

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CIÊNCIA DA COMPUTAÇÃO

Luan Diniz Moraes

**Cadeia de Custódia Forense Usando Hyperledger Fabric e Controle de Acesso via
Hyperledger Identities**

Florianópolis
2025

Luan Diniz Moraes

Cadeia de Custódia Forense Usando Hyperledger Fabric e Controle de Acesso via Hyperledger Identus

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em Ciência da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientadora: Prof^a. Carla Merkle Westphall, Dr^a.

Coorientador: Prof. Leandro Loffi, Dr.

Florianópolis

2025

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Diniz Moraes, Luan
Cadeia de Custódia Forense Usando Hyperledger Fabric e
Controle de Acesso via Hyperledger Identity / Luan Diniz
Moraes ; orientadora, Carla Merkle Westphall,
coorientador, Leandro Loffi, 2025.
131 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Centro Tecnológico,
Graduação em Ciências da Computação, Florianópolis, 2025.

Inclui referências.

1. Ciências da Computação. 2. Cadeia de Custódia. 3.
Identidade Autossobrevivente. 4. Blockchain. 5. Segurança da
Informação. I. Merkle Westphall, Carla. II. Loffi, Leandro.
III. Universidade Federal de Santa Catarina. Graduação em
Ciências da Computação. IV. Título.

Luan Diniz Moraes

**Cadeia de Custódia Forense Usando Hyperledger Fabric e Controle de Acesso via
Hyperledger Identus**

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo curso de Graduação em Ciência da Computação.

Florianópolis, 29 de setembro de 2025.

Prof. Dr. Renato Cislighi
Coordenador do Curso

Banca Examinadora:

Prof^ª. Carla Merkle Westphall, Dr^a.
Orientadora
Universidade Federal de Santa Catarina

Prof. Leandro Loffi, Dr.
Coorientador
Universidade Federal de Santa Catarina

Cristian Alves dos Santos, Me.
Avaliador
Universidade Federal de Santa Catarina

Dedico este trabalho aos meus pais.

*"Welcome on board, over here is the ship of your life
So rotten that will cast away
I'll be your sweet lullaby all night
And if you get lost, you can hold my hand
And I'll be here when fire burns (inside your heart)
Climb up the hills and mountains
Don't forget what you've learned
Life makes us feel the time we cannot hold
Time makes us live a tale already told
Time makes us heal a feeling inside
A feeling that lies in our hearts that we stole away"
Time — Angra
(álbum Angels Cry, 1993)*

RESUMO

As evidências digitais são mais voláteis e suscetíveis à adulteração do que as físicas. Nesse contexto, a preservação da integridade e autenticidade da cadeia de custódia é essencial para a validade jurídica destes vestígios. Tendo isso em vista, tecnologias como Identidade Autossobrerana e *blockchain* apresentam características relevantes que podem ser aplicadas a esse cenário. Embora existam estudos sobre esses temas, existem lacunas a serem exploradas, principalmente na integração entre Identidade Autossobrerana, *blockchain* e cadeia de custódia.

Neste trabalho, foram estudados os fundamentos teóricos dessas áreas e desenvolvida uma aplicação prática utilizando o Hyperledger Fabric e o Hyperledger Identus, introduzindo novas abordagens que contribuem para o avanço do estado da arte. A Identidade Autossobrerana foi aplicada em conjunto com provas de conhecimento zero para controlar o acesso de peritos a um sistema de gerenciamento de cadeias de custódia de evidências digitais.

Foi construída uma prova de conceito onde a cadeia de custódia é modelada como um encadeamento de credenciais verificáveis, armazenando metadados críticos *on-chain* e dados sensíveis *off-chain*. O sistema implementou um controle de acesso granular para peritos, utilizando credenciais AnonCreds e provas de conhecimento zero. Isso permitiu validar níveis de permissão específicos sem expor desnecessariamente os dados dos peritos.

Palavras-chave: Cadeia de Custódia. Identidade Autossobrerana. *Blockchain*. Segurança da Informação.

SUMÁRIO

1	INTRODUÇÃO	15
1.1	MOTIVAÇÃO	15
1.2	OBJETIVOS	16
1.2.1	Objetivo Geral	16
1.2.2	Objetivos Específicos	16
1.3	ORGANIZAÇÃO DO TEXTO	16
2	FUNDAMENTAÇÃO TEÓRICA	19
2.1	BLOCKCHAIN	19
2.1.1	Definição	19
2.1.2	Breve Histórico	19
2.1.3	<i>Tipos de Blockchains</i>	20
2.2	IDENTIDADE AUTOSSOBERANA	20
2.2.1	Registro de Dados Verificável	20
2.2.2	Identificadores Descentralizados	21
2.2.2.1	<i>Sintaxe de um DID</i>	22
2.2.2.2	<i>Métodos DID</i>	22
2.2.2.3	<i>Classificação de DIDs</i>	23
2.2.2.4	<i>DID Key</i>	24
2.2.3	Credenciais Verificáveis	24
2.2.4	<i>Framework de Governança</i>	26
2.2.5	<i>Trust over IP</i>	26
2.3	PROVAS DE CONHECIMENTO ZERO	28
2.4	ANONCREDS	28
2.4.1	Preparo para assinar uma credencial AnonCreds	28
2.4.2	Assinatura de uma credencial AnonCreds	29
2.4.3	Apresentação de uma credencial AnonCreds	32
2.5	CADEIA DE CUSTÓDIA	32
2.6	FERRAMENTAS RELACIONADAS	33
2.6.1	Hyperledger Foundation	33
2.6.2	Hyperledger Fabric	34
2.6.3	Hyperledger Identus	34
2.6.3.1	<i>Terminologia</i>	35
2.6.3.2	<i>Limitações</i>	37
3	ESTADO DA ARTE	39
3.1	BLOCKCHAIN E DOCUMENTOS DIGITAIS	39
3.1.1	<i>Blockchain family deed certificate for privacy and data security</i>	39

3.1.2	<i>Fusion of IoT, blockchain and artificial intelligence for developing smart cities</i>	39
3.1.3	<i>The use of blockchain technology and OCR in e-government for document management: Inbound invoice management as an example</i>	40
3.2	BLOCKCHAIN E CADEIA DE CUSTÓDIA	40
3.2.1	Inteligência artificial, <i>blockchain</i> e a cadeia de custódia da prova no processo penal	40
3.2.2	<i>Management of the Chain of Custody of Digital Evidence Using Blockchain and Self-Sovereign Identities: A Systematic Literature Review</i>	41
3.2.3	<i>Blockchain-Based Chain of Custody for Digital Evidence with Hyperledger Fabric</i>	42
3.3	IDENTIDADE AUTOSSOBERANA	42
3.3.1	<i>Self-Sovereign Identity: Decentralized Digital Identity and Verifiable Credentials</i>	42
3.3.2	<i>A Self-Sovereign Identity Blockchain Framework for Access Control and Transparency in Financial Institutions</i>	43
4	IMPLEMENTAÇÃO	45
4.1	DESCRIÇÃO GERAL DO SISTEMA	45
4.2	MODELO GERAL	46
4.2.1	Cadeia de Custódia	46
4.2.2	Casos de Uso	47
4.2.3	Diagrama de Sequência	48
4.3	VISÃO GERAL DA ARQUITETURA	50
4.4	GERENCIADOR DE CADEIA DE CUSTÓDIA	52
4.4.1	Chaincode	52
4.4.2	Aplicativo Typescript	54
4.4.2.1	Inicialização	55
4.4.2.2	Interação com o Chaincode	56
4.4.2.3	Criando DID e Credenciais	56
4.4.2.4	Lógica	57
4.4.2.5	Servidor Web	59
4.4.3	Script para Execução	62
4.5	CONTROLE DE ACESSO SSI	62
4.5.1	Inicializando os Agentes	62
4.5.2	Script para Execução	63
4.5.3	Componente Holder	63
4.5.4	Componente Issuer/Verifier	64
4.5.5	Erros Documentação	65
4.6	LIMITAÇÕES	65

4.7	DEMONSTRAÇÃO DE EXECUÇÃO	66
5	CONCLUSÃO	73
	REFERÊNCIAS	75
	APÊNDICE A - Dependências da Aplicação Typescript	79
	APÊNDICE B - Códigos de Funções do <code>vc-handler.ts</code>	81
	APÊNDICE C - Códigos de Funções do <code>logic.ts</code>	83
	APÊNDICE D - Docker Compose Identus	87
	APÊNDICE E - <i>Presentation Request</i> da documentação e o da imple- mentação	93
	APÊNDICE F - <i>Forensic Evidence Schema</i>	95
	APÊNDICE G - <i>Outputs</i> completos da execução	97
	APÊNDICE F - Protocolos Hyperledger Identus	105
.1	CRIANDO UMA CONEXÃO	105
.2	CRIANDO UM DID	106
.3	CRIANDO UM <i>CREDENTIAL SCHEMA</i>	107
.4	CRIANDO UMA CREDENTIAL DEFINITION	109
.5	CRIANDO UMA CREDENCIAL	109
.6	APRESENTANDO UMA PROVA	111
	APÊNDICE G - Código Fonte	115
	APÊNDICE H - Artigo Síntese do TCC	117

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

A manipulação de dados digitais cresce de forma exponencial (LOFFI et al., 2025a). Nesse contexto, surge a problemática das evidências digitais, que são frágeis, voláteis e suscetíveis à adulteração. A tecnologia *blockchain* desponta como uma possível solução diante da necessidade de garantir a integridade e a autenticidade da cadeia de custódia (ARAÚJO, 2023). Observa-se, de fato, sua crescente adoção nesse cenário de preservação de evidências digitais (LOFFI et al., 2025a) e e, como mostra a seção 3.1 deste trabalho, também no armazenamento de documentos digitais em geral.

Concomitantemente, a internet, desde sua concepção, não possui uma camada nativa de identidade digital, pois foi projetada para a comunicação entre máquinas, não entre pessoas (PREUKSCHAT; REED, 2021). Essa ausência levou ao surgimento de diferentes modelos de identidade ao longo do tempo. O modelo centralizado, baseado em contas (*account-based identity*), associa a identidade de um usuário ao cadastro em um serviço específico, com nome de usuário e senha. Tal abordagem, ainda amplamente utilizada, apresenta problemas de reusabilidade, usabilidade e segurança, como o acúmulo de grandes bases de dados vulneráveis a vazamentos (PREUKSCHAT; REED, 2021).

Em seguida, o modelo federado surgiu como tentativa de simplificar o processo de autenticação, permitindo que um provedor de identidade — como o Google (Google, 2025) — intermediasse o acesso a outros serviços. Contudo, esse modelo também introduziu novas fragilidades, como riscos à privacidade e à centralização do controle de dados pessoais (ALDOSARY; ALQAHTANI, 2021).

Nesse contexto evolutivo, surge o modelo de Identidade Autossobrerana, considerado o próximo estágio da identidade digital. Inspirado na tecnologia *blockchain*, ele adota uma arquitetura descentralizada, baseada em Identificadores Descentralizados e Credenciais Verificáveis, emitidas por entidades confiáveis. Esse paradigma devolve ao indivíduo o controle sobre seus próprios dados e permite a utilização de técnicas criptográficas, como as provas de conhecimento zero para autenticação sem exposição de informações sensíveis (PREUKSCHAT; REED, 2021).

A motivação deste trabalho é investigar como a integração da Identidade Autossobrerana pode fortalecer o modelo de cadeia de custódia baseado em *blockchain*, indo além do simples registro de documentos em um *ledger*. Busca-se também abordar o sistema que o circunda, exigindo uma arquitetura descentralizada e robusta para o controle de acesso. A Identidade Autossobrerana apresenta potencial para atender a essa demanda (BELCHIOR et al., 2020; OU; CHEN; LIN, 2025), além de contribuir para a preservação da privacidade dos atores envolvidos nesses processos.

Portanto, este trabalho justifica-se por seu potencial de contribuir para o avanço do estado da arte, explorando, por meio de estudos teóricos e práticos, a integração entre Identi-

dade Autossoberana e cadeia de custódia — uma combinação emergente e ainda subexplorada (LOFFI et al., 2025a).

1.2 OBJETIVOS

1.2.1 Objetivo Geral

O objetivo deste trabalho é estudar e integrar os conceitos de Identidade Autossoberana e cadeia de custódia forense, visando desenvolver uma prova de conceito que utilize a tecnologia *blockchain*, implementada com o uso do *Hyperledger Fabric*, e adote o *Hyperledger Identus* para controle de acesso baseado em Identidade Autossoberana. Além disso, a aplicação deve utilizar provas de conhecimento zero para validar o nível de acesso do perito no sistema, visando obter maior privacidade no processo.

Busca-se, assim, garantir a imutabilidade dos registros, a rastreabilidade das evidências digitais e o controle descentralizado de acesso, contribuindo para o fortalecimento da segurança e da confiabilidade em ambientes forenses digitais.

1.2.2 Objetivos Específicos

Os objetivos específicos são:

- Estudar os fundamentos teóricos da tecnologia *blockchain*, da Identidade Autossoberana, cadeia de custódia forense e provas de conhecimento zero;
- Analisar trabalhos relacionados que utilizam *blockchain* e Identidade Autossoberana em contextos de preservação e controle de acesso a evidências digitais;
- Projetar uma arquitetura que integre o *Hyperledger Fabric* como base para o registro imutável das evidências e o *Hyperledger Identus* como mecanismo de controle de acesso;
- Utilizar provas de conhecimento zero para validar o nível de acesso do perito ao sistema através de credenciais AnonCreds, suportadas pelo Hyperledger Identus.
- Implementar uma prova de conceito da arquitetura proposta;
- Identificar limitações no trabalho realizado e propor sugestões para estudos futuros.

1.3 ORGANIZAÇÃO DO TEXTO

O texto está organizado em:

- Fundamentação Teórica
- Estado da Arte

- Implementação
- Conclusão
- Referências
- Apêndices

No capítulo Fundamentação Teórica, os principais conceitos e ferramentas relacionadas são descritos, provendo explicações para a compreensão do trabalho.

Já o capítulo Estado da Arte analisa primeiramente três trabalhos que aplicam *blockchain* como solução para gerenciamento de algum tipo de documento digital, mostrando as vantagens dessa abordagem. Então, são analisados os trabalhos que discorrem sobre a aplicação de *blockchain* especificamente para a Cadeia de Custódia. Por último, são analisados trabalhos que discorrem especificamente sobre Identidade Autossoberana.

A seguir, o capítulo Implementação destrincha a solução desenvolvida, explicando arquitetura e cada componente dela, além de dificuldades de desenvolvimento, pontos de melhoria e mostra a execução da prova de conceito desenvolvida.

Por fim, o capítulo de *Conclusão* sintetiza os resultados obtidos, discute contribuições do trabalho e sugere possíveis extensões.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 BLOCKCHAIN

2.1.1 Definição

Uma *blockchain* pode ser definida como uma "estrutura de dados distribuída composta por uma cadeia de blocos. A *blockchain* atua como um banco de dados distribuído ou um livro-razão global que mantém registros de todas as transações em uma rede *blockchain*" (BAHGA; MADISSETTI, 2016).

As transações são armazenadas em blocos, os quais também possuem, necessariamente, *timestamp* e *hash* criptográfico. Eles armazenam seu *hash* e também o do bloco anterior a ele, dessa forma criando um encadeamento linear.

Dessa forma, a *blockchain* garante propriedades importantes, como imutabilidade, transparência e confiabilidade, já que os nós participantes da rede têm acesso aos dados nela gravados e uma ação, após registrada no *ledger* (livro-razão) não pode ser removida nem alterada (GOLOSOVA; ROMANOV, 2018).

A figura 1 ilustra a arquitetura de uma *blockchain*.

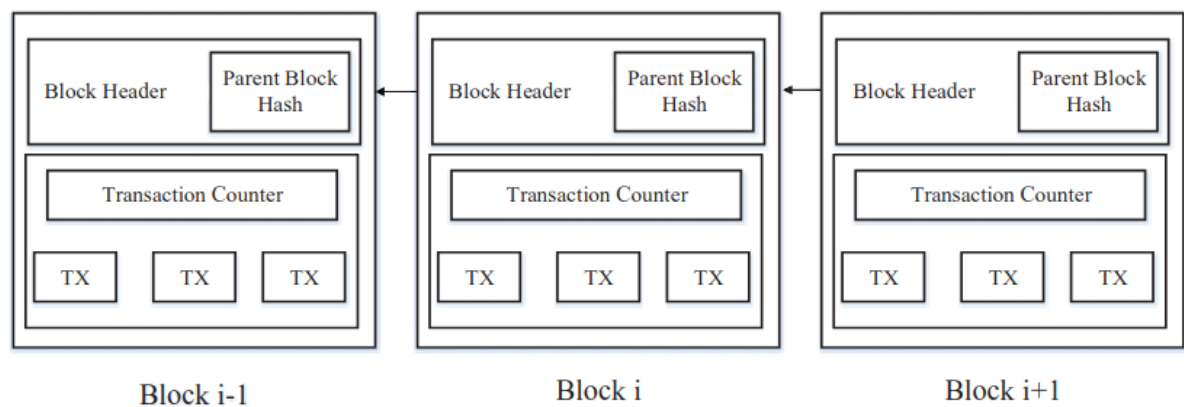


Figura 1 – Arquitetura da *blockchain*. Imagem retirada de (ZHENG et al., 2017).

2.1.2 Breve Histórico

Muitas pessoas consideram que o marco inicial da tecnologia *blockchain* foi a publicação do *whitepaper* de Nakamoto descrevendo o Bitcoin (NAKAMOTO, 2008). Embora não tenha sido o primeiro a apresentar um trabalho relacionado a *blockchain*, sua proposta inovadora de adicionar incentivos financeiros ao sistema, por meio de um mecanismo de mineração (*Proof of Work*), foi o que possibilitou o surgimento das criptomoedas (WHITAKER, 2019).

Posteriormente, foi publicado o *whitepaper* do Ethereum, no qual é introduzido o conceito de *smart contracts* (contratos inteligentes), definidos, de forma simplificada, como a exe-

cução de funções (*scripts*) diretamente na *blockchain*. Embora já existissem *scripts* no Bitcoin, estes eram bastante limitados em suas capacidades (BUTERIN et al., 2013). Com essa inovação, tornou-se possível o desenvolvimento de aplicações mais complexas — e não necessariamente financeiras — utilizando a tecnologia *blockchain*.

O Bitcoin foi concebido com o objetivo de eliminar a necessidade de confiança (*trust*) entre os participantes de uma transação financeira, substituindo-a por mecanismos criptográficos (NAKAMOTO, 2008).

Em contrapartida, plataformas como o Hyperledger Fabric eventualmente surgiram, reintroduzindo o elemento de confiança — ainda que de forma controlada — por meio de permissões e identidades conhecidas entre os participantes. Essa abordagem possibilita um modelo mais flexível, abrindo espaço para uma gama mais ampla de aplicações, especialmente em contextos corporativos e interorganizacionais (ANDROULAKI et al., 2018).

2.1.3 Tipos de Blockchains

No contexto de sistemas *blockchain*, o armazenamento de dados pode ser classificado como *On-chain* ou *Off-chain*. O primeiro consiste no registro de informações diretamente na *blockchain*, enquanto o segundo refere-se ao armazenamento delas em sistemas externos à *blockchain* (HEPP et al., 2018).

Além disso, as *blockchains* podem ser classificadas como públicas, privadas (ou permissionadas) ou ainda como híbridas (ou de consórcio), dependendo do nível de acesso, controle e participação dos usuários na rede (LOFFI et al., 2025a).

2.2 IDENTIDADE AUTOSSOBERANA

A identidade autossoberana (SSI – *Self Sovereign Identity*) apresenta-se como um novo modelo para a identidade digital, respondendo à necessidade fundamental de como indivíduos provam quem são para estabelecer confiança no ambiente online (PREUKSCHAT; REED, 2021). Ela dá ao usuário o controle sobre seus dados identitários, de forma análoga à gestão da identidade no mundo físico.

Para criar uma infraestrutura de confiança descentralizada, a SSI baseia-se em conceitos como credenciais verificáveis — representando digitalmente atributos de forma segura — e identificadores descentralizados, que são endereços únicos e persistentes sob controle do indivíduo, apoiando-se em um registro de dados verificável (PREUKSCHAT; REED, 2021).

2.2.1 Registro de Dados Verificável

Existe um consenso na comunidade SSI, de que um registro de dados verificável (VDR – *Verifiable Data Registry*) é a melhor maneira de prover as funcionalidades desejadas nesse contexto. Um VDR deve ser acessível pela internet e possui todos os dados e metadados neces-

sários para que um ecossistema de credenciais verificáveis funcione, como por exemplo chaves públicas de emissores, listas de revocações, *schemas* descrevendo as propriedades que as credenciais podem conter, entre outros (PREUKSCHAT; REED, 2021).

Na prática, o VDR pode ser implementado através de diferentes tecnologias descentralizadas, como um sistema de arquivos ou diretórios descentralizado, *ledgers* distribuídos, dentre outras tecnologias (PREUKSCHAT; REED, 2021). No contexto deste trabalho, o VDR é uma *blockchain*.

2.2.2 Identificadores Descentralizados

Os identificadores descentralizados (DIDs) foram projetados para tirar proveito de redes descentralizadas. Eles representam um novo tipo de identificador global único, voltado para identidades digitais na internet.

Na prática, um DID é um identificador uniforme de recurso (URI – *Uniform Resource Identifier*) que atua como o bloco fundamental de uma nova camada de identidade digital descentralizada e infraestrutura de chave pública (PKI – *Public Key Infrastructure*) para a internet (PREUKSCHAT; REED, 2021).

Um DID pode ser consultado — ou resolvido — por meio de um software chamado resolvidor de DID (*DID resolver*), a fim de recuperar uma estrutura de dados padronizada conhecida como documento DID (*DID document*), que contém a chave pública e outros metadados que descrevem o sujeito do DID.

A verificabilidade criptográfica baseia-se em pares de chaves pública e privada, permitindo que o controlador da chave privada comprove o controle do DID, frequentemente por meio da assinatura digital de seu respectivo documento DID. Sua natureza descentralizada possibilita que os DIDs sejam gerados e utilizados sem a necessidade de uma autoridade de registro centralizada (PREUKSCHAT; REED, 2021).

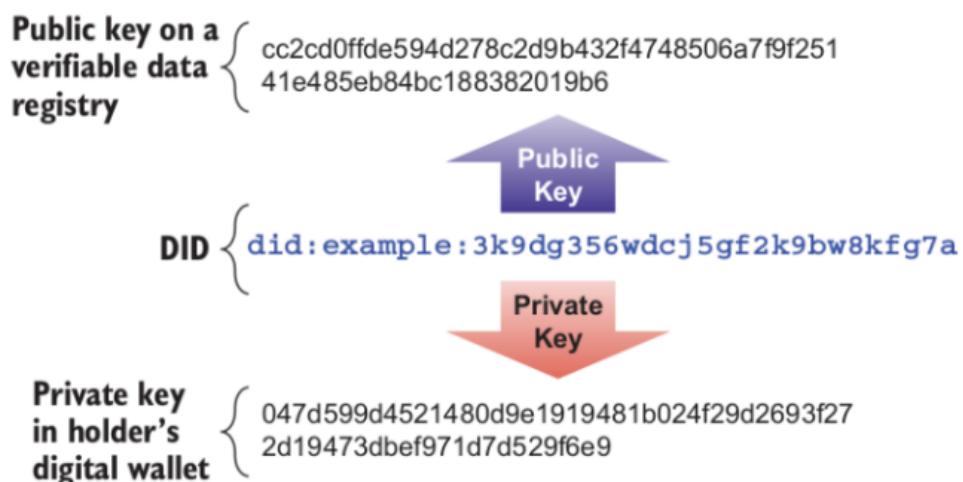


Figura 2 – Exemplo de um DID e seu par de chaves associado (PREUKSCHAT; REED, 2021).

De forma simplificada, e com base na Figura 2, pode-se dizer que o DID funciona

como se fosse um endereço, indicando onde está o documento DID no VDR (PREUKSCHAT; REED, 2021).

Já a Figura 3 mostra a relação entre as chaves públicas e privadas do controlador do DID e o DID em si.

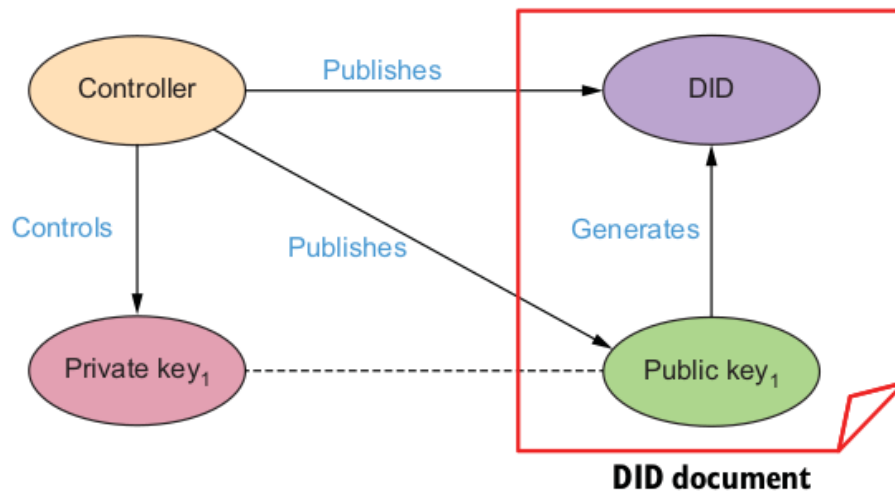


Figura 3 – Visão geral da arquitetura ao usar DIDs (PREUKSCHAT; REED, 2021)

2.2.2.1 Sintaxe de um DID

Todos os DIDs respeitam uma sintaxe específica, como é mostrado pela Figura 4.

```

did           = "did:" method-name ":" method-specific-id
method-name   = 1*method-char
method-char   = %x61-7A / DIGIT
method-specific-id = *( *idchar ":" ) 1*idchar
idchar        = ALPHA / DIGIT / "." / "-" / "_"

```

Figura 4 – Sintaxe de um DID (REED et al., 2020)

2.2.2.2 Métodos DID

Existem diferentes tipos de DIDs. Eles podem ser reconhecidos pelo método DID (*DID method*), que é a parte descrita entre o primeiro e o segundo dois-pontos de um DID, como pode ser observado pela Figura 4.

Os DIDs e seus respectivos documentos DID são criados e gerenciados de acordo com regras específicas definidas pelos *DID methods*. Esses métodos são definidos para sistemas específicos, implementando quatro operações fundamentais de qualquer DID (*Create, Read, Update, Deactivate*) (PREUKSCHAT; REED, 2021):

- 1. Criar o DID e seu respectivo Documento DID.
- 2. Utilizar o DID para recuperar seu Documento DID no sistema correspondente.
- 3. Atualizar um Documento DID de um DID em específico.
- 4. Desativar um DID, encerrando sua utilização.

Naturalmente, cada método DID possui sua própria especificação técnica discorrendo sobre esses itens.

A Figura 5 traz exemplos de DIDs gerados utilizando diferentes métodos DIDs, neste caso, respectivamente, `sov` `btc` `v1` `ethr` e `jolo`.

```
did:sov:WRfXPg8dantKVubE3HX8pw
did:btc:xz35-jzv2-qqs2-9wjt
did:v1:test:nym:3AEJTDMSxDDQpyUftjuoeZ2Bazp4Bswj1ce7FJGybCUu
did:ethr:0xE6Fe788d8ca214A080b0f6aC7F48480b2AEfa9a6
did:jolo:1fb352353ff51248c5104b407f9c04c3666627fcf5a167d693c9fc84b75964e2
```

Figura 5 – Exemplos de DIDs gerados com métodos DID diferentes (PREUKSCHAT; REED, 2021)

2.2.2.3 Classificação de DIDs

Considerando a diversidade de métodos de DIDs existentes, é possível agrupá-los em diferentes categorias (PREUKSCHAT; REED, 2021).

Uma delas compreende os métodos baseados em ledger, como o `sov` e o `ethr`, que representam a categoria original dos DIDs e se fundamentam em blockchain ou em outras tecnologias de ledger distribuído.

Há também os métodos estáticos, geralmente mais simples, porém não oferecem operações de *update* ou *deactivate* — o `key` é um exemplo desse tipo.

Por fim, destaca-se uma terceira categoria, denominada métodos alternativos, como `git`, `ipid` e `web`, caracterizados por uma arquitetura suficientemente flexível para serem aplicados em protocolos já existentes (PREUKSCHAT; REED, 2021).

A Figura 6 apresenta alguns exemplos de DIDs classificados como alternativos.

```
did:git:625557b5a9cdf399205820a2a716da897e2f9657
did:ipid:QmYA7p467t4BGgBL4NmyHtsXMoPrYH9b3kSG6dbgFYskJm
did:web:uport.me
```

Figura 6 – Exemplos de DIDs alternativos (PREUKSCHAT; REED, 2021)

2.2.2.4 DID Key

Classificado como método estático, o método DID key segue a sintaxe estabelecida pela Figura 7.

```
did-key-format := did:key:<mb-value>
mb-value      := z[a-km-zA-HJ-NP-Z1-9]+
```

Figura 7 – Sintaxe do DID key (W3C Credentials Community Group, 2025)

O mb-value – destacado em vermelho na Figura 7 – é obtido seguindo a lógica mostrada na Figura 8, ou seja, são concatenados o identificador Multikey (CABALLERO, 2023) – que identifica o tipo de chave utilizada – e os bytes brutos da chave pública utilizada para criar este DID. O resultado dessa concatenação é então codificado em Multibase (CABALLERO et al., 2021) base58-btc.

```
did-key-format := did:key:MULTIBASE(base58-btc, MULTICODEC(public-key-type, raw-public-key-bytes))
```

Figura 8 – Formato com a lógica para a formação do mb-value (W3C Credentials Community Group, 2025).

Com a chave pública presente no DID key é criado o documento DID, a partir da derivação dela. O valor `did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLGpbnnEGta2doK`, por exemplo, gera o documento DID ilustrado na Figura 9. Note que o mb-value possui o prefixo `z6Mk`. Todo DID key gerado a partir de uma chave Ed25519 terá esse mesmo prefixo (W3C Credentials Community Group, 2025), como ilustram os exemplos apresentados na Figura 10. De forma análoga, o mb-value possuirá prefixo `zQ3s` se for gerado por uma chave Secp256k1, ou prefixo `zUC7` se uma chave BLS 12-381 for utilizada, por exemplo (W3C Credentials Community Group, 2025).

2.2.3 Credenciais Verificáveis

As credenciais verificáveis (VCs – *Verifiable Credentials*) são o equivalente digital de credenciais físicas, como passaportes ou carteiras de habilitação. As VCs permitem que uma parte confiável (*Issuer*) faça declarações sobre um sujeito (geralmente o próprio detentor da credencial, chamado de *Holder*). Essas declarações, juntamente com metadados, são organizadas em uma credencial digital. Para torná-la verificável, o *Issuer* assina criptograficamente a credencial, criando uma prova (assinatura digital) que confirma sua origem e integridade (PREUKSCHAT; REED, 2021).

O *Holder* armazena suas VCs em uma carteira digital. Quando precisa comprovar alguma informação a um *Verifier*, seu agente digital — geralmente integrado à carteira — apresenta a VC ou uma prova derivada.

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1",
    "https://w3id.org/security/suites/x25519-2020/v1"
  ],
  "id": "did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK",
  "verificationMethod": [{
    "id": "did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK#z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK",
    "type": "Multikey",
    "controller": "did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK",
    "publicKeyMultibase": "z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK"
  }],
  "authentication": [
    "did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK#z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK"
  ],
  "assertionMethod": [
    "did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK#z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK"
  ],
  "capabilityDelegation": [
    "did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK#z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK"
  ],
  "capabilityInvocation": [
    "did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK#z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK"
  ],
  "keyAgreement": [{
    "id": "did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK#z6LSj72tK8brWgZja8NLRwPigth2T9QRiG1uH9oKZuKjdh9p",
    "type": "Multikey",
    "controller": "did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLgpbnnEGta2doK",
    "publicKeyMultibase": "z6LSj72tK8brWgZja8NLRwPigth2T9QRiG1uH9oKZuKjdh9p"
  }]
}
```

Figura 9 – Exemplo de documento DID gerado a partir de um DID key (W3C Credentials Community Group, 2025).

