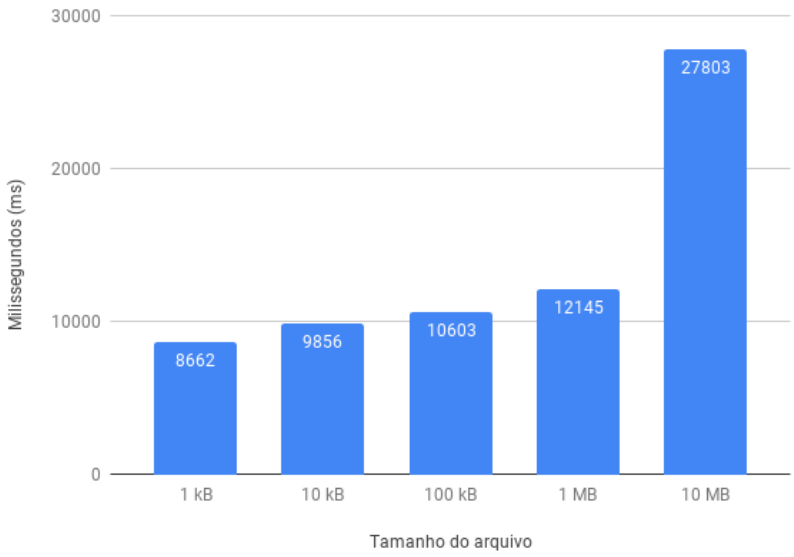
Figura 9 – Resultado experimental do tempo de envio da hash do arquivo à blockchain.



Ao encerrar o processo de upload de arquivos no sistema, obtiveram-se os tempos totais de execução para cada caso de teste, apresentados na Figura 10. O arquivo de 1kB apresentou um tempo médio total de 8.662 ms, o de 10kB 9.856 ms, o de 100kB 10.603 ms, o de 1MB 12.145 ms e o de 10MB 27.803 ms. Com os resultados obtidos, foi possível observar-se que o desempenho da funcionalidade de upload variou relativamente pouco para arquivos de até aproximadamente 1MB - 3,48 s foi a diferença de tempo de um arquivo de 1kB para um arquivo de 1MB. Já para arquivos com tamanhos superiores a 1MB, o tempo de conclusão tendeu a subir consideravelmente - 15,66 s foi a diferença de tempo de um arquivo de 1MB para um arquivo

de 10MB, o que representou um aumento de aproximadamente 228,93%.

Figura 10 – Resultado experimental do tempo total de execução do processo de upload de arquivos.



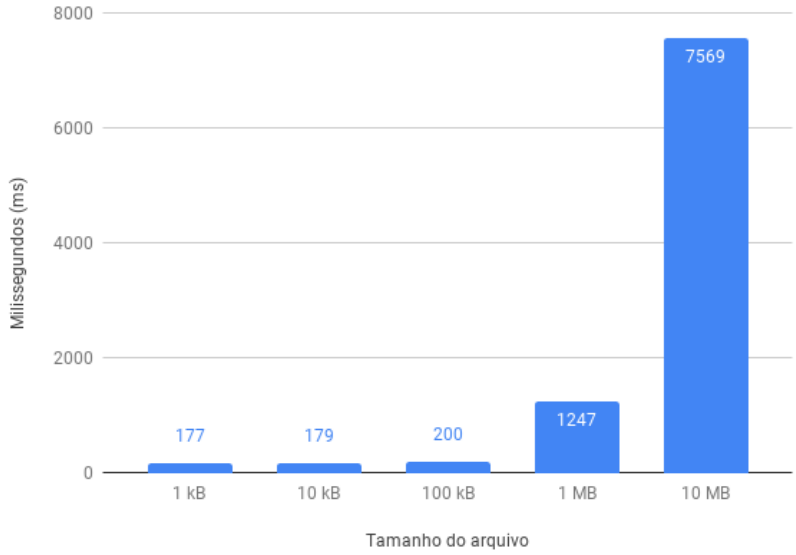
6.2 RESULTADOS EXPERIMENTAIS DO PROCESSO DE DOWNLOAD DE ARQUIVO

No processo de download de arquivo, foram avaliados os tempos de execução das etapas de busca da hash do arquivo encriptado no IPFS, decriptação do buffer recuperado e conversão deste dado decriptado em um arquivo, bem como o tempo total de execução do processo.

