Sistema de gestão de senhas com armazenamento distribuído baseado em *blockchain*

Luis Fernando Vicarri[1], **Maria Lorena Muralha**[1], **Robson Costa**[1], **Glaidson Menegazzo Verzeletti**[1]

[1]Instituto Federal de Santa Catarina (IFSC)
Rua Heitor Vila Lobos, 225 – Bairro São Francisco – 88506-400 – Lages – SC – Brasil

luis.v01@aluno.ifsc.edu.br, maria.ml2004@aluno.ifsc.edu.br

robson.costa@ifsc.edu.br, glaidson.verzeletti@ifsc.edu.br

Abstract. *Traditional password managers rely on centralized architectures that create a single point of failure in credential storage, making them vulnerable to large-scale data breaches. This work proposes a password management architecture that mitigates this risk by distributing encrypted fragments across independent infrastructures. The developed solution encrypts credentials locally and then fragments the ciphertext into two independent parts, which are stored in two private blockchains. This fragmentation model mitigates the single point of failure associated with centralized credential storage, increases resilience, and reduces user dependency on a single trusted repository, improving control when compared to traditional centralized approaches.*

Resumo. *Gerenciadores de senhas tradicionais adotam arquiteturas centralizadas, o que cria um ponto único de falha no armazenamento de credenciais e os torna vulneráveis a vazamentos massivos de dados. Este trabalho propõe uma arquitetura de gestão de senhas que mitiga esse risco ao distribuir fragmentos cifrados entre infraestruturas independentes. A solução desenvolvida criptografa as credenciais localmente e, em seguida, fragmenta o dado cifrado em duas partes autônomas, armazenadas em duas blockchains privadas. Esse modelo de fragmentação mitiga o ponto único de falha associado ao armazenamento centralizado de credenciais, aumenta a resiliência e reduz a dependência do usuário de um repositório único de confiança, ampliando o controle em relação às abordagens centralizadas tradicionais.*

1. Introdução

A crescente conectividade da vida moderna tornou a gestão de senhas um aspecto crítico da segurança digital. Estudos demonstram que manter um grande número de credenciais fortes e atualizadas regularmente é uma tarefa desafiadora para a maioria dos usuários, o que compromete a conformidade com boas práticas de segurança. Essa dificuldade é evidenciada por Choong et al. (2014), que, ao estudarem funcionários do governo dos Estados Unidos, observaram comportamentos recorrentes como reutilização de senhas, uso de anotações físicas e esquecimento frequente de credenciais — práticas que aumentam o risco de exposição. Nesse contexto, surgem os gerenciadores de senhas, sistemas projetados para armazenar, organizar e gerar senhas de maneira segura, geralmente por meio de ambientes criptografados. Contudo, apesar da conveniência proporcionada por

essas ferramentas, a arquitetura predominante utilizada por elas baseia-se em modelos de armazenamento centralizado, introduzindo um ponto único de falha no repositório central de credenciais, cujo comprometimento pode ser mais prejudicial do que o de uma única conta de aplicativo (Smith et al., 2018).

Os gerenciadores de senhas mais modernos, como Bitwarden[1] e NordPass[2], têm adotado soluções baseadas em computação em nuvem para seus *backends*, com o objetivo de garantir redundância de dados, escalabilidade e facilidade na implementação de políticas de *backup*. No entanto, embora essa abordagem ofereça vantagens operacionais significativas, seu uso como infraestrutura centralizada para aplicativos móveis tem sofrido com vulnerabilidades críticas, incluindo configurações incorretas na nuvem, controles de acesso inadequados e fragilidades na arquitetura dos provedores de serviços em nuvem (Alanoud et al., 2024).

Nos últimos anos, incidentes envolvendo vazamentos massivos de dados armazenados em serviços de nuvem revelaram falhas na proteção de registros altamente sensíveis — incluindo informações pessoais, registros médicos e dados corporativos confidenciais. Casos como o da Oracle (Kovacs, 2025), o da 23andMe (Blount, 2025) e o vazamento massivo da CAM4 (TeamPassword, 2021) ilustram a gravidade dessas vulnerabilidades.

Estudos recentes indicam que bancos de dados inseguros utilizados por aplicativos móveis resultaram na exposição de aproximadamente 280 milhões de registros sensíveis de usuários, contendo dados como credenciais de acesso e informações pessoais (Zuo et al., 2019). Tais ocorrências evidenciam que a centralização da informação em estruturas de armazenamento na nuvem representa uma das principais vulnerabilidades dos sistemas atuais, uma vez que concentra grandes volumes de dados sensíveis em um único repositório, tornando-os alvos atrativos para agentes maliciosos.

Os ataques a credenciais constituem uma ameaça significativa tanto para usuários comuns quanto para grandes corporações. Relatórios em cibersegurança, como o *Verizon Data Breach Investigations Report* (2023), o comunicado oficial da IBM sobre o relatório *Cost of a Data Breach Report 2024* (IBM Newsroom, 2024) e o ENISA *Threat Landscape Report* (2023), apontam que grande parte das violações de dados está diretamente relacionada ao comprometimento de senhas, resultando em prejuízos financeiros, vazamento de informações sensíveis e roubo de identidade (Verizon Business, 2023; IBM, 2024; European Union Agency for Cybersecurity (ENISA), 2023). Segundo o *Verizon Data Breach Investigations Report*, aproximadamente 49% das violações de dados analisadas envolveram o uso de credenciais roubadas, evidenciando a fragilidade dos sistemas atuais frente a esse tipo de ameaça.

Nesse cenário, modelos de armazenamento distribuído surgem como alternativas promissoras para mitigar riscos associados à centralização. Don Tapscott e Alex Tapscott destacam que tecnologias inspiradas em *blockchain* podem reduzir a dependência de repositórios centralizados e mitigar riscos inerentes a bancos de dados tradicionais (Tapscott e Tapscott, 2016). Embora sua aplicação prática varie conforme o tipo de rede e o nível de controle existente, o princípio de distribuição de dados contribui para reduzir a atratividade e o impacto de ataques direcionados a um único ponto.

[1] `https://bitwarden.com`
[2] `https://nordpass.com`

A tecnologia *blockchain* destaca-se por permitir o armazenamento distribuído e imutável de informações, reduzindo significativamente a possibilidade de um ponto único de falha no armazenamento. Esse modelo aumenta a segurança contra ataques direcionados ao repositório de credenciais, uma vez que os dados não ficam concentrados em um único servidor. Além disso, o uso de técnicas criptográficas sofisticadas impede a exposição direta de senhas, reforçando a proteção das informações tanto para usuários quanto para organizações.

Diante desse cenário, o objetivo geral deste trabalho é desenvolver um gerenciador de senhas baseado em um modelo de armazenamento distribuído, utilizando a tecnologia *blockchain* para elevar os níveis de segurança, integridade e confiabilidade do armazenamento. A proposta visa mitigar vulnerabilidades presentes em sistemas centralizados, frequentemente exploradas em ataques cibernéticos, ao distribuir fragmentos de dados cifrados entre duas infraestruturas independentes. O sistema permite o gerenciamento seguro de credenciais de acesso, reduzindo a dependência de um único *backend* centralizado e aumentando a autonomia do usuário em relação a modelos tradicionais.

Para alcançar o objetivo geral, foram definidos os seguintes objetivos específicos:

- Estudar os fundamentos da tecnologia *blockchain* e sua aplicação em sistemas de armazenamento distribuído;
- Realizar o levantamento de requisitos de segurança e privacidade para o gerenciamento de senhas, identificando as linguagens de programação e *frameworks* mais adequados às necessidades do projeto;
- Desenvolver a solução proposta, contemplando desde o projeto dos módulos (interface, gerenciador de senhas, *blockchain* e banco de dados) até a implementação de um protótipo funcional;
- Avaliar o desempenho do protótipo por meio de testes funcionais, de desempenho e de segurança, executados em ambiente controlado para comprovar sua viabilidade em cenários simulados.

Quanto à classificação metodológica, este trabalho caracteriza-se como uma pesquisa aplicada, com abordagem quali–quantitativa, pois combina análise qualitativa da arquitetura proposta com a coleta e interpretação de métricas quantitativas obtidas experimentalmente. Em relação aos objetivos, enquadra-se como pesquisa exploratória e descritiva.

Do ponto de vista dos procedimentos técnicos, o estudo integra pesquisa bibliográfica, desenvolvimento tecnológico de um protótipo funcional e experimentação em ambiente controlado.

O trabalho foi desenvolvido em quatro etapas principais. Como ponto de partida, foi realizada uma revisão bibliográfica abrangente sobre os temas de segurança da informação, *blockchain*, gerenciamento de senhas e armazenamento distribuído, fundamentada em livros, artigos científicos e relatórios técnicos obtidos em bases acadêmicas e fontes confiáveis.

Em seguida, foram conduzidas duas frentes de trabalho: a identificação dos requisitos de segurança e privacidade para o gerenciamento de senhas e a seleção das tecnologias utilizadas, considerando diretrizes de boas práticas da área, bem como critérios de desempenho, compatibilidade e segurança. Também foram utilizados recursos de mode-

lagem para representar as funcionalidades e a estrutura do sistema.

A terceira etapa correspondeu ao desenvolvimento do protótipo funcional, abrangendo a implementação dos principais módulos do sistema, incluindo a interface do usuário, o gerenciador de senhas, a estrutura de *blockchain* e o mecanismo de armazenamento distribuído, seguindo os requisitos previamente definidos.

Por fim, na etapa de testes, foram realizados experimentos em ambiente controlado, contemplando testes funcionais, de desempenho e de segurança e resiliência. Os resultados foram analisados por meio de quadros comparativos, com o objetivo de avaliar a aderência da solução aos requisitos definidos e sua viabilidade técnica em cenários simulados.

Além desta introdução, este trabalho está organizado da seguinte forma: na Seção 2, é apresentado o referencial teórico, com os principais conceitos sobre segurança da informação, gerenciamento de senhas, *blockchain* e armazenamento distribuído. A Seção 3 descreve a solução proposta, abordando a arquitetura do sistema, as tecnologias utilizadas e os procedimentos de implementação. Na Seção 4, são apresentados os testes realizados e a análise dos resultados obtidos. Por fim, a Seção 5 traz as conclusões do estudo, bem como sugestões para trabalhos futuros.

2. Referencial teórico

Esta seção apresenta uma visão geral das soluções existentes para o gerenciamento de credenciais, contemplando tanto os modelos centralizados quanto as alternativas distribuídas. Como parte fundamental da proposta deste trabalho, introduz-se inicialmente o conceito de *blockchain*, a fim de fornecer os fundamentos necessários para compreender as soluções que utilizam essa tecnologia em sua arquitetura.

2.1. *Blockchain*

A tecnologia *blockchain* tem sido amplamente aplicada em sistemas distribuídos de armazenamento e autenticação, tornando-se relevante para o contexto deste trabalho. Nesta subseção, são apresentados seus conceitos centrais, o funcionamento técnico, as características fundamentais e as principais variações existentes, estabelecendo o embasamento necessário para a compreensão das soluções baseadas em *blockchain*.

2.1.1. Origem da tecnologia

A tecnologia *blockchain* emergiu como uma das inovações mais relevantes da computação nas últimas décadas, oferecendo uma nova forma de registrar e validar transações digitais sem a necessidade de intermediários. Inicialmente proposta por Nakamoto (2008) como a base do Bitcoin, essa estrutura de dados distribui informações em blocos encadeados e protegidos por técnicas criptográficas, formando um livro-razão imutável, descentralizado e auditável. Ao reduzir a dependência de uma autoridade central — especialmente em implementações públicas — a *blockchain* permite que múltiplas partes mantenham consenso sobre o estado dos dados de maneira transparente e resistente a alterações maliciosas (Wood, 2014).

Com o amadurecimento da tecnologia, seu uso passou a abranger aplicações além de criptomoedas, incluindo rastreabilidade de cadeias de suprimento, autenticação de

identidade, contratos inteligentes e o fortalecimento da segurança em sistemas de armazenamento distribuído e gerenciamento de credenciais (Zheng et al., 2017).

2.1.2. Contratos inteligentes

Contratos inteligentes são programas autoexecutáveis armazenados e executados em uma *blockchain*, operando de forma determinística e automatizada conforme condições previamente definidas são atendidas (Christidis e Devetsikiotis, 2016). Szabo (1994) os define como "protocolos de transações que executam os termos de um contrato", de forma confiável e sem a necessidade de intermediários.

Essa tecnologia permite a criação de regras automáticas — por exemplo: "se A entregar o serviço X, o pagamento é liberado" — reduzindo custos operacionais e eliminando etapas manuais (Kosba et al., 2016). Por estarem integrados à *blockchain*, contratos inteligentes herdam propriedades como imutabilidade, auditabilidade e execução confiável, discutidas nas seções posteriores (Zheng et al., 2017).

Apesar de seu potencial, os contratos inteligentes apresentam limitações importantes. A execução é estritamente determinada pelo código implantado, o que reduz sua capacidade de lidar com exceções complexas ou interpretar nuances jurídicas sem mecanismos externos (Atzei et al., 2017). Além disso, muitos contratos dependem de oráculos — serviços que fornecem dados externos, como preços ou eventos do mundo real — introduzindo novos pontos de confiança no sistema (Gudgeon et al., 2019). Esses componentes exigem auditorias rigorosas, especificação cuidadosa e mecanismos complementares para garantir segurança e confiabilidade em cenários reais.

2.1.3. Funcionamento geral

Do ponto de vista técnico, a *blockchain* opera como uma cadeia de blocos criptograficamente interligados, na qual cada bloco contém um conjunto de transações e o *hash* do bloco anterior. Essa vinculação sequencial impede alterações retroativas sem que todo o histórico subsequente também seja modificado, o que tornaria qualquer tentativa de adulteração facilmente detectável (Narayanan et al., 2016).

A segurança da rede é garantida por mecanismos de consenso distribuído, como o *Proof of Work* (PoW) — utilizado no Bitcoin — e o *Proof of Stake* (PoS), empregado por plataformas como o *Ethereum 2.0*. Esses mecanismos validam transações e registram blocos sem depender de uma única autoridade central nas redes públicas; nas redes privadas, esse consenso é estabelecido entre entidades previamente autorizadas, mantendo a integridade e a resistência a ataques (Bonneau et al., 2015).

Cada nó da rede mantém uma cópia integral da cadeia, assegurando transparência e resiliência mesmo diante de falhas ou ataques direcionados a nós individuais. A Figura 1 ilustra essa dinâmica, destacando a ligação entre os blocos e a distribuição dos nós na rede.

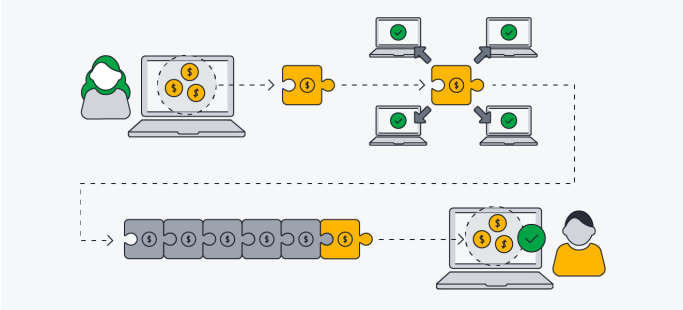

Figura 1. Rede *blockchain*. Fonte: AVG Technologies (2022).

2.1.4. Características fundamentais

A tecnologia *blockchain* apresenta um conjunto de características técnicas que a tornam adequada para aplicações que exigem integridade, transparência e resistência a falhas, entre as quais destacam-se:

- **Descentralização**: em *blockchains* públicas, não há um servidor central responsável pelo controle das transações. Em redes privadas, embora exista governança e controle de acesso, a validação permanece distribuída entre múltiplos nós, reduzindo a dependência de um único ponto operacional e aumentando a resistência a falhas e manipulações (Crosby et al., 2016).
- **Imutabilidade**: após registrados, os dados não podem ser modificados sem alterar os blocos subsequentes e obter o consenso da maioria dos nós, tornando a adulteração de registros altamente inviável na prática (Narayanan et al., 2016).
- **Transparência e auditabilidade**: em *blockchains* públicas, as transações ficam registradas de forma verificável, permitindo auditoria independente e ampliando a confiança no sistema (Zheng et al., 2017). Em redes privadas, o acesso pode ser restrito conforme políticas organizacionais.
- **Segurança criptográfica**: a integridade dos dados e a autenticação das transações são garantidas por técnicas como criptografia assimétrica e funções *hash*, assegurando que apenas os detentores das chaves privadas possam autorizar operações (Bonneau et al., 2015).

2.1.5. Classificações de *blockchain*

Embora originalmente concebida como uma rede pública e aberta, a tecnologia *blockchain* evoluiu para atender a diferentes necessidades organizacionais e operacionais, podendo ser classificada em três tipos principais:

- **Pública**: permite que qualquer usuário participe da rede, envie transações e valide blocos sem necessidade de permissão prévia. Exemplos incluem as *blockchains* do Bitcoin e do Ethereum. Esse tipo privilegia transparência e resistência à censura, embora apresente maior latência e custos operacionais (Buterin, 2014).
- **Privada**: restringe o acesso a um conjunto definido de participantes autorizados, geralmente sob controle de uma única organização ou consórcio. É utilizada quando há necessidade de maior desempenho, confidencialidade e governança

centralizada, como em soluções internas de auditoria ou rastreamento (Zheng et al., 2017).

- **Híbrida (ou permissionada)**: combina características das redes públicas e privadas. O acesso à rede e à validação pode ser restrito, enquanto determinados dados ou provas criptográficas podem ser disponibilizados para fins de transparência e auditabilidade. Essa abordagem é comum em sistemas corporativos que exigem controle granular de acesso entre organizações distintas, como ocorre no Hyperledger Fabric (Zheng et al., 2017).

