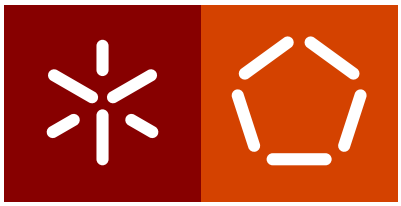
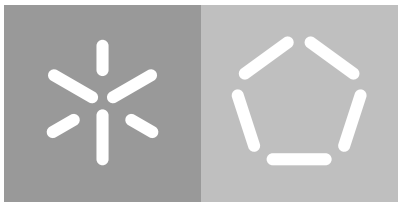


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Ricardo Jorge Marques Peixoto

DIDs, Claims, Credentials e Blockchains (Self-sovereign Identity)

May 2021



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Ricardo Jorge Marques Peixoto

DIDs, Claims, Credentials e Blockchains (Self-sovereign Identity)

Master dissertation
Master Degree in Informatics Engineering

Dissertation supervised by
Professor José Carlos Bacelar Almeida

May 2021

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Ricardo Jorge Marques Peixoto

_____

AGRADECIMENTOS

Em primeiro lugar quero agradecer ao Dr. José Eduardo Pina Miranda, orientador da dissertação (por parte da empresa Devise Futures, Lda), por sugerir um tema tão interessante e inovador e por se mostrar sempre disponível para me ajudar e encaminhar durante este longo percurso. A sua dedicação e entusiasmo pelo tema impulsionaram-me a explorar todo um mundo de inovação e a pensar fora da caixa.

Agradeço, de igual forma, ao Professor José Carlos Bacelar Almeida, orientador da dissertação, por aceitar orientar esta dissertação, por contribuir com diversas sugestões e opiniões relevantes e, acima de tudo, por me levar a questionar e explorar vários conceitos importantes.

Quero também agradecer à comunidade Hyperledger por disponibilizar vários projetos, nos quais pude basear o meu trabalho, e pela constante disponibilidade que os seus membros demonstraram em esclarecer todas as dúvidas que surgiram e ajudar a resolver todos os problemas com os quais me deparei, sendo eu um simples estudante. Agradeço em particular ao Stephen Curran por estar sempre disposto a ajudar mesmo quando as minhas dúvidas transcendiam as suas áreas de especialização. Deixo ainda o meu obrigado a todos os membros desta comunidade que direta ou indiretamente contribuíram para a realização desta dissertação.

Agradeço ainda a todos os meus amigos por me proporcionarem vários momentos de descontração e convívio nos quais pude recarregar as minhas energias.

Por fim, agradeço aos meus pais pelo constante apoio e motivação, e pela ajuda na superação dos momentos mais adversos. Sem eles esta jornada não teria sido possível, razão pela qual lhes dedico esta dissertação.

ABSTRACT

The high growth in the use of digital identities creates the need to develop mechanisms that can protect the personal data of each individual. The way identity is treated today prevents each of us from being able to control our personal information. This is due to the centralized architecture in which the personal data are inserted, that is, all these data are kept together and controlled by the entities responsible for providing the most varied services, which is wrong since the identity belongs to the person and thus it must be responsible for controlling that identity. Centralized identity management brings within itself several problems, whether intentional (that is, data correlation for profiling) or unintentional (that is, data breach).

To face this problem, multiple entities across the world are developing decentralized identity managment systems based on a self-sovereign identity architecture where each individual is responsible for managing and storing a set of credentials, each with parts of their personal information. A self-sovereign identity architecture allows users to provide only small parts of their personal information or even to omit any type of personal identification, using cryptographic techniques like selective disclosure and zero-knowledge proofs, which allows them to have more control over their privacy.

Taking into account the current problems of digital identity, this dissertation aims to explore the state of the art and develop a proof of concept, through the implementation of a system based on self-sovereign identity, which is able to cover the use cases for digital identity. Thus, this document shows the architecture implemented, with a blockchain, responsible for the storage of all public data, and a user agent, responsible for facilitating all interactions of the various users with the developed system.

The proof of concept developed allows not only to validate that it is possible to correct many of the problems associated with centralized identity management, but also to explore new cryptographic strategies in order to improve the way each of us manages our own identity.

Keywords: decentralized identifiers, digital identity, distributed ledger technology, privacy, self-sovereign identity, verifiable credentials.

RESUMO

O elevado crescimento no uso de identidades digitais gera a necessidade de desenvolver mecanismos que sejam capazes de proteger os dados pessoais de cada indivíduo. O modo como a identidade é tratada atualmente impede cada um de nós seja capaz de controlar as suas informações pessoais. Isto deve-se à arquitetura centralizada na qual os dados pessoais estão inseridos, ou seja, todos estes dados são mantidos em conjunto e controlados pelas entidades encarregues de fornecer os mais variados serviços, o que está errado por definição uma vez que a identidade pertence à pessoa e esta deverá ser responsável por controlá-la. A gestão centralizada de identidades traz consigo diversos problemas, sejam estes intencionais (i.e. correlação de dados para criação de perfis) ou não intencionais (i.e. *data breaches*).

Para enfrentar este problema, várias entidades em todo o mundo estão a desenvolver sistemas de gestão de identidade descentralizados com base numa arquitetura de *self-sovereign identity* na qual cada indivíduo é responsável por gerir e armazenar um conjunto de credenciais que contêm partes da sua identidade digital. Uma arquitetura de *self-sovereign identity* permite que os utilizadores partilhem apenas pequenas partes de suas informações pessoais ou até mesmo que omitam qualquer tipo de identificação pessoal, usando técnicas criptográficas como divulgação seletiva e provas de conhecimento zero, o que lhes permite ter mais controlo sobre sua privacidade.

Tendo em consideração os problemas atuais da identidade digital, esta dissertação tem como objetivo explorar o estado da arte e desenvolver uma prova de conceito, através da implementação de um sistema baseado em *self-sovereign identity*, que seja capaz de cobrir os vários casos de uso da identidade digital. Deste modo, este documento apresenta a arquitetura implementada, com uma *blockchain*, responsável por armazenar todos os dados de caráter público, e um agente de utilizador, responsável por facilitar todas as interações dos vários utilizadores com o sistema desenvolvido.

A prova de conceito desenvolvida permite não só validar que é possível corrigir muitos dos problemas associados à gestão centralizada de identidades mas também explorar novas estratégias criptográficas com o objetivo de melhorar o modo como cada um de nós gere a sua própria identidade.

Palavras-Chave: credenciais verificáveis, identidade auto-soberana, identidade digital, identificadores descentralizados, privacidade, tecnologia de registo distribuído.

CONTEÚDO

LISTA DE FIGURAS

GLOSSÁRIO

A

AGENTE DO UTILIZADOR Programa que atua como intermediário na comunicação entre *holders*, *issuers* e *verifiers*, devendo sempre atuar de acordo com a vontade do seu controlador. Por defeito todas as entidades do sistema recorrem a agentes para interagirem entre si e com o sistema.

C

CLAIM Declaração feita sobre um *subject*, também designada por atributo.

CREDENTIAL Conjunto de uma ou mais *claims* emitidas por um *issuer*.

CREDENTIAL DEFINITION Indica a intenção de um *issuer* de criar credenciais verificáveis, que devem corresponder a um determinado esquema, e especifica as chaves criptográficas que o *issuer* irá utilizar para assinar essas credenciais.

D

DISTRIBUTED LEDGER TECHNOLOGY Uma base de dados distribuída, gerida através de protocolos de consenso, na qual cada transação é assinada criptograficamente e encadeada à transação anterior.

DIVULGAÇÃO SELETIVA Capacidade fornecida ao *holder* para partilhar apenas uma parte dos dados presentes na *verifiable credential*.

DOCUMENTO DE UM DID Documento acessível através de um *verifiable data registry* que contém informação relacionada com o respetivo identificador descentralizado, tal como a chave pública do *subject* da credencial. Este documento pode conter também outros atributos ou *claims* que descrevam o *subject*. Todas as informações presentes neste documento devem ser de caráter público dado que qualquer entidade o pode consultar.

G

GRAPHQL Linguagem de procura utilizada para obter dados de APIs.

H

HASH DIGEST Resultado da execução de uma função de *hash* sobre uma mensagem..

HOLDER Entidade dententora das *verifiable credentials* e capaz de gerar *verifiable presentations* com os dados presentes nas mesmas. O *holder* nem sempre é o *subject* da credencial (i.e. quando o *subject* é um animal de estimação, o *holder* será o seu proprietário).

I

IDENTIFICADOR DESCENTRALIZADO Identificador global único, designado de DID, que está associado a uma entidade. Estes identificadores são armazenados num registo descentralizado e podem ser consultados por qualquer entidade a qualquer altura, anulando assim a necessidade de autoridades centralizadas para validação a singularidade do identificador. Um exemplo de um DID é did:example:njdf832nfo3.

ISSUER Entidade que emite *claims* sobre um ou mais *subjects*, agrupando-as em *verifiable credentials*.

L

LEDGER Registo de transações imutável e ordenado por tempo de inserção.

M

MINIMIZAÇÃO DE DADOS Ato de limitar a quantidade de dados que são partilhados ao mínimo necessário para efetuar uma tarefa.

MONGODB Base de dados NoSQL orientada a documentos.

N

NOSQL Termo genérico que representa os bases de dados não relacionais.

NÓ Máquina física que está conectada à rede da *blockchain* e tem como função efetuar escritas e/ou leituras de transações na mesma bem como participar no protocolo de consenso.

O

OFF-CHAIN Fora de uma *blockchain*.

ON-CHAIN Dentro de uma *blockchain*.

P

POOL Coleção de nós que estão a executar o protocolo de consenso.

PREDICADO DERIVADO Uma declaração booleana verificável sobre o valor de um atributo numa *verifiable credential*. Por exemplo, se apenas for requerida uma prova de maioridade do *subject* não é necessário fornecer a idade específica presente na credencial. Deste modo é possível criar uma prova baseada no predicado "idade é superior a 18 anos".

PROVER Num contexto geral, *prover* e *holder* representam a mesma entidade. Contudo, no ecossistema Hyperledger Aries, *holder* é o papel desempenhado pelo ator que pretende adquirir uma credencial enquanto que *prover* é o papel desempenhado pelo ator que pretende provar que possui certos atributos através de *verifiable presentations*.

R

REPOSITÓRIO Programa responsável por armazenar e proteger o acesso às credenciais do *holder*.

S

SCHEMA Esquema de dados que procura impor o uso de uma estrutura específica numa certa coleção de dados.

SERVICE ENDPOINT Ponto de contacto no qual o serviço pode ser acedido. O endereço num qual o agente de uma entidade recebe mensagens de outras agentes é um *service endpoint*.

SNAPSHOT Cópia do estado em que um sistema se encontra num determinado momento.

SQL Linguagem de procura utilizada para gerir dados armazenados numa base de dados relacional.

SQLITE Biblioteca escrita através da linguagem de programação C que implementa uma base de dados SQL.

SUBJECT Entidade ou objeto sobre o qual as *claims* são efetuadas.

V

VERIFIABLE CREDENTIAL Credencial digital inviolável que permite efetuar a validação de autoria através de técnicas criptográficas. *Verifiable credentials* são usadas para provar a posse de determiandos atributos tais como nome, idade, nacionalidade, entre outros, através do uso de *verifiable presentations*.

VERIFIABLE DATA REGISTRY Sistema que atua como mediador na criação e verificação de identificadores, chaves, esquemas de *verifiable credentials*, registos de revogação, entre outros. De uma forma geral este registo de dados é responsável por armazenar toda a informação pública necessária num sistema de gestão de identidades descentralizado.

VERIFIABLE PRESENTATION Estrutura utilizada pelo *holder* para provar a posse de certos atributos. Sempre que um *holder* pretende apresentar provas pode agrupar vários atributos de múltiplas credenciais numa única estrutura, *verifiable presentation*, e apresentá-la à entidade verificadora. Uma *verifiable presentation* é inviolável e está codificada de tal forma que permite validar a autoria dos dados.

VERIFIER Entidade que recebe *verifiable presentations* geradas pelos *holders*, e efetua o processo de validação das mesmas.

W

WALLET *Software* que armazena chaves privadas, credenciais e outras informações privadas e fornece mecanismos CRUD para controlar essas informações.

ACRÓNIMOS

A

ACL Access Control List.

AEAD Authenticated Encryption with Associated Data.

AEAD Authenticated Encryption with Associated Data.

AES-CBC Advanced Encryption Standard - Cipher Block Chaining.

API Application Programming Interface.

C

CA Certificate Authority.

CNE Comissão Nacional de Eleições.

CRUD Create, Read, Update and Delete.

CSRF Cross Site Request Forgery.

D

DI Departamento de Informática.

DID Decentralized Identifier.

DLT Distributed Ledger Technology.

DPKI Decentralized Public Key Infrastructure.

E

ECDH Elliptic-curve Diffie–Hellman.

ECDSA Elliptic Curve Digital Signature Algorithm.

EDDSA Edwards-curve Digital Signature Algorithm.

EID Cross Site Request Forgery.

EIDAS Cross Site Request Forgery.

H

HTTP Hypertext Transfer Protocol.

I

ID Identificador.

IIW Internet Identity Workshop [1].

IPFS InterPlanetary File System.

IRN Instituto dos Registos e Notariado.

J

JSON JavaScript Object Notation.

M

MAC Message Authentication Code.

MEI Mestrado em Engenharia Informática.

N

NIC Número de Identificação do Cidadão.

NIF Número de Identificação Fiscal.

NISS Número de Identificação de Segurança Social.

P

PII Personally Identifiable Information.

PKI Public Key Infrastructure.

R

RBFT Byzantine Fault Tolerance.

RBFT Plenum Redundant Byzantine Fault Tolerance.

RBFT Redundant Byzantine Fault Tolerance.

RFC Request for Comments.

RGPD Regulamento Geral sobre a Proteção de Dados.

S

SDK Software Development Kit.

SSI Self-Sovereign Identity.

T

TLS Transport Layer Security.

U

UM Universidade do Minho.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

UUID Universally Unique Identifier.

V

VC Verifiable Credential.

VP Verifiable Presentation.

W

W3C World Wide Web Consortium [2].

X

XML Extensible Markup Language.

XSS Cross Site Scripting.

Z

ZKP Zero-Knowledge Proofs.

1

INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

A informação de identificação pessoal (PII) descreve as caraterísticas únicas de uma entidade. Esta informação é usada diariamente para executar as mais variadas ações, desde levantamentos de dinheiro num banco até à conexão a serviços *web*, podendo estar contida tanto em formato físico como em formato digital. Um subconjunto de PII é geralmente designado por credencial.

A necessidade de utilizar mecanismos de identificação pessoal de forma generalizada remonta a 1938, maioritariamente devido à segunda guerra mundial. O Reino Unido foi a primeira nação a implementar um sistema de cartões de identificação para todos os residentes, seguindo-se a Alemanha ainda em 1938 e, mais tarde, a França, a Grécia e a Polónia, em 1940 [3].

Desde então, a representação da informação de identificação pessoal foi evoluindo, até que nos últimos anos começou a migrar acentuadamente para formato digital. À medida que o uso de credenciais tem expandido, várias instituições, tais como universidades, bancos, ou empresas, emitem as suas próprias credenciais de forma a disponibilizarem os seus serviços. Devido à inexistência de padrões para a representação destas credenciais, cada indivíduo acaba por possuir inúmeras credenciais diferentes para ter acesso aos vários serviços, em particular no mundo digital. Como consequência, os indivíduos acabam com a sua informação de identificação pessoal espalhada por várias organizações, o que reduz drasticamente o controlo que estes possuem sobre os seus dados pessoais. Deste modo, a informação passa a ser controlada por terceiros, alguns dos quais o respetivo dono nem sequer tem conhecimento, e é usada e partilhada várias vezes sem a sua autorização.

No final do século XX começaram a ser desenvolvidas soluções que visavam colocar o utilizador como dono absoluto dos seus dados pessoais e mais tarde, em 2016, surgiu o conceito de *Self-Sovereign Identity* (SSI), que procura não só desenvolver novas formas de gerir identidades de modo a garantir que os seus donos têm total controlo sobre elas, isto é, como e com quem as suas informações pessoais são partilhadas, mas também descrever a interoperabilidade da identidade do indivíduo com as várias plataformas existentes [4]. Apesar desta ideia estar a ser desenvolvida há mais de duas décadas foi apenas na última

década que o interesse em torno desta área começou a crescer exponencialmente, em parte devido ao rápido crescimento do uso de *blockchains* e da tecnologia descentralizada em geral. O aparecimento destas tecnologias descentralizadas (DLT), nomeadamente de *blockchains*, proporcionou uma nova forma de abordar os sistemas de gestão de identidades, passando a ser possível substituir os sistemas tradicionais, onde o controlo das identidades era centralizado e detido por organizações, por um modelo descentralizado onde cada utilizador possui controlo total e individual da sua informação de identificação pessoal. Adicionalmente, a DLT oferece imutabilidade dos dados, o que garante a sua integridade ao longo do tempo [5]. Esta caraterística permite utilizar *blockchains* como fonte de verdade, o que possibilita a verificação e validação dos dados armazenados a qualquer momento e por qualquer indivíduo.

1.2 Motivação

A identidade está na base da maior parte das tarefas que efetuamos e o seu uso espalha-se pelos vários setores da sociedade, desde serviços financeiros, como gestão de uma conta bancária ou acesso a crédito, até serviços de saúde, como o acesso a seguros de saúde e vacinação, passando por serviços públicos (i.e. acesso à segurança social), educação e direitos políticos e legais.

Atualmente os indivíduos são obrigados a criar múltiplas identidades (i.e. uma identidade para cada serviço de web) sem possuírem controlo sobre qualquer uma delas. O uso de endereços de email ou identificadores sociais para as mais variadas funções fez com que os utilizadores desenvolvessem o hábito de usar o mesmo identificador e senha em plataformas totalmente distintas, o que leva a uma fraca segurança e possibilita a correlação de contas e, consequentemente, de dados privados. Citando o Evernym INC [6], "nenhum de nós possui uma identidade digital", afirmando que "apenas 'alugamos' identidades de cada serviço de *web* ou aplicação que usamos, resultando numa confusão ineficiente, cheia de fraudes e que viola a nossa privacidade". Como consequência os utilizadores acabam com vários nomes de utilizador e senhas, informação de identificação pessoal espalhada pela *internet*, roubo de identidade derivado do uso de senhas fracas, e partilha da sua informação pessoal sem a sua autorização. Adicionalmente, como a informação de identificação pessoal de uma pessoa é mantida por várias organizações em diferentes formas, algumas destas organizações acabam por possuir dados desatualizados, o que pode levar a conflitos com outras organizações. Além disso, estas organizações estão constantemente a partilhar enormes quantidades de informação, correlacionando-as de forma a obter desde os hábitos de compras dos indivíduos até à sua localização durante o dia ou com quem convivem. Empresas e organizações que lidam com elevadas quantidades de dados pessoais conseguem gerar milhões de euros em lucro através do uso dessas informações. Outro problema de armazenar informação de identificação pessoal do lado das organizações é o facto destes

dados estarem armazenados em repositórios centralizados atrativos a *hackers*, o que resulta em notícias diárias de quebras de segurança e divulgação massiva de informações pessoais armazenadas nesses repositórios.

Por outro lado, também as organizações são vítimas do atual sistema de identidades. Segundo o artigo *The Financial Cost of Fraud 2019* [7], em 2019 foram gastos 3.5 biliões de euros devido a despesas resultantes de fraudes. Adicionalmente, quebras de segurança dos dados pessoais levam à perda da confiança depositada na organização. Aliado a isto, as organizações são forçadas a atuarem como fornecedores de identidades e a contratar peritos na área da segurança, bem como a respeitar todos os aspetos legais advindos do manuseamento de dados privados (i.e. RGPD). Todos os fatores indicados obrigam a que estas organizações invistam muito dinheiro na segurança e proteção dos dados armazenados. Como consequência, grande parte das organizações não quer ter este fardo de armazenar informação de identificação pessoal, contudo foram forçadas a tal de maneira a poderem fornecer os seus serviços. A própria Microsoft acredita [8] que um sistema de identidades descentralizado baseado em padrões pode permitir um novo conjunto de experiências que possibilitam utilizadores e organizações a manter maior controlo sobre os seus dados e a fornecer um maior grau de confiança e segurança às suas aplicações, dispositivos e fornecedores de serviços.

Outro fator importante passa pelo desrespeito à minimização dos dados. Num sistema de gestão de identidades tradicional é nos requisitado o cartão de cidadão para compra de bebidas alcoólicas, com todas as informações nele presentes, quando na realidade seria apenas necessário comprovar a foto e a data de nascimento. Este caso não apresenta um risco elevado pois o funcionário do estabelecimento dificilmente irá decorar todas as informações por forma a roubá-las. Contudo, quando passamos para sistemas digitais, estes geralmente registam todos os dados fornecidos, o que aumenta o impacto em caso de quebras de segurança.

Por fim, uma autoridade pode negar a identidade de um indivíduo ou confirmar uma identidade falsa. Isto é possível devido ao controlo de identidades ser centralizado. Outro problema grave derivado da ausência de identidade deve-se à ausência de registo de nascimento de bebés em países pouco desenvolvidos [9]. Esta ausência de identidade torna mais difícil impedir casamento infantil, aumenta o risco enfrentado pelos refugiados ao atravessar fronteiras, facilita a desigualdade de salários e pode levar a outros problemas muito graves.

O objetivo das *verifiable credentials* passa por dar às pessoas o controlo sobre a sua identidade digital, criptograficamente protegida, e estabelecer uma forte relação entre a pessoa e a sua identidade de forma a dificultar personificação ou roubo de identidade. A *Self-sovereign identity* visa criar um sistema de gestão de identidades descentralizado no qual cada indivíduo é responsável por gerir e armazenar um conjunto de credenciais, cada uma delas

contendo partes da sua informação pessoal, designadas por *verifiable credentials*, que na sua totalidade formam a identidade do indivíduo. Uma arquitetura de *self-sovereign identity* permite que os utilizadores partilhem apenas pequenas partes das suas informações pessoais ou até mesmo que omitam qualquer tipo de identificação pessoal, recorrendo a técnicas criptográficas como divulgação seletiva e provas de conhecimento zero, o que lhes permite ter mais controlo sobre sua privacidade. Complementarmente, se cada indivíduo armazenar as suas informações pessoais dentro do seu domínio, não irá existir um repositório centralizado com informações pessoais de milhares de pessoas que possa ser explorado de forma a comprometer a privacidade das mesmas. Por fim, as *verifiable credentials* procuram ainda fornecer identidade digital a pessoas de países pouco desenvolvidos de modo a fornecer-lhes melhor acesso aos direitos humanos e à economia global [10].

1.3 OBJETIVOS

O principal objetivo desta dissertação é desenvolver uma prova de conceito com base numa arquitetura que respeite os princípios de *self-sovereign indenty*. Para tal, será desenvolvida uma solução capaz de substituir os sistemas de gestão de identidades tradicionais por um sistema de gestão de identidades apoiado em tecnologia descentralizada, que siga uma arquitetura de *self-sovereign identity*, de modo a que os indivíduos controlem a sua informação de identificação pessoal que circula no mundo digital, ou seja, a sua identidade digital. Por conseguinte, a solução desenvolvida deverá permitir que um indivíduo armazene as suas informações pessoais em repositórios controlados por si e disponibilizar um registo descentralizado, que atue como fonte de verdade, no qual serão armazenados todos os dados de caráter público necessários para o correto funcionamento do sistema. Este sistema de gestão de identidades descentralizado deverá ser capaz de executar as várias operações do ciclo de vida de uma credencial, desde a sua emissão até à sua validação e/ou destruição, respeitando o Regulamento Geral sobre a Proteção de Dados [11]. Deverá também fornecer mecanismos que permitam que uma entidade emissora possa emitir credenciais criptograficamente seguras e verificáveis, que um indivíduo seja capaz de requisitar a emissão de credenciais a entidades emissoras e fornecer provas criptográficas dos atributos presentes nessas credenciais a fornecedores de serviços, e que os fornecedores de serviços sejam capazes de validar as credenciais que lhes forem fornecidas.

Tratando-se de uma prova de conceito, esta dissertação também tem como objetivo desenvolver uma solução suficientemente genérica de modo a permitir a sua utilização em diversos casos de uso reais que estão presentes no dia-a-dia dos cidadãos, e validar a sua aplicabilidade através da aplicação da mesma a três casos de uso distintos.

Por fim, esta dissertação irá avaliar a possibilidade de coexistência e integração dos sistemas de gestão de identidades tradicionais com a solução desenvolvida.

1.4 ESTRUTURA DO DOCUMENTO

O documento começa por apresentar o estado da arte, no capítulo 2, onde são discutidos vários aspetos importantes de um sistema de gestão de identidades baseado em *self-sovereign identity*. Este capítulo começa por mostrar a evolução da identidade digital, na secção 2.1, e o que é o conceito *self-sovereign identity* bem como uma explicação detalhada sobre os identificadores descentralizados utilizados neste tipo de sistemas, na secção 2.2. De seguida, a secção 2.3 ilustra a principal arquitetura de *self-sovereign identity* definida pela comunidade até ao momento e define as várias componentes que a constituem. Nesta arquitetura é apresentada a componente *data registry*, baseada em tecnologia DLT (*Distributed Ledger Technology*). Sendo DLT um tema muito complexo e um dos principais pilares desta dissertação, a secção 2.4 introduz o conceito de DLT, aborda o tema de *blockchains* (um tipo de DLT) e explica a importância do uso deste tipo de tecnologias num ecossistema SSI. Por fim, as secções 2.5 e 2.6 apresentam considerações de segurança e privacidade que devem ser consideradas durante o desenvolvimento de soluções baseadas em *self-sovereign identity*.

Após a apresentação do estado da arte, o documento descreve a implementação de uma solução baseada em *self-sovereign identity* com o objetivo de efetuar uma prova de conceito, no capítulo 3. Neste capítulo é inicialmente apresentada a arquitetura da solução desenvolvida e as várias componentes que a constituem. Devido à integração da solução com algumas componentes desenvolvidas pela comunidade *Hyperledger* [12], este capítulo começa com a apresentação de vários projetos que estão a ser desenvolvidos por esta comunidade. Posteriormente, a secção 3.2 aborda o agente do utilizador desenvolvido e a secção 3.3 explica que estratégias foram utilizadas para garantir a interoperabilidade do agente desenvolvido com outros agentes externos, nomeadamente a implementação de protocolos de comunicação universais. Este capítulo aborda também as duas componentes responsáveis por armazenar a informação envolvida num sistema baseado em *self-sovereign identity*: a *wallet* (para a informação privada), na secção 3.4.1, e a *blockchain* (para a informação pública), na secção 3.5.

Por fim, no capítulo 4 são explorados três casos de uso comuns no nosso dia-a-dia aos quais pode ser aplicado o sistema desenvolvido.

2

ESTADO DA ARTE

2.1 EVOLUÇÃO DA IDENTIDADE DIGITAL

Fornecer aos utilizadores controlo sobre as suas identidades digitais tem sido o objetivo de diversos indivíduos e organizações ao longo das últimas três décadas. A informação que se segue, com os vários eventos que ocorreram nos últimos anos, foi retirada do documento *The Path to Self-Sovereign Identity* [4], publicado por Christopher Allen.

Em 1991, *Pretty Good Privacy* (PGP), um *software* de criptografia, deu os primeiros passos na criação de uma solução viável ao introduzir a *Web of Trust*, onde os nós da rede inseriam e validavam chaves públicas, fornecendo assim o primeiro modelo de gestão de confiança descentralizada. Esta solução acabou por nunca ser adotada como padrão. Em 1995 surgiram as primeiras autoridades de certificação, para ajudar os sites de comércio *online* a provarem ser quem afirmavam ser. Contudo, este modelo não era capaz de remover a necessidade de uma identidade centralizada. Apesar disso, este modelo assegurava uma confiança aceitável, o que fez com que fosse adotado e utilizado a nível mundial até à atualidade. Em 1996 Carl Ellison, ao tentar conjugar as duas ideias anteriores, sugeriu um método de verificar entidades *online* recorrendo a uma troca de segredos partilhados através de um canal seguro. Mais tarde, em 1999, Ellison foi também o principal elemento do projeto SPKI/SDSI [13] [14], cujo objetivo era construir uma infraestrutura pública para certificados de identidade que pudesse substituir o sistema "X.509". Nesta infraestrutura, as autoridades centralizadas eram uma opção mas não eram a única opção. Ainda em 1999, a Microsoft iniciou o projeto *Microsoft's Passport* [15] que visava permitir o uso da mesma identidade em vários *sites*. Contudo, este esquema mantinha o controlo centralizado na Microsoft. Em 2001, A Microsoft organizou a *Liberty Alliance* [16], que dividia o controlo de uma autoridade central por várias autoridades poderosas, o que apenas endereça parcialmente o problema apresentado.

Foi apenas no século XXI que começaram a surgir propostas que se focavam numa identidade centrada no utilizador. Em 2000, a *Augmented Social Network* (ASN), afirmando que o *Passport* e a *Liberty Alliance* depositavam muito ênfase na privatização da informação e modelização dos utilizadores como consumidores, começou a explorar uma nova forma de identidade digital. Esta identidade digital seria persistente e cada individuo teria o direito

de controlar a sua própria identidade, aspetos estes que não eram cumpridos pelas duas abordagens da Microsoft. Esta nova forma de abordar a identidade digital, introduzida pela ASN num relatório extensivo [17], tornou-se a fundação para as várias soluções que se seguiram. Em 2001, a *Identity Commons* [1] começou a consolidar informação sobre identidade digital com foco na descentralização e em 2005, em conjunto com o *Identity Gang*, criou o grupo de trabalho *Internet Identity Workshop* (IIW), que tem avançado esta ideia de identidade descentralizada (centralizada no utilizador) até à atualidade. O trabalho do *IIW* deu suporte a muitos métodos de criação de identidade digital, nomeadamente OpenID (2005) [19], OpenID 2.0 (2006) [20], OAuth (2010) [21], FIDO (2013) [22] e OpenID Connect (2014) [23].

Em 2008 apareceu o Facebook Connect [24], que apesar de ser uma implementação centralizada no utilizador, mantinha o controlo das identidades centralizado no Facebook (único fornecedor de identidades digitais). Como resultado, os utilizadores estavam sujeitos a perder as várias identidades de vários *sites* em simultâneo se o Facebook assim o desejasse. Por esta razão, a abordagem do Facebook foi rapidamente rejeitada.

Até este ponto as implementações centralizadas nos utilizadores apenas forneciam interoperabilidade e algum nível de consenso do utilizador sobre como e com quem partilhar as suas identidades. Contudo, ainda era necessário passar o controlo das identidades para o lado dos utilizadores. Na última década começou a aumentar o uso do termo *self-sovereign-identity* para indicar o utilizador como dono da sua própria identidade. Uma das primeiras referências a este termo surgiu em 2012 através de um documento do escritor Moxie Marlinspike, "*Sovereign Source Authority*", onde este escreveu que os indivíduos têm direito a uma identidade. Desde então esta nova abordagem para o controlo de identidades tem proliferado, impulsionando o desenvolvimento de estratégias criptográficas para proteger a autonomia e o controlo do utilizador. Adicionalmente, o crescimento das tecnologias descentralizadas, nomeadamente *blockchains*, proporcionou uma abordagem viável para a implementação de um registo que atuasse como fonte de verdade, o que rapidamente levou à introdução destas tecnologias descentralizadas em vários modelos de SSI, cujo papel seria atuar como um registo de dados públicos para armazenar esquemas de credenciais, chaves públicas, entre outros. A utilização de tecnologia descentralizada em sistemas SSI funcionou tão bem que permitiu um rápido crescimento do termo ao longo dos últimos anos, trazendo um grande número de novos indivíduos e entidades para o desenvolvimento de soluções baseadas em identidade descentralizada.

Em 2016 surgiu então o conceito de *Self-Sovereign Identity*, cujo objetivo é não só garantir que os os indivíduos têm controlo total sobre as suas entidades mas também assegurar a interoperabilidade das várias soluções desenvolvidas.

1 *Identity Commons* é uma comunidade de grupos cujo objetivo é desenvolver a camada social e de identidade da rede [18].

2.2 SELF-SOVEREIGN IDENTITY

O termo *self-sovereign identity* (SSI) representa uma abordagem à identidade digital que visa colocar o utilizador no centro do modelo, ou seja, o utilizador deve ser capaz de controlar totalmente a sua identidade, decidindo como, quando e com quem deseja partilhar as suas informações pessoais. Adicionalmente, SSI visa atingir a interoperabilidade de identidades através de vários sistemas e plataformas. Para ser capaz de realizar tais objetivos, uma arquitetura SSI não pode estar presa a um único local que possa ser facilmente controlado por terceiros. Adicionalmente, deve permitir que utilizadores comuns possam obter *claims* (atributos), que podem incluir informação de identificação pessoal, filiação a grupos, etc. Citando Christopher Allen [4], "na criação de um sistema SSI deve-se ter sempre em consideração a proteção do indivíduo, nomeadamente defendendo-o contra perdas financeiras, prevenindo abusos aos direitos humanos, suportando quaisquer direitos associados aos indivíduos", entre outros. A subsecção 2.2.1 irá detalhar cada um dos princípios de *self-sovereign identity* que devem ser considerados, segundo Christopher Allen [4].

2.2.1 *Princípios da Self-Sovereign Identity*

Numa tentativa de definir mais rigorosamente o termo *Self-Sovereign-Identity*, diversas entidades, nomeadamente o W3C, partilharam várias ideias sobre os pontos mais importantes que deveriam ser tidos em consideração. Christopher Allen aglomerou todos esses pontos chave e propôs os *Dez Princípios da Self-Sovereign-Identity* [4]. Estes princípios tentam assegurar o controlo do utilizador sobre a sua identidade, garantindo a segurança da mesma, mas também definir como esta identidade deve operar a nível do sistema como um todo.

1. **Existência**

   Os utilizadores devem ter uma existência independente. Isto implica que a entidade nunca pode existir unicamente em formato digital, ou seja, a entidade apenas torna públicos e acessíveis um conjunto de atributos da sua identidade que já existem.

2. **Controlo**

   Os utilizadores devem controlar as suas identidades. Deve-se garantir a validade contínua de uma identidade através de algoritmos seguros, mantendo o utilizador como a autoridade máxima sobre a sua credencial, podendo referir-se à mesma, atualizá-la ou ocultá-la a qualquer momento. Contudo, o utilizador não controla todas as *claims* associadas à sua identidade uma vez que outros utilizadores podem efetuar *claims* sobre esse indivíduo.

3. **Acesso**

   Os utilizadores devem possuir acesso permanente aos seus próprios dados. Um utilizador deve poder, a qualquer momento, obter todas as *claims* e outros dados sobre a sua identidade não podendo haver dados escondidos. Isto não significa que o utilizador pode alterar todas as *claims* associadas à sua identidade, mas deve pelo menos ser capaz de saber que informações referentes a essa entidade estão a ser mantidas por terceiros e quem são esses terceiros.

4. **Transparência**

   Os sistemas usados para administrar e operar a rede de identidades devem ser transparentes tanto no seu funcionamento como na sua gestão e atualização. Os algoritmos devem ser de código aberto, conhecidos, o mais independentes possível de qualquer arquitetura particular, e qualquer entidade deverá ser capaz de verificar o seu funcionamento.

5. **Persistência**

   Identidades devem ser persistentes. Idealmente, identidades devem durar tanto tempo quanto o utilizador quiser. Contudo, devido ao elevado ritmo de mudança na *internet* tal pode não ser possível pelo que, no mínimo, as identidades devem persistir tanto tempo quanto o sistema de identidades no qual elas estão inseridas. É importante referir que isto não pode contradizer o direito a ser esquecido. Deste modo, é necessária uma separação evidente entre a identidade e as *claims* feitas sobre a mesma, para que o utilizador possa "esquecer" uma identidade e todas as *claims* feitas sobre essa identidade serem alteradas ou removidas apropriadamente ao longo do tempo.

6. **Portabilidade**

   Informações e serviços sobre uma identidade devem ser transportáveis. Isto significa que identidades não devem ser mantidas numa única entidade terceira, mesmo que esta seja de confiança. Identidades transportáveis asseguram que o utilizador tem controlo sobre a sua identidade a qualquer altura e, consequentemente, melhoram também a persistência das identidades.

7. **Interoperabilidade**

   Identidades devem ser usadas da forma mais abrangente possível. Um dos principais objetivos dos sistemas de identidades digitais é garantir que a informação presente nas identidades seja reconhecida a nível global sem que os utilizadores percam o controlo dessas identidades.

8. **Consentimento**

   Utilizadores deve concordar com o uso das suas identidades por terceiros. Apesar dos sistemas de identidades serem construídos para partilhar identidades e, em particular, as respetivas *claims*, esta partilha de informações apenas pode ocorrer com o consentimento do utilizador. De notar que este consentimento não precisa necessariamente de ser interativo mas deve estar sempre bem definido.

9. **Minimização**

   Divulgação de *claims* deve ser minimizada. Ao divulgar informação presente numa credencial, essa divulgação deve conter apenas a quantidade mínima de dados necessários para completar a tarefa. Por exemplo, se um utilizador quiser adquirir a carta de condução, não é necessário fornecer a sua idade exata desde que consiga provar que a sua idade é superior ao mínimo requerido. Entre as diversas técnicas criptográficas existentes para alcançar este objetivo destacam-se divulgação seletiva (i.e. revelar apenas a fotografia na apresentação de uma credencial que representa o cartão de cidadão) e prova de conhecimento zero (i.e. provar maioridade sem revelar a idade em concreto).

10. **Proteção**

    Os direitos dos utilizadores devem ser protegidos. Quando há conflitos entre as necessidades da rede de identidades e os direitos dos utilizadores individuais, a rede deve sempre proteger a liberdade e os direitos dos utilizadores.

2.2.2 *Identificadores descentralizados*

A viabilidade de um sistema de gestão de identidades descentralizado depende do uso de identificadores descentralizados (DIDs) que permitam identificar unicamente e de forma descentralizada os respetivos utilizadores do sistema. Estes identificadores descentralizados procuram geralmente, mas nem sempre, garantir a privacidade dos utilizadores. O facto destes identificadores descentralizados poderem ser criados de acordo com a necessidade dos utilizadores faz com que seja possível isolar diferentes credenciais, e os seus respetivos atributos, de modo a dificultar a correlação de dados.

Evolução dos identificadores únicos

Produzir identificadores únicos tem sido uma necessidade básica desde o aparecimento da *world wide web*. Para servir esta necessidade, surgiu inicialmente o *Universally Unique Identifier* - UUID [25]. As primeiras versões do UUID dependiam de uma autoridade de registo central (Institute of Electrical and Electronics Engineers [26]) para a atribuição de

identificadores únicos, tipicamente recorrendo aos endereços MAC dos aparelhos eletrónicos dos utilizadores. Por sua vez, a versão 4 do UUID eliminou a necessidade de um registo central para a criação de identificadores únicos através geração e utilização de números pseudo-aleatórios como identificadores únicos. Apesar de não ser necessário um registo central para validar a criação de novos identificadores únicos, este continuava a ser necessário para validar a ligação entre o identificador único e a entidade a ele associada pelo que esta solução ainda não conseguiu ser completamente descentralizada. Mais tarde surgiu a criptografia de chave pública. Ao contrário de um UUID, um par de chaves pública/-privada elimina a necessidade de verificação de terceiros uma vez que consegue efetuar a autenticação, assumindo a correta ligação entre a chave pública e o respetivo detentor dessa chave. Contudo, nestes esquemas o mapeamento entre a chave pública e informação legível ao utilizador com as informações do detentor da chave é feito através de uma entidade de certificação centralizada que apresenta elevado risco caso seja comprometida.

Com o surgimento da DLT (*Distributed Ledger Technology*) tornou-se possível combinar identificadores únicos com a infraestrutura de chave pública (PKI) de forma a permitir o uso de identificadores completamente descentralizados. Isto deve-se às caraterísticas da tecnologia DLT, que permitem criar bases de dados imutáveis, cujas informações publicadas no registo não podem ser apagadas ou alteradas, com total transparência dos dados armazenados a todos os participantes da rede, e com uma estrutura descentralizada na qual todos os participantes possuem uma cópia da base de dados. Devido a estas caraterísticas, a tecnologia DLT permite gerar bases de dados que atuam como fonte de confiança para o mapeamento entre identificadores (descentralizados) e chaves públicas. Aplicando o mesmo conceito de uma infraestrutura de chave pública, se o utilizador demonstrar ter conhecimento da chave privada associada à chave pública consegue provar que é o dono do identificador descentralizado ligado a essa chave pública na base de dados descentralizada (i.e. blockchain). A secção 2.4.2 explica com mais detalhe o funcionamento de uma infraestrutura de chave pública baseada em tecnologia DLT.

Identificadores descentralizados

Todo o funcionamento de um sistema de gestão de identidades descentralizado depende de identificadores descentralizados (DIDs) sobre os quais os atributos das várias entidades são emitidos. Deste modo, cada utilizador gera e controla o seu próprio DID e, consequentemente, controla todos os atributos associados a esse mesmo identificador. Um DID deve ser persistente e imutável, isto é, vinculado exclusiva e permanentemente ao seu único sujeito. Mesmo após o DID ser desativado, nunca poderá ser reaproveitado.

A comunidade W3C define o termo **identificador descentralizado** [27] como sendo "um novo tipo de identificador para identidades digitais verificáveis". Estes identificadores "estão totalmente sobre o controlo do sujeito", entidade sobre a qual estes são gerados, e são

"independentes de qualquer registo centralizado, fornecedor de identidades ou autoridade de certificação". Segundo o W3C, "DIDs são URIs que se relacionam com um sujeito" e a sua principal função é servir de meio para estabelecer "interações de confiança com esse sujeito". Para tal é necessário utilizar uma componente que sirva como base de confiança para estabelecer uma associação entre o identificador descentralizado e um documento que contenha as várias informações importantes sobre a entidade à qual esse DID pertence (i.e. métodos de verificação de autenticidade como chaves públicas). Na subsecção 2.4.2 é explicado com detalhe o mecanismo de vinculação entre uma entidade e um identificador descentralizado, nomeadamente como esta entidade demonstra ter controlo sobre esse respetivo DID.

Para efetuar a vinculação de certos atributos a um DID, o emissor de credenciais deve introduzir esse DID na credencial. Deste modo qualquer verificador terá acesso ao DID e poderá consultar um registo descentralizado de modo a obter a chave pública associada a esse DID. Para verificar que o indivíduo que forneceu a credencial é de facto o dono da mesma, é apenas necessário requisitar-lhe uma prova de conhecimento da chave privada associada a chave pública adquirida. Se o indivíduo não conseguir gerar tal prova, não conseguirá provar que os atributos da credencial lhe pertencem. Um aspeto importante que se pode retirar desta arquitetura é que os atributos são emitidos sobre os DIDs e não sobre os indivíduos. A prova de posse dos atributos depende única e exclusivamente da prova de posse dos DIDs sobre os quais esses atributos foram emitidos. Por esta razão, o armazenamento de chaves privadas é de extrema importância.

Seguindo os princípios de *privacy by design* [28], cada entidade pode ter tantos DIDs quantos necessários, de modo a permitir a separação desejada de identidades, personas, e contextos e, consequentemente, diminuir a possibilidade de correlação de atributos. Adicionalmente, DIDs de diferentes sistemas podem não ser interoperacionais e, consequentemente, as entidades podem precisar de vários DIDs para oferecer suporte a diferentes relacionamentos. As entidades podem ainda precisar de vários DIDs para oferecer suporte aos diferentes esquemas criptográficos de diferentes métodos de DIDs. Métodos de DIDs são definições de como esquemas de DIDs específicos podem ser implementados num registo distribuído. Estes métodos são explicados com mais detalhe na subsecção 2.2.3.

Por fim, DIDs podem ser divididos em duas classes [8]: DIDs públicos e DIDs de emparelhamento. DIDs públicos são identificadores com os quais os utilizadores se querem ligar publicamente e que contêm alguma informação pessoal sobre os mesmos (i.e. uma fotografia ou uma biografia). Contudo, como foi referido anteriormente, ligar todas as relações ao mesmo identificador pode levar a quebras de privacidade através da correlação dos dados. Deste modo, os DIDs de emparelhamento são gerados sempre que os utilizadores querem isolar interações com outras entidades, pelo que devem ser o principal tipo de DIDs utilizados pelo utilizador comum.

Propriedades dos identificadores descentralizados

Por forma a definir os objetivos que os identificadores descentralizados devem procurar atingir, a comunidade W3C apresentou um conjunto de propriedades que estes devem seguir [27]:

- **Descentralização:** Eliminar a necessidade de autoridades centralizadas ou pontos únicos de falha na gestão de identificadores, permitindo o registo de identificadores únicos a nível global, verificação de chaves públicas, entre outros;
- **Controlo:** Dar às entidades mecanismos para controlarem diretamente os seus identificadores descentralizados sem necessidade de confiarem em entidades externas;
- **Privacidade:** Permitir às entidades controlar a privacidade da sua informação através de divulgação seletiva e progressiva de atributos da sua identidade;
- **Segurança:** Garantir segurança suficiente para que terceiros possam confiar nos documentos dos DIDs e nas informações que estes armazenam. A subsecção 2.2.3 procura explicar com mais detalhe o conceito de documentos dos DIDs bem como as informações que estes armazenam;
- **Baseado em provas:** Permitir que o sujeito do DID forneça provas criptográficas em interações com outras entidades (i.e. para provar posse de um conjunto de atributos);
- **Descoberta:** Possibilitar que uma entidade obtenha mais informações sobre outras entidades através da resolução dos DIDs dessas entidades. Resolução do DID representa o processo de obtenção do documento associado a esse DID e que está armazenado num registo que atua como fonte de confiança. Por outras palavras, através do DID de uma entidade deve ser possível obter o documento associado a esse DID que contém mais informações sobre a entidade em causa.
- **Interoperabilidade:** Usar padrões de forma a possibilitar o uso de ferramentas e bibliotecas projetadas para fornecer interoperabilidade.
- **Portabilidade:** Ser independente do sistema para permitir o seu uso em qualquer sistema que suporte DIDs.
- **Simplicidade:** Favorecer um conjunto reduzido de caraterísticas simples de modo a que a tecnologia seja fácil de entender e implementar.
- **Extensibilidade:** Permitir extensibilidade tendo em consideração o seu impacto na interoperabilidade, portabilidade e simplicidade.

Sintaxe padrão para representação de DIDs

Um identificador descentralizado pode ser separado em 3 partes: esquema (*scheme*), método (*method*) e especificação do método (*method specification*) como mostra a figura 1.

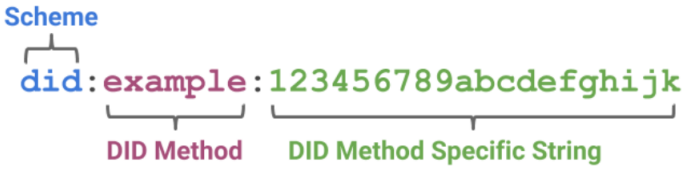


Figura 1: Estrutura base de um DID (SelfKey [29])

O esquema especifica a estrutura do DID. O método refere-se à plataforma em particular que fornece as funcionalidades do DID, definindo um conjunto de operações (i.e. criar, remover, atualizar e apagar). Todos os métodos dos DIDs devem seguir as especificações definidas pelo W3C [27]. A subsecção 2.2.3 aborda com mais detalhe os vários métodos de um DID. Por fim a especificação do método é o documento do DID em si. Com estes três campos do DID é possível localizar e obter o seu respetivo documento, como é explicado de seguida na subsecção *Resolver universal de DIDs*.

Na figura 2 é possível observar com mais detalhe a sintaxe padrão de um identificador descentralizado seguindo o padrão definido pelo W3C [27]. Segundo este padrão, o esquema deve ser escrito em minúsculas, o método deve conter apenas carateres alfanuméricos e deve ser escrito em minúsculas, e a especificação do método deve conter apenas carateres alfanuméricos com a adição de alguns carateres especiais, entre os quais ":", ".", -", e "_".

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

Figura 2: Sintaxe de um DID [27]

Resolver universal de DIDs

O DID por si só é apenas um URI tal como foi mencionado anteriormente. Este URI especifica o local no qual é possível encontrar o respetivo documento do DID com as várias informações públicas afetas ao mesmo. Para localizar estes documentos através dos DIDs são usados *resolvers* universais, que recebem o identificador descentralizado e retornam o documento correspondente. Este tipo de documentos serão explorados na subsecção, 2.2.3.

Para obter o documento, um *resolver* localiza o sistema descentralizado que o contém, através da *driver* correspondente a esse sistema. O campo do método do DID indica ao

resolver que *driver* deve utilizar. Por sua vez, o campo correspondente à identificação do método refere-se ao identificador do documento nesse mesmo sistema descentralizado. Utilizando o DID presente na figura 1, um *resolver* identifica que o esquema usado é o de um identificador descentralizado, obtém o método com o valor "example", que deverá usar para procurar na sua lista de *drivers* a qual delas se deverá conectar, e por fim utiliza a especificação do método (o identificador atribuído ao documento armazenado no registo em questão) para obter o respetivo documento. A Figura 3 mostra o processo iterativo no qual a Alice fornece um DID a um *resolver*, que lhe retorna o documento correspondente a esse DID.

Figura 3: Resolução de um DID (Adaptado de Microsoft [8])

2.2.3 *Documentos de identificadores descentralizados*

Utilizando a definição do W3C [27], "um documento de um DID é um recurso que está associado a um identificador descentralizado (DID)". Estes documentos contêm normalmente métodos de verificação (i.e. chaves públicas), informações sobre o esquema usado (campo **context**), identificação da entidade à qual o documento diz respeito (campo **subject**), entre outros. A subsecção *Propriedades padrão em documentos de DIDs* apresenta as várias propriedades que podem ser inseridas num documento de um DID. Na subsecção *Sintaxe de Documentos de DIDs* é indicada a sintaxe que deve ser usada para a serialização de documentos de DIDs. A subsecção seguinte, *Métodos de DIDs*, retrata as operações disponíveis para a atualização de documentos. Por fim, a subsecção *Extensibilidade em documentos de DIDs* explica como é possível estender um documento através de JSON-LD.

Propriedades padrão em documentos de DIDs

Tendo como objetivo normalizar as propriedades presentes no documento de um DID o W3C definiu um conjunto de padrões que devem ser seguidos [27]. As propriedades apresentadas de seguida são algumas das propriedades mais comuns, contudo é possível introduzir novas propriedades. É importante ter em consideração que os documentos dos DIDs são interpretados por agentes pelo que estes podem não ser capazes de interpretar algumas das propriedades presentes no documento. Deste modo, com a exceção de casos muito específicos é recomendado seguir os padrões definidos o mais precisamente possível para garantir a interoperabilidade entre diferentes agentes.

- **Contexto** (obrigatório)

Quando dois sistemas comunicam entre si, devem usar a mesma terminologia e um protocolo que ambos percebam. Deste modo, os documentos usam a propriedade *@context* para indicar o esquema que estão a seguir de modo a que dois sistemas que operem com esse documento usem a mesma terminologia. O valor desta propriedade deve ser uma lista de URIs ordenada, onde o primeiro URI é "https://w3id.org/did/v1".

- **Sujeito** (obrigatório)

  O sujeito do documento de um DID representa a entidade que controla o DID sobre o qual o documento. Esta propriedade é representada por *id* e o seu valor corresponde ao DID sobre o qual o documento foi emitido. Cada documento deve conter exatamente um sujeito.

- **Chave pública** (opcional)

  A chave pública é usada para efetuar operações criptográficas como assinaturas digitais e cifra de dados, e estabelecer relações seguras entre as entidades. Estas chaves públicas podem ser usadas em mecanismos de autorização das operações CRUD em DIDs. Para representar o conjunto de chaves públicas de autorização é utilizada a propriedade *publicKey*. Autorização é o mecanismo usado para declarar como devem ser efetuadas operações em nome do sujeito do DID. Autorização permite que outras entidades atualizem o documento de um DID (i.e. para recuperação de chaves) sem lhes fornecer autenticação e, consequentemente, impedir a personificação do dono do DID. Apenas quem conhece as chaves públicas apresentadas na propriedade *authentication* (propriedade seguinte) consegue provar a sua autenticação quanto a esse DID. Cada método deve definir como é que a autorização e delegação são implementadas, incluindo quaisquer operações criptográficas necessárias. Se não for encontrada uma chave pública no documento do DID deve-se assumir que a chave foi revogada ou não é válida. Um documento que contenha uma chave revogada deve conter ou referir um local onde se pode encontrar informação sobre a razão de revogação dessa chave.

  A propriedade *publicKey* corresponde a uma lista de objetos que representam chaves públicas. Cada chave pública deve incluir as propriedades *id*, *type* e *controller*, podendo adicionalmente conter outras informações. De notar que:

  - A lista de chaves públicas não deve conter entradas duplicadas com o mesmo identificador.
  - O valor da propriedade *type* deve ser do tipo chave pública, seguindo os formatos definidos em *Linked Data Cryptographic Suite Registry* [30].

  - O valor da propriedade *controller* deve indicar o DID da entidade que possui permissão para alterar o documento.

O excerto de código apresentado na figura 4 mostra um exemplo de possíveis tipos de chaves públicas.

```
"publicKey": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }, {
    "id": "did:example:123456789abcdefghi#keys-2",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:pqrstuvwxyz0987654321",
    "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }, {
    "id": "did:example:123456789abcdefghi#keys-3",
    "type": "Secp256k1VerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyHex": "02b97c30de767f084ce308...5d7116a3263d29f1450936b71"
}]
```

Figura 4: Exemplo do valor da propriedade *publicKey* [27]

- **Autenticação** (opcional)

  Autenticação é o meio através do qual o dono de um DID pode provar criptograficamente que está associado a esse DID. Para representar a autenticação deve-se usar a propriedade *authentication* cujo valor é uma lista de métodos de verificação. Cada método de verificação pode ser incorporado ou referenciado.

  A figura 5 mostra duas formas possíveis de indicar métodos de verificação: incorporado ou referenciado. Caso a chave pública seja usada noutras propriedades (i.e. na propriedade *publicKey*), basta apenas referenciá-la. Caso a chave pública seja apenas utilizada para autenticação é necessário incorporá-la.

```
"authentication": [
  // chave úpblica referenciada
  "did:example:123456789abcdefghi#keys-1",
  // chave úpblica incorporada
  {
    "id": "did:example:123456789abcdefghi#keys-2",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }
]
```

Figura 5: Exemplo do valor da propriedade *authentication* [27]

- **Created** (Opcional)

  O documento de um DID deve apresentar no máximo uma propriedade *created* que representa o timestamp da sua criação, cujo valor deve ser um XML de data-tempo válido.

```
  "created": "2002-10-10T17:00:00Z"
```

Figura 6: Exemplo do valor da propriedade *created* [27]

- **Updated** (Opcional)

  O documento de um DID deve apresentar no máximo uma propriedade *updated* que represente o *timestamp* da última atualização, cujo valor deve seguir o mesmo formato da propriedade *created*.

- ***Service Endpoints*** (opcional)

  Um *service endpoint* pode representar qualquer tipo de serviço que o sujeito deseja anunciar (i.e. gestão descentralizada de identidades, autenticação, autorização ou interação). A propriedade *service* é responsável por representar todos os serviços disponibilizados pelo DID e o seu valor deve ser uma lista de objetos do tipo *service endpoint*. Cada objeto deve incluir as propriedades *id*, *type* e *serviceEndpoint*, podendo incluir propriedades adicionais. O valor da propriedade "serviceEndpoint"deve ser um objeto JSON-LD [31] ou um URI válido. A figura 7 apresenta vários tipos de serviços disponibilizados pelo DID *did:example:123456789abcdefghi*.

```
"service": [{
    "id": "did:example:123456789abcdefghi#openid",
    "type": "OpenIdConnectVersion1.0Service",
    "serviceEndpoint": "https://openid.example.com/"
  }, {
    "id": "did:example:123456789abcdefghi#hub",
    "type": "IdentityHub",
    "publicKey": "did:example:123456789abcdefghi#key-1",
    "serviceEndpoint": {
      "@context": "https://schema.identity.foundation/hub",
      "type": "UserHubEndpoint",
      "instances": ["did:example:456", "did:example:789"]
    }
  }, {
    "id": "did:example:123456789abcdefghi#messages",
    "type": "MessagingService",
    "serviceEndpoint": "https://example.com/messages/8377464"
  }, {
    "id": "did:example:123456789abcdefghi#inbox",
    "type": "SocialWebInboxService",
    "serviceEndpoint": "https://social.example.com/83hfh37dj",
    "description": "My public social inbox",
    "spamCost": {
      "amount": "0.50",
      "currency": "USD"
    }
}]
```

Figura 7: Exemplo do valor da propriedade *service* [27]

- **Prova** (Opcional)

  O Documento de um DID pode apresentar uma prova criptográfica para comprovar a sua integridade. Essa prova, se existir, deve estar representada na propriedade *proof*. A figura 8 mostra um exemplo de uma prova.

```
"proof": {
  "type": "LinkedDataSignature2015",
  "created": "2016-02-08T16:02:20Z",
  "creator": "did:example:8uQhQMGzWxR8vw5P3UWH1ja#keys-1",
  "signatureValue": "QNB13Y7Q9...1tzjn4w=="
}
```

Figura 8: Exemplo do valor da propriedade *proof* [27]

Em anexo, na subsecção A.1, é apresentado um documento completo de um DID com os vários campos abordados acima.

Sintaxe de Documentos de DIDs

O documento de um DID deve ser um único objeto JSON e deve respeitar as normas definidas para a representação dos vários tipos de dados:

- Valores numéricos devem ser representados com o tipo **Number**;
- Valores booleanos devem ser representados com o tipo **Boolean**;
- Sequências de valores devem ser representados com o tipo **Array**;
- Conjuntos de valores não ordenados devem ser representados com o tipo **Array**;
- Conjuntos de propriedades devem ser representados com o tipo **Object**;
- Valores vazios devem ser representados com o tipo **null**;
- Outros valores devem ser representados com o tipo **String**.

Documentos de DIDs recorrem ainda ao formato JSON-LD [31], um formato baseado em JSON que é usado para serializar dados ligados (*Linked Data*). Este formato é útil quando se pretende estender os dados do documento de um DID, como é apresentado duas subsecções abaixo, em "Extensibilidade em documentos de DIDs".

Métodos de DIDs

Os métodos de um DID são mecanismos através dos quais um DID e o seu respetivo documento são criados, lidos, atualizados, e desativados numa rede ou registo distribuído. Este conjunto de mecanismos permite que o dono tenha controlo sobre o seu DID.

Para permitir total funcionalidade dos DIDs e dos respetivos documentos, a especificação de métodos deve especificar como cada uma das operações **CRUD** é executada por um cliente. Estas operações podem ser usadas para executar todas as ações requeridas num sistema de gestão de chaves criptográficas (e.g. registo, substituição, rotação, recuperação, expiração).

- **Create** deve especificar como um cliente cria uma DID e o seu respetivo documento no registo público, incluindo todas as operações criptográficas necessárias para estabelecer prova de controlo.

- **Read/Verify** deve especificar como um cliente usa uma DID para pedir um documento DID do registo público, incluindo como o cliente pode verificar a autenticidade da resposta.
- **Update** deve especificar como um cliente pode atualizar um documento DID no registo público incluindo todas as operações criptográficas necessárias para estabelecer prova de controlo, ou indicar que *updates* não são possíveis.
- **Deactivate** deve especificar como um cliente pode desativar uma DID no registo público, incluindo todas as operações criptográficas necessárias para estabelecer prova de desativação, ou indicar que *deactivation* não é possível.

Ao criar uma nova especificação para um método de um DID deve-se usar um nome para o método que seja único entre todos os nomes de métodos conhecidos aquando ao momento da publicação. Como não existe uma autoridade central, não há maneira de saber se o nome de um método em particular é único. O registo de métodos de DIDs [32] é uma ferramenta utilizada para atingir um consenso face à nomeação de um novo método. Apesar desta ferramenta não ser uma lista oficial, é aconselhável adicionar os novos nomes de métodos a esta lista para que outros membros da comunidade tenham um espaço para ver todos os métodos DID existentes.

Extensibilidade em documentos de DIDs

É possível fornecer mecanismos de extensibilidade através do uso de **JSON-LD**[31] (*JSON Linked Data*). *Linked Data* refere-se a dados estruturados que estão interligados com outros dados. Esta abordagem é tipicamente designada por suposição de mundo aberto, ou seja, qualquer pessoa pode dizer qualquer coisa sobre quaisquer outras coisas. O artigo "On JSON-LD and the Semantics of Identity"[33] informa da importância dos dados ligados e do uso JSON-LD para a criação de padrões de representação de informação em oposição à utilização de esquemas unicamente JSON.

O software escrito para um sistema que permite extensibilidade terá de determinar se o documento de um DID com extensões é aceitável ou não com base no perfil de risco da aplicação. Algumas aplicações podem optar por aceitar mas ignorar extensões, outras podem optar por apenas aceitar certas extensões, enquanto que ambientes mais seguros podem optar por não permitir quaisquer extensões.