Na etapa de busca da hash do arquivo encriptado no IPFS ilustrado pela Figura 11, o arquivo de 1kB apresentou um tempo médio de 177 ms, o de 10kB 179 ms, o de 100kB 200 ms, o de 1MB 1.247 ms e o de 10MB 7.569 ms. Como dependem do desempenho de um sistema externo ao front-end desenvolvido, os resultados obtidos podem sofrer consideráveis variações, todavia demonstraram que houve um impacto do tamanho do arquivo recuperado do IPFS no tempo de execução. Isto ocorreu porque, quanto maior o arquivo armazenado, maior será também o dado resultante da sua encriptação.

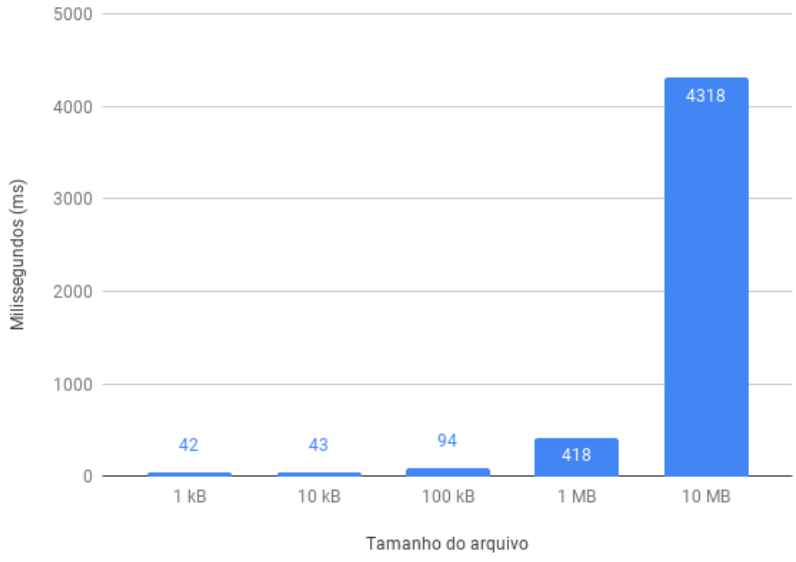
Durante a etapa de decriptação do buffer recuperado do IPFS, os tempos de duração mensurados foram de 42 ms para o arquivo de 1kB, 43 ms para o de de 10kB, 94 ms para o de 100kB, 418 ms para o de 1MB e 4.318 ms para o de 10MB, conforme a Figura 12. Da mesma forma que na etapa de encriptação do arquivo, observou-se um aumento significativo do tempo de execução da etapa de decriptação para arquivos maiores do que 1MB: 3,9 s foi a diferença

Figura 11 – Resultado experimental do tempo de busca da hash do arquivo encriptado no IPFS.



de tempo de um arquivo de 1MB para um arquivo de 10MB, o que representou um aumento de aproximadamente 1.033%.

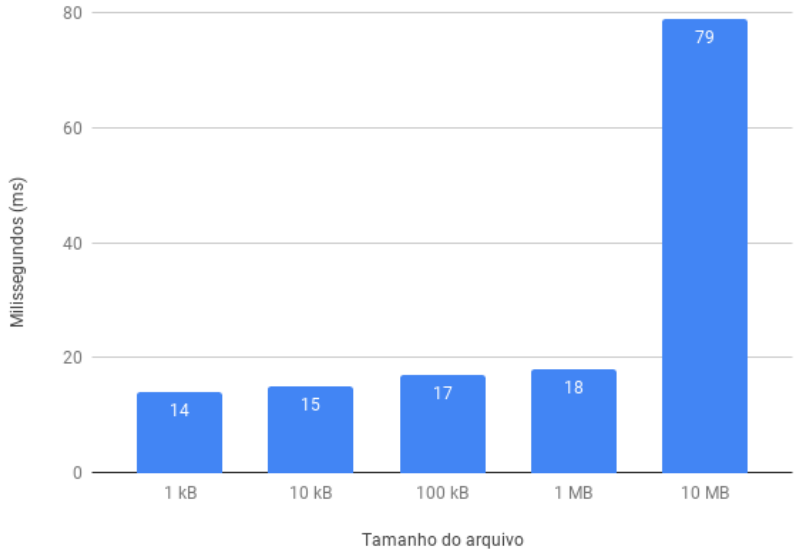
Figura 12 – Resultado experimental do tempo de decriptação do buffer recuperado do IPFS.