```
did:key:z6MkiTBzlymuepAQ4HEHYSF1H8quG5GLVVQR3djdX3mDooWp
did:key:z6MkjchhfUsD6mmvni8mCdXHw216Xrm9bQe2mBH1P5RDjVJG
did:key:z6MknGc3ocHs3zdPiJbnaaqDi58NGb4pk1Sp9WxWufuXSdx
```

Figura 10 – Exemplos de DID key gerados com Ed25519 (W3C Credentials Community Group, 2025).

Assim é composta a arquitetura da SSI: *Issuers*, *HOLDERS* e *Verifiers* formam o chamado Triângulo de Confiança, como ilustra a Figura 11.

O *Verifier* utiliza técnicas criptográficas — normalmente envolvendo o DID do *Issuer* e informações de um VDR — para validar diferentes aspectos da credencial apresentada. Entre os principais pontos verificados, estão:

- 1. Se a credencial apresenta os dados exigidos em um formato padronizado.
- 2. Se a assinatura digital do *Issuer* é válida, garantindo sua autenticidade.
- 3. Se a credencial permanece válida, ou seja, não foi revogada nem está expirada.

O princípio fundamental é que o *Verifier* confia na credencial pois ele confia no *Issuer*. Além disso, as VCs podem oferecer suporte a mecanismos de divulgação seletiva e a provas de conhecimento zero, permitindo que o *Holder* revele apenas o mínimo necessário de informação, sem expor os dados subjacentes (PREUKSCHAT; REED, 2021).

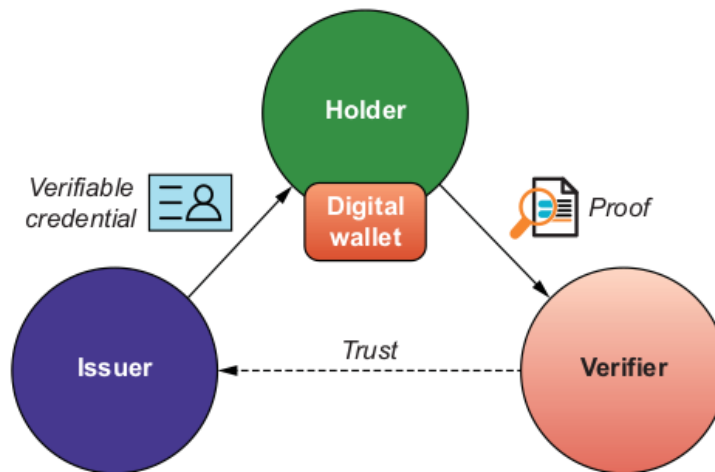


Figura 11 – Triângulo de confiança em um ecossistema SSI (PREUKSCHAT; REED, 2021).

Algumas das principais especificações de VCS incluem o W3C Verifiable Credentials Data Model (SPORNY et al., 2025), o formato SD-JWT (FETT; YASUDA; CAMPBELL, 2025), que possui divulgação seletiva de dados de forma nativa, e as credenciais com suporte a ZKPs do Hyperledger AnonCreds (The Linux Foundation, s.d.a).

2.2.4 *Framework de Governança*

Os *frameworks* de governança especificam políticas e procedimentos para habilitar a emissão de credencial de um emissor. Eles são a união de regras legais, de negócios e de tecnologia para que a infraestrutura SSI seja utilizada (PREUKSCHAT; REED, 2021).

Ao utilizar um *framework* de governança, é criado mais um triângulo de confiança na arquitetura SSI, como ilustra a Figura 12. Isso facilita o trabalho do verificador no cenário em que ele recebe uma credencial emitida por alguém que ele não conhece, pois basta ele verificar se a credencial foi assinada utilizando algum *framework* de governança que ele confie. É dessa forma que esse ecossistema de VCs pode ser escalado para funcionar em diversos níveis:

Eles são o segredo para que as VCs possam escalar para funcionar em comunidades de confiança de qualquer tamanho, desde uma única cidade até toda uma indústria [...] [Tradução livre]. (PREUKSCHAT; REED, 2021)

2.2.5 *Trust over IP*

Todos os conceitos expostos nesta seção compõem a arquitetura SSI confiável, que é compreendido como um modelo arquitetural de quatro camadas chamado *Trust over IP* (ToIP). A Figura 13 mostra em detalhes quais partes compõem cada uma dessas camadas.

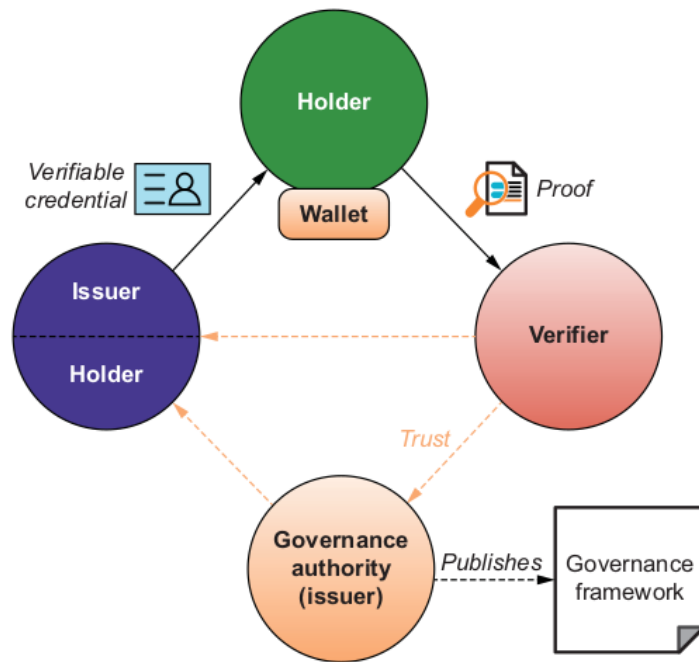


Figura 12 – Os triângulos de confiança em SSI adicionando um *framework* de governança (PREUKSCHAT; REED, 2021).

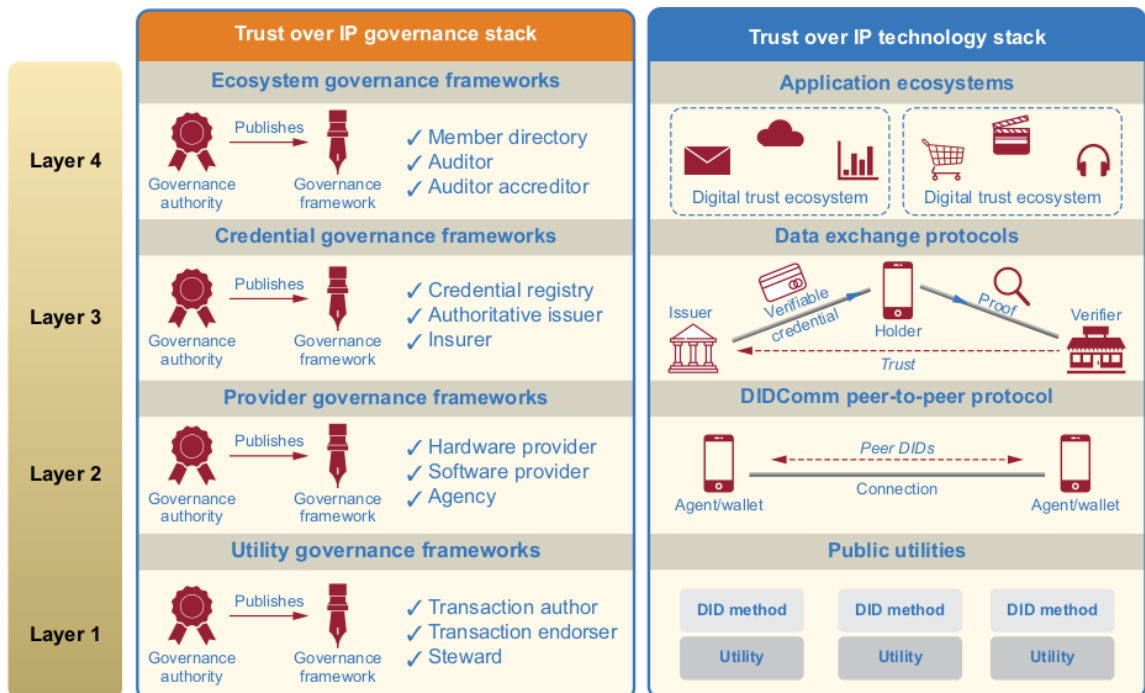


Figura 13 – As camadas da ToIP (PREUKSCHAT; REED, 2021).

2.3 PROVAS DE CONHECIMENTO ZERO

As provas de conhecimento zero (ZKPs — *Zero Knowledge Proofs*) são técnicas criptográficas que permitem a prova de algum dado ou declaração para um verificador sem revelar a informação em si para ele, isto é, ela se mantém em segredo (HASAN, 2019).

Para ser classificada como uma ZKP, o processo de prova precisa ter três características — tradução livre de (HASAN, 2019):

- **Completeness** (*Completeness*): "Se a afirmação for verdadeira, um provador honesto convencerá um verificador honesto (ou seja, um que segue o protocolo corretamente) de que ela é verdadeira";
- **Soundness** (*Soundness*): "Se a afirmação for falsa, o verificador honesto não poderá ser persuadido por nenhum provador trapaceiro de que ela é verdadeira, exceto com uma probabilidade ínfima"; e
- **Conhecimento Zero** (*Zero Knowledge*): "Se a afirmação for verdadeira, nenhum verificador aprende nada além do fato de que a declaração é verdadeira".

Uma das principais possíveis aplicações do ZKP é em sistemas de autenticação, na qual protocolos envolvendo ZKPs podem ser utilizados no lugar de autenticações mais clássicas baseadas em chave pública (HASAN, 2019).

2.4 ANONCREDS

É um tipo de credencial verificável, com suporte a ZKP. O nome AnonCreds vem de Anonymous Credentials (Credenciais Anônimas). Inicialmente sua implementação fazia parte do projeto Hyperledger Indy, porém suas funcionalidades foram realocadas para um projeto próprio — o Hyperledger AnonCreds (The Linux Foundation, s.d.a).

A Figura 14 mostra uma visão geral dos objetos do AnonCreds.

2.4.1 Preparo para assinar uma credencial AnonCreds

Uma sequência de operações de inicialização deve ser feita antes de iniciar-se o protocolo para a assinatura de uma credencial para o *Holder*.

A Figura 15 ilustra o fluxo de dados relacionados a esse preparo.

Primeiramente, é necessário que um *schema* seja publicado no VDR. Um *schema* é um modelo público que define os atributos que uma credencial terá. A partir desse momento, algum *Issuer* pode utilizar esse *schema* como base para a criação de uma *credential definition* (definição de credencial), que, por sua vez, contém dados públicos e privados que são usados para a assinatura — e posteriormente, validação — de uma credencial, como mostra a Figura 14. Um dos que a *credential definition* tem, por exemplo, é um par de chaves assimétricas

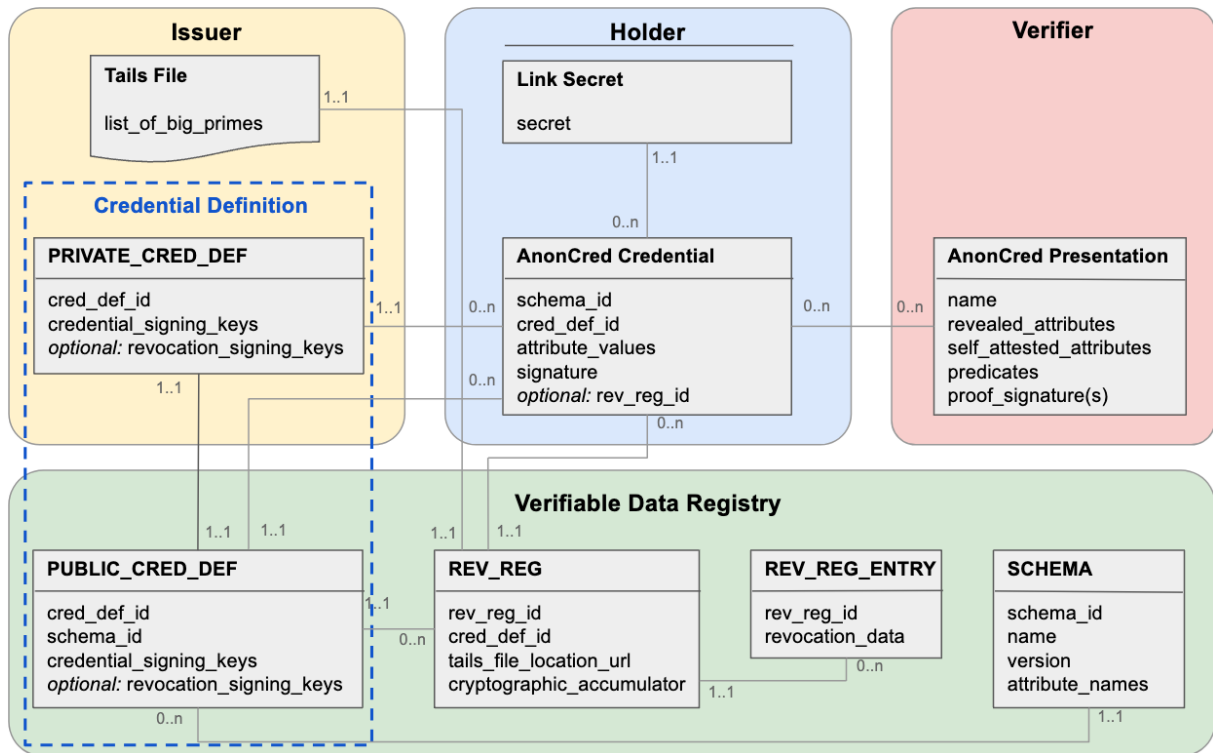


Figura 14 – Visão geral dos objetos AnonCreds (The Linux Foundation, s.d.a).

para cada atributo da credencial que, potencialmente, será assinada. Quando uma credencial é criada, as chaves privadas assinam seus respectivos atributos (The Linux Foundation, s.d.a).

Por fim, o *Holder* deve criar e armazenar um *link secret*, que é um valor secreto e único utilizado para conectar o *Holder* às credenciais assinadas recebidas por ele. É o *link secret* que habilita o uso de Provas de Conhecimento Zero posteriormente (The Linux Foundation, s.d.a).

2.4.2 Assinatura de uma credencial AnonCreds

A Figura 16 mostra o fluxo de assinatura da credencial. Primeiramente, o *Issuer* cria uma oferta de credencial (*Credential Offer*) e a envia para o possível *Holder*, que a verificará e a avaliará (The Linux Foundation, s.d.a).

Baseado no *Credential Offer*, o *Holder* cria o requerimento da credencial (*Credential Request*), que contém os campos que ele quer assinados, além de seu segredo ofuscado (*Blinded Secret*) e provas necessárias, e envia ao *Issuer* que, por sua vez, avaliará o pedido, decidindo se assinara a credencial ou não (The Linux Foundation, s.d.a).

O último passo é a criação da credencial em si e o envio dela para o *Holder*, que a armazenará.

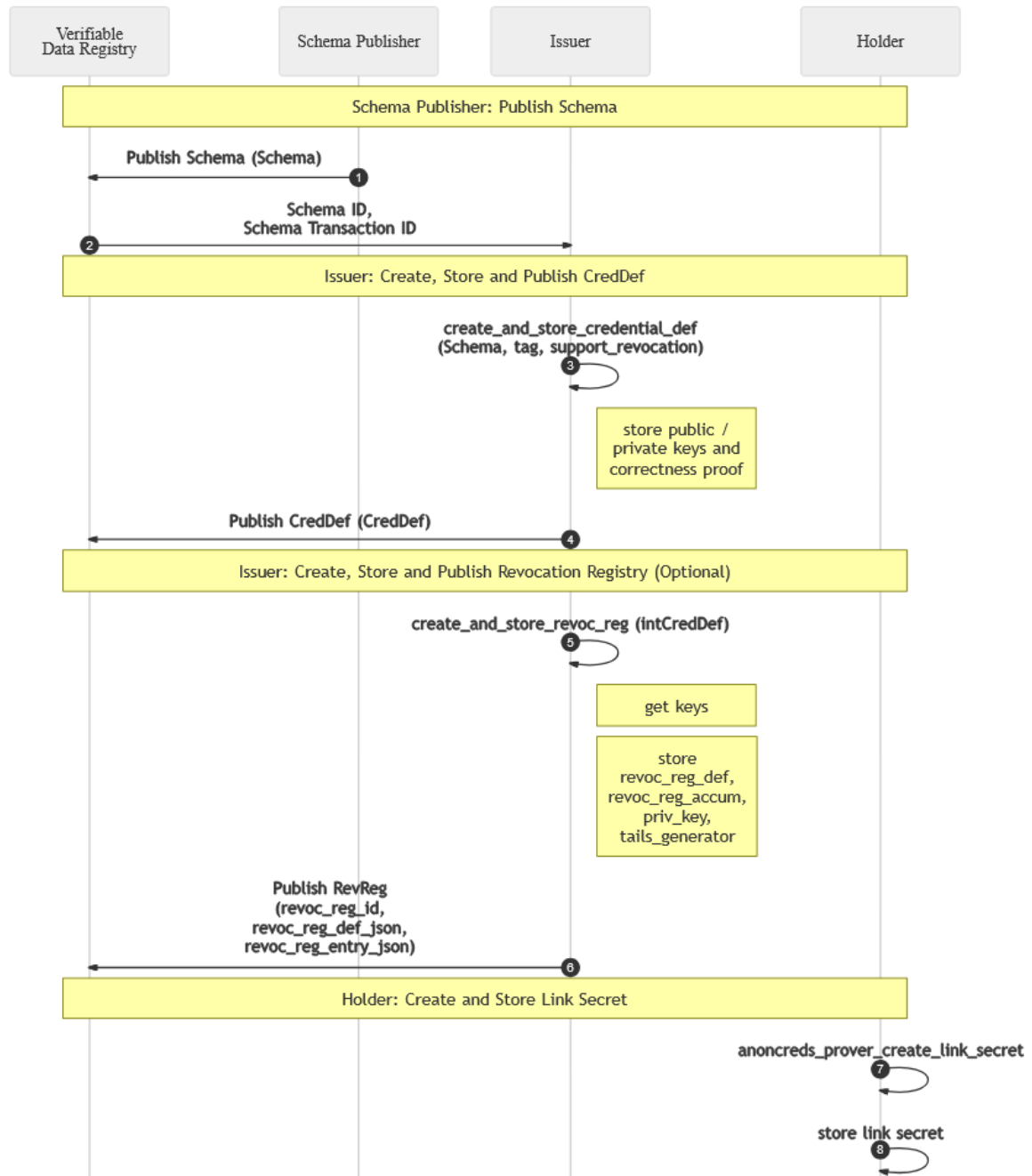


Figura 15 – Fluxo de inicialização para assinatura de credencial AnonCreds (The Linux Foundation, s.d.a).

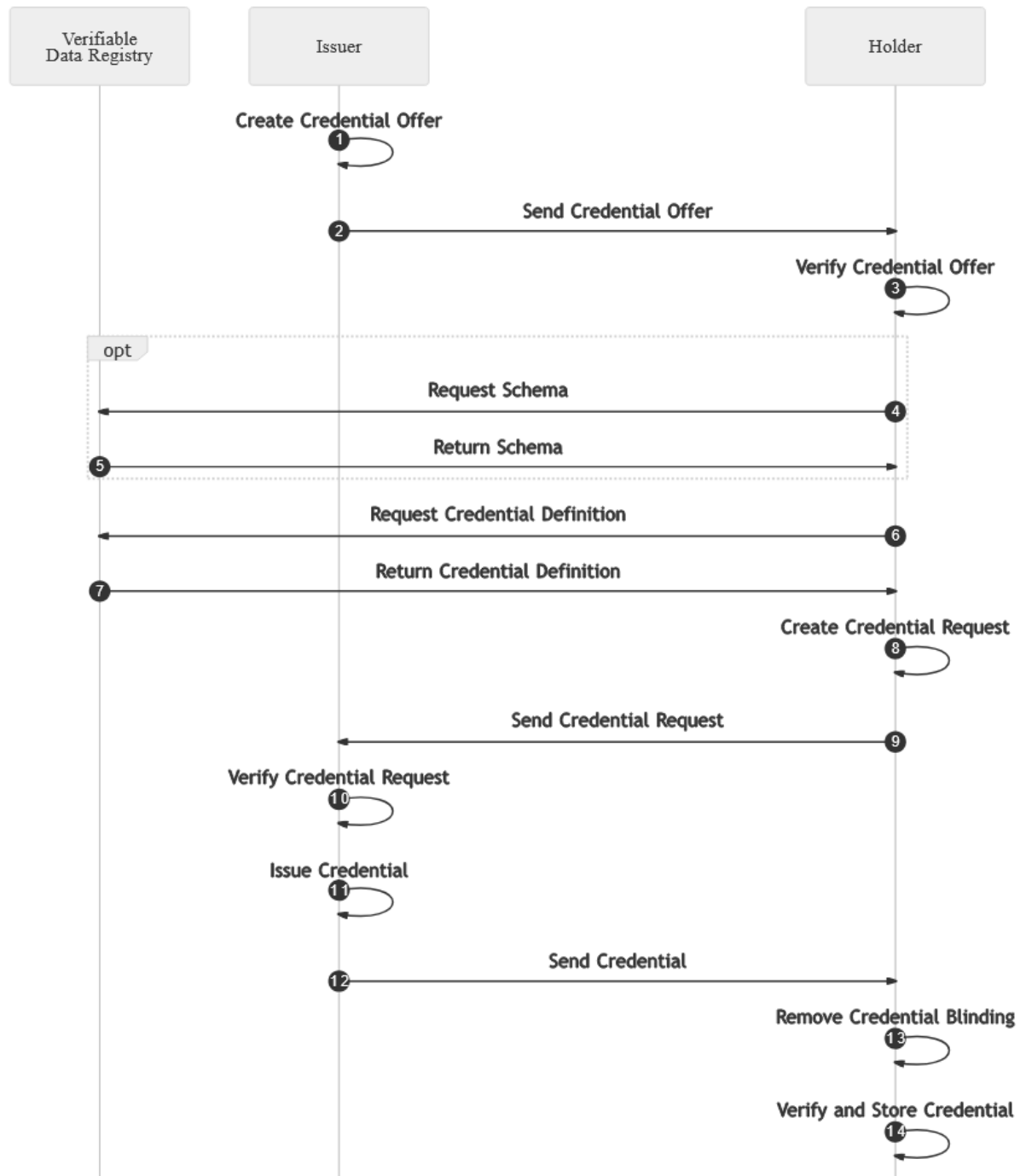


Figura 16 – Fluxo de assinatura de credencial AnonCreds (The Linux Foundation, s.d.a).

2.4.3 Apresentação de uma credencial AnonCreds

O *Verifier* cria um requerimento de apresentação (*Presentation Request*), onde são especificados quais atributos são pedidos e quais serão revelados, além de outras restrições, e envia-o para o *Holder* (The Linux Foundation, s.d.a).

A Figura 18 ilustra o formato JSON deste requerimento. O campo `requested_attributes` lista os atributos que teriam seu valor revelado, enquanto no `requested_predicates` estariam os predicados relacionados a ZKP, como por exemplo a expressão `birth_dateint <= 20020116`.

Ao receber o *Presentation Request*, o dono da credencial, se ele quiser, cria uma apresentação que satisfaça seus requisitos e então a envia de volta para o *Verifier*, que a verifica e aceita, se for válida (The Linux Foundation, s.d.a).

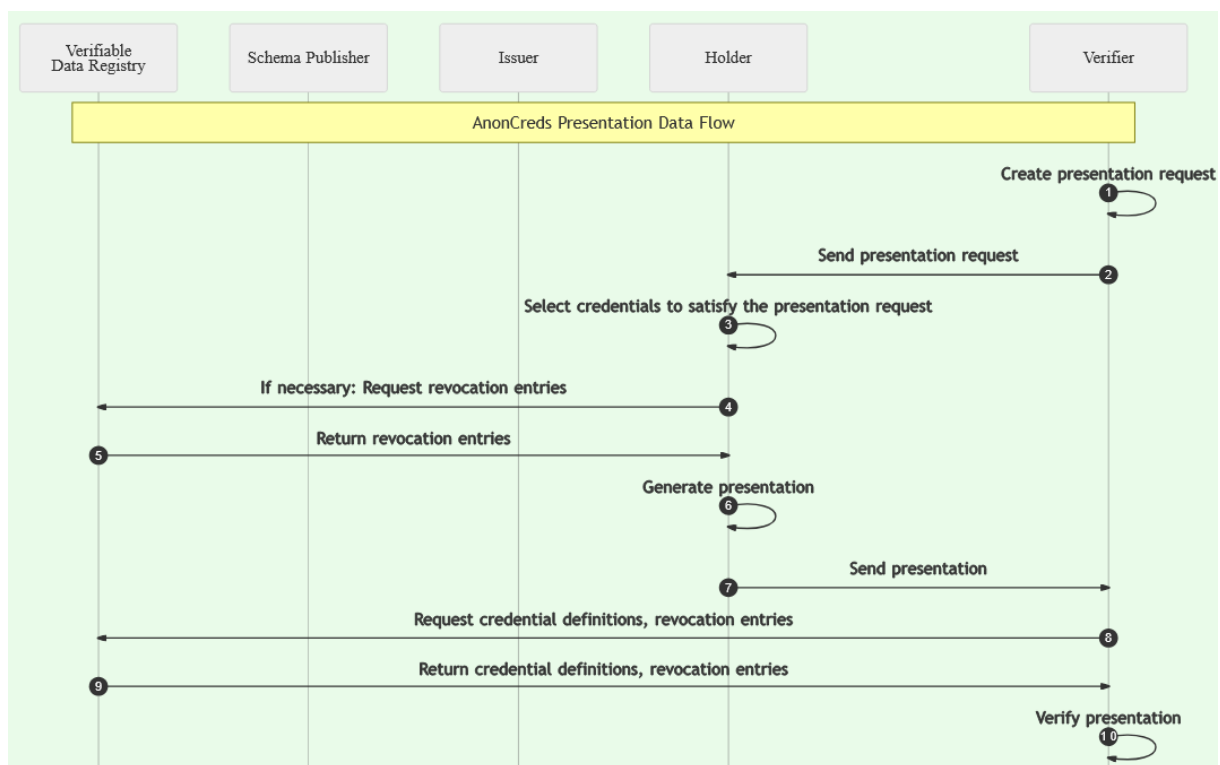


Figura 17 – Fluxo de apresentação de credencial AnonCreds (The Linux Foundation, s.d.a).

2.5 CADEIA DE CUSTÓDIA

O Art. 158-A. do Código de Processo Penal define o que é a cadeia de custódia:

Considera-se cadeia de custódia o conjunto de todos os procedimentos utilizados para manter e documentar a história cronológica do vestígio coletado em locais ou em vítimas de crimes, para rastrear sua posse e manuseio a partir de seu reconhecimento até o descarte (BRASIL, 1941; BRASIL, 2019).