De seguida é descrito o processo incremental para adicionar extensibilidade seguindo o exemplo apresentado em [34]. Inicialmente temos um documento simples, apresentado na figura 9. Dado que as propriedades *publickey*, *authentication* e *service* não são importantes neste contexto serão omitidas por forma a simplificar o exemplo.

```
{
  "@context": "https://example.org/example-method/v1",
  "id": "did:example:123456789abcdefghi",
  "publicKey": [{...}],
  "authentication": [{...}],
  "service": [{...}]
}
```

Figura 9: Exemplo de um documento simples [27]

Pretende-se estender o documento apresentado para retratar informação adicional, neste caso uma *stream* de fotos públicas do sujeito. Para tal cria-se um **JSON-LD** contendo o novo termo, como é possível observar na figura 10.

```
{
  "@context": {
    "PhotoStreamService": "https://example.com/vocab#PhotoStreamService"
  }
}
```

Figura 10: Exemplo de um documento JSON-LD [27]

Este JSON-LD deve ser posteriormente publicado em algum sítio (unicamente identificável) que seja acessível a qualquer processador de documentos de DIDs. Neste exemplo o documento foi colocado no URI *did:example:contexts:987654321*. De seguida, basta adicionar a nova propriedade aos serviços do documento original do DID, como mostra a figura 11.

```
{
  "@context": "https://example.org/example-method/v1",
  "id": "did:example:123456789abcdefghi",
  "publicKey": [{...}],
  "authentication": [{...}],
  "service": [{
    "@context": "did:example:contexts:987654321",
    "id": "did:example:123456789abcdefghi#photos",
    "type": "PhotoStreamService",
    "serviceEndpoint": "https://example.org/photos/379283"
  }, ...]
}
```

Figura 11: Documento do DID alterado [27]

Através do serviço adicionado qualquer pessoa pode consultar o URI *did:example:contexts:987654321* e obter a estrutura de dados usada mesmo que essa informação não esteja representada no documento em si.

2.3 ARQUITETURA SELF-SOVEREIGN IDENTITY

Contrariamente aos sistemas de gestão de identidades tradicionais, nos quais o fornecedor do serviço se encontra no centro do modelo (i.e. facebook, google, etc), os modelos de *Self-Sovereign-Identity* colocam o utilizador no seu centro. Apesar de serem vários os modelos de SSI apresentados até à data, todos eles seguem uma arquitetura semelhante à apresentada na Figura 12. Na maior parte destes modelos a *blockchain* efetua o papel de *data registry*, isto é, garantir a integridade e a autenticidade da informação. Contudo, em alguns modelos a componente *blockchain* é substituída por outra componente que desempenha o mesmo papel. Neste documento a componente utilizada para implementar o *data registry* na solução final será a *blockchain* pelo que será dada maior atenção a esta tecnologia.

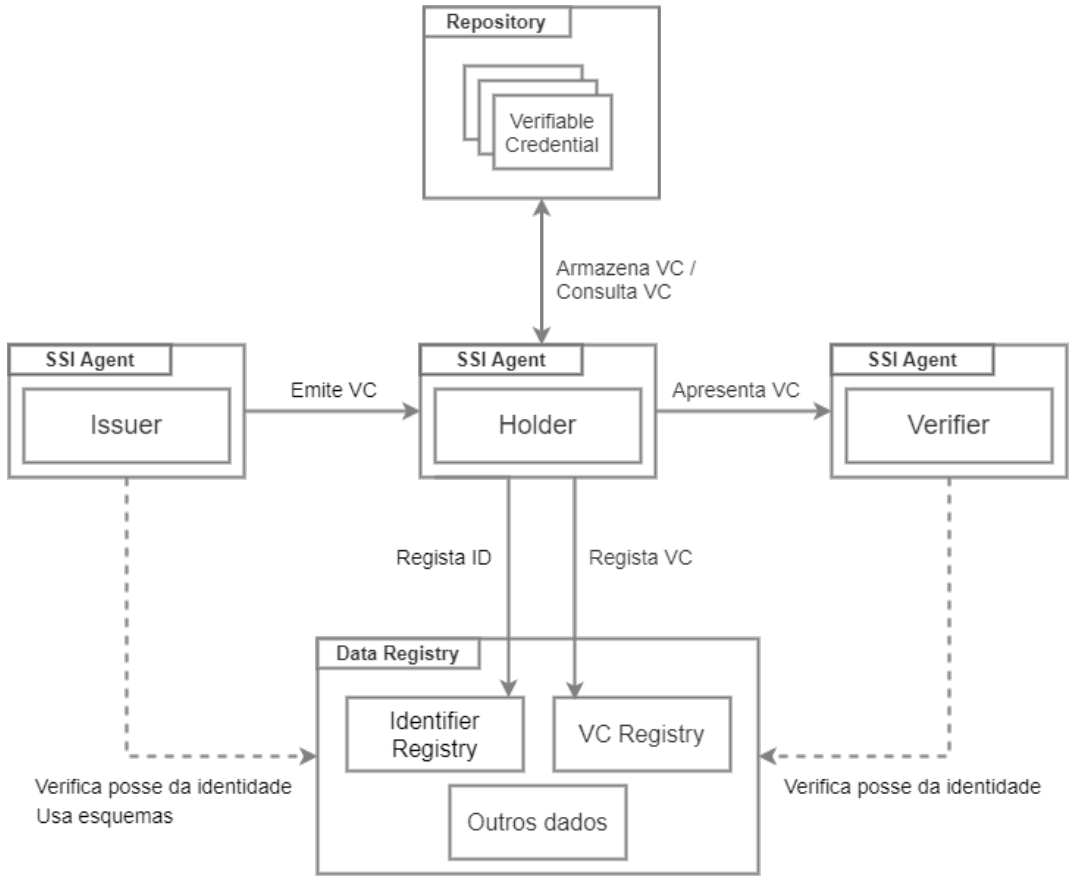


Figura 12: Arquitetura SSI (adaptado de Hasso Plattner Institute [25])

2.3.1 *Ecossistema*

O armazenamento de dados em *Self-Sovereign Identity* está diretamente dependente da privacidade desejada nos mesmos. Em certos casos, é desejável que a informação seja

guardada de forma pública (i.e. chaves públicas ou perfis sociais que queiram garantir a sua autenticidade ao público em geral). Nestes casos, a solução mais viável é armazenar estas informações num registo público, *data registry*, que seja simultaneamente inviolável e descentralizado. Contudo, na maioria dos casos o utilizador não quer revelar informações sobre a sua identidade pelo que a sua privacidade deve ser preservada. Deste modo, as informações de caráter privado devem ser armazenadas *off-chain*, num local escolhido pelo utilizador - *repository*.

Na Figura 12 podemos observar as três principais entidades do ecossistema: *Issuer*, *Holder*, e *Verifier*. É possível observar também um local onde as *verifiable credentials* são armazenadas, *repository*, e o registo descentralizado responsável por gerir e garantir a autenticidade e integridade da informação pública, *data registry*. A arquitetura apresentada mostra ainda um *SSI Agent*, cuja função é ajudar o utilizador a interagir com o sistema. Abaixo é apresentada a descrição de cada uma das componentes presentes numa arquitetura SSI:

- **Holder:** Entidade que possui uma ou mais *verifiable credentials* e que possui a capacidade de gerar *verifiable presentations* através da junção dos atributos das suas credenciais. Exemplos desta entidade incluem alunos, empregados, clientes, entre outros.
- **SSI Agent:** Programa que atua como intermediário na comunicação entre *holders*, *issuers* e *verifiers*, devendo sempre atuar de acordo com a vontade do seu controlador. Por defeito todas as entidades do sistema recorrem a agentes para interagirem entre si e com o sistema.
- **Subject:** Entidade ou objeto sobre o qual as *claims* são emitidas. Na maioria dos casos o sujeito é o *holder* contudo, em alguns casos tal não se verifica. Por exemplo, o dono de um animal de estimação (*holder*) possui as *verifiable credentials* emitidas ao seu animal (*subject*).
- **Issuer:** Entidade responsável por atribuir *claims* a um ou mais *subjects*, agrupar essas *claims* numa *verifiable credential* e transmiti-la a um **Holder**. São exemplos desta entidade corporações, governos, entre outros.
- **Verifier:** Entidade que verifica uma ou mais *verifiable credentials*, opcionalmente dentro de *verifiable presentations*. Se a *verifiable credencial* não for incluída dentro de uma apresentação, esta atua como *verifiable presentation*. São exemplos desta entidade fornecedores de serviços, websites, entre outros.
- **Repository:** Local escolhido pelo *holder* para armazenar as suas *verifiable credentials*. Esta componente é também muitas vezes designada por *storage*.
- **Data Registry:** Sistema que fornece meios de criação e verificação de identificadores, chaves, e outros dados relevantes, tais como esquemas de *verifiable credentials*. O

data registry é ainda responsável por estabelecer a identificação e autenticação do *holder*. São exemplos destes sistemas bases de dados confiáveis, bases de dados descentralizadas, bases de dados governamentais de IDs, etc. A subsecção 2.4.2 explica detalhadamente o mecanismo de autenticação do *holder* numa infraestrutura de chave pública descentralizada através do estabelecimento de uma ligação entre o seu identificador único (DID) e a sua chave pública com o auxilio de uma *blockchain*.

Nesta arquitetura a *verifiable credential* é totalmente controlada pelo *Holder*, através de um *Holder Agent*, sendo o *Holder* responsável por armazenar as suas *Verifiable Credentials* num *repository* à sua escolha.

Quanto à componente *data registry*, a tecnologia *blockchain* demonstra ser uma opção bastante viável uma vez que apresenta todas as caraterísticas necessárias, isto é, é inviolável, descentralizada e mantém o registo de todas as alterações efetuadas ao longo do tempo desde a sua criação. Esta tecnologia descentralizada é apresentada com mais detalhe na secção 2.4, incluindo as suas vantagens para o caso de uso *self-sovereign identity* e possíveis problemas.

Nesta arquitetura, a componente *data registry* não se limita a uma única solução, podendo ser integrados vários registos, cada um com o seu próprio espaço de nomes e identificadores. Deste modo, é possível utilizar várias tecnologias descentralizadas para atuarem como registos confiáveis no mesmo sistema, e mapear os identificadores descentralizados para o seu respetivo *registry* através de *resolvers* universais. Como foi mencionado na secção 2.2.2, um *resolver* é responsável por obter documentos dos identificadores descentralizados, que se encontram armazenados nos *data registries*, através do respetivo DID. É importante mencionar que um *resolver* apenas consegue adquirir documentos de registos cujas *drivers* foram adicionadas a esse *resolver*.

2.3.2 *Modelo de confiança numa arquitetura SSI*

Tento em conta a arquitetura apresentada na figura 12, verifica-se que existem várias relações que requerem confiança:

- o *verifier* confia no *issuer* que emitiu as *verifiable credentials*. Para estabelecer esta confiança, o *issuer* deve incluir uma prova que garanta que foi ele quem gerou a credencial. Essa confiança pode ser enfraquecida dependendo da avaliação de risco do *verifier*.
- o *issuer*, o *holder* e o *verifier* confiam que o *registry* não pode ser corrompido e que pode ser usado para armazenamento de informações públicas importantes como o mapeamento entre identificadores e respetivas entidades do mundo físico, esquemas de credenciais, entre outros.

- o *holder* confia no *issuer* para emitir credenciais verdadeiras e para as revogar rapidamente assim que lhe for requisitado.
- O *holder* confia no *repository* para armazenar as credenciais em segurança, para não as fornecer a ninguém além dele próprio e para não as corromper ou perder.

Como consequência, o *issuer* e o *verifier* não precisam de confiar no *repository* que o *holder* utiliza para armazenar as credenciais e o *issuer* não precisa de conhecer ou confiar no *verifier*.

De acordo com as relações de confiança necessárias que foram apresentadas, todas elas são facilmente estabelecidas. Quanto à primeira relação de confiança, o *verifier* pode possuir uma lista de *issuers* nos quais confia e aceitar apenas credenciais emitidas por eles. Já na segunda relação de confiança é visível que o *registry* desempenha um papel fundamental no funcionamento de todo o sistema através mapeamento entre identificadores e as respetivas entidades, importante para efetuar a autenticação das entidades, bem como outras informações públicas como esquemas, algumas credenciais de caráter público (i.e. uma credencial de uma marca de roupa com o objetivo de dar a conhecer mais informações públicas sobre si), entre vários outros dados. Apesar de todas as entidades necessitarem de depositar a sua confiança neste registo, tal não apresenta qualquer problema na implementação da solução dado que existem várias tecnologias de caráter descentralizado que podem ser usadas para implementar esta componente, como é o caso da tecnologia *blockchain*. Relativamente à terceira relação de confiança, caso o *holder* verifique que alguma informação não está correta, o *issuer* tem a obrigação de corrigir tal informação, podendo ser processado caso tente prejudicar o *holder*. Por fim, relativamente à quarta relação de confiança, o *holder* é responsável por escolher o repositório no qual pretende armazenar as suas credenciais, podendo armazenar as várias credenciais em múltiplos repositórios. Caso opte por um serviço especializado no armazenamento de credenciais fora do seu controle, este serviço deverá sempre agir tendo em conta os melhores interesses do *holder*, podendo enfrentar repercussões jurídicas caso não o faça. Como o proprietário do repositório pode tentar correlacionar dados do *holder* através das suas múltiplas credenciais, é de extrema importância que o *holder* confie nos repositórios que escolheu, que divida as suas credenciais por múltiplos repositórios para evitar perda de credenciais, e que possua vários identificadores descentralizados de modo a obter um melhor isolamento das suas credenciais e mitigar assim alguns problemas de correlação de dados.

Após a análise efetuada é possível afirmar que apesar de ser necessário estabelecer certas relações de confiança, estas são facilmente estabelecidas através do uso da arquitetura de *self-sovereign identity* apresentada anteriormente.

2.3.3 *Verifiable Credentials*

Credenciais são parte do nosso dia-a-dia, seja a carta de condução, o diploma da universidade, o passaporte, ou até mesmo o cartão de membro de um clube. Estas credenciais têm um uso físico valioso mas o seu uso digital ainda é muito abstrato. Atualmente é difícil expressar qualificações educacionais, dados de saúde, detalhes de contas financeiras, entre outros através da rede. Esta dificuldade gera o desafio de fazer com que as credenciais digitais apresentem as mesmas carateristicas e funcionalidades que as credenciais físicas. Esta secção apresenta uma forma padrão de expressar credenciais na rede de uma maneira criptograficamente segura, que respeitem a privacidade e que sejam computacionalmente verificáveis. É contudo necessário ter em atenção a persistência da informação digital e a facilidade com que se podem recolher e correlacionar os dados presentes nessas credenciais, o que pode comprometer a privacidade e segurança das *verifiable credentials*. A secção 2.6 endereça esses problemas.

O que é uma credencial?

No mundo físico, uma credencial consiste em:

- Informação que identifica o sujeito da credencial (foto, nome, número de identificação, entre outros);
- Informação referente à autoridade emissora (governo, entidade certificadora, entre outros);
- Informação referente ao tipo de credencial que representa (Passaporte Português, Carta de condução Americana, entre outros);
- Informação referente a atributos específicos ou propriedades que estão a ser declaradas pela autoridade emissora sobre o sujeito (nacionalidade, classe do veículo que pode conduzir, entre outros);
- Prova referente a como a credencial foi obtida;
- Informação referente às restrições da credencial (data de validade, termos de uso, entre outros).

No mundo digital, uma *credential* é um atestado que atribui um conjunto de atributos (também designados por *claims*) a uma identidade. Esta secção começa por explicar o que é uma *claim*, o que é uma *verifiable credential* e introduz o conceito de *verifiable presentation*, sendo esta última a estrutura que será apresentada ao fornecedor do serviço (*verifier*) e que contém os vários atributos requeridos pelo mesmo para que este os possa validar. Ainda nesta secção serão explicados vários conceitos associados ao uso de *verifiable credential* tais como a delegação de autoridade, a resolução de disputas, a divulgação seletiva de atributos e as estratégias para lidar com os atributos negativos e, por fim, as provas de conhecimento zero.

Claims

Uma *claim* é uma declaração feita sobre um sujeito, podendo este ser uma entidade ou um objeto. Dan Gisolfi define *claim* [35] como sendo "uma declaração feita por um indivíduo ou organização que confirma que a entidade efetuou ações específicas de forma a estabelecer a verdade sobre um atributo de identidade específico".

Uma *verifiable claim* deve conter o sujeito (*subject*) sobre o qual as declarações são efetuadas e metadados das mesmas. Além disso, uma *verifiable claim* deve conter um período de validade, a identidade da entidade que a emitiu (*issuer*), e os algoritmos usados para assinaturas/cifras. De modo a tornar uma *claim* verificável, o *issuer* deve assiná-la com a sua chave privada associada à sua chave pública que publicou em algum local de confiança, seja através de certificados ou através de registos imutáveis como *blockchains*. *Claims* são expressas através de relações **sujeito-propriedade-valor**, como é possível verificar na figura 13. Data de nascimento, altura ou número da segurança social são alguns exemplos de *claims*.

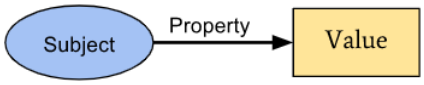


Figura 13: Estrutura básica de uma *claim* (W3C [34])

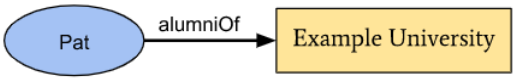


Figura 14: *Claim* que expressa que Pat é um ex-aluno da universidade (W3C [34])

Na Figura 14 é possível observar uma *claim* individual que afirma que Pat se graduou na universidade indicada. *Claims* individuais podem ser conjugadas de maneira a representar um grafo de informação sobre um sujeito. A Figura 15 mostra uma possível extensão à *claim* anterior através da adição das *claims* "Pat conhece Sam" e "Sam é professor".

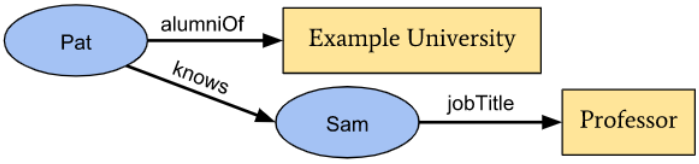


Figura 15: Combinação de várias *claims* (W3C [34])

Verifiable Credentials

Como indicado anteriormente, uma credencial é um conjunto de uma ou mais *claims*, sendo cada uma verificável através da assinatura do *issuer* que a emitiu ou que pode certificar o conteúdo da mesma. Uma *verifiable credential* é um conjunto de *claims* e metadados invioláveis que provam criptograficamente quem os emitiu e pode representar toda a informação que uma credencial física representa. A adição de tecnologias, como assinaturas digitais, torna estas credenciais mais difíceis de violar e, consequentemente, mais confiáveis do que as credenciais físicas. *Verifiable credentials* podem também incluir um identificador e metadados

para descrever as propriedades da credencial, como por exemplo o *issuer*, a data e hora de expiração, uma imagem representativa, uma chave pública para propósitos de verificação, o mecanismo de revogação, etc. Os metadados devem ser assinados pelo *issuer*. É importante referir que um *holder* pode possuir uma credencial que contenha múltiplas *claims* sobre diferentes sujeitos, não sendo necessário que estes estejam relacionados ou que ele esteja incluído nesses sujeitos. Um caso comum é a credencial emitida a um animal de estimação (sujeito), ficando o seu dono (*holder*) responsável por ela.

A figura 16 mostra os componentes básicos de uma *verifiable credential*: os metadados, a lista de *verifiable claims* e a lista de provas que permitem verificar as várias *claims*, enquanto que a figura 17 representa uma visualização mais completa de uma *verifiable credential*, na qual o primeiro grafo contem todas as *claims* e metadados enquanto que o segundo grafo contém a prova dessas informações.

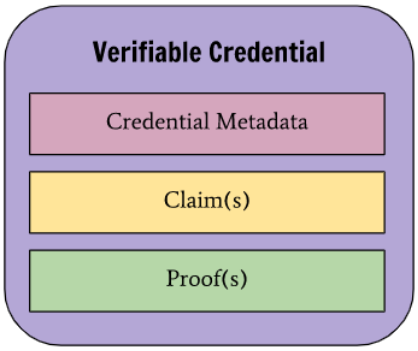


Figura 16: Componentes básicos de uma *verifiable credential* (W3C [34])

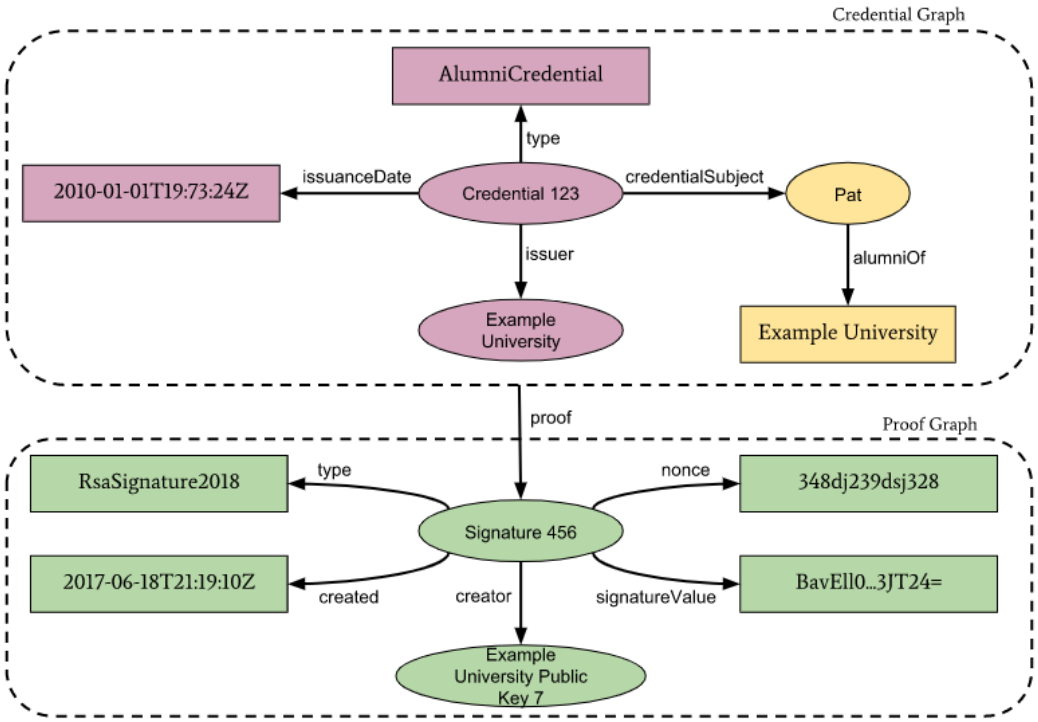


Figura 17: Grafos de informação de uma *verifiable credential* básica (W3C [34])

Verifiable Presentations

Dado que a privacidade é uma necessidade básica de uma arquitetura descentralizada, é importante que as entidades que usem a tecnologia *SSI* possam mostrar apenas as partes da sua *persona* que são necessárias para uma dada situação. A expressão de um subconjunto de atributos de uma dada *verifiable credential* é chamada *verifiable presentation*. Por sua vez, uma *verifiable presentation* pode expressar subconjuntos de atributos de uma ou mais credenciais, agregados de forma a que a autoria dos dados seja verificável. As *verifiable credentials* podem também ser apresentadas diretamente, atuando nesse caso como *verifiable presentations*.

Os *holders*, titulares das *verifiable credentials*, podem gerar *verifiable presentations* e partilhá-las com *verifiers* para provar que possuem certos atributos (*claims*). Os dados presentes numa *verifiable presentation* são normalmente referentes ao mesmo sujeito, mas podem ter sido emitidos por diferentes *issuers*. É contudo possível ter uma *verifiable presentation* que possui múltiplas credenciais sobre diferentes sujeitos sendo que estes não necessitam de estar relacionados.

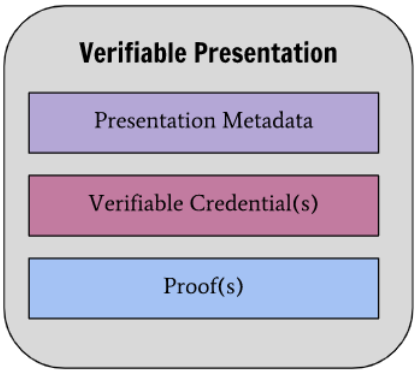


Figura 18: Componentes básicos de uma *verifiable presentation* (W3C [34])

A figura 18 mostra os componentes de uma *verifiable presentation*, contudo abstrai os detalhes sobre como as *verifiable credentials* estão organizadas em grafos de informação. Já a figura 19 mostra uma descrição mais completa de uma *verifiable presentation*, que normalmente é composta por, pelo menos, quatro grafos de informação. O primeiro grafo mostra os metadados da apresentação. A propriedade **verifiable Presentation** refere-se a uma ou mais *verifiable credentials* (cada uma num grafo auto-contido), que contem os metadados e as *claims*. Os últimos dois grafos expressam a prova da credencial e da apresentação, respetivamente.

Figura 19: Grafos de informação de uma *verifiable presentation* básica (W3C [34])

Delegação de autoridade

Verifiable credentials permitem que os intervenientes no sistema transmitam certas permissões a outras entidades. O *issuer* pode dar permissões a outros *issuers* (*issuers* subordinados) para emitirem *verifiable credentials* no seu lugar. Já o *holder* pode transmitir certas credenciais a outros *holders*. Por sua vez, o *verifier* pode fornecer permissões a outras entidades para

autenticarem os *holders*. Esta subsecção irá explorar estes três tipos principais de delegação de autoridade: delegação pelo *issuer*, delegação pelo *holder* e delegação pelo *verifier*.

A estratégia de delegação de autoridade efetuada pelo *issuer* para a emissão de *verifiable credentials* permite-lhe aumentar a escalabilidade do processo de emissão destas credenciais através da adição de mais *issuers*. O *issuer* inicial é designado por *source of authority* (SoA) enquanto que as entidades que recebem a permissão para agir em nome do *issuer* são designadas por *issuers* subordinados. São geralmente usados dois mecanismos de delegação de autoridade:

- **Descendente**

  A SoA publica uma lista de *subordinate issuers* aos quais delegou permissão para emissão de credenciais. Esta lista deve ser acessível a todos os *verifiers* que confiam na SoA e pode ser atualizada a qualquer altura. O *verifier* pode validar qualquer credencial emitida em nome da SoA verificando se o subordinado que assinou a credencial se encontra na lista assinada pela SoA e não se encontra em qualquer lista de revogação.

- **Ascendente**

  Não há necessidade da SoA publicar uma lista de subordinados de confiança. Estes subordinados efetuam todo o processo de aquisição e agregação de dados para a credencial e a SoA assina diretamente essaa credencial. Nesta estratégia o processo de verificação é mais simples pois todas as credenciais são emitidas (e revogadas) pela SoA na qual todos os *verifiers* confiam.

Por sua vez, a delegação de autoridade efetuada pelo *holder* permite que terceiros solicitem uma credencial para si ou em nome do *holder*. Por exemplo, um *holder* pode fornecer um cartão de sócio de um clube a um amigo, permitindo-o a entrar no clube. Neste caso a credencial deverá ser uma credencial de portador, ou seja, pertence a quem a possui. Num segundo exemplo o *holder* fornece a outro indivíduo a autoridade legal de agir em nome do *holder* para assinar um formulário de prova de autoridade. Em ambos os casos, é possível aplicar o conceito de recursividade, onde os novos *holders* podem nomear novos *holders*.

Por fim, a delegação de autoridade efetuada pelo *verifier* permite-lhe atribuir a validação de credenciais a outra entidade de modo a que esta possa facultar o serviço ao *holder* em nome do *verifier*. Se o processo de delegação não for visível ao *holder*, a entidade delegada pode agir em nome do *verifier* sem necessidade de que entidades externas tenham conhecimento, nomeadamente o *holder*.

Resolução de disputas

A informação que um *issuer* mantém sobre um *holder* pode estar incorreta ou desatualizada. Consequentemente todas as credenciais emitidas pelo *issuer* a esse *holder* contêm dados incorretos. Se o *holder* detetar que as suas credenciais apresentam dados incorretos deverá

informar o *issuer* para que estes dados sejam atualizados. O *issuer* deverá permitir que o *holder* atualize as suas informações sempre que for necessário. Contudo o *holder* deverá apresentar evidências de que a informação está de facto incorreta, de que quem está a solicitar a alteração é de facto o *holder* e não um impostor, e de que ele não está a tentar inserir informação falsa ou remover informação verdadeira. Se o *issuer* detetar alguma anormalidade em algum dos requisitos apresentados poderá recusar-se a atualizar a informação. De maneira a respeitar a vontade do dono da informação (*holder*), o *issuer* deverá publicar procedimentos através dos quais o *holder* pode:

- Obter o conjunto completo de informação mantida pelo *issuer* sobre si.
- Requisitar a emissão de uma ou mais *verifiable credentials*, indicando o subconjunto de informação que deve ser emitido nas mesmas.
- Requisitar a revogação de uma *verifiable credential*.
- Requisitar a atualização de informação mantida pelo *issuer*.

Divulgação seletiva e atributos negativos

Outra caraterística importante de *verifiable credentials* é a possibilidade de usar divulgação seletiva. Esta técnica permite aos *holders* incluírem nas *verifiable presentations* apenas subconjuntos dos atributos presentes nas credenciais de forma a melhorar a privacidade dos dados, ou seja, se o *verifier* não requisitar todos os dados presentes na credencial o *issuer* pode selecionar individualmente cada um dos atributos requeridos. Se o *issuer* optar por permitir o uso de divulgação seletiva, deverá assinar as *claims* (atributos) presentes na credencial individualmente de maneira a que possam ser verificados como atributos singulares.

Apesar desta técnica ser desejável por aumentar a privacidade do *holder*, nem sempre é a mais apropriada. Se o *holder* poder escolher que informações são emitidas nas credenciais irá sempre tentar esconder as informações negativas (i.e. ocultar os pontos que possui na carta de condução mostrando apenas que está habilitado a conduzir). Para resolver este problema podem ser usadas três estratégias distintas:

- *Issuers* não podem isolar informação negativa em credenciais separadas. Para tal, a informação deve ser agregada de modo a que não seja possível apresentar partes isoladas dessa informação;
- *Verifiers* podem requisitar que toda a informação negativa seja apresentada, contendo um valor nulo se o sujeito não possuir atributos negativos (i.e. credenciais de registo criminal aquando à aplicação para um trabalho de cuidar de crianças);
- *Verifiers* usam outros meios para entrar em contacto com os *issuers* e requisitar qualquer informação negativa que estes possuam sobre o *holder*.

Provas de conhecimento zero (ZKP)

"Em criptografia, uma prova de conhecimento zero é um método através do qual uma entidade (provador) consegue provar a outra entidade (verificador) que conhece um valor, x, sem fornecer qualquer informação sobre esse valor"[36]. No ecossistema SSI o provador é o *holder* e o verificador é o *verifier*. Este tipo de provas permite a um *holder* provar que possui mais do que uma determinada idade sem revelar a sua idade em concreto ou que recebe um vencimento acima de um determinado valor sem indicar o valor exato. Citando Simari [37], "o conceito de prova em ZKP é diferente do conceito matemático tradicional. Provas matemáticas são rígidas que usam declarações auto-evidentes ou obtidas a partir de provas estabelecidas previamente. Provas de conhecimento zero estão mais próximas de um processo dinâmico usado por humanos para estabelecer a verdade de uma declaração através de troca de informação". Traduzindo o conceito apresentado para SSI, esta interação consiste em perguntas complicadas feitas pelo *verifier* às quais o *holder* terá de responder de forma convincente. Uma prova de conhecimento zero deve satisfazer três parâmetros:

- **completude:** se uma afirmação é verdadeira, o *verifier* será convencido pelo *holder*;
- **solidez:** se a afirmação é falsa, o *holder* não consegue convencer o *verifier*, exceto com uma probabilidade que pode ser desconsiderada;
- **conhecimento zero:** se a afirmação é verdadeira, o *verifier* não aprende nada além do facto de que a afirmação é verdadeira

Uma ZKP não tem como objetivo fornecer total anonimato mais sim melhorar a privacidade da melhor forma possível. Não seria possível a Alice provar que possui mais de 18 anos ao segurança de uma discoteca se não lhe poder revelar qualquer informação sobre a sua idade.

Passemos agora a um exemplo concreto do funcionamento de uma prova de conhecimento zero. Usando o exemplo da idade [38], vamos assumir que a Alice possui 19 anos. A Alice consegue provar ao Bob que possui mais de 18 anos através de uma prova de conhecimento zero baseada em cadeias de *Hashs*. Para tal, Alice começa por contactar a autoridade emissora (na qual o Bob confia) e pede-lhe o segredo **S**. Com este segredo, a autoridade emissora calcula uma cadeia de *hashs* de tamanho superior à idade da Alice em uma unidade (i.e. se a idade da Alice for 19 anos, a autoridade emissora irá computar 20 iterações de calculo de *hashs* sobre o segredo **S**).

$$EncryptedAge = HASH^{ActualAge+1}(S)$$

De seguida Alice efetua o mesmo processo mas executa apenas 2 iterações (diferença entre a idade dela e a idade mínima com a adição de uma unidade).

$$Proof = HASH^{1+ActualAge-AgeToProve}(S) = HASH^{2}(S)$$

Tendo a idade cifrada fornecida pela autoridade emissora, **EncryptedAge**, e a prova calculada por si, **Proof**, Alice fornece estes valores ao Bob. Bob por sua vez computa a cadeia de *hashs* com um número de iterações igual à idade mínima requerida sobre o valor **Proof** fornecido pela Alice.

$$VerifiedAge = HASH^{AgeToProve}(Proof)$$

Se os valores de **EncryptedAge** e **VerifiedAge** coincidirem, Bob pode confirmar que a idade da Alice está acima da idade mínima. Esta técnica só é possível se o *verifier* confiar no *issuer*. Se o *verifier* não confiar no *issuer* deverá rejeitar esta prova. De notar que se a Alice tentar falsificar o valor **Proof**, o valor de **EncryptedAge** nunca será igual ao valor de **VerifiedAge**. Por outras palavras, se o comprimento da cadeia de *hashs* computada pela Alice não for o correto, as cadeias computadas pela autoridade emissora e pelo Bob não terão o mesmo comprimento e a prova irá falhar.

O uso desta técnica não está restrito à criação de provas sobre a idade de um indivíduo. Pelo contrário, a técnica de ZKP pode ser aplicada em qualquer caso de uso cujo requisito seja apenas a prova de um predicado como é o caso de um banco requisitar uma prova que garanta que a Alice possui um vencimento mensal superior a mil euros por forma a disponibilizar um empréstimo bancário.

É importante referir que a técnica apresentada é apenas um exemplo simples da aplicação de provas de conhecimento zero. Em cenários reais as técnicas criptográficas utilizadas apresentam mais cuidados a nível da sua segurança criptográfica.

O artigo N*o Paradox Here: ZKPs Deliver Savvy Trust* [39] publicado por Daniel Hardman, uma das pessoas mais influentes na área de SSI, refuta de forma clara vários argumentos utilizados para provar a insegurança de provas de conhecimento, explicando com detalhe as várias razões destas provas poderem ser usadas com segurança na construção de credenciais numa arquitetura SSI. Esta técnica pode ainda ser usada fora do contexto dos atributos. O livro *An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials* [40] detalha uma abordagem baseada em provas de conhecimento zero para efetuar revogação de credenciais.

2.3.4 *Ciclo de vida de uma Verifiable Credential*

Efetuada a análise do ecossistema SSI e das suas várias componentes, é também importante analisar o fluxo de informação que circula pelo sistema, isto é, o ciclo de vida das *verifiable credentials* desde a sua emissão, pelo *issuer*, até à sua revogação, expiração ou esquecimento, passando pela geração e apresentação de provas baseadas nessas credenciais. Com o auxílio

do documento Verifiable Credentials Lifecycle do W3C [41] foram extraídos os seguintes estados para a representação da informação:

1. O *holder* obtém um identificador descentralizado (DID) e um par de chaves assimétricas.
2. O *holder* contacta o *issuer* e requisita uma *verifiable credential*.
3. O *issuer* autentica o *holder*, valida o DID fornecido e verifica, nos seus registos, se o *holder* tem direito à credencial pedida.
4. O *issuer* emite a *verifiable credential* e envia-a ao *holder*.
5. O *holder* armazena a *verifiable credential* no seu repositório (i.e. uma *wallet* digital).
6. o *holder* agrega os atributos das suas *verifiable credentials* numa *verifiable presentation* e apresenta-a a um *verifier*.
7. O *verifier* pode optar por validar parcial ou totalmente uma apresentação verificável de acordo com o modelo de risco adotado e a natureza da sua interação com o *holder*. A validação de credenciais inclui verificar se os atributos da apresentação pertencem ao *holder*, se o *issuer* é credível para emitir a informação presente nas mesmas, se as credenciais originais não foram revogadas pelos respetivos *issuers* e se ainda se encontram dentro do período de validade, e se outras políticas da credencial são respeitadas (i.e. restrições no número de uso ou nos *verifiers* permitidos).
8. O ciclo de vida de uma credencial termina quando o *issuer* a revoga, a sua validade expira ou o *holder* a elimina completamente.

O anexo A.2 retrata um exemplo do ciclo de vida de uma *verifiable credential* comum.

2.3.5 *Casos de uso e requisitos*

Tendo por base um documento de casos de uso em SSI [42], o W3C organizou, no documento *Data Model* [34], uma lista de caraterísticas que o ecossistema deve apresentar. Esta subsecção tem como objetivo agrupar as várias propriedades de um sistema de gestão de identidades descentralizado recorrendo ao documento elaborado pelo W3C.

- As *verifiable credentials* representam declarações feitas por um *issuer* de maneira a evitar violações de informação e respeitar a privacidade.
- As *verifiable presentations* permitem que qualquer *verifier* valide a autenticidade das *verifiable credentials* de múltiplos *issuers*. Estas apresentações podem divulgar os atributos de uma *verifiable credential* ou satisfazer predicados derivados solicitados pelo *verifier*, processo este geralmente designado por provas de conhecimento zero. Os predicados derivados são condições booleanas como "maior que", "menor ou igual que", entre outros. Tais provas são úteis para preservar a privacidade do *holder* em

situações nas quais este precisa de provar que possui um atributo com um certo valor sem necessidade de revelar o valor em questão.

- Tanto *verifiable credentials* como *verifiable presentations* devem suportar a serialização da informação para um ou mais formatos de dados legíveis à máquina. O processo de serialização e/ou desserialização deve ser determinístico, bidirecional e sem perdas.
- Os *Holders* (titulares) podem receber *verifiable credentials* de qualquer entidade, armazenar *verifiable credentials* em qualquer local sem afetar sua verificabilidade e sem que o *issuer* consiga inferir algo sobre onde elas estão armazenadas ou quando são acedidas, agregar atributos de *verifiable credentials* de diferentes *issuers* numa única *verifiable presentation* e compartilhar *verifiable presentations*, que podem ser verificadas sem revelar a identidade do *verifier* ao *issuer*.
- Os *issuers* podem emitir *verifiable credentials* sobre qualquer sujeito. Estas credenciais podem ou não suportar revogação, divulgação seletiva e provas de conhecimento zero de acordo com a vontade do *issuer*. Em caso de revogação, os *issuers* podem divulgar o motivo da revogação, devendo distinguir entre revogação por integridade criptográfica (por exemplo, a chave de assinatura está comprometida) e revogação por alteração de estado (por exemplo, a carta de condução está suspensa). A revogação de credenciais efetuada pelo *issuer* não deve revelar qualquer informação de identificação sobre o sujeito, o *holder*, a *verifiable credential* ou o *verifier*. Se a *verifiable credential* suportar divulgação seletiva, os *holders* poderão apresentar provas de *claims* sem revelar toda a informação da *verifiable credential*. Os *issuers* podem ainda fornecer um serviço para a atualização de *verifiable credentials*.
- O *verifier* pode verificar as *claims* presentes na *verifiable presentation* que o *holder* lhe forneceu independentemente do *issuer* que a emitiu. O processo de verificação não deve depender de interações diretas entre *issuers* e *verifiers* nem revelar a identidade do *verifier* ao *issuer*.
- Atuar como *issuer*, *holder* ou *verifier* não requer registo nem aprovação de nenhuma autoridade dado que a confiança envolvida é bilateral entre as partes. Qualquer entidade do sistema pode interagir com outras entidades do sistema através de agentes do utilizador. Estes agentes são nada mais do que *software* com ações pré-definidas que atua e interage com o ecossistema de acordo com a vontade dos seus controladores. Um agente de utilizador tem como objetivo facilitar a interação dos vários utilizadores com o sistema, efetuando todas as operações criptográficas e comunicações necessárias.
- O modelo de dados e a serialização devem ser facilmente extensíveis. Para tal é recomendado o uso de *JSON-LD* tal como é explicado na subsecção 2.2.3, em *Extensibilidade em documentos de DIDs*.

Casos de uso offline

Todas as soluções para construir ecossistemas de *self-sovereign identity* são baseadas em comunicações *online* entre as diferentes entidades de forma a comprovar a autenticidade das mesmas. Contudo, esta solução não é viável para muitos países em desenvolvimento nos quais não existe acesso à *internet* ou o acesso é muito lento. Adicionalmente, alguns ambientes de segurança elevada não permitem conexão ao exterior. É necessário encontrar uma forma de contornar estas situações e possibilitar o uso destes sistemas em ambientes *offline*. O documento *Use Cases and Proposed Solutions for Verifable Offline Credentials* [43] aborda com mais detalhe este problema e apresenta um vasto conjunto de casos de uso específicos, bem como algumas possíveis soluções. Entre essas soluções encontra-se a solução da Sovrin, que suporta a criação de *snapshots* do registo público (i.e. blockchain). Estes *snapshots* podem ser armazenados em dispositivos *offline* e atualizados periodicamente. Dito isto, os dispositivos devem apresentar poder computacional suficiente para efetuar as várias operações criptográficas. A transmissão de informação em ambientes *offline* pode ser feita através de códigos QR, códigos de áudio, RF, entre outros.

2.4 TECNOLOGIA DLT

Nos últimos anos a tecnologia DLT (*Distributed Ledger Technology*) tem sido fortemente associada à gestão de carteiras digitais de criptomoedas, em particular a Bitcoin [44] e o Ethereum [45]. Contudo, esta tecnologia apresenta um vasto conjunto de casos de uso além da representação de valor monetário digital, entre os quais a implementação de um arquitetura *self-sovereign identity*. O aparecimento desta tecnologia descentralizada proporcionou a possibilidade de gerir identidades de forma completamente descentralizada e de remover a necessidade de autoridades centralizadas ou pontos únicos de falha.

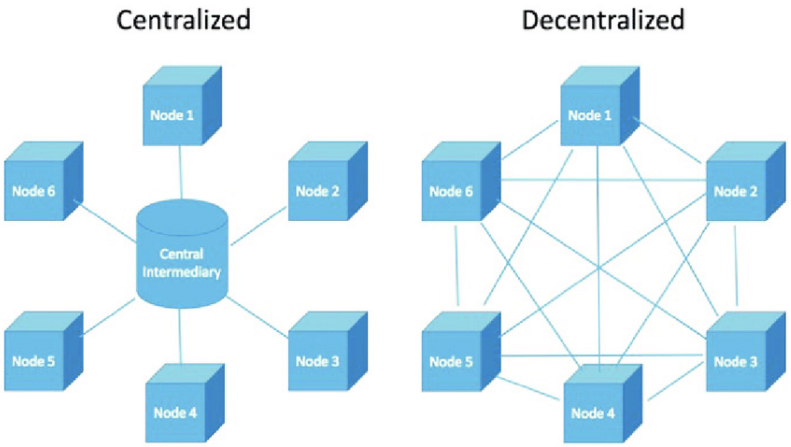


Figura 20: BD centralizada vs BD descentralizada (retirada de [46])

Com esta tecnologia, qualquer entidade pode criar e gerir a sua própria identidade digital. Como foi mencionado na secção 2.3, a tecnologia DLT, em particular a tecnologia *blockchain*, será usada como *data registry* da implementação desenvolvida. Esta secção explica, de um forma geral, o que é uma *blockchain* e descreve quais são as aplicações e as vantagens desta tecnologia na criação de identidades descentralizadas. A Figura 20 mostra uma comparação entre uma base de dados centralizada, na qual toda a informação está contida num único ponto que é controlado por um único indivíduo ou organização, e uma base de dados descentralizada, na qual qualquer utilizador consegue aceder e comprovar a veracidade de todas as informações armazenadas e consultar todo o histórico de alterações efetuadas desde a criação da base de dados. Num sistema descentralizado cada membro da rede possui uma cópia do registo descentralizado, que deve ser atualizado de acordo com o protocolo de consenso em vigor. Este protocolo é responsável por indicar aos vários nós da rede que procedimentos devem seguir para adicionar e atualizar informações nos seus registos e, no caso de *blockchains permissioned* (publicação de novos blocos baseada em permissões de utilizador), quem deverá ser o próximo utilizador a publicar blocos na rede. Mais à frente nesta secção será feita uma comparação entre *blockchains permissioned* e *permissionless* (qualquer utilizador pode publicar novos blocos no registo).

2.4.1 *O que é a tecnologia blockchain*

Antes de iniciar a análise da aplicação e vantagens de usar uma *blockchain* num sistema de gestão de identidades descentralizado é importante entender o que é uma *blockchain* e quais as suas propriedades. Segundo o documento *Blockchain Technology Overview* [5], *blockchains* são "registos digitais invioláveis, implementados de forma distribuída e sem necessidade de uma autoridade central". Estes registos "permitem que uma comunidade de utilizadores registe transações num registo público de tal forma que, assegurado o seu correto funcionamento, não é possível alterar transações após a sua publicação". Entre as várias propriedades das *blockchains* destacam-se a imutabilidade, a integridade e a persistência da informação. Como o próprio nome indica, uma *blockchain* é um registo que se encontra estruturado numa cadeia de blocos e a informação que se pretende armazenar está contida nesses blocos. Uma *blockchain* trata novos blocos como atualizações ou modificações dos blocos mais antigos. Admitindo que não é possível modificar blocos após estes serem publicados, a *blockchain* armazena o registo de todas as alterações efetuadas, podendo ser usada como fonte de verdade. A garantia de imutabilidade deriva diretamente dos participantes da rede, que devem concordar no estado atual da *blockchain*, através de protocolos de consenso. Protocolos de consenso são discutidos detalhadamente mais à frente nesta secção.

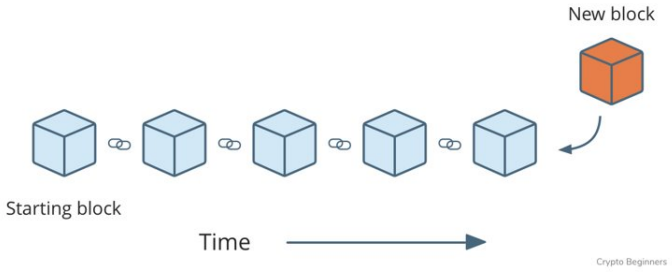


Figura 21: Estrutura blockchain (retirada de [47])

De modo a publicar dados na *blockchain*, os utilizadores submetem transações para a rede, através dos seus nós, sendo estas transações propagadas pelos restantes nós da rede. De entre todos os nós da rede, apenas alguns são capazes de efetuar publicações de blocos para a *blockchain*. Este nós são chamados *publishing nodes*. Um bloco está dividido em duas componentes: o cabeçalho e o corpo. No cabeçalho estão representados todos os metadados associados ao bloco. Por sua vez o corpo contém uma lista de transações válidas que foram submetidas para a rede. Uma transação só é efetivamente adicionada à *blockchain* após um bloco, no qual ela está representada, ser publicado na *blockchain*. Assim que um bloco é publicado na *blockchain*, os restantes nós da rede verificam a validade e autenticidade de todas as transações incluídas no corpo do bloco, podendo rejeitar o bloco caso este contenha transações inválidas. Na sua generalidade, os blocos de uma *blockchain* apresentam a seguinte estrutura:

- Cabeçalho do bloco
  - Número do bloco.
  - Valor da *hash* do cabeçalho do bloco anterior.
  - Um *timestamp*.
  - O tamanho do bloco.
  - Valor da *hash* do corpo do bloco.
  - O valor do *nonce*, típicamente usado para resolução de *hash puzzles* em *blockchains permissionless*.
- Corpo do bloco
  - Lista de transações
  - Outros dados

Como é possível observar nesta estrutura, cada bloco (exceto o primeiro bloco, *genesis block*) contém o *hash digest* do cabeçalho do bloco anterior, que por sua vez possui o *hash digest* do seu corpo, com todas as suas transações, e o *hash digest* do cabeçalho do bloco anterior. Devido a esta estrutura, se a informação de blocos anteriores fosse alterada, seria

gerada uma *hash* diferente e, consequentemente, todos os blocos posteriores à alteração iriam também derivar *hashs* diferentes. Esta propriedade permite validar se o novo bloco que está a ser publicado respeita todos os dados publicados até à altura através de uma simples comparação entre a *hash* do cabeçalho do último bloco da cadeia e a *hash* que está contida no cabeçalho do bloco a ser adicionado. A Figura 22 ilustra a estrutura apresentada, com a informação separada em cabeçalho (*Block Header*) e corpo (*Block Data*).

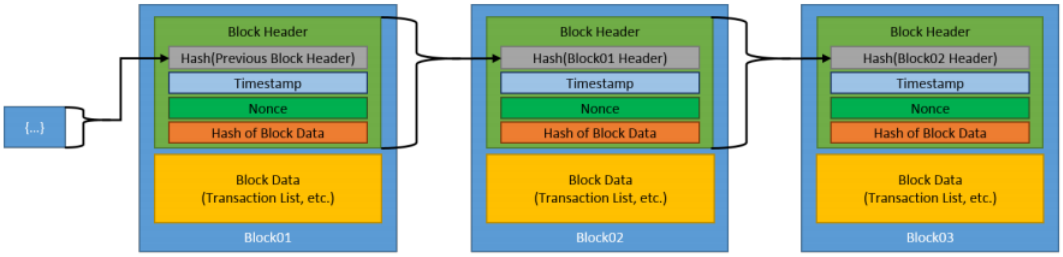


Figura 22: Cadeia de blocos genérica (adaptada de Blockchain Technology Overview [5])

Protocolo de consenso

Sendo a *blockchain* uma estrutura de dados distribuída, é possível que sejam adicionados dois blocos válidos ao mesmo tempo ou que certos nós contenham informação desatualizada por algum tempo. O fenómeno de publicação simultânea de blocos é designado por *fork* e ocorre quando dois blocos válidos, que apontam para o mesmo bloco pai, são submetidos ao mesmo tempo. A figura 23 ilustra a ocorrência de um *fork* no qual os blocos A e B são publicados em simultâneo.

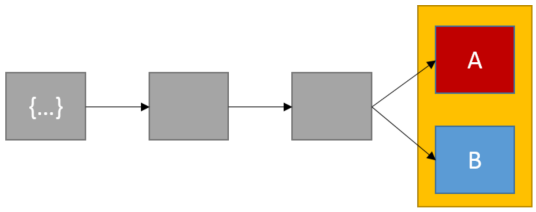


Figura 23: *Fork* (retirada de [5])

Como consequência alguns nós irão aceitar um dos blocos enquanto que os restantes irão aceitar o outro bloco, fazendo com que a *blockchain* entre temporariamente num estado inconsistente. Devido à possibilidade dos blocos serem alterados, uma transação não é aceite até que sejam criados vários blocos adicionais por cima do bloco que contém essa transação. A aceitação de um bloco é normalmente probabilística, ou seja, quanto mais blocos forem adicionados por cima desse bloco, mais provável é que esse bloco não seja substituído. Para atingir o consenso todos os nós devem esperar pela submissão de novos blocos. Se um nó receber duas cadeias de blocos de dois nós diferentes deverá adotar a cadeia de blocos mais

longa. Tomando como exemplo o caso presente na figura 24, o bloco aceite deverá ser o B, seguido do novo bloco encadeado no mesmo.

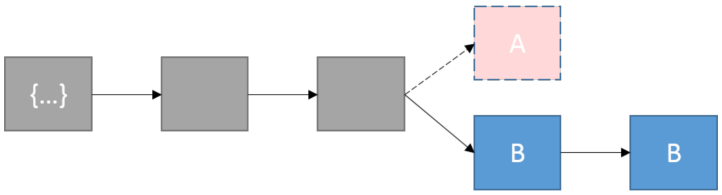


Figura 24: Novo bloco adicionado (retirada de [5])

Assim que a maioria dos nós aceitar a mesma cadeia de blocos a ser publicada na *blockchain* todos os nós atingem rapidamente o consenso fazendo com que a *blockchain* volte novamente a um estado consistente. Estes conflitos são, por norma, resolvidos rapidamente. As transações presentes nos blocos descartados que não foram adicionadas pelos blocos publicados devem ser inseridas na lista de transações pendentes e submetidas novamente. Cada nó mantém a sua própria lista de transações pendentes.

De maneira a explicar mais detalhadamente o mecanismo de resolução de conflitos vamos assumir que a *blockchain* se encontra atualmente no bloco #57 e os nós 1 e 2 adicionaram o bloco #58 simultaneamente. O bloco adicionado pelo nó 1 contém as transações A, C e D enquanto que o bloco adicionado pelo nó 2 é composto pelas transações B e C. Apesar de os blocos conterem transações diferentes, nenhum dos nós está errado. Cada nó responsável por efetuar publicações de blocos recebe informações sobre transações que deverão ser publicadas na *blockchain*. Contudo, dado que se trata de um registo distribuído, as informações que cada um dos nós recebe poderão diferir.

Digamos que o bloco #59, com as transações A e E, é adicionado pelo nó 3 e encadeado ao bloco adicionado pelo nó 2. A estratégia utilizada passa pela aceitação da cadeia que apresenta a maior sequência de blocos. Assim que um nó receba esta nova cadeia de blocos deverá descartar o bloco sobre o qual estava aguardar e aceitar a cadeia mais longa com o bloco #58, adicionado pelo nó 2, e o bloco #59, adicionado pelo nó 3. A nova cadeia contém as transações A, B, C e E, contudo não contém a transação D. Esta transação deverá portanto ser submetida novamente na adição de um novo bloco. Dado que alguns nós podem não conter esta transação, não há garantias de que será inserida imediatamente no bloco #60 mas será eventualmente adicionada à *blockchain*.

Como consequência do protocolo de consenso, se algum utilizador mal intencionado tentar alterar parte ou a totalidade da cadeia de blocos publicados, as suas alterações serão rejeitadas pelos restantes participantes uma vez que é necessário que pelo menos metade dos nós participantes concordem num estado para que haja uma transição do estado anterior para o novo estado. Para o atacante conseguir obter sucesso precisa de controlar pelo menos 50% dos nós da rede, o que se torna cada vez mais difícil à medida que novos utilizadores se

juntam à rede. Eventualmente a rede irá atingir um número de participantes suficientemente elevado para que não seja possível efetuar este ataque com sucesso.

Modelo de confiança

Sendo o principal objetivo da *blockchain* garantir a integridade e autenticidade dos dados armazenados, a principal questão sobre esta tecnologia gira em torno da sua capacidade de estabelecer confiança. Dito isto, a confiança dentro de uma *blockchain* é garantida por quatro caraterísticas:

- Imutabilidade - Todos os novos blocos são inseridos no final da cadeia, o que fornece todo o historial de transações efetuadas desde a criação do primeiro bloco, *genesis block*.
- Criptografia - Uso de técnicas criptográficas para impedir a violação dos dados armazenados e possibilitar a verificação da validade e autenticidade dos mesmos.
- Transparência - Todos os participantes possuem uma cópia da *blockchain*, o que garante total transparência entre eles.
- Descentralização - Permite escalar o número de nós da *blockchain* de forma a adicionar tantos utilizadores quantos forem necessários. Como não existe um ponto único de falha torna-se muito mais difícil (se não impossível) subverter a rede através de ataques como negação de serviço ou sabotagem dos dados.

Apesar destas quatro caraterísticas permitirem que uma *blockchain* possa ser usada como fonte de verdade é necessário garantir que a imutabilidade não é violada. Dado que uma *blockchain* altera de estado de acordo com o protocolo de consenso, se a maioria dos nós concordar com a remoção de determinados nós é possível remover esses nós. Por esta razão é importante garantir que nenhuma entidade ou organização controla mais de 50% da rede. Assim que a *blockchain* atinja um número elevado de utilizadores a sua propriedade de imutabilidade é confiável.

Modelo de permissões

Blockchains podem ser categorizadas em duas categorias distintas, com base nos seus modelos de permissões: *permissionless* e *permissioned*.

Uma *blockchain* ***permissionless*** permite que qualquer utilizador leia blocos existentes e publique novos blocos na rede sem necessidade de obter permissão de alguma autoridade. Como consequência, utilizadores mal intencionados podem tentar subverter o sistema e levar ao seu incorreto funcionamento. Dado que neste tipo de *blockchains* é extremamente difícil publicar um novo bloco e verificar um bloco adicionado é fácil, um utilizador mal intencionado não consegue publicar blocos a um ritmo superior ao ritmo que esses blocos são descartados, ou seja, a única forma de um utilizador mal intencionado subverter o

sistema é conseguindo que mais de 50% dos nós da rede aceitem os blocos publicados por ele. Garantindo que mais de 50% dos utilizadores apresenta boas intenções, é impossível subverter o sistema. Para promover um bom comportamento, os *publishing nodes* são normalmente recompensados de alguma forma.

Por sua vez, uma *blockchain* ***permissioned*** permite a uma autoridade restringir os utilizadores com autorização para ler ou publicar blocos na rede. Ao contrário das *blockchains permissionless*, estas *blockchains* não requerem que os utilizadores resolvam *puzzles*, ou outros desafios computacionais, para poderem publicar novos blocos uma vez que o utilizador precisa de fornecer a sua identidade para participar como membro da *blockchain*. Caso um utilizador tente subverter o sistema de alguma forma, é possível revogar a sua autorização e abrir um processo legal contra esse utilizador.

Smart Contracts

O termo *smart contract* foi introduzido por Nick Szabo em 1994. Segundo a sua definição, um *smart contract* "é um protocolo de transação informatizado que executa os termos de um contrato" [48]. Na sua definição, Nick declara que os principais objetivos do *smart contract* são satisfazer as condições contratuais comuns (tais como termos de pagamento, confidencialidade e até mesmo obrigar ao cumprimento), minimizar condições maliciosas ou acidentais, e minimizar a necessidade de intermediários de confiança. Por sua vez o NIST (National Institute of Standards and Technology) define o termo *smart contract* como sendo uma coleção de código e dados, também designados por funções e estado, que é implementado através de transações assinadas criptograficamente na *blockchain* [5]. A principal funcionalidade de um *smart contract* é fornecer serviços aos utilizadores através das funções nele contidas, sendo capaz de efetuar cálculos, armazenar informação, disponibilizar a informação armazenada, entre outras operações.

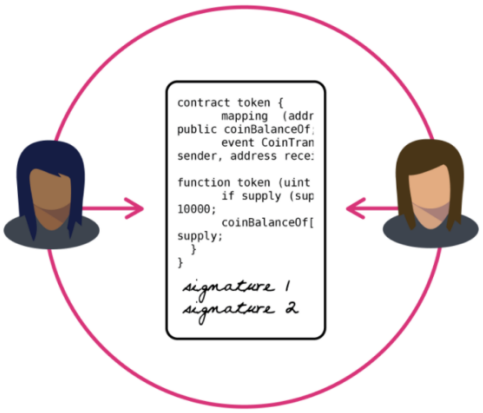


Figura 25: *Smart contract* (retirada de [49])

À semelhança da publicação normal de transações na *blockchain*, todos os nós que executarem o contrato devem acordar no estado que é obtido após esta execução, ou seja, a execução do *smart contract* deve ser determinística de modo a que para determinados dados de entrada ele produza sempre os mesmos dados de saída. Para alcançar este objetivo, o *smart contract* não pode operar com dados externos pelo que todos os dados têm de lhe ser passados diretamente como argumentos das funções. Para utilizarem o *smart contract*, os utilizadores da *blockchain* podem criar transações que enviam dados para as funções públicas oferecidas pelo mesmo. De entre as muitas vantagens de um *smart contract* destacam-se a sua transparência (código fonte visível a qualquer utilizador), a redução dos custos associados e a redução do tempo para completar uma transação. Além disso, um fator importante destes contratos é que, por estarem na *blockchain*, são também armazenados permanentemente e não podem ser alterados ou apagados, o que faz com que seja possível usá-los como intermediários de confiança.

2.4.2 *Aplicação da tecnologia DLT em self-sovereign identity*

Como indicado na subsecção 2.4.1, em "Modelo de confiança", a confiança dentro de uma blockchain é garantida por quatro caraterísticas: **imutabilidade**, **criptografia**, **transparência** e **descentralização**. Estas quatro caraterísticas permitem que a *blockchain* atue como base de confiança numa arquitetura *self-sovereign identity*, mais concretamente como *data registry* dessa mesma arquitetura.

Na subsecção 2.3.1 o *data registry* está definido como sendo um "sistema que fornece meios de criação e verificação de identificadores, chaves, e outros dados relevantes, tais como esquemas de *verifiable credentials*", sendo ainda "responsável por estabelecer a identificação e autenticação do *holder*". Devido às caraterísticas apresentadas anteriormente, a tecnologia *blockchain* permite:

- garantir a integridade e autenticidade dos dados registados através de técnicas criptográficas.
- garantir que todos os nós têm conhecimento de todas as transações efetuadas na rede dado que cada nó possui a sua própria cópia da *blockchain*.
- registar dados públicos de forma permanente, tais como chaves públicas, esquemas de *verifiable credentials* ou até mesmo as próprias *verifiable credentias* no caso de entidades públicas que querem provar a todos os utilizadores que possuem certos atributos. Tal é possível devido à *blockchain* atuar como registo no qual apenas é possível adicionar novos dados no final da cadeia, não sendo possível alterar dados já publicados.
- garantir que os identificadores gerados são únicos e descentralizados (DIDs), recorrendo ao mecanismo de consenso em que todos os nós da rede devem concordar

no estado da mesma. Qualquer transação, adicionada por um nó, que tente inserir identificadores já existentes será rejeitada pelos restantes nós da rede.