Nos testes executados durante a etapa de conversão do dado decriptado em um arquivo, apresentados na Figura 13, obtiveram-se os tempos de 14 ms para o arquivo de 1kB, 15 ms para o de de 10kB, 17 ms para o de 100kB, 18 ms para o de 1MB e 79 ms para o de 10MB. Os resultados obtidos revelam que, para arquivos com tamanho de até 1MB, o tempo de execução da etapa de reconstrução do arquivo praticamente não varia. Para arquivos maiores aqui repre-

sentados pelo arquivo de 10MB, o tempo de execução teve um aumento de 61 ms em relação ao arquivo de 1MB, porém este é inexpressivo em relação ao tempo total do processo.

Figura 13 – Resultado experimental do tempo de conversão do dado decriptado em um arquivo.

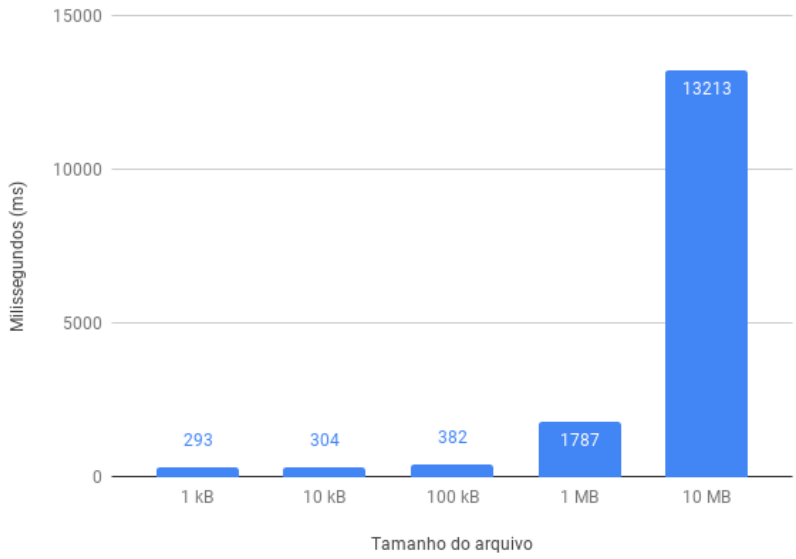


Ao encerrar o processo de download de arquivos no sistema, obtiveram-se os tempos totais de execução para cada caso de teste, apresentados na Figura 14. O arquivo de 1kB apresentou um tempo médio total de 293 ms, o de 10kB 304 ms, o de 100kB 382 ms, o de 1MB 1.787 ms e o de 10MB 13.213 ms. Da mesma forma que foram constatados pelas etapas do processo, observou-se um impacto significativo quando o tamanho do arquivo superou 1MB. A diferença entre o arquivo de 1MB e o arquivo de 10MB ficou na casa dos 11,43 s, um aumento de aproximadamente 7,39 vezes.

6.3 RESULTADOS EXPERIMENTAIS DO PROCESSO DE COMPARTILHAMENTO DE ARQUIVO

No processo de compartilhamento de arquivos, foram avaliados o tempo de execução da etapa de envio dos dados de compartilhamento à blockchain e o tempo total de execução do processo. As demais etapas que compõe este processo já foram mensuradas nos processos anteriores e, por isso, não foram novamente apresentadas, mas estão compreendidas nos resultados do tempo total de execução. São elas: busca da hash do arquivo encriptado no IPFS, decriptação do buffer recuperado do IPFS, encriptação do buffer com chave pública do usuário que receberá o acesso, conversão do arquivo encriptado em um buffer e adição do buffer no IPFS.

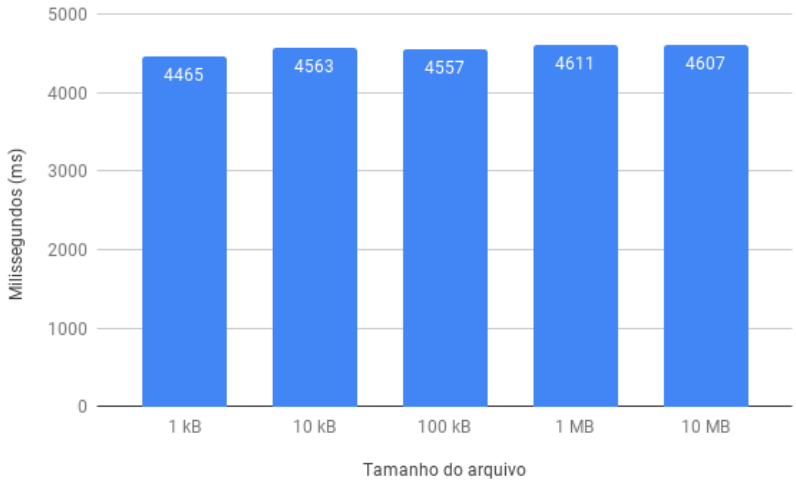
Figura 14 – Resultado experimental do tempo total de execução do processo de download de arquivos.