2.1.6. Aplicações em segurança da informação

A versatilidade da tecnologia *blockchain* tem permitido sua adoção em diversas áreas que demandam segurança, transparência e resistência a fraudes. Na segurança da informação, destaca-se seu uso para garantir a integridade de arquivos e registros, assegurar a rastreabilidade de transações digitais e viabilizar sistemas de identidade descentralizada (*Decentralized Identity*) (Zyskind et al., 2015; Li et al., 2020). Nesses sistemas, a autenticação e o controle de acesso são realizados por meio de chaves criptográficas, reduzindo a dependência de autoridades centralizadas e ampliando a autonomia do usuário sobre suas credenciais.

Essas aplicações são particularmente relevantes em cenários que exigem proteção reforçada de credenciais e mecanismos confiáveis de autenticação. Soluções modernas de gerenciamento de senhas, controle de acesso e registro de eventos sensíveis podem se beneficiar da imutabilidade e auditabilidade proporcionadas pela *blockchain*, que atua como uma camada adicional de confiança ao registrar operações críticas de forma rastreável e resistente a adulterações.

2.1.7. Desafios e limitações

Apesar de seu potencial, a tecnologia *blockchain* enfrenta desafios relevantes em aplicações de segurança. Redes públicas podem apresentar custos elevados de transação (*gas fees*) e latência significativa, o que inviabiliza o registro frequente de operações sensíveis, como atualizações de senhas (Buterin, 2014). Além disso, vulnerabilidades em contratos inteligentes podem ser exploradas por atacantes. Um exemplo clássico ocorreu em 2016, com o ataque ao DAO — um *Decentralized Autonomous Organization* em Ethereum — quando exploradores abusaram de uma vulnerabilidade de reentrância, chamando repetidamente uma função antes que seu estado fosse atualizado, drenando cerca de US$ 50 milhões em *Ether* (Atzei et al., 2017; Siegel, 2016).

Outro desafio é o estouro de gás (*out-of-gas*). No Ethereum, cada operação consome uma quantidade limitada de gás, definida para evitar *loops* infinitos ou cálculos excessivos. Caso esse limite seja ultrapassado, a transação é revertida, mas o custo computacional ainda é cobrado, o que pode interromper serviços e gerar custos imprevistos (Wood, 2014).

Além disso, o modelo de segurança da *blockchain* exige controle rigoroso das chaves privadas. A perda dessas chaves implica a impossibilidade definitiva de recuperar

dados protegidos (Zyskind et al., 2015), e a necessidade de lidar com carteiras, transações e conceitos de criptoeconomia pode representar uma barreira de adoção para usuários menos experientes (Gudgeon et al., 2019).

Esses fatores indicam que, embora a *blockchain* possa reforçar a integridade e a auditabilidade de sistemas distribuídos, sua adoção requer planejamento cuidadoso, desenvolvimento seguro e auditoria contínua.

2.2. Soluções centralizadas

Soluções centralizadas para gerenciamento de senhas armazenam credenciais sob o controle de uma entidade central, responsável pela autenticação, sincronização e proteção das informações. Embora ofereçam conveniência e integração entre dispositivos, esses sistemas concentram confiança e governança em um único ponto, o que pode se tornar um ponto único de falha (Bonneau et al., 2012).

Os gerenciadores de senhas — aplicações que armazenam, geram e preenchem automaticamente credenciais — podem ser classificados em três categorias principais: (1) **tradicionais baseados em nuvem**, (2) **integrados a navegadores** e (3) **corporativos** (Li et al., 2014).

2.2.1. Gerenciadores de senhas tradicionais

O armazenamento em nuvem é um modelo de computação que possibilita o armazenamento de dados e arquivos em uma rede privada dedicada ou por meio de provedores acessados pela Internet pública (Amazon Web Services, 2024), como a *Amazon Web Services* (AWS), *Microsoft Azure* ou *Google Cloud*.

A Figura 3 do Anexo A.1 apresenta um exemplo simplificado dessa arquitetura, ilustrando a centralização do armazenamento em servidores de nuvem e a interconexão entre clientes e provedores.

Apesar de sua ampla adoção, esse modelo apresenta vulnerabilidades estruturais, entre as quais se destacam:

1. **Ponto único de falha**: a centralização das credenciais torna o servidor um alvo de alto valor. Embora a criptografia reduza significativamente o risco de acesso não autorizado, uma violação do ambiente do provedor ou o comprometimento da senha mestra pode expor todas as credenciais associadas (Silverback Consulting, 2025).
2. **Armazenamento de dados sensíveis em memória**: segundo pesquisa conduzida por Chatzoglou et al. (2024), 75% dos gerenciadores de senhas para *desktop* e 83% das extensões de navegador armazenam senhas em texto plano na memória do sistema durante o uso, elevando o risco de vazamento em caso de comprometimento do dispositivo (Chatzoglou et al., 2024).
3. **Dependência de servidores de terceiros**: a utilização de infraestrutura operada por terceiros adiciona riscos operacionais à segurança dos dados, pois falhas, configurações inadequadas ou comprometimentos podem afetar simultaneamente um grande número de usuários (Silverback Consulting, 2025).

Gerenciadores como LastPass[3], 1Password[4] e Dashlane[5] utilizam criptografia local para proteger os cofres de senhas antes de transmiti-los aos servidores centrais (Bonneau et al., 2012; Li et al., 2014). Apesar de essa abordagem oferecer um nível relevante de segurança, incidentes como o ocorrido com o LastPass em 2022 evidenciam limitações nos mecanismos auxiliares de segurança e recuperação.

Nesse incidente, embora os cofres estivessem criptografados, metadados essenciais — como nomes de usuário, URLs e estrutura das pastas — permaneceram armazenados em texto claro e foram expostos (Demartini, 2022). Além disso, os invasores obtiveram acesso a dados de recuperação de contas e a cópias criptografadas dos cofres, ampliando o risco para usuários com senhas mestras fracas, suscetíveis a ataques de força bruta *offline* (Greenberg, 2022).

Esses fatores evidenciam que, apesar dos avanços técnicos, o modelo centralizado em nuvem permanece um vetor relevante de risco. Configurações incorretas em serviços de nuvem figuram entre as vulnerabilidades mais recorrentes, frequentemente resultando em acesso não autorizado e violações de dados (Rathod et al., 2025).

2.2.2. Gerenciadores integrados em navegadores

Soluções nativas de navegador, como o Google Password Manager[6] e o iCloud Keychain[7], oferecem sincronização automática de credenciais vinculadas à conta principal do usuário. Mesmo com suporte a autenticação multifator e criptografia ponta a ponta, a segurança dessas senhas depende diretamente da robustez da conta mestra, cujo comprometimento pode resultar na exposição de todos os registros armazenados (Jackson e Boneh, 2014).

A Figura 4 do Anexo A.2 apresenta uma representação simplificada do fluxo de funcionamento desses gerenciadores nativos, destacando a sincronização entre o navegador, a conta do usuário e os serviços em nuvem.

Embora proporcionem uma experiência de autenticação simplificada e integração nativa com os sistemas operacionais, os gerenciadores de senhas embutidos em navegadores apresentam limitações significativas em termos de segurança. A proteção das credenciais está atrelada à segurança geral do navegador e do dispositivo utilizado, o que pode expor os dados em ambientes mal configurados ou comprometidos por *malware*.

Um estudo conduzido por pesquisadores da Universidade do Tennessee e da Universidade Brigham Young (Gautam et al., 2024) analisou vulnerabilidades associadas a gerenciadores de senhas baseados em navegador, identificando três tipos de ataques:

1. **Preenchimento automático**: quando campos de *login* são preenchidos sem interação explícita do usuário, um site comprometido pode capturar a senha apenas com a visita à página.

[3] `https://www.lastpass.com/`
[4] `https://1password.com/`
[5] `https://www.dashlane.com/`
[6] `https://passwords.google.com/`
[7] `https://support.apple.com/pt-br/HT204085`

2. **Ataques baseados em manipulação do DOM**: um adversário com controle sobre o *Document Object Model* (DOM) pode interceptar ou modificar campos de senha antes que o usuário perceba, obtendo acesso às credenciais armazenadas.
3. **Ataques de *phishing***: ao simular interfaces legítimas e controlar o servidor receptor, atacantes podem induzir o usuário a fornecer suas credenciais, interceptando sua transmissão.

Essas vulnerabilidades ressaltam que, embora convenientes, os gerenciadores integrados aos navegadores devem ser utilizados com cautela, especialmente em contextos que exijam elevados padrões de segurança.

2.2.3. Soluções corporativas

Em ambientes corporativos, ferramentas como o Vault[8] da HashiCorp (modo centralizado) e o Credential Manager[9] do Active Directory proporcionam controle de acesso e auditoria de credenciais (HashiCorp, 2019). Entretanto, a concentração de privilégios e dados em um único servidor mantém vivo o risco de ponto único de falha, seja por invasões ou por uso indevido de credenciais administrativas.

A Figura 5 do Anexo A.3 apresenta uma visão conceitual desse modelo centralizado, evidenciando a concentração de credenciais em um servidor único que fornece autenticação e acesso a diversos usuários e sistemas corporativos.

Apesar dos avanços em criptografia, controle de permissões e registro de atividades, o modelo centralizado ainda apresenta uma vulnerabilidade estrutural relevante: a concentração de privilégios e dados sensíveis em um único ponto da arquitetura. Embora esses gerenciadores melhorem significativamente a usabilidade e a gestão de credenciais, introduzem um ponto único de falha, no qual uma invasão pode ter consequências mais graves do que o comprometimento isolado de uma única conta (Gautam et al., 2024), resultando em exposição massiva de informações pessoais e sensíveis.

Nesse modelo, os usuários tendem a ter menor controle sobre a infraestrutura e o fluxo de armazenamento das credenciais, uma vez que uma autoridade central governa o ambiente e define as políticas de acesso, mesmo quando a criptografia local impede o acesso direto ao conteúdo (Team, 2024). Além disso, a dependência de um único servidor implica riscos operacionais relevantes, como falhas internas ou indisponibilidades que podem afetar diretamente operações críticas. Segundo o relatório anual da IBM Security, o tempo médio para conter uma violação de dados é de 277 dias, e os custos médios globais superam 4 milhões de dólares (IBM Security, 2023).

A gestão de senhas centralizadas também depende fortemente da adoção de práticas rigorosas de segurança, como rotação frequente de credenciais, divisão de privilégios administrativos (princípio do menor privilégio) e múltiplas camadas de autenticação. Sem essas medidas, o risco associado ao ponto único de falha torna-se ainda mais pronunciado.

[8] `https://www.vaultproject.io/`

[9] `https://learn.microsoft.com/en-us/windows/security/identity-protection/credential-manager-concept`

Assim, embora soluções corporativas centralizadas ofereçam maior controle administrativo e usabilidade, sua adoção exige uma abordagem contínua de mitigação de riscos e reforço de boas práticas operacionais.

2.3. Soluções descentralizadas

Soluções descentralizadas para gerenciadores de senhas armazenam credenciais de forma distribuída, eliminando a dependência de um servidor central e, em alguns casos, fragmentando informações entre múltiplos nós. De acordo com Tanenbaum, um sistema distribuído é aquele que opera sobre uma coleção de máquinas independentes, sem memória compartilhada, mas que se apresentam aos usuários como um único sistema coerente (Tanenbaum e Bos, 2014). Essa arquitetura mitiga os riscos associados à centralização de dados — especialmente o ponto único de falha no armazenamento —, aumentando a segurança, a resiliência operacional e a tolerância a falhas, mesmo diante da indisponibilidade de alguns nós (LogaP, 2024).

As soluções descentralizadas podem ser classificadas conforme suas abordagens tecnológicas em três categorias principais: (1) **gerenciadores com sincronização *Peer-to-Peer* (P2P)**, (2) **gerenciadores baseados em *hardware***, como *tokens* físicos ou chaves USB, e (3) **gerenciadores baseados em *blockchain***.

2.3.1. Gerenciadores com sincronização *Peer-to-Peer*

As redes *Peer-to-Peer* (P2P) são amplamente utilizadas na Internet por permitirem o compartilhamento direto de recursos computacionais entre usuários, sem a necessidade de servidores centrais (Androutsellis-Theotokis e Spinellis, 2004). Essas redes são compostas por múltiplos nós distribuídos geograficamente, que operam de forma autônoma e independente, mas interconectados em uma estrutura colaborativa. Diferentemente do modelo centralizado, cada nó pode atuar simultaneamente como cliente e servidor, promovendo maior descentralização e resiliência na comunicação (Pereira, 2004).

No contexto do gerenciamento de senhas, arquiteturas *Peer-to-Peer* baseiam-se na comunicação direta entre os dispositivos do usuário, eliminando a necessidade de servidores centrais para sincronização. Nesse modelo, as credenciais são criptografadas localmente e compartilhadas apenas entre dispositivos previamente autorizados, assegurando que somente o próprio usuário, ou dispositivos sob seu controle, tenha acesso às informações armazenadas (JumpCloud, 2022).

Apesar de soluções como o Web3Password[10] e o KryptPass[11] mitigarem o ponto único de falha associado à dependência de servidores centrais, esse modelo apresenta vulnerabilidades próprias que podem comprometer a segurança das credenciais, como:

1. **Exposição de dados pessoais**: a eficácia da proteção em redes P2P depende fortemente dos protocolos de segurança empregados. Configurações inadequadas podem expor os usuários a vazamentos de credenciais e à disseminação de conteúdo malicioso (NordVPN, 2024).

[10] `https://web3password.com`
[11] `https://www.fyeo.io/kryptpass`

2. **Ataques de negação de serviço (DoS/DDoS)**: a ausência de controle centralizado facilita a orquestração de ataques distribuídos, nos quais múltiplos nós comprometidos geram tráfego excessivo para esgotar recursos computacionais e comprometer a disponibilidade do sistema (Araujo et al., 2009).
3. **Acesso não autorizado**: falhas de configuração ou vulnerabilidades na aplicação P2P podem permitir acesso indevido aos dados armazenados no dispositivo do usuário (LNCC, 2023).

Assim, embora soluções baseadas em P2P ofereçam avanços relevantes em descentralização e controle do usuário, sua segurança depende fortemente de implementações cuidadosas e da adoção de mecanismos eficazes de proteção.

2.3.2. Gerenciadores baseados em *hardware*

Um gerenciador de senhas baseado em *hardware* consiste em um dispositivo físico dedicado ao armazenamento e gerenciamento de credenciais. Diferentemente das soluções baseadas em *software*, que armazenam informações em computadores ou na nuvem, esses dispositivos operam de forma isolada, reduzindo a exposição a vulnerabilidades associadas a sistemas operacionais e serviços externos (Dashlane, 2024). Contudo, esse isolamento não elimina totalmente os riscos, uma vez que dispositivos físicos ainda podem ser alvo de ataques especializados. Exemplos notáveis incluem o RecZone Password Safe[12] e o YubiKey[13].

O funcionamento desses dispositivos depende do componente utilizado para o armazenamento das credenciais. O RecZone Password Safe, por exemplo, armazena senhas e outras informações sensíveis de forma *offline*, utilizando um chip de memória *flash* SPI acessado por um PIN de 4 a 16 caracteres, permitindo o armazenamento de até 400 conjuntos de credenciais (Sharper Image, 2013; Tiansun, 2012; GetUSCart, 2025).

Apesar de sua portabilidade e capacidade de armazenamento, análises conduzidas pela Pen Test Partners identificaram vulnerabilidades críticas nesse tipo de dispositivo, evidenciando riscos relevantes mesmo em soluções dedicadas:

1. **Dados em texto simples**: as credenciais são armazenadas em texto não criptografado, permitindo acesso direto ao conteúdo da memória (Eveleigh, 2019).
2. **Persistência de dados após *reset***: mesmo após um *reset* de fábrica, os dados permanecem na memória, indicando ausência de procedimentos adequados de limpeza (Eveleigh, 2019).
3. **Risco de exposição de dados sensíveis**: em caso de perda ou roubo do dispositivo, terceiros com conhecimento técnico básico podem extrair diretamente os dados do chip de memória, que permanecem armazenados em texto claro (Eveleigh, 2019).

Esses resultados indicam que, embora dispositivos de *hardware* ofereçam vantagens em termos de isolamento e portabilidade, eles ainda estão sujeitos a vulnerabilidades críticas que devem ser consideradas antes de sua adoção.

[12] `https://manuals.plus/asin/B003VV17DO`
[13] `https://www.yubico.com/products/yubikey-5-overview/`

Outro dispositivo amplamente utilizado é o YubiKey, uma chave de autenticação física USB desenvolvida pela Yubico[14], projetada para reforçar a segurança digital por meio de autenticação forte — incluindo autenticação de dois fatores (2FA) e autenticação sem senha — com base em padrões como FIDO2 e U2F (Yubico, 2024b). O dispositivo também oferece funcionalidades como assinatura digital, criptografia e autenticação com chaves RSA e ECC, sem expor as chaves privadas ao sistema *host* (Yubico, 2024c).

Apesar de sua ampla adoção, vulnerabilidades já foram identificadas nesses dispositivos:

1. **Vulnerabilidade EUCLEAK**: descoberta pela NinjaLab[15], essa vulnerabilidade de canal lateral[16] afeta dispositivos YubiKey 5 com *firmware* anterior à versão 5.7. Ela permite que um atacante, com acesso físico ao dispositivo, clone chaves privadas ECDSA, PINs e dados de contas (CRA News Service, 2024);
2. ***Bypass* de autenticação de dois fatores (2FA)**: em cenários específicos de configuração inadequada, a autenticação com YubiKey pode ser indevidamente ignorada ou considerada válida sem a presença do dispositivo físico (Winder, 2025).
3. **Problemas de privacidade**: falhas que permitem inferir informações sobre aplicações e serviços associados a credenciais FIDO2 armazenadas no dispositivo, comprometendo a privacidade do usuário (Yubico, 2024a).

