UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

ARTHUR ALEXANDRE CUNHA CUSTODIO

USO DE BLOCKCHAIN CONTRA FALSIFICAÇÃO DE DOCUMENTOS

Ouro Preto
2026

ARTHUR ALEXANDRE CUNHA CUSTODIO

USO DE BLOCKCHAIN CONTRA FALSIFICAÇÃO DE DOCUMENTOS

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Carlos Frederico Marcelo da Cunha Cavalcanti

Ouro Preto
2026

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Arthur Alexandre Cunha Custódio

USO DE BLOCKCHAIN CONTRA FALSIFICAÇÃO DE DOCUMENTOS

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 30 de Julho de 2026.

Membros da banca

Carlos Frederico M. da Cunha Cavalcanti (Orientador) - Doutor - Universidade Federal de Ouro Preto
Fernando Cortez Sica (Examinador) - Doutor - Universidade Federal de Ouro Preto
Ricardo Augusto Rabelo Oliveira (Examinador) - Doutor - Universidade Federal de Ouro Preto

Carlos Frederico M. da Cunha Cavalcanti, Orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 24/07/2026.

Documento assinado eletronicamente por **Carlos Frederico Marcelo da Cunha Cavalcanti**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 11/08/2026, às 08:24, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **1142658** e o código CRC **06C766BD**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591692 - www.ufop.br

Resumo

Este trabalho propõe uma arquitetura baseada em *blockchain* permissionado para combater a falsificação de documentos legais no contexto brasileiro, considerando suas particularidades socioeconômicas e tecnológicas. A pesquisa teve como objetivo desenvolver uma arquitetura modular capaz de tratar documentos legais heterogêneos, integrando mecanismos de privacidade, incluindo uma prova de conceito de Provas de Conhecimento Zero (ZK-SNARKs) sobre o *toolchain* ZoKrates, armazenamento descentralizado via Sistema de Arquivos Interplanetário (IPFS) e identificadores descentralizados (DIDs) do padrão W3C. Utilizou-se uma metodologia aplicada, exploratória e descritiva, compreendendo etapas de análise de requisitos, seleção tecnológica, desenvolvimento da arquitetura e validação experimental do protótipo. O Hyperledger Fabric foi escolhido devido à sua capacidade de controle granular de acesso e ao seu potencial de interoperabilidade com sistemas legados por meio de integrações via API, aspecto motivador da escolha tecnológica, não avaliado experimentalmente neste trabalho, enquanto o IPFS assegurou a integridade e o endereçamento por conteúdo dos documentos armazenados, ficando a disponibilidade em cenário de produção condicionada a estratégias de persistência distribuída não implementadas no protótipo. O protótipo desenvolvido, *DoChain*, foi validado por meio de 154 casos de teste automatizados (unitários, de integração, *end-to-end* e de segurança, estes últimos incluindo 15 cenários de ataque controlados), todos aprovados, além de experimentos de desempenho que quantificaram a latência das operações principais, o comportamento sob concorrência e o *overhead* das provas de conhecimento zero. Os resultados demonstram a viabilidade técnica da proposta, evidenciando potencial para reduzir fraudes documentais, aumentar a eficiência dos processos de verificação e garantir maior segurança e interoperabilidade entre diferentes setores. Este estudo contribui academicamente ao propor um modelo adaptado às necessidades brasileiras, oferecendo bases sólidas para futuras implementações e pesquisas na área.

Palavras-chave: Blockchain. Falsificação Documental. Hyperledger Fabric. IPFS. Zero-Knowledge Proofs.

Abstract

This study proposes a permissioned blockchain-based architecture to combat the forgery of legal documents within the Brazilian context, considering its socioeconomic and technological particularities. The research aimed to develop a modular architecture capable of handling heterogeneous legal documents, integrating privacy mechanisms, including a proof of concept of Zero-Knowledge Proofs (ZK-SNARKs) built on the ZoKrates toolchain, decentralized storage through the InterPlanetary File System (IPFS), and W3C-standard decentralized identifiers (DIDs). An applied, exploratory, and descriptive methodology was employed, encompassing requirement analysis, technological selection, architecture development, and experimental validation of the prototype. Hyperledger Fabric was selected for its granular access control capabilities and its potential for interoperability with legacy systems through API integrations, a motivating factor in the technology choice, not experimentally evaluated in this work, while IPFS ensured the integrity and content-addressing of stored documents, with availability in a production scenario depending on distributed persistence strategies not implemented in the prototype. The resulting prototype, DoChain, was validated through 154 automated test cases (unit, integration, end-to-end, and security tests, the latter including 15 controlled attack scenarios), all passed, along with performance experiments quantifying the latency of core operations, behavior under concurrency, and the overhead of zero-knowledge proofs. The results demonstrate the technical feasibility of the proposed approach, showing potential to reduce document fraud, improve verification process efficiency, and ensure greater security and interoperability across sectors. This study contributes academically by proposing a model tailored to Brazilian needs, providing a solid foundation for future implementations and research in the field.

Keywords: Blockchain. Document Forgery. Hyperledger Fabric. IPFS. Zero-Knowledge Proofs.

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Abreviaturas e Siglas

ABNT	Associação Brasileira de Normas Técnicas
API	*Application Programming Interface*
BFT	*Byzantine Fault Tolerance*
CaaS	*Chaincode as a Service*
CFM	Conselho Federal de Medicina
CID	Identificador de Conteúdo (*Content Identifier*)
CNH	Carteira Nacional de Habilitação
CPF	Cadastro de Pessoas Físicas
CRI	Cartório de Registro de Imóveis
CRM	Conselho Regional de Medicina
CRS	Sequência de Referência Comum (*Common Reference String*)
CTB	Código de Trânsito Brasileiro
DECOM	Departamento de Computação
DETRAN	Departamento Estadual de Trânsito
DID	Identificadores Descentralizados (*Decentralized Identifiers*)
DLT	*Distributed Ledger Technology*
DPoS	*Delegated Proof of Stake*
DSL	Linguagem de Domínio Específico (*Domain-Specific Language*)
EBSI	*European Blockchain Services Infrastructure*
gRPC	*Google Remote Procedure Call*
ICP-Brasil	Infraestrutura de Chaves Públicas Brasileira
IES	Instituição de Ensino Superior
IPFS	Sistema de Arquivos Interplanetário

ITI	Instituto Nacional de Tecnologia da Informação
LDB	Lei de Diretrizes e Bases da Educação Nacional
LGPD	Lei Geral de Proteção de Dados Pessoais
MEC	Ministério da Educação
MPC	Computação Multipartidária (*Multi-Party Computation*)
MSP	Provedor de Serviço de Membresia (*Membership Service Provider*)
MVCC	Controle de Concorrência Multiversão (*Multiversion Concurrency Control*)
P2P	*Peer-to-Peer*
PBFT	*Practical Byzantine Fault Tolerance*
PoS	*Proof of Stake*
PoW	*Proof of Work*
R1CS	Sistema de Restrições de Posto 1 (*Rank-1 Constraint System*)
Raft	Protocolo de consenso por eleição de líder e replicação de log
REST	*Representational State Transfer*
RNP	Rede Nacional de Ensino e Pesquisa
SDK	Kit de Desenvolvimento de *Software* (*Software Development Kit*)
SENATRAN	Secretaria Nacional de Trânsito
TLS	*Transport Layer Security*
UFOP	Universidade Federal de Ouro Preto
UUID	Identificador Único Universal (*Universally Unique Identifier*)
VC	Credencial Verificável (*Verifiable Credential*)
WSL2	*Windows Subsystem for Linux 2*
ZKP	Prova de Conhecimento Zero (*Zero-Knowledge Proof*)
ZK-SNARKs	Prova de Conhecimento Zero Sucinta Não Interativa

Lista de Símbolos

π	Prova de conhecimento zero gerada pelo esquema ZK-SNARKs

Sumário

1 Introdução

Desde tempos tão antigos quanto a própria noção de burocracia, a falsificação de documentos é um fenômeno cuja escala e natureza foram profundamente transformadas no século XXI. Em um mundo cada vez mais dependente de credenciais para acesso a direitos como educação, saúde, propriedade, trabalho e prestação de serviços essenciais, a integridade documental tornou-se um dos pilares fundamentais para a justiça social e a eficiência econômica. No entanto, a globalização, a transformação digital acelerada e o aumento da sofisticação das técnicas fraudulentas transformaram essa questão em uma crise transversal a múltiplos setores, com ramificações que alcançam desde a negociação de imóveis até a emissão de atestados médicos e a concessão de habilitações.

No Brasil, essa questão é particularmente crítica, com destaque para a falsificação de diplomas, adulteração de contratos, uso de CPFs falsos e registros imobiliários fraudulentos, que geram prejuízos econômicos diretos, como as contratações baseadas em currículos fraudulentos e transferências imobiliárias ilegítimas, e indiretos, como custos de investigação e reparação de danos.

A amplitude do problema exige uma solução igualmente abrangente. Documentos como escrituras de imóveis, atestados médicos, diplomas universitários, laudos periciais e certidões de estado civil compartilham um denominador comum: todos são alvos frequentes de falsificação, todos envolvem direitos fundamentais e todos dependem, em seu modelo atual, de intermediários centralizados como cartórios, conselhos profissionais, instituições de ensino e órgãos governamentais para atestar sua autenticidade.

Nesse cenário, a tecnologia *blockchain* surge não como um mero avanço técnico, mas como uma mudança paradigmática no modelo de gestão da confiança. Desenvolvida inicialmente para dar suporte a criptomoedas como o Bitcoin (NAKAMOTO, 2008), ela é baseada em registros imutáveis e consenso distribuído, oferecendo uma nova perspectiva para a autenticação de informações. Projetos internacionais demonstram sua viabilidade em diferentes domínios documentais:

- A UoB (*University of Bahrain*) emite diplomas por *blockchain* desde 2019, reduzindo o tempo de verificação pelos empregadores (COINTELEGRAPH, 2019);
- A União Europeia introduziu a EBSI (*European Blockchain Services Infrastructure*), utilizada para certificação de qualificações profissionais e documentos públicos em 29 países (EUROPEAN COMMISSION, 2020);
- Em Singapura, o governo utilizou *blockchain* para validar certificados de saúde durante

a pandemia de COVID-19, garantindo a integridade dos dados (GOVERNMENT OF SINGAPORE, 2021);

No entanto, o Brasil ainda apresenta uso incipiente dessa tecnologia para autenticação documental. O Diploma Digital, iniciativa do Ministério da Educação (MEC) em parceria com a Rede Nacional de Ensino e Pesquisa (RNP) para a emissão e verificação de diplomas de instituições federais de ensino superior por meio de *blockchain* (MINISTÉRIO DA EDUCAÇÃO (MEC); REDE NACIONAL DE ENSINO E PESQUISA (RNP), 2021), é uma exceção isolada, não a regra. A resistência cultural, a ausência de regulamentação clara e a fragmentação institucional geram um vazio que este estudo busca preencher, propondo o *DoChain*, uma arquitetura modular capaz de atender a diferentes categorias documentais em um único sistema integrado.

1.1 Descrição do problema

Esta pesquisa busca responder como a tecnologia *blockchain* pode ser implementada para prevenir a falsificação de documentos legais de natureza heterogênea, garantindo segurança, escalabilidade e integração com sistemas legados. O problema manifesta-se em dois planos principais:

No plano **técnico**, a maioria das soluções atuais é construída para um único tipo documental utilizando estruturas de dados rígidas e lógicas de autorização não generalizáveis. Isso impede a reutilização da infraestrutura entre setores e obriga cada instituição a manter sistemas isolados e incompatíveis. Além disso, *blockchains* públicas apresentam custos e limitações de escala incompatíveis com a esfera pública brasileira.

No plano **operacional**, diferentes categorias documentais envolvem diferentes emissores, regulações e ciclos de vida. Uma escritura de imóvel é emitida por um cartório, regulada pela Lei n.º 8.935/1994 e pode ser substituída por novas escrituras. Um atestado médico é emitido por um profissional registrado no Conselho Federal de Medicina (CFM) e contém dados sensíveis sujeitos ao Art. 11 da LGPD (BRASIL, 2018). Um diploma universitário é emitido por uma instituição de ensino superior e raramente é revogado. Essas diferenças exigem uma arquitetura que suporte diferentes perfis de emissores, múltiplos ciclos de vida documentais e distintos níveis de controle de acesso.

1.2 Justificativa

A falsificação de documentos representa uma ameaça multidimensional. No setor educacional, a fraude em diplomas compromete a credibilidade de instituições e empregadores (BRUNNER, 2017). No setor imobiliário, escrituras adulteradas resultam em disputas patrimoniais de difícil resolução judicial. No âmbito da saúde, atestados e laudos falsos podem

levar a decisões médicas incorretas, colocando vidas em risco. No âmbito jurídico, documentos adulterados podem resultar em decisões judiciais injustas ou na perda de direitos fundamentais.

A motivação para este estudo reside na intersecção entre a segurança cibernética e a busca por soluções descentralizadas que reduzam a dependência de intermediários, alinhando-se aos princípios de transparência e autonomia digital. Mais especificamente, três fatores justificam a abordagem ampla adotada:

1. **Impacto Social e Econômico Transversal:** A falsificação documental afeta direitos fundamentais em múltiplos setores simultaneamente. Uma arquitetura capaz de proteger diferentes categorias documentais em um único sistema oferece retorno econômico e social proporcionalmente maior do que soluções verticais.

2. **Inovação Técnica:** Sistemas legados são fragmentados e incompatíveis entre si. A *blockchain* permissionada propõe descentralização, interoperabilidade e automação via contratos inteligentes, reduzindo tanto custos quanto falhas humanas em processos de autenticação que hoje dependem de verificação manual.

3. **Contribuição Acadêmica:** Há escassez de modelos que abranjam categorias heterogêneas de documentos legais em um único *framework* integrado. Este trabalho preenche essa lacuna ao propor uma classificação documental baseada em sensibilidade e ao projetar uma arquitetura modular capaz de acomodar as particularidades de cada categoria.

1.3 Objetivos

1.3.1 Objetivo Geral

Desenvolver uma arquitetura de *blockchain* modular para autenticação de documentos legais de natureza heterogênea, capaz de implementar mecanismos de privacidade diferenciados por nível de sensibilidade documental e de garantir interoperabilidade entre diferentes setores e sistemas.

1.3.2 Objetivos Específicos

Para atingir o objetivo principal, definem-se os seguintes objetivos específicos:

1. Classificar documentos legais por critérios de sensibilidade, frequência de uso e perfil de emissor como escrituras de imóveis, atestados médicos, diplomas universitários, certidões e carteiras de habilitação (CNH), definindo níveis distintos de controle de acesso e ciclo de vida para cada categoria;

2. Elaborar um sistema de identificadores únicos descentralizados (DIDs) segundo o padrão W3C, com os métodos *did:web* para instituições emissoras e *did:key* para titulares, vinculando documentos a identidades verificáveis sem dependência de autoridade central;

3. Propor um modelo de metadados genérico e extensível para interoperabilidade entre as diferentes categorias documentais suportadas, permitindo que novos tipos de documento sejam incorporados à arquitetura sem alterações estruturais no *chaincode*;

4. Implementar e validar um protótipo funcional que demonstre os fluxos de emissão, verificação e revogação para ao menos duas categorias documentais distintas, com coleta de métricas de desempenho e corretude.

1.4 Organização do Trabalho

Os demais capítulos estão organizados da seguinte forma: após esta introdução, o Capítulo 2 detalha a fundamentação teórica, abordando conceitos de *blockchain*, mecanismos de consenso e trabalhos relacionados. O Capítulo 3 descreve a metodologia de pesquisa, as decisões de projeto e a implementação do protótipo *DoChain*, incluindo os experimentos práticos conduzidos. No Capítulo 4, são apresentados os resultados do protótipo e a análise de sua eficácia. Por fim, o Capítulo 5 conclui o trabalho, discutindo contribuições, limitações e direções futuras.

2 Fundamentação Teórica

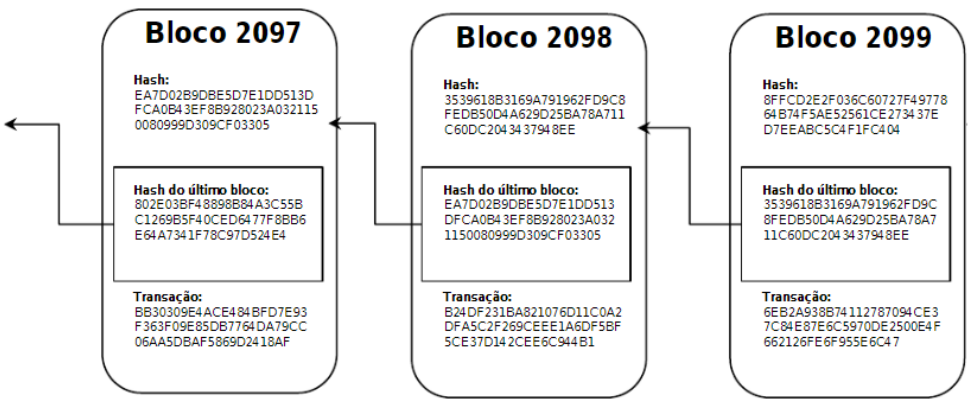
A tecnologia de *blockchain*, originada como base para criptomoedas como o Bitcoin, evoluiu para se tornar uma ferramenta fundamental em diversos domínios, incluindo sistemas de autenticação e segurança de dados. Baseada em estruturas de dados encadeadas e protegida por algoritmos criptográficos, a *blockchain* oferece propriedades de imutabilidade, transparência e descentralização que são essenciais para a emissão e gestão de documentos legais. Este capítulo aborda os conceitos fundamentais do *blockchain*, os mecanismos de consenso, as técnicas de criptografia subjacentes e os fundamentos da classificação documental, destacando sua aplicação na prevenção de fraudes e na autenticação de informações sensíveis.

2.1 Blockchain: Conceitos Fundamentais e Propriedades Técnicas

A *blockchain* apresenta um conjunto de características essenciais que garantem sua segurança, descentralização e confiabilidade. Essas propriedades são fundamentais para compreender como essa tecnologia pode ser aplicada no combate à falsificação de documentos em diferentes domínios e como pode ser adaptada para suportar categorias documentais heterogêneas.

A rede *blockchain* funciona como um livro-razão digital distribuído (*Distributed Ledger Technology* – DLT), no qual os dados são organizados em blocos interligados criptograficamente, formando uma cadeia sequencial e imutável. Conforme ilustrado na Figura 2.1, cada bloco é composto por três elementos essenciais: o *hash* único do bloco atual, o *hash* do bloco anterior e um conjunto de transações registradas.

Figura 2.1 – Exemplo de blocos encadeados.



Fonte: (ARATA; FERREIRA, 2021).

2.1.1 Arquitetura Distribuída e Imutabilidade

Ao contrário de bancos de dados tradicionais, que dependem de um servidor centralizado, a *blockchain* opera em uma arquitetura *peer-to-peer* (P2P), na qual cada nó mantém uma cópia integral do *ledger*. Essa redundância não apenas garante alta disponibilidade, já que a falha de um único nó não compromete a rede, como também elimina pontos únicos de ataque ou corrupção.

A imutabilidade é garantida pela função *hash* SHA-256 (Equação 2.1):

$$H(x) = \text{SHA256}(x) \tag{2.1}$$

onde x representa o conjunto de transações em um bloco. Qualquer alteração em x alteraria drasticamente $H(x)$, invalidando a cadeia subsequente (NAKAMOTO, 2008). Essa propriedade é especialmente relevante para documentos como escrituras de imóveis, nos quais o histórico completo de transações, aquisições, hipotecas e desmembramentos precisa ser preservado de forma inviolável.

2.1.2 Transparência e Auditabilidade

Todas as transações registradas na *blockchain* podem ser verificadas pelos participantes da rede, garantindo total transparência. Em *blockchains* públicas, qualquer pessoa pode visualizar o histórico completo de operações; enquanto em redes permissionadas, o acesso pode ser restrito a entidades autorizadas. No contexto de documentos legais heterogêneos, a auditabilidade diferenciada é crítica: atestados médicos exigem acesso restrito por questões de privacidade, enquanto escrituras de imóveis demandam transparência controlada entre as partes envolvidas.

2.1.3 Consenso na Rede

O consenso é o mecanismo central da *blockchain*, garantindo que todos os nós concordem com o estado válido do *ledger*. Em contextos empresariais ou governamentais, como no combate à falsificação de documentos, redes permissionadas adotam mecanismos mais eficientes como o *Practical Byzantine Fault Tolerance* (PBFT) ou o Raft, que priorizam velocidade e escalabilidade sem comprometer a segurança (ANDROULAKI et al., 2018).

2.2 Mecanismos de Consenso na Blockchain

Para garantir a integridade e a confiabilidade das transações, a *blockchain* utiliza mecanismos de consenso que permitem que múltiplos participantes da rede concordem sobre o estado atual do livro-razão distribuído. Esses protocolos eliminam a necessidade de uma autoridade central e protegem a rede contra fraudes (NARAYANAN et al., 2016).

2.2.1 Proof of Work (PoW) – Prova de Trabalho

O *Proof of Work* (PoW) foi o primeiro algoritmo de consenso implementado, sendo utilizado pelo Bitcoin (NAKAMOTO, 2008). Embora seja altamente seguro, o PoW apresenta desafios como alto consumo de energia e baixa escalabilidade, tornando-o inadequado para sistemas de autenticação documental em escala governamental (STOLL; KLAASSEN; GALLERSDÖRFER, 2019).

2.2.2 Proof of Stake (PoS) – Prova de Participação

Como alternativa ao alto consumo energético do PoW, surgiu o *Proof of Stake* (PoS). O Ethereum, originalmente projetado sobre PoW (BUTERIN, 2014), migrou para o PoS em setembro de 2022, na atualização conhecida como *The Merge*. O PoS é mais eficiente em termos energéticos e melhora a escalabilidade, sendo uma opção a ser considerada para implementações futuras em escala.

2.2.3 Delegated Proof of Stake (DPoS)

O DPoS é uma variação do PoS introduzida pela rede EOS (LARIMER, 2014). Nesse modelo, detentores de tokens votam em validadores confiáveis, melhorando a escalabilidade, embora possa gerar concentração de poder.

2.2.4 Practical Byzantine Fault Tolerance (PBFT)

O PBFT (CASTRO; LISKOV, 1999) é um mecanismo de consenso projetado para redes permissionadas, amplamente utilizado em *blockchains* empresariais por garantir alta eficiência e baixa latência, sendo adequado para sistemas de autenticação documental que exigem confirmação de transações em poucos segundos. O *Hyperledger Fabric*, especificamente, não emprega o PBFT: seu serviço de ordenação é baseado no protocolo Raft (*etcdraft*) (LINUX FOUNDATION, 2021), detalhado a seguir.

2.2.5 Raft

O Raft (ONGARO; OUSTERHOUT, 2014) é um protocolo de consenso por eleição de líder e replicação de logs, projetado como alternativa mais compreensível ao Paxos para sistemas distribuídos que toleram falhas por interrupção (*crash fault tolerance*), sem, no entanto, tolerar falhas bizantinas como o PBFT. O Hyperledger Fabric adota o Raft como serviço de ordenação (*ordering service*) desde a versão 1.4.1, substituindo a alternativa baseada em Kafka: um nó líder eleito entre os *orderers* recebe as transações endossadas, agrupa-as em blocos e replica o log para os demais nós ordenadores, garantindo tolerância a falhas por parada desde que a maioria dos nós permaneça operacional (ONGARO; OUSTERHOUT, 2014). Essa escolha é

adequada ao modelo de confiança do protótipo, no qual as organizações participantes (cartórios, universidades, conselhos profissionais) são entidades identificadas e permissionadas, e não participantes anônimos potencialmente maliciosos, cenário para o qual a tolerância bizantina do PBFT seria exigida.