- estabelecer a relação entre o DID e o respetivo *holder* através de criptografia assimétrica, ligando o identificador à chave pública do *holder*. Se o *holder* demonstrar ter conhecimento da chave privada associada a essa chave pública consegue provar que é o titular da credencial.

É, contudo, importante esclarecer que as caraterísticas da tecnologia *blockchain* não são desejáveis para armazenar todos os tipos de informação. Informação de identificação pessoal (PII) cuja privacidade seja um requisito deve ser armazenada *off-chain* dado que qualquer participante da rede tem acesso a todas as informações armazenadas na *blockchain*. Apesar de ser possível armazenar esta informação na *blockchain* recorrendo a cifras de maneira a manter esta informação privada, tal não é uma boa prática uma vez que não é possível determinar quando as técnicas criptográficas utilizadas serão quebradas. Dado que todos os participantes possuem uma cópia com todos os dados da *blockchain* e podem efetuar *snapshots* da mesma sempre que desejarem, assim que a cifra utilizada for quebrada, todas as informações de identificação pessoal serão divulgadas.

A subsecção "Infraestrutura de chave pública centralizada" explica mais detalhadamente a utilização de uma infraestrutura de chave pública tradicional enquanto que a subsecção "Infraestrutura de chave pública descentralizada" explora a utilização de uma infraestrutura de chave pública descentralizada como meio de autenticação de entidades numa *blockchain*.

Infraestrutura de chave pública centralizada

As primeiras soluções criptográficas eram baseadas em esquemas de cifras simétricas assentes em algoritmos que recebiam uma mensagem e a chave secreta e geravam um criptograma. Quem possuísse a mesma chave secreta conseguia efetuar o processo inverso e decifrar o criptograma de forma a obter a mensagem original. Isto obrigava a que todos os intervenientes na comunicação precisassem de conhecer a chave secreta de modo a conseguirem cifrar e decifrar as mensagens trocadas entre eles. Além disso, dado que todos os intervenientes utilizavam a mesma chave para cifrar uma mensagem não era possível provar a autoria da mesma, isto é, não havia autenticação da origem. De modo a satisfazer estas duas necessidades surgiu a criptografia assimétrica, ou **criptografia de chave pública**, um esquema baseado em duas chaves: chave pública (para cifrar uma mensagem) e chave privada (para decifrar essa mesma mensagem). A chave pública pode ser partilhada com qualquer entidade e deve ser usada para cifrar mensagens cujo destinatário seja a entidade associada a essa chave. Já a chave privada deve ser de conhecimento exclusivo do seu titular. Uma mensagem cifrada com a chave pública de uma entidade só pode ser decifrada com a respetiva chave privada associada. Contudo, este tipo de técnicas criptográficas na sua

forma mais simples está suscetível a ataques de *man in the middle* nos quais o atacante se faz passar pelas duas entidades que querem comunicar de forma segura, estabelecendo uma comunicação com cada uma dessas entidades. Por exemplo, digamos que a Alice pretende estabelecer uma ligação segura com o Bob. Para tal, tenta adquirir a chave pública do Bob e estabelecer uma conexão com ele, cifrando os dados com a chave pública que adquiriu. Um atacante pode fornecer a sua chave pública à Alice fazendo-se passar pelo Bob e estabelecer uma conexão com ela. Como a Alice não sabe que a chave pública que adquiriu pertence ao atacante irá cifrar todos os dados com essa chave. Do mesmo modo, o atacante estabelece uma conexão com o Bob fazendo-se passar pela Alice. Sempre que a Alice tenta enviar uma mensagem para o Bob irá enviá-la para o atacante, podendo este apenas intercetar a mensagem e reencaminhá-la para o Bob ou alterá-la de maneira a prejudicar a Alice. Do mesmo modo o atacante é capaz de intercetar todas as mensagens que o Bob envia à Alice. O atacante só é capaz de fornecer a sua chave pública no lugar das chaves públicas corretas por não haver forma de a Alice confirmar que a chave pública que adquiriu é de facto a chave pública do Bob e vice-versa. Este tipo de ataques introduz a necessidade de garantir a autenticidade da chave pública, ou seja, garantir que a chave pública pertence realmente ao destinatário da mensagem. Tal é conseguido através de certificados digitais: documentos que ligam metadados (i.e. nome da organização, email e nacionalidade do utilizador, chave pública, etc) à respetiva pessoa ou organização no mundo real.

O esquema de autenticação atual é baseado num modelo de confiança centralizado, isto é, depende unicamente da hierarquia de entidades de certificação (ECs), que são responsáveis por estabelecer a autenticidade da chave pública e o seu respetivo titular através da emissão de certificados digitais. Estes certificados digitais estão organizados de forma hierárquica culminando numa **raiz de confiança**, isto é, um documento auto-assinado. O processo de validação de um certificado digital deve percorrer a cadeia de certificados até atingir o certificado auto-assinado (de confiança) ou um certificado intermédio no qual já se confie. Dado que uma entidade de certificação é responsável por estabelecer a ligação entre chaves públicas e os seus respetivos titulares, se esta for comprometida pode pôr em causa todo o sistema. Por esta razão, uma das vantagens de utilizar *blockchains* em sistemas de gestão de identidades descentralizados é precisamente remover a necessidade de recorrer a uma entidade de certificação centralizada, usando a própria *blockchain* como registo de confiança para estabelecer a ligação entre chaves públicas e os respetivos titulares. Esta estratégia permite utilizar a infraestrutura de chave pública tradicional num ambiente distribuído, como será explicado de seguida.

Infraestrutura de chave pública descentralizada

Devido à tecnologia *blockchain* ser um registo imutável, totalmente transparente e descentralizado, é possível utilizá-la para verificar a autenticidade das associações das chaves públicas

com os respetivos titulares e eliminar qualquer dependência face a hierarquias de entidades de certificação centralizadas. Assim que uma associação entre uma chave pública e uma entidade for publicada na *blockchain*, essa associação fica visível para todos utilizadores da rede e não é possível alterá-la. É contudo possível submeter uma nova transação que visa substituir essa associação, ficando sempre registado o histórico de mudanças. Como foi explicado na subsecção 2.2.3, a chave pública é inserida dentro de um documento, o documento do DID, que por sua vez é indexado através do identificador descentralizado do *holder*. Deste modo, qualquer pessoa que saiba o DID do *holder* consegue facilmente obter o documento associado a esse DID e as respetivas chaves públicas.

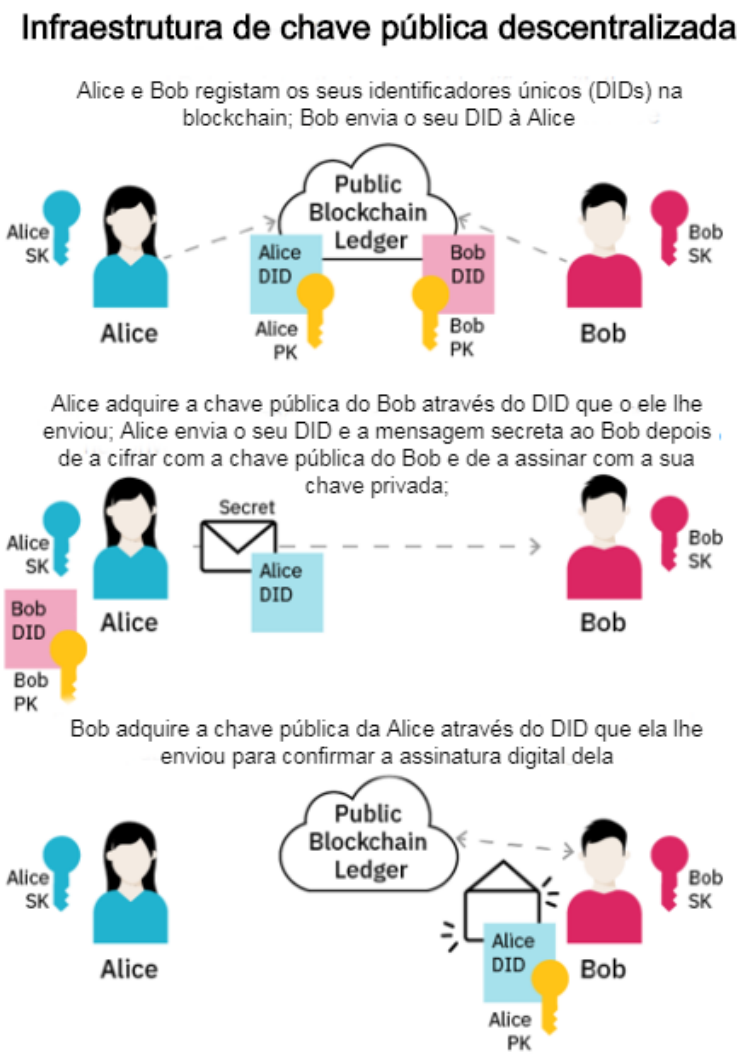


Figura 26: Infraestrutura de chave pública descentralizada (adaptada de IBM [50])

A figura 26 representa o mecanismo de autenticação da Alice e do Bob através de uma infraestrutura de chave pública descentralizada. Neste exemplo, ambos geram os seus identificadores descentralizados (DIDs), associando-lhes uma chave pública e registando-os

na *blockchain*. Sempre que é necessário obter a chave pública de uma entidade basta apenas consultar a *blockchain* utilizando o DID dessa entidade.

Além da ajuda no estabelecimento de conexões seguras entre duas entidades, a infraestrutura de chave pública descentralizada permite ao *issuer* emitir *verifiable credential* de forma a que o *holder* consiga provar a posse das mesmas através da simples inserção do DID do *holder* nessas credenciais, ou seja, as credenciais incluem o DID do *holder* sobre o qual estas foram emitidas. No processo de verificação, o *verifier* usa o DID presente na credencial para obter a chave pública do *holder*, através da consulta do documento armazenado na *blockchain*, e gera um desafio para autenticar o *holder*, isto é, para obter a prova de que o *holder* conhece a chave privada associada a essa chave pública e, consequentemente, possui os atributos emitidos na credencial. É importante realçar que o DID presente da credencial é também assinado pelo *issuer*.

Contudo, assim como qualquer caso de uso da tecnologia *blockchain*, também uma infraestrutura de chave pública descentralizada está vulnerável à perda de chaves privadas. Como nesta arquitetura a autenticação depende totalmente do conhecimento da chave privada e a perda de chaves é algo muito frequente, como é possível observar nos inúmeros casos de perda de chaves na Bitcoin e outras criptomoedas, é necessário ter um mecanismo de recuperação das mesmas. Em SSI a solução passa por separar autenticação de autorização, autorizando outros utilizadores a alterar o documento do DID e, em particular, a chave pública de autenticação do *holder*, como é explicado em *Propriedades padrão em documentos de DIDs*, na subsecção 2.2.3.

2.5 Considerações de segurança

Com o objetivo de promover a segurança a nível das *verifiable credentials*, o W3C definiu um pequeno conjunto de considerações [34] que *issuers*, *verifiers* e *holders* devem conhecer. Esta secção apresenta cada uma dessas considerações, bem como uma breve explicação do que cada uma delas significa.

Uso de técnicas criptográficas

Quando se pensa em aumentar a segurança de dados digitais a primeira solução passa pelo uso de técnicas criptográficas. Contudo, todas as técnicas criptográficas são eventualmente quebradas por novos ataques e avanços na tecnologia. Por esta razão devem ser implementados mecanismos que permitam facilmente atualizar técnicas criptográficas ou bibliotecas que tenham expirado ou que tenham sido quebradas e, consequentemente, invalidar e substituir as credenciais existentes. Além disso, deve existir uma monitorização constante de modo a detetar atempadamente qualquer ameaça de possa comprometer a segurança de técnicas criptográficas.

Proteção da integridade

As *verifiable credentials* apresentam normalmente *URLs* para dados externos à própria credencial devido ao uso de *JSON-LD* (i.e. imagens e contextos). Por norma estes dados não estão protegidos contra adulteração uma vez que residem fora da proteção da prova da *verifiable credential*. Para assegurar proteção de integridade devem-se usar esquemas URL que forneçam integridade do conteúdo, como por exemplo *hashlinks*.

Na grande maioria dos casos tal não será necessário uma vez que as implementações devem ser iniciadas já com cópias estáticas dos contextos *JSON-LD* mais importantes.

Roubo de dispositivos e/ou personificação

Quando as *verifiable credentials* estão armazenadas num dos dispositivos do *holder* e esse dispositivo é roubado, é possível que um atacante consiga utilizar essas credenciais. Para mitigar este problema, a solução passa por incluir *password*, *pin* ou dados biométricos, preferencialmente com autenticação multi-fator, para desbloquear o dispositivo, obter acesso ao repositório das credenciais e obter acesso às chaves criptográficas. Idealmente o utilizador deverá ter de se autenticar sempre que pretender utilizar/alterar credenciais ou efetuar qualquer outra operação de risco.

Especificações de métodos dos DIDs

As especificações dos métodos têm como objetivo definir claramente as operações usadas para criar, resolver e remover DIDs, bem como criar e alterar os seus respetivos documentos. Consequentemente, estas especificações devem incluir as suas próprias secções de considerações de segurança, devendo estas:

1. Considerar todos os requisitos presentes em *Guidelines for Writing RFC Text on Security Considerations* [51].
    - Devem ser consideradas pelo menos as seguintes formas de ataques: *espionagem*, *repetição*, *inserção de mensagens*, *eliminação*, *modificação*, *man-in-the-middle*, e negação de serviço;
    - Se o protocolo incorporar mecanismos de proteção criptográficos, deve também ser indicado claramente que partes dos dados estão protegidas e quais são as proteções utilizadas (i.e. integridade, confidencialidade, autenticação dos *endpoints*, entre outros). Em adição devem ser indicados os tipos de ataques aos quais a proteção criptográfica está suscetível;
    - Dados que são mantidos em segredo (chaves criptográficas, *seeds* aleatórias, etc.) devem ser claramente tabelados;

    - Se a tecnologia envolve autenticação, a segurança do método de autenticação deve ser claramente especificada.
2. Discutir riscos residuais (i.e. riscos de comprometimento num protocolo, riscos de implementação incorreta, ou riscos de cifra) após a implementação dos mecanismos de mitigação de ameaças;
3. Fornecer proteção de integridade e autenticação a todas as operações CRUD;
4. Discutir como é feita a autenticação de *endpoints*;
5. Explicitar o mecanismo pelo qual os DIDs são comprovadamente atribuídos de forma única.

Service endpoints de autenticação

Se um documento de um DID publica um *service endpoint* para autenticação ou autorização do sujeito, é responsabilidade do fornecedor desse *service endpoint*, sujeito, e/ou parte confiável cumprir os requisitos do protocolo de autenticação suportado nesse *service endpoint*.

Notificação de alterações em documentos de DIDs

Se um atacante obtiver acesso a uma chave de autorização incorporada no documento do DID pode efetuar um conjunto de operações sem a permissão do respetivo dono. Uma possível forma de mitigar este problema consiste em monitorizar ativamente o documento e notificar o *holder* sempre que o documento seja alterado. Como não existe registo intermediário ou provedor de contas para gerar a notificação, é recomendado utilizar:

1. **subscrições**: se o registo no qual o DID está registado suportar notificações de alterações, devem ser enviadas notificações diretamente para os *service endpoints* relevantes registados no DID;
2. **auto-monitorização**: o sujeito de um DID pode usar o seu próprio agente para monitorizar periodicamente as alterações num documento DID;
3. **monitorização por terceiros**: o sujeito de um DID pode recorrer a um serviço de monitorização de terceiros para monitorizar constantemente alterações no seu documento.

2.6 CONSIDERAÇÕES DE PRIVACIDADE

Quando o ato de se envolver com outra parte é, por si só, um ato reconhecível, a privacidade diminui consideravelmente. Deve-se por isso optar sempre por tecnologias e interfaces humanas que tenham como padrão preservar o anonimato e o uso de pseudónimos. Para reduzir as pegadas digitais deve-se usar a menor quantidade possível de protocolos, recorrer a camadas criptográficas de transporte e preencher as mensagens com comprimentos padrão.

É também crítico que sejam aplicados os princípios de *privacy by design* em todos os aspetos de uma arquitetura SSI. Nesta secção são abordados os vários aspetos que a comunidade W3C acordou [34] que devem ser tidos em consideração durante o desenvolvimento de uma arquitetura SSI.

Espectro de privacidade

Algumas entidades desejam que os seus identificadores apontem diretamente para as suas identidades no mundo real, o que torna estes identificadores altamente correlacionáveis entre si. Outras entidades desejam ocultar qualquer informação que possa ser utilizada para as identificar. Dependendo do caso de uso, uma pessoa pode estar disposta a revelar mais ou menos informações sobre si. Por exemplo, uma entidade governamental utiliza identificadores globais para que qualquer outra entidade possa associar esses identificadores ao respetivo governo. No outro extremo do espetro, uma pessoa que deseja comprar tabaco não quer revelar qualquer informação sobre si, tendo apenas de provar que é maior de idade. Deste modo, não existe um método único para lidar com a privacidade que funcione para todos os casos de uso, ou seja, cada caso de uso possui a sua solução para atingir o grau de privacidade desejado.

Informação de identificação pessoal (PII)

Informações de identificação pessoal tais como identificadores emitidos pelo governo, moradas ou nomes completos podem ser facilmente usadas para correlacionar os dados de uma entidade. Até mesmo informações pessoais pouco comprometedoras individualmente, como data de nascimento e código postal, podem-se demonstrar extremamente comprometedoras quando correlacionadas, como é explicado de seguida, em **Correlação de identificadores**. Por outro lado, o registo de dados público armazena publica e permanentemente todos os dados lá inseridos (i.e. documentos dos DIDs) não sendo possível apagá-los. Por esta razão é altamente recomendado que esses documentos não contenham qualquer PII, salvo algumas exceções em que entidades públicas querem conscientemente tornar essas informações públicas. Em geral toda a informação privada deve ser armazenada nos *service endpoints*, sob controlo do *holder*.

Correlação de identificadores

Os identificadores usados para identificar o sujeito podem criar elevado risco de correlação quando são usados por muito tempo ou em vários domínios. Como foi mencionado na consideração de privacidade anterior, até mesmo atributos de identificação pessoal que não aparentam possuir qualquer risco de identificação pessoal quando usados individualmente, podem ser extremamente identificáveis quando correlacionados com outros atributos. Por

exemplo, em todo o mundo existem inúmeras pessoas que nasceram em 1990, contudo, quando se correlaciona esta informação com o código postal é muito provável que seja possível mapear diretamente o ano de nascimento para uma pessoa específica ou para um pequeno conjunto de pessoas. À medida que o *holder* fornece mais informação, o *verifier* pode associá-la à idade e ao código postal que já lhe haviam sido fornecidos e construir assim o perfil do *holder*. De modo a diminuir o risco de correlação é recomendado usar identificadores pessoais o menor número de vezes possível ou, idealmente, não os usar de todo. Contudo, certos casos de uso requerem a utilização de informação de identificação pessoal do sujeito da credencial (i.e. endereços de *email*, moradas, idade, e nome). Para mitigar problemas de correlação de dados, os *issuers* devem optar por opções de preservação de privacidade sempre que possível através do uso de técnicas criptográficas que possibilitem o uso de divulgação seletiva de atributos e provas de conhecimento zero. Apesar destas técnicas permitirem uma redução drástica na correlação de dados, a existência de identificadores de longa duração e técnicas de *browser tracking* conseguem superar até as técnicas criptográficas mais modernas. Por esta razão não se deve utilizar identificadores de forma descuidada. Adicionalmente, é recomendado utilizar vários identificadores descentralizados e evitar assim associar informação de identificação pessoal a identificadores de longa duração, reduzindo o fornecimento dessa informação ao mínimo necessário. Também os responsáveis pela implementação dos sistemas de gestão de identidades são aconselhados a cifrar tanto os dados em trânsito como os dados que estão armazenados. Além disso, devem ser implementados mecanismos que identifiquem atributos de *verifiable credentials* que contenham dados de identificação pessoal e alertem o *holder* sempre que estes dados forem requeridos por terceiros.

Claims abstratas

Sempre que possível, o *issuer* deve optar pelo uso de atributos abstratos para retratar a informação presente nas *verifiable credentials*. Voltando de novo ao exemplo da idade mínima, as credenciais cujo objetivo é provar que o sujeito possui a idade mínima necessária para executar alguma ação devem retratar tal informação de forma binária (tem ou não tem) em vez de fornecer a data de nascimento do sujeito. Em *self-sovereign identity* tal é possível através do uso de provas de conhecimento zero, como foi explicado na subsecção 2.3.3.

Bearer Credentials

Uma *bearer credential* é nada mais do que uma *verifiable credential* capaz de dar acesso a algum recurso sem divulgar qualquer informação privada do *holder*. Para tal, a estratégia utilizada passa por não especificar o sujeito da credencial. Este tipo de credenciais é mais adequado para casos de uso nos quais não existe um elevado risco associado. Um exemplo de um caso de uso de *bearer credentials* seria um bilhete para um concerto.

Apesar deste tipo de credenciais não revelar o sujeito da credencial, o seu uso repetido em várias plataformas pode possibilitar a correlação. Por este motivo, os *issuers* de *bearer credentials* devem-se assegurar de que estas credenciais são de uso único (sempre que possível) e não contêm qualquer informação de identificação pessoal. Além disso, o *holder* deve ser notificado sempre que o seu *software* detetar que *bearer credentials* apresentam informação sensível ou que a combinação de várias *bearer credentials* apresenta risco de correlação.

Minimização dos dados

A quantidade de informação presente numa *verifiable credential* deve ser restrita ao mínimo necessário para que os *verifiers* sejam capazes de fornecerem o serviço. Além disso, os *issuers* devem atomizar a informação presente nas credenciais de modo a possibilitar o uso de divulgação seletiva. Para tal, em vez de emitir uma credencial com todo o conjunto de atributos do sujeito, o *issuer* deve optar por assinar cada um dos atributos presentes individualmente de modo a permitir que estes possam ser verificados individualmente pelo *verifier*, o que permite ao *holder* fornecer apenas os atributos relevantes para a obtenção do serviço disponibilizado pelo *verifier*. É importante referir que em alguns casos de uso é desejável que certos atributos sejam agregados de forma a impedir o seu uso individual como está exemplificado na subsecção 2.3.3, em "Divulgação seletiva e atributos negativos".

Fornecedores de armazenamento e mineração de dados

Entidades que fornecem serviços de armazenamento de credenciais podem estar a minerar informação pessoal e a vendê-la a organizações. Os *holders* devem estar cientes dos termos de serviço dos seus repositórios de credenciais, nomeadamente no que diz respeito à proteções contra correlação e mineração de dados. É possível mitigar a mineração dos dados através do uso de *software* para cifrar as *verifiable credentials* antes de as armazenar em serviços.

Verificações de validade

Para validar as credenciais, os *verifiers* realizam uma série de verificações. Estas verificações podem desencadear fugas de informação que violem a privacidade do *holder*. Por exemplo, a verificação da lista de revogação pode informar o *issuer* de que o *holder* está a usar a credencial para interagir com um determinado *verifier*. Deste modo, os *issuers* não devem utilizar listas de revogação únicas por credencial. O *software* do *holder* deve alertá-lo sempre que detetar informação que pode levar à violação da sua privacidade durante as verificações de validade.

2.7 Integração do SSI com esquemas existentes

Após serem discutidos os vários aspetos de um sistema SSI surge uma questão relevante: estes sistemas vão substituir completamente os esquemas de identidade existentes ou poderão ambos coexistir? Para responder a esta pergunta será usado o exemplo do eIDAS [52], um regulamento definido pela União Europeia sobre a identificação eletrónica, autenticação e serviços de confiança para transações eletrónicas. Este regulamento tem como objetivo assegurar que pessoas e empresas possam usar as suas identidades digitais para aceder a serviços públicos disponíveis noutros países da UE, bem como criar um mercado europeu interno de serviços de confiança, entre os quais assinaturas eletrónicas e autenticação em *websites* [53].

Dependendo de como a identidade digital será utilizada, a *framework* eIDAS pode declarar identidades digitais através duas formas distintas [54]:

- Autenticação através de um esquema de identificação eletrónica (eID);
- Geração de uma assinatura eletrónica através de certificados eletrónicos emitidos por autoridades de certificação qualificadas.

A primeira opção permite criar uma ligação entre um DID e uma pessoa do mundo real. Se o *verifier* apenas precisar de validar que o *holder* é capaz de se autenticar com o eID (i.e. para provar que é um cidadão europeu), esta opção é viável. Contudo, o *holder* não irá possuir qualquer prova de que possui determinados atributos devido à identificação eletrónica (eID) ser utilizada apenas para autenticar os utilizadores e não para fornecer declarações sobre uma entidade que possam ser verificadas por outras entidades, ou seja, o utilizador não será capaz de utilizar o eID para fornecer provas assinadas pelo eIDAS aos *verifiers*, tendo portanto de auto-declarar os atributos, o que por si só não é o suficiente na maior parte das ocasiões. Deste modo, esta opção não trará valor acrescido na emissão e apresentação de credenciais.

Por outro lado, a segunda opção baseia-se na utilização de certificados qualificados emitidos por autoridades de certificação qualificadas. Estes certificados apresentam um par de chaves associado, uma chave privada para assinar e uma chave pública para verificar a assinatura. Tal como foi explicado anteriormente, os sistemas de gestão de identidades descentralizados recorrem à criptografia de chave pública para efetuar a autenticação do *holder* face a um determinado DID. Esta ligação pode facilmente ser efetuada através da utilização do par de chaves associado ao certificado qualificado como o par de chaves associado a um DID, em alternativa à utilização do par de chaves gerado automaticamente pelo agente do utilizador. Esta estratégia permite criar uma ligação criptográfica entre o DID e o certificado e, consequentemente, sempre que o *holder* assinar algo com a chave privada associada ao seu DID, essa assinatura terá o mesmo estado que uma assinatura avançada

produzida por um certificado qualificado, de acordo com o regulamento eIDAS [54]. Estas assinaturas permitem ao *verifier* obter uma maior confiança legal relativamente aos dados assinados pelo *holder*. Esta ligação criptográfica entre o DID e o certificado qualificado é implicitamente garantida pela utilização do mesmo par de chaves, contudo é possível estabelecer uma relação explícita através da inserção do DID no respetivo certificado. Ao contrário da primeira opção, os atributos apresentados pelo *holder* podem ser validados por outras entidades através de uma verificação de validade do certificado qualificado criptograficamente ligado à chave pública associada ao DID desse *holder*. É importante indicar que apesar da chave pública ser acessível por qualquer entidade, o certificado correspondente não precisa de o ser, isto é, o *holder* pode manter o controlo sobre esse certificado e partilhá-lo apenas com entidades que precisem de verificar a sua identidade real. Apesar de tal ser possível, esta estratégia pode levar à exposição da ligação entre o DID e o respetivo proprietário, razão pela qual é aconselhado utilizar múltiplos DIDs, de modo a isolar partes da identidade por cada um desses DIDs, e minimizar assim exposição de dados pessoais caso a ligação entre um desses DID e o respetivo proprietário seja exposta.

Tendo em consideração o que foi abordado nesta secção é possível afirmar que os esquemas de identidades utilizados atualmente podem não só coexistir com sistemas baseados numa arquitetura SSI mas também complementá-los através do estabelecimento de uma relação confiável entre um DID e uma pessoa do mundo real.

3

IMPLEMENTAÇÃO DE UM SISTEMA DE GESTÃO DE IDENTIDADES DESCENTRALIZADO

No capítulo anterior foi apresentado o estado da arte, em particular as componentes que servem de pilar para uma arquitetura SSI e algumas recomendações de segurança e privacidade. Este capítulo visa detalhar o processo de implementação de um sistema de gestão de identidades descentralizado baseado numa arquitetura SSI e que vá de encontro aos vários aspetos abordados no capítulo anterior.

De modo a explicar a solução desenvolvida, este capítulo começa por ilustrar uma arquitetura que apresenta uma visão geral das várias componentes que constituem o sistema e como estas se relacionam entre si. De seguida explora detalhadamente cada uma dessas componentes, descrevendo em que consistem, qual é o seu objetivo num sistema SSI e quais as suas vantagens. Por conseguinte, o agente do utilizador será abordado nas secções 3.2 e 3.3, sendo a primeira secção responsável por explicar o agente desenvolvido num contexto individual e a segunda secção encarregue de apresentar a forma como o agente desenvolvido comunica com outros agentes. Posteriormente é abordado o repositório privado do agente do utilizador (*wallet*) no qual são armazenados todos os dados de caráter privado do utilizador, na secção 3.4. O capítulo termina com a explicação da componente *data registry*, na secção 3.5, que consiste numa *blockchain* desenvolvida pela comunidade Hyperledger.

Dada a complexidade do sistema, a secção A.5 (nos anexos) apresenta um tutorial de configuração do ambiente de modo a ser possível executar a aplicação. Para facilitar a implementação do sistema é também disponibilizado o *url* para uma máquina virtual que já contém o ambiente todo configurado.

Tal como foi ilustrado na figura 12, a arquitetura de um sistema SSI é constituída por três componentes principais:

- agente do utilizador;
- registo público descentralizado;
- repositório de credenciais e outros dados privados.

Adicionalmente, esta arquitetura engloba três entidades distintas: o *issuer*, o *holder* e o *verifier*, cada um deles com um ou mais agentes de utilizador que lhes permitem interagir com o sistema.

A arquitetura da solução desenvolvida segue o mesmo modelo, sendo o repositório privado substituído por uma *wallet* e o registo público descentralizado substituído por uma *blockchain*. Contudo, ao contrário da arquitetura apresentada na figura 12 na qual apenas o *holder* possuía um repositório privado, na arquitetura projetada para a solução também o *issuer* e o *verifier* possuem repositórios privados, necessários para armazenar dados privados tais como chaves criptográficas, definições de credenciais, entre outros. A figura 27 ilustra a arquitetura da solução implementada.

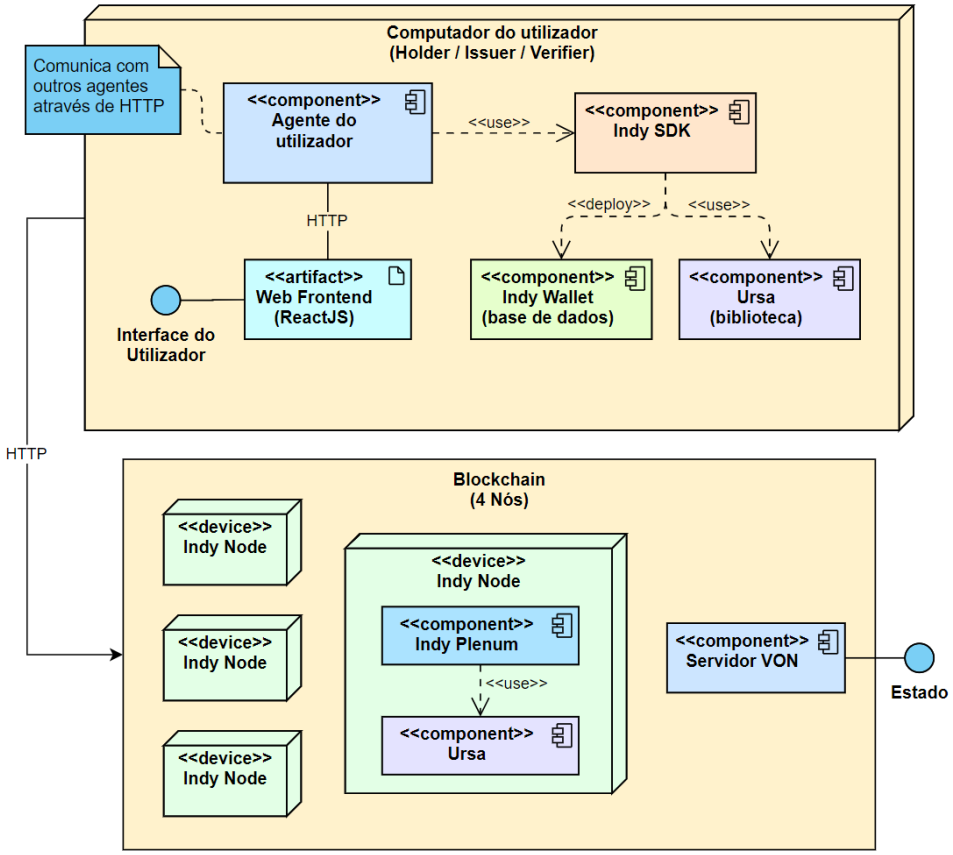


Figura 27: Arquitetura da solução desenvolvida

Nesta arquitetura podemos distinguir três componentes principais: o agente do utilizador, a *wallet* e a *blockchain* e ainda duas componentes adicionais: Indy SDK e Ursa. O agente do utilizador é composto por um servidor *backend* (NodeJS), com toda a lógica do agente, e por um servidor *frontend* (React) com a sua respetiva interface. A divisão do agente em duas componentes distintas é explicada na secção 3.2. Podemos também observar que o agente desenvolvido consegue comunicar com outros agentes através de HTTP. Este agente utiliza ainda duas componentes adicionais: Indy SDK e Ursa, duas bibliotecas responsáveis

por fornecer várias operações necessárias ao ecossistema. Já a *blockchain* implementada é composta por quatro nós, todos eles constituídos pelas componentes Indy Node, Indy Plenum e Ursa.

As componentes Indy SDK, Ursa, Indy Node e Indy Plenum são projetos que estão atualmente a ser desenvolvidos pela comunidade Hyperledger. Deste modo, o capítulo apresenta uma secção inicial, 3.1, que visa introduzir esta comunidade e explicar em que consiste cada um dos projetos indicados.

Wallet

No SSI todas as entidades precisam de armazenar chaves privadas, para assinar documentos e decifrar mensagens que recebem de outras entidades, bem como armazenar várias outras informações privadas. Por esta razão, cada entidade possui uma *wallet* à qual consegue aceder, através do seu agente de utilizador, de modo a armazenar segredos e a utilizá-los sempre que necessário. A subsecção 3.4 irá explicar mais detalhadamente as várias caraterísticas e funcionalidades da componente *wallet*, desenvolvida pela comunidade do Hyperledger [12]. Esta *wallet* foi desenvolvida tendo em vista o seu uso em sistemas de *self-sovereign identity*, o que faz com que apresente fortes medidas de segurança e privacidade. Na solução desenvolvida a componente *wallet* está armazenada no dispositivo que o *holder* utiliza para interagir com o sistema, tal como mostra a figura 27. Contudo, de maneira a aumentar a disponibilidade e a redundância da informação armazenada, é possível recorrer alternativa ou complementarmente a *wallets* armazenadas em serviços de *cloud*.

Agente do utilizador

O agente do utilizador foi desenvolvido com auxílio à biblioteca disponibilizada pelo Hyperledger, Indy SDK [55]. Esta biblioteca contém um vasto conjunto de operações necessárias a um sistema de gestão de identidades descentralizado tais como criação de esquemas, emissão e revogação de credenciais, e criação de apresentações verificáveis. Este agente tem como finalidade ajudar o seu proprietário a interagir com o sistema e com outros agentes para emitir/obter credenciais verificáveis e apresentar/verificar apresentações verificáveis. Deste modo, o agente do utilizador é responsável por estabelecer conexões seguras com outros agentes de modo a permitir o envio de mensagens protegidas, sendo esta segurança garantida pela utilização de uma infraestrutura de chave pública descentralizada (com o suporte da *blockchain*). Para comunicar com outros agentes, o agente desenvolvido disponibiliza uma porta através da qual recebe mensagens desses agentes, processando-as de modo a extrair o tipo da mensagem e agir em conformidade. Este agente suporta apenas comunicações HTTP, dado que são as mais comuns, sendo contudo possível dar suporte a vários tipos de comunicação para aumentar a capacidade de comunicação do agente.

A secção 3.2 indica o que são agentes de utilizadores, que tipos de agentes existem e quais as estrategias utilizadas para desenvolver o agente incorporado nesta solução, enquanto que a secção 3.3 explica como eles se comunicam entre si através de protocolos e de criptografia.

Blockchain

O papel de registo descentralizado é desempenhado por uma *blockchain* desenvolvida pelo Hyperledger Indy [56] especificamente para o caso de uso da identidade digital [57]. Cada nó desta *blockchain*, **Indy Node** [58]), possui uma componente, **Indy Plenum** [59], que contém toda a lógica referente ao funcionamento da *blockchain*, nomeadamente o protocolo de consenso. A secção 3.5 explica o funcionamento da *blockchain* desenvolvida pelo Hyperledger, nomeadamente a diferença entre os projetos Indy Node e Indy Plenum, o protocolo de consenso utilizado pelos nós para acordarem no estado atual da *blockchain* e as várias operações de escrita e de consulta disponibilizadas.

A figura 27 mostra a arquitetura da *blockchain* implementada. Tal como é possível observar, esta *blockchain* é constituída por quatro nós, baseados nos projetos **Indy Node** e **Indy Plenum**. De maneira a facilitar a integração dos **Indy Nodes** com as soluções de SSI de várias entidades terceiras, a comunidade Hyperledger desenvolveu um pacote para distribuições **debian** com várias funcionalidades integradas. Este pacote pode ser instalado através da execução sequencial dos comandos apresentados na figura 28.

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys CE7709D068DB5E88
sudo bash -c 'echo "deb https://repo.sovrin.org/deb xenial stable" >> /etc/apt/sources.list'
sudo apt-get update
sudo apt-get install indy-node
```

Figura 28: Comandos de instalação do pacote *Indy Node* [60]

Uma das funcionalidades incluídas neste pacote é a criação e inicialização automática de nós através da execução de comandos específicos. Aliando este pacote à utilização de *containers*, é possível criar múltiplos nós de forma simples e rápida. Para facilitar ainda mais a integração dos nós com soluções de SSI externas, a comunidade Hyperledger desenvolveu também um ficheiro de **Docker** [61] responsável por configurar o ambiente e inicializar uma *pool* de teste com quatro nós capazes de comunicarem entre si, sendo este o mínimo necessário para que a *blockchain* continue a funcionar corretamente face à falha de um dos nós.

Como o objetivo desta dissertação é efetuar uma prova de conceito, a estratégia adotada inicialmente passou pelo uso deste ficheiro de Docker, com apenas quatro nós, para implementar uma pequena *blockchain* capaz de atuar como *data registry* do sistema desenvolvido. Nesta estratégia todos os nós da *blockchain* são executados localmente dentro do mesmo *container*. Apesar de não ser uma representação ideal de um cenário real, esta estratégia

permitiu testar o funcionamento do sistema com uma instância da *blockchain* desenvolvida nos projetos **Indy Plenum** e **Indy Node**. Por outro lado, apesar desta implementação apenas possuir quatro nós, tal não apresenta qualquer problema uma vez que mais nós apenas iriam conferir maior segurança e maior tolerância a falhas. Além disso poderiam ser facilmente adicionados vários outros nós a qualquer momento através da submissão de transações para o *ledger* referente à *pool* dos nós, ou seja, o *ledger* que permite gerir todos os nós da *blockchain*. A subsecção 3.5.1 explora a divisão da *blockchain* Indy em múltiplos *ledgers* individuais (*domain*, *pool* e iconfig), cada um com um âmbito específico.

Mais tarde, a estratégia apresentada anteriormente foi substituída por uma implementação [62] desenvolvida pela Verifiable Organizations Network (VON) [63], que também se baseia nos projetos Indy Node e Indy Plenum, também com quatro nós. Em semelhança à implementação anterior, esta implementação recorre ao pacote *Indy Node* e ao uso do *docker* para criar os vários nós da *blockchain*. Contudo, cada nó está incluído no seu próprio *container*, ficando assim individualizado dos restantes, o que faz com que este cenário esteja mais próximo da realidade. Adicionalmente, esta implementação apresenta um servidor, desenvolvido pela Verifiable Organizations Network (VON), responsável por interagir com os nós da *blockchain* e disponibilizar informações referentes aos mesmos e ao estado dos vários *ledgers* que constituem *blockchain* (*domain*, *pool* e *config*), permitindo assim que qualquer indivíduo consiga consultar essas informações através de uma interface gráfica. Este servidor é independente da *blockchain*, o que permite que possa ser conectado a qualquer *blockchain* e fornecer uma interface gráfica para a consulta de dados dessa *blockchain*. Caso se pretendesse implementar uma *blockchain* diferente no futuro, seria possível e relativamente fácil conectar o servidor desenvolvido pela VON a essa *blockchain*.

3.1 HYPERLEDGER INDY

Dado que a solução desenvolvida integra algumas componentes concebidas pela comunidade Hyperledger, tais como a *blockchain* (projetos **Indy Plenum** e **Indy Node**), a biblioteca (projeto **Indy SDK** e implementações de técnicas criptográficas tais como esquemas de assinaturas (projeto **Ursa**), esta secção irá apresentar uma breve explicação dos vários projetos que estão atualmente a ser desenvolvidos pela comunidade e em que consiste cada um deles.

O Hyperledger Indy é um projeto *open source* que está a ser desenvolvido pela comunidade **Hyperledger**, e apoiado pela **Linux Foundation** [64], e implementa os padrões de identidade descentralizada definidos pelo W3C. Este projeto tem como base uma *blockchain* pública e *permissioned*, na qual qualquer utilizador pode efetuar operações de leitura mas apenas utilizadores com permissões podem efetuar operações de escrita, construída com o propósito de ser usada para gestão de identidades descentralizadas. Adicionalmente, este projeto disponibiliza ferramentas, bibliotecas e componentes reutilizáveis com o objetivo de fornecer

identidades digitais enraizadas em *blockchains* (ou outros registos distribuídos) de forma a que estas sejam interoperáveis.

Os projetos desenvolvidos por esta comunidade apresentaram uma elevada aprovação e já estão a ser utilizados atualmente por várias organizações. Em particular, a **Sovrin Foundation** [65], uma organização mundial sem fins lucrativos, implementou uma instância da *blockchain* desenvolvida pelo Hyperledger de modo a que esta possa ser utilizada como *data registry* em soluções desenvolvidas pelas várias entidades e organizações.

Através das suas várias caraterísticas e funcionalidades, o Hyperledger Indy é um projeto que [66] [67]:

- Apresenta um registo distribuído construído propositadamente para identidade descentralizada;
- É resistente à correlação por *design*;
- Possui identificadores descentralizados que são globalmente únicos e resolúveis (através de um *ledger*) sem necessidade de uma autoridade de resolução centralizada;
- Permite usar identificadores descentralizados de emparelhamento para criar relações seguras ponto-a-ponto entre quaisquer duas entidades;
- Utiliza declarações de atributos (*verifiable claims*) que seguem o formato interoperacional para a partilha de atributos entre identidade digitais de acordo com os padrões definidos pelo W3C;
- Disponibiliza o uso de divulgação seletiva e provas de conhecimento zero;
- Fornece ao utilizador o controlo sobre as suas credenciais;
- Dá suporte aos dez princípios de *self-sovereign identity* abordados na secção 2.2.1.

Dada a sua dimensão, o projeto Hyperledger Indy está dividido em várias componentes. Destas componentes, as principais são o **Indy Plenum** [59], o **Indy Node** [58], o **Indy SDK** [55], o **Aries Agent** [68] (anteriormente designado por Indy Agent) e o **Ursa** [69] (anteriormente designado por Indy Crypto). Desde a sua criação, o Hyperledger Indy tem sido desenvolvido como um único projeto. Contudo, partes desse projeto foram dividas em projetos separados por serem genéricas e poderem ser usadas numa grande variedade de outros projetos [70]. É o caso da biblioteca criptográfica, **Indy Crypto**, que mostrou potencial para ser usada em vários outros projetos devido ao seu caráter genérico e à relevância das técnicas criptográficas desenvolvidas, individualizando-se assim num novo projeto, designado por **Hyperledger Ursa** [69], no final de 2018. Mais tarde, em março de 2019, a componente **Indy Agent**, cuja finalidade é atuar como agente do utilizador, tornou-se um projeto separado designado **Hyperledger Aries** [71], também com o objetivo de poder ser utilizada noutros projetos, mais concretamente de modo a fornecer suporte a várias tecnologias descentralizadas.

As subsecções seguintes visam explicar de forma resumida em que consiste cada um dos projetos indicados.

3.1.1 *Indy Plenum e Indy Node*

Indy Plenum e Indy Node apresentam várias sobreposições, contudo são projetos distintos. De uma forma geral, o Indy Plenum contém todas as funcionalidades principais ligadas ao protocolo de consenso, ao *ledger*, e a outros aspetos relacionados com a parte criptográfica da *blockchain*. Por sua vez, o Indy Node é uma extensão ao Indy Plenum, com transações específicas do ambiente da identidade digital, tais como criação de esquemas e definições de credenciais, gestão de DIDs (designados por nyms no contexto da *blockchain* do Hyperledger), entre outras. Esta divisão deve-se ao facto de que a componente Indy Plenum amadureceu ao ponto de poder ser usada por várias *blockchains*, até mesmo fora do contexto da identidade descentralizada. A subsecção 3.5.1 detalha o registo distribuído (*blockchain*) bem como o protocolo de consenso utilizado e a parte criptográfica que dá suporte à *blockchain*. A subsecção 3.5.2 explica os vários tipos de transações existentes e o sistema de permissões de escrita no registo distribuído.

3.1.2 *Indy SDK*

O Indy SDK consiste numa biblioteca *c-callable* e num conjunto de *wrappers* em várias linguagens para essa biblioteca. Uma biblioteca *c-callable* é uma biblioteca escrita em C que permite construir APIs para várias outras linguagens diretamente por cima dessa biblioteca, o que elimina a necessidade de criar implementações para cada uma dessas linguagens. Atualmente esta biblioteca apresenta *wrappers* para Java, Python, iOS, NodeJS, .Net e Rust. A biblioteca Indy SDK contém várias operações afetas à criação de esquemas e credenciais, gestão de DIDs, entre outros, de acordo com os padrões propostos pelo W3C. Adicionalmente, implementa ainda uma *wallet* criptograficamente protegida, baseada em SQLite, que é armazenada do lado do utilizador (i.e. computador pessoal ou *smartphone*) e disponibiliza mecanismos que permitem substituir o núcleo desta *wallet* por uma implementação externa [72]. A arquitetura e funcionalidades de uma *wallet* no ecossistema Indy são exploradas em profundidade na subsecção 3.4.1.

3.1.3 *Aries Agent*

O Aries Agent representa todo o *software* responsável por interagir com o ecossistema em nome do utilizador através de comunicações ponto-a-ponto e com o auxílio da *blockchain*. O *software* deve ainda ser responsável por fornecer uma *wallet* segura ao utilizador e geri-la de acordo com a vontade do mesmo, criando, armazenando e transmitindo *verifiable claims*. Um agente de *self-sovereign identity* necessita de realizar operações tais como criar esquemas, credenciais ou provas, armazenar ou consultar registos na *wallet*, entre muitas outras. Estas

operações são disponibilizadas pela biblioteca Indy SDK, razão pela qual é comum que um Aries Agent inclua a biblioteca Indy SDK para realizar tais operações.

Apesar desta componente não estar a ser utilizada na solução desenvolvida, a comunidade que está a desenvolver os agentes nas várias linguagens de programação discute várias boas práticas e padrões de interoperabilidade de modo a que diferentes agentes sejam capazes de comunicarem entre si. Várias das sugestões sugeridas por esta comunidade foram seguidas para a implementação de algumas funcionalidades na solução desenvolvida tais como os protocolos de comunicação entre agentes, razão pela qual é importante mencionar este projeto.

3.1.4 *Ursa*

Por fim, a biblioteca Ursa contém várias operações criptográficas, entre as quais se destacam o esquemas de assinaturas *Boneh–Lynn–Shacham* (BLS), utilizado na *blockchain* para garantir que todos os nós consentem com o estado atual do registo, o esquemas de assinaturas *Camenisch-Lysyanskaya* (CL), um formato de assinatura de provas de conhecimento zero utilizado pelo *issuer* para assinar os atributos presentes na credencial do *holder*, e ainda a técnica de criptografia simétrica *Chacha20-Poly1305*, utilizada para cifrar os dados armazenados na *wallet*. Além destas técnicas, o projeto Ursa apresenta implementações de ECDSA, EdDSA, ECDH, AES-CBC, entre várias outras técnicas criptográficas que podem ser utilizadas fora do contexto de identidades digitais, razão pela qual esta biblioteca se individualizou num projeto isolado.

3.2 AGENTE DO UTILIZADOR

No *Self-sovereign identity*, pessoas e organizações não são, por norma, capazes de interagir diretamente entre si e com o ecossistema, seja para enviar, receber, consultar ou armazenar informação, ou efetuar as várias operações criptográficas necessárias à criação de credenciais, provas, entre outros [73]. Consequentemente, uma arquitetura SSI inclui agentes que atuam em nome dos utilizadores, efetuando todas as operações necessárias para que os respetivos utilizadores sejam capazes de interagir com o ecossistema e tendo sempre em consideração os seus melhores interesses. Para tal, estes agentes armazenam um conjunto de chaves criptográficas que o utilizador usa para provar autenticação e autorização e recorrem a protocolos de comunicação, que procuram garantir interoperabilidade, para comunicarem entre si.

Os agentes de SSI são normalmente divididos em duas categorias, tendo em conta a sua localização:

- agente de *cloud* (*cloud agent*);
- agente de ponto (*edge agent*).

Um agente de *cloud*, como o próprio nome indica, está hospedado na *cloud*, o que lhe permite estar constantemente disponível para receber e enviar mensagens bem como estabelecer conexões com outros agentes e entidades. Por outro lado, um *edge agent* é executado num dispositivo físico diretamente sob o controlo do seu dono (i.e. servidor de uma empresa ou telemóvel de um cidadão). Consequentemente, um agente de ponto fornece mais segurança enquanto que um agente de *cloud* consegue garantir uma maior disponibilidade uma vez que o agente pode estar constantemente ativo a comunicar com outros agentes.

Na solução desenvolvida, o agente implementado é do tipo *edge agent*, apesar de poder facilmente ser usado como *cloud agent*. Para possibilitar esta fácil transação de agente de ponto para agente de *cloud*, a solução desenvolvida separa o núcleo do agente (um servidor em NodeJS) da sua interface (uma aplicação em ReactJS), tal como é mostrado na figura 27. O servidor é responsável por toda a lógica afeta à gestão de dados, interações com outros agentes, execução de protocolos (i.e. protocolo de emissão de credenciais), entre outras operações. Esta separação permite que as duas partes do agente possam ser executadas diretamente na máquina física do utilizador (agente de ponto) ou que apenas a aplicação em ReactJS seja executada nessa mesma máquina, ficando o servidor com o núcleo do agente hospedado em *cloud*. Outra vantagem de separar o agente em núcleo (NodeJS) e interface (ReactJS) é que se mais tarde for necessário desenvolver uma aplicação para *smartphone* será apenas necessário desenvolver a nova componente da interface e conectá-la ao núcleo já desenvolvido. Neste descrito o agente é classificado como agente de *cloud* uma vez que o servidor do núcleo seria hospedado em *cloud* e a aplicação do *smartphone* iria comunicar com esse servidor.

Apesar do agente desenvolvido apresentar o servidor (NodeJS) e a interface (ReactJS) na mesma máquina, este foi projetado para poder também operar como agente de *cloud*. Deste modo, a comunicação entre o servidor e a interface é efetuada através de HTTP e recorre a *json web tokens* (JWT) [74] para efetuar a autenticação do dono do respetivo agente. Durante a implementação da autenticação surgiram várias soluções sobre como usar estes *tokens*:

- Inicialmente a estratégia passou por enviar um *token* para a aplicação da interface e armazenar esse *token* na *local storage* do navegador. O principal problema desta solução é a possibilidade de atacantes obterem o *token* através de um ataque de XSS (*Cross Site Scripting*) [75], no qual o atacante insere código malicioso na página e o navegador não tem maneira de identificar esse código como sendo malicioso. Ao ser executado, esse código é capaz de aceder às informações armazenadas na *local storage* e enviá-las para o atacante. Após obter acesso ao *token*, o atacante consegue personificar o utilizador;

- Para remover a vulnerabilidade de XSS, a segunda estratégia recorria às *cookies* do navegador para armazenar o *token* de acesso. Contudo, apesar desta estratégia ser consideravelmente mais segura, estava por sua vez sujeita a ataques de CSRF (*Cross*

Site Request Forgery) [76]. Nestes ataques o atacante consegue enganar o utilizador de forma a que este envie um pedido ao servidor sem que se aperceba da ação que está realmente a executar. Como a *cookie* é adicionada automaticamente ao pedido pelo navegador, o servidor irá validar o acesso do utilizador e executar a ação. Esta ação pode ser apagar a conta, transferir dinheiro ou, neste ecossistema, emitir uma credencial;

- A terceira e última estratégia é a mais completa e segura das três, não estando vulnerável a nenhum dos ataques indicados anteriormente. Esta estratégia baseia-se no uso de dois *tokens*, um para o acesso (*access token*) e outro para a atualização do primeiro (*refresh token*). O *token* de acesso é responsável por autenticar o utilizador. Por sua vez, o *token* de atualização é utilizado sempre que for necessário gerar um novo *token* de acesso. Nesta estratégia o servidor envia o *access token* no corpo da resposta e o *refresh token* numa *cookie*. Ao receber a resposta do servidor, a aplicação ReactJS armazena o *access token* na memória local recorrendo ao redux [77], uma biblioteca que permite gerir o estado global da aplicação e atuar como fonte única de verdade da aplicação. De modo a aumentar a segurança desta solução, cada *token* de acesso gerado possui uma validade de quinze minutos, sendo necessário utilizar o *token* de atualização sempre que a validade do *token* expirar. Por outro lado, sempre que o utilizador tentar efetuar um pedido ao servidor sem fornecer o *token* de acesso, se fornecer o *token* de atualização o servidor irá gerar um novo *token* de acesso e fornecê-lo à aplicação ReactJS. Em ambos os casos, após a aplicação ReactJS receber o novo *token* de acesso irá efetuar o mesmo pedido de forma automática, adicionando-lhe o *token* de acesso que recebeu do servidor. Se o pedido não incluir os *tokens* de acesso e de atualização, ou incluir *tokens* inválidos, o servidor irá responder com um erro de autenticação.

  Como o *token* de acesso está armazenado em memória (no redux), um atacante não consegue adquiri-lo através de ataques de XSS. Por sua vez, o *token* de atualização está armazenado na *cookie* e, consequentemente, está sujeito a ataques de CSRF. Contudo, a única ação do servidor que depende deste *token* é a ação responsável por atualizar o *token* de acesso, ou seja, mesmo que um atacante seja capaz de levar o utilizador a executar esta ação, apenas irá atualizar o *token* de acesso desse utilizador sem nunca ter acesso ao mesmo uma vez que o novo *token* de acesso será enviado para o utilizador. Deste modo, a estratégia apresentada não está sujeita a ataques de XSS ou de CSRF, sendo portanto a estratégia mais segura para efetuar a autenticação do utilizador. Adicionalmente esta estratégia atualiza o *token* de acesso de forma automática e transparente ao utilizador sempre que for necessário. O artigo *Authentication Using JWT and Refresh Token — Part 1* [78] apresenta informações mais detalhadas sobre a implementação desta estratégia no lado do servidor.

Ainda no segmento da comunicação, o aspeto mais importante é a comunicação entre diferentes agentes de utilizadores. Estes agentes podem comunicar entre si através de HTTP, WebSockets, Bluetooth, entre muitos outros métodos de comunicação. Na solução implementada os agentes comunicam sempre através de HTTP uma vez que este é o transporte mais utilizado a nível global e um dos objetivos do agente implementado é fornecer elevada interoperabilidade. É importante referir que todos os tipos de comunicação num sistema de *self-sovereign identity* são baseados em mensagens e a segurança destas mensagens é garantida por técnicas modernas de criptografia de chave pública aceites pela comunidade, o que fornece segurança independente ao transporte usado. Para os agentes comunicarem entre si de forma a que ambos consigam entender-se, existem protocolos que servem de padrão para dar resposta aos casos de uso em questão. Deste modo, na solução desenvolvida foram implementados três protocolos padrão definidos pela comunidade Hyperledger tendo em vista a standardização das interações num sistema SSI:

- Protocolo de comunicação;
- Protocolo de emissão da credencial;
- Protocolo de apresentação da prova.

A secção 3.3 explica detalhadamente a estratégia utilizada para a comunicação entre agentes, nomeadamente os protocolos que foram implementados no agente desenvolvido e as técnicas criptográficas utilizadas a nível das mensagens trocadas nestes protocolos.

As próximas subsecções irão abordar as estratégias implementadas no agente do utilizador para gestão de DIDs, a criação de esquemas e definições de credenciais, de registos de revogação e das próprias credenciais e apresentações verificáveis, e ainda a verificação de apresentações e a revogação de credenciais.

3.2.1 *Identificadores descentralizados*

Os identificadores descentralizados são um dos elementos mais importantes de um sistema de gestão de identidades descentralizado uma vez que são responsáveis por identificar os vários indivíduos no ecossistema. Adicionalmente, todos os atributos são emitidos sobre estes identificadores, ou seja, para um indivíduo provar que possui certos atributos emitidos sobre um DID tem de provar que esse DID está sob o seu controlo.

No início da escrita deste documento a comunidade do W3C apresentava apenas um tipo de DIDs: DIDs públicos. Contudo, durante o desenvolvimento desta dissertação a comunidade do Hyperledger sugeriu a definição de um outro tipo de DIDs: DIDs de emparelhamento (*peer DIDs*). DIDs de emparelhamento são utilizados em conexões privadas e armazenados unicamente na *wallet* dos agentes que interagem nessa conexão enquanto que DIDs públicos são armazenados na *blockchain* e utilizados para construir a imagem pública de uma entidade no ecossistema. Após analisar as vantagens da utilização dos DIDs

de emparelhamento no estabelecimento de conexões privadas entre agentes foi decidido que seria interessante implementar esta estratégia no agente do utilziador. Deste modo, o agente desenvolvido implementa dois tipos de identificadores descentralizados: **DIDs de emparelhamento** e **DIDs públicos**. A subsecção 3.3.1 explica mais detalhadamente cada um destes dois tipos de DIDs, quais as suas caraterísticas e em que situações se deve optar por um ou por outro.

Quanto à sua sintaxe, tanto os DIDs públicos como os DIDs de emparelhamento seguem a sintaxe padrão definida pelo W3C em *Decentralized Identifiers (DIDs) v1.0* [27]. As figuras 1 e 2 apresentadas na secção 2.2.2 ilustram as regras seguidas para a construção dos DIDs utilizados na solução implementada. A figura 29 apresenta um exemplo de um DID público enquanto que a figura 30 demonstra um exemplo de um DID de emparelhamento.

```
did:mybc:TuBaNHWJKXtCSnz8w2C4qa
```

Figura 29: DID público

```
did:peer:WEdD5eCyxEUZCHELPiCPod
```

Figura 30: DID de emparelhamento

Como é possível observar nas figuras apresentadas, ambos os identificadores indicam o esquema no qual estão inseridos ("did"), qual é o seu método, isto é, que *driver* deve ser utilizado para obter o documento do DID ("mybc" e "peer"), e qual é a especificação desse método, ou seja, qual é o endereço no qual o documento se encontra. Tal como foi explicado na secção 2.2.2, o *resolver* universal de DIDs verifica que *driver* deverá utilizar para obter o documento do DID através do método do DID. O método "mybc" corresponde à *blockchain* implementada nesta solução, ou seja, após verificar o método, o servidor irá consultar o documento com o identificador "TuBaNHWJKXtCSnz8w2C4qa" na *blockchain* desenvolvida. Já o método "peer" indica que o identificador corresponde a um DID de emparelhamento e que o respetivo documento está portanto armazenado localmente, na *wallet* do agente. Dado que o agente desenvolvido tem como objetivo efetuar uma prova de conceito, apenas disponibiliza o uso dos métodos "peer" e "mybc". Contudo, está preparado para incluir outros métodos, tanto de *blockchains* como de outros registos descentralizados.

Documentos de identificadores descentralizados

Cada identificador descentralizado possui um documento associado que contém vários dados referentes ao identificador tais como chaves públicas de autorização e autenticação, serviços disponibilizados e os respetivos *endpoints* desses serviços, entre outros. A secção 2.2.3 refere algumas das propriedades mais usuais destes documentos. Na solução implementada o agente do utilizador cria documentos apenas com as propriedades **contexto**, **sujeito**, **chave pública**, **autenticação** e **serviços** uma vez que são as propriedades mais relevantes à interação com o ecossistema. A figura 31 demonstra o documento associado ao DID público apresentado anteriormente, *did:mybc:TuBaNHWJKXtCSnz8w2C4qa*.