Em síntese, embora gerenciadores de senhas baseados em *hardware* acrescentem uma camada adicional de segurança ao manter os dados isolados de redes e sistemas vulneráveis, eles não são imunes a falhas. Vulnerabilidades de *hardware*, *firmware* ou uso inadequado podem comprometer sua eficácia, exigindo atualização constante, auditoria e boas práticas operacionais.

Diante das limitações observadas tanto em soluções *Peer-to-Peer* quanto em gerenciadores baseados em *hardware*, torna-se evidente a necessidade de abordagens mais robustas de armazenamento distribuído. Nesse contexto, emergem as soluções baseadas em *blockchain*, que combinam criptografia avançada e imutabilidade dos registros distribuídos, reduzindo a dependência de servidores centrais e ampliando a resistência a fraudes e interferências externas.

2.3.3. Gerenciadores baseados em *blockchain*

Em um gerenciador de senhas baseado em *blockchain*, as credenciais não são armazenadas em texto simples em servidores centrais. O usuário criptografa localmente seu conjunto de senhas e registra apenas um *hash* criptográfico desse conteúdo em um bloco da cadeia (*on-chain*), no livro-razão distribuído da rede. Dessa forma, mesmo que o *hash* seja obtido por terceiros, não é possível recuperar as senhas sem a chave privada do usuário, uma vez que os dados sensíveis permanecem criptografados fora da *blockchain* (*off-chain*) (Zyskind et al., 2015).

[14] `https://www.yubico.com`

[15] `https://ninjalab.io`

[16] Em segurança da informação, uma *vulnerabilidade de canal lateral* ocorre quando informações sensíveis, como chaves criptográficas, podem ser extraídas por meio da análise de sinais físicos emitidos pelo dispositivo — por exemplo, variações de tempo, consumo de energia, ruído eletromagnético ou calor — durante a execução de operações criptográficas.

Nesse modelo, o cliente — como uma extensão de navegador ou aplicação local — utiliza uma senha mestra ou um par de chaves criptográficas para cifrar todas as credenciais antes do armazenamento. O *hash* do conteúdo criptografado é então enviado a um contrato inteligente na rede, garantindo integridade e registro auditável. Quando o usuário solicita o acesso a uma credencial, o cliente recupera o pacote completo do armazenamento *off-chain*, por exemplo por meio de protocolos de distribuição de arquivos *peer-to-peer* como o *InterPlanetary File System* (IPFS) (Benet, 2014), verifica a correspondência do *hash* registrado e, caso válida, realiza a descriptografia local das informações (Kosba et al., 2016).

Projetos atuais ilustram essa abordagem híbrida de armazenamento, combinando criptografia local, persistência *off-chain* e validação *on-chain*:

- **PolyPass**[17] — extensão de navegador que utiliza o banco de dados distribuído Polybase para armazenar credenciais criptografadas e grava seus *hashes on-chain*, garantindo integridade e disponibilidade via Ethereum (ETHGlobal, 2023) .
- **Dassword**[18] — cofre de senhas construído sobre IPFS e Filecoin, permitindo ao usuário manter controle direto sobre suas credenciais, já que somente ele detém as chaves criptográficas, enquanto reduz a dependência de um *backend* centralizado (Dassword, 2025).
- **LockEth**[19] — aplicação descentralizada em Ethereum para gerenciamento de senhas via contratos inteligentes, criada por Reshma Haridhas, que proporciona armazenamento seguro e validação *on-chain* das credenciais (Haridhas, 2019).

Esses sistemas aplicam princípios como descentralização, imutabilidade e segurança criptográfica ao gerenciamento de credenciais, reduzindo a dependência de servidores centrais e mitigando pontos únicos de falha (Christidis e Devetsikiotis, 2016). Ao combinarem armazenamento *off-chain* com verificações *on-chain*, preservam a privacidade dos dados e minimizam os custos associados ao uso direto da *blockchain* (Buterin, 2014). Além disso, o uso de contratos inteligentes possibilita um controle mais refinado de acesso às credenciais, incluindo permissões temporárias, revogações automatizadas e rastreamento de ações (Kosba et al., 2016).

Apesar desses benefícios, a adoção em larga escala ainda depende da superação de desafios técnicos e de usabilidade, já discutidos na Seção 2.1, como a necessidade de interfaces intuitivas, mecanismos eficazes de recuperação de chaves e auditorias contínuas do código dos contratos inteligentes.

Em síntese, gerenciadores baseados em *blockchain* representam uma alternativa promissora às abordagens tradicionais ao ampliarem a segurança, a transparência e a autonomia do usuário sobre suas credenciais, estabelecendo um novo paradigma para sua proteção em ambientes digitais.

[17] `https://ethglobal.com/showcase/polypass-fg8ru`
[18] `https://dassword.com/index.html`
[19] `https://github.com/reshmaharidhas/LockEth`

2.3.4. Sistemas similares

Nesta subseção, apresenta-se uma comparação entre a solução proposta neste trabalho e quatro soluções existentes que utilizam a tecnologia *blockchain* com foco em segurança e gerenciamento de credenciais: PolyPass, Dassword, LockEth e Safeguard[20].

O PolyPass é uma extensão de navegador desenvolvida durante o evento *Scaling Ethereum 2023*, projetada para o gerenciamento descentralizado de senhas por meio de tecnologias *blockchain* (ETHGlobal, 2023). A solução integra-se ao navegador, detectando formulários de *login* e oferecendo ao usuário a opção de salvar novas credenciais.

Além do PolyPass, foi analisado o gerenciador Dassword, que opera sem servidores centralizados, armazenando os dados dos usuários diretamente na rede IPFS, com persistência garantida pela integração com a Filecoin (Dassword, 2025). O projeto é *open source*, permitindo contribuições da comunidade, e oferece suporte ao armazenamento de senhas, cartões de crédito, notas pessoais e arquivos sensíveis.

O LockEth é um gerenciador de senhas descentralizado que utiliza contratos inteligentes desenvolvidos para a *blockchain* da Ethereum (Haridhas, 2019). O projeto foi originalmente implantado em redes de teste da Ethereum, utilizadas para validação de funcionalidades antes da implantação em ambiente produtivo.

Os contratos inteligentes do LockEth são escritos em Solidity, linguagem projetada especificamente para o desenvolvimento de *smart contracts* na plataforma Ethereum. Esses contratos controlam a lógica de armazenamento e recuperação das credenciais, assegurando que apenas o proprietário autorizado possa acessá-las. O projeto é *open source*, favorecendo transparência, auditabilidade e colaboração contínua da comunidade.

O Safeguard é um gerenciador de senhas descentralizado baseado na tecnologia Blockstack (atual Stacks) (One Identity, 2025; Investopedia, 2025), utilizando o sistema de armazenamento descentralizado *Gaia* para manter dados pessoais fora da *blockchain*, sob controle do usuário (Stacks Project, 2025). Apesar disso, a solução depende da infraestrutura da rede Stacks para validação e disponibilização dos dados, caracterizando uma descentralização baseada em rede permissionada. O Safeguard também integra notificações de segurança via Twilio e disponibiliza interface por meio de aplicação *web* e extensão de navegador, conciliando segurança e usabilidade.

A proposta deste trabalho consiste no desenvolvimento de um gerenciador de senhas que alia segurança a um modelo de armazenamento distribuído, ampliando a autonomia do usuário sobre seus dados, uma vez que apenas ele detém as chaves criptográficas necessárias ao acesso. As credenciais são cadastradas e criptografadas localmente no dispositivo do usuário, garantindo que nenhuma informação sensível seja trafegada ou armazenada em texto claro. Em seguida, o *backend* aplica um processo de fragmentação do *ciphertext*, gerando duas partes independentes, que são armazenadas em *blockchains* privadas distintas. Essa abordagem mitiga o risco de ponto único de falha e aumenta a resiliência do sistema, mesmo em cenários de comprometimento de uma das redes.

O Quadro 1 apresenta uma comparação entre as principais características dessas plataformas e o sistema desenvolvido neste projeto, evidenciando as diferenças técnicas e

[20] `https://www.oneidentity.com/one-identity-safeguard/`

os aspectos que tornam a proposta mais robusta e segura.

Quadro 1. Comparação entre plataformas de gerenciamento de senhas.

Características	**PolyPass**	**Dassword**	**LockEth**	**Safeguard**	**Proposta deste Trabalho**
Criptografia Local	Sim	Sim	Sim	Sim	**Sim**
Fragmentação dos Dados	Não	Não	Não	Não	**Sim (duas *blockchains*)**
Descentralização	Média	Alta	Alta	Alta	**Média (armazenamento distribuído; governança centralizada)**
Ponto Único de Falha no Armazenamento	Sim	Não	Não	Não	**Não**
Recuperação de Conta	Baixa	Média	Média	Média	**Média**
Imutabilidade	Alta	Alta	Alta	Alta	**Alta**
Suporte à Interface *Web*	Sim	Sim	Sim	Sim	**Não (CLI local)**[21]
Escalabilidade	Média	Média	Média	Média	**Média**
Privacidade dos Metadados	Média	Alta	Alta	Alta	**Alta (dados e metadados locais)**
Controle de Dados (criptografia e chaves locais)	Sim	Sim	Sim	Sim	**Sim (controle local das chaves e do conteúdo criptografado)**

A partir do Quadro 1, observa-se que a solução proposta se diferencia das demais ao adotar um modelo de **armazenamento duplamente distribuído**, no qual as credenciais são **criptografadas localmente** e **fragmentadas em duas partes independentes**, armazenadas em ***blockchains* privadas distintas**. Essa arquitetura elimina o ponto único de falha no armazenamento centralizado e amplia a resiliência e a privacidade dos dados, uma vez que nenhuma das redes, isoladamente, possui acesso à informação completa. Além disso, a execução **totalmente local**, por meio de uma interface CLI, reduz a superfície de ataque e evita a exposição de metadados em serviços *web*, reforçando o controle do usuário sobre o ciclo de vida de suas credenciais.

Embora o protótipo opere em um ambiente controlado com duas *blockchains* privadas locais — o que resulta em escalabilidade classificada como média —, essa característica está alinhada ao escopo do estudo. Em cenários futuros, a escalabilidade poderia ser ampliada com a distribuição dos nós ou adoção de ambientes corporativos, sem alterar o modelo conceitual da solução.

A seguir, são detalhadas algumas das principais características apresentadas no Quadro 1, com o objetivo de esclarecer os critérios técnicos adotados para sua avaliação e os significados dos níveis atribuídos.

[21] A ausência de interface *web* decorre de uma decisão de escopo: o protótipo concentra-se na arquitetura, criptografia e armazenamento distribuído. A implementação de uma GUI está prevista como trabalho futuro.

Descentralização refere-se ao grau em que os dados e a lógica do sistema estão distribuídos entre múltiplos nós, sem depender de um servidor central para armazenamento ou validação. Um nível alto indica que a solução opera sobre uma rede distribuída — pública ou permissionada — em que nenhum nó isolado possui autoridade exclusiva sobre os dados. Já o nível médio caracteriza sistemas em que, embora o armazenamento esteja distribuído, a governança ou a infraestrutura permaneçam parcialmente centralizadas.

Nesse contexto, a descentralização da solução proposta é classificada como média, pois, embora o armazenamento das credenciais seja distribuído entre duas *blockchains* privadas independentes — eliminando o ponto único de falha no armazenamento — a infraestrutura opera em ambiente local sob controle de uma única organização. Assim, o sistema apresenta forte descentralização no armazenamento, mas não em toda a camada infraestrutural, justificando a classificação adotada.

A recuperação de conta refere-se à capacidade do sistema de permitir que o usuário recupere o acesso às suas credenciais em situações como perda de dispositivos, senhas mestras ou chaves privadas. Um nível alto indica a existência de mecanismos robustos, como recuperação social, *backups* criptografados ou múltiplas camadas de autenticação. Um nível médio caracteriza soluções que oferecem opções limitadas de recuperação ou que dependem de configurações prévias por parte do usuário. Já um nível baixo indica a ausência de mecanismos viáveis de recuperação, tornando a perda de acesso irreversível.

O controle de dados diz respeito à autonomia do usuário sobre o conteúdo sensível armazenado. Quando essa característica está presente, a criptografia é realizada localmente e apenas o próprio usuário detém as chaves necessárias para acessar as credenciais, reduzindo a necessidade de confiar em terceiros — ainda que a infraestrutura de armazenamento seja operada por entidades autorizadas. Esse critério refere-se exclusivamente ao acesso ao conteúdo criptografado, não implicando controle sobre a governança ou os nós da rede *blockchain*.

Por fim, a imutabilidade refere-se à capacidade do sistema de garantir que os dados, uma vez registrados, não possam ser modificados ou removidos sem deixar vestígios. Um nível alto indica o uso de tecnologias como *blockchain* ou registros invioláveis, assegurando a integridade e a rastreabilidade das informações ao longo do tempo.

3. Solução proposta

Esta seção descreve o processo de desenvolvimento da solução proposta[22], detalhando as decisões técnicas, as tecnologias selecionadas e as etapas implementadas ao longo do desenvolvimento. O foco principal foi criar uma aplicação segura, com armazenamento distribuído e fragmentado em duas *blockchains* privadas, voltada para o ambiente *desktop*, com interação por meio de uma interface em linha de comando (do inglês *Command-Line Interface* — CLI), capaz de oferecer ao usuário controle local sobre o ciclo de vida de suas credenciais, desde o armazenamento até a recuperação.

A escolha pelo ambiente *desktop* e pela interface em linha de comando não representa uma limitação conceitual da solução, mas uma decisão arquitetural deliberada.

[22] `https://github.com/LorenaMuralha23/password-manager.git`

Nesse modelo, o *backend* da aplicação é executado integralmente no dispositivo do usuário, não havendo servidores remotos ou componentes centralizados para processamento de dados sensíveis. Essa opção reduz significativamente a superfície de ataque, elimina dependências de serviços externos e favorece maior previsibilidade do ambiente de execução, fatores essenciais para avaliar de forma controlada os mecanismos de segurança, criptografia local e armazenamento distribuído que constituem o foco central deste trabalho. Assim, a estrutura adotada prioriza segurança e controle, em vez de conveniência de interface, estando alinhada ao objetivo de construir um protótipo voltado à investigação da arquitetura e não à usabilidade final.

O desenvolvimento seguiu as definições estabelecidas na fase de modelagem, priorizando a privacidade, a resiliência contra ataques externos e a redução de riscos associados a pontos únicos de falha no armazenamento das credenciais. As ferramentas, *frameworks* e arquiteturas escolhidas tiveram como critério principal a capacidade de atender aos requisitos de segurança e desempenho estabelecidos na solução proposta.

Nas subseções a seguir, serão abordados os principais componentes e etapas técnicas que compõem o sistema, incluindo a arquitetura geral da aplicação, o sistema de *login* e cadastro, a estrutura modular do *backend* desenvolvida com Spring Boot, a interface em linha de comando (CLI), a implementação da criptografia local, o processo de fragmentação das credenciais, o armazenamento distribuído em *blockchains* privadas, o uso do banco de dados H2 como solução de armazenamento local e o fluxo de recuperação das informações.

3.1. Levantamento de requisitos

O processo de levantamento de requisitos foi uma etapa fundamental para garantir que a solução desenvolvida atendesse às necessidades funcionais e não funcionais previstas. A definição criteriosa desses requisitos permitiu direcionar o desenvolvimento de forma estruturada, alinhando as escolhas de tecnologia e as estratégias de implementação aos objetivos de segurança, desempenho e usabilidade estabelecidos.

Para garantir que a solução atenda aos objetivos propostos e ofereça ao usuário todas as funcionalidades necessárias, foi realizada a definição dos requisitos funcionais do sistema (Quadro 2). Esses requisitos descrevem as principais ações e comportamentos que a aplicação deve executar, abrangendo desde o armazenamento seguro das credenciais até o processo de recuperação de dados.

Quadro 2. Requisitos Funcionais do Sistema.

Requisito	Descrição
Cadastro de Credenciais	Permitir o cadastro de novas credenciais, com criptografia local antes de qualquer envio para armazenamento.
Fragmentação de Dados	Implementar a fragmentação dinâmica e aleatória das credenciais criptografadas.
Armazenamento Distribuído	Distribuir os fragmentos das credenciais entre duas *blockchains* privadas distintas.
Gerenciamento de Metadados	Armazenar, de forma segura, no banco de dados local (H2 Database), as informações de mapeamento necessárias para recuperação das credenciais.
Recuperação de Credenciais	Permitir a recuperação das credenciais por meio da reagrupação dos fragmentos e posterior descriptografia local.

Login Local	Implementar um sistema de autenticação local para controlar o acesso ao *backend* da aplicação.

Além das funcionalidades essenciais, foram definidos também os requisitos não funcionais (Quadro 3), que tratam de aspectos relacionados à qualidade, desempenho, segurança e portabilidade do sistema. Esses requisitos são fundamentais para assegurar que a aplicação opere de maneira eficiente, segura e compatível com diferentes ambientes de execução.

Quadro 3. Requisitos Não Funcionais do Sistema.

Requisito	Descrição
Segurança	Mitigar ao máximo a exposição de dados sensíveis, garantindo que o sistema não armazene nem transmita credenciais em texto claro.
Desempenho	As operações de armazenamento, recuperação e criptografia devem ocorrer de forma eficiente, com baixo tempo de resposta para o usuário.
Portabilidade	O sistema deve ser projetado para execução em ambiente *desktop*, com arquitetura independente de sistema operacional, permitindo sua adaptação a diferentes plataformas, como Windows, Linux e macOS. No escopo do protótipo desenvolvido, os testes e validações foram realizados exclusivamente em ambiente Windows.
Escalabilidade Local	Estruturar o banco de dados e a integração com as *blockchains* de forma que permita o armazenamento de um número crescente de credenciais, mantendo desempenho adequado mesmo com o aumento do volume de dados locais.
Resiliência a Falhas	Implementar mecanismos para tratamento de falhas de comunicação com as *blockchains* para garantir a integridade dos dados em caso de erros inesperados.

O levantamento desses requisitos serviu como base para as etapas seguintes de modelagem, projeto da arquitetura e implementação do sistema, garantindo alinhamento entre a proposta inicial e o produto final desenvolvido.