2.3 Provas de Conhecimento Zero

Os mecanismos de consenso apresentados na Seção 2.2 garantem a integridade das transações, mas nem sempre preservam a privacidade dos dados. Nesse contexto, surgem as Provas de Conhecimento Zero (*Zero-Knowledge Proofs* – ZKPs), que permitem a verificação de um documento sem revelar seu conteúdo (GOLDWASSER; MICALI; RACKOFF, 1985). A prova π é gerada conforme a Equação 2.2:

$$\pi = \text{Prove}(\text{CRS},\, x,\, w) \tag{2.2}$$

onde CRS (*Common Reference String*) é um parâmetro público, x é a entrada pública e w é o segredo privado (GOLDWASSER; MICALI; RACKOFF, 1985). Essa propriedade é especialmente relevante para documentos de alta sensibilidade, como atestados médicos e laudos diagnósticos, nos quais a autenticidade precisa ser verificada sem exposição de dados de saúde.

2.3.1 ZK-SNARKs

Os ZK-SNARKs (*Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge*) permitem que uma prova seja gerada e verificada de forma rápida e eficiente, sem necessidade de interação contínua entre provador e verificador. Essa tecnologia é implementada no protótipo por meio da ferramenta *ZoKrates*.

2.3.2 ZK-STARKs

Os ZK-STARKs eliminam a necessidade de uma configuração confiável e oferecem maior escalabilidade e segurança (BEN-SASSON et al., 2018), constituindo uma alternativa promissora para implementações futuras do sistema.

2.4 Redes Permissionadas

As redes permissionadas são *blockchains* privadas nas quais o acesso é restrito a entidades autorizadas, permitindo que empresas e instituições utilizem os benefícios da tecnologia sem abdicar do controle sobre os dados. A Tabela 2.1 compara as principais características entre *blockchains* públicas e permissionadas.

Tabela 2.1 – Comparação entre *Blockchain* Pública e Permissionada.

Característica	**Pública (Bitcoin, Ethereum)**	**Permissionada (Fabric)**
Acesso	Aberto a qualquer usuário	Restrito a participantes autorizados
Desempenho	Menor escalabilidade	Alto desempenho e eficiência
Privacidade	Dados acessíveis a todos	Dados acessíveis conforme perfil
Governança	Descentralizada e aberta	Controlada por conjunto de entidades
Conformidade	Difícil adequação à LGPD	Adequada à LGPD por design

Fonte: Elaborado pelo autor (2026).

Para sistemas de autenticação de documentos legais no contexto brasileiro, as redes permissionadas são a escolha natural: permitem definir com precisão quem pode emitir, verificar ou revogar cada categoria documental, em conformidade com as respectivas regulações setoriais.

2.5 Hyperledger Fabric

O *Hyperledger Fabric* (ANDROULAKI et al., 2018) é um *framework* de *blockchain* permissionado projetado para ambientes empresariais e institucionais que demandam alta customização, privacidade e desempenho. Diferentemente de redes públicas como o Ethereum, o Fabric adota uma abordagem modular, permitindo que organizações selecionem componentes específicos conforme suas necessidades. Essa flexibilidade é especialmente valiosa para sistemas de autenticação documental multi-setorial, pois possibilita a criação de canais privados segmentados, por exemplo, um canal dedicado a escrituras imobiliárias, outro a documentos de saúde, nos quais apenas as entidades autorizadas participam da validação e auditoria.

2.5.1 Componentes Principais da Arquitetura

2.5.1.1 Peers

São os nós responsáveis por armazenar o *ledger* e executar *chaincodes*. Cada *peer* mantém um *ledger* de transações, registro imutável de operações, criptografado com SHA-256 e um banco de estados, que armazena o estado atual dos ativos em bancos NoSQL como CouchDB (ANDROULAKI et al., 2018). Para documentos heterogêneos, o CouchDB é especialmente relevante por suportar consultas ricas sobre atributos como tipo de documento, status e identificador da instituição emissora.

2.5.1.2 Ordenadores

Responsáveis por ordenar transações e agrupá-las em blocos. No protótipo, utiliza-se o protocolo Raft, que garante consenso por eleição de líderes e replicação de logs, sendo adequado para redes com múltiplas organizações emissoras de tipos distintos de documentos.

2.5.1.3 Canais

Sub-redes privadas dentro do Fabric que permitem comunicação confidencial entre participantes específicos. Em uma evolução futura da arquitetura, canais distintos poderiam isolar categorias documentais de alta sensibilidade das de sensibilidade média ou baixa, garantindo que informações sensíveis não sejam expostas a partes não autorizadas; o protótipo atual, descrito na Seção 3.3.1.2, opera sobre um único canal (`documentos-channel`) compartilhado por todos os tipos documentais.

2.5.1.4 Provedores de Serviço de Membresia (MSP)

O MSP gerencia identidades digitais via certificados X.509, atribuindo papéis a cada entidade. Em sistemas multi-setoriais, o MSP permite definir papéis distintos por tipo de documento: cartórios como emissores de escrituras, médicos como emissores de atestados, universidades como emissoras de diplomas.

2.5.1.5 Chaincode

São contratos inteligentes que definem a lógica de negócio da rede. Para suportar documentos heterogêneos, o *chaincode* do protótipo adota uma estrutura genérica com campo de tipo documental e metadados extensíveis, conforme detalhado na Seção 3.3.2, em vez de uma estrutura rígida por categoria.

2.6 Sistema de Arquivos Interplanetário (IPFS)

O *InterPlanetary File System* (IPFS), proposto por Benet (2014), é um protocolo de rede *peer-to-peer* (P2P) projetado para revolucionar o armazenamento e a distribuição de dados na internet. Diferentemente do modelo tradicional baseado em localização (por exemplo, URLs), o IPFS utiliza identificação por conteúdo (*Content Identifier* – CID), na qual cada arquivo é representado por um hash criptográfico único derivado de seu conteúdo. Essa abordagem elimina a dependência de servidores centralizados, tornando o IPFS ideal para integrar sistemas anti-falsificação, como registros de documentos imobiliários, laudos médicos ou contratos legais, nos quais a imutabilidade e a disponibilidade permanente são essenciais.

2.6.1 Funcionamento Técnico do IPFS

Ao armazenar um documento, o IPFS fragmenta o arquivo em blocos, gera um CID usando SHA-256 e distribui os blocos entre nós da rede. Qualquer alteração no conteúdo, mesmo mínima, gera um novo CID distinto, o que torna a detecção de adulteração automática e determinística (BENET, 2014). A rede distribuída dos nós garante resiliência: mesmo que alguns nós saiam da rede, o conteúdo permanece acessível (MUKNE et al., 2019).

2.6.2 Integração Blockchain–IPFS

A *blockchain* armazena o CID do documento e seus metadados, enquanto o conteúdo original é mantido no IPFS. A verificação opera em dupla camada: a validação do *hash* na *blockchain* confirma a autenticidade do registro, e a comparação do CID no IPFS assegura que o conteúdo não foi alterado pós-registro. Esse modelo é aplicável a qualquer categoria documental suportada pelo sistema, independentemente do volume ou da natureza do arquivo.

2.7 Identificadores Descentralizados (DIDs) – Padrão W3C

Os Identificadores Descentralizados (DIDs) são um padrão W3C (SPORNY et al., 2022b) para identificação digital que elimina a dependência de autoridades centrais. Um DID é um identificador único e persistente, cuja forma geral é `did:método:identificador-específico`, e que resolve para um *DID Document* contendo a chave pública do sujeito e os métodos de verificação suportados.

No contexto de documentos legais heterogêneos, os DIDs resolvem um problema estrutural: diferentes categorias documentais envolvem diferentes tipos de emissores e diferentes tipos de titulares, que não compartilham sistemas de identidade comuns. Os métodos `did:web` para instituições que possuem domínio *web* estabelecido, como cartórios, universidades e hospitais, e `did:key` para titulares individuais de documentos, permitem identificação verificável em todos esses contextos sem infraestrutura adicional de *blockchain*.

2.8 Certificados X.509

Os certificados X.509, definidos pela RFC 5280 (COOPER et al., 2008), desempenham papel central em redes *blockchain* permissionadas como o *Hyperledger Fabric*. O MSP utiliza certificados X.509 para autenticar participantes e atribuir funções específicas. Em sistemas multisetoriais, os certificados X.509 funcionam em camada complementar aos DIDs W3C: enquanto os primeiros controlam o acesso à rede *blockchain*, os segundos fornecem identidade verificável publicamente sem dependência de infraestrutura específica.

No Brasil, o padrão X.509 é adotado pela ICP-Brasil (*Infraestrutura de Chaves Públicas Brasileira*), instituída pela Medida Provisória n.º 2.200-2/2001 (BRASIL, 2001), garantindo a autenticidade de documentos eletrônicos por meio de uma cadeia hierárquica de Autoridades Certificadoras. A compatibilidade entre o MSP do Fabric e a ICP-Brasil representa uma vantagem relevante para a adoção do sistema por instituições públicas brasileiras.

2.9 Classificação de Documentos Legais

A heterogeneidade das categorias documentais tratadas pelo sistema exige uma classificação estruturada, que oriente as decisões de controle de acesso, ciclo de vida e nível de privacidade aplicados a cada tipo de documento. Neste trabalho, propõe-se uma classificação baseada em três critérios: **sensibilidade** dos dados contidos, **perfil do emissor** autorizado e **ciclo de vida** esperado.

A Tabela 2.2 apresenta a taxonomia adotada, com exemplos representativos de cada categoria.

Tabela 2.2 – Classificação de documentos legais por sensibilidade, emissor e ciclo de vida.

Nível	**Exemplos**	**Emissor**	**Regulação**	**Ciclo de vida**
Alta sensibilidade	Atestados médicos, laudos diagnósticos, receituários	Médico (CRM)	LGPD Art. 11; CFM	Válido por prazo; revogável
Média sensibilidade	Escrituras de imóveis, contratos de compra e venda, certidões de estado civil	Cartório (Lei n.º 8.935/1994)	Lei n.º 8.935/1994; Lei n.º 6.015/1973; CRI	Encadeado: cada novo ato sucede o anterior
Baixa sensibilidade	Diplomas universitários, CNH	IES; DETRAN	LDB; CTB	Emitido uma vez; raramente revogado

Fonte: Elaborado pelo autor (2026).

Essa classificação fundamenta diretamente a lógica do *chaincode*: documentos de alta sensibilidade ativam automaticamente os mecanismos de ZK-SNARKs, com acesso restrito via canal privado como evolução possível da arquitetura (Seção 2.5.1.3); documentos de média sensibilidade preservam o histórico de versões do próprio documento por meio do `GetHistoryForKey` do chaincode (Seção 3.3.2) – mecanismo distinto do encadeamento entre atos sucessivos de uma mesma escritura sugerido na Tabela 2.2, que não foi implementado no protótipo; documentos de baixa sensibilidade permitem consultas mais abertas e fluxos de emissão simplificados. A implementação detalhada é descrita na Seção 3.3.2.

2.10 Trabalhos Relacionados

Diversos estudos exploram o uso de *blockchain* e tecnologias associadas para emissão, autenticação, transferência e prevenção de fraude de documentos legais. Esta seção revisa os principais trabalhos relacionados ao tema, destacando suas contribuições, abordagens metodológicas e limitações, oferecendo uma base sólida para o desenvolvimento deste estudo.

Nakamoto (2008) introduziu o conceito fundamental do *blockchain* como uma rede descentralizada, garantindo a imutabilidade e autenticidade das transações através do consenso por prova de trabalho (Proof-of-Work). Esse sistema transformou a forma como a confiança é estabelecida em ambientes distribuídos e serve como alicerce para muitas das aplicações atuais, incluindo aquelas voltadas para documentos legais.

Brunner (2017) propôs o *Eduthereum*, uma solução baseada em *blockchain* público para armazenar certificados educacionais. Este trabalho destaca como características de imutabilidade e transparência do *blockchain* podem ser aplicadas na emissão e verificação de certificados, assegurando autenticidade sem a necessidade de intermediários confiáveis. Apesar disso, a abordagem apresenta desafios quanto à escalabilidade e privacidade dos dados.

Christidis e Devetsikiotis (2016) exploraram o uso de *blockchains* e contratos inteligentes para a Internet das Coisas (IoT), sugerindo frameworks que podem ser adaptados para autenticação e transferência de documentos. Sua aplicação no contexto legal é promissora, pois contratos inteligentes permitem verificar e executar condições automaticamente, reduzindo a possibilidade de fraude.

Deshpande et al. (2015) analisaram mecanismos para prevenir vazamentos de informações em documentos, propondo frameworks que podem ser integrados com sistemas baseados em *blockchain* para assegurar confidencialidade. Contudo, o trabalho não aborda explicitamente a integração de *blockchains* para autenticação descentralizada.

Kosba et al. (2016) apresentaram o *Hawk*, um modelo de contratos inteligentes voltado para privacidade e segurança. Essa abordagem se mostra relevante para o contexto de documentos legais, onde é crucial proteger informações sensíveis enquanto se mantém um histórico verificável das transações.

Tschorsch e Scheuermann (2016) realizaram um levantamento técnico abrangente sobre moedas digitais descentralizadas, com foco nas limitações e melhorias potenciais dos sistemas baseados em *blockchain*. As análises técnicas fornecem perspectivas sobre como adaptar as estruturas subjacentes para aplicações específicas, como a prevenção de fraudes em documentos.

Vukolić (2016) comparou mecanismos de consenso, como Proof-of-Work e Byzantine Fault Tolerance (BFT), destacando os prós e contras de cada abordagem no contexto de redes escaláveis. Sua análise fornece fundamentos para escolher a tecnologia apropriada ao projetar sistemas robustos de emissão e autenticação de documentos.

Além dos trabalhos acadêmicos anteriores, três iniciativas de aplicação prática são especialmente relevantes por serem retomadas como parâmetro de comparação no Capítulo 4 (Seção 4.5). O *OpenCerts* (GOVERNMENT TECHNOLOGY AGENCY OF SINGAPORE (GOVTECH), 2020), iniciativa do governo de Singapura, emite certificados educacionais sobre a *blockchain* pública Ethereum, registrando o hash do documento *on-chain* e reduzindo o tempo de verificação por empregadores de um processo manual de dias para poucos segundos; sua limitação central para o contexto deste trabalho é o escopo restrito a um único tipo documental e a ausência de mecanismos de privacidade como ZK-SNARKs.

A *European Blockchain Services Infrastructure* (EBSI) (EUROPEAN COMMISSION, 2020) é a iniciativa da União Europeia para certificação de qualificações profissionais e documentos públicos entre países-membros, apoiada em *blockchain* permissionada (Hyperledger Besu) e

em identidades EBSI-DID compatíveis com o padrão W3C. Sua abrangência multinacional e multi-norma é maior que a do protótipo aqui proposto, mas sua camada de privacidade seletiva ainda não incorpora provas de conhecimento zero com garantias formais como o Groth16.

Por fim, Mukne et al. (2019) propuseram um sistema de registro de propriedades imobiliárias combinando Hyperledger Fabric e IPFS, arquitetura conceitualmente próxima à adotada neste trabalho: metadados e identificadores no *ledger*, documentos originais no IPFS. A principal diferença é o escopo, restrito a um único tipo documental (registros de imóveis) e sem camada de identidade W3C (DIDs e Credenciais Verificáveis), o que limita a verificação por terceiros sem acesso direto à rede Fabric.

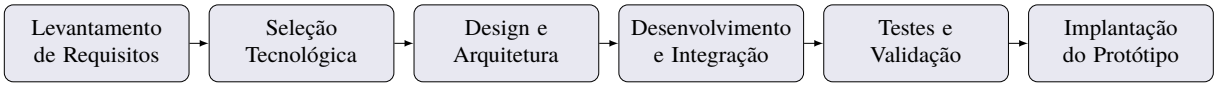
Esses trabalhos oferecem perspectivas complementares sobre o uso do *blockchain* e tecnologias associadas em diferentes contextos, sendo fundamentais para o desenvolvimento de soluções inovadoras e seguras no campo de documentos legais.

3 Desenvolvimento

3.1 Metodologia

Esta seção descreve o percurso metodológico adotado para o desenvolvimento do protótipo de autenticação de documentos legais baseado em tecnologia *blockchain*. A pesquisa estrutura-se em torno de uma abordagem aplicada, exploratória e descritiva, compreendendo as seguintes etapas sequenciais e iterativas: levantamento de requisitos, seleção tecnológica, projeto da arquitetura modular, implementação incremental dos componentes, testes e validação, e implantação do protótipo. A Figura 3.1 apresenta a linha do tempo representativa do desenvolvimento.

Figura 3.1 – Linha do Tempo do Desenvolvimento da Solução.



Fonte: Elaborado pelo autor (2026).

3.1.1 Abordagem Metodológica

Esta seção descreve a abordagem metodológica adotada no desenvolvimento do protótipo, integrando as modalidades de pesquisa empregadas e os critérios que orientaram a seleção das tecnologias utilizadas.

3.1.2 Integração de Pesquisas Aplicada, Exploratória e Descritiva

A metodologia adotada combina três modalidades complementares de pesquisa, cujas fronteiras se interpenetram ao longo de todo o ciclo de desenvolvimento.

Pesquisa Aplicada: Visando solucionar um problema concreto da sociedade, a falsificação de documentos legais, este trabalho utiliza conhecimentos técnicos e científicos para desenvolver uma solução prática e demonstrável. A aplicação de tecnologias como *blockchain* permissionado, armazenamento descentralizado e identificadores descentralizados destaca seu caráter aplicado, pois busca implementar um sistema funcional, seguro e auditável para autenticação documental no contexto brasileiro.

Pesquisa Exploratória: Dado o caráter inovador e a adoção ainda incipiente dessas tecnologias em escala governamental no Brasil, o estudo explora: (*i*) a integração entre *Hyperledger Fabric* e *IPFS*, avaliando sinergias e desafios práticos; (*ii*) a viabilidade do padrão W3C de *Decentralized Identifiers* (DIDs) como mecanismo de identidade em sistemas de autenticação

documental; e (*iii*) as barreiras técnicas e regulatórias específicas do contexto brasileiro, como a heterogeneidade da infraestrutura de TI, a ausência de padrões interoperáveis e as exigências da Lei Geral de Proteção de Dados Pessoais (LGPD, Lei n.º 13.709/2018).

Pesquisa Descritiva: A fase descritiva detalha cada tecnologia empregada e o funcionamento dos componentes do sistema. São documentados os protocolos de segurança como o uso de ZK-SNARKs para validação sem exposição de dados sensíveis, a arquitetura modular que integra *blockchain* com armazenamento descentralizado, e os critérios técnicos que norteiam a implementação, facilitando sua reprodução e futuras melhorias.

3.1.3 Seleção de Ferramentas: *Blockchain* – Hyperledger Fabric

O *Hyperledger Fabric* (ANDROULAKI et al., 2018) foi selecionado como plataforma *blockchain* pela sua adequação a ambientes permissionados e pela necessidade de segurança reforçada no controle de documentos sensíveis. A escolha fundamenta-se nos seguintes critérios:

a) **Controle de Acesso Granular:** O Fabric utiliza mecanismos de listas de controle de acesso (ACLs) e políticas de endosso configuráveis, garantindo que operações específicas sejam realizadas somente por entidades autorizadas. Por exemplo, apenas universidades credenciadas no sistema podem invocar a função de emissão de diplomas, enquanto qualquer participante autenticado pode executar consultas de verificação.

b) **Desempenho e Baixa Latência:** Diferentemente de redes que empregam *Proof-of-Work* (PoW), o Fabric implementa o protocolo de consenso Raft, que possibilita a finalização de blocos em aproximadamente 1 a 2 segundos (LINUX FOUNDATION, 2021), sendo adequado para sistemas que requerem respostas em tempo quase real.

c) **Arquitetura Modular:** A rede é organizada em múltiplos nós (*Orderer*, *Peer*, CouchDB, CLI) implantados em contêineres Docker, simulando um ambiente distribuído realista. Essa modularidade favorece a escalabilidade e a manutenção do sistema ao longo do ciclo de vida do sistema.

d) **Compatibilidade com a LGPD:** Por ser uma rede permissionada, o Fabric permite restringir o acesso a dados pessoais apenas às entidades autorizadas, em conformidade com o princípio da finalidade e com os demais princípios previstos no art. 6.º da LGPD (BRASIL, 2018).

e) **Gerenciamento de Identidade via MSP:** O *Membership Service Provider* (MSP) do Fabric autentica participantes por meio de certificados X.509, atribuindo papéis funcionais, emissor, validador, auditor, a cada entidade. Esse mecanismo é complementado, no presente trabalho, pelos Identificadores Descentralizados do padrão W3C, conforme descrito na Seção 3.1.6.

O *chaincode* equivalente aos contratos inteligentes em plataformas públicas é implementado na linguagem Go e expõe cinco operações principais sobre um modelo de documento genérico: `EmitirDocumento`, `VerificarDocumento`, `RevogarDocumento`, `ListarPorTipo` e `HistoricoDocumento`, detalhadas na Seção 3.3.2. O banco de dados de estado CouchDB é empregado em substituição ao LevelDB padrão, viabilizando consultas ricas (Mango Queries) sobre atributos dos documentos registrados, como identificador da instituição emissora e status de validade. A implantação dos nós é realizada por meio do seguinte comando ilustrativo:

Código 3.1 – Deploy dos contêineres Docker do Hyperledger Fabric

```
docker-compose -f docker-compose-fabric.yaml up -d
```

3.1.4 Seleção de Ferramentas: Armazenamento Descentralizado – IPFS

Dada a necessidade de manipular documentos potencialmente volumosos e garantir a imutabilidade dos dados sem sobrecarregar o *ledger* da *blockchain*, o *InterPlanetary File System* (IPFS) foi integrado ao sistema para oferecer armazenamento distribuído de alta confiabilidade (BENET, 2014).

A escolha do IPFS como camada de armazenamento decorre de três propriedades fundamentais:

a) **Identificação por Conteúdo (Content Addressing):** Cada arquivo armazenado no IPFS recebe um *Content Identifier* (CID) exclusivo, derivado do hash criptográfico SHA-256 de seu conteúdo. Qualquer alteração no arquivo, mesmo mínima, gera um CID completamente distinto, tornando a adulteração detectável de forma automática e determinística.

b) **Escalabilidade e Descentralização:** Ao distribuir os dados por meio de uma rede *peer-to-peer*, o IPFS elimina pontos únicos de falha e possibilita acesso robusto mesmo em cenários de alta demanda ou infraestrutura heterogênea, característica relevante para o contexto brasileiro (MUKNE et al., 2019).

c) **Eficiência de Custo:** A combinação do IPFS com a *blockchain* para armazenar exclusivamente metadados, como o CID e o hash SHA-256 do documento, permite reduzir significativamente o volume de dados registrados no *ledger*, tornando as transações mais eficientes.