```

{
  "name": string,
  "version": string,
  "nonce": string,
  "requested_attributes": {
    "<attr_referent>": <attr_info>,
    ...,
  },
  "requested_predicates": {
    "<predicate_referent>": <predicate_info>,
    ...,
  },
  "non_revoked": Optional<non_revoc_interval>,
  "ver": Optional<str>
}

```

Figura 18 – Formato de um *Presentation Request* (The Linux Foundation, s.d.a).

O objetivo primordial da cadeia de custódia é assegurar a idoneidade do vestígio, garantindo sua integridade, ou seja, que ele não sofreu perdas ou diminuições, e sua autenticidade, que corresponde ao seu caráter de legitimidade, livre de adulterações. A quebra desse elo documental e cronológico compromete o valor probatório da evidência, podendo, a depender do grau da violação, torná-la legalmente inadmissível. Para isso, cada etapa do procedimento — que compreende desde o reconhecimento do vestígio até o seu descarte — deve ser rigorosamente registrada para permitir a análise de seu histórico. Tradicionalmente, esse registro depende de processos que podem apresentar vulnerabilidades, especialmente no que tange às provas digitais, que são mais suscetíveis à manipulação e adulteração (ARAÚJO, 2023).

2.6 FERRAMENTAS RELACIONADAS

2.6.1 Hyperledger Foundation

Com código aberto (*open source*), segurança e escalabilidade sendo seus princípios, o Hyperledger Foundation funcionava como um projeto guarda-chuva da The Linux Foundation,

porém ele foi incorporado em 2024 ao LF Decentralized Trust. Essa nova configuração mantém os projetos anteriores da Hyperledger Foundation, mas também abrange mais amplamente outras iniciativas relacionadas à confiança descentralizada (LF Decentralized Trust, 2024).

Dentre seus projetos mantidos, é possível citar o Hyperledger Fabric, Hyperledger Identus, Hyperledger AnonCreds, entre vários outros (FOUNDATION, 2025).

2.6.2 Hyperledger Fabric

O Hyperledger Fabric é uma plataforma de *blockchain* permissionada e privada, a plataforma foi projetada para oferecer suporte à implementação de aplicações distribuídas, especialmente voltado a casos de uso corporativos.

Ele adota uma arquitetura de execução-ordenação-validação (*execute-order-validate*), na qual as transações são primeiramente executadas e endossadas por pares, posteriormente ordenadas por um mecanismo de consenso configurável e, por fim, validadas antes de serem registradas no ledger (ANDROULAKI et al., 2018).

Por isso, o Fabric possui três tipos de nós: Clientes, Pares e Nós do Serviço de Ordenação (OSNs). Os OSNs estabelecem a ordem total das transações na rede. Os clientes criam propostas de transações, e, após ordenadas, serão validadas e executadas pelos seus pares. (ANDROULAKI et al., 2018).

Uma característica central nesta plataforma é a de que os nós da rede devem ser conhecidos e confiáveis. Para isso, por padrão, são utilizados certificados digitais X.509, emitidos por uma Autoridade Certificadora (CA), para autenticar e autorizar os participantes da rede (HYPERLEDGER, 2025b).

O conceito de *chaincode* é análogo ao de contrato inteligente, ambos os termos sendo utilizados, por muitas vezes, de forma intercambiável na própria documentação do Hyperledger Fabric (HYPERLEDGER, 2025c). Além disso, eles podem ser escritos nas linguagens de programação Javascript, Go ou Java.

Para a escrita de aplicações clientes, é utilizado o Fabric Gateway, que é um serviço que provê uma API (*Application Programming Interface*) para que transações sejam submetidas à rede do Fabric. As APIs estão disponíveis para Go, Node (Typescript, Javascript) e Java (HYPERLEDGER, 2025a).

A figura 19 mostra um fluxo de transação exemplo, seguindo a arquitetura de execução-ordenação-validação.

2.6.3 Hyperledger Identus

O Hyperledger Identus é um projeto que fornece um conjunto de ferramentas e componentes para a construção de soluções baseadas em identidade autossobrerana. A sua arquitetura foi projetada para suportar os padrões essenciais do ecossistema SSI, incluindo DIDs e VCs (The Linux Foundation, 2025).

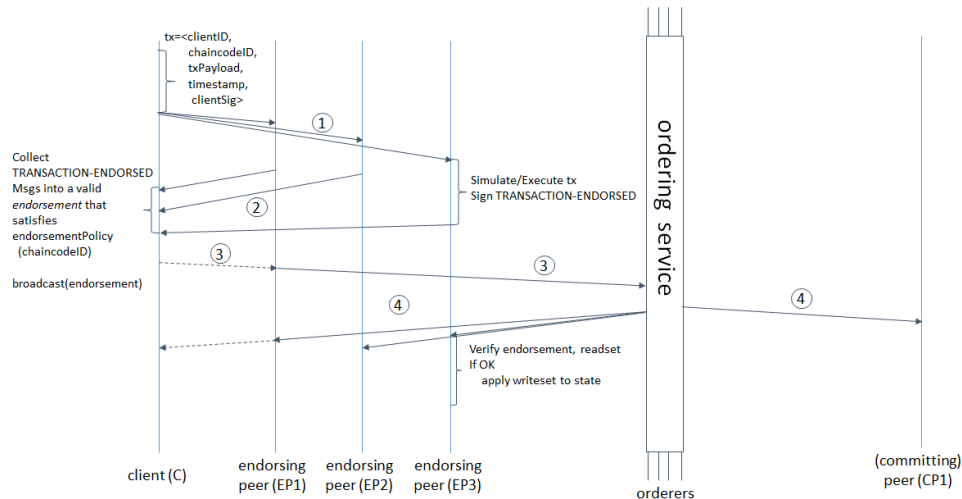


Figura 19 – Exemplo de fluxo de transação no Hyperledger Fabric (HYPERLEDGER, 2025d)

Um dos principais elementos do projeto é o *Identus Cloud Agent*, um agente que atua como uma camada de abstração para facilitar a implementação dessas aplicações descentralizadas. Ao expor suas funcionalidades por meio de uma API REST, o agente permite que os desenvolvedores interajam com a infraestrutura de identidade sem a necessidade de gerenciar diretamente a complexidade dos protocolos de comunicação e dos fluxos de criptografia. Com isso, tarefas como a emissão, o armazenamento e a verificação de credenciais são simplificadas (FOUNDATION, 2023).

O Identus utiliza a *blockchain* Cardano, através do PRISM Node. Segundo a documentação oficial, “o PRISM Node atua como um nó de segunda camada para o *ledger* distribuído. Ele funciona como um registro de dados verificáveis, fornecendo uma maneira segura e confiável de armazenar e gerenciar dados” (The Linux Foundation, s.d.b).

As figuras 20 e 21 mostram, respectivamente, a arquitetura do ecossistema do Hyperledger Identus, ou seja qual é a relação e quais componentes estão envolvidos na comunicação entre uma aplicação e o *Cloud Agent*, e a arquitetura interna do *Identus Cloud Agent*.

É importante ressaltar que, como mostra a Figura 20, cada indivíduo ou serviço terá seu próprio agente em nuvem. Nesse panorama, ao pensar no triângulo de confiança SSI, existirá um agente para o *Holder*, outro para o *Issuer* e um último para o *Verifier*.

2.6.3.1 Terminologia

Existem três termos que serão frequentemente utilizados neste trabalho. O primeiro deles é o *agent* (agente) já comentado neste capítulo. Na documentação oficial, a definição é simplesmente de que ele é “um serviço que provê APIs para interagir com a plataforma Identus e utilizar os recursos SSI”(FOUNDATION, 2024b).

O segundo termo é *wallet* (carteira), e é definido como o “componente lógico do agente que armazena segredos e fornece o isolamento lógico ou físico dos dados”(FOUNDATION,

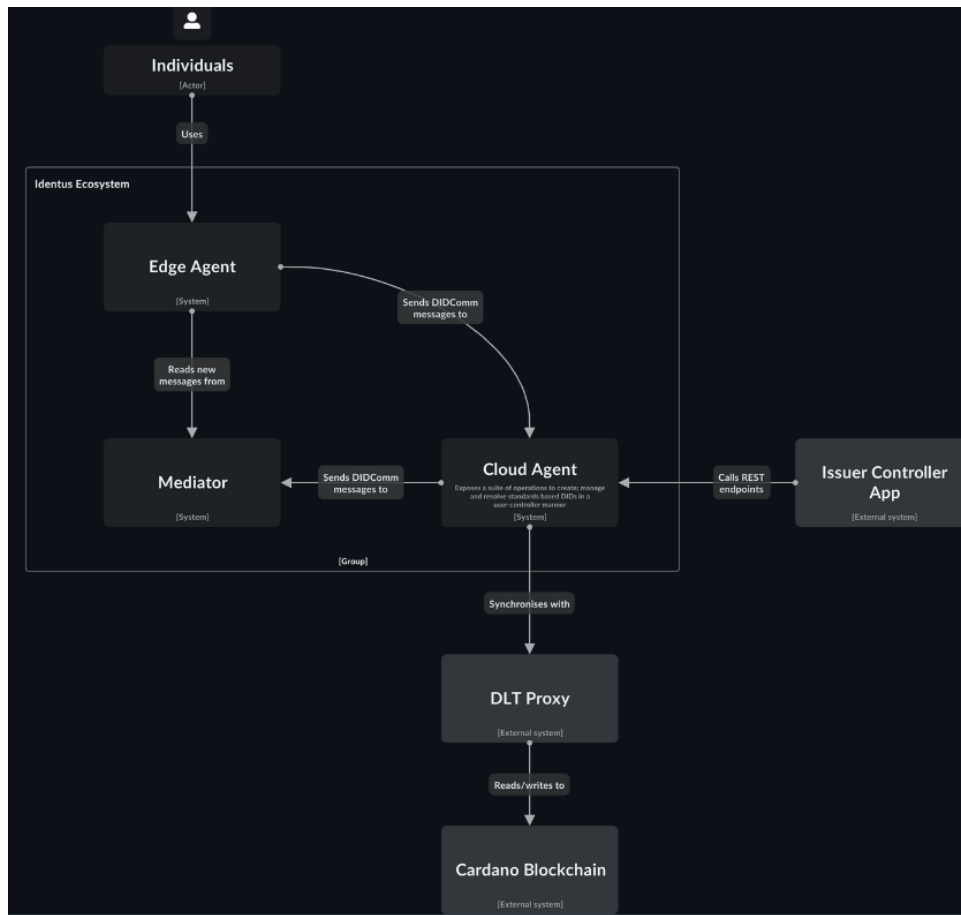


Figura 20 – Arquitetura do Ecosystema Identus (HYPERLEDGER, 2024b)

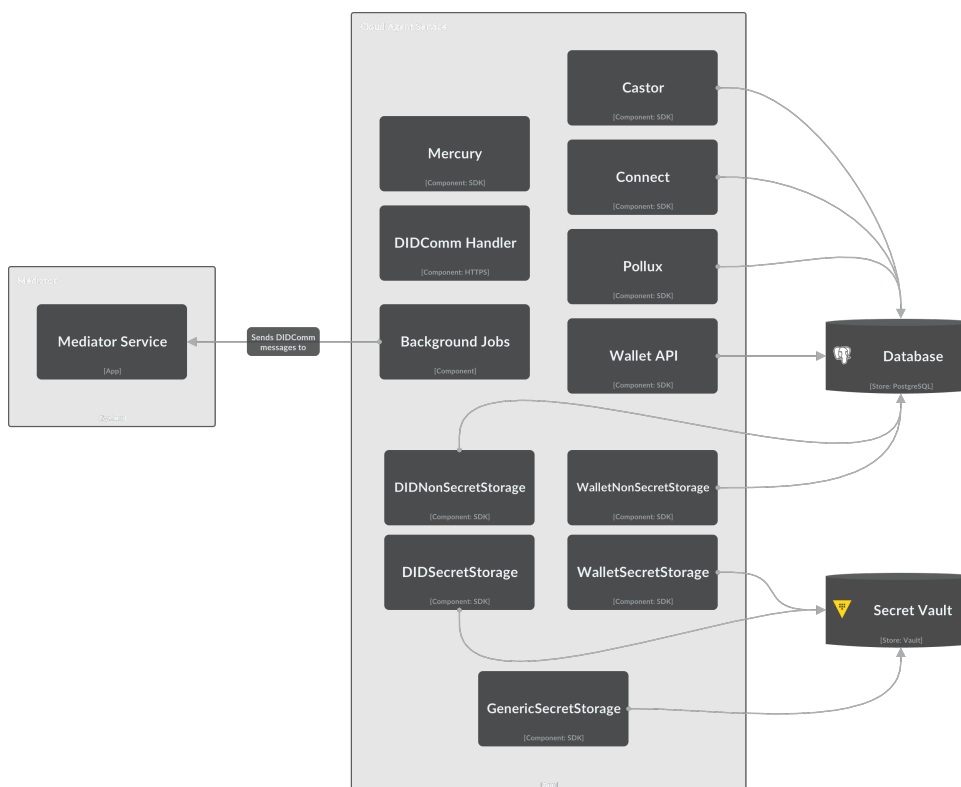


Figura 21 – Arquitetura do Identus Cloud Agent (HYPERLEDGER, 2024b)

2024a). Nela, são armazenados dados como DIDs, conexões, chaves, credenciais verificáveis e outros.

Já o último termo, é o *controller* (controlador), que é a parte da implementação de uma aplicação que irá se comunicar e controlar, via requisições, o agente.

2.6.3.2 Limitações

A documentação oficial em alguns momentos mostrou-se errada ou desatualizada, essas dificuldades serão comentadas posteriormente dentro do capítulo Implementação. O Apêndice F traz *requests* baseados na documentação oficial e ilustra como criar conexões, DIDs, credenciais Anoncreds e o processo de verificação, destacando as várias fases desses protocolos.

O Hyperledger Indentus possui algumas limitações de implementação e elas são apontadas por sua documentação (Hyperledger Indentus, 2025a):

- O *Credential Schema* não é assinado pelo *Issuer*;
- O *Credential Schema* não é publicado no VDR;
- A *Credential Definition* não é publicada no VDR; e
- Não existe *endpoint* para deletar um *Credential Schema* ou *Credential Definition*.

Essas são funcionalidades que serão implementadas em versões posteriores do projeto (Hyperledger Indentus, 2025a).

3 ESTADO DA ARTE

3.1 *BLOCKCHAIN* E DOCUMENTOS DIGITAIS

Nesta seção, são analisados três artigos que tratam da aplicação da tecnologia *blockchain* no gerenciamento de documentos digitais, com ênfase naqueles utilizados em contextos governamentais. Os trabalhos selecionados discutem as fragilidades dos sistemas tradicionais e propõem soluções digitais mais seguras, transparentes e eficientes.

3.1.1 *Blockchain family deed certificate for privacy and data security*

Na Indonésia, o Certificado de Escritura de Família (*Family Deed Certificate*) é um documento de muita importância para seus cidadãos, já que ele "é uma identidade familiar que contém dados sobre acordos, relacionamentos e o número de membros da família"(SUNARYA et al., 2020).

O artigo (SUNARYA et al., 2020) discute diversas fragilidades associadas a esse tipo de documentação, como a possibilidade de perda física do certificado, uso indevido, falsificação, erros nos dados, entre outros. Além disso, destaca-se a dificuldade de atualização das informações, o que torna o documento pouco flexível diante de mudanças familiares, como nascimento, óbito ou divórcio de um dos membros.

Portanto, o trabalho sugere o uso de *blockchain* para apaziguar esses problemas, gerenciando os documentos de forma digital e assegurando a segurança dos dados dos Certificados de Escritura de Família, devido a características como transparência, descentralização e imutabilidade. Ao mesmo tempo, essa solução oferece mais flexibilidade e reduz a burocracia envolvida nesses processos, tornando-os mais simples e eficientes (SUNARYA et al., 2020).

3.1.2 *Fusion of IoT, blockchain and artificial intelligence for developing smart cities*

Esse trabalho (KIRUTHIKA; PONNUSWAMY, 2021) discute a integração entre Internet das Coisas IoT, *blockchain* e inteligência artificial como pilares para o desenvolvimento de cidades inteligentes. Dentre os diversos tópicos abordados, destaca-se a proposta de utilização da tecnologia *blockchain* para o armazenamento de certidões civis, como registros de nascimento e óbito, com o objetivo de aumentar a segurança, a confiabilidade e a eficiência desses processos governamentais.

De acordo com os autores, os dados dessas certidões seriam mantidos em contratos inteligentes dentro de um livro-razão distribuído, protegido e privado, o que permitiria uma troca de informações mais eficiente entre diferentes instituições. Além disso, o sistema proposto contribui para o aumento da transparência, facilita o acesso a registros civis e reduz a redundância de dados. A tecnologia também seria útil na identificação de entradas duplicadas, especialmente em contextos interestaduais (KIRUTHIKA; PONNUSWAMY, 2021).

3.1.3 *The use of blockchain technology and OCR in e-government for document management: Inbound invoice management as an example*

O artigo (AZZAM et al., 2023) discute as limitações dos sistemas governamentais tradicionais, baseados em papel ou estruturas centralizadas, os quais exigem grande esforço para o gerenciamento de registros físicos, demandam espaço de armazenamento e apresentam riscos de erros, perda de documentos e possibilidade de fraudes e falsificações. Nesse contexto, destaca-se a relevância do governo eletrônico (*e-government*), que visa utilizar tecnologias da informação e comunicação para modernizar os serviços públicos e aumentar a confiabilidade dos processos administrativos.

Com base nessa perspectiva, os autores propõem o modelo SECHash, que integra tecnologias de reconhecimento óptico de caracteres (OCR) com *blockchain*, de forma a garantir a integridade e a segurança dos dados documentais. O modelo automatiza a extração das informações dos documentos e os registra em uma rede *blockchain*, impedindo sua modificação posterior. Segundo os autores, isso contribui para a redução de transações fraudulentas, elimina etapas manuais sujeitas a erros e assegura a rastreabilidade e autenticidade dos documentos armazenados (AZZAM et al., 2023).

3.2 BLOCKCHAIN E CADEIA DE CUSTÓDIA

Esta seção comentará de artigos que relacionam cadeia de custódia e *blockchain*.

3.2.1 Inteligência artificial, *blockchain* e a cadeia de custódia da prova no processo penal

Este estudo, (ARAÚJO, 2023), analisa, do âmbito do Direito, a possibilidade de utilizar inteligência artificial e *blockchain* em processos relacionados à cadeia de custódia, onde o autor demonstra a compatibilidade dessas tecnologias com as normas do processo penal brasileiro.

São analisadas as características fundamentais de uma *blockchain* e conclui-se que elas estão alinhadas com os fundamentos da cadeia de custódia, já que contribuem para a preservação da integridade e autenticidade da prova.

Segundo o autor:

Tais funcionalidades permitem um relatório mais fidedigno do caminho traçado pela prova, pois asseguram um registro imutável de todos os acontecimentos da cadeia, sem o risco de alterações que possam ocultar desvios eventualmente ocorridos no percurso, além de possibilitar que as partes, especialmente o acusado, tenham um conhecimento exato de todo o sucedido (ARAÚJO, 2023).

Além disso, o estudo destaca a maior fragilidade das evidências digitais em compa-

ração às provas físicas, uma vez que são mais suscetíveis a manipulações e adulterações em razão de sua natureza volátil. Justamente por essa vulnerabilidade, o autor direciona parte significativa da análise para demonstrar que, nesse contexto, o uso da tecnologia *blockchain* se mostra especialmente pertinente, oferecendo mecanismos eficazes para garantir a integridade e a autenticidade dos vestígios digitais.

Por fim, destaca-se mais uma funcionalidade relevante abordada pelo autor:

No que tange o controle de acesso, o Blockchain, por meio de chaves criptográficas e permissões, pode controlar o acesso aos vestígios digitais, de modo que somente as partes autorizadas poderiam ter contato com os vestígios digitais registrados na rede. Trata-se de excelente forma de evitar contatos desnecessários de pessoas não autorizadas, aumentando a confiabilidade do vestígio (ARAÚJO, 2023).

3.2.2 *Management of the Chain of Custody of Digital Evidence Using Blockchain and Self-Sovereign Identities: A Systematic Literature Review*

Este trabalho de revisão sistemática da literatura (LOFFI et al., 2025a) comenta sobre a grande importância das evidências digitais no contexto jurídico, devido ao crescimento exponencial da "criação, armazenamento e transmissão de dados digitais", e tem seu foco na união da cadeia de custódia com o conceito de SSI.

De forma geral, a conclusão do estudo é que há uma grande adoção de *blockchain* no quesito da preservação de evidências digitais, porém indica que "o SSI permanece um conceito emergente e subexplorado em aplicações forenses".

Mais especificamente, vários pontos relevantes foram levantados, como a necessidade de mais pesquisas em técnicas de preservação da privacidade, por exemplo, as Provas de Conhecimento Zero. Entretanto, foram destacados alguns pontos que se caracterizaram como lacunas nos artigos analisados pelo estudo, sendo alguns deles:

- a falta de implementações práticas;
- o armazenamento seguro de evidências *off-chain*; e
- a necessidade de incluir novos campos nos metadados das evidências.

Além disso, o trabalho ressalta que, em geral, plataformas *blockchain* permissionadas facilitam a implementação de mecanismos de confidencialidade e controle de acesso. Já as *blockchains* privadas apresentam vantagens em relação às públicas e às baseadas em consórcio, ao eliminarem taxas de transação e reduzirem a necessidade de medidas criptográficas adicionais para garantir a segurança, especialmente no que se refere à privacidade e aos requisitos de controle de acesso relacionados às evidências (LOFFI et al., 2025a).

3.2.3 *Blockchain-Based Chain of Custody for Digital Evidence with Hyperledger Fabric*

O artigo (LOFFI et al., 2025b) fundamenta sua proposta na necessidade de adequar os processos tradicionais de cadeia de custódia às exigências da evidência digital no contexto judicial. Os autores destacam que o registro de uma cadeia de custódia deve responder a um conjunto de questões conhecido como 5WH (*Who, What, When, Where, Why e How* — Quem, O quê, Quando, Onde, Por quê e Como), a fim de documentar a história cronológica da evidência e garantir sua rastreabilidade.

Além disso, os metadados armazenados devem assegurar propriedades de segurança cruciais como integridade, autenticidade, rastreabilidade e verificabilidade. Nesse contexto, a tecnologia *blockchain* é apresentada como uma solução eficaz, pois seu uso de funções hash e criptografia, aliado à sua natureza descentralizada, permite encadear informações e garantir a consistência dos dados.

A solução foi implementada de forma prática, utilizando a plataforma Hyperledger Fabric, com um contrato inteligente definindo as operações *CreateAsset*, *UpdateAsset* e *TransferAsset*, permitindo o registro, atualização e transferência de custódia das evidências digitais.

Dessa forma, o estudo conclui que essa implementação de cadeia de custódia possui vários benefícios, principalmente devido às características inerentes à *blockchain*. Entre eles, estão a precisão cronológica dos dados, a redução do risco de adulteração das provas, a melhoria da segurança dos dados e da integridade, além de maior transparência, pois as auditorias são facilitadas, já que todas as movimentações das evidências estão registradas no livro-razão (LOFFI et al., 2025b).

3.3 IDENTIDADE AUTOSSOBERANA

Esta seção comentará alguns trabalhos que demonstram o porquê da Identidade Autossoberana ser um tema relevante no estado da arte.

3.3.1 *Self-Sovereign Identity: Decentralized Digital Identity and Verifiable Credentials*

Este livro (PREUKSCHAT; REED, 2021) aborda diferentes aspectos da Identidade Autossoberana, fornecendo uma visão abrangente sobre sua origem, princípios e aplicações práticas. Os autores exploram a evolução dos modelos de identidade digital — do modelo centralizado, no qual cada serviço gerencia suas próprias credenciais de acesso — geralmente compostos por usuário e senha —, passando pelo modelo federado até chegar ao modelo autossoberano, em que o indivíduo detém total controle sobre seus próprios dados de identificação. Todos esses modelos buscam sanar a lacuna deixada pelo fato da internet ter sido criada sem uma camada de identidade. Essa transição é apresentada como uma resposta às limitações dos sistemas centralizados e federados, como a dependência de provedores de identidade e a vulnerabilidade à exposição indevida de dados pessoais.

Dessa forma, segundo os autores, a SSI representa o estado da arte da gestão de identidades digitais, por combinar princípios de privacidade, interoperabilidade e descentralização, alinhando-se às exigências contemporâneas de segurança e conformidade regulatória. Essa análise é relevante para o presente trabalho, pois reforça o papel da SSI como um componente essencial em arquiteturas que buscam garantir autenticidade e autorização descentralizada — aspectos fundamentais em uma cadeia de custódia digital baseada em *blockchain*.

3.3.2 A Self-Sovereign Identity Blockchain Framework for Access Control and Transparency in Financial Institutions

O artigo (OU; CHEN; LIN, 2025) propõe o *Framework* de Identidade Autossoberana e Controle de Acesso do Cliente (CSSI – *Customer Self-Sovereign Identity and Access-Control Framework*), que aplica a tecnologia de Identidade Autossoberana especificamente para atender às necessidades de avaliação de crédito em instituições financeiras. O trabalho surge em um contexto onde a SSI se tornou um tópico em ascensão, impulsionado pelo desenvolvimento da tecnologia *blockchain* e pela crescente conscientização sobre a privacidade pessoal.

Para otimizar a avaliação de crédito e a concessão de empréstimos, o *framework* permite que os clientes armazenem ativos avaliáveis e dados de crédito de forma segura na *blockchain*. Então, o CSSI confere aos clientes direitos aprimorados de controle de acesso, permitindo-lhes definir a duração do acesso e o conteúdo exato que será compartilhado com as instituições financeiras. Uma vez que o período de acesso expira, as instituições ficam impedidas de acessar os dados, prevenindo o uso não autorizado de dados a longo prazo.

Este trabalho, em especial, mostra que a SSI é uma poderosa ferramenta para controle de acesso.

4 IMPLEMENTAÇÃO

O objetivo primário deste trabalho sempre foi explorar e relacionar a área de Identidade Autossobrerana com a cadeia de custódia. A princípio, foi avaliado se isso seria possível utilizando somente o *Hyperledger Identus*. Porém, no Identus, todas as credenciais são armazenadas diretamente nas *wallets* dos agentes. A solução imaginada da cadeia de custódia ser um encadeamento de credenciais precisaria importar e exportar credenciais dessas *wallets*. Após o estudo da API do agente em nuvem (IDENTUS, 2025), concluiu-se a inviabilidade dessa ideia, já que essas funcionalidades não estão presentes no Identus.

Portanto, adotou-se o projeto *Hyperledger Fabric* para desenvolver essa aplicação, visto sua modularidade e extensibilidade (ANDROULAKI et al., 2018), o que permitiu que a solução pensada fosse implementada de fato.

Posteriormente, pensou-se em como seria feito o controle de acesso a essa implementação, e optou-se por baseá-lo em SSI. Nesse cenário, o uso do Identus é muito favorável, pois ele já implementa as operações criptográficas necessárias para criar esse controle de acesso, sendo necessário apenas desenvolver controladores para os agentes em nuvem, além de possuir suporte a *Anoncreds*, o que possibilita o uso de provas de conhecimento zero.

Em outras palavras, o uso do *Hyperledger Fabric* e do *Hyperledger Identus* advieram de diferentes necessidades, e sua união surge a partir da oportunidade de dar a este trabalho um caráter mais exploratório, unindo e experimentando novos projetos no contexto da cadeia de custódia (como *Anoncreds*, ZKP e o Identus).

4.1 DESCRIÇÃO GERAL DO SISTEMA

O sistema desenvolvido conta com dois papéis: o perito e o emissor. O perito deseja documentar algo na cadeia de custódia, mas para isso ele primeiro deve obter acesso a ela. Esse acesso é concedido pelo emissor, que assina uma credencial detalhando o nível de permissões do perito ao utilizar o sistema e envia a ele através de uma conexão estabelecida entre os dois. Essa sequência de passos se baseia no Hyperledger Identus.

Após obter a credencial, o perito pode se comunicar com o gerenciador de cadeia de custódia e tentar realizar diversas ações, como registrar ou atualizar uma nova evidência. Sempre que ele fizer a requisição para executar alguma ação, a sua credencial é verificada e suas permissões, validadas. No caso de sucesso, a execução é feita, caso contrário, algum erro ocorrerá.

Dentro desse gerenciador, existem outros tipos de credenciais. Elas são criadas fora da *blockchain*, encadeadas e representam uma cadeia de custódia em si. Essa parte da implementação é baseada no Hyperledger Fabric, pois ele armazena no *ledger* metadados importantes para a garantia da autenticidade e integridade da cadeia de custódia.

A Figura 22 mostra em alto nível o fluxo de dados no sistema quando o perito, após obter sua credencial de acesso, faz uma requisição para executar alguma operação no sistema.

As próximas seções detalham a modelagem, casos de uso e arquitetura do sistema construído.

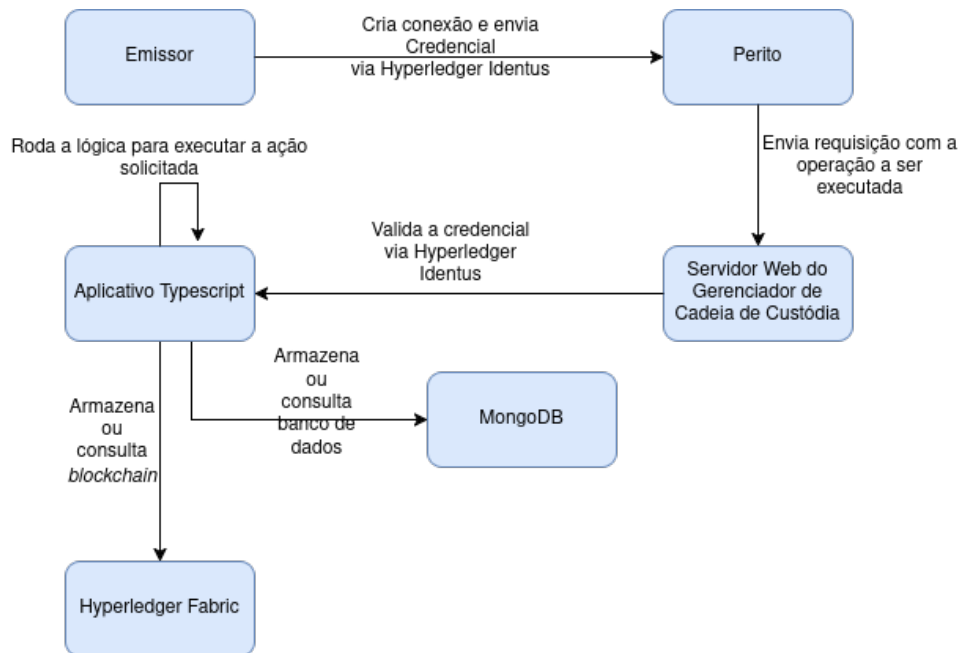


Figura 22 – Fluxo de dados para executar uma ação no sistema desenvolvido

4.2 MODELO GERAL

4.2.1 Cadeia de Custódia

A cadeia de custódia foi modelada como um encadeamento de credenciais, como evidência a Figura 23. Cada credencial nesse encadeamento é uma documentação sobre algum processo ou análise feito sobre a evidência digital em questão. Note que a “Credencial N” foi a última a ser adicionada, isto é, ela é a ponta da cadeia. Toda credencial tem uma referência ao identificador da credencial que a precede neste encadeamento, além das informações necessárias sobre a evidência (5WH).

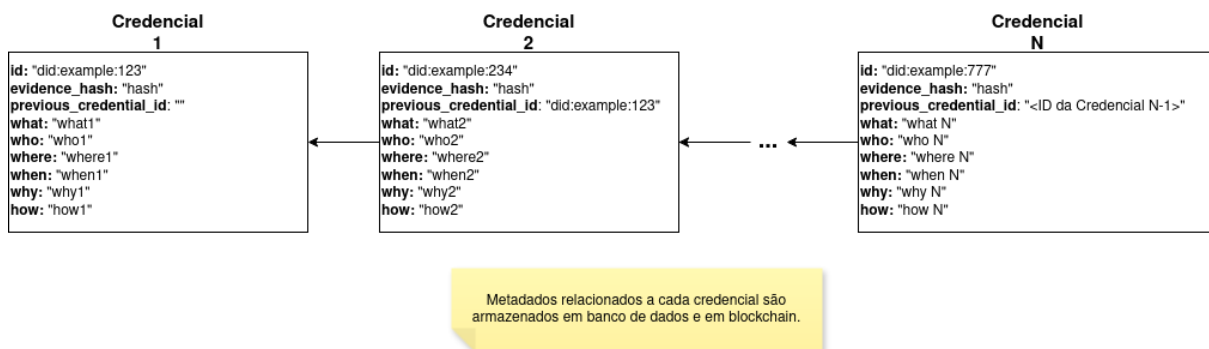
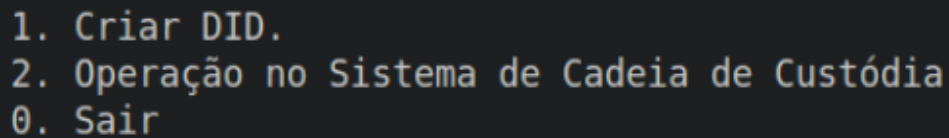


Figura 23 – A cadeia de custódia como encadeamento de credenciais

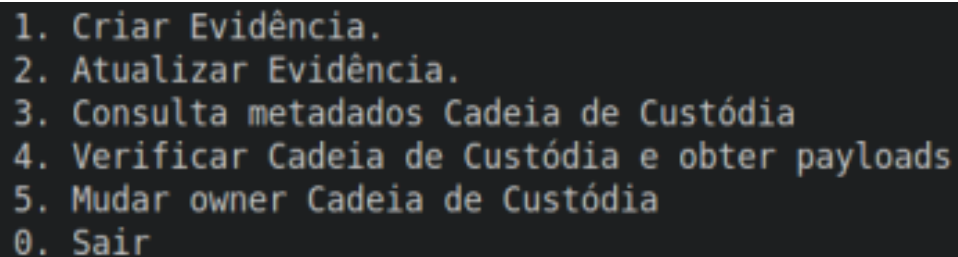
4.2.2 Casos de Uso

No sistema desenvolvido, existem dois atores: o Perito (*Holder*) e o Emissor (*Issuer*). As possíveis ações que o Perito pode fazer nesse sistema são descritas pelas Figuras 24 e 25. Já as escolhas do Emissor são mostradas pela Figura 26



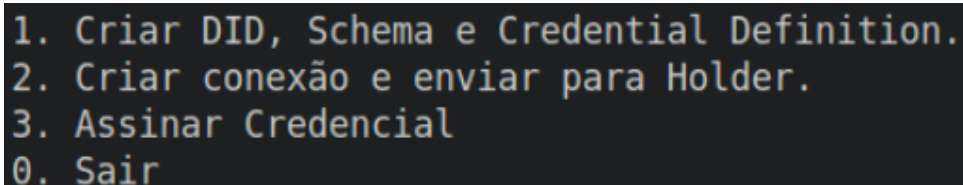
```
1. Criar DID.  
2. Operação no Sistema de Cadeia de Custódia  
0. Sair
```

Figura 24 – Ações que o Perito pode escolher



```
1. Criar Evidência.  
2. Atualizar Evidência.  
3. Consulta metadados Cadeia de Custódia  
4. Verificar Cadeia de Custódia e obter payloads  
5. Mudar owner Cadeia de Custódia  
0. Sair
```

Figura 25 – Ações de gerência de cadeia de custódia que o Perito pode escolher



```
1. Criar DID, Schema e Credential Definition.  
2. Criar conexão e enviar para Holder.  
3. Assinar Credencial  
0. Sair
```

Figura 26 – Ações que o Emissor pode escolher

A Figura 27 é um diagrama de caso de uso, evidenciando de forma gráfica quais são os casos de uso para o Perito e para o Emissor na aplicação desenvolvida.

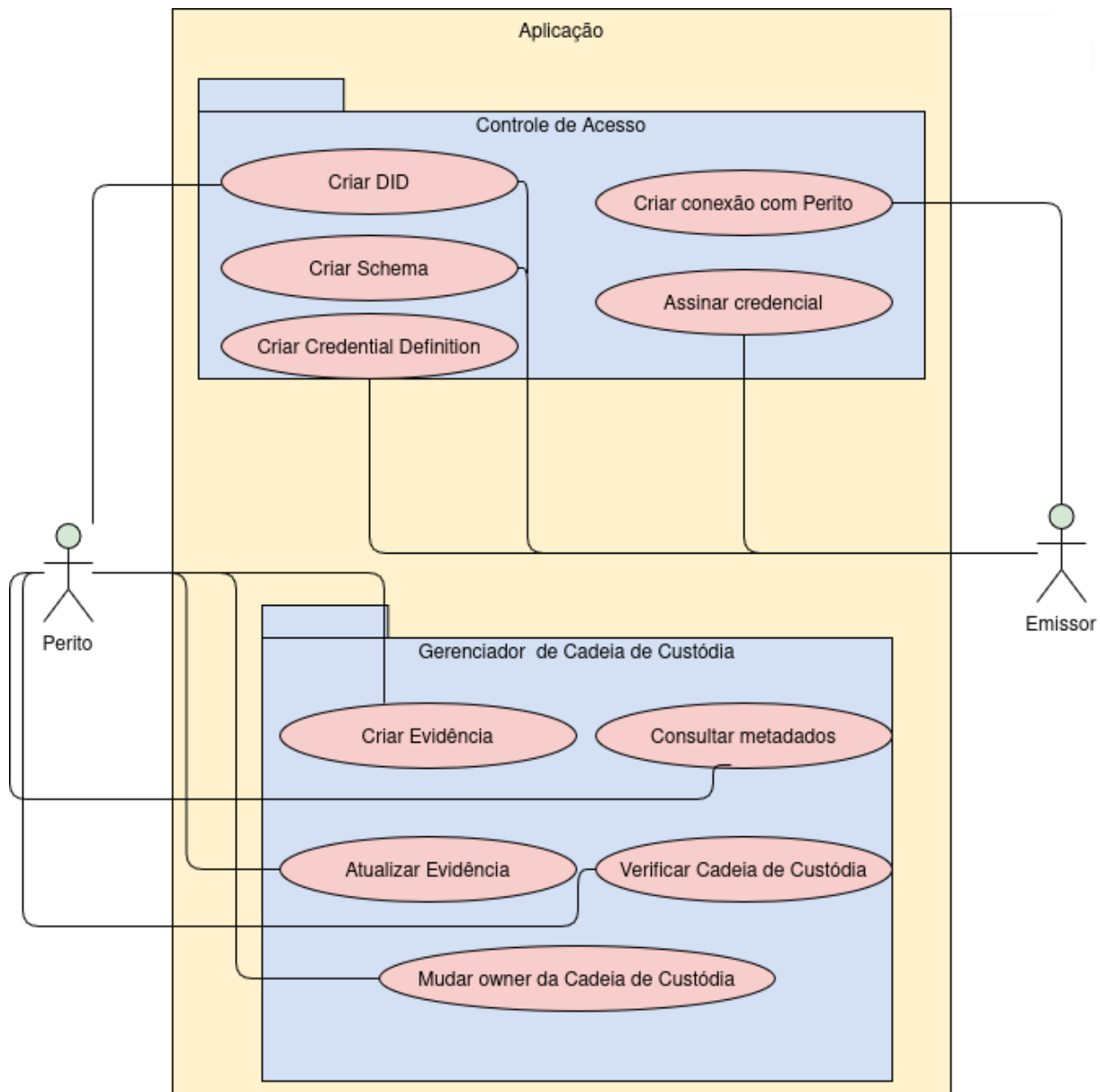
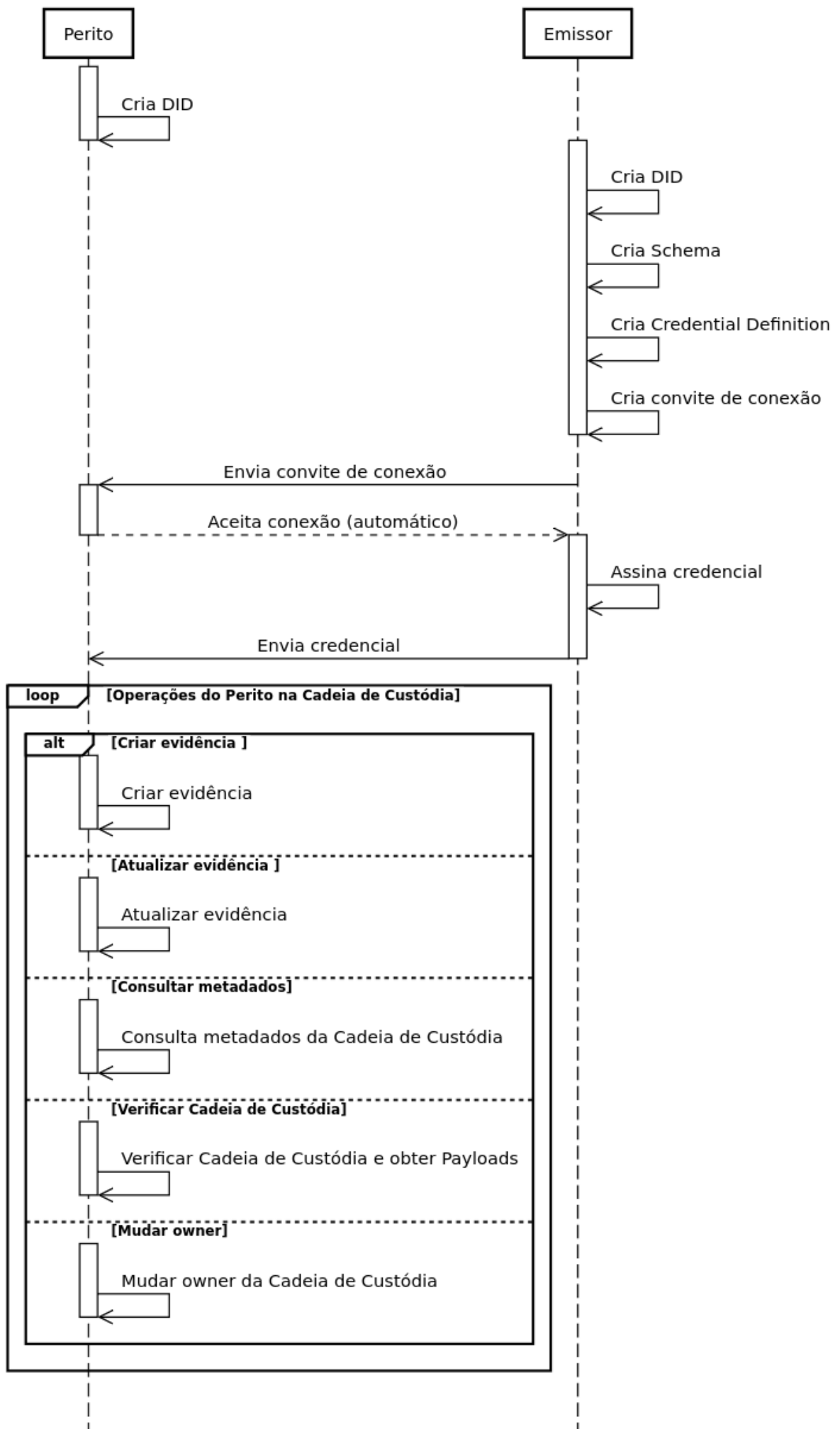


Figura 27 – Casos de Uso da Aplicação

4.2.3 Diagrama de Sequência

Nem toda ordem de execução das ações descritas pelos casos de uso resulta em sucesso. A Figura 28 é um diagrama de sequência que explicita a ordem correta das execuções para que o fluxo aconteça sem erros. Nota-se que as operações do perito na cadeia de custódia só terão sucesso após o recebimento da credencial.



4.3 VISÃO GERAL DA ARQUITETURA

A implementação feita pode ser dividida em duas grandes partes. A primeira é um sistema de gerenciamento de cadeia de custódia, criado utilizando o Hyperledger Fabric. A segunda parte é um sistema SSI de controle de acesso construído com o Hyperledger Identities que permite peritos obterem uma credencial verificável que funciona como permissão para que eles manipulem a cadeia de custódia.

O processo de desenvolvimento seguiu uma abordagem incremental. Após o estudo teórico dos conceitos envolvidos, a documentação das bibliotecas que seriam utilizadas foi analisada. Em seguida, foram criadas algumas demonstrações (demos) para testar e entender seu funcionamento, o que serviu de base para a implementação posterior. Após possuir demos funcionais, a arquitetura final foi projetada e a implementação realizada.

A Figura 29 mostra a visão geral da arquitetura da aplicação de forma simplificada. A comunicação entre o Controle de Acesso SSI e o Gerenciador de Cadeia de Custódia se dá via REST API. O primeiro usa, em teoria, a *blockchain* Cardano, através do PRISM Node. Já o segundo se comunica com o *ledger* do Hyperledger Fabric através do Fabric Gateway.

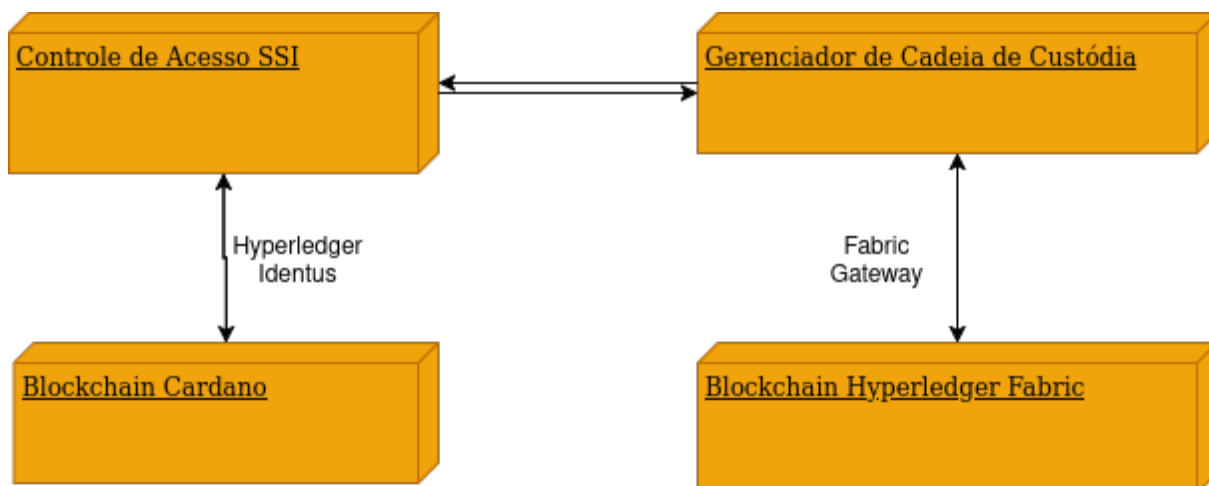


Figura 29 – Visão Geral da Arquitetura

Já a Figura 30 ilustra a arquitetura de forma mais detalhada. Dentro do controle de acesso, há componentes criados em Python, através deles o perito — ou quem for emitir a credencial para ele — se comunica, através de uma interface via terminal, com o controlador *controller* de seu agente. O *controller* então comunica-se com o agente via REST API, e então as operações desejadas — criar um DID, por exemplo — são executadas.

O Gerenciador de Cadeia de Custódia possui um aplicativo escrito em Typescript que recebe *requests* e então faz operações específicas, como adicionar uma nova evidência, ou seja, iniciando uma cadeia de custódia. A cadeia de custódia é representada por um encadeamento de credenciais verificáveis JWT, criadas dentro desse aplicativo Typescript, sendo que parte dos metadados e a credencial em si são armazenados *off-chain* em um banco de dados, o MongoDB.

Outros metadados, como por exemplo o hash da credencial, são armazenados diretamente na *blockchain*, através de contratos inteligentes escritos na linguagem Go. Ressalta-se a distinção entre os dois tipos de credenciais utilizadas nesse sistema: as JWT, encadeadas e usadas para armazenar os dados da cadeia de custódia, e as AnonCreds, emitidas para peritos e utilizadas somente para controle de acesso.

Nota-se que enquanto a Figura 29 mostrou a arquitetura pensada em alto nível e em sua forma ideal, a Figura 30 destrincha os detalhes de implementação. Nela, é perceptível a ausência da *blockchain* Cardano. Isso se dá ao fato da aplicação construída utilizar uma construção do PRISM Node que utiliza somente um banco de dados para simular essa interação com a *blockchain* de fato, como evidenciado pelo arquivo .yaml presente no Apêndice D.

Toda a implementação foi desenvolvida com base em um modelo assíncrono, fazendo uso de funções *async* e *await* em praticamente todas as definições e requisições, a fim de permitir a execução simultânea de tarefas e melhorar o desempenho do sistema.

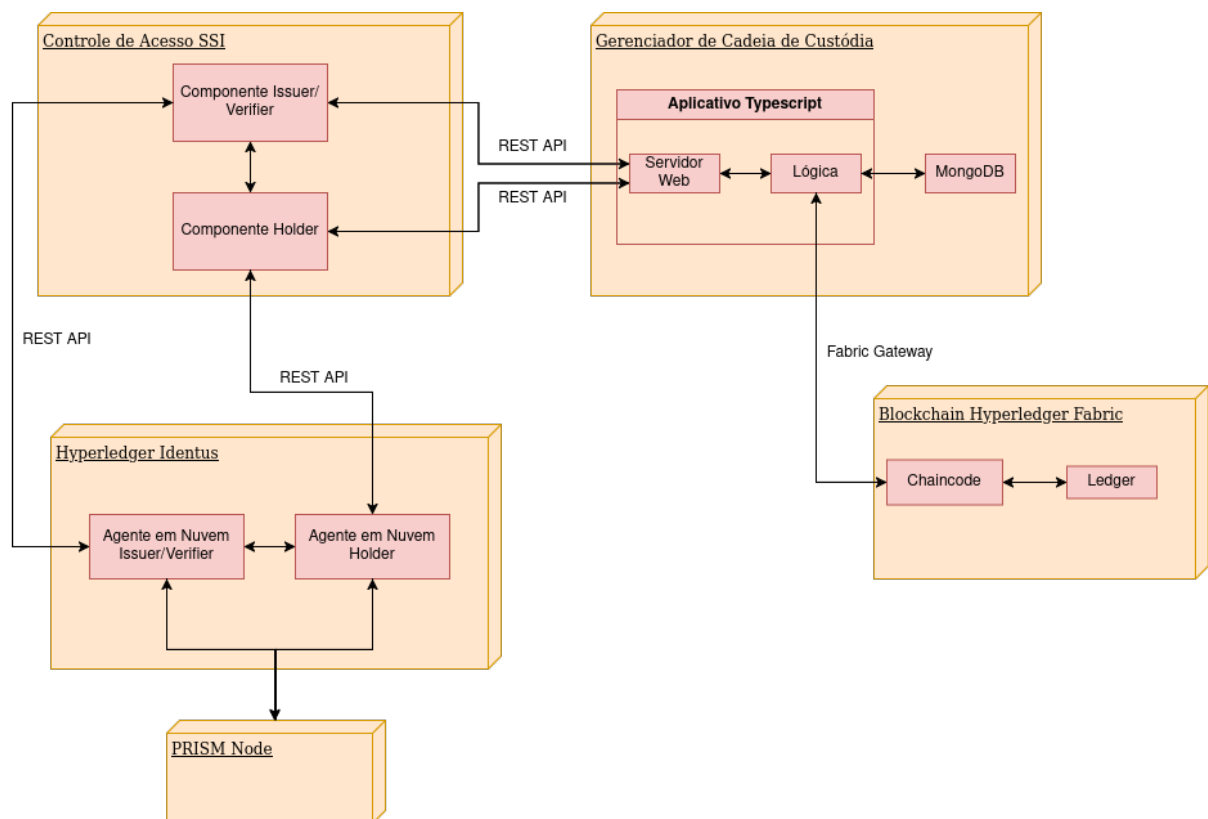


Figura 30 – Arquitetura da implementação mais detalhada

As próximas seções discutirão detalhadamente cada um dos componentes ilustrados pela Figura 30.

4.4 GERENCIADOR DE CADEIA DE CUSTÓDIA

É importante notar que a versão do Hyperledger Fabric utilizada nesta prova de conceito é a 2.5.12. Esta versão foi obtida no início do desenvolvimento, através do script oficial `install-fabric.sh` (The Hyperledger Foundation, 2025). Além disso, o início do desenvolvimento foi baseado no exemplo *asset-transfer-basic*, disponível no repositório `fabric-samples` no Github (HYPERLEDGER, 2024a).

4.4.1 Chaincode

Uma parte fundamental do funcionamento do *Chaincode* no Fabric é o pacote `shim`: “O pacote `shim` provê APIs para o *chaincode* acessar suas variáveis de estado, contexto de transações e chamar outros *chaincodes*” (Hyperledger Fabric, 2024).

Ele possui o `ChaincodeStub` e, segundo um comentário no código fonte: “O `ChaincodeStub` é um objeto passado ao *chaincode* para que a biblioteca `shim` gerencie as APIs” (HYPERLEDGER, 2024c).

Por exemplo, dentro do pacote `shim`, existe a declaração do tipo `ChaincodeStubInterface`, onde é definida a função `PutState` e a `GetState`, de assinaturas, respectivamente, `PutState(key string, value []byte) error` e `GetState(key string) ([]byte, error)` (Hyperledger Fabric, 2024).

O `GetState` consulta o *ledger* e retorna o valor da *key* especificada, enquanto o `PutState`, adiciona a chave e valor especificados na transação como uma proposta para que eles sejam escritos no *ledger*, o que ocorrerá somente após a validação da transação e seu `commit` (confirmação).

A Figura 31 é uma captura de tela mostrando a definição do tipo `ChaincodeStub`, comentado anteriormente.

Outro ponto fundamental para o entendimento é o *Asset*. Ele é uma estrutura de dados que representa algo, no caso desse trabalho, dados sobre uma credencial gerada. Um *Asset* é representado como uma coleção de pares chave-valor, sendo que eles podem ser representados em forma JSON ou binária (FABRIC, 2024).

A Figura 32 mostra a estrutura do *Asset* utilizado na construção dos contratos inteligentes implementados.

A Figura 33 mostra a implementação da função `ReadAsset`, que lê e retorna algum *Asset* do *ledger* baseado em seu identificador, enquanto a função `AssetExists` consulta o livro-razão para avaliar a existência ou não de um *Asset*.

Na Figura 34 está o código da função `CreateAsset`, que cria um *Asset* baseado nos dados recebidos por argumento. Nota-se que a maior parte deles são imutáveis, já que não existe uma função para fazer a atualização de um *Asset*. Isso ocorre pois a cadeia de custódia é representada por um encadeamento de credenciais verificáveis. Quando ela for atualizada, o *status* da credencial mais recente é alterado para *revoked*, enquanto uma nova VC é criada