3.2. Modelagem da plataforma

Com base nas decisões arquiteturais apresentadas anteriormente, esta subseção apresenta os principais componentes que integram a arquitetura da solução proposta. A modelagem foi estruturada de forma modular, contemplando o fluxo de registro e autenticação de usuários, a lógica do *backend*, o uso de *blockchain* para armazenamento distribuído e o papel do banco de dados local. Cada parte será descrita a seguir, com foco em suas responsabilidades, interações e contribuições para a segurança e funcionamento da plataforma.

3.2.1. Registro e autenticação inicial de usuários

A etapa inicial de interação do usuário com a plataforma ocorre por meio da CLI, composta por duas operações fundamentais: o registro de uma nova conta e a autenticação de acessos subsequentes. Por se tratar de um sistema voltado ao gerenciamento de informações sensíveis, esses procedimentos foram projetados com foco em segurança

e controle local, reduzindo a possibilidade de acesso por usuários não autorizados e fortalecendo a proteção sobre as identidades digitais criadas na plataforma.

Essa etapa tem como principal objetivo estabelecer, para cada usuário, uma identidade digital segura vinculada exclusivamente ao seu ambiente local. Essa identidade é associada a uma chave criptográfica gerada internamente pelo sistema, garantindo que todo o processo de autenticação ocorra de maneira autônoma e controlada no ambiente local do usuário. Dessa forma, favorece-se que o acesso aos recursos da aplicação — bem como às informações nela armazenadas — ocorra de modo restrito, confidencial e devidamente validado, em continuidade ao fluxo iniciado pela CLI.

Para atender a essas premissas de segurança, o processo de primeiro acesso ocorre inteiramente por meio da CLI, sem a necessidade de componentes externos ou conexões remotas. O *backend*, desenvolvido em Spring Boot, é responsável por executar as operações de segurança, incluindo a geração das chaves criptográficas, a cifragem dos dados sensíveis e a verificação das credenciais informadas pelo usuário. A autenticação em dois fatores (2FA – *Two-Factor Authentication*) está prevista para versões futuras da aplicação, integrando-se ao mesmo fluxo de validação local. O banco de dados — implementado com H2 Database — armazena exclusivamente dados protegidos, como a chave simétrica cifrada e o *hash* de verificação associado, reduzindo significativamente o risco de exposição de informações sensíveis em texto claro.

No momento do registro, o usuário fornece um identificador único, como um nome de usuário ou e-mail, juntamente com uma senha. A partir dessas informações, o *backend* gera uma chave simétrica exclusiva que constitui o núcleo da identidade criptográfica do usuário e servirá como base para operações seguras na aplicação. Para garantir sua confidencialidade, essa chave é protegida por meio de cifragem simétrica a partir de uma chave derivada da senha do usuário, assegurando que o valor original jamais seja utilizado diretamente no processo de criptografia. Métodos de derivação mais robustos, como PBKDF2 ou Argon2, estão previstos para versões futuras do sistema, a fim de reforçar a resistência contra ataques de força bruta e aprimorar o gerenciamento seguro das credenciais.

Além dessas etapas, o fluxo de registro também prevê, para versões futuras, a geração local de um segredo compartilhado (*shared secret*) destinado à autenticação em dois fatores (2FA) baseada no protocolo TOTP (*Time-based One-Time Password*). Esse segredo seria apresentado ao usuário durante o processo de cadastro, permitindo sua configuração em aplicativos autenticadores compatíveis. Embora ainda não implementado, esse mecanismo complementará o modelo de segurança ao fornecer um segundo fator atrelado à posse de um dispositivo físico.

O sistema também calcula, durante o registro, um *hash* da chave simétrica original (antes de ser cifrada), cuja finalidade é possibilitar a validação de sua integridade durante o processo de *login*. Ao final do registro, são armazenados localmente no banco os seguintes elementos: o identificador do usuário, a chave simétrica cifrada, o *hash* de verificação correspondente e o campo reservado para futura integração do 2FA. Importante destacar que a senha do usuário não é armazenada, nem mesmo de forma cifrada ou mascarada, em nenhuma fase do processo, o que reduz significativamente o risco de exposição indevida.

A utilização de um *hash* verificável da chave simétrica representa uma decisão

de projeto voltada à validação local da integridade da chave após sua decifragem, evitando que chaves corrompidas ou incorretas sejam utilizadas nas etapas subsequentes do sistema. Entretanto, reconhece-se que, em um cenário de comprometimento completo do banco de dados local, a presença simultânea da chave cifrada e de seu *hash* pode introduzir um vetor teórico de ataque *offline*, no qual um adversário poderia tentar validar tentativas de decifragem de forma independente do sistema.

No contexto deste protótipo, esse risco é mitigado por múltiplas camadas de proteção, incluindo a cifragem da chave simétrica, a ausência de armazenamento da senha do usuário em qualquer forma, a criptografia do banco de dados em repouso e a execução integral do sistema em ambiente local controlado. Ainda assim, para versões futuras da aplicação, está prevista a adoção de mecanismos mais robustos de verificação, como funções de derivação de chave com custo computacional elevado (por exemplo, PBKDF2 ou Argon2) ou esquemas alternativos que reduzam a possibilidade de validação *offline*, reforçando a resistência do sistema contra ataques pós-comprometimento.

A Figura 6 do Anexo A.4 apresenta o diagrama do fluxo de cadastro, representando as etapas realizadas desde a coleta das credenciais pelo usuário até o armazenamento seguro das informações no banco de dados local. O processo contempla a geração da chave simétrica, sua cifragem a partir de uma chave derivada da senha e o cálculo do *hash* utilizado para verificação futura de integridade e autenticidade, além da previsão do espaço para integração do 2FA.

Já no fluxo de autenticação, o usuário fornece novamente suas credenciais por meio da CLI. A senha informada é utilizada pelo *backend* para derivar uma chave criptográfica empregada na validação da chave simétrica previamente armazenada de forma cifrada. Caso a operação seja bem-sucedida, o sistema calcula o *hash* da chave recuperada e o compara com o valor registrado no banco de dados local. A coincidência entre os *hashes* confirma a legitimidade da senha informada, validando a identidade do usuário e concluindo a primeira etapa do processo de autenticação.

Na sequência, o mecanismo de autenticação em dois fatores (2FA), previsto para versões futuras, deverá adicionar uma camada complementar de segurança ao processo de *login*. Nesse modelo, o usuário informará um código temporário gerado por um aplicativo autenticador compatível com o protocolo TOTP, como Google Authenticator[23] ou Authy[24]. O *backend* calculará o código esperado com base no segredo compartilhado configurado no registro e no horário atual, comparando-o ao valor fornecido pelo usuário. Somente mediante a validação bem-sucedida de ambas as etapas — senha e código temporário — o acesso à aplicação será autorizado, completando o processo de autenticação em sua forma ampliada.

A incorporação desse modelo de autenticação tende a fortalecer consideravelmente a proteção contra tentativas de acesso não autorizado, mitigando especialmente riscos associados a ataques baseados no roubo de credenciais, como *phishing* ou vazamento de senhas. Ao introduzir um fator adicional de verificação vinculado à posse física de um dispositivo autenticador, o sistema amplia sua capacidade de resistência frente a ameaças direcionadas a senhas ou chaves comprometidas.

[23] `https://support.google.com/accounts/topic/7189195`
[24] `https://www.authy.com/`

O processo de autenticação descrito é representado de forma conceitual na Figura 7 do Anexo A.4. O sistema utiliza a senha informada pelo usuário para derivar a chave necessária à validação da chave simétrica previamente armazenada, verificando sua integridade por meio do *hash* correspondente. Em uma etapa complementar, prevista para versões futuras, será realizada a autenticação em dois fatores (2FA), baseada na validação de um código temporário gerado a partir do segredo compartilhado. O acesso à aplicação será autorizado apenas quando todas as etapas forem concluídas com êxito.

A adoção desse modelo de autenticação proporciona uma série de vantagens em termos de segurança e autonomia do usuário. Como nenhuma senha é armazenada diretamente — nem mesmo na forma de *hash* ou de qualquer derivado —, e considerando que a chave criptográfica do usuário permanece cifrada com base em uma chave derivada de sua senha, o risco de exposição de credenciais é significativamente reduzido, mesmo em eventuais cenários de comprometimento do banco de dados local. Além disso, a inclusão planejada da autenticação em dois fatores (2FA) tende a elevar ainda mais o nível de proteção contra acessos indevidos. O funcionamento autônomo do sistema, aliado à integração com redes *blockchain* privadas, reforça sua resiliência e seu modelo de armazenamento distribuído, permitindo o registro distribuído de fragmentos criptografados e fortalecendo o controle do usuário sobre suas próprias credenciais.

Por fim, é importante destacar que os dados e estruturas gerados durante essa etapa desempenham um papel essencial na continuidade da experiência do usuário e no funcionamento integrado da plataforma. Tanto o *backend* quanto o banco de dados local, além dos módulos responsáveis pela integração com as redes *blockchain* privadas, dependem diretamente das informações estabelecidas durante o primeiro acesso, como as chaves criptográficas e os dados associados às credenciais do usuário. Esse conjunto de informações reforça a necessidade de que o processo de registro e autenticação seja conduzido, desde sua concepção até sua implementação, com elevados padrões de segurança, precisão e integridade, uma vez que sua proteção impacta diretamente toda a operação segura e o modelo de armazenamento distribuído da plataforma.

3.2.2. *Backend*

O *backend* desenvolvido para esta aplicação foi implementado em Java, utilizando o *framework* Spring Boot, e projetado especificamente para execução local em ambientes *desktop*, integrando-se à CLI como ponto de interação principal com o usuário. Esse modelo reduz a dependência de servidores remotos e de infraestruturas externas, assegurando que todo o processamento e armazenamento de dados sensíveis ocorram no próprio dispositivo do usuário, comunicando-se apenas com instâncias locais das *blockchains* privadas. Dessa forma, reduz-se significativamente a exposição a riscos como vazamentos de informações, ataques externos e falhas decorrentes de pontos únicos de centralização.

A Figura 2 apresenta uma visão geral da comunicação entre os principais componentes do *backend*, evidenciando o papel central deste módulo na mediação entre o usuário, o banco de dados local e as duas redes *blockchain* privadas. O diagrama ilustra as etapas de interação envolvidas no armazenamento distribuído e na recuperação das credenciais, representando de forma conceitual o fluxo de dados executado internamente pelo sistema. Cada uma dessas etapas é detalhada nas subseções a seguir.

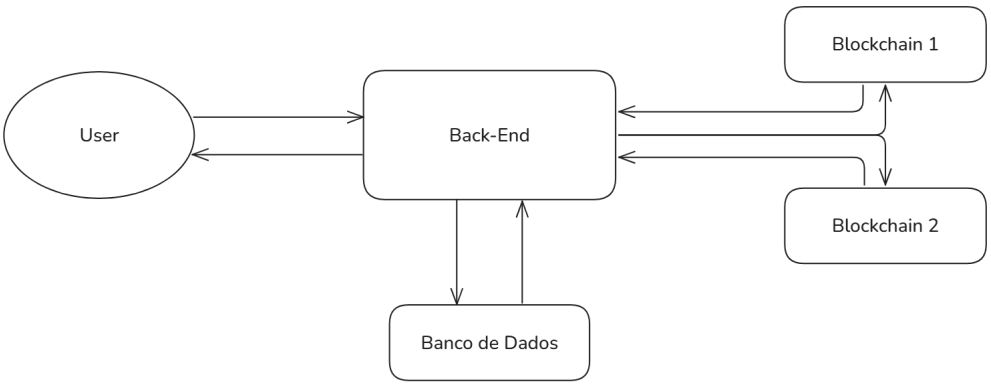

Figura 2. Comunicação entre o *backend* e os módulos da aplicação.

Ao concentrar todas as funcionalidades críticas diretamente na máquina local, o sistema, desenvolvido sob o *framework* Spring Boot, passa a operar de forma local e independente de servidores externos para suas funções críticas, permanecendo totalmente integrado à CLI. Essa configuração garante ao usuário controle local sobre o ciclo de vida do conteúdo criptografado de suas credenciais — desde a criação até a exclusão segura —, mantendo toda a lógica sensível sob sua posse, ainda que o armazenamento distribuído utilize duas instâncias privadas de *blockchain*. Todas as etapas do processo — incluindo o armazenamento, a fragmentação, a recuperação e a decriptação das senhas — ocorrem exclusivamente no ambiente local, sem que quaisquer informações trafeguem ou sejam mantidas em servidores de terceiros.

Além de reforçar a privacidade e a segurança, essa arquitetura promove a independência operacional do sistema, reduzindo sua suscetibilidade a falhas externas e indisponibilidades típicas de aplicações hospedadas na *web* ou em ambientes de nuvem. Embora se comunique com redes *blockchain* privadas, o *backend* mantém toda a lógica de execução e controle de dados de forma local, sob controle local do usuário. Essa configuração elimina vulnerabilidades decorrentes de integrações remotas com servidores de terceiros e favorece a continuidade operacional da aplicação em cenários restritos de conectividade, ao reduzir sua dependência de serviços externos, embora não elimine riscos associados a falhas no ambiente local do usuário.

Para viabilizar a implementação de todas essas funcionalidades com a robustez, a escalabilidade e o nível de segurança exigidos pelo projeto, optou-se pela adoção de um conjunto de tecnologias adequadas ao contexto de uma aplicação *desktop* de alto desempenho e execução local. Nesse cenário, o *backend* foi desenvolvido em Java, utilizando o *framework* Spring Boot, cuja arquitetura modular e suporte nativo à injeção de dependência favorecem o desenvolvimento seguro, a manutenção do código e a integração entre os diferentes serviços do sistema. A escolha do Java como linguagem principal deve-se à sua maturidade tecnológica, ampla portabilidade e comprovada eficiência na implementação de aplicações críticas voltadas à segurança da informação.

De forma geral, o *backend* desenvolvido concentra todas as operações fundamentais relacionadas ao ciclo de vida das credenciais dentro da aplicação. A cada interação do usuário — realizada por meio da CLI —, como o cadastro, a recuperação ou a exclusão de uma senha, uma sequência de serviços é acionada internamente no *backend*, de forma mo-

dular e controlada pelo *framework* Spring Boot. Esses serviços compõem o núcleo lógico da aplicação e asseguram que todas as etapas críticas de processamento sejam executadas localmente e de maneira segura. Entre os principais processos realizados, destacam-se:

- **Autenticação do usuário**: validação das credenciais fornecidas via CLI, garantindo que apenas usuários legítimos possam interagir com o sistema;
- **Criptografia das credenciais**: aplicação do algoritmo AES-256-GCM, com vetor de inicialização aleatório, para proteger os dados sensíveis antes de qualquer forma de armazenamento;
- **Fragmentação dos dados**: divisão dinâmica e aleatória da credencial criptografada em partes independentes, dificultando a reconstrução não autorizada;
- **Armazenamento nas *blockchains***: envio de cada fragmento para uma instância privada distinta da rede Hyperledger Fabric, garantindo redundância e armazenamento distribuído;
- **Persistência de metadados**: registro local, no banco de dados H2, das informações essenciais para posterior reconstrução, como os identificadores dos fragmentos e o mapa de fragmentação;
- **Recuperação de credenciais**: reagrupamento dos fragmentos obtidos nas duas *blockchains*, seguido da decriptação local e da apresentação segura ao usuário.

3.2.3. Criptografia local

Uma das premissas centrais deste projeto é assegurar que as credenciais do usuário permaneçam protegidas ao longo de todo o seu ciclo de processamento local, inclusive contra a própria aplicação e seus desenvolvedores. Todas as operações de cifragem e decifragem das senhas ocorrem exclusivamente no *backend* local desenvolvido em Java, utilizando o *framework* Spring Boot, e acessado pelo usuário por meio da CLI. Dessa forma, antes de qualquer envio de dados para armazenamento externo — seja no banco de dados local ou nas redes *blockchain* —, as credenciais são protegidas por meio de criptografia simétrica, técnica na qual uma mesma chave é utilizada tanto para cifrar quanto para decifrar as informações, equilibrando segurança e eficiência computacional.

O algoritmo adotado para a proteção das credenciais é o AES-256-GCM (*Advanced Encryption Standard – 256 bits, Galois/Counter Mode*), amplamente reconhecido por sua robustez e desempenho em aplicações de segurança da informação. O modo GCM combina os processos de cifragem e autenticação em uma única operação, assegurando não apenas a confidencialidade, mas também a integridade das informações. Dessa forma, qualquer tentativa de modificação nos dados cifrados pode ser detectada durante a decifragem, dificultando ataques de falsificação ou adulteração de conteúdo. Cada processo de cifragem utiliza um vetor de inicialização (*Initialization Vector* – IV) de 96 bits, gerado de forma aleatória e segura por meio de um gerador criptograficamente forte (*SecureRandom*), sendo esse vetor armazenado junto ao fragmento correspondente para permitir a posterior decifragem.

A chave criptográfica responsável pela proteção das credenciais é gerada e armazenada localmente, vinculada à identidade criptográfica do usuário estabelecida durante o processo de registro. Essa vinculação reduz a possibilidade de que ambientes não autorizados realizem operações. Dessa forma, mesmo que um invasor obtenha acesso físico

ao banco de dados local ou às redes *blockchain*, os fragmentos armazenados tendem a permanecer indecifráveis sem a posse dessa chave, que constitui o elemento central da confidencialidade do sistema.

A comunicação entre o *backend* e as redes *blockchain* privadas ocorre por meio de canais seguros com suporte à criptografia de transporte (*Transport Layer Security* – TLS), assegurando a confidencialidade e a integridade das transações. No protótipo desenvolvido, essa comunicação é executada em ambiente local: o *backend* e os nós das redes *blockchain* operam na mesma máquina apenas para fins de desenvolvimento, simulando um ambiente distribuído em escala reduzida. Essa configuração não representa o modelo distribuído completo que uma rede permissionada pode adotar. Ainda assim, o sistema mantém o mesmo rigor de segurança aplicado em redes reais, contribuindo para que a troca de informações seja protegida pela infraestrutura criptográfica nativa do Hyperledger Fabric. Essa camada adicional de proteção reforça a confiabilidade das interações entre os módulos do sistema e as redes de armazenamento distribuído.