No protótipo, é utilizado o cliente *Kubo* (implementação de referência do IPFS em Go) implantado localmente via Docker. A integração com a camada de aplicação ocorre por meio da biblioteca `kubo-rpc-client` para Node.js. O trecho de código a seguir ilustra o procedimento de armazenamento de um documento PDF no IPFS:

Código 3.2 – Upload de documento para o IPFS via kubo-rpc-client

```
const { create } = require('kubo-rpc-client');
const ipfs = create({ url: 'http://localhost:5001' });

async function armazenarDocumento(buffer) {
  const resultado = await ipfs.add(buffer);
  const cid = resultado.cid.toString();
  // Ex.: "QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco"
  return cid;
}
```

O mecanismo de verificação de integridade baseia-se na comparação entre o hash SHA-256 recalculado a partir do arquivo recuperado do IPFS e o hash registrado imutavelmente no *ledger* do Fabric. Qualquer divergência entre esses valores indica adulteração do conteúdo, independentemente de o CID permanecer o mesmo no registro.

3.1.5 Privacidade: ZK-SNARKs com ZoKrates

O tratamento de dados sensíveis como laudos médicos, exames e documentos de identificação impõe requisitos de privacidade que transcendem a simples restrição de acesso. Nesse contexto, são empregadas Provas de Conhecimento Zero (*Zero-Knowledge Proofs* – ZKPs) para permitir a verificação da autenticidade de um documento sem que seu conteúdo seja revelado ao verificador (GOLDWASSER; MICALI; RACKOFF, 1985).

A implementação utiliza ZK-SNARKs (*Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge*) por meio da ferramenta *ZoKrates* (EBERHARDT; TAI, 2018), que disponibiliza uma linguagem de domínio específico para a definição de circuitos aritméticos e ferramentas de linha de comando para compilação, geração de testemunhos (*witnesses*), geração de provas e verificação. Essa abordagem permite incorporar provas ZK-SNARKs reais ao sistema sem a necessidade de implementar os algoritmos criptográficos subjacentes diretamente.

A escolha por ZK-SNARKs (esquema Groth16) em vez de ZK-STARKs (Seção 2.3.2) para o protótipo se justifica por três fatores práticos, além da maturidade do *toolchain* ZoKrates: o tamanho da prova Groth16 é constante e da ordem de algumas centenas de *bytes*, contra provas STARK tipicamente de dezenas a centenas de quilobytes; o custo computacional de verificação é significativamente menor, o que é relevante para um sistema que pode envolver verificação por dispositivos com poucos recursos; e o ZoKrates oferece um fluxo de compilação, *setup* e geração de prova integrado e documentado, sem equivalente direto e igualmente maduro para STARKs no momento do desenvolvimento. Em contrapartida, o Groth16 exige um *trusted setup* por circuito (Seção 3.3.5.2), risco ausente nos ZK-STARKs, o que é registrado como limitação do protótipo (L2, Seção 4.6) e mantido como direção de evolução para versões futuras do sistema.

O circuito implementado no protótipo realiza a seguinte afirmação verificável: *"O pro-*

vador conhece um valor privado (`hashPrivado`) igual ao valor já registrado publicamente no ledger (`hashPublico`)". Formalmente, a prova π é gerada conforme a Equação (2.2), já apresentada na Seção 2.3, na qual CRS (*Common Reference String*) é um parâmetro público gerado durante o *trusted setup*; x representa a entrada pública (o hash já registrado no *ledger*); e w é o testemunho privado fornecido pelo provador, que deve ser numericamente igual a x para que a prova seja válida (GOLDWASSER; MICALI; RACKOFF, 1985).

Cabe uma ressalva importante quanto ao alcance dessa prova: como `hashPublico` é, por definição, o mesmo hash SHA-256 já publicado no *ledger* (Seção 2.6.2), o circuito não computa o SHA-256 do documento internamente nem vincula criptograficamente a prova ao conteúdo do documento original; ele apenas atesta a posse de um valor numericamente igual a um dado que já é público. O circuito constitui, portanto, uma prova de conceito do fluxo de compilação, *trusted setup*, geração e verificação de provas do *toolchain* ZoKrates (Seção 3.2), sem oferecer, na versão atual, garantia de privacidade documental: qualquer parte que já conheça o hash publicado, o que inclui qualquer verificador com acesso de leitura ao *ledger*, pode inspecioná-lo diretamente, sem necessidade da prova ZK. A incorporação do cálculo do SHA-256 dentro do próprio circuito, de modo que a prova vincule genuinamente o documento original sem expor o hash publicamente, é indicada como extensão necessária em Trabalhos Futuros (Seção 5.2).

O fluxo de geração e verificação de provas com ZoKrates compreende as seguintes etapas:

a) **Definição do circuito:** O circuito é escrito na linguagem ZoKrates, declarando entradas públicas e privadas e a condição a ser provada (`assert(hashPrivado == hashPublico)`);

b) **Compilação e *trusted setup*:** O circuito é compilado e o *setup* confiável gera as chaves de prova (*proving key*) e verificação (*verification key*);

c) **Geração da prova:** O provador fornece o testemunho privado e obtém a prova π, que pode ser transmitida publicamente;

d) **Verificação:** O verificador utiliza apenas a *verification key* pública e a prova π para confirmar a afirmação, sem acesso ao testemunho privado.

O circuito é exposto por meio de um *endpoint* adicional na API (`POST /documentos/:id/proof`), que gera e retorna a prova ZKP para um documento previamente registrado. A verificação é disponibilizada em `GET /documentos/:id/verify-proof`.

3.1.6 Identificação Descentralizada: DIDs W3C (*did:web* e *did:key*)

A identificação descentralizada constitui um pilar fundamental para garantir a autenticidade e a rastreabilidade dos documentos emitidos. Este trabalho adota o padrão de *Decentralized Identifiers* (DIDs) definido pelo *World Wide Web Consortium* (W3C) (SPORNY et al., 2022b), em substituição à abordagem originalmente proposta com o *Hyperledger Indy*. Essa decisão

decorre de três razões objetivas: (*i*) a maturidade e o amplo suporte do padrão W3C, formalmente publicado como Recomendação em julho de 2022; (*ii*) a possibilidade de implementação sem dependência de infraestrutura adicional de *blockchain*; e (*iii*) a compatibilidade nativa com o ecossistema de Credenciais Verificáveis (VCs) do W3C (SPORNY et al., 2022a), que fundamenta a arquitetura de identidade do protótipo.

Um DID é um identificador único e persistente, controlado pelo próprio sujeito identificado, cuja forma geral é:

$$\texttt{did:}\textit{método}\texttt{:}\textit{identificador-específico}$$

O DID resolve para um *DID Document*, um documento JSON que contém a chave pública do sujeito, os métodos de verificação suportados e, opcionalmente, *endpoints* de serviço. A resolução é realizada de acordo com o método DID empregado, sem necessidade de autoridade central.

O presente protótipo adota dois métodos DID complementares, aplicados a perfis distintos de participantes:

3.1.6.1 *did:web* – Identidade das Instituições Emissoras

O método `did:web` (TERBU et al., 2023) vincula a identidade descentralizada ao domínio web já estabelecido de uma instituição. O DID Document é hospedado em um caminho padronizado do servidor web da organização e resolvido por meio de uma requisição HTTPS convencional:

$$\texttt{did:web:ufop.br} \longrightarrow \texttt{GET https://ufop.br/.well-known/did.json}$$

Essa abordagem apresenta vantagens significativas para o contexto institucional brasileiro: (*i*) não exige infraestrutura de *blockchain* adicional para a resolução do identificador; (*ii*) é imediatamente verificável por qualquer entidade com acesso à internet, aproveitando a confiança já estabelecida no domínio institucional; e (*iii*) permite rotação de chaves criptográficas sem alteração do identificador.

No protótipo, o DID Document da instituição emissora contém o método de verificação associado à chave pública Ed25519 utilizada para assinar as Credenciais Verificáveis. Durante o desenvolvimento, um servidor local simulou o endpoint de resolução, mantendo a aderência ao padrão sem dependência do domínio real da instituição.

3.1.6.2 *did:key* – Identidade dos Titulares dos Documentos

Para os titulares dos documentos como os alunos, no caso do diploma universitário, adota-se o método `did:key` (W3C CREDENTIALS COMMUNITY GROUP, 2023), que deriva

o identificador diretamente de um par de chaves criptográficas, sem qualquer registro externo ou infraestrutura adicional. O DID é gerado por meio da codificação multibase da chave pública Ed25519 do titular:

`did:key:z6Mkf5rGMoatrSj1f4CyvuHBeXJELe9RPdzo2PKGNCKVtZxP`

A propriedade fundamental do `did:key` é que o DID Document é derivado deterministicamente da chave pública, sendo resolvido localmente sem qualquer requisição de rede. Isso elimina dependências de infraestrutura para a identificação dos titulares, tornando o método adequado para contextos onde os sujeitos não dispõem de servidores web ou registros prévios.

3.1.6.3 Integração dos DIDs na Arquitetura do Protótipo

Na arquitetura proposta, os dois DIDs são registrados diretamente na estrutura do documento armazenado no *ledger* do Fabric, independentemente do tipo documental. A Tabela 3.1 apresenta os campos adicionados ao modelo de dados do documento para suportar a identificação descentralizada.

Tabela 3.1 – Campos de identificação descentralizada no modelo de dados do documento.

Campo	**Tipo**	**Descrição**
`EmissorDID`	`string`	DID W3C da instituição emissora (*did:web*)
`TitularDID`	`string`	DID W3C do titular do documento (*did:key*)
`ProvaVC`	`string`	Assinatura `Ed25519Signature2020` (`proofValue` multibase) da Credencial Verificável

Fonte: Elaborado pelo autor (2026).

O fluxo de emissão é estendido para incorporar a geração e a assinatura da Credencial Verificável (VC): após o armazenamento do PDF no IPFS e o cálculo do hash SHA-256, a API gera uma VC estruturada segundo o padrão W3C (SPORNY et al., 2022a), cujo *credentialSubject* referencia o `TitularDID` e cujos metadados incluem o CID e o hash do documento. A VC é então assinada com a chave privada da instituição cuja chave pública está publicada no DID Document `did:web` e o campo `ProvaVC` é registrado no *ledger* juntamente com os demais metadados do documento.

Na verificação, além das etapas de consulta ao *ledger* e de recalculação do hash via IPFS já descritas, é realizada a verificação criptográfica da assinatura da VC: o verificador resolve o DID da instituição (`did:web`), obtém a chave pública correspondente e valida a assinatura. Esse mecanismo garante não-repúdio e rastreabilidade criptograficamente verificáveis, independentemente de qualquer autoridade central.

O trecho a seguir ilustra a estrutura JSON da Credencial Verificável gerada pelo sistema:

Código 3.3 – Estrutura da Credencial Verificável (VC) gerada pelo protótipo

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
  ],
  "type": ["VerifiableCredential"],
  "issuer": "did:web:ufop.br",
  "issuanceDate": "2026-04-16T00:00:00Z",
  "credentialSubject": {
    "id": "did:key:z6Mkf5rGMoatrSj1f4CyvuHBeXJELe9RPdzo2PKGNCKVtZxP",
    "tipo": "DIPLOMA",
    "cid": "QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco",
    "hash": "a3f8b2c1d9e4f768...",
    "metadados": {
      "curso": "Ciencia da Computacao",
      "grau": "Bacharelado"
    }
  },
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "2026-04-16T00:00:00Z",
    "verificationMethod": "did:web:ufop.br#key-1",
    "proofPurpose": "assertionMethod",
    "proofValue": "z3sN1oBu5tRv2WCbxJZBLtWSHZLQMcQ..."
  }
}
```

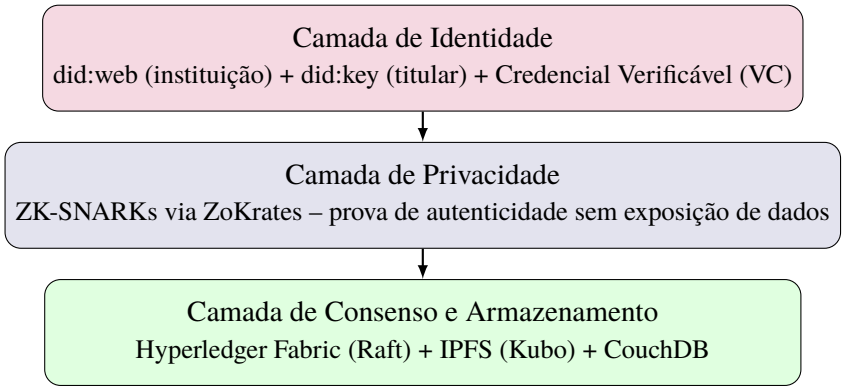
3.1.7 Desenvolvimento da Arquitetura Modular: Sinergia entre Identidade, Privacidade e Consenso

A arquitetura do protótipo é organizada em três camadas funcionais interdependentes, cada uma com responsabilidades bem definidas e interfaces padronizadas, conforme ilustra a Figura 3.2.

Camada de Identidade: Responsável por registrar e gerenciar as identidades dos participantes e dos documentos. O fluxo de processamento inclui: (*i*) geração do par de chaves Ed25519 para o titular e derivação do `did:key`; (*ii*) resolução do `did:web` da instituição e obtenção da chave pública de assinatura; (*iii*) emissão e assinatura da Credencial Verificável (VC) estruturada segundo o padrão W3C; e (*iv*) registro dos DIDs e da prova da VC no *ledger* do Fabric.

Camada de Privacidade: Sobre a camada de identidade, esta camada incorpora as técnicas de ZK-SNARKs via ZoKrates, assegurando que a verificação da autenticidade documental possa ocorrer sem a exposição de dados sensíveis. A camada é ativada sob demanda, por meio dos *endpoints* específicos de geração e verificação de provas da API.

Figura 3.2 – Arquitetura Modular Integrada do Protótipo.



Fonte: Elaborado pelo autor (2026).

Camada de Consenso e Armazenamento: Responsável pela persistência imutável dos registros e pela disponibilidade dos documentos originais. O protocolo de consenso Raft do Hyperledger Fabric valida as transações propostas pelos clientes e garante que somente blocos aprovados pela maioria dos nós ordenadores sejam adicionados à cadeia. O IPFS assegura a disponibilidade descentralizada dos PDFs originais, enquanto o CouchDB mantém o estado atual dos registros e viabiliza consultas complexas.

A função de registro de documentos no *chaincode* Go, que materializa a integração entre as três camadas, opera sobre a *struct* `Documento` genérica (válida para os cinco tipos documentais suportados, e não apenas para diplomas): verifica a ausência prévia do ID, deriva a sensibilidade a partir do tipo, registra a identidade MSP do emissor e persiste o registro via `PutState`. A implementação completa da função `EmitirDocumento` é apresentada no Código 3.6, na Seção 3.3.2.

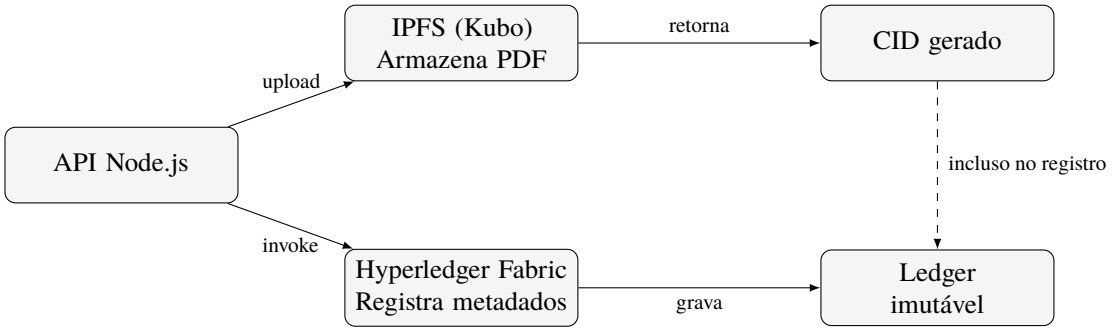
3.1.8 Camada de Armazenamento e Interoperabilidade

A camada de armazenamento e interoperabilidade articula o armazenamento seguro dos documentos com a capacidade de acessá-los e validá-los a partir de diferentes componentes do sistema. Ela opera como elo entre as informações sumarizadas (hashes, CIDs e DIDs) armazenadas imutavelmente no *ledger*, e os documentos reais, mantidos de forma descentralizada no IPFS.

O fluxo de integração, ilustrado na Figura 3.3, opera da seguinte forma: ao receber um documento PDF, a API Node.js realiza simultaneamente o upload para o IPFS, obtendo o CID, e o cálculo do hash SHA-256 do arquivo. Esses dois identificadores, juntamente com os DIDs gerados na camada de identidade, são então transmitidos ao *chaincode* por meio do SDK oficial do Hyperledger Fabric para Node.js (`@hyperledger/fabric-gateway`).

A comunicação entre a API e o Hyperledger Fabric ocorre via gRPC com autenticação mútua por certificados X.509, garantindo a confidencialidade e a integridade do canal de comunicação. A API é implementada com Node.js e Express.js, expondo os *endpoints* REST detalhados na Tabela 3.4 (Seção 3.3.6), e adota o padrão de tratamento centralizado de erros por

Figura 3.3 – Fluxo Integrado entre IPFS e *Blockchain*.



Fonte: Elaborado pelo autor (2026).

meio de *middleware* dedicado, que mapeia erros semânticos do *chaincode* para os códigos HTTP correspondentes.

3.1.9 Testes e Validação do Sistema

A fase de testes e validação constitui etapa crítica para assegurar a corretude funcional, o desempenho e a robustez do sistema proposto. Os testes são conduzidos de forma iterativa e paralela ao desenvolvimento, combinando testes unitários do *chaincode*, testes de integração da API e simulações de carga.

Os cenários de teste contemplam, ao menos, os seguintes casos:

a) **Emissão e verificação de documento íntegro:** Emissão de um documento, recuperação via *endpoint* de verificação e confirmação de status `ATIVO` e `integrityOk: true`;

b) **Detecção de adulteração:** Substituição do arquivo no IPFS por uma versão modificada sem alteração do registro no *ledger* e confirmação de que a verificação retorna `integrityOk: false`;

c) **Revogação e verificação posterior:** Revogação de um documento com motivo registrado e confirmação de que consultas subsequentes retornam status `REVOGADO`;

d) **Tentativa de revogação não autorizada:** Invocação de `RevogarDocumento` por entidade diferente do emissor original e confirmação do erro `ErrNaoAutorizado` com código HTTP 403;

e) **Verificação de Credencial Verificável (VC):** Resolução do `did:web` da instituição, validação da assinatura Ed25519 da VC e confirmação da correspondência entre o `TitularDID` e os dados registrados no *ledger*;

f) **Geração e verificação de prova ZKP:** Solicitação de prova ZK-SNARK para um documento registrado, transmissão da prova para o *endpoint* de verificação e confirmação do resultado `valido: true`.

Para a coleta de métricas de desempenho, são executados *benchmarks* em Node.js que medem a latência média das operações de emissão e verificação (em milissegundos), o tempo de geração de prova ZKP e o tempo de upload ao IPFS em função do tamanho do arquivo (100 KB, 500 KB, 1 MB, 5 MB e 10 MB). Os resultados são registrados em formato CSV para posterior análise e apresentação no Capítulo 4.

3.1.10 Implantação do Protótipo e Análise Preliminar dos Resultados

A implantação do protótipo representa a etapa de convergência de todas as fases anteriores em um ambiente integrado e reproduzível. A infraestrutura de *blockchain* e armazenamento é configurada por meio de um arquivo Docker Compose unificado (dez serviços, detalhados na Seção 3.3.1.3): os nós *peer* e *orderer* do Hyperledger Fabric, as Autoridades Certificadoras, o banco de dados CouchDB para o estado do *ledger* e o nó IPFS (Kubo). A API Node.js e o *frontend* React, também implementados no protótipo, são executados como processos Node.js separados fora do Docker Compose, conectando-se à infraestrutura containerizada. A inicialização completa do ambiente é realizada por meio de um único comando:

Código 3.4 – Inicialização do ambiente integrado de desenvolvimento