```
// ChaincodeStub is an object passed to chaincode for shim side handling of
// APIs.
type ChaincodeStub struct {
    TxID          string
    ChannelID     string
    chaincodeEvent *pb.ChaincodeEvent
    args          [][]byte
    handler       *Handler
    signedProposal *pb.SignedProposal
    proposal      *pb.Proposal
    validationParameterMetakey string

    // Additional fields extracted from the signedProposal
    creator []byte
    transient map[string][]byte
    binding []byte

    decorations map[string][]byte
}
```

Figura 31 – ChaincodeStub, do arquivo stub.go do pacote shim (Hyperledger Fabric, 2024)

```
// Asset represents a verifiable credential on the ledger.
type Asset struct {
    Status      string `json:"status"`           // "active" or "revoked"
    Timestamp   string `json:"timestamp"`       // ISO8601 format or Unix timestamp
    OwnerDID    string `json:"owner_did"`       // Credential owner's DID
    IssuerDID   string `json:"issuer_did"`      // Issuer's DID
    CredentialID string `json:"credential_id"`    // Unique identifier of the credential
    CredentialHash string `json:"credential_hash"`
    LastModifierDID string `json:"last_modifier_did"` // DID of the last modifier
    PreviousCredentialID string `json:"previous_credential_id"`
}
```

Figura 32 – Asset representando dados relacionados a uma credencial verificável

```
// ReadAsset retrieves a credential from the ledger by its ID.
func (s *SmartContract) ReadAsset(
    ctx contractapi.TransactionContextInterface,
    credentialID string) (*Asset, error) {

    assetJSON, err := ctx.GetStub().GetState(credentialID)
    if err != nil {
        return nil, fmt.Errorf("falha ao ler do world state: %v", err)
    }
    if assetJSON == nil {
        return nil, fmt.Errorf("a credencial %s não existe", credentialID)
    }

    var asset Asset
    err = json.Unmarshal(assetJSON, &asset)
    if err != nil {
        return nil, err
    }
    return &asset, nil
}

// AssetExists checks whether a credential with the specified ID exists in the ledger.
func (s *SmartContract) AssetExists(ctx contractapi.TransactionContextInterface,
    credentialID string) (bool, error) {

    assetJSON, err := ctx.GetStub().GetState(credentialID)
    if err != nil {
        return false, fmt.Errorf("falha ao ler do world state: %v", err)
    }
    return assetJSON != nil, nil
}
```

Figura 33 – Funções de consulta de dados implementadas

com *status active* e adicionada na ponta desse encadeamento. A Figura 35 mostra o contrato inteligente que revogar um *Asset*, enquanto a Figura 36 ilustra como é feita a transferência do “dono” da credencial.

```
// CreateAsset creates a new credential on the ledger.
func (s *SmartContract) CreateAsset(
    ctx contractapi.TransactionContextInterface,
    credentialID string, status string, issuerDID string,
    ownerDID string, credentialHash string, timestamp string, previousCredentialID string) error {

    exists, err := s.AssetExists(ctx, credentialID)
    if err != nil {
        return err
    }
    if exists {
        return fmt.Errorf("a credencial %s já existe", credentialID)
    }

    asset := Asset{
        CredentialID:    credentialID,
        Status:          status,
        IssuerDID:       issuerDID,
        OwnerDID:        ownerDID,
        CredentialHash:  credentialHash,
        Timestamp:       timestamp,
        LastModifierDID: ownerDID,
        PreviousCredentialID: previousCredentialID,
    }
    assetJSON, err := json.Marshal(asset)
    if err != nil {
        return err
    }

    return ctx.GetStub().PutState(credentialID, assetJSON)
}
```

Figura 34 – Função CreateAsset, do *chaincode*

4.4.2 Aplicativo Typescript

Foram utilizadas bibliotecas externas para a criação de DIDs, VCs e sua verificação. A lista de todas as dependências deste componente se encontra no Apêndice A.

```
// RevokeAsset changes the status of a credential to "revoked".
func (s *SmartContract) RevokeAsset(ctx contractapi.TransactionContextInterface,
    credentialID string) error {

    asset, err := s.ReadAsset(ctx, credentialID)
    if err != nil {
        return err
    }

    asset.Status = "revoked"
    asset.Timestamp = time.Now().UTC().Format(time.RFC3339)

    assetJSON, err := json.Marshal(asset)
    if err != nil {
        return err
    }

    return ctx.GetStub().PutState(credentialID, assetJSON)
}
```

Figura 35 – Função RevokeAsset, do *chaincode*

```
// TransferOwnership updates the owner (OwnerDID) of an existing credential.
func (s *SmartContract) TransferOwnership(ctx contractapi.TransactionContextInterface,
    credentialID string, newOwnerDID string) error {

    asset, err := s.ReadAsset(ctx, credentialID)
    if err != nil {
        return err
    }

    asset.OwnerDID = newOwnerDID

    assetJSON, err := json.Marshal(asset)
    if err != nil {
        return err
    }

    return ctx.GetStub().PutState(credentialID, assetJSON)
}
```

Figura 36 – Função TransferOwnership, do *chaincode*

4.4.2.1 Inicialização

A Figura 37 mostra o código que inicializa a aplicação. Primeiramente, é criada uma conexão com o banco de dados MongoDB, em seguida é estabelecida uma conexão gRPC com o Fabric Gateway. Por fim, uma instância representando o contrato inteligente é salva em uma variável global, é através dela que as transações serão submetidas.

```
// Async function to handle the setup process
const startServer = async () => {
    try {
        console.log('Initializing server...');

        // Connect to MongoDB
        dbHandler = new DatabaseHandler(MONGO_URI);
        await dbHandler.connect();

        // Connect to Hyperledger Fabric Gateway
        const { gateway } = await createGatewayConnection();
        const network = gateway.getNetwork(channelName);
        contract = network.getContract(chaincodeName);

        // Start listening for HTTP requests
        app.listen(PORT, () => {
            console.log(`Server is running and listening on http://localhost:${PORT}`);
        });
    }
};
```

Figura 37 – Inicialização da aplicação Typescript

4.4.2.2 Interação com o Chaincode

As figuras 38 e 39 exemplificam como os contratos inteligentes são evocados no código Typescript. A função `createAsset` submete uma transação para `CreateAsset`, enquanto `readAssetByID` envia para `ReadAsset`. Além delas, foram definidas de forma análoga as funções `transferOwnership` e `revokeAsset`

```
* assetID is the credential ID (did)
*/
export async function createAsset(contract: Contract, assetID: string, owner_did: string,
  issuer_did: string, credential_hash: string,
  previous_credential_id: string): Promise<void> {
  //console.log(`\n-> Submit Transaction: CreateAsset, creates a new credential`);
  const status = 'active';
  const timestamp = new Date().toISOString();

  await contract.submitTransaction(
    'CreateAsset',
    assetID,
    status,
    issuer_did,
    owner_did,
    credential_hash,
    timestamp,
    previous_credential_id
  );
}
```

Figura 38 – Função `createAsset` no código Typescript

```
export async function readAssetByID(contract: Contract, assetId: string): Promise<Asset> {
  // console.log(`\n-> Evaluate Transaction: ReadAsset for ID: ${assetId}`);
  const resultBytes = await contract.evaluateTransaction('ReadAsset', assetId);

  const resultJson = utf8Decoder.decode(resultBytes);
  // We cast the parsed JSON to our Asset interface to get type safety.
  const result = JSON.parse(resultJson) as Asset;

  return result;
}
```

Figura 39 – Função `readAssetByID` no código Typescript

4.4.2.3 Criando DID e Credenciais

No arquivo `utils.ts`, há a função `createCredentialID`. Ela cria um DID do tipo `key` a partir de um par de chaves `Ed25519` e retorna seu identificador. Cada credencial criada no Gerenciador de Cadeia de Custódia tem seu próprio `did:key` e ele é usado como seu identificador único. Também possui a função `createIssuer` que automaticamente cria um assinador para o sistema.

Já no arquivo `vc-handler.ts`, a função `createCredential` é responsável por criar a credencial no formato JWT que fará parte da cadeia de custódia, enquanto a `verifyCreatedCredential` pode ser utilizada posteriormente para verificar sua autenticidade. O código de ambas estão disponíveis no Apêndice B.

A Figura 40 mostra o código da interface que define quais os valores que a VC criada terá como *payload*.

```

export interface EvidenceRecordData {
  what: string;
  who: string;
  where: string;
  when: string;
  why: string;
  how: string;
}

export interface CredentialSubjectData {
  id: string;
  evidence_hash: string;
  previous_credential_id: string | null;
  evidence_record: EvidenceRecordData;
}

```

Figura 40 – Código da Interface CredentialSubjectData

4.4.2.4 Lógica

A implementação inclui o arquivo `logic.ts`, que reúne as funções responsáveis pela comunicação direta com o *chaincode*, já abordadas anteriormente, além daquelas que compõem a lógica central do sistema, as quais serão discutidas a seguir.

Inicialmente, foi definida uma interface contendo os campos que um registro no MongoDB teria, como mostrado na Figura 41.

O campo `evidencehash` é referente ao hash calculado a partir da evidência digital, ele funciona como um identificador para a cadeia de custódia inteira já que é único e imutável.

Como discutido anteriormente, a cadeia de custódia é modelada como um encadeamento de credenciais verificáveis. Por esse motivo, existem os campos `credentialId`, `previousCredentialId`, `sequence` e `status`, sendo que este último pode assumir apenas os valores `'active'` ou `'revoked'` (ativo ou revogado). Somente a credencial mais recente na cadeia possui o status ativo, indicando que ela é a ponta do encadeamento; todas as anteriores apresentam o status como `'revoked'`.

Enquanto isso, `vcJwt` armazenará a credencial em si, codificada em base64. Por fim, `last_modifier_did` armazena o DID do último perito a realizar alguma modificação na cadeia de custódia, enquanto `ownerDid` é o DID do perito que possui a custódia da evidência no momento.

As funções implementadas são:

- `create_evidence;`
- `update_evidence;`

```

/**
 * Interface for the custody credential record in MongoDB.
 */
export interface CustodyCredentialRecord {
  _id?: ObjectId;
  evidencehash: string;
  credentialId: string;
  vcJwt: string;
  previousCredentialId: string | null;
  last_modifier_did: string;
  sequence: null | number;
  status: 'active' | 'revoked';
  issuerDid: string;
  ownerDid: string;
  createdAt: Date;
  tags: string[];
}

```

Figura 41 – Código da Interface CustodyCredentialRecord

- get_chain_of_custody;
- verify_chain_of_custody; e
- transfer_evidence_ownership.

A imagem 42 mostra a função `create_evidence`. Baseando-se nos dados recebidos em seu argumento, ela primeiro cria uma credencial verificável e então cria um registro no banco de dados, armazenando-a junto com outros metadados. Em seguida, o hash SHA256 da VC é calculado e é feita a requisição para escrita dele e outros metadados no *ledger*. Por fim, a função retorna o ID da credencial criada.

```

export async function create_evidence(
  subject_data: CredentialSubjectData, issuer: Issuer,
  dbHandler: DatabaseHandler, RecordData: CustodyCredentialRecord, contract: Contract): Promise<string> {
  // Creates credential and store in database.
  const vcJwt = await createCredential(subject_data, issuer);
  RecordData.vcJwt = vcJwt;
  RecordData.last_modifier_did = RecordData.ownerDid;
  RecordData.sequence = 1;
  RecordData.evidencehash = subject_data.evidence_hash;

  await dbHandler.createRecord(RecordData);

  const credentialHash = createHash('sha256').update(vcJwt).digest('hex');
  // Stores some data in blockchain.
  await createAsset(contract, RecordData.credentialId, RecordData.ownerDid,
    RecordData.issuerDid, credentialHash, "");

  return RecordData.credentialId;
}

```

Figura 42 – Código da `create_evidence` do arquivo `logic.ts`

Os demais códigos das funções podem ser encontrados no Apêndice C. A Figura 43 é um fluxograma mostrando em mais alto nível os passos para atualizar uma cadeia de custódia através da função `update_evidence`.

Já a Figura 44 é um fluxograma representando o algoritmo, implementado pela função `get_chain_of_custody` para retornar os dados referentes a toda a cadeia de custódia, tanto os armazenados *off-chain*, como os armazenados *on-chain*.

A obtenção do *payload* das credenciais geradas é feita somente após a verificação de sua autenticidade, como mostra o fluxograma da função `verify_chain_of_custody` na Figura 45.

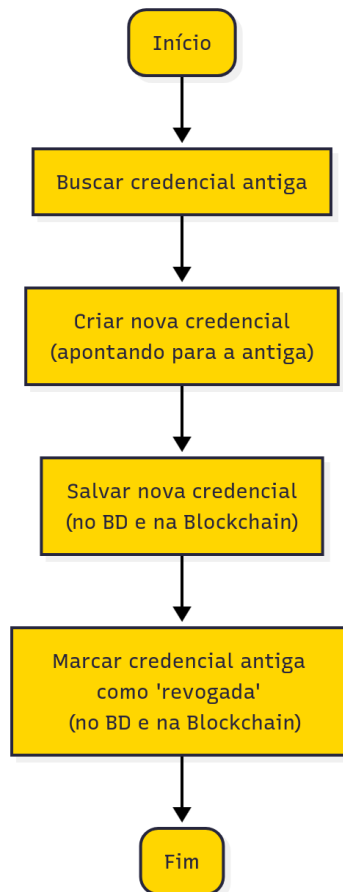


Figura 43 – Fluxograma da função `update_evidence` do arquivo `logic.ts`

4.4.2.5 Servidor Web

Foi construído um servidor com a biblioteca *express* contendo os *endpoints* `/create-evidence`, `/chain-of-custody`, `/verify-chain-of-custody`, `/update-evidence` e `/transfer-ownership`.

Em geral, os *endpoints* fazem validações dos dados recebidos e então chamam funções do arquivo `logic.ts`. Por fim, mais algumas validações para certificar-se de que os dados recebidos pelo *payload* correspondem aos armazenados na Cadeia de Custódia, ou seja, uma verificação de segurança.

Uma ressalva importante é o `/chain-of-custody`. A credencial está armazenada por inteiro no MongoDB e, a princípio, teria que ser retornada junto com os outros dados *off-chain* e *on-chain*. Porém, isso violaria o nível de acesso — que será comentado ainda nesta seção —

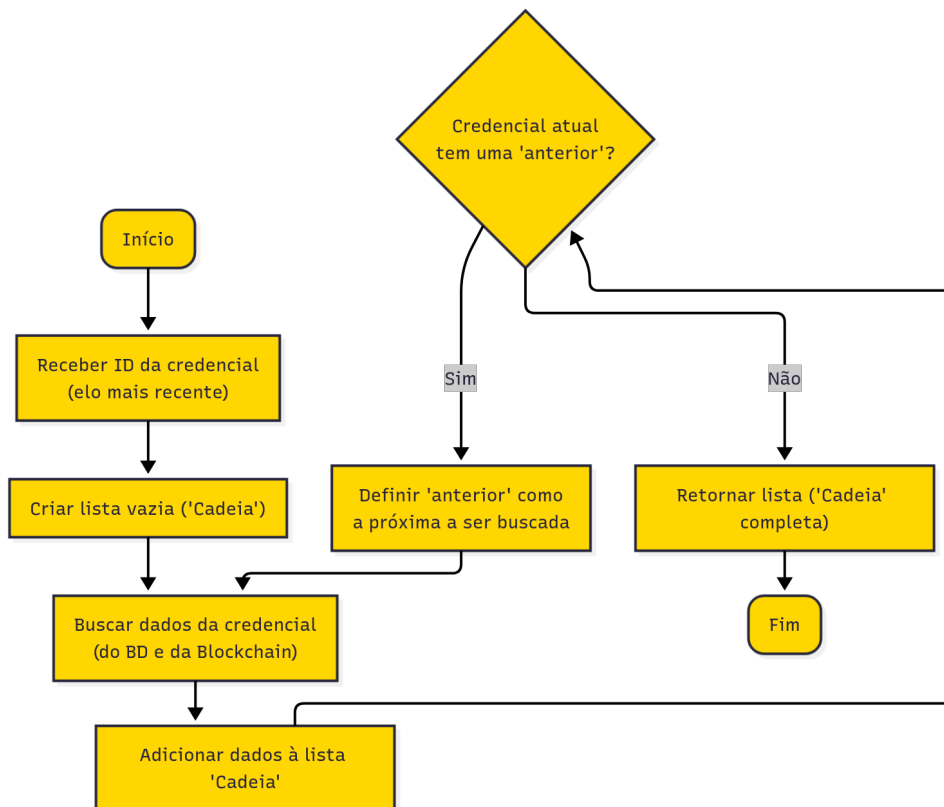


Figura 44 – Fluxograma da função `get_chain_of_custody` do arquivo `logic.ts`

utilizado por este *endpoint*. Portanto, a credencial é substituída por um *placeholder* genérico no retorno dos dados.

Este parágrafo inicia a descrição dos *payloads* esperados por cada um dos *endpoints*. O `/create-evidence` espera receber um JSON contendo o hash da evidência (`evidence_hash`), o DID do *Holder* (`owner_did`) e um `evidence_record`, de formato análogo ao `EvidenceRecordData` ilustrado pela Figura 41.

O `/update-evidence` espera receber exatamente o mesmo *payload*. Porém, a partir de agora é possível enviar o `credential_id` para localizar a cadeia de custódia ao invés do hash da evidência. Esses dois *endpoints* comentados aceitam *tags*, que são opcionais.

O `/chain-of-custody` e o `/verify-chain-of-custody` devem receber o DID do *Holder* e o hash da evidência. Enquanto isso, o `/transfer-ownership` espera receber o hash da evidência, o DID do novo dono da custódia (`new_owner_did`) e o DID do *Holder* (`current_owner_did`).

A toda requisição recebida, será exigida uma VC contendo dados do *Holder*, como por exemplo o nível de acesso que lhe foi concedido. Esses dados são obtidos através do protocolo de verificação do Hyperledger Indutus. Após a verificação dela sem erros, o *payload* da VC é obtida.

A Figura 46 mostra parte do código do *endpoint* `/create-evidence` que realiza o processo de iniciar a verificação da credencial, começando com um *request* ao *Verifier* para gerar um *PresentationRequest* e então realizar um *poll* dos dados na expectativa do protocolo ter sido um sucesso e de que em breve o *payload* da apresentação da prova estará disponível

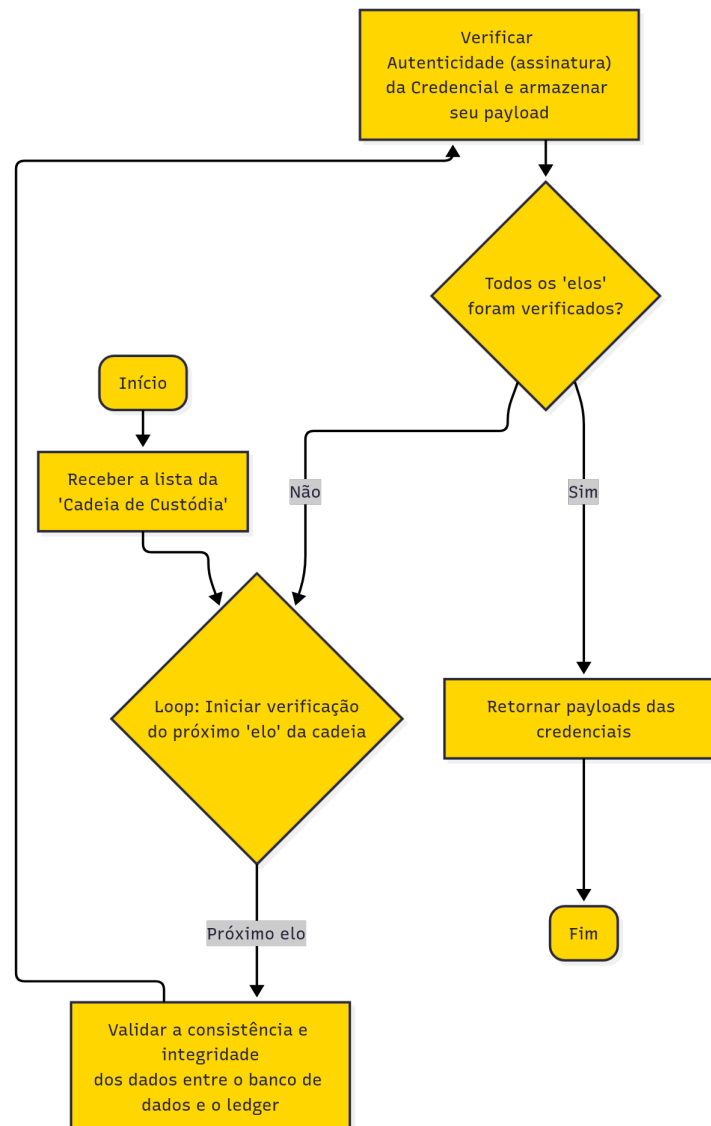


Figura 45 – Fluxograma da função `verify_chain_of_custody` do arquivo `logic.ts`

no agente do verificador. Todos os *endpoints* possuem um código análogo a esse. Destaca-se o campo `level_required`, que é o nível de acesso mínimo necessário que o *Holder* precisa ter para pedir que a ação de um *endpoint* seja realizada. Este atributo estará presente em todas as credenciais verificáveis emitidas para um *Holder*.

Os níveis de acesso são representados por inteiros não negativos. Os possíveis modos pensados para esta implementação são:

- *Restricted Read-Only*: representado pelo número 0, é o modo de apenas leitura restrito. Somente alguns dados e metadados poderão ser lidos. O *endpoint* `/chain-of-custody` é o único acessível com este nível de permissão.
- *Read-Only*: representado pelo número 1, é o modo de apenas leitura. Nele, todos os dados da cadeia de custódia podem ser obtidos. Os *endpoints* acessíveis neste modo são o `/chain-of-custody` e o `/verify-chain-of-custody`.

- *Read and Write*: representado pelo número 2, tem acesso total à cadeia de custódia, podendo adicionar dados. Todos os *endpoints* são acessíveis com este nível de permissão, com a ressalva de que para realizar a mudança de custódia de uma evidência, ainda é necessário que o DID do *Holder* seja igual *ownerDid* armazenado nos registros sobre a credencial.
- *ADMIN*: nível máximo de permissão nesse sistema. Permite, por exemplo, alterar o *ownerDid* como quiser. Em outras palavras, ele pode mudar a custódia de uma evidência sem possuir previamente a custódia da mesma.

```
// --- Request Presentation ---
console.log('Requesting presentation with identifier: ${identifier}');
await axios.post(`${VERIFIER_API_URL}/presentation_request`, {
  connection_id: connectionId,
  cred_def_guid: credential_guid,
  level_required: 2,
  id_database: identifier
});

// --- Poll for Verified Data ---
const verified_data = await pollForVerifiedData(identifier);
console.log(verified_data['requested_proof']['revealed_attrs']);

const subject_did = verified_data['requested_proof']['revealed_attrs']['subject_did_proof']['raw']
const expert_name = verified_data['requested_proof']['revealed_attrs']['expert_name_proof']['raw']
console.log(expert_name)
const evidence_hash_cred = verified_data['requested_proof']['revealed_attrs']['evidence_hash_proof']['raw']
```

Figura 46 – Processo de verificação da credencial do *Holder* que ocorre similarmente em todos os *endpoints*.

4.4.3 Script para Execução

Para a inicialização do Gerenciador de Cadeia de Custódia foram construídos os *scripts* *run.sh*, *stop.sh* e *run_network.sh*, que configuram e inicializam a rede do Hyperledger Fabric, o MongoDB através do Docker e a aplicação em Typescript.

Nota-se que esta prova de conceito foi feita pensando na execução local do programa, utilizando a rede de teste padrão do Hyperledger Fabric.

4.5 CONTROLE DE ACESSO SSI

É utilizado o *release* v2.0.0 que, até o momento da escrita deste trabalho, é a mais recente do *Cloud Agent* do Hyperledger Identus.

4.5.1 Inicializando os Agentes

O repositório do agente em nuvem do Identus possui diversos exemplos de como inicializar os agentes com diferentes configurações (HYPERLEDGER, 2024b). Utilizou-se o mo-

delo `st-multi` como base para criar o arquivo, cujo código está disponível no Apêndice D, `compose.yaml`. Através do Docker Compose, três agentes são inicializados, um para o *Holder*, outro para o *Issuer* e o último para o *Verifier*.

Destaca-se a necessidade de adicionar `ENABLE_ANONCRED: true` como variável ambiente, o que não era necessário em versões anteriores. Tal mudança aparece nas notas de atualização da *release* `v2.0.0`, porém ainda não foi adicionada nas documentações (Hyperledger Identus, 2025a). Situações como essa ocorreram algumas vezes durante o desenvolvimento e, geralmente, os testes de integração que o repositório possui eram suficientes para sanar dúvidas. Além disso, existe um canal de comunicação oficial através do Discord para obter suporte dos desenvolvedores e comunidade: <https://discord.gg/hyperledger>.

Através desse canal de comunicação foi descoberto que os exemplos disponibilizados estavam desatualizados. Antes, a imagem dos agentes era obtida através de `ghcr.io/hyperledger/identus-cloud-agent:1.38.0`, uma versão mais antiga do agente, quando o correto seria `hyperledgeridentus/identus-cloud-agent:2.0.0`. No dia 29 de setembro de 2025, um *commit* ao repositório principal tratou desse problema, atualizando os arquivos de exemplo (LOSOPONKUL, 2025).

4.5.2 Script para Execução

Foram criados os *scripts* `start_services.sh` e `stop_services.sh` para auxiliar na inicialização e depois encerramento dos serviços necessário do Controle de Acesso SSI, excetuando-se os agentes em nuvem do Identus.

4.5.3 Componente Holder

Escrito em Python, possui três arquivos. O primeiro é o `holder_controller.py`, que implementa funções para enviar *requests* – os quais já foram explicados na seção 2.6.3 – para o agente em nuvem. O arquivo `holder_api.py` cria um servidor no qual o *issuer* pode mandar notificações para ele. Poderia ter sido usado um *webhook* ao invés das funções do `holder_api.py`, com exceção do *endpoint* `/receive\_oob\_invitation` que simula o recebimento do convite OOB por parte do *Holder*.

O último arquivo é o `holder_interface.py`. Ele implementa a interface em terminal que é utilizada pelo usuário. Ao iniciar o programa, é possível escolher entre “Criar DID” ou “Operação no Sistema de Cadeia de Custódia”. Para realizar uma operação, é necessário ter criado um DID e possuir credencial verificável assinada pelo *Issuer*. Os passos de aceitar conexão, credencial e apresentar prova são feitas automaticamente nessa prova de conceito.

Ao escolher a alternativa para realizar uma operação, são mostradas as opções que representam cada um dos *endpoints* implementados pela aplicação em Typescript no Gerenciador de Cadeia de Custódia. Os dados necessários para construir o *payload* são então pedidos ao usuário via terminal. A Figura 47 mostra a parte do código em Python que se comunica com o

usuário mostrando as possíveis opções, além de ilustrar como o programa pede dados adicionais ao perito.

