

**UNIVERSIDADE FEDERAL DE SÃO CARLOS**

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

**AUDITCHAIN: UM MECANISMO PARA  
ATESTAR A INTEGRIDADE DE *LOGS*  
BASEADO NA PROVA DE EXISTÊNCIA EM  
*BLOCKCHAIN* PÚBLICA**

**BRUNO DE AZEVEDO MENDONÇA**

**ORIENTADOR: PROF. DR. PAULO MATIAS**

São Carlos – SP

Março, 2021

**UNIVERSIDADE FEDERAL DE SÃO CARLOS**

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

**AUDITCHAIN: UM MECANISMO PARA  
ATESTAR A INTEGRIDADE DE *LOGS*  
BASEADO NA PROVA DE EXISTÊNCIA EM  
*BLOCKCHAIN* PÚBLICA**

**BRUNO DE AZEVEDO MENDONÇA**

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Sistemas Distribuídos e Redes de Computadores.

Orientador: Prof. Dr. Paulo Matias

São Carlos – SP

Março, 2021



# UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia  
Programa de Pós-Graduação em Ciência da Computação

---

## Folha de Aprovação

---

Defesa de Dissertação de Mestrado do candidato Bruno de Azevedo Mendonça, realizada em 12/04/2021.

### Comissão Julgadora:

Prof. Dr. Paulo Matias (UFSCar)

Prof. Dr. Helio Crestana Guardia (UFSCar)

Prof. Dr. Diego de Freitas Aranha (UA)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

## **AGRADECIMENTOS**

---

---

Agradeço a minha família, namorada e amigos por todo apoio. A meu orientador pelo suporte ao longo do projeto. A Dra. Kamila Rios e Ma. Vanessa Carvalho por me orientarem nos momentos difíceis. E a todos que direta, ou indiretamente, contribuíram para o sucesso deste trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

# RESUMO

Dados digitais são de extrema importância para pessoas e empresas, servindo como peça chave na tomada de decisões em diferentes áreas. Por esse motivo, atestar a integridade dos dados é primordial, pois grandes prejuízos podem decorrer de uma falsificação. O registro de *logs* pode servir como um rastro de alteração desses dados ao longo do tempo e é base para ações de auditoria. A abordagem tradicional para assegurar a integridade de *logs* é armazená-los em servidores bem guardados tanto do ponto de vista de segurança física quanto digital. No entanto, essa abordagem assume que é difícil explorar vulnerabilidades nesses servidores e que é possível confiar no administrador desses sistemas. Uma alternativa para não depender dessas premissas é distribuir a confiança e, nesse sentido, abordagens baseadas em *blockchain* têm se mostrado promissoras. Alguns métodos para a atestação da integridade de *logs* após o armazenamento já foram propostos na literatura, mas não foram encontrados trabalhos que identifiquem a Prova de Existência (PoE) como uma possível operação básica para a realização dessa tarefa. PoE consiste em enviar a uma *blockchain* pública o *hash* de um objeto, comprovando que o objeto existia antes do instante de tempo em que sua inclusão na *blockchain* foi registrada. Também não foram encontrados trabalhos com propostas para facilitar a aplicação prática da técnica, permitindo a ingestão de *logs* a partir de diversas fontes. Com isso, o objetivo principal deste trabalho é propor uma arquitetura capaz de integrar a realização de PoE em *blockchain* públicas ao Elasticsearch, uma ferramenta amplamente utilizada atualmente para manter indexados *logs* de diversas fontes. O método aplicado é o estudo de caso, considerado um método eficaz para investigar e compreender questões complexas em cenários do mundo real. O contexto abordado é o de técnicos de informática, em parceria com a Secretaria da Informática (SIn) da Universidade Federal de São Carlos, de modo a obter resultados que possam ser generalizados para outros contextos. Realizou-se uma análise comparativa entre Interfaces de Programação de Aplicações para PoE, resultando na escolha da OpenTimestamps para o desenvolvimento do protótipo de software. A última versão do protótipo desenvolvido foi colocada à prova atestando *logs* contidos no servidor Elasticsearch da SIn de maneira ininterrupta por 10 dias. Verificou-se o correto funcionamento do protótipo, além de resolução temporal compatível com a taxa de transações emitidas pelos servidores públicos da OpenTimestamps. Espera-se que o conhecimento obtido no desenvolvimento deste protótipo, documentado no presente trabalho, possa auxiliar na melhor compreensão de como a tecnologia *blockchain* pode contribuir para a atestação de *logs*, e como as técnicas envolvidas podem tornar-se mais acessíveis aos administradores de sistemas.

**Palavras-chave:** *Blockchain*. Integridade de dados. *Logs*. Prova de existência.

# ABSTRACT

Digital data are critical to people and companies, acting as a crucial element in the decision-making process in different areas. Thus, attesting to data integrity is crucial because forgery can result in significant losses. Logs can track how this data changes over time and are essential to enable auditing. The traditional approach to ensure log integrity is to store them on well-kept servers, both from a physical and a digital security standpoint. However, this approach assumes that it is difficult to exploit these servers' vulnerabilities and that these systems' administrators can be trusted. Distributing the trust is an alternative that does not rely on these assumptions, and blockchain-based approaches are promising in that aspect. Even though the literature has already proposed some methods for attesting logs' integrity after their storage, we found no works that identify Proof of Existence (PoE) as a potential basic block for carrying that task. PoE consists of sending the hash of an object to a public blockchain, proving that the object existed before its hash became included in the blockchain. Furthermore, we found no works that paid attention to ease the technique's practical use by allowing for the ingestion of logs from different sources. Thus, this work's primary goal is to propose an architecture capable of integrating PoE on public blockchains with Elasticsearch, a tool widely used today for indexing logs from different sources. We applied the case study method, an effective method to investigate and understand complex issues in real-world scenarios. The context addressed is that of computer technicians, in partnership with the Secretariat of Informatics (SIn) of the Federal University of São Carlos, to obtain results that can be generalized to other contexts. We conducted a comparative analysis between PoE Application Programmer Interfaces, resulting in the choice of OpenTimestamps for the development of our software prototype. We tested the last version of our prototype by attesting logs stored in SIn's Elasticsearch service uninterruptedly for 10 days. We verified that the prototype was working correctly and presented a temporal resolution compatible with the transaction fulfillment rate sustained by public OpenTimestamps servers. We hope the knowledge obtained through the development of our software prototype, as documented in this work, will help to foster an understanding of how blockchain technology may contribute to log attestation and to raise awareness of the related techniques amongst system administrators.

**Keywords:** Blockchain. Data integrity. Logs. Proof of existence.

## LISTA DE SIGLAS

---

---

API	Application Programmer Interface
CA	Certification Authority
CLI	Command Line Interface
ECDSA	Elliptical Curve Digital Signature Algorithm
GCP	Google Cloud Platform
IDE	Integrated Development Environment
HSM	Hardware Security Module
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
JDK	Java Development Kit
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
NIST	National Institute of Standards and Technology
ORM	Object-Relational Mapping
P2P	Peer-to-peer
PBFT	Practical Byzantine Fault Tolerance
PoE	Proof of Existence
PoS	Proof of Stake
PoW	Proof of Work
REST	Representational State Transfer

SHA	Secure Hash Algorithms
SIn	Secretaria de Informática
SQL	Structured Query Language
TPM	Trusted Platform Module
UFSCar	Universidade Federal de São Carlos
USP	Universidade de São Paulo
UTXO	Unspent Transaction Outputs
USD	United States Dollar
XOF	Extendable-Output Functions

## LISTA DE FIGURAS

---

---

Figura 1 – Esquema de comunicação cifrada . . . . .	21
Figura 2 – Função de <i>Hash</i> . . . . .	23
Figura 3 – Árvores de Merkle . . . . .	24
Figura 4 – <i>Blockchain Framework</i> . . . . .	26
Figura 5 – Armazenamento descentralizado na <i>blockchain</i> . . . . .	27
Figura 6 – Porcentagem de Transações por Preço do Combustível no Ethereum . . . . .	29
Figura 7 – Média de Custo por Transação na Bitcoin e Ethereum (\$) . . . . .	30
Figura 8 – Transações na <i>Blockchain</i> . . . . .	31
Figura 9 – Estrutura de dados nas transações do Bitcoin . . . . .	32
Figura 10 – Blocos encadeados por <i>timestamps</i> . . . . .	33
Figura 11 – Ramificações na <i>blockchain</i> . . . . .	33
Figura 12 – Estrutura do bloco . . . . .	34
Figura 13 – Estrutura dos dados na <i>blockchain</i> . . . . .	35
Figura 14 – Elementos chaves da <i>blockchain</i> . . . . .	36
Figura 15 – Cenários de aplicações da <i>blockchain</i> . . . . .	38
Figura 16 – Recibo de uma PoE realizada pela API ChainPoint e Árvore de Merkle . . . . .	42
Figura 17 – Realização de PoE utilizando APIs com árvores de Merkle . . . . .	43
Figura 18 – Rede Chainpoint . . . . .	52
Figura 19 – Servidor de calendário OpenTimestamps Alice . . . . .	56
Figura 20 – <i>Hash rate</i> da Bitcoin nos últimos doze meses . . . . .	63
Figura 21 – <i>Hash rate</i> da Ethereum nos últimos doze meses . . . . .	63
Figura 22 – <i>Hash rate</i> da Aion nos últimos doze meses . . . . .	63
Figura 23 – Visão Geral do Fluxo de Dados na Auditchain . . . . .	64
Figura 24 – Arquitetura da Aplicação Auditchain . . . . .	66
Figura 25 – Fluxograma do comando stamp-elasticsearch . . . . .	68
Figura 26 – Execução do comando stamp-elasticsearch . . . . .	69
Figura 27 – Fluxograma do comando verify-elasticsearch . . . . .	70
Figura 28 – Execução do comando verify-elasticsearch . . . . .	71
Figura 29 – Tempo entre transações Bitcoin do OpenTimestamps em janeiro de 2021 . . . . .	75
Figura 30 – Tempo entre transações Bitcoin do OpenTimestamps em fevereiro de 2021 . . . . .	76
Figura 31 – Tempo entre transações Bitcoin do OpenTimestamps em março de 2021 . . . . .	77

## LISTA DE TABELAS

---

---

Tabela 1 – Análise das APIs - Visão geral . . . . .	60
Tabela 2 – Estatísticas do tempo entre transações da OpenTimestamps . . . . .	78

# SUMÁRIO

---

---

<b>CAPÍTULO 1–INTRODUÇÃO</b>	<b>12</b>
1.1 Contexto	12
1.2 Revisão de Literatura	14
1.3 Objetivos	16
1.4 Síntese da Metodologia	17
1.5 Resultados	18
1.6 Organização do Trabalho	18
<b>CAPÍTULO 2–BLOCKCHAIN E POE</b>	<b>20</b>
2.1 Conceitos Preliminares	20
2.1.1 Criptografia	20
2.1.2 Hash Criptográfico	22
2.1.3 Árvore de Merkle	23
2.2 Visão Geral	24
2.3 Rede Ponto-a-Ponto	25
2.4 Consenso	27
2.5 Transações	29
2.5.1 Cadeia de Blocos	31
2.5.2 Estrutura dos Blocos	34
2.6 Identidade e Contratos inteligentes	35
2.7 Conclusões sobre núcleo da <i>blockchain</i>	36
2.8 Aplicações e PoE	37
2.8.1 Prova de Existência	38
<b>CAPÍTULO 3–METODOLOGIA</b>	<b>44</b>
3.1 Materiais	44
3.2 Procedimentos	46
3.2.1 Estudo de caso	46
3.2.2 Análise sobre APIs de realização de PoE	47
3.2.3 Aplicação desenvolvida: Auditchain	48
<b>CAPÍTULO 4–DESENVOLVIMENTO</b>	<b>51</b>
4.1 Análise das APIs	51
4.1.1 Chainpoint	51
4.1.2 OpenTimestamps	54
4.1.3 OriginStamp	58

4.1.4	Comparativo . . . . .	59
4.1.5	Análise de maturidade das <i>blockchains</i> . . . . .	61
4.2	Auditchain . . . . .	62
4.2.1	Discussão acerca do modelo adversarial . . . . .	71
<b>CAPÍTULO 5–RESULTADOS . . . . .</b>		<b>74</b>
5.1	Tempo entre transações da OpenTimestamps . . . . .	74
5.2	Aproveitamento de transações para ancoragem de PoEs . . . . .	79
5.3	Sumário qualitativo . . . . .	79
<b>CAPÍTULO 6–CONCLUSÕES . . . . .</b>		<b>81</b>
<b>REFERÊNCIAS . . . . .</b>		<b>83</b>

# Capítulo 1

## INTRODUÇÃO

---

---

### 1.1 Contexto

Dados digitais são de extrema importância para pessoas e empresas, atuando como peça chave na tomada de decisões em diferentes áreas como finanças, saúde, educação e administração pública (GAETANI et al., 2017). Por consequência, a adulteração de dados destacou-se como uma das maiores ameaças cibernéticas dos últimos anos, pois a influência de agentes maliciosos a dados científicos, comerciais ou do governo pode gerar grandes prejuízos a corporações e a toda uma nação (KALIS; BELLOUM, 2018). Lidar com essa ameaça é um dos grandes desafios envolvendo arquivos e dados que são confiados a serviços online.

Em serviços online, os dados são controlados e armazenados por seus provedores e podem ser manipulados sem autorização prévia do proprietário, sobretudo em estruturas de armazenamento centralizadas que têm limitações nos aspectos de segurança quanto a falsificações, perda e roubo de dados (XU et al., 2017). Dessas limitações se destacam (1) o problema de ponto único de falha, pois há um único servidor responsável pelo armazenamento de dados que, caso seja invadido, pode incidir em danos irreparáveis aos dados; e (2) a necessidade de um terceiro confiável para realização do armazenamento, visto que é necessário confiar no provedor, que possui acesso total aos dados armazenados e pode manipulá-los como desejar (WANG et al., 2018). Uma das maneiras de checar a integridade dos dados é por meio do registro de *logs*.

*Logs* são considerados documentos que podem evidenciar a qualidade de um serviço, sendo essenciais para investigações na área da computação. Há sistemas de banco de dados atualmente que permitem realizar o registro de *logs* de cada alteração realizada nos dados, possibilitando até mesmo sua restauração. Com o registro correto de *logs* é possível identificar quando e quem manipulou os dados para que o autor seja responsabilizado e as alterações sejam revertidas, caso necessário. Devido aos *logs* serem dados valiosos, geralmente sabe-se da importância de manter backups dos mesmos. No entanto, ainda é necessário confiar que as pessoas que possuem acesso a esses dados não comprometerão sua integridade. Isso ocorre principalmente nos casos em que o armazenamento dos *logs* é igualmente realizado em estruturas centralizadas, geralmente no próprio provedor, com o agravante de que empresas podem ter

maior interesse em forjar *logs* que revelem falhas técnicas ou operacionais para poupar-se de multas em uma possível auditoria do sistema (POURMAJIDI; MIRANSKY, 2018; ANIELLO et al., 2017).

Em auditorias de sistemas são coletadas e avaliadas evidências buscando-se certificar, dentre outras coisas, a integridade dos dados armazenados. Atualmente, grande parte das auditorias são realizadas por empresas que atuam como terceiro confiável, sem possuir vínculos com usuário ou provedor, buscando assegurar uma avaliação imparcial. Dos pontos negativos desse tipo de auditoria estão o alto custo para sua realização e a dificuldade em se encontrar um terceiro confiável, pois o conluio com os auditores é uma das formas de empresas ocultarem suas fraudes (RAVI; SUNITHA, 2017; ABREU et al., 2018). Desse modo, a necessidade de um terceiro confiável e seus entraves torna-se recorrente, e é visando reduzir essas limitações que diversas pesquisas vêm sendo realizadas tendo como base a tecnologia *blockchain*.

Introduzida por Nakamoto et al. (2008) com a criptomoeda Bitcoin, *blockchain* destacou-se por ser uma estrutura de armazenamento distribuída em nós sem controle central, imutável, transparente e consistente. Todos os nós possuem uma cópia dos dados e, para proteger a integridade na rede, a inserção de novos dados é realizada em conformidade com o consenso da maioria dos participantes. Após inserida, uma informação nunca pode ser apagada ou alterada, assim os dados são monitorados por todos e controlados por ninguém, assegurando que nenhuma pessoa, empresa ou mesmo governo possa forjá-los facilmente (ABREU et al., 2018; ZIKRATOV et al., 2017). Basicamente, há três tipos de *blockchains*: pública, de consórcio (ou federada) e privada. Todas possuem o mesmo conceito mas com algumas diferenças de funcionalidade e implementação (ALKETBI et al., 2018).

Em *blockchains* públicas, como a Bitcoin, qualquer pessoa pode fazer parte da rede e do processo de consenso, aumentando a segurança dos dados devido ao alto custo para tomar controle do consenso da rede. Em contrapartida, esse tipo de *blockchain* possui as limitações de baixa taxa de transferência e alta latência, o que torna a rede extremamente lenta, além de ser necessário pagar cada transação que se deseja realizar. Por outro lado, em *blockchains* de consórcio ou privadas há maior flexibilidade na implementação porque somente nós selecionados fazem parte do processo de consenso, entretanto há necessidade de confiança nestes nós pois, se uma maioria for maliciosa, ela pode tomar o controle da rede e comprometer a integridade dos dados (ZHENG et al., 2017).

Devido a essas características, *blockchain* viabiliza processos para que duas partes que não se conhecem (usuário e provedor, por exemplo) possam se assegurar da autenticidade dos dados com menos necessidade de confiança em um terceiro. Tanto provedor quanto usuário se beneficiam pela possibilidade de auditar os dados com maior clareza e imparcialidade. Tal qualidade é desejável sobretudo em sistemas governamentais, por contribuir para reduzir custos operacionais e conter fraudes com maior transparência entre governo e civis (LIU; XU, 2018; KALIS; BELLOUM, 2018; ALKETBI et al., 2018). À vista disso, diversos estudos vêm sendo

realizados acerca da integridade de dados com base nessa tecnologia.

## 1.2 Revisão de Literatura

Renner et al. (2018) apresentam um *framework* para auditoria de arquivos armazenados em estruturas de computação em nuvem utilizando *blockchain* Ethereum. Seu *framework* coleta e envia à *blockchain* metadados sobre alterações realizadas em arquivos que o usuário deseja monitorar, e com isso é possível detectar se algum deles foi modificado sem a autorização do proprietário. Por utilizar uma *blockchain* pública, reduz-se necessidade de um terceiro confiável. Entretanto, devido alta latência e baixa taxa de transferência, além da necessidade de se pagar por cada uma das transações, esse *framework* é indicado somente para monitorar arquivos que sofrem poucas modificações, caso contrário sua utilização torna-se inviável.

Xu et al. (2017) propõem uma aplicação *blockchain* de certificados educacionais na qual buscam atender as demandas de baixa latência e alto rendimento utilizando algoritmo de consenso por cooperação e estrutura de dados em árvore. Pourmajidi e Miranskyy (2018) fazem uso de uma *blockchain* hierárquica para armazenar *logs* que são coletados de diferentes fontes, buscando evitar a adulteração destes. Yang et al. (2018) armazenam na *blockchain* informações sobre a remoção de arquivos armazenados em nuvem, sendo possível validar se o arquivo foi ou não apagado com o consentimento do proprietário. Kumar e Sunitha (2017) utilizam o Hyperledger Fabric<sup>1</sup> para preservar a integridade de dados de servidores de localização. Todos esses trabalhos utilizam *blockchains* privadas, o que geralmente implica que haverá uma quantidade menor de nós, controlados também por uma quantidade menor de entidades, o que reduz a segurança da integridade das informações em comparação com *blockchains* públicas.

Aniello et al. (2017) avaliam um protótipo de *blockchain* de duas camadas, uma privada e uma pública, para registro de *logs* de bancos de dados em redes federadas. Os dados são armazenados primeiramente na *blockchain* privada, por esta possuir um algoritmo de consenso rápido, e periodicamente são gerados *hashes* dos dados de um dos nós dessa camada para serem enviados à *blockchain* pública, sendo possível provar a integridade dos dados da *blockchain* privada em determinado momento do tempo. Essa proposta aumenta a segurança sobre a realização de ataques pelo fato de em um curto espaço de tempo os *logs* serem inseridos em uma rede descentralizada. No entanto, por utilizar uma camada de *blockchain* privada, ainda há necessidade de confiança nos nós desta rede. Caso os dados nesta camada estejam comprometidos, o *hash* enviado à *blockchain* pública também estará. Além disso, não há como recuperar dados da *blockchain* pública, somente validá-los. O ato de gerar *hashes* de um conjunto de dados e enviá-los à *blockchain* para comprovar a integridade dos dados é análogo à operação de prova de existência (PoE)<sup>2</sup>, muito embora o trabalho citado não empregue essa nomenclatura e utilize sua própria implementação da técnica.

<sup>1</sup> <<https://www.hyperledger.org/projects/fabric>>

<sup>2</sup> Também análogo a um *commitment*

Explicitar o uso de PoE pelo método de atestação de *logs* tem o potencial de levar a um melhor reaproveitamento de recursos e de código, portanto foram pesquisados também trabalhos relacionados à técnica de PoE.

[Meneghetti et al. \(2019\)](#) propõem um design de PoE com segurança incremental também com uso de duas camadas de *blockchain*, privada e pública. Realiza-se a PoE primeiro na *blockchain* privada, que possui retorno mais rápido porém menos confiança, e posteriormente na *blockchain* pública. Essa proposta se difere da proposta de [Aniello et al. \(2017\)](#) por não ter a *blockchain* privada como intermediária. Assim, mesmo que a camada privada esteja corrompida, os dados enviados à *blockchain* pública não estarão. Contudo, isso torna a camada de *blockchain* privada não primordial, já que a PoE considerada legítima é a da *blockchain* pública. Seria necessário um complicado processo de auditoria caso houvesse PoEs diferentes para o mesmo conjunto de dados em ambas as camadas.

[Kalis e Belloum \(2018\)](#) apresentam uma proposta de aplicação de PoE (mesmo não o nomeando dessa forma) em qualquer dado arbitrário utilizando somente *blockchain* Ethereum. Assume-se que os dados reais são armazenados separadamente, possibilitando validá-los com custo moderado de transações e menos necessidade de um terceiro confiável, embora não seja possível recuperar dados perdidos ou corrompidos. [Cheng et al. \(2018\)](#) fazem uso da PoE buscando atestar a integridade de certificados educacionais. [Gao e Nobuhara \(2017\)](#) propõem um meio de expandir o limite de dados enviados à *blockchain* Bitcoin, de forma a permitir armazenar metadados relacionados a uma PoE em uma mesma transação, e citam Interfaces para Programação de Aplicativos (API) que realizam PoE.

Atualmente há diferentes APIs para realização de PoEs. Algumas dessas APIs fazem uso de uma estrutura em árvore de Merkle ([MERKLE, 1980](#)) com objetivo de reduzir a quantidade de transações na *blockchain*. Em vez de realizar a PoE de cada *hash* recebido, essas APIs concatenam *hashes* de diferentes clientes e a cada determinado intervalo de tempo realizam a PoE somente da raiz de Merkle. Após a PoE ser realizada, clientes podem baixar os ramos necessários para reconstrução da raiz de Merkle a partir de seu *hash* e realizar a validação diretamente na *blockchain*. Esse processo auxilia na redução de transações na *blockchain* e diluição de custos entre diferentes clientes. [Liang et al. \(2017\)](#) e [Zhang et al. \(2017\)](#) utilizam a API Chainpoint<sup>3</sup> para realizar a PoE de metadados provenientes de arquivos armazenados em nuvem. [Schönhals et al. \(2018\)](#) utilizam a API OriginStamp<sup>4</sup> para realizar a PoE de ideias que surjam em processos de *Design Thinking*, com intenção de garantir sua propriedade intelectual. [Hepp et al. \(2018b\)](#) empregam essa mesma API para realização de PoE de dados em cadeias de suprimentos.

Após análise do estado da arte conclui-se que há grande uso da tecnologia *blockchain* buscando assegurar a integridade de dados provenientes de diversas fontes, tais como arquivos

<sup>3</sup> <<https://www.chainpoint.org>>

<sup>4</sup> <<https://www.originstamp.org>>

em nuvem, dados educacionais, cadeias de suprimento, dentre outras. Para reduzir a necessidade de confiança em terceiros é recomendada a utilização de *blockchains* públicas robustas, como Bitcoin ou Ethereum. Contudo, é inviável o envio e armazenamento de grandes volumes de dados em transações devido às limitações enfrentadas em *blockchains* públicas. Nesse sentido, a PoE aplicada juntamente a árvores de Merkle apresenta-se como uma opção viável para atestar a integridade de *logs*. Armazenar somente a raiz da árvore de Merkle é suficiente para comprovar a integridade de todos os *hashes* concatenados, viabilizando auditar *logs* de diversas fontes com custo de transações baixo e fixo.

### 1.3 Objetivos

Segundo Ravi e Sunitha (2017), uma boa auditoria deve permitir verificação rápida, pública e ilimitada, viabilizando baixo custo de comunicação e preservando a privacidade dos dados. O método de realização de PoE em conjunto com árvores de Merkle proporciona processos de auditoria com essas características e existem APIs que o efetuam sem muitas complicações. No entanto, embora haja trabalhos que exploram PoE para outras aplicações, não foram encontrados trabalhos que realizem atestação de *logs* utilizando PoE de forma explícita. Também não foram encontrados trabalhos que façam uma análise das APIs de realização de PoE disponíveis atualmente, tampouco trabalhos focados na integração desse método com dados provenientes de diversas fontes. Com isso, levanta-se a problemática de como simplificar a aplicação da técnica por equipes de técnicos de informática que empregam o registro de *logs* de seus dados.

Uma ferramenta amplamente utilizada para indexação de *logs* atualmente é o Elasticsearch, um mecanismo para pesquisa e análise de dados quase em tempo real. Essa ferramenta é capaz de realizar indexação e armazenamento de diversos tipos de dados, oferecendo suporte a pesquisas rápidas e agregação de informações. É muito utilizada como base para análise de *logs*, métricas e dados de eventos de segurança por possuir uma máquina de busca robusta para lidar com esses tipos de dados. A companhia disponibiliza ferramentas como Filebeat<sup>5</sup> e Logstash<sup>6</sup> para coleta de *logs* de diversas fontes. Além disso, há ferramentas de terceiros com suporte ao Elasticsearch, como o Fluentd<sup>7</sup> e o Fluent Bit<sup>8</sup>. Com essas ferramentas é possível indexar no mesmo servidor Elasticsearch *logs* gerados pelo Apache ou log4j, *logs* provenientes de arquivos, requisições HTTP, registros do mecanismo de auditoria do *kernel* Linux, dentre outros (ELASTICSEARCH B.V., 2021). Com isso, assegurar a integridade de *logs* indexados em um servidor Elasticsearch representa assegurar a integridade de *logs* provenientes de diversas outras fontes.

<sup>5</sup> <https://www.elastic.co/pt/beats/filebeat>

<sup>6</sup> <https://www.elastic.co/pt/logstash>

<sup>7</sup> <https://www.fluentd.org>

<sup>8</sup> <https://fluentbit.io>

Em vez de armazenar informações como linhas de dados em colunas, o Elasticsearch armazena estruturas de dados complexas que são serializadas como documentos em Notação de Objetos JavaScript (JSON). Quando um documento é armazenado, ele é indexado e totalmente pesquisável dentro de 1 segundo. Um índice pode ser considerado uma coleção otimizada de documentos e cada documento é uma coleção de campos, que são os pares de chave-valor que contêm seus dados (ELASTICSEARCH B.V., 2021). A indexação de um *log*, por exemplo, pode ser um documento contendo o *log* em si e também informações a respeito da máquina ou sistema que o gerou. Cada novo documento indexado no Elasticsearch recebe um carimbo de data e hora do momento em que foi indexado. Esse carimbo de data e hora possibilita obter documentos indexados em um determinado intervalo de tempo de forma assertiva, assegurando que os mesmos documentos serão retornados para aquele intervalo, mesmo após a inserção de novos documentos.