As transações enviadas às redes *blockchain* são assinadas digitalmente, de modo a garantir sua integridade, autenticidade e rastreabilidade. Essa assinatura é gerenciada pelo *Software Development Kit* (SDK) oficial do Hyperledger Fabric, que administra o uso das chaves criptográficas e dos certificados digitais vinculados à identidade da aplicação. Com esse mecanismo, assegura-se que apenas participantes devidamente autorizados possam registrar ou consultar informações nas redes, mantendo a confiabilidade e a validade das transações executadas.

Embora o canal de comunicação seja protegido por criptografia de transporte, o conteúdo das transações — como os fragmentos de credenciais — não é cifrado pela própria *blockchain*. As redes *blockchain* garantem integridade e imutabilidade das transações, mas não realizam proteção criptográfica do conteúdo armazenado, o que reforça a necessidade da cifra local prévia. Por esse motivo, o sistema adota uma camada adicional de criptografia simétrica aplicada localmente antes da transmissão dos dados sensíveis. Essa medida assegura que, mesmo em caso de acesso não autorizado aos registros armazenados, as informações permaneçam indecifráveis sem a chave correta, consolidando o princípio de segurança em profundidade que orienta toda a arquitetura proposta.

Em síntese, a adoção da criptografia local e prévia ao armazenamento constitui um dos pilares centrais da arquitetura proposta. Essa estratégia fortalece o modelo de segurança da aplicação ao mitigar riscos como vazamentos internos, ataques de interceptação e exploração de vulnerabilidades em camadas externas, contribuindo para que as operações críticas ocorram sob controle local do ambiente da aplicação, sem dependência de servidores externos.

3.2.4. Fragmentação

Após a etapa de criptografia local, o *backend* executa um processo adicional de embaralhamento estrutural (*shuffling*) aplicado ao *ciphertext*, reorganizando de forma pseudoaleatória a ordem de seus *bytes*. Esse procedimento aumenta a descontinuidade interna do dado cifrado e dificulta correlações entre regiões adjacentes, reforçando sua imprevisibi-

lidade antes da divisão em fragmentos.

Em seguida, o sistema realiza a fragmentação das credenciais, dividindo o *ciphertext* embaralhado em dois fragmentos independentes. Essa divisão é executada de forma dinâmica e aleatória, por meio de um gerador seguro de números pseudoaleatórios (*SecureRandom*), garantindo que a posição do corte varie a cada operação. Essa aleatoriedade criptográfica aumenta a imprevisibilidade do processo e dificulta tentativas de reconstrução não autorizada das informações originais.

Cada fragmento gerado contém apenas uma parte incompleta da credencial criptografada e embaralhada, o que torna altamente improvável a recuperação das informações originais a partir de um único fragmento. Essa estratégia introduz uma camada adicional de segurança ao reduzir a superfície de ataque e eliminar o ponto único de falha associado ao armazenamento centralizado de credenciais, assegurando que, mesmo em caso de comprometimento de uma das *blockchains* privadas, o fragmento isolado não forneça material suficiente para reconstrução, correlação ou exploração do conteúdo sensível. Além disso, a fragmentação reduz correlações semânticas entre blocos cifrados, atuando não apenas como mecanismo de resiliência, mas também como componente ativo de proteção da confidencialidade dos dados.

O mapeamento da fragmentação — isto é, o registro que indica a posição de corte e a correspondência entre os fragmentos — é armazenado separadamente no banco de dados local. Esse mapa é protegido pela mesma camada de criptografia simétrica aplicada ao restante da estrutura, e sua separação física e lógica em relação aos fragmentos impede correlações diretas e reforça o isolamento entre as partes necessárias para a reconstrução.

A fragmentação, portanto, configura-se como uma camada complementar à criptografia, elevando o nível de proteção do sistema ao dificultar a exploração de vulnerabilidades associadas a acessos não autorizados. Ao distribuir de forma autônoma e criptograficamente isolada os fragmentos das credenciais, o modelo reforça os princípios de confidencialidade, integridade e defesa em profundidade que fundamentam a arquitetura de segurança da solução proposta.

3.2.5. Armazenamento da credencial

Com os fragmentos das credenciais devidamente criptografados e separados, o processo de armazenamento distribuído é conduzido pelo módulo de *backend*, desenvolvido em Java com o *framework* Spring Boot. Nessa etapa, cada fragmento cifrado é destinado a uma instância distinta da rede Hyperledger Fabric, assegurando que nenhuma das *blockchains* privadas contenha a totalidade da informação. Essa estratégia de distribuição mitiga o ponto único de falha associado ao armazenamento centralizado de credenciais e reduz significativamente o risco de exposição decorrente de acessos indevidos ou comprometimentos parciais de infraestrutura.

As redes utilizadas são *blockchains* privadas e permissionadas, implementadas com a tecnologia Hyperledger Fabric. Nelas, o acesso aos nós, bem como as operações de leitura e gravação, é rigidamente controlado por certificados digitais e políticas de identidade definidas pelo sistema. Essa configuração reduz vulnerabilidades típicas de redes públicas, como a exposição ampla de metadados e a dependência de mecanismos

de consenso abertos, favorecendo maior previsibilidade, privacidade e controle sobre as transações — especialmente em um ambiente local e de menor escala, como o protótipo desenvolvido.

Embora executadas no mesmo ambiente físico, as duas instâncias da *blockchain* operam de forma isolada do ponto de vista lógico, armazenando fragmentos distintos do *ciphertext*. Com isso, mesmo que uma das *blockchains* seja comprometida, o fragmento obtido tende a ser criptograficamente insuficiente para reconstruir ou inferir a credencial original. Além disso, a imutabilidade inerente ao livro-razão distribuído (*ledger*) assegura que, uma vez registrados, os fragmentos se tornem bastante difíceis de alterar ou remover sem gerar evidências criptográficas, reforçando a integridade e a rastreabilidade das informações armazenadas.

O registro dos fragmentos nas *blockchains* é mediado pelo módulo de *backend*, que realiza a comunicação com as redes por meio do *Software Development Kit* (SDK) oficial da Hyperledger Fabric. Para cada operação de armazenamento, são gerados identificadores de transação e metadados associados à posição de cada fragmento, os quais são preservados localmente no banco de dados H2. Esses metadados — mantidos sob criptografia — funcionam como um índice seguro que possibilita a posterior reconstrução da credencial, sem expor o conteúdo sensível ou comprometer a confidencialidade do processo.

No modelo proposto, a reconstrução integral de uma credencial requer o acesso simultâneo aos três componentes do sistema: o banco de dados local, que armazena os metadados e o mapa de fragmentação; e as duas redes *blockchain* privadas, responsáveis por preservar separadamente os fragmentos criptografados. Essa interdependência reforça a confidencialidade e a resiliência da solução proposta, uma vez que nenhuma das partes, isoladamente, contém informação suficiente para revelar ou reconstruir o conteúdo original.

3.2.6. Recuperação das credenciais

A recuperação de uma credencial no sistema proposto é iniciada por meio da CLI e segue um fluxo inverso ao processo de armazenamento, preservando os mesmos princípios de segurança, fragmentação e controle local de dados. Todo o procedimento é conduzido pelo *backend* da aplicação, garantindo que as operações ocorram integralmente no ambiente local do usuário, sem dependência de servidores externos.

Quando o usuário solicita o acesso a uma credencial, o *backend* da aplicação consulta o banco de dados local H2, onde estão armazenados os metadados necessários para o processo de recuperação, como os identificadores de localização dos fragmentos nas *blockchains* e os índices referentes à posição da fragmentação. Esses dados permanecem cifrados em repouso e são decifrados exclusivamente em memória volátil durante a execução do processo. A credencial recuperada existe em texto claro apenas no espaço de memória do processo e pelo menor tempo necessário para sua apresentação ao usuário, não sendo persistida em disco nem transmitida em texto claro por rede em nenhuma etapa da operação.

De posse desses metadados, o sistema estabelece comunicação autenticada com as

duas redes Hyperledger Fabric privadas por meio do *Software Development Kit* (SDK) em Java, recuperando de forma segura os fragmentos criptografados correspondentes. Após a obtenção dos fragmentos, o *backend* realiza seu reagrupamento conforme o mapeamento definido no momento da fragmentação, reconstruindo o *ciphertext* original. A verificação de integridade ocorre na etapa subsequente, durante a decifragem, por meio do mecanismo de autenticação provido pelo modo GCM *(Galois/Counter Mode)*.

Em seguida, o *backend* executa o processo de decifragem local utilizando o algoritmo AES-256-GCM (*Advanced Encryption Standard* com modo GCM), restituindo a credencial ao seu formato original exclusivamente em memória e dentro do ambiente controlado da aplicação. Essa abordagem evita a persistência de partes da credencial em texto claro ou sua transmissão externa, restringindo sua existência em texto claro ao espaço de memória do processo e ao tempo estritamente necessário para apresentação ao usuário.

Ao manter todas as etapas críticas sob o controle exclusivo do ambiente local e ao distribuir os fragmentos das credenciais em redes *blockchain* privadas e imutáveis, o sistema reforça a confidencialidade, a integridade e a resiliência do processo de recuperação. Essa arquitetura mitiga significativamente os riscos de interceptação e de acesso indevido, contribuindo para a proteção de ponta a ponta das informações sensíveis do usuário.

Cabe ressaltar que, embora o texto claro exista transitoriamente em memória para apresentação ao usuário, o sistema não realiza qualquer persistência, cache ou transmissão desse conteúdo, mantendo a exposição limitada ao escopo do processo em execução.

3.2.7. *Blockchain*

A plataforma proposta adota uma arquitetura híbrida que integra componentes tradicionais de *backend*, desenvolvidos em Java com o *framework* Spring Boot, a soluções baseadas em *blockchain*, com o objetivo de fortalecer a segurança e a integridade no armazenamento distribuído das credenciais dos usuários. Essa combinação permite que a aplicação mantenha controle local sobre todas as operações — por meio de uma CLI — enquanto utiliza redes *blockchain* privadas para garantir a imutabilidade e a distribuição controlada dos fragmentos criptografados.

Antes da definição da tecnologia de *blockchain* a ser adotada, foi analisada a distinção entre redes públicas e privadas, considerando as necessidades específicas da aplicação, que prioriza a confidencialidade, o controle de acesso e a integridade dos dados armazenados. Embora as *blockchains* públicas ofereçam alto grau de transparência e descentralização, tais características não se alinham aos objetivos do sistema proposto, uma vez que este manipula fragmentos criptografados de credenciais e não requer qualquer tipo de exposição pública. Nesse contexto, a transparência — frequentemente considerada uma vantagem em redes abertas — torna-se um fator de risco, pois poderia revelar informações sobre a estrutura ou a existência dos fragmentos armazenados, mesmo que o conteúdo permaneça criptografado.

Dessa forma, optou-se pela utilização de duas redes *blockchain* privadas e permissionadas, que oferecem um ambiente permissionado com controle rigoroso de identidades e operações, adequado ao contexto da aplicação. Essa abordagem viabiliza a implementação de mecanismos de auditoria interna e garante a confidencialidade dos re-

gistros, uma vez que apenas identidades autorizadas pelo próprio sistema podem interagir com o *ledger*, mantendo controle operacional sobre o acesso sem implicar controle total sobre a infraestrutura da rede — característica típica de *blockchains* privadas. Além disso, a adoção de múltiplas redes privadas reduz significativamente o risco de ponto único de falha, reforçando a resiliência e a segurança da arquitetura proposta.

Entre as plataformas disponíveis nesse modelo, foram adotadas duas instâncias independentes da *blockchain* privada Hyperledger Fabric[25], cada uma operando como uma rede permissionada, isolada e segura. Essas instâncias funcionam de forma complementar, de modo que cada uma armazena um fragmento distinto das credenciais criptografadas geradas pelo *backend* da aplicação. O papel central dessas redes é garantir a integridade, a imutabilidade e a distribuição controlada dos fragmentos, reduzindo a probabilidade de que uma única instância da *blockchain* detenha acesso suficiente para reconstruir os dados originais.

A escolha pelo Hyperledger Fabric fundamenta-se em um conjunto de características técnicas que o tornam particularmente adequado ao contexto do sistema desenvolvido. Além de oferecer suporte nativo à integração com o *backend* em Java Spring Boot, o Fabric proporciona um ambiente seguro, modular e auditável para o armazenamento distribuído de dados sensíveis. Entre suas principais vantagens, destacam-se:

- **Código aberto (*open source*):** o projeto é mantido pela Linux Foundation[26] sob uma licença permissiva, garantindo independência tecnológica, transparência no código-fonte e possibilidade de customização conforme as necessidades específicas da aplicação.
- **Suporte à linguagem Java:** a compatibilidade nativa com Java simplifica o desenvolvimento de contratos inteligentes (*chaincodes*) e sua integração direta com o *backend*, aproveitando a *stack* tecnológica do sistema e o uso do *Software Development Kit* (SDK) oficial do Fabric.
- **Modelo permissionado:** o Hyperledger Fabric permite configurar redes privadas com controle rigoroso de identidade e autenticação, assegurando que apenas entidades autorizadas possam validar, registrar ou consultar transações.
- **Arquitetura modular e escalável:** sua estrutura modular possibilita adaptar componentes como mecanismos de consenso, gerenciamento de identidades e controle de canais, favorecendo a personalização conforme os requisitos de segurança e desempenho do sistema.
- **Imutabilidade e auditabilidade das transações:** todas as operações registradas são permanentes, verificáveis e rastreáveis, reforçando a integridade das informações e possibilitando auditorias internas sem comprometer a confidencialidade dos dados criptografados.

No fluxo da aplicação, o *backend*, desenvolvido em Java Spring Boot e acessado por meio da CLI, é responsável por realizar a fragmentação e a criptografia de cada credencial cadastrada pelo usuário. Esse processo garante que nenhuma instância isolada da *blockchain* contenha a informação completa, mesmo em sua forma cifrada, reduzindo os riscos associados a eventuais comprometimentos de infraestrutura. Cada fragmento gerado é encapsulado em uma transação assinada digitalmente e enviada, por meio do

[25] `https://hyperledger-fabric.readthedocs.io/en/release-2.5/`
[26] `https://www.linuxfoundation.org`

Software Development Kit (SDK) oficial do Hyperledger Fabric, à respectiva rede permissionada, onde é registrado de forma permanente e imutável no *ledger*.

Esse processo é conduzido por contratos inteligentes (*chaincodes*) desenvolvidos em Java, que recebem os fragmentos, validam sua estrutura e os registram no *ledger* — termo que, no contexto de *blockchain*, refere-se ao livro-razão digital distribuído, responsável por armazenar todas as transações de forma permanente, imutável e auditável. Cada fragmento é tratado como um ativo digital independente dentro desse registro, o que garante rastreabilidade e integridade a cada operação de armazenamento realizada nas redes Hyperledger Fabric.

Como consequência, mesmo que um agente externo obtenha acesso não autorizado a uma das instâncias da *blockchain*, o conteúdo armazenado será criptograficamente insuficiente para reconstruir ou decifrar a credencial original. A segurança do modelo proposto baseia-se na combinação entre criptografia local, fragmentação dinâmica e armazenamento distribuído em duas redes privadas logicamente independentes do Hyperledger Fabric, o que reduz o ponto único de falha no armazenamento centralizado de credenciais, mitigando riscos decorrentes de comprometimentos parciais da infraestrutura, e reforça a confidencialidade dos dados em todas as etapas do processo.

Durante o processo de autenticação, o *backend* é novamente responsável por interagir diretamente com as duas instâncias da *blockchain* privada. As transações de leitura, executadas por meio do *Software Development Kit* (SDK) do Hyperledger Fabric, recuperam os fragmentos previamente armazenados, que são então reagrupados e decifrados localmente no ambiente controlado da aplicação. Todo esse procedimento é conduzido de forma autônoma pelo sistema, sem a necessidade de servidores intermediários, garantindo que os dados não sejam expostos nem em trânsito, nem em repouso.

A adoção desse modelo contribui de maneira significativa para a segurança e a confiabilidade da plataforma. Ao distribuir os fragmentos de dados entre duas redes permissionadas, logicamente independentes e operadas de forma isolada dentro do ambiente local, reduz significativamente a probabilidade de um ponto único de falha e reforça-se a resiliência frente a ataques, vazamentos ou tentativas de acesso não autorizado. Além disso, a auditabilidade nativa do Hyperledger Fabric permite rastrear todas as interações registradas no *ledger*, garantindo transparência operacional sem comprometer a confidencialidade das credenciais e dos metadados protegidos localmente.

3.2.8. Banco de dados

No contexto da solução proposta, o banco de dados local atua como uma camada de persistência auxiliar, responsável por armazenar metadados criptográficos e informações de referência que apoiam a operação da plataforma, sem reter qualquer forma de credencial do usuário. Essa estrutura complementa os dados distribuídos entre as *blockchains* privadas, garantindo acesso rápido e seguro aos elementos necessários para os processos de autenticação, reconstrução das credenciais e gerenciamento das identidades digitais no ambiente local.

A tecnologia escolhida para essa finalidade foi o H2 Database[27], um banco de

[27] `https://www.h2database.com/html/main.html`

dados relacional leve e embarcado, amplamente utilizado em aplicações que exigem simplicidade de configuração, portabilidade e rapidez no acesso local. Integrado ao *backend* desenvolvido em Spring Boot, o H2 é executado diretamente na aplicação, eliminando a necessidade de um servidor de banco de dados externo. Essa característica reduz a complexidade da infraestrutura, facilita a persistência automatizada por meio do Spring Data JPA e reforça a autonomia operacional do sistema no ambiente local.