```
docker-compose up -d
./scripts/init-network.sh   # sobe canal, instala e aprova chaincode
```

Essa configuração assegura a reprodutibilidade do experimento, permitindo que o protótipo seja inicializado sem configurações manuais adicionais a partir do código-fonte do projeto, em conformidade com boas práticas de reprodutibilidade experimental. O repositório do projeto é atualmente privado; sua publicização está prevista para uma etapa posterior a este trabalho.

A análise preliminar dos resultados, a ser detalhada no próximo capítulo, abordará: (*i*) as métricas de latência e *throughput* coletadas nos *benchmarks*; (*ii*) a corretude funcional atestada pelos testes automatizados; (*iii*) a viabilidade da integração entre as camadas de identidade (DIDs W3C), privacidade (ZK-SNARKs) e consenso (Hyperledger Fabric + IPFS); e (*iv*) as limitações identificadas durante a implantação, que fundamentarão os trabalhos futuros.

3.2 Considerações sobre as Decisões de Projeto

Algumas decisões de projeto merecem registro explícito, uma vez que implicam afastamento em relação à proposta original do TCC1, com justificativa técnica e acadêmica fundamentada.

Substituição do Hyperledger Indy pelo padrão W3C DID: O *Hyperledger Indy*, solução originalmente considerada para a gestão de identidade descentralizada, apresenta curva de aprendizado elevada, fator que representaria risco significativo para o prazo disponível. A

substituição pelo padrão W3C DID, com os métodos `did:web` e `did:key`, é academicamente mais robusta: o padrão é uma Recomendação oficial do W3C desde julho de 2022, conta com múltiplas implementações maduras em Node.js e é amplamente adotado em projetos de identidade digital de grande escala (SPORNY et al., 2022b). Ademais, a combinação com Credenciais Verificáveis (VCs) do W3C confere ao sistema propriedades de interoperabilidade superiores às de uma solução proprietária.

Custódia da chave do titular como limitação em aberto: A Seção 3.1.6 descreve o `did:key` como um identificador controlado pelo próprio sujeito, propriedade central da noção de identidade auto-soberana. Na versão atual do protótipo, porém, o par de chaves Ed25519 do titular é gerado de forma efêmera pelo próprio servidor da API no momento da emissão (passo 3 do fluxo descrito na Seção 3.3.6.2), e não pelo titular em um dispositivo ou carteira sob seu controle. Isso significa que, na prática, o titular não controla nem retém a chave privada associada ao seu próprio `did:key`, uma tensão análoga à discutida para a centralização institucional na Seção 5.1. A custódia da chave pelo titular, tipicamente via uma carteira digital (*wallet*) executada em seu próprio dispositivo, é reconhecida como evolução necessária para que a identidade do titular seja de fato auto-soberana, e é retomada na Seção 5.2.

ZKP como prova de conceito deliberada: A implementação completa de ZK-SNARKs para todos os tipos de documentos e para a totalidade dos atributos da credencial exigiria circuitos de maior complexidade, cujo desenvolvimento ultrapassa o escopo deste trabalho. O circuito implementado, que prova a posse de um valor igual a um hash já público no *ledger* (Seção 3.1.5), é uma prova de conceito deliberada do *toolchain* ZoKrates, não uma garantia de privacidade documental efetiva: demonstra a viabilidade técnica do fluxo de compilação, *trusted setup*, geração e verificação de provas no contexto proposto, e constitui base para a incorporação futura do cálculo do hash dentro do próprio circuito (Seção 5.2), etapa necessária para que a prova de fato vincule o documento original sem expor seu hash publicamente.

IPFS local em vez de Filecoin ou serviços de *pinning*: O uso do *Kubo* local via Docker é adequado para o protótipo e para a coleta de métricas de desempenho. Estratégias de persistência a longo prazo como Filecoin ou serviços comerciais de *pinning* como o Pinata são analisadas nas considerações finais como extensão necessária para um cenário de produção.

3.3 Experimentação Prática

Esta seção descreve a implementação e os experimentos executados com o protótipo *DoChain*, sistema de autenticação de documentos legais desenvolvido no contexto desta pesquisa. As bases teóricas dos componentes utilizados (Hyperledger Fabric, IPFS, DIDs W3C, Credenciais Verificáveis e ZK-SNARKs) foram apresentadas no Capítulo 2; a metodologia de projeto e as decisões arquiteturais foram descritas na Seção 3.1. A presente seção foca nos aspectos práticos: configuração do ambiente, código implementado em cada camada, resultados da validação

funcional e de segurança, e métricas de desempenho.

A experimentação abrange desde a inicialização da rede Hyperledger Fabric até a execução de provas ZKP via ZoKrates, passando pelo armazenamento descentralizado em IPFS e pela emissão de Credenciais Verificáveis (VC) conforme o padrão W3C. Os resultados obtidos são discutidos ao final do capítulo e retomados nas considerações finais.

3.3.1 Ambiente de Experimentação

O protótipo foi implantado integralmente em um ambiente local, contido em contêineres Docker, sem dependência de infraestrutura de nuvem. Essa escolha assegura reprodutibilidade integral dos experimentos: a infraestrutura de *blockchain* e armazenamento distribuído (peers, orderer, CAs, CouchDB e IPFS) é orquestrada por um único arquivo `docker-compose.yaml`; a API e o *frontend* executam como processos Node.js externos a esse arquivo (Seção 3.1.10), e o ZoKrates é invocado sob demanda em contêiner efêmero (Seção 3.1.5).

3.3.1.1 Configuração de Hardware e Software

A Tabela 3.2 apresenta as especificações do ambiente de experimentação. O ambiente executa sobre o *Windows Subsystem for Linux 2* (WSL2), que provê um núcleo Linux nativo sobre Windows, permitindo a execução dos contêineres Hyperledger Fabric sem modificações.

Tabela 3.2 – Especificações do ambiente de experimentação.

Componente	**Especificação**
Sistema Operacional	WSL2, Linux 6.6.87 (kernel Microsoft) sobre Windows 11
Docker Engine	24.x (plugin Docker Compose 2.x)
Hyperledger Fabric	2.5.0
Fabric CA	1.5.7
Go	1.21+ (compilação do chaincode)
Node.js	20 LTS (API Express)
IPFS (Kubo)	Imagem `ipfs/kubo:latest`
CouchDB	3.3.3 (banco de estado dos peers)
ZoKrates	Imagem `zokrates/zokrates` (via Docker)
Rede Docker	`fabric_net` (bridge isolada)

Fonte: Elaborado pelo autor (2026).

3.3.1.2 Topologia da Rede *Blockchain*

Conforme o modelo arquitetural definido na Seção 2.5, a rede foi configurada com duas organizações participantes (`Org1MSP` e `Org2MSP`), cada qual operando um único *endorsing peer*, e um serviço de ordenação baseado no protocolo Raft (LINUX FOUNDATION, 2021), implantado nesta topologia com um único nó `orderer.example.com`. Trata-se de uma configuração de

desenvolvimento: com um único ordenador não há eleição de líder nem replicação entre múltiplos nós, de modo que a rede não possui tolerância a falhas por parada no serviço de ordenação. As propriedades de replicação e de decisão por maioria descritas na Seção 2.5 pressupõem três ou mais nós ordenadores e não são exercidas neste ambiente experimental. A topologia foi definida nos arquivos `configtx.yaml` e `crypto-config.yaml`, e os contêineres foram declarados em `docker-compose.yaml`.

O canal `documentos-channel` conecta todos os peers e o orderer, constituindo o espaço compartilhado de ledger sobre o qual o chaincode `documentos:1.0` é executado. A adoção do CouchDB como banco de estado, em detrimento do LevelDB padrão do Fabric, foi motivada pela necessidade de executar consultas seletivas (*Mango queries*) no chaincode, conforme detalhado na Seção 3.3.2.2.4. Cada peer possui sua instância CouchDB dedicada: `couchdb0` para `peer0.org1.example.com` (porta 5984) e `couchdb1` para `peer0.org2.example.com` (porta 7984).

A Figura 3.4 ilustra a topologia da rede implantada.

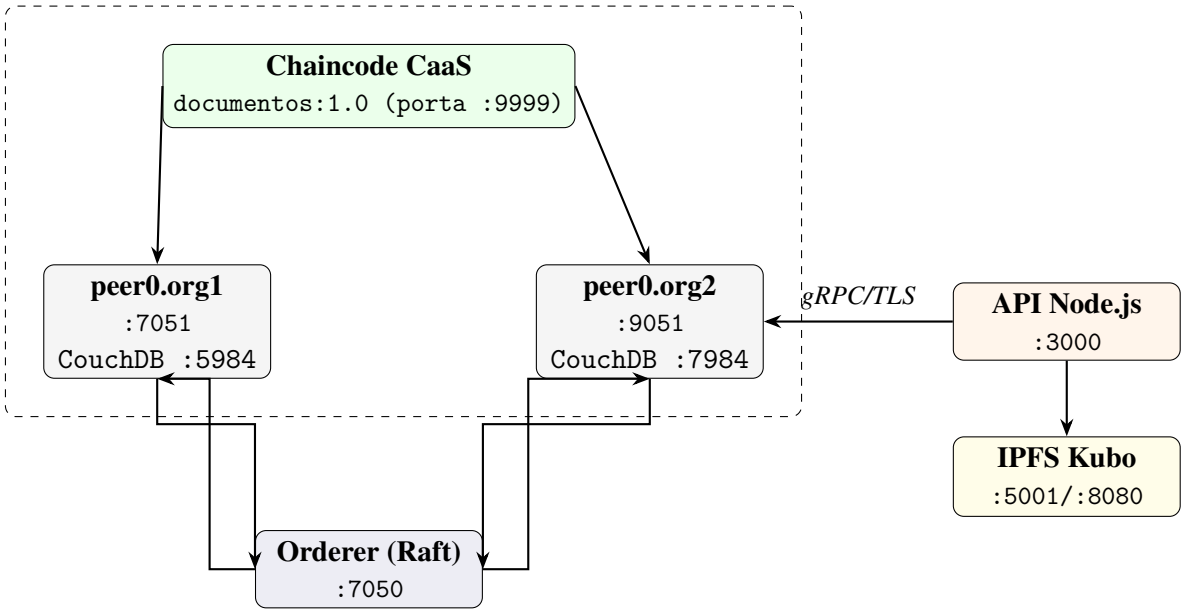


Figura 3.4 – Topologia da rede Hyperledger Fabric implantada.

Fonte: Elaborado pelo autor (2026).

A política de endosso definida em `configtx.yaml` exige a aprovação de ambas as organizações para que qualquer transação seja incluída no bloco: `AND('Org1MSP.peer', 'Org2MSP.peer')`. Isso significa que nenhuma organização isolada pode modificar o ledger unilateralmente, propriedade central para o modelo de confiança proposto (ANDROULAKI et al., 2018).

O orderer opera no modo *etcdraft* (Raft) sem o canal de sistema (*system channel*), característica introduzida no Fabric 2.3 e mantida na versão 2.5.0 utilizada no protótipo (Tabela 3.2).

O canal é criado diretamente pela *Channel Participation API*, acessada via `osnadmin channel join`. O `BatchTimeout` foi configurado em 2 s e o `MaxMessageCount` em 10 transações por bloco, parâmetros adequados para a carga do ambiente de desenvolvimento.

3.3.1.3 Serviços de Suporte

O arquivo `docker-compose.yaml` declara dez serviços, todos conectados à rede Docker `fabric_net`: dois peers, dois bancos CouchDB, duas Autoridades Certificadoras (CA), um orderer Raft, um nó IPFS Kubo, um container CaaS do chaincode e um container CLI para operações administrativas.

O nó IPFS Kubo expõe a porta 5001 para a API RPC (utilizada pela API Node.js para operações de adição e leitura de conteúdo) e a porta 8080 para o gateway HTTP público. Dois volumes Docker persistentes foram configurados: `ipfs_data` para o repositório de blocos do IPFS e `ipfs_staging` para a área de trabalho temporária. O parâmetro `IPFS_PROFILE=server` desabilita o anúncio de endereços locais, adequado para o contexto de contêiner.

O chaincode é executado no modo *Chaincode as a Service* (CaaS), em um contêiner Docker separado (`chaincode-documentos:9999`), comunicando-se com os peers via gRPC interno sem TLS. Esse modo evita o problema de *Docker-in-Docker* que ocorre em ambientes WSL2 e é a abordagem recomendada pelo Hyperledger para implantações de produção.

3.3.2 Implementação do Chaincode

O chaincode é o componente que encapsula a lógica de negócio executada diretamente na *blockchain*. Conforme a arquitetura definida na Seção 2.5, o chaincode foi implementado em Go 1.21 utilizando a biblioteca `contractapi` do Hyperledger Fabric, que abstrai o ciclo de vida da transação e provê tipagem estática para os parâmetros das funções do contrato (ANDROULAKI et al., 2018).

3.3.2.1 Modelo de Dados

A unidade de dado central do sistema é a *struct* `Documento`, definida em `chaincode/pkg/types.go`. Cada instância representa um documento legal emitido e é serializada como JSON para ser armazenada no ledger via `PutState`. O Código 3.5 apresenta a definição completa.

Código 3.5 – Struct `Documento`: unidade de dado armazenada no ledger

```
// Documento e a struct principal armazenada no ledger.
type Documento struct {
    ID              string             `json:"id"`
    Tipo            TipoDocumento      `json:"tipo"`
    Sensibilidade   NivelSensibilidade `json:"sensibilidade"`
    EmissorDID      string             `json:"emissorDID"`
```

```
    TitularDID      string              `json:"titularDID"`
    EmissorMSP      string              `json:"emissorMSP"`
    CID             string              `json:"cid"`
    HashDocumento   string              `json:"hashDocumento"`
    ProvaVC         string              `json:"provaVC"`
    Metadados       string              `json:"metadados"` // JSON
        livre por tipo
    Status          StatusDocumento     `json:"status"`
    DataEmissao     string              `json:"dataEmissao"`
    DataRevogacao   string              `json:"dataRevogacao,omitempty
        "`
    MotivoRevogacao string              `json:"motivoRevogacao,
        omitempty"`
}
```

O campo `Metadados` é intencionalmente um `string` que armazena JSON livre, permitindo que cada tipo documental carregue atributos específicos sem alteração da estrutura principal do chaincode. Um diploma pode registrar `{"curso":"Ciência da Computação", "grau":"Bacharelado"}`, enquanto um atestado médico armazena `{"crm":"12345/MG", "diasAfastamento":3}`. Essa decisão de design admite extensibilidade sem requerer atualização do chaincode para cada novo tipo de metadado.

O campo `EmissorMSP` registra a identidade X.509 completa do cliente Fabric que submeteu a transação de emissão, obtida via `ctx.GetClientIdentity().GetID()`. Essa identidade é persistida no ledger e comparada nas operações de revogação para garantir que apenas o emissor original possa revogar o próprio documento.

A Tabela 3.3 apresenta os cinco tipos de documento suportados, seus níveis de sensibilidade e os emissores esperados. O nível de sensibilidade é derivado automaticamente pelo chaincode na função `SensibilidadePorTipo`, definida em `chaincode/pkg/sensibilidade.go`.

Tabela 3.3 – Tipos documentais suportados e seus níveis de sensibilidade.

TipoDocumento	**Sensibilidade**	**Emissor Esperado**	**Regulação**
`DIPLOMA`	BAIXA	Instituição de Ensino Superior	Lei 9.394/1996
`CNH`	BAIXA	DETRAN / SENATRAN	Lei 9.503/1997
`ESCRITURA`	MÉDIA	Cartório (Tabelionato)	Lei 8.935/1994
`CERTIDAO`	MÉDIA	Cartório de Registro Civil	Lei 6.015/1973
`ATESTADO_MEDICO`	ALTA	Médico credenciado no CRM	CFM Res. 2.299/2021

Fonte: Elaborado pelo autor (2026).

3.3.2.2 Funções do Chaincode

O chaincode expõe cinco funções públicas, implementadas em `chaincode/chaincode.go`:

3.3.2.2.1 EmitirDocumento

Registra um novo documento no ledger após validação dos campos obrigatórios, verificação do tipo e checagem de ID duplicado. O Código 3.6 ilustra a lógica principal.

Código 3.6 – Função `EmitirDocumento`: trecho principal de validação e persistência (listagem já refletindo a correção de determinismo descrita na Seção 3.3.9.1)

```
func (s *SmartContract) EmitirDocumento(
    ctx contractapi.TransactionContextInterface,
    id, tipo, emissorDID, titularDID, cid, hashDoc, provaVC,
       metadados string,
) error {
    if id == "" || tipo == "" || emissorDID == "" ||
       titularDID == "" || cid == "" || hashDoc == "" {
        return pkg.ErrCamposObrigatorios
    }
    if _, ok := tiposValidos[pkg.TipoDocumento(tipo)]; !ok {
        return fmt.Errorf("tipo de documento invalido: %q", tipo)
    }
    existente, err := ctx.GetStub().GetState(id)
    if err != nil { return fmt.Errorf("falha ao consultar ledger: %w"
       , err) }
    if existente != nil { return pkg.ErrDocumentoJaExiste }

    identidade, _ := ctx.GetClientIdentity().GetID()

    // Usa o timestamp assinado da proposta, identico em todos os
       peers
    // endossantes, nunca time.Now(), que diverge entre peers e
       quebra o endosso.
    txTimestamp, _ := ctx.GetStub().GetTxTimestamp()
    dataEmissao := time.Unix(txTimestamp.Seconds, int64(txTimestamp.
       Nanos)).UTC().Format(time.RFC3339)

    tipoDoc := pkg.TipoDocumento(tipo)
    doc := pkg.Documento{
        ID:             id,
        Tipo:           tipoDoc,
        Sensibilidade: pkg.SensibilidadePorTipo(tipoDoc),
        EmissorDID:    emissorDID,   TitularDID: titularDID,
        EmissorMSP:    identidade,   CID:        cid,
```

```
        HashDocumento: hashDoc,       ProvaVC:    provaVC,
        Metadados:     metadados,     Status:     pkg.StatusAtivo,
        DataEmissao:   dataEmissao,
    }
    docJSON, _ := json.Marshal(doc)
    return ctx.GetStub().PutState(id, docJSON)
}
```

3.3.2.2.2 VerificarDocumento

Recupera o JSON do ledger via `GetState(id)`, desserializa e retorna o objeto `Documento` completo. Retorna `ErrDocumentoNaoEncontrado` se o ID não existir ou erro de deserialização se o JSON estiver corrompido.

3.3.2.2.3 RevogarDocumento

Verifica que o chamador é o mesmo MSP que emitiu o documento comparando `doc.EmissorMSP != identidade`. Recusa revogação de documentos já revogados via `ErrDocumentoRevogado`, e rejeita motivo vazio via `ErrCamposObrigatorios`. Ao revogar, atualiza o campo `Status` para `REVOGADO`, grava `DataRevogacao` e `MotivoRevogacao`, e persiste via `PutState`.

3.3.2.2.4 ListarPorTipo

Executa uma consulta Mango diretamente no CouchDB:

Código 3.7 – Query CouchDB Mango na função `ListarPorTipo`

```
query := fmt.Sprintf(`{"selector":{"tipo":"%s","status":"ATIVO"}}`,
    tipo)
it, err := ctx.GetStub().GetQueryResult(query)
```

Essa consulta é viabilizada pelo CouchDB. O LevelDB padrão do Fabric não suporta consultas por campo; admite apenas recuperação por chave exata, o que tornaria a listagem por tipo inviável sem varredura completa do ledger.

3.3.2.2.5 HistoricoDocumento

Utiliza `GetHistoryForKey(id)` para retornar o histórico completo de transações associadas ao documento. Cada entrada contém o identificador da transação (`txId`), o *timestamp* do bloco, o indicador de deleção (`isDelete`) e o valor serializado do estado naquele momento. O histórico é imutável: está gravado nos blocos da cadeia, não apenas no banco de estado.

3.3.2.3 Políticas de Controle de Acesso

O controle de acesso do chaincode opera em duas camadas complementares. A primeira é estrutural, imposta pelo *Membership Service Provider* (MSP): apenas clientes com certificados X.509 emitidos pelas CAs das organizações participantes podem submeter transações ao canal. A segunda é lógica, verificada dentro do próprio chaincode em tempo de execução.

Na função `RevogarDocumento`, o chaincode compara a identidade do chamador atual com a identidade armazenada no campo `EmissorMSP` no momento da emissão:

Código 3.8 – Verificação de identidade na revogação

```
identidade, err := ctx.GetClientIdentity().GetID()
if err != nil {
    return fmt.Errorf("falha ao obter identidade: %w", err)
}
if doc.EmissorMSP != identidade {
    return pkg.ErrNaoAutorizado
}
```

Esse mecanismo impede que uma organização revogue documentos emitidos por outra. Por exemplo, uma universidade (`Org1MSP`) não pode revogar uma escritura emitida por um cartório (`Org2MSP`), ainda que ambos sejam membros legítimos da rede.

3.3.2.4 Deploy e Ciclo de Vida do Chaincode

O deploy seguiu o ciclo de vida do Fabric 2.x, composto por dez etapas automatizadas no script `scripts/deploy-chaincode.sh`:

1. Validação da compilação Go (`go build ./...`);
2. Geração do pacote CaaS com `connection.json` apontando para `chaincode-documentos:9999`;
3. Verificação e reinicialização do contêiner CaaS com o Package ID correto;
4. Cópia do pacote para o contêiner CLI;
5. Instalação em `peer0.org1` e `peer0.org2` (`peer lifecycle chaincode install`);
6. Confirmação do Package ID instalado via `queryinstalled`;
7. Aprovação pela `Org1MSP` (`peer lifecycle chaincode approveformyorg`);
8. Aprovação pela `Org2MSP` com variáveis de ambiente adequadas;

9. Verificação de prontidão (`checkcommitreadiness`): ambas as organizações devem constar como `true`;

10. Commit da definição no canal (`peer lifecycle chaincode commit`).

O conceito de endosso exige que as propostas de transação sejam assinadas pelos peers das organizações listadas na política antes de serem enviadas ao orderer. Com a política `AND('Org1MSP.peer','Org2MSP.peer')`, nenhuma transação pode ser incluída no ledger sem a anuência explícita de ambas as partes, garantindo o modelo de confiança distribuída (ANDROULAKI et al., 2018).

3.3.3 Implementação da Camada de Armazenamento

O armazenamento dos arquivos PDF diretamente no ledger do Fabric seria inviável: cada peer replicaria todos os dados, e arquivos de vários megabytes inchariam o banco de estado, tornando a sincronização de novos peers proibitivamente lenta. Conforme discutido na Seção 2.6.2, o IPFS resolve essa limitação: apenas o CID e o hash SHA-256 do arquivo são registrados no ledger (BENET, 2014).

3.3.3.1 Integração com o IPFS

O módulo `api/src/ipfs/ipfsService.js` encapsula toda a comunicação com o nó Kubo via a biblioteca `kubo-rpc-client`. O Código 3.9 apresenta a função de armazenamento.

Código 3.9 – Função `armazenarDocumento`: upload para o IPFS e cálculo do hash

```
const ipfs = create({ url: process.env.IPFS_URL ?? 'http://localhost
    :5001' });

export async function armazenarDocumento(buffer) {
  const resultado = await ipfs.add(buffer, { pin: true });
  const hash = createHash('sha256').update(buffer).digest('hex');
  return { cid: resultado.cid.toString(), hash };
}
```

O parâmetro `pin: true` instrui o nó IPFS a manter o conteúdo no repositório local, impedindo que o coletor de lixo do IPFS remova o arquivo. O hash SHA-256 é calculado diretamente sobre o `buffer` na memória, antes de qualquer transmissão, garantindo que o hash armazenado no ledger corresponde ao arquivo original, não a uma representação codificada.

Formalizando a propriedade de determinismo do CID já justificada como critério de escolha do IPFS (Seção 3.1.4), e considerando que o IPFS fragmenta arquivos maiores que o tamanho de bloco em um *Merkle DAG* UnixFS (Seção 2.6.1), o CID é derivado como:

$$\text{CID} = \begin{cases} \text{multibase}(\text{multihash}(\text{SHA-256}(\text{conteúdo}))), & \text{arquivo cabe em um único bloco} \\ \text{multibase}(\text{multihash}(\text{raiz do } \textit{Merkle DAG})), & \text{arquivo fragmentado em múltiplos blocos} \end{cases} \tag{3.1}$$

onde *multibase* codifica o resultado em Base58BTC e *multihash* prefixa o hash com o identificador do algoritmo. Para os arquivos de teste utilizados nos experimentos deste trabalho (até 10 MB, Seção 4), o segundo caso se aplica: o CID corresponde à raiz do *Merkle DAG* construído sobre os blocos, e não ao SHA-256 direto do conteúdo integral. Em ambos os casos, a propriedade relevante para a verificação de integridade é preservada: qualquer alteração no conteúdo, mesmo em um único bloco, altera o CID resultante.

3.3.3.2 Mecanismo de Verificação de Integridade

O mecanismo de verificação em dupla camada, já descrito como decisão de projeto na Seção 3.1.4, é implementado pela função `verificarIntegridade`:

Código 3.10 – Função `verificarIntegridade`: verificação em dupla camada

```
export async function verificarIntegridade(cid, hashEsperado) {
  const chunks = [];
  for await (const chunk of ipfs.cat(cid)) {
    chunks.push(chunk);
  }
  const buffer = Buffer.concat(chunks);
  const hash = createHash('sha256').update(buffer).digest('hex');
  return hash === hashEsperado;
}
```

A propriedade de integridade pode ser enunciada formalmente como:

$$\text{verificacaoIntegridade} = \left[\, \text{SHA-256}(\text{arquivo}_{\text{IPFS}}(cid)) = \text{hashLedger} \,\right] \tag{3.2}$$

Se o arquivo for adulterado após o armazenamento, o SHA-256 recalculado diferirá do hash gravado no ledger e a verificação retornará `false`. Como o CID é determinístico, é também impossível substituir o conteúdo no IPFS sem alterar o CID; além disso, o CID registrado no ledger é imutável.

3.3.4 Implementação da Identificação Descentralizada

Os Identificadores Descentralizados (DID) e as Credenciais Verificáveis (VC) foram adotados conforme os padrões W3C apresentados na Seção 3.1.6. O sistema utiliza dois métodos

distintos: `did:key` para titulares de documentos (pessoas físicas) e `did:web` para instituições emissoras (SPORNY et al., 2022b).

3.3.4.1 Geração de `did:key` para Titulares

O método `did:key` gera um identificador descentralizado diretamente a partir de uma chave pública, sem necessitar de registro em nenhum servidor ou *blockchain*. O algoritmo Ed25519 foi escolhido por sua eficiência e segurança comprovadas para assinaturas digitais (BERNSTEIN et al., 2012).

Código 3.11 – Função `gerarDidKey`: geração de DID para titular