Assim, este trabalho teve como base dois objetivos principais: (1) realizar a análise de três APIs de realização de PoE disponíveis atualmente: Chainpoint, OpenTimestamps<sup>9</sup> e OriginStamp; e (2) desenvolver uma arquitetura capaz de integrar a realização de PoE em *blockchains* públicas à ferramenta Elasticsearch, de modo a poder atestar a integridade de *logs* indexados com essa ferramenta. Essa arquitetura buscou avançar na proposta feita por Kalis e Belloum (2018) através da utilização de APIs que realizam a PoE com árvores de Merkle e da integração com a ferramenta Elasticsearch para atestação de *logs* provenientes de diversas fontes. Partiu-se da hipótese de que haverá maior segurança quanto a integridade dos dados se houver uma arquitetura que facilite a realização de PoE de *logs* de maneira integrada a uma plataforma amplamente utilizada. O alcance do objetivo geral passou pelo cumprimento das seguintes tarefas:

- Aprofundamento do conhecimento na tecnologia *blockchain* e no método de PoE
- Análise das APIs Chainpoint, OpenTimestamps e OriginStamp
- Criação de uma arquitetura para integração do método de PoE à plataforma Elasticsearch
- Desenvolvimento de um estudo de caso para a arquitetura proposta com a criação de uma ferramenta de Interface de Linhas de Comando (CLI)
- Testes e avaliações da ferramenta desenvolvida
- Comparação dos resultados com trabalhos levantados na revisão da literatura

## 1.4 Síntese da Metodologia

Do modo como propomos no presente trabalho, as premissas necessárias para possibilitar a atestação da integridade dos dados são:

<sup>9</sup> <<https://opentimestamps.org/>>

- que os sistemas ou serviços possuam registro correto de *logs*, rastreando todas as operações que modifiquem seus dados
- que os *logs* estejam armazenados em algum lugar para poderem ser validados (o método proposto armazenará apenas *hashes*)
- que os *logs* não sejam forjados antes da realização da PoE

Além disso, a PoE é utilizada com o único intuito de atestar os dados, sendo possível somente verificar se eles foram forjados e quando o foram, mas sem a possibilidade de recuperá-los. Se houver a necessidade de recuperação de dados, esta deverá ser assegurada por meio de mecanismos de *backup* com um nível de redundância adequado, que não abordaremos aqui. Por fim, enfatizamos que uma auditoria com base na tecnologia *blockchain* pode oferecer segurança jurídica somente se assim for regulamentado na jurisdição onde ela é aplicada.

O método de pesquisa é o estudo de caso, considerado um método eficaz para investigar e compreender questões complexas em cenários do mundo real por meio de uma análise aprofundada de um problema em um contexto específico, com a perspectiva dos participantes, de forma mais exploratória do que confirmatória. A partir desse método, realizamos uma análise aprofundada sobre atestação da integridade de dados utilizando *blockchain* no contexto de equipes de técnicos de informática, com o auxílio da secretaria de informática (SIn) da Universidade Federal de São Carlos (UFSCar). Adotamos procedimentos de análise qualitativos e quantitativos por meio de análises estatísticas, análises de documentos e análises de trabalhos anteriores para uma maior generalização dos resultados e buscando minimizar os níveis de subjetividade (HARRISON et al., 2017; HANCOCK; ALGOZZINE, 2016).

## 1.5 Resultados

Desenvolveu-se um protótipo de software denominado Auditchain para atestar *logs* realizando PoEs com a API da OpenTimestamps. A última versão do protótipo foi capaz de atestar e validar um índice do servidor Elasticsearch da SIn quando executada por 10 dias ininterruptos. A resolução temporal obtida foi compatível com a taxa de transações emitidas pelos servidores públicos da OpenTimestamps. Além disso, constatou-se que 96% de todas as transações emitidas por esses servidores durante o período dos testes continham PoEs solicitadas pelo protótipo.

## 1.6 Organização do Trabalho

Este trabalho está organizado em seis capítulos. O primeiro compreende a introdução, a qual já foi apresentada. O segundo capítulo detalha o funcionamento da tecnologia *blockchain* e o método de PoE, abordando conceitos acerca de criptografia, estrutura de dados na *blockchain* e

---

sua arquitetura de rede e, por fim, como *blockchain* se relaciona com o método de PoE. O terceiro capítulo apresenta com detalhes a metodologia utilizada no desenvolvimento deste projeto, indicando os materiais utilizados e procedimentos realizados com base no método de estudo de caso. No quarto capítulo é apresentado o desenvolvimento deste trabalho, o qual contém a análise e comparação das APIs de realização de PoE e arquitetura para integração do método de PoE à ferramenta Elasticsearch. No quinto capítulo são apresentados os resultados acerca dos testes realizados, comparando-os com a revisão da literatura. Por fim, no sexto capítulo são apresentadas as conclusões deste trabalho e recomendações para trabalhos futuros.

# Capítulo 2

## BLOCKCHAIN E PoE

---

---

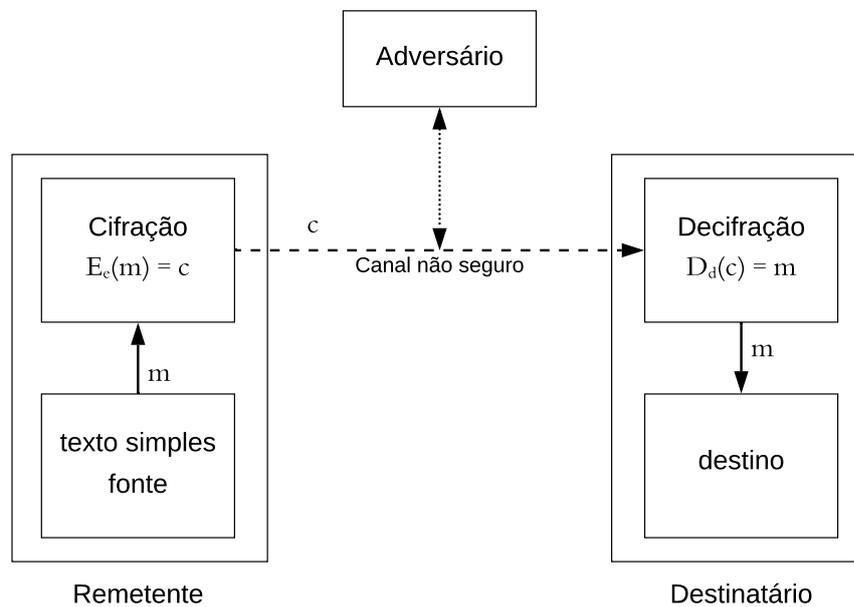
Neste capítulo são apresentados os principais conceitos por trás do funcionamento da *blockchain* em uma abordagem *bottom-up*. Inicialmente, são revisados conceitos preliminares acerca de segurança da informação, em especial os relacionados à base da arquitetura da *blockchain*. Em seguida, a tecnologia em questão é detalhada com base nas camadas propostas por [Concordium \(2019\)](#). São abordados o funcionamento da rede distribuída ponto-a-ponto e algoritmos de consenso, esclarecendo a forma como *blockchain* organiza-se em uma arquitetura descentralizada sem a necessidade de um nó central. Detalha-se a estrutura das transações e cadeia de blocos, de modo a se fazer entender como *blockchain* assegura a imutabilidade e integridade dos dados, por meio de assinaturas digitais, *timestamps* e funções de *hash*. Os conceitos de identidade e contratos inteligentes são apresentados e, por fim, comenta-se a respeito das aplicações em *blockchain*, relacionando e detalhando o método de PoE com essa tecnologia.

## 2.1 Conceitos Preliminares

### 2.1.1 Criptografia

Etimologicamente, o termo criptografia vem da junção das palavras “escrita” e “escondida”. Por mais que hoje a criptografia envolva um conjunto muito maior de técnicas, a cifração continua uma das operações mais emblemáticas da área. Cifrar dados consiste em mascará-los de modo que fiquem compreensíveis apenas para quem possa restaurá-los em sua forma original. Essa operação é aplicada quando diferentes pessoas comunicam-se por um meio inseguro para impedir que terceiros tenham acesso às informações compartilhadas ([PRIANGA et al., 2018](#)). Um esquema de comunicação de duas partes utilizando encriptação pode visto na [Figura 1](#). Há dois tipos de algoritmos amplamente usados que se baseiam na utilização de chaves para cifrar os dados: os algoritmos de chaves simétricas e os de chaves assimétricas. Basicamente, um algoritmo é considerado de chaves simétricas quando é eficiente calcular a chave de encriptação a partir da chave de decifração, enquanto que um algoritmo de chaves assimétricas é aquele onde não é eficiente calcular a chave de decifração a partir da chave de encriptação ([MENEZES et al., 1996](#)).

Figura 1 – Esquema de comunicação cifrada



Fonte: Adaptado de [Menezes et al. \(1996\)](#)

Um dos maiores problemas com sistemas de chaves simétricas é encontrar um método eficiente para trocar as chaves com segurança, já que a mesma chave precisa ser conhecida entre remetente e destinatário dos dados. Já com sistemas de chaves assimétricas é possível utilizar esquemas como criptografia de chave pública ou assinatura digital. Em esquemas de criptografia de chave pública, considera-se um par de chaves  $(e, d)$ , onde  $d$  é a chave secreta (ou privada) e  $e$  é a chave pública. Como o próprio nome diz, a chave pública  $e$  pode ser compartilhada publicamente, desde que seja compartilhada de maneira autenticada e a chave privada  $d$  seja mantida secreta e segura. Esse esquema permite que um remetente use a chave pública de um destinatário para cifrar a mensagem de forma que somente o destinatário possa decifrá-la usando sua chave privada, possibilitando que qualquer remetente compartilhe dados cifrados e somente o destinatário tenha acesso a eles.

Outra técnica criptográfica importante é a assinatura digital, essencial para autenticação, autorização e não repúdio de dados. Assim como assinaturas por escrito, as digitais devem ser verificáveis e não falsificáveis, de forma a provar que um documento foi verdadeiramente assinado por certa entidade (pessoa, organização, serviço, dentre outros). Como uma descrição genérica de assinatura, podemos considerar que uma entidade  $A$  (o signatário) assina uma mensagem  $m$  calculando  $s = S_A(m)$ , onde  $S_A$  é o algoritmo de assinatura de  $A$  e  $s$  é a assinatura dos dados. Com isso, caso uma entidade  $B$  (o verificador) queira verificar se uma assinatura  $s$  em uma mensagem  $m$  foi criada por  $A$ , basta que ele obtenha a função de verificação de  $A$  ( $V_A$ )

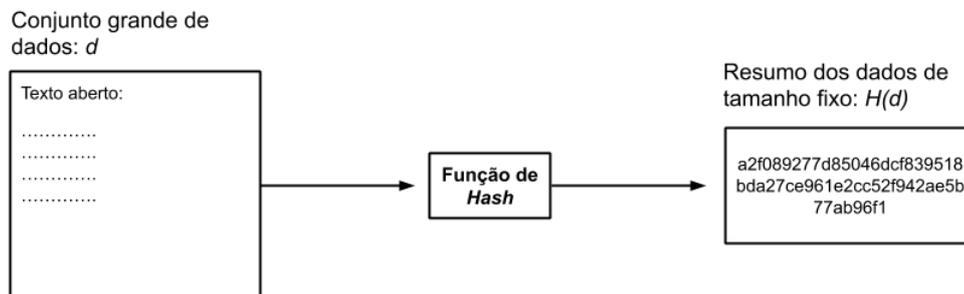
e calcule  $u = V_A(m, s)$ . Caso  $u$  seja verdadeiro, o verificador  $B$  pode considerar a assinatura como criada pela entidade  $A$  (MENEZES et al., 1996; KUROSE; ROSS, 2012).

As transformações  $S_A$  e  $V_A$  são tipicamente caracterizadas de forma mais compacta por um par assimétrico de chaves. Desse modo, o algoritmo de assinatura  $S_A$  de  $A$  é determinado por uma chave  $k_A$  e  $A$  precisa apenas para manter  $k_A$  em segredo. Da mesma forma, o algoritmo de verificação  $V_A$  de  $A$  é determinado por uma chave  $l_A$  que é tornada pública (MENEZES et al., 1996). Para certificar que os dados foram de fato assinados pelo remetente, o destinatário deve ser capaz de verificar se possui sua chave pública verdadeira. Uma das formas de fazê-lo sem necessitar de um canal seguro com o remetente é utilizando um terceiro confiável, neste caso denominado Autoridade Certificadora (CA). Uma CA possui as funções de validar identidades por meio de verificações rigorosas e atribuir certificados que vinculem a chave pública à entidade validada, assegurando que as chaves públicas divulgadas realmente pertencem a suas entidades. As chaves públicas das CAs são amplamente divulgadas e geralmente vem junto com os pacotes de instalação de sistemas operacionais e navegadores web (KUROSE; ROSS, 2012). Por último, é custoso operar com grandes volume de dados utilizando algoritmos de chave assimétrica, e por esse motivo a assinatura digital geralmente é realizada em conjunto com operações de *hash*.

### 2.1.2 Hash Criptográfico

Como ilustrado na Figura 2, funções de *hash* recebem como entrada qualquer conjunto de dados e calculam uma cadeia de tamanho fixo como saída ( $H(d)$ ). *Hashes* criptográficos seguem essa mesma definição com a adição de que, em tempo computacionalmente viável, é impraticável encontrar duas entradas diferentes  $x$  e  $y$  tais que  $H(x) = H(y)$  ou determinar  $z$  caso somente  $H(z)$  seja conhecido. Mais especificamente, uma boa função de *hash* criptográfica deve atender às seguintes propriedades: (1) ser determinística, garantindo exatamente o mesmo *hash* para a mesma entrada; (2) gerar o valor de *hash* rapidamente; (3) ser impraticável, em tempo computacionalmente viável, encontrar qual entrada gerou um valor de *hash*; (4) uma pequena alteração nos dados de entrada deve gerar um valor de *hash* completamente diferente; e (5) dois conjuntos de dados diferentes jamais devem ter o mesmo valor de *hash* (DEBNATH et al., 2017; KUROSE; ROSS, 2012).

Assim, funções de *hash* criptográficas são resistentes a colisões, fazendo com que cada valor de *hash* seja considerado único. Além disso, elas são de mão única, de modo que é inviável reverter um valor de *hash* para seus dados originais, assumindo distribuição uniforme das entradas. Com isso, *hashes* criptográficos são amplamente utilizados para validar o conjunto de dados que os originou ou como identificador de seus dados (ALKETBI et al., 2018; GAO; NOBUHARA, 2017). Dos algoritmos de *hash* criptográfico atualmente recomendados destacam-se os Algoritmos de *Hash* Seguros (SHA), mais especificamente SHA-2 e SHA-3, que são parte de uma família de algoritmos padronizados pelo Instituto Nacional de Padrões e Tecnologia (NIST) dos Estados Unidos.

**Figura 2 – Função de Hash**

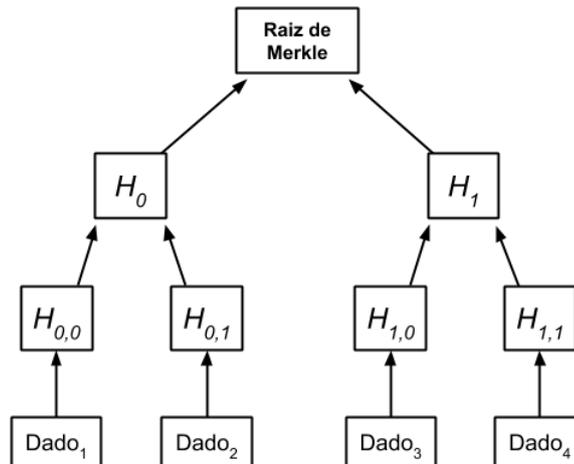
Fonte: adaptado de [Kurose e Ross \(2012\)](#)

O conjunto de algoritmos SHA-2 baseia-se na construção de Merkle–Damgård aplicada sobre uma função de mão única criada com a estrutura de Davies–Meyer, e consiste em seis funções *hash* com diferentes tamanhos de resultado – 224, 256, 384 ou 512 bits – comumente chamados de SHA-224, SHA-256, SHA-384 e SHA-512, respectivamente. A família SHA-3 consiste em quatro funções hash criptográficas, chamadas SHA3-224, SHA3-256, SHA3-384 e SHA3-512, e duas funções de saída extensível (XOF), chamadas SHAKE128 e SHAKE256. Cada uma das funções SHA-3 é baseada em uma instância do algoritmo Keccak. Dada uma mensagem  $M$ , as quatro funções hash SHA-3 são definidas a partir da função Keccak[ $c$ ] anexando um sufixo de dois bits a  $M$  e especificando o comprimento da saída. Em cada caso, a capacidade  $c$  é o dobro do comprimento do resumo, ou seja,  $c = 2d$ , e a entrada  $N$  para Keccak[ $c$ ] é a mensagem com o sufixo anexado, ou seja,  $N = M||01$ . O sufixo serve para fazer separação de domínio, isto é, ele distingue as entradas para Keccak[ $c$ ] decorrentes das funções hash SHA-3 das entradas decorrentes das XOFs SHA-3 ([DWORKIN, 2015](#)).

### 2.1.3 Árvore de Merkle

Árvore de Merkle ([MERKLE, 1980](#)) é uma estrutura de dados em árvore binária na qual os dados são representados pelos nós folhas. Todo nó interno mantém o valor de *hash* da concatenação dos nós-filho esquerdo e direito. No topo da árvore, encontra-se a raiz de Merkle, que nada mais é que um identificador que representa a junção de todos os nós-filhos ([Figura 3](#)). Essa estrutura é usada para verificar a autenticidade de dados digitais reduzindo a sobrecarga de computação e comunicação, sendo uma maneira mais eficiente e segura de atestar a integridade grandes volumes de dados ([KUMAR; SUNITHA, 2017; PRIANGA et al., 2018; YANG et al., 2018](#)).

Figura 3 – Árvores de Merkle



Fonte: adaptado de [Meng et al. \(2018\)](#)

## 2.2 Visão Geral

*Blockchain* é equivalente a um banco de dados distribuído que, em vez de utilizar estruturas em tabelas relacionais, é organizado em uma lista ordenada de blocos de dados. Cada bloco contém um conjunto de novos registros de dados e é vinculado ao bloco anterior, formando uma estrutura de dados contínua similar a uma cadeia — por esse motivo o nome de cadeia de blocos (ou *block-chain*) ([ALKETBI et al., 2018](#)). Sua arquitetura é resultado da integração de múltiplas tecnologias já existentes, combinadas para manter um banco de dados único e confiável por meio de uma abordagem descentralizada. Seu diferencial está em assegurar a confiança na rede com base em premissas de dificuldade computacional em vez de usar um terceiro confiável como intermediário ([LIU; XU, 2018](#); [DAI et al., 2017](#)).

*Blockchain* utiliza princípios criptográficos com objetivo de garantir integridade e autenticidade dos dados, que são armazenados em forma de transações vinculadas umas às outras. Essas transações são registradas de forma cronológica e imutável, sendo que todos os participantes validam os registros seguindo algoritmos de consenso. Algoritmos de consenso são utilizados com objetivo de garantir que as transações sejam efetivadas em conformidade com a maioria da rede. Com isso, *blockchain* assume a presença de agentes maliciosos infiltrados na rede e anula suas estratégias por meio dos participantes honestos, fazendo dessa uma tecnologia indicada para operar em ambientes altamente competitivos ([MENG et al., 2018](#); [LIANG et al., 2017](#)).

A rede *blockchain* consiste em uma arquitetura ponto-a-ponto (P2P) com múltiplos nós, em que cada nó possui uma cópia local dos dados da rede. A cada transação validada, os nós participantes coletivamente mantêm o registro de dados atualizado, dessa maneira todos os participantes são capazes de compartilhar os dados sem que nenhuma entidade possa controlá-los. Algumas implementações de *blockchain* ainda permitem armazenar e executar programas em sua

estrutura, chamados de Contratos Inteligentes. Esses contratos são utilizados para automatizar processos e formalizar normas entre duas ou mais partes envolvidas, trabalhando sem inatividade, censura, fraude ou interferência de terceiros (LIANG et al., 2017; RENNER et al., 2018).

Alketbi et al. (2018) apresenta a evolução da *blockchain* em cinco estágios: (1) Iniciou com *blockchain* como registro distribuído de transações sem uma autoridade central, introduzido por Nakamoto et al. (2008) e aplicado pela primeira vez na Bitcoin, em 2009; (2) Devido ao código aberto, houve a construção de outras moedas digitais, resultando no reconhecimento e efetivação dessa tecnologia; (3) *Blockchain* expandiu-se para além das moedas digitais e começou a ser aplicada para gerenciar transações de bens e serviços; (4) Com a expansão da *blockchain* e de sua confiabilidade, ela passou a ser usada também como registro de propriedade e direitos autorais, garantindo autenticidade de dados; (5) Por fim, o último estágio está voltado para a automação de processos a partir de programas que são armazenados e executados pela *blockchain*, os contratos inteligentes.

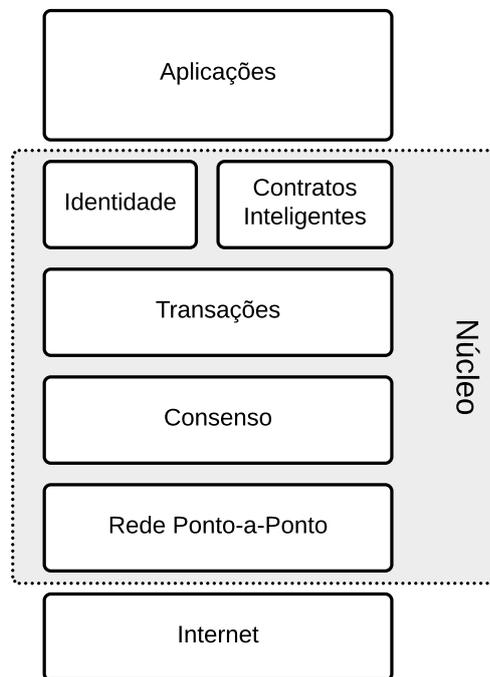
Dadas essas informações, podemos dizer que as características chave da *blockchain* são: (1) uma arquitetura de fato descentralizada, que não depende de um nó central; (2) persistência de dados assegurada por algoritmos de consenso de modo que transações inválidas são descobertas rapidamente e, após incluído, é inviável alterar um dado; (3) pseudonimato<sup>1</sup>, já que as interações com *blockchain* são realizadas por meio de endereços gerados; e (4) em *blockchains* públicas, um sistema totalmente auditável, haja vista que os dados são públicos e interligados (ZHENG et al., 2017). Para melhor compreensão da tecnologia *blockchain*, Concordium (2019) propõe uma representação de alto nível subdividida em camadas, conforme mostra a Figura 4.

A camada mais baixa é uma camada de Internet responsável pela conexão com a rede. Logo acima, há uma camada de rede P2P, responsável por encaminhar mensagens de forma descentralizada e sem censura. Mais acima, uma camada de consenso responsável por estabelecer um livro de registros global. Sobre ela, uma camada de transações responsável por determinar quais transações entram na *blockchain*. Depois, uma camada de identidades e uma de contratos inteligentes e, por fim, uma camada de aplicação (CONCORDIUM, 2019). As camadas do meio formam o núcleo da tecnologia *blockchain* e serão utilizadas aqui como base para o detalhamento dessa tecnologia. Vale ressaltar que os detalhes aqui apresentados, em sua maioria, são baseados na *blockchain* Bitcoin, podendo ter variações em outras implementações.

## 2.3 Rede Ponto-a-Ponto

Diversos sistemas e serviços web utilizam estrutura de armazenamento centralizada que necessita de grande confiança em um terceiro. O uso desse tipo de estrutura traz vulnerabilidades quanto à segurança de dados como: ponto único de falha, já que todos os dados estão em um

<sup>1</sup> Utiliza-se o neologismo *pseudonimato* ao invés de anonimato pois os endereços gerados (pseudônimos) podem acabar sendo conectados a identidades reais.

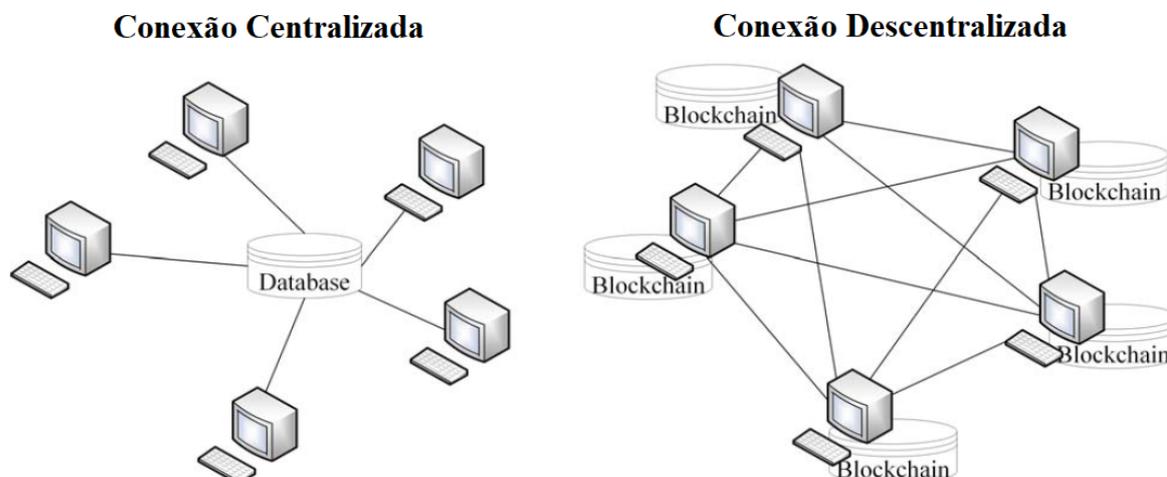
**Figura 4 – Blockchain Framework**

Fonte: adaptado de [Concordium \(2019\)](#)

único servidor; insegurança sobre a forma em que os dados são armazenados, já que o terceiro confiável possui total controle sobre eles; e políticas inconsistentes, já que vários sistemas especificam políticas e regulamentos diferentes, que são mutuamente incompatíveis ([WANG et al., 2018](#)). A utilização de redes descentralizadas busca minimizar esses problemas por meio da transparência e distribuição dos dados ([Figura 5](#)).

*Blockchain* tem como base um livro de registros distribuídos, que é descentralizado por natureza e mantém uma réplica dos dados com todos os participantes da rede, que auxiliam na manutenção do livro ([KUMAR; SUNITHA, 2017](#)). Ainda que esse mecanismo possa causar redundância, ele gera maior tolerância a falhas pois caso um ou mais nós sejam atacados, não ocorrem grandes danos sobre a rede como um todo. Dessa forma, *blockchain* atende os requisitos de transparência e confiança construindo protocolos de compartilhamento de código aberto e protegendo os dados por meio do poder de computação em vez de um terceiro confiável. Além disso, como os dados são replicados aos participantes da rede, há uma distribuição no poder de auditoria, pois cada participante pode verificar as informações inseridas na rede por conta própria ([KUMAR; SUNITHA, 2017](#); [DAI et al., 2017](#)). Por esses motivos, *blockchain* requer uma rede P2P.

Em uma rede P2P, cada participante é considerado igual e é referenciado como um nó. Os nós da rede possuem duas tarefas: manter os dados da rede em seu próprio nó e fiscalizar os

**Figura 5 – Armazenamento descentralizado na *blockchain***

Fonte: [Dai et al. \(2017\)](#)

demais nós. Na *blockchain*, apesar de todos os nós serem iguais, eles podem assumir funções diferentes na rede. Há nós que são conhecidos como nós completos, que possuem uma cópia inteira da *blockchain*. Esses nós asseguram a integridade da rede pois, para que as informações da *blockchain* fossem destruídas, seria necessário atacar e destruir os dados de cada um deles. Há também os nós conhecidos como mineradores, que são responsáveis pela realização do algoritmo de consenso na rede ([GAO et al., 2018](#); [LIU; XU, 2018](#)).

## 2.4 Consenso

A tecnologia *blockchain* resolve um problema clássico em comunicações P2P conhecido como Falha Bizantina. Essencialmente, a grande questão por de trás da Falha Bizantina é a impossibilidade de obter consistência em trocas de mensagens entre diversas partes que se comunicam por um canal não confiável em que haja perda de informações, como a Internet. Por esse motivo, em redes P2P que utilizam a Internet como canal de comunicação, até então, se fazia necessário o uso de um nó central para evitar fraudes e decisões incorretas. Para construir uma rede descentralizada sem uma unidade central controladora, era preciso definir alguma forma de selecionar qual o nó responsável pela inserção de dados na rede. *Blockchain* resolve esse problema utilizando algoritmos de consenso que possibilitam uma rede P2P verdadeiramente descentralizada ([LIU; XU, 2018](#); [ZHENG et al., 2017](#)).