```
{
    "@context": "https://w3id.org/did/v1",
    "id": "did:mybc:TuBaNHWJKXtCSnz8w2C4qa",
    "publicKey": [
        {
            "controller": "did:mybc:TuBaNHWJKXtCSnz8w2C4qa",
            "id": "did:mybc:TuBaNHWJKXtCSnz8w2C4qa#1",
            "publicKeyBase58": "FfMSUfKQqUh7dEJng1tEuppXMvrD1Ef6RScR3jgZykwY",
            "type": "Ed25519VerificationKey2018"
        }
    ],
    "authentication": [
        "did:mybc:TuBaNHWJKXtCSnz8w2C4qa#1"
    ],
    "service": [
        {
            "id": "did:mybc:TuBaNHWJKXtCSnz8w2C4qa#agent",
            "recipientKeys": [
                "FfMSUfKQqUh7dEJng1tEuppXMvrD1Ef6RScR3jgZykwY"
            ],
            "routingKeys": [],
            "serviceEndpoint": "http://localhost:5001",
            "type": "agent"
        }
    ]
}
```


Figura 31: Documento do DID *did:mybc:TuBaNHWJKXtCSnz8w2C4qa*

Nesta figura é possível observar que o documento segue o esquema definido em "https://w3id.org/did/v1", tal como está indicado na propriedade **contexto** (*@context*). A propriedade **id** indica o identificador descentralizado ao qual o documento pertence. De seguida estão apresentadas a propriedade **chave pública** (*publicKey*), uma lista de chaves públicas de autorização, e a propriedade **authentication**, uma lista de chave públicas de autenticação. Nas chaves públicas de autenticação está presente uma referência à chave pública de autorização indicada no mesmo documento. Estas referências permitem utilizar chaves públicas presentes na lista de autorização sem ser necessário reescrevê-las. No caso apresentado, a chave pública com o identificador *did:mybc:TuBaNHWJKXtCSnz8w2C4qa1* é utilizada tanto para autorização como para autenticação. Por fim, a propriedade **serviços** (*service*) descreve todos os serviços disponibilizados pelo DID. Neste exemplo o documento apenas possui informações sobre o serviço do agente do utilizador responsável por receber as comunicações destinadas ao dono do respetivo DID, nomeadamente a chave pública que deverá ser utilizada para cifrar as mensagens e o ponto de acesso ao serviço. Este serviço apresenta ainda uma lista de chaves de reencaminhamento (*routingKeys*), que é utilizada

para enviar uma mensagem através de um conjunto de mediadores. Caso esta lista esteja vazia, a mensagem é enviada diretamente para o destinatário, ou seja, o agente do utilizador. Apesar desta funcionalidade não ter sido implementada na solução desenvolvida, a secção 3.3.5 explica o mecanismo de encaminhamento de mensagens através de mediadores e *relays* que seria utilizado para desenvolver esta funcionalidade.

Documentos de identificadores descentralizados são armazenados em diferentes localizações dependendo da natureza dos respetivos DIDs. Deste modo, documentos de DIDs públicos são armazenados na *blockchain* enquanto que documentos de DIDs privados são colocados na *wallet* do seu respetivo proprietário.

3.2.2 *Esquemas e definições de credenciais verificáveis*

Uma credencial verificável corresponde a um conjunto de atributos e metadados assinados por um *issuer*. O *issuer* pode optar por assinar cada atributo individualmente, permitindo que o *holder* possa utilizar esses atributos de forma singular na construção das apresentações verificáveis, ou assinar conjuntos de atributos, obrigando assim o *holder* a apresentar esses atributos em conjunto de forma a que a prova seja válida.

Todas as credenciais seguem esquemas de dados que indicam quais os atributos que estas devem conter. Deste modo não é possível criar uma credencial sem fornecer um esquema válido. Estes esquemas são armazenados em registos descentralizados (i.e. *blockchain*) de modo a garantir a sua integridade. Na solução implementada os esquemas são armazenados na *blockchain* incluída nesta mesma solução e apenas podem ser adicionados pelas entidades mais confiáveis do sistema, isto é, *trustees*, *stewards* e *endorsers*. Após serem adicionados, não podem ser editados por nenhuma entidade. A subsecção 3.5.2 explica detalhadamente os vários papéis que as entidades podem possuir na *blockchain* utilizada bem como o sistema de permissões em vigor. A figura 32 ilustra um esquema simples que representa um cartão de identificação de uma pessoa com apenas dois atributos, o nome e a idade.

```
{
    "id": "schema:mybc:did:mybc:Th7MpTaRZVRYnPiabds81Y:2:identification_card:1.0",
    "name": "identification_card",
    "ver": "1.0",
    "version": "1.0",
    "attrNames": ["name", "age"]
}
```

Figura 32: Estrutura de um esquema de uma credencial

Qualquer *issuer* que pretenda emitir credenciais baseadas num dado esquema deverá primeiramente criar a respetiva definição da credencial, que indica o DID do *issuer* e qual o esquema que este irá utilizar para emitir as credenciais, bem como quais as chaves

criptográficas que serão usadas para assinar essas credenciais. É também na definição da credencial que os *issuers* indicam se as credenciais irão ou não suportar revogação. Caso optem por incorporar a possibilidade de revogação de credenciais, os *issuers* devem ainda gerar um registo de revogação após criarem a definição da credencial. A subsecção 3.2.3 explica o mecanismo de revogação utilizado.

A figura 33 mostra um exemplo da definição de uma credencial baseada no esquema apresentado acima.


```
{
  "id": "creddef:...:c727cf04-fffc-4513-a1cf-14255f4edce4",
  "schemaId": "118",
  "type": "CL",
  "tag": "c727cf04-fffc-4513-a1cf-14255f4edce4",
  "value": {
    "primary": {
      "n": "10022858066865115869291478076 5...",
      "s": "19624730044964992381425246552 3...",
      "r": {
        "master_secret": "350938093289543103383642483660...",
        "name": "82393606776241757085506096495 9...",
        "age": "71676144052303098162546405912 1..."
      },
      "rctxt": "73460272736287587320364482473 6...",
      "z": "49743581359826017341090375148 5..."
    },
    "revocation": {
      "g": "1 17C05...BE5B5 1 03BEA...A8270 2 095E4...7A8A8",
      "g_dash": "1 1A378...0C155 1 13708...F3606 1 1E0A1...D95FD ...",
      "h": "1 0B664...0860A 1 05261...68B79 2 095E4...7A8A8",
      "h0": "1 04BF4...C8A06 1 03AE5...85559 2 095E4...7A8A8",
      "h1": "1 08D42...09DA0 1 0DD57...5C865 2 095E4...7A8A8",
      "h2": "1 158E3...7033B 1 15B10...C8595 2 095E4...7A8A8",
      "htilde": "1 1698C...E627C 1 010AC...FBBF9 2 095E4...7A8A8",
      "h_cap": "1 1E27B...59375 1 0C687...8D9BA 1 232C7...6B058 ...",
      "u": "1 02D74...30AF1 1 0EA4B...AA8EA 1 0FA78...64F65 ...",
      "pk": "1 24A36...B07B8 1 08594...899FD 2 095E4...7A8A8",
      "y": "1 00DB7...6144F 1 0D86F...2B84E 1 18F3F...CD7B8 ..."
    }
  }
}
```


Figura 33: Estrutura de uma definição de uma credencial que suporta revogação

Apesar do identificador do esquema incluído nesta estrutura ser diferente do identificador do esquema apresentado anteriormente, ambos correspondem ao mesmo esquema. O

identificador utilizado na definição da credencial corresponde ao identificador atribuído ao esquema pela *blockchain* no momento da sua adição à mesma.

A definição da credencial apresentada é do tipo "CL", ou seja, recorre ao esquema de assinaturas Camenisch-Lysyanskaya para gerar provas sobre os atributos das credenciais. Esta técnica criptográfica permite utilizar provas de conhecimento zero para provar a posse de um determinado atributo sem fornecer informações concretas desse atributo, ou seja, um *holder* pode provar que possui mais de dezoito anos sem divulgar a sua idade em concreto. Os valores presentes no atributo "primary" correspondem à chave pública associada à chave privada que o *issuer* utiliza para assinar as credenciais referentes a esta definição da credencial. Por sua vez, a chave privada é armazenada na *wallet* do *issuer*, dentro da componente dos segredos, tal como é explicado na secção 3.4.1.

Continuando a análise a esta definição da credencial podemos inferir que o *issuer* optou por incorporar revogação de credenciais na sua definição da credencial devido à presença do atributo "revocation", que contém um conjunto de pontos de curvas elípticas referentes ao esquema de assinaturas CKS [79], a técnica criptográfica baseada em acumuladores criptográficos utilizada para monitorizar o estado de revogação das credenciais verificáveis e gerar provas de não revogação. Os atributos "g" e "g_dash" são geradores de números pseudo-aleatórios criptograficamente seguros. Os restantes atributos constituem a chave pública do *issuer*. A chave privada correspondente é armazenada na *wallet* do *issuer*, também dentro da componente dos segredos. Caso o *issuer* optasse por não incorporar revogação, a definição da credencial seria semelhante à que foi apresentada mas não iria conter o atributo "revocation" nem seria utilizada a técnica criptográfica de assinaturas CKS.

Utilizando ambas as chaves públicas, os *verifiers* são capazes de validar tanto a prova de posse dos atributos como a prova de não revogação fornecidas pelo *holder* através das apresentações verificáveis, que serão discutidas na subsecção 3.2.5.

3.2.3 *Revogação de credenciais*

Antes de discutir as estruturas das credenciais e apresentações verificáveis, é importante explicar como funciona o mecanismo de revogação de credenciais. Para implementar o mecanismo de revogação de credenciais a comunidade do Hyperledger recorreu ao uso de **acumuladores criptográficos**. Um acumulador criptográfico pode ser interpretado como o resultado da multiplicação de vários números. Na equação $a * b * c * d = e$ o acumulador é o **e**. Substituindo letras por números, se a=2, b=3, c=5 e d=7, então o valor do acumulador **e** será 210. Se pretendermos remover o fator **b** ao acumulador **e** basta apenas dividir o acumulador pelo valor de b, ou seja, $210/3$. Consequentemente, o valor do acumulador passa então a ser 70, ou seja, o produto da multiplicação dos restantes fatores [80].

Aplicando acumuladores criptográficos ao contexto de credenciais verificáveis, b=3 seria o valor privado correspondente a uma dada credencial, conhecido apenas pelo dono e

pelo emissor da credencial. O produto dos restantes fatores (a * c * d) é designado por ***witness***. Por sua vez, o valor do acumulador corresponde à multiplicação de todos os fatores associados a credenciais que ainda não foram revogadas. Esta caraterística é interessante pois permite que uma entidade seja capaz de provar matematicamente que é capaz de derivar o valor do acumulador através do valor privado associado à sua credencial (o valor de b) sem revelar qualquer informação relativamente a esse valor privado, ou seja, essa entidade consegue provar que o valor privado associado à sua credencial ainda está a contribuir para o valor do acumulador e, consequentemente, que essa credencial ainda não foi revogada.

No exemplo apresentado são usados apenas quatro fatores, todos eles com valores reduzidos. Para esta técnica ser útil e fiável na revogação de credenciais, os acumuladores não podem ser revertíveis, isto é, a única maneira de obter o valor do acumulador é multiplicando os vários fatores. De modo a satisfazer essa condição é usada aritmética modular e os fatores apresentam valores primos extremamente elevados.

Registos de revogação

Na parte introdutória desta secção é mencionado que cada credencial que suporta revogação possui um fator associado, que é usado para determinar o acumulador, e que o processo de revogar uma credencial se baseia na exclusão do fator associado a essa credencial no calculo do acumulador. Apesar de cada credencial (que suporta revogação) possuir um fator associado, o valor inserido na credencial não é o fator em si mas um índice que referencia o fator em questão. Deste modo é necessário algo que seja capaz de estabelecer a associação entre índices e fatores, o ***tails file***. O *tails file* é um ficheiro binário criado pelo *issuer* que contém todos os fatores associados a uma determinada definição de credencial sendo estes fatores atribuídos às credenciais emitidas sobre essa definição de credencial de modo a suportarem revogação. Todos os fatores presentes no ficheiro *tails file* são gerados aleatoriamente aquando da sua criação e o seu conteúdo nunca é alterado. Este ficheiro deve estar disponível publicamente de modo a que qualquer entidade o possa consultar (i.e. numa *blockchain*). Dado que o valor inserido em cada credencial é o índice correspondente a um determinado fator, a relação entre este índice e a credencial deve ser conhecida apenas pelo *issuer* que emitiu a credencial e pelo dono da mesma, ou seja, o índice passa então a ser o valor secreto inserido na credencial. A figura 34 ilustra a ligação entre índices, fatores e o acumulador, sendo o último a multiplicação de todos os fatores com a exclusão dos fatores associados às credenciais revogadas.

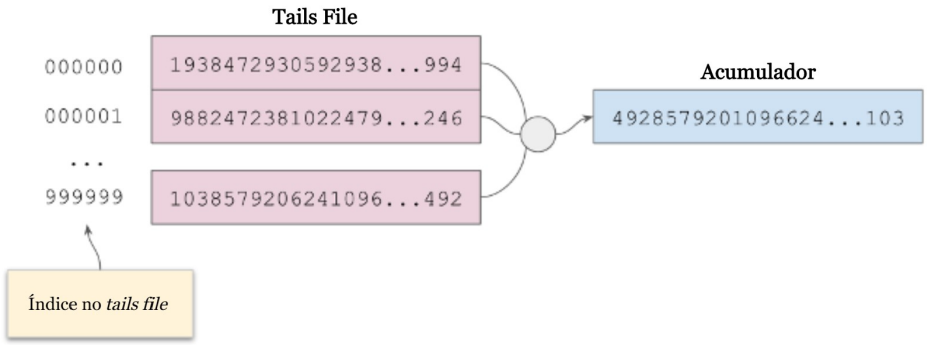


Figura 34: *Tails file* (Adaptado de [80])

Antes do *issuer* poder emitir credenciais que suportem revogação deve primeiro criar o *tails file*, tornando-o publicamente acessível, e um registo de revogação referente à definição da credencial, que deverá publicar na *blockchain*. Este registo de revogação indica o identificador da definição da credencial à qual está associado, a técnica utilizada para a revogação, a estratégia que deverá ser utilizada na emissão de credenciais e qual a capacidade de credenciais que podem ser associadas ao registo de revogação, o URI através do qual é possível obter o *tails file* e a *hash* do *tails file* de modo a ser possível validar a sua integridade, tal como mostra a figura 35.

```
{
  "credDefId": "creddef:...:118:c727cf04-fffc-4513-a1cf-14255f4edce4",
  "id": "revreg:...:25bc0bd3-4c44-45b5-86ec-32078c4e52a7",
  "revocDefType": "CL_ACCUM",
  "tag": "25bc0bd3-4c44-45b5-86ec-32078c4e52a7",
  "value": {
    "issuanceType": "ISSUANCE_BY_DEFAULT",
    "maxCredNum": 1000,
    "publicKeys": {
      "accumKey": {
        "z": "1 0ECFEA4DD3DB0DC24068496A8F7BD83F86CAD4BA08733FD9701C6F5656F3DF16 1
             12FE429D466F3B6BD3A3CB747D77E1005FF90AE38A6A7C32066340B7165A6102 ... 1
             0F3B6C1791EE0BCB8BC9C45BF571407C0AD5805737BA22763E6C02537580BC45"
      }
    }
    "tailsHash": "CpmRPd1sb5YcN59EGFYHXXc1Z7wEZjaDCWVqp1Wyikyh",
    "tailsLocation": "/tmp/indy_acme_tails/CpmRPd1sb5YcN59EGFYHXXc1Z7wEZjaDCWVqp1Wyikyh"
  },
  "ver": "1.0"
}
```

Figura 35: Estrutura da definição de um registo de revogação

O registo de revogação apresentado na figura anterior está associado à definição da credencial apresentada na figura 33 da subsecção anterior. Cada registo de revogação possui um *tails file* com um conjunto finito de índices. Consequentemente, se o *issuer* desejar adicionar mais índices pode simplesmente criar um novo registo de revogação, com o respetivo *tails file*, e associá-lo à definição da credencial.

Relativamente à estratégia utilizada na emissão de credenciais, existem duas estratégias: **emissão por padrão** e **emissão por pedido**. Na emissão por padrão é assumido que todas as credenciais são emitidas aquando à criação do registo de revogação, ou seja, o acumulador é calculado sobre todos os índices do *tails file*, o que faz com que o registo de revogação apenas seja atualizado na execução de ações de revogação. Por outro lado, na emissão por pedido nenhum índice contribui para o calculo do acumulador pelo que o seu valor inicial é 1. A primeira estratégia deve ser utilizada quando é expectável que o número de operações de revogação seja inferior ao número de operações de emissão.

Adicionalmente, o *issuer* deve ainda publicar no *ledger* o valor atual do acumulador criptográfico que deve ser utilizado para verificar o estado de revogação das credenciais como é possível ver na figura 36.

```
{
    "value": {
        "accum": "21 147C7DC231097DDD68B5C76340B2AF56099B91102B9E7276BA45C942360956F2B 21
            139A4289C07EE745DFC533D13073E22EAC31465DC607F8C2C2828C230B582D05E ... 4
            277B65CCE17341DDF1BAAFF84B67AC9C61CA783033688FA289CE3F2077449634"
    }
}
```

Figura 36: Acumulador do registo de revogação

Revogação de credenciais

No processo de revogação de uma credencial o *tails file* permanece inalterado, sendo apenas calculado o novo valor do acumulador com a exclusão do fator cujo índice está associado à credencial revogada, ou seja, apenas o registo do acumulador publicado na *blockchain* é alterado. Do mesmo modo, o *issuer* pode revogar múltiplas credenciais calculando simplesmente o novo valor do acumulador com a exclusão de todos os fatores associados às credencias revogadas. Contudo, esta variação do valor do acumulador faz com que o *holder* não possa usar o valor de *witness* que o *issuer* lhe forneceu inicialmente (produto dos fatores associados a todas as credenciais, incluindo as credenciais já revogadas) para calcular diretamente o novo valor do acumulador. Consequentemente, o *issuer* deve também publicar o *witness delta*, que representa a variação do valor de *witness* desde a criação do registo de revogação, de modo a que todos os *holders* sejam capazes de ajustar o valor do *witness* que o *issuer* lhes forneceu inicialmente, tendo em conta as credenciais revogadas, e derivar o valor correto do acumulador.

O acumulador e a variação do *witness* (*witness delta*) devem ser atualizados periodicamente ou conforme seja necessário de modo a conter informação atualizada sobre as credenciais revogadas. Para tal o *issuer* pode atualizar o acumulador e o delta sempre que revogar uma credencial, uma vez por dia (removendo todas as credenciais pretendidas de uma só vez), ou conforme lhe seja mais prático.

Validação de credenciais

Se o *holder* utilizar uma credencial que suporte revogação para provar possuir certos atributos deve apresentar também uma prova de não revogação, isto é, de que a credencial que está a utilizar na prova não foi revogada. Na solução desenvolvida todas estas provas são efetuadas automaticamente pelo agente do utilizador pelo que este não precisa de se preocupar com aspetos criptográficos durante a construção de apresentações verificáveis. Com a técnica dos acumuladores criptográficos, o *holder* prova que a credencial não foi revogada se conseguir provar matematicamente que consegue derivar o valor do acumulador através da multiplicação do fator que conhece com o produto dos restantes fatores ativos, ou seja, fatores que não estão associados a credenciais revogadas. Uma vez que o *holder* não conhece os fatores ativos, utiliza o *witness* que o *issuer* lhe forneceu inicialmente, que corresponde ao valor da multiplicação de todos os fatores exceto o dele. Contudo, a multiplicação do valor deste *witness* pelo fator associado à sua credencial resulta no acumulador inicial antes de qualquer credencial ter sido revogada, ou seja, a sua prova irá falhar uma vez que o valor do acumulador foi alterado. É portanto fulcral que o *issuer* atualize a variação do *witness* (*witness delta*) sempre que atualizar o acumulador de modo a que os *holders* consigam ajustar o valor do *witness* e conseguirem assim derivar o valor correto do acumulador. Durante esta prova o *holder* não divulga o valor do seu fator ao *verifier*.

Considerações finais

A técnica de revogação baseada em acumuladores permite que os *issuers* possam associar múltiplos registos de revogação a uma única definição de credencial, revogar múltiplas credenciais através de uma simples alteração de dois números na *blockchain*, **acumulador** e ***witness delta***, o que torna o processo de revogação bastante eficiente e ainda reverter revogações através de simples multiplicações do acumulador pelos fatores associados às credenciais revogadas. Por outro lado, os *holders* podem demonstrar a prova de não revogação sem reduzirem a sua privacidade dado que não é possível correlacionar o *holder* através do identificador da credencial ou do índice do fator de revogação. Por fim, os *verifiers* não precisam de contactar os *issuers* para verificar se certas credenciais foram revogadas, e a verificação das provas de não revogação geradas pelos *holders* é extremamente fácil, rápida e barata.

3.2.4 *Credenciais verificáveis*

Após a criação da definição da credencial e, se necessário, do registo de revogação, o *issuer* pode finalmente começar a emitir credenciais verificáveis. A figura 37 mostra a estrutura de uma credencial verificável emitida através da definição da credencial ilustrada na figura 33.

```
{
  schema_id: "schema:mybc:did:mybc:Th7MpTaRZVRYnPiabds81Y:2:identification_card:1.0",
  cred_def_id: "creddef:...:c727cf04-fffc-4513-a1cf-14255f4edce4",
  rev_reg_id: "revreg:...:46d5b1b8-5969-4181-99d7-8aec6d7121e2",
  values: {
    age: { raw: "23", encoded: "23" },
    name: { raw: "Jon", encoded: "4307168083...4218340983"}
  },
  signature: {
    p_credential: {
      m_2: "353151046837968621381167058800...",
      a: "825684911437287066190078790279...",
      e: "259344723055062059907025491480...",
      v: "604348989985369035181261319997..."
    },
    r_credential: {
      sigma: "1 239EA...2CE3A 1 1E78E...2F12B 2 095E4...7A8A8",
      c: "019EA760AE2CDB59E6BF6CDD1289E3...",
      vr_prime_prime: "19FB849DBD650CB6DBD022E23A5378...",
      witness_signature: [Object],
      g_i: "1 09611...5AEE7 1 03C0C...3E191 2 095E4...7A8A8",
      i: 1,
      m2: "07CEC39B815BF65EAED7D17B82C5158..."
    }
  },
  signature_correctness_proof: {
    se: "2018527210814820024357369392 27...",
    c: "617669126379245590734067334966..."
  },
  rev_reg: {
    accum: "21 12BF2...6512F 21 12693...50C67 ... 4 2CA66...214E6"
  },
  witness: {
    omega: "21 1293E...5DF03 21 10F72...E5F66 ... 4 2DD84...DAECD"
  }
}
```

Figura 37: Estrutura de uma credencial verificável

Nesta figura é possível observar um conjunto de metadados tais como os identificadores do esquema e da definição da credencial utilizados. É também possível inferir que esta

credencial suporta revogação devido à presença de metadados como o identificador do registo de revogação, o acumulador e o *witness*.

É também possível observar a presença da assinatura do *issuer* com os atributos "p_credential" e "r_credential". O primeiro atributo corresponde à assinatura efetuada sobre a **credencial primária**, ou seja, a credencial que contém as *claims* atribuídas ao *holder*. O segundo atributo corresponde à assinatura efetuada sobre a **credencial de não revogação**, sendo portanto utilizada para provar que a credencial primária ainda não foi revogada. Ao gerar uma apresentação verificável, o *holder* irá utilizar estes atributos para a construção da prova.

É importante indicar que as credenciais são armazenadas dentro da componente dos segredos da *wallet*. Uma vez que os segredos são mantidos num local seguro e nunca são expostos ao mundo exterior, são criados objetos que apenas contêm uma pequena parte das credenciais de modo a permitir que os *holders* possam consultar as suas credenciais. Estes objetos encontram-se armazenados na componente dos não segredos da *wallet*. A figura 38 mostra a estrutura destes objetos.


```
{
  "attrs": {
    "name": "Jon",
    "age": "23"
  },
  "cred_def_id": "creddef:...:c727cf04-fffc-4513-a1cf-14255f4edce4",
  "cred_rev_id": "2",
  "referent": "e1d9cf4b-1ecf-4f4f-9a82-d097bfe776a5",
  "rev_reg_id": "revreg:...:46d5b1b8-5969-4181-99d7-8aec6d7121e2",
  "schema_id": "schema:mybc:did:mybc:Th7MpTaRZVRYnPiabds81Y:2:identification_card:1.0"
}
```


Figura 38: Estrutura da credencial verificável armazenada na *wallet*

3.2.5 *Apresentações Verificáveis*

Apresentações verificáveis são estruturas de dados capazes de provar que um *holder* possui certos atributos e que estes ainda não foram revogados. Para tal, o *holder* pode recorrer à utilização do mecanismo de divulgação seletiva, ou seja, pode agrupar subconjuntos de atributos de várias credenciais numa única apresentação da maneira que achar mais apropriado, gerando as provas apropriadas para cada um desses atributos. A figura 39 mostra a estrutura de uma apresentação verificável com as respetivas provas de posse dos atributos e de não revogação. Dado que esta estrutura apresenta uma dimensão elevada, a prova primária (prova de posse dos atributos) e a prova de não revogação são apresentadas nas figuras 40 e 41, respetivamente.

```
{
  "proof": {
    "proofs": [
      {
        "primary_proof": { ... },
        "non_revoc_proof": { ... }
      }
    ],
    ...
  },
  "requested_proof": {
    "revealed_attrs": {
      "attribute1": {
        "sub_proof_index": 0,
        "raw": "Jon",
        "encoded": "43071680839755638477115980806 7..."
      }
    },
    "self_attested_attrs": {},
    "unrevealed_attrs": {},
    "predicates": {
      "predicate1": {
        "sub_proof_index": 0
      }
    }
  },
  "identifiers":[
    {
      "schema_id": "schema:mybc:did:mybc:Th7MpTaRZVRYnPiabds81Y:2:identification_card:1.0",
      "cred_def_id": "creddef:...:c727cf04-fffc-4513-a1cf-14255f4edce4",
      "rev_reg_id": "revreg:...:46d5b1b8-5969-4181-99d7-8aec6d7121e2",
      "timestamp": 1607468936
    }
  ]
}
```

Figura 39: Estrutura de uma apresentação verificável

Nesta figura é possível observar os atributos que foram requisitados pelo *verifier* ("requested_proof") e os identificadores das várias estruturas de dados utilizadas para gerar as credenciais verificáveis às quais os atributos presentes na apresentação verificável pertencem. É através destes identificadores que o *verifier* irá validar as provas apresentadas pelo *holder*.

Uma das provas que o *holder* inclui na apresentação verificável é a prova primária, ou seja, a prova que atesta que o *holder* é detentor os atributos apresentados. A figura 40 mostra a estrutura de uma prova primária com os atributos nome e idade. Nesta prova o *holder*

está a declarar que o seu nome é "Jon", através do atributo "eq_proof", e que possui mais de dezoito anos, através do primeiro elemento do atributo "ge_proofs".


```
"primary_proof": {
  "eq_proof": {
    "revealed_attrs": {
      "name": "43071680839755638477115980806 7..."
    },
    "a_prime": "36828430845595151076553462879 3...",
    "e": "193318900696657403670256666627...",
    "v": "129239511969463291327727801806...",
    "m":{
      "age": "14265204852895570777531092185 0...",
      "master_secret": "79672612320741378829203989342 5..."
    },
    "m2": "13748857052055062860377228364 0..."
  },
  "ge_proofs": [
    {
      "u": {
        "0": "26972764428528792934398753851 6...", "1": "15707140947192443673519813718 3...",
        "2": "11913767092189389001165745593 6...", "3": "62258700885003412888469232025 5..."
      },
      "r":{
        "0": "11860295122062439724640155481 1...", "1": "62961464961458980389183289310 5...",
        "2": "14527878763505644268763741592 5...", "3": "10670345315216466673962692300 2...",
        "DELTA": "58508673654866481209605234787 1..."
      }
      "mj": "14265204852895570777531092185 0...",
      "alpha":"77004147076517388473976664945 6...",
      "t":{
        "0": "83240027558467064547859747019 0...", "1": "46579011147193328050819208855 3...",
        "2": "14399878631021344839798067144 0...", "3": "48882999778617752838722161174 0...",
        "DELTA": "31096582569282775961077002..."
    },
      "predicate":{
        "attr_name":"age",
        "p_type":"GE",
        "value":18
      }
    }
  ]
}
```


Figura 40: Prova de posse dos atributos

Além da prova primária, o *holder* precisa também de apresentar uma prova que confirme que os atributos da prova primária não foram revogados. A figura 41 apresenta a estrutura de dados referentes a uma prova de não revogação sobre as declarações efetuadas na prova primária.


```
"non_revoc_proof":{
  "x_list":{
    "rho":"10304D4B4731214B7603C1AC96CBE7EC8FA6604D61AF6B0B8DF29D2D1A489ACE",
    "r":"1900F6A9258285152512702EEDA2C14020144604EDFAECBB2D72049C0C64EFFD",
    "r_prime":"0763729176D2429CC95DF0B2EBAA4D9055A776B2BE1C1C3F93E7D4D7654C0DD5",
    "r_prime_prime":"0C6501D0497DCF973B9C02C5E53048E7E58ED10DF29C442E590611F48B26A466",
    "r_prime_prime_prime":"198FD094A738BF43F8EC08315458FE477677C5F47AD018F3932DF7B5334D7B1B",
    "o":"079A03917433B9A220C0CF8BEBEAC07FBD0F8404152D1D8644BB17F0EC7EF6B0",
    "o_prime":"0645BF132FB7FAF27712AFA57E1C5F82E0EAC79DA34A0396135DC630497E14B2",
    "m":"0C21D8151657D7BEFAD1C5018B16238764194BC1B2B2DF9A0C99D9391A3043CE",
    "m_prime":"0B2507628A63DDF673FF6708A400B98145DDB59DA3EF59859685F4A036E2F119",
    "t":"21D152115EF57874DE9B01EF7443058015AEE07A1BD9F97C8F357FD48F365EFA",
    "t_prime":"1AA24784E98B831D900CE1DB2BF51CC7F1678A353C25D2B0D0216CBAFD4027F9",
    "m2":"10DA914DE2DEF54F9DF4BDB8F2A95DD59BC599F333BAF969B4FDB0A620E3BB5A",
    "s":"16054495C221DA96F6E7DB1CAC2CCF775133E4156C2D8A1C0A0484C3D6A95EC5",
    "c":"151BBF20A8E840A28673237B3F3B50D80BE708976623873ABC170EF7CB0D918F"
  },
  "c_list":{
    "e":"6 48C8B...5AB35 4 17F49...BF442 4 25DB0...B898B",
    "d":"6 5BD1F...D881A 4 397FF...109A9 4 1F40E...0C8E7",
    "a":"6 48D14...F4A7B 4 06557...04D02 4 3F21F...9CCB9",
    "g":"6 2969A...21E01 4 24979...FE459 4 3039C...707D7",
    "w":"21 13080...56E45 21 118F8...55276 6 61A9F...C15F2 4 18E47...EBCC4 6 63983...2E80C 4
        2A0D5...C1D3D",
    "s":"21 12957...86945 21 122B9...50686 6 5C7AD...9E093 4 2796C...EF7E8 6 7BD28...F41A0 4
        26E03...13FFF",
    "u":"21 13F6C...8A535 21 1310E...7E586 6 79151...18E78 4 1D31E...13FE9 6 81C6B...5657C 4
        34A8D...33A29"
  }
}
```


Figura 41: Prova de não revogação

Ao receber a apresentação verificável, o *verifier* irá validar as provas de posse dos atributos e de não revogação apresentadas através das chaves públicas presentes na definição da credencial criada pelo *issuer*, cujo identificador está incluído na apresentação verificável. Para efetuar a verificação, o *verifier* deverá obter da *blockchain* todos os esquemas, definições de credenciais, registos de revogação e acumuladores associados aos atributos presentes na apresentação verificável.

3.3 COMUNICAÇÃO COM DIDS

Num ecossistema descentralizado, os agentes precisam de comunicar uns com os outros de maneira a atingirem os seus objetivos. Num sistema de gestão de identidades descentralizado essa comunicação é feita através de identificadores descentralizados (DIDs) e dos seus respetivos documentos. A comunicação com DIDs procura ser segura, privada, interoperável, independente do transporte e extensível [81]:

- Segurança e privacidade implicam que os protocolos sejam descentralizados e tão opacos quanto possível;
- A interoperabilidade procura garantir que a comunicação com DIDs funciona de forma independente a linguagens de programação, *blockchains*, fornecedores, sistemas operativos / plataformas, redes, jurisdições, criptografia e *hardware*;
- Independente do transporte significa que deve ser possível usar comunicação por DIDs através de HTTP, WebSockets, IRC, Bluetooth, AMQP, Signal, email, entre outros;
- Extensibilidade procura possibilitar a inserção de novas funcionalidades de forma simples e natural.

Ao contrário do paradigma de desenvolvimento web, que se baseia em comunicações *duplex* com curtos períodos de espera por uma resposta (através do mesmo canal) após se efetuar uma chamada à API, numa comunicação com DIDs o intervalo entre uma mensagem enviada e uma mensagem recebida em resposta pode ser de horas ou até dias, podendo ser utilizados canais diferentes para a mesma comunicação. Deste modo, uma comunicação por DIDs é uma comunicação assíncrona, *simplex* e baseada em mensagens.

Relativamente à segurança ao nível da mensagem, os servidores tradicionais dependem de certificados e sessões, bem como da segurança do TLS. Por sua vez, a comunicação com DIDs usa criptografia de chave pública descentralizada, o que fornece segurança independente do transporte e remove a necessidade de recorrer a certificados ou sessões.

Esta secção visa explorar as várias estratégias utilizadas pelo agente desenvolvido de modo a que seja capaz de se comunicar com outros agentes. Assim sendo, a secção começa por discutir a primitiva de toda a comunicação em SSI: os identificadores descentralizados, explicando os dois tipos de DIDs implementados no agente desenvolvido e em que situações o agente utiliza cada um deles. Posteriormente são apresentados os protocolos implementados que permitem ao agente comunicar com quaisquer outros agentes que sigam os mesmos protocolos. Por fim é explicado o mecanismo de comunicação orientado às mensagens utilizado pelo agente e as técnicas criptográficas envolvidas na proteção dessas mensagens.

3.3.1 *DIDs de emparelhamento vs DIDs públicos*

No capítulo 2, mais concretamente na subsecção 2.2.2, foram discutidos vários aspetos dos identificadores descentralizados, nomeadamente a sua sintaxe e a sintaxe dos seus respetivos documentos. Na implementação da solução a maioria desses aspetos são respeitados. Contudo, é feita uma separação de DIDs em duas categorias: **DIDs de emparelhamento** (*peer DIDs*) e **DIDs públicos**.

Os DIDs públicos respeitam todas as normas originais de identificadores descentralizados cujos documentos são armazenados na *blockchain* e podem ser consultados por qualquer entidade a qualquer momento desde que essa entidade sabia qual é o identificador descentralizado ao qual o documento em questão corresponde. Apesar desta estratégia ser bem construída e necessária em alguns casos, nomeadamente na construção de reputação por parte de entidades públicas, nem sempre é necessário recorrer à *blockchain* para armazenar estes documentos. Pelo contrário, a maior parte das entidades intervenientes não precisam de associar uma reputação pública aos seus identificadores descentralizados e a grande maioria das relações de um sistema de gestão de identidades descentralizado são de caráter privado pelo que podem (e devem) ser estabelecidas através de trocas diretas de informação entre dois agentes, eventualmente partilhando os respetivos documentos dos DIDs de forma privada. Esta estratégia é conseguida através do uso de DIDs de emparelhamento e tem como objetivo remover a necessidade de recorrer a uma *blockchain* para mediar relacionamentos privados, o que permite manter a *blockchain* mais rápida (por não precisar de armazenar tanta informação) mas também reduzir os custos associados às taxas de transações das *blockchains*.

O conceito de *peer DIDs* foi introduzido por Daniel Hardman numa apresentação que este realizou em Novembro de 2019 [82]. Um *peer DID* segue a mesma especificação que um DID normal. As principais diferenças estão no local onde este identificador e o respetivo documento são armazenados e no contexto em que são utilizados. Enquanto que os restantes DIDs e respetivos documentos são armazenados em locais que atuam como fonte de verdade (i.e. *blockchains*) e são utilizados para que uma entidade possa provar a todas as entidades do sistema que possui certos atributos, associando-se a um DID, os *peer DIDs* e os seus respetivos documentos são usados para estabelecer conexões privadas entre duas entidades e são armazenados na *wallet* do seu dono e, eventualmente, partilhados com uma segunda entidade de modo a possibilitarem o estabelecimento de uma comunicação segura. Ao receber o documento associado ao DID que a primeira entidade está a usar para estabelecer a conexão, a segunda entidade armazena-o na sua *wallet* e consulta-o sempre que necessitar de informações sobre a outra entidade (i.e. chaves públicas). Todo o processo de criação e gestão de DIDs e dos seus respetivos documentos está ao cargo do agente do utilizador.

A figura 42 mostra um exemplo do estabelecimento de uma conexão entre duas entidades através de *peer DIDs*. Neste exemplo, tanto Acme como Bob pretendem estabelecer uma

relação privada para poderem trocar informações. Para tal, geram os DIDs e respetivos documentos necessários para estabelecer a conexão e partilham-os um com o outro. Dado que as entidades possuem os documentos localmente e não existe uma fonte de verdade que possam usar para atualizar esses documentos, sempre que uma das partes precisar de atualizar algo no seu documento deverá contactar a outra entidade e enviar o documento atualizado para que essa entidade possa atualizar o documento que armazenou localmente.

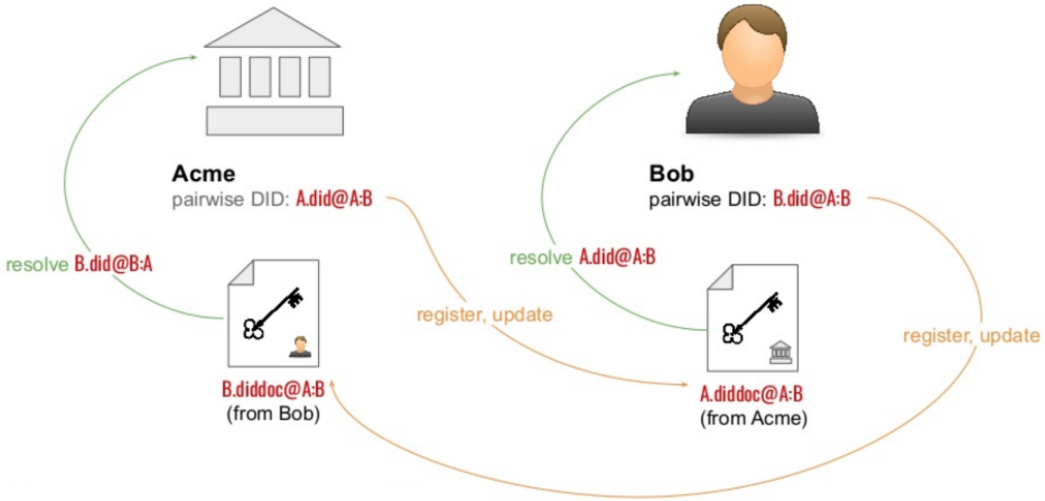


Figura 42: Relacionamento baseado em *peer DIDs* (Retirado de [82])

Nesta situação não há necessidade de recorrer a um registo que atue como fonte de verdade dado que entidades externas ao relacionamento não precisam de conhecer o conteúdo destes dois documentos. Quando tal acontece, os DIDs de emparelhamento são a solução ideal.

Atualmente a comunidade W3C está a desenvolver um documento, *Peer DID Method Specification* [83], que procura definir a especificação de um *peer did* - identificador descentralizado independente de qualquer fonte de verdade - que seja barato, rápido, escalável e seguro. Este tipo de identificadores procura reduzir a quantidade de relações estabelecidas com base em *blockchains*. De notar que este documento não representa o consenso na comunidade W3C, estando ainda em fase de análise. Apesar disso, uma vez que o conceito é muito interessante, foi implementado no agente solução proposta nesta dissertação. A utilização destes dois tipos de DIDs obriga o agente a determinar se os documentos estão situados numa *blockchain* ou localmente, na *wallet*. Esta distinção é feita através do método do DID, ou seja, se o método for do tipo "peer", o respetivo documento está armazenado na *wallet*. Caso contrário, o agente irá proceder à utilização do *resolver universal* para determinar a localização do documento, tal como foi explicado na secção 2.2.2.

Tendo sido explicado o conceito de *peer DIDs*, é importante referir as razões pelas quais o uso de *peer DIDs* é desejável. Segundo Daniel Hardman [82], 99% dos DIDs não precisam das caraterísticas que uma *blockchain* proporciona, isto é, o seu uso é exclusivo ao estabelecimento

de conexões privadas entre agentes. Ao retirar todos estes DIDs da *blockchain* é possível evitar as taxas de transações, custos de eletricidade, entre outros, mas também aumentar a segurança e privacidade dos dados uma vez que não existe uma *blockchain* ou nó para usar como vetor de ataque e apenas os intervenientes na comunicação têm conhecimento dos DIDs utilizados nessa comunicação e respetivas informações. Adicionalmente, a performance da *blockchain* aumenta consideravelmente com a redução da quantidade de transações que são efetuadas. Além disso, a necessidade de individualizar credenciais (e informações pessoais) através do uso de múltiplos DIDs de modo a mitigar os vários problemas de correlação de dados faz com que o uso de DIDs de emparelhamento seja ainda mais desejado.

A subsecção 3.3.2 explica o protocolo utilizado para que dois agentes possam partilhar o seu DID e o respetivo documento - **protocolo de conexão**.

3.3.2 *Protocolos*

Para permitir o estabelecimento de uma comunicação entre diferentes utilizadores de um ecossistema descentralizado é necessário estabelecer protocolos de comunicação universais de modo a que todos estes agentes sejam capazes de se entenderem mutuamente. A comunidade do Hyperledger Aries [84] tem vindo a definir vários protocolos visando a sua utilização como padrão a nível mundial [85], entre os quais:

- Conexão entre agentes;
- Pedido e emissão de credenciais;
- Prova de atributos através de credenciais;
- Delegação de Autoridade (explicado na subsecção 2.3.3);
- Promulgação e execução de contratos;
- Comunicação de erros.

Estes protocolos são descentralizados, o que implica que não existe uma autoridade superior que garanta o fluxo de dados ou imponha certos comportamentos. Deste modo, todos os intervenientes interagem através de consenso mútuo e conhecimento das regras e objetivos de um dado protocolo [85]. Protocolos mais simples podem ser incluídos em protocolos mais complexos. Neste caso, o protocolo interno é designado por **sub-protocolo** e o protocolo externo é designado por **super-protocolo**. Protocolos descentralizados introduzem ainda a noção de "papéis" (com responsabilidades e privilégios), nomeadamente *holder*, *prover*, *issuer* e *verifier*. Num contexto geral, *prover* e *holder* representam a mesma entidade. Contudo, no contexto de agentes de utilizador, *holder* é o papel desempenhado pelo ator que pretende adquirir uma credencial enquanto que *prover* é o papel desempenhado pelo ator que pretende provar que possui certos atributos através de *verifiable presentations*.

Estes protocolos correspondem apenas a fluxos de transição de dados entre os intervenientes, ficando ao cargo de cada agente gerir o modo como trata as várias informações trocadas,

ou seja, o objetivo destes protocolos é conseguir que todos os agentes do ecossistema sejam capazes de reconhecer as várias informações contidas nas mensagens que recebem de outros agentes e de responder numa linguagem que os esses agentes também consigam entender. Ao longo desta subsecção serão abordados os três principais protocolos implementados na solução desenvolvida:

- Protocolo de conexão;
- Protocolo de emissão da credencial;
- Protocolo de apresentação da prova.

Estes três protocolos representam o núcleo de todas as interações dentro de um sistema de gestão de identidades descentralizado e são indispensáveis a qualquer agente de utilizador. Contudo, o agente desenvolvido implementa vários protocolos adicionais tais como o *Aries RFC 0008*[86], o *Aries RFC 0011*[87], o *Aries RFC 0017*[88], o *Aries RFC 0020*[89], o *Aries RFC 0015*[90] e o *Aries RFC 0035*[91]. Alguns destes protocolos ainda não foram oficialmente aceites pela comunidade, contudo apresentam propriedades interessantes, razão pela qual foram implementados na solução desenvolvida. Dos protocolos adicionais indicados, é importante dar especial destaque aos protocolos *Aries RFC 0015* (ACKs) e *Aries RFC 0035* (Report Problem Protocol 1.0) uma vez que são sub-protocolos incluídos nos três protocolos principais (indicados acima) e serão referenciados de seguida, na explicação detalhada destes três protocolos.

Protocolo de Conexão

Antes de poderem trocar informações sobre credenciais e atributos, ambas as entidades precisam de estabelecer uma comunicação segura entre elas. O protocolo de conexão define todos os passos que devem ser tomados durante o estabelecimento de uma comunicação segura [92] de modo a que os agentes consigam interagir entre si sem qualquer dificuldade. Neste protocolo existem dois papéis: *inviter* e *invitee*. O *inviter* é a entidade que inicia o protocolo através da criação e partilha de um convite de conexão. O *invitee* é a entidade que vê e aceita o convite através do envio de um pedido de conexão à entidade que o publicou. Adicionalmente, este protocolo apresenta vários estados que representam o ponto no qual o estabelecimento da conexão se encontra:

- **null**: a conexão ainda não existe;
- **invited**: o convite foi partilhado com o *invitee*;
- **requested**: o *invitee* enviou um pedido de conexão ao *inviter*;
- **responded**: o *inviter* respondeu ao pedido de conexão do *invitee*;
- **complete**: a conexão está ativa e pronta a utilizar.

A figura 43 mostra todo o processo de estabelecimento de uma conexão através de um diagrama de coreografia, desde a criação do convite até à ativação de uma conexão.

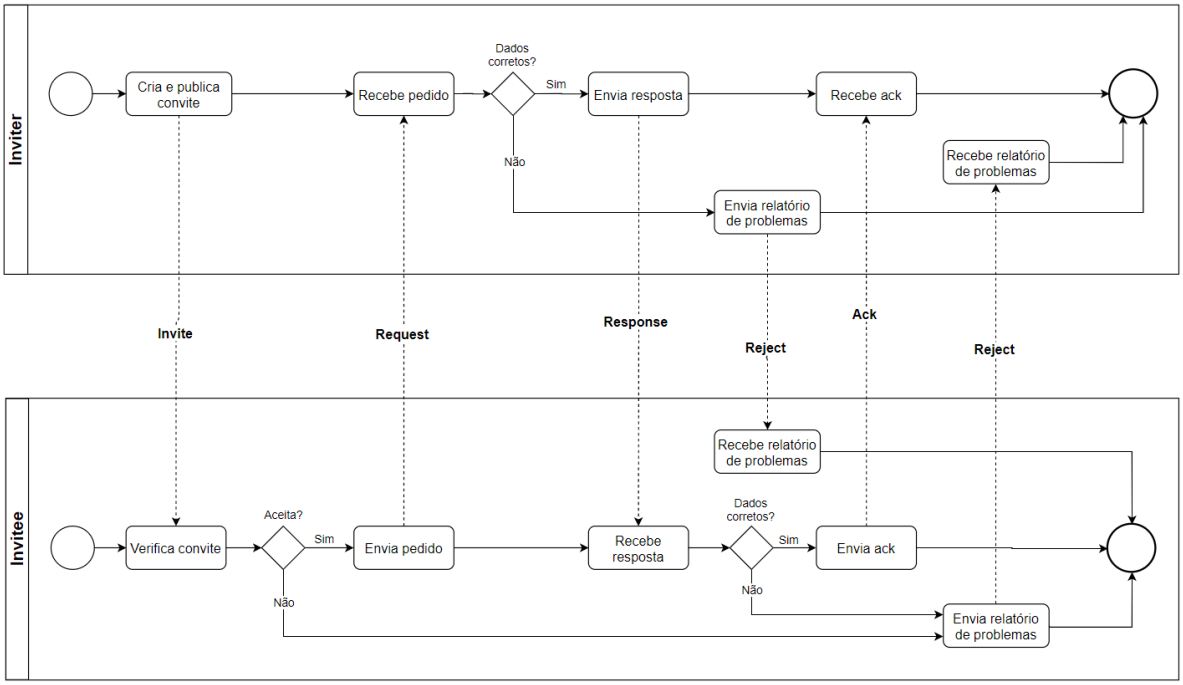


Figura 43: Protocolo de conexão

O protocolo inicia-se com uma das partes, o *inviter*, a partilhar um convite (através de um objeto JSON, QRCode, URL, etc) que contém informações sobre como entrar em contacto com ela. Todos os convites possuem um DID associado de acordo com o tipo de convite, podendo este ser um DID público ou um DID de emparelhamento. Por conseguinte, um convite pode ser:

- **Privado** - contém um DID de emparelhamento, a chave pública que outras partes devem usar para cifrar os dados destinados a esta entidade, o *endpoint* de serviço no qual o agente dessa entidade estará a escutar e ainda um conjunto de chaves públicas de mediadores pelos quais a mensagem deve ser encaminhada. Caso não sejam especificadas chaves públicas de mediadores, a mensagem deve ser enviada diretamente para o agente da entidade que criou o convite. A subsecção 3.3.5 explica com mais detalhe o conceito de mediadores;
- **Público** - contém um DID público cujo documento pode ser obtido consultando uma *blockchain*. Este documento contém todas as informações adicionais apresentadas num convite privado, nomeadamente a chave pública do *invitee*, o seu *endpoint* de serviço e as chaves públicas dos mediadores. Qualquer identificador descentralizado público atua por si só como um convite, isto é, qualquer pessoa que tenha conhecimento de um DID pode adquirir o respetivo documento através de um *resolver* universal e, consequentemente, obter as informações necessárias para contactar o dono desse DID.

Se uma entidade (*invitee*) pretender estabelecer uma conexão com o *inviter* deve primeiro criar um DID de emparelhamento e um documento associado, que contém as típicas informações de um documento de um DID, e, de seguida, enviar uma mensagem de pedido, *connection request*, ao respetivo *inviter*, cifrando esta mensagem com a chave pública presente no convite (se o convite for privado) ou no documento do DID (se o convite for público) e encaminhando-a pelos vários mediadores especificados. Neste pedido de conexão, o *invitee* deve enviar o documento referente ao DID de emparelhamento que acabou de criar para esta conexão de modo a que o *inviter* tenha acesso às informações desse DID, em particular do serviço de comunicação do agente do mesmo.

Ao receber o pedido de conexão, o *inviter* processa o documento recebido e obtém as informações sobre como contactar o *invitee* (chaves públicas, *endpoint* de serviço, chaves públicas de mediadores, entre outros), armazenando o documento na sua *wallet*. Apesar de o *inviter* poder usar um DID público para criar um convite, deverá também criar um DID de emparelhamento e um documento associado para esta nova conexão de modo a isolar a interação com o *invitee*. Tal é necessário devido à possibilidade de um observador conseguir correlacionar as mensagens trocadas entre o *inviter* e o *invitee* utilizando a chave pública presente no convite. Ao criar um DID de emparelhamento para cada conexão, o *inviter* cria também um par de chaves para cada uma dessas conexões, o que faz com que cada comunicação possua uma chave pública diferente e, consequentemente, não seja possível correlacionar mensagens através da chave pública. Esta estratégia permite que qualquer *invitee* seja capaz de confirmar a identidade da entidade pública associada ao convite e, ao mesmo tempo, melhorar a segurança de cada conexão individual. Após criar o DID e o respetivo documento, o *inviter* envia-os ao *invitee* através de uma mensagem de resposta, *connection response*, cifrando a mensagem com a chave pública presente no documento que recebeu e encaminhando-a pelos vários mediadores especificados no mesmo. O campo que contém a informação sobre o DID e o documento associado deve ser assinado pelo *inviter* com a chave privada associada ao convite de forma a autenticar estas informações.

Ao receber a resposta de conexão, o *invitee* verifica a assinatura efetuada sobre o documento do DID, recorrendo à chave pública contida no convite inicial, de forma a comprovar que foi o *inviter* quem lhe enviou esse documento. Se a verificação for bem sucedida, o *invitee* armazena o documento que recebeu na sua *wallet* e envia uma mensagem de confirmação, designada por *connection acknowledgement*, utilizando para tal o sub-protocolo *ACKs* [90].

Após a conexão estar estabelecida, as duas entidades podem comunicar entre si de forma segura, recorrendo às chaves públicas contidas nos respetivos documentos recebidos para cifrar todas as mensagens.

Se durante este protocolo ocorrer algum erro ou desvio face ao fluxo apresentado, tal deve ser comunicado à outra entidade através do sub-protocolo *Report Problem Protocol 1.0* [91].

Protocolo de emissão da credencial

A emissão de credenciais envolve dois intervenientes: o *issuer* e o *holder*. Para que o *issuer* emita uma credencial é necessário que ambas as partes estejam de acordo quanto ao tipo de informações que essa credencial deverá conter. O protocolo de emissão da credencial [93] especifica as mensagens trocadas desde o processo inicial de negociação até à mensagem de confirmação enviada pelo *holder* a confirmar que a credencial foi recebida. À semelhança do protocolo anterior, este protocolo também apresenta vários estados. Contudo, existe uma separação de estados tendo em conta o papel desempenhado pela entidade como é possível observar na tabela apresentada de seguida. A execução do protocolo termina assim que ambas as entidades atingirem o estado "completo".

Issuer	*Holder*
Proposta recebida	Proposta enviada
Oferta enviada	Oferta recebida
Pedido recebido	Pedido enviado
Credencial emitida	Credencial recebida
Completo	Completo

A figura 44 representa o processo de emissão de credenciais através de um diagrama de coreografia.

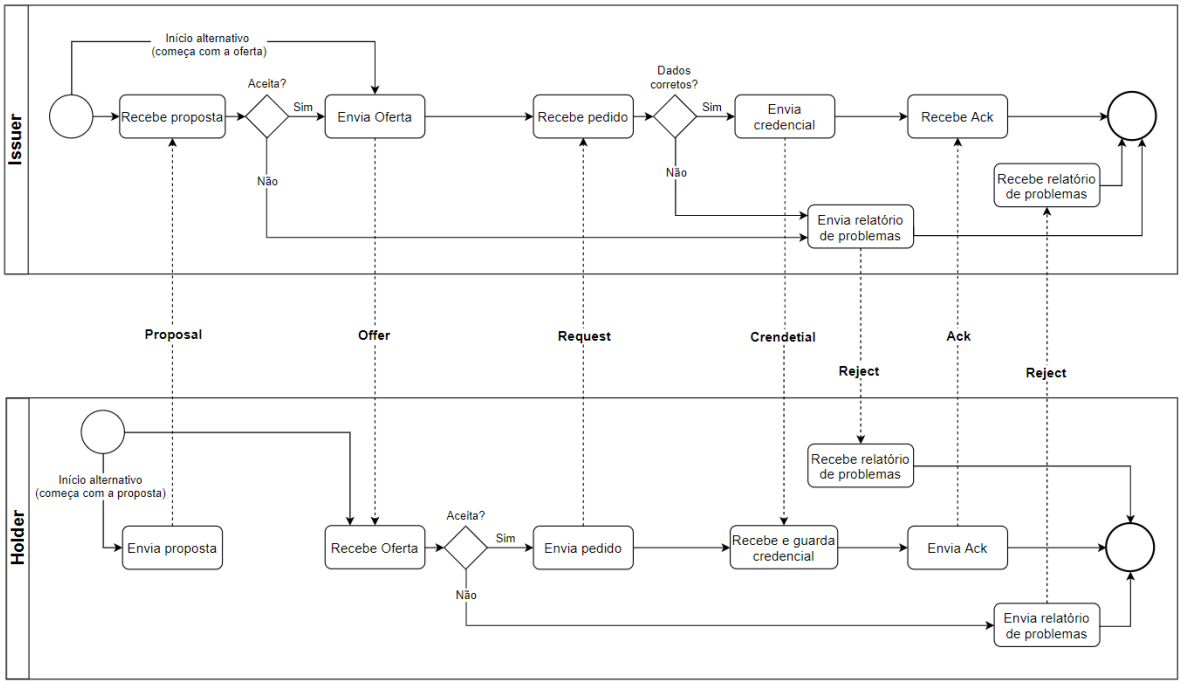


Figura 44: Protocolo de emissão da credencial

Na solução implementada, este protocolo apresenta dois inícios alternativos. A primeira alternativa implica que seja o *holder* a iniciar o protocolo através do envio de uma proposta ao *issuer* na qual indica que esquema de dados pretende usar e qual o valor que cada um dos atributos desse esquema deve possuir. Já a segunda alternativa exclui a fase inicial de envio e receção de uma proposta, sendo o *issuer* a iniciar o protocolo através da oferta de uma credencial. Esta oferta deve conter a *credential definition* que o *issuer* pretende usar para emitir a credencial, uma pré-visualização dos atributos que irão ser incluídos na credencial e os seus respetivos valores, e ainda uma prova de correção de chaves de modo a comprometer o *issuer* criptograficamente à emissão de um conjunto específico de atributos (de acordo com a *credential definition*) na credencial final. Esta prova visa prevenir o *issuer* de inserir dados ilegítimos ou falsos que vão contra o que foi acordado durante o protocolo. Nesta última alternativa, o *issuer* deve possuir previamente todos os dados do *holder* necessários à emissão da credencial.

Ao receber a oferta, o *holder* verifica se todos os atributos apresentados estão corretos. Se encontrar algum problema deverá cancelar o processo e notificar o *issuer*, indicando a razão do cancelamento. Para tal é utilizado o sub-protocolo *Report Problem Protocol 1.0* [91]. Caso tudo esteja correto, o *holder* continua o processo através do envio de um pedido que contém a *credential definition* que deverá ser usada pelo *issuer* para a emissão da credencial bem como um segredo ofuscado, **master secret**, que o *issuer* deverá incorporar na credencial para que o *holder* possa provar que os atributos da credencial lhe pertencem. Esta técnica criptográfica é designada por assinatura *Camenisch-Lysyanskaya*, um esquema de assinatura desenvolvido por Jan Camenisch e Anna Lysyanskaya e que apresenta várias carateristicas desejáveis na emissão de credenciais anónimas [94]:

- Permite que o controlo de uma credencial verificável seja vinculado a um **master secret** conhecido apenas pelo *holder*, ou seja, apenas o *holder* é capaz de provar posse dos atributos presentes numa credencial dado que é o único que possui conhecimento do **master secret**;
- Fornece ao *issuer* a capacidade de assinar subconjuntos de atributos de acordo com a granularidade que pretende atribuir à credencial, isto é, em vez de assinar todos os atributos com uma única assinatura, o *issuer* pode optar por assinar atributos individualmente, permitindo que o *holder* possa apresentá-los aos *verifiers* de forma individual, ou em subconjuntos, obrigando o *holder* a apresentar todos os atributos desse conjunto em simultâneo. Assinar conjuntos de atributos pode ser necessário no caso da existência de atributos inseparáveis ou atributos negativos, tal como foi mencionado na subsecção 2.3.3, mais concretamente em *Divulgação seletiva e atributos negativos*;

- Permite que o *holder* possa combinar subconjuntos de atributos de diferentes credenciais numa única prova, apresentação verificável, de maneira a que todos os atributos sejam verificáveis;
- Apresenta a capacidade de emitir credenciais que suportam o uso de provas de conhecimento zero.

Após receber o pedido para a emissão de uma credencial, o *issuer* verifica se esse pedido está em conformidade com a oferta que enviou anteriormente e, caso não sejam detetados problemas, emite uma credencial com os atributos acordados e envia-a ao *holder*. Se a *credential definition* usada pelo *issuer* suportar revogação, a credencial emitida deve ainda incluir o identificador do registo de revogação usado para todas as credenciais emitidas através dessa *credential definition*.

Por fim, ao receber a credencial, o *holder* deverá notificar o *issuer* que recebeu a credencial através do sub-protocolo de ACKs [90], terminando assim o protocolo de emissão da credencial.

Uma vez mais, caso ocorra algum problema, deverá ser utilizado o sub-protocolo *Report Problem Protocol 1.0* [91] para comunicar esse problema à outra entidade do protocolo.

Protocolo de apresentação da prova

Em muitas situações, para um *holder* ter acesso a um serviço disponibilizado por uma entidade precisa de provar a essa entidade que possui certos atributos, seja provar que possui mais de 18 anos para comprar bebidas alcoólicas, provar que tem licenciatura numa determinada área ou até mesmo provar que possui um emprego. Esta prova envolve duas entidades: o *prover* (papel desempenhado pelo *holder* no ato de apresentação de prova) e o *verifier*. O protocolo de apresentação da prova [95] procura padronizar as interações que estas duas entidades têm uma com a outra de forma a que o *holder* consiga provar ao *verifier* que possui certos atributos. Assim como os protocolos anteriores, também este protocolo apresenta um conjunto de estados possíveis para cada entidade tendo em conta o papel desempenhado e a sua execução termina assim que ambas as entidades atingirem o estado "completo".

Prover	*Verifier*
Proposta enviada	Proposta recebida
Pedido recebido	Pedido enviado
Prova enviada	Prova recebida
Completo	Completo

A figura 45 representa o processo de geração e validação de uma prova através de um diagrama de coreografia.

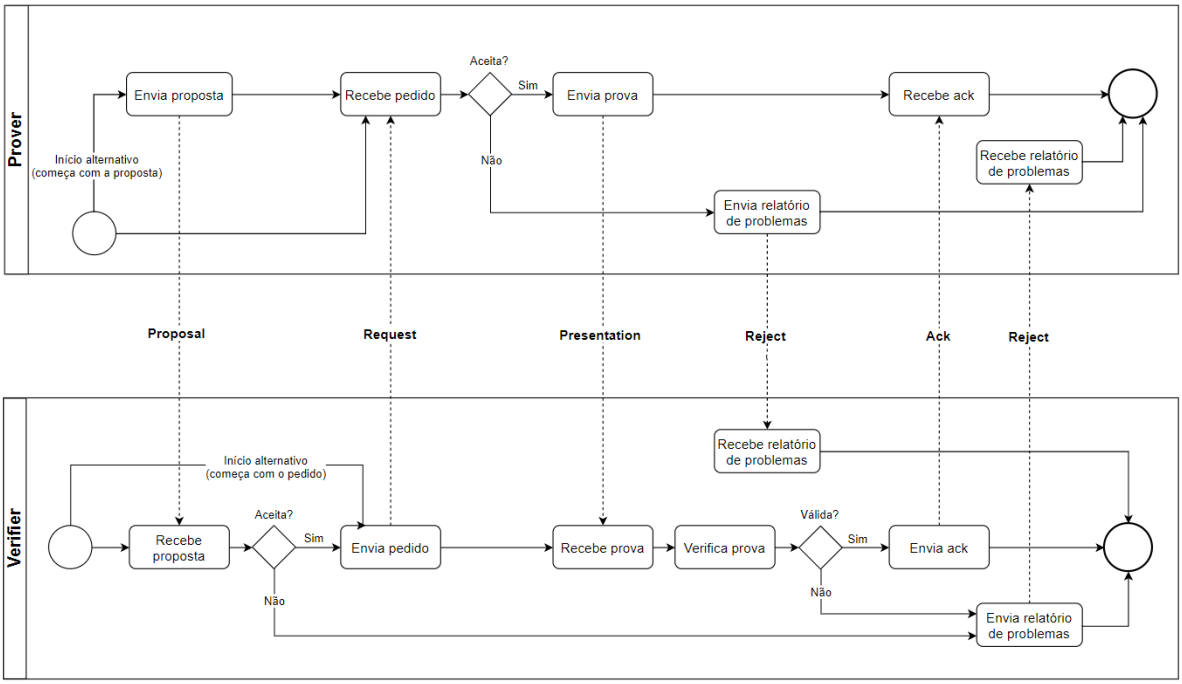


Figura 45: Protocolo de apresentação da prova

O protocolo de apresentação da prova pode ser iniciado com a apresentação de uma proposta, por parte do *prover*, ou com um pedido de prova, por parte do *verifier*. Se o *prover* iniciar o protocolo, deverá enviar uma proposta na qual indica os atributos que quer provar divididos em três categorias: atributos simples, predicados (i.e. provar maioridade sem revelar a idade específica) e auto-atribuídos. Na solução implementada, os atributos auto-atribuídos estão implementados mas não são utilizados uma vez que não trazem valor aos casos de uso explorados.

Ao receber a proposta, o *verifier* decide se pretende ou não requisitar essa prova ao *prover*. Em alternativa, o *verifier* pode tomar a iniciativa e iniciar o protocolo, enviando um pedido de prova ao *prover*. Este pedido de prova deverá conter os nomes dos atributos pretendidos, divididos nas três categorias indicadas, um conjunto de restrições para cada um destes atributos (i.e. esquemas, *issuers* e *crendential definitions* aceites) e, opcionalmente, um intervalo de tempo para cada atributo no qual a prova desse atributo deve ser válida, ou seja, não esteja revogada. Caso este intervalo não seja especificado, será utilizado o momento em que o pedido de prova for criado. Em geral a segunda alternativa é a mais utilizada.

Após receber o pedido de prova, o *prover* verifica atributos simples e predicados que estão a ser requeridos pelo *verifier* e escolhe que credenciais deseja utilizar para cada um desses atributos, assumindo que essas credenciais cumpram todas as restrições impostas pelos atributos. Caso não concorde em fornecer certos atributos ao *verifier* ou não possua credenciais válidas com os atributos desejados deverá rejeitar o pedido e informar o *verifier*.

Se o *prover* possuir todos os atributos necessários e concordar com a partilha dos mesmos deverá gerar uma apresentação verificável com esses atributos e enviá-la ao *verifier*.

Recebendo a apresentação, o *verifier* procede à verificação da mesma e, se a prova for válida, informa o *holder* de que recebeu e validou a apresentação sem detetar problemas, através do sub-protocolo *ACKs* [90], terminado assim o protocolo.

Assim como nos dois protocolos anteriores, também este protocolo recorre ao sub-protocolo *Report Problem Protocol 1.0* [91] para que uma entidade possa comunicar à outra entidade qualquer problema que ocorra durante a execução do protocolo.

3.3.3 *Tipo de mensagem*

Os agentes de *self-sovereign identity* determinam qual o protocolo a usar através do campo "type" presente na mensagem recebida. Este campo deve seguir o formato apresentado na figura 46.