```
export async function gerarDidKey() {
  const keyPair = await Ed25519VerificationKey2020.generate();
  const { didDocument } = await didKeyDriver.fromKeyPair({
    verificationKeyPair: keyPair,
  });
  const did = didDocument.id;
  keyPair.controller = did;
  keyPair.id = `${did}#${keyPair.fingerprint()}`;
  return { did, keyPair };
}
```

O processo gera um par de chaves Ed25519, deriva a *fingerprint* multibase da chave pública (prefixo `z6Mk` para `Ed25519VerificationKey2020`) e constrói o DID na forma `did:key:z6Mk<fingerprint>`. Exemplo de DID gerado: `did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLGpbnnEGta2doK`. O DID é completamente auto-descritivo: a chave pública está codificada no próprio identificador, dispensando consulta a qualquer servidor para resolução.

3.3.4.2 Resolução de `did:web` para Instituições

Conforme o mecanismo de resolução já descrito na Seção 3.1.6 (`did:web:ufop.br` → `GET https://ufop.br/.well-known/did.json`), esta seção detalha sua implementação. Em ambiente de desenvolvimento, domínios `localhost` são resolvidos a partir de arquivos locais em `api/mock-dids/`. O Código 3.12 mostra a lógica de resolução implementada.

Código 3.12 – Função `resolverDidWeb`: resolução local e via HTTPS

```
export async function resolverDidWeb(dominio) {
  if (dominio.startsWith('localhost')) {
    const caminho = resolve(MOCK_DIR, `${dominio}.json`);
    return JSON.parse(readFileSync(caminho, 'utf8'));
  }
  const url  = `https://${dominio}/.well-known/did.json`;
  const resp = await fetch(url);
  if (!resp.ok) {
```

```
    throw new Error(
      `Falha ao resolver did:web para ${dominio}: HTTP ${resp.status}`
    );
  }
  return resp.json();
}
```

O DID Document do emissor de desenvolvimento (`did:web:localhost:3001`) contém um método de verificação do tipo `Multikey` com chave Ed25519, conforme o Código 3.13.

Código 3.13 – DID Document do emissor de desenvolvimento (`localhost:3001`)

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/multikey/v1"
  ],
  "id": "did:web:localhost:3001",
  "verificationMethod": [{
    "id": "did:web:localhost:3001#
       z6Mktcj5M1nFUvNnxKbYQaEbSJ9HAyen7CgVEsf33stcxgZK",
    "type": "Multikey",
    "controller": "did:web:localhost:3001",
    "publicKeyMultibase": "
       z6Mktcj5M1nFUvNnxKbYQaEbSJ9HAyen7CgVEsf33stcxgZK"
  }],
  "authentication": [
    "did:web:localhost:3001#
       z6Mktcj5M1nFUvNnxKbYQaEbSJ9HAyen7CgVEsf33stcxgZK"
  ],
  "assertionMethod": [
    "did:web:localhost:3001#
       z6Mktcj5M1nFUvNnxKbYQaEbSJ9HAyen7CgVEsf33stcxgZK"
  ],
  "service": [{
    "id": "did:web:localhost:3001#documentos",
    "type": "DocumentoService",
    "serviceEndpoint": "http://localhost:3001/api/documentos"
  }]
}
```

3.3.4.3 Emissão e Verificação de Credenciais Verificáveis

A VC é um envelope JSON assinado que encapsula as informações do documento conforme o padrão W3C Verifiable Credentials 1.1 (SPORNY et al., 2022a). A função `assinarVC`

utiliza a biblioteca `@digitalbazaar/vc` com a suíte de assinatura `Ed25519Signature2020`:

Código 3.14 – Função `assinarVC`: emissão de Credencial Verificável

```
export async function assinarVC(credentialSubject, emissorDID,
                                keyPair, opcoes = {}) {
  const { contextos = [], tipos = [] } = opcoes;
  const credencial = {
    '@context': [CONTEXT_VC_V1, CONTEXT_ED25519,
                 CONTEXT_MULTIKEY, ...contextos],
    id: `urn:uuid:${randomUUID()}`,
    type: ['VerifiableCredential', ...tipos],
    issuer: emissorDID,
    issuanceDate: new Date().toISOString(),
    credentialSubject,
  };
  const suite = new Ed25519Signature2020({ key: keyPair });
  const documentLoader = criarDocumentLoader();
  return vc.issue({ credential: credencial, suite, documentLoader });
}
```

A VC emitida contém os campos `@context` (especificação dos vocabulários W3C), `type` (`VerifiableCredential`), `issuer` (DID do emissor), `issuanceDate` (carimbo de tempo RFC3339), `credentialSubject` (dados do documento: `id` do titular, `tipo`, `cid` e `hash`) e `proof` (assinatura Ed25519 e referência ao método de verificação). A seção `proof` permite a verificação de não-repúdio: qualquer terceiro pode obter a chave pública do emissor via DID Document e confirmar que a assinatura é válida, sem necessitar contactar a instituição emissora (SPORNY et al., 2022a).

A verificação da VC (`verificarVC`) resolve o DID Document do emissor, extrai o método de verificação, reconstrói a suíte Ed25519 e invoca `vc.verifyCredential`. Qualquer adulteração no payload da credencial invalida a assinatura criptográfica.

3.3.5 Implementação das Provas de Conhecimento Zero

As provas de conhecimento zero (ZK-SNARKs) implementadas no protótipo demonstram, como prova de conceito, que um titular pode comprovar a posse de um valor privado igual a um valor já público sem revelar esse valor diretamente na prova, conforme a ressalva sobre o alcance do circuito discutida na Seção 3.1.5. Conforme os fundamentos apresentados naquela seção, o sistema utiliza ZoKrates (EBERHARDT; TAI, 2018), uma DSL e *toolchain* para ZK-SNARKs que implementa o esquema de prova Groth16 (GROTH, 2016) sobre a curva BN128.

3.3.5.1 Definição do Circuito Aritmético

O circuito implementado, armazenado em `zkp/diploma_valid.zok`, prova que o provador conhece os dois *field elements* que compõem o hash SHA-256 do documento:

Código 3.15 – Circuito ZoKrates `diploma_valid.zok`: prova de conhecimento do hash

```
// Prova que o provador conhece o hash de um documento
// sem revelar o hash completo.
//
// O SHA-256 (256 bits) e representado como dois field elements
// de 128 bits cada (hashPartes[0] = bits 255..128,
//                   hashPartes[1] = bits 127..0).
def main(private field[2] hashPrivado, field[2] hashPublico) {
    assert(hashPrivado[0] == hashPublico[0]);
    assert(hashPrivado[1] == hashPublico[1]);
    return;
}
```

O hash SHA-256 de 256 bits não pode ser representado diretamente como um único *field element* do BN128, pois o módulo primo do campo ($\approx 2^{254}$) é menor que 2^{256}. Por isso, o hash é dividido em dois *field elements* de 128 bits cada: `hashPartes[0]` corresponde aos bits 255 a 128 (primeiros 32 caracteres hexadecimais), e `hashPartes[1]` aos bits 127 a 0 (últimos 32 caracteres). O circuito afirma que os elementos privados fornecidos pelo provador são iguais aos elementos públicos comprometidos, sem revelar quais são esses valores ao verificador (GROTH, 2016).

3.3.5.2 Configuração Confiável (*Trusted Setup*)

O *trusted setup* Groth16 é executado uma única vez pelo script `scripts/zkp-setup.sh`, em dois passos via contêiner Docker:

Código 3.16 – Script `zkp-setup.sh`: compilação e *trusted setup*

```
# 1. Compilacao do circuito
docker run --rm -v "${ZKP_DIR}:/zkp" zokrates/zokrates \
  zokrates compile -i /zkp/diploma_valid.zok -o /zkp/out

# 2. Trusted setup Groth16
docker run --rm -v "${ZKP_DIR}:/zkp" zokrates/zokrates \
  zokrates setup \
    -i /zkp/out \
    -p /zkp/proving.key \
    -v /zkp/verification.key
```

O primeiro passo compila o circuito ZoKrates para uma representação aritmética de

baixo nível (*Rank-1 Constraint System*, R1CS) armazenada em `zkp/out`. O segundo passo gera a *Common Reference String* (CRS) do esquema Groth16, produzindo dois arquivos: `proving.key`, utilizada pelo servidor para gerar provas, e `verification.key`, utilizada por qualquer verificador para conferir provas (GROTH, 2016). A `verification.key` pode ser tornada pública; a `proving.key` deve ser protegida, pois o conhecimento do *toxic waste* gerado no setup permitiria forjar provas válidas para entradas falsas.

Em um sistema de produção, o *trusted setup* deveria ser realizado em cerimônia multipartidária (*Multi-Party Computation*, MPC), garantindo que nenhuma das partes individualmente possua o *toxic waste* completo (BOWE; GABIZON; GREEN, 2017).

3.3.5.3 Geração e Verificação de Provas

A geração de provas é implementada em `api/src/zkp/zkpService.js`. O hash SHA-256 hexadecimal é convertido em dois *field elements* decimais antes de ser passado ao ZoKrates:

Código 3.17 – Conversão do hash em *field elements* para o ZoKrates

```
function hashParaFieldElements(hashHex) {
  if (typeof hashHex !== 'string' || hashHex.length !== 64) {
    throw new Error(`hash invalido: esperado 64 chars hex`);
  }
  const parte0 = BigInt('0x' + hashHex.slice(0, 32)).toString(10);
  const parte1 = BigInt('0x' + hashHex.slice(32, 64)).toString(10);
  return [parte0, parte1];
}
```

A função `gerarProva` executa dois comandos ZoKrates em sequência via Docker: `compute-witness` (calcula o testemunho aritmético para as entradas fornecidas) e `generate-proof` (gera a prova π Groth16 como objeto JSON com os pontos `a`, `b` e `c` na curva BN128). Um diretório temporário de sessão (UUID aleatório) é criado para isolar execuções concorrentes, sendo removido ao final independentemente de erros. A função `verificarProva` executa `zokrates verify` com a `verification.key` e retorna `true` apenas se a saída contiver a string `PASSED`.

3.3.6 Implementação da API REST

A API Node.js atua como orquestradora dos quatro componentes: Fabric, IPFS, DIDs e ZKP. Sua existência é justificada pelo fato de que os clientes Fabric utilizam gRPC com TLS mútuo, protocolo incompatível com navegadores web, e pelas chaves criptográficas de administrador que não devem ser expostas ao cliente.

3.3.6.1 Arquitetura e Organização

A API é um servidor Express 4 (Node.js 20 LTS, módulos ESM) organizado em módulos especializados:

a) `src/fabric/gateway.js`: conexão gRPC *singleton* reutilizada entre requisições, carregando os certificados X.509 do administrador da Org1 a partir das variáveis de ambiente;

b) `src/ipfs/ipfsService.js`: armazenamento, recuperação e verificação de integridade via `kubo-rpc-client`;

c) `src/did/didService.js`: geração de `did:key`, resolução de `did:web`, emissão e verificação de VCs;

d) `src/zkp/zkpService.js`: geração e verificação de provas via Docker ZoKrates;

e) `src/routes/documentos.js`: todos os endpoints REST de documentos;

f) `src/app.js`: configuração do Express com `express-async-errors` e middleware centralizado de tratamento de erros.

O middleware centralizado de erros (`app.js`) mapeia erros do chaincode para códigos HTTP específicos: `LIMIT_FILE_SIZE` gera 413, `ENDORSEMENT_FAILURE` gera 502, e mensagens do chaincode contendo "não encontrado", "não autoriz" ou "já se encontra revogado" são mapeadas para 404, 403 e 409, respectivamente.

3.3.6.2 Fluxo de Emissão de Documentos

O endpoint `POST /documentos` executa o seguinte fluxo sequencial:

1. **Validação da entrada**: verifica presença do arquivo PDF (`req.file`), MIME type `application/pdf`, campo `tipo` presente e no conjunto de tipos válidos, campo `emissorDominio` presente, e parseia `metadados` como JSON;

2. **Armazenamento IPFS**: invoca `armazenarDocumento(req.file.buffer)`, obtendo `cid` e `hash` (SHA-256 hexadecimal);

3. **Geração do DID do titular**: invoca `gerarDidKey()`, produzindo um par de chaves Ed25519 efêmero e o `did:key` correspondente;

4. **Resolução do DID do emissor**: invoca `resolverDidWeb(emissorDominio)`, obtendo o DID Document da instituição;

5. **Carregamento do par de chaves do emissor**: invoca `obterChaveEmissor(emissorDominio)` para carregar a chave privada de assinatura;

6. **Assinatura da VC**: invoca `assinarVC(credentialSubject, emissorDID, keyPair)`, produzindo a Credencial Verificável completa com *proof* Ed25519;
7. **Geração do UUID**: `randomUUID()` produz o identificador único do documento, garantindo que replays do payload geram IDs distintos;
8. **Submissão ao Fabric**: invoca `contract.submitTransaction('EmitirDocumento', ...)`, aguardando o consenso Raft e o retorno do endosso de ambos os peers.

3.3.6.3 Fluxo de Verificação de Autenticidade

O endpoint `GET /documentos/:id` executa quatro verificações em sequência:

a) **Consulta ao ledger**: `contract.evaluateTransaction('VerificarDocumento', id)` recupera o documento do banco de estado do peer (sem criar transação);

b) **Verificação de integridade IPFS**: `verificarIntegridade(doc.cid,doc.hashDocumento)`, com resultado em `integrityOk`;

c) **Verificação da VC**: `verificarVC(doc.provaVC,doc.emissorDID)`, com resultado em `vcValida`;

d) **Composição do resultado**: o campo `autenticoGeral` é calculado conforme a Equação (3.3).

$$\text{autenticoGeral} = \text{integrityOk} \ \wedge \ \text{vcValida} \ \wedge \ (\text{status} = \texttt{ATIVO}) \tag{3.3}$$

Qualquer falha em uma das três condições implica `autenticoGeral = false`, sinalizando ao verificador que o documento não pode ser considerado autêntico naquele instante.

3.3.6.4 Endpoints Implementados

A Tabela 3.4 lista todos os endpoints expostos pela API.

3.3.7 Experimentos de Validação Funcional

3.3.7.1 Estratégia de Testes

A validação funcional foi conduzida por meio de uma suíte de testes automatizados organizada em quatro níveis: unitário, de integração, *end-to-end* (E2E) e de segurança, cobrindo o chaincode Go e os quatro módulos de serviço da API (`ipfsService`, `didService`, `zkpService` e as rotas de `documentos.js`). A distribuição quantitativa dos casos por módulo e os resultados de execução são apresentados na Seção 4.1 do Capítulo 4.

Tabela 3.4 – Endpoints da API REST do DoChain.

Método	Rota	Descrição	Sucesso
GET	`/health`	Verificação de disponibilidade da API	200
POST	`/documentos`	Emissão de novo documento (multipart/PDF)	201
GET	`/documentos/:id`	Verificação completa em quatro etapas	200
GET	`/documentos/tipo/:tipo`	Listagem de documentos ativos por tipo	200
GET	`/documentos/:id/historico`	Histórico imutável de transações	200
POST	`/documentos/:id/proof`	Geração de prova ZKP	200
GET	`/documentos/:id/verify-proof`	Verificação de prova ZKP	200
DELETE	`/documentos/:id`	Revogação (body: `motivo`)	200

Fonte: Elaborado pelo autor (2026).

Todos os testes unitários e de integração da API utilizam `jest.unstable_mockModule` para substituir os módulos de Fabric, IPFS, DID e ZKP por implementações simuladas (*mocks*), eliminando dependência de infraestrutura Docker em execução. Os testes E2E e de segurança operam sobre um ledger simulado em memória compartilhado entre os casos do mesmo arquivo, permitindo validação de fluxos com estado persistido.

3.3.7.2 Testes dos Fluxos Principais

Três arquivos E2E (`fluxo-diploma.test.js`, `fluxo-atestado.test.js` e `fluxo-escritura.test.js`) validam, de ponta a ponta, os tipos documentais de maior relevância para o trabalho: emissão, verificação em quatro etapas, geração e verificação de prova ZKP (no caso do diploma), revogação e consulta ao histórico imutável. Cada arquivo cobre também cenários específicos do tipo documental testado, como a preservação do metadado `crm` no atestado médico (sensibilidade ALTA) e a rejeição de metadados malformados na escritura. Adicionalmente, `fluxo-completo.test.js` (nível de integração) valida cenários com ledger simulado compartilhado, incluindo a adulteração direta do hash no estado e a tentativa de revogação cruzada entre organizações distintas. Os resultados detalhados de cada fluxo são discutidos na Seção 4.1.2.

3.3.7.3 Testes de Regressão do Chaincode

Os 17 testes unitários do chaincode utilizam implementações *mock* das interfaces `ChaincodeStubInterface`, `ClientIdentity` e `TransactionContextInterface`, dispensando qualquer contêiner Docker. O `mockStub` mantém o estado em um `map[string][]byte`

em memória, e a `mockClientIdentity` retorna a string de identidade configurada por parâmetro.

Os casos cobrem todas as cinco funções do chaincode: emissão com todos os cinco tipos documentais (verificando `Tipo` e `Sensibilidade` na resposta), seis cenários de campos obrigatórios ausentes, tipo inválido, ID duplicado, verificação de documento existente e inexistente, JSON corrompido no estado, revogação pelo emissor correto, revogação por terceiro (deve retornar `ErrNaoAutorizado`), revogação dupla (`ErrDocumentoRevogado`), motivo vazio, listagem com e sem resultados, e histórico com *timestamp* real.

3.3.8 Experimentos de Segurança

3.3.8.1 Metodologia dos Experimentos de Segurança

Os experimentos de segurança foram conduzidos de forma controlada contra a instância local da API, utilizando a suíte `api/tests/security/attack.test.js`, executada em 22 de maio de 2026. A seleção dos quinze cenários não foi arbitrária: partiu-se de uma modelagem de ameaças inspirada no *framework* STRIDE (**??**) (*Spoofing*, *Tampering*, *Repudiation*, *Information Disclosure*, *Denial of Service* e *Elevation of Privilege*), aplicado às superfícies de entrada do protótipo (rota de emissão, ledger, rotas de consulta e revogação, e as três camadas de verificação criptográfica). Cada um dos quatro grupos de cenários corresponde a uma ou mais categorias STRIDE:

a) **Ataques à emissão** (ATK-01 a ATK-05, *Spoofing* e *Tampering* na entrada): tentativas de registrar documentos com identificadores duplicados, tipos inválidos, arquivos não-PDF, domínios DID malformados e operadores de injeção NoSQL nos metadados. Este grupo mira a fronteira de confiança mais exposta do sistema: a rota `POST /documentos`, único ponto em que dados fornecidos por um agente externo (emissor) entram no protótipo antes de qualquer validação;

b) **Ataques à integridade** (ATK-06 a ATK-08, *Tampering* e *Repudiation*): adulteração do hash do documento no ledger, tentativa de sobrescrita de metadados e reenvio de payload (*replay attack*). Este grupo verifica se o encadeamento IPFS → SHA-256 → ledger (Seção 3.3.3.2) de fato impede que um documento seja alterado ou repudiado após o registro;

c) **Ataques ao controle de acesso** (ATK-09 e ATK-10, *Elevation of Privilege*): revogação por MSP não autorizado e revogação dupla de documento já revogado, exercitando diretamente a checagem de identidade do chaincode (`ctx.GetClientIdentity()`, Seção 3.3.2);

d) **Ataques à verificação** (ATK-11 a ATK-15, *Tampering*, *Information Disclosure* e *Denial of Service*): adulteração da assinatura da VC, adulteração da prova ZKP, tipo inexistente na listagem, ID inexistente no histórico e envio de arquivo de 15 MB

acima do limite configurado, este último uma tentativa elementar de esgotamento de memória do processo Node.js.

Ficam fora do escopo desta modelagem, e portanto da suíte, os vetores que exigiriam alterar a infraestrutura subjacente ao invés da lógica de aplicação: *fuzzing* sistemático de payloads, ataques ao protocolo de consenso Raft e varredura de vulnerabilidades em Docker, CouchDB ou no runtime Node.js, formalizados como limitação L5 na Seção 4.6.

3.3.8.1.1 Arquitetura da suíte e fronteira de *mock*

Cada cenário é um teste Jest que percorre a pilha real da API (*middlewares* Express, `multer`, validação de rota, tratamento de erro centralizado) contra uma requisição HTTP construída com `supertest`; apenas as quatro dependências externas de infraestrutura, Fabric Gateway, IPFS, resolução de DID e ZoKrates, são substituídas por *mocks* configuráveis por cenário, conforme o Código 3.18, de modo que a suíte completa executa sem exigir a rede Fabric, o nó IPFS ou os contêineres ZoKrates ativos.

Código 3.18 – Fronteira de *mock* da suíte de segurança: apenas os módulos de infraestrutura são substituídos, preservando a pilha real da API (`api/tests/security/attack.test.js`)

```
jest.unstable_mockModule('../../src/fabric/gateway.js', () => ({
  getContract:     mockGetContract,
  closeConnection: jest.fn(),
}));
jest.unstable_mockModule('../../src/ipfs/ipfsService.js', () => ({
  armazenarDocumento:   mockArmazenarDocumento,
  recuperarDocumento:   jest.fn(),
  verificarIntegridade: mockVerificarIntegridade,
}));
jest.unstable_mockModule('../../src/did/didService.js', () => ({
  gerarDidKey: mockGerarDidKey, resolverDidWeb: mockResolverDidWeb,
  assinarVC:   mockAssinarVC,   verificarVC:    mockVerificarVC,
}));
jest.unstable_mockModule('../../src/zkp/zkpService.js', () => ({
  gerarProva: mockGerarProva, verificarProva: mockVerificarProva,
}));
```

Essa fronteira de *mock* tem uma consequência metodológica importante, detalhada na discussão da Seção 4.2.1: nos cenários em que a defesa depende de uma primitiva criptográfica (Ed25519 em ATK-11, Groth16 em ATK-12), o que a suíte de segurança valida é se a API *propaga corretamente* um veredito de verificação negativo até a resposta final (`vcValida`, `autenticoGeral`), e não a execução da primitiva em si, que é mockada para permitir determinismo e velocidade. A corretude matemática das próprias primitivas é vali-

dada em outros pontos da suíte: o recálculo de SHA-256 é exercitado com a função real de `node:crypto` nos testes unitários de `ipfsService` (sem mock de hash, apenas do transporte `kubo-rpc-client`), e a execução real dos contêineres ZoKrates ocorre nos *benchmarks* de desempenho (`zkp-overhead.bench.js`, Seção 3.3.9.1), cujos tempos de geração e verificação de prova (Tabela 4.3) só são explicáveis por execução criptográfica genuína, não simulada.

Como exemplo de cenário adversarial concreto, o Código 3.19 reproduz a tentativa de injeção NoSQL do ATK-05: o payload inclui operadores literais do MongoDB/CouchDB (`$where`, `$gt`, `$regex`) no campo `metadados`, testando se a camada de persistência os interpretaria como parte de uma consulta.