Algoritmos de consenso usados em *blockchain* geralmente envolvem “mineração” para adição de dados, que significa a resolução de algum problema difícil de ser superado porém facilmente verificável. Os nós que desejarem adicionar dados na rede competem entre si para ter esse direito e são conhecidos como mineradores. Com a utilização de algoritmos de consenso,

*blockchain* garante que os nós sejam capazes de confirmar a validade dos blocos à medida em que são adicionados na cadeia e, caso um nó queira publicar um novo bloco, ele deve provar que não deseja executar um ataque na rede (RENNER et al., 2018; GAO et al., 2018).

Há vários mecanismos de consenso propostos atualmente, dos quais se destacam a Prova de Trabalho (PoW), a Prova de Participação (PoS) e a Tolerância a Falha Bizantina Prática (PBFT). Todos possuem o mesmo objetivo: determinar precisamente quais blocos estão corretos, checando o trabalho investido na criação de cada bloco. A diferença entre cada algoritmo consiste em quem pode adicionar novos blocos e em que velocidade, além de quais condições precisam ser satisfeitas para provar à rede que determinado nó pode incluir uma nova transação em um bloco (GAO et al., 2018). Atualmente, PoW é o algoritmo utilizado nos maiores *blockchains* públicos, como Bitcoin e Ethereum.

A realização do PoW é um procedimento de força bruta. O nó que deseja criar um novo bloco deve gerar um *hash* de todas as informações contidas no cabeçalho deste bloco obedecendo a quantidade máxima de *bits* especificada pela propriedade *nBits*, ou seja, um *hash* que comece com uma quantidade específica de zeros. A única informação do cabeçalho que pode ser alterada é a propriedade *Nonce*, um número que os nós mineradores incrementam até encontrar uma combinação que gere um *hash* válido. O nó minerador que resolver esse problema primeiro repassa o bloco com o *hash* e *Nonce* para que demais nós possam verificar e, caso a maioria da rede concorde que o valor está correto, esse nó ganha o direito de inserir o novo bloco. Assim, o tempo investido para verificação do PoW é  $O(1)$ , pois basta gerar um único *hash* dos dados com o *Nonce* repassado. O grande trabalho computacional é realizado somente pelos nós mineradores, que são recompensados por isso (RENNER et al., 2018; CAI et al., 2018).

De fato, o custo computacional imposto pelo PoW é intencional. A quantidade de *bits* exigida na criação do *hash* do bloco é determinada por quantos mineradores e quanto poder computacional está conectado à rede. Com isso, esse algoritmo aumenta enormemente a dificuldade de ataques para forjar os dados devido ao grande investimento em hardware exigido. Ademais, como recompensa pela produção dos blocos, os nós mineradores obtêm uma quantia em moeda digital da rede em que mineram a cada bloco que criam. Dessa forma, mesmo que um nó possua tremenda capacidade computacional, é mais lucrativo para ele utilizar essa capacidade na criação de novos blocos para ser recompensado do que atacar a rede (RENNER et al., 2018; CAI et al., 2018). As moedas pagas aos nós mineradores são provenientes de duas fontes: inserção de novas moedas na rede e taxas cobradas a cada transação realizada.

Por convenção, a primeira transação de cada bloco é referente à inserção de uma nova moeda na rede, destinada ao nó minerador do bloco. Além de servir como incentivo para criação dos blocos, essas transações também oferecem uma forma de colocar novas moedas em circulação, já que não há uma autoridade central para emití-las. O incentivo é baseado também em taxas que são cobradas de cada participante da rede que deseja realizar uma transação, sendo que quando houver um número predeterminado de moedas em circulação o incentivo pode ser

realizado somente através das taxas, de modo a conter a inflação da moeda (NAKAMOTO et al., 2008). Ao realizar uma transação, o emitente pode escolher quanto deseja pagar de taxa para que ela seja efetivada, possibilitando aumentar ou diminuir a prioridade e tempo para que essa transação seja adicionada a um bloco.

Já no Ethereum, o cálculo é realizado de uma forma diferente, com base no conceito de limite de combustível (*Gas*). Combustível é a nomenclatura utilizada como unidade de medida para especificar o poder computacional investido em transações do Ethereum. Basicamente, cada transação exige uma quantidade de combustível para ser realizada e ao efetuar uma transação o emitente deve especificar qual limite de combustível ela poderá consumir. Caso a transação seja concluída, o combustível restante é devolvido ao emitente. Porém, caso falte combustível, a transação não é efetivada e nada é devolvido ao emitente, uma vez que houve poder computacional investido (WOOD, 2019). Assim, o que determina a prioridade de uma transação no Ethereum é o valor pago pelo combustível, também especificado a cada transação. Atualmente esse valor fica acima de 50 Gwei<sup>2</sup> para a maior parte das transações na rede, como mostra a Figura 6. A flutuação do custo de transações em dólares americanos (USD) para as redes Bitcoin e Ethereum de dezembro de 2020 a março de 2021 é exibida na Figura 7.

**Figura 6 – Porcentagem de Transações por Preço do Combustível no Ethereum**



Fonte: <<https://ethgasstation.info>> (Acesso em 06/03/2021)

## 2.5 Transações

As transações são os registros de dados na *blockchain*, que podem ser provenientes de transferências monetárias entre usuários ou da execução de códigos e armazenamento de dados na rede. Na Bitcoin, Nakamoto et al. (2008) estabelece uma moeda digital como uma cadeia de assinaturas digitais, onde explica:

Cada proprietário transfere a moeda para o próximo assinando digitalmente um hash da transação anterior e a chave pública do próximo proprietário e

<sup>2</sup> Unidade de moeda do Ethereum equivalente a 0.00000001 ETH

**Figura 7 – Média de Custo por Transação na Bitcoin e Ethereum (\$)**

Fonte: <<https://bitinfocharts.com/comparison/transactionfees-btc-eth.html#3m>> (Acesso em 06/03/2021)

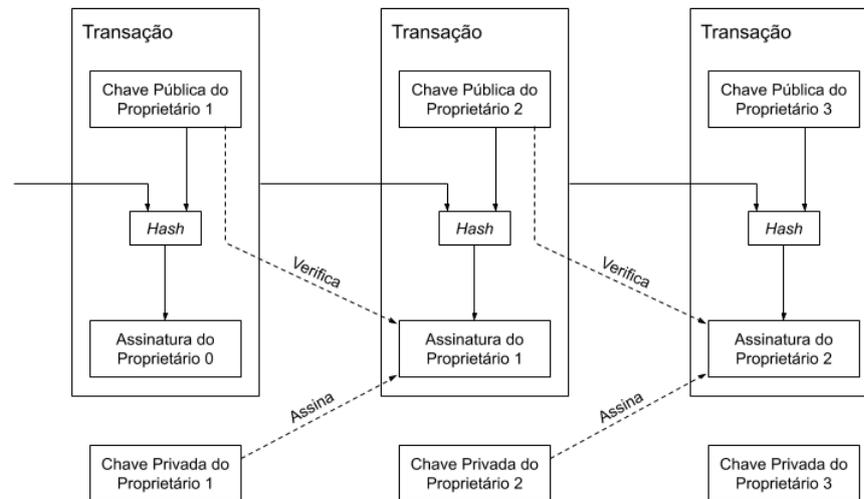
adicionando-os ao final da moeda. Um beneficiário pode verificar as assinaturas para verificar a cadeia de propriedade.<sup>3</sup>

Essa definição é a base para entender as transações na *blockchain*. De forma genérica, na *blockchain* temos o seguinte processo: o Proprietário *A* deseja realizar uma transação (Tx) ao Proprietário *B*. Para isso, o Proprietário *A* deve assinar digitalmente ( $S_A$ ) um *hash* ( $H$ ) dos dados de uma transação em que foi anteriormente beneficiado acrescidos da chave pública ( $l_B$ ) do Proprietário *B*. Assim obtém-se:  $S_A(H(Tx || l_B))$ , de modo que qualquer outro participante da rede pode conferir se o Proprietário 1 realmente realizou a transação e o Proprietário 2 é o único a quem a transação pertence (GAO et al., 2018). Esse processo pode ser visualizado na Figura 8.

Com esse processo é possível atestar a origem das transações realizadas, já que cada transação é conectada às transações anteriores. Cada transação é transmitida pela rede e os nós verificam sua autenticidade a partir da assinatura digital. Caso a transação seja válida, ela é adicionada ao montante de transações ainda não confirmadas que serão incluídas no próximo bloco, sendo que as transações só são efetivadas de fato ao fim da criação de cada bloco. A assinatura e verificação das transações utilizam o Algoritmo de Assinatura Digital com Curva Elíptica (ECDSA) (ALKETBI et al., 2018; MOUBARAK et al., 2018). Na Bitcoin e em muitas *blockchains* derivadas dela, o saldo de um usuário é determinado com uma lista de transações não gastas (UTXO). Basicamente essa lista contém as transações destinadas ao usuário que ainda não foram destinadas a mais ninguém (ZHANG et al., 2019).

Tomando como exemplo a Bitcoin, uma transação em sua *blockchain* contém um identificador, versão, tempo de bloqueio, entrada(s) e saída(s) (Figura 9). Seu identificador é um *hash* SHA-256 dos dados da transação. A versão se refere às regras de consenso utilizadas na

<sup>3</sup> Texto original: "Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership."

Figura 8 – Transações na *Blockchain*

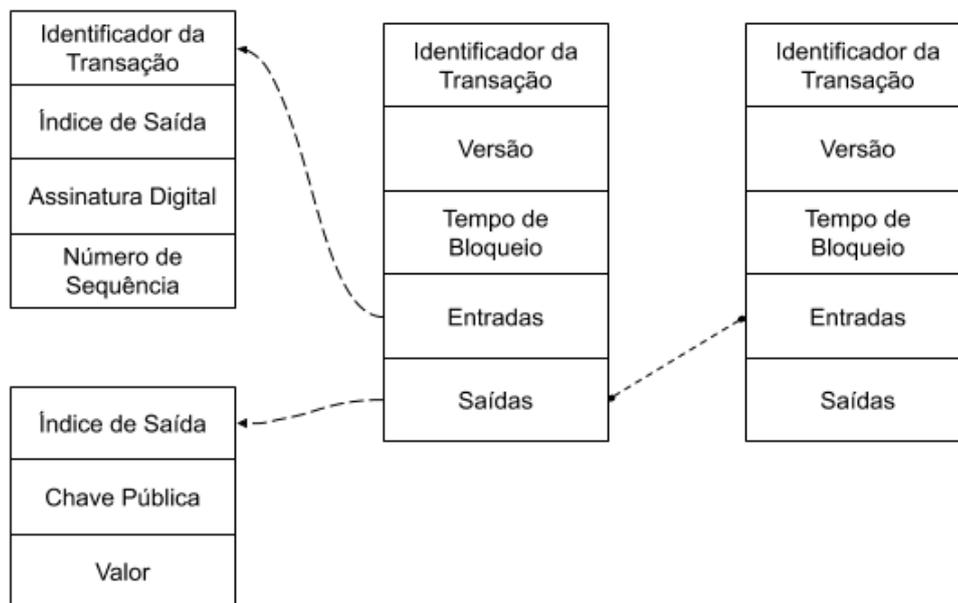
Fonte: Nakamoto et al. (2008)

transação. O tempo de bloqueio indica o horário a partir do qual transação pode ser adicionada a um novo bloco. As entradas identificam demais transações que são usadas como referência para a atual, contendo seus identificadores, índice de saída, assinatura digital e número de sequência (utilizado para permitir que transações ainda não confirmadas sejam atualizadas). As saídas identificam valores e destinatários da transação, possuindo um índice para indicar a ordem de saída, a chave pública do destinatário e um campo de valor para cada saída (WANG et al., 2018).

### 2.5.1 Cadeia de Blocos

A estrutura dos dados na *blockchain* baseia-se em blocos que são encadeados uns aos outros. Cada bloco contém um novo conjunto de transações sendo que o número máximo de transações que um bloco pode conter depende de seu tamanho e do tamanho das transações. Após ser criado e vinculado à cadeia, os dados do bloco são concluídos e não podem mais sofrer alterações. Basicamente, *blockchain* gera um *hash* de todo o conjunto de dados contido em um bloco e o utiliza como índice desse bloco, que será vinculado no bloco seguinte de forma que cada bloco da cadeia aponte para o bloco anterior (GAO et al., 2018; LIU; XU, 2018). Além do conjunto de dados contidos em um bloco, a função *hash* recebe como entrada um carimbo de data e hora do momento em que o *hash* está sendo criado.

A adição do carimbo de data e hora (*timestamp*) ao *hash* possibilita a garantia de que um dado foi registrado em determinado momento no tempo (YANG et al., 2018). Os demais nós da rede fazem uma verificação para aceitar um bloco somente se esse carimbo representar, ao menos de forma aproximada, a data e a hora atual. Caso haja qualquer alteração, o *hash* resultante seria completamente diferente, e por esse motivo esse método é também usado como forma de assegurar a integridade dos dados, sendo a base para a realização da PoE (detalhada na

**Figura 9 – Estrutura de dados nas transações do Bitcoin**

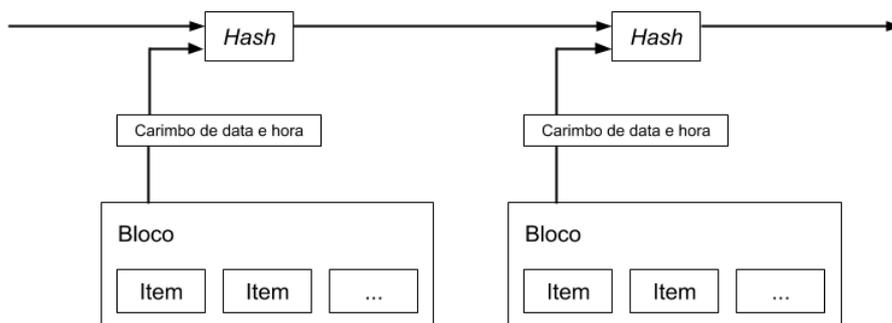
Fonte: Wang et al. (2018)

### Seção 2.8.1).

Para garantir a integridade dos dados em toda *blockchain*, o *hash* de cada bloco é usado no cálculo do *hash* bloco seguinte, de forma que cada *hash* adicional reforça os anteriores (NAKAMOTO et al., 2008), como demonstrado na Figura 10. Essa cadeia de *hashes* é que faz a *blockchain* aceitar somente inclusão de dados e não alterações ou exclusões, assegurando a imutabilidade dos dados pois, para fazer qualquer alteração em um bloco que já foi adicionado na cadeia, seria necessário alterar também todos os blocos posteriores a esse em todos os nós da rede (MENG et al., 2018). Segundo Nakamoto et al. (2008), a execução da *blockchain* contém os seguintes passos:

1. novas transações são repassadas e validadas pelos nós vizinhos;
2. cada nó agrupa as transações em um bloco;
3. cada nó busca a solução do algoritmo de consenso para esse bloco;
4. quando um nó acha o resultado ele o repassa para todos os nós;
5. os nós aceitam o bloco somente se todas as transações forem válidas; e
6. os nós demonstram que aceitaram o bloco passado trabalhando na criação de um novo bloco que aponta para ele.

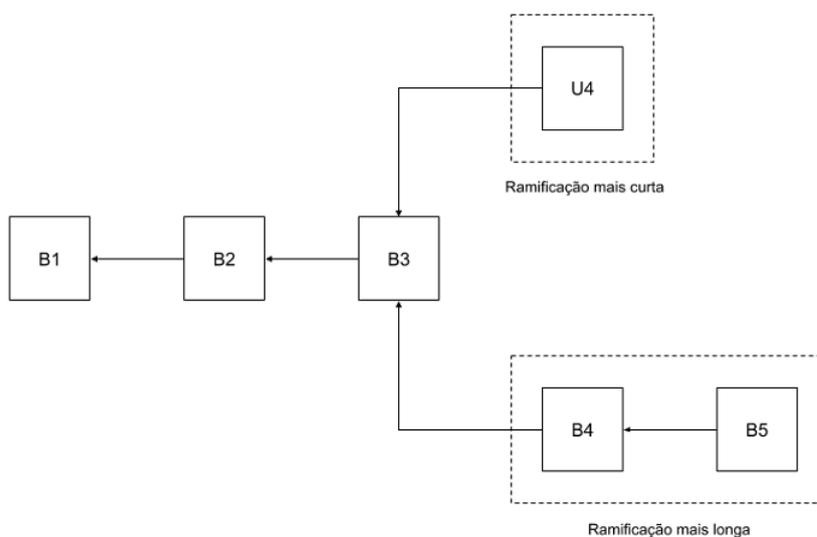
**Figura 10 – Blocos encadeados por *timestamps***



Fonte: adaptado de Nakamoto et al. (2008)

Os nós sempre consideram a cadeia mais longa como a correta e trabalham para estendê-la. Caso dois nós repassem diferentes versões do próximo bloco, uma ramificação é criada na cadeia. Os nós continuam trabalhando no bloco que receberam primeiro, mas mantêm o outro salvo para o caso dele tornar-se a cadeia mais longa. Assim que o próximo algoritmo de consenso for realizado, uma das bifurcações se tornará mais longa, e então todos os blocos voltarão a trabalhar nela (NAKAMOTO et al., 2008). Um exemplo desse processo é ilustrado na Figura 11, em que a ramificação contendo o bloco *U4* foi descartada por tornar-se a mais curta após a criação do bloco *B5*, que aponta para *B4*.

**Figura 11 – Ramificações na *blockchain***



Fonte: Zheng et al. (2017)

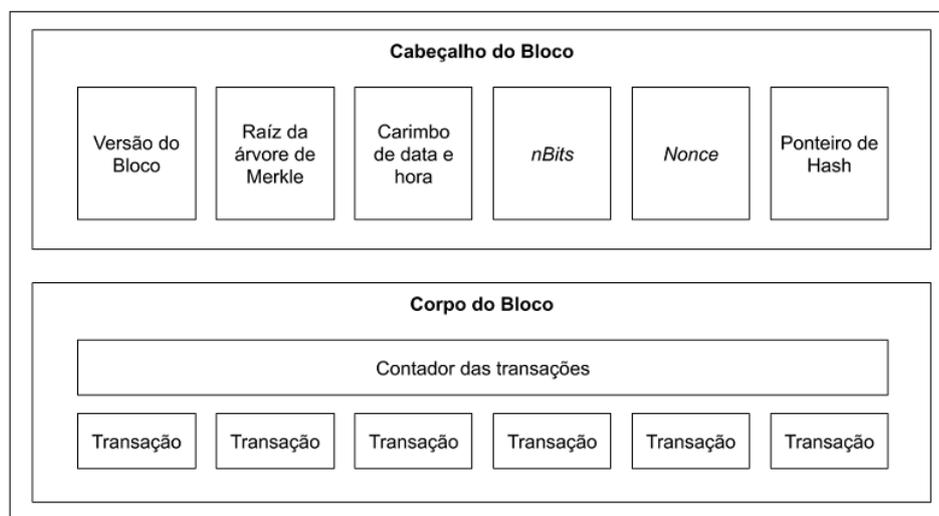
Portanto, caso um agente malicioso alterasse os dados de um bloco já inserido na cadeia, o *hash* desse bloco também seria alterado. Com isso, o novo *hash* não estaria vinculado a nenhum outro bloco e não seria aceito pela rede, quebrando a cadeia de blocos e criando uma nova

ramificação na *blockchain*. Como a regra para bifurcações é aceitar somente a mais longa, para que o bloco alterado seja aceito pela rede o agente malicioso precisaria crescer sua ramificação mais rápido que o restante da rede. Em algoritmos de consenso baseados em PoW, a criação de novos blocos é uma tarefa computacional custosa, de forma que para estender sua ramificação o agente malicioso precisaria ter um poder computacional maior que o do restante da rede. Em *blockchains* públicas robustas como Bitcoin ou Ethereum isso é impraticável, uma vez que o poder computacional da rede é muito grande (KALIS; BELLOUM, 2018).

## 2.5.2 Estrutura dos Blocos

Blocos são a unidade de armazenamento da *blockchain* e possuem, essencialmente, corpo e cabeçalho, como mostrado na Figura 12. O corpo agrega todas as transações executadas desde a criação do último bloco até a criação do atual, acrescido de um contador de transações. As transações são organizadas em estrutura de Árvore de Merkle, sendo que cada transação representa um nó folha. São gerados *hashes* de cada transação e de cada conjunto de *hashes* filho até que se chegue à Raiz de Merkle. Essa estrutura possibilita reduzir o tamanho do armazenamento da *blockchain* por meio do descarte de transações antigas, e também é eficiente para a verificação do bloco a partir da Raiz de Merkle que é armazenada no cabeçalho (GAO et al., 2018).

Figura 12 – Estrutura do bloco

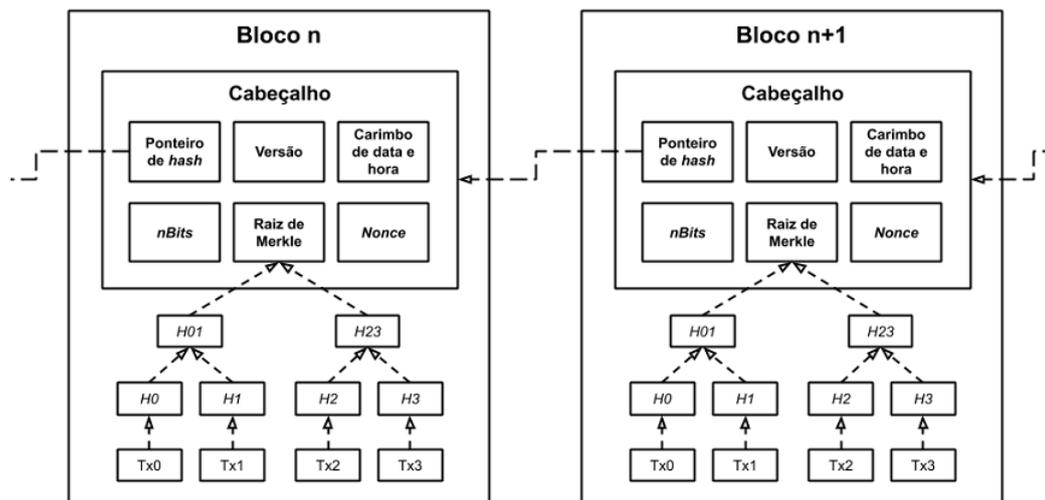


Fonte: adaptado de Zheng et al. (2017)

O cabeçalho de cada bloco é composto por: (1) versão do bloco, utilizada para identificar quais regras foram usadas em sua validação; (2) a Raiz de Merkle correspondente a todas as transações do bloco; (3) carimbo de data e hora que indica o momento em que o bloco foi criado; (4) *nBits*, uma maneira compacta de armazenar a quantidade máxima de *bits* que o *hash* do bloco deve ter para ser aceito em sua criação (utilizado para ajustar a dificuldade do algoritmo de consenso); (5) *Nonce*, um número inteiro utilizado para encontrar um *hash* válido para o

bloco no algoritmo de consenso; e (6) índice do bloco anterior em forma de ponteiro de *hash*, responsável pelo encadeamento dos blocos (ZHENG et al., 2017; GAO et al., 2018). Uma versão resumida da camada de dados pode ser visualizada na Figura 13.

Figura 13 – Estrutura dos dados na *blockchain*



Fonte: adaptado de Renner et al. (2018)

## 2.6 Identidade e Contratos inteligentes

*Blockchain* se propõe a preservar a privacidade da identidade de usuários por meio de chaves pública e privada. Como as transações são feitas somente através das chaves dos usuários, não há exposição de identidade real. No entanto há trabalhos que revelam que a *blockchain* não pode garantir a privacidade transacional, uma vez que os valores de todas as transações e saldos para cada chave pública são visíveis publicamente. Várias soluções têm sido propostas para melhorar o anonimato na rede, que podem ser categorizados em dois tipos: Mistura, que possibilita transferir fundos de vários endereços de entrada para vários endereços de saída; e Anônimo, no qual a origem do pagamento é desvinculada das transações para evitar análises do grafo de transações (ZHENG et al., 2017).

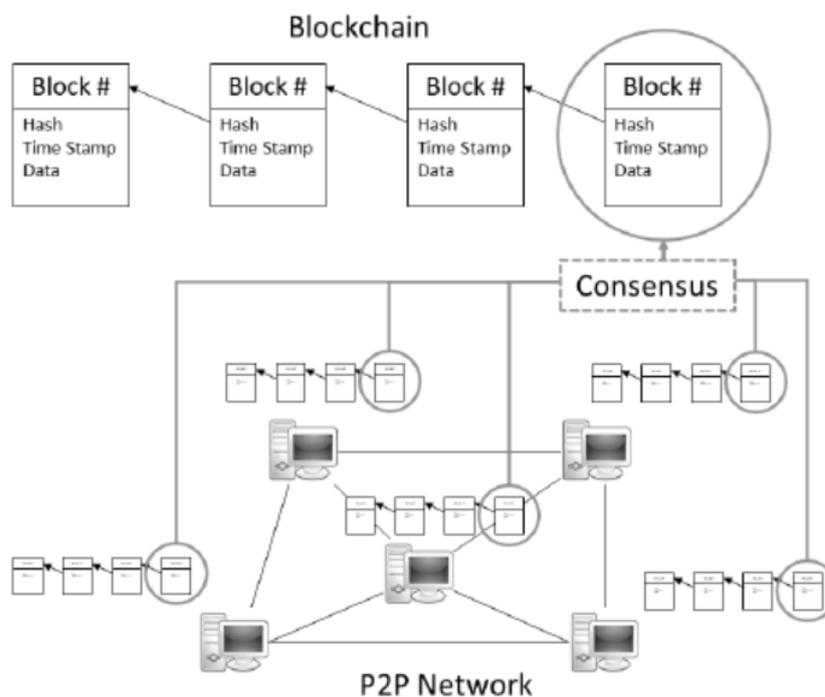
Contratos inteligentes são uma maneira de digitalizar contratos reais, especificando um acordo entre as partes por meio de código sem a necessidade de um terceiro confiável. Embora o conceito seja antigo, a tecnologia *blockchain* facilitou a implementação dessa ideia (KALIS; BELLOUM, 2018). Após um contrato inteligente ser desenvolvido e enviado à *blockchain*, ele não pode mais ser alterado, devido à imutabilidade da rede. Assim, publicar um contrato inteligente cria um conjunto de funcionalidades confiáveis que os demais participantes da rede podem conferir e às quais podem aderir. Ao invocar um contrato inteligente, ele é executado

de maneira descentralizada entre os nós da rede por meio do algoritmo de consenso, que é o responsável por garantir que os contratos serão executados corretamente. Ethereum foi pioneiro na utilização de contratos inteligentes ao propor a linguagem de programação Solidity, que é Turing completa (CAI et al., 2018). A utilização de contratos inteligentes é um marco da expansão do desenvolvimento de aplicações na *blockchain* para além das moedas digitais.

## 2.7 Conclusões sobre núcleo da *blockchain*

Temos então os elementos chaves do núcleo da *blockchain*: uma rede distribuída P2P sem unidade central, que mantém uma base de dados estruturada em blocos, cujos dados são assinados com esquemas de criptografia assimétrica, têm sua integridade assegurada por *hashes* e *timestamps*, e que reduz a necessidade de um terceiro confiável graças à realização de algoritmos de consenso que levam em consideração a opinião da maioria da rede para inserção de dados. Um resumo desses elementos pode ser visualizado na Figura 14. É por meio da junção de diversas tecnologias já existentes que *blockchain* oferece não somente um sistema financeiro, mas sim uma maneira completamente inovadora de realizar interações pela Internet, de forma segura sem a necessidade de um terceiro confiável. Graças a essas características, muitas pesquisas vem sendo realizadas acerca dessa tecnologia buscando sua aplicação em diversas áreas.

Figura 14 – Elementos chaves da *blockchain*



Fonte: Cai et al. (2018)

Pode-se concluir que há grandes benefícios na utilização da tecnologia *blockchain*, mas

que esta não é a solução ideal para todos os casos. Segundo [Chowdhury et al. \(2018\)](#), *blockchain* é recomendada para os casos em que:

- há múltiplas partes envolvidas com insuficiência de confiança;
- não há um terceiro confiável;
- os dados armazenados devem ser imutáveis; e
- escalabilidade não é um requisito crítico.