```
type = uri + delim + nome-do-protocolo + "/" + ãverso-do-protocolo + "/" + tipo-de-mensagem
delim = "?" | "/" | "&" | ":" | ";" | "="
```

Figura 46: Exemplo do valor da propriedade *authentication* [27]

O URI resultante especifica a localização do documento, o protocolo a ser usado e a respetiva versão, bem como a interação pretendida. Um exemplo de um valor para este campo é "https://didcomm.org/connections/1.0/request". Neste caso o agente que enviou a mensagem pretende estabelecer uma comunicação com outro agente utilizando a versão 1.0 do protocolo de conexão definido em *https://didcomm.org/*.

É importante referir que o agente recetor pode não suportar um protocolo ou uma versão específica desse protocolo. Nesse caso, o agente recetor deve enviar uma mensagem ao agente emissor, através do protocolo de comunicação de problemas, a indicar que não suporta o protocolo ou a versão pretendida desse protocolo e, possivelmente, que versões suporta. O agente emissor pode enviar uma nova mensagem com uma das versões suportadas ou terminar o protocolo.

3.3.4 *Envelopes criptográficos*

Todos os protocolos de um ecossistema descentralizado são baseados em trocas de mensagens de texto simples, que são transmitidas dentro de envelopes criptográficos. Um envelope criptográfico é nada mais do que um "embrulho" em torno de uma mensagem, cifrado com a chave pública do recetor de maneira a permitir a transmissão segura dessa mensagem entre os agentes. Contudo, não é possível iniciar uma comunicação utilizando imediatamente envelopes criptográficos. O **protocolo de conexão** discutido na subsecção 3.3.2 explica como

os agentes partilham os DIDs e respetivos documentos (com as chaves públicas que devem ser utilizadas na comunicação) de maneira a estabelecerem uma conexão segura antes de começarem a trocar mensagens. Estes envelopes usam um formato padrão definido no RFC *JSON Web Encryption - RFC 7516* [96]. Este RFC fornece uma solução genérica para cifrar, decifrar e encaminhar mensagens de forma independente ao transporte.

Uma comunicação entre dois agentes pode envolver nós intermédios que atuam como encaminhadores de mensagens. Caso estes nós intermédios atuem como *relays*, a sua função é apenas reencaminhar as mensagens recebidas sem necessidade de decifrar nada. Por outro lado, se os nós intermédios atuarem como mediadores, o emissor deverá utilizar um envelope criptográfico para cada mediador. Deste modo, se a Alice pretender efetuar uma comunicação com o Bob através de uma rota com dois agentes mediadores deverá criar um envelope criptográfico para o Bob e dois envelopes criptográficos adicionais, um para cada salto na rota, isto é, um para cada mediador tal como mostra a figura 47.

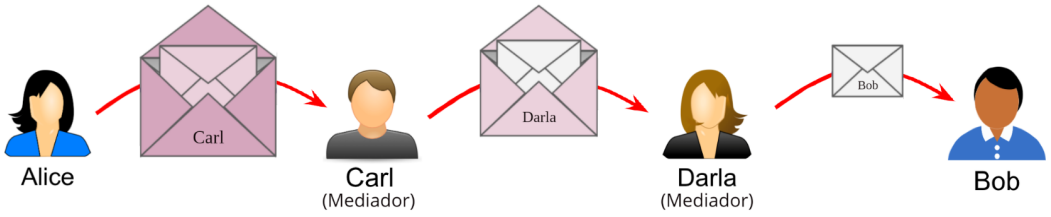


Figura 47: Rota com dois mediadores (Adaptada de Aries RFC 0046 [97])

Para enviar uma mensagem dentro dos vários envelopes, o agente da Alice deve primeiro criar o envelope criptográfico destinado ao Bob com a mensagem original que pretende transmitir e, recursivamente, incluir o envelope criptográfico de cada um dos agentes mediadores no envelope criptográfico do agente antecessor na cadeia de transmissão da mensagem. Deste modo, cada agente consegue decifrar o envelope que recebeu e avaliar se é efetivamente o destinatário da mensagem original ou se deverá encaminhar o resultado obtido para o próximo agente na cadeia de transmissão. Caso o agente não seja o destinatário da mensagem não irá conseguir retirar qualquer informação do conteúdo presente na mensagem original uma vez que não consegue decifrar o envelope destinado ao próximo agente na cadeia de transmissão. A subsecção 3.3.5 explica com mais detalhe os dois tipos de nós intermédios existentes na arquitetura: **mediadores** e ***relays***.

A técnica criptográfica usada em envelopes criptográficos designa-se por ChaCha20-Poly1305 e baseia-se na junção da cifra *ChaCha20* [98] com com autenticador de uso único *Poly1305* [99] de modo a criar um algoritmo de cifra autenticada com dados associados (AEAD). Tanto a *ChaCha20* como o *Poly1305* foram desenhados de forma a fornecer elevada performance e a reduzir a fuga de informação através de ataques *side-channel*. Adicionalmente, estas implementações necessitam de recursos reduzidos e apenas efetuam operações leves, o que permite que sejam usadas numa vasta gama de arquiteturas. [100]

ChaCha20

O *Chacha* é uma cifra utilizada para *stream* de dados e foi desenvolvida por D. J. Bernstein em 2008. A variante da cifra *ChaCha* utilizada em envelopes criptográficos é designada por *ChaCha20*, uma cifra baseada em 20 rondas, com um *nonce* de 96 bits e uma chave de 256 bits, o que lhe assegura uma segurança de 256 bits. Esta cifra é uma redefinição da famosa cifra de *stream Salsa20*, e foi usada como o núcleo da função de *hash BLAKE*, uma das finalistas da competição do NIST de funções de *hash* para integrar o algoritmo SHA-3.

Poly1305

Poly1305 é um autenticador de uso único, também desenvolvido por D. J. Bernstein. Este algoritmo recebe uma chave de 256 bits (de uso único) e uma mensagem, e gera uma *tag* de 16 bytes que autentica a mensagem de modo a que um atacante tenha uma chance negligível de produzir uma *tag* válida para uma mensagem não autenticada [101].

Considerações de segurança

Sendo a *ChaCha20-Poly1305* um algoritmo projetado para ser usado sobretudo em comunicação, é importante referir os vários aspetos de segurança que foram considerados durante a construção do algoritmo. De seguida são apresentados alguns desses aspetos [100]:

- O *poly1305* foi desenhado com o objetivo de garantir que mensagens adulteradas são rejeitadas com uma probabilidade de $1 - \frac{n}{2^{102}}$ para mensagens com 16n bytes mesmo após o envio de 2^{64} mensagens, o que lhe fornece uma forte segurança contra ataques de mensagem escolhida;
- O *nonce* utilizado no algoritmo *ChaCha20* deve ser único a cada mensagem. Se o *nonce* for repetido, tanto a chave do algoritmo *Poly1305* como a *keystream* são idênticos nessas mensagens, o que revela o *XOR* das mensagens originais dado que é igual ao *XOR* dos criptogramas;
- A chave do algoritmo *Poly1305* deve ser imprevisível;
- O *ChaCha20* recorre apenas a adições, *XORs* e rotações fixas, enquanto que o *Poly1305* apresenta adições, multiplicações e operações modulares. Todas estas operações podem ser feitas em tempo constante, ou seja, ambos os algoritmos podem ser implementados em tempo constante;
- A validação da autenticação de uma mensagem inclui uma comparação bit a bit da *tag* calculada com a *tag* recebida. É recomendado recorrer a funções que garantam a execução desta operação em tempo constante em alternativa a funções otimizadas de modo a impedir que um atacante seja capaz de inferir o comprimento do prefixo idêntico presente em ambas as *tags* com base no tempo de execução da função.

O relatório *Security Analysis of ChaCha20-Poly1305 AEAD* [102] apresenta uma avaliação de segurança do algoritmo *ChaCha20-Poly1305*. Segundo este relatório não é possível efetuar (de forma eficiente) análise diferencial, criptoanálise linear, ataque distintivo, análise de adivinhar e determinar ou ataque algébrico no *ChaCha*. O relatório conclui com a afirmação de que não foi possível encontrar qualquer ponto fraco no *ChaCha20-Poly1305 AEAD* assumindo que a implementação do *ChaCha* foi efetuada com cuidado e seguindo todas as recomendações.

Authcrypt vs Anoncrypt

No ecossistema do Hyperledger, os envelopes criptográficos apresentam dois modos de uso: **anoncrypt** e **authcrypt**. Anoncrypt é usado quando se pretende enviar uma mensagem sem associar criptograficamente o emissor da mensagem ao conteúdo da mesma. Por sua vez, authcrypt é usado quando o emissor pretende associar a sua entidade à autoria da mensagem, assinando a mensagem com a sua chave privada..

Relativamente ao modo authcrypt, é importante indicar que apesar de assinaturas estarem fortemente ligadas à autenticação, as consequências do seu uso nem sempre são desejáveis, nomeadamente a redução da privacidade. Vejamos o caso em que a Alice pretende enviar uma mensagem ao Bob. Para tal, pode enviar uma mensagem assinada digitalmente (de forma a que seja não repudiável) ou através da técnica de cifra autenticada (de forma a que seja repudiável). A técnica que Alice deve usar depende do contexto e do objetivo da mensagem. Se assinar a mensagem digitalmente, Bob irá conseguir provar a qualquer pessoa que foi, de facto, a Alice que escreveu aquela mensagem uma vez que a mensagem foi assinada com a chave privada da Alice. Contudo, se ela optar por usar uma cifra autenticada, o Bob consegue saber que foi Alice quem escreveu a mensagem, dado que apenas eles os dois conhecem a chave simétrica utilizada para cifrar e decifrar as mensagens trocadas entre eles, mas não o conseguirá provar a mais ninguém. Deste modo, Alice pode contar algo ao Bob sem lhe dar controlo dessa informação, isto é, se o Bob tentar divulgar a mensagem que a Alice lhe enviou, esta poderá argumentar que foi o Bob quem escreveu a mensagem dado que ele também possui conhecimento da chave simétrica.

Num paradigma SSI, o comportamento padrão do sistema deve assegurar que o dono de um segredo é sempre capaz de decidir se esse segredo pode ou não ser partilhado. Isto implica que se a Alice possuir um segredo e o partilhar com o Bob, apenas a Alice pode partilhar esse segredo de forma não repudiável com outras pessoas ou entidades, ou seja, o Bob não pode divulgar uma mensagem que a Alice lhe enviou de modo a danificar a reputação dela. Resumindo, a comunicação por DIDs suporta tanto assinaturas digitais como cifras autenticadas, contudo é aconselhado recorrer a mensagens repudiáveis por padrão e permitir o uso de mensagens não repudiáveis apenas após uma escolha deliberada e consciente.

3.3.5 *Mediadores e Relays*

Tendo como objetivo aumentar a privacidade e a disponibilidade dos agentes de *self-sovereign identity* foram introduzidas duas primitivas de comunicação: mediador e *relay*.

Um mediador participa ativamente no reencaminhamento de mensagens entre agentes e possui as suas próprias chaves. Por outras palavras, um mediador é um agente conhecido pelo emissor que deverá ser incluído na rota de encaminhamento de mensagens para um recetor, interagindo ativamente com a mensagem que recebe, decifrando-a e reencaminhando o resultado obtido para o próximo agente da rota. Dado que o agente destinatário da mensagem pode ser um *edge agent*, há a possibilidade de que quando a mensagem seja enviada ele não se encontre conectado à rede. Se o último agente mediador da rota não conseguir comunicar com o agente destinatário, deverá armazenar a mensagem e tentar efetuar a entrega mais tarde. Como foi referido na subsecção 3.3.4, se a Alice quiser adicionar um agente mediador à rota deverá criar um envelope para esse agente e incluir no envelope o conteúdo que esse agente deverá encaminhar para o próximo mediador da rota. O envelope criptográfico exterior será cifrado de forma a que apenas este mediador o possa decifrar. Ao decifrar o envelope, o agente mediador irá deparar-se com uma mensagem do tipo "Forward" que contém a informação referente ao próximo nó da cadeia de transmissão e o respetivo envelope criptográfico que lhe deverá enviar. De notar que o agente mediador não consegue aceder ao conteúdo presente no envelope criptográfico que deverá enviar ao próximo agente da rota.

Por sua vez, um *relay* é uma entidade que também reencaminha mensagens, mas que não decifra nada. Estes *relays* podem ser usados de modo a alterar o transporte de uma mensagem.

Estas duas primitivas de comunicação permitem suportar vários transportes diferentes e fornecem anonimato parcial das rotas tanto para o emissor e recetor como para terceiros, o que garante anonimato total do emissor e do recetor. Tal como foi referido na subsecção 3.2, numa arquiterura SSI os agentes envolvidos podem ser agentes de extremidade ou agentes de *cloud*. Em geral mediadores e *relays* são agentes de *cloud* devido à necessidade de estarem constantemente conectados à rede.

De maneira a entender de forma mais clara a diferença entre mediador e *relay* serão apresentados dois exemplos do mundo real em que a Alice pretende enviar uma mensagem ao Bob [97]:

- Num primeiro exemplo, o Bob interage com o sistema SSI através de um agente de ponto (com o seu telemóvel). Por um lado, Bob não quer disponibilizar um endereço de IP público. Por outro lado, o seu agente só será capaz de receber mensagens no período de tempo em que ele estiver *online*. Deste modo, Bob decide recorrer a um agente mediador que:

  - seja capaz de lhe fornecer *herd hosting*, ou seja, um agente que atua como *host* de agentes de múltiplas entidades distintas e, consequentemente, aumenta a privacidade do destinatário;
  - seja responsável por receber e armazenar todas as mensagens destinadas ao Bob enquanto o seu agente está *offline* e reencaminhá-las assim que o agente ficar *online*;
  - seja capaz de filtrar todas as mensagens que não interessam ao Bob.

  Neste exemplo, se a Alice pretender enviar uma mensagem ao Bob deverá criar um envelope para o Bob e incluí-lo dentro de um envelope adicional, destinado ao agente mediador. Ao abrir o envelope, o agente mediador irá obter os metadados que a Alice lhe enviou, um segundo envelope e uma indicação de que esse envelope deverá ser entregue ao Bob.

  Agentes mediadores podem ainda ser usados como pontos de entrada de um domínio. Digamos que o Bob possui um domínio com vários agentes de ponto que não possuem um endereço de IP permanente. Neste exemplo, o Bob poderá utilizar um agente de *cloud*, controlado por si, que seja responsável por receber todas as mensagens destinadas aos vários agentes de ponto do seu domínio. Este agente de *cloud* irá atuar como agente mediador e, consequentemente, a Alice deverá criar um envelope adicional para este agente mediador seguindo a estratégia descrita no exemplo anterior.

  É importante referir que o Bob deve indicar à Alice que todas as mensagens que lhe forem destinadas deverão ser enviadas através do agente mediador indicado, sendo este agente escolhido pelo Bob.

- Num segundo exemplo, o Bob apenas deseja ter algum mecanismo responsável por armazenar as mensagens que lhe forem enviadas enquanto o seu agente estiver *offline*. Deste modo, se a Alice tentar comunicar com o agente do Bob e este estiver *offline*, poderá enviar a mensagem através de *relays*, que irão armazenar a mensagem até a conseguirem estabelecer uma conexão com o agente do Bob. Neste exemplo a Alice não precisa de criar envelopes adicionais para os *realys* uma vez que estes agentes são usados apenas para reencaminhar mensagens, não tendo a necessidade de processar qualquer informação.

É importante referir que o fluxo de mensagens é de sentido único. Se o Bob quiser enviar uma mensagem para a Alice, terá de criar uma nova rota. A inclusão de um agente mediador ou *relay* no fluxo de mensagens da Alice para o Bob não implica a sua inclusão no fluxo contrário. A notação utilizada para indicar uma mensagem enviada da Alice, A, para o Bob, B, é A –> B ou B <– A. Por fim, mediadores e *relays* podem ser combinados em qualquer ordem e quantidade.

3.4 *Indy wallet*

Uma *wallet* de identidade é "um recipiente digital para dados que é necessário para controlar uma identidade auto-soberana" [103]. O SSI adotou o termo *wallet* do mundo das criptomoedas (*crypto wallets*), como sendo o local no qual toda a informação referente à identidade de um individuo é armazenada. Contudo, uma "crypto wallet" possui apenas um pequeno subconjunto de funcionalidades que uma *wallet* de identidade necessita. Enquanto que uma *crypto wallet* armazena apenas um pequeno conjunto de chaves, uma *wallet* de identidade armazena uma elevada variedade de dados, desde relacionamentos até credenciais verificáveis, e, no caso de uma organização de elevada dimensão, é necessário gerir milhares (ou até milhões) de relacionamentos, cada relacionamento com um conjunto de dados associados. Por este motivo na maioria dos casos de uso uma *crypto walllet* não irá funcionar de forma "out of the box" numa arquitetura SSI, isto é, uma *crypto wallet* não irá conter todas as funcionalidades requeridas pelo que será necessário efetuar algumas alterações a essa *wallet* de modo a torná-la compatível com um sistema SSI. Ao longo desta secção o termo *wallet* será utilizado para referenciar uma *wallet* de identidade.

Como foi mencionado, uma *wallet* de identidade armazena uma grande variedade de dados. Alguns exemplos destes dados são chaves criptográficas de cifra e assinatura, informação de identificação pessoal, credenciais verificáveis, fotografias e vídeos, receitas e registos de saúde. Nos exemplos apresentados é possível verificar que possuem não só diferentes estruturas e tamanhos como também diferentes riscos de segurança, isto é, perder uma credencial é algo que irá afetar temporariamente o *holder* mas um atacante não é capaz de personificar o *holder* apenas com essa credencial dado que não irá conseguir provar que a credencial lhe pertence sem ter conhecimento da chave privada associada. Por outro lado, se um atacante tiver acesso a uma chave privada do *holder*, pode usá-la para provar controlo de algo e levar a consequências mais graves. Isto obriga à necessidade de tratar cada tipo de dados de forma diferente tendo em conta as suas caraterísticas. Alguns dados requerem a criação de *backups*, outros requerem o seu armazenamento em locais mais seguros, e outros requerem uma velocidade de acesso mais elevada. É por isso necessário dividir a *wallet* em vários setores de modo a separar os dados de acordo com as propriedades desejadas.

É importante referir que uma *wallet* não é uma base de dados de propósito geral para armazenar todos os tipos de dados. Apesar de permitir armazenar qualquer tipo de dados, é recomendado que não seja utilizada como *cache* ou para armazenar dados irrelevantes.

3.4.1 *Arquitetura de uma Indy wallet*

Para lidar com os diferentes tipos de dados que são armazenados na *wallet*, a comunidade Hyperledger propôs a arquitetura representada na figura 48. Nesta arquitetura é possível distinguir dois componentes principais: "wallet core" e "pluggable storage".

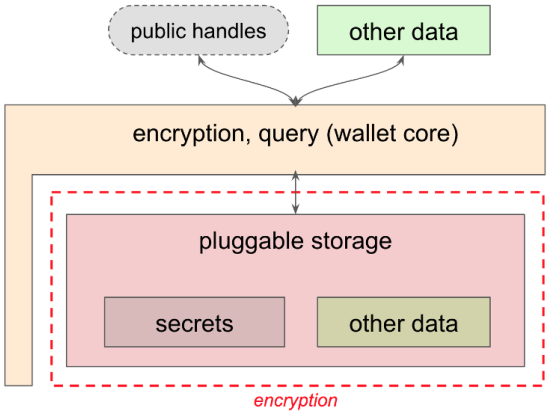


Figura 48: Arquitetura da *wallet* (Retirado de [103])

A *wallet core* é semelhante em todas as implementações de *wallets* no ecossistema de identidades descentralizadas e incluí a lógica afeta à criptografia, pesquisa e cuidados especiais no manuseamento dos dados armazenados na *pluggable storage*. A *pluggable storage* é uma base de dados que armazena todos os dados necessários a um sistema de gestão de identidades digital, agrupando-os em duas categorias: "segredos" e "não segredos". Entre os segredos estão as chaves privadas, DIDs, credenciais verificáveis, entre outros. Devido ao elevado risco de segurança os segredos constituem, nomeadamente chaves privadas, estes são mantidos num local seguro dentro da *wallet* durante todo o seu ciclo de vida, isto é, são gerados e preservados dentro de uma "câmara isolada" e nunca são expostos ao mundo exterior. Qualquer operação que necessite de chaves criptográficas secretas é efetuada dentro desse local de modo a maximizar a segurança das mesmas. Entre os não segredos encontram-se registos afetos aos protocolos de estabelecimento de conexões e de emissão e de apresentação de credenciais. É também nos não segredos que são armazenados quaisquer outros dados privados que o utilizador pretenda armazenar mas que não representem elevado risco de segurança ou privacidade. Na solução implementada são armazenados nos não segredos seis tipos principais de registos: convites, conexões, documentos de DIDs de emparelhamento, registos de revogação, emissões de credenciais e apresentações de provas.

A principal diferença entre segredos e não segredos está no modo como estes são armazenados e acedidos na *wallet*, isto é, enquanto que não segredos podem ser acedidos diretamente pelo seu dono, segredos apenas podem ser acedidos através dos *handlers* que lhes estão associados e não é possível extrair estes segredos para fora da *wallet*. Por exemplo, se o *holder* pretender utilizar a sua chave privada deverá indicar à *wallet* a respetiva chave pública, ficando a *wallet* responsável por efetuar as operações desejadas pelo *holder* sem expor a chave privada para o exterior. É importante referir que todos os dados são cifrados antes de serem armazenados na *wallet*, sejam eles armazenados na componentes dos segredos ou na componente dos não segredos, e que apenas o dono desses dados os pode consultar.

Uma *pluggable storage* pode ser constituída por um sistema de ficheiros, uma base de dados relacional ou não relacional, ou qualquer outro sistema de armazenamento de dados que possibilite a execução de procuras. O Hyperledger Indy [56] fornece uma base de dados SQLite, que opera de forma *out of the box* e que está conetada ao núcleo da *wallet* mas permite que esta base de dados seja facilmente substituída por uma base de dados externa através de uma API que permite registar uma nova implementação recorrendo a um conjunto de *handlers*.

A figura 49 explica a estrutura interna de uma *wallet* Indy [103]. Como podemos ver, a *wallet* apresenta três APIs: uma para a gestão da *wallet* (criar, abrir, fechar, apagar, etc), outra para o armazenamento de segredos (i.e. DIDs, credenciais, chaves criptográficas, entre outros), e uma última para o armazenamento de não segredos (dados específicos da aplicação). Tal como foi referido anteriormente, é possível observar nesta figura que tanto segredos como não segredos são cifrados antes de entrarem na base de dados (padrão ou externa) e decifrados após serem retirados dessa mesma base de dados.

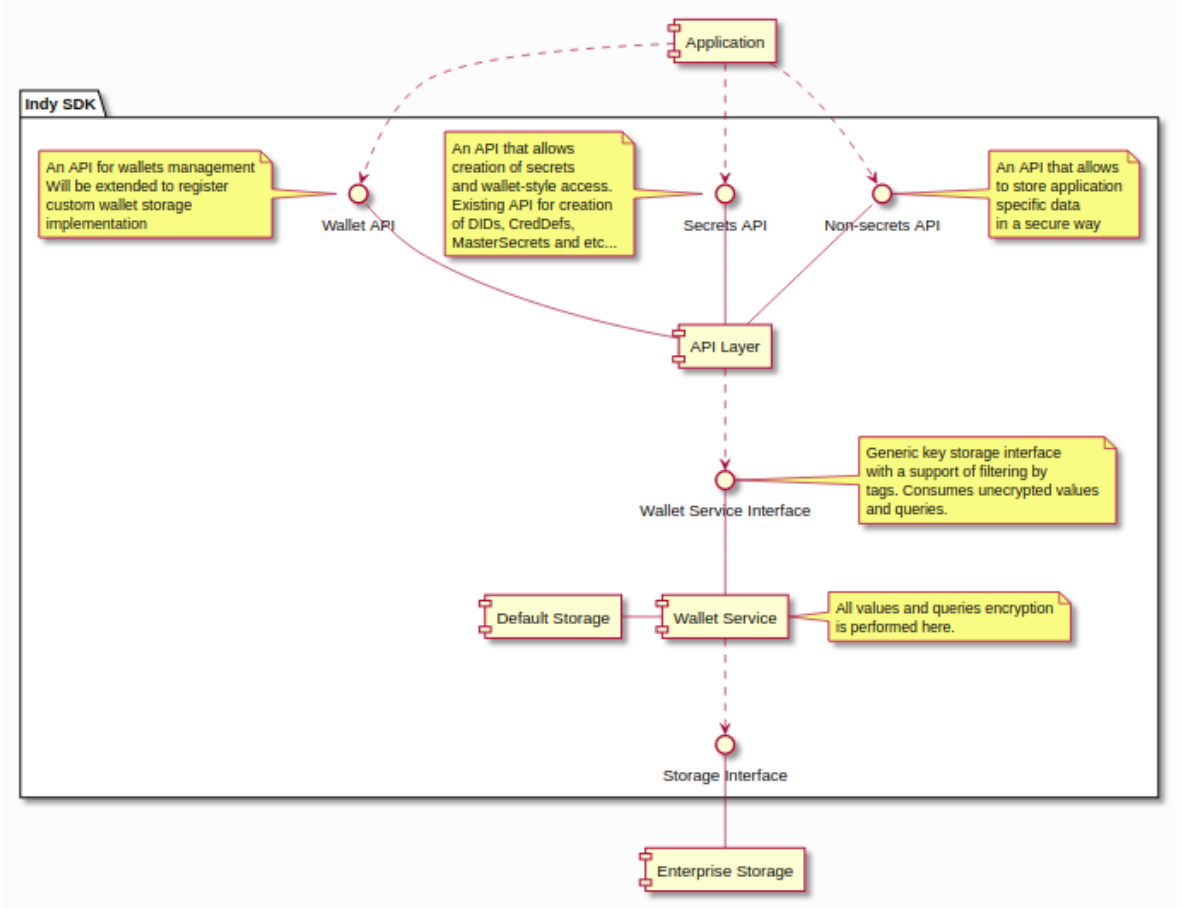


Figura 49: Componentes da *wallet* (Retirado de [103])

Na figura 49, a componente "wallet service" representa todos os serviços fornecidos pela componente "wallet core" apresentada na figura 48.

Esta arquitetura coloca toda a lógica de criptografia no núcleo da *wallet* (*wallet core*), ou seja, todos os dados são cifrados antes de entrarem no armazém de dados e decifrados após saírem desse mesmo armazém, com a exceção de certas *tags*, que serão explicadas na subsecção 3.4.2. O facto dos dados estarem cifrados durante todo o seu ciclo de vida dentro do armazém garante um elevado grau de segurança criptográfica a qualquer tipo de armazenamento que seja conectado a este núcleo. Consequentemente, o sistema de armazenamento de dados pode ignorar todos os aspetos ligados à proteção da segurança e privacidade dos dados, e focar-se apenas em garantir escritas e leituras eficientes.

Como foi referido anteriormente, é importante garantir que segredos, tais como chaves privadas, nunca saem para fora da *wallet*. Isto implica que todos os acessos a segredos são efetuados através dos seus respetivos *handlers* públicos. Todos os dados que pertencem à categoria dos não segredos podem ser acedidos diretamente. Adicionalmente, os processos de uma *Indy wallet* usam listas de controlo de acesso, baseiam-se no princípio do privilégio mínimo e todos os segredos gerados apenas permanecem em memória por tempo ínfimo até ser chamada uma função segura para os apagar.

Por fim, uma das principais preocupações com as *Indy wallets* é que não prendam um cliente a uma agência ou dispositivo específicos. Para tal, estas *wallets* suportam importação e exportação **total** de dados. Relativamente à importação de dados, uma *wallet* só é capaz de importar dados de outra *wallet* se ainda não possuir quaisquer dados. Já o mecanismo de exportação não suporta exportação seletiva dos dados sendo portanto necessário exportar os dados na sua totalidade. O mecanismo de exportação de dados pode também ser usado para criar *backups* de forma a mitigar perdas de chaves privadas ou dados importantes. Neste momento a comunidade do Hyperledger está a tentar implementar importação e exportação seletiva dos dados, bem como importação de dados para *wallets* que já possuem os seus próprios dados.

3.4.2 *Wallet Query Language (WQL)*

De maneira a consultar os dados armazenados, a *wallet* apresenta uma linguagem de procura própria [103]. Esta linguagem é baseada em JSON e apresenta uma gramática simples e semelhante à linguagem de procura do MongoDB, que pode ser facilmente mapeada para SQL ou GraphQL. O excerto de código apresentado na figura 50 mostra expressões regulares que definem as *queries* suportadas pela linguagem de procura da *wallet*. Neste excerto de código é possível observar que todas as *queries* recorrem aos valores de *tags* associadas aos registos armazenados para efetuar procuras robustas através de filtros. Estas *tags* são pares chave-valor (*tagName-tagValue*) e cada registo pode possuir zero ou mais *tags*. Tanto o nome como o valor destas *tags* devem ser representado por carateres alfanuméricos.

```
subquery = {subquery, ..., subquery} // subquery AND ... AND subquery
subquery = $or: [{subquery},..., {subquery}] // subquery OR ... OR subquery
subquery = $not: {subquery} // NOT (subquery)
subquery = "tagName": tagValue // tagName == tagValue
subquery = "tagName": {$neq: tagValue} // tagName != tagValue
subquery = "tagName": {$gt: tagValue} // tagName > tagValue
subquery = "tagName": {$gte: tagValue} // tagName >= tagValue
subquery = "tagName": {$lt: tagValue} // tagName < tagValue
subquery = "tagName": {$lte: tagValue} // tagName <= tagValue
subquery = "tagName": {$like: tagValue} // tagName LIKE tagValue
subquery = "tagName": {$in: [tagValue, ..., tagValue]} // tagName IN (tagValue, ..., tagValue)
```

Figura 50: *Queries* suportadas pela Indy *wallet* [103]

Apesar do sistema de *tags* apresentado permitir uma procura robusta e eficiente, algumas funcionalidades de um sistema de gestão de bases de dados relacionais não estão disponíveis, nomeadamente *joins* [103]. Uma *wallet* Indy permite o uso de *tags* cifradas e *tags* não cifradas. *Tags* cifradas permitem efetuar apenas procuras com os operadores "$eq", "$neq" e "$in", ou seja, apenas é possível verificar se o valor da *tag* é igual ou diferente de um determinado valor ou se está incluído num *array*. Já as *tags* não cifradas permitem efetuar procuras com todas as operações apresentadas anteriormente, isto é, em adição às operações indicadas para as *tags* cifradas permitem ainda efetuar as operações "$gt" (maior do que), "$lt" (menor do que) e "$like". *Tags* não cifradas diminuem a segurança e a privacidade dos dados mas permitem aumentar a complexidade das *queries*. Como segurança e privacidade são o foco de um sistema de gestão de identidades, a *wallet* usa *tags* cifradas por padrão.

3.4.3 *Criptografia*

Atualmente toda a criptografia aplicada aos dados é efetuada através das técnicas *ChaCha20-Poly1305* e *HMAC-SHA256*. A técnica *HMAC-Sha256* é usada múltiplas vezes para a criação de vetores de inicialização. Por sua vez, a técnica *ChaCha20-Poly1305* é utilizada principalmente para a cifragem de dados e para a criação de chaves. Uma *wallet* Indy recorre a diferentes estratégias para cifrar os dados de acordo com o seu tipo: item, *tag* cifrada, *tag* não cifrada e chaves criptográficas.

Itens

Cada item é constituído por três valores:

- *type*, que indica qual o tipo de dados a que o item pertence;
- *name*, que atua como um identificador do registo;
- *value*, um objeto *json* com a informação completa que se pretende armazenar.

O processo de cifragem de cada um destes valores varia ligeiramente conforme ilustrado na figura 51.

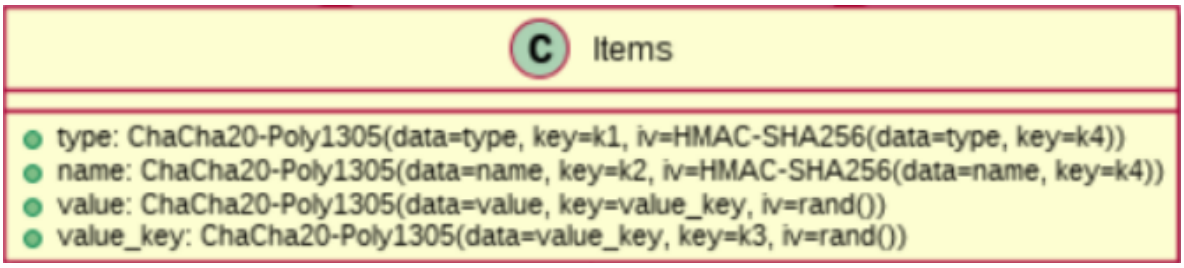


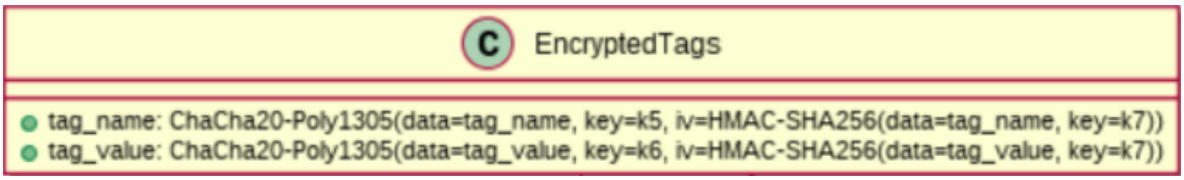
Figura 51: Cifra aplicada aos itens (Retirada de [103])

Como podemos observar, é usada a mesma técnica criptográfica para cifrar os três tipos de valores indicados, contudo a cifragem de cada valor é efetuada individualmente, sendo usada uma chave específica para cada valor, isto é, *type*, *name* e *value* são cifrados com as chaves **k1**, **k2** e **k3**, respetivamente. Adicionalmente as estratégias utilizadas para cifrar cada um destes campos são diferentes. Tanto o *type* como o *name* são cifrados de forma semelhante utilizando, respetivamente, as chaves k1 e k2 diretamente no algoritmo e recorrendo a uma quarta chave, **k4**, para gerar o vetor de inicialização com o auxílio da técnica *HMAC-SHA256*. Por sua vez, o campo *value* não utiliza diretamente a chave k3 no algoritmo mas sim uma chave derivada de k3, **value_key**, e recorre a um valor aleatório como vetor de inicialização. As chaves **k1**, **k2**, **k3** e **k4** são usadas na cifra de todos os itens armazenados na wallet enquanto que a chave **value_key** deve ser gerada de forma aleatória para cada item.

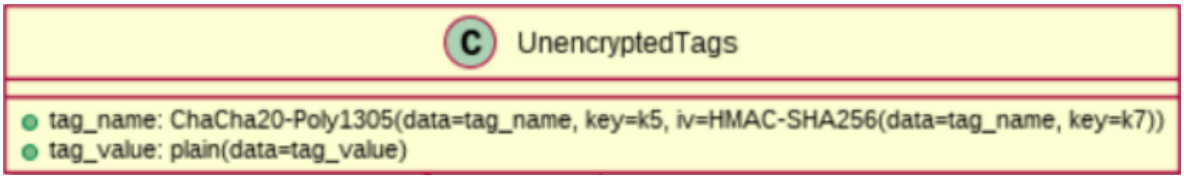
A razão de recorrer a estratégias de cifra diferentes para os valores de *type* e *name* e para os valores de *value* deve-se à necessidade de utilizar o *type* e o *name* para efetuar procuras sobre os dados cifrados e, consequentemente, recorrer a uma estratégia que dado um valor de entrada produza sempre o mesmo valor de saída. Deste modo ao efetuar procuras é possível aplicar a cifra apresentada acima ao valor que se pretende encontrar e compará-lo diretamente com os valores armazenados na *wallet*. Por outro lado, o objeto *json* presente no *value* nunca será acedido diretamente durante a execução de procuras na *wallet*, ou seja, é possível recorrer a uma estratégia que produza valores de saída aleatórios para um dado valor de entrada de modo a aumentar a segurança dos dados armazenados. Apesar dos valores de *type* e *name* não apresentarem uma segurança tão forte, tal não é necessário dado que estes valores não revelam informação sensível.

Tags

De maneira a disponibilizar procuras mais complexas com recurso a vários filtros, a *wallet* permite associar zero ou mais *tags* a cada item. Estas *tags* podem ou não ser cifradas, tal como foi explicado na subsecção 3.4.2. A figura 52 expõe a estratégia utilizada para *tags* cifradas (a) e *tags* não cifradas (b).



(a) *Tags* cifradas



(b) *Tags* não cifradas

Figura 52: Cifra aplicada às *tags* (Retirada de [103])

Na sub-figura 52a é possível observar o mecanismo de cifra aplicado a uma *tag* cifrada. Para este tipo de *tag*, tanto o seu nome como o seu valor são cifrados através do algoritmo *ChaCha20-Poly1305*, com as chaves **k5** e **k6**, respetivamente. Adicionalmente é usada uma terceira chave, **k7**, para gerar o vetor de inicialização através do algoritmo *HMAC-SHA256*. Já na sub-figura 52b é indicado o mecanismo de cifra aplicado a uma *tag* não cifrada. Contrariamente às *tags* cifradas, em *tags* não cifradas apenas é cifrado o seu nome, utilizando o mesmo algoritmo, chave e vetor de inicialização que são utilizados para o nome de uma *tag* cifrada. A necessidade de efetuar operações complexas sobre os valores destas *tags* (i.e. "maior que" ou "menor ou igual que") impossibilita a cifragem dos mesmos uma vez que não é possível efetuar comparações de superioridade sobre valores cifrados.

Enquanto que cifrar os valores das *tags* fornece maior privacidade, armazená-los em texto simples possibilita o uso operações de pesquisa mais complexas mas diminui a privacidade. Deste modo é necessário analisar cada caso de uso de forma individual e determinar a estratégia mais adequada para esse caso de uso.

Chaves

As quatro chaves utilizadas para cifrar os itens (k1, k2, k3 e k4) e as três chaves usadas para cifrar as *tags* (k5, k6 e k7) são concatenadas e cifradas com a chave mestra da *wallet* e o resultado é armazenado nos metadados da *wallet*. Isto permite que seja possível modificar a chave mestra da *wallet* sem necessidade de decifrar e cifrar novamente todos os dados armazenados na base de dados. A figura 53 mostra como é efetuada a cifra destas sete chaves, recorrendo uma vez mais ao algoritmo *ChaCha20-Poly1305*.

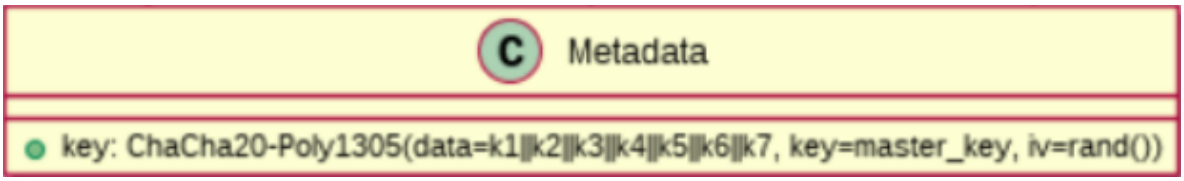


Figura 53: Cifra aplicada às chaves (Retirada de [103])

3.4.4 *Integração da indy wallet com o agente desenvolvido*

As secções anteriores procuraram descrever todas as caraterísticas de uma *indy wallet*. De seguida são apresentadas as várias razões de se ter optado por integrar este tipo de *wallet* no agente desenvolvido:

- A primeira razão deve-se ao facto desta *wallet* ser uma componente desenvolvida com a comunidade Hyperledger e que pode, por isso, ser integrada diretamente através do Indy SDK [55]. Esta integração permite utilizar as funcionalidades do Indy SDK para armazenar e adquirir vários dados necessários a um sistema SSI tais como definições de credenciais e as próprias credenciais, criar chaves criptográficas e utilizá-las para cifrar ou assinar dados, entre muitas outras operações;
- A segunda razão, e sem dúvida a mais significativa, deriva do modo como a componente foi criada, ou seja, desde a fase de desenho o principal foco sempre foi a sua utilização no ecossistema SSI, o que fez com que a segurança e a privacidade fossem uma preocupação durante todo o seu desenvolvimento. Tal como podemos ver na secção 3.4.1, a estrutura da *wallet* desenvolvida pelo Hyperledger protege os dados de diferentes formas tendo em conta a sua natureza e as estratégias utilizadas foram pensadas cuidadosamente;
- Por fim, mas não menos importante, esta *wallet* possibilita o armazenamento de qualquer tipo de dados através de uma associação de chave-valor. Esta caraterística é muito importante por permitir que o agente desenvolvido seja capaz de armazenar os dados associados à execução dos três protocolos apresentados na secção 3.3.2, que estão estruturados em formato JSON.

Resumidamente, a *wallet* desenvolvida pela comunidade *Indy* apresenta fortes preocupações com a segurança e privacidade, disponibilizada diversas funcionalidades necessárias a um sistema SSI e possibilita ainda o armazenamento de dados não estruturados de modo a que o agente do utilizador não precise de recorrer a bases de dados adicionais. Adicionalmente, apesar do nível de segurança associado os vários dados depender da sua natureza, todos os dados são cifrados antes de serem armazenados.

3.5 BLOCKCHAIN (INDY PLENUM + INDY NODE)

Uma das principais componentes de um sistema de gestão de identidades descentralizado é o registo descentralizado, também designado por *data registry*. Na solução desenvolvida este registo é uma *blockchain* criada com o auxilio de duas componentes semelhantes mas distintas: **Indy Plenum** [59] e **Indy Node** [58]. Todos os dispositivos que possuem estas duas componentes e mantêm uma cópia do registo descentralizado são designados por nós. Como foi indicado na parte introdutória da secção 3, o Indy Plenum é o coração da DLT, sendo responsável pela implementação do protocolo de consenso, enquanto que o Indy Node é uma extensão ao Indy Plenum que permite efetuar a gestão do *ledger* e dar suporte a transações específicas de um sistema SSI, atribuindo-lhes um tipo que vai influenciar a maneira como essas transações são processadas. A subsecção 3.5.1 aborda a componente Indy Plenum, nomeadamente os tipos de *blockchains* que existem e em qual deles se enquadra a *blockchain* desenvolvida bem como qual é e em que consiste o protocolo utilizado pela mesma. Já a subsecção 3.5.2 indica as várias operações disponibilizadas pela *blockchain*, assim como o sistema de permissões em vigor na mesma.

A *blockchain* implementada possuí quatro nós, cada um deles a correr isoladamente dentro do seu *container*. Estes quatro nós estão definidos no ficheiro de génese da *blockchain*, o que lhes confere o estatuto de nós de confiança. Através do consenso entre estes nós é possível integrar múltiplos nós adicionais. Devido à individualização de nós da *blockchain* em *containers*, é relativamente fácil correr várias instâncias através do *docker* e adicionar os respetivos nós à *blockchain*. Dado que esta dissertação tem como objetivo mostrar uma prova de conceito, quatro nós são suficientes para suportar o sistema desenvolvido. Num cenário real seria necessário ter vários nós a executar o protocolo de consenso, não só para tornar a *blockchain* mais resistente a ataques, falhas e ações mal intencionadas mas também para possibilitar a distribuição das transações enviadas pelos utilizadores pelos vários nós da rede de modo a proporcionar uma maior performance.

3.5.1 *Indy Plenum*

O Indy Plenum [59] é responsável por gerir o registo de transações (*ledger*), definindo o protocolo de consenso e as várias operações criptográficas da *blockchain*. Cada nó executa este protocolo de consenso, baseado em Tolerância Redundante a Falhas Bizantinas (RBFT) [104], de maneira a atingir consenso sobre o estado dos *ledgers*, isto é, quais transações já foram publicadas e qual a ordem e o momento da sua publicação.

Apesar da *blockchain* Indy ser muitas vezes mencionada como sendo um único *ledger*, é na verdade constituída por um conjunto de *ledgers*, cada um deles com um propósito distinto. Entre estes *ledgers* encontram-se o *pool ledger*, que é utilizado para controlar os participantes da rede (i.e. adição ou suspensão de nós, alteração de IP/porta ou chaves de

um nó, entre outros), o *domain ledger*, para controlar as identidades descentralizadas (i.e. adição de esquemas, alteração de chaves em documentos de DIDs, entre outros), e o *config ledger*, que é utilizado para autenticação e autorização (através de papeis e de regras) [105]. Por questões de simplicidade todos estes *ledgers* serão retratados como sendo um único *ledger* ao longo deste documento.

Nesta secção serão abordados os aspetos principais do núcleo da *blockchain*, ou seja, qual é o seu tipo e que protocolo é usado para atingir o consenso.

Tipo de blockchain

As *Blockchains* podem ser caraterizadas em duas dimensões: **acesso** e **validação** [72]. Quanto ao acesso, as *blockchains* podem ser **públicas**, nas quais qualquer utilizador pode ler o conteúdo da mesma, ou **privadas**, onde apenas utilizadores autorizados podem efetuar operações de leitura. Quanto à validação, *blockchains* podem ser ***permissioned***, isto é, apenas utilizadores com permissões podem submeter transações, ou ***permissionless***, onde não existe um sistema de permissões e, consequentemente, qualquer utilizador pode enviar novas transações. A secção 2.4 define com maior detalhe a distinção entre *blockchains permissioned* e *permissionless*.

Como foi referido na secção 3.1, o Indy possui uma *blockchain* pública e *permissioned*. O facto de ser pública implica que apenas informação pública deve ser armazenada. Toda a informação de caráter privado, nomeadamente informação de identificação pessoal, deve ser armazenada fora da *blockchain*. Por outro lado, o facto de ser *permissioned* permite-lhe individualizar várias operações de escrita diferentes e fornecer-lhes diferentes níveis de permissões. A subsecção *Sistema de permissões da blockchain* aborda os vários tipos de operações permitidas numa *blockchain* Indy bem como os níveis de permissões associados a cada uma delas.

Protocolo de consenso

Tal como as restantes implementações de *blockchains*, o Hyperledger Indy usa uma implementação de um algoritmo de consenso para decidir o conteúdo do próximo bloco a ser adicionado à *blockchain*. Esta implementação é designada por *Plenum Byzantine Fault Tolerant Protocol* (PRBFT) [59], uma implementação do algoritmo de tolerância redundante a falhas bizantinas (RBFT) [104] com algumas melhorias. O RBFT por sua vez é baseado no algoritmo *Byzantine Fault Tolerance* (BFT).

Em computação, uma falha bizantina é uma falha arbitrária que ocorre durante a execução de um algoritmo num sistema distribuído. Estas falhas podem dever-se a interrupções devido à própria execução do algoritmo (*crash failure*), problemas de *hardware* ou nós maliciosos na rede. *Bizantine Fault Tolerance* (BFT) é um algoritmo que procura resolver o problema dos generais bizantinos, que se baseia na necessidade de assegurar que os generais estabelecem

consenso num plano de batalha, comunicando entre si através de mensageiros. Contudo, um ou mais generais podem ser traidores que irão tentar confundir os restantes. Este problema é resolúvel se e só se mais de dois terços dos generais forem leais. Deste modo, por cada N generais traidores serão necessários 2N+1 generais leais para que seja possível atingir um consenso sobre o plano de batalha. O artigo *Understanding Blockchain Fundamentals, Part 1: Byzantine Fault Tolerance* [106] explica de uma forma simples e detalhada este conceito.

Como está descrito no relatório *Redundant Byzantine Fault Tolerance* [104], os protocolos de BFT existentes usam um nó especial, designado por *primary*, que indica aos restantes nós que transações devem ser processadas e qual a ordem de processamento. Este nó primário pode ser malicioso e agir contra o sistema sem ser detetado pelos restantes nós. O protocolo RBFT combate esse problema através da utilização de múltiplos nós, um nó *master* e um ou mais nós de apoio (*backups*). Todos estes nós emitem os pedidos, mas apenas os pedidos emitidos pelo nó *master* são efetivamente executados para serem adicionados à *blockchain*. Todos os restantes nós responsáveis por validar as transações comparam o desempenho do nó *master* com o dos nós de apoio, tanto a nível de performance como de resultados obtidos. Se o *master* não apresentar um desempenho aceitável é substituído por um dos nós de apoio. Este nó também será substituído se deixar de funcionar ou se atuar de forma maliciosa. O algoritmo usa a técnica de *round robin* para selecionar um novo líder. O nó selecionado para atuar como líder comunica o seu estado e adquire o estado dos restantes nós e, se estiver atrás deles, executa um protocolo para adquirir as transações em falta de forma a que quando começar a correr, todos os nós apresentem o mesmo estado [105].

O algoritmo do Plenum possui um desempenho semelhante aos restantes algoritmos Byzantine Fault Tolerance (BFT) em condições ideais, isto é, sem falhas de participantes, mas apresenta uma degradação de desempenho notavelmente inferior à dos restantes algoritmos quando falhas ocorrem na rede. Representando esta diferença em valores concretos, a degradação de desempenho no algoritmo do Plenum ronda os 3% enquanto que os restantes algoritmos BFT apresentam uma degradação de até 78% [72].

É importante referir que o número de falhas concorrentes de nós numa rede Indy é um fator crucial no algoritmo do Plenum. Para a rede conseguir sobreviver a **F** falhas concorrentes deve conter pelo menos **3F+1** nós. Consequentemente, para uma rede conseguir sobreviver à perda de um nó deve possuir no mínimo quatro nós. Do mesmo modo, para uma rede sobreviver à perda de dez nós deve possuir no mínimo trinta e um nós. Quanto maior for o número de nós na rede, mais resistente a falhas esta se torna. Contudo, quanto mais nós uma rede possuir, mais tempo será necessário para atingir um consenso. É portanto necessário ter em conta ambos os fatores e tentar encontrar o melhor equilíbrio entre estes eles. Anteriormente foi indicado que a Sovrin Foundation é uma organização mundial sem fins lucrativos que implementou uma instância da *blockchain* desenvolvida pelo Hyperledger. A estratégia utilizada pela Sovrin Foundation para combater o dilema apresentado passa por

ter um número elevado de nós mas apenas um subconjunto destes nós, designados *validators*, contribui no processo de validação e, consequentemente, no algoritmo de consenso. Os restantes nós, *observers*, apenas efetuam leituras à *blockchain*, podendo ser ativados como *validators* quando são detetadas falhas de nós que estavam a atuar como *validators* [72]. Com esta estratégia é possível manter um número razoável de nós a contribuir para o algoritmo de consenso e ainda retirar bastante carga destes nós através da utilização dos *observers* para a execução de operações de leitura.

Diferenças entre Plenum e RBFT

O protocolo Plenum tenta melhorar alguns aspetos do algoritmo RBFT nomeadamente no que diz respeito a fatores relacionados com a *blockchain*. Com base na documentação do Indy Plenum [105] foram extraídas duas diferenças entre estes algoritmos:

- O RBFT executa um *commit* de três fases (*3 phase commit*) para cada pedido (transação) enquanto que o algoritmo do Plenum executa um *commit* de três fases sobre um lote de pedidos. O relatório RBFT [104] contém informações mais detalhadas sobre *commits* de três fases. Esta melhoria demonstra ser consideravelmente significativa uma vez que cada um destes *commits* envolve n^2 comunicações, onde n corresponde ao número de nós. Deste modo é possível executar um *commit* para um lote de mil transações em vez de executar mil *commits*, um para cada transação;

- O Plenum introduziu um protocolo de agregação de assinaturas durante o processo do *commit* de três fases no qual todos os nós submetem a sua assinatura sobre o estado da *ledger*. Consequentemente, o cliente não é obrigado a depender da resposta de múltiplos nós sempre que pretender consultar o estado da *ledger* uma vez que lhe é fornecida a assinatura agregada. O esquema de assinatura usado é BLS (Boneh–Lynn–Shacham), uma das técnicas criptográficas disponibilizadas no projeto Ursa.

O facto desta implementação apresentar bons resultados levou à separação da *blockchain* Indy em dois projetos distintos, Hyperledger Plenum e Hyperledger Node, o que permite que o Plenum possa não só ser usado na implementação do Hyperledger Indy mas também em outras implementações de *blockchains*.

3.5.2 *Indy Node*

O Indy Node [58] é responsável por fornecer suporte a transações relativas à gestão de nós (i.e. adição e remoção de nós), adição, alteração e remoção de permissões e ainda a transações específicas de um sistema de gestão de identidades, sendo responsável por processar cada transação de acordo com o seu tipo. Dado que para uma *blockchain* todas as transações são vistas da mesma forma, o Indy Node adiciona um tipo a cada transação

antes de a publicar na *blockchain* de forma a ser capaz de processar os dados de forma diferente de acordo com o tipo de transação em questão. Entre os vários tipos de transação destacam-se o "0", para a gestão dos nós, o "1", para a gestão de permissões, o "100", para a publicação de atributos, o "101", para a publicação de esquemas, o "102", para a publicação de definições de credenciais e os "113" e "114", para a gestão dos registos de revogação. A figura 55 mostra todos os tipos de transações suportados pelo Indy Node. Adicionalmente também apresenta um conjunto de operações de leitura da *blockchain* de maneira a processar os dados de acordo com o seu tipo tal como mostra a figura 56. De uma forma geral o Indy Node é uma implementação da *blockchain* definida pelo Plenum sobre um sistema de identidades descentralizado e implementa os vários métodos necessários ao funcionamento desse sistema desde a criação de esquemas até à configuração dos nós da *pool*, isto é, a coleção de nós que estão a executar o protocolo de consenso. A subsecção *Operações de escrita e leitura* indica todas as operações de escrita e de leitura disponibilizadas pelo Indy Node e explica em que consistem algumas das operações mais comuns.

Sendo o Indy Node uma *blockchain permissioned*, apresenta um conjunto de papéis com permissões diferentes para lidar com os vários tipos de utilizadores da rede: *trustee*, *steward*, *endorser*, *network monitor* e *identity owner*. A subsecção *Sistema de permissões da blockchain* explora cada um destes cinco papéis e apresenta as várias permissões associadas aos mesmos.

Tal como em qualquer *blockchain*, também a *blockchain* Indy precisa de definir um conjunto de transações iniciais que definem quais os nós de confiança que irão executar o protocolo de consenso, designadas por transações génese da *pool*. Adicionalmente, por ser uma *blockchain* baseada em permissões, deve também conter um conjunto de transações iniciais que definem as várias entidades com as permissões mais elevadas, ou seja, os *trustees* e os *stewards*, designadas por transações de génese do domínio. Os *trustees* e os *stewards* presentes nas transações de génese são responsáveis por atribuir permissões aos restantes utilizadores da rede.

Operações de escrita e leitura

Dado que para uma *blockchain* todas as transações são equivalentes, isto é, são apenas conjuntos de dados, o Indy Node diferencia os vários tipos de transações através da adição de um atributo designado *type*. Este atributo determina qual o tipo de dados presente numa transação específica, o que permite efetuar procuras mais eficientes usando o tipo de dados como filtro. Por esta razão tanto os pedidos de escrita como os pedidos de leitura seguem uma estrutura JSON específica [107] tal como mostra a figura 54.