Durante a etapa de envio dos dados de compartilhamento à blockchain, os tempos de duração mensurados foram de 4.465 ms para o arquivo de 1kB, 4.563 ms para o de de 10kB, 4.557 ms para o de 100kB, 4.611 ms para o de 1MB e 4.607 ms para o de 10MB, conforme a Figura 15. Assim como verificou-se na etapa de envio da hash do arquivo à blockchain (processo de upload), o tempo de execução da etapa de envio dos dados de compartilhamento à blockchain não sofreu impactos relacionados ao tamanho do arquivo original devido à arquitetura do sistema proposto. Independentemente do tamanho do arquivo, o tamanho dos dados enviados para a função correspondente no smart contract e a complexidade para executá-la serão sempre do mesmo grau. De qualquer modo, é importante frisar que estas etapas dependem do desempenho de um sistema externo ao front-end desenvolvido, sofrendo variações de acordo com a conexão com a rede, com a interação com o usuário e com o valor pago pela execução da operação na Ethereum blockchain.

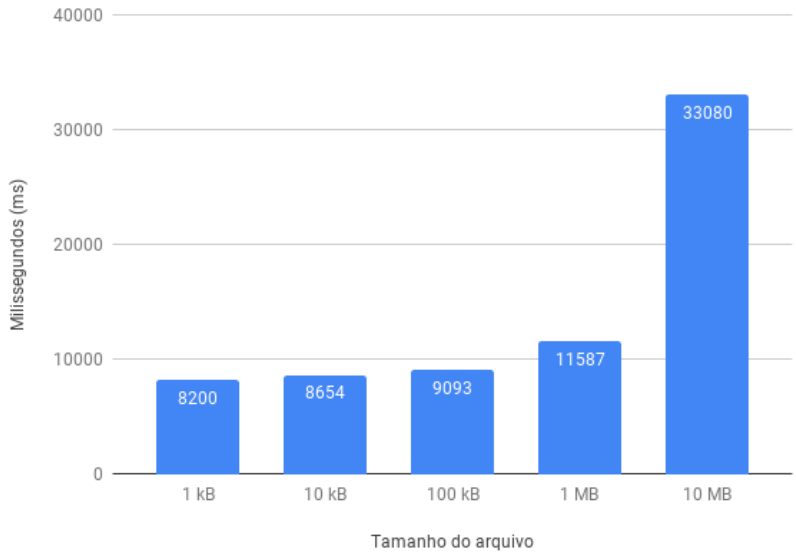
O tempo total de execução do processo de compartilhamento de arquivos, que compreende algumas etapas em comum com os processos de upload e download de arquivos, está representado na Figura 16. Obtiveram-se os tempos de 8.200 ms para o arquivo de 1kB, 8.654 ms para o de de 10kB, 9.093 ms para o de 100kB, 11.587 ms para o de 1MB e 33.080 ms para o de 10MB. Com os resultados obtidos, observou-se que o impacto do tamanho do arquivo em relação ao tempo de execução do processo foi significativo, especialmente para arquivos maiores do que 1MB. A diferença de tempo entre o arquivo de 1kB e o de 1MB foi de 3,39 s, o que

Figura 15 – Resultado experimental do tempo de envio dos dados de compartilhamento à blockchain.



representa 140,30% de aumento. Já a diferença de tempo entre o arquivo de 1kB e o de 1MB foi de 21,49 s, o que representa 285,49% de aumento.

Figura 16 – Resultado experimental do tempo total de execução do processo de compartilhamento de arquivos.