Caso a solução a ser desenvolvida não atenda a algum desses requisitos, provavelmente é possível encontrar alguma outra tecnologia que se adeque melhor ao desenvolvimento. [Chowdhury et al. \(2018\)](#) acrescenta ainda que caso haja necessidade de verificação pública é recomendada a utilização de *blockchains* públicas. Dentre os pontos negativos dessa tecnologia estão a demora para efetivação das transações e o grande consumo de energia pelo uso de algoritmos de consenso ([GATTESCHI et al., 2018](#)). De acordo com essas definições, a utilização de *blockchains* públicas na proposta deste trabalho justifica-se pelos seguintes motivos:

- há múltiplas partes envolvidas e dificuldade de se encontrar um terceiro confiável para armazenamento e auditoria de *logs*;
- por se tratarem de registros de eventos ocorridos ao longo do tempo, *logs* são dados tipicamente imutáveis;
- realizando a validação com base em *hashes* calculados a partir de uma entrada que possui um componente não-determinístico, tornar estes *hashes* públicos não representa risco à privacidade dos dados (devido à resistência à pré-imagem), possibilitando uma auditoria mais transparente com a utilização de *blockchains* públicas; e
- o uso de árvores de Merkle para combinar diversos *hashes* em um único registro é capaz de mitigar os problemas de escalabilidade das *blockchains* nas quais o registro é efetuado.

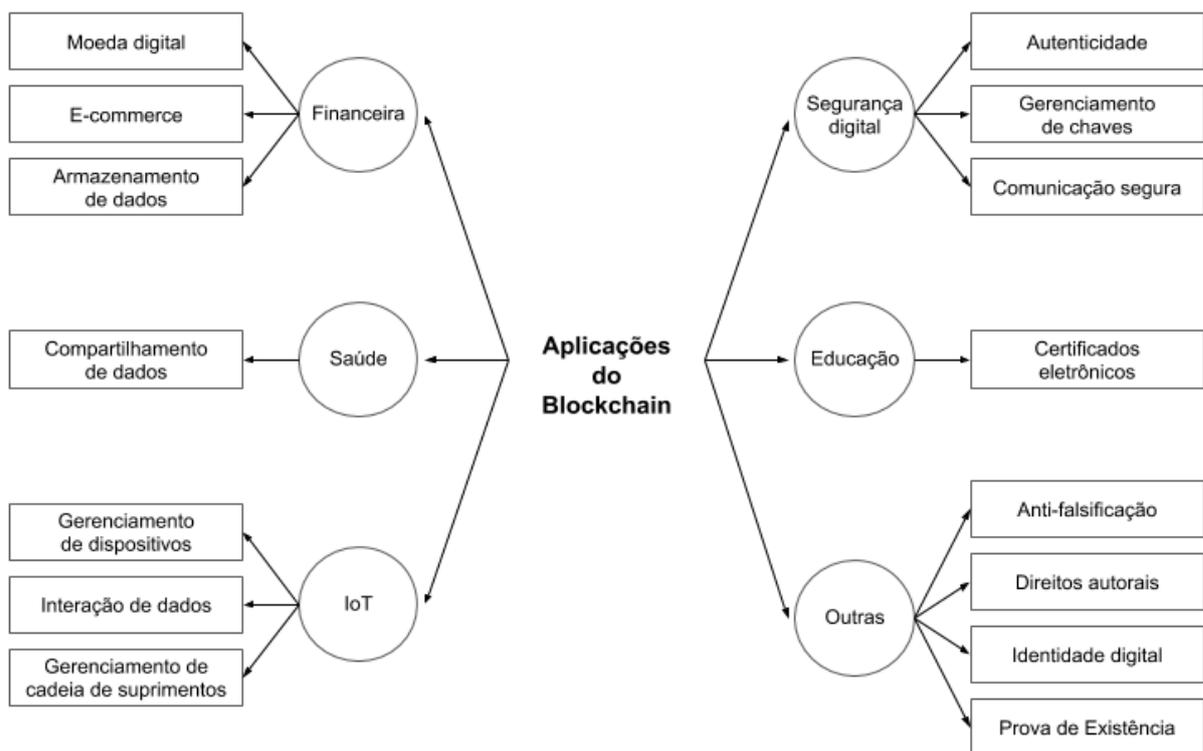
## 2.8 Aplicações e PoE

A camada de aplicações é a de mais alto nível na arquitetura *blockchain*, logo acima de seu núcleo. Ela é composta pelas diversas aplicações descentralizadas que integram a rede e se beneficiam do seu registro contínuo, consenso sobre nós distribuídos, contratos inteligentes e componentes de criptografia. A aplicação da tecnologia *blockchain* fora do contexto de moedas digitais é um conceito que tem se expandido cada vez mais, principalmente em áreas onde há uma demanda crescente por segurança e controle distribuído ([GAO et al., 2018](#)). Nessa camada

existem aplicações que são totalmente descentralizadas e outras que são parcialmente, por contar com algum serviço centralizado complementar.

Conforme exibido na [Figura 15](#), há diferentes categorias de aplicações que fazem uso da arquitetura *blockchain*, dentre as quais se destacam: a financeira, por todos os benefícios já citados; saúde, com aplicações que possibilitam o compartilhamento de informações entre hospitais de forma segura; Internet das coisas (IoT), na qual se vem aplicando *blockchain* em larga escala para automação de processos e para garantir a segurança de dados em dispositivos com baixo poder computacional; educação, atualmente com foco na geração de certificados eletrônicos; segurança digital, com aplicações para autenticação, gerenciamento de chaves e comunicação segura; e demais aplicações como direitos autorais, identidade digital e a própria PoE ([DAI et al., 2017](#)).

**Figura 15 – Cenários de aplicações da *blockchain***



Fonte: adaptado de [Dai et al. \(2017\)](#)

### 2.8.1 Prova de Existência

Integridade de dados refere-se ao processo de manter os dados inalterados desde sua criação até que sejam consumidos novamente. Assegurar a integridade dos dados é uma demanda

crucial, uma vez que estes podem ser corrompidos por ataques ou até mesmo por erros no desenvolvimento de aplicações. Um bom sistema de informação deve ser capaz de manter informações sobre os dados desde que foram inseridos, de forma a poder auditá-los e confirmar sua integridade. Algumas das técnicas utilizadas por profissionais de segurança de base de dados são: *backup*, para restaurar os dados caso sejam perdidos ou corrompidos; controle de acesso, para gerenciar quem tem o privilégio de ver ou alterar os dados do sistema; e validação de dados, para certificar que um dado não foi corrompido enquanto armazenado ou durante sua transmissão (KUMAR; SUNITHA, 2017).

Kumar e Sunitha (2017) apresentam diferentes atributos a serem considerados em um esquema de integridade de dados, dos quais se destacam: (1) abordagem, que se refere à forma de validação dos dados oferecida pelo esquema, podendo ser determinística ou probabilística, ou seja, se baseia no arquivo inteiro para validação ou somente em pedaços do arquivo escolhidos de maneira aleatória; (2) natureza dos dados, que especifica se os dados são estáticos e não permitem alterações ou se são dinâmicos; e (3) tendência, que especifica se o esquema é somente de verificação dos dados ou de verificação com recuperação. Esses atributos devem ser estudados e analisados antes da implementação de qualquer esquema de integridade de dados visando garantir maior segurança e melhor desempenho. De acordo com essas características, a PoE possui abordagem determinística e tendência somente de verificação.

A realização de PoE baseia-se no conceito de *timestamp*, de modo que, para comprovar o estado de um conjunto de dados em determinado momento, gera-se um *hash* desse conjunto de dados que é enviado a uma entidade certificadora responsável por registrar o *hash* e comprovar o momento em que ele foi salvo. Dessa forma, é possível provar que naquele momento aqueles dados existiam exatamente daquela forma. Para validação, basta gerar um novo *hash* e comparar com o *hash* salvo anteriormente. Caso os *hashes* não coincidam, significa que o conjunto de dados não está íntegro. Essa abordagem é considerada determinística pois a validação é realizada sobre o *hash* de todo o conjunto de dados e a tendência é somente de verificação pois caso seja constatado que a integridade foi ferida, não há como recuperar os dados originais a partir do *hash* gerado anteriormente (MENEGETTI et al., 2019; LIANG et al., 2017).

Esse método é bem conhecido e é aplicado a outros cenários além da comprovação de integridade de dados, como por exemplo comprovação de autoria para disputa por direitos autorais, certificação de informações, dentre outros. Antes da criação da *blockchain*, a PoE era realizada com a mediação de empresas, que centralizavam os dados e atuavam como terceiro confiável. Com *blockchain*, diminuiu-se a necessidade de confiança pois, como discutido anteriormente, os dados são controlados de forma pública e distribuída, assegurados por criptografia e por poder de computação. É notório que a PoE e a *blockchain* estão intrinsecamente ligados, já que ambos utilizam o conceito de *timestamp* para validação dos dados (WANG et al., 2018; SCHÖNHALS et al., 2018). Dentre as vantagens da realização de PoE em *blockchains* públicas, Wang et al. (2018) e Abreu et al. (2018) destacam:

1. como qualquer ator do sistema pode participar do processo de validação dos blocos, a segurança é relegada ao protocolo e aos princípios de criptografia;
2. graças à imutabilidade da rede, os dados não podem ser modificados por ninguém, nem pelo proprietário, nem por terceiros, ou mesmo pelo governo;
3. em *blockchains* públicas maduras (Bitcoin ou Ethereum, por exemplo) há um grande poder computacional na rede, tornando inviável ataques de falsificação; e
4. a transparência e a interoperabilidade de *blockchains* públicas maduras podem auxiliar a resolução de problemas judiciais.

Há mais de uma maneira de se realizar a PoE em *blockchains* públicas, bastando enviar para rede o *hash* do conjunto de dados desejado. A partir do momento que o *hash* é ancorado em um bloco, que possui data e hora de criação, é possível provar sua existência. Para realizar o envio do *hash* à *blockchain*, pode-se adotar uma das seguintes abordagens: agregar o *hash* aos dados de uma transação, utilizando a operação OP\_RETURN da Bitcoin, por exemplo; fazer uso de algum contrato inteligente que realize o armazenamento de *hashes*; ou realizar uma transação com valor mínimo utilizando como endereço de destino o *hash* gerado (MENEGETTI et al., 2019; WANG et al., 2018). Esta última abordagem é menos desejável por gerar UTXOs que nunca serão gastas, uma ação análoga a queimar dinheiro.

Tendo em vista o custo para realizar as transações e o número de transações que podem ser processadas por segundo pela rede, é impraticável enviar à *blockchain* o *hash* de cada pequeno conjunto de dados que se deseja assegurar (HEPP et al., 2018b). Para resolver essa demanda, existem APIs que propõem realizar PoEs com auxílio de árvores de Merkle. Ao invés de mandar à *blockchain* cada *hash*, essas APIs recebem *hashes* de diferentes fontes e os concatenam em uma árvore de Merkle. Após certo período de tempo, a raiz de Merkle é enviada a uma *blockchain* pública por meio de alguma das abordagens citadas acima. Armazenando somente a raiz de Merkle na *blockchain*, é possível comprovar a existência de todos os *hashes* concatenados na árvore com o custo de uma única transação. Dessa forma, é possível realizar a PoE de vários conjuntos de dados a um custo baixo e fixo. As APIs não possuem conhecimento sobre o conjunto de dados originais, pois trabalham somente com seus *hashes* (POURMAJIDI; MIRANSKY, 2018; SCHÖNHALS et al., 2018; HEPP et al., 2018b). De forma geral, os passos para realização de PoEs utilizando APIs com árvores de Merkle são:

1. O cliente deve gerar o *hash* do conjunto de dados que deseja atestar e enviar este *hash* à API. O envio do *hash* geralmente é realizado por meio de uma arquitetura de Transferência de Estado Representacional (REST) ou por ferramentas CLI fornecidas pela API.
2. Após o envio do *hash* à API, o cliente deve aguardar até que esta realize a PoE na(s) *blockchain(s)* da raiz de Merkle de todos os *hashes* recebidos em determinado período de

tempo. O período de tempo e a quantidade de *blockchains* utilizadas variam de uma API para outra.

3. Após a efetivação da raiz de Merkle em um bloco da *blockchain*, a API gera um recibo (ou atestado) para cada *hash* recebido, como exemplificado na [Figura 16](#). Esse recibo contém informações sobre a transação realizada na *blockchain* e os dados necessários para reconstrução da raiz de Merkle a partir do *hash* específico. Após a criação do recibo o cliente pode baixá-lo e armazená-lo localmente.
4. Com essas informações, auditores e proprietários dos dados podem reconstruir a árvore de Merkle a partir dos *hashes* que constam no recibo e verificar se a raiz de Merkle encontra-se na transação que também é especificada no recibo.
5. A verificação pode ser realizada por meio da própria API ou através da utilização de alguma ferramenta de acesso a *blockchains*, como, por exemplo, a Block Explorer<sup>4</sup>.
6. Dessa forma, ainda que a PoE seja realizada com intermédio da API, os dados podem ser validados sem a mediação desta ou de qualquer terceiro. Uma visão geral desse processo é demonstrada na [Figura 17](#).

O recibo retornado pelas APIs traz segurança tanto no sentido de poder realizar a verificação diretamente na *blockchain* quanto pelo fato de que, caso um dia a API deixe de existir, basta ter salvo os recibos retornados que sempre será possível realizar a verificação dos dados. Vale ressaltar também a presença de outras vantagens para verificação de dados baseada em *hashes*, como o bom desempenho por não ter que comparar *bit a bit* dos conjuntos de dados em um processo de validação e também a possibilidade de validar o conjunto de dados sem tornar público o seu conteúdo ([RENNER et al., 2018](#)). Todas essas características contribuem para a qualidade do processo de auditoria.

Em um esquema de garantia de integridade de dados, estão envolvidos: (1) o proprietário dos dados, que deseja obter auditoria de seus dados sem vazar nenhuma informação confidencial; (2) provedor de serviços, que armazena e lida com dados de seus usuários, desejando oferecer auditorias seguras e eficientes que comprovem a integridade de seu serviço; e (3) o auditor dos dados, que pode ser o próprio usuário ou auditor terceirizado escolhido pelo usuário, responsável por validar se o serviço mantém os dados íntegros ([KUMAR; SUNITHA, 2017](#); [WANG et al., 2018](#)). Tendo em vista a dificuldade e o alto custo para a realização de auditoria por terceiros, a técnica de PoE se apresenta como grande facilitadora para esquemas de atestação de integridade.

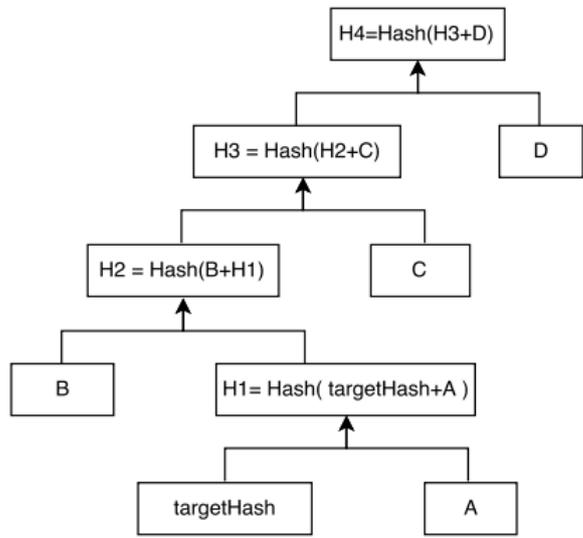
Por fim, a PoE aplicada a *logs* de sistemas de informação traz mais facilidades na validação de dados, já que a natureza dos dados de *logs* é estática. Dessa forma, esquemas de auditoria se baseiam em verificar se o conjunto de *logs* armazenado pelo provedor de serviços

<sup>4</sup> <<https://www.blockchain.com/explorer>>

**Figura 16 – Recibo de uma PoE realizada pela API ChainPoint e Árvore de Merkle**

```

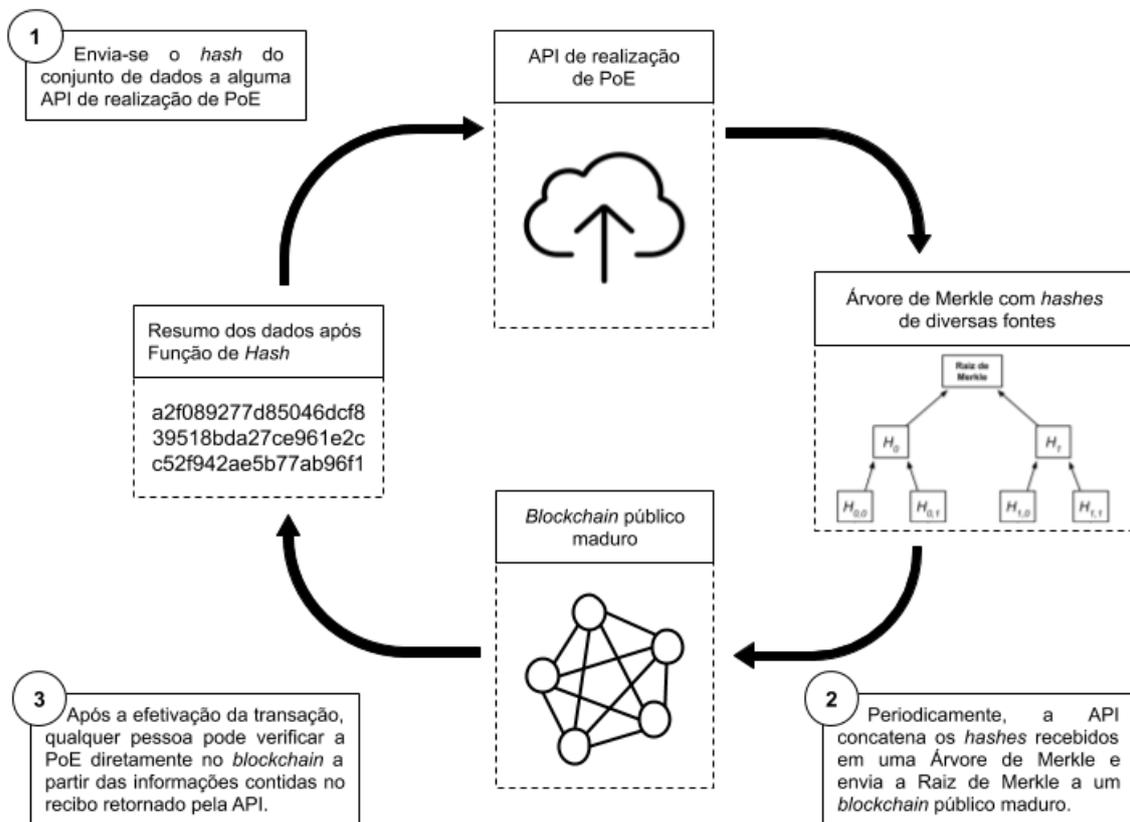
{
  "@context": "https://w3id.org/chainpoint/v2",
  "type": "ChainpointSHA256v2",
  "targetHash": "82e46ffd212d680b3e1a169e6a8b59472985ac55398b8740832fe94fd5e5fd63",
  "merkleRoot": "9f0100055a430539796817ce626d84ccb5485453e4d558cf3353e4d4a7e59031",
  "proof": [
    {
      "right": "0f6117e8bddd7fdc713aa5365e74aafe34f5cc31fd654ed84ea37976d873c087"
    },
    {
      "left": "f860e7697ba57d944d925f311cce786e6d20833071d1c16e6e5fef3fc4749c96"
    },
    {
      "right": "de4b5b29183d193b95905ae9741a928ab056cbbbefb9a537ac9282fe180c78bd"
    },
    {
      "right": "e75da94bc44a3a9778b2ec7a5ffd58e4a622d4ce4c20676215eb88a4764bb335"
    }
  ],
  "anchors": [
    {
      "type": "BTCOpReturn",
      "sourceId": "0b956b057330591cd63c90e5572ba364c6f9f08299c3e8ee0c893411db1c30a6"
    }
  ]
}
    
```



Fonte: [Liang et al. \(2017\)](#)

é o mesmo que foi atestado pela PoE na *blockchain*. Porém, ainda que esquemas como esse aparentem ser triviais, são pouco adotados por empresas, instituições e governos. Aumentar a confiança sobre a integridade dos dados armazenados nesses diversos contextos é essencial para uma sociedade mais justa e segura, evitando que usuários saiam prejudicados por negligência dos provedores de serviços.

Figura 17 – Realização de PoE utilizando APIs com árvores de Merkle



Fonte: adaptado de <<https://originstamp.org>>

# Capítulo 3

## METODOLOGIA

---

---

Este capítulo apresenta os materiais e métodos utilizados ao longo deste trabalho. São fornecidas aqui informações e referências a respeito dos materiais e ferramentas empregadas durante desenvolvimento, testes e análises. O estudo de caso é detalhado, de forma a embasar as etapas realizadas e técnicas adotadas. São apresentados os critérios para seleção das APIs de realização de PoEs, assim como os procedimentos para analisá-las. Os procedimentos de teste da aplicação desenvolvida são descritos e, por fim, são apresentados modelos de ataque para análise teórica da solução.

### 3.1 Materiais

Para que fosse possível criar uma ferramenta multiplataforma, o Auditchain foi desenvolvido utilizando a linguagem de programação Kotlin em uma arquitetura compatível com a Java Virtual Machine (JVM), e diversas bibliotecas auxiliaram seu desenvolvimento. A Retrofit obtém dados do Elasticsearch por meio de sua API REST; a Libsodium assina os dados; a Java Opentimestamps executa PoEs; e a Clikt auxilia a criar um aplicativo CLI. Os dados dos atestados de PoE são armazenados em um banco de dados SQLite e o IntelliJ IDEA Community foi o Ambiente de Desenvolvimento Integrado (IDE) utilizado em todo o processo de desenvolvimento. O Gradle ajuda na construção de um .jar executável e, por fim, uma máquina virtual Linux no Google Cloud Platform (GCP) foi utilizada para execução dos testes. Essas e demais ferramentas utilizadas durante a pesquisa são detalhadas abaixo:

- **Clikt** (<<https://github.com/ajalt/clikt>>): biblioteca Kotlin que auxilia no desenvolvimento de ferramentas CLI de forma simples e intuitiva. Oferece suporte a uma ampla variedade de casos de uso e permite a personalização avançada quando necessário.
- **Elasticsearch** (<<https://elastic.co>>): plataforma de pesquisa e análise quase em tempo real. Possui armazenamento distribuído com motores de busca e análise capazes de atender a um número crescente de cenários. Largamente adotada para indexação de *logs*.

- **Exposed** (<<https://github.com/JetBrains/Exposed>>): estrutura de Mapeamento de Objeto-Relacional (ORM) para Kotlin. Oferece acesso *typesafe* a banco de dados e objetos de acesso a dados leves.
- **Git** (<<https://www.git-scm.com>>): um dos sistemas de controle de versão mais utilizados atualmente. Possibilita ramificar o desenvolvimento e auxilia na segurança dos dados que podem ser retrocedidos a alterações anteriores.
- **GitHub** (<<https://github.com/>>): repositório online de Git que permite armazenar projetos na nuvem de forma pública ou privada. Projetos armazenados de forma pública, considerados de código aberto, podem receber colaboração de demais programadores da comunidade.
- **Google Cloud Platform** (<<https://cloud.google.com>>): conjunto de recursos físicos e recursos virtuais localizados nos *data centers* do Google por todo o mundo para computação em nuvem. Oferece vantagens como redundância em caso de falha e latência reduzida.
  - Na realização dos testes foi utilizada uma instância n2d-standard-2 com configurações padrões.
- **Gradle Build Tool** (<<https://gradle.org>>): Gradle é uma ferramenta de código aberto para automação de compilação focada em flexibilidade e desempenho. Seus scripts de compilação são escritos usando Groovy ou Kotlin. Possui suporte para aplicações desenvolvidas em Kotlin.
- **IntelliJ IDEA Community** (<<https://jetbrains.com/idea>>): Ambiente de desenvolvimento integrado com suporte para a linguagem Kotlin. Possui diversas ferramentas que auxiliam na produtividade durante o desenvolvimento.
- **Java Opentimestamps** (<<https://github.com/opentimestamps/java-opentimestamps>>): ferramenta CLI Java para realização e validação de PoEs utilizando a API OpenTimestamps. Possui disponibilidade para integração em aplicações desenvolvidas em base Java.
- **Kit de Desenvolvimento Java (JDK)** (<<https://www.java.com>>): inclui o Ambiente de Execução do Java (JRE), seu compilador e APIs. Necessário para o desenvolvimento de qualquer aplicação com base em Java.
- **Kotlin** (<<https://kotlinlang.org>>): linguagem de programação de sintaxe concisa e segura, que reduz a quantidade de código desnecessário e evita erros como valores nulos. Compilada e estaticamente tipada, disponível para diversas plataformas. Possui interoperabilidade com Java, tendo seu código compilado compatível com a JVM.

- **Libsodium** (<<https://libsodium.gitbook.io>>): biblioteca de software moderna e fácil de usar para criptografia, descritografia, assinaturas, *hashing* de senha, dentre outras operações. Tem como objetivo fornecer todas as operações básicas necessárias para construir ferramentas criptográficas de alto nível.
- **Retrofit** (<<https://square.github.io/retrofit>>): biblioteca para configuração de clientes com Protocolo de Transferência de Hipertexto (HTTP) *typesafe* em aplicações com base em Java.
- **SQLite** (<<https://sqlite.org>>): biblioteca em linguagem C que implementa um mecanismo de banco de dados de Linguagem de Consulta Estruturada (SQL) pequeno, rápido, independente, de alta confiabilidade e com recursos completos.
- **SciPo-Farmácia** (<<http://www.nilc.icmc.usp.br/scipo-farmacia>>): conjunto de ferramentas computacionais construídas para auxiliar na escrita acadêmica, desenvolvido a partir do Suporte Computacional à Escrita Científica em Português – SciPo. Fornece apoio para estruturar textos de acordo com as diretrizes de “boa escrita” propostas pela literatura especializada. Resultado de um projeto realizado no Núcleo Interinstitucional de Linguística Computacional no Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) de São Carlos, em parceria com a Faculdade de Ciências Farmacêuticas da USP-São Paulo.
- **Lucidchart** (<[www.lucidchart.com](http://www.lucidchart.com)>): plataforma online para criação de diagramas UML, mapas mentais, fluxo de dados, dentre outros.

## 3.2 Procedimentos

### 3.2.1 Estudo de caso

O estudo de caso é considerado uma metodologia eficaz para investigar e compreender questões complexas em ambientes do mundo real. [Harrison et al. \(2017\)](#) a descrevem como uma forma versátil de investigação qualitativa, mais adequada para uma investigação abrangente e profunda de uma questão complexa em determinado contexto. O requisito essencial para o emprego do estudo de caso é a compreensão de fenômenos complexos. [Heale e Twycross \(2018\)](#) definem um estudo de caso como um estudo intensivo sobre uma unidade, onde se tem como objetivo generalizar em várias unidades. [Hancock e Algozzine \(2016\)](#) afirmam que pesquisa de estudo de caso é geralmente mais exploratória do que confirmatória. É ricamente descritiva por se basear em fontes de informação profundas e variadas, podendo envolver coleta e a análise de informações de fontes como entrevistas, observações e documentos.

Segundo [Harrison et al. \(2017\)](#) o foco decai mais no que é estudado (o caso) do que em como é estudado (o método). O estudo de caso possui uma versatilidade prática em sua

abordagem agnóstica, em que não é atribuído uma posição metodológica fixa. As etapas ao utilizar esta metodologia não se difere de outras. A princípio é necessário definir o caso único ou grupo de casos do estudo. É então realizada uma pesquisa para determinar o que se sabe e o que ainda não se sabe sobre o(s) caso(s), para se estabelecer uma compreensão básica e formar questões de pesquisa. Então é iniciada a fase de obtenção dos dados, que são frequentemente qualitativos. Por fim, os dados são sumarizados, reportados e confirmados (HEALE; TWYLCROSS, 2018; HARRISON et al., 2017). Harrison et al. (2017) apresentam diversos exemplos de ações a serem tomadas em cada uma das etapas supracitadas.

A obtenção dos dados deve incluir o maior número possível de documentos relevantes, englobando material extraído da internet, registros públicos e privados, evidências físicas e instrumentos criados pelo pesquisador. Os instrumentos criados pelo pesquisador podem fornecer um meio poderoso de coleta de informações. É desejável também identificar participantes-chave do contexto estudado, cujos conhecimentos e opiniões podem fornecer percepções importantes sobre as questões da pesquisa. Diferente de algumas outras metodologias onde os dados são examinados somente no final, a pesquisa de estudo de caso envolve o exame e a interpretação contínuos dos dados para chegar a conclusões provisórias e refinar as questões de pesquisa. Por fim, fazer estudos de caso cria oportunidades para explorar questões adicionais pelo ato de investigar um tópico em detalhes (HANCOCK; ALGOZZINE, 2016).