```

user_input = input("""Selecione:
\t1. Criar Evidência.
\t2. Atualizar Evidência.
\t3. Consulta metadados Cadeia de Custódia
\t4. Verificar Cadeia de Custódia e obter payloads
\t5. Mudar owner Cadeia de Custódia
\t0. Sair\n""")
if user_input == '1':
    tags_input = input('Digite tags (separadas por vírgula) se desejar, ou aperte enter: ')
    tags = [tag.strip() for tag in tags_input.split(',')] if tags_input else []
    evidence_hash = input('Digite o hash da evidência: ')
    print("Responda...")
    what = input("What? ")
    who = input("Who? ")
    where = input("Where? ")
    when = input("When? ")
    why = input("Why? ")
    how = input("How? ")

```

Figura 47 – Parte do código que se comunica com o perito via terminal

4.5.4 Componente Issuer/Verifier

Nesta prova de conceito, o *Issuer* e o *Verifier* são o mesmo agente. Ambos possuem seu controlador, através do `verifier_controller.py` e do `issuer_controller.py`, de forma análoga ao *Holder*.

O `webhook_handler.py` implementa um *webhook* (Hyperledger Identus, 2025b) para o *Issuer*. Ele é notificado quando algum estado interno do agente é alterado e toma ações para continuar os protocolos já comentados na seção 2.6.3 do *Identus*.

O `anoncreds_schema.py` define o *schema* que é publicado para futura criação de credenciais e pode ser encontrado no Apêndice F.

Enquanto isso, o `verifier_api.py` é usado para notificar o *Verifier* para que alguma ação seja tomada. Ele é usado pelo Gerenciador de Cadeia de Custódia para iniciar o processo de apresentação de provas.

Por fim, o `issuer_interface.py` é a interface em terminal que será usada por quem dá acesso à evidência ao perito. Três ações são possíveis: “Criar DID, Schema e Credential Definition”, “Criar conexão e enviar para Holder” e “Assinar Credencial”. Todas essas alternativas simplesmente se comunicam com as funções correspondentes no `issuer_controller.py` e, quando necessário, requerem dados pelo terminal.

4.5.5 Erros Documentação

Em alguns pontos, a documentação está desatualizada ou com *payloads* exemplo errados. Já foram comentados na seção 4.2.1 a existência de um *Discord* oficial, da existência de testes de integração que geralmente suprem erros de documentação e também um exemplo de erro – o do *ENABLE_ANONCRED*. Outros erros foram encontrados e serão comentar a seguir.

O Código 4.1 mostra como deveria ser formatado o campo `schemaId` para criar um *Credential Definition* segundo a documentação oficial, enquanto o Código 4.2 mostra a formatação correta a ser utilizada. O *placeholder* `schemaRegistryURL` usado na implementação foi `http://caddy-issuer:8080/cloud-agent/`, enquanto a documentação utiliza `http://host.docker.internal:8080/cloud-agent/`, o que não é um erro, essa diferença se explica pelo `compose.yaml` que é utilizado para iniciar os agentes neste trabalho.

O *Presentation Request* para *AnonCreds* também estava errado na documentação. A versão utilizada no tutorial e a versão utilizada no código da implementação compõem o Apêndice E deste trabalho.

Código 4.1 – Campo `schemaId` na documentação oficial (Hyperledger Indentus, 2025a)

```
"schemaId": "http://host.docker.internal:8080/cloud-agent/schema-registry/schemas/{{SCHEMA_ID}}"
```

Código 4.2 – Campo `schemaId` utilizado na implementação

```
"schemaId": f"{schemaRegistryURL}/schema-registry/schemas/{schema_guid}/schema"
```

4.6 LIMITAÇÕES

A PoC criada possui algumas limitações. Um exemplo é o já citado fato do *Holder* não estar usando *webhook*, o que simplificaria algumas interações e deixaria a arquitetura mais natural.

Além disso, a prova de conceito foi pensada para que o *Issuer* consiga ter uma interação completa com um *Holder* somente. Para ampliar isso, seria necessário implementar uma maneira do *Issuer* gerenciar corretamente suas conexões que podem vir a estar em fases diferentes de protocolos.

Em alguns momentos, foram utilizados bancos de dados locais para o compartilhamento de determinadas informações — por vezes entre o *Holder* e o *Issuer*, por vezes entre o Gerenciador da Cadeia de Custódia e outro agente. Nesse aspecto, a arquitetura ainda precisa ser estudada para identificar uma solução mais adequada.

Atrelado a esse ponto, permanecem em aberto alguns questionamentos sobre a integração com sistemas externos, isto é, sistemas que poderiam interagir com a solução proposta neste trabalho. Por exemplo, supõe-se que os peritos possuam cadastro em algum sistema institucio-

nal, o qual poderia ser utilizado para associar o perito aos DIDs gerados. Assim, os bancos de dados locais funcionaram, de certa forma e em alguns momentos, como uma solução provisória para representar a integração esperada com sistemas externos reais.

Ressalta-se que a integridade e autenticidade da cadeia de custódia também está ligada a esses sistemas externos, em como um perito ou um emissor se autenticam e nos protocolos de comunicação utilizados. Isso ocorre pois embora as tecnologias empregadas visem garantir maior segurança, vulnerabilidades nesses acessos, por exemplo, podem violar a cadeia de custódia.

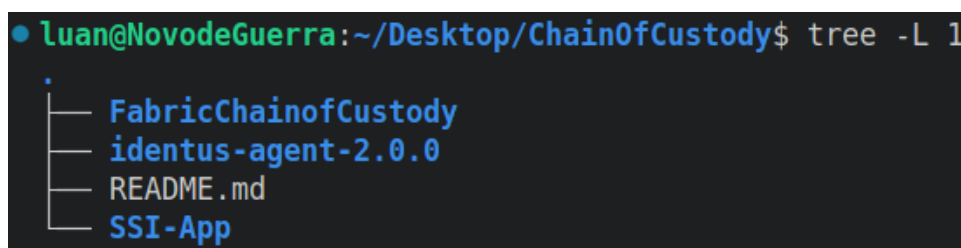
Por fim, reforça-se o já citado fato do PRISM Node não estar utilizando uma rede da Cardano na prática. Como próximos passos, podem ser adicionados a execução de uma *testnet* local e então fazer com que o PRISM Node rode em cima dela.

Todos esses seriam objetos de melhoria e de futura pesquisa sobre a arquitetura que foi implementada.

4.7 DEMONSTRAÇÃO DE EXECUÇÃO

O código-fonte completo desta implementação está disponível publicamente em um repositório no GitHub: <https://github.com/Luan-Diniz/ChainOfCustody>.

Considerando que o arquivo raiz do projeto é o diretório *ChainOfCustody*, a Figura 48 mostra como a implementação está estruturada. A pasta *SSI-App* é referente ao componente Controle de Acesso SSI, enquanto o *FabricChainofCustody* se refere ao Gerenciador de Cadeia de Custódia. O *identus-agent-2.0.0* contém o arquivo *compose.yaml* que é usado para inicializar os agentes e o *README.md* contém instruções para a execução da implementação como um todo.



```

• luan@NovodeGuerra:~/Desktop/ChainOfCustody$ tree -L 1
.
├── FabricChainofCustody
├── identus-agent-2.0.0
├── README.md
└── SSI-App
  
```

Figura 48 – Estrutura do projeto

Primeiro, o comando `./run.sh` deve ser executado dentro do diretório *FabricChainofCustody/*, dessa forma inicializando a rede do Hyperledger Fabric e a Aplicação em TypeScript.

Em outro terminal, dentro do diretório *identus-agent-2.0.0/st-multi*, deve-se executar o comando `docker compose up -d`. Isso irá inicializar os agentes e demais componentes do Identus, como o PRISM Node.

Agora dentro de *SSI-App*, o comando `./start_services.sh` deve ser executado. Antes de rodar as interfaces, é necessário instalar os requisitos listados em *requirements.txt*, pode-se usar o comando `pip install -r requirements.txt`, embora o recomendável seja

criar um ambiente virtual (venv) (Python Software Foundation, 2025) para instalar essas dependências.

Em outro terminal, dentro de SSI-App executa-se `python3 issuer/issuer_interface.py`. E em um terminal separado, dentro de SSI-App executa-se `python3 holder/holder_interface.py`.

A interface do perito (*Holder*) é mostrada pela Figura 49, nela é possível ver que o sistema pede o nome completo e então dá as opções disponíveis, nesse caso, o perito decidiu criar um DID para ele.

```

Luan@NovodeGuerra:~/Desktop/ChainOfCustody/SSI-App$ python3 holder/holder_interface.py
(LogIn) Digite seu nome completo:
Luan Diniz
Atenção! Nome não registrado no banco de dados.
Crie um DID!
Selecione:
1. Criar DID.
2. Operação no Sistema de Cadeia de Custódia
0. Sair
1
DID criado: did:prism:ea5331907b71bebdb7b38fa2bde8b5194741b9af86777cd1663bc416883e04cc
DID atualizado no sistema.
(LogIn) Digite seu nome completo:
Luan Diniz
Seu DID atual: did:prism:ea5331907b71bebdb7b38fa2bde8b5194741b9af86777cd1663bc416883e04cc
Selecione:
1. Criar DID.
2. Operação no Sistema de Cadeia de Custódia
0. Sair

```

Figura 49 – *Holder* criando um DID

A interface do *Issuer* é a captura de tela da Figura 50. Nela, aparecem as ações possíveis, das quais é escolhida a primeira delas, que cria um DID para o *Issuer* e então publica o *Schema* e a *Credential Definition*.

Então, é possível estabelecer uma conexão com um *Holder*, como mostra a Figura 51.

Por fim, o *Issuer* pode assinar uma VC AnonCreds que dá acesso ao perito para que ele consiga usar o sistema que gerencia a cadeia de custódia. Esse processo está ilustrado na Figura 52 e destaca-se a necessidade de definir o tempo de validade da credencial, o hash da evidência digital na qual o perito será autorizado a interagir e o nível de acesso que ele possuirá, nesse caso, *Read and Write*, acesso de nível 2.

Na verificação este nível de acesso é verificado via ZKP. Por exemplo, o predicado para avaliar se o perito é um ADMIN (nível 99) seria `authorization_level >= 99`, isto é definido no corpo do *Presentation Request*, que já foi comentado previamente e possui um exemplo no Apêndice E. Usar ZKP para avaliar este atributo só é possível pois quanto maior o acesso ao sistema, maior o inteiro que o representa.

Após ter sua credencial de acesso, o *Holder* pode fazer operações no Gerenciador de

```

Luan@NovodeGuerra:~/Desktop/ChainOfCustody/SSI-App$ python3 issuer/issuer_interface.py
Selecione:
1. Criar DID, Schema e Credential Definition.
2. Criar conexão e enviar para Holder.
3. Assinar Credencial
0. Sair

1
DID criado: did:prism:b75ae6bba52d76ed92e52e36726a22b17fc0f95b01a86fb60d9b30c216a3d1c2
DID adicionado à lista de Issuers confiáveis!
Schema criado: 72920dfb-0d69-3c7d-8059-89654a0d7871
Credential Definition criado: 6f91f22a-3f63-344f-a03d-52974501f320
Selecione:
1. Criar DID, Schema e Credential Definition.
2. Criar conexão e enviar para Holder.
3. Assinar Credencial
0. Sair

```

Figura 50 – *Issuer* criando DID, *Schema* e *Credential Definition*

```

Selecione:
1. Criar DID, Schema e Credential Definition.
2. Criar conexão e enviar para Holder.
3. Assinar Credencial
0. Sair

2
Para que a conexão prossiga e a credencial seja gerada responda...
Qual o nome completo do perito que receberá a credencial?
Luan Diniz
Sucesso! A identidade de 'Luan Diniz' foi encontrada.
Dados: {'current_did': 'did:prism:ea5331907b71bebdb7b38fa2bde8b5194741b9af86777cd1663bc416883e04cc', 'older_dids': []}
Dê um rótulo para esta nova conexão: ConexaoTeste
MockDB responded with status: 200
Invitation sent. Holder agent responded with status: 200

```

Figura 51 – *Issuer* estabelecendo conexão com um perito

```

2. Criar conexão e enviar para Holder.
3. Assinar Credencial
0. Sair

3
Esteja certo de que o perito aceitou sua solicitação.
Qual será o tempo de validade da credencial gerada (em segundos)? 7200
Qual o hash da evidência digital? 4c0e0460dde9152e10c9099aaf22ddb7fe07726fe6cbbab245d213d39321dbd
Digite a jurisdição do tribunal (ex: Comarca de Florianópolis, 4ª Vara Federal): TJSC
Qual seu identificador (id)? 1
Níveis de Acesso
0 (Restricted Read-only): Apenas alguns dados e metadados poderão ser lidos.
1 (Read-Only): Todos os dados da cadeia de custódia poderão ser visualizados
2 (Read and Write): Tem acesso à cadeia de custódia e pode adicionar dados.
99 (ADMIN): Acesso total para modificações, apenas deve ser usados em casos extremos.
(Como para transferir a custódia de uma evidência de um perito que não está mais elegível
a utilizar este ecossistema a outro.)

Qual o nível de acesso o perito terá à evidência (digite somente um número)? 2
Invitation sent. Holder agent responded with: {'status': 'success', 'connection_details': {'recordId': 'bbf5fab7-628b-4ecf-a080-d28cba69206',
thid': '7d35649f-358b-4678-8f15-f2283a77828c', 'credentialFormat': 'AnonCreds', 'claims': {'authorization_level': '2', 'issuing_judge_id':

```

Figura 52 – *Issuer* assinando credencial para um perito

Cadeia de Custódia. A Figura 53 mostra o perito escolhendo a opção de “Criar Evidência”, ou seja, iniciar uma cadeia de custódia. A partir do terminal ele escreve os campos necessários para documentar esse processo relacionado à evidência.

Em toda interação do perito com o sistema, sua credencial é verificada e seus atributos avaliados. Caso algo não seja aprovado por esta verificação, um erro é retornado ao usuário.

```

Selecionar:
    1. Criar Evidência.
    2. Atualizar Evidência.
    3. Consulta metadados Cadeia de Custódia
    4. Verificar Cadeia de Custódia e obter payloads
    5. Mudar owner Cadeia de Custódia
    0. Sair
1
Digite tags (separadas por vírgula) se desejar, ou aperte enter:
Digite o hash da evidência: 4c0e0460dde9152e10c9099aaf22ddb7fe07726fe6cbbcab245d213d39321dbd
Responda...
What? Uma imagem forense bit a bit do disco principal (SSD de 1TB, modelo genérico) do servidor 'SRV-WEB-01'
Who? Imagem criada remotamente por uma analista de Resposta a Incidentes, com certificação XXX, da equipe interna de cibersegurança
Where? O servidor físico está localizado no rack 10, posição 5 da empresa 'Dados S.A', na Rua X, 777, Bloco C, Florianópolis-SC
When? Aquisição da imagem iniciada em 15 de fevereiro de 2025, às 03:00 (horário local) e concluída 05:00 no mesmo dia.
Why? Para apurar um incidente de segurança, provavelmente um ataque 'SQL Injection'
How? Foi utilizado uma ferramenta de cópia forense via SSH a partir de uma estação de trabalho de análise.
CreateEvidence response: {'message': 'Evidence created successfully', 'credentialId': 'did:key:z6MkuYULpnQELRWUQxB6a5EgK2NLNkdXxFroiygcoaozAE6j'}

```

Figura 53 – Perito criando uma cadeia de custódia

A 54 mostra o perito “atualizando a evidência”, isto é, adicionando um novo registro sobre a evidência à cadeia.

```

    2. Atualizar Evidência.
    3. Consulta metadados Cadeia de Custódia
    4. Verificar Cadeia de Custódia e obter payloads
    5. Mudar owner Cadeia de Custódia
    0. Sair
2
Digite tags (separadas por vírgula) se desejar, ou aperte enter:
Digite o hash da evidência: 4c0e0460dde9152e10c9099aaf22ddb7fe07726fe6cbbcab245d213d39321dbd
Responda...
What? Início da análise pericial da imagem forense. O hash foi conferido e montado como 'read-only'.
Who? Perito John Doe, ID 12345, da equipe de Análise Forense
Where? Estação de trabalho forense 'FOR-WS-01', localizada no Laboratório de Análise Forense da 'Dados S.A', Sala 10B
When? 28 de outubro de 2025, 14:00 (horário local)
Why? Para extrair artefatos (logs do servidor web, scripts, arquivos de banco de dados) relacionados à suspeita de 'SQL Injection', conforme solicitação do incidente
How? A imagem foi montada em modo 'read-only' utilizando a ferramenta 'Arsenal Image Mounter' e a análise foi iniciada com o software 'Autopsy 4.19'
UpdateEvidence response: {'message': 'Evidence updated successfully', 'newCredentialId': 'did:key:z6MkmjDu9n3w76FCsEqZfsRUvSYU1nmnCtXNTtDmWk6dKhy'}

```

Figura 54 – Perito adicionando um elo à cadeia de custódia

As Figuras 55 e 56 mostram, respectivamente, o perito obtendo os metadados relacionados à cadeia de custódia e então verificando e obtendo o *payload* das credenciais que representam essa cadeia de custódia. Ambos os *outputs* completos estão disponíveis no Apêndice G deste trabalho.

Por fim, a Figura 57 mostra o perito realizando a transferência de custódia para alguém com o DID `did:example:123`. Após realizada essa ação, o perito tenta realizar a transferência novamente, para outro DID, porém ele obtém um erro, já que não possui mais a custódia e não é um ADMIN, como mostra a Figura 58.

```

3. Consulta metadados Cadeia de Custódia
4. Verificar Cadeia de Custódia e obter payloads
5. Mudar owner Cadeia de Custódia
0. Sair

3
Digite o hash da evidência: 4c0e0460dde9152e10c9099aaf22ddb7fe07726fe6cbbbeab245d213d39321dbd
Chain of Custody Metadata:

--- INÍCIO DA CADEIA DE CUSTÓDIA ---

----- Elo 1 -----

[+] Registro do Banco de Dados:
{
  "id": "6900dccb68082c1aa359f892",
  "evidencehash": "4c0e0460dde9152e10c9099aaf22ddb7fe07726fe6cbbbeab245d213d39321dbd",
  "credentialId": "did:key:z6MkmjDu9n3w76FCsEqZfsRUVsYU1nmnCETXNTtDmwk6dKhy",
  "vcJwt": "vcJWTPlaceholder",
  "previousCredentialId": "did:key:z6MkuYULpnQELRWUQxB6a5EgK2NLNKDxxFroiYgcoaozAE6j",
  "last_modifier_did": "did:prism:ea5331907b71bebdb7b38fa2bde8b5194741b9af86777cd1663bc416883e04cc",
  "sequence": 2,
  "status": "active",
  "issuerDid": "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWmo1W5q3BYTgXXVgx",
  "ownerDid": "did:prism:ea5331907b71bebdb7b38fa2bde8b5194741b9af86777cd1663bc416883e04cc",
  "createdAt": "2025-10-28T15:10:03.266Z",
  "tags": []
}

```

Figura 55 – Perito obtendo os metadados da cadeia de custódia

```

4. Verificar Cadeia de Custódia e obter payloads
5. Mudar owner Cadeia de Custódia
0. Sair

4
Digite o hash da evidência: 4c0e0460dde9152e10c9099aaf22ddb7fe07726fe6cbbbeab245d213d39321dbd
Chain of Custody Payloads:
----- INÍCIO PAYLOADS -----
{
  "message": "Chain of custody verified successfully",
  "payloads": [
    {
      "verified": true,
      "payload": {
        "vc": {
          "@context": [
            "https://www.w3.org/2018/credentials/v1"
          ],
          "type": [
            "VerifiableCredential",
            "Evidence"
          ],
          "credentialSubject": {
            "evidence_hash": "4c0e0460dde9152e10c9099aaf22ddb7fe07726fe6cbbbeab245d213d39321dbd",
            "previous_credential_id": "did:key:z6MkuYULpnQELRWUQxB6a5EgK2NLNKDxxFroiYgcoaozAE6j",
            "evidence_records": {
              "what": "In\u00e9dicio da an\u00e9lise pericial da imagem forense. O hash foi conferido e montado como 'read-only'.",
              "who": "Perito John Doe, ID 12345, da equipe de An\u00e9lise Forense",
              "where": "Esta\u00e9 no 7\u00e9 de trabalho forense 'FOR-WS-01', localizada no Laborat\u00f3rio de An\u00e9lise Forense da 'Dados S.A', Sala 10B",
              "when": "28 de outubro de 2025, 14:00 (hor\u00e9rio local)",
              "why": "Para extrair artefatos (logs do servidor web, scripts, arquivos de banco de dados) relacionados \u00e0 suspeita de 'SQL Injection', conforme solicita\u00e7\u00e3o do incidente",
              "how": "A imagem foi montada em modo 'read-only' utilizando a ferramenta 'Arsenal Image Mounter' e a an\u00e9lise foi iniciada com o software 'Autopsy 4.19'"
            }
          }
        },
        "sub": "did:key:z6MkmjDu9n3w76FCsEqZfsRUVsYU1nmnCETXNTtDmwk6dKhy",
        "nbf": 1761664203,
        "iss": "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWmo1W5q3BYTgXXVgx"
      }
    ]
  }
}

```

Figura 56 – Perito verificando a cadeia de custódia e obtendo o *payload* das credenciais

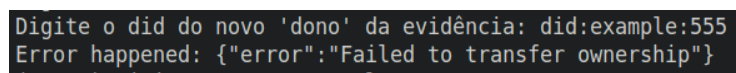
```

5. Mudar owner Cadeia de Custódia
0. Sair

5
Digite o hash da evidência: 4c0e0460dde9152e10c9099aaf22ddb7fe07726fe6cbbbeab245d213d39321dbd
Digite o did do novo 'dono' da evidência: did:example:123
Transfer Ownership response: {'message': 'Ownership transferred successfully', 'credentialId': 'did:key:z6MkmjDu9n3w76FCsEqZfsRUVsYU1nmnCETXNTtDmwk6dKhy', 'newOwner': 'did:example:123'}

```

Figura 57 – Perito transferindo a custódia

A terminal window with a dark background and light-colored text. The first line shows a prompt followed by the command 'did:example:555'. The second line shows an error message in JSON format: {'error': 'Failed to transfer ownership'}.

```
Digite o did do novo 'dono' da evidência: did:example:555  
Error happened: {"error": "Failed to transfer ownership"}
```

Figura 58 – Perito sem a custódia tentando transferir novamente e obtendo erro.

5 CONCLUSÃO

Este Trabalho de Conclusão de Curso propôs e implementou uma arquitetura para a cadeia de custódia forense que integra a imutabilidade da tecnologia *blockchain* ao controle de acesso proporcionado pela Identidade Autossoberana. A proposta surgiu da necessidade de garantir a integridade e a autenticidade de evidências digitais, contemplando tanto seu registro seguro quanto a definição de quem possui autorização para acessá-las.

Os objetivos definidos foram alcançados por meio da elaboração de uma prova de conceito funcional. A solução desenvolvida empregou o Hyperledger Fabric para o registro *on-chain* de metadados das evidências e o Hyperledger Identus para o controle de acesso dos peritos. A cadeia de custódia foi modelada como um encadeamento de credenciais verificáveis no formato JWT, enquanto o acesso foi concedido por meio de VCs baseadas em AnonCreds. A aplicação de provas de conhecimento zero possibilitou a verificação do nível de autorização dos peritos sem a exposição de informações sensíveis.

Além disso, em comparação aos trabalhos correlatos, este TCC avança no estado da arte ao preencher lacunas relevantes. Enquanto implementações anteriores concentraram-se na cadeia de custódia com Hyperledger Fabric (LOFFI et al., 2025b), a revisão sistemática de (LOFFI et al., 2025a) indicou que a integração com a Identidade Autossoberana permanecia subexplorada e que havia necessidade de aprofundar pesquisas sobre privacidade, especialmente no uso de provas de conhecimento zero. Este trabalho responde diretamente a essas lacunas ao integrar o Fabric com um controle de acesso robusto baseado em SSI e detalhar uma implementação prática que explora o uso do Hyperledger Identus e AnonCreds neste contexto, aplicando ZKPs para a verificação de permissões. Adicionalmente, propõe-se uma arquitetura inovadora que articula uma rede permissionada (Fabric) e uma rede pública (idealmente a *Cardano*, por meio do componente PRISM Node, voltado ao gerenciamento de identidades descentralizadas).

Apesar dos resultados obtidos, a prova de conceito apresenta limitações que podem ser aprimoradas em estudos futuros. Recomenda-se o aperfeiçoamento da arquitetura proposta e sua integração com sistemas externos reais. Sugere-se, ainda, a comparação entre diferentes formatos de credenciais, como JWT e SD-JWT, com o objetivo de avaliar suas vantagens e limitações em relação às evidenciadas pelo AnonCreds nesta implementação.

Além disso, trabalhos futuros podem realizar análises de desempenho, escalabilidade e segurança da arquitetura proposta. Nota-se também que a aplicação alcança seu objetivo de integridade e autenticidade desejada no ambiente controlado da prova de conceito. Porém, em um cenário de implementação real do proposto nesse trabalho, será necessário a realização de análises e verificações de segurança mais aprofundadas, não só com testes de penetração como também em estudos formais a respeito de ataques típicos realizados em *blockchains*, por exemplo.

Conclui-se que o trabalho contribui com um modelo arquitetural de código aberto e uma implementação prática que combina o controle de acesso baseado em SSI com a integridade proporcionada pela *blockchain* forense. Assim, estabelece fundamentos sólidos para o

desenvolvimento de sistemas descentralizados de cadeia de custódia digital, favorecendo soluções mais seguras, auditáveis e interoperáveis no contexto forense.

REFERÊNCIAS

- ALDOSARY, M.; ALQAHTANI, N. A survey on federated identity management systems limitation and solutions. **International Journal of Network Security & Its Applications (IJNSA)** Vol, v. 13, 2021.
- ANDROULAKI, E. et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In: **Proceedings of the thirteenth EuroSys conference**. [S.l.: s.n.], 2018. p. 1–15.
- ARAÚJO, M. Inteligência artificial, blockchain e a cadeia de custódia da prova no processo penal. **Revista da UFMG**, v. 30, 2023.
- AZZAM, F. et al. The use of blockchain technology and ocr in e-government for document management: Inbound invoice management as an example. **Applied Sciences**, MDPI, v. 13, n. 14, p. 8463, 2023.
- BAHGA, A.; MADISETTI, V. K. Blockchain platform for industrial internet of things. **Journal of Software Engineering and Applications**, Scientific Research Publishing, v. 9, n. 10, p. 533–546, 2016.
- BELCHIOR, R. et al. Ssibac: self-sovereign identity based access control. In: IEEE. **2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)**. [S.l.], 2020. p. 1935–1943.
- BRASIL. **Código de Processo Penal: Decreto-Lei n. 3.689, de 3 de outubro de 1941**. 1941. Online; acesso em 7 de junho de 2025. Disponível em: https://www.planalto.gov.br/ccivil_03/decreto-lei/del3689.htm.
- BRASIL. **Lei nº 13.964, de 24 de dezembro de 2019**. 2019. Conhecida como Pacote Anticrime. Altera o Código de Processo Penal e inclui o art. 158-A. Acesso em 7 de junho de 2025. Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2019-2022/2019/lei/L13964.htm.
- BUTERIN, V. et al. Ethereum white paper. **GitHub repository**, v. 1, n. 22-23, p. 5–7, 2013.
- CABALLERO, J. **Content Identifiers (CIDs) v1.0**. [S.l.], 2023. Seção 3.5: Multikey. Disponível em: <https://www.w3.org/TR/cid-1.0/>.
- CABALLERO, J. et al. **Verifiable Credentials Data Model v1.1**. [S.l.], 2021. Seção 5.6: Multibase. Disponível em: <https://www.w3.org/TR/vc-data-model/>.
- DIDComm Community. **DIDComm .org - Share protocols built on DIDComm**. 2025. <https://didcomm.org/>. Acessado em 21 out. 2025.
- FABRIC, H. **Fabric Model - Architecture Reference**. 2024. [Online; acesso em 23 de outubro de 2025]. Disponível em: https://hyperledger-fabric.readthedocs.io/en/latest/fabric_model.html.
- FETT, D.; YASUDA, K.; CAMPBELL, B. **Selective Disclosure for JWTs (SD-JWT)**. 2025. <https://datatracker.ietf.org/doc/draft-ietf-oauth-selective-disclosure-jwt/>. [Online; acesso em 5 de junho de 2025; Internet-Draft, IETF, Work in Progress].

FOUNDATION, T. L. **Overview | Hyperledger Identus Cloud Agent**. 2023. Online; Acesso em: 9 jun. 2025. Disponível em: <https://hyperledger-identus.github.io/docs/home/identus/cloud-agent/overview>.

FOUNDATION, T. L. **Glossary | Hyperledger Identus**. 2024. Online; Acesso em: 27 nov. 2025. Disponível em: <https://hyperledger-identus.github.io/docs/documentation/learn/glossary>.

FOUNDATION, T. L. **Secret Storage | Hyperledger Identus Cloud Agent**. 2024. Online; Acesso em: 27 nov. 2025. Disponível em: <https://hyperledger-identus.github.io/docs/cloud-agent/docs/docusaurus/secrets/secret-storage>.

FOUNDATION, T. L. **About - LF Decentralized Trust**. 2025. <https://www.lfdecentralizedtrust.org/about>. [Online; acesso em 18 de maio de 2025].

GOLOSOVA, J.; ROMANOV, A. The advantages and disadvantages of the blockchain technology. In: IEEE. **2018 IEEE 6th workshop on advances in information, electronic and electrical engineering (AIEEE)**. [S.l.], 2018. p. 1–6.

Google. **Identity**. 2025. <https://developers.google.com/identity>. Acessado em 17/10/2025.

HASAN, J. Overview and applications of zero knowledge proof (zkp). **International Journal of Computer Science and Network**, v. 8, n. 5, p. 2277–5420, 2019.

HEPP, T. et al. On-chain vs. off-chain storage for supply-and blockchain integration. **it-Information Technology**, De Gruyter Oldenbourg, v. 60, n. 5-6, p. 283–291, 2018.

HYPERLEDGER. **Hyperledger Fabric Samples - main branch**. 2024. <https://github.com/hyperledger/fabric-samples/tree/main>. [Online; acesso em 23 de outubro de 2025].

HYPERLEDGER. **Identus Cloud Agent**. 2024. <https://github.com/hyperledger-identus/cloud-agent>. [Online; acesso em 5 de julho de 2025].

HYPERLEDGER. **stub.go - shim package (commit 9fb89180dc17)**. 2024. <https://github.com/hyperledger/fabric-chaincode-go/blob/9fb89180dc17/shim/stub.go>. [Online; acesso em 23 de outubro de 2025].

HYPERLEDGER. **Fabric Gateway**. 2025. <https://hyperledger-fabric.readthedocs.io/en/latest/gateway.html>. [Online; acesso em 18 de maio de 2025].

HYPERLEDGER. **How Fabric networks are structured**. 2025. <https://hyperledger-fabric.readthedocs.io/en/latest/network/network.html>. [Online; acesso em 18 de maio de 2025].

HYPERLEDGER. **Smart Contracts and Chaincode**. 2025. <https://hyperledger-fabric.readthedocs.io/en/latest/smartcontract/smartcontract.html>. [Online; acesso em 18 de maio de 2025].

HYPERLEDGER. **Transaction Flow**. 2025. <https://hyperledger-fabric.readthedocs.io/en/latest/txflow.html>. [Online; acesso em 5 de julho de 2025].

Hyperledger Fabric. **shim package**. 2024. <https://pkg.go.dev/github.com/hyperledger/fabric-chaincode-go/shim>. Acessado em 23 out. 2025.

Hyperledger Identus. **Tutorials – Identus Platform**. 2025. <https://hyperledger-identus.github.io/docs/tutorials/>. Acessado em 21 out. 2025.

Hyperledger Identus. **Webhooks Tutorial – Identus Platform**. 2025. <https://hyperledger-identus.github.io/docs/tutorials/webhooks/webhook>. Acessado em 27 out. 2025.

IDENTUS, H. **Identus Cloud Agent API Reference**. 2025. <https://hyperledger-identus.github.io/docs/agent-api/>. Acessado em 21 out. 2025. Disponível em: <https://hyperledger-identus.github.io/docs/agent-api/>.

KIRUTHIKA, M.; PONNUSWAMY, P. P. Fusion of iot, blockchain and artificial intelligence for developing smart cities. In: **Blockchain, Internet of Things, and Artificial Intelligence**. [S.l.]: Chapman and Hall/CRC, 2021. p. 155–177.

LF Decentralized Trust. **Linux Foundation Decentralized Trust Launches with 17 Projects, 100+ Founding Members**. 2024. <https://www.lfdecentralizedtrust.org/announcements/linux-foundation-decentralized-trust-launches-with-17-projects-100-founding-members>. Acesso em 25 de agosto de 2025.

LOFFI, L. et al. Management of the chain of custody of digital evidence using blockchain and self-sovereign identities: A systematic literature review. **IEEE Access**, IEEE, 2025.

LOFFI, L. et al. Blockchain-based chain of custody for digital evidence with hyperledger fabric. In: IEEE. **2025 International Conference on Information Networking (ICOIN)**. [S.l.], 2025. p. 437–442.

LOSOPONKUL, P. **fix: update examples to ensure they work correctly**. 2025. <https://github.com/hyperledger-identus/cloud-agent/commit/b8f58c6337d56bd906c8abe8e5e2fe79ad0be45f>. Acessado em 27 out. 2025.

NAKAMOTO, S. Bitcoin whitepaper. **URL: https://bitcoin.org/bitcoin.pdf**-(: 17.07. 2019), v. 9, p. 15, 2008.

OU, H.-H.; CHEN, G.-Y.; LIN, I.-C. A self-sovereign identity blockchain framework for access control and transparency in financial institutions. **Cryptography (2410-387X)**, v. 9, n. 1, 2025.

PREUKSCHAT, A.; REED, D. **Self-Sovereign Identity: Decentralized Digital Identity and Verifiable Credentials**. 1. ed. Shelter Island, NY: Manning Publications Co., 2021. ISBN 9781617296598.

Python Software Foundation. **venv — Creation of virtual environments – Python 3.14.0 documentation**. 2025. <https://docs.python.org/3/library/venv.html>. Acessado em 28 out. 2025.

REED, D. et al. Decentralized identifiers (dids) v1. 0. **Draft Community Group Report**, W3C Cambridge, MA, USA, 2020.

SPORNY, M. et al. **Verifiable Credentials Data Model v2.0**. 2025. <https://www.w3.org/TR/vc-data-model-2.0/>. Online; acesso em 5 de junho de 2025;.

SUNARYA, P. A. et al. Blockchain family deed certificate for privacy and data security. In: IEEE. **2020 Fifth International Conference on Informatics and Computing (ICIC)**. [S.l.], 2020. p. 1–4.

The Hyperledger Foundation. **Hyperledger Fabric Documentation**. 2025. Official Documentation. Acessado em 23 out. 2025. Disponível em: <https://hyperledger-fabric.readthedocs.io/en/latest/>.

The Linux Foundation. **Identus**. 2025. Online; Acesso em 9 de junho de 2025. Disponível em: <https://www.lfdecentralizedtrust.org/projects/identus>.

The Linux Foundation. **AnonCreds Specification**. s.d. <https://hyperledger.github.io/anoncreds-spec/>. [Online; acesso em 5 de junho de 2025;].

The Linux Foundation. **PRISM Node Overview**. s.d. Online; Acesso em: 9 jun. 2025. Disponível em: <https://hyperledger-identus.github.io/docs/home/identus/prism-node/overview>.

W3C Credentials Community Group. **The did:key Method v0.7: Draft Community Group Report, 26 March 2025**. 2025. <https://w3c-ccg.github.io/did-key-spec/>. Acessado em: 25 de agosto de 2025.

WHITAKER, A. Art and blockchain: A primer, history, and taxonomy of blockchain use cases in the arts. **Artivate**, JSTOR, v. 8, n. 2, p. 21–46, 2019.

ZHENG, Z. et al. An overview of blockchain technology: Architecture, consensus, and future trends. In: IEEE. **2017 IEEE international congress on big data (BigData congress)**. [S.l.], 2017. p. 557–564.

APÊNDICE A - DEPENDÊNCIAS DA APLICAÇÃO TYPESCRIPT