Entre os fatores que motivaram essa escolha destacam-se a facilidade de uso e configuração, o suporte ao padrão SQL, o bom desempenho em operações locais de leitura e escrita e a adequação ao contexto de um protótipo seguro. Outro aspecto relevante é sua integração nativa com a linguagem Java, que simplifica a comunicação com a aplicação e reduz a necessidade de camadas adicionais de configuração. Além disso, o H2 oferece recursos de criptografia nativos, permitindo proteger o arquivo de banco de dados com senha e reforçar a segurança no armazenamento de metadados sensíveis. O suporte a diferentes modos de execução, como o funcionamento em memória, também contribui para a flexibilidade da arquitetura e facilita o uso da base de dados em ambientes controlados de teste.

No escopo atual do sistema, o banco de dados é responsável por armazenar, de forma cifrada e referencial (utilizando a mesma chave mestra de usuário gerada no registro), um conjunto mínimo — porém essencial — de metadados associados a cada usuário. Esses registros não correspondem às senhas originais, mas a elementos de suporte à gestão criptográfica e ao processo de reconstrução segura das credenciais. Entre eles, destacam-se:

- **Identificador do usuário:** definido como chave primária, garante a associação consistente entre os diferentes registros e facilita a recuperação eficiente das informações vinculadas;
- **Chave simétrica cifrada:** elemento central da identidade criptográfica do usuário, protegida por meio do algoritmo AES-256-GCM com vetor de inicialização (IV) aleatório, garantindo que apenas a senha pessoal informada no processo de autenticação permita sua recuperação;
- ***Hash* da chave simétrica:** utilizado como mecanismo de verificação de integridade e autenticidade da chave após a descriptografia, prevenindo adulterações ou corrupções de dados;
- **Segredo compartilhado (2FA) [planejado]:** armazenamento futuro do segredo compartilhado, a ser utilizado quando a autenticação em dois fatores for ativada, conforme o protocolo TOTP (*Time-based One-Time Password*);
- **Mapeamento da fragmentação:** estrutura de referência que descreve a estratégia aplicada para a fragmentação de cada credencial criptografada antes do envio às instâncias de *blockchain*;
- **Localização dos fragmentos:** identificadores e endereços das transações registradas nas diferentes *blockchains*, indispensáveis para a reconstrução segura das credenciais quando necessário.

A Figura 8 do Anexo A.5 ilustra o modelo lógico simplificado do banco de dados local (H2 Database), evidenciando as principais entidades e os campos correspondentes aos metadados descritos anteriormente. O esquema representa as tabelas `user` e `passwords_ref`, nas quais são armazenadas as informações associadas às identidades dos usuários, chaves criptográficas e referências às transações registradas nas duas

instâncias de *blockchain*. Essa estrutura reflete o caráter referencial do banco, que mantém apenas os elementos necessários para a reconstrução das credenciais, sem reter dados sensíveis em texto claro.

A adoção de um banco de dados local mostra-se estratégica para equilibrar segurança, desempenho e disponibilidade. Todas as informações sensíveis armazenadas são previamente cifradas na camada de aplicação, e sua integridade é verificada no momento da decifragem por meio do mecanismo de autenticação do modo GCM, impedindo que a exposição do banco represente qualquer comprometimento direto das credenciais dos usuários. Ao manter localmente os metadados criptográficos e as referências de fragmentação, a aplicação favorece rapidez no processo de autenticação e reconstrução das credenciais, mesmo em situações de instabilidade ou latência nas redes *blockchain* privadas. Essa arquitetura reforça a autonomia operacional do sistema, assegurando que operações locais, como autenticação do usuário e gestão de metadados, permaneçam funcionais mesmo diante de instabilidades temporárias nos componentes distribuídos. Ainda assim, a recuperação das credenciais depende do acesso simultâneo às duas instâncias da *blockchain*.

É importante destacar que o H2 Database oferece suporte à criptografia nativa do arquivo de banco (por exemplo, *CIPHER=AES*), recurso utilizado apenas como camada complementar, já que a proteção principal dos dados sensíveis ocorre na cifragem prévia realizada na aplicação, desenvolvida em Spring Boot, utilizando o algoritmo AES-256 no modo GCM e vetores de inicialização (IVs) aleatórios gerados por *SecureRandom*. Dessa forma, mesmo que o arquivo físico do banco seja obtido por terceiros, seu conteúdo permanece ilegível, assegurando uma camada adicional de proteção e reforçando a confidencialidade das informações armazenadas.

Outro aspecto relevante é a exigência de autenticação no acesso ao banco de dados: a abertura do arquivo requer credenciais válidas de usuário e senha, configuradas na própria aplicação. Esse controle de acesso, aliado às camadas de cifragem aplicacional e de arquivo, reforça a defesa em profundidade do sistema, dificultando tentativas não autorizadas de leitura ou manipulação das informações armazenadas.

A estratégia adotada, que combina ciframento em múltiplos níveis — na camada de aplicação e no próprio arquivo de banco —, autenticação obrigatória no acesso e persistência local segura, contribui de forma decisiva para a robustez da plataforma. Essa abordagem estabelece um modelo de defesa em profundidade, mitigando riscos associados a falhas de segurança e a tentativas de acesso indevido, ao mesmo tempo em que preserva a confidencialidade e a integridade dos metadados armazenados.

A Figura 9 do Anexo A.5 apresenta a arquitetura da camada de persistência de dados da aplicação, evidenciando a interação entre a interface de linha de comando (CLI), o *backend* desenvolvido em Spring Boot, o banco de dados local (H2 Database) e as instâncias de *blockchain* privadas baseadas na plataforma Hyperledger Fabric. O diagrama demonstra como as responsabilidades são distribuídas entre os componentes: o *backend* realiza a criptografia dos dados sensíveis, o mapeamento da fragmentação e a comunicação com o banco local e as *blockchains*; o H2 Database armazena metadados criptográficos e referências de fragmentação; e as redes *blockchain* mantêm, de forma imutável, os fragmentos cifrados das credenciais. Essa estrutura garante que cada ca-

mada da arquitetura contribua de maneira complementar para a segurança, integridade e disponibilidade das informações.

4. Testes e resultados

Esta seção apresenta o planejamento e a execução experimental realizados para avaliar a solução proposta quanto ao funcionamento, segurança, desempenho e resiliência. Os ensaios foram conduzidos em ambientes locais controlados e documentados de forma reprodutível, tendo como referência: (i) as características comparativas consolidadas no Quadro 1; e (ii) os requisitos funcionais e não funcionais definidos na fase de modelagem (Seção 3).

O objetivo é demonstrar que a arquitetura baseada em *criptografia local*, *fragmentação dupla* e *armazenamento distribuído em duas blockchains privadas* atende aos requisitos estabelecidos, mitigando o ponto único de falha associado ao armazenamento centralizado de credenciais.

4.1. Metodologia dos testes

A metodologia experimental foi definida para avaliar o comportamento da aplicação sob quatro dimensões — funcionamento, segurança, desempenho e resiliência — com foco na reprodutibilidade dos resultados. Os experimentos foram organizados nas seguintes categorias:

1. **Testes funcionais**: avaliam o correto funcionamento dos componentes do sistema, verificando se as operações de registro, fragmentação, armazenamento e recuperação ocorrem conforme o fluxo especificado na modelagem;
2. **Testes de segurança**: têm como foco avaliar a confidencialidade e a integridade das informações, assegurando que nenhum dado sensível seja armazenado ou transmitido em texto claro em qualquer etapa do fluxo;
3. **Testes de desempenho**: mensuram o tempo médio das operações críticas, como criptografia, fragmentação, gravação em *blockchain* e reconstrução das credenciais, avaliando a eficiência da aplicação sob diferentes volumes de dados;
4. **Testes de resiliência**: simulam falhas parciais — como a indisponibilidade temporária de uma das redes *blockchain* — para analisar o comportamento da plataforma diante de interrupções e avaliar sua capacidade de recuperação.

4.2. Ambientes e configuração experimental

Os experimentos foram realizados em dois ambientes locais distintos, configurados com a mesma estrutura de *software*, arquitetura de comunicação e componentes de rede, diferindo apenas quanto ao *hardware*. Essa separação permitiu avaliar o comportamento do protótipo sob diferentes capacidades de processamento, mantendo consistência tecnológica entre os testes.

Em ambos os casos, a aplicação (*backend* Java/Spring Boot) executou as operações de criptografia local (AES-256-GCM), fragmentação em dois fragmentos independentes e armazenamento distribuído desses fragmentos nas duas redes Hyperledger Fabric (`Org1` e `Org2`), além da persistência dos metadados no banco H2. O Quadro 4 resume as principais configurações técnicas empregadas em ambos os ambientes.

Quadro 4. Resumo dos ambientes de execução utilizados nos testes.

Componente	Ambiente A	Ambiente B
Finalidade	Testes funcionais, resiliência e desempenho	Testes de segurança
Sistema Operacional	Windows 11 Pro + WSL2 (Ubuntu 22.04 LTS)	Windows 10 Pro + WSL2 (Ubuntu 22.04 LTS)
Processador / Memória	Intel Core i5 / 8 GB RAM	AMD Ryzen 5 / 8 GB RAM
Softwares principais	Java 21, Maven 3.8, Docker 28.2, Hyperledger Fabric v2.5, H2 Database v2.2	Mesmos *softwares* e versões do Ambiente A
Rede *blockchain*	Duas organizações (`Org1MSP`, `Org2MSP`), dois canais (`mychannel`, `secondchannel`)	Estrutura idêntica

Ambos os ambientes seguiram o mesmo fluxo operacional, no qual a CLI local aciona o *backend* para execução das operações previstas, com registro das métricas de tempo (T_{enc}, T_{frag}, T_{store}, T_{rec}) por meio da classe utilitária `LogTimer`, responsável por padronizar as medições e garantir consistência entre execuções. A variação de *hardware* permitiu observar diferenças naturais de desempenho sem comprometer a integridade metodológica dos experimentos.

4.3. Avaliação funcional e comparativa

A avaliação funcional e comparativa foi conduzida com o objetivo de confirmar que o sistema desenvolvido cumpre os requisitos definidos na modelagem e apresenta o comportamento esperado quanto às operações fundamentais de proteção e gerenciamento das credenciais.

Foram definidos três Casos de Teste (CT), correspondentes às operações centrais do sistema: (i) cadastro e armazenamento de credenciais, (ii) recuperação e reconstrução dos dados armazenados e (iii) verificação da persistência cifrada no banco local. Os experimentos foram planejados com base nos requisitos funcionais definidos no **Quadro 2** e nos critérios técnicos apresentados no **Quadro 1**.

O Quadro 6, apresentado no Anexo A.6, sintetiza os resultados obtidos e as principais evidências registradas durante a execução dos testes funcionais.

No **CT01**, o sistema executou o ciclo completo de registro de usuário, cifragem da credencial, fragmentação em duas partes e armazenamento distribuído nas *blockchains* privadas das organizações `Org1` e `Org2`. Os *logs* confirmaram a geração da chave AES-256, a proteção da *User Master Key* (UMK) por meio de uma chave derivada da senha do usuário, o uso do modo autenticado *Galois/Counter Mode* (GCM) e a persistência dos metadados no banco H2. O teste foi repetido dez vezes consecutivas, desconsiderando-se a primeira execução por corresponder ao aquecimento da JVM e à inicialização das conexões com as redes. O tempo médio estabilizado das nove execuções subsequentes foi de aproximadamente 5,22 s, contemplando as etapas de cifragem, fragmentação e comunicação com ambas as redes Fabric.

O **CT02** avaliou a recuperação e reconstrução das credenciais armazenadas. O sistema obteve corretamente os fragmentos das duas redes Fabric, recompôs o conteúdo cifrado e decifrou a senha utilizada no teste, exibindo-a em texto claro apenas no terminal

local do usuário, conforme previsto pelo modelo de uso controlado. O tempo total registrado foi de cerca de 1,24 s, indicando estabilidade e eficiência do processo de leitura e recomposição.

Por fim, o **CT03** consistiu na inspeção direta do banco H2, confirmando que os campos `ENCRYPTED_DATA`, `IV` e `DATA_REF` armazenam apenas valores cifrados e metadados criptográficos. A Figura 10, apresentada no Anexo A.7, ilustra a evidência observada na consulta SQL, na qual o campo `ENCRYPTED_DATA` contém um objeto JSON com os parâmetros "`alg: AES-GCM-256`", "`iv`" e "`ciphertext`", sem qualquer texto legível.

Os resultados dos três testes funcionais confirmam que o protótipo opera de acordo com a modelagem definida, executando corretamente as etapas de cifragem, fragmentação, armazenamento distribuído e recuperação de credenciais. Em todos os casos, as informações permaneceram cifradas em repouso e em trânsito, sem qualquer exposição de dados sensíveis.

Além de demonstrar a aderência aos requisitos funcionais, os experimentos evidenciam que o modelo de armazenamento distribuído evita o ponto único de falha no armazenamento de credenciais, reforçando a resiliência do sistema sem comprometer a confidencialidade dos dados. Esses resultados fundamentam as análises de segurança, desempenho e resiliência apresentadas nas subseções seguintes.

4.4. Testes de segurança e resiliência

Os testes de segurança e resiliência foram conduzidos com o objetivo de avaliar a robustez da arquitetura proposta diante de cenários de vulnerabilidade, falhas e interrupções controladas. Essa etapa demonstrou que a combinação de *criptografia local*, *fragmentação dupla* e *armazenamento distribuído em duas blockchains privadas* assegura a confidencialidade, a integridade e a disponibilidade das credenciais, mesmo diante de eventos adversos. A análise abrangeu quatro dimensões:

(i) **Isolamento e indecifrabilidade dos fragmentos**: verificação de que cada parte da credencial armazenada nas *blockchains* é individualmente inútil e indecifrável;
(ii) **Proteção criptográfica e confidencialidade**: confirmação de que nenhuma informação sensível é armazenada ou transmitida em texto claro;
(iii) **Integridade e tolerância a falhas**: avaliação da resposta do sistema diante de indisponibilidade parcial das redes *blockchain*;
(iv) **Resiliência operacional e recuperação automática**: análise da capacidade de retomada das operações após falhas ou interrupções abruptas.

O **CT01 (isolamento de fragmento)** consistiu na execução manual do comando: `peer chaincode query -C mychannel -n passwordmanager -c '{"Args":["ReadPassword","user1_google"]}'`. O resultado retornou apenas um fragmento cifrado em formato hexadecimal, sem qualquer dado legível. A análise do *log* (`test-security-00-all.log`) confirmou que nenhuma parte da senha original foi exposta, comprovando a indecifrabilidade dos fragmentos individuais e a efetividade da fragmentação como camada de segurança.

Em seguida, no **CT02 (tentativa de reconstrução parcial)**, o *peer* da `Org2` foi desligado manualmente (`docker stop peer0.org2.example.com`)

para simular a perda temporária de uma das *blockchains*. Ao tentar recuperar uma credencial, o sistema retornou o erro controlado "`Fragmento ausente: reconstrução não permitida`". Após a reativação do *peer* (`docker start peer0.org2.example.com`), a recuperação foi concluída normalmente, demonstrando que a reconstrução só é possível com a presença de ambos os fragmentos íntegros, eliminando o ponto único de falha no armazenamento centralizado.

Por fim, o **CT03 (verificação de *logs* e banco local)** inspecionou os registros gerados pelo *Logback* e os dados persistidos no banco H2. Nenhuma informação sensível foi encontrada nos *logs*, que continham apenas mensagens de execução (*INFO*, *WARN*), e as tabelas do banco armazenavam exclusivamente *hashes*, identificadores e metadados de fragmentação. Esses resultados confirmam a conformidade com o princípio de *confidencialidade por design*, assegurando a proteção dos dados mesmo nas camadas internas de auditoria.

O Quadro 7, apresentado no Anexo A.8, sintetiza os resultados obtidos nos três casos de teste de segurança.

Os tempos observados nesses testes possuem caráter apenas descritivo, não sendo tratados como métricas formais de desempenho, uma vez que o foco dessa etapa está no comportamento seguro e na tolerância a falhas do sistema.

Essas evidências são corroboradas pela Figura 10, no Anexo A.7, que apresenta a visualização parcial do campo `ENCRYPTED_DATA` no banco H2, evidenciando que apenas dados criptografados são persistidos, sem informações legíveis.

Os testes de resiliência tiveram como objetivo avaliar a capacidade do sistema de manter a integridade dos dados e retomar o funcionamento normal após falhas parciais ou interrupções abruptas.

No **CT01 (falha temporária de *peer* da Org2)**, a `Org2` foi desligada durante o processo de armazenamento de uma nova credencial. A tentativa falhou de forma controlada (*ServiceDiscoveryException*), sem corromper o banco H2. Após a reativação do *peer*, a operação foi concluída normalmente, evidenciando consistência e resiliência da arquitetura. O tempo médio de recuperação registrado foi de aproximadamente 6,5 s.

No **CT02 (interrupção do *backend* durante gravação)**, simulou-se a interrupção abrupta da aplicação entre o início da operação e a submissão do primeiro fragmento à *blockchain*. Após a reinicialização, não foram identificados registros parciais no banco H2, e uma nova tentativa de cadastro foi realizada com sucesso, demonstrando integridade transacional e ausência de efeitos colaterais.

Por fim, no **CT03 (reconexão e recuperação automática)**, o *peer* da `Org2` foi desligado durante uma operação ativa e posteriormente reativado. O *backend* manteve-se funcional, registrou o erro como "*tentativa de comunicação falha – fragmento ausente*" e retomou automaticamente a comunicação após o restabelecimento da rede. O tempo médio de reconexão foi de aproximadamente 5,9 s, sem perda de dados ou necessidade de reinicialização manual.

O Quadro 8, no Anexo A.9, sintetiza os resultados dos testes de resiliência.