### 3.2.2 Análise sobre APIs de realização de PoE

Como detalhado anteriormente, o uso de APIs para realização de PoE traz diferentes benefícios, dos quais destacamos (1) o fato da API concatenar *hashes* de diversos clientes em uma única árvore de Merkle e ancorar somente a raiz de Merkle na *blockchain*, diluindo custos com transações; (2) APIs pré-existentes já foram testadas e utilizadas por outras aplicações, assim a chance de encontrar *bugs* com elas é relativamente menor do que caso se optasse pela implementação do zero; e (3) poupa-se esforços no desenvolvimento da aplicação, já que a comunicação com a *blockchain* é realizada pela API, reduzindo o tempo de desenvolvimento. Visto que há variadas APIs disponíveis atualmente, foi realizada uma análise e comparação com os seguintes critérios:

- **Resolução**, correspondente à periodicidade com que a API envia transações à rede do *blockchain* (para ancorar a raiz de Merkle);
- **Latência**, referindo-se ao tempo que as transações são efetivadas em um novo bloco após enviadas à rede do *blockchain* (a depender do valor que a API investe em cada transação);
- **Solidez**, considerando a(s) *blockchain(s)* públicas em que a API realiza PoE e o grau de maturidade dessa(s) *blockchain(s)*;

- **Forma de ancoragem**, validando a operação utilizada pela API para registrar PoEs na(s) *blockchain(s)*;
- **Codificação**, para validar se a API é de código aberto ou privado e se recebe manutenção regularmente;
- **Documentação**, a partir da análise dos documentos disponibilizados pela API; e
- **Custo**, referindo-se as despesas por mês para utilização da API, como transações na *blockchain*, servidores, planos de utilização, dentre outras.

Os critérios para a seleção das APIs analisadas foram (1) ser possível realizar a validação da PoE diretamente na *blockchain*, sem intermédio da API; (2) que haja documentação o suficiente para compreender o funcionamento da API e poder avaliá-la; e (3) que a API realize operações de *commitments* com os dados recebidos, de modo a concatenar diversos *hashes* e enviar somente o resumo para a *blockchain*, possibilitando a realização de grandes volumes de PoEs sem sobrecarregar a *blockchain*. Com isso, as APIs selecionadas para análise foram a Chainpoint, OpenTimestamps e OriginStamp. As análises foram feitas principalmente por meio de suas documentações, incluindo site oficial, artigos publicados e repositórios de código. Além disso, entramos em contato com os desenvolvedores da API OriginStamp e OpenTimestamps para esclarecer o código usado pela API, o procedimento usado para realizar PoE e tempo de latência de PoE. Por fim, realizamos testes com OpenTimestamps para gerar dados que forneceram análises estatísticas. Os testes só foram possíveis com OpenTimestamps pois apenas esta API realiza PoEs gratuitamente e possui servidores públicos.

Para calcular os custos de realização de PoEs, é necessário considerar a granularidade dos dados: o volume de dados inseridos em uma única PoE. A granularidade dos *logs* pode ser medida pelo número de *logs* ou pela janela de tempo usada para obtê-los. Uma baixa granularidade (por exemplo, um *log* por PoE) pode reduzir a segurança em relação à privacidade de dados. Por outro lado, com alta granularidade (por exemplo, uma PoE por dia), a especificidade da verificação de dados é reduzida, tornando toda a janela de tempo invalidada se houver fraude em uma pequena quantidade de dados. Em discussão com a SIn, concluímos que uma granularidade de 25 minutos é adequada para a maioria dos sistemas. Portanto, usamos essa granularidade em nossos testes e análises, exigindo 57,6 PoEs por dia e 1.728 PoEs por mês. Como ambas as APIs requerem um dispositivo cliente que realiza o PoE, armazena os atestados localmente e realiza a validação, não o consideramos no cálculo de custo.

### 3.2.3 Aplicação desenvolvida: Auditchain

Como estudo de caso deste trabalho, foi desenvolvida uma aplicação CLI para realização de PoEs de forma integrada ao Elasticsearch, a Auditchain. Sua arquitetura e fluxo de dados são apresentados em detalhes na [Seção 4.2](#). Os requisitos para o seu desenvolvimento foram

levantados e refinados de acordo com revisão da literatura, contato constante com a SIn e documentos analisados. A execução dos testes da Auditchain contaram com uma instância n2d-standard-2 da GPC e foram realizados do dia 17/03 a 26/03/2021. A aplicação realizou PoEs com uma granularidade de 25 minutos de *logs* gerados em laboratório. Ela foi executada a cada 10 minutos com auxílio do agendador de tarefas do Linux (cron), seja pra realizar PoE ou atualizar dados de PoEs anteriores. Para análise teórica acerca das soluções utilizadas no desenvolvimento desta aplicação, levantamos um modelo adversarial com possíveis ataques a sistemas. Para este modelo, listamos abaixo os componentes de arquitetura envolvidos e as possíveis estratégias de ataque dependendo dos acessos obtidos pelo adversário. Ressaltamos que é inviável impedir um ataque caso adversário tenha acesso a todos os componentes da arquitetura ao mesmo tempo, o que foge ao escopo deste trabalho.

- **Componentes da arquitetura:**

1. Aplicação – sistema ao qual se busca atestar a integridade dos dados;
2. Base de dados – base com dados da aplicação;
3. Arquivos de *log* – conjunto dos *logs* gerados localmente pela aplicação;
4. Servidor Elasticsearch – onde os *logs* são centralizados e indexados após serem registrados nos arquivos de *log*;
5. Base de atestados – base mantida pelo Auditchain contendo dados de atestados gerados após realização da PoE; e
6. Servidores de Calendário – servidores da API OpenTimestamps responsáveis por ancorar a PoE na *blockchain*.

- **Estratégias de ataque:**

1. Com acesso ao servidor da Aplicação um adversário pode forjar a geração de *logs* antes destes serem registrados, buscando ocultar suas ações futuras;
2. Com acesso à base de dados um adversário pode operar os dados sem a permissão do proprietário;
3. Com acesso aos arquivos de *logs* um adversário pode forjar *logs* do passado para limpar rastros de uma operação ilegal na aplicação;
4. Com acesso ao servidor Elasticsearch um adversário pode forjar *logs* do passado para limpar rastros de uma operação ilegal na aplicação;
5. Com acesso à base de atestados um adversário pode forjar atestados do passado para limpar rastros de uma operação ilegal na aplicação; e
6. Com acesso aos servidores de calendário o adversário pode interceptar *hashes* por ele recebidos.

**• Modelo adversarial:**

1. Adversário com acesso ao servidor da aplicação passa a forjar *logs*;
2. Adversário com acesso à base de dados forja um registro;
3. Adversário com acesso aos arquivos de *logs* forja *logs* já registrados;
4. Adversário com acesso ao servidor Elasticsearch forja *logs* após certo tempo destes terem sido indexados;
5. Adversário com acesso ao servidor Elasticsearch forja *logs* logo após estes terem sido indexados;
6. Adversário com acesso à base de atestados apaga o atestado correspondente a um determinado intervalo de tempo;
7. Adversário com acesso à base de atestados substitui atestados verdadeiros por atestados forjados através da realização de PoE paralela de *logs* verossímeis por determinado período de tempo;
8. Adversário com acesso aos arquivos de *logs* e à base de atestados forja um *log* e deleta o atestado antes que este seja ancorado na *blockchain*; e
9. Adversário com acesso aos servidores de calendário intercepta *hashes* com a intenção de identificar padrões nos *logs* e forjar atestados.

# Capítulo 4

## DESENVOLVIMENTO

---

---

Neste capítulo são apresentadas as etapas do desenvolvimento deste trabalho. A princípio, as APIs Chainpoint, OpenTimestamps e OriginStamp são apresentadas e analisadas de acordo com os parâmetros definidos. Então esses parâmetros são comparados e uma discussão a respeito é apresentada. Após análise das APIs, optou-se por utilizar neste trabalho a OpenTimestamps, e a partir dela foi construída a solução Auditchain, que serviu como base de estudo de caso. Os detalhes a respeito da Auditchain são apresentados incluindo sua arquitetura, fluxo de dados e fluxogramas de suas principais funcionalidades. Por fim, os modelos de ataque apresentados na metodologia são discutidos em comparação com as soluções obtidas com a Auditchain.

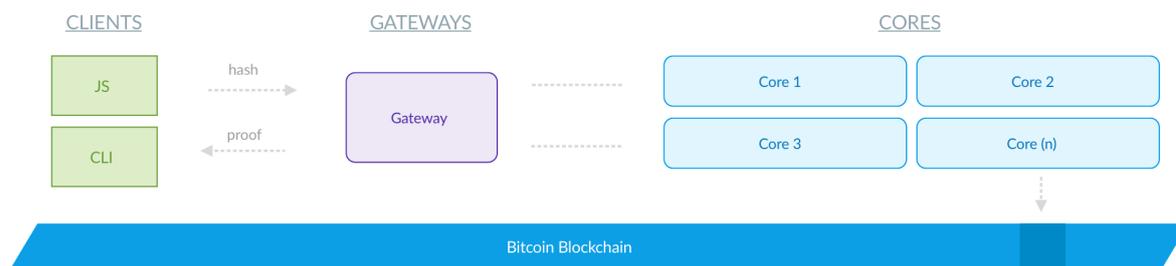
### 4.1 Análise das APIs

#### 4.1.1 Chainpoint

Chainpoint é uma ferramenta de código aberto para realização de PoE de qualquer dado, arquivo ou processo através de uma rede de nós mundialmente distribuída. Para utilizar essa ferramenta é necessário a configuração de ao menos um servidor *gateway* que irá prover a API REST para realização das PoEs, e um cliente que será responsável por enviar os *hashes* dos dados a este servidor por meio de requisições HTTP. Os servidores *gateways* são responsáveis por agregar esses *hashes* em uma árvore de Merkle e enviar a raiz de Merkle a *cores*. Os *cores*, por sua vez, publicam a raiz de Merkle em uma transação da *blockchain*. No fim do processo é disponibilizado um atestado, um conjunto de dados que possui uma série de operações que vinculam os dados criptograficamente a um bloco da *blockchain* (VAUGHAN et al., 2016; VAUGHAN et al., 2020). Uma ilustração gráfica dessa arquitetura é apresentada na Figura 18. A rede Chainpoint é composta de uma série de agregações para criação do atestado. Segundo Vaughan et al. (2020), o processo pode ser resumido em cinco passos:

1. Clientes submetem *hashes* aos *gateways*;

Figura 18 – Rede Chainpoint



Fonte: Vaughan et al. (2020)

2. *Gateways* agregam esses *hashes* em uma árvore de Merkle e periodicamente enviam a raiz de Merkle a um ou mais *cores*;
3. *Cores* realizam um rodada adicional de agregações e enviam a raiz de Merkle em transações em uma *blockchain* baseada na tecnologia Tendermint<sup>1</sup>, chamada de Calendário;
4. A cada hora um *core* é eleito para ancorar o estado do calendário na *blockchain*, aguardar pela confirmação e então escrever os resultados no Calendário;
5. *Gateways* e clientes usam os dados do Calendário para construir o atestado final.

Após a finalização desses passos, o que segundo Vaughan et al. (2020) leva em torno de 90 minutos, é necessário realizar uma requisição GET ao *gateway* para obter os dados do atestado e armazená-lo localmente, pois o *gateway* mantém esses dados por somente 24 horas. Com esses dados é possível recriar a raiz de Merkle enviada à *blockchain* e validar se esta consta no bloco indicado (VAUGHAN et al., 2016). Segundo Vaughan et al. (2016), Chainpoint 2.0 suporta ancoragem em transações na Bitcoin com uma saída OP\_RETURN, e ancoragem em transações na Ethereum usando campos de dados. No entanto, segundo Vaughan et al. (2020), atualmente Chainpoint realiza ancoragem somente na Bitcoin, portanto será considerado aqui o documento mais atual.

Como as transações na Bitcoin são pagas, os *cores* cobram uma taxa de ancoragem que por padrão atualmente é configurada para 2 satoshis<sup>2</sup>, podendo ser alterada pelos *cores* para se adaptar às condições de mercado (VAUGHAN et al., 2016). Assim, para a execução do servidor *gateway* é necessária a criação de uma carteira virtual, a qual deverá ter crédito o suficiente para pagamento das taxas de transações. Para realização de 1728 PoEs são necessários 3456 satoshis, com custo aproximado de US\$1,25<sup>3</sup>. Além dos custos com transações, para a instalação do servidor *gateway* é necessário um *hardware* com ao menos 4GB de RAM, 1 core de CPU,

<sup>1</sup> <<https://github.com/tendermint/tendermint>>

<sup>2</sup> Atualmente a menor unidade registrada da moeda Bitcoin

<sup>3</sup> Cotação do satoshi em 03/02/2021: US\$1 = 2761 satoshis

128+ GB SSD e um endereço IPv4 público. Essas configurações equivalem a uma instância de máquina virtual e2-medium com 128GB de SSD na GCP, com custo de US\$46,22 por mês. Ainda que a própria organização forneça ferramentas cliente de código aberto, um desenvolvedor pode implementar sua própria ferramenta com a tecnologia que desejar, bastando que essa se comunique com o servidor gateway por meio da API fornecida. Assim, temos as seguintes variáveis para sumarizar essa ferramenta:

- Resolução: 60 minutos
- Latência: 30 minutos
- Solidez: Bitcoin
- Forma de Ancoragem: A raiz de Merkle é publicada em transações na Bitcoin com uma saída OP\_RETURN
- Codificação: Código aberto para as seguintes ferramentas:
  - Cliente JavaScript:
    - \* Acesso: <<https://github.com/chainpoint/chainpoint-js>>
    - \* Última atualização: 19/06/2020 (Acesso em 04/01/2021)
  - Cliente CLI:
    - \* Acesso: <<https://github.com/chainpoint/chainpoint-cli>>
    - \* Última atualização: 17/06/2020 (Acesso em 04/01/2021)
  - Servidor *Gateway*
    - \* Acesso: <<https://github.com/chainpoint/chainpoint-gateway>>
    - \* Última atualização: 27/05/2020 (Acesso em 04/01/2021)
  - *Core*:
    - \* Acesso: <<https://github.com/chainpoint/chainpoint-core>>
    - \* Última atualização: 20/12/2020 (Acesso em 04/01/2021)
- Documentação: Possui *White Paper*, site e vasta documentação nos repositórios, mas não há contato para suporte
- Custo: Por mês, aproximadamente US\$46,22 para manter um servidor com configurações mínimas estipuladas e US\$1,25 com transações

### 4.1.2 OpenTimestamps

OpenTimestamps é uma ferramenta de código aberto para realização de PoE desenvolvida por Peter Todd, ex-desenvolvedor do Projeto Bitcoin. Ela é baseada na utilização de servidores de agregação e servidores de calendário que trabalham em conjunto para realizar operações de *commitment* em *hashes* recebidos por clientes e a ancoragem da raiz de Merkle desses *hashes* na *blockchain*. Ainda que seja possível implantar seu próprio servidor de calendário, não há necessidade, pois existem servidores de calendário públicos disponibilizados gratuitamente, mantidos por meio de doações. Atualmente há quatro servidores de calendário públicos disponíveis:

- alice (<[alice.btc.calendar.opentimestamps.org](http://alice.btc.calendar.opentimestamps.org)>)
- bob (<[bob.btc.calendar.opentimestamps.org](http://bob.btc.calendar.opentimestamps.org)>)
- finney (<[finney.calendar.eternitywall.com](http://finney.calendar.eternitywall.com)>)
- catallaxy (<[ots.btc.catallaxy.com](http://ots.btc.catallaxy.com)>)

Os servidores de calendário públicos trabalham simultaneamente, provendo redundância e alternando qual realiza a ancoragem da raiz de Merkle. Estes servidores realizam a ancoragem somente na Bitcoin, por meio de uma operação OP\_RETURN. Já demais servidores implementados individualmente podem utilizar outras operações e até mesmo outras *blockchains*<sup>4</sup>. Com isso, os servidores de calendário públicos fazem duas promessas: (1) realizar PoEs na Bitcoin da raiz de Merkle de todo *hash* recebido pelos clientes em um tempo razoável; e (2) árvores de Merkle completas serão disponibilizadas ao público e mantidas indefinidamente (TODD, 2016).

O contato com os servidores de calendário é realizado por meio de bibliotecas cliente de código aberto disponibilizadas pela própria organização. Atualmente há bibliotecas disponibilizadas para as linguagens Java, JavaScript (com a plataforma NodeJS), Python e Rust, além de ferramentas de linha de comando escritas nessas mesmas linguagens, que servem como exemplos de uso das bibliotecas. Essas bibliotecas são também as responsáveis por baixar os dados da PoE e realizar sua validação. Caso a máquina executando a ferramenta cliente seja um nó Bitcoin, a validação é realizada direto na rede, caso contrário a validação é realizada por meio da API Blockstream<sup>5</sup>.

Com a utilização de calendários públicos, OpenTimestamps oferece uma conveniência a mais em comparação a demais APIs. É possível criar uma PoE verificável por terceiros em questão de segundos, sem a necessidade de esperar a confirmação da Bitcoin. Essa funcionalidade é fundamental para aplicações que precisam de uma validação rápida e não podem aguardar por sistemas de consenso descentralizados. É uma proposta de segurança incremental, com a primeira validação nos servidores de calendário OpenTimestamps públicos e posteriormente na

<sup>4</sup> Comunicação pessoal de Petter Todd, em 26 de Fevereiro de 2021, recebida por correio eletrônico

<sup>5</sup> <<https://blockstream.info/>>

Bitcoin (TODD, 2016). Contudo, essa conveniência traz o problema de pontos centrais de falha, que OpenTimestamps tenta amenizar de diferentes formas.

Primeiramente, utilizar segurança incremental com calendários públicos é opcional pois clientes podem realizar e validar a PoE sem depender de dados não ancorados em *blockchain* utilizando a *flag --wait*. Somado a isso, há redundância de ao menos dois calendários públicos que são usados simultaneamente, desse modo contanto que ao menos um calendário público esteja disponível é possível realizar e validar PoEs. Além disso há a possibilidade de configurar seu próprio calendário privado, que utiliza os calendários públicos e entra em ação somente caso os públicos fiquem inativos (TODD, 2016). Quanto aos riscos legais da utilização dos servidores de calendário, Todd (2016) comenta que são baixos pelos seguintes motivos:

- calendários não têm autoridade e portanto não podem produzir provas falsas;
- derrubar servidores de calendário seria inútil pois os dados são facilmente espelhados;
- os calendários não armazenam dados dos clientes, somente os *commitments* são armazenados permanentemente para validação da PoE pelos clientes.

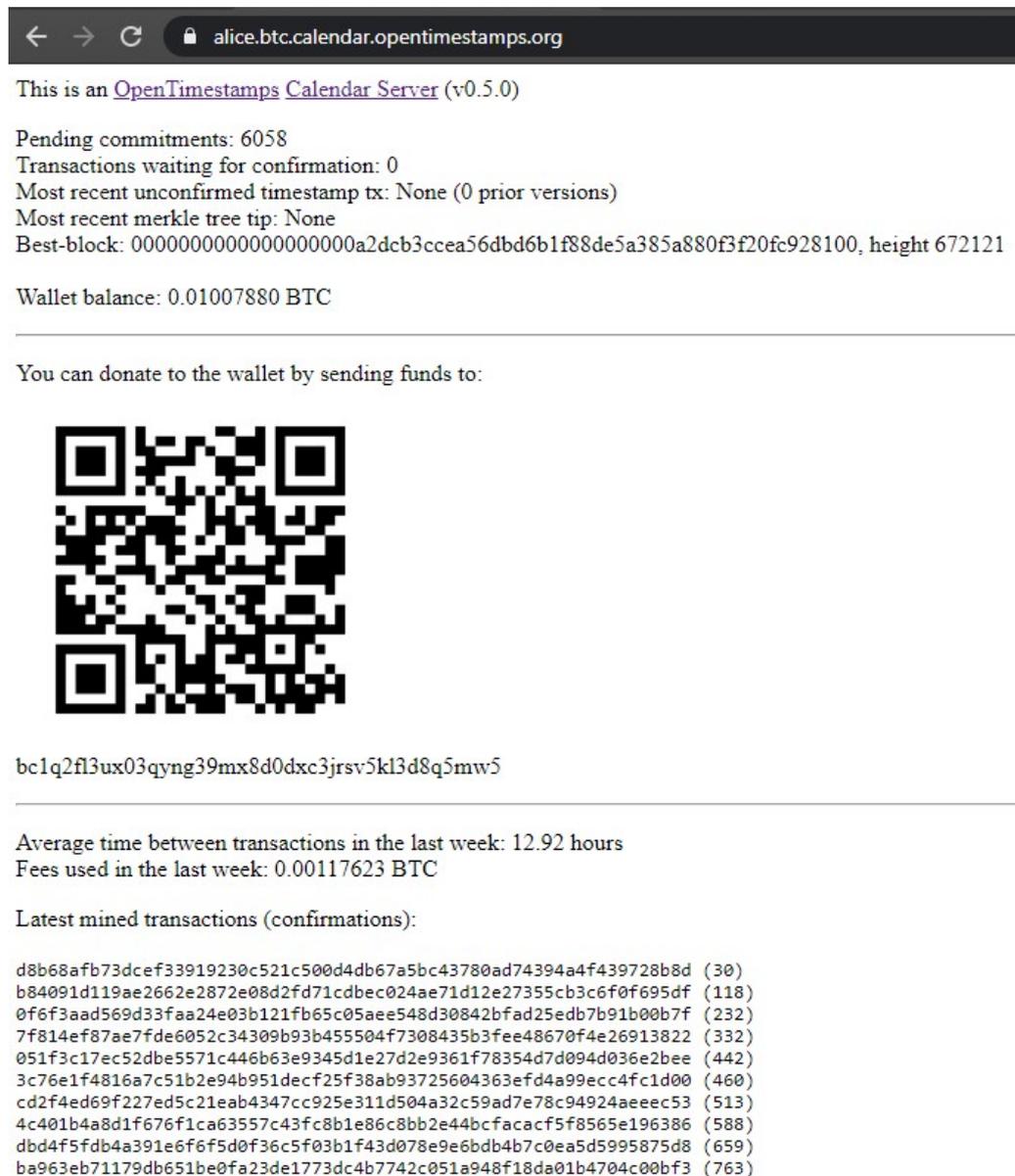
Acessando as URLs dos servidores de calendário é possível obter algumas informações referentes à realização de PoEs como: quantidade de *commitments* pendentes para ancoragem na Bitcoin; transações aguardando confirmação (serem minadas) em um bloco na Bitcoin; últimas transações minadas; saldo da carteira do servidor de calendário; endereço para realização de doações; valor gasto na última semana; e o tempo médio entre transações na última semana. Um exemplo das informações do servidor de calendário pode ser visualizado na [Figura 19](#).

A ancoragem da raiz de Merkle é realizada pelo servidor calendário em intervalos de tempo sorteados por uma distribuição uniforme no intervalo  $[i; 2i]$ , onde o valor de  $i$  é controlado pela opção `--btc-min-tx-interval` configurada no servidor. Espera-se, portanto, que a resolução média seja de  $1,5 \times i$ . Contudo, o valor de  $i$  não é tornado público na página contendo informações sobre o servidor. O valor padrão é de  $i = 6$  horas, o que levaria a uma resolução média de 9 horas considerando um único servidor calendário.

O tempo médio entre transações na última semana (em 28/03/2020) dos servidores *alice* e *bob* é compatível com a configuração padrão – 9,88 e 9,33 horas, respectivamente. Já o tempo médio entre transações do servidor *finney* – 18,67 horas – é compatível com uma resolução de 18 horas, que seria obtida configurando  $i = 12$  horas. Por fim, o servidor *catallaxy* apresenta um tempo médio entre transações de 2,67 horas, compatível com uma resolução de 2 horas, obtida com  $i = 80$  minutos (embora este seja um valor um tanto peculiar para a configuração).

A latência pode ser estimada subtraindo a resolução do tempo médio entre transações exibido na página de informações. As taxas de transação utilizadas pelos quatro servidores

**Figura 19 – Servidor de calendário OpenTimestamps Alice**



This is an [OpenTimestamps Calendar Server](#) (v0.5.0)

Pending commitments: 6058  
 Transactions waiting for confirmation: 0  
 Most recent unconfirmed timestamp tx: None (0 prior versions)  
 Most recent merkle tree tip: None  
 Best-block: 000000000000000000a2dcb3ccea56dbd6b1f88de5a385a880f3f20fc928100, height 672121

Wallet balance: 0.01007880 BTC

---

You can donate to the wallet by sending funds to:



bc1q2fl3ux03qyng39mx8d0dxc3jrsv5kl3d8q5mw5

---

Average time between transactions in the last week: 12.92 hours  
 Fees used in the last week: 0.00117623 BTC

Latest mined transactions (confirmations):

d8b68afb73dcef33919230c521c500d4db67a5bc43780ad74394a4f439728b8d (30)  
 b84091d119ae2662e2872e08d2fd71cdbec024ae71d12e27355cb3c6f0f695df (118)  
 0f6f3aad569d33faa24e03b121fb65c05aee548d30842bfad25edb7b91b00b7f (232)  
 7f814ef87ae7fde6052c34309b93b455504f7308435b3fee48670f4e26913822 (332)  
 051f3c17ec52dbe5571c446b63e9345d1e27d2e9361f78354d7d094d036e2bee (442)  
 3c76e1f4816a7c51b2e94b951decf25f38ab93725604363efd4a99ecc4fc1d00 (460)  
 cd2f4ed69f227ed5c21eab4347cc925e311d504a32c59ad7e78c94924aeec53 (513)  
 4c401b4a8d1f676f1ca63557c43fc8b1e86c8bb2e44bcfacac5f8565e196386 (588)  
 ddb4f5fdb4a391e6f6f5d0f36c5f03b1f43d078e9e6bdb4b7c0ea5d5995875d8 (659)  
 ba963eb71179db651be0fa23de1773dc4b7742c051a948f18da01b4704c00bf3 (763)

Fonte: <https://alice.btc.calendar.opentimestamps.org/>. Acesso em: 25/02/2021.

calendário são similares, portanto espera-se que estes obtenham uma latência média similar. A média aritmética entre as latências estimadas dessa forma é de 0,64 horas, ou cerca de 40 minutos.

A resolução média tende atingir valores menores conforme adicionam-se mais servidores calendário, pois o sorteio dos intervalos de tempo faz com que estes fiquem fora de sincronia, e a biblioteca cliente tenta realizar a PoE simultaneamente em todos os servidores configurados. A resolução total, considerando-se os quatro servidores calendário, pode ser estimada, portanto, somando-se as taxas de ancoragem (inverso da resolução) de cada servidor, ou seja, calculando-se  $(9^{-1} + 9^{-1} + 18^{-1} + 2^{-1})^{-1} \approx 1,29$  horas, ou cerca de 77 minutos.

Com isso, temos o seguinte resultado para essa API:

- Resolução: 77 minutos
- Latência: 40 minutos
- Solidez: Bitcoin
- Forma de Ancoragem: A raiz de Merkle é publicada em transações na Bitcoin com uma saída OP\_RETURN
- Codificação: Código aberto para as seguintes ferramentas:
  - Servidor OpenTimestamps (*Calendar*)
    - \* Acesso: <<https://github.com/opentimestamps/opentimestamps-server>>
    - \* Última atualização: 02/02/2021 (Acesso em 04/01/2021)
  - Cliente:
    - \* Acesso: <<https://github.com/opentimestamps/opentimestamps-client>>
    - \* Última atualização: 31/10/2019 (Acesso em 04/01/2021)
  - Cliente JavaScript:
    - \* Acesso: <<https://github.com/opentimestamps/javascript-opentimestamps>>
    - \* Última atualização: 12/12/2020 (Acesso em 04/01/2021)
  - Cliente Python:
    - \* Acesso: <<https://github.com/opentimestamps/python-opentimestamps>>
    - \* Última atualização: 26/06/2020 (Acesso em 04/01/2021)
  - Cliente Java:
    - \* Acesso: <<https://github.com/opentimestamps/java-opentimestamps>>
    - \* Última atualização: 15/09/2020 (Acesso em 04/01/2021)
  - Cliente Rust:
    - \* Acesso: <<https://github.com/opentimestamps/rust-opentimestamps>>
    - \* Última atualização: 08/03/2019 (Acesso em 04/01/2021)
- Documentação: Possui site com documentação resumida, listas de email, artigos publicados por Peter Todd, documentação nos repositórios e informações fornecidas pelos servidores de calendário.
- Custo: Gratuita

### 4.1.3 OriginStamp

OriginStamp é um serviço de PoE privado oferecido pela companhia OriginStamp.com, que promete realizar PoEs de forma segura e anônima. A realização de PoE nesse serviço segue o padrão dos demais: gera-se um *hash* dos dados que será concatenado em uma árvore de Merkle a qual terá sua raiz armazenada em uma transação na *blockchain*. Além da possibilidade de submeter *hashes* através de sua API, é possível também realizar a PoE de arquivos por meio de sua plataforma web ou aplicação móvel<sup>6</sup>. Por ser um serviço privado, para utilizá-lo é necessário a criação de um usuário e a utilização da API exige uma chave que é associada ao usuário, de modo a poder identificar suas transações (HEPP et al., 2018a).