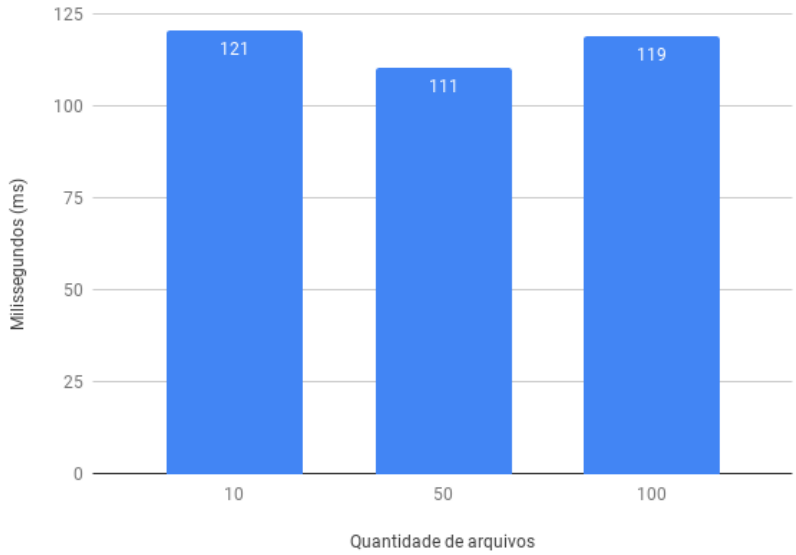
6.4 RESULTADOS EXPERIMENTAIS DO PROCESSO DE BUSCA DE TODOS OS ARQUIVOS DO USUÁRIO NA BLOCKCHAIN

O processo de busca dos arquivos do usuário na blockchain é o primeiro a ser executado toda vez que o sistema é carregado no navegador. Como resultado, são apresentados na tela todos os arquivos aos quais o usuário tem acesso. A avaliação deste processo compreendeu as etapas de identificação da conta do usuário na Metamask, busca da quantidade total de arquivos aos quais o usuário tem acesso na blockchain e recuperação de cada um deles, além do tempo total de execução do processo.

Diferentemente das avaliações dos processos anteriores, optou-se por utilizar neste processo a quantidade de arquivos armazenada ao invés de arquivos com tamanhos originalmente distintos. O motivo foi a estrutura dos dados armazenados na blockchain, que apresenta o mesmo tamanho e complexidade para retorno ao front-end independentemente do tamanho do arquivo originalmente enviado à blockchain. Já a quantidade de total de arquivos armazenados pode provocar alterações maiores no desempenho, como será apresentado nos gráficos a seguir.

Conforme visto na Figura 17, a quantidade total de arquivos armazenados não influencia na identificação da conta do usuário na Metamask. Durante a execução desta etapa, a comunicação direta entre o front-end e a Metamask apresentou os resultados de 121 ms para os testes com 10 arquivos, 111 ms para 50 arquivos e 119 ms para 100 arquivos armazenados.

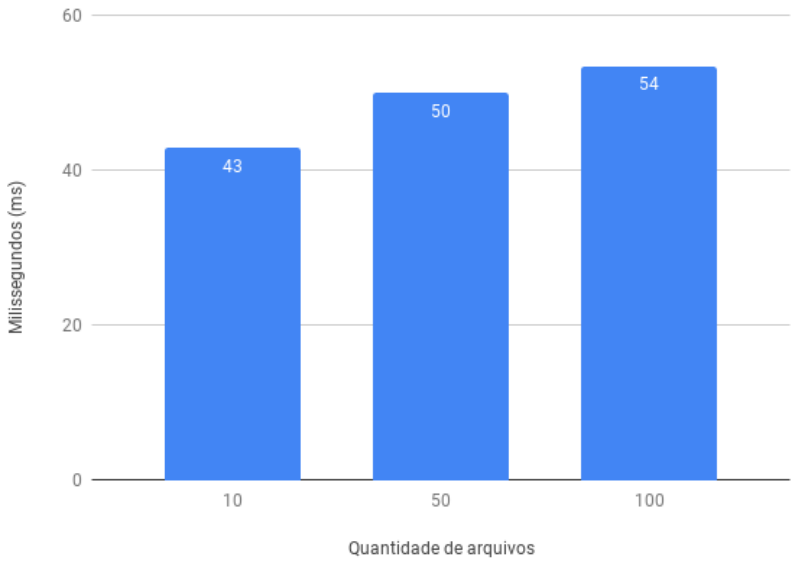
Figura 17 – Resultado experimental do tempo de identificação da conta do usuário na Metamask.