Código 3.19 – ATK-05: payload de injeção de operadores NoSQL no campo `metadados` (`api/tests/security/attack.test.js`)

```
const payloadInjecao = JSON.stringify({
  '$where': 'sleep(5000)',
  '$gt':    '',
  '$regex': '.*',
});

await request(app)
  .post('/documentos')
  .field('tipo',           'DIPLOMA')
  .field('emissorDominio', 'localhost:3001')
  .field('metadados',      payloadInjecao)
  .attach('arquivo', PDF_BUFFER, { filename: 'diploma.pdf',
     contentType: 'application/pdf' });
```

Cada cenário documenta o comportamento observado (`BLOQUEADO` ou `DETECTADO`), o código HTTP retornado, o tempo de resposta e a evidência textual, persistidos automaticamente em `results.json` ao final da execução (função `registrar()` chamada ao término de cada teste). A distinção entre `BLOQUEADO` (erro retornado ao atacante antes de qualquer efeito no ledger) e `DETECTADO` (operação aceita pela API, mas o sistema expõe o ataque na resposta de verificação subsequente) é a categoria central da análise do modelo de ameaças: ela separa defesas que impedem a ação maliciosa de defesas que apenas tornam seu efeito visível a quem verifica o documento posteriormente; sem essa distinção, um ataque `DETECTADO` poderia ser mal interpretado como falha de segurança.

Os resultados obtidos para os quinze cenários, incluindo código HTTP, mecanismo de defesa acionado e evidência observada, são apresentados e discutidos na Seção 4.2 do Capítulo 4, junto com a distinção entre ataques `BLOQUEADO` e `DETECTADO`.

3.3.9 Experimentos de Desempenho

3.3.9.1 Metodologia dos Benchmarks

Os benchmarks de desempenho foram executados com a infraestrutura Docker completamente inicializada e estabilizada. Os scripts em `api/tests/benchmark/` cobrem cinco dimensões de análise: latência por operação, *throughput* sob concorrência, impacto do tamanho do arquivo no IPFS, escalabilidade do *ledger* e *overhead* das provas ZKP.

A metodologia principal (`latency.bench.js`) executa cada operação 50 vezes sequencialmente e descarta as 3 primeiras execuções como *warm-up* do *gateway* Fabric e do IPFS, resultando em $n_{\text{ef}} = 47$ amostras por operação. As operações de ZKP utilizam $N = 30$ repetições com $W = 3$ descartes, resultando em $n_{\text{ef}} = 27$ amostras, em função do maior custo computacional por execução. Para cada operação são calculadas: média aritmética, mediana, mínimo, máximo, desvio padrão (DP), percentil 95 (P95) e percentil 99 (P99). As tabelas de desempenho do Capítulo 4 utilizam exclusivamente essa execução completa ($n_{\text{ef}} = 47$ ou 27, conforme a operação), reunida nos arquivos `api/tests/benchmark/data/*.csv`.

Durante a execução do benchmark de escalabilidade (`scaling.bench.js`) sob carga sustentada (≈220 documentos emitidos consecutivamente), o Fabric reportou uma falha de endosso (*"ProposalResponsePayloads do not match"*) entre os peers das duas organizações. A causa raiz identificada foi o uso de `time.Now()` no chaincode Go para preencher os campos `DataEmissao` e `DataRevogacao`: como a simulação da transação ocorre de forma independente em cada peer endossante, pequenas divergências de relógio entre os processos geravam *write-sets* distintos, violando a política de endosso. A correção substituiu `time.Now()` por `ctx.GetStub().GetTxTimestamp()`, o *timestamp* assinado da proposta do cliente, idêntico em todos os peers endossantes por construção, eliminando a causa raiz do não determinismo. Após a correção e o *redeploy* do *chaincode*, o benchmark de escalabilidade foi reexecutado integralmente até 500 documentos sem novas falhas de consenso, e os dados finais no Capítulo 4 já refletem essa correção.

As cinco dimensões de desempenho medidas (latência por operação, *throughput* sob concorrência, impacto do tamanho do arquivo no IPFS, escalabilidade do ledger e *overhead* das provas ZK-SNARKs) são apresentadas e discutidas na Seção 4.3 do Capítulo 4, com as respectivas tabelas geradas a partir dos scripts `latency.bench.js`, `throughput.bench.js`, `ipfs.bench.js`, `scaling.bench.js` e `zkp-overhead.bench.js`.

3.3.10 Considerações sobre a Experimentação

A experimentação prática descrita nesta seção permitiu validar o protótipo DoChain em três dimensões complementares, materializando as decisões arquiteturais tomadas nas seções anteriores.

Do ponto de vista funcional, os 154 casos de teste automatizados, cobrindo chaincode Go, serviços Node.js, integração de API, fluxos E2E e segurança, demonstraram que as regras de negócio implementadas (emissão, verificação, revogação e histórico de documentos nos cinco tipos suportados) comportam-se conforme o especificado, incluindo os cenários de erro e rejeição de entradas inválidas.

Do ponto de vista da segurança, os quinze cenários de ataque executados (já contabilizados nos 154 casos totais) não resultaram em comprometimento efetivo do sistema. A discussão detalhada de cada mecanismo de defesa, da independência entre as camadas criptográficas e das oportunidades de melhoria de usabilidade identificadas (mapeamento de erros para códigos HTTP nos cenários ATK-01, ATK-04 e ATK-07) é apresentada na Seção 4.2.1.

Do ponto de vista do desempenho, as medições indicam que o sistema é viável para uso em contextos institucionais com carga moderada. A emissão de um documento completo (PDF + IPFS + DID + VC + Fabric) leva em média 2.106,9 ms, e a verificação em quatro etapas completa em 28,4 ms. O principal gargalo é a geração de provas ZKP (1.254,0 ms), restrita a documentos de alta sensibilidade e opcional do ponto de vista do fluxo principal.

Ressalta-se que todos os experimentos foram realizados em ambiente local sobre máquina única com subsistema WSL2, o que constitui uma limitação relevante. Em ambiente de produção com rede real entre os peers, latências de rede acrescentariam componente adicional ao tempo de endosso. Por outro lado, a latência de emissão em ambiente local inclui o tempo de inicialização de contêineres ZoKrates, que em produção poderia ser amortizado por instâncias persistentes. A análise comparativa entre os resultados experimentais e os dados da literatura, bem como a discussão das implicações para implantação em escala, são desenvolvidas no capítulo seguinte.

4 Resultados

Neste capítulo são apresentados, interpretados e discutidos os resultados obtidos com o protótipo *DoChain*. Os experimentos foram organizados em três eixos complementares: validação funcional, segurança e desempenho. Para cada eixo, os dados são primeiramente apresentados por meio de tabelas e, em seguida, analisados criticamente em relação aos objetivos específicos do trabalho e à literatura relacionada. As limitações identificadas são discutidas ao final do capítulo.

4.1 Resultados da Validação Funcional

A validação funcional foi conduzida por meio de uma suíte de testes automatizados organizada em quatro níveis: unitário, integração, *end-to-end* (E2E) e segurança. A Tabela 4.1 apresenta a distribuição dos casos de teste por módulo.

Tabela 4.1 – Distribuição dos testes automatizados por módulo e resultado.

Módulo	**Ferramenta**	**Escopo**	**Casos**	**Aprovados**
Chaincode Go (`pkg/`)	`go test`	Constantes e `SensibilidadePorTipo`	4	4
Chaincode Go (principal)	`go test`	Cinco funções do contrato com *mock stub*	17	17
`ipfsService`	Jest (ESM)	*Upload*, *download* e verificação de integridade	9	9
`didService`	Jest (ESM)	`did:key`, `did:web`, assinarVC, verificarVC	16	16
`zkpService`	Jest (ESM)	Geração de prova, *field elements*, limpeza	14	14
Integração (API)	Jest + Supertest	Oito rotas em 57 casos	57	57
E2E – Diploma	Jest + Supertest	Ciclo completo com ZKP	8	8
E2E – Atestado Médico	Jest + Supertest	Sensibilidade ALTA, rejeições	7	7
E2E – Escritura	Jest + Supertest	Metadados, sensibilidade MÉDIA	7	7
Segurança	Jest + Supertest	Vetores de ataque documentados	15	15
Total	—	—	**154**	**154**

Fonte: Elaborado pelo autor (2026).

Todos os 154 casos foram aprovados. A estratégia de *mock* adotada nos níveis unitário e de integração, substituindo Fabric, IPFS, DID e ZKP por implementações simuladas, permitiu

executar a suíte completa sem a infraestrutura Docker ativa, reduzindo o ciclo de *feedback* para dezenas de segundos. Os testes E2E e de segurança operam sobre um ledger simulado em memória, validando fluxos com estado persistido entre os casos do mesmo arquivo.

4.1.1 Corretude das Regras de Negócio

Os 17 casos unitários do chaincode Go cobrem as cinco funções do contrato com os cenários mais críticos para a integridade do sistema. Os resultados confirmam as seguintes invariantes:

- **Unicidade de ID**: a função `EmitirDocumento` verifica a existência do ID via `GetState` antes de qualquer escrita, retornando `ErrDocumentoJaExiste` em caso de duplicidade;
- **Classificação automática de sensibilidade**: os cinco tipos documentais foram emitidos e o campo `Sensibilidade` verificado na resposta, confirmando que a função `SensibilidadePorTipo` deriva corretamente os níveis BAIXA, MÉDIA e ALTA sem intervenção do cliente;
- **Exclusividade da revogação**: apenas o MSP que emitiu o documento pode revogá-lo; tentativas por outros emissores retornam `ErrNaoAutorizado`;
- **Idempotência da revogação**: revogar um documento já revogado retorna `ErrDocumentoRevogado`, preservando a data e o motivo da primeira revogação.

4.1.2 Validação dos Fluxos End-to-End

Os três arquivos E2E validam os tipos documentais de maior relevância, cada um percorrendo um ciclo de vida completo:

- **Fluxo DIPLOMA** (8 casos): emissão → verificação com `autenticoGeral: true` → geração de prova ZKP → verificação da prova (`valido: true`) → revogação → tentativa de nova revogação (HTTP 409) → histórico com 2 entradas → verificação pós-revogação (`autenticoGeral: false`). O fluxo valida o ciclo de vida integral do documento, incluindo a irrevogabilidade do histórico no ledger;
- **Fluxo ATESTADO_MEDICO** (7 casos): emissão com `sensibilidade: ALTA` na resposta → verificação da preservação do metadado `crm` no ledger → revogação com motivo registrado → rejeição de tipo inválido (HTTP 400) → rejeição de arquivo não-PDF (HTTP 400). O nível ALTA implica que o fluxo ZKP é recomendado para este tipo, em conformidade com o art. 11 da LGPD (BRASIL, 2018), que exige tratamento especial de dados de saúde;

- **Fluxo ESCRITURA** (7 casos): emissão com metadados imobiliários → preservação dos metadados no ledger simulado → emissão sem metadados opcionais → rejeição de JSON malformado no campo `metadados`. Esses casos cobrem o comportamento com campos parcialmente preenchidos, situação comum em documentos imobiliários reais.

A aprovação integral dos 22 casos E2E confirma que os fluxos de negócio funcionam de ponta a ponta para os três tipos documentais mais representativos do problema abordado neste trabalho. O cenário de adulteração direta do campo `hashDocumento` no estado simulado resultou corretamente em `autenticoGeral: false` e `integrityOk: false`, demonstrando que a verificação em quatro etapas (das quais três compõem condições booleanas de autenticidade, Equação (3.3)) detecta discrepâncias introduzidas fora do fluxo normal de emissão.

4.2 Resultados dos Experimentos de Segurança

Os experimentos de segurança foram conduzidos por meio de uma suíte automatizada (`api/tests/security/attack.test.js`) composta por 15 cenários de ataque organizados em quatro grupos de vetores, derivados de uma modelagem de ameaças STRIDE (**??**) sobre as superfícies de entrada do protótipo, conforme detalhado na Seção 3.3.8.1. Trata-se de testes funcionais adversariais controlados, direcionados a superfícies de ataque conhecidas do sistema (validação de entrada, controle de acesso e integridade criptográfica), não de uma avaliação adversarial abrangente no sentido de um teste de penetração ou auditoria de segurança formal, que incluiria técnicas como *fuzzing* e tentativas de quebra do protocolo de consenso, fora do escopo deste trabalho (Seção 4.6). Como discutido na Seção 3.3.8.1, apenas as dependências de infraestrutura (Fabric Gateway, IPFS, resolução de DID e ZoKrates) são substituídas por *mocks* configuráveis por cenário; a pilha de validação e roteamento da API é exercitada de forma real e integral em cada requisição. A Tabela 4.2 apresenta os resultados de cada cenário.

4.2.1 Discussão dos Resultados de Segurança

Nenhum dos 15 cenários resultou em comprometimento efetivo do sistema. Os ataques classificados como `DETECTADO` foram corretamente identificados pelas camadas criptográficas ou de verificação, mesmo quando a API retornou HTTP 200, o que é semanticamente adequado, pois a consulta foi processada com sucesso e o resultado informa ao verificador que o documento não é autêntico.

O grupo de ataques à emissão (ATK-01 a ATK-05) evidencia que a rota `POST /documentos` implementa validação em camadas antes de qualquer interação com o chaincode: o tipo documental é checado contra o array `TIPOS_VALIDOS` na própria rota (ATK-02, 4 ms), o MIME type do arquivo é verificado logo após o processamento do `multer` (ATK-03, 3 ms), e apenas então a requisição alcança a rede Fabric, onde o contrato rejeita identificadores duplicados

Tabela 4.2 – Resultados dos experimentos de segurança.

ID	Cenário	Mecanismo de Defesa	Evidência	Status	HTTP	Tempo (ms)
ATK-01	Emissão com ID duplicado	`GetState` retorna `ErrDocumentoJaExiste` antes do `PutState`	2ª emissão com mesmo ID retorna erro no chaincode	BLOQUEADO	500	218
ATK-02	Tipo de documento inválido	Validação do array `TIPOS_VALIDOS` na rota antes de qualquer chamada ao chaincode	Mensagem: *Tipo inválido: "PASSAPORTE_FALSO"*	BLOQUEADO	400	4
ATK-03	Arquivo não-PDF (MIME inválido)	Verificação `req.file.mimetype` pós-multer	Mensagem: *Apenas arquivos PDF são aceitos*	BLOQUEADO	400	3
ATK-04	Domínio DID emissor malformado	Falha na resolução `did:web` propagada pelo *handler* centralizado	HTTP 500 (ausência de validação prévia do formato de domínio)	BLOQUEADO	500	29
ATK-05	Injeção NoSQL nos metadados	Metadados armazenados como *string* literal; query CouchDB usa seletor fixo	Operadores `$where`/`$gt` armazenados como texto, não executados	DETECTADO	201	5
ATK-06	Adulteração do hash pós-emissão	SHA-256 recalculado sobre conteúdo IPFS comparado ao `hashDocumento` do ledger	`integrityOk=false`, `autenticoGeral=false`	DETECTADO	200	10
ATK-07	Adulteração de metadados no ledger	Imutabilidade da blockchain + `ErrDocumentoJaExiste` como segunda camada	Sobrescrita arquiteturalmente impossível sem MSP válido + consenso Raft	BLOQUEADO	500	13
ATK-08	*Replay attack*	`randomUUID()` gerado pelo servidor por requisição	IDs distintos garantidos; atacante não controla o identificador	DETECTADO	201	18
ATK-09	Revogação por MSP não autorizado	`ctx.GetClientIdentity().GetID()` comparado a `doc.EmissorMSP`	*Operação não autorizada para este emissor*	BLOQUEADO	403	178
ATK-10	Revogação dupla	Verificação `doc.Status == StatusRevogado` no chaincode	*Documento já se encontra revogado*	BLOQUEADO	409	6
ATK-11	Assinatura VC adulterada	`Ed25519Signature2020.verify()` com chave pública do DID Document	`vcValida=false`, `autenticoGeral=false`	DETECTADO	200	2
ATK-12	Prova ZKP com byte adulterado	Verificação matemática Groth16/BN128 pelo ZoKrates	`valido=false`: equação de paridade inválida	DETECTADO	200	3
ATK-13	Listagem com tipo inexistente	Validação `TIPOS_VALIDOS` na rota `GET /tipo/:tipo`	*Tipo inválido: "TIPO_QUE_NAO_EXISTE"*	BLOQUEADO	400	3
ATK-14	Histórico de ID inexistente	`GetState` verificado antes de `GetHistoryForKey`	*Documento não encontrado*	DETECTADO	404	3
ATK-15	Upload acima do limite (15 MB)	`limits.fileSize` configurado no multer (10 MB)	Rejeição antes de alocar memória	BLOQUEADO	413	113

Fonte: Elaborado pelo autor (2026), com base em `api/tests/security/results-table.csv`.

via `GetState` (ATK-01, 218 ms). A diferença de duas ordens de grandeza entre os tempos de resposta de ATK-02/ATK-03 (validação síncrona em memória, sem I/O) e ATK-01 (que depende do ciclo completo de simulação e endosso do Fabric antes de falhar) ilustra o custo de rejeitar uma entrada apenas na camada mais profunda do sistema, reforçando o valor de validar o mais cedo possível na cadeia de chamadas. O ATK-05 (Código 3.19, Seção 3.3.8.1) testa uma classe de vulnerabilidade real em sistemas que utilizam bancos de documentos: operadores como `$where` podem, em esquemas mal projetados, ser interpretados como parte de uma consulta dinâmica ao CouchDB/MongoDB, permitindo desde negação de serviço (`sleep(5000)`) até contorno de

condições de busca. No protótipo, o campo `metadados` é serializado como *string* JSON opaca no chaincode (Seção 3.3.2) e as *Mango queries* do CouchDB usam seletores fixos sobre os campos `tipo` e `status` (Seção 4.3.4), nunca sobre `metadados`; por isso o payload é armazenado como texto literal e o ataque é classificado como `DETECTADO`: aceito, porém inerte. Essa defesa é incidental à modelagem de dados (metadados como *string* opaca), e não uma sanitização de entrada deliberada: caso uma versão futura do sistema passe a indexar ou consultar o conteúdo de `metadados` diretamente no CouchDB, a mesma superfície deixaria de estar protegida, o que recomenda tratar esse campo com uma biblioteca de sanitização explícita (por exemplo, removendo chaves iniciadas por $) antes de qualquer exposição futura a consultas dinâmicas.

Os ataques ao controle de acesso (ATK-09 e ATK-10) revelam duas propriedades complementares: a revogação por MSP não autorizado foi bloqueada em 178 ms com HTTP 403, enquanto a tentativa de revogação dupla foi bloqueada em apenas 6 ms, diferença explicada pelo fato de que a verificação de status no chaincode retorna `ErrDocumentoRevogado` imediatamente, sem executar a sequência completa de endosso.

Os ataques criptográficos (ATK-06, ATK-11 e ATK-12) demonstram que os mecanismos de defesa do sistema operam em camadas independentes entre si: a verificação SHA-256 detecta adulterações no conteúdo IPFS, a verificação Ed25519 detecta adulterações na assinatura da Credencial Verificável (BERNSTEIN et al., 2012), e a verificação Groth16 detecta qualquer modificação nos *field elements* da prova ZKP (GROTH, 2016). Essa independência entre camadas é a propriedade mais relevante do ponto de vista de segurança: um ataque que comprometesse exclusivamente o IPFS seria detectado pela verificação de hash; um ataque à assinatura VC seria detectado pela verificação Groth16.

Uma ressalva metodológica qualifica a leitura de ATK-11 e ATK-12. Como detalhado na Seção 3.3.8.1, a suíte de segurança mocka os módulos `didService` e `zkpService` na fronteira de infraestrutura, de modo que `verificarVC` e `verificarProva` são configurados para retornar o veredito esperado (`false`) em vez de executarem, respectivamente, o algoritmo Ed25519 real ou o binário ZoKrates dentro do teste. O Código 4.1 mostra o payload de ATK-11: uma Credencial Verificável cuja `proof.proofValue` foi substituída por uma *string* de assinatura de outra chave; o Código 4.2 mostra ATK-12, em que um único caractere do *field element* `proof.a[0]` da prova Groth16/BN128 é alterado.

Código 4.1 – ATK-11: Credencial Verificável com `proofValue` substituído por assinatura de outra chave (`api/tests/security/attack.test.js`)

```
const provaAdulterada = JSON.stringify({
  '@context': ['https://www.w3.org/2018/credentials/v1'],
  type:       ['VerifiableCredential'],
  issuer:     'did:web:localhost:3001',
  proof: {
    type:       'Ed25519Signature2020',
    proofValue: 'zASS1NATURA_ADUL73RADA_D3_OUTR4_CH4V3_Ed25519XyZ',
```

```
  },
});
```

Código 4.2 – ATK-12: alteração de um único caractere do *field element* `proof.a[0]` da prova ZKP (`api/tests/security/attack.test.js`)

```
const provaAdulterada = JSON.parse(JSON.stringify(PROVA_ZKP_VALIDA));
const a0original = provaAdulterada.proof.a[0];
provaAdulterada.proof.a[0] =
  a0original.slice(0, -1) + (a0original.endsWith('1') ? '2' : '1');
```

O que ATK-11 e ATK-12 efetivamente validam, portanto, é que a API compõe corretamente o veredito final (`vcValida=false` e `autenticoGeral=false` em ATK-11; `valido=false` em ATK-12) a partir do resultado retornado pela camada de verificação, e não a rejeição criptográfica em si. O fato de uma assinatura Ed25519 ou uma equação de paridade Groth16 realmente falhar diante de um único bit alterado é uma propriedade matemática das próprias primitivas (BERNSTEIN et al., 2012; GROTH, 2016), cuja implementação é exercitada de forma real (sem mock de criptografia) na geração e verificação de prova dos *benchmarks* de desempenho (Tabela 4.3, Seção 4.3.1), onde os tempos médios de 1.254,0 ms e 602,1 ms só são explicáveis por execução genuína dos contêineres ZoKrates, e não por resultados simulados. O recálculo de SHA-256 do ATK-06, por sua vez, é uma exceção dentro da própria suíte de segurança: a função de *hash* de `node:crypto` nunca é mockada nos testes de `ipfsService` (apenas o transporte `kubo-rpc-client` o é), de modo que a detecção de adulteração de conteúdo em ATK-06 já reflete um cálculo criptográfico real, e não apenas a propagação de um veredito pré-configurado.

O ATK-08 (*replay attack*) merece leitura mais cautelosa que a classificação `DETECTADO` sugere isoladamente: como o servidor gera um novo `randomUUID()` a cada requisição, o reenvio do mesmo payload é aceito e produz um segundo registro do mesmo conteúdo sob um ID distinto do original. Do ponto de vista criptográfico, o ataque não compromete a integridade de nenhum registro existente; do ponto de vista documental, porém, o resultado é uma duplicidade: por exemplo, uma escritura reenviada é registrada duas vezes no *ledger*, sob identificadores diferentes, potencialmente problemática em contextos nos quais a unicidade de um ato tem valor jurídico, como o cartorial. A defesa atual evita apenas a sobrescrita silenciosa de um estado existente, mas não impede a duplicação de intenção; uma defesa mais completa exigiria deduplicação por conteúdo (associando de forma unívoca cada CID/hash já registrado a um único ID de documento) ou controle de idempotência na API, ambos fora do escopo do protótipo atual.