Os seis experimentos realizados nos testes de segurança e resiliência comprova-

ram que o sistema mantém a confidencialidade, integridade e disponibilidade das credenciais em todas as camadas de operação. Nenhum dado sensível foi registrado em texto claro em qualquer etapa, e as falhas foram tratadas de forma controlada, com recuperação automática após eventos adversos.

Os resultados indicam que o modelo de armazenamento fragmentado e distribuído em múltiplas *blockchains* mitiga o ponto único de falha associado a repositórios centralizados de credenciais, contribuindo para a continuidade operacional. Além disso, a política de *logging* seguro e a persistência exclusiva de metadados cifrados no banco local reforçam a conformidade com os princípios da tríade de segurança da informação (confidencialidade, integridade e disponibilidade).

Em síntese, os testes de segurança e resiliência confirmam empiricamente a solidez da proposta, evidenciando que a solução é capaz de resistir a falhas parciais, prevenir a exposição de dados sensíveis e preservar a consistência dos registros em todas as camadas do sistema.

4.5. Testes de desempenho e eficiência

Os testes de desempenho e eficiência foram conduzidos com o propósito de avaliar o comportamento temporal do sistema em diferentes níveis de carga, mensurando o tempo médio das operações críticas — cifragem, fragmentação, armazenamento distribuído e recuperação de credenciais.

A avaliação baseou-se em três cenários experimentais, executados de forma automática por meio da classe `PerformanceTestInitializer`, que reproduziu o ciclo completo de uso (*registro* $\rightarrow$ *login* $\rightarrow$ *armazenamento* $\rightarrow$ *recuperação* $\rightarrow$ *logout*) múltiplas vezes em sequência controlada. As medições foram obtidas com auxílio da classe utilitária `LogTimer`, responsável por registrar os tempos parciais de cada etapa:

- T_{enc} — tempo médio de cifragem AES-256-GCM;
- T_{frag} — tempo médio de fragmentação da credencial;
- T_{store} — tempo médio de armazenamento dos fragmentos nas duas redes Fabric;
- T_{rec} — tempo médio de recuperação e reconstrução completa da credencial.

Cada cenário foi repetido dez vezes consecutivas, sendo desconsiderada a primeira execução (responsável pelo *warm-up* da JVM, inicialização do cache criptográfico e estabelecimento das primeiras conexões gRPC com a rede). As médias foram calculadas a partir das nove execuções subsequentes, registradas nos arquivos de *log* correspondentes. Ressalta-se que, devido ao número reduzido de execuções por cenário e ao caráter exploratório dos testes, a análise baseou-se nas médias observadas, sem o emprego de métricas estatísticas de dispersão, como desvio-padrão, intervalos de confiança ou percentis.

O **cenário A (carga unitária)** caracteriza-se pela inserção e recuperação de uma única credencial, destinada a medir o tempo base de operação do sistema. O **cenário B (carga moderada)** caracteriza-se pelo processamento de dez credenciais, simulando um uso cotidiano e verificando a estabilidade sob carga intermediária. Por fim, o **cenário C (carga ampliada)** caracteriza-se pela inserção e recuperação de cinquenta credenciais, utilizado para observar o comportamento temporal do protótipo sob um volume maior de operações consecutivas. O Quadro 5 apresenta as médias consolidadas obtidas a partir dos três cenários de teste.

Quadro 5. Médias de tempo obtidas nos testes de desempenho.

Cenário	T_{enc} **(ms)**	T_{frag} **(ms)**	T_{store} **(s)**	T_{rec} **(s)**	T_{total} **por credencial (s)**	**Tempo total estimado** (s)
A (1 cred)	7	10	15,1	1,7	16,8	16.8
B (10 creds)	2	1	6,7	1,7	8,4	84.0
C (50 creds)	1,19	0,62	6,49	1,52	8,01	400,0

O tempo total estimado de cada cenário foi calculado a partir das médias das operações (T_{enc}, T_{frag}, T_{store} e T_{rec}), obtidas nas dez execuções, multiplicadas pelo número de credenciais processadas. Esse cálculo é válido porque, no protótipo, cada credencial é tratada de forma independente e as operações são executadas sequencialmente, sem paralelismo. Assim, o valor estimado representa o tempo aproximado necessário para concluir todo o cenário, enquanto as médias refletem o comportamento interno de cada etapa.

Nos três cenários, observou-se estabilidade nos tempos de cifragem e fragmentação, que permaneceram constantes na faixa de milissegundos, independentemente do volume de dados processados. O tempo de armazenamento em *blockchain* (T_{store}) apresentou variação proporcional ao número de credenciais, representando a maior parcela do tempo total de operação no contexto do protótipo desenvolvido, em razão da necessidade de submissão de transações, validação pelos *peers* e ordenação pelo serviço de *ordering* do Fabric. Já o tempo médio de recuperação (T_{rec}) manteve-se estável, entre 1,5 s e 1,7 s, confirmando a eficiência da recomposição e decifragem das credenciais.

Os resultados obtidos sugerem que o aumento da carga (de 1 para 50 credenciais) está associado a um crescimento proporcional do tempo total de execução, indicando uma tendência de comportamento previsível em função do número de operações, dentro do escopo experimental avaliado. Durante as execuções prolongadas, não foram observados erros, desconexões ou degradações críticas, o que indica um comportamento estável do sistema no contexto e nas condições avaliadas.

A análise detalhada dos *logs* indicou que a primeira conexão com cada organização da rede Fabric é significativamente mais custosa (cerca de 12 s), pois envolve a fase inicial de *service discovery* (descoberta automática dos *peers* e do *orderer*) e o carregamento do MSP (*Membership Service Provider*), executados apenas na primeira sessão de comunicação com a rede. Após essa etapa inicial, as operações subsequentes reutilizam conexões gRPC já estabelecidas e estruturas de cache do SDK (como canal, identidade e roteamento), reduzindo o tempo médio de armazenamento para o intervalo de 6 s a 8 s. Essa característica contribui para a estabilidade dos tempos observados nos cenários B e C.

A tendência de crescimento consistente em função do número de operações observada nos tempos médios sugere que a sobrecarga introduzida pelos mecanismos de segurança — criptografia local e fragmentação dupla — é pequena quando comparada ao custo da operação distribuída em *blockchain*, indicando um equilíbrio favorável entre segurança e desempenho no contexto experimental analisado. Resultados mais conclusivos sobre variabilidade, percentis e comportamento sob carga elevada exigiriam um conjunto ampliado de execuções e métricas estatísticas adicionais, configurando uma pos-

sibilidade natural de aprofundamento metodológico em trabalhos futuros.

5. Conclusão e trabalhos futuros

Após a execução dos testes descritos na seção anterior, observou-se que os experimentos evidenciaram a **viabilidade técnica e prática da arquitetura proposta**, demonstrando que o uso combinado de *criptografia local*, *fragmentação dupla* e *armazenamento distribuído em duas blockchains privadas* constitui uma alternativa sólida frente às limitações típicas de abordagens centralizadas de gerenciamento de credenciais. A aplicação executou de forma estável todas as operações críticas — registro, cifragem, fragmentação, armazenamento, recuperação e *logout* — sem erros ou inconsistências observadas nos *logs*, demonstrando a execução completa e consistente do ciclo de funcionamento previsto na modelagem.

Os **testes de segurança** indicaram que não foi identificada a exposição de credenciais, chaves criptográficas ou vetores de inicialização em texto claro em qualquer camada do sistema. Os fragmentos armazenados em cada *blockchain* revelaram-se individualmente inúteis e indecifráveis, contribuindo para o isolamento e a proteção do dado original. No eixo de **resiliência**, os testes indicaram que a arquitetura é capaz de manter sua integridade mesmo diante de falhas parciais controladas, como a indisponibilidade temporária de um *peer* ou interrupções abruptas no *backend*. Em todos os cenários avaliados, o sistema recuperou-se de forma consistente, sem necessidade de intervenção manual e sem perda de dados.

Os **testes de desempenho** evidenciaram que a sobrecarga introduzida pelos mecanismos de segurança é mínima: as operações de cifragem e fragmentação permaneceram na ordem de milissegundos, enquanto os tempos mais elevados concentraram-se nas transações distribuídas em *blockchain*, características inerentes a esse modelo arquitetural. O comportamento geral foi previsível dentro do escopo experimental, apresentando uma tendência aproximadamente proporcional, mesmo em cenários de carga aumentada (50 credenciais consecutivas), indicando potencial de escalabilidade dentro do escopo experimental avaliado.

Com base nesses resultados, as implicações do trabalho podem ser analisadas sob duas perspectivas complementares:

- **Perspectiva técnica**: os experimentos evidenciaram um equilíbrio satisfatório entre segurança e desempenho. A fragmentação dupla e o uso de duas redes Fabric acrescentaram camadas de proteção sem causar degradação significativa no tempo total de operação. A latência observada — concentrada na escrita distribuída — manteve-se estável ao longo das execuções, reforçando a eficiência da arquitetura de armazenamento distribuído e sua aderência aos requisitos definidos na modelagem.
- **Perspectiva prática**: a solução mostrou-se tecnicamente viável para cenários reais de gerenciamento de credenciais, sobretudo em ambientes que exigem alto nível de confiabilidade. O modelo de armazenamento distribuído mitigou o ponto único de falha característico de repositórios centralizados, assegurando que nenhuma das infraestruturas de armazenamento, isoladamente, detenha os fragmentos necessários para reconstruir o dado completo. A execução local, aliada ao

controle direto pelo usuário, reforça a privacidade e a autonomia sobre as próprias credenciais.

Além dos resultados positivos, algumas oportunidades de aprimoramento foram identificadas, abrindo espaço para a evolução da plataforma. Entre as melhorias potenciais destacam-se:

- **Implementação completa de autenticação multifator (2FA)**, integrando o protocolo TOTP ao fluxo de *login* para elevar a robustez contra acessos não autorizados;
- **Uso de funções de derivação de chave mais robustas**, como PBKDF2 ou Argon2, fortalecendo a resistência contra ataques de força bruta;
- **Integração de auditoria automatizada via *smart contracts***, permitindo verificação contínua da integridade dos fragmentos e do fluxo transacional nas redes Fabric;
- **Expansão dos testes para ambientes distribuídos reais**, com múltiplos nós Fabric em rede externa, a fim de mensurar latência, *throughput* e tolerância a falhas em cenários corporativos;
- **Investigação de técnicas avançadas de fragmentação**, como esquemas de divisão criptográfica (por exemplo, *Shamir's Secret Sharing*), para aumentar a resiliência e reduzir correlações estruturais;
- **Desenvolvimento de uma interface gráfica (GUI)** multiplataforma, ampliando a usabilidade sem comprometer a lógica *local-first* da aplicação;
- **Projetos de mecanismos de recuperação de conta**, explorando abordagens seguras para restauração do acesso criptográfico do usuário em cenários de perda de credenciais, sem comprometer o modelo de confidencialidade e autenticação adotado.
- **Aprimoramento da escalabilidade da solução**, por meio da distribuição dos nós Fabric em máquinas distintas ou ambientes de nuvem privada, possibilitando avaliar o comportamento do sistema em cenários multiusuário e sob cargas elevadas, sem alterar o modelo conceitual proposto.
- **Incorporação de mecanismos de *backup* e recuperação do ambiente local**, explorando estratégias seguras para preservação dos metadados criptográficos e das referências de fragmentação, de forma a mitigar riscos associados a falhas físicas no dispositivo do usuário, sem comprometer o modelo de confidencialidade e controle local adotado pela arquitetura.
- **Adoção de mecanismos de limitação de requisições (*rate limiting*)**, visando mitigar tentativas de abuso, automação maliciosa ou ataques de força bruta ao *backend*, particularmente em cenários futuros de exposição em rede ou uso multiusuário.

Em síntese, as evidências empíricas obtidas reforçam a contribuição científica e tecnológica deste trabalho ao demonstrar que a arquitetura desenvolvida é capaz de unir princípios de **privacidade**, **controle local** e **armazenamento distribuído** em um modelo funcional, seguro e estável de gerenciamento de credenciais. Os resultados experimentais indicaram que a solução reduz significativamente o ponto único de falha associado ao armazenamento centralizado de senhas e oferece um nível de proteção mais elevado no que se refere ao armazenamento e à manipulação criptográfica das credenciais, quando comparada a abordagens tradicionais baseadas em repositórios únicos.

Referências

Akamai Technologies (2024). What are cloud databases? Acessado em 18 de maio de 2025.

Alanoud, A., Rawabi, A., e Mounir, F. (2024). Prominent security vulnerabilities in cloud computing. *International Journal of Advanced Computer Science and Applications*, 15(2).

Amazon Web Services (2024). O que é armazenamento em nuvem? - aws. Acessado em 28 de abril de 2025.

Androutsellis-Theotokis, S. e Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371.

Araujo, C. A., Moreira, A., e Bernardino, J. (2009). A framework to detect and avoid ddos attacks in p2p networks. In *2009 Fourth International Conference on Systems and Networks Communications*. IEEE.

Atzei, N., Bartoletti, M., e Cimoli, T. (2017). A survey of attacks on ethereum smart contracts (sok). In *International Conference on Principles of Security and Trust (POST)*, pages 164–186, Paris, France. Springer.

AVG Technologies (2022). What is blockchain technology? Acessado em 26 de maio de 2025.

Benet, J. (2014). Ipfs – content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561. Acessado em 17 de maio de 2025.

Blount, A. (2025). Understanding the 23andme data breach and ensuring cybersecurity. Acessado em 8 de abril de 2025.

Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J. A., e Felten, E. W. (2015). Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *IEEE Symposium on Security and Privacy*, pages 104–121, San Jose, CA. IEEE.

Bonneau, J., Preibusch, S., e Anderson, R. (2012). The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy*, pages 553–567. IEEE.

Buterin, V. (2014). Ethereum whitepaper. Ethereum Project. Acessado em 4 de maio de 2025.

Chatzoglou, E., Kampourakis, V., Tsiatsikas, Z., Karopoulos, G., e Kambourakis, G. (2024). Keep your memory dump shut: Unveiling data leaks in password managers. *arXiv preprint arXiv:2404.00423*.

Choong, Y.-Y., Greene, M. M., e Theofanos, M. F. (2014). User perceptions on usability and security of personal authentication behavior. Technical Report NIST IR 7983, National Institute of Standards and Technology. Acessado em 4 de maio de 2025.

Christidis, K. e Devetsikiotis, M. (2016). Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303.

CRA News Service (2024). New vulnerability exposes yubikey 5 devices to cloning attacks. Acessado em 4 de maio de 2025.

Crosby, M., Pattanayak, N., Verma, S., e Kalyanaraman, V. (2016). Blockchain technology: Beyond bitcoin. *Applied Innovation Review*, 2:6–19.

Dashlane (2024). Hardware security key purpose, benefits & use cases. Acessado em 4 de maio de 2025.

Dassword (2025). Dassword: The first decentralized password manager. Site Oficial. Acessado em 13 de maio de 2025.

Demartini, F. (2022). Lastpass confirma vazamento de senhas criptografadas dos usuários. Acessado em 4 de maio de 2025.

ETHGlobal (2023). Polypass: Decentralized password manager using polybase. ETHGlobal Showcase. Acessado em 13 de maio de 2025.

European Union Agency for Cybersecurity (ENISA) (2023). Enisa threat landscape report 2023. `https://www.enisa.europa.eu/sites/default/files/publications/ENISA%20Threat%20Landscape%202023.pdf`. Relatório técnico oficial. Acessado em 9 de outubro de 2025.

Eveleigh, P. (2019). Hacking hardware password managers: The reczone. Acessado em 4 de maio de 2025.

Gautam, A., Seamons, K., Yadav, T. K., e Ruoti, S. (2024). Passwords are meant to be secret: A practical secure password entry channel for web browsers.

GetUSCart (2025). Reczone password safe vault electronic storage organizer keeper device and eva carry case bundle. Acessado em 4 de maio de 2025.

Greenberg, A. (2022). Lastpass breach exposes password vault metadata. *Wired*.

Gudgeon, L., Moreno-Sanchez, P., Leonardos, S., Gervais, A., e Knottenbelt, W. J. (2019). Sok: Layer-two blockchain protocols. Cryptology ePrint Archive, Paper 2019/360. Acessado em 5 de maio de 2025.

Haridhas, R. (2019). Locketh: Ethereum-based password dapp. GitHub Repository. Acessado em 13 de maio de 2025.

HashiCorp (2019). *Vault: Securing Secrets and Protecting Sensitive Data*.

IBM (2024). IBM report: Escalating data breach disruption pushes costs to new highs.

IBM Security (2023). Cost of a data breach report 2023. Technical report, IBM Corporation. Acessado em 28 de abril de 2025.

Investopedia (2025). Blockstack (stacks). Investopedia. Acessado em 13 de maio de 2025.

Jackson, C. e Boneh, D. (2014). Password managers: Attacks and defenses. In *23rd USENIX Security Symposium*, pages 447–461. USENIX Association.

JumpCloud (2022). Password management architecture matters. Acessado em 3 de maio de 2025.

Kosba, A., Miller, A., Shi, E., Wen, Z., e Papamanthou, C. (2016). Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 839–858, San Jose, CA. IEEE.

Kovacs, E. (2025). Oracle confirms cloud hack. `https://www.securityweek.com/oracle-confirms-cloud-hack/`. Acessado em 8 de abril de 2025.

Li, H., Liang, X., Zhao, J., Dong, M., e Shen, X. S. (2020). Blockchain-based data preservation system for medical data. *IEEE Access*, 8:59389–59401.

Li, Z., He, W., Akhawe, D., e Song, D. (2014). The emperor's new password manager: Security analysis of web-based password managers. In *23rd USENIX Security Symposium*, pages 465–479. USENIX Association.

LNCC (2023). P2p - distribuição de arquivos. Acessado em 4 de maio de 2025.

LogaP (2024). Sistemas distribuídos: O que são, vantagens e desafios. Acessado em 4 de maio de 2025.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Acessado em 4 de maio de 2025.