```

1 {
2   "name": "asset-transfer-basic",
3   "version": "1.0.0",
4   "description": "Asset Transfer Basic Application implemented in TypeScript using
5     fabric-gateway",
6   "main": "dist/index.js",
7   "typings": "dist/index.d.ts",
8   "engines": {
9     "node": ">=18"
10  },
11  "scripts": {
12    "build": "tsc",
13    "build:watch": "tsc -w",
14    "lint": "eslint src",
15    "prepare": "npm run build",
16    "pretest": "npm run lint",
17    "start": "node dist/app.js"
18  },
19  "engineStrict": true,
20  "author": "Hyperledger",
21  "license": "Apache-2.0",
22  "dependencies": {
23    "@grpc/grpc-js": "^1.12.2",
24    "@hyperledger/fabric-gateway": "^1.7.0",
25    "@transmute/did-key-ed25519": "^0.3.0-unstable.10",
26    "axios": "^1.12.2",
27    "did-jwt": "^8.0.17",
28    "did-jwt-vc": "^4.0.15",
29    "did-resolver": "^4.1.0",
30    "express": "^5.1.0",
31    "key-did-resolver": "^4.0.0",
32    "mongodb": "^6.18.0",
33    "tslib": "^2.8.1"
34  },
35  "devDependencies": {
36    "@eslint/js": "^9.3.0",
37    "@tsconfig/node18": "^18.2.2",
38    "@types/body-parser": "^1.19.6",
39    "@types/connect": "^3.4.38",
40    "@types/express": "^5.0.3",
41    "@types/mime": "^3.0.4",
42    "@types/node": "^18.18.6",
43    "@types/serve-static": "^1.15.9",
44    "@types/webidl-conversions": "^7.0.3",
45    "@types/whatwg-url": "^13.0.0",
46    "eslint": "^8.57.0",
47    "typescript": "^5.4",
48    "typescript-eslint": "^7.13.0"
49  }

```

Figura 59 – Arquivo package.json da aplicação Typescript

APÊNDICE B - CÓDIGOS DE FUNÇÕES DO VC-HANDLER.TS

```

1  export async function createCredential(subject_data: CredentialSubjectData,
2                                     issuer: Issuer
3                                     ): Promise<string>
4  {
5      console.log("Gerando DID do emissor e criando a credencial...");
6      // ***** Dynamic importing for ESM modules *****
7      const { createVerifiableCredentialJwt } = await import('did-jwt-vc');
8
9      // VC claims
10     const vcPayload = {
11         '@context': ['https://www.w3.org/2018/credentials/v1'],
12         type: ['VerifiableCredential', 'Evidence'],
13         issuer: issuer.did,
14         issuanceDate: new Date().toISOString(),
15         credentialSubject: subject_data,
16     };
17
18     // Create and sign a VC using JWT format.
19     const vcJwt = await createVerifiableCredentialJwt(vcPayload, issuer);
20     //console.log(vcJwt);
21
22     return vcJwt;
23 }

```

Figura 60 – Função createCredential do arquivo vc-handler.ts

```

1  export async function verifyCreatedCredential(vcJwt: string): Promise<VerifiedCredential> {
2      // ***** Dynamic importing for ESM modules *****
3      const { verifyCredential } = await import('did-jwt-vc');
4      const { Resolver } = await import('did-resolver');
5      const { getResolver: getKeyResolver } = await import('key-did-resolver');
6
7      // Sets up the resolver
8      const keyResolver = getKeyResolver();
9      const resolver = new Resolver({
10         ...keyResolver
11     });
12
13     // Verifies Credential
14     try {
15         const verifiedVc = await verifyCredential(vcJwt, resolver);
16         console.log('\n Verificação concluída com sucesso!');
17         console.log('A credencial é autêntica. ');
18         return verifiedVc;
19     } catch (error: any) {
20         console.error('\n Falha ao verificar a credencial:', error.message);
21         throw error;
22     }
23 }

```

Figura 61 – Função verifyCreatedCredential do arquivo vc-handler.ts

APÊNDICE C - CÓDIGOS DE FUNÇÕES DO LOGIC.TS

```

1 export async function update_evidence(old_credential_did: string,
2   subject_data: CredentialSubjectData, issuer: Issuer,
3   dbHandler: DatabaseHandler, newRecordData: CustodyCredentialRecord,
4   contract: Contract): Promise<void>
5 {
6   const credential_record = await dbHandler.findRecordByCredentialId(old_credential_did);
7   if (credential_record?.sequence) {
8     // Calculates sequence number
9     newRecordData.sequence = credential_record.sequence + 1;
10  }
11
12  // Creates new credential.
13  const vcJwt = await createCredential(subject_data, issuer);
14  newRecordData.vcJwt = vcJwt;
15  newRecordData.evidencehash = subject_data.evidence_hash;
16  newRecordData.last_modifier_did = newRecordData.ownerDid;
17  newRecordData.previousCredentialId = old_credential_did;
18
19
20  await dbHandler.createRecord(newRecordData);
21
22  const credentialHash = createHash('sha256').update(vcJwt).digest('hex');
23  // Stores data of the new credential in blockchain.
24  await createAsset(contract, newRecordData.credentialId, newRecordData.ownerDid,
25    newRecordData.issuerDid, credentialHash, old_credential_did
26    );
27
28  // Change status of old credential in database
29  await dbHandler.updateRecordStatus(old_credential_did, "revoked");
30
31  // Change status of old credential in blockchain.
32  await revokeAsset(contract, old_credential_did);
33 }

```

Figura 62 – Função update_evidence do arquivo logic.ts

```

1  export async function get_chain_of_custody(credential_id: string, dbHandler:
2      DatabaseHandler, contract: Contract):
3      Promise<Array<{databaseRecord: CustodyCredentialRecord, ledgerData: Asset}>> {
4      // This list will store the combined data from the database and the ledger.
5      const chainOfCustody: Array<{databaseRecord: CustodyCredentialRecord, ledgerData: Asset}>
6          = [];
7
8      // Start with the initial credential ID provided.
9      let current_cred_id: string | null = credential_id;
10
11      do {
12          // Fetch the record from your local database.
13          const cred_record = await dbHandler.findRecordByCredentialId(current_cred_id) as
14              CustodyCredentialRecord;
15
16          // Fetch the corresponding asset data from the ledger
17          const ledger_data = await readAssetByID(contract, current_cred_id);
18
19          // Store the combined results as an object in our list.
20          chainOfCustody.push({
21              databaseRecord: cred_record,
22              ledgerData: ledger_data
23          });
24
25          // Check if a previous credential exists to continue the chain.
26          if (cred_record && cred_record.previousCredentialId) {
27              current_cred_id = cred_record.previousCredentialId;
28          } else if (cred_record) {
29              // If there's a record but no previous ID, the chain ends here.
30              current_cred_id = null;
31          }
32          else {
33              // If no record is found at all, the chain is broken.
34              throw new Error(`The chain of custody is broken. Could not find a record for
35                  credential ID: ${current_cred_id}`);
36          }
37      } while (current_cred_id);
38
39      // Return the complete list of records.
40      return chainOfCustody;
41  }

```

Figura 63 – Função get_chain_of_custody do arquivo logic.ts

```

1  export async function verify_chain_of_custody(
2    chain_of_custody: Array<{databaseRecord: CustodyCredentialRecord, ledgerData: Asset}>
3  ) : Promise<Array<VerifiedCredential>> {
4
5    let payloads: Array<VerifiedCredential> = [];
6    for (const [index, link] of chain_of_custody.entries()) {
7      try {
8        // Verify whether the credential hash matches the one stored in the ledger
9        const calculatedHash =
10          createHash('sha256').update(link.databaseRecord.vcJwt).digest('hex');
11
12        if (link.ledgerData.credential_id !== link.databaseRecord.credentialId) {
13          throw new Error(`Credentials id do not match at index ${index}.`);
14        }
15        if (calculatedHash !== link.ledgerData.credential_hash) {
16          // Throws a specific error if the hash does not match
17          throw new Error(`Hash mismatch for credential at index ${index}.`);
18        }
19        // If empty string, converts it to null so it can be compared with the database.
20        const ledgerPrevId = link.ledgerData.previous_credential_id === "" ? null :
21          link.ledgerData.previous_credential_id;
22        if (ledgerPrevId !== link.databaseRecord.previousCredentialId) {
23          throw new Error(`Previous credential link mismatch at index ${index}.`);
24        }
25        if (link.ledgerData.status !== link.databaseRecord.status) {
26          throw new Error(`Status mismatch at index ${index}.`);
27        }
28        if (link.ledgerData.credential_id !== link.databaseRecord.credentialId) {
29          throw new Error(`Credentials id do not match at index ${index}.`);
30        }
31        if (link.ledgerData.issuer_did !== link.databaseRecord.issuerDid) {
32          throw new Error(`Issuer DID mismatch at index ${index}.`);
33        }
34        // Verify the authenticity and integrity of the JWT credential
35        const verified_credential = await
36          verifyCreatedCredential(link.databaseRecord.vcJwt);
37        payloads.push(verified_credential);
38      } catch (error: any) {
39        // Capture verification errors either from hash either from credential.
40        console.error(`Failed to verify credential ${index} in the chain of custody:`,
41          error.message);
42        throw error;
43      }
44    }
45    return payloads;
46  }

```

Figura 64 – Função verify_chain_of_custody do arquivo logic.ts

```
1 export async function transfer_evidence_ownership(  
2   credential_id: string, new_owner_did: string,  
3   dbHandler: DatabaseHandler, contract: Contract): Promise<void> {  
4   // Change owner --> in blockchain data and db data.  
5   // Changes owner, but not the "last_modifier_did" field  
6   await dbHandler.updateRecordOwner(credential_id, new_owner_did);  
7  
8   // blockchain operation  
9   await transferOwnership(contract, credential_id, new_owner_did);  
10 }
```

Figura 65 – Função transfer_evidence_ownership do arquivo logic.ts

APÊNDICE D - DOCKER COMPOSE IDENTUS

```

1  configs:
2    caddyfile_holder:
3      content: |-
4        :8083 { # Changed from 8081 to 8083
5          handle_path /didcomm* {
6            reverse_proxy agent-holder:8090
7          }
8          handle_path /cloud-agent* {
9            reverse_proxy agent-holder:8085
10         }
11         handle_path /keycloak* {
12           reverse_proxy keycloak-holder:8080
13         }
14         handle_path /vault* {
15           reverse_proxy vault-holder:8200
16         }
17       }
18    caddyfile_issuer:
19      content: |-
20        :8080 {
21          handle_path /didcomm* {
22            reverse_proxy agent-issuer:8090
23          }
24          handle_path /cloud-agent* {
25            reverse_proxy agent-issuer:8085
26          }
27          handle_path /keycloak* {
28            reverse_proxy keycloak-issuer:8080
29          }
30          handle_path /vault* {
31            reverse_proxy vault-issuer:8200
32          }
33        }
34    caddyfile_verifier:
35      content: |-
36        :8082 {
37          handle_path /didcomm* {
38            reverse_proxy agent-verifier:8090
39          }
40          handle_path /cloud-agent* {
41            reverse_proxy agent-verifier:8085
42          }
43          handle_path /keycloak* {
44            reverse_proxy keycloak-verifier:8080
45          }
46          handle_path /vault* {
47            reverse_proxy vault-verifier:8200
48          }
49        }
50  services:
51    agent-holder:
52      depends_on:
53        node:

```

```

54     condition: service_started
55 environment:
56     ADMIN_TOKEN: admin
57     AGENT_DB_HOST: db-holder
58     AGENT_DB_NAME: agent
59     AGENT_DB_PASSWORD: postgres
60     AGENT_DB_PORT: '5432'
61     AGENT_DB_USER: postgres
62     API_KEY_ENABLED: 'false'
63     CONNECT_DB_HOST: db-holder
64     CONNECT_DB_NAME: connect
65     CONNECT_DB_PASSWORD: postgres
66     CONNECT_DB_PORT: '5432'
67     CONNECT_DB_USER: postgres
68     DIDCOMM_SERVICE_URL: http://caddy-holder:8083/didcomm # Changed from 8081 to 8083
69     POLLUX_DB_HOST: db-holder
70     POLLUX_DB_NAME: pollux
71     POLLUX_DB_PASSWORD: postgres
72     POLLUX_DB_PORT: '5432'
73     POLLUX_DB_USER: postgres
74     POLLUX_STATUS_LIST_REGISTRY_PUBLIC_URL: http://caddy-holder:8083/cloud-agent # Changed
    from 8081 to 8083
75     PRISM_NODE_HOST: node
76     PRISM_NODE_PORT: '50053'
77     REST_SERVICE_URL: http://caddy-holder:8083/cloud-agent # Changed from 8081 to 8083
78     SECRET_STORAGE_BACKEND: postgres
79     ENABLE_ANONCRED: true
80 image: hyperledgeridentus/identus-cloud-agent:2.0.0
81 restart: always
82 agent-issuer:
83     depends_on:
84         node:
85             condition: service_started
86 environment:
87     ADMIN_TOKEN: admin
88     AGENT_DB_HOST: db-issuer
89     AGENT_DB_NAME: agent
90     AGENT_DB_PASSWORD: postgres
91     AGENT_DB_PORT: '5432'
92     AGENT_DB_USER: postgres
93     API_KEY_ENABLED: 'false'
94     CONNECT_DB_HOST: db-issuer
95     CONNECT_DB_NAME: connect
96     CONNECT_DB_PASSWORD: postgres
97     CONNECT_DB_PORT: '5432'
98     CONNECT_DB_USER: postgres
99     DIDCOMM_SERVICE_URL: http://caddy-issuer:8080/didcomm
100    POLLUX_DB_HOST: db-issuer
101    POLLUX_DB_NAME: pollux
102    POLLUX_DB_PASSWORD: postgres
103    POLLUX_DB_PORT: '5432'
104    POLLUX_DB_USER: postgres
105    POLLUX_STATUS_LIST_REGISTRY_PUBLIC_URL: http://caddy-issuer:8080/cloud-agent
106    PRISM_NODE_HOST: node
107    PRISM_NODE_PORT: '50053'
108    REST_SERVICE_URL: http://caddy-issuer:8080/cloud-agent

```

```

109     SECRET_STORAGE_BACKEND: postgres
110     ENABLE_ANONCRED: true
111     GLOBAL_WEBHOOK_URL: http://host.docker.internal:5000/webhook
112     extra_hosts:
113     - "host.docker.internal:host-gateway"
114     image: hyperledgeridentus/identus-cloud-agent:2.0.0
115     restart: always
116 agent-verifier:
117     depends_on:
118     node:
119     condition: service_started
120     environment:
121     ADMIN_TOKEN: admin
122     AGENT_DB_HOST: db-verifier
123     AGENT_DB_NAME: agent
124     AGENT_DB_PASSWORD: postgres
125     AGENT_DB_PORT: '5432'
126     AGENT_DB_USER: postgres
127     API_KEY_ENABLED: 'false'
128     CONNECT_DB_HOST: db-verifier
129     CONNECT_DB_NAME: connect
130     CONNECT_DB_PASSWORD: postgres
131     CONNECT_DB_PORT: '5432'
132     CONNECT_DB_USER: postgres
133     DIDCOMM_SERVICE_URL: http://caddy-verifier:8082/didcomm
134     POLLUX_DB_HOST: db-verifier
135     POLLUX_DB_NAME: pollux
136     POLLUX_DB_PASSWORD: postgres
137     POLLUX_DB_PORT: '5432'
138     POLLUX_DB_USER: postgres
139     POLLUX_STATUS_LIST_REGISTRY_PUBLIC_URL: http://caddy-verifier:8082/cloud-agent
140     PRISM_NODE_HOST: node
141     PRISM_NODE_PORT: '50053'
142     REST_SERVICE_URL: http://caddy-verifier:8082/cloud-agent
143     SECRET_STORAGE_BACKEND: postgres
144     ENABLE_ANONCRED: true
145     image: hyperledgeridentus/identus-cloud-agent:2.0.0
146     restart: always
147 caddy-holder:
148     configs:
149     - source: caddyfile_holder
150       target: /etc/caddy/Caddyfile
151     image: caddy:2.7.6-alpine
152     ports:
153     - 8083:8083 # Changed from 8081:8081 to 8083:8083
154     restart: always
155 caddy-issuer:
156     configs:
157     - source: caddyfile_issuer
158       target: /etc/caddy/Caddyfile
159     image: caddy:2.7.6-alpine
160     ports:
161     - 8080:8080
162     restart: always
163 caddy-verifier:
164     configs:

```

```

165 - source: caddyfile_verifier
166   target: /etc/caddy/Caddyfile
167 image: caddy:2.7.6-alpine
168 ports:
169 - 8082:8082
170 restart: always
171 db-holder:
172   environment:
173     POSTGRES_MULTIPLE_DATABASES: pollux,connect,agent
174     POSTGRES_PASSWORD: postgres
175     POSTGRES_USER: postgres
176   healthcheck:
177     interval: 10s
178     retries: 5
179     test:
180     - CMD
181     - pg_isready
182     - -U
183     - postgres
184     - -d
185     - postgres
186     timeout: 5s
187 image: postgres:13
188 restart: always
189 volumes:
190 - pg_data_holder:/var/lib/postgresql/data
191 - ../shared/postgres/init-script.sh:/docker-entrypoint-initdb.d/init-script.sh
192 - ../shared/postgres/max_conns.sql:/docker-entrypoint-initdb.d/max_conns.sql
193 db-issuer:
194   environment:
195     POSTGRES_MULTIPLE_DATABASES: pollux,connect,agent
196     POSTGRES_PASSWORD: postgres
197     POSTGRES_USER: postgres
198   healthcheck:
199     interval: 10s
200     retries: 5
201     test:
202     - CMD
203     - pg_isready
204     - -U
205     - postgres
206     - -d
207     - postgres
208     timeout: 5s
209 image: postgres:13
210 restart: always
211 volumes:
212 - pg_data_issuer:/var/lib/postgresql/data
213 - ../shared/postgres/init-script.sh:/docker-entrypoint-initdb.d/init-script.sh
214 - ../shared/postgres/max_conns.sql:/docker-entrypoint-initdb.d/max_conns.sql
215 db-verifier:
216   environment:
217     POSTGRES_MULTIPLE_DATABASES: pollux,connect,agent
218     POSTGRES_PASSWORD: postgres
219     POSTGRES_USER: postgres
220   healthcheck:

```

```

221     interval: 10s
222     retries: 5
223     test:
224     - CMD
225     - pg_isready
226     - -U
227     - postgres
228     - -d
229     - postgres
230     timeout: 5s
231 image: postgres:13
232 restart: always
233 volumes:
234 - pg_data_verifier:/var/lib/postgresql/data
235 - ../.shared/postgres/init-script.sh:/docker-entrypoint-initdb.d/init-script.sh
236 - ../.shared/postgres/max_conns.sql:/docker-entrypoint-initdb.d/max_conns.sql
237 node:
238   depends_on:
239     node-db:
240       condition: service_healthy
241   environment:
242     NODE_PSQL_DATABASE: node_db
243     NODE_PSQL_HOST: node-db:5432
244     NODE_PSQL_PASSWORD: postgres
245     NODE_PSQL_USERNAME: postgres
246 image: ghcr.io/input-output-hk/prism-node:2.4.0
247 restart: always
248 node-db:
249   environment:
250     POSTGRES_MULTIPLE_DATABASES: node_db
251     POSTGRES_PASSWORD: postgres
252     POSTGRES_USER: postgres
253   healthcheck:
254     interval: 10s
255     retries: 5
256     test:
257     - CMD
258     - pg_isready
259     - -U
260     - postgres
261     - -d
262     - postgres
263     timeout: 5s
264 image: postgres:13
265 restart: always
266 volumes:
267 - pg_data_node:/var/lib/postgresql/data
268 - ../.shared/postgres/init-script.sh:/docker-entrypoint-initdb.d/init-script.sh
269 - ../.shared/postgres/max_conns.sql:/docker-entrypoint-initdb.d/max_conns.sql
270 volumes:
271 pg_data_holder: {}
272 pg_data_issuer: {}
273 pg_data_node: {}
274 pg_data_verifier: {}
275

```

Figura 66 – Arquivo `compose.yaml` que inicializa os agentes

APÊNDICE E - *PRESENTATION REQUEST* DA DOCUMENTAÇÃO E O DA IMPLEMENTAÇÃO

```

1  {
2      "connectionId": "872ddfa9-4115-46c2-8a1b-22c24c7431d7",
3      "anoncredPresentationRequest": {
4          "requested_attributes": {
5              "attribute1": {
6                  "name": "Attribute 1",
7                  "restrictions": [
8                      {
9                          "cred_def_id": "credential_definition_id_of_attribute1"
10                     }
11                 ],
12                 "non_revoked": {
13                     "from": 1635734400,
14                     "to": 1735734400
15                 }
16             }
17         },
18         "requested_predicates": {
19             "predicate1": {
20                 "name": "age",
21                 "p_type": ">=",
22                 "p_value": 18,
23                 "restrictions": [
24                     {
25                         "schema_id": "schema_id_of_predicate1"
26                     }
27                 ],
28                 "non_revoked": {
29                     "from": 1635734400
30                 }
31             }
32         },
33         "name": "Example Presentation Request",
34         "nonce": "1234567890",
35         "version": "1.0"
36     },
37     "credentialFormat": "AnonCreds"
38 }

```

Figura 67 – Exemplo *Presentation Request* AnonCreds da documentação oficial (Hyperledger Indentus, 2025a)

```

1  {
2    "connectionId": connectionId,
3    "credentialFormat": "AnonCreds",
4    "anoncredPresentationRequest": {
5      "name": "proof_of_expertise_and_authorization",
6      "version": "1.0",
7      "nonce": "1234567890123456789", # Should be random
8      "requested_attributes": {
9        "expert_name_proof": {
10          "name": "expert_name",
11          "restrictions": [{ # Url should be configurable
12            "cred_def_id": f"http://caddy-issuer:8080/cloud-agent/credential-definiti
              on-registry/definitions/{credential_definition_guid}/definition"
13          }]
14        },
15        "evidence_hash_proof": {
16          "name": "evidence_hash",
17          "restrictions": [{
18            "cred_def_id": f"http://caddy-issuer:8080/cloud-agent/credential-definiti
              on-registry/definitions/{credential_definition_guid}/definition"
19          }]
20        },
21        "subject_did_proof": {
22          "name": "subject_did",
23          "restrictions": [{
24            "cred_def_id": f"http://caddy-issuer:8080/cloud-agent/credential-definiti
              on-registry/definitions/{credential_definition_guid}/definition"
25          }]
26        }
27      },
28      "requested_predicates": {
29        "auth_level_proof": {
30          "name": "authorization_level",
31          "p_type": ">=",
32          "p_value": int(level_required),
33          "restrictions": [{
34            "cred_def_id": f"http://caddy-issuer:8080/cloud-agent/credential-definiti
              on-registry/definitions/{credential_definition_guid}/definition"
35          }]
36        }
37      }
38    },
39    "proofs": [],
40    "options": None
41  }

```

Figura 68 – *Presentation Request* AnonCreds utilizado na implementação

APÊNDICE F - *FORENSIC EVIDENCE SCHEMA*

```

1  {
2      "name": "anoncred-forensic-cert",
3      "version": "1.0.0",
4      "description": "A certificate for a piece of forensic evidence.",
5      "type": "AnoncredSchemaV1",
6      "author": "did:prism:example", # Must overwrite it!
7      "tags": [
8          "forensic",
9          "evidence",
10         "chain-of-custody"
11     ],
12     "schema": {
13         "$schema": "https://json-schema.org/draft/2020-12/schema",
14         "type": "object",
15         "properties": {
16             "name": {
17                 "type": "string",
18                 "minLength": 1
19             },
20             "version": {
21                 "type": "string",
22                 "minLength": 1
23             },
24             "attrNames": {
25                 "type": "array",
26                 "items": {
27                     "type": "string",
28                     "minLength": 1
29                 },
30                 "minItems": 1,
31                 "maxItems": 125,
32                 "uniqueItems": True
33             },
34             "issuerId": {
35                 "type": "string",
36                 "minLength": 1
37             },
38         },
39         "name": "Forensic Evidence Schema",
40         "version": "1.0",
41         "attrNames": [
42             "expert_name",
43             "issuing_judge_id",
44             "evidence_hash",
45             "authorization_level",
46             "court_jurisdiction",
47             "subject_did"
48         ],
49         "issuerId": "did:prism:example" # Must overwrite it!
50     },
51     "required": [
52         "name",
53         "version"
54     ],
55     "additionalProperties": True
56 }

```

Figura 69 – *Schema* utilizado na implementação

APÊNDICE G - *OUTPUTS* COMPLETOS DA EXECUÇÃO

Chain of Custody Metadata:

--- INÍCIO DA CADEIA DE CUSTÓDIA ---

----- Elo 1 -----

[+] Registro do Banco de Dados:

```
{
  "_id": "6900dccb68082c1aa359f892",
  "evidencehash": "4c0e0460dde9152e10c9099aaf22ddb7fe07726fe6cbbeab245d213d39321dbd",
  "credentialId": "did:key:z6MkmjDu9n3w76FCsEqZfsRUvSYU1nmnCETXNTtDmwk6dKhy",
  "vcJwt": "vcJWTPplaceholder",
  "previousCredentialId": "did:key:z6MkuYULpnQELRWUQxB6a5EgK2NLNKDxxFroi ygcoaozAE6j",
  "last_modifier_did":
    ↪ "did:prism:ea5331907b71bebdb7b38fa2bde8b5194741b9af86777cd1663bc416883e04cc",
  "sequence": 2,
  "status": "active",
  "issuerDid": "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx",
  "ownerDid": "did:prism:ea5331907b71bebdb7b38fa2bde8b5194741b9af86777cd1663bc416883e04cc",
  "createdAt": "2025-10-28T15:10:03.266Z",
  "tags": []
}
```

[+] Dados do Ledger (Blockchain):

```
{
  "status": "active",
  "timestamp": "2025-10-28T15:10:03.306Z",
  "owner_did": "did:prism:ea5331907b71bebdb7b38fa2bde8b5194741b9af86777cd1663bc416883e04cc",
  "issuer_did": "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx",
  "credential_id": "did:key:z6MkmjDu9n3w76FCsEqZfsRUvSYU1nmnCETXNTtDmwk6dKhy",
  "credential_hash": "eb5bb9521b94e35b19e00f9f1edfe6f368023d56bfd3f6dabb07d6dfbd1210ff",
  "last_modifier_did":
    ↪ "did:prism:ea5331907b71bebdb7b38fa2bde8b5194741b9af86777cd1663bc416883e04cc",
  "previous_credential_id": "did:key:z6MkuYULpnQELRWUQxB6a5EgK2NLNKDxxFroi ygcoaozAE6j"
}
```

----- Elo 2 -----

[+] Registro do Banco de Dados:

```
{
  "_id": "6900daad68082c1aa359f891",
  "evidencehash": "4c0e0460dde9152e10c9099aaf22ddb7fe07726fe6cbbeab245d213d39321dbd",
  "credentialId": "did:key:z6MkuYULpnQELRWUQxB6a5EgK2NLNKDxxFroi ygcoaozAE6j",
  "vcJwt": "vcJWTPplaceholder",
  "previousCredentialId": null,
  "last_modifier_did":
    ↪ "did:prism:ea5331907b71bebdb7b38fa2bde8b5194741b9af86777cd1663bc416883e04cc",
  "sequence": 1,
  "status": "revoked",
  "issuerDid": "did:key:z6Mkii73NPAsEvwDU7zd3zN73eGzkskhkfwcstrWpTrUoH1oP",
  "ownerDid": "did:prism:ea5331907b71bebdb7b38fa2bde8b5194741b9af86777cd1663bc416883e04cc",
  "createdAt": "2025-10-28T15:01:01.495Z",
  "tags": []
}
```

```

}

[+] Dados do Ledger (Blockchain):
{
  "status": "revoked",
  "timestamp": "2025-10-28T15:10:05Z",
  "owner_did": "did:prism:ea5331907b71bebdb7b38fa2bde8b5194741b9af86777cd1663bc416883e04cc",
  "issuer_did": "did:key:z6Mkii73NPAsEvwDU7zd3zN73eGzkshkfwcstrWpTrUoH1oP",
  "credential_id": "did:key:z6MkuYULpnQELRWUQxB6a5EgK2NLNKDxxFroi ygcoaozAE6j",
  "credential_hash": "f5a59e03790a28cb127363b2882b4f6ffd4b411a80cee906eb4a240a0fa5b96b",
  "last_modifier_did":
    ↪ "did:prism:ea5331907b71bebdb7b38fa2bde8b5194741b9af86777cd1663bc416883e04cc",
  "previous_credential_id": ""
}

----- FIM DA CADEIA -----

```

Figura 70 – Metadados da Cadeia de Custódia obtidos

```

Chain of Custody Payloads:
----- INÍCIO PAYLOADS -----
{
  "message": "Chain of custody verified successfully",
  "payloads": [
    {
      "verified": true,
      "payload": {
        "vc": {
          "@context": [
            "https://www.w3.org/2018/credentials/v1"
          ],
          "type": [
            "VerifiableCredential",
            "Evidence"
          ],
          "credentialSubject": {
            "evidence_hash":
              ↪ "4c0e0460dde9152e10c9099aaf22ddb7fe07726fe6cbbeab245d213d39321dbd",
            "previous_credential_id":
D ↪ "did:key:z6MkuYULpnQELRWUQxB6a5EgK2NLNKDxxFroi ygcoaozAE6j",
            "evidence_record": {
              "what": "In\u00edcio da an\u00e1lise pericial da imagem forense. O hash foi
              ↪ conferido e montado como 'read-only'.",
              "who": "Perito John Doe, ID 12345, da equipe de An\u00e1lise Forense",
              "where": "Esta\u00e7\u00e3o de trabalho forense 'FOR-WS-01', localizada no
              ↪ Laborat\u00f3rio de An\u00e1lise Forense da 'Dados S.A', Sala 10B",
              "when": "28 de outubro de 2025, 14:00 (hor\u00e1rio local)",
              "why": "Para extrair artefatos (logs do servidor web, scripts, arquivos de banco
              ↪ de dados) relacionados \u00e0 suspeita de 'SQL Injection', conforme
              ↪ solicita\u00e7\u00e3o do incidente",
              "how": "A imagem foi montada em modo 'read-only' utilizando a ferramenta
              ↪ 'Arsenal Image Mounter' e a an\u00e1lise foi iniciada com o software
              ↪ 'Autopsy 4.19'"
            }
          }
        }
      ]
    }
  ]
}

```

```

    }
  },
  "sub": "did:key:z6MkmjDu9n3w76FCsEqZfsRUvSYU1nmnCETXNTtDmwk6dKhy",
  "nbf": 1761664203,
  "iss": "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx"
},
"didResolutionResult": {
  "didResolutionMetadata": {
    "contentType": "application/did+json"
  },
  "didDocument": {
    "id": "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx",
    "verificationMethod": [
      {
        "id": "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx#z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx",
        "type": "Ed25519VerificationKey2018",
        "controller": "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx",
        "publicKeyBase58": "4CvjMxMdLEHkUPx4BggzBEbRHYgewum9Pp8FiBiWcGua"
      }
    ],
    "authentication": [
      "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx#z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx"
    ],
    "assertionMethod": [
      "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx#z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx"
    ],
    "capabilityDelegation": [
      "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx#z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx"
    ],
    "capabilityInvocation": [
      "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx#z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx"
    ],
    "keyAgreement": [
      {
        "id": "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx#z6LSkVxqKVeGcGD8o2nysNZLre2RNWFFDaqx1fKZDSh7ecK",
        "type": "X25519KeyAgreementKey2019",
        "controller": "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx",
        "publicKeyBase58": "9pnfoBqQWoVPheRDFDrc2GRYaDy8YcQh52we4koAQGqZ"
      }
    ]
  },
  "didDocumentMetadata": {}
},
"issuer": "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx",
"signer": {
  "id": "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx#z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx",
  "type": "Ed25519VerificationKey2018",
  "controller": "did:key:z6MkhfBmxCc4fmnDatnksFeq2L9R77xWMo1W5q3BYTgXXVgx",

```