Na etapa de busca da quantidade total de arquivos do usuário na blockchain ilustrado

pela Figura 18, 43 ms foi o tempo médio de resposta para os testes com 10 arquivos, 50 ms para 50 arquivos e 54 ms para 100 arquivos armazenados. Com os resultados obtidos, pode-se afirmar que a quantidade total de arquivos armazenados praticamente não influencia no desempenho do processo como um todo. Embora tenha apresentado um aumento, o tempo de execução para 100 arquivos foi apenas 1,24 vezes maior do que o tempo utilizado para 10 arquivos, mesmo esta quantidade sendo 10 vezes maior, e essa diferença foi de 11 ms.

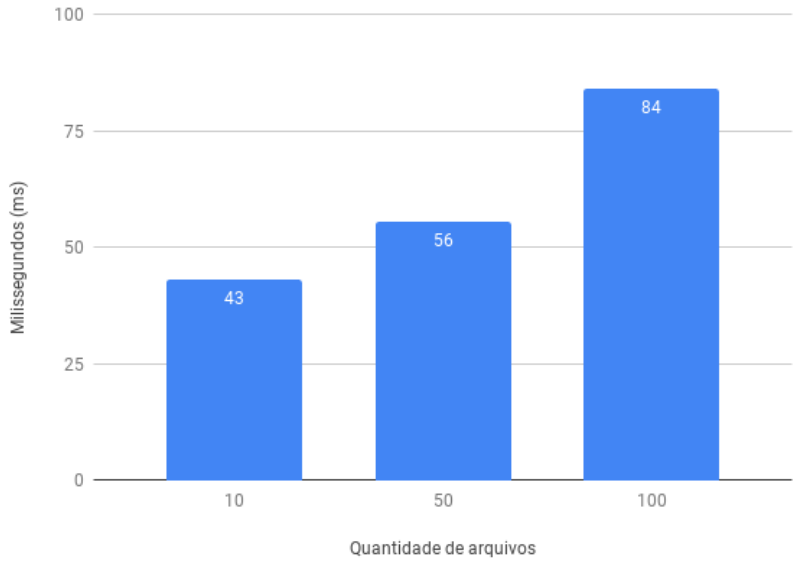
Figura 18 – Resultado experimental do tempo de busca da quantidade total de arquivos do usuário na blockchain.



O tempo transcorrido durante a etapa de recuperação de cada arquivo do usuário da blockchain apresentou um aumento à medida em que se aumentou a quantidade de arquivos armazenados no sistema. Obtiveram-se os tempos de 43 ms com 10 arquivos, 56 ms com 50 arquivos e 84 ms nos testes com 100 arquivos armazenados. Estes resultados tem um impacto significativo no tempo total de execução do processo, visto que os resultados de tempo obtidos são multiplicados pela quantidade de arquivos a serem recuperados. Esta relação representa um tempo total de recuperação de 430 ms para 10 arquivos, 2.880 ms para 50 arquivos e 8.400 ms para 100 arquivos. Os resultados registrados representam um aumento de aproximadamente 19,53 vezes no tempo total de recuperação na comparação entre os testes de 10 e 100 arquivos, enquanto a diferença na quantidade de arquivos é de 10 vezes.

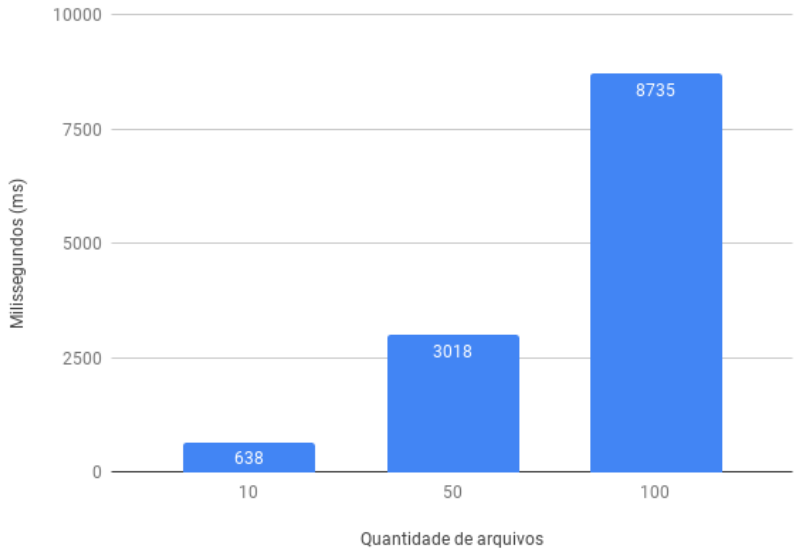
Ao encerrar o processo de busca dos arquivos do usuário na blockchain, obtiveram-se os tempos totais de execução para cada caso de teste, apresentados na Figura 20. Os tempos de duração mensurados foram de 638 ms para 10 arquivos, 3.018 ms para 50 arquivos e 8.735 ms