Há um repositório de código aberto<sup>7</sup> cuja ultima atualização foi realizada em 09/10/2016. Esse repositório não é mais utilizado e o serviço atualmente conta com um repositório privado com mais de 200 mil linhas de código<sup>8</sup>. A PoE é realizada em quatro *blockchains*: Bitcoin, Ethereum, Aion e Südkurier (uma *blockchain* de mídia física<sup>9</sup>). As informações necessárias para reconstrução da raiz de Merkle e comprovação do *hash* diretamente na *blockchain* podem ser baixadas e armazenadas localmente. Segundo Hepp et al. (2018a), a ancoragem na Bitcoin é possível tanto com uma saída OP\_RETURN quanto com a criação de novos endereços e transferência de um valor mínimo para estes, mas a preferência é para a operação de OP\_RETURN para evitar UTXO. Já na Ethereum e Aion, a ancoragem é feita por meio de contratos inteligentes, e no jornal Südkurier o *hash* é impresso diretamente. Aion e Südkurier não são consideradas aqui como *blockchains* maduras, conforme detalhado na Seção 4.1.5.

A ancoragem na Bitcoin e Südkurier é realizada uma vez ao dia (na Südkurier com exceção de domingos e feriados), Ethereum a cada 60 minutos e Aion a cada 10 minutos. A latência na Bitcoin é de 14 minutos, na Ethereum 2 minutos, Aion 10 segundos e Südkurier 24 horas (impresso no dia seguinte). Para lidar com os custos a companhia disponibiliza planos que variam de gratuito a US\$249 ao mês, com a possibilidade de 5 a 100.000 PoEs por mês, respectivamente<sup>10</sup>. Para realizar 1728 PoEs por mês é necessário o plano Pro, com a possibilidade de até 10 mil PoEs por mês, com custo de US\$79. A vantagem de ser um serviço privado é a possibilidade de contar com o suporte da companhia, porém há a desvantagem de não haver total clareza de todo procedimento realizado com os dados por não se ter acesso ao código da aplicação. Ainda assim, há uma vasta documentação no site, possibilidade de contato com o suporte e também uma página de apoio a pesquisadores com artigos publicados envolvendo a ferramenta<sup>11</sup>.

<sup>6</sup> <<https://goo.gl/nQkx5A>>

<sup>7</sup> <<https://github.com/OriginStamp/os.core.api>>

<sup>8</sup> Comunicação pessoal de Thomas, após contato com suporte técnico da OriginStamp, em 28 de Janeiro de 2020, recebida por correio eletrônico

<sup>9</sup> Embora a documentação da API refira-se à Südkurier como uma *blockchain* de mídia física, consideramos que esta assemelha-se mais a um serviço de *timestamp* físico, tendo como base somente a impressão do *hash* em um jornal. Esse jornal pode ser consultado em <[blockchain.suedkurier.de](http://blockchain.suedkurier.de)>.

<sup>10</sup> <<https://originstamp.com/pricing/>>

<sup>11</sup> <<https://dke.uni-wuppertal.de/en/projects/originstamp.html>>

Dessa forma, temos o seguinte resumo:

- Resolução: Bitcoin 24 horas, Ethereum 60 minutos, Aion 10 minutos e Suedkurier 24 horas exceto domingos e feriados.
- Latência: Bitcoin 14 minutos, Ethereum 2 minutos, Aion 10 segundos e Südkurier 24 horas
- Solidez: Bitcoin e Ethereum
- Forma de Ancoragem: A raiz de Merkle é publicada em transações na Bitcoin com uma saída OP\_RETURN, na Ethereum e Aion através de contratos inteligentes, e é impressa diretamente no jornal Südkurier.
- Codificação: Código privado
- Documentação: Possui site, páginas dedicadas para documentação da ferramenta e da API REST, email para contato com suporte e página para apoio a pesquisadores com artigos publicados envolvendo a ferramenta.
- Custo: US\$79 com a contratação do plano Pro.

#### 4.1.4 Comparativo

Acima foram avaliadas individualmente três APIs selecionadas de acordo com os critérios apresentados, sendo elas Chainpoint, OpenTimestamps e OriginStamp. A avaliação individual analisou a API de forma geral, demonstrando sua arquitetura e modo de trabalho com foco nas variáveis de resolução, latência, solidez, forma de ancoragem, codificação, documentação e custo. Um resumo comparativo dessas variáveis é apresentado na [Tabela 1](#). O aprofundamento nessas variáveis é realizado aqui, comparando pontos fortes e fracos de cada API. Essa comparação visa auxiliar o processo de seleção da API que melhor se encaixa no contexto a ser empregada.

Dentre as três APIs, o menor tempo para realização e ancoragem na Bitcoin de uma PoE é realizado com a Chainpoint, com um total de 90 minutos (resolução + latência). Embora a OriginStamp tenha uma latência para Bitcoin de 14 minutos, sua resolução é de 24 horas, tornando o tempo total de aproximadamente 1454 minutos para efetivação de uma PoE nesta *blockchain*. Caso haja necessidade de validação em um intervalo menor, a API OriginStamp deverá recorrer à *blockchain* Ethereum, que necessita de aproximadamente 62 minutos para efetivação de uma PoE. Ainda que seja possível recorrer à *blockchain* Aion para validações em intervalos de tempo menores, essa *blockchain* não é considerada madura conforme critérios analisados na [Seção 4.1.5](#). Ainda em relação aos tempos de resolução e latência, é importante ressaltar que os tempos levantados para as APIs Chainpoint e OriginStamp se baseiam unicamente em informações fornecidas por seus criadores, através da documentação ou contato realizado. Somente foi

**Tabela 1 – Análise das APIs - Visão geral**

Critérios de Análise	APIs		
	<i>Chainpoint</i>	<i>OpenTimestamps</i>	<i>OriginStamp</i>
Resolução	60 min	77 min	Bitcoin 24h, Ethereum 60 min, Aion 10 min, Südkurier 24h
Latência	30 min	40 min	Bitcoin 14 min, Ethereum 2 min, Aion 10 seg, Südkurier 24h
Solidez	Bitcoin	Bitcoin	Bitcoin e Ethereum
Ancoragem	OP_RETURN	OP_RETURN	Bitcoin: OP_RETURN, Ethereum e Aion: contratos inteligentes, Südkurier: impresso
Codificação	código aberto	código aberto	código privado
Documentação	<i>white paper</i> , site e repositórios Git	site, lista de email, artigos de Peter Todd, repositórios Git e servidores de calendários	site, documentação, suporte e página de suporte a pesquisadores
Custos p/ mês	US\$46,22 com servidor e ≈ US\$1,25 com transações	Gratuita	Plano Pro: US\$79

possível realizar testes para análise e validação destas variáveis para a API OpenTimestamps, por esta ser a única que realiza PoE de forma gratuita. Portanto é preciso considerar um possível viés e contar que estes valores poderiam ser diferentes com a execução de testes.

A respeito da solidez, as três APIs realizam a PoE na Bitcoin e a OriginStamp realiza também na Ethereum. Algumas das documentações das APIs Chainpoint e OpenTimestamps falam sobre a intenção de realizar PoE também na Ethereum, mas até o momento não foi encontrado nada concreto a respeito. A OriginStamp realiza PoE também nas *blockchains* Aion e Südkurier, mas ainda que essas *blockchains* possibilitem atestações extras das PoEs, elas não são consideradas maduras conforme apresentado na [Seção 4.1.5](#). Com a análise das operações de ancoragem é possível validar se alguma API gera UTXOs que nunca serão gastas, queimando dinheiro da *blockchain*. Segundo os documentos analisados e contatos realizados com os desenvolvedores das APIs, ambas utilizam uma operação OP\_RETURN para ancorar *hashes* na Bitcoin, e a OriginStamp usa contratos inteligentes na Ethereum e Aion. Portanto, não foram encontrados problemas relacionados a UTXO.

Por serem APIs de código aberto, Chainpoint e OpenTimestamps oferecem maior segurança quanto à clareza das operações realizadas. Além disso essas APIs acabam criando comunidades de desenvolvedores interessados em suas evoluções, o que auxilia na correção de *bugs* e a acrescentar esquemas de segurança. A OriginStamp, por outro lado, ainda que não

ofereça tanta clareza nos processos em comparação às demais APIs, oferece uma equipe de suporte para auxiliar desenvolvedores em sua utilização. Isso pode auxiliar também a acelerar a correção de *bugs*, já que há um time totalmente alocado para manutenção da ferramenta. Além disso, ela oferece suporte a pesquisadores interessados na temática de realização de PoEs, com um site que realiza a indexação de arquivos publicados que citam essa API.

Em relação aos custos, como a OpenTimestamps oferece servidores de calendário gratuitos, não há custos para utilizá-la. Para utilizar a Chainpoint executando um servidor *gateway* com as configurações mínimas especificadas somando custos com transações, haveria um custo de aproximadamente US\$47,5 por mês. Para OriginStamp, é necessário o Plano Pro, que permite até 10 mil PoEs por mês e tem um custo de US\$79. No entanto, o Plano Pro é necessário para que seja possível atingir a granularidade de 25 minutos. Para reduzir esse custo seria possível adotar o Plano Grow, que possibilita a realização de até 1000 PoEs por mês. Para isso seria necessária uma granularidade de 45 minutos, que realizaria no máximo 992 PoEs em um único mês (considerando meses com 31 dias). O Plano Grow possui um custo de US\$39 por mês, tornando essa opção mais barata que a Chainpoint, pois mesmo com uma granularidade de 45 minutos os custos com a API Chainpoint não sofreriam grandes alterações.

Das conclusões possíveis a se chegar com essa análise, destacamos: (1) as variáveis que mais influenciam na escolha de uma dessas três APIs são custo, codificação, latência e resolução, já que as demais não possuem diferenças tão grandes de uma API para outra; (2) para sistemas públicos ou governamentais, as APIs Chainpoint e OpenTimestamps são preferíveis pelo fato de possuir código aberto e assim ter maior clareza nas operações realizadas; e (3) a API OpenTimestamps é preferível em cenários onde se busca a realização de PoEs com custo mínimo, já que esta possui servidores de calendário gratuitos. A partir destas conclusões, a API selecionada para aplicação neste trabalho foi a OpenTimestamps.

#### 4.1.5 Análise de maturidade das *blockchains*

Para verificar a maturidade de uma *blockchain* analisamos: a taxa de *hashes* por segundo (*hash rate*); a complexidade computacional do cálculo de um *hash* no caso de PoW; ou o valor monetário total escorado no caso de PoS. As *blockchains* encontradas durante a análise das APIs foram: Bitcoin, Ethereum, Aion e Südkurier. A Südkurier é uma *blockchain* de mídia física. Basicamente, um jornal em que a raiz de Merkle é impressa diariamente. Embora se trate de uma proposta interessante, é difícil comparar a dificuldade de se forjar uma edição de um jornal com a dificuldade de se forjar blocos em *blockchains* digitais. Não se sabe como exatamente o *hash* é impresso no jornal, nem se há ou não encadeamento com o *hash* de dias anteriores, nem o procedimento para verificar a validade desse encadeamento, nem quais cuidados existem para preservar os exemplares das diversas edições passadas do jornal. Portanto, foge ao escopo deste trabalho declarar Südkurier como uma *blockchain* madura.

Na [Figura 20](#) é exibido um gráfico com histórico da evolução do *hashrate* no último ano

da Bitcoin, na [Figura 21](#) da Ethereum e na [Figura 22](#) da Aion. Conforme apresentado nestes gráficos, nos últimos três meses a Bitcoin manteve seu *hash rate* acima dos 100 *exahash/s*, seguida pela Ethereum que se manteve acima de 250 *terahash/s* e por último a Aion se manteve acima de 300 *gigahash/s*.

Ethereum e Aion trabalham com funções *hash* do tipo *memory-hard* para PoW, que são difíceis de paralelizar e exigem maior esforço computacional, ao passo que Bitcoin utiliza SHA-256, uma função *hash* tradicional, eficiente e facilmente implementável em hardware especializado. Portanto, é natural que Ethereum e Aion apresentem um *hash rate* inferior ao do Bitcoin. Ainda assim, o *hash rate* de Aion é três ordens de grandeza inferior ao do Ethereum, evidenciando um nível de maturidade muito inferior. No entanto, Aion combina PoW e PoS em seu algoritmo de consenso, de forma que é necessário analisá-la também sob o ponto de vista do PoS.

Em 29/03/2020, Aion possuía um valor total escorado de cerca de US\$ 500 mil<sup>12</sup>. Esse valor é inferior à receita anual de uma empresa de pequeno porte segundo a legislação brasileira, além de ser diversas ordens de grandeza inferior ao volume de Bitcoin e de Ethereum movimentado ao longo de 24 horas – US\$ 61 bilhões<sup>13</sup> e US\$ 33 bilhões<sup>14</sup>, respectivamente. Desta forma, considera-se Aion uma *blockchain* sustentada por uma rede imatura tanto em termos de poder computacional quanto de valor total escorado, quando comparada às alternativas Bitcoin e Ethereum.

## 4.2 Auditchain

Com base na análise das APIs chegou-se a conclusão que a API OpenTimestamps é a mais adequada para este trabalho, tendo em vista que (1) é possível realizar PoEs de forma gratuita, já que essa API disponibiliza servidores de calendário públicos mantidos por doação; (2) há maior clareza na sua implementação por possuir repositório online de código aberto, o que pode inclusive auxiliar em disputas judiciais; e (3) possibilidade de atestar PoEs antes da ancoragem na Bitcoin com base nos dados dos calendários públicos. Além disso, com a API OpenTimestamps é possível empregar sua ferramenta CLI Java, auxiliando no desenvolvimento de uma aplicação multiplataforma com base na JVM. Dado isso, foi idealizada a arquitetura com o fluxo de dados ilustrado na [Figura 23](#).

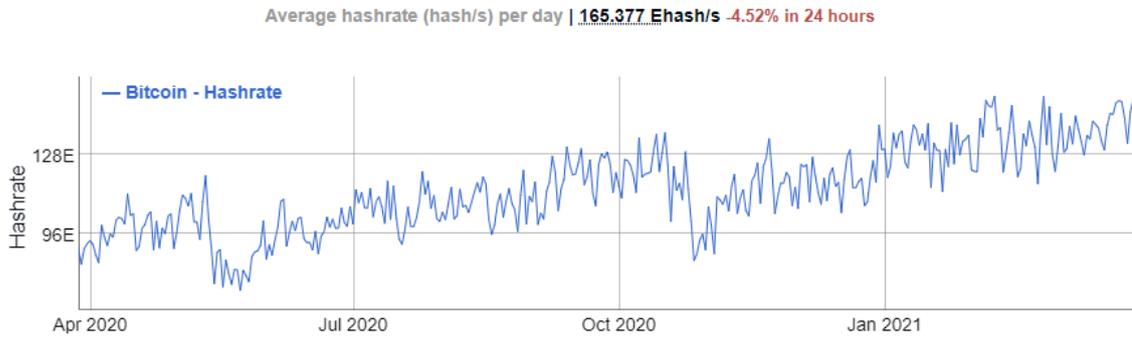
Conforme as premissas apresentadas neste trabalho, para que seja possível atestar a integridade dos dados de uma aplicação a partir da realização de PoE de seus *logs*, é essencial que a aplicação realize o registro de *logs* de todas as operações de alteração de dados. Há diversas formas para uma aplicação realizar o registro de *logs*, desde armazená-los em arquivos de texto até enviá-los a base de dados específicas para *logs*. Com as ferramentas de coleta de

<sup>12</sup> Fonte: <<https://mainnet.theoan.com/#/dashboard>> (Acesso em 29/03/2021)

<sup>13</sup> Fonte: <<https://nomics.com/assets/btc-bitcoin>> (Acesso em 29/03/2021)

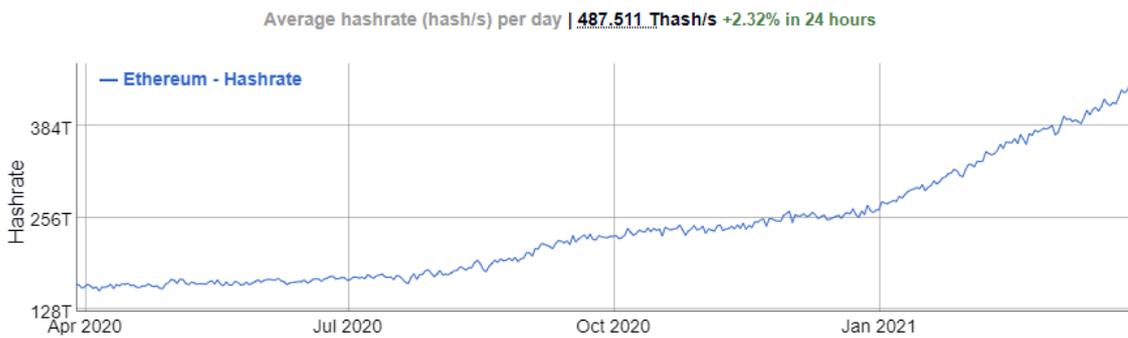
<sup>14</sup> Fonte: <<https://nomics.com/assets/eth-ethereum>> (Acesso em 29/03/2021)

**Figura 20 – Hash rate da Bitcoin nos últimos doze meses**



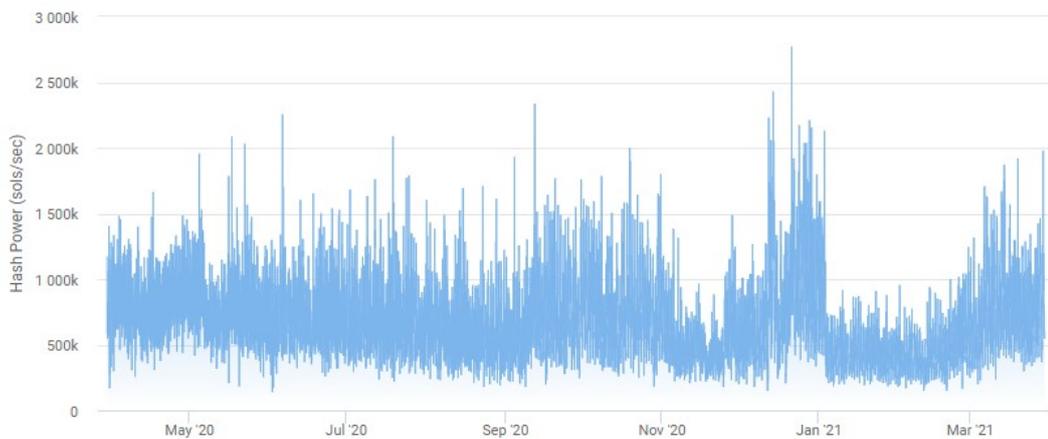
Fonte: <<https://bitinfocharts.com/comparison/bitcoin-hashrate.html#1y>>  
(Acesso em 29/03/2021)

**Figura 21 – Hash rate da Ethereum nos últimos doze meses**



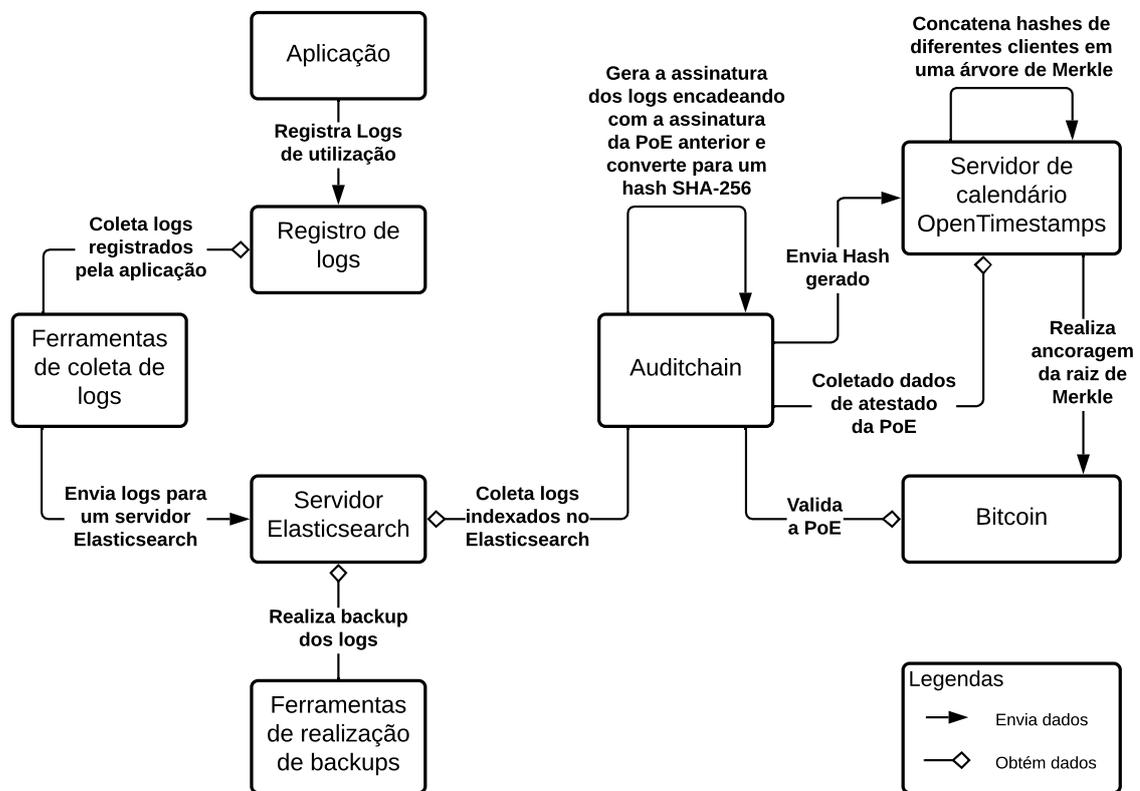
Fonte: <<https://bitinfocharts.com/comparison/ethereum-hashrate.html#1y>>  
(Acesso em 29/03/2021)

**Figura 22 – Hash rate da Aion nos últimos doze meses**



Fonte: <<https://mainnet.theoan.com/#/charts/HashingPower>>  
(Acesso em 29/03/2021)

**Figura 23 – Visão Geral do Fluxo de Dados na Auditchain**



Fonte: Próprio autor

*logs* disponibilizadas pela Elasticsearch é possível reunir *logs* de diferentes bases em um único servidor com sistema de busca robusta e otimizada para análise desse tipo de dado. Por se tratar de dados sensíveis à aplicação, é importante configurar sistemas de backups com redundância adequada para estes servidores, para caso seja necessário realizar restaurações de dados. A aplicação Auditchain entra em ação a partir dos dados indexados no servidor Elasticsearch.

Periodicamente a Auditchain realiza a coleta de *logs* de intervalos de tempo pré-configurados, definidos através de um arquivo de configuração. O intervalo de tempo é o que define a granularidade a qual será realizada a PoE dos *logs*. Após a coleta dos *logs*, Auditchain gera uma assinatura desses dados concatenados com a assinatura dos dados do período anterior. Logo após é gerado o *hash* SHA-256 da assinatura, que é enviado para servidores de calendário OpenTimestamps. Os servidores de calendário OpenTimestamps, por sua vez, ficam responsáveis pela agregação dos *hashes* com diversos outros *hashes* (enviados por outros clientes) em uma árvore de Merkle. A cada determinado período de tempo, um servidor de calendário é eleito para realizar a ancoragem da raiz de Merkle na Bitcoin. A partir do momento que a raiz de Merkle é ancorada na Bitcoin, os servidores de calendário atualizam os dados dos atestados para que possuam todos os ramos necessários para formar a raiz de Merkle a partir de um ramo específico,

e inserem também no atestado o ID da transação na Bitcoin. Dessa forma, é possível baixar todos os dados necessários para conseguir validar a PoE diretamente na *blockchain* (MENDONCA; MATIAS, 2021).

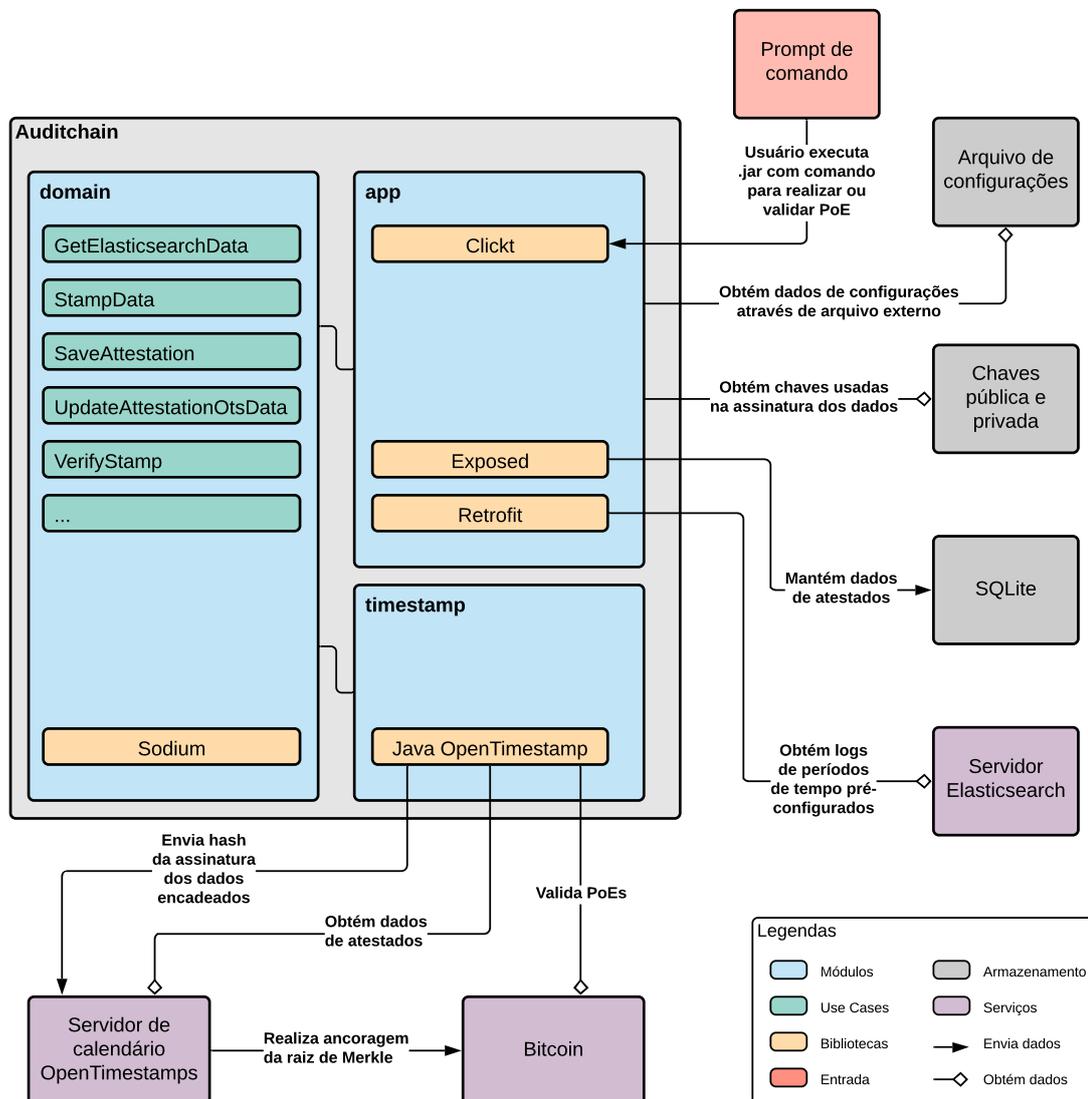
A Auditchain foi desenvolvida de forma modular utilizando a arquitetura *Clean Architecture* (MARTIN, 2012). Foram desenvolvidos três módulos: **app** responsável pelas fontes de dados e comunicação com usuário; **timestamp** responsável pela realização e validação de PoEs; e **domain** responsável pela lógica de domínio da aplicação, de maneira a orquestrar demais módulos. Essa arquitetura com suas principais bibliotecas pode ser visualizada com detalhes na Figura 24. Com a biblioteca Clickt foram configurados dois comandos de linha: `stamp-elasticsearch` e `verify-elasticsearch`. A biblioteca Exposed realiza o acesso a base de dados SQLite para manter os dados dos atestados. A Retrofit é utilizada para obter os *logs* registrados no servidor Elasticsearch. A Java OpenTimestamps é responsável por toda a comunicação com servidores de calendário OpenTimestamps, para realização de PoEs e obtenção dos dados de atestados. É ela também a responsável por validar a PoE na Bitcoin. Por fim, a biblioteca Sodium é usada pra realização e validação da assinatura dos dados.