```

{
  "publicKeyBase58": "4CvjMxMdLEHkUPx4BggzBebRHYgewum9Pp8FiBiWcGua",
  "jwt": "eyJhbGciOiJIJzERTQSIIsInR5cCI6IkpXVCJ9.eyJ2YyI6eyJAY29udGV4dCI6WyJodHRwc2ovL3d3dyIjE5M3My5vcmcvMjAxOC9jcmVhZmV5aWZ5cy92MSJdLCJ0eXB1IjpbI1ZlcmleWVibGVkdGVkZW50aWZ5IiwiaXZpZGVuY2UiXSwiY3JlZGVudGh1bFN1YmplY3QiOnsiZXZpZGVuY2VfaGFzaCI6IjRjMGUwNDYwZGRlOTE1MmUxMGM5MDk5YWZ5IiwiaXZpZGVuY2UwNmZlNmNiYmVhYjIIONWQyMTNkMzkzMjFkYmQiLCJwcmV2aW91c19jcmVhZmV5aWZ5X2lkIjoizG1kOmtleTp6Nk1rdVlVTHBuUUVMUldVUXhCNmE1RwIdLMk5MTktEeHhGcm9peWdjbj2FvekFFNmoiLCJldmklZW5jZV9yZWNVcmQiOnsid2hhdCI6IkludW61jaW8gZGEgYW7DoWxpc2UgcGVyaWNpYWwgZGEgaW1hZ2VtIGZvcml0byBkb2huIERvZSwgSUQgMTIzNDUsIGRhIGVxdWlwZSBkZSBBS0hbGlzZSBGb3JlbnN1Iiwid2hlcmUiOiJFc3Rhw6fDo28gZGUgdHJhYmFsaG8gZm9yZW5zZSAnRk9SLVdTLTAxJywgY29yZW50aWZ5IiwiaXZpZGVuY2UwNmZlNmNiYmVhYjIIONWQyMTNkMzkzMjFkYmQiLCJwcmV2aW91c19jcmVhZmV5aWZ5X2lkIjoizG1kOmtleTp6Nk1rdVlVTHBuUUVMUldVUXhCNmE1RwIdLMk5MTktEeHhGcm9peWdjbj2FvekFFNmoiLCJldmklZW5jZV9yZWNVcmQiOnsid2hhdCI6IkludW61jaW8gZGEgYW7DoWxpc2UgcGVyaWNpYWwgZGEgaW1hZ2VtIGZvcml0byBkb2huIERvZSwgSUQgMTIzNDUsIGRhIGVxdWlwZSBkZSBBS0hbGlzZSBGb3JlbnN1Iiwid2hlcmUiOiJFc3Rhw6fDo28gZGUgdHJhYmFsaG8gZm9yZW5zZSAnRk9SLVdTLTAxJywgY29yZW50aWZ5IiwiaXZpZGVuY2UwNmZlNmNiYmVhYjIIONWQyMTNkMzkzMjFkYmQiLCJwcmV2aW91c19jcmVhZmV5aWZ5X2lkIjoizG1kOmtleTp6Nk1rdVlVTHBuUUVMUldVUXhCNmE1RwIdLMk5MTktEeHhGcm9peWdjbj2FvekFFNmoiLCJldmklZW5jZV9yZWNVcmQiOnsid2hhdCI6IkludW61jaW8gZGEgYW7DoWxpc2UgcGVyaWNpYWwgZGEgaW1hZ2VtIGZvcml0byBkb2huIERvZSwgSUQgMTIzNDUsIGRhIGVxdWlwZSBkZSBBS0hbGlzZSBGb3JlbnN1Iiwid2hlcmUiOiJFc3Rhw6fDo28gZGUgdHJhYmFsaG8gZm9yZW5zZSAnRk9SLVdTLTAxJywgY29yZW50aWZ5IiwiaXZpZGVuY2UwNmZlNmNiYmVhYjIIONWQyMTNkMzkzMjFkYmQiLCJwcmV2aW91c19jcmVhZmV5aWZ5X2lkIjoizG1kOmtleTp6Nk1rdVlVTHBuUUVMUldVUXhCNmE1RwIdLMk5MTktEeHhGcm9peWdjbj2FvekFFNmoiLCJldmklZW5jZV9yZWNVcmQiOnsid2hhdCI6IkludW61jaW8gZGEgYW7DoWxpc2UgcGVyaWNpYWwgZGEgaW1hZ2VtIGZvcml0byBkb2huIERvZSwgSUQgMTIzNDUsIGRhIGVxdWlwZSBkZSBBS0hbGlzZSBGb3JlbnN1Iiwid2hlcmUiOiJFc3Rhw6fDo28gZGUgdHJhYmFsaG8gZm9yZW5zZSAnRk9SLVdTLTAxJywgY29yZW50aWZ5IiwiaXZpZGVuY2UwNmZlNmNiYmVhYjIIONWQyMTNkMzkzMjFkYmQiLCJwcmV2aW91c19jcmVhZmV5aWZ5X2lkIjoizG1kOmtleTp6Nk1rdVlVTHBuUUVMUldVUXhCNmE1RwIdLMk5MTktEeHhGcm9peWdjbj2FvekFFNmoiLCJldmklZW5jZV9yZWNVcmQiOnsid2hhdCI6IkludW61jaW8gZGEgYW7DoWxpc2UgcGVyaWNpYWwgZGEgaW1hZ2VtIGZvcml0byBkb2huIERvZSwgSUQgMTIzNDUsIGRhIGVxdWlwZSBkZSBBS0hbGlzZSBGb3JlbnN1Iiwid2hlcmUiOiJFc3Rhw6fDo28gZGUgdHJhYmFsaG8gZm9yZW5zZSAnRk9SLVdTLTAxJywgY29yZW50aWZ5IiwiaXZpZGVuY2UwNmZlNmNiYmVhYjIIONWQyMTNkMzkzMjFkYmQiLCJwcmV2aW91c19jcmVhZmV5aWZ5X2lkIjoizG1kOmtleTp6Nk1rdVlVTHBuUUVMUldVUXhCNmE1RwIdLMk5MTktEeHhGcm9peWdjbj2FvekFFNmoiLCJldmklZW5jZV9yZWNVcmQiOnsid2hhdCI6IkludW61jaW8gZGEgYW7DoWxpc2UgcGVyaWNpYWwgZGEgaW1hZ2VtIGZvcml0byBkb2huIERvZSwgSUQgMTIzNDUsIGRhIGVxdWlwZSBkZSBBS0hbGlzZSBGb3JlbnN1Iiwid2hlcmUiOiJFc3Rhw6fDo28gZGUgdHJhYmFsaG8gZm9yZW5zZSAnRk9SLVdTLTAxJywgY29yZW50aWZ5IiwiaXZpZGVuY2UwNmZlNmNiYmVhYjIIONWQyMTNkMzkzMjFkYmQiLCJwcmV2aW91c19jcmVhZmV5aWZ5X2lkIjoizG1kOmtleTp6Nk1rdVlVTHBuUUVMUldVUXhCNmE1RwIdLMk5MTktEeHhGcm9peWdjbj2FvekFFNmoiLCJldmklZW5jZV9yZWNVcmQiOnsid2hhdCI6IkludW61jaW8gZGEgYW7DoWxpc2UgcGVyaWNpYWwgZGEgaW1hZ2VtIGZvcml0byBkb2huIERvZSwgSUQgMTIzNDUsIGRhIGVxdWlwZSBkZSBBS0hbGlzZSBGb3JlbnN1Iiwid2hlcmUiOiJFc3Rhw6fDo28gZGUgdHJhYmFsaG8gZm9yZW5zZSAnRk9SLVdTLTAxJywgY29yZW50aWZ5IiwiaXZpZGVuY2UwNmZlNmNiYmVhYjIIONWQyMTNkMzkzMjFkYmQiLCJwcmV2aW91c19jcmVhZmV5aWZ5X2lkIjoizG1kOmtleTp6Nk1rdVlVTHBuUUVMUldVUXhCNmE1RwIdLMk5MTktEeHhGcm9peWdjbj2FvekFFNmoiLCJldmklZW5jZV9yZWNVcmQiOnsid2hhdCI6IkludW61jaW8gZGEgYW7DoWxpc2UgcGVyaWNpYWwgZGEgaW1hZ2VtIGZvcml0byBkb2huIERvZSwgSUQgMTIzNDUsIGRhIGVxdWlwZSBkZSBBS0hbGlzZSBGb3JlbnN1Iiwid2hlcmUiOiJFc3Rhw6fDo28gZGUgdHJhYmFsaG8gZm9yZW5zZSAnRk9SLVdTLTAxJywgY29yZW50aWZ5IiwiaXZpZGVuY2UwNmZlNmNiYmVhYjIIONWQyMTNkMzkzMjFkYmQiLCJwcmV2aW91c19jcmVhZmV5aWZ5X2lkIjoizG1kOmtleTp6Nk1rdVlVTHBuUUVMUldVUXhCNmE1RwIdLMk5MTktEeHhGcm9peWdjbj2FvekFFNmoiLCJldmklZW5jZV9yZWNVcmQiOnsid2hhdCI6IkludW61jaW8gZGEgYW7DoWxpc2UgcGVyaWNpYWwgZGEgaW1hZ2VtIGZvcml0byBkb2huIERvZSwgSUQgMTIzNDUsIGRhIGVxdWlwZSBkZSBBS0hbGlzZSBGb3JlbnN1Iiwid2hlcmUiOiJFc3Rhw6fDo28gZGUgdHJhYmFsaG8gZm9yZW5zZSAnRk9SLVdTLTAxJywgY29yZW50aWZ5IiwiaXZpZGVuY2UwNmZlNmNiYmVhYjIIONWQyMTNkMzkzMjFkYmQiLCJwcmV2aW91c19jcmVhZmV5aWZ5X2lkIjoizG1kOmtleTp6Nk1rdVlVTHBuUUVMUldVUXhCNmE1RwIdLMk5MTktEeHhGcm9peWdjbj2FvekFFNmoiLCJldmklZW5jZV9yZWNVcmQiOnsid2hhdCI6IkludW61jaW8gZGEgYW7DoWxpc2UgcGVyaWNpYWwgZGEgaW1hZ2VtIGZvcml0byBkb2huIERvZSwgSUQgMTIzNDUsIGRhIGVxdWlwZSBkZSBBS0hbGlzZSBGb3JlbnN1Iiwid2hlcmUiOiJFc3Rhw6fDo28gZGUgdHJhYmFsaG8gZm9yZW5zZSAnRk9SLVdTLTAxJywgY29yZW50aWZ5IiwiaXZpZGVuY2UwNmZlNmNiYmVhYjIIONWQyMTNkMzkzMjFkYmQiLCJwcmV2aW91c19jcmVhZmV5aWZ5X2lkIjoizG1kOmtleTp6Nk1rdVlVTHBuUUVMUldVUXhCNmE1RwIdLMk5MTktEeHhGcm9peWdjbj2FvekFFNmoiLCJldmklZW5jZV9yZWNVcmQiOnsid2hhdCI6IkludW61jaW8gZGEgYW7DoWxpc2UgcGVyaWNpYWwgZGEgaW1hZ2VtIGZvcml0byBkb2huIERvZSwgSUQgMTIzNDUsIGRhIGVxdWlwZSBkZSBBS0hbGlzZSBGb3JlbnN1Iiwid2hlcmUiOiJFc3Rhw6fDo28gZGUgdHJhYmFsaG8gZm9yZW5zZSAnRk9SLVdTLTAxJywgY29yZW50aWZ5IiwiaXZpZGVuY2UwNmZlNmNiYmVhYjIIONWQyMTNkMzkzMjFkYmQiLCJwcmV2aW91c19jcmVhZmV5aWZ5X2lkIjoizG1kOmtleTp6Nk1rdVlVTHBuUUVMUldVUXhCNmE1RwIdLMk5MTktEeHhGcm9peWdjbj2FvekFFNmoiLCJldmklZW5jZV9yZWNVcmQiOnsid2hhdCI6IkludW61jaW8gZGEgYW7DoWxpc2UgcGVyaWNpYWwgZGEgaW1hZ2VtIGZvcml0byBkb2huIERvZSwgSUQgMTIzNDUsIGRhIGVxdWlwZSBkZSBBS0hbGlzZSBGb3JlbnN1Iiwid2hlcmUiOiJFc3Rhw6fDo28gZGUgdHJhYmFsaG8gZm9yZW5zZSAnRk9SLVdTLTAxJywgY29yZW50aWZ5IiwiaXZpZGVuY2UwNmZlNmNiYmVhYjIIONWQyMTNkMzkzMjFkYmQiLCJwcmV2aW91c19jcmVhZmV5aWZ5X2lkIjoizG1kOmtleTp6Nk1rdVlVTHBuUUVMUldVUXhCNmE1RwIdLMk5MTktEeHhGcm9peWdjbj2FvekFFNmoiLCJldmklZW5jZV9yZWNVcmQiOnsid2hhdCI6IkludW61jaW8gZGEgYW7DoWxpc2UgcGVyaWNpYWwgZGEgaW1hZ2VtIGZvcml0byBkb2huIERvZSwgSUQgMTIzNDUsIGRhIGVxdWlwZSBkZSBBS0hbGlzZSBGb3JlbnN1Iiw
```

```

    }
  },
  {
    "verified": true,
    "payload": {
      "vc": {
        "@context": [
          "https://www.w3.org/2018/credentials/v1"
        ],
        "type": [
          "VerifiableCredential",
          "Evidence"
        ],
        "credentialSubject": {
          "evidence_hash":
            ↪ "4c0e0460dde9152e10c9099aaf22ddb7fe07726fe6cbbcab245d213d39321dbd",
          "previous_credential_id": null,
          "evidence_record": {
            "what": "Uma imagem forense bit a bit do disco principal (SSD de 1TB, modelo
            ↪ gen\u00e9rico) do servidor 'SRV-WEB-01'",
            "who": "Imagem criada remotamente por uma analista de Resposta a Incidentes, com
            ↪ certifica\u00e7\u00e3o XXX, da equipe interna de ciberseguran\u00e7a",
            "where": "O servidor f\u00edsico est\u00e1 localizado no rack 10,
            ↪ posi\u00e7\u00e3o 5 da empresa 'Dados S.A', na Rua X, 777, Bloco C,
            ↪ Florian\u00f3polis-SC",
            "when": "Aquisi\u00e7\u00e3o da imagem iniciada em 15 de fevereiro de 2025,
            ↪ \u00e0s 03:00 (hor\u00e1rio local) e conclu\u00edda \u00e0s 05:00 no mesmo dia.",
            "why": "Para apurar um incidente de seguran\u00e7a, provavelmente um ataque 'SQL
            ↪ Injection'",
            "how": "Foi utilizado uma ferramenta de c\u00f3pia forense via SSH a partir de
            ↪ uma esta\u00e7\u00e3o de trabalho de an\u00e1lise."
          }
        }
      }
    },
    "sub": "did:key:z6MkuYULpnQELRWUQxB6a5EgK2NLNKDxxFroiygcoaozAE6j",
    "nbf": 1761663661,
    "iss": "did:key:z6Mkii73NPAsEvwDU7zd3zN73eGzkshkfwcstrWpTrUoH1oP"
  },
  "didResolutionResult": {
    "didResolutionMetadata": {
      "contentType": "application/did+json"
    },
    "didDocument": {
      "id": "did:key:z6Mkii73NPAsEvwDU7zd3zN73eGzkshkfwcstrWpTrUoH1oP",
      "verificationMethod": [
        {
          "id": "did:key:z6Mkii73NPAsEvwDU7zd3zN73eGzkshkfwcstrWpTrUoH1oP#z6Mkii73NPAsEvw
          ↪ DU7zd3zN73eGzkshkfwcstrWpTrUoH1oP",
          "type": "Ed25519VerificationKey2018",
          "controller": "did:key:z6Mkii73NPAsEvwDU7zd3zN73eGzkshkfwcstrWpTrUoH1oP",
          "publicKeyBase58": "5Fqzn8vRuPSkMd9vNRQGCYizwJRuG4NXCqbtadawnMo21"
        }
      ]
    },
    "authentication": [
      "did:key:z6Mkii73NPAsEvwDU7zd3zN73eGzkshkfwcstrWpTrUoH1oP#z6Mkii73NPAsEvwDU7zd3zN
      ↪ 73eGzkshkfwcstrWpTrUoH1oP"
    ]
  }
}

```



```
]
}
----- FIM PAYLOADS -----
```

Figura 71 – *Payloads* obtidos após sucesso na verificação da cadeia de custódia

APÊNDICE F - PROTOCOLOS HYPERLEDGER IDENTUS

.1 CRIANDO UMA CONEXÃO

Dois agentes podem estabelecer entre si uma comunicação que utiliza o protocolo DIDComm (DIDComm Community, 2025).

Dois papéis existem para criar uma conexão: o convidador (*Inviter*) e o convidado (*Invitee*). Ao longo do processo, ambos terão seus próprios estados internos indicando em que fase do protocolo eles estão.

As figuras 72 e 73 mostram, respectivamente, a evolução dos estados dos agentes do *Inviter* e do *Invitee* ao longo do processo de criar uma conexão entre eles.

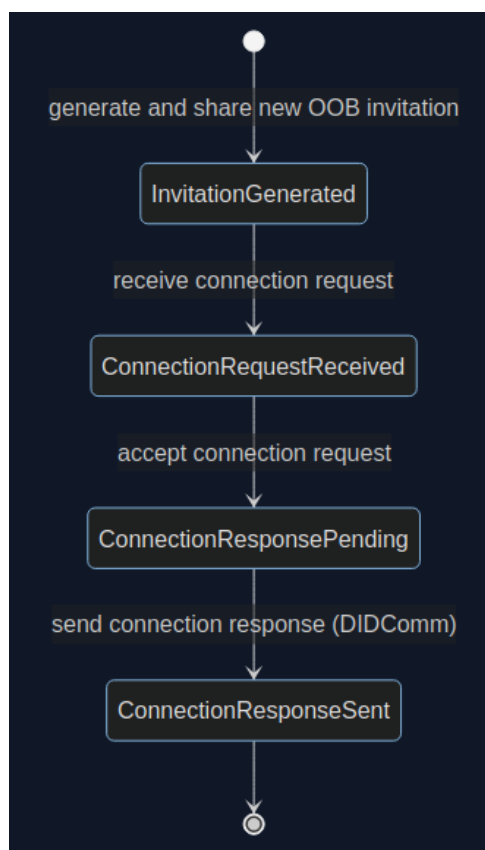


Figura 72 – Estados do *Inviter* durante estabelecimento de conexão (Hyperledger Identus, 2025a)

O convite gerado é do tipo fora de banda (OOB — *Out-of-Band*). Primeiro, o *Inviter* faz uma requisição (*request*) POST para seu agente no *endpoint* /connections. O Código 1 mostra uma resposta (*response*) exemplo ao gerar um convite. O campo *invitationUrl* deve ser transmitido ao *Invitee* de alguma forma — através de um QR code, por exemplo — pois ele será usado em um POST *request* ao *endpoint* /connection-invitations para aceitar o convite e estabelecer a conexão.

Código 1 – Exemplo de resposta ao criar o convite

```
{
```

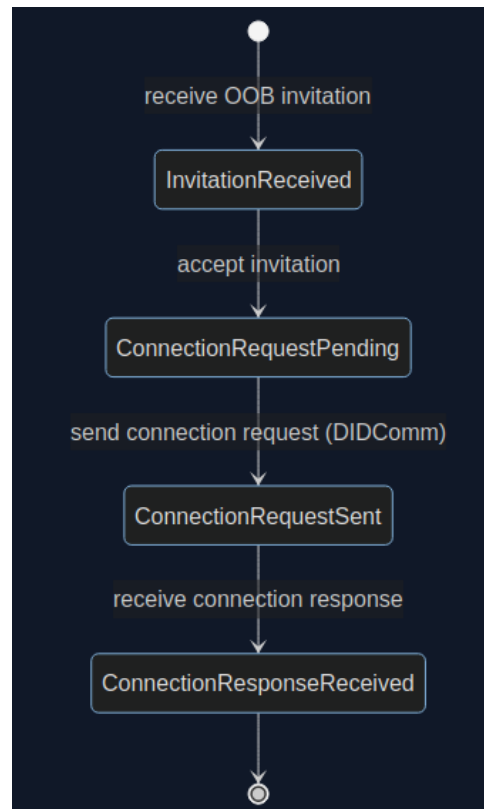


Figura 73 – Estados do *Invitee* durante estabelecimento de conexão (Hyperledger Indutus, 2025a)

```

"connectionId": "1893e207-4cba-4792-8179-067c78d2acc2",
"createdAt": "2023-02-02T17:14:13.593647Z",
"invitation": {
  "from": "did:peer:inviter-did",
  "id": "1893e207-4cba-4792-8179-067c78d2acc2",
  "invitationUrl": "https://domain.com/path?_oob={RAW_INVITATION}",
  "type": "https://didcomm.org/out-of-band/2.0/invitation"
},
"kind": "/connections/1893e207-4cba-4792-8179-067c78d2acc2",
"label": "Connect with Alice",
"self": "Connection",
"state": "InvitationGenerated"
}
  
```

.2 CRIANDO UM DID

Para um agente obter um PRISM DID, ele precisa fazer um POST *request* para o *endpoint* /did-registrar/dids. Ao fazer isso, um longFormDid é obtido, porém ele ainda não está registrado na *blockchain*. Para agendar a publicação dele no PRISM Node, é necessário

fazer uma requisição POST para `/did-registrar/dids/{didRef}/publications`, onde *{didRef}* deve ser substituído pelo `longFormDid` obtido anteriormente.

A resposta para a requisição irá conter a forma curta do DID que será publicado, como mostra o Código 2. Além disso, essa operação é assíncrona e demora um pouco até que o *status* do DID mude para PUBLISHED (Hyperledger Identities, 2025a).

Código 2 – Exemplo de DID criado (Hyperledger Identities, 2025a)

```
"didRef": "did:prism:4
a5b5cf0a513e83b598bbea25cd6196746747f361a73ef77068268bc9bd732ff"
```

.3 CRIANDO UM *CREDENTIAL SCHEMA*

Ele é análogo ao *schema* comentado na seção sobre AnonCreds. A Figura 74 mostra um exemplo de *schema* AnonCreds. Para criar a *Credential Schema*, deve-se realizar um POST request para o *endpoint* `/schema-registry/schemas` com um *schema* como parte do *payload*. A resposta conterá, dentre outras informações, um *id*, que o identifica localmente, além de um *guid*, que o identifica globalmente (Hyperledger Identities, 2025a).

```

{
  "name": "anoncred-birthday-cert",
  "version": "1.0.0",
  "description": "Birthday certificate",
  "type": "AnoncredSchemaV1",
  "author": "did:prism:e0266ee8d80a00163e5f922dc2567ab9611724a00db92423301154282169dff9",
  "tags": [
    "birth",
    "certificate"
  ],
  "schema": {
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "type": "object",
    "properties": {
      "name": {
        "type": "string",
        "minLength": 1
      },
      "version": {
        "type": "string",
        "minLength": 1
      },
      "attrNames": {
        "type": "array",
        "items": {
          "type": "string",
          "minLength": 1
        },
        "minItems": 1,
        "maxItems": 125,
        "uniqueItems": true
      },
      "issuerId": {
        "type": "string",
        "minLength": 1
      }
    },
    "name": "Birth Certificate Schema",
    "version": "1.0",
    "attrNames": [
      "location",
      "birthday"
    ],
    "issuerId": "did:prism:e0266ee8d80a00163e5f922dc2567ab9611724a00db92423301154282169dff9"
  },
  "required": [
    "name",
    "version"
  ],
  "additionalProperties": true
}

```

Figura 74 – Exemplo de *schema* AnonCreds (Hyperledger Indentus, 2025a)

.4 CRIANDO UMA CREDENTIAL DEFINITION

Este passo é exclusivo para a criação de uma credencial AnonCreds.

O *Issuer* deve fazer a requisição ao *endpoint* `/credential-definition-registry/definitions`, com um *payload* semelhante ao mostrado no Código 3. A resposta, similarmente à criação do *schema*, terá, além de outros dados, um *id* e um *guid* (Hyperledger Identus, 2025a).

Código 3 – Exemplo de *payload* para criar um *Credential Definition*

```
{
  "name": "Birth Certificate location",
  "description": "Birth certificate Anoncred Credential
    Definition",
  "version": "1.0.0",
  "tag": "Licence",
  "author": "{{ISSUER_DID_SHORT}}",
  "schemaId": "http://host.docker.internal:8080/cloud-agent/
    schema-registry/schemas/{{SCHEMA_ID}}",
  "signatureType": "CL",
  "supportRevocation": true
}
```

.5 CRIANDO UMA CREDENCIAL

Existe mais de um protocolo para assinar uma credencial. O que será abordado a seguir supõe a existência de uma conexão DIDComm entre *Holder* e *Issuer*. Importante ressaltar que os *payloads* esperados variam conforme o tipo da credencial que será assinada (Hyperledger Identus, 2025a).

As Figuras 75 e 75 mostram, respectivamente, os estados que os agentes do *Issuer* e do *Holder* transitarão durante o protocolo.

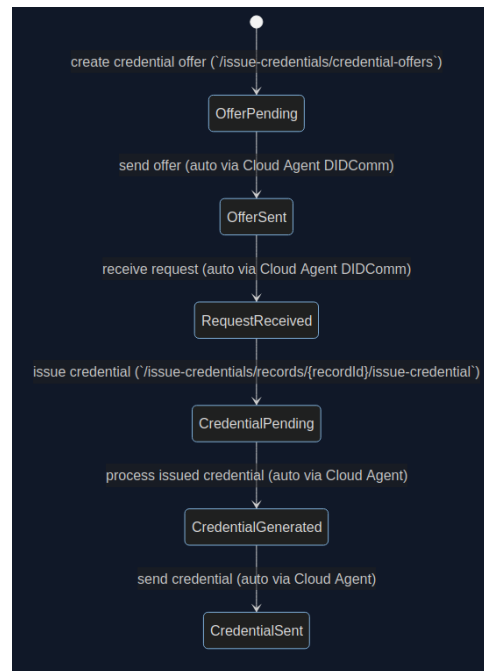


Figura 75 – Estados do *Issuer* durante a assinatura de uma credencial (Hyperledger Identus, 2025a)

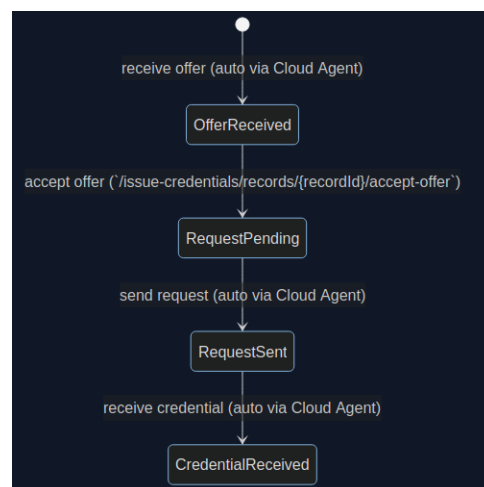


Figura 76 – Estados do *Holder* durante a assinatura de uma credencial (Hyperledger Identus, 2025a)

Primeiramente, o *Issuer* define os atributos que ele deseja que a credencial tenha e realiza um POST *request* para `/issue-credentials/credential-offers`, com *payload* similar ao mostrado no Código 4, criando uma oferta de credencial e automaticamente enviando-a ao *Holder* através do DIDComm (Hyperledger Identus, 2025a).

Código 4 – Exemplo de *payload* para criar um *Credential Definition*

```

{
  "connectionId": "9d075518-f97e-4f11-9d10-d7348a7a0fda",
  "credentialFormat": "AnonCreds",
  "anoncredsVcPropertiesV1": {
    "claims": {

```

```

        "emailAddress": "alice@wonderland.com",
        "givenName": "Alice",
        "familyName": "Wonderland",
        "dateOfIssuance": "2020-11-13T20:20:39+00:00",
        "drivingLicenseID": "12345",
        "drivingClass": "3"
    },
    "issuingDID": "did:prism:example",
    "credentialDefinitionId": "5d737816-8fe8-3492-bfe3-1
        b3e2b67220b"
}

```

Ao receber a oferta, o *Holder* pode aceitá-la através de uma requisição POST para `/issue-credentials/records/{recordId}/accept-offer`. Sendo que `recordId` é o identificador local dessa *Credential Offer*.

O último passo seria o *Issuer* assinar a credencial manualmente através do *endpoint* `issue-credentials/records/{recordId}/issue-credential` com o método POST. Porém, este passo é realizado automaticamente, a menos que o atributo `automaticIssuance = false` tenha sido adicionado no início do protocolo de assinatura de credencial (IDENTUS, 2025).

.6 APRESENTANDO UMA PROVA

É necessário que o *Verifier* e o provador (*Prover*) tenham estabelecido previamente uma conexão DIDComm entre si.

As Figuras 77 e 78 mostram os estados que os agentes do *Verifier* e do *Prover*, respectivamente, transitarão durante o protocolo.

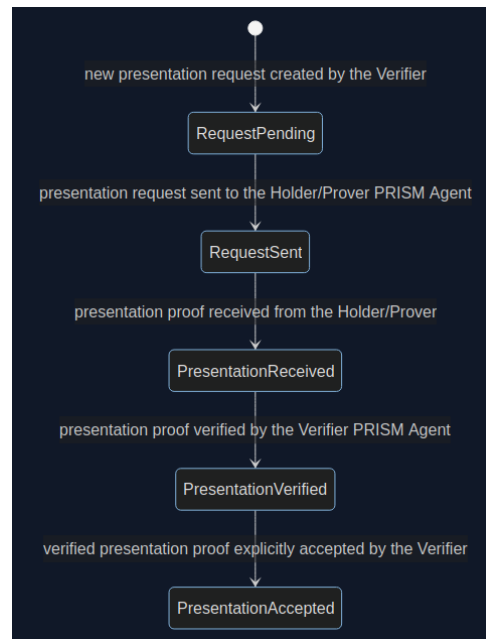


Figura 77 – Estados do *Verifier* durante a apresentação de uma prova (Hyperledger Identus, 2025a)

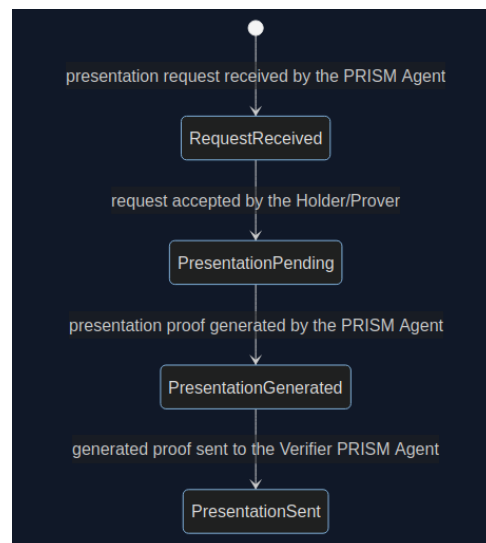


Figura 78 – Estados do *Prover* durante a apresentação de uma prova (Hyperledger Identus, 2025a)

Primeiramente, o *Verifier* cria um *Presentation Request*, que contém informações sobre quais atributos são exigidos e, no caso do AnonCreds, os predicados que serão verificados. O *Verifier* faz um POST *request* para `/present-proof/presentations` com um *payload* nos moldes do que é ilustrado na Figura 79.

Essa requisição é enviada ao *Prover*, que monta sua apresentação baseada nela e então pode enviar o método PATCH para o *endpoint* `/present-proof/presentations/{id}`, com *payload* similar ao ilustrado na Figura 80. Nota-se que o *id* é referente ao identificador local do *Presentation Request* recebido (Hyperledger Identus, 2025a).