Figura 19 – Resultado experimental do tempo de recuperação de cada arquivo do usuário da blockchain.



para 100 arquivos armazenados. Os resultados analisados demonstram claramente o impacto da quantidade de arquivos armazenados na inicialização do sistema, representado pelo aumento de 13,69 vezes (1.369,04%) no tempo total de busca quando comparados os testes com 10 e 100 arquivos.

Figura 20 – Resultado experimental do tempo total de execução do processo de busca de todos os arquivos do usuário na blockchain.



7 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou um sistema para armazenamento e compartilhamento de arquivos através da combinação de smart contracts e IPFS, que garante a segurança dos arquivos através do emprego de criptografia assimétrica. Embora todos os dados armazenados na Ethereum blockchain sejam visíveis para qualquer usuário da rede, a solução proposta garante que estes dados não sejam legíveis para qualquer usuário que não seja detentor da chave privada correspondente à chave pública empregada na encriptação do arquivo. Além disso, armazenar a massa de dados no IPFS e adicionar apenas a hash correspondente na blockchain garante custos de mineração viáveis, considerando os benefícios apresentados.

Os resultados obtidos nas avaliações demonstram que os tempos de espera para upload, compartilhamento e download aumentam consideravelmente quando o tamanho dos arquivos ultrapassa a casa de 1MB. Dependendo das circunstâncias em que o sistema for empregado, o uso de arquivos maiores que este podem ser comuns. Portanto, como trabalhos futuros, pretende-se melhorar a eficiência do sistema nas etapas de encriptação e decriptação e implementar um mecanismo de paginação para exibição dos arquivos do usuário, permitindo-se assim o armazenamento eficiente de arquivos maiores. Pretende-se ainda desenvolver novas funcionalidades como remoção do acesso de um usuário à um determinado arquivo e exclusão de arquivos armazenados.

REFERÊNCIAS

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

CHRISTIDIS, K.; DEVETSIKIOTIS, M. Blockchains and smart contracts for the internet of things. **IEEE Access**, v. 4, p. 2292–2303, 2016. ISSN 2169-3536.

COSTA, L. da; PINHEIRO, B.; ARAÚJO, R.; ABELÉM, A. Compartilhamento Seguro de Arquivos de Saúde usando Criptografia Baseada em Atributos e Redes Descentralizadas. ago. 2018. Disponível em: <http://portaldeconteudo.sbc.org.br/index.php/sbcas/article/view/3682/3634>. Acesso em: 30 out. 2018.

GRÄTHER, W.; KOLVENBACH, S.; RULAND, R.; SCHÜTTE, J.; TORRES, C.; WENDLAND, F. Blockchain for education: Lifelong learning passport. In: EUROPEAN SOCIETY FOR SOCIALLY EMBEDDED TECHNOLOGIES (EUSSET). **Proceedings of 1st ERCIM Blockchain Workshop 2018**. [S.l.], 2018.

MCCORRY, P.; SHAHANDASHTI, S. F.; HAO, F. A smart contract for boardroom voting with maximum voter privacy. In: SPRINGER. **International Conference on Financial Cryptography and Data Security**. [S.l.], 2017. p. 357–375.

PRATHYUSHA, T.; KAVYA, M.; AKSHITA, P. S. L. Block chain technology. 2018.

RAY, S. Blockchains versus Traditional Databases. nov. 2017. Disponível em: <https://hackernoon.com/blockchains-versus-traditional-databases-c1a728159f79>. Acesso em: 30 out. 2018.

SZABO, N. Smart contracts. **Unpublished manuscript**, 1994. Disponível em: <http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html>. Acesso em: 01 out. 2018.

VIEIRA, A. B. Transmissão de mídia contínua ao vivo em p2p: modelagem, caracterização e implementação de mecanismo de resiliência a ataques. UFMG, 2010.

WOOD, G. Ethereum: A secure decentralised generalised transaction ledger. **Ethereum project yellow paper**, v. 151, p. 1–32, 2014.