O quarto grupo (ATK-13 a ATK-15) mira comportamentos de disponibilidade e o tratamento de entradas fora do domínio esperado, sem explorar diretamente falhas criptográficas ou de controle de acesso. ATK-13 e ATK-14 confirmam que consultas com identificadores inexistentes

(um tipo documental fora do enum e um ID de documento nunca emitido) retornam respostas rápidas e específicas (3 ms e HTTP 400; 3 ms e HTTP 404, respectivamente) sem chegar a consultar o CouchDB ou o *ledger* desnecessariamente, o que também mitiga, de forma incidental, um vetor elementar de exaustão de recursos por requisições malformadas repetidas.

O ATK-15 testa diretamente um vetor de negação de serviço por consumo de memória: o Código 4.3 envia um arquivo de 15 MB, acima do limite configurado na rota de emissão.

Código 4.3 – ATK-15: envio de arquivo de 15 MB para testar o limite de tamanho do `multer` (`api/tests/security/attack.test.js`)

```
const bigBuffer = Buffer.alloc(15 * 1024 * 1024, 37);

const resp = await request(app)
  .post('/documentos')
  .field('tipo',           'DIPLOMA')
  .field('emissorDominio', 'localhost:3001')
  .attach('arquivo', bigBuffer, { filename: 'grande.pdf', contentType
     : 'application/pdf' });
```

O teste é deliberadamente escrito para aceitar qualquer um de quatro códigos de status (`[201, 400, 413, 500]`) em vez de exigir apenas HTTP 413: o objetivo do cenário é documentar o comportamento real do sistema perante um arquivo acima do limite, não presumir de antemão em qual camada da pilha a rejeição ocorreria. O resultado observado confirma que o `multer` está configurado com `limits: { fileSize: 10 * 1024 * 1024 }` (`api/src/routes/documentos.js`) e rejeita o *upload* com HTTP 413 antes de alocar o *buffer* completo em memória (113 ms, Tabela 4.2). Esse desenho de teste tolerante ao resultado é uma escolha metodológica deliberada para cenários de disponibilidade: como a rejeição de um arquivo grande poderia, em princípio, ser imposta em diferentes pontos da pilha (proxy reverso, middleware do Express ou o próprio `multer`), fixar um único código HTTP esperado tornaria o teste frágil a mudanças de configuração que não afetam a garantia de segurança relevante: a de que arquivos grandes não sejam integralmente alocados em memória sem controle antes de qualquer rejeição.

Três aspectos requerem atenção por comprometerem a usabilidade sem, no entanto, representar vulnerabilidades de segurança, já que a operação falha corretamente em todos os casos. O ATK-04 revela a ausência de validação prévia do formato do campo `emissorDominio` na rota de emissão, resultando em HTTP 500 em vez de HTTP 400; a correção consiste em adicionar validação de formato de domínio antes da chamada a `resolverDidWeb`. Defeito análogo ocorre no ATK-01 e no ATK-07 (Tabela 4.2): ambos retornam HTTP 500 quando o chaincode responde com `ErrDocumentoJaExiste`, um erro de conflito de estado que deveria ser mapeado para HTTP 409 (*Conflict*) pelo middleware centralizado de erros da API (Seção 3.3.6.1), em vez de propagado como falha genérica de servidor.

4.3 Resultados dos Experimentos de Desempenho

4.3.1 Latência por Operação

O *benchmark* de latência (`latency.bench.js`) executou a metodologia completa descrita na Seção 3.3.9.1 ($n_{\text{ef}} = 47$ para as operações gerais; $n_{\text{ef}} = 27$ para as operações de ZKP) sobre a infraestrutura Docker completamente inicializada, já com a correção de não determinismo do *chaincode* aplicada (Seção 3.3.9.1). A Tabela 4.3 apresenta os resultados para as sete operações medidas.

Tabela 4.3 – Latência das operações principais do sistema.

Operação	n	Média (ms)	Mediana (ms)	Mín (ms)	Máx (ms)	DP (ms)	P95 (ms)	P99 (ms)
Emissão (POST /documentos)	47	2.106,9	2.105,9	2.096,8	2.120,7	6,5	2.120,2	2.120,7
Verificação (GET /documentos/:id)	47	28,4	26,1	21,9	44,3	5,6	39,3	44,3
Revogação (DELETE /documentos)	47	2.082,5	2.082,7	2.071,0	2.095,8	5,3	2.090,9	2.095,8
Listagem por tipo	80	138,7	133,1	100,5	192,2	21,7	174,4	192,2
Histórico	40	7,5	7,3	5,7	10,9	1,3	10,5	10,9
Geração de prova ZKP	27	1.254,0	1.212,4	1.142,9	1.732,5	115,5	1.386,5	1.732,5
Verificação de prova ZKP	27	602,1	599,8	561,3	658,0	25,4	653,6	658,0

Fonte: Elaborado pelo autor (2026), com base no *benchmark* de latência (`latency.bench.js`) descrito na Seção 3.3.9.1. Nota: n variável conforme a operação: emissão, verificação e revogação usam $n_{\text{ef}} = 47$ (metodologia principal, Seção 3.3.9.1); geração e verificação de prova ZKP usam $n_{\text{ef}} = 27$; listagem por tipo ($n = 80$) agrega 20 execuções em cada um de quatro tamanhos de ledger (10, 50, 100 e 200 documentos), medidas em estado estável do ledger, sem escritas concorrentes; histórico ($n = 40$) agrega 20 execuções em cada uma das duas profundidades de versão testadas (1 e 2 versões). Os valores de listagem por tipo aqui reportados (100,5–192,2 ms) *não* são diretamente comparáveis aos da Tabela 4.6 para os mesmos tamanhos de ledger (340,7–688,9 ms): esta última é medida pelo `scaling.bench.js`, um *benchmark* distinto que mede a listagem imediatamente após cada lote de emissões, quando a *Mango query* subjacente (sem índice CouchDB declarado explicitamente, Seção 4.3.4) sofre o custo adicional de escritas recentes ainda em processamento pelo CouchDB. As duas tabelas respondem a perguntas diferentes, latência em estado estável *versus* latência imediatamente após carga de escrita e não devem ser lidas como uma série única.

A emissão (2.106,9 ms em média) e a revogação (2.082,5 ms) são dominadas pelo consenso e são da mesma ordem de grandeza entre as operações de escrita (≈2,1 s), ambas próximas do `BatchTimeout` de 2 s configurado no orderer Raft (LINUX FOUNDATION, 2021). Esse resultado indica que o consenso é o fator determinante da latência de escrita: o custo adicional da emissão (*upload* ao IPFS, geração do par de chaves Ed25519, resolução do DID Document e assinatura da Credencial Verificável) é comparativamente pequeno (dezenas de milissegundos) frente ao tempo de finalização de bloco, o que explica por que emissão e revogação, apesar de envolverem cadeias de operação muito diferentes, apresentam latências praticamente indistinguíveis.

A verificação (28,4 ms) é ordens de grandeza mais rápida por ser executada como `evaluateTransaction`, resolvida localmente no peer sem passar pelo orderer, mesmo incluindo o *download* do conteúdo no IPFS, o recálculo do SHA-256 e a verificação criptográfica da VC. A listagem por tipo (138,7 ms em média, com variação de 100,5 a 192,2 ms) é mais custosa que a verificação por chave por executar uma *Mango query* no CouchDB sem índice declarado

explicitamente (Seção 4.3.4), e o histórico (7,5 ms) é a operação mais rápida do sistema, por consultar diretamente o log de versões da chave via `GetHistoryForKey`.

A geração de prova ZKP (1.254,0 ms) permanece a operação de maior custo computacional isolado, por envolver dois comandos Docker sequenciais (`compute-witness` e `generate-proof`) que incluem o tempo de inicialização do contêiner ZoKrates. A verificação da prova já gerada (602,1 ms), medida diretamente, é significativamente menor que o da geração, ainda que não desprezível frente às demais operações do sistema.

4.3.2 Comportamento sob Concorrência

O *benchmark* de *throughput* (`throughput.bench.js`) avaliou o comportamento do sistema sob quatro níveis de concorrência (1, 5, 10 e 20 requisições simultâneas), com 5 lotes por nível, tanto para emissão quanto para verificação. A Tabela 4.4 apresenta os resultados observados.

Tabela 4.4 – Comportamento do sistema sob concorrência – emissão e verificação de DIPLOMA (PDF de 200 KB).

Concorrência	Operação	Sucessos	Erros	Throughput (req/s)	Lat. média (ms)
1	Emissão	5	0	0,47	2.134,0
5	Emissão	25	0	2,35	2.129,1
10	Emissão	49	1	24,60	654,2
20	Emissão	100	0	32,84	552,6
1	Verificação	5	0	40,54	24,7
5	Verificação	25	0	52,43	66,0
10	Verificação	50	0	56,00	111,0
20	Verificação	100	0	59,54	207,7

Fonte: Elaborado pelo autor (2026), com base no *benchmark* de *throughput* descrito na Seção 3.3.9.1.

O resultado mais relevante da emissão sob concorrência é contraintuitivo à primeira vista: a latência média por requisição *cai* de ≈2.130 ms (concorrência 1–5) para ≈550–650 ms (concorrência 10–20), enquanto o *throughput* agregado sobe de 0,47 req/s (concorrência 1) a 32,84 req/s (concorrência 20). Cabe uma ressalva metodológica sobre a linha de concorrência 10: o *throughput* de 24,60 req/s e a latência média de 654,2 ms da mesma linha não devem ser interpretados como grandezas reciprocamente consistentes por uma relação direta $\textit{throughput} = \textit{concorrência}/\textit{latência}$. Essa relação vale exatamente para um único lote homogêneo, mas a Tabela 4.4 agrega 5 lotes por nível de concorrência (Seção 4.3.2): o *throughput* de cada lote é calculado como sucessos dividido pelo tempo total do lote, e a latência reportada é a média das latências individuais das requisições daquele lote, duas grandezas calculadas independentemente por lote e só então agregadas, cada uma, por média aritmética simples entre os 5 lotes. A média das razões não coincide, em geral, com a razão das médias, de modo que a relação

throughput $\leq$ *concorrência*$/$*latência* não é garantida para os valores agregados, ainda que valha, aproximadamente, em cada lote individual. O efeito é evidente nesta linha: dos 5 lotes de emissão em concorrência 10, um único lote apresentou 1 erro em 10 requisições e um tempo total de lote muito acima dos demais, o que eleva desproporcionalmente a média das latências (de $\approx$333 ms nos outros quatro lotes para 654,2 ms no agregado) sem reduzir na mesma proporção a média dos *throughputs* por lote, já que o *throughput* desse lote isolado permanece uma razão limitada (sucessos sobre tempo), não uma grandeza que colapsa com o mesmo peso relativo da inflação de sua própria latência. Reexaminando apenas os quatro lotes sem esse ponto atípico, a média de *throughput* ($\approx$29,7 req/s) e o teto *concorrência*$/$*latência* calculado sobre a média de latência desses mesmos lotes ($10/0,333$ s $\approx 30,0$ req/s) convergem, como esperado. O valor de 24,60 req/s reportado na tabela é, portanto, correto e consistente com os dados brutos do *benchmark*; a causa do erro pontual observado nesse lote é discutida adiante nesta seção.

A explicação para a queda de latência e o aumento de *throughput* entre os níveis de concorrência está no mecanismo de *batching* do orderer Raft, que combina dois parâmetros configurados em `configtx.yaml` (Seção 3.3.1.2) com papéis distintos: o `BatchTimeout` (2 s) define o tempo máximo de espera antes de fechar um bloco parcialmente preenchido, enquanto o `MaxMessageCount` (10 transações) fecha o bloco imediatamente por lotação assim que esse número de transações é atingido, independentemente do tempo decorrido. Em concorrência 1 e 5, o número de transações simultâneas é insuficiente para atingir o `MaxMessageCount`, de modo que o bloco permanece aberto até o `BatchTimeout` expirar e cada submissão paga o custo quase integral da janela de 2 s, por isso a concorrência 5, embora também caiba dentro da janela do timeout, não se beneficia de qualquer redução de latência. A partir da concorrência 10, o número de transações simultâneas iguala ou supera o `MaxMessageCount`, o bloco fecha por lotação antes do `BatchTimeout` expirar, e o custo fixo de finalização é amortizado entre todas as transações do lote, o que explica a queda abrupta de latência e o correspondente aumento de *throughput* observados a partir dessa faixa de concorrência. O único erro observado (concorrência 10) é consistente com um `MVCC_READ_CONFLICT` (LINUX FOUNDATION, 2021) pontual; como o DoChain usa `randomUUID()` como chave do documento, o risco estrutural de conflito de chave é mitigado, e a taxa de erro observada (1 em 50) é compatível com colisões de agendamento no bloco, não de chave.

A verificação escala de forma mais previsível (40,54 a 59,54 req/s), com aumento moderado da latência média sob contenção (24,7 ms a 207,7 ms), atribuível ao enfileiramento de requisições no laço de eventos único do Node.js, não ao Fabric. Para o contexto de uso de documentos legais, no qual a frequência de emissão é tipicamente de dezenas a centenas de documentos por dia por instituição, o *throughput* observado (superior a 30 req/s de emissão e a 55 req/s de verificação sob concorrência) é amplamente suficiente.

4.3.3 Impacto do Tamanho do Arquivo no IPFS

O *benchmark* IPFS (`ipfs.bench.js`) avaliou, com 20 execuções por operação, as latências de *upload*, *download* e verificação de integridade para cinco tamanhos de arquivo entre 100 KB e 10 MB. A Tabela 4.5 apresenta os resultados.

Tabela 4.5 – Latência das operações IPFS por tamanho de arquivo ($n = 20$ por operação).

Tamanho	*Upload* – Média (ms)	*Download* – Média (ms)	Verificação – Média (ms)
100 KB	21,7	3,0	3,2
500 KB	24,6	7,5	7,7
1 MB	28,9	10,7	13,8
5 MB	68,8	46,2	51,9
10 MB	100,8	70,7	95,9

Fonte: Elaborado pelo autor (2026), com base no *benchmark* de IPFS descrito na Seção 3.3.9.1.

Observa-se relação aproximadamente linear entre o tamanho do arquivo e a latência de *upload* a partir de 1 MB, com coeficiente de ≈8 ms/MB, consideravelmente mais rápido que o esperado para uma rede de longa distância, consistente com o fato de o nó IPFS operar na mesma rede virtual Docker que a API, sem atravessar a Internet pública. O *download* segue o mesmo padrão, com coeficiente de ≈7 ms/MB, ligeiramente mais rápido que o *upload* por o IPFS poder iniciar a entrega em blocos antes de completar a leitura do conteúdo inteiro. A latência de verificação de integridade equivale ao tempo de *download* acrescido do cálculo SHA-256, com *overhead* perceptível apenas a partir de 5 MB. Para documentos legais típicos em PDF (200 KB a 1 MB), a latência de *upload* ao IPFS é da ordem de 20 a 30 ms, uma fração pequena do tempo total de emissão observado na Tabela 4.3, reforçando a conclusão da Seção 4.3.1 de que o consenso Raft, e não o IPFS, é o fator dominante na latência de escrita.

4.3.4 Escalabilidade do Ledger

O *benchmark* de crescimento do ledger (`scaling.bench.js`) mediu as latências de verificação e listagem, e o tamanho estimado do banco CouchDB, após emissões em lote crescente (10, 50, 100, 200 e 500 documentos). A Tabela 4.6 apresenta os resultados. Conforme descrito na Seção 3.3.9.1, a execução completa até 500 documentos só foi possível após a correção do não determinismo do *chaincode* (`time.Now()` → `GetTxTimestamp()`); a execução anterior à correção falhava por violação da política de endosso ao redor de 220 documentos emitidos sob carga sustentada.

A latência de verificação por ID permanece na ordem de dezenas de milissegundos em todo o intervalo testado, consistente com a resolução por chave exata (`GetState`) em tempo $O(1)$ no CouchDB; o valor pontual mais alto em 200 documentos (73,3 ms) não se repete em 500 documentos e é atribuído à variância amostral, dado o tamanho reduzido da amostra

Tabela 4.6 – Comportamento do sistema com crescimento do ledger ($n = 5$ amostras por ponto).

Total de documentos	Verificação (ms)	Listagem por tipo (ms)	CouchDB (MB est.)
10	27,9	340,7	1,910
50	26,3	301,6	2,170
100	24,7	355,2	2,500
200	73,3	688,9	2,270
500	31,2	563,9	3,480

Fonte: Elaborado pelo autor (2026), com base no *benchmark* de escalabilidade descrito na Seção 3.3.9.1.

($n = 5$ por ponto); uma medição mais rigorosa, com mais repetições por ponto, é indicada como trabalho futuro. A listagem por tipo (`ListarPorTipo`), que executa uma *Mango query* sem índice explícito declarado, apresenta tendência de crescimento a partir de 100 documentos, ainda que não estritamente monotônica pelo mesmo motivo. Em produção, a declaração de um índice CouchDB para os campos `tipo` e `status` em `META-INF/statedb/couchdb/indexes/` reduziria a latência de listagem para comportamento $O(\log n)$ sem necessidade de alteração do *chaincode*. O tamanho do banco CouchDB cresce de forma geral com o número de documentos, mas também não de forma estritamente monotônica entre os pontos intermediários, refletindo processos internos de compactação do próprio CouchDB que ocorrem de forma assíncrona em relação à medição.

4.3.5 Overhead das Provas ZK-SNARKs

O *benchmark* de overhead ZKP (`zkp-overhead.bench.js`) comparou dois modos de verificação: Modo A (apenas verificação convencional) e Modo B (verificação + geração + verificação de prova ZK-SNARK). A Tabela 4.7 apresenta os resultados com $n = 27$ amostras efetivas após descarte de *warm-up*.

Tabela 4.7 – Overhead das provas ZK-SNARKs – comparação entre Modo A e Modo B.

Modo	Média (ms)	Mediana (ms)	Overhead (ms)	Overhead (%)
A – Verificação convencional	33,0	32,4	—	—
B – Verificação + geração + ZKP	2.037,6	2.022,1	2.004,6	≈6.074,5%

Fonte: Elaborado pelo autor (2026), com base no *benchmark* de *overhead* ZKP descrito na Seção 3.3.9.1.

O *overhead* absoluto (≈2.004,6 ms) é dominado pelo tempo de geração da prova no contêiner ZoKrates (≈1.254 ms, Tabela 4.3) somado à verificação da prova já gerada (≈602 ms); a soma aproximada dessas duas parcelas com a verificação convencional (33,0 ms + 1.254,0 ms + 602,1 ms ≈1.889 ms) é consistente, com a diferença residual explicada pelo custo de orquestração sequencial entre as três chamadas HTTP do Modo B. O *overhead percentual* (≈6.074,5%) é numericamente elevado porque decorre de uma razão entre um denominador pequeno, a verificação convencional (≈33 ms), e um numerador dominado pelo custo absoluto de geração e

verificação da prova ZKP; por essa razão, o *overhead* absoluto, não o percentual, é a métrica mais estável e relevante para dimensionar o custo real da operação. Esse custo adicional corresponde ao mecanismo de conhecimento implementado no protótipo como prova de conceito do *toolchain* ZoKrates (Seção 3.2), sem garantia de privacidade documental efetiva na versão atual: o circuito verifica apenas a igualdade entre o hash fornecido pelo provador e o hash já registrado publicamente no *ledger* (Código 3.15), sem calcular o SHA-256 do documento nem vincular a prova ao conteúdo original. Ainda assim, o custo medido é relevante para dimensionar o impacto de uma futura versão do circuito que incorpore o cálculo do hash internamente (Seção 5.2), aplicável sobretudo a documentos de sensibilidade ALTA sujeitos ao art. 11 da LGPD (BRASIL, 2018). Documentos de sensibilidade BAIXA (diplomas, CNHs) podem ser verificados exclusivamente via Modo A, sem envolvimento do ZoKrates.

4.4 Atendimento aos Objetivos Específicos

Com base nos resultados apresentados nas seções anteriores, é possível avaliar o atendimento de cada objetivo específico definido na Seção 1.3.2. A Tabela 4.8 sintetiza essa avaliação.

Tabela 4.8 – Avaliação do atendimento aos objetivos específicos.

OE	Objetivo (síntese)	Evidência de Atendimento	Resultado
OE-1	Classificação documental	5 tipos com 3 níveis de sensibilidade derivados automaticamente; rejeição de tipo inválido (ATK-02, HTTP 400); ciclo de vida diferenciado validado nos fluxos E2E	Atendido
OE-2	DIDs W3C	`did:key` e `did:web` implementados e testados (15 casos); verificação autônoma por terceiros sem acesso à rede Fabric; adulteração de VC detectada (ATK-11)	Atendido
OE-3	Metadados extensível	Campo `Metadados` como JSON livre; três schemas distintos validados nos testes E2E; novos tipos incorporáveis sem redeployment do chaincode	Atendido
OE-4	Protótipo funcional com métricas	154 casos aprovados; fluxos E2E para DIPLOMA, ATESTADO_MEDICO e ESCRITURA; 5 dimensões de desempenho quantificadas em tabelas	Atendido

Fonte: Elaborado pelo autor (2026).

O OE-1 foi atendido com uma contribuição adicional em relação ao planejado: a derivação automática do nível de sensibilidade no chaincode impede que um cliente mal-intencionado registre um atestado médico com nível BAIXA, transferindo o controle da classificação para a lógica imutável do contrato.

O OE-2 foi atendido com a propriedade de verificação autônoma: dado o DID Document da instituição, qualquer terceiro pode verificar a assinatura Ed25519 da Credencial Verificável sem contatar a instituição emissora nem ter acesso à rede Fabric (SPORNY et al., 2022a). Esse resultado é especialmente relevante para o cenário brasileiro, onde a interoperabilidade entre sistemas de diferentes instituições é historicamente limitada. Ressalva-se, contudo, que o atendimento é parcial quanto à autonomia do titular: o par de chaves `did:key` é gerado de forma efêmera pelo próprio servidor no momento da emissão (Seção 3.3.6.2), e não pelo titular em um dispositivo sob seu controle, de modo que a identidade auto-soberana descrita na Seção 3.1.6 não é, ainda, efetivamente auto-soberana na versão atual do protótipo; a custódia da chave pelo titular é discutida como evolução necessária na Seção 5.2.

O OE-3 foi atendido com a abordagem de JSON livre no campo `Metadados`, que demonstrou ser suficientemente flexível para acomodar schemas completamente distintos (dados acadêmicos, clínicos e imobiliários) em um único sistema, sem nenhuma alteração no chaincode entre os três tipos testados.

4.5 Comparação com a Literatura

A Tabela 4.9 compara o DoChain com as iniciativas mais relevantes identificadas na revisão bibliográfica, destacando os aspectos técnicos de maior impacto para o problema de autenticação de documentos legais.

Tabela 4.9 – Comparação do DoChain com iniciativas relacionadas da literatura.

Critério	**DoChain**	**OpenCerts** (GOVERNMENT TECHNOLOGY AGENCY OF SINGAPORE (GOVTECH), 2020)	**EBSI** (EUROPEAN COMMISSION, 2020)	**Mukne et al.** (MUKNE et al., 2019)
Plataforma	Fabric (permissionado)	Ethereum (público)	Besu (permissionado)	Fabric (permissionado)
Tipos documentais	5 (heterogêneos)	1 (certificado)	Múltiplos (por norma)	1 (imóvel)
Identidade	DIDs W3C + VC Ed25519	Hash on-chain	EBSI-DID (W3C)	Certificado X.509
Armazenamento	IPFS (Kubo)	On-chain (hash)	Off-chain (variável)	IPFS
Privacidade ZKP	Groth16/BN128 (ZoKrates)	Não implementada	Seletiva	Não implementada
Custo por transação	Sem *gas*	*Gas* Ethereum	Variável por país	Sem *gas*
Metadados	JSON livre (extensível)	Schema fixo	Schema por norma nacional	Schema fixo

Fonte: Elaborado pelo autor (2026), com base nas referências citadas.