```
{
    "operation": {
        "type": <request type>, // tipo de pedido
        <request-specific fields> // campos íespecficos do pedido
    },
    "identifier": <author DID>,
    ‘endorser‘: <endorser DID>,
    "reqId": <req_id unique integer>,
    "protocolVersion": 2,
    "signature": <signature_value> // valor da assinatura
}
```

Figura 54: Estrutura de um pedido à *blockchain*

Nesta figura é possível observar que o valor do campo *type* identifica qual a operação que deverá ser executada. Esta operação pode ser de escrita ou de leitura dependendo do código passado. Se o código for "101" a operação corresponde à adição de um esquema à *blockchain* (operação de escrita) enquanto que se o código for "107" corresponde à consulta de um esquema (operação de leitura). Entre as várias operações disponíveis, algumas das mais utilizadas são a adição e consulta de esquemas, atributos e registos de revogação, a adição de remoção de nós da *blockchain* e a alteração de permissões dos utilizadores da *blockchain*. As figuras 55 e 56 indicam todas as operações disponibilizadas pelo Indy Node e quais os códigos associados as mesmas. Adicionalmente, com o objetivo de aumentar a flexibilidade das várias operações, podem ser adicionados atributos adicionais específicos do pedido ao atributo *operation*. No final da estrutura JSON é possível observar ainda o atributo *signature*. Este atributo é utilizado para autenticar os autores das transações enviadas para a *blockchain* e é obrigatório em todas as operações de escrita.

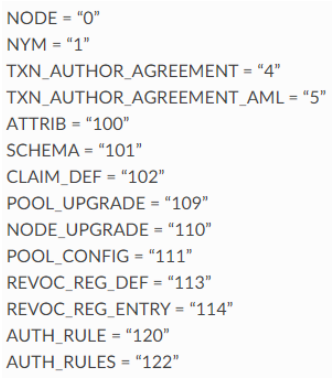


Figura 55: Tipos de operações de escrita

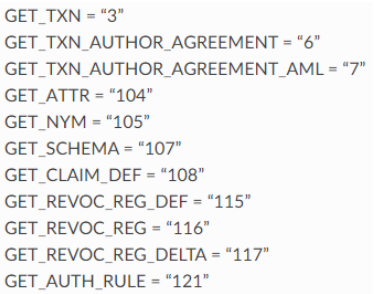


Figura 56: Tipos de operações de leitura

Apresentando resumidamente algumas das operações, "NODE" é utilizado para adicionar novos nós ou atualizar nós existentes, "NYM" é usado para adicionar ou alterar permissões de utilizadores, "SCHEMA" permite adicionar novos esquemas e "GET_ATTR" permite

consultar atributos associados a um utilizador, entre os quais se encontra o documento do DID. O documento *Hyperledger Indy Node Documentation* [107] detalha cada uma destas operações bem como os atributos adicionais que cada uma delas pode receber. Cada operação apresenta um código associado.

As subsecções A.3.1 e A.3.2 dos anexos apresentam exemplos de algumas operações de leitura e de escrita, respetivamente.

Sistema de permissões da blockchain

Nas figuras 55 e 56 foram apresentadas as várias operações disponibilizadas pelo Indy Node. A execução de operações de leitura não requer quaisquer permissões, ou seja, todos os utilizadores do sistema de gestão de identidades as podem efetuar. Relativamente às operações de escrita, cada operação apresenta um conjunto de papéis/entidades que as pode executar, isto é, algumas operações só estão disponíveis a utilizadores com um determinado tipo de papel (i.e. apenas *trustees* podem adicionar outros *trustees*) enquanto que outras só estão disponíveis a utilizadores específicos (i.e. alterar um atributo dentro de um documento de um DID apenas pode ser efetuado pelo dono desse DID). Nesta subsecção apenas serão discutidas permissões relativas às operações de escrita mais comuns: NODE, NYM, ATTRIB e SCHEMA. O documento *Default AUTH_MAP Rules* [108] detalha as permissões associadas a todas as operações apresentadas na figura 55.

Antes de analisar o sistema de permissões de cada uma das operações indicadas anteriormente é necessário caracterizar os vários tipos de utilizadores existentes: *trustee*, *steward*, *endorser*, *network monitor* e *identity owner*.

- *Trustee* representa as entidades mais confiáveis e com mais permissões do sistema e é principalmente responsável pela gestão da *pool* de nós que estão a correr o protocolo de consenso *blockchain* e pelo estabelecimento das regras de autenticação. Exemplos destas entidades são organizações relevantes interessadas em garantir que a rede é bem gerida;

- *Steward* é o papel desempenhado por entidades públicas bem conhecidas que estão dispostas a contribuir com dispositivos (nós) para a execução do protocolo de consenso da *blockchain*, sendo portanto responsáveis por adicionar, atualizar e remover os nós ativos (nós que estão a contribuir para estabelecer o consenso) e inativos (nós que estão em espera) que estão sob o seu controlo;

- *Endorser* é o papel atribuído às entidades que estão dispostas a assumir a responsabilidade legal e o esforço necessário para ajudar os autores das transações a escrever itens no *ledger*. Estas entidades também podem escrever no *ledger* transações da sua autoria, sendo esta a razão de alguns deles quererem ser *endorsers*. Deste modo, uma grande parte dos *issuers* acaba por pertencer a este grupo;

- *Network monitor* corresponde ao papel desempenhado por dispositivos ou entidades cuja função é garantir que os nós e a rede no geral estão a operar de forma eficiente. Por este motivo, *network monitors* possuem acesso a mais informações sobre os nós e a rede do que aquelas que estão disponíveis aos utilizadores comuns;
- *identity owner* corresponde a todas as entidades que possuem identidade, geralmente designados por *holders*.

As únicas operações de escrita que os *identity owners* podem executar são a adição e a alteração de atributos associados ao seu perfil público presente na *blockchain*, nomeadamente o documento do DID. Cada um dos papéis apresentados possui um código associado através do qual é representado na *blockchain* tal como mostra a tabela seguinte.

Código	Papel
0	trustee
2	steward
101	endorser
201	network monitor
None	identity owner

Dado que o Indy Node disponibiliza várias operações não é possível efetuar uma análise de permissões afetas a todas essas operações. Deste modo serão analisadas apenas algumas das operações mais comuns: NODE, NYM, ATTRIB e SCHEMA. Esta seleção de operações permite explicar todos os pontos fundamentais da *blockchain* Indy de forma clara e resumida.

As operações do tipo NYM estão fortemente ligadas à gestão de permissões dos utilizadores, permitindo registar, promover ou rebaixar utilizadores. Uma transação NYM permite associar um DID a um dos cinco papéis indicados anteriormente, sendo esta transação necessária para que o dono desse DID possa efetuar operações de escrita na *blockchain*. Utilizadores que não estão registados na *blockchain* com algum dos papéis indicados apenas podem efetuar operações de leitura. Quanto ao registo e promoção de utilizadores, um *trustee* pode atribuir qualquer um dos cinco papéis apresentados a qualquer utilizador e um *steward* apenas pode adicionar *endorsers*, *monitors* e *identity owners* e promover utilizadores já registados até ao nível de *endorser*, ou seja, não pode adicionar ou promover utilizadores para o nível *steward* ou *trustee*. Os restantes tipos de utilizadores não possuem permissões para alterar o papel de outros utilizadores. Quanto a rebaixar papéis, um *trustee* pode rebaixar qualquer outro utilizador, inclusive outros *trustees*, enquanto que um *steward* apenas pode rebaixar utilizadores de *monitor* para *identity owner*. Como consequência, um *trustee* pode alterar o papel de qualquer utilizador para *identity owner*, impedindo-o de continuar a tentar sabotar as informações da *blockchain*. É importante referir que as transações de adição e alteração de permissões de utilizadores ficam registadas na *blockchain* com a assinatura

de quem as efetuou pelo que é fácil detetar alguém que está a tentar prejudicar o correto funcionamento da mesma e reverter todas as transações efetuadas por essa entidade. As operações do tipo NYM permitem ainda que o dono de um DID efetue rotação das chaves associadas ao seu DID. Apenas o dono do DID pode alterar as chaves associadas a esse DID.

Por sua vez, as operações do tipo NODE permitem efetuar uma gestão granular dos vários nós que estão a executar a *blockchain* sendo possível adicionar nós no estado ativo ou inativo, promover (ativar) e rebaixar (desativar) nós e alterar atributos desses nós tais como o endereço de IP e a porta. As operações de adicionar, promover e rebaixar um nó podem ser efetuadas por qualquer *trustee* e pelo *steward* responsável por esse nó enquanto que as operações de alterar atributos desse nó apenas podem ser efetuadas pelo *steward* responsável.

Outro tipo de operações muito usado é o ATTRIB. As operações do tipo ATTRIB permitem adicionar e editar atributos associados a um DID registado na *blockchain*. O sistema permite que cada utilizador possa adicionar os atributos que desejar, contudo a maioria dos utilizadores de um sistema de gestão de identidades interage com esse sistema através de agentes de utilizadores. É portanto responsabilidade destes agentes disponibilizarem mecanismos de adição e edição de atributos. Na grande maioria dos casos de uso, o único atributo que precisa de ser armazenado na *blockchain* associado ao DID é o seu respetivo documento. Por este motivo o agente de utilizador desenvolvido apenas disponibiliza operações do tipo ATTRIB afetas ao atributo referente ao documento do DID. Apenas o dono de um DID presente na *blockchain* pode adicionar, alterar ou remover atributos associados a esse mesmo DID.

Por fim as operações do tipo SCHEMA permitem criar esquemas. Apenas *trustees*, *stewards* ou *endorsers* podem adicionar esquemas à *blockchain* e ninguém pode alterar esquemas. Se for necessário alterar algum valor num esquema é necessário criar um novo esquema.

Os quatro tipos de operações apresentados mostram que o sistema de permissões pode ser tão flexível quanto necessário, permitindo fornecer permissões de acordo com o papel dos utilizadores ou com os seus DIDs específicos.

4

CASOS DE USO IMPLEMENTADOS

Com o objetivo de analisar o modo de funcionamento do sistema de identidades descentralizado desenvolvido quando aplicado ao mundo real foram desenvolvidos três casos de uso distintos. Estes três casos de uso procuram demonstrar que este sistema pode ser aplicado em vários setores totalmente distintos seguindo os mesmos padrões. A subsecção 4.1 descreve um caso de uso mais simples no qual um cidadão obtém um atributo (número de identificação civil) do serviço de identificação nacional e usa esse mesmo atributo para exercer o seu direito de voto. Já a subsecção 4.2 exemplifica uma aplicação prática de sistemas de identidades descentralizados no combate a um dos problemas com mais impacto social da atualidade, o vírus COVID-19. Este caso de uso visa representar uma forma de controlar o acesso de indivíduos a espaços de risco, tais como cinemas ou estádios de futebol, através da validação de credenciais que indicam se esses indivíduos se encontram ou não infetados pelo vírus. Por fim, a subsecção 4.3 descreve um caso de uso um pouco mais complexo no qual o cidadão precisa de obter atributos de duas entidades distintas, o serviço de finanças e o seu atual empregador, de modo a apresentá-los ao banco no momento da requisição de um empréstimo bancário. Apesar de nesta secção serem retratados apenas três casos de uso com uma complexidade simples/moderada, sistemas de gestão de identidades descentralizados podem ser aplicados em diversas áreas e com vários graus de complexidade.

Este documento apresenta ainda uma secção nos anexos, A.4, na qual é apresentada uma demonstração prática da aplicação do sistema desenvolvido ao caso de uso da eleição, acrescentando-lhe mais alguma complexidade com o objetivo de demonstrar as várias funcionalidades deste sistema, desde o uso de divulgação seletiva e de provas de conhecimento zero até à revogação de credenciais.

Antes de iniciar a análise detalhada dos três casos de uso apresentados, é importante referir que qualquer interação no sistema de gestão de identidades descentralizado desenvolvido requer o pré-estabelecimento de uma conexão entre as entidades intervenientes. A figura 57 mostra as várias interações que cada entidade envolvida deverá executar antes da emissão ou apresentação de credenciais ou de qualquer outro tipo de comunicação de dados privados. Nesta figura, a entidade 1 representa a entidade que está à espera de receber conexões de outras entidades, seja para requisitarem credenciais ou para apresentarem provas. Deste

modo, a entidade 1 deve publicar um convite num local acessível ao seu público alvo (i.e. URL, QR Code em espaços físicos, entre outros).

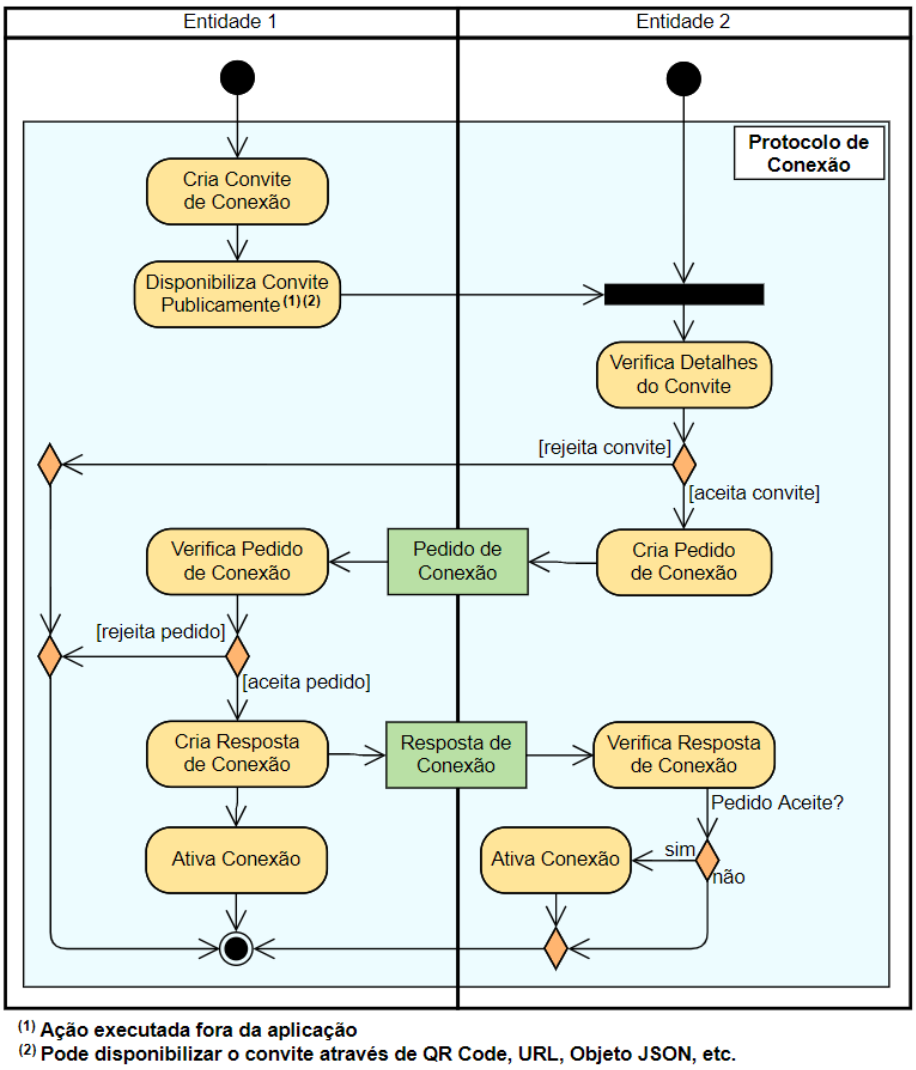


Figura 57: Diagrama de atividade do protocolo de conexão

Por questões de simplicidade, o estabelecimento de uma conexão não foi representado nos diagramas das figuras 58, 59 e 60, apesar de ser algo estritamente necessário em todas as comunicações representadas nos vários casos de uso. É, contudo, possível reutilizar uma conexão já existente de forma a não ser necessário acordar uma nova conexão sempre que duas entidades específicas pretenderem comunicar entre si.

Por fim, é também importante realçar que os diagramas apresentados representam uma versão simplificada dos protocolos apresentados na subsecção 3.3.2. Nestes diagramas são ignoradas mensagens de confirmação (ACKs), mensagens de rejeição de propostas e pedidos, mensagens de comunicação de erros, entre outras.

4.1 ELEIÇÃO

Como primeiro caso de uso será analisado o cenário no qual um cidadão obtém um atributo simples e utiliza-o para executar o seu direito de voto. O serviço eleitoral requer que o cidadão forneça o NIC (número de identificação civil), que o identifica unicamente como cidadão daquele país. Deste modo, caso o cidadão não possua credenciais verificáveis que contenham o NIC, deverá contactar o serviço de identificação para obter uma credencial com o NIC. Após a obtenção dessa credencial, o cidadão deverá gerar uma apresentação verificável com o NIC e enviá-la ao serviço eleitoral de maneira a poder exercer o seu direito de voto.

Tendo em conta o caso de uso descrito é possível destacar 3 entidades intervenientes:

- cidadão (***holder***) - entidade sobre a qual as credenciais são emitidas;
- serviço de identificação (***issuer***) - entidade responsável por emitir a credencial verificável com o NIC do cidadão;
- serviço eleitoral (***verifier***) - entidade responsável por validar se o cidadão está apto a efetuar o seu direito de voto;

A figura 58 mostra o diagrama de atividade referente ao caso de uso descrito. Neste diagrama podemos ver duas partes distintas: a execução do protocolo de emissão de credencial e a execução do protocolo de apresentação da prova.

Como foi referido na parte introdutória deste capítulo, antes da execução de qualquer um destes dois protocolos é necessário que os intervenientes tenham estabelecido uma conexão através do protocolo de conexão, como demonstrado na figura 43.

Após a conexão estar estabelecida, o cidadão propõe uma proposta para a emissão de uma credencial ao serviço de identificação. Recebendo a proposta, este verifica se os dados estão corretos e, em caso afirmativo, apresenta uma oferta ao cidadão. É importante referir que o serviço pode iniciar o protocolo através do envio direto de uma oferta de uma credencial. Ao receber a oferta, o cidadão verifica se os dados que serão emitidos na credencial são os corretos e procede à efetuação de um pedido para emissão da credencial. Se os dados do pedido estiverem coerentes com os que foram indicados na oferta, o serviço de identificação emite e envia a credencial ao cidadão, com os atributos digitalmente assinados de modo a que o cidadão os possa usar como prova futuramente.

Ao receber a credencial, o *holder* deverá armazená-la no seu repositório privado (i.e. *wallet*) de modo a poder utilizá-la mais tarde.

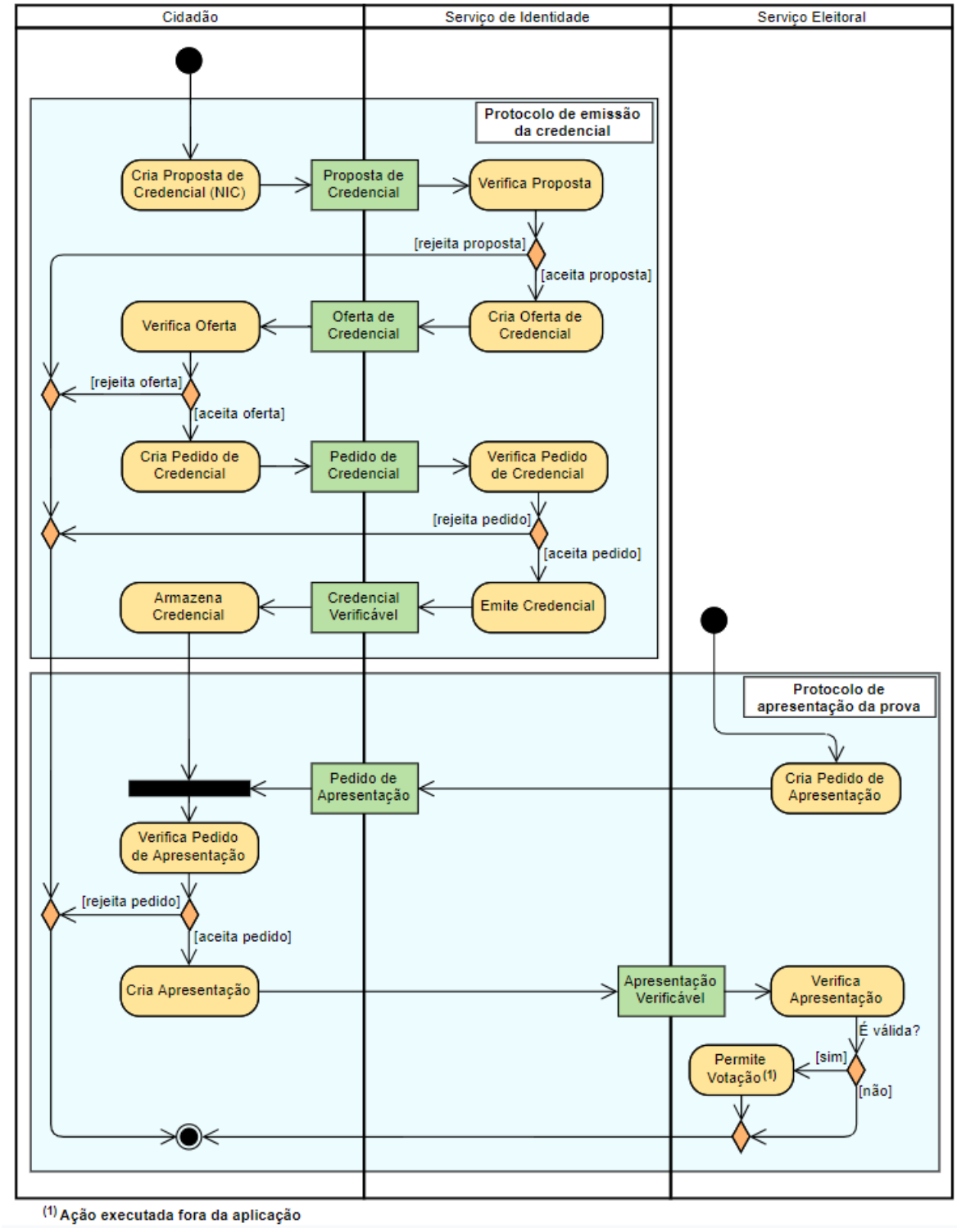


Figura 58: Diagrama de atividade do processo de votação

Relativamente à segunda parte do diagrama, o cidadão dirige-se a uma mesa de voto e estabelece uma conexão com o serviço eleitoral. Como foi explicado anteriormente, o

estabelecimento de conexões não é representado no diagrama para simplificar o processo. Após a conexão estar estabelecida, o serviço eleitoral inicia o protocolo através do envio de um pedido de apresentação da prova ao cidadão, no qual requisita o atributo NIC. Como o cidadão adquiriu uma credencial com este atributo anteriormente, seleciona esse atributo da sua credencial, adiciona-o à prova e envia esta prova ao serviço eleitoral. Apesar de neste caso a credencial conter apenas um atributo, se esta contiver vários atributos e se o *issuer* fornecer à credencial a capacidade de divulgação seletiva, o cidadão pode adicionar apenas um desses atributos à prova de tal forma que os restantes atributos não seriam divulgados ao *verifier*. É obrigatório que o *issuer* indique explicitamente a permissão para o uso de divulgação seletiva em certos atributos uma vez que alguns atributos devem ser apresentados em conjunto, e quando apresentados individualmente não deverão possuir qualquer valor de prova. Por fim, o serviço eleitoral verifica a prova apresentada e, se não detetar nenhum problema, permite ao cidadão exercer o seu direito de voto.

4.2 VERIFICAÇÃO DE RESULTADOS DE COVID-19

Um dos principais problemas da atualidade trata-se do combate ao vírus designado por COVID-19. O caso de uso apresentado nesta subsecção mostra uma forma de lidar com o controlo de pessoas infetadas através do uso de um sistema de identidades descentralizado e com o auxílio de credenciais verificáveis capazes de provar o resultado de um teste efetuado por um indivíduo. Tais credenciais são úteis para restringir a entrada de pessoas infetadas em cinemas, complexos desportivos ou quaisquer outros eventos sociais. Neste caso de uso, um indivíduo pretende ir ao cinema. Contudo, apenas é permitida a entrada a pessoas que tenham efetuado o teste nos últimos 5 dias cujo resultado tenha sido negativo. Para obter uma credencial verificável que comprove que o resultado do teste ao COVID-19 foi negativo, o indivíduo deve deslocar-se ao serviço de saúde, efetuar o teste e obter a respetiva credencial verificável com o resultado desse teste.

O caso de uso indicado é baseado em duas interações simples, uma para a obtenção da credencial e outra para a apresentação, nas quais intervêm:

- indivíduo (***holder***) - entidade que efetua o teste e apresenta o resultado do mesmo ao cinema;
- serviço de saúde (***issuer***) - entidade responsável por efetuar os testes ao COVID-19 e emitir credenciais que comprovem o resultado do teste;
- cinema (***verifier***) - entidade que organiza eventos sociais e que verifica se os indivíduos estão ou não infetados com COVID-19 à entrada desses eventos.

Na figura 59 é possível observar o diagrama de atividades que representa o processo de obtenção de uma credencial verificável com a prova do resultado obtido ao teste de COVID-19 bem como o processo de apresentação dessa mesma prova.

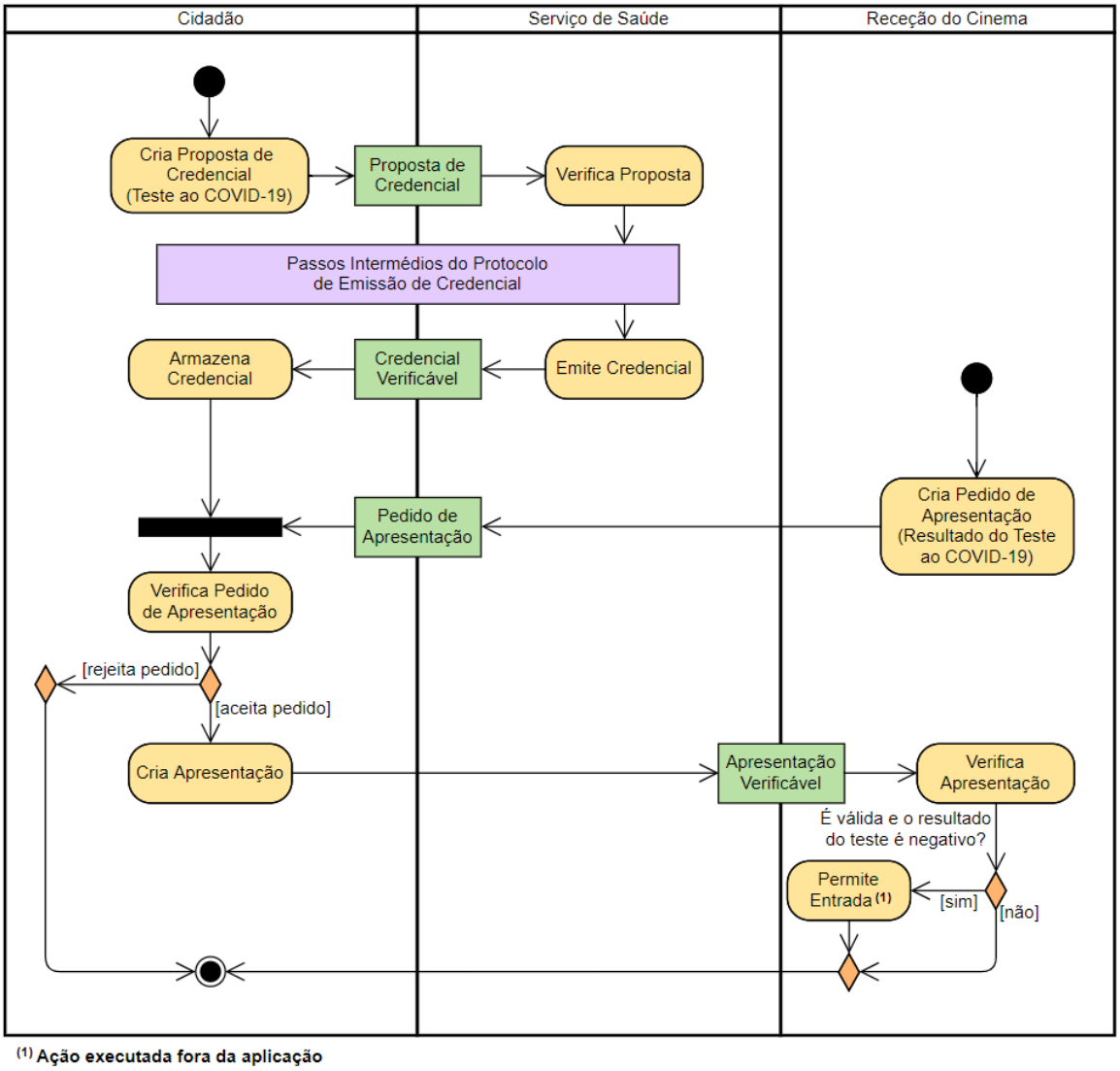


Figura 59: Diagrama de atividade da verificação de testes de COVID-19

No diagrama apresentado existem duas interações, uma entre o cidadão e o serviço de saúde e outra entre o cidadão e a receção do cinema. Assim como no caso de uso anterior, também estas duas interações requerem o estabelecimento prévio de conexões para efetuar uma comunicação segura. Deste modo, as entidades precisam de estabelecer uma conexão através do protocolo de conexão, como exemplificado na figura 43. Por questões de simplicidade, o processo de estabelecimento destas conexões não está representado no diagrama.

Após o estabelecimento da conexão entre o cidadão e o serviço de saúde, o processo de emissão da credencial que contém o resultado do teste ao vírus é semelhante ao processo de emissão da credencial com o NIC do cidadão descrito na subsecção anterior. Dada a semelhança entre ambos os diagramas, o diagrama presente na figura 59 oculta os passos

intermédios do protocolo de emissão da credencial. Possuindo a credencial, o cidadão pode então estabelecer uma conexão com a receção do cinema e fornecer o resultado do teste assim que a receção requisitar tal informação. Após receber a apresentação da prova por parte do cidadão, a receção do cinema verifica o resultado do teste e a validade da prova (i.e. se foi emitida nos últimos cinco dias e se o emissor é uma entidade de confiança) e, caso toda a informação esteja correta, permite que o cidadão entre no cinema.

4.3 EMPRÉSTIMO BANCÁRIO

Como último caso de uso será analisada uma situação do quotidiano um pouco mais complexa que irá envolver duas entidades emissoras e a possibilidade de recorrer a provas de conhecimento zero que permitem a um *holder* provar predicados (i.e. idade superior ou igual a dezoito anos ou salário mensal superior a mil euros) sem revelar o valor em concreto, bem como a junção de atributos de duas credenciais distintas numa única apresentação verificável que será apresentada ao *verifier*.

Neste caso de uso, o cidadão pretende obter um empréstimo bancário. Contudo, o banco requisita o seu NIF (número de identificação fiscal) e uma prova, emitida pelo seu empregador, que prove que recebe um salário mensal superior a mil euros. Para continuar com o processo, o cidadão adquire uma credencial do serviço de finanças que contém o seu NIF e outra credencial do seu empregador que contém as informações relativas ao seu emprego. Tendo ambas as credenciais, o cidadão gera uma prova com os dois atributos requeridos e apresenta-a ao banco.

Tendo em conta o caso de uso descrito é possível destacar 4 entidades intervenientes:

- cidadão (***holder***) - entidade sobre a qual as credenciais são emitidas;
- serviço de finanças (***issuer***) - entidade encarregada por emitir a credencial verificável com o NIF do cidadão;
- empresa (***issuer***) - entidade que emprega atualmente o cidadão, sendo responsável por emitir a credencial verificável com os atributos de empregado do respetivo cidadão;
- banco (***verifier***) - entidade que atua como instituição financeira e que faculta um empréstimo ao cidadão após validar as informações requisitadas.

A figura 60 demonstra o diagrama de atividade que representa o caso de uso de um empréstimo bancário. Neste diagrama, o cidadão requisita credenciais ao serviço de finanças e ao seu empregador, cria uma apresentação verificável com os atributos requeridos pelo banco e envia-lhe essa apresentação verificável de maneira a obter o empréstimo bancário. A requisição das duas credenciais pode ser efetuada em paralelo.

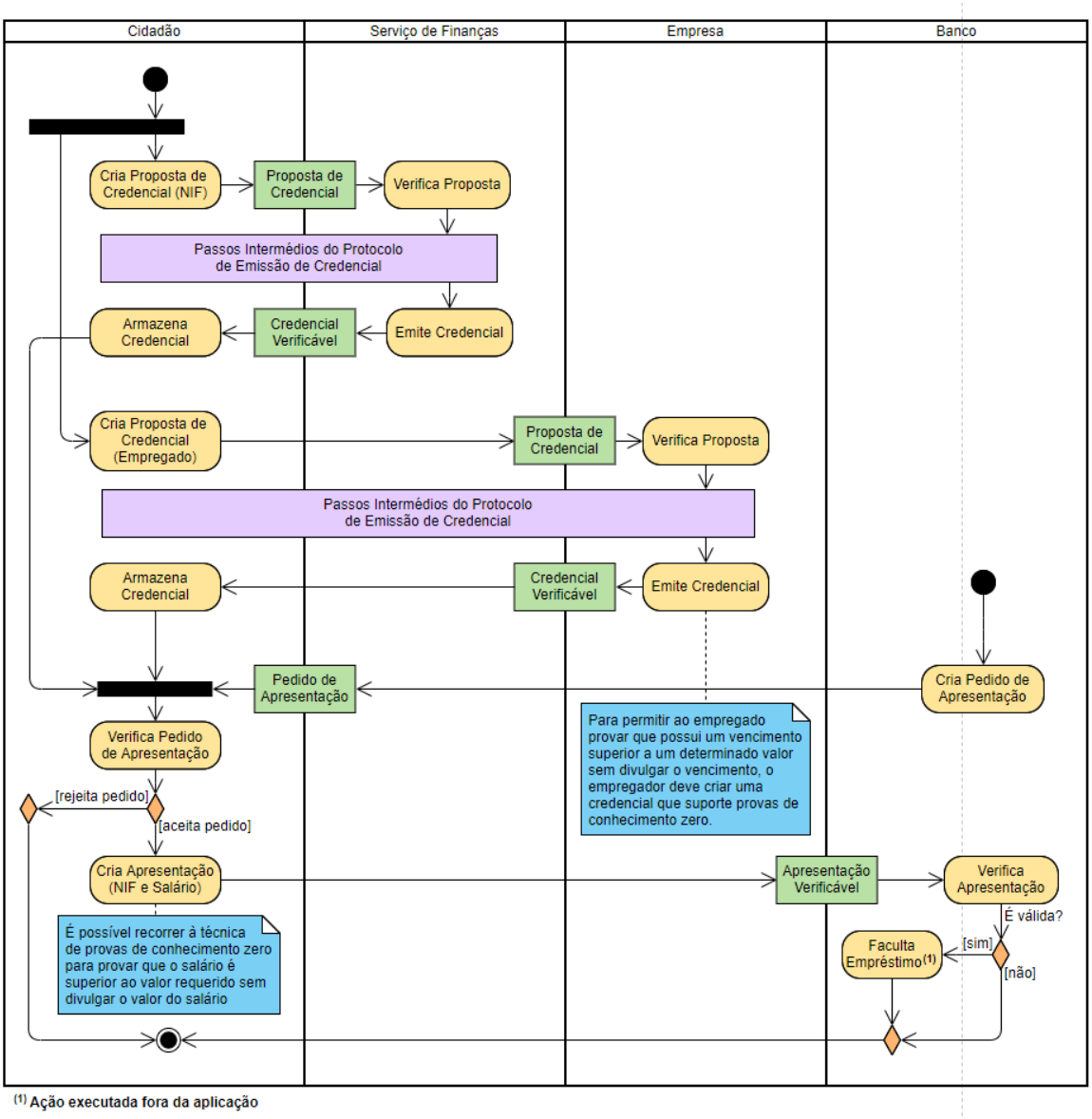


Figura 60: Diagrama de atividade de um empréstimo bancário

O diagrama apresentado inclui interações do cidadão com as outras três entidades indicadas anteriormente: o serviço de finanças, a empresa que emprega o cidadão, e o banco ao qual o cidadão pretende requisitar um empréstimo bancário. Para comunicar com estas entidades o cidadão deve estabelecer previamente uma conexão com cada uma delas recorrendo ao protocolo de conexão, como mostra a figura 43. Uma vez mais, por questões de simplicidade, o estabelecimento destas conexões não se encontra representado no diagrama.

A primeira parte do diagrama representa a execução dos dois protocolos de emissão de credenciais, uma que contém o NIF do cidadão e outra com as informações relativas ao seu emprego. Os passos intermédios destes dois protocolos são semelhantes ao protocolo de

emissão da credencial com o NIC, representado na figura 58, razão pela qual são omitidos neste diagrama. Ao emitir a credencial para o seu empregado, o empregador assegura-se de utilizar a técnica de prova de conhecimento zero no atributo referente ao salário mensal do empregado para que o cidadão possa provar que recebe mais que um determinado valor sem revelar o valor concreto.

Relativamente à segunda parte do diagrama, o cidadão contacta o banco informando-o de que tem a intenção de pedir um empréstimo. A primeira interação do protocolo de apresentação da prova pode ser iniciada pelo cidadão, através do envio de uma proposta de apresentação utilizando a aplicação, ou pelo banco, através do envio direto de um pedido de apresentação. No caso de uso apresentado é utilizada a segunda opção, isto é, o cidadão contacta o banco através de um telefonema a informar da sua intenção de obter um empréstimo e o banco fica assim encarregue de iniciar o protocolo através da requisição do NIF (número de identificação fiscal) e de uma prova de que o salário mensal do cidadão é superior a mil euros. Apesar de o cidadão ter efetuado o telefonema, esta ação não está incluída no protocolo de apresentação da prova dado que é efetuada fora da aplicação. Após o cidadão obter a credencial do serviço de finanças, com o NIC, e a credencial do seu empregador, com as informações do seu emprego, verifica os atributos requeridos pelo banco, indicados no pedido de prova que recebeu do mesmo, e seleciona os atributos de ambas as credencias que correspondem aos atributos requeridos. Entre estes atributos, o cidadão repara que um deles corresponde a uma prova de que recebe um salário mensal superior a mil euros. Consequentemente, na criação da apresentação o cidadão poderá recorrer à técnica de provas de conhecimento zero para provar que o seu salário é superior a mil euros ser revelar o valor exato. O processo de utilização de provas de conhecimento zero está incluído na aplicação de forma transparente ao cidadão, ou seja, este apenas precisa de indicar que pretende gerar uma prova de que o valor do seu salário é superior a mil euros. É importante reforçar que o *issuer* está responsável por permitir o uso de provas de conhecimento zero e divulgação seletiva de atributos nas credenciais que emite. Para o cidadão poder gerar uma prova de conhecimento zero referente ao seu salário, o seu empregador precisa de inserir criptograficamente na credencial que tal operação é suportada para esse atributo em específico. Ao receber a prova, o banco verifica a validade da mesma e, se todas as informações requeridas estiverem corretas, inicia o processo de empréstimo bancário ao cidadão.

5

CONCLUSÃO

A necessidade de partilhar informação no mundo digital tem crescido exponencialmente e a falta de uma solução genérica para armazenar e gerir a informação de identificação pessoal obrigou os fornecedores de serviços a desenvolver estratégias para armazenar os dados pessoais dos seus clientes. Contudo estas estratégias têm-se demonstrado bastante inviáveis devido aos custos de armazenamento e segurança que estes fornecedores são obrigados a despender, aos ataques que as bases de dados sofrem constantemente ou até mesmo à partilha indevida de dados pessoais de clientes tendo em vista a obtenção de lucro. Por outro lado os dados de identificação pessoal que constroem a identidade de indivíduo devem ser armazenados num local à sua escolha e controlados por si. São vários os pontos de falha que os sistemas atuais apresentam no que diz respeito ao armazenamento e gestão de dados pessoais, contudo todos eles convergem num único problema: a necessidade por parte dos fornecedores de serviços em armazenar e gerir todas as informações afetas aos seus clientes de forma centralizada. Deste modo é extremamente necessário desenvolver soluções baseadas em *self-sovereign identity*, ou seja, soluções que colocam os utilizadores no centro do modelo, sendo cada um deles responsável por controlar todos os seus dados de caráter pessoal. As várias vantagens de recorrer a uma arquitetura *self-sovereign identity*, nomeadamente o aumento de privacidade e segurança dos dados privados dos utilizadores e a remoção da responsabilidade e dos custos que uma empresa é obrigada a despender para manter esses dados fizeram com que este tema ganhasse bastante notoriedade a nível mundial e atraísse grandes empresas para o desenvolvimento de soluções baseadas em SSI, entre as quais a Microsoft. Adicionalmente, SSI apresenta vários casos de uso interessantes e complexos que requerem a aplicação de tecnologias descentralizadas como a *blockchain*, o que atrai vários entusiastas por este tipo de tecnologias. O elevado crescimento de interesse de governos e outras grandes organizações, bem como da comunidade mundial em geral, na substituição dos modelos de gestão de identidades atuais por modelos descentralizados, isto é, centrados nos utilizadores, e o facto de serem usadas tecnologias inovadoras faz com que este seja atualmente um dos temas mais interessante para explorar. Devido à velocidade com que novas soluções viáveis têm aparecido e ao facto das mesmas poderem ser usadas em todos os setores da indústria é expectável que em poucos anos uma grande parte dos

cidadãos já irá recorrer a sistemas baseados em *self-sovereign identity* no seu dia-a-dia para executar uma variada panóplia de ações, desde interações com entidades bancárias até à utilização de credenciais verificáveis no acesso a complexos desportivos e cinemas.

Como esta nova forma de lidar com os dados privados é transversal a todos os setores de negócio e a todas as idades e pretende substituir todas as formas de identificação digital existentes, é necessário desenvolver agentes intermediários capazes de ajudar todos indivíduos a interagirem com um sistema complexo de forma fácil e intuitiva, independentemente do seu nível de conhecimento tecnológico. Se a solução não obtiver adesão por parte da população em geral, não será possível substituir os sistemas de gestão de identidades tradicionais por sistemas de gestão de identidades descentralizados. É portanto fulcral criar aplicações (agentes de utilizadores) que sejam não só fáceis de usar mas também apelativas, e que permitam ao utilizador despender menos esforço na execução das suas tarefas do dia-a-dia.

Esta dissertação visa não só explorar o estado da arte referente ao conceito de *self-sovereign identity* mas também fornecer uma solução que permita efetuar uma prova desse conceito. Deste modo foi também desenvolvida uma solução que oferece uma alternativa, baseada em *self-sovereign identity*, com um agente de utilizador que procura ajudar qualquer indivíduo a interagir com um sistema de gestão de identidades descentralizado intuitivamente de modo a estabelecer conexões seguras com outras entidades através de técnicas criptográficas modernas, controlar todas as suas informações privadas, requisitar a emissão de credenciais verificáveis digitalmente e gerar provas criptográficas que incluem os vários atributos requisitados pelos fornecedores dos serviços. O sistema desenvolvido integra ainda uma *blockchain* (*data registry*) com quatro nós e uma *wallet* (*repository*) criptograficamente segura, ambas desenvolvidas pelo Hyperledger. Tanto a *blockchain* como a *wallet* foram projetadas especificamente para serem utilizadas dentro de sistemas de gestão de identidades descentralizados, o que fez com que a segurança e a privacidade fossem uma preocupação durante todo o processo de desenvolvimento destas componentes. Consequentemente, estas componentes apresentam elevada segurança e proteção de privacidade em toda a sua arquitetura interna.

Durante o desenvolvimento da solução foram tidas em consideração as várias boas práticas de desenvolvimento de software bem como as boas práticas definidas pelo W3C para a representação das credenciais, comunicação entre agentes, geração de provas, entre outras. Adicionalmente a solução desenvolvida implementa várias das sugestões indicadas pelo W3C para o desenvolvimento de uma arquitetura baseada em *self-sovereign identity*. No geral todas as componentes foram implementadas/integradas com sucesso e os agentes são capazes de estabelecer comunicações seguras entre si de modo a realizarem todos os processos de troca de informação presentes num sistema de gestão de identidades descentralizado. Por fim o agente do utilizador desenvolvido apresenta uma interface intuitiva e com todas as funcionalidades necessárias ao correto funcionamento de um sistema de gestão de identidades descentralizado.

A quantidade de informação privada a circular no mundo digital tem crescido exponencialmente e tudo indica que irá continuar a crescer a um ritmo muito elevado. Consequentemente, a preocupação com a proteção destes dados fará com que *self-sovereign identity* se torne num dos temas mais discutidos no futuro, razão pela qual recomendo vivamente aos leitores deste documento que procurem saber mais sobre este tema através de comunidades abertas ao público em geral, tais como o W3C [2] e o Hyperledger [12]. Em particular, o Hyperledger possuí vários projetos *open-source* em diversas áreas distintas, desde implementação de técnicas criptográficas até ao desenvolvimento de agentes de utilizadores em várias linguagens de programação, e a sua comunidade é extremamente recetiva relativamente a pessoas que pretendam integrar estes projetos e contribuir para o desenvolvimento dos mesmos. Deste modo, sugiro a todos os leitores com interesse em contribuir em projetos *open-source* a explorarem os vários projetos do Hyperleger Indy [56] e a registarem-se no *chat* global [109] desta comunidade.

5.1 PROBLEMAS ENCONTRADOS

Uma das principais dificuldades sentidas na fase inicial de procura de informação relativa ao estado da arte foi que, por este ser um tema que ainda está a ser desenvolvido pelas várias comunidades e a receber contribuições constantes por parte dessas comunidades, as melhores estratégias e práticas estão constantemente a mudar, o que dificultou um pouco todo o processo de aquisição e organização de informação. Outra dificuldade sentida ainda nesta fase de pesquisa deveu-se a este ser um tema que envolve diversas áreas e ser necessário ter conhecimentos sólidos em várias dessas áreas.

Passando para a fase de desenvolvimento, o objetivo inicial seria implementar o agente do utilizador através de uma aplicação *mobile*. Contudo, durante a fase inicial do desenvolvimento da solução percebi que as bibliotecas do Hyperledger Indy não possuíam compatibilidade direta com tecnologias *mobile*. Apesar da comunidade ser muito atenciosa e prestável, não obtive sucesso em nenhuma das soluções propostas pela mesma, o que me levou a envergar por outra estratégia: o desenvolvimento de uma aplicação *web*. Nesta estratégia o agente do utilizador foi portanto dividido em duas componentes, um servidor em **NodeJS** (núcleo), que contém toda a lógica do agente bem como um ponto de serviço através do qual recebe mensagens de outros agentes, e um servidor **React**, que contém toda a interface gráfica do mesmo. A principal razão de se ter optado por esta estratégia deve-se à possibilidade de utilizar o servidor em NodeJS tanto em agentes de ponto como em agentes de *cloud*.

Após repensar a estratégia para implementar o agente do utilizador e começar a desenvolver esse agente surgiu mais um problema de integração, desta vez referente à *blockchain* desenvolvida pelo Hyperledger. De maneira a disponibilizar o projeto **Indy Node** para que outros desenvolvedores o pudessem integrar facilmente nas suas soluções, o Hyperledger

optou por desenvolver um pacote **debian** responsável pela criação dos vários nós de uma *blockchain* através da execução de *scripts*. Contudo, este pacote apresentava várias dependências, sendo que algumas delas não eram compatíveis com certas distribuições de linux ou eram extremamente difíceis de integrar. Em particular, a solução incluída nesta dissertação estava a ser desenvolvida na distribuição *Ubuntu 18.04*, o que me levou a enfrentar vários destes problemas de dependências. Após comunicar os problemas à comunidade, eles confirmaram que era expectável encontrar estes problemas com as dependências e aconselharam-me a utilizar um ficheiro **Docker**, desenvolvido por eles, que se baseava na distribuição *Ubuntu 16.04* e já integrava todas as dependências necessárias. Além de resolver os problemas de dependências encontrados, o uso do ficheiro de **Docker** permitiu ainda individualizar cada um dos quatro nós no seu respetivo *container*, o que permitiu aproximar a solução desenvolvida de um cenário real no qual os nós geralmente são executados por máquinas diferentes.

Mais tarde, ainda na fase de desenvolvimento, deparei-me com um problema no estabelecimento de uma comunicação segura entre dois agentes. Após alguma pesquisa encontrei um conjunto de protocolos elaborados pela comunidade do Hyperledger que indicavam os vários passos que deveriam ser seguidos em cada tipo de comunicação de modo a permitir a interoperabilidade do agente que eu estava a desenvolver com outros agentes, desde estabelecer a conexão inicial, até emitir credenciais ou apresentar provas. O resultado final da implementação destes protocolos está detalhadamente explicado na secção 3.3.2 deste documento.

5.2 TRABALHO FUTURO

Apesar de o agente do utilizador apresentar uma interface intuitiva e com todas as funcionalidades importantes, estas funcionalidade são baseadas em preenchimentos de formulários, nomeadamente para oferecer credenciais verificáveis ou criar apresentações verificáveis. Consequentemente, o esforço que um utilizador precisa de despender ainda não se encontra totalmente otimizado. Como o esforço que um utilizador precisa de despender para executar uma ação é um aspeto muito importante para que a solução desenvolvida seja bem recebida pelos vários utilizadores, é necessário garantir que através desta aplicação o utilizador consegue ser mais eficiente que na utilização de métodos tradicionais. Assim sendo, como trabalho futuro é necessário efetuar testes de usabilidade da aplicação para perceber a opinião dos utilizadores e eventualmente adicionar, remover ou melhorar certas funcionalidades.

Outro ponto importante prende-se na impossibilidade de implementar certas funcionalidades menos importantes devido à restrição de tempo imposta. Apesar de terem sido implementadas todas as funcionalidades principais e várias funcionalidades adicionais relativamente ao agente do utilizador, não foi possível implementar algumas funcionalidades

interessantes que trariam valor adicional devido à falta de tempo. Entre estas funcionalidades destaca-se a implementação de mediadores e *relays*, ou seja, agentes cuja função é atuar como pontos intermediários numa comunicação entre dois agentes de utilizadores. Esta funcionalidade é particularmente importante quando os agentes dos utilizadores estão a ser executados localmente e não possuem uma ligação constante à rede, sendo então responsabilidade dos agentes mediadores armazenar as mensagens até que o agente do utilizador destinatário se volte a conectar à rede. Outra funcionalidade que poderia trazer valor adicional à solução desenvolvida é a possibilidade do utilizador armazenar a sua *wallet* em múltiplos locais, diminuindo assim o risco de perder todos seus os dados, ou até mesmo permitir que o utilizador possa utilizar várias *wallets* de serviços diferentes.

Apesar da solução desenvolvida ter como objetivo efetuar uma prova de conceito, seria também interessante permitir que o agente do utilizador pudesse consultar informação de múltiplos registos descentralizados, desenvolvidos por outras entidades, que atuam como fontes de verdade (i.e. outras *blockchains*). Deste modo, se uma entidade possuir um identificador descentralizado externo à solução desenvolvida com um conjunto de atributos associados, o agente do utilizador implementado seria capaz de efetuar pedidos de leitura e de escrita ao registo no qual o documento desse DID está armazenado e permitir assim o uso desses atributos no sistema desenvolvido nesta dissertação, não sendo necessário replicar todos os processos já efetuados no outro sistema desde a criação de esquemas até à emissão das credenciais. Para implementar esta funcionalidade bastaria apenas conectar o agente do utilizador a um *resolver* universal responsável por efetuar a conexão a outros registos descentralizados que desenvolveram *drivers* através das quais o agente se pode conectar, tal como foi abordado na subsecção 2.2.2.

Por fim, um dos fatores mais importantes na mudança para um sistema baseado em SSI prende-se na possibilidade de integração desse sistema com os esquemas de identidades centralizados utilizados atualmente (i.e. documento de identificação única do cidadão). Tal como foi discutido na secção 2.7, é possível utilizar certificados qualificados (i.e. emitidos pelo eIDAS) dentro de um sistema SSI sempre que for necessário estabelecer uma ligação forte entre um DID e uma pessoa real, caso este que não é diretamente coberto pelo SSI. Deste modo, esquemas de credenciais existentes podem coexistir com os novos sistemas de gestão de identidades descentralizados, tendo como objetivo complementar a fraca ligação entre DIDs e pessoas do mundo real. Sendo esta integração possível e até desejável em alguns casos de uso, é uma mais valia implementar um mecanismo que permita utilizar certificados qualificados dentro do sistema desenvolvido.

BIBLIOGRAFIA

[1] *Internet Identity Workshop*. URL: https://internetidentityworkshop.com/.

[2] URL: https://www.w3.org/.

[3] FXB Center for Health and Human Rights - Harvard University. *A Brief History of National ID Cards*. 2015. URL: https://fxb.harvard.edu/2015/11/12/a-brief-history-of-national-id-cards/.

[4] C. Allen. *The Path to Self-Sovereign Identity*. 2016. URL: https://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html.

[5] D. Yaga, P. Mell, N. Roby e K. Scarfone. *Blockchain Technology Overview*. National Institute of Standards e Technology, 2018.

[6] E. INC. *Self-Sovereign Identity: The Problem*. 2019. URL: https://www.evernym.com/problem/.

[7] J. Gee e M. Button. Em: *The Financial Cost of Fraud 2019*. 2019, p. 6.

[8] Microsoft. *Decentralized Identity - Own and control your identity*. 2018. URL: https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RE2DjfY.

[9] F. Wang e P. D. Filippi. *Self-Sovereign Identity in a Globalized World: Credentials-Based Identity Systems as a Driver for Economic Inclusion*. 2020. URL: https://www.frontiersin.org/articles/10.3389/fbloc.2019.00028/full.

[10] F. Wang e P. D. Filippi. *Self-Sovereign Identity in a Globalized World: Credentials-Based Identity Systems as a Driver for Economic Inclusion*. URL: https://www.frontiersin.org/articles/10.3389/fbloc.2019.00028/full.

[11] URL: https://gdpr.eu/.

[12] *Hyperledger*. URL: https://www.hyperledger.org/.

[13] C. Ellison. *RFC 2692: SPKI Requirements*. 1999. URL: https://tools.ietf.org/html/rfc2692.

[14] C. Ellison. *RFC 2693: SPKI Certificate Theory*. 1999. URL: https://tools.ietf.org/html/rfc2693.

[15] Microsoft. *Microsoft Passport: Streamlining Commerce and Communication on the Web*. 1999. URL: https://news.microsoft.com/1999/10/11/microsoft-passport-streamlining-commerce-and-communication-on-the-web/.

[16] C. Metz. *Microsoft and the Liberty Alliance*. 2001. URL: https://www.eweek.com/security/microsoft-and-the-liberty-alliance.

[17] K. Jordan, J. Hauser e S. Foster. *The Augmented Social Network: Building Identity and Trust into the Next-Generation Internet*. 2003. URL: http://asn.planetwork.net/asn-archive/AugmentedSocialNetwork.pdf.

[18] *Identity Commons*. URL: https://www.idcommons.org/.

[19] OpenID. *What is OpenID?* URL: https://openid.net/what-is-openid/.

[20] OpenID. *OpenID Authentication 2.0 - Final*. URL: https://openid.net/specs/openid-authentication-2_0.html.

[21] E. Hammer-Lahav. *The OAuth 1.0 Protocol*. 2010. URL: https://tools.ietf.org/html/rfc5849.

[22] OpenID. *FIDO UAF Protocol Specification v1.0*. 2014. URL: https://fidoalliance.org/specs/uaf-v1.0-id-20141122/fido-uaf-protocol-v1.0-id-20141122.html.

[23] OpenID. *Welcome to OpenID Connect*. URL: https://openid.net/connect/.

[24] B. Stone. "New Tool From Facebook Extends Its Web Presence". Em: (2008). URL: https://www.nytimes.com/2008/07/24/technology/24facebook.html.

[25] A. Mühle, A. Grüner, T. Gayvoronskaya e C. Meinel. Em: (2018), pp. 1–6.

[26] I. de Engenheiros Eletricistas e Eletrônicos IEEE. URL: https://www.ieee.org/.

[27] D. Reed, M. Sporny, D. Longley, C. Allen, R. Grant e M. Sabadello. *Decentralized Identifiers (DIDs)*. 2019. URL: https://w3c.github.io/did-core/.

[28] A. Cavoukian. "Privacy by Design: The 7 Foundational Principles". Em: (2010). URL: https://iapp.org/media/pdf/resource_center/pbd_implement_7found_principles.pdf.

[29] SelfKey. *Why Decentralized Identifiers Are Changing The Future of the Internet, Identity and Finance*. 2019. URL: https://selfkey.org/decentralized-identifiers-article/.

[30] C. C. Group. *Linked Data Cryptographic Suite Registry*. 2019. URL: https://w3c-ccg.github.io/ld-cryptosuite-registry/.

[31] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, P.-A. Champin e N. Lindström. *JSON-LD 1.1 - A JSON-based Serialization for Linked Data*. 2020. URL: https://www.w3.org/TR/json-ld/.

[32] W3C. *DID Specification Registries*. 2020. URL: https://w3c.github.io/did-spec-registries/.

[33] O. Steele. *On JSON-LD and the Semantics of Identity*. 2020. URL: https://medium.com/transmute-techtalk/on-json-ld-and-the-semantics-of-identity-42d051d3ce14.

[34] M. Sporny, D. Longley e D. Chadwick. *Verifiable Credentials Data Model 1.0: Expressing verifiable information on the Web*. 2019. URL: https://www.w3.org/TR/vc-data-model/.

[35] D. Gisolfi. *Self-sovereign identity: Unraveling the terminology*. 2018. URL: https://www.ibm.com/blogs/blockchain/2018/06/self-sovereign-identity-unraveling-the-terminology/.

[36] D. Lavrenov. *A Zero-Knowledge Proof: Improving Privacy on a Blockchain*. 2019. URL: https://www.altoros.com/blog/zero-knowledge-proof-improving-privacy-for-a-blockchain/.

[37] G. Simari. *A Primer on Zero Knowledge Protocols*. 2002. URL: http://www.cs.ox.ac.uk/people/gerardo.simari/personal/publications/zkp-simari2002.pdf.

[38] A. S. Delight. *Zero Knowledge Proof of Age Using Hash Chains*. 2017. URL: https://www.stratumn.com/thinking/zero-knowledge-proof-of-age-using-hash-chains/.

[39] D. Hardman. *No Paradox Here: ZKPs Deliver Savvy Trust*. 2020. URL: https://www.evernym.com/blog/no-paradox-here-zkps-deliver-savvy-trust/.

[40] J. Camenisch, M. Kohlweiss e C. Soriente. *An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials*. 2018. URL: https://eprint.iacr.org/2008/539.pdf.

[41] D. Chadwick. *Verifiable Credentials Lifecycle 1.0*. 2019. URL: https://w3c-ccg.github.io/vc-lifecycle/#identify-information-lifecycle.

[42] N. Otto, S. Lee, B. Sletten, D. Burnett, M. Sporny e K. Ebert. *Verifiable Credentials Use Cases*. 2019. URL: https://www.w3.org/TR/vc-use-cases/.

[43] M. Lodder, S. M. Chase e W. McNally. *Use Cases and Proposed Solutions for Verifable Ofine Credentials*. 2019. URL: https://nbviewer.jupyter.org/github/WebOfTrustInfo/rwot7-toronto/blob/master/final-documents/offline-use-cases.pdf.

[44] Bitcoin. 2020. URL: https://bitcoin.org/en/.

[45] Ethereum. 2020. URL: https://ethereum.org/en/.

[46] N. Scott. *A Decentralized Blockchain Ledger for the Management of Medication Histories - Scientific Figure on ResearchGate*. 2020. URL: https://www.researchgate.net/figure/Network-structures-of-centralized-system-left-and-decentralized-system-right_fig1_330139613.

[47] J. Dossman. 2020. URL: https://cruzlaw.gi/what-is-blockchain/.

[48] N. Szabo. *Smart Contracts*. 1994. URL: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html.

[49] CoinDesk. *Ethereum 101*. 2020. URL: https://www.coindesk.com/learn/ethereum-101/ethereum-smart-contracts-work.

[50] D. Gisolfi. *Self-sovereign identity: Why blockchain?* 2018. URL: https://www.ibm.com/blogs/blockchain/2018/06/self-sovereign-identity-why-blockchain/.

[51] N. W. Group. *Guidelines for Writing RFC Text on Security Considerations*. 2003. URL: https://tools.ietf.org/html/rfc3552.

[52] eIDAS. URL: https://www.eid.as/.

[53] eGovernment e T. U. H.4). *Trust Services and Electronic identification (eID)*. URL: https://ec.europa.eu/digital-single-market/en/policies/trust-services-and-eidentification.

[54] eIDAS. *EIDAS SUPPORTED SELF-SOVEREIGN IDENTITY*. URL: https://ec.europa.eu/futurium/en/system/files/ged/eidas_supported_ssi_may_2019_0.pdf.

[55] Hyperledger. *Hyperledger Indy SDK*. 2018. URL: https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/index.html.

[56] *Hyperledger Indy*. URL: https://www.hyperledger.org/use/hyperledger-indy.

[57] *Hyperledger*. URL: https://indy.readthedocs.io/en/latest/.

[58] Hyperledger. *Welcome to Hyperledger Indy Node's documentation*. 2018. URL: https://hyperledger-indy.readthedocs.io/projects/node/en/latest/index.html.

[59] Hyperledger. *Welcome to Indy Plenum's documentation!* 2018. URL: https://hyperledger-indy.readthedocs.io/projects/plenum/en/latest/index.html.

[60] Hyperledger. *Create a Network and Start Nodes*. URL: https://hyperledger-indy.readthedocs.io/projects/node/en/latest/start-nodes.html.

[61] Hyperledger. URL: https://github.com/hyperledger/indy-sdk/blob/master/ci/indy-pool.dockerfile.

[62] V. O. Network. *VON Network*. URL: https://github.com/bcgov/von-network.

[63] *Verifiable Organizations Network: Global digital trust for organizations*. URL: https://vonx.io/.

[64] *Sovrin*. URL: https://www.linuxfoundation.org/.

[65] *Sovrin*. URL: https://sovrin.org.

[66] Hyperledger. *Hyperledger Indy*. 2019. URL: https://hyperledger-indy.readthedocs.io/en/latest/index.html.

[67] H. Anwar. *Do You Really Need Hyperledger Indy?* 2020. URL: https://101blockchains.com/hyperledger-indy/.

[68] Hyperledger. *Hyperledger Aries*. 2020. URL: https://github.com/hyperledger/aries.

[69] Hyperledger. *Hyperledger Ursa*. 2020. URL: https://www.hyperledger.org/use/ursa.

[70] K. Tam. *Exploring Agent Demonstration in Hyperledger Indy*. 2019. URL: https://medium.com/@kctheservant/exploring-agent-demonstration-in-hyperledger-indy-3e1e8278b3d1.

[71] Hyperledger. URL: https://www.hyperledger.org/projects/aries.

[72] Hyperledger. 2019. URL: https://github.com/hyperledger-archives/education/blob/master/LFS171x/docs/introduction-to-hyperledger-indy.md.

[73] D. Hardman. *Aries RFC 0004: Agents*. 2020. URL: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0004-agents/README.md.

[74] *JSON Web Tokens*. URL: https://jwt.io/.

[75] KirstenS. *Cross Site Scripting (XSS)*. URL: https://owasp.org/www-community/attacks/xss/.

[76] KirstenS. *Cross Site Request Forgery (CSRF)*. URL: https://owasp.org/www-community/attacks/csrf.

[77] D. Abramov. *Redux*. URL: https://redux.js.org/.

[78] S. Das. *Authentication Using JWT and Refresh Token — Part 1*. URL: https://medium.com/swlh/authentication-using-jwt-and-refresh-token-part-1-aca5522c14c8.

[79] J. Camenisch, M. Kohlweiss e C. Soriente. "An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials." Em: *Public Key Cryptography*. Ed. por S. Jarecki e G. Tsudik. Vol. 5443. Lecture Notes in Computer Science. Springer, 2009, pp. 481–500. ISBN: 978-3-642-00467-4. URL: http://dblp.uni-trier.de/db/conf/pkc/pkc2009.html#CamenischKS09.

[80] Hyperledger. *How Credential Revocation Works*. 2018. URL: https://github.com/fabienpe/indy-sdk/blob/master/docs/concepts/revocation/cred-revocation.md.

[81] Hyperledger. *Aries RFC 0005: DID Communication*. 2020. URL: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0005-didcomm/README.md.

[82] A. Preukschat. *Peer DIDs: a secure and scalable method for DIDs that's entirely off-ledger – Daniel Hardman – Webinar 42*. 2019. URL: https://ssimeetup.org/peer-dids-secure-scalable-method-dids-off-ledger-daniel-hardman-webinar-42/.

[83] W3C. *Peer DID Method Specification*. 2020. URL: https://identity.foundation/peer-did-method-spec/.

[84] Hyperledger. *Hyperledger Aries*. 2020. URL: https://www.hyperledger.org/use/aries.

[85] Hyperledger. *Aries RFC 0003: Protocols*. 2020. URL: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0003-protocols/README.md.

[86] Hyperledger. *Message ID and Threading*. URL: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0008-message-id-and-threading/README.md.

[87] Hyperledger. *Decorators*. URL: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0011-decorators/README.md.

[88] Hyperledger. *Attachments*. URL: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0017-attachments/README.md.

[89] Hyperledger. *Message Types*. URL: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0020-message-types/README.md.

[90] Hyperledger. *ACKs*. URL: https://github.com/hyperledger/aries-rfcs/blob/master/features/0015-acks/README.md.

[91] Hyperledger. *Report Problem Protocol 1.0*. URL: https://github.com/hyperledger/aries-rfcs/blob/master/features/0035-report-problem/README.md.

[92] R. West, D. Bluhm, M. Hailstone, S. Curran e S. Curren. *0160: Connection Protocol*. 2020. URL: https://github.com/hyperledger/aries-rfcs/blob/master/features/0160-connection-protocol/README.md.

[93] N. Khateev. *Aries RFC 0036: Issue Credential Protocol 1.0*. 2020. URL: https://github.com/hyperledger/aries-rfcs/blob/master/features/0036-issue-credential/README.md.

[94] W. Abramson. *CL Signatures for Anonymous Credentials*. URL: https://misterwip.uk/cl-signatures.

[95] N. Khateev. *Aries RFC 0037: Present Proof Protocol 1.0*. 2020. URL: https://github.com/hyperledger/aries-rfcs/blob/master/features/0037-present-proof/README.md.

[96] M. Jones e J. Hildebrand. *JSON Web Encryption (JWE)*. 2015. URL: https://tools.ietf.org/html/rfc7516.

[97] D. Hardman. *Aries RFC 0046: Mediators and Relays*. 2020. URL: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0046-mediators-and-relays/README.md.

[98] D. J. Bernstein. *ChaCha, a variant of Salsa20*. 2008. URL: http://cr.yp.to/chacha/chacha-20080128.pdf.

[99] D. J. Bernstein. *The Poly1305-AES message-authentication code*. 2005. URL: http://cr.yp.to/mac/poly1305-20050329.pdf.

[100] Y. Nir e A. Langley. *ChaCha20 and Poly1305 for IETF Protocols*. 2015. URL: https://tools.ietf.org/html/rfc7539.

[101] A. Langley, W. Chang, N. Mavrogiannopoulos, J. Strombergson e S. Josefsson. *ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS)*. 2016. URL: https://tools.ietf.org/html/rfc7905.

[102] I. KDDI Research. *Security Analysis of ChaCha20-Poly1305 AEAD*. 2017. URL: https://www.cryptrec.go.jp/exreport/cryptrec-ex-2601-2016.pdf.

[103] D. Hardman, D. Kulic, V. Gudkov e M. Lodder. *Aries RFC 0050: Wallets*. 2019. URL: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0050-wallets/README.md.

[104] P.-L. Aublin, S. B. Mokhtar e V. Quema. "RBFT: Redundant Byzantine Fault Tolerance". Em: *2013 IEEE 33rd International Conference on Distributed Computing Systems*. 2013, pp. 297–306. DOI: 10.1109/ICDCS.2013.53.

[105] Hyperledger. *Indy Plenum Documentation*. 2019. URL: https://readthedocs.org/projects/indy-plenum/downloads/pdf/latest/.

[106] G. Konstantopoulos. *Understanding Blockchain Fundamentals, Part 1: Byzantine Fault Tolerance*. 2017. URL: https://medium.com/loom-network/understanding-blockchain-fundamentals-part-1-byzantine-fault-tolerance-245f46fe8419.

[107] Hyperledger. *Hyperledger Indy Node Documentation*. 2019. URL: https://readthedocs.org/projects/indy-node/downloads/pdf/latest/.

[108] Hyperledger. *Default AUTH_MAP Rules*. 2018. URL: https://hyperledger-indy.readthedocs.io/projects/node/en/latest/auth_rules.html.

[109] URL: https://chat.hyperledger.org/home.

[110] *NodeJs*. URL: https://nodejs.org/en/about/.

[111] *Get Started with Docker*. URL: https://www.docker.com/.

[112] *Git*. URL: https://git-scm.com/.

A

MATERIAL DE SUPORTE

A.1 EXEMPLO DE UM DOCUMENTO DE UM DID

Na secção 2.2.3 foram explicados alguns dos atributos mais usuais em documentos de DIDs. As figuras 61 e 62 apresentam a junção de todos esses atributos num único documento.