```
{
  "connectionId": "872ddfa9-4115-46c2-8a1b-22c24c7431d7",
  "anoncredPresentationRequest": {
    "requested_attributes": {
      "attribute1": {
        "name": "Attribute 1",
        "restrictions": [
          {
            "cred_def_id": "credential_definition_id_of_attribute1"
          }
        ],
        "non_revoked": {
          "from": 1635734400,
          "to": 1735734400
        }
      }
    },
    "requested_predicates": {
      "predicate1": {
        "name": "age",
        "p_type": ">=",
        "p_value": 18,
        "restrictions": [
          {
            "schema_id": "schema_id_of_predicate1"
          }
        ],
        "non_revoked": {
          "from": 1635734400
        }
      }
    },
    "name": "Example Presentation Request",
    "nonce": "1234567890",
    "version": "1.0"
  },
  "credentialFormat": "AnonCreds"
}
```

Figura 79 – Exemplo de *payload* para criar um *Presentation Request* (Hyperledger Indentus, 2025a)

```
{
  "action": "request-accept",
  "anoncredPresentationRequest": {
    "credentialProofs": [
      {
        "credential": "3e849b98-f0fd-4cb4-ae96-9ea527a76267",
        "requestedAttribute": [
          "age"
        ],
        "requestedPredicate": [
          "age"
        ]
      }
    ]
  }
}
```

Figura 80 – Exemplo de *payload* para criar um *Presentation* (Hyperledger Identus, 2025a)

Por fim, o *Verifier* deve aceitar a apresentação por meio do método PATCH no *endpoint* `/present-proof/presentations/{id}` (Hyperledger Identus, 2025a). Ressalta-se que o fluxo mostrado representa o caso em que não ocorrem erros e os dados são válidos e autênticos. Caso contrário, mensagens de erro seriam retornadas em algum ponto da execução.

APÊNDICE G - CÓDIGO FONTE

O código fonte da implementação está disponível em <https://github.com/Luan-Diniz/ChainOfCustody> e também em <https://codigos.ufsc.br/luan.diniz.moraes/chainofcustody>.

Para sua execução, basta seguir as instruções escritas no arquivo `README.md`.

APÊNDICE H - ARTIGO SÍNTESE DO TCC

Cadeia de Custódia Forense Usando Hyperledger Fabric e Controle de Acesso via Hyperledger Identus

Luan Diniz Moraes¹

¹ Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)
Florianópolis, SC - Brasil

Abstract. *Digital evidence requires rigorous preservation of integrity and authenticity. Technologies such as Self-Sovereign Identity (SSI) and blockchain offer promising characteristics in this context; however, their integration applied to the chain of custody remains underexplored in the literature. This work presents an application developed using Hyperledger Fabric and Hyperledger Identus, utilizing SSI and zero-knowledge proofs (ZKP) to control forensic expert access. A proof of concept was built in which the chain of custody is modeled as a chain of verifiable credentials, storing metadata on-chain and sensitive data off-chain. The system implemented granular access control via AnonCreds and ZKP, validating permissions without unnecessarily exposing expert data.*

Resumo. *As evidências digitais exigem rigorosa preservação de integridade e autenticidade. Tecnologias como Identidade Autossobrerana (SSI) e blockchain oferecem características promissoras nesse contexto, mas sua integração aplicada à cadeia de custódia permanece subexplorada pela literatura. Este trabalho apresenta uma aplicação desenvolvida com Hyperledger Fabric e Hyperledger Identus, utilizando SSI e provas de conhecimento zero (ZKP) para controlar o acesso de peritos. Foi construída uma prova de conceito onde a cadeia de custódia é modelada como um encadeamento de credenciais verificáveis, com metadados on-chain e dados sensíveis off-chain. O sistema implementou controle de acesso granular via AnonCreds e ZKP, validando permissões sem expor desnecessariamente os dados dos peritos.*

1. Introdução

A manipulação de dados digitais cresce de forma exponencial [Loffi et al. 2025a], trazendo à tona a fragilidade e volatilidade das evidências digitais. Para garantir a validade jurídica desses vestígios, a preservação da integridade e autenticidade da cadeia de custódia é essencial. Nesse cenário, a tecnologia *blockchain* tem sido amplamente adotada como solução para o registro imutável e seguro de evidências [Araújo 2023, Loffi et al. 2025a].

Contudo, a robustez do armazenamento deve ser acompanhada por um controle de acesso igualmente seguro. A arquitetura atual da internet, carente de uma camada nativa de identidade, historicamente dependeu de modelos centralizados ou federados. Tais abordagens apresentam vulnerabilidades significativas, como a criação de silos de dados suscetíveis a vazamentos e riscos à privacidade [Preukschat and Reed 2021, Aldosary and Alqahtani 2021, Google 2025]. Como alternativa, a Identidade Autossobrerana (SSI) emerge como um paradigma descentralizado, devolvendo ao usuário o controle de seus dados através de Identificadores Descentralizados e Credenciais Verificáveis.

A integração entre os temas cadeia de custódia, SSI e *blockchain* é uma área subexplorada pela literatura [Loffi et al. 2025a, Belchior et al. 2020, Ou et al. 2025]. Este trabalho visa preencher essa lacuna investigando como a SSI pode fortalecer o modelo de cadeia de custódia, indo além do registro em *ledger* para oferecer um controle de acesso granular e privado.

O objetivo principal deste artigo é apresentar o desenvolvimento de uma prova de conceito que integra a cadeia de custódia forense com os conceitos de SSI. A solução proposta utiliza o *Hyperledger Fabric* para garantir a imutabilidade dos registros e o *Hyperledger Identus* para o gerenciamento de identidade. Um diferencial da abordagem é o uso de provas de conhecimento zero (ZKPs) e credenciais *AnonCreds*, permitindo validar o nível de acesso dos peritos sem expor desnecessariamente informações sensíveis.

2. Fundamentação Teórica

2.1. Cadeia de Custódia

A cadeia de custódia é definida pelo Art. 158-A do Código de Processo Penal como o conjunto de procedimentos utilizados para manter e documentar a história cronológica do vestígio, desde o reconhecimento até o descarte [Brasil 1941, Brasil 2019]. O objetivo primordial é assegurar a integridade e a autenticidade da evidência, sendo que qualquer quebra nesse elo pode comprometer o valor probatório e a admissibilidade jurídica da prova [Araújo 2023]. Essa documentação é realizada através de uma série de perguntas conhecidas como 5WH [Loffi et al. 2025a]: Quem (*Who*), O que (*What*), Quando (*When*), Onde (*Where*), Por que (*Why*) e Como (*How*).

2.2. Blockchain

Uma *blockchain* pode ser definida como uma "estrutura de dados distribuída composta por uma cadeia de blocos. A *blockchain* atua como um banco de dados distribuído ou um livro-razão global que mantém registros de todas as transações em uma rede *blockchain*" [Bahga and Madisetti 2016].

2.3. Identidade Autossoberana

A Identidade Autossoberana (SSI – *Self Sovereign Identity*) apresenta-se como um novo modelo para a identidade digital, respondendo à necessidade fundamental de como indivíduos provam quem são para estabelecer confiança no ambiente online. Ela dá ao usuário o controle sobre seus dados identitários, de forma análoga à gestão da identidade no mundo físico, através de uma infraestrutura de confiança descentralizada [Preukschat and Reed 2021].

Para viabilizar essa arquitetura de confiança descentralizada, o modelo SSI fundamenta-se em três pilares tecnológicos principais: o Registro de Dados Verificável, os Identificadores Descentralizados e as Credenciais Verificáveis.

2.3.1. Registro de Dados Verificável

Também conhecido pela sigla VDR (*Verifiable Data Registry*), Um Registro de Dados Verificável deve ser acessível pela internet e possui todos os dados e metadados necessários para que um ecossistema de credenciais verificáveis funcione, como por exemplo chaves públicas de emissores e listas de revocações [Preukschat and Reed 2021].

2.3.2. Identificadores Descentralizados

Os Identificadores Descentralizados (DIDs – *Decentralized Identifiers*) representam um novo tipo de identificador global único, voltado para identidades digitais na internet. Foram projetados para tirar proveito de redes descentralizadas [Preukschat and Reed 2021].

2.3.3. Credenciais Verificáveis

Conhecidas também pela sigla VCs (*Verifiable Credentials*), as Credenciais Verificáveis permitem que uma parte confiável (*Issuer*) faça declarações sobre um sujeito (geralmente o próprio detentor da credencial, chamado de *Holder*). Essas declarações, juntamente com metadados, são organizadas em uma credencial digital. Para torná-la verificável, o *Issuer* assina criptograficamente a credencial, criando uma prova (assinatura digital) que confirma sua origem e integridade [Preukschat and Reed 2021].

Um exemplo de VC é o AnonCreds, que possui suporte à provas de conhecimento zero [The Linux Foundation sda].

2.4. Hyperledger Fabric

O Hyperledger Fabric é uma plataforma de *blockchain* permissionada e privada, a plataforma foi projetada para oferecer suporte à implementação de aplicações distribuídas, especialmente voltado a casos de uso corporativos. Os nós da rede devem ser conhecidos e confiáveis [Androulaki et al. 2018].

Possui o conceito de *chaincode*, que é análogo ao de contrato inteligente, ambos os termos sendo utilizados, por muitas vezes, de forma intercambiável na própria documentação do Hyperledger Fabric [Hyperledger 2025b]. Para a escrita de aplicações clientes, é utilizado o Fabric Gateway, que é um serviço que provê uma API (*Application Programming Interface*) para que transações sejam submetidas à rede do Fabric [Hyperledger 2025a].

2.5. Hyperledger Identus

O Hyperledger Identus é um projeto que fornece um conjunto de ferramentas e componentes para a construção de soluções baseadas em identidade autossobranana [The Linux Foundation 2025].

Um dos principais elementos do projeto é o *Identus Cloud Agent*, um agente que atua como uma camada de abstração para facilitar a implementação dessas aplicações descentralizadas.

O Identus utiliza a *blockchain* Cardano, através do PRISM Node. Segundo a documentação oficial, “o PRISM Node atua como um nó de segunda camada para o *ledger* distribuído. Ele funciona como um registro de dados verificáveis, fornecendo uma maneira segura e confiável de armazenar e gerenciar dados” [The Linux Foundation sdb].

Existem três termos que serão frequentemente utilizados neste trabalho. O primeiro deles é o *agent* (agente), que é “um serviço que provê APIs para interagir com a plataforma Identus e utilizar os recursos SSI”[Foundation 2024b]. O segundo termo é

wallet (carteira), definido como sendo o "componente lógico do agente que armazena segredos e fornece o isolamento lógico ou físico dos dados"[Foundation 2024a]. Nela, são armazenados dados como DIDs, conexões, chaves, credenciais verificáveis e outros. Por fim, o *controller* (controlador), é a parte da implementação de uma aplicação que irá se comunicar e controlar, via requisições, o agente.

3. Estado da Arte

A aplicação de tecnologias descentralizadas para a gestão de documentos e identidades tem sido amplamente discutida na literatura recente. Diversos trabalhos [Sunarya et al. 2020, Kiruthika and Ponnuswamy 2021, Azzam et al. 2023] apontam o uso de *blockchain* como uma solução eficaz em contextos governamentais e de registros civis, mitigando problemas de falsificação, burocracia e centralização, ao mesmo tempo em que asseguram integridade e rastreabilidade.

Especificamente no âmbito forense, a integridade da cadeia de custódia é crítica. Araújo [Araújo 2023] demonstra a compatibilidade jurídica do uso de *blockchain* no processo penal brasileiro, destacando a criação de registros imutáveis contra manipulações. Em uma análise mais ampla, Loffi et al. [Loffi et al. 2025a] realizaram uma revisão sistemática que confirma a crescente adoção de *blockchain* para a preservação de evidências, mas aponta lacunas críticas, como a escassez de implementações práticas e a ausência de técnicas de privacidade como ZKPs. O estudo conclui que a integração de Identidade Autossobrerana nesse cenário ainda é incipiente. Já no campo prático, Loffi et al. [Loffi et al. 2025b] apresentaram uma implementação utilizando *Hyperledger Fabric*, estruturando a cadeia de custódia para responder às questões essenciais (5WH) e garantir a precisão cronológica dos registros.

Por fim, a literatura sobre gestão de identidade apresenta a SSI como o estado da arte para modelos descentralizados [Preukschat and Reed 2021]. No que tange ao controle de acesso, Ou et al. [Ou et al. 2025] demonstram a aplicação prática dessa tecnologia para o gerenciamento de permissões e privacidade em instituições financeiras. A presente proposta se apropria desses conceitos para suprir a lacuna de integração entre SSI e cadeia de custódia de evidências digitais.

4. Implementação

O sistema desenvolvido conta com dois papéis: o Perito e o Emissor. O Perito deseja documentar algo na cadeia de custódia, mas para isso ele primeiro deve obter acesso a ela. Esse acesso é concedido pelo Emissor, que assina uma credencial detalhando o nível de permissões do Perito ao utilizar o sistema e envia a ele através de uma conexão estabelecida entre os dois. Essa sequência de passos se baseia no Hyperledger Identus.

Após obter a credencial, o Perito pode se comunicar com o gerenciador de cadeia de custódia e tentar realizar diversas ações, como registrar ou atualizar uma nova evidência. Sempre que ele fizer a requisição para executar alguma ação, a sua credencial é verificada e suas permissões, validadas. No caso de sucesso, a execução é feita, caso contrário, algum erro ocorrerá.

Dentro desse gerenciador, existem outros tipos de credenciais. Elas são criadas fora da *blockchain*, encadeadas e representam uma cadeia de custódia em si. Essa parte da

implementação é baseada no Hyperledger Fabric, pois ele armazena no *ledger* metadados importantes para a garantia da autenticidade e integridade da cadeia de custódia.

A princípio foi avaliado se isso seria possível implementar utilizando somente o *Hyperledger Identus*. Porém, no *Identus*, todas as credenciais são armazenadas diretamente nas *wallets* dos agentes. A solução imaginada da cadeia de custódia ser um encaideamento de credenciais precisaria importar e exportar credenciais dessas *wallets*. Após o estudo da API do agente em nuvem [Identus 2025], concluiu-se a inviabilidade dessa ideia, já que essas funcionalidades não estão presentes no *Identus*.

Portanto, adotou-se o projeto *Hyperledger Fabric* para desenvolver essa aplicação, visto sua modularidade e extensibilidade [Androulaki et al. 2018], o que permitiu que a solução pensada fosse implementada de fato.

Posteriormente, pensou-se em como seria feito o controle de acesso a essa implementação, e optou-se por baseá-lo em SSI. Nesse cenário, o uso do *Identus* é muito favorável, pois ele já implementa as operações criptográficas necessárias para criar esse controle de acesso, sendo necessário apenas desenvolver controladores para os agentes em nuvem, além de possuir suporte a *Anoncreds*, o que possibilita o uso de provas de conhecimento zero.

4.1. Casos de Uso

No sistema desenvolvido, existem dois atores: o Perito (*Holder*) e o Emissor (*Issuer*). A Figura 1 é um diagrama de caso de uso, evidenciando de forma gráfica quais são os casos de uso para ambos na aplicação desenvolvida.

4.2. Diagrama de sequência

Nem toda ordem de execução das ações descritas pelos casos de uso resulta em sucesso. A Figura 2 é um diagrama de sequência que explicita a ordem correta das execuções para que o fluxo aconteça sem erros. Nota-se que as operações do perito na cadeia de custódia só terão sucesso após o recebimento da credencial.

4.3. Visão Geral da Arquitetura

A implementação feita pode ser dividida em duas grandes partes. A primeira é um sistema de gerenciamento de cadeia de custódia, criado utilizando o Hyperledger Fabric. A segunda parte é um sistema SSI de controle de acesso construído com o Hyperledger Identus que permite peritos obterem uma credencial verificável que funciona como permissão para que eles manipulem a cadeia de custódia.

A Figura 3 mostra a visão geral da arquitetura da aplicação de forma simplificada. A comunicação entre o Controle de Acesso SSI e o Gerenciador de Cadeia de Custódia se dá via REST API. O primeiro usa, em teoria, a *blockchain* Cardano, através do PRISM Node. Já o segundo se comunica com o *ledger* do Hyperledger Fabric através do Fabric Gateway.

Já a Figura 4 ilustra a arquitetura de forma mais detalhada. Dentro do controle de acesso, há componentes criados em Python, através deles o perito — ou quem for emitir a credencial para ele — se comunica, através de uma interface via terminal, com o controlador *controller* de seu agente. O *controller* então comunica-se com o agente

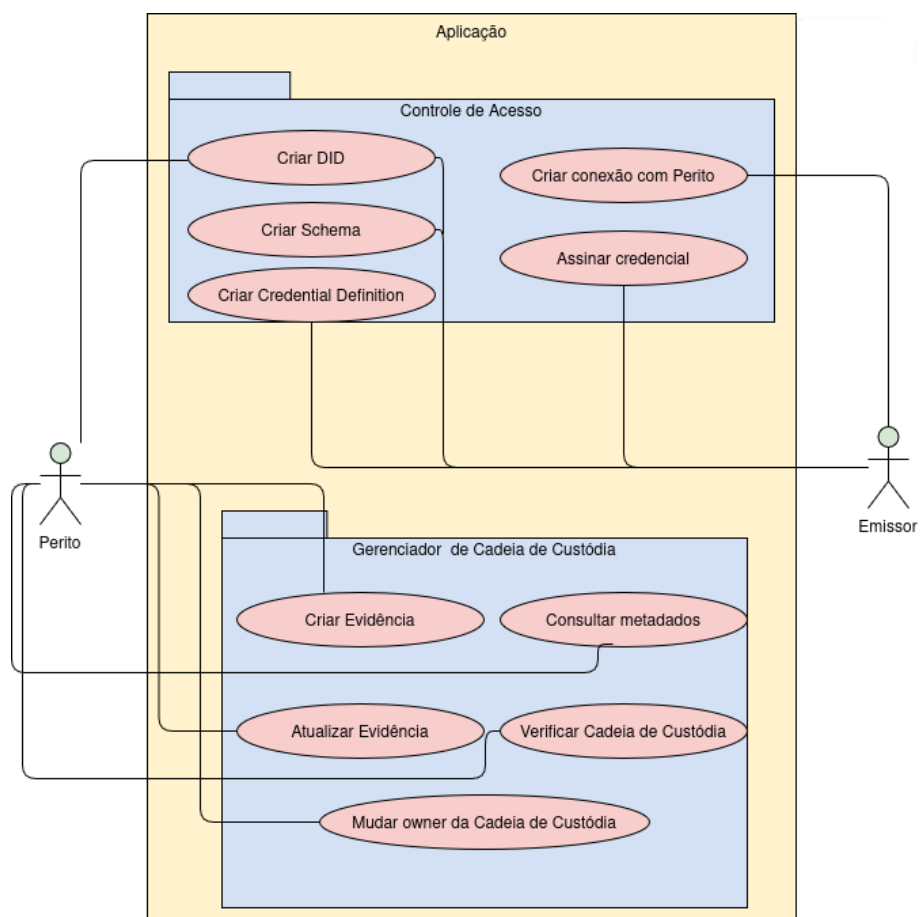


Figura 1. Casos de Uso da Aplicação

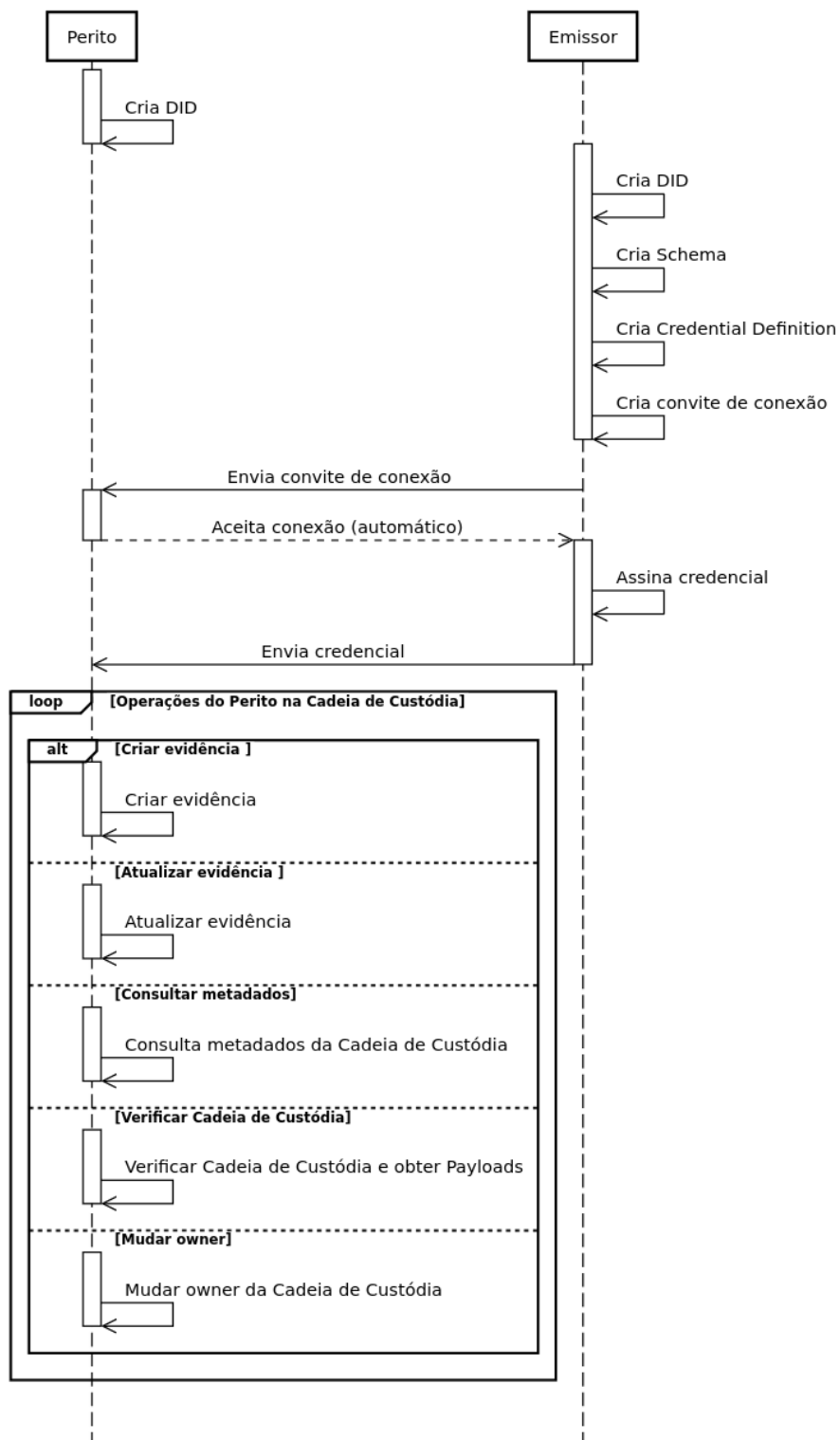


Figura 2. Diagrama de sequência envolvendo o Perito e o Emissor

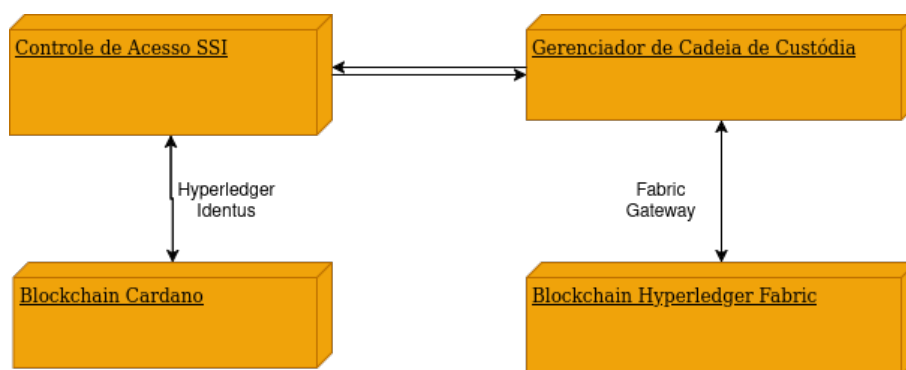


Figura 3. Visão Geral da Arquitetura

via REST API, e então as operações desejadas — criar um DID, por exemplo — são executadas.

O Gerenciador de Cadeia de Custódia possui um aplicativo escrito em Typescript que recebe requisições e então faz operações específicas, como adicionar uma nova evidência, ou seja, iniciando uma cadeia de custódia. A cadeia de custódia é representada por um encadeamento de credenciais verificáveis, sendo que parte dos metadados e a credencial em si são armazenados *off-chain* em um banco de dados, o MongoDB. Outros metadados, como por exemplo o hash da credencial, são armazenados diretamente na *blockchain*, através de contratos inteligentes.

Nota-se que enquanto a Figura 3 mostrou a arquitetura pensada em alto nível e em sua forma ideal, a Figura 4 destrincha os detalhes de implementação. Nela, é perceptível a ausência da *blockchain* Cardano. Isso se dá ao fato da aplicação construída utilizar uma construção do PRISM Node que utiliza somente um banco de dados para simular essa interação com a *blockchain* de fato.

Toda a implementação foi desenvolvida com base em um modelo assíncrono, fazendo uso de funções *async* e *await* em praticamente todas as definições e requisições, a fim de permitir a execução simultânea de tarefas e melhorar o desempenho do sistema.

4.3.1. Gerenciador de Cadeia de Custódia

A primeira parte deste componente é composto por um aplicativo escrito em Typescript que recebe requisições e a partir delas se comunica com um banco de dados MongoDB e, via Fabric Gateway, envia transações para serem executadas na rede *blockchain*. Os peritos enviam requisições para este componente e sempre sua credencial é verificada e avaliada conforme os requisitos de permissão do sistema.

Dentro do código Typescript, a cadeia de custódia foi modelada como um encadeamento de credenciais, como evidencia a Figura 5. Nota-se que a “Credencial N” foi a última a ser adicionada, isto é, ela é a ponta da cadeia. Toda credencial tem uma referência ao identificador da credencial que a precede neste encadeamento, além das informações necessárias sobre a evidência (5WH).

Cada credencial nesse encadeamento é uma documentação sobre algum processo ou análise feito sobre a evidência digital em questão. Essas credenciais estão no formato

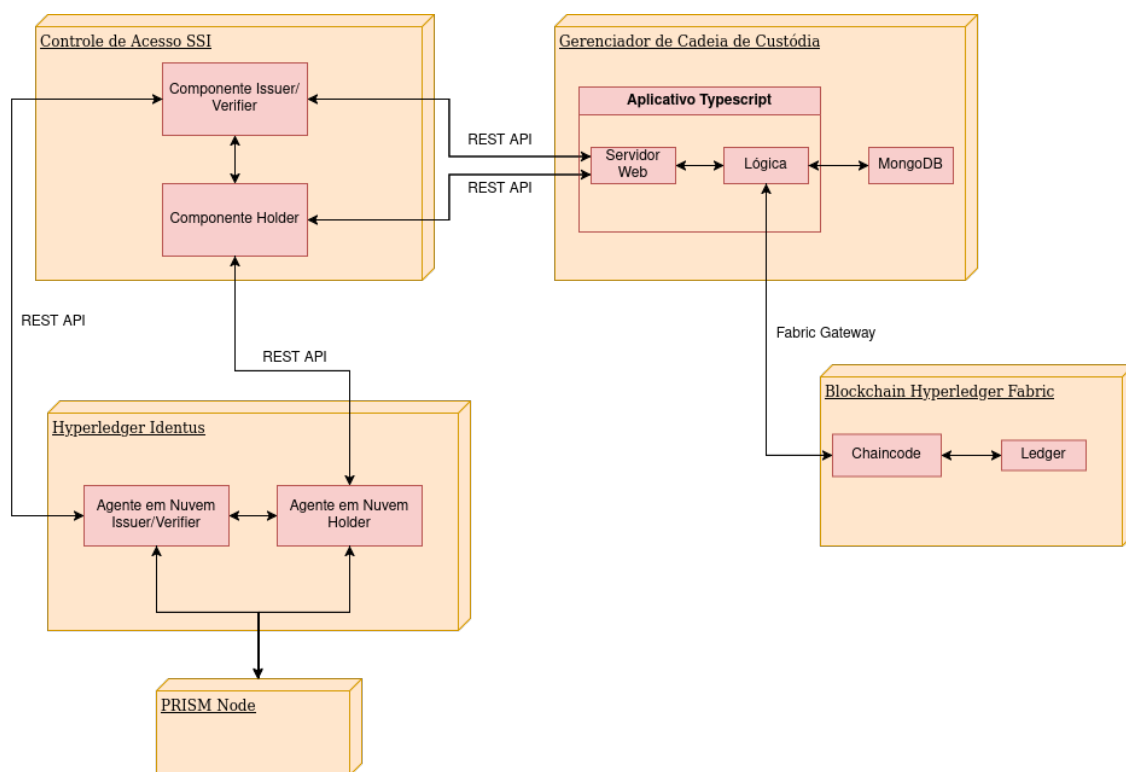


Figura 4. Arquitetura da implementação mais detalhada

JWT e são armazenadas, junto com metadados, no banco de dados. Além disso, o *hash* calculado dela é armazenado na *blockchain*, também ao lado de metadados para garantir a integridade e autenticidade dessas informações.

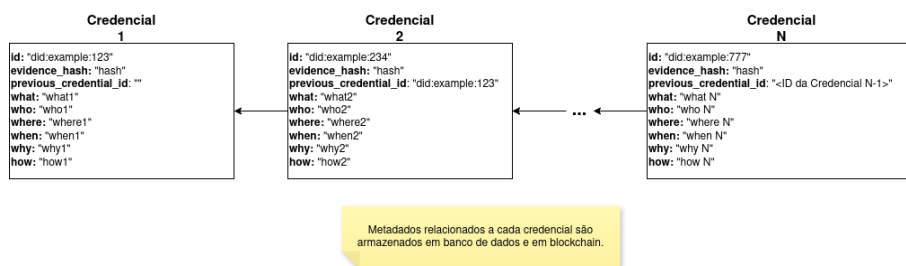


Figura 5. A cadeia de custódia como encadeamento de credenciais

Já a segunda parte deste componente é o *chaincode* escrito em Go. Ele implementa os contratos inteligentes que permitem escrever e ler os dados a respeito de cada credencial no *ledger*.

Foi utilizada a versão v2.5 do Hyperledger Fabric.

4.3.2. Controle de Acesso SSI

O controle de acesso foi implementado sobre o *Hyperledger Identus* v2.0.0, utilizando uma arquitetura de três agentes em nuvem rodando em *containers* Docker: um para o Emissor, um para o Perito (*Holder*) e um para o Verificador. A comunicação entre os

componentes é orquestrada por controladores desenvolvidos em Python, que interagem com os agentes via API REST.

Além disso, a interação dos usuários com o sistema é via terminal. Essa interface também foi implementada via Python.

O sistema utiliza credenciais no formato *AnonCreds*, o que habilita o uso de ZKPs. A parte mais importante desse fluxo é o Emissor estabelecendo uma conexão DIDComm com o Perito e emitindo uma credencial que contém, dentre outros atributos, um nível de autorização (representado por um número inteiro) e o hash da evidência permitida.

Quando o Perito tenta interagir com o Gerenciador de Cadeia de Custódia, o sistema atua como Verificador e solicita uma prova de autorização. Graças às ZKPs, o sistema não precisa ler o valor exato do nível de acesso do Perito, ele apenas verifica matematicamente se o atributo satisfaz um predicado, por exemplo `authorization_level >= 2`.

4.4. Limitações

A prova de conceito desenvolvida apresenta limitações técnicas e arquiteturais. A implementação atual do componente *Holder* não utiliza *webhooks*, e o sistema do *Issuer* foi projetado para gerenciar uma interação completa por vez, necessitando de aprimoramentos para lidar com conexões simultâneas em diferentes fases de protocolo.

Além disso, o nó PRISM opera de forma isolada, sem conexão com uma rede Cardano real. Em termos de integração, foram utilizados bancos de dados locais como solução provisória para compartilhar informações entre agentes e simular sistemas externos (como cadastros institucionais de peritos). Essa abordagem deixa em aberto a definição de uma arquitetura definitiva para essas integrações, ressaltando que vulnerabilidades na autenticação desses sistemas externos podem comprometer a integridade da cadeia de custódia.

5. Conclusão

Este trabalho apresentou uma arquitetura para a cadeia de custódia forense que integra a imutabilidade da tecnologia *blockchain* ao controle de acesso proporcionado pela Identidade Autoss soberana (SSI). Através de uma prova de conceito utilizando *Hyperledger Fabric* e *Hyperledger Identus*, demonstrou-se a viabilidade de modelar a custódia como um encadeamento de credenciais verificáveis, garantindo integridade e autenticidade. O uso de provas de conhecimento zero (ZKP) permitiu a verificação de permissões de peritos sem expor dados sensíveis, atendendo aos requisitos de privacidade do domínio forense.

A proposta avança o estado da arte ao preencher lacunas identificadas na literatura [Loffi et al. 2025a], especificamente a integração prática entre redes permissionadas e SSI, um cenário até então subexplorado. A solução diferencia-se pela aplicação de controle de acesso granular e descentralizado, superando as limitações de modelos que utilizam *blockchain* apenas para o registro passivo de documentos.

Como trabalhos futuros, sugere-se a realização de análises de desempenho e escalabilidade da arquitetura, bem como auditorias de segurança mais aprofundadas para cenários de produção. Recomenda-se também o estudo comparativo entre diferentes for-

matos de credenciais, como JWT e SD-JWT, para avaliar vantagens em relação ao *Anon-Creds*, além da integração da solução com sistemas de identidade institucionais reais.

Referências

- Aldosary, M. and Alqahtani, N. (2021). A survey on federated identity management systems limitation and solutions. *International Journal of Network Security & Its Applications (IJNSA)* Vol, 13.
- Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. (2018). Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15.
- Araújo, M. (2023). Inteligência artificial, blockchain e a cadeia de custódia da prova no processo penal. *Revista da UFMG*, 30.
- Azzam, F., Jaber, M., Saies, A., Kirresh, T., Awadallah, R., Karakra, A., Barghouthi, H., and Amarneh, S. (2023). The use of blockchain technology and ocr in e-government for document management: Inbound invoice management as an example. *Applied Sciences*, 13(14):8463.
- Bahga, A. and Madiseti, V. K. (2016). Blockchain platform for industrial internet of things. *Journal of Software Engineering and Applications*, 9(10):533–546.
- Belchior, R., Putz, B., Pernul, G., Correia, M., Vasconcelos, A., and Guerreiro, S. (2020). Ssibac: self-sovereign identity based access control. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1935–1943. IEEE.
- Brasil (1941). Código de processo penal: Decreto-lei n. 3.689, de 3 de outubro de 1941. Online; acesso em 7 de junho de 2025.
- Brasil (2019). Lei nº 13.964, de 24 de dezembro de 2019. Conhecida como Pacote Anticrime. Altera o Código de Processo Penal e inclui o art. 158-A. Acesso em 7 de junho de 2025.
- Foundation, T. L. (2024a). Glossary — hyperledger identus. Online; Acesso em: 27 nov. 2025.
- Foundation, T. L. (2024b). Secret storage — hyperledger identus cloud agent. Online; Acesso em: 27 nov. 2025.
- Google (2025). Identity. <https://developers.google.com/identity>. Acessado em 17/10/2025.
- Hyperledger (2025a). Fabric gateway. <https://hyperledger-fabric.readthedocs.io/en/latest/gateway.html>. [Online; acesso em 18 de maio de 2025].
- Hyperledger (2025b). Smart contracts and chaincode. <https://hyperledger-fabric.readthedocs.io/en/latest/smartcontract/smartcontract.html>. [Online; acesso em 18 de maio de 2025].
- Identus, H. (2025). Identus cloud agent api reference. <https://hyperledger-identus.github.io/docs/agent-api/>. Acessado em 21 out. 2025.

- Kiruthika, M. and Ponnuswamy, P. P. (2021). Fusion of iot, blockchain and artificial intelligence for developing smart cities. In *Blockchain, Internet of Things, and Artificial Intelligence*, pages 155–177. Chapman and Hall/CRC.
- Loffi, L., Camillo, G. L., De Souza, C. A., Westphall, C. M., and Westphall, C. B. (2025a). Management of the chain of custody of digital evidence using blockchain and self-sovereign identities: A systematic literature review. *IEEE Access*.
- Loffi, L., Camillo, G. L., Westphall, C. M., and Westphall, C. B. (2025b). Blockchain-based chain of custody for digital evidence with hyperledger fabric. In *2025 International Conference on Information Networking (ICOIN)*, pages 437–442. IEEE.
- Ou, H.-H., Chen, G.-Y., and Lin, I.-C. (2025). A self-sovereign identity blockchain framework for access control and transparency in financial institutions. *Cryptography* (2410-387X), 9(1).
- Preukschat, A. and Reed, D. (2021). *Self-Sovereign Identity: Decentralized Digital Identity and Verifiable Credentials*. Manning Publications Co., Shelter Island, NY, 1 edition.
- Sunarya, P. A., Khoirunisa, A., Nursaputri, P., et al. (2020). Blockchain family deed certificate for privacy and data security. In *2020 Fifth International Conference on Informatics and Computing (ICIC)*, pages 1–4. IEEE.
- The Linux Foundation (2025). Identus. Online; Acesso em 9 de junho de 2025.
- The Linux Foundation (s.d.a). Anoncreds specification. <https://hyperledger.github.io/anoncreds-spec/>. [Online; acesso em 5 de junho de 2025;].
- The Linux Foundation (s.d.b). Prism node overview. Online; Acesso em: 9 jun. 2025.