Em relação ao OpenCerts (GOVERNMENT TECHNOLOGY AGENCY OF SINGAPORE (GOVTECH), 2020), solução adotada em Singapura que reduziu o tempo de verificação de certificados pelos empregadores de um processo manual de dias para poucos segundos (GOVERNMENT TECHNOLOGY AGENCY OF SINGAPORE (GOVTECH), 2020), o DoChain apresenta duas vantagens principais: a ausência de custo por transação (relevante no contexto de instituições públicas brasileiras com orçamento limitado) e a heterogeneidade documental, que permite atender múltiplos setores (educação, saúde, imóveis) em uma única plataforma. Em contrapartida, o OpenCerts opera sobre *blockchain* pública, eliminando a necessidade de configuração e manutenção de infraestrutura Fabric.

Frente à EBSI (EUROPEAN COMMISSION, 2020), o DoChain apresenta escopo menor em termos de infraestrutura, porém avança na camada de privacidade com a integração de ZK-

SNARKs Groth16 (GROTH, 2016), que oferece garantias formais de privacidade não endereçadas pela especificação atual da EBSI.

Em relação ao sistema de Mukne et al. (2019), cujo contexto de IPFS + Hyperledger Fabric é conceitualmente similar, a principal contribuição do DoChain é a camada de identidade W3C: ao utilizar DIDs e Credenciais Verificáveis (SPORNY et al., 2022b), terceiros podem verificar a autenticidade de um documento sem acesso direto à rede Fabric, o que elimina um ponto de centralização crítico no fluxo de verificação.

Do ponto de vista de desempenho, a latência de verificação de 28,4 ms obtida posiciona o DoChain de forma competitiva para o cenário de verificação presencial, como a validação de um diploma em uma entrevista de emprego ou de uma escritura em um cartório. Comparada ao processo manual atual, que envolve contato com a instituição emissora por telefone ou e-mail e pode levar dias, a verificação em 28,4 ms representa uma melhoria de várias ordens de grandeza.

4.6 Limitações Identificadas

A análise dos resultados permite identificar cinco limitações que devem ser consideradas na interpretação dos dados e endereçadas em trabalhos futuros:

L1 – Ambiente de experimentação único: Todos os experimentos foram conduzidos em uma única máquina física com WSL2, eliminando a latência de rede real entre os peers. Em um ambiente distribuído com nós geograficamente separados, o tempo de endosso tenderia a aumentar na ordem de dezenas a centenas de milissegundos. Os valores de latência reportados constituem um limite inferior para ambientes de produção.

L2 – *Trusted setup* monopartidário: O *trusted setup* Groth16 foi realizado pelo próprio autor, o que significa que o *toxic waste* poderia ser utilizado para forjar provas válidas. Em produção, a cerimônia deveria ser conduzida em protocolo multipartidário (*Multi-Party Computation* – MPC) (BOWE; GABIZON; GREEN, 2017), garantindo que nenhuma das partes individualmente possua o segredo completo.

L3 – Persistência IPFS sem *pinning* descentralizado: O nó IPFS utilizado é local e não está integrado a serviços de persistência de longo prazo como Filecoin ou Pinata. A disponibilidade dos documentos depende da operação contínua do nó local, o que não é adequado para produção.

L4 – Mapeamento incompleto de erros para códigos HTTP (ATK-01, ATK-04, ATK-07): A falta de validação prévia do formato do campo `emissorDominio` resulta em HTTP 500 em vez de HTTP 400 (ATK-04); de forma análoga, o erro `ErrDocumentoJaExiste` do chaincode não é mapeado para HTTP 409 no middleware da API, resultando em HTTP 500 em ID duplicado (ATK-01) e em tentativa de sobrescrita de metadados (ATK-07). Ambos prejudicam a usabilidade sem comprometer a segurança, e as correções são triviais, não requerendo alteração do chaincode.

L5 – Escopo restrito da avaliação de segurança: Os 15 cenários executados cobrem vetores de ataque conhecidos e específicos da aplicação (validação de entrada, controle de acesso, adulteração criptográfica), mas não constituem um teste de penetração ou auditoria de segurança formal. Não foram realizados *fuzzing* sistemático, tentativas de quebra ou manipulação do protocolo de consenso Raft, nem análise de vulnerabilidades na infraestrutura subjacente (Docker, CouchDB, Node.js). Uma avaliação de segurança abrangente, incluindo auditoria externa do chaincode e testes de penetração formais, é recomendada antes de qualquer implantação em produção.

L6 – Verificação criptográfica mockada nos cenários ATK-11 e ATK-12: Conforme discutido na Seção 4.2.1, os testes que envolvem a assinatura Ed25519 da Credencial Verificável e a prova ZK-SNARK Groth16 mockam os módulos `didService` e `zkpService` para isolar a lógica de composição da resposta da API do custo e da complexidade de infraestrutura de executar essas primitivas a cada execução da suíte. Esses cenários comprovam que a API propaga corretamente um veredito de verificação negativo até a resposta final, mas não exercitam, eles próprios, a rejeição criptográfica real de uma assinatura ou prova adulterada. Essa lacuna é parcialmente mitigada pelos testes unitários de `ipfsService` (SHA-256 real, apenas o transporte IPFS é mockado) e pelos *benchmarks* de desempenho (execução real dos contêineres ZoKrates), mas um teste de integração dedicado, que gere um par de chaves Ed25519 real e uma prova Groth16 real, adultere um byte e verifique a rejeição de ponta a ponta sem nenhum mock de criptografia, é uma extensão natural e recomendada para trabalhos futuros.

Adicionalmente, o sistema não implementa rotação de chaves criptográficas dos emissores: caso a chave privada de uma instituição seja comprometida, não há mecanismo automático para invalidar as Credenciais Verificáveis emitidas com aquela chave. Esse aspecto (*key rotation*) é uma extensão natural do padrão W3C DID (SPORNY et al., 2022b) e representa uma das evoluções mais importantes para uma versão de produção do sistema.

5 Considerações Finais

5.1 Conclusão

O presente trabalho teve como objetivo explorar a aplicação da tecnologia *blockchain* para prevenir a falsificação de documentos legais de natureza heterogênea, garantindo maior segurança, escalabilidade, interoperabilidade e integração com sistemas legados. Ao longo do desenvolvimento do estudo, foram analisadas as propriedades fundamentais do *blockchain*, os protocolos de consenso aplicáveis, os padrões de identificação descentralizada W3C, e os requisitos distintos de cada categoria documental considerada: escrituras de imóveis, atestados médicos, diplomas universitários, certidões e carteiras de habilitação (CNH).

Uma contribuição central deste trabalho é a classificação de documentos legais em três níveis de sensibilidade: alta, média e baixa, que fundamenta as decisões de controle de acesso, privacidade e ciclo de vida implementadas no sistema. Essa taxonomia reconhece que documentos de saúde, documentos imobiliários e documentos educacionais impõem requisitos regulatórios e técnicos distintos, e que uma arquitetura eficaz precisa acomodar essa heterogeneidade sem impor ao desenvolvedor a necessidade de construir sistemas paralelos para cada categoria.

A adoção do *Hyperledger Fabric* como plataforma de *blockchain* permissionado, integrado ao IPFS para armazenamento descentralizado, ao padrão W3C de DIDs para identificação verificável e à biblioteca ZoKrates para provas ZK-SNARKs, demonstrou viabilidade técnica no protótipo desenvolvido. O modelo genérico de metadados no *chaincode* com campo `TipoDocumento` e atributos extensíveis por categoria permitiu que os fluxos de emissão, verificação e revogação fossem reutilizados com mínimas adaptações entre as diferentes categorias testadas.

A substituição do *Hyperledger Indy* originalmente previsto para gestão de identidade pelo padrão W3C DID com os métodos `did:web` e `did:key` mostrou-se tecnicamente adequada e academicamente sólida. O padrão W3C oferece maior interoperabilidade, documentação atualizada e compatibilidade com o ecossistema de Credenciais Verificáveis (VCs), que constitui a base para emissão criptograficamente verificável de documentos em ambientes multi-institucionais.

Cabe reconhecer uma tensão conceitual inerente à proposta: o trabalho parte da premissa de reduzir a dependência de intermediários centralizados, mas a arquitetura resultante não elimina essa centralização, apenas a desloca. A validade de uma transação depende do *Membership Service Provider* (MSP) da rede Fabric, controlado pelas organizações participantes, e a identidade das instituições emissoras é ancorada no método `did:web`, que herda a confiança do domínio institucional já estabelecido. Diferentemente de uma *blockchain* pública, na qual a confiança é distribuída entre participantes anônimos por meio do consenso, o DoChain confia nas mesmas

instituições (universidades, cartórios, conselhos profissionais) que já desempenham esse papel no modelo atual. A contribuição do trabalho não está, portanto, em eliminar intermediários, mas em tornar suas atestações auditáveis, imutáveis e verificáveis por terceiros sem contato direto com o emissor, o que já representa um ganho substancial de transparência e rastreabilidade em relação ao processo manual, mesmo sem eliminar a necessidade de confiança institucional subjacente. A escolha por um modelo permissionado, em vez de uma *blockchain* pública, foi deliberada: reflete a necessidade de conformidade regulatória (LGPD, MSP auditável) e de controle de custos operacionais no contexto de instituições públicas brasileiras, que uma rede pública sem permissionamento não ofereceria.

Essa mesma ancoragem na confiança institucional delimita o alcance real da proposta frente ao título do trabalho. O DoChain garante a integridade e a proveniência do registro, isto é, que o que foi gravado no *ledger* não foi alterado após a emissão e que é atribuível de forma não repudiável a uma identidade institucional específica, mas não garante a veracidade do ato de emissão em si. Trata-se do problema do oráculo, inerente a qualquer sistema de *blockchain* que registra fatos do mundo real: se uma instituição legitimamente credenciada emitir um documento fraudulento, um diploma sem o cumprimento da carga horária devida, por exemplo, o DoChain o registrará e o verificará como autêntico indefinidamente, uma vez que a fraude ocorre a montante do sistema, no ato de emissão, e não no registro subsequente. A contribuição do trabalho está em deslocar a superfície de fraude do documento isolado, hoje falsificável de forma anônima e não rastreável, para o emissor, tornando cada emissão atribuível, auditável e associada a uma identidade institucional verificável. Isso reduz significativamente a fraude por adulteração e por falsificação de proveniência, mas não substitui os mecanismos de integridade e fiscalização já responsáveis pela veracidade do ato de emissão na origem.

Desafios relacionados à integração com sistemas legados, especialmente nos setores cartorial e de saúde, ao desempenho da rede sob cargas elevadas e à conformidade integral com a LGPD para documentos de alta sensibilidade foram identificados durante a implementação e devem ser investigados em trabalhos futuros.

Este trabalho estabelece as bases para uma série de investigações futuras. A metodologia, estruturada de forma sequencial e integrada, demonstra a eficácia da abordagem iterativa e incremental, evidenciando que, mesmo diante de desafios técnicos e regulatórios complexos, é possível construir soluções inovadoras que respondam às necessidades contemporâneas de segurança documental em um contexto institucional tão diverso quanto o brasileiro.

5.2 Trabalhos Futuros

Com base nas limitações identificadas e nas perspectivas abertas pelo protótipo, propõem-se as seguintes direções para trabalhos futuros:

1. **Expansão para novos tipos documentais:** A arquitetura modular proposta pode ser estendida para acomodar categorias ainda não implementadas no protótipo, como laudos periciais e contratos de locação, além das cinco categorias já suportadas (diplomas, CNH, escrituras, certidões e atestados médicos). Cada nova categoria exigirá a definição de seu perfil de emissor no MSP, de seus metadados específicos e de suas políticas de ciclo de vida no *chaincode*.

2. **Integração com a ICP-Brasil:** A compatibilidade entre o MSP do *Hyperledger Fabric* e a infraestrutura de chaves públicas brasileira pode ser investigada, viabilizando o uso de certificados digitais já emitidos por Autoridades Certificadoras credenciadas pelo ITI como chaves de emissão no sistema.

3. **Conformidade integral com a LGPD para documentos de saúde:** Documentos classificados como de alta sensibilidade como atestados médicos e laudos exigem tratamento específico sob o Art. 11 da LGPD. Trabalhos futuros podem explorar a fragmentação de dados via *Shamir's Secret Sharing* e o uso de canais privados com políticas de acesso baseadas em consentimento explícito, alinhando o sistema às exigências legais de proteção de dados de saúde.

4. **Aprimoramento dos circuitos ZKP:** O circuito ZK-SNARK implementado neste trabalho é uma prova de conceito deliberadamente simples (Seção 3.2), que atesta a posse de um valor igual a um hash já público no *ledger*, sem calcular o SHA-256 do documento internamente nem vincular a prova ao conteúdo original. O passo seguinte natural é incorporar o próprio cálculo do hash dentro do circuito, de modo que a prova vincule genuinamente o documento sem expor seu hash publicamente; a partir dessa base, circuitos mais complexos poderiam provar, por exemplo, que o emissor possui CRM ativo em determinada especialidade sem revelar o número do registro, ampliando as garantias de privacidade para categorias documentais específicas.

5. **Persistência de longo prazo via Filecoin:** O uso do IPFS local é adequado para o protótipo, mas uma implementação em produção requer estratégias de persistência garantida. A integração com Filecoin ou serviços de *pinning* comerciais asseguraria a disponibilidade dos documentos em horizontes de tempo compatíveis com os exigidos legalmente para escrituras e certidões.

6. **Interface para órgãos verificadores:** O desenvolvimento de interfaces especializadas para perfis verificadores como empregadores, cartórios de segunda instância e operadoras de saúde pode ampliar significativamente a usabilidade do sistema e facilitar sua adoção institucional.

7. **Custódia da chave do titular:** Na versão atual, o par de chaves Ed25519 do `did:key` do titular é gerado de forma efêmera pelo servidor da API (Seção 3.2), o que contraria a noção

de identidade auto-soberana proposta na Seção 3.1.6. Trabalhos futuros devem investigar a geração e a custódia da chave em um dispositivo ou carteira digital (*wallet*) sob controle direto do titular, com o servidor limitado a resolver e verificar identidades, não a gerá-las.

8. **Sessões de validação com usuários:** Este trabalho não conduziu sessões de validação de usabilidade com usuários representativos (por exemplo, emissores institucionais e titulares de documentos); a avaliação restringiu-se aos testes técnicos automatizados e aos experimentos de desempenho e segurança descritos no Capítulo 4. Sessões estruturadas de *feedback* qualitativo sobre os fluxos de emissão e verificação são recomendadas como etapa anterior a qualquer piloto institucional.

A continuidade deste projeto dependerá do monitoramento constante dos resultados obtidos e da abertura para parcerias com instituições de pesquisa, cartórios, conselhos profissionais e órgãos governamentais, viabilizando a adaptação contínua das tecnologias empregadas às demandas do cenário institucional real. Essa integração entre academia, setor público e privado revela-se essencial para transformar o protótipo em uma solução consolidada e de amplo impacto social.

Referências

ANDROULAKI, E. et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In: *Proceedings of the Thirteenth EuroSys Conference*. Porto, Portugal: ACM, 2018. p. 1–15. Evento: EuroSys '18.

ARATA, E.; FERREIRA, A. d. C. Blockchain: Aporte teórico e sugestões de aplicações em diferentes setores. *FatecSeg - Congresso de Segurança da Informação*, v. 1, out. 2021. Acesso em: 08 jul. 2026. Disponível em: <https://www.fatecourinhos.edu.br/fatecseg/index.php/fatecseg/article/view/15>.

BEN-SASSON, E. et al. *Scalable, Transparent, and Post-Quantum Secure Computation with zk-STARKs*. 2018. Acesso em: 25 mar. 2025. Disponível em: <https://eprint.iacr.org/2018/046.pdf>.

BENET, J. *IPFS - Content Addressed, Versioned, P2P File System*. 2014. Acesso em: 25 mar. 2025. Disponível em: <https://arxiv.org/abs/1407.3561>.

BERNSTEIN, D. J. et al. High-speed high-security signatures. *Journal of Cryptographic Engineering*, v. 2, n. 2, p. 77–89, 2012.

BOWE, S.; GABIZON, A.; GREEN, M. D. *A Multi-party Protocol for Constructing the Public Parameters of the Pinocchio zk-SNARK*. 2017. Cryptology ePrint Archive, Report 2017/602. Disponível em: <https://eprint.iacr.org/2017/602>. Acesso em: 25 maio 2026.

BRASIL. *Medida Provisória nº 2.200-2, de 24 de agosto de 2001 — Institui a Infraestrutura de Chaves Públicas Brasileira (ICP-Brasil)*. 2001. <https://www.planalto.gov.br/ccivil_03/mpv/antigas_2001/2200-2.htm>. Acesso em: 06 jul. 2026.

BRASIL. *Lei nº 13.709, de 14 de agosto de 2018 (Lei Geral de Proteção de Dados Pessoais - LGPD)*. 2018. Diário Oficial da União. Acesso em: 06 jul. 2026. Disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm>.

BRUNNER, C. *Eduthereum: A System for Storing Educational Certificates in a Public Blockchain*. Dissertação (Mestrado) — Universität Innsbruck, 2017. Acesso em: 06 jul. 2026. Disponível em: <https://informationsecurity.uibk.ac.at/theses/eduthereum/>.

BUTERIN, V. *Ethereum Whitepaper: A Next-Generation Smart Contract and Decentralized Application Platform*. 2014. Acesso em: 25 mar. 2025. Disponível em: <https://ethereum.org/en/whitepaper/>.

CASTRO, M.; LISKOV, B. Practical byzantine fault tolerance. In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI)*. [s.n.], 1999. p. 173–186. Acesso em: 08 jul. 2026. Disponível em: <https://css.csail.mit.edu/6.824/2014/papers/castro-practicalbft.pdf>.

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

COINTELEGRAPH. *University of Bahrain Will Issue Diplomas on the Blockchain Employing Blockcerts*. 2019. Acesso em: 06 jul. 2026. Disponível em: <https://cointelegraph.com/news/university-of-bahrain-will-issue-diplomas-on-the-blockchain-employing-blockcerts>.

COOPER, D. et al. *RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. [S.l.], 2008. Acesso em: 08 jul. 2026. Disponível em: <https://tools.ietf.org/html/rfc5280>.

DESHPANDE, P. M. et al. The mask of zorro: Preventing information leakage from documents. *Knowledge and Information Systems*, Springer, v. 45, n. 3, p. 705–730, 2015.

EBERHARDT, J.; TAI, S. ZoKrates: Scalable Privacy-Preserving Off-Chain Computations. In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. [S.l.]: IEEE, 2018. p. 1084–1091.

EUROPEAN COMMISSION. *European Blockchain Services Infrastructure (EBSI)*. Bruxelas, 2020. Acesso em: 06 jul. 2026. Disponível em: <https://ec.europa.eu/digital-building-blocks/sites/spaces/EBSI/pages/447687044/Home>.

GOLDWASSER, S.; MICALI, S.; RACKOFF, C. The knowledge complexity of interactive proof-systems. In: *Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC)*. [S.l.: s.n.], 1985. p. 291–304.

GOVERNMENT OF SINGAPORE. *HealthCerts: Open Standard for Verification of COVID-19 Test Results*. 2021. Informação disponível online. Acesso em: 01 mar. 2025. Disponível em: <https://opengovasia.com/2021/03/02/singapore-government-uses-blockchain-to-verify-covid-19-test-results/>.

GOVERNMENT TECHNOLOGY AGENCY OF SINGAPORE (GOVTECH). *Losing Your Educational Certificates Is Now a Thing of the Past*. 2020. GovTech Singapore Tech News. Acesso em: 08 jul. 2026. Disponível em: <https://www.tech.gov.sg/technews/losing-your-educational-certs-a-thing-of-the-past/>.

GROTH, J. On the Size of Pairing-Based Non-Interactive Arguments. In: *Advances in Cryptology – EUROCRYPT 2016*. [S.l.]: Springer, Berlin, Heidelberg, 2016. (Lecture Notes in Computer Science, v. 9666), p. 305–326.

KOSBA, A. et al. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: *2016 IEEE Symposium on Security and Privacy (SP)*. [S.l.: s.n.], 2016. p. 839–858.

LARIMER, D. *Delegated Proof of Stake (DPoS)*. 2014. White Paper. Acesso em: 16 mar. 2025. Disponível em: <https://bitshares.org/technology/delegated-proof-of-stake/>.

LINUX FOUNDATION. *Hyperledger Fabric Documentation*. [S.l.], 2021. Acesso em: 15 fev. 2025. Disponível em: <https://hyperledger-fabric.readthedocs.io/>.

MINISTÉRIO DA EDUCAÇÃO (MEC); REDE NACIONAL DE ENSINO E PESQUISA (RNP). *Diploma Digital: agilidade e segurança para alunos do Ensino Superior*. 2021. RNP. Acesso em: 08 jul. 2026. Disponível em: <https://www.rnp.br/noticias/diploma-digital>.

MUKNE, H. et al. Land record management using hyperledger fabric and ipfs. In: *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. [S.l.]: IEEE, 2019. p. 1–8. ISBN 978-1-7281-1261-9.

NAKAMOTO, S. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. Acesso em: 20 dez. 2024. Disponível em: <https://bitcoin.org/bitcoin.pdf>.

NARAYANAN, A. et al. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton, NJ: Princeton University Press, 2016.

ONGARO, D.; OUSTERHOUT, J. In search of an understandable consensus algorithm. In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, 2014. p. 305–319. Disponível em: <https://raft.github.io/raft.pdf>.

SPORNY, M. et al. *Verifiable Credentials Data Model v1.1*. [S.l.], 2022. Acesso em: 25 maio 2026. Disponível em: <https://www.w3.org/TR/vc-data-model/>.

SPORNY, M. et al. *Decentralized Identifiers (DIDs) v1.0*. [S.l.], 2022. Acesso em: 06 jul. 2026. Disponível em: <https://www.w3.org/TR/did-1.0/>.

STOLL, C.; KLAASSEN, L.; GALLERSDÖRFER, U. The carbon footprint of bitcoin. *Joule*, Elsevier, v. 3, n. 7, p. 1647–1661, 2019.

TERBU, O. et al. *did:web Method Specification*. [S.l.], 2023. Acesso em: 08 jul. 2026. Disponível em: <https://w3c-ccg.github.io/did-method-web/>.

TSCHORSCH, F.; SCHEUERMANN, B. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys & Tutorials*, IEEE, v. 18, n. 3, p. 2084–2123, 2016.

VUKOLIĆ, M. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In: *Open Problems in Network Security*. [S.l.]: Springer, 2016. p. 112–125.

W3C CREDENTIALS COMMUNITY GROUP. *The did:key Method v0.9*. [S.l.], 2023. W3C Community Group Draft. Acesso em: 08 jul. 2026. Disponível em: <https://w3c-ccg.github.io/did-key-spec/>.