```
{
  "@context": "https://w3id.org/future-method/v1",
  "id": "did:example:123456789abcdefghi",

  "publicKey": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }, {
    "id": "did:example:123456789abcdefghi#keys-3",
    "type": "Ieee2410VerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }],

  "authentication": [
    // este mecanismo pode ser usado para autenticar como did:...fghi
    "did:example:123456789abcdefghi#keys-1",
    // este mecanismo pode ser usado para autenticar biometricamente como did:...fghi
    "did:example:123456789abcdefghi#keys-3",
    // este mecanismo apenas é autorizado para çâautenticao, ãno pode ser
    // utilizado para outro ópropsito de prova, por isso a sua çãdescrio
    // completa encontra-se embutida aqui em vez de usar apenas uma êreferncia
```

Figura 61: Exemplo de um documento de um DID (parte 1) [27]

```
  (çãcontinuao)

    {
      "id": "did:example:123456789abcdefghi#keys-2",
      "type": "Ed25519VerificationKey2018",
      "controller": "did:example:123456789abcdefghi",
      "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
    }
  ],
  "service": [{
    "id": "did:example:123456789abcdefghi#oidc",
    "type": "OpenIdConnectVersion1.0Service",
    "serviceEndpoint": "https://openid.example.com/"
  }, {
    "id": "did:example:123456789abcdefghi#vcStore",
    "type": "CredentialRepositoryService",
    "serviceEndpoint": "https://repository.example.com/service/8377464"
  }, {
    "id": "did:example:123456789abcdefghi#hub",
    "type": "HubService",
    "serviceEndpoint": "https://hub.example.com/.identity/did:example:0123456789abcdef/"
  }, {
    "id": "did:example:123456789abcdefghi#messaging",
    "type": "MessagingService",
    "serviceEndpoint": "https://example.com/messages/8377464"
  }, {
    "type": "SocialWebInboxService",
    "id": "did:example:123456789abcdefghi#inbox",
    "serviceEndpoint": "https://social.example.com/83hfh37dj",
    "description": "My public social inbox",
    "spamCost": {
      "amount": "0.50",
      "currency": "USD"
    }
  }, {
    "type": "DidAuthPushModeVersion1",
    "id": "did:example:123456789abcdefghi#push",
    "serviceEndpoint": "http://auth.example.com/did:example:123456789abcdefghi"
  }]
}
```

Figura 62: Exemplo de um documento de um DID (parte 2) [27]

A.2 EXEMPLO CONCRETO DO CICLO DE VIDA DE UMA CREDENCIAL

No exemplo que se segue, Pat recebe uma *verifiable credential* da universidade e armazena-a numa *wallet* digital. Esta credencial tem como objetivo usufruir de um desconto para ex-alunos da universidade. A figura 63 representa esta credencial.

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1"
  ],
  // especifica o identificador da credencial
  "id": "http://example.edu/credentials/1872",
  // os tipos da credencial declaram que dados esperar na mesma
  "type": ["VerifiableCredential", "AlumniCredential"],
  // a entidade de emitiu a credencial
  "issuer": "https://example.edu/issuers/565049",
  // quando a credencial foi emitida
  "issuanceDate": "2010-01-01T19:73:24Z",
  // claims sobre os sujeitos da credencial
  "credentialSubject": {
    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    "alumniOf": {
      "id": "did:example:c276e12ec21ebfeb1f712ebc6f1",
      "name": [
        { "value": "Example University", "lang": "en" },
        { "value": "Exemplo de Universidade", "lang": "pt" }
      ]
    }
  },
  // Prova digital que torna a credencial áinviolvel
  "proof": {
    // A étcnica de assinaturas ácriptogrficas que foi usada para gerar a assinatura
    "type": "RsaSignature2018",
    // A data em que a assinatura foi criada
    "created": "2017-06-18T21:19:10Z",
    // Objetivo desta prova
    "proofPurpose": "assertionMethod",
    // O identificador da chave úpblica que pode verificar esta assinatura
    "verificationMethod": "https://example.edu/issuers/keys/1",
    // O valor da assinatura digital
    "jws": "eyJhbGciOiJSUzI1NiIsImI2 ... LtjPAYuNzVBAh4vGHSrQyHUdBBPM"
  }
}
```

Figura 63: Exemplo de uma *verifiable credential* [34]

O *verifier* declara que qualquer ex-aluno da "Example University" recebe um desconto nos bilhetes para eventos desportivos. Consequentemente, após obter a credencial, o Pat usa-a para usufruir do desconto de ex-aluno, iniciando o processo através do seu dispositivo móvel. Durante este processo é-lhe requerida uma prova de que é, de facto, ex-aluno da Example University, pelo que ele recorre à credencial que recebeu para gerar uma apresentação verificável e envia-a ao *verifier*. As figuras 64 e 65 mostram a estrutura de uma apresentação verificável.

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1"
  ],
  "type": "VerifiablePresentation",
  // Lista de credenciais com a credential emitida no exemplo anterior
  "verifiableCredential": [{
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://www.w3.org/2018/credentials/examples/v1"
    ],
    "id": "http://example.edu/credentials/1872",
    "type": ["VerifiableCredential", "AlumniCredential"],
    "issuer": "https://example.edu/issuers/565049",
    "issuanceDate": "2010-01-01T19:73:24Z",
    "credentialSubject": {
      "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
      "alumniOf": {
        "id": "did:example:c276e12ec21ebfeb1f712ebc6f1",
        "name": [{
          "value": "Example University",
          "lang": "en"
        }, {
          "value": "Exemplo de Universidade",
          "lang": "pt"
        }]
      }
    },
```

Figura 64: Exemplo de uma *verifiable presentation* (parte 1) [34]

```
  (çãcontinuao)

    "proof": {
      "type": "RsaSignature2018",
      "created": "2017-06-18T21:19:10Z",
      "proofPurpose": "assertionMethod",
      "verificationMethod": "https://example.edu/issuers/keys/1",
      "jws": "eyJhbGciOiJSUzI1NiIsImI2 ... LtjPAYuNzVBAh4vGHSrQyHUdBBPM"
    }
  }],
  // Assinatura digital do Pat na çãapresentao
  "proof": {
    "type": "RsaSignature2018",
    "created": "2018-09-14T21:19:10Z",
    "proofPurpose": "authentication",
    "verificationMethod": "did:example:ebfeb1f712ebc6f1c276e12ec21#keys-1",
    // "challenge" e "domain" protegem contra ataques de çãrepetio
    "challenge": "1f44d55f-f161-4938-a659-f8026467f126",
    "domain": "4jt78h47fh47",
    "jws": "eyJhbGciOiJSUzI1NiIsImI2NC...NzVBAh4vGHSrQyHUGlcTwLtjPAnKb78"
  }
}
```

Figura 65: Exemplo de uma *verifiable presentation* (parte 2) [34]

A.3 TRANSAÇÕES DA BLOCKCHAIN IMPLEMENTADA

Esta secção apresenta a estrutura de algumas das transações mais comuns de um ecossistema SSI. A subsecção A.3.1 aborda as estruturas de algumas operações de leitura enquanto que a subseção A.3.2 explora as estruturas de algumas operações de escrita.

A.3.1 *Operações de leitura na blockchain*

Consulta de um Nym

Numa *blockchain* Indy a palavra "Nym" refere-se aos utilizadores da *blockchain* cujos dados estão armazenados na mesma. Consequentemente, a consulta de um Nym é uma operação de leitura que obtém todos os dados públicos associados a um DID específico (Nym). A figura 66 mostra um exemplo de uma operação de consulta de um Nym. O atributo "dest" identifica o DID que se pretende consultar, que corresponde ao valor "2VkbBskPNNyWrLrZq7DBhk".

```
{
  'operation': {
    'type': '105'
    'dest': '2VkbBskPNNyWrLrZq7DBhk'
  },
  'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
  'reqId': 1514308188474704,
  'protocolVersion': 2
}
```

Figura 66: Exemplo de uma operação de consulta de um Nym (Retirado de [107])

Consulta de um esquema

Entre as várias operações de consulta efetuadas a uma *blockchain*, consulta de esquemas é uma das mais comuns. A figura 67 demonstra um exemplo de uma operação de consulta do esquema "Degree", mais concretamente da versão 1.0 desse esquema. O atributo "dest" identifica o DID do *issuer* que emitiu o esquema.

```
{
  'operation': {
    'type': '107'
    'dest': '2VkbBskPNNyWrLrZq7DBhk',
    'data': {
      'name': 'Degree',
      'version': '1.0'
    },
  },
  'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
  'reqId': 1514308188474704,
  'protocolVersion': 2
}
```

Figura 67: Exemplo de uma operação de consulta de um Nym (Retirado de [107])

A.3.2 *Operações de escrita na blockchain*

Adição de um esquema

Operações de adição de esquemas são operações muito importantes num sistema de gestão de identidades uma vez que permitem que estes esquemas sejam inseridos na *blockchain* de modo a assegurar a sua integridade. A figura 68 mostra um exemplo de uma operação de adição de um esquema. Na operação apresentada é possível observar que se pretende adicionar a versão 1.0 do esquema "Degree" com um conjunto de atributos indicados no campo "attr_names".

```
{
  'operation': {
    'type': '101',
    'data': {
      'version': '1.0',
      'name': 'Degree',
      'attr_names': ['undergrad', 'last_name', 'first_name', 'birth_date', 'postgrad',
          'expiry_date']
    },
  },
  'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
  'endorser': 'D6HG5g65TDQr1PPHHRoiGf',
  'reqId': 1514280215504647,
  'protocolVersion': 2,
  'signature': '5ZTp9g4SP6t73rH2s8zgmtqdX ... PXfJue3vJBWjy89bSHvyMSdS'
}
```

Figura 68: Exemplo de uma operação de adição de um esquema (Retirado de [107])

Adição de um nó

Entre as várias operações de escrita existentes encontram-se algumas afetas à gestão da *blockchain* e dos nós que a compõem, entre as quais a operação de adição de um nó. A figura 69 apresenta uma operação de adição de um nó a uma *blockchain*. Neste exemplo podemos observar o atributo "data" contém todas as informações importantes relativas aos nó que se pretende adicionar, nomeadamente o ip, *node_ip*, e a porta, *node_port*.