As chaves privada e pública para assinatura dos dados devem utilizar o esquema ED25519 e devem ser disponibilizadas em arquivos. As configurações da Auditchain devem estar em um arquivo de propriedades com o nome `config.properties` localizado no mesmo diretório do arquivo `.jar` da aplicação. Neste arquivo são configuradas as informações para acesso ao servidor do Elasticsearch, informações sobre a granularidade da realização de PoEs e o diretório onde estão localizadas as chaves pública e privada. Um exemplo destas configurações pode ser visualizado em Código 1. A obtenção dos dados no Elasticsearch é com base em um padrão de índice (*index pattern*). Os dados são coletados de todos os índices com o padrão configurado, nos intervalos de tempo de acordo com a granularidade especificada. O intervalo de tempo para realização de PoEs, ou seja, a granularidade, é definido no parâmetro `frequency`, que no exemplo citado é de 25 minutos.

De acordo com a granularidade configurada, a Auditchain calcula em quais momentos do dia deverão ser realizadas as PoEs. Para uma granularidade de 30 minutos, por exemplo, as PoEs serão realizadas nos intervalos de 00:00 a 00:30, 00:30 a 01:00, e assim por diante. Quando é configurada uma granularidade cujo tempo não coincida com as 24 horas do dia, uma última PoE é realizada com o tempo restante. Com uma granularidade de 25 minutos, por exemplo, é realizada uma última PoE no dia de 15 minutos, das 23:45 às 00:00. O parâmetro de tempo utilizado para seleção dos dados é o carimbo de data e hora do documento indexado no Elasticsearch, de forma a assegurar que sempre serão retornados os mesmos dados para determinado intervalo de tempo. Isso garante que o *hash* gerado no momento da validação da PoE será igual ao *hash* gerado na criação da mesma.

O parâmetro `delay` é utilizado para configurar alguns segundos de atraso para obtenção dos dados, com a intenção de garantir que todos os *logs* já tenham sido indexados. Com os

Figura 24 – Arquitetura da Aplicação Auditchain



Fonte: Próprio autor

parâmetros definidos em [Código 1](#), por exemplo, caso o programa seja executado exatamente às 00:25, a obtenção dos dados não será realizada, pois com um atraso de 120 segundos, a PoE deste intervalo só poderá ser feita a partir das 00:27. A lógica de tratamento dos intervalos de tempo, assim como demais lógicas de domínio da aplicação ficam no módulo `domain`.

No módulo `domain` as funcionalidades principais da aplicação são organizadas em *use cases*. Esses *use cases* são os responsáveis por orquestrar a lógica de demais módulos, de modo a abstrair o acesso às fontes de dados. Ou seja, o módulo `domain` não sabe como os dados são obtidos, salvos ou enviados à *blockchain*. Essa abstração permite que o módulo `domain` seja desacoplado dos demais, de maneira que caso alguma fonte de dados seja alterada ele seja pouco (ou até mesmo nada) impactado. Com isso, a Auditchain se torna facilmente escalonável para

**Código 1 – Arquivo de configurações**

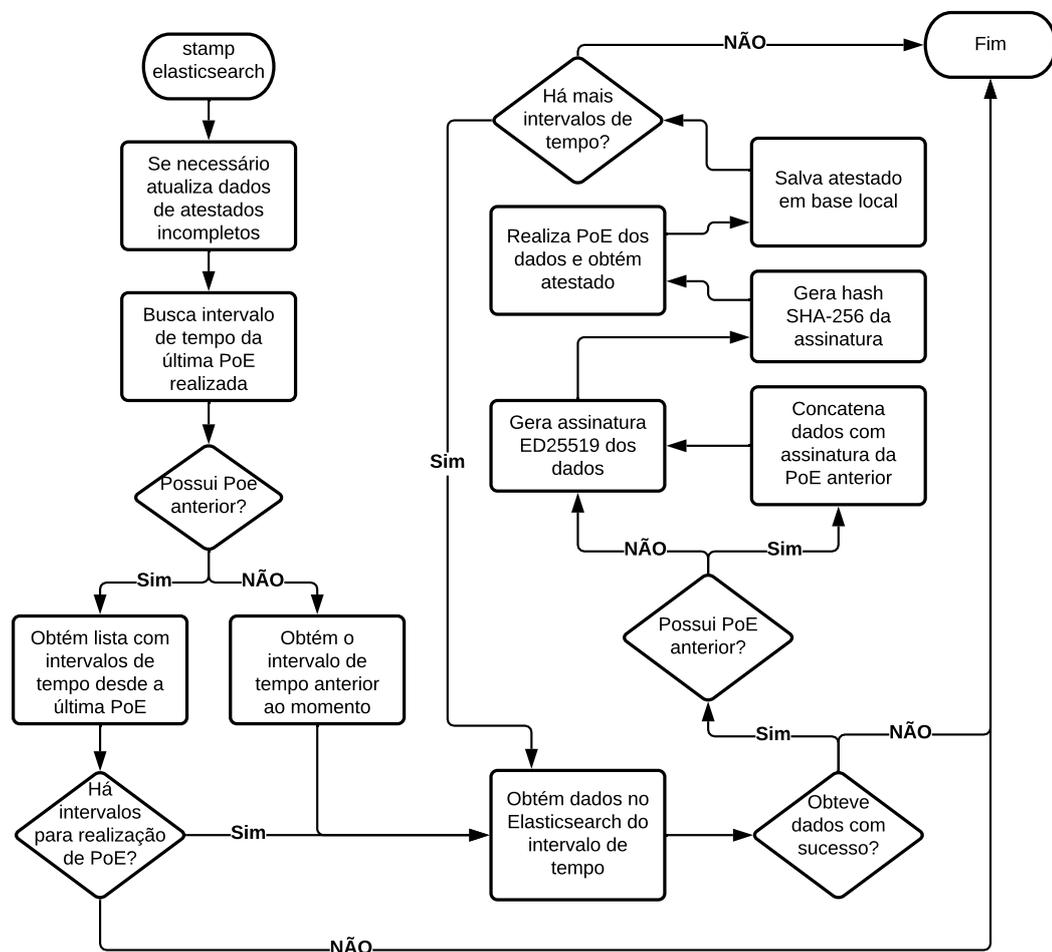
```
1 # Elasticsearch infos
2 elasticHost = https://elasticsearch.host/
3 elasticUser = elasticsearch-user-name
4 elasticPwds = elasticsearch-user-password
5 indexPattern = indexPattern*
6
7 # Attestation infos
8 # frequency in minutes
9 frequency = 25
10 # delay in seconds
11 delay = 120
12 # keys to sign data
13 signingKey = path/to/signing_key
14 verifyKey = path/to/verify_key
```

realização de PoEs com obtenção de dados em outras fontes além da Elasticsearch. Basta que essa obtenção de dados seja implementada no módulo `app` utilizando a lógica para realização de PoEs contidas no módulo `domain`. Além disso, não seria complicado utilizar alguma outra API de realização de PoE, bastaria alterar o módulo `timestamp` seguindo a abstração definida no módulo `domain`. O ponto de entrada da aplicação para execução das ações é através dos comandos pré-definidos.

O comando `stamp-elasticsearch` é utilizado para executar a ação de criação de PoE de dados provenientes dos servidores Elasticsearch. A cada vez que este comando é executado, a `Auditchain` realiza as etapas definidas no fluxograma da [Figura 25](#). Quando necessário, é realizada a atualização de atestados incompletos, ou seja, atestados que ainda não possuem todos os dados para validação diretamente na *blockchain*. Esses dados para validação só são disponibilizados pelos servidores de calendário `OpenTimestamps` após a raiz de Merkle ser ancorada na *blockchain*. Por esse motivo, a `Auditchain` busca por atualizações toda vez que a aplicação é executada. Após este processo, busca-se o intervalo de tempo da última PoE realizada. Caso não haja PoE anterior, a `Auditchain` reconhece a PoE atual como sendo a primeira PoE registrada e obtém-se o intervalo de tempo anterior ao momento atual, com base na granularidade especificada. Caso haja PoE anterior, a `Auditchain` verifica qual foi o intervalo de tempo da última realização de PoE e obtém uma lista com todos os intervalos de tempo desde então, também com base na granularidade especificada. Caso a última PoE for do intervalo de tempo mais recente, significa que não há PoEs a serem realizadas no momento e é necessário aguardar até que se atinja o próximo intervalo de tempo de acordo com a granularidade. A lista com os intervalos de tempo é utilizada como base para as próximas etapas, que são realizadas de forma sequencial para cada intervalo de tempo.

A `Auditchain` obtém os dados de um intervalo de tempo e caso essa obtenção falhe (seja

Figura 25 – Fluxograma do comando stamp-elasticsearch



Fonte: Próprio autor

por problemas de acesso ao servidor, conexão com internet, etc) todo o processo de realização de PoE é interrompido. Havendo sucesso na obtenção dos dados, caso haja PoE anterior ao momento atual, os dados baixados são concatenados com a assinatura dos dados da PoE anterior. Caso não haja PoE anterior, a atual será considerada a primeira PoE da cadeia de PoEs. Então é gerada a assinatura ED25519 dos dados finais, e posteriormente é gerado o *hash* SHA-256 desta assinatura. O *hash* é então enviado aos servidores de calendário OpenTimestamps e os dados de atestados incompletos são baixados e armazenados localmente. Todo este processo é realizado de forma síncrona para que seja possível realizar o encadeamento das PoEs. Ao realizar o comando `stamp-elasticsearch` é possível adicionar a opção `-v` ou `--verbose` para que sejam exibidos detalhes sobre as operações realizadas. Um exemplo da execução do comando `stamp-elasticsearch` pode ser visto em [Figura 26](#).

O comando `verify-elasticsearch`, por sua vez, é utilizado para executar a verificação de PoE de dados provenientes dos servidores Elasticsearch. A cada vez que este comando

**Figura 26 – Execução do comando stamp-elasticsearch**

```
c:\Auditchain>java -jar Auditchain-0.3.1-RELEASE.jar stamp-elasticsearch -v
Updating OTS data from previous stamps...
Stamping data from: 2021-03-24 19:35 to 2021-03-24 20:00

-----
Data from:
Interval: 2021-03-24 19:35 - 2021-03-24 20:00
Source: ELASTICSEARCH
Stamped at 2021-03-24 20:02
.
.
.
Process completed

c:\Auditchain>
```

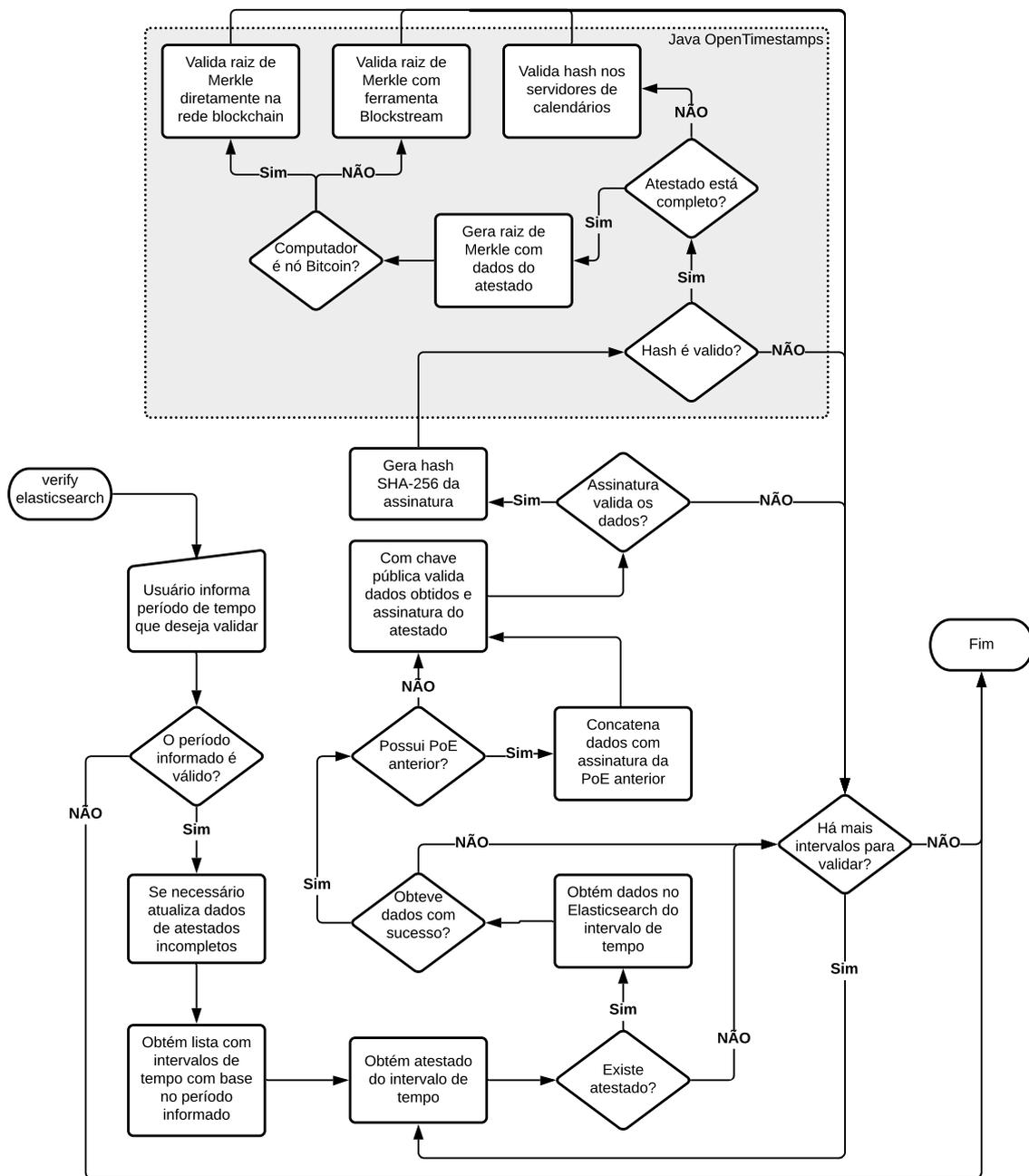
Fonte: Próprio autor

é executado, a Auditchain realiza as etapas definidas no fluxograma da [Figura 27](#). Ao executar este comando o usuário deve informar qual período de tempo deseja verificar através das opções `--start-at` e `--finish-in`. Caso o período informado seja inválido, o usuário é informado e o comando não é executado. Antes da verificação iniciar, os atestados incompletos são atualizados. Logo após, a Auditchain obtém uma lista com os intervalos de tempo a serem validados com base no período informado pelo usuário e na granularidade configurada. Em uma granularidade de 25 minutos, por exemplo, caso o usuário queira verificar o período de 00:10 a 00:30, serão validados os intervalos de 00:00 a 00:25 e 00:25 a 00:50. Com base nessa lista de intervalos, a Auditchain inicia o processo de validação para cada intervalo individualmente.

O primeiro passo é obter o atestado para o intervalo de tempo. Caso o atestado não seja localizado, o usuário é informado e o ciclo se repete para o próximo intervalo. Após a obtenção do atestado, são obtidos os dados do servidor Elasticsearch. Se houver falha na obtenção dos dados do Elasticsearch, o usuário é informado e o ciclo se repete para o próximo intervalo. Caso haja PoE anterior, sua assinatura é concatenada com os dados obtidos do Elasticsearch. Então os dados finais são validados com a assinatura armazenada no atestado, com base na chave pública do usuário. Caso os dados não sejam validados de acordo com a assinatura, o usuário é informado e o ciclo se repete para o próximo intervalo. Com os dados válidos de acordo com a assinatura, resta somente gerar seu *hash* SHA-256 e validar se este *hash* consta na *blockchain*. A Java OpenTimestamps é a responsável pela execução dos próximos passos.

A Java Opentimestamps verifica se o *hash* a ser validado condiz com o armazenado localmente no atestado. Caso não seja, o usuário é informado e o ciclo se repete para o próximo intervalo. Logo após, é validado se o atestado está completo, ou seja, se todos os dados para criação da raiz de Merkle estão disponíveis. Caso o atestado não esteja completo é sinal de que a raiz de Merkle ainda não foi ancorada na *blockchain*, então os dados são validados de acordo

Figura 27 – Fluxograma do comando verify-elasticsearch



Fonte: Próprio autor

com os servidores de calendário OpenTimestamps públicos. Caso o atestado esteja completo, a raiz de Merkle é recriada e validada na *blockchain*. Caso a máquina que está executando a verificação seja um nó Bitcoin, a validação é realizada diretamente na rede. Caso a máquina não seja um nó Bitcoin, a validação é realizada com auxílio da ferramenta Blockstream. Assim como o comando `stamp-elasticsearch`, ao realizar o comando `verify-elasticsearch` é possível adicionar a opção `-v` ou `--verbose` para que sejam exibidos detalhes sobre as

operações realizadas. Um exemplo da execução do comando `verify-elasticsearch` pode ser visualizado na [Figura 28<sup>15</sup>](#).

**Figura 28 – Execução do comando `verify-elasticsearch`**

```
c:\Auditchain>java -jar Auditchain-0.3.1-RELEASE.jar verify-elasticsearch
-v --start-at="2021-03-24 19:35" --finish-in="2021-03-24 20:00"
Updating OTS data from previous stamps...
Verifying data from: 2021-03-24 19:35 to 2021-03-24 20:00

-----
Data from:
Interval: 2021-03-24 19:35 - 2021-03-24 20:00
Source: ELASTICSEARCH
Is attested by:
BITCOIN since 2021-03-24 20:25
Latency time from this attestation: 82186 seconds

.
.
.
Process completed

c:\Auditchain>
```

Fonte: Próprio autor

### 4.2.1 Discussão acerca do modelo adversarial

De acordo com a solução desenvolvida no estudo de caso Auditchain, são discutidos abaixo os modelos de ataque levantados para este trabalho. A discussão considera as seguintes premissas: a chave privada usada para assinar os dados é mantida somente no computador que executa o Auditchain; a chave pública usada para validar os dados é amplamente divulgada; existem backups dos dados; e são registrados *logs* de todas as operações que alteram dados no sistema.

#### 1. Adversário com acesso ao servidor da aplicação passa a forjar *logs*.

Nesse caso, o adversário conseguiria modificar o comportamento da aplicação ou alterar a geração de arquivos de *log* de alguma outra forma para forjá-los. Foge ao escopo do Auditchain proteger contra esse tipo de ataque.

No entanto, vale ressaltar que o Auditchain ainda é útil para investigar como foi possível obter acesso à máquina, já que o invasor provavelmente deixou rastros antes de concretizar o acesso. Se o adversário tentar apagar esses rastros, o Auditchain permitirá identificar o instante de tempo aproximado da invasão, guiando esforços de recuperação de *backup* dos *logs* e demais evidências digitais.

<sup>15</sup> O tempo de latência é calculado com base no momento que a PoE foi realizada e o momento em que ela foi atualizada e seu atestado completo. No caso desse teste, a latência ficou com valor elevado por ser um ambiente de teste onde a Auditchain não é executado automaticamente.

## 2. Adversário com acesso à base de dados forja um registro.

Basta consultar o *log* da operação para revertê-la e responsabilizar o adversário.

## 3. Adversário com acesso aos arquivos de *logs* forja *logs* já registrados.

As ferramentas de coleta de *logs* fazem a indexação no Elasticsearch assim que o *log* é registrado nos arquivos de *log*. Portanto, apagar o *log* dos arquivos não surte grande efeito. Basta consultar sempre os *logs* indexados no servidor Elasticsearch.

No entanto, é importante notar que as permissões de acesso ao índice pelas credenciais utilizadas pela ferramenta de coleta precisam ser corretamente configuradas, para evitar que um comprometimento no servidor de aplicação transforme-se imediatamente em um comprometimento do índice no servidor Elasticsearch.

As credenciais utilizadas para coleta devem possuir apenas a permissão `create_doc` no índice que armazena os *logs*, para permitir apenas a adição de novos registros, impedindo a exclusão ou modificação de registros previamente incluídos. A ferramenta de coleta deve suportar o uso do Elasticsearch configurado dessa maneira. As ferramentas Filebeat e Logstash o fazem por padrão. A Fluentd precisa ter sua configuração `write_operation` alterada do padrão `index` para `create`. A Fluent Bit precisa ser compilada com o *patch* de uma *pull request* recentemente integrada ao projeto<sup>16</sup>.

## 4. Adversário com acesso ao servidor Elasticsearch forja *logs* após certo tempo destes terem sido indexados.

Considerando que a PoE dos *logs* no período de tempo em que os dados foram forjados já tinha sido realizada, basta executar a verificação com Auditchain para atestar que os *logs* de determinado intervalo de tempo foram forjados. Mesmo que a PoE ainda não tenha sido ancorada na *blockchain*, é possível realizar a atestação dos dados com base nos servidores de calendário públicos OpenTimestamps.

## 5. Adversário com acesso ao servidor Elasticsearch forja *logs* logo após estes terem sido indexados.

Caso o adversário consiga forjar os *logs* no servidor Elasticsearch logo após realizar uma operação, há grandes chances desse *log* não ter sido atestado pela Auditchain. Nesse caso, o adversário conseguiria forjar os *logs*.

## 6. Adversário com acesso à base de atestados apaga o atestado correspondente a um determinado intervalo de tempo.

Caso as PoEs fossem realizadas diretamente na *blockchain*, seria possível listar todas as transações assinadas com certa chave pública, constatando a ausência de uma PoE que deveria estar na base de atestados.

<sup>16</sup> <<https://github.com/fluent/fluent-bit/pull/2026>>

Como o Auditchain utiliza APIs que agregam diversas PoEs em uma árvore de Merkle antes de ancorá-las, não é possível listar todas as transações efetuadas por determinada instância do Auditchain. Por isso, o Auditchain implementa um segundo nível de encadeamento, entre uma PoE e a próxima, além do encadeamento já existente na própria *blockchain*. Desta forma, a exclusão de um atestado pode ser detectada pois o encadeamento com as PoEs seguintes é quebrado.

**7. Adversário com acesso à base de atestados substitui atestados verdadeiros por atestados forjados através da realização de PoE paralela de logs verossímeis por determinado período de tempo.**

Haveria falha na validação da assinatura dos atestados forjados quando estes fossem verificados com a chave pública do par de chaves configurado no Auditchain.

**8. Adversário com acesso aos arquivos de logs e à base de atestados forja um log e deleta o atestado antes que este seja ancorado na *blockchain*.**

Caso o atestado deletado não seja o último da cadeia de atestados, é possível detectar sua exclusão. No entanto, caso o adversário delete o último atestado da cadeia de PoEs antes que este seja ancorado, o adversário conseguiria realizar o forjamento.

**9. Adversário com acesso aos servidores de calendário intercepta hashes com a intenção de identificar padrões nos logs e forjar atestados.**

A informação do período de tempo é adicionada à assinatura da PoE com intuito de que cada PoE seja única, principalmente para diferenciar períodos em que não há dados registrados. Com isso, mesmo que o sistema realize a PoE de intervalos de tempo com dados vazios seguidas vezes, cada *hash* terá um valor diferente.

# Capítulo 5

## RESULTADOS

---

---

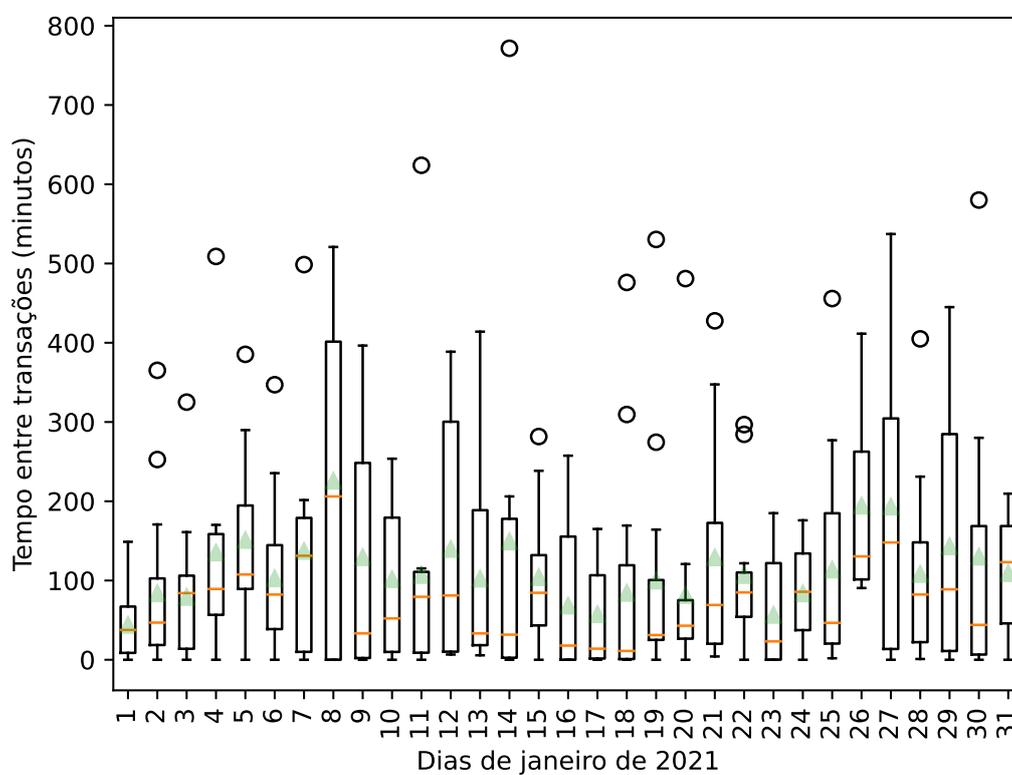
Paralelamente aos testes com a ferramenta Auditchain, monitorou-se a página de informações de cada servidor calendário da OpenTimestamps a cada 10 minutos, a fim de identificar quais as últimas transações realizadas na Bitcoin por cada servidor. Esse levantamento serviu a dois propósitos: (1) determinar com maior clareza estatísticas relacionadas ao tempo entre transações ao longo de um período mais extenso, já que a página de informações apresenta apenas a média desse valor ao longo da última semana; (2) validar se a biblioteca cliente da OpenTimestamps e, conseqüentemente, o Auditchain, faziam o melhor uso possível dessas transações para determinar os tempos de ancoragem em *blockchain*. Este capítulo apresenta esses dados quantitativos e, em seguida, finaliza resumindo as contribuições qualitativas alcançadas com a proposta da ferramenta.

### 5.1 Tempo entre transações da OpenTimestamps

As Figuras 29, 30 e 31 mostram o tempo entre transações emitidas por qualquer um dos quatro servidores calendário públicos da OpenTimestamps, respectivamente nos meses de janeiro, fevereiro e março (somente até o dia 28) de 2021. As barras laranjas e triângulos verdes representam a mediana e a média, respectivamente. As caixas cobrem o intervalo entre o primeiro e o terceiro quartis, os *whiskers* cobrem o mínimo e o máximo e os círculos representam *outliers*.