Narayanan, A., Bonneau, J., Felten, E., Miller, A., e Goldfeder, S. (2016). *Bitcoin and Cryptocurrency Technologies*. Princeton University Press, Princeton, NJ.

NordVPN (2024). O que é p2p? entenda como funcionam as conexões ponto a ponto. Acessado em 3 de maio de 2025.

One Identity (2025). One identity safeguard: Privileged access management. Site Oficial. Acessado em 13 de maio de 2025.

Pereira, C. A. C. (2004). Redes peer-to-peer: Estudo sobre arquitetura, aplicações e desafios. `https://lume.ufrgs.br/handle/10183/12008`. Universidade Federal do Rio Grande do Sul. Trabalho de conclusão de curso.

Rathod, B., Zagade, T., Parkhe, N., e Waghmare, P. (2025). Cloud security vulnerabilities: How to identify and address risks. *International Journal of Multidisciplinary and Scientific Emerging ResearcH*, 13:1–20.

Sharper Image (2013). Password safe user manual. `https://cdn4.sharperimage.com/si/pdf/manuals/201776.pdf`. Acessado em 4 de maio de 2025.

Siegel, D. (2016). Understanding the dao attack. Acessado em 17 de maio de 2025.

Silverback Consulting (2025). Is your password manager actually safe? Acessado em 4 de maio de 2025.

Smith, T., Ruoti, S., e Seamons, K. (2018). Augmenting centralized password management with application-specific passwords. In *USENIX Symposium on Usable Privacy and Security (SOUPS)*. USENIX.

Stacks Project (2025). Gaia: Stacks storage system. Stacks Documentation. Acessado em 13 de maio de 2025.

Szabo, N. (1994). Smart contracts: Automated transaction protocols. Manuscrito disponível online via Fon.Hum.Uva.nl. Acessado em 17 de maio de 2025.

Tanenbaum, A. S. e Bos, H. (2014). *Modern Operating Systems*. Pearson, 4 edition.

Tapscott, D. e Tapscott, A. (2016). *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World*. Penguin, New York, NY.

Team, T. (2024). Centralized vs decentralized identity management explained. Acessado em 29 de abril de 2025.

TeamPassword (2021). What happened with the cam4 data leak? Acessado em 8 de abril de 2025.

Tiansun (2012). How to use the rec zone password vault. Acessado em 4 de maio de 2025.

Verizon Business (2023). 2023 data breach investigations report. Acessado em 8 de abril de 2025.

Winder, D. (2025). Yubico issues security advisory as 2fa bypass vulnerability confirmed. Acessado em 4 de maio de 2025.

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. Yellow paper, Ethereum Foundation. Acessado em 5 de maio de 2025.

Yubico (2024a). Security advisory ysa-2024-02. Acessado em 4 de maio de 2025.

Yubico (2024b). Yubikey 5 series overview. `https://www.yubico.com/products/yubikey-5-overview/`. Acessado em 4 de maio de 2025.

Yubico (2024c). Yubikey technical manual. `https://docs.yubico.com/hardware/yubikey/yk-tech-manual/`. Acessado em 4 de maio de 2025.

Zheng, Z., Xie, S., Dai, H.-N., Chen, X., e Wang, H. (2017). An overview of blockchain technology: Architecture, consensus, and future trends. *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 557–564.

Zuo, C., Lin, Z., e Zhang, Y. (2019). Why Does Your Data Leak? Uncovering the Data Leakage in Cloud from Mobile Apps. *IEEE Symposium on Security and Privacy (SP)*, pages 1–3.

Zyskind, G., Nathan, O., e Pentland, A. (2015). Decentralizing privacy: Using blockchain to protect personal data. In *Proceedings of the 2015 IEEE Security and Privacy Workshops (SPW)*, pages 180–184. IEEE. Acessado em 5 de maio de 2025.

A. Apêndice

A.1. Arquiteturas centralizadas

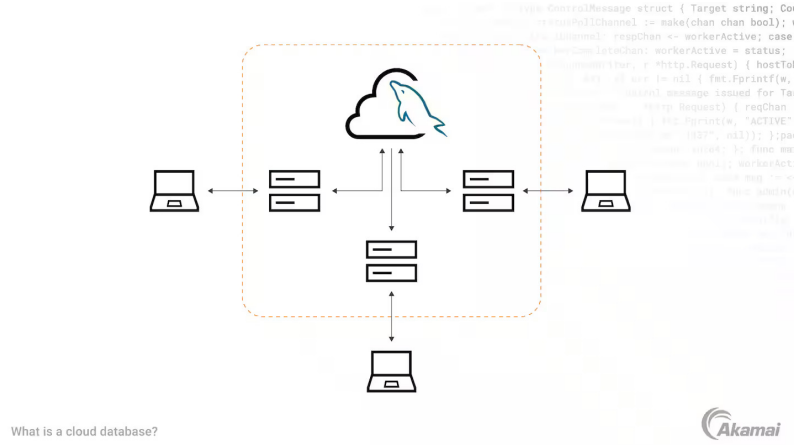

Figura 3. Banco de dados em nuvem. Fonte: Akamai Technologies (2024).

A.2. Fluxos de sincronização em gerenciadores nativos

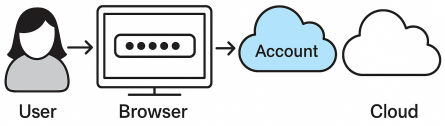

Figura 4. Fluxo de sincronização de senhas em gerenciadores nativos.

A.3. Arquiteturas centralizadas corporativas

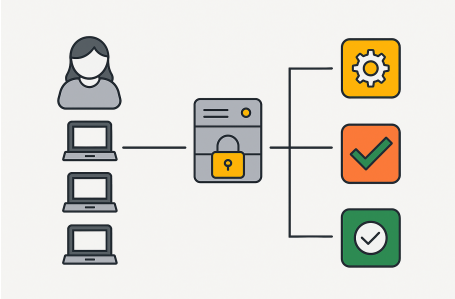

Figura 5. Arquitetura centralizada de gerenciamento de credenciais.

A.4. Arquitetura proposta

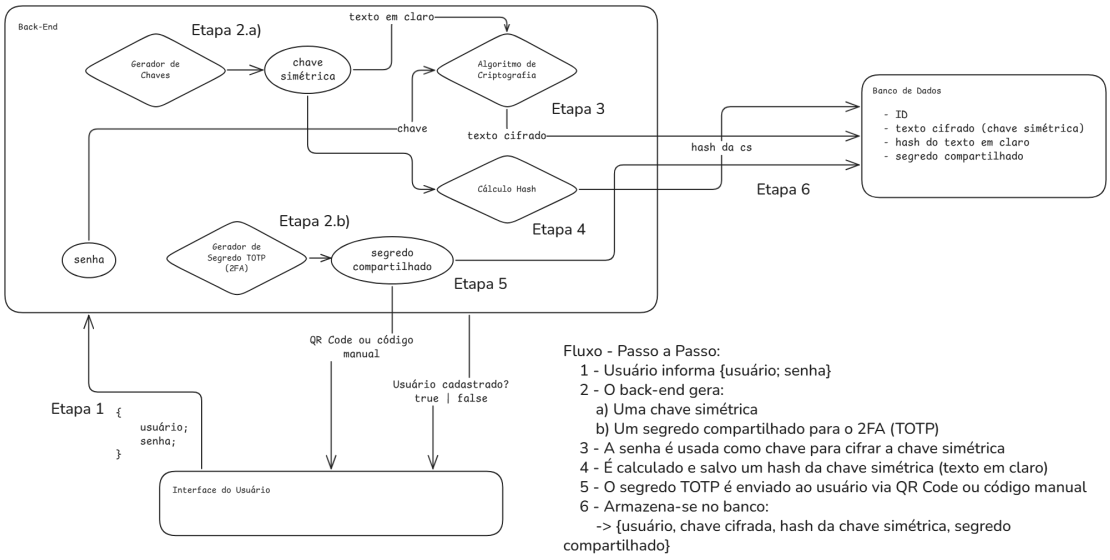

Figura 6. Fluxo de cadastro: etapas de geração, cifragem e armazenamento das credenciais.

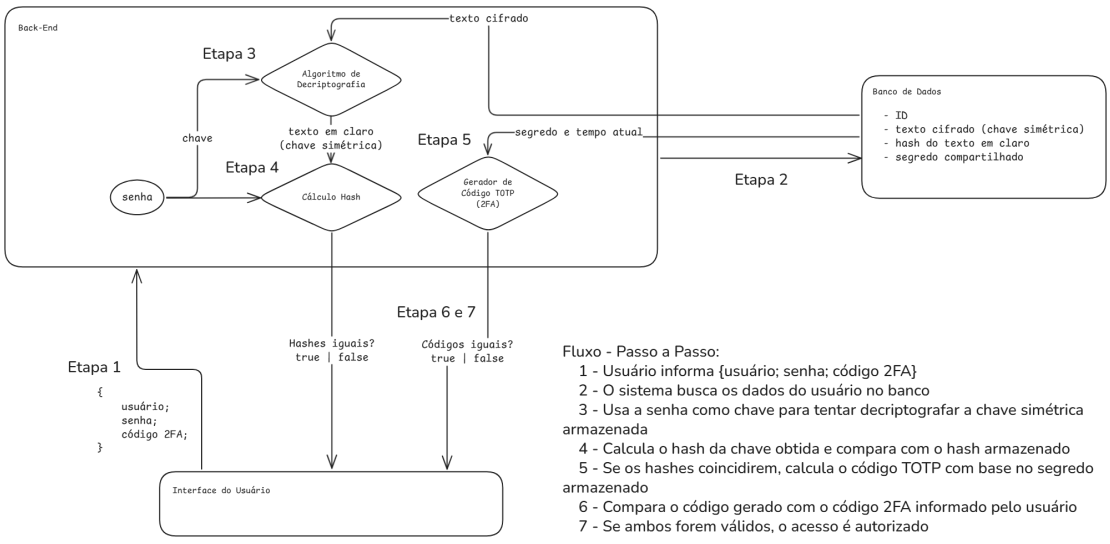

Figura 7. Fluxo de autenticação: validação da senha e do código 2FA.

A.5. Modelagem do banco de dados

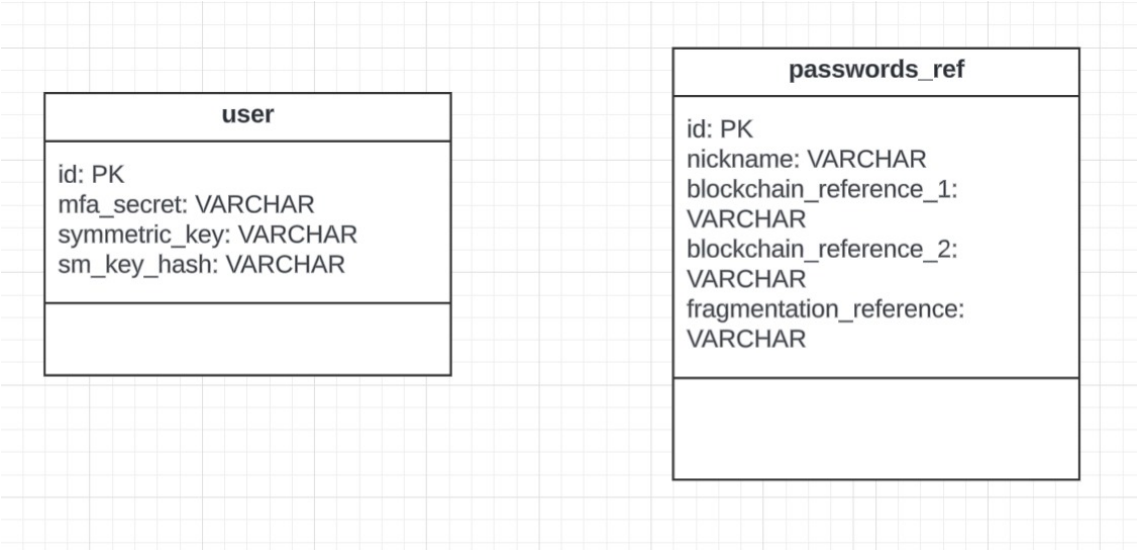

Figura 8. Modelo lógico simplificado das entidades armazenadas no banco de dados local.

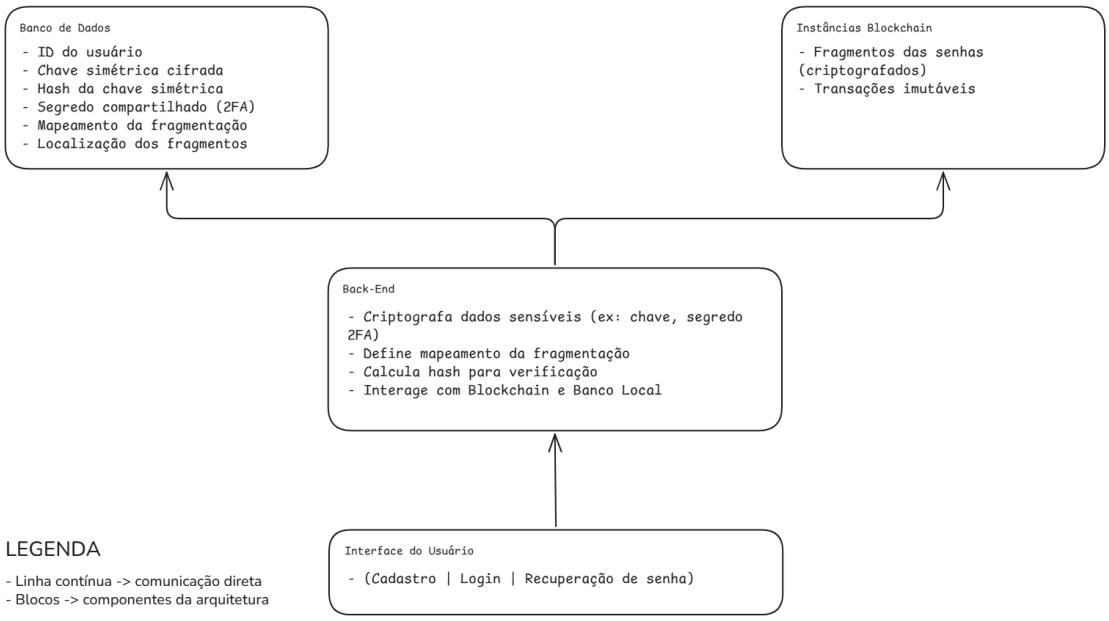

Figura 9. Arquitetura de persistência de dados da aplicação.

A.6. Resultados detalhados dos testes funcionais

Quadro 6. Testes funcionais executados no Ambiente A.

CT	Descrição	Evidência	Resultado	Tempo Total
CT01	Cadastro e armazenamento de credencial	test-functional-01-register.log	Sucesso	5,22 s
CT02	Recuperação e decifragem de credencial	test-functional-02-recover.log	Sucesso	1,24 s

CT03	Verificação da persistência cifrada (H2)	h2-verificacao.png	Sucesso	–

A.7. Evidências de dados cifrados no banco H2

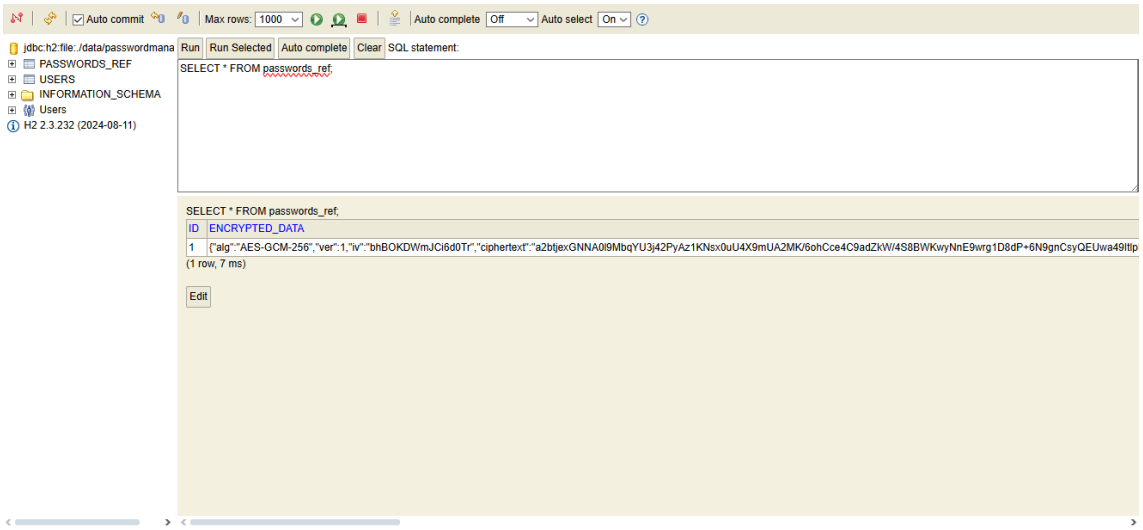

Figura 10. Visualização parcial do campo `ENCRYPTED_DATA` cifrado no banco H2.

A.8. Resultados detalhados dos testes de segurança

Quadro 7. Testes de segurança executados no Ambiente B.

CT	Descrição	Evidência	Resultado	Tempo Total
CT01	Isolamento do fragmento	test-security-00-all.log	Sucesso	0,8 s
CT02	Tentativa de reconstrução parcial	test-security-00-all.log	Sucesso	3,2 s
CT03	Verificação de logs e banco local	test-security-00-all.log	Sucesso	1,1 s

A.9. Resultados detalhados dos testes de resiliência

Quadro 8. Testes de resiliência executados no Ambiente A.

CT	Descrição	Evidência	Resultado	Tempo Total
CT01	Falha temporária de *peer* da Org2	test-resilience-01-falha-peer.log	Sucesso	6,5 s
CT02	Interrupção do *backend* durante gravação	test-resilience-02-interrupcao.log	Sucesso	5,9 s
CT03	Reconexão e recuperação automática	test-resilience-03-reconexao.log	Sucesso	5,9 s