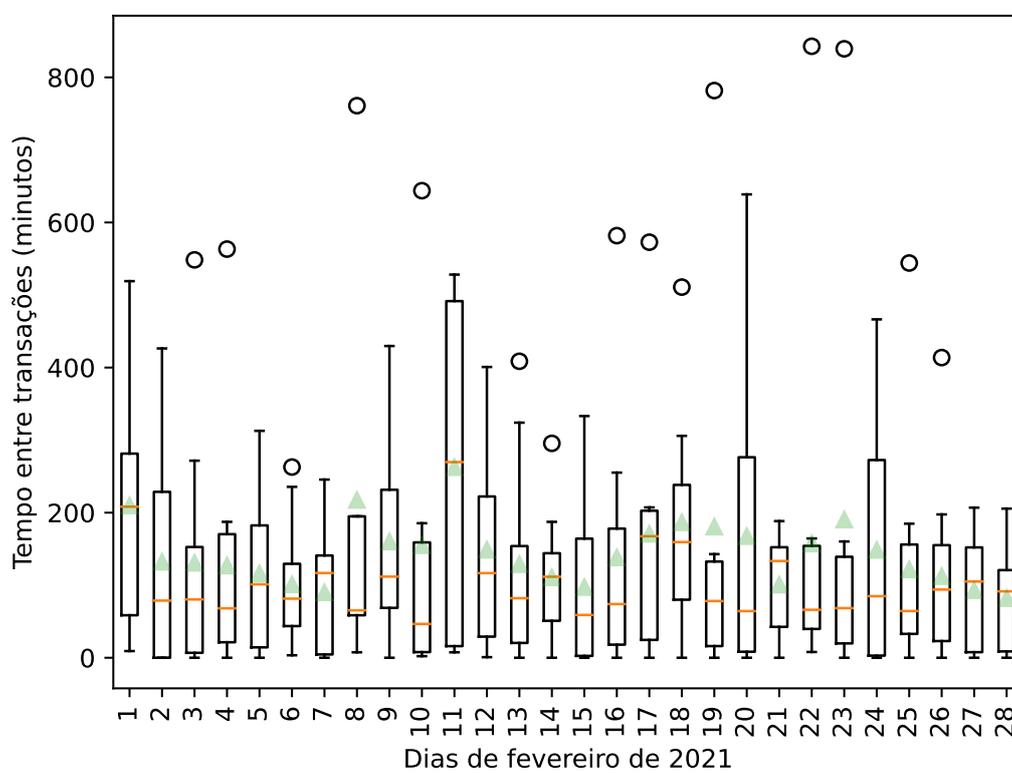
As estatísticas de cada mês são resumidas na Tabela 2. Como esperado, os valores obtidos para a média de tempo entre transação são próximos à soma entre os valores de resolução e latência estimados para a API em Seção 4.1.2. É interessante notar, no entanto, a grande dispersão desses valores de tempo: em alguns momentos, ocorre um grande tempo de espera até a próxima transação. No mês de março, muito embora a média e a mediana dos valores tenham sido um pouco desfavoráveis quando comparadas às do mês de janeiro, houve menos incidência de casos de grande tempo de espera, como evidenciado pelo 85º percentil da Tabela 2 e pelo valor máximo dos *outliers* apresentados na Figura 31.

**Figura 29 – Tempo entre transações Bitcoin do OpenTimestamps em janeiro de 2021**



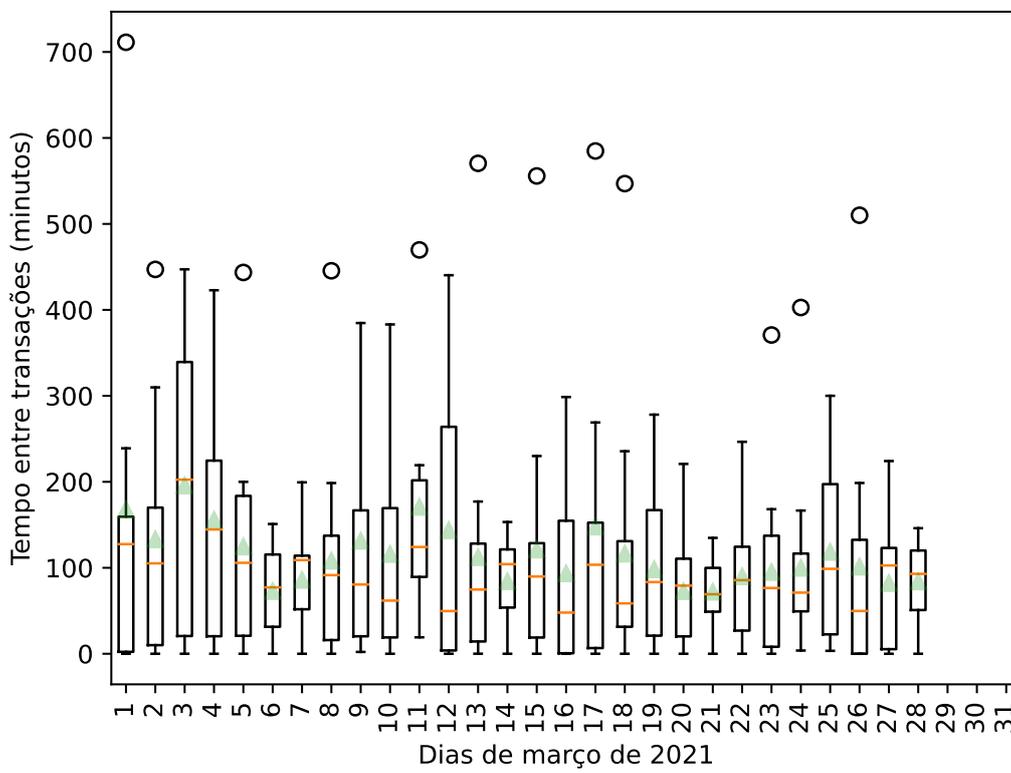
Fonte: Próprio autor

**Figura 30 – Tempo entre transações Bitcoin do OpenTimestamps em fevereiro de 2021**



Fonte: Próprio autor

**Figura 31 – Tempo entre transações Bitcoin do OpenTimestamps em março de 2021**



Fonte: Próprio autor

**Tabela 2 – Estatísticas do tempo entre transações da OpenTimestamps**

<b>Janeiro de 2021</b>	
<b>Mediana</b>	68 min
<b>Média</b>	104 min
<b>Desvio padrão</b>	121 min
<b>15º percentil</b>	1 min
<b>85º percentil</b>	201 min
<b>Fevereiro de 2021</b>	
<b>Mediana</b>	100 min
<b>Média</b>	140 min
<b>Desvio padrão</b>	157 min
<b>15º percentil</b>	3 min
<b>85º percentil</b>	263 min
<b>Março de 2021</b>	
<b>Mediana</b>	93 min
<b>Média</b>	109 min
<b>Desvio padrão</b>	114 min
<b>15º percentil</b>	4 min
<b>85º percentil</b>	184 min

## 5.2 Aproveitamento de transações para ancoragem de PoEs

De 17/03 a 26/03/2021, foram realizadas 512 PoEs, que acabaram sendo ancoradas em 112 transações. Observando-se as transações realizadas por todos os servidores calendário no mesmo período, foram encontradas as seguintes transações que deixaram de ser utilizadas para ancoragem de PoEs:

- a) 61abe99385e69ddb2eff128f2b71cab2bf35cbfd8f7914e25491b64e9c9af3, emitida em 17/03 às 21:27 pelo servidor catallaxy;
- b) aec75af2a2117cfa60e89ca0cf0bc74e1b76ed5edd60ae9815815324fe791854, emitida em 18/03 às 03:19 pelo servidor finney;
- c) 4ac06fec4bc30d963aeed87000867378321e8798b6c63669be900891fb74ea10, emitida em 18/03 às 17:49 pelo servidor bob;
- d) f99a0484b3dee771fec1d276a50b6288f0c0b156ce31c534f5bb3edc55c3ea3f, emitida em 22/03 às 22:54 pelo servidor catallaxy;
- e) 09b59353a7235d530676234721747493006a38e7b642108130838265f2d42de8, emitida em 23/03 às 06:32 pelo servidor finney.

Ou seja, foram aproveitadas 96% de todas as transações emitidas no período, o que corresponde a um aproveitamento satisfatório. No entanto, é interessante investigar o motivo pelo qual as transações acima não foram aproveitadas.

A princípio, levantou-se a hipótese de que os servidores correspondentes poderiam ter excedido o limite de *commitments* pendentes próximo aos instantes em questão. No entanto, por meio do monitoramento da página de informações dos servidores calendário, foi possível constatar que a quantidade de *commitments* pendentes nos instantes em questão era inferior ao máximo observado durante o período dos testes. Observou-se, também, que o máximo de *commitments* pendentes dentre todos os servidores durante o período dos testes foi de 50298, no servidor finney. Mesmo esse valor é inferior ao limite padrão de 100000 *commitments* pendentes.

Este assunto requer, portanto, uma depuração mais cuidadosa que será deixada para trabalhos futuros. Vale ressaltar que esses 4% de transações não aproveitadas não impedem o correto funcionamento da ferramenta, uma vez que as PoEs acabam por ser ancoradas em transações posteriores. O efeito de um aproveitamento inferior a 100% é tão somente reduzir a precisão temporal dos instantes de ancoragem.

## 5.3 Sumário qualitativo

Embora a realização de PoE com uso de APIs seja um método consolidado, não há trabalhos anteriores que apliquem esse método à atestação de *logs*. Comparado à proposta de [Kalis e Belloum \(2018\)](#), este trabalho trouxe avanços com (1) utilização de API para realização

da PoEs, reduzindo custos a praticamente gratuito; e (2) integração do método de PoE com a plataforma Elasticsearch, possibilitando atestar a integridade de dados de diversas fontes de maneira simplificada. A proposta de segurança incremental oferecida pela OpenTimestamps no contexto deste trabalho é parecida com as propostas realizadas por [Meneghetti et al. \(2019\)](#) e [Aniello et al. \(2017\)](#). Em comparação com ambas as propostas, a solução deste trabalho traz maior praticidade e transparência por utilizar os servidores de calendários públicos da OpenTimestamps como primeira camada de segurança. Por outro lado, a proposta de [Aniello et al. \(2017\)](#) oferece maior resistência a ataques de falsificação dos dados por armazená-los diretamente em uma *blockchain* privada. Assim, para forjar os dados seria necessário atacar essa *blockchain*.

# Capítulo 6

## CONCLUSÕES

---

---

Neste trabalho foi desenvolvido um estudo de caso acerca de uma arquitetura para integrar a realização de PoE de *logs* à plataforma Elasticsearch. Essa integração visa tornar a realização de PoE mais simples e abrangente, possibilitando que técnicos de informática em diversos contextos que indexam seus *logs* na Elasticsearch possam atestar seus dados com menos necessidade de confiança em terceiros. A redução da necessidade de confiança se dá pelo fato de utilizarmos, nesta solução, *blockchains* públicas, que são totalmente auditáveis. Além disso, há maior transparência por utilizar-se aplicações de código aberto.

Para realizar PoEs, optou-se pela utilização de APIs que agregam *hashes* em árvores de Merkle e realizam a ancoragem somente da raiz de Merkle. Foram realizadas análises e comparações entre três APIs disponíveis atualmente, sendo elas a Chainpoint, OpenTimestamps e OriginStamp. A análise contou principalmente com a documentação fornecida pelas APIs, mas também foram realizados contatos com responsáveis e testes. Os principais critérios para análise foram resolução, latência, solidez, ancoragem, codificação, documentação e custos. Após comparação, a API definida como mais adequada para desenvolvimento do estudo de caso deste trabalho foi a OpenTimestamps.

Foi desenvolvida então a Auditchain, uma ferramenta CLI capaz de obter *logs* indexados pela ferramenta Elasticsearch e realizar a PoE destes com incrementos de segurança. Dentre estes incrementos, destacamos (1) a utilização de assinatura digital na realização de PoE ao invés dos dados puros, o que permite confirmar a autenticidade da PoE; (2) encadeamento de PoEs, que possibilita identificar rapidamente quando um atestado foi excluído mesmo utilizando esquemas de árvores de Merkle; e (3) adição de metadados referente ao período de tempo da PoE na assinatura dos dados, que permite um *hash* diferente para cada PoE realizada evitando assim riscos envolvendo padrões. Com isso, foram discutidos modelos de ataque e como a Auditchain poderia resistir ou não a cada um deles.

Foram então realizados testes que permitiram aferir o correto funcionamento da ferramenta coletando *logs* de um servidor Elasticsearch instalado na SIn. Comparando os instantes de ancoragem em *blockchain* das PoEs emitidas pelo teste com os instantes das transações geradas

pelos quatro servidores calendários públicos da OpenTimestamps, chegou-se à conclusão que 96% das transações continham PoEs solicitadas pelo Auditchain.

Destacamos como o principal benefício para o estado da arte fornecido com este trabalho uma abordagem inovadora para integrar a realização de PoEs a *logs* provenientes de diversas fontes de forma gratuita e imediatamente aplicável, sem necessitar que o administrador de sistemas configure antes seu próprio nó completo da rede Bitcoin. Além disso, há também o estudo e análise das APIs de realização de PoE, que não foi encontrado em nenhum outro trabalho. Nos demais trabalhos encontrados que citam APIs, os autores simplesmente apresentam a API que vão utilizar. Por fim o avanços nos estudos sobre atestação de integridade de dados a partir dos testes e análise dos dados obtidos. Embora a pesquisa tenha sido feita no contexto de técnicos de informática com auxílio da SIn, os resultados aqui obtidos podem ser generalizados para diferentes contextos onde haja necessidade de atestação de dados. Como sugestões para trabalhos futuros, destacamos:

- a) sugerir mudanças na OpenTimestamps de forma que o instante de recepção da PoE pelo servidor calendário passe a ser armazenado, tenha sua integridade atestada pela árvore de Merkle e seja retornado pela API, a fim de aumentar a precisão temporal dos *timestamps*;
- b) implementar e testar a arquitetura proposta com outra API de realização de PoE, com intuito de comparar os resultados aqui obtidos;
- c) implementar e testar a arquitetura proposta sem utilizar APIs de realização de PoE, realizando-as diretamente na *blockchain*, com intuito de comparar os resultados e verificar se há ganhos de segurança;
- d) analisar o tempo médio que um adversário leva para conseguir acesso a um sistema ou base de dados, para validar o tempo de granularidade mais adequado para realização de PoEs;
- e) evoluir a Auditchain para tratar mudanças no arquivo de configuração de uma instância já em operação e implementar algum mecanismo automatizado de redundância da base de atestados (por exemplo, armazená-la em um *blockchain* privado, já que ela possui histórico imutável);
- f) utilizar esquemas de hardware criptográfico para armazenamento das chaves, como um token criptográfico, um módulo de segurança de hardware (HSM) ou um módulo de plataforma confiável (TPM). Esquemas deste tipo evitariam ponto único de falha no servidor de atestados, onde as chaves são armazenadas atualmente.

Por fim, embora existam trabalhos com abordagem parecidas, este trabalho é o primeiro a explicitar que atestar a integridade de *logs* é equivalente a fazer registro de PoEs.

## REFERÊNCIAS

---

ABREU, P. W.; APARICIO, M.; COSTA, C. J. Blockchain technology in the auditing environment. In: *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*. [S.l.: s.n.], 2018. p. 1–6. Citado 2 vezes nas páginas 13 e 39.

ALKETBI, A.; NASIR, Q.; TALIB, M. A. Blockchain for government services — Use cases, security benefits and challenges. In: *2018 15th Learning and Technology Conference (LT)*. [S.l.: s.n.], 2018. p. 112–119. Citado 5 vezes nas páginas 13, 22, 24, 25 e 30.

ANIELLO, L.; BALDONI, R.; GAETANI, E.; LOMBARDI, F.; MARGHERI, A.; SASSONE, V. A Prototype Evaluation of a Tamper-Resistant High Performance Blockchain-Based Transaction Log for a Distributed Database. In: *2017 13th European Dependable Computing Conference (EDCC)*. [S.l.: s.n.], 2017. p. 151–154. Citado 4 vezes nas páginas 13, 14, 15 e 80.

CAI, W.; WANG, Z.; ERNST, J. B.; HONG, Z.; FENG, C.; LEUNG, V. C. M. Decentralized Applications: The Blockchain-Empowered Software System. *IEEE Access*, v. 6, p. 53019–53033, 2018. ISSN 2169-3536. Citado 2 vezes nas páginas 28 e 36.

CHENG, J.; LEE, N.; CHI, C.; CHEN, Y. Blockchain and smart contract for digital certificate. In: *2018 IEEE International Conference on Applied System Invention (ICASI)*. [S.l.: s.n.], 2018. p. 1046–1051. Citado na página 15.

CHOWDHURY, M. J. M.; COLMAN, A.; KABIR, M. A.; HAN, J.; SARDA, P. Blockchain Versus Database: A Critical Analysis. In: *Proceedings of the 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications and 12th IEEE International Conference on Big Data Science and Engineering: Trustcom/BigDataSE 2018*. [S.l.]: IEEE, Institute of Electrical and Electronics Engineers, 2018. p. 1348–1353. Citado na página 37.

CONCORDIUM. *Professor Jesper Buus Nielsen introduces the Concordium Blockchain Research Center*. 2019. Disponível em: <<https://www.youtube.com/watch?v=owR7QsMngUw>>. Citado 3 vezes nas páginas 20, 25 e 26.

DAI, F.; SHI, Y.; MENG, N.; WEI, L.; YE, Z. From Bitcoin to cybersecurity: A comparative study of blockchain application and security issues. In: *2017 4th International Conference on Systems and Informatics (ICSAI)*. [S.l.: s.n.], 2017. p. 975–979. Citado 4 vezes nas páginas 24, 26, 27 e 38.

DEBNATH, S.; CHATTOPADHYAY, A.; DUTTA, S. Brief review on journey of secured hash algorithms. In: *2017 4th International Conference on Opto-Electronics and Applied Optics (Optronix)*. [S.l.: s.n.], 2017. p. 1–5. Citado na página 22.

DWORKIN, M. J. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. [S.l.], 2015. NIST FIPS 202 p. Disponível em: <<https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>>. Citado na página 23.

- ELASTICSEARCH B.V. *Elastic Stack and Product Documentation*. 2021. Disponível em: <<https://www.elastic.co/guide/index.html>>. Citado 2 vezes nas páginas 16 e 17.
- GAETANI, E.; ANIELLO, L.; BALDONI, R.; LOMBARDI, F.; MARGHERI, A.; SASSONE, V. Blockchain-based database to ensure data integrity in cloud computing environments. In: *Italian Conference on Cybersecurity (20/01/17)*. [S.l.: s.n.], 2017. Citado na página 12.
- GAO, W.; HATCHER, W. G.; YU, W. A Survey of Blockchain: Techniques, Applications, and Challenges. In: *2018 27th International Conference on Computer Communication and Networks (ICCCN)*. [S.l.: s.n.], 2018. p. 1–11. ISSN 1095-2055. Citado 7 vezes nas páginas 27, 28, 30, 31, 34, 35 e 37.
- GAO, Y.; NOBUHARA, H. A Decentralized Trusted Timestamping Based on Blockchains. *IEEJ Journal of Industry Applications*, v. 6, n. 4, p. 252–257, jul. 2017. ISSN 2187-1094, 2187-1108. Citado 2 vezes nas páginas 15 e 22.
- GATTESCHI, V.; LAMBERTI, F.; DEMARTINI, C.; PRANTEDA, C.; SANTAMARÍA, V. To Blockchain or Not to Blockchain: That Is the Question. *IT Professional*, v. 20, n. 2, p. 62–74, mar. 2018. ISSN 1520-9202. Citado na página 37.
- HANCOCK, D. R.; ALGOZZINE, B. *Doing Case Study Research: A Practical Guide for Beginning Researchers*. 3ª edição. ed. New York: Teachers College Press, 2016. ISBN 978-0-8077-5813-7. Citado 3 vezes nas páginas 18, 46 e 47.
- HARRISON, H.; BIRKS, M.; FRANKLIN, R.; MILLS, J. Case Study Research: Foundations and Methodological Orientations. *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research*, v. 18, n. 1, jan. 2017. ISSN 1438-5627. Number: 1. Disponível em: <<https://www.qualitative-research.net/index.php/fqs/article/view/2655>>. Citado 3 vezes nas páginas 18, 46 e 47.
- HEALE, R.; TWYLCROSS, A. What is a case study? *Evidence-Based Nursing*, Royal College of Nursing, v. 21, n. 1, p. 7–8, 2018. ISSN 1367-6539. Disponível em: <<https://ebn.bmj.com/content/21/1/7>>. Citado 2 vezes nas páginas 46 e 47.
- HEPP, T.; SCHOENHALS, A.; GONDEK, C.; GIPP, B. OriginStamp: A blockchain - backed system for decentralized trusted timestamping. *it - Information Technology*, Walter de Gruyter GmbH, nov 2018. Disponível em: <<https://doi.org/10.1515/itit-2018-0020>>. Citado na página 58.
- HEPP, T.; WORTNER, P.; SCHÖNHALS, A.; GIPP, B. Securing Physical Assets on the Blockchain: Linking a Novel Object Identification Concept with Distributed Ledgers. In: *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*. New York, NY, USA: ACM, 2018. (CryBlock'18), p. 60–65. ISBN 978-1-4503-5838-5. Event-place: Munich, Germany. Citado 2 vezes nas páginas 15 e 40.
- KALIS, R.; BELLOUM, A. Validating Data Integrity with Blockchain. *ResearchGate*, p. 272–277, dec 2018. Citado 7 vezes nas páginas 12, 13, 15, 17, 34, 35 e 79.
- KUMAR, K. M. M.; SUNITHA, N. R. Preserving Location Data Integrity in Location Based Servers using Blockchain Technology. In: *2017 2nd International Conference On Emerging Computation and Information Technologies (ICECIT)*. [S.l.: s.n.], 2017. p. 1–6. Citado 5 vezes nas páginas 14, 23, 26, 39 e 41.

- KUROSE, J. F.; ROSS, K. W. *Computer Networking: A Top-Down Approach (6th Edition)*. 6th. ed. [S.l.]: Pearson, 2012. ISBN 0132856204, 9780132856201. Citado 2 vezes nas páginas 22 e 23.
- LIANG, X.; SHETTY, S.; TOSH, D.; KAMHOUA, C.; KWIAT, K.; NJILLA, L. ProvChain: A Blockchain-Based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability. In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. [S.l.: s.n.], 2017. p. 468–477. Citado 5 vezes nas páginas 15, 24, 25, 39 e 42.
- LIU, L.; XU, B. Research on information security technology based on blockchain. In: . [S.l.: s.n.], 2018. p. 380–384. Citado 4 vezes nas páginas 13, 24, 27 e 31.
- MARTIN, R. C. *The Clean Architecture*. 2012. Disponível em: <<https://blog.cleancoder.com/un-cle-bob/2012/08/13/the-clean-architecture.html>>. Citado na página 65.
- MENDONCA, B.; MATIAS, P. Auditchain: a mechanism for ensuring logs integrity based on proof of existence in a public blockchain. In: *NTMS'2021 - Security Track (NTMS'2021 Security Track)*. Paris, France: [s.n.], 2021. Citado na página 65.
- MENEGHETTI, A.; QUINTAVALLE, A. O.; SALA, M.; TOMASI, A. Two-tier blockchain timestamped notarization with incremental security. *arXiv:1902.03136 [cs]*, fev. 2019. ArXiv: 1902.03136. Citado 4 vezes nas páginas 15, 39, 40 e 80.
- MENEZES, A. J.; VANSTONE, S. A.; OORSCHOT, P. C. V. *Handbook of Applied Cryptography*. 1st. ed. USA: CRC Press, Inc., 1996. ISBN 0849385237. Citado 3 vezes nas páginas 20, 21 e 22.
- MENG, W.; TISCHHAUSER, E.; WANG, Q.; WANG, Y.; HAN, J. When intrusion detection meets blockchain technology: A review. *IEEE Access*, v. 6, p. 10179–10188, 2018. Citado 2 vezes nas páginas 24 e 32.
- MERKLE, R. C. Protocols for public key cryptosystems. In: *1980 IEEE Symposium on Security and Privacy*. [S.l.: s.n.], 1980. p. 122–122. ISSN 1540-7993. Citado 2 vezes nas páginas 15 e 23.
- MOUBARAK, J.; FILIOL, E.; CHAMOUN, M. On blockchain security and relevant attacks. In: *2018 IEEE Middle East and North Africa Communications Conference (MENACOMM)*. [S.l.: s.n.], 2018. p. 1–6. Citado na página 30.
- NAKAMOTO, S. et al. Bitcoin: A peer-to-peer electronic cash system. 2008. Disponível em: <<https://bitcoin.org/bitcoin.pdf>>. Citado 6 vezes nas páginas 13, 25, 29, 31, 32 e 33.
- POURMAJIDI, W.; MIRANSKY, A. Logchain: Blockchain-Assisted Log Storage. In: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. [S.l.: s.n.], 2018. p. 978–982. ISBN 978-1-5386-7235-8. ISSN 2159-6190. Citado 3 vezes nas páginas 13, 14 e 40.
- PRIANGA, S.; SAGANA, R.; SHARON, E. Evolutionary Survey On Data Security In Cloud Computing Using Blockchain. In: . [S.l.: s.n.], 2018. p. 1–6. Citado 2 vezes nas páginas 20 e 23.
- RAVI, N.; SUNITHA, N. R. Introduction of Blockchain to Mitigate The Trusted Third Party Auditing for Cloud Security: An Overview. In: *2017 2nd International Conference On Emerging Computation and Information Technologies (ICECIT)*. [S.l.: s.n.], 2017. p. 1–6. Citado 2 vezes nas páginas 13 e 16.

- RENNER, T.; MÜLLER, J.; KAO, O. Endolith: A Blockchain-Based Framework to Enhance Data Retention in Cloud Storages. In: *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. [S.l.: s.n.], 2018. p. 627–634. Citado 5 vezes nas páginas 14, 25, 28, 35 e 41.
- SCHÖNHALS, A.; HEPP, T.; GIPP, B. Design Thinking Using the Blockchain: Enable Traceability of Intellectual Property in Problem-Solving Processes for Open Innovation. In: *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*. New York, NY, USA: ACM, 2018. (CryBlock'18), p. 105–110. ISBN 978-1-4503-5838-5. Event-place: Munich, Germany. Citado 3 vezes nas páginas 15, 39 e 40.
- TODD, P. *OpenTimestamps: Scalable, Trust-Minimized, Distributed Timestamping with Bitcoin*. 2016. Disponível em: <<https://petertodd.org/2016/opentimestamps-announcement>>. Citado 2 vezes nas páginas 54 e 55.
- VAUGHAN, W.; BUKOWSKI, J.; WILKINSON, S. *Chainpoint: A scalable protocol for anchoring data in the blockchain and generating blockchain receipts*. 2016. Disponível em: <[https://github.com/chainpoint/whitepaper/blob/master/chainpoint\\_white\\_paper.pdf](https://github.com/chainpoint/whitepaper/blob/master/chainpoint_white_paper.pdf)>. Citado 2 vezes nas páginas 51 e 52.
- VAUGHAN, W.; HENDERSON, J.; PERLEY, B. *Chainpoint Start*. [S.l.]: GitHub, 2020. <<https://github.com/chainpoint/chainpoint-star>>. Citado 2 vezes nas páginas 51 e 52.
- WANG, M.; WU, Q.; QIN, B.; WANG, Q.; LIU, J.; GUAN, Z. Lightweight and Manageable Digital Evidence Preservation System on Bitcoin. *Journal of Computer Science and Technology*, v. 33, n. 3, p. 568–586, 2018. Citado 7 vezes nas páginas 12, 26, 31, 32, 39, 40 e 41.
- WOOD, G. Ethereum: a secure decentralised generalised transaction ledger byzantium version dbc2f9b - 2019-03-28. 2019. Disponível em: <<https://ethereum.github.io/yellowpaper/paper.pdf>>. Citado na página 29.
- XU, Y.; ZHAO, S.; KONG, L.; ZHENG, Y.; ZHANG, S.; LI, Q. ECBC: A high performance educational certificate blockchain with efficient query. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 10580 LNCS, p. 288–304, 2017. Citado 2 vezes nas páginas 12 e 14.
- YANG, C.; CHEN, X.; XIANG, Y. Blockchain-based publicly verifiable data deletion scheme for cloud storage. *Journal of Network and Computer Applications*, v. 103, p. 185–193, 2018. Citado 3 vezes nas páginas 14, 23 e 31.
- ZHANG, R.; XUE, R.; LIU, L. Security and privacy on blockchain. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 52, n. 3, jul. 2019. ISSN 0360-0300. Disponível em: <<https://doi.org/10.1145/3316481>>. Citado na página 30.
- ZHANG, Y.; WU, S.; JIN, B.; DU, J. A blockchain-based process provenance for cloud forensics. In: *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. [S.l.: s.n.], 2017. p. 2470–2473. Citado na página 15.
- ZHENG, Z.; XIE, S.; DAI, H.; CHEN, X.; WANG, H. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. In: *2017 IEEE International Congress on Big Data (BigData Congress)*. [S.l.: s.n.], 2017. p. 557–564. Citado 6 vezes nas páginas 13, 25, 27, 33, 34 e 35.

---

ZIKRATOV, I.; KUZMIN, A.; AKIMENKO, V.; NICULICHEV, V.; YALANSKY, L. Ensuring data integrity using blockchain technology. In: *2017 20th Conference of Open Innovations Association (FRUCT)*. [S.l.: s.n.], 2017. p. 534–539. ISSN 2305-7254. Citado na página [13](#).