```
{
  'operation': {
    'type': '0'
    'data': {
      'alias': 'Node1',
      'client_ip': '127.0.0.1',
      'client_port': 7588,
      'node_ip': '127.0.0.1',
      'node_port': 7587,
      'blskey': '00000000000000000000000000000000',
      'services': ['VALIDATOR']}
    },
    'dest': '6HoV7DUEfNDiUP4ENnSC4yePja8w7JDQJ5uzVgyW4nL8'
  },
  'identifier': '21BPzYYrFzbuECcBV3M1FH',
  'reqId': 1514304094738044,
  'protocolVersion': 2,
  'signature': '3YVzDtSxxnowVwAXZmxCG2f ... 9mCqFLa7aBzt4MKXk4MeunVj',
}
```

Figura 69: Exemplo de uma operação de adição de um nó (Retirado de [107])

A.4 Caso de uso: Eleição

No capítulo 4 foram descritos três casos de uso aos quais é possível aplicar a solução desenvolvida. Esta secção irá apresentar a aplicação prática do sistema desenvolvido ao primeiro caso de uso, isto é, ao caso de uso da eleição. Com o objetivo de demonstrar as várias funcionalidades do sistema, este caso de uso foi alargado para a emissão de uma credencial com três atributos: o nome, a idade e o NIC (número de identificação civil). Por sua vez, o *verifier* apenas necessita do NIC do cidadão e de uma prova de que este é maior de idade. Deste modo, o *holder* terá a possibilidade de não revelar o seu nome (através do uso de divulgação seletiva) e de ocultar a sua idade exata, provando apenas ser maior de idade (através de provas de conhecimento zero). Durante a execução deste caso de uso serão criados esquemas e definições de credenciais, bem como registos de revogação aos quais as credenciais emitidas serão associadas, tal como explicado na secção 3.2.3. No final será ainda demonstrada a revogação, por parte do *issuer*, da credencial utilizada pelo *holder*, o que o irá impedir de continuar a utilizar essa credencial.

A.4.1 *Criação de utilizadores*

Tal como foi referido, este caso de uso envolve três utilizadores:

- *Issuer:* Instituto dos Registos e Notariado (IRN);
- *Holder*: Alice;
- *Verifier*: Comissão Nacional de Eleições (CNE).

Cada uma das entidades indicadas possui um agente de utilizador que lhe permite interagir com o sistema e com outras entidades. Após inicializar a aplicação, as entidades precisam de se registar de modo a proteger os seus dados com um *login*. Na figura 70 é possível observar os utilizadores criados para a execução deste caso de uso.

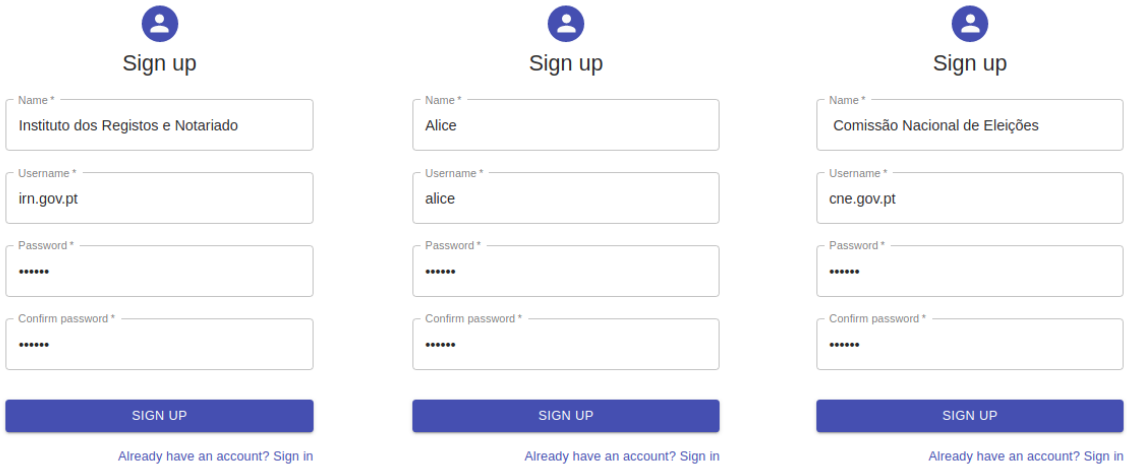


Figura 70: Página de *Login*

É importante indicar que esta aplicação corre localmente no dispositivo de cada utilizador e não existe um repositório centralizado que armazena estas informações.

A.4.2 *Criação de identificadores descentralizados*

Identificadores descentralizados constituem a base de um sistema de gestão de identidades descentralizado sendo necessários para efetuar a grande maioria das interações com o sistema, desde estabelecimento de conexões até à escrita de transações na *blockchain*. Adicionalmente, tal como já foi indicado no corpo do relatório, os atributos são emitidos não sobre o *holder* mas sim sobre um DID controlado por esse *holder*.

No estabelecimento de conexões privadas são utilizados DIDs de emparelhamento, tal como é explicado na secção 3.3. Estes DIDs são automaticamente gerados pelo agente do utilizador durante o estabelecimento da conexão. O *issuer* emite os atributos sobre o DID de emparelhamento que o *holder* lhe forneceu durante o estabelecimento da conexão privada entre ambos. Consequentemente, o *holder* não precisa de possuir DIDs públicos.

Por outro lado, tanto o IRN (*issuer*) como a CNE (*verifier*) precisam de possuir DIDs públicos de modo a associarem a sua identidade digital à respetiva entidade física. Adicionalmente, o IRN precisa ainda de ser capaz de publicar esquemas, definições de credenciais e dados de revogação de credenciais na *blockchain*. Para tal, precisa de um DID público que lhe forneça as permissões necessárias. As permissões de autorização para publicação de transações na *blockchain* estão descritas na secção 3.5.2.

Após estarem autenticadas, as três entidades poderão gerir todos os seus DIDs através da página de gestão de DIDs, apresentada na figura 71.

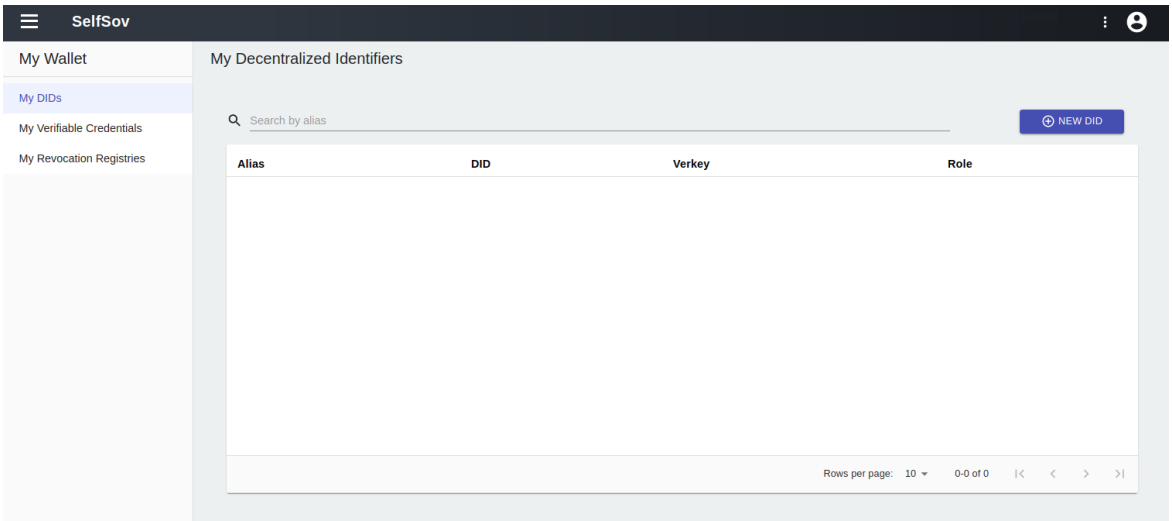


Figura 71: Página de gestão de DIDs

No sistema implementado, o registo de DIDs públicos requer que uma entidade com permissões adequadas, isto é, uma entidade que possua um DID público com as permissões

necessárias, submeta transações de registo desses DIDs à *blockchain*. Uma vez mais, a secção 3.5.2 descreve as permissões necessárias para a escrita de transações na *blockchain*, indicando também que papeis são necessários para criar DIDs com certos papeis. Tendo isso por base, tanto o IRN como a CNE necessitam de registar DIDs públicos na *blockchain*, recorrendo para tal a uma entidade de maior autoridade (i.e. o estado português), que desempenhe o papel de *trustee*, ou seja, o papel com mais permissões dentro da *blockchain*. Por questões de simplicidade, esta entidade de maior autoridade não é considerada no caso de uso apresentado, sendo utilizadas *seeds* que permitem importar DIDs já registados na *blockchain* para a *wallet*, tal como mostram as figuras 72 e 73.

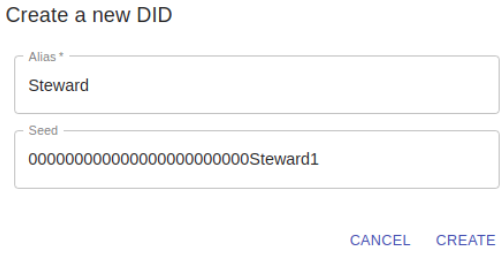


Figura 72: DID público do IRN (*steward*)

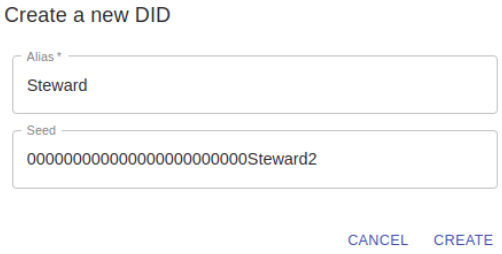


Figura 73: DID público da CNE (*steward*)

As *seeds* utilizadas correspondem a DIDs com o papel de *steward*, ou seja, o segundo papel com mais permissões da *blockchain*. Com este papel, o IRN e a CNE podem criar múltiplos DIDs (com menos privilégios) de modo a construir hierarquias de DIDs. Deste modo, tanto o IRN como a CNE decidem criar um DID adicional para expor os serviços que ambos disponibilizam, ou seja, os novos DIDs serão usados para criar convites públicos que visam o estabelecimento de conexões. As figuras 74 e 75 mostram a criação de dois DIDs que serão posteriormente registados com o papel de *trust anchor*, sendo este o terceiro papel com mais permissões.

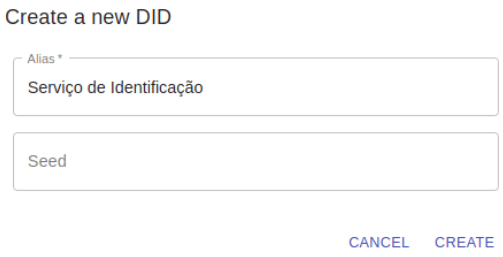


Figura 74: DID público do IRN (*trust anchor / endorser*)

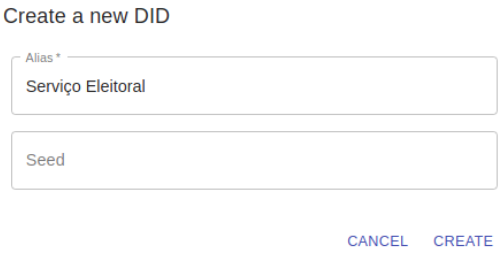


Figura 75: DID público da CNE (*trust anchor / endorser*)

Neste caso não é usada uma *seed* dado que se pretende criar um novo DID. De notar que os DIDs criados ainda não podem ser utilizados como DIDs públicos uma vez que ainda não foram adicionados à *blockchain*. A figura 76 mostra o estado atual do DID criado pelo IRN, que se encontra atualmente no estado "não utilizado", o que significa que ainda não foi registado na *blockchain*. O mesmo acontece com o DID criado pela CNE.

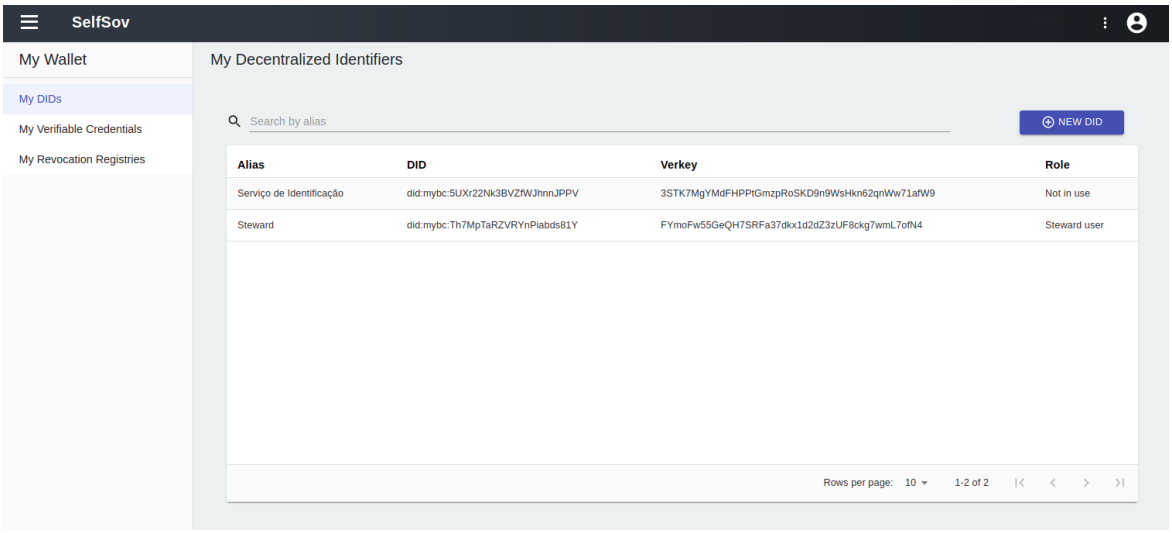


Figura 76: Página de gestão de DIDs

Deste modo, o IRN e a CNE recorrem aos seus DIDs *steward* para registar os DIDs acabados de criar na *blockchain*. DIDs com papel de *steward* permitem que os seus donos sejam capazes de registar DIDs com papel de *trust anchor* ou inferior. A figura 77 mostra a página da aplicação que permite submeter transações para a *blockchain*, em particular transações de criação de DIDs públicos (**Nyms**).

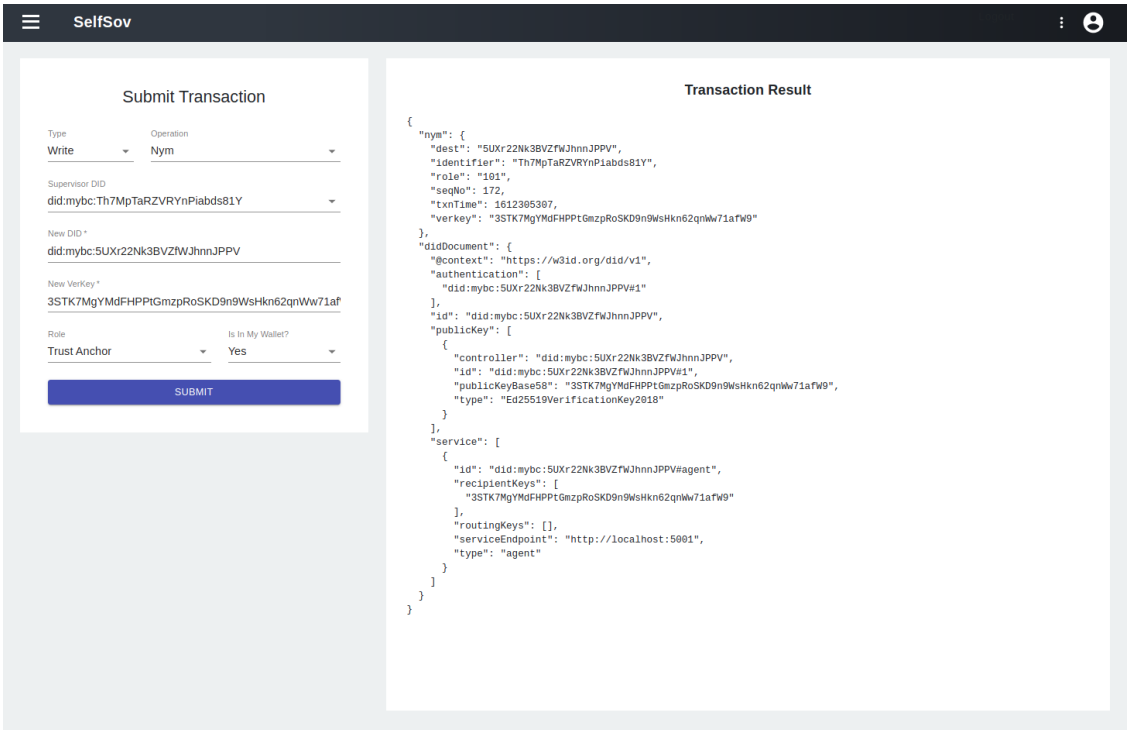


Figura 77: Registo do DID na *blockchain* com o papel de *trust anchor*

Nesta figura podemos verificar que o IRN utiliza o seu DID *steward* para registar o seu outro DID com o papel de *trust anchor* e a resposta enviada pela *blockchain* com os detalhes da transação efetuada.

Voltando para a página de gestão de DIDs, o IRN poderá agora verificar que o seu DID foi registado na *blockchain* e que possui o papel de *trust anchor*, tal como mostra a figura 78.

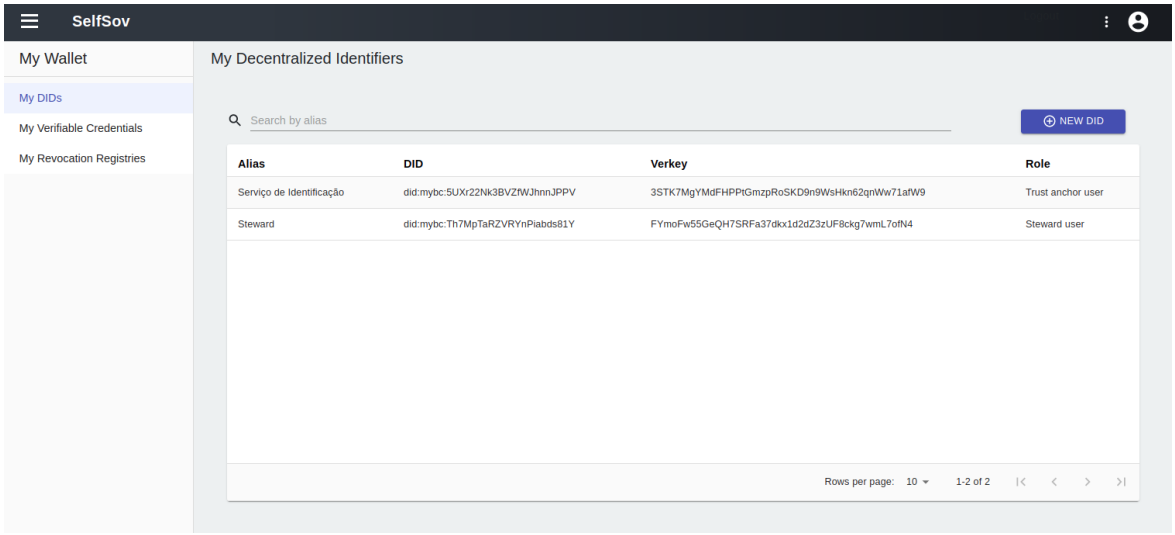


Figura 78: Página de gestão de DIDs: DID do serviço registado

Do mesmo modo, a CNE deverá executar os passos apresentados para registar o seu novo DID através do seu DID *steward*.

Ao longo do caso de uso apresentado, o DID *steward* será utilizado para registo de esquemas, definições de credenciais, entre outras operações, na *blockchain* enquanto que o DID do serviço (*trust anchor*) será utilizado apenas para criar convites de conexão. Contudo, a utilização dos vários DIDs depende da natureza do caso de uso e das permissões que esses DIDs possuem. Por exemplo, apesar de ser utilizado o DID *steward* para adicionar esquemas e definições de credenciais à *blockchain*, DIDs com o papel de *trust anchor* também podem efetuar essas operações.

A.4.3 *Criação de esquemas, definições de credenciais e registos de revogação*

De modo a poder iniciar o processo de emissão de credenciais, o IRN (*issuer*) deverá utilizar um esquema que indique quais atributos as suas credenciais irão conter. Como ainda não existem esquemas na *blockchain*, o IRN começa por criar um esquema de acordo com as suas necessidades, isto é, um esquema que contém os atributos "nome", "idade" e "nic" (número de identificação civil). Ao publicar o esquema na *blockchain*, o IRN decide utilizar o seu DID *steward* para submeter a transação, tal como mostra a figura 79, e chama o esquema de "cartao_de_identificacao".

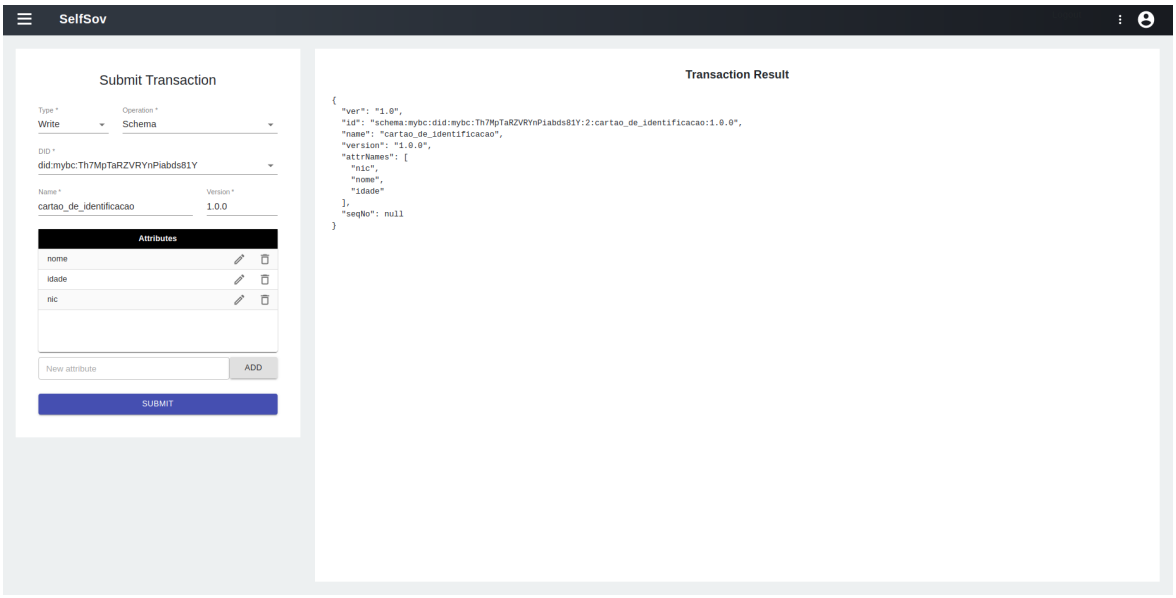


Figura 79: Submissão de um esquema para a *blockchain*

Após a criação do esquema, o IRN procede então para a criação da definição da credencial, que especifica o esquema e as chaves criptográficas que o *issuer* irá utilizar na emissão de credenciais. Na figura 80 é possível verificar que o IRN utiliza o identificador do esquema que acabou de criar e recorre novamente ao DID *steward* para efetuar a transação.

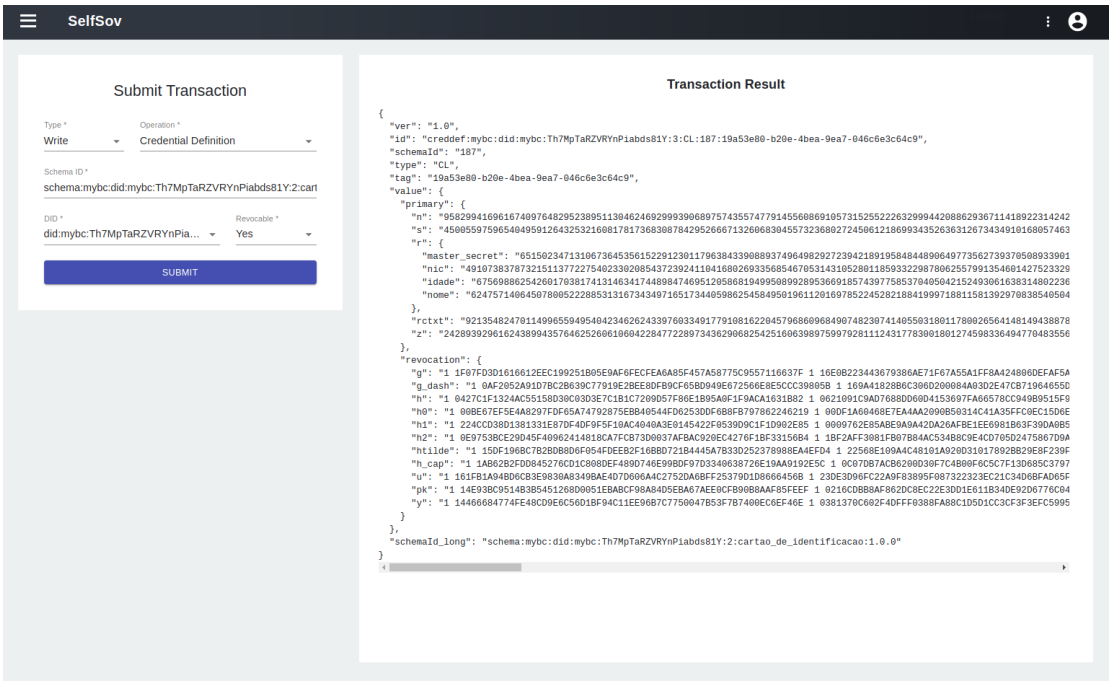


Figura 80: Submissão da definição de uma credencial para a *blockchain*

Este formulário apresenta ainda o campo "Revocable"que irá determinar se as credenciais emitidas através desta definição da credencial serão ou não revogáveis. Caso o *issuer* opte

por permitir a revogação de credenciais, deverá também criar um registo de revogação ao qual irá associar todas as credenciais emitidas para esta definição da credencial. O *issuer* não poderá emitir credenciais enquanto não associar um registo válido à definição da credencial.

Através da figura apresentada é possível observar que o IRN optou por permitir a revogação de credenciais pelo que terá de proceder à criação de um registo de revogação antes de começar a emitir credenciais.

Assim como os DIDs, o IRN pode aceder a todos os seus registos de revogação através da página da *wallet*. A figura 81 mostra a página de gestão de registos de revogação na qual é possível efetuar toda a gestão desses registos.

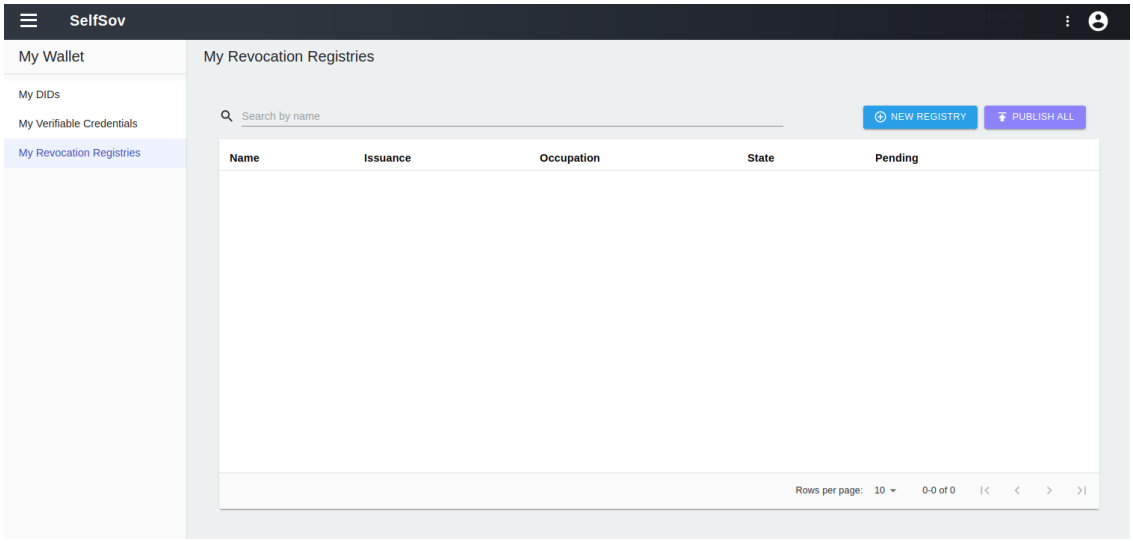


Figura 81: Página de gestão de registos de revogação

Para adicionar um novo registo de revogação, o IRN precisa de indicar o identificador da definição da credencial à qual ele quer associar o registo, qual a capacidade desejada para o registo e qual o modo de emissão que quer utilizar (**emissão por padrão** ou **emissão por pedido**), tal como foi explicado na secção 3.2.3. A figura 82 mostra os dados introduzidos pelo IRN na criação do registo de revogação.

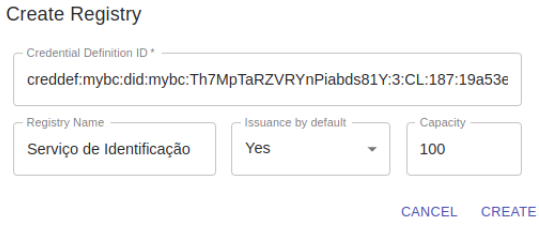


Figura 82: Criação de um registo de revogação

Após criar o registo de revogação, o IRN poderá geri-lo através da página de gestão dos registos de revogação. A figura 83 mostra a tabela dos registos de revogação com o registo acabado de criar.

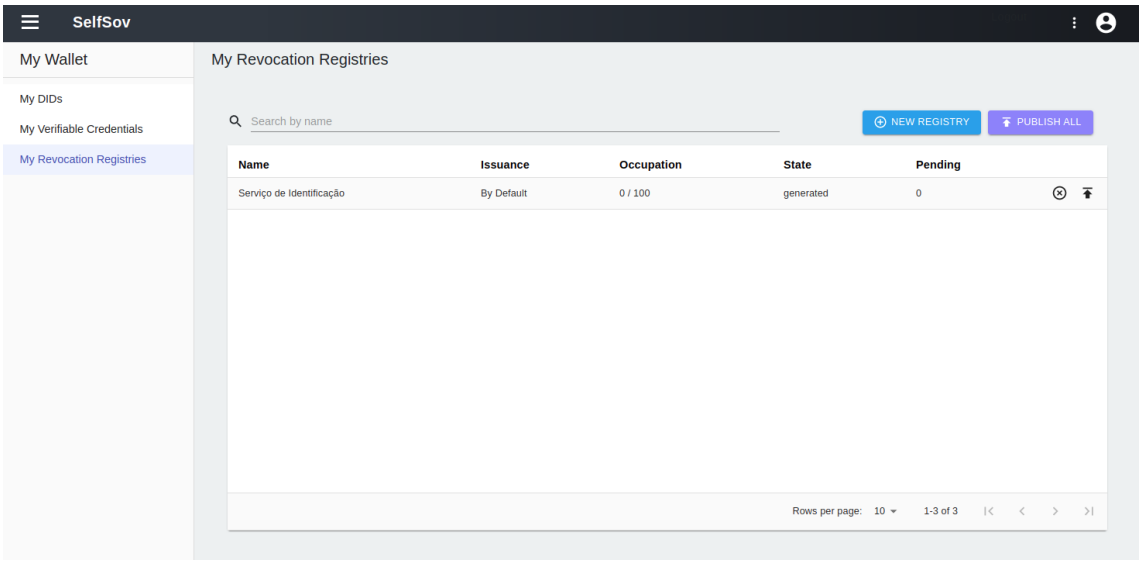


Figura 83: Tabela de registos de revogação com o registo criado

Nesta tabela o IRN consegue revogar todas as credenciais que emitir, podendo optar por apenas revogar localmente ou enviar imediatamente a ordem de revogação para a *blockchain*. Revogar localmente permite que o *issuer* seja capaz de marcar várias credenciais para serem revogadas e revogá-las todas em simultâneo a qualquer momento. A tabela ilustrada apresenta apenas algumas informações do registo de revogação. O IRN pode visualizar mais informações sobre o registo clicando na linha da tabela correspondente a esse registo, tal como é possível observar na figura 84.

Figura 84: Informações detalhadas do registo de revogação

A.4.4 *Criação de conexões*

Após a base de todo o sistema estar preparada, resta apenas que as várias entidades comuniquem entre si de modo a partilharem todas as informações desejadas. Para que esta comunicação ser possível, é necessário estabelecer conexões seguras entre as entidades intervenientes, tal como é explicado na secção 3.3. Na aplicação desenvolvida, a gestão de todas as conexões é feita através da página ilustrada na figura 85.

Uma conexão pode ser iniciada de duas formas:

;**Criar um convite:** opção utilizada por quem pretende disponibilizar serviços e aguardar que outras entidades entrem em contacto. Nesta opção será criada uma conexão para cada entidade que aceitar o convite disponibilizado; **Utilizar um convite:** opção utilizada por entidades que visam obter os serviços disponibilizados pelo autor do convite.

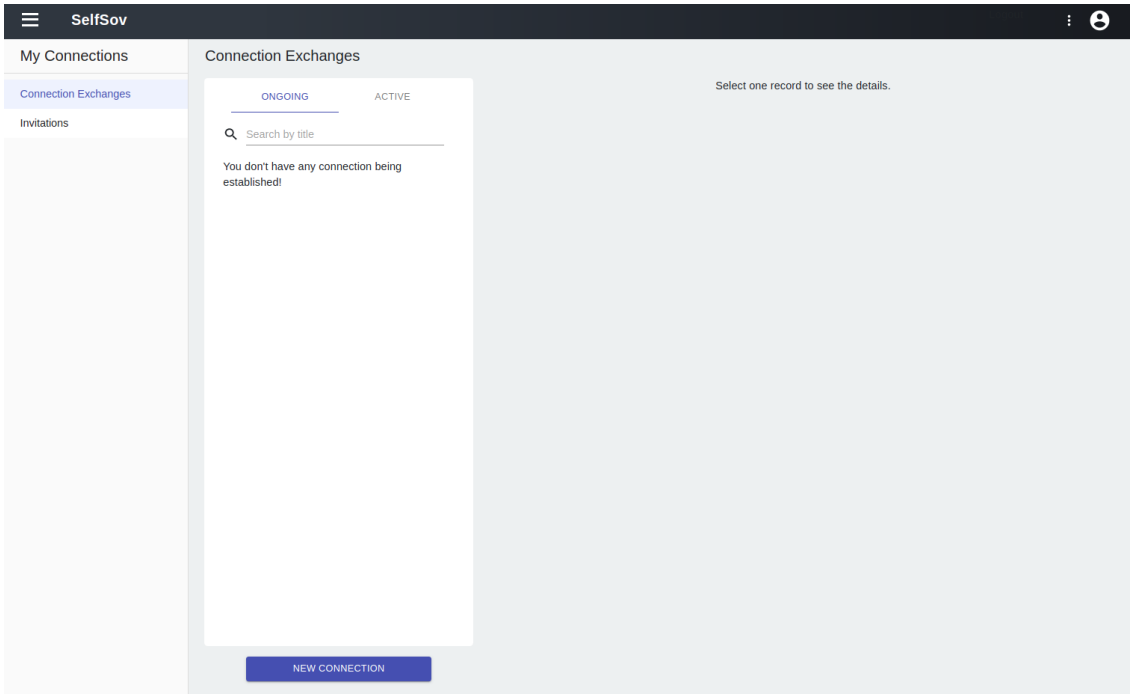


Figura 85: Página de gestão de conexões

No caso de uso apresentado, tanto o IRN como a CNE pretendem disponibilizar serviços que outras entidades (i.e. a Alice) possam utilizar. Deste modo, o IRN e a CNE irão criar convites, disponibilizando-os eventualmente em algum local acessível aos *holders* (i.e. QR Code no *website* ou nos respetivos estabelecimentos físicos). Por sua vez, a Alice irá aceitar cada um desses convites de maneira a estabelecer uma conexão individual com cada um deles. As figuras 86 e 87 mostram a criação dos convites de conexão do IRN (Serviço de Identificação) e da CNE (Serviço Eleitoral), respetivamente.

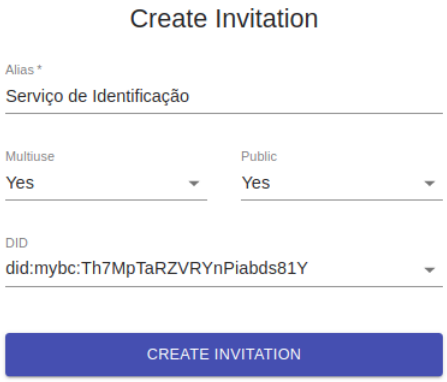


Figura 86: Criação do convite (IRN)

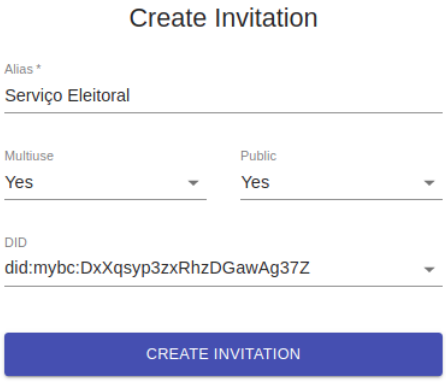


Figura 87: Criação do convite (CNE)

Durante a criação dos convites, as entidades podem escolher se o convite será utilizado apenas para o estabelecimento de uma conexão ou de múltiplas conexões. Independentemente da escolha, estas entidades poderão a qualquer momento desativar ou eliminar o convite, rejeitando assim qualquer tentativa de contacto a partir desse momento. Um convite desativado poderá ser ativado novamente a qualquer momento. A figura 88 mostra a página através da qual o IRN pode gerir todos os convites que criou.

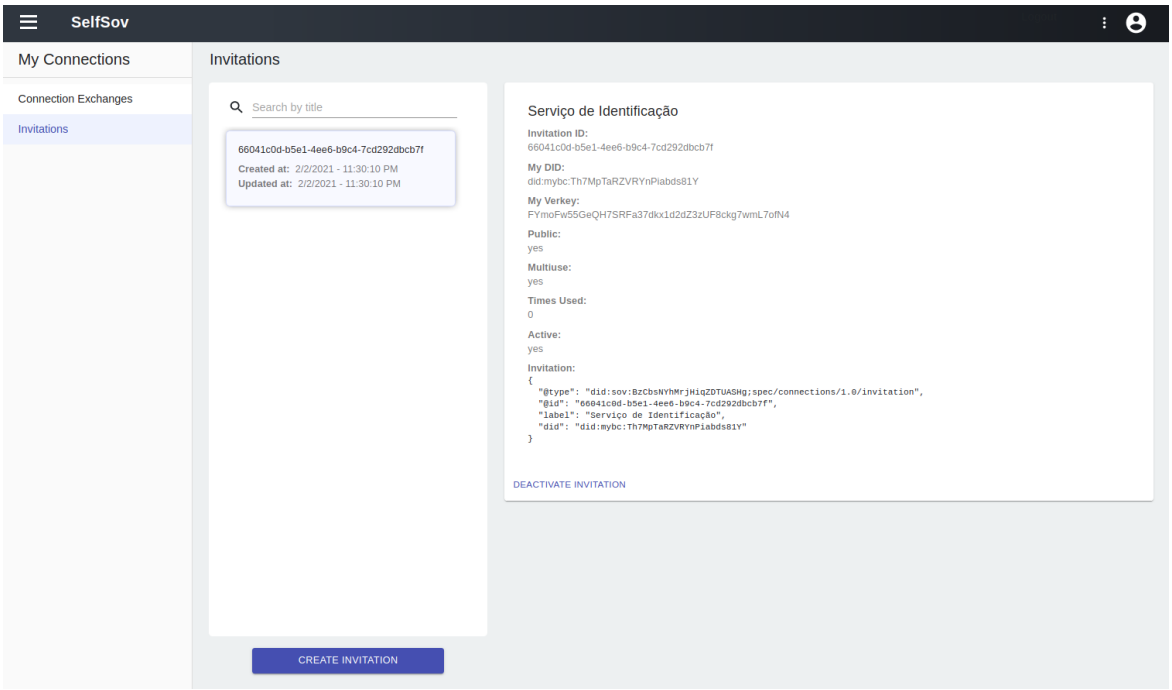


Figura 88: Página de gestão de convites de conexão

Nesta figura podemos observar a presença do convite criado anteriormente para o serviço de identificação. Do mesmo modo, também a CNE pode gerir todos os convites criados através da sua respetiva página.

O próximo passo será disponibilizar estes convites de forma a que outras entidades sejam capazes de os adquirir. Por questões de simplicidade, este exemplo assume que o convite é enviado diretamente à Alice em formato JSON, tal como representado na figura acima. Após receber ambos os convites, a Alice procede à importação dos mesmos através da página de gestão de conexões do seu agente. Para importar estes convites, a Alice deverá clicar no botão de criação de uma nova conexão e selecionar a opção de utilizar um convite. De seguida é apenas necessário escolher um nome para essa conexão e inserir os dados do convite que lhe foram enviados. As figuras 89 e 90 mostram a utilização dos convites do IRN e da CNE, respetivamente.

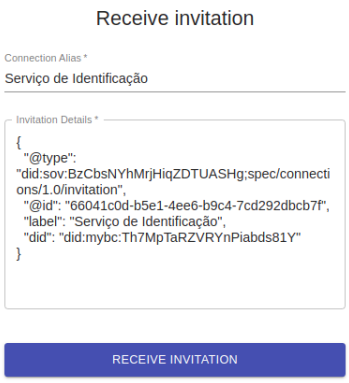


Figura 89: Utilização do convite do IRN

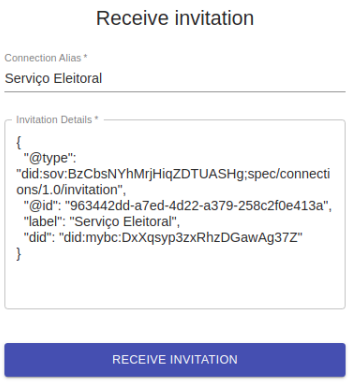


Figura 90: Utilização do convite da CNE

Devido à semelhança no estabelecimento de conexões, apenas será demonstrado o processo de estabelecimento de uma conexão entre a Alice e o IRN. Contudo, é necessário efetuar o mesmo processo entre a Alice e a CNE de modo a que ambos possam comunicar entre si.

Após utilizar o convite que recebeu do IRN, a Alice poderá optar por o aceitar ou rejeitar.

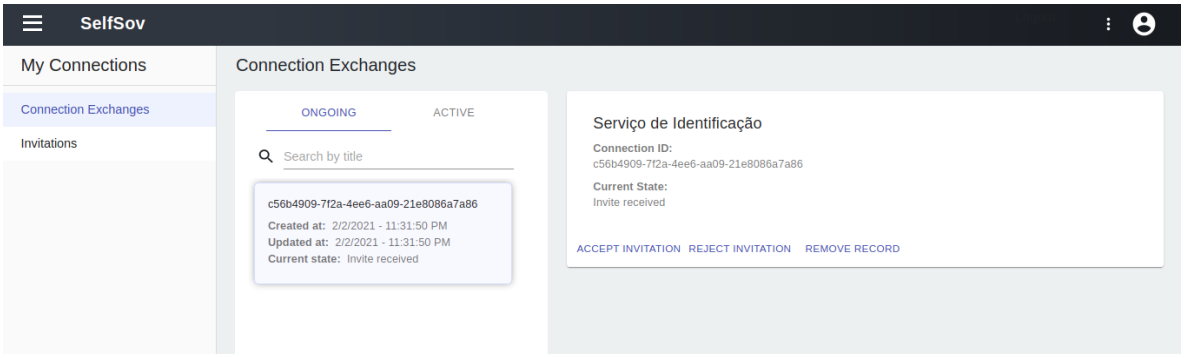


Figura 91: Página de gestão de conexões da Alice com uma nova conexão em progresso

Se optar por aceitar o convite, irá efetuar um pedido de conexão ao IRN e o estado do estabelecimento da conexão irá transitar para "Request Sent" (pedido enviado).

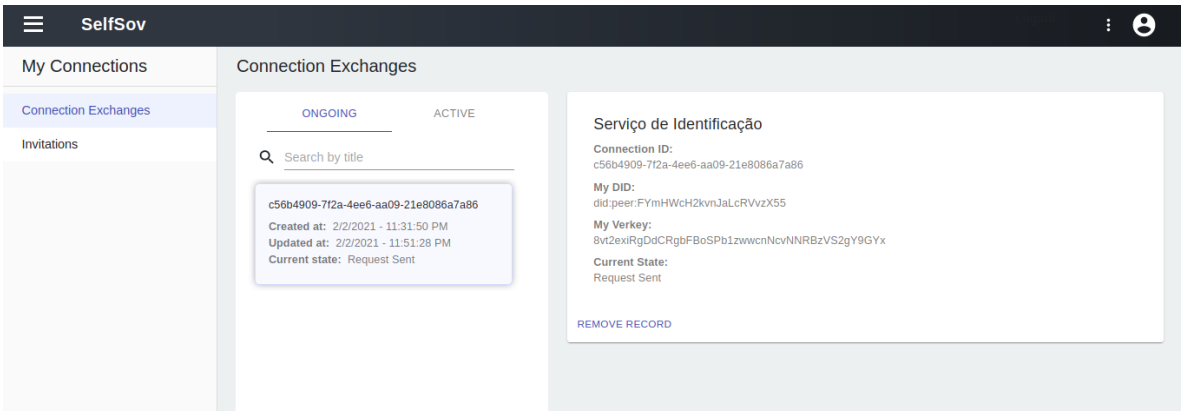


Figura 92: Página de gestão de conexões da Alice com um pedido enviado ao IRN

Ao receber o pedido da Alice, o IRN poderá também aceitá-lo ou rejeitá-lo, tal como mostra a figura 93. Se o IRN optar por aceitar o pedido da Alice, o estabelecimento da conexão será concluído com sucesso e as duas partes poderão começar a comunicar entre si.

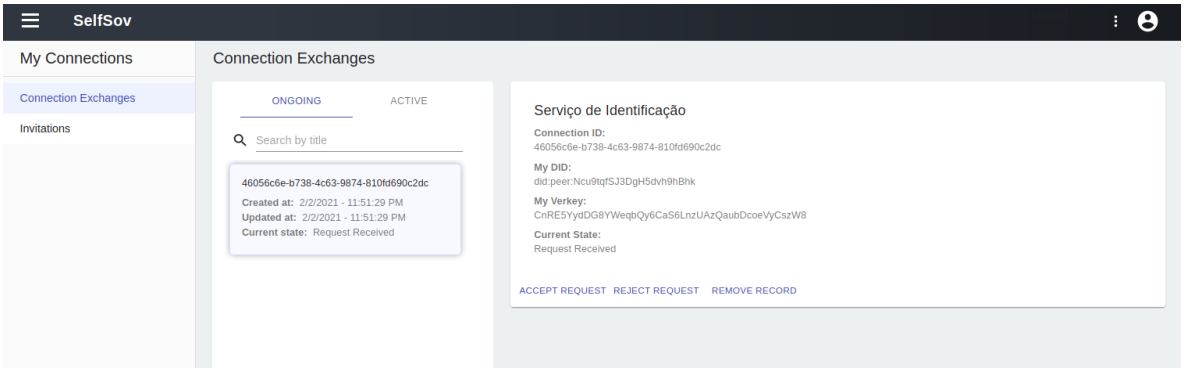


Figura 93: Página de gestão de conexões com um novo pedido

A figura 94 mostra novamente a página de gestão de conexões do IRN, na qual o IRN pode verificar que possui uma nova conexão com a Alice no estado ativo.

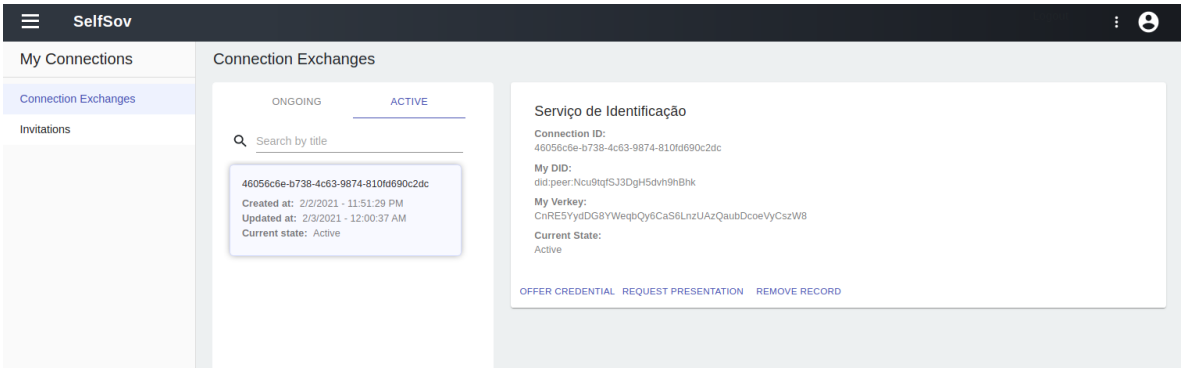


Figura 94: Conexão ativa entre o IRN e a Alice

Após o estabelecimento das duas conexões, a Alice terá dois DIDs de emparelhamento (*peer DIDs*), um para cada conexão, enquanto que o IRN e a CNE terão um novo DID de emparelhamento, tal como é possível verificar através das figuras 95, 96 e 97, respetivamente.

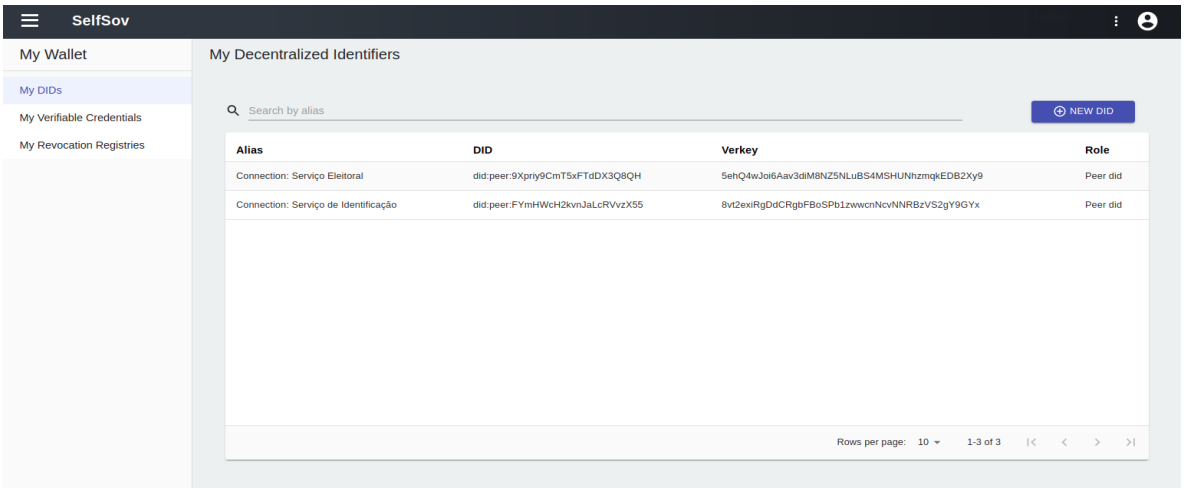


Figura 95: DIDs da Alice

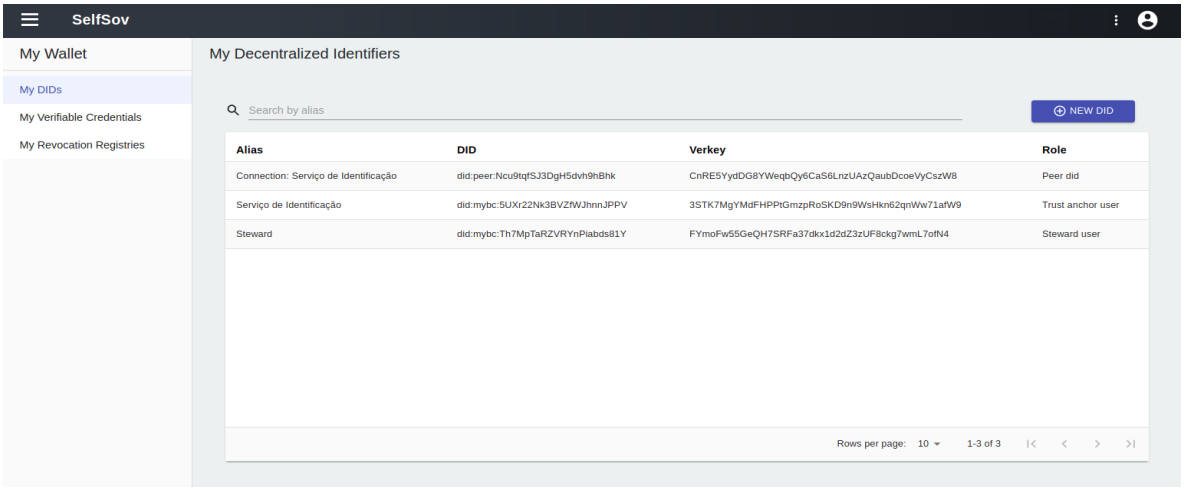


Figura 96: DIDs do IRN

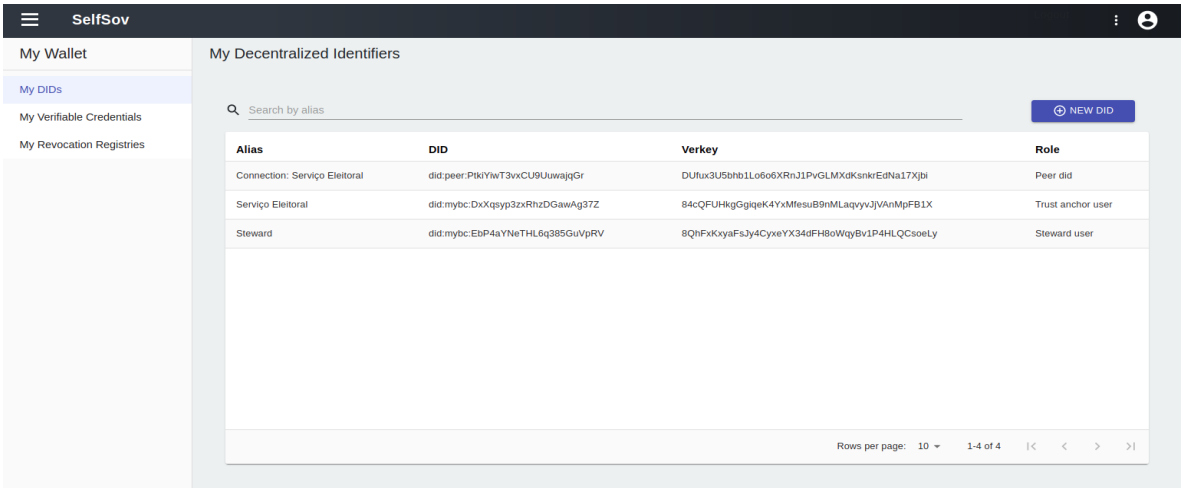


Figura 97: DIDs da CNE

A.4.5 *Emissão de uma credencial verificável*

Após a conexão entre a Alice e o IRN estar estabelecida, o IRN poderá emitir credenciais verificáveis sobre o DID de emparelhamento que a Alice está a utilizar nessa conexão e que ele já conhece. Para iniciar o processo de emissão da credencial destinada à Alice, o IRN pode fazê-lo através da conexão ativa que estabeleceu com a Alice (na página de gestão de conexões) ou através da página de gestão de emissão de credenciais ilustrada na figura 98.

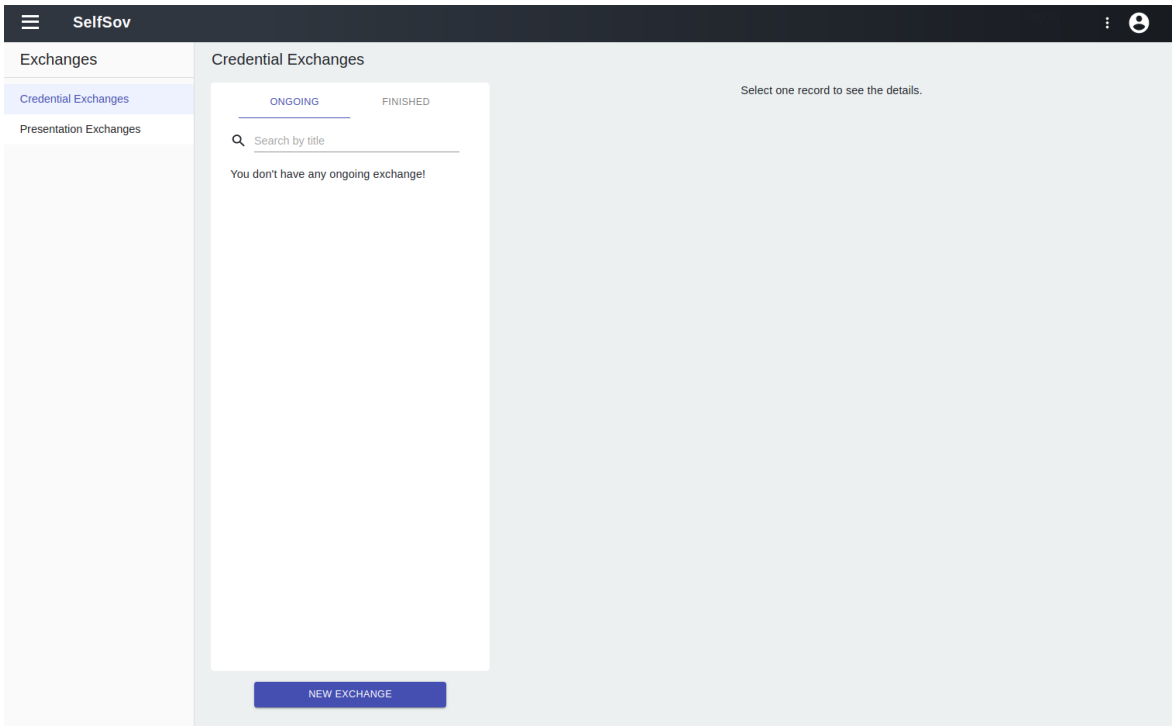


Figura 98: Página de gestão da emissão de credenciais

Ao clicar no botão para iniciar o processo de emissão de uma nova credencial ("New Exchange"), o agente de utilizador permite iniciar o processo de duas formas:

- O *holder* pode propor ao *issuer* a emissão de uma credencial específica;
- O *issuer* pode oferecer ao *holder* uma credencial.

No caso de uso apresentado, será o IRN a iniciar o processo, começando assim por oferecer uma credencial à Alice através da conexão que ambos estabeleceram. Após selecionar a opção de oferecer uma credencial, o IRN preenche o formulário de acordo com o exemplo ilustrado nas figuras 99, 100 e 101. Através das figuras apresentadas é possível observar que o IRN selecionou a conexão que estabeleceu com a Alice para enviar a oferta, selecionou o identificador da definição da credencial que criou anteriormente e inseriu o valor de cada um dos atributos presentes no esquema sob o qual a definição da credencial se baseia.

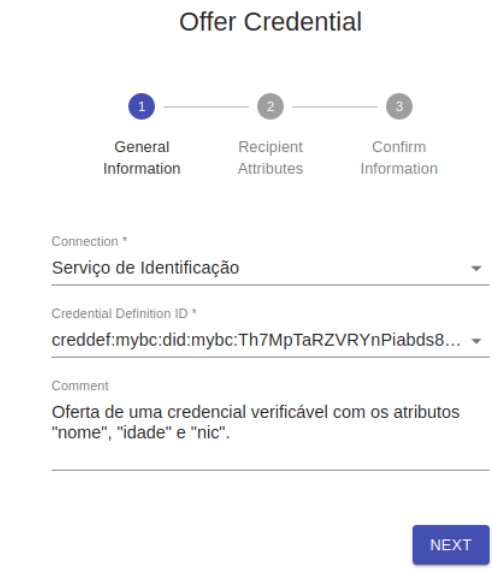


Figura 99: Formulário de emissão de uma credencial (I)

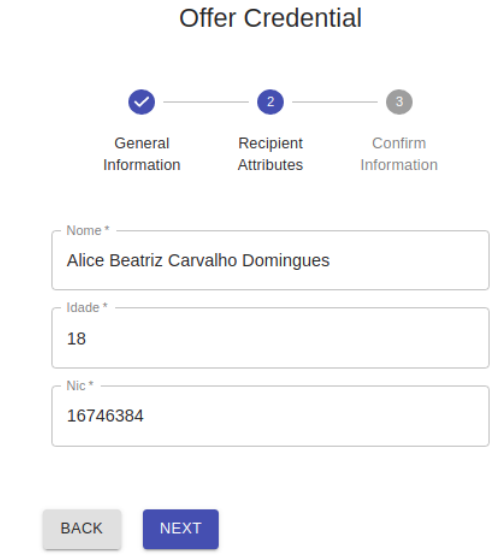


Figura 100: Formulário de emissão de uma credencial (II)

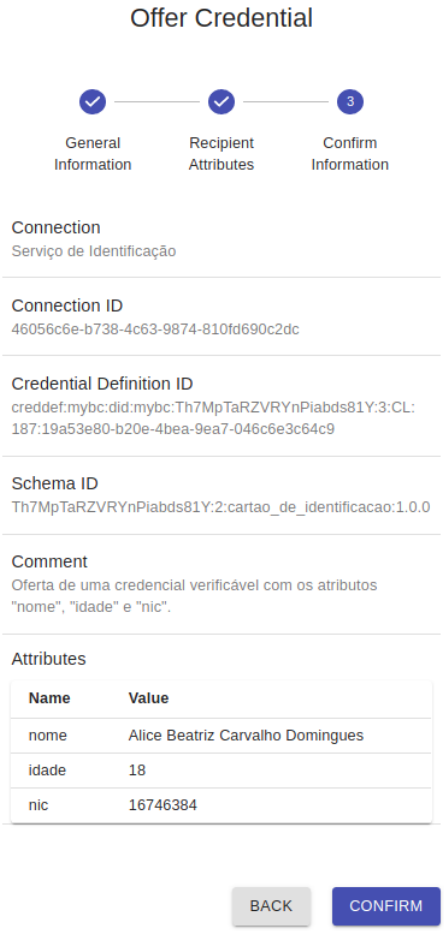


Figura 101: Formulário de emissão de uma credencial (III)

Após clicar em "Confirm", a oferta da credencial é enviada ao agente de utilizador da Alice e ambos poderão acompanhar o processo de emissão da credencial através da página de gestão da emissão de credenciais dos seus respetivos agentes. As figuras 102 e 103 mostram o registo gerado para o IRN e para a Alice, respetivamente.

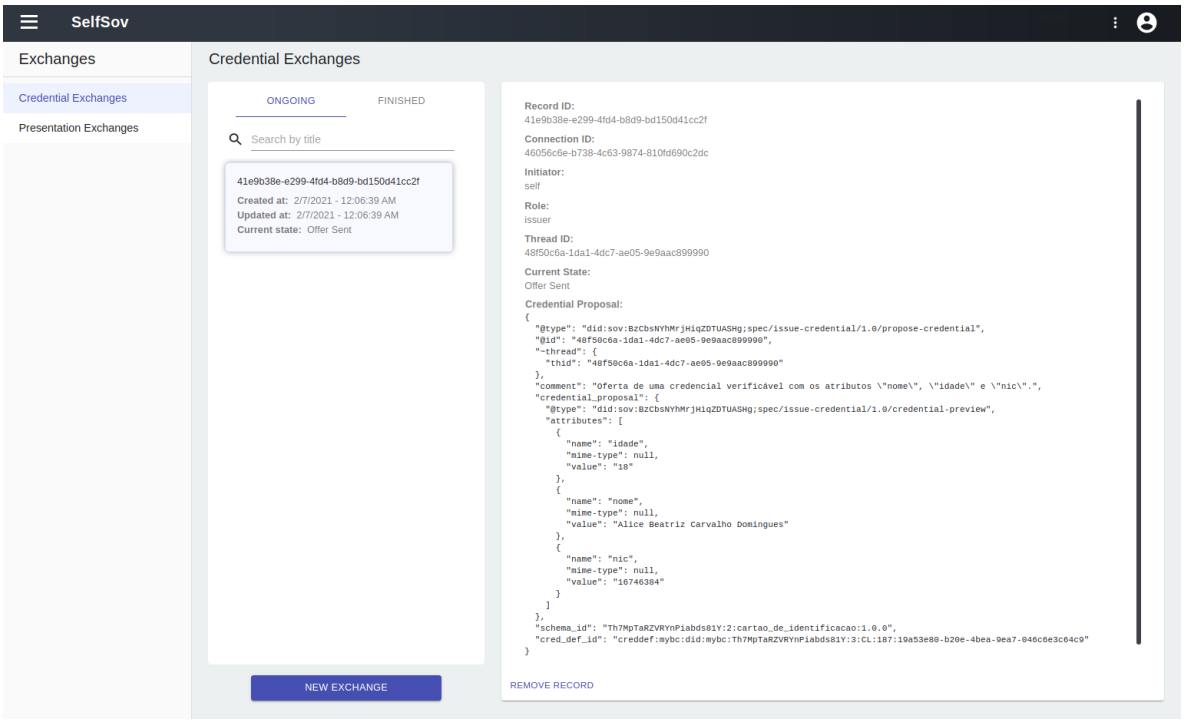


Figura 102: Página de gestão da emissão de credenciais do IRN

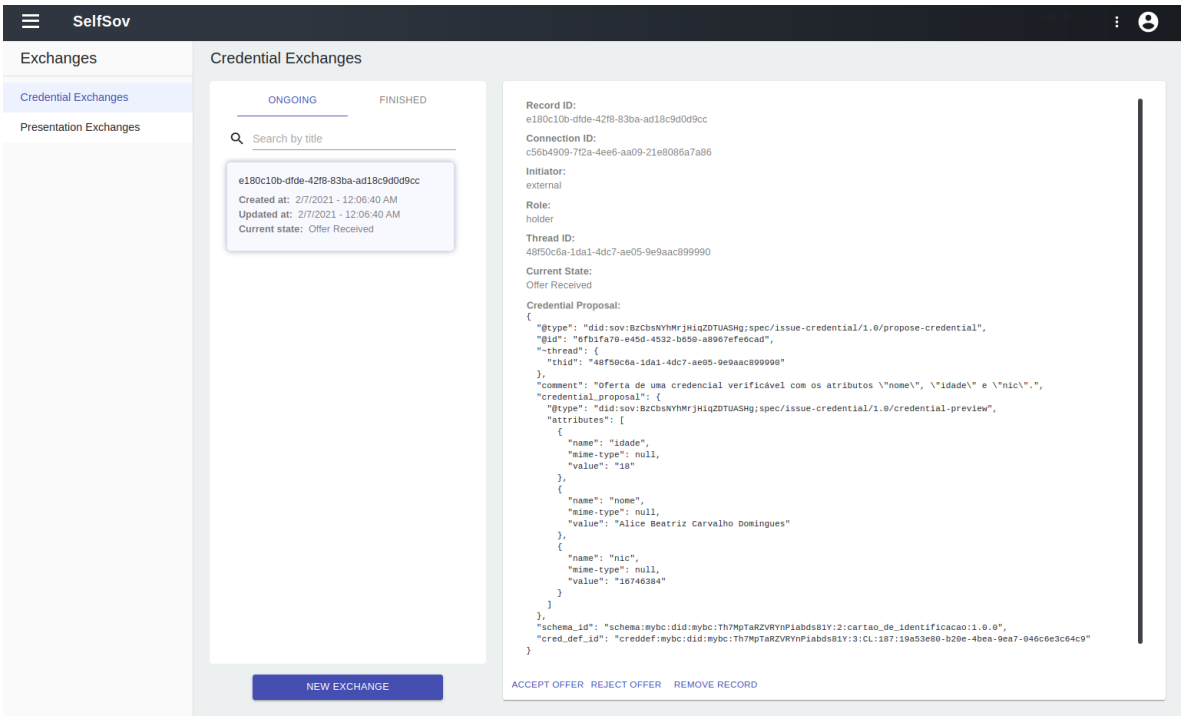


Figura 103: Página de gestão da emissão de credenciais da Alice

Ao receber a oferta do IRN para a emissão da credencial, a Alice poderá ver os detalhes dessa oferta e optar por aceitá-la ou rejeitá-la. Caso decida aceitar, será enviado um pedido de emissão de credencial ao IRN baseado na oferta que este enviou inicialmente.

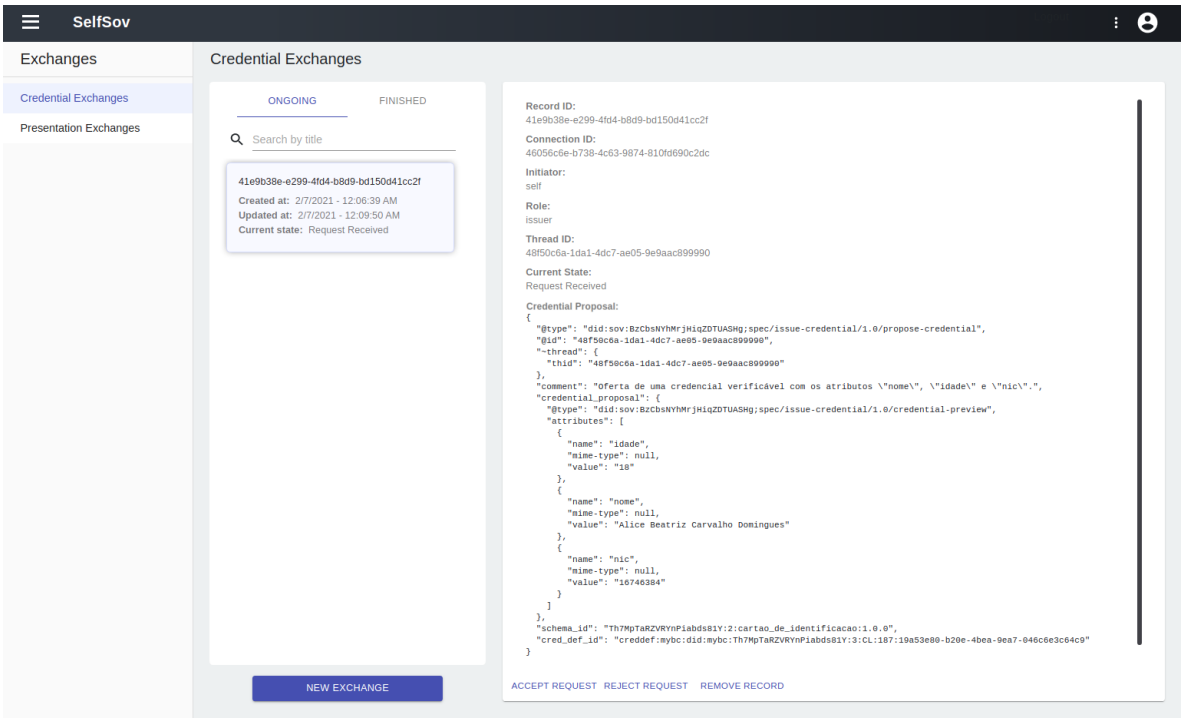


Figura 104: Página de gestão da emissão de credenciais do IRN (pedido recebido)

Assim que o IRN aceitar o pedido recebido, este irá emitir uma nova credencial e enviá-la à Alice, terminando assim o processo de emissão da credencial. Ao receber a credencial, o agente da Alice irá armazená-la na *wallet* e a Alice poderá consultá-la a qualquer momento, tal como mostra a figura 105.

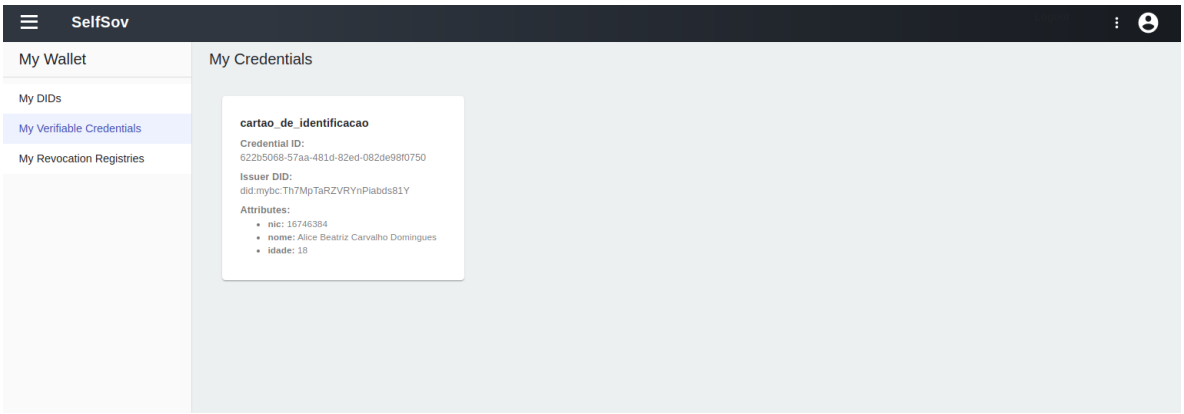


Figura 105: Página de gestão de credenciais da Alice

O IRN pode também verificar que o registo de revogação que criou anteriormente possui agora uma credencial associada, a credencial que emitiu para a Alice, e o estado do registo passou de "generated" (gerado) para "active" (ativo), tal como é possível observar na figura 106.

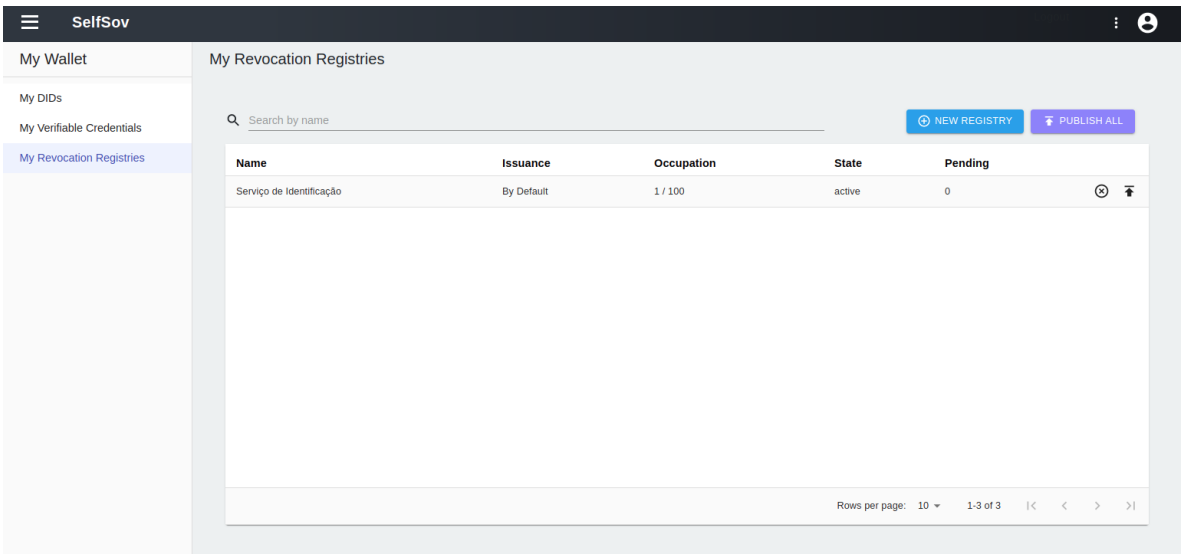


Figura 106: Página de gestão dos registo de revogação do IRN

A.4.6 *Apresentação de uma prova*

Após obter a credencial, a Alice deseja então exercer o seu direito ao voto. Para tal, deverá indicar a sua intenção à CNE, que por sua vez lhe irá enviar um pedido com os atributos necessários. Quando receber o pedido, a Alice deverá criar uma apresentação verificável com os atributos requisitados e fornecer essa apresentação à CNE.

A indicação dada pela Alice de que pretende exercer o seu direito de voto é uma ação externa à aplicação aplicação. Ao verificar que a Alice pretende exercer o seu direito de voto, a CNE precisa de confirmar que a Alice é uma cidadã portuguesa e que possui mais de dezoito anos. Para iniciar um novo pedido de apresentação de prova, a CNE pode aceder à página de gestão de conexões e iniciar o processo através da conexão ativa que possui com a Alice. A CNE também pode iniciar este processo através da página de gestão de apresentações de provas, ilustrada na figura 107.

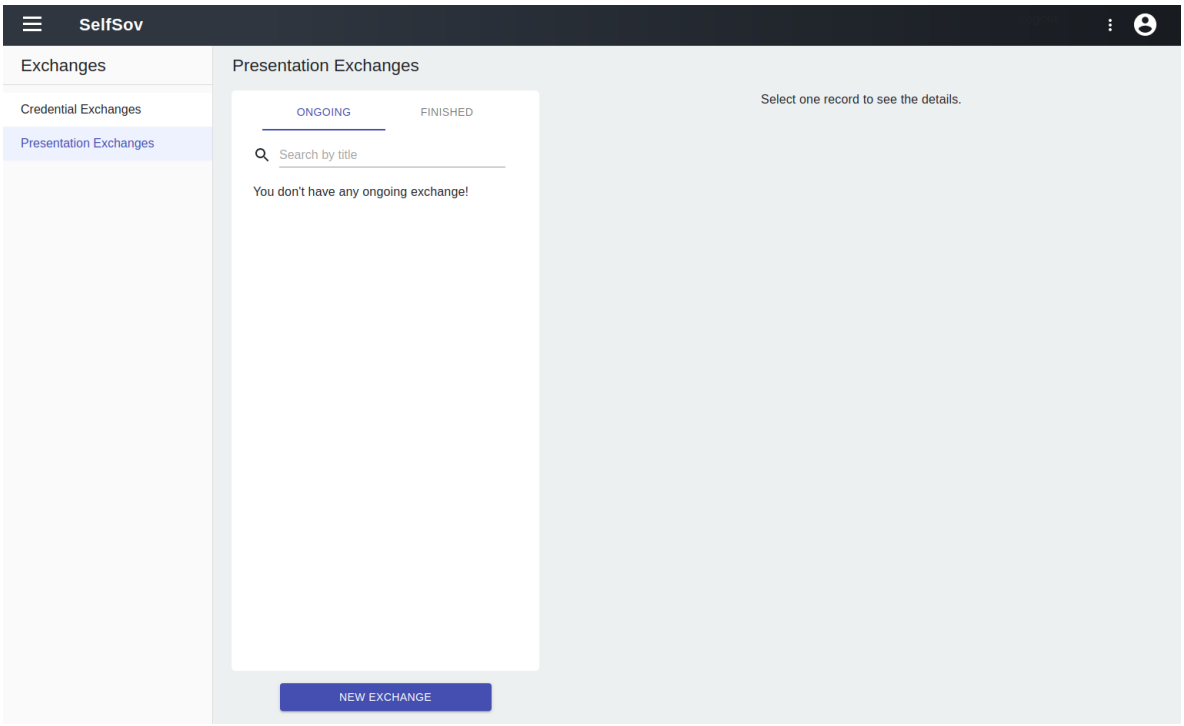


Figura 107: Página de gestão de apresentações de provas

Com o objetivo de validar se a Alice está autorizada a votar, a CNE cria um pedido de apresentação de prova que visa obter provas criptograficamente válidas de que a Alice possui os atributos necessários. Consequentemente, no seu pedido, a CNE indica que pretende obter uma prova com o número de identificação civil e outra prova que assegure que a Alice possui mais de dezoito anos. A figura 108 mostra o formulário preenchido pela CNE para a criação de um pedido de prova. Por sua vez, as figuras 109 e 110 mostram os atributos requisitados pela CNE bem como as restrições aplicadas aos mesmos. Através destas figuras é possível verificar que a CNE pretende obter uma prova de que todos os atributos requisitados sejam válidos e não tenham sido revogados até ao momento de criação do pedido de prova. Alternativamente, a CNE poderia estabelecer um intervalo específico para a não revogação dos atributos presentes na apresentação da prova. As figuras 109 e 110 mostram ainda a indicação de que a CNE apenas aceita atributos emitidos pelo IRN, através da inserção do DID que o IRN associou à definição da credencial, e que se baseiem no esquema criado pelo mesmo no inicio deste caso de uso.

É ainda possível observar que tanto o pedido geral como cada atributo desse pedido possuem um intervalo de não revogação. O intervalo especificado no pedido é usado apenas sobre os atributos requisitados no pedido que não possuírem um intervalo de revogação próprio, ou seja, o intervalo de revogação utilizado será sempre o mais específico.

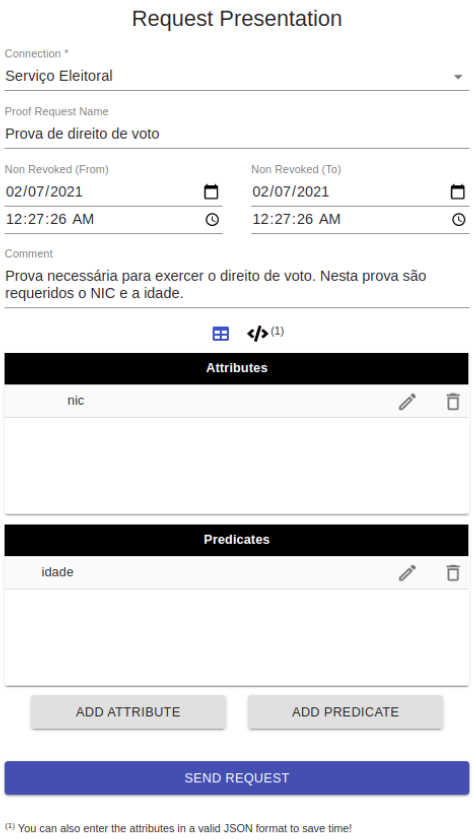


Figura 108: Formulário de pedido de prova

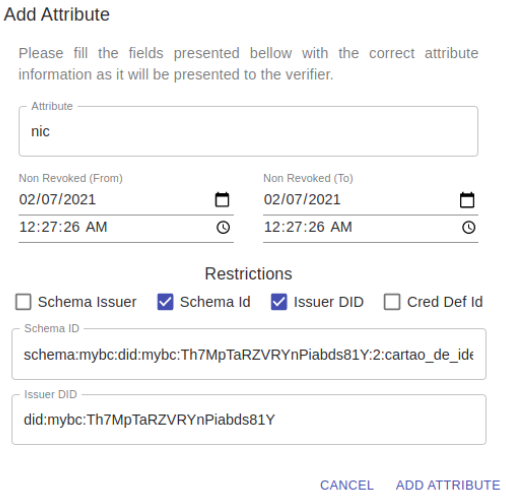


Figura 109: Adição do NIC ao pedido de prova

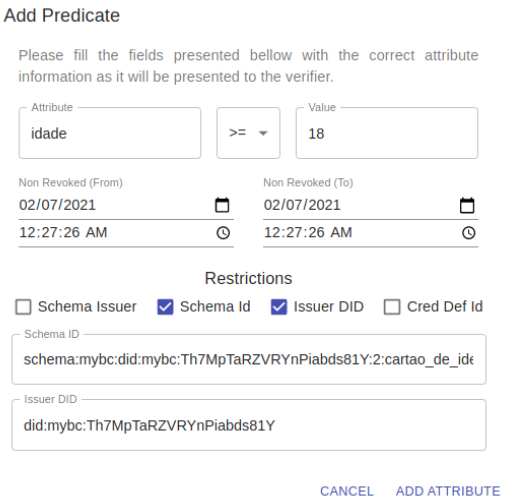


Figura 110: Adição do predicado "idade >= 18" ao pedido de prova

Em semelhança à criação de uma oferta para a emissão de uma credencial, após a CNE enviar o pedido de apresentação da prova serão criados registos que indicam o estado do processo tal como mostram as figuras 111 e 112.

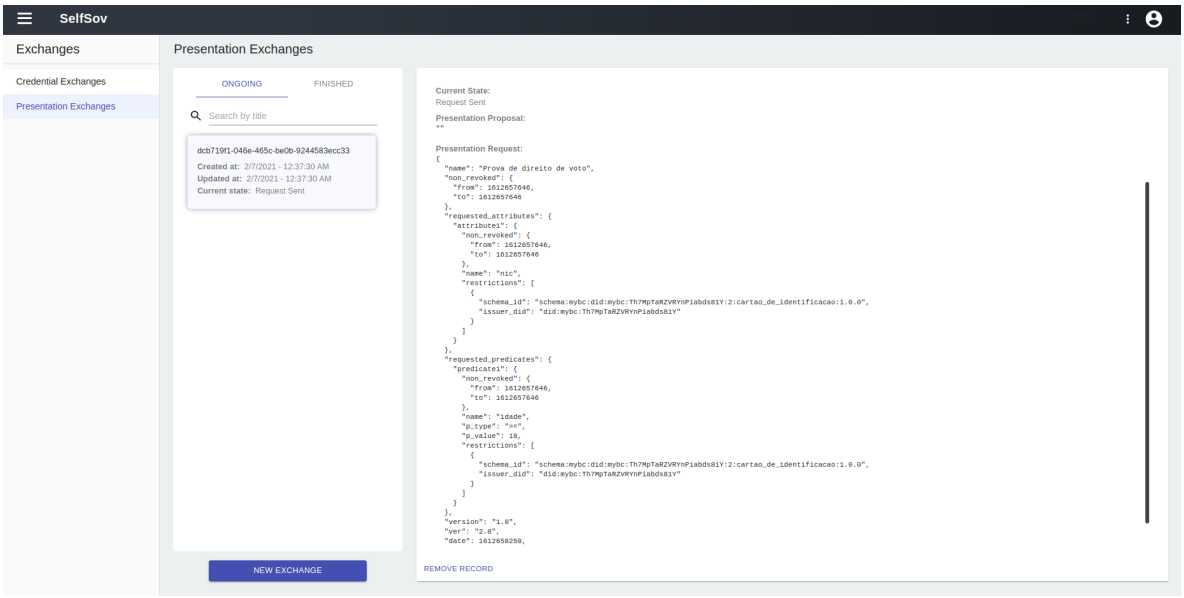


Figura 111: Página de gestão de apresentações de provas do IRN

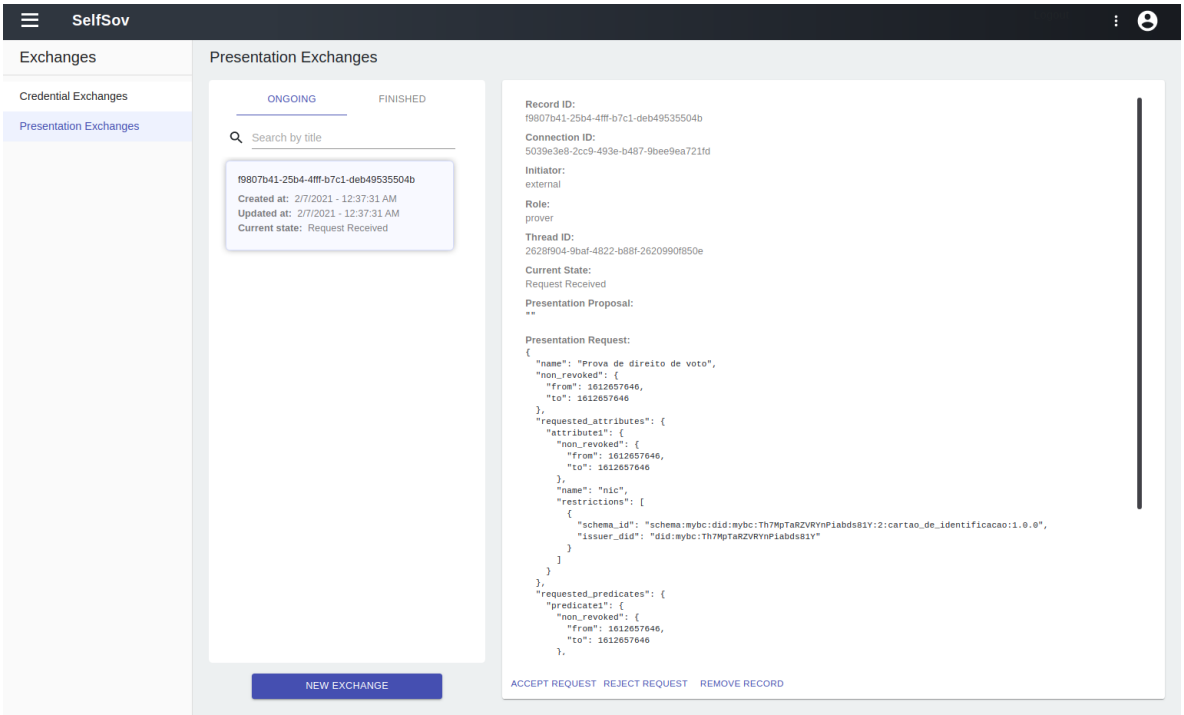


Figura 112: Página de gestão de apresentações de provas da Alice

Ao receber o pedido de apresentação de prova da CNE, a Alice pode aceitar ou rejeitar esse pedido. Caso opte por aceitar, ser-lhe-á apresentado um formulário no qual ela poderá escolher de quais credenciais pretende selecionar os atributos para construir a apresentação da prova. Se a Alice desejar, poderá agregar atributos de diferentes credenciais numa única

apresentação verificável desde que cada atributo individual cumpra as restrições impostas no pedido enviado pela CNE. É também importante referir que o agente do utilizador efetua todo o processo de filtragem de credenciais inválidas face ao pedido recebido de forma transparente ao utilizador e apenas apresenta as credenciais que cumprem as restrições indicadas nesse pedido.

Neste caso de uso a Alice possui uma única credencial pelo que apenas poderá escolher os atributos dessa credencial durante a construção da prova, tal como ilustra a figura 113.

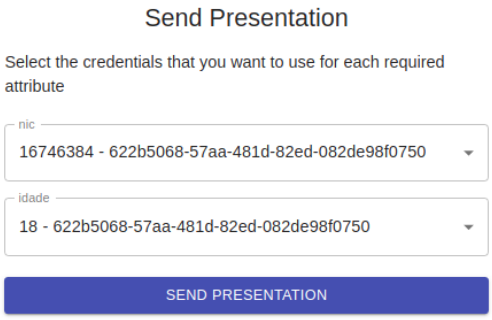


Figura 113: Criação da apresentação verificável

Ao receber a apresentação criada pela Alice, a CNE poderá verificar a validade da mesma clicando em "Verify Presentation" (Verificar Apresentação), como é possível observar na figura 114.

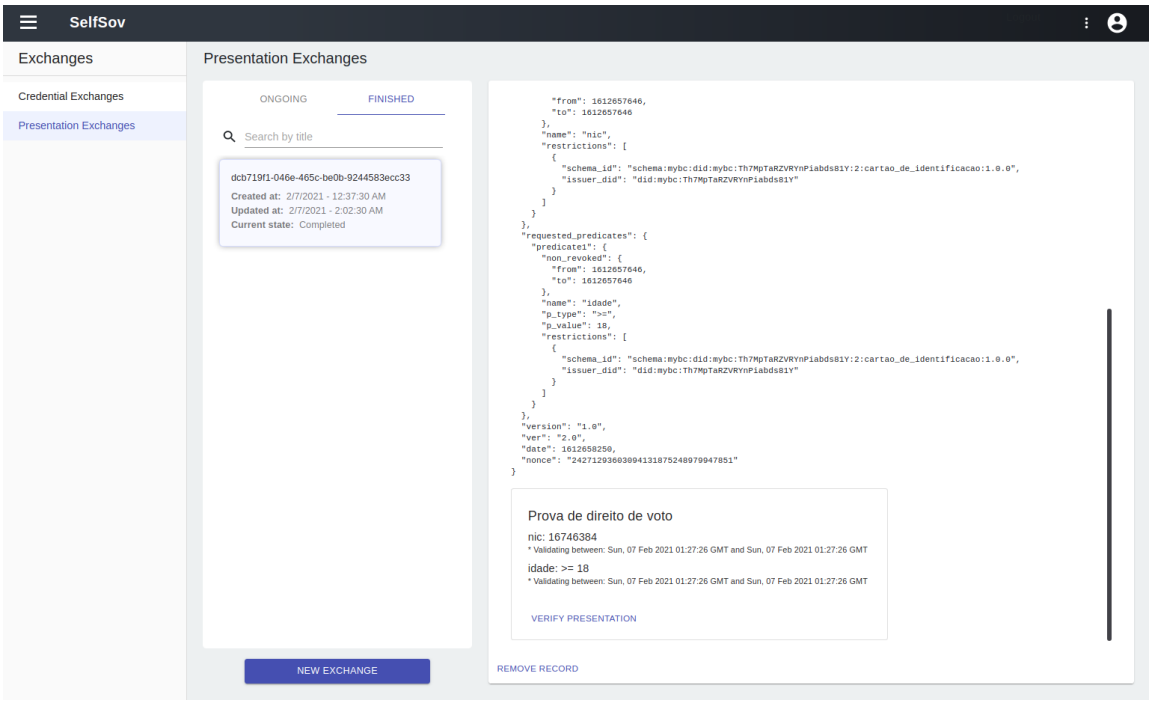


Figura 114: CNE recebe a apresentação verificável

Após verificar a validade da prova, a CNE irá obter a indicação de que a prova é válida dentro dos parâmetros estabelecidos, tal como mostra a figura 115.

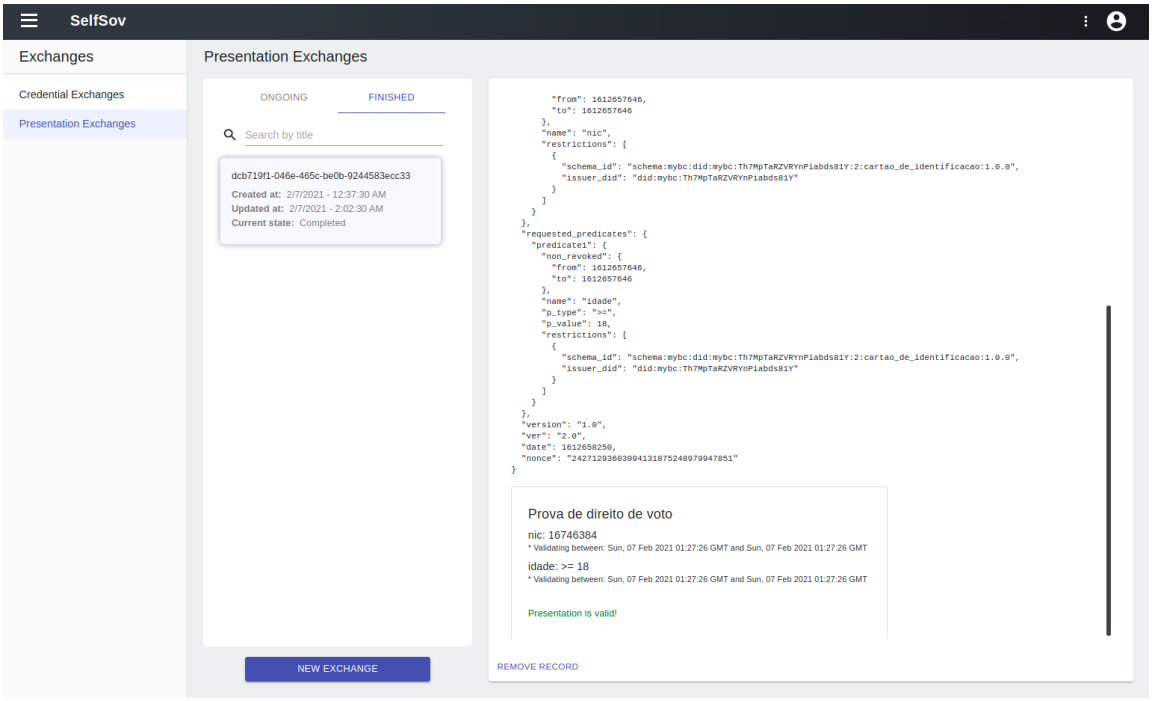


Figura 115: CNE verifica a validade da prova

Esta prova será sempre válida dado que os parâmetros de validação são estáticos, ou seja, mesmo que a credencial seja revogada, a verificação será sempre efetuada sobre o intervalo de não revogação indicado no pedido de prova pelo que a prova será sempre válida. Se a CNE pretender validar novamente a validade atual dos atributos requisitados à Alice, deverá criar um novo pedido de apresentação de uma prova seguindo o mesmo processo.

A.4.7 *Revogação de uma credencial*

Assumindo que por algum motivo o IRN decide revogar a credencial que emitiu à Alice anteriormente, pode fazê-lo através da sua página de gestão de registos de revogação, figura 83, clicando no "x" situado no lado direito da tabela. Ao clicar nesse botão, ser-lhe-á apresentado um formulário no qual deve indicar o identificador que identifica a credencial que pretende revogar dentro do respetivo registo de revogação e se pretende que a revogação seja efetuada de imediato ou se deverá ficar pendente para que possa ser revogada mais tarde junto com outras credenciais. Deste nodo, o IRN passa o identificador da credencial dentro do registo de revogação e indica que pretende revogá-la imediatamente, tal como mostra a figura 116.

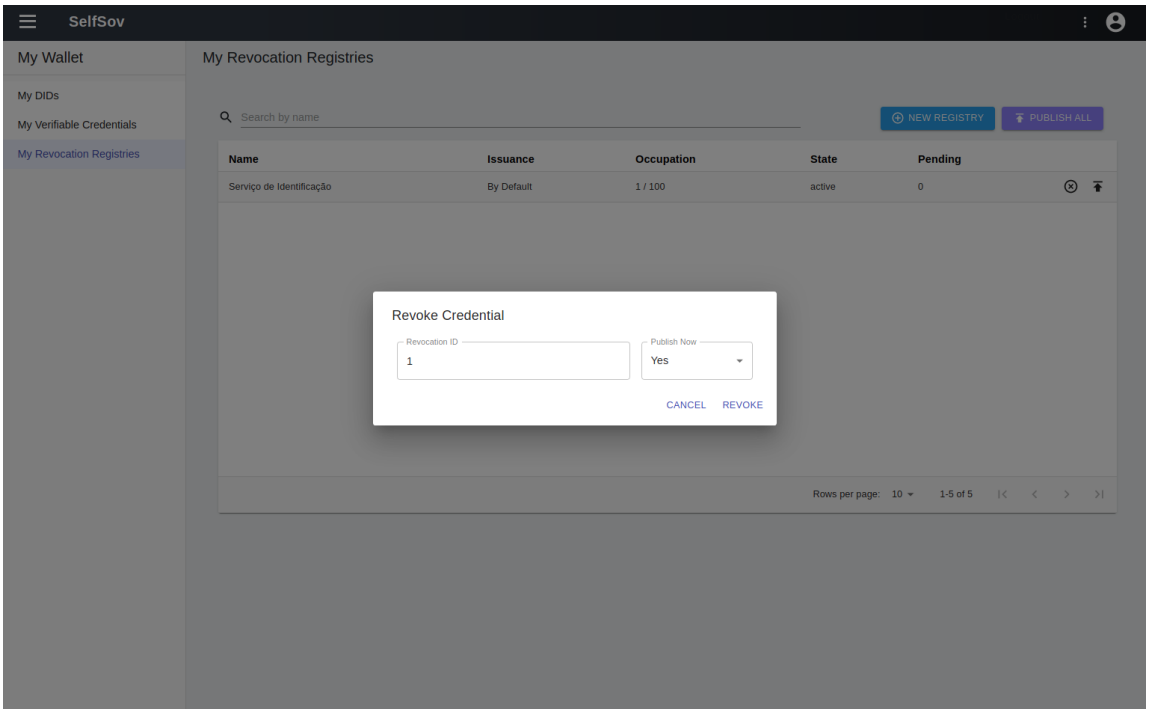


Figura 116: IRN revoga a credencial da Alice

Após a credencial ser revogada, o registo de revogação continuará com um espaço preenchido e, apesar da credencial da Alice ter sido revogada, nenhuma credencial emitida posteriormente poderá ser associada ao identificador utilizado para a credencial da Alice.

Se a CNE voltar a requisitar a apresentação de uma prova de acordo com as restrições indicadas anteriormente, mas com o intervalo de revogação aplicado a qualquer período posterior à revogação da credencial da Alice, esta não será capaz de gerar uma apresentação verificável válida. Na figura 117 é possível observar que apesar dos atributos respeitarem as restrições do esquema e o *issuer*, a apresentação que será gerada não é válida.

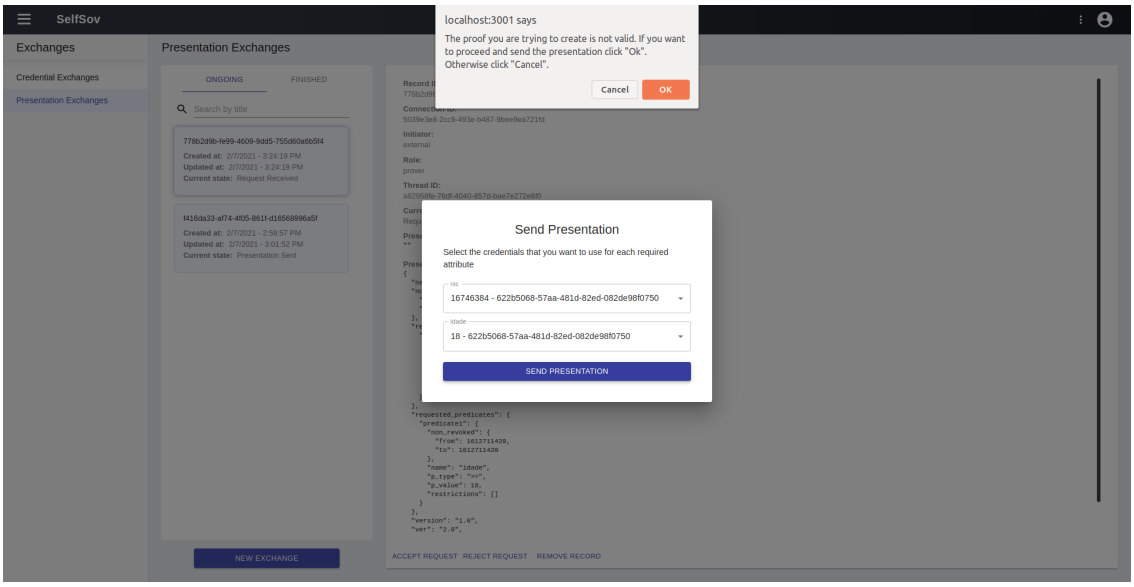


Figura 117: Criação de uma apresentação verificável inválida

Com o objetivo de demonstrar a verificação de uma prova inválida por parte da CNE, a Alice poderá enviar a prova mesmo que esta não seja válida.

Ao receber a apresentação verificável, a CNE procede à sua validação seguindo o mesmo processo. Contudo, desta vez a verificação da prova irá resultar na indicação de que essa prova não é válida, sendo isso visível através da figura 118.

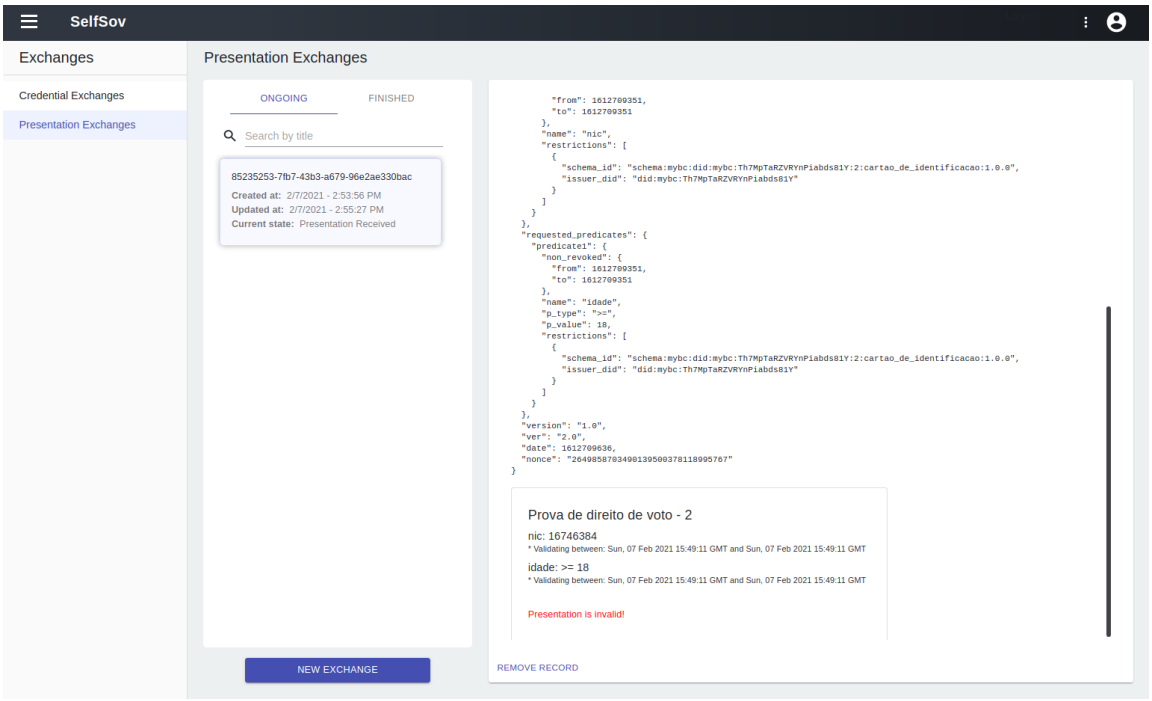


Figura 118: Apresentação verificável inválida

A.5 CONFIGURAÇÃO DA APLICAÇÃO

Esta secção irá apresentar os vários passos necessários para efetuar a configuração do ambiente linux no qual a aplicação desenvolvida irá correr bem como disponibilizar um *link* para uma máquina virtual desenvolvida já com todas as componentes a funcionarem corretamente.

A.5.1 *Configuração Manual*

A configuração apresentada nesta subsecção visa as distribuições de **Ubuntu 16.04** e **Ubuntu 18.04**. Relativamente às restantes distribuições, é possível que não suportem algumas das dependências do Indy SDK.

Pré requisitos

Para que seja possível executar as várias componentes da solução desenvolvida é necessário instalar primeiro o **NodeJs** [110] e o **npm**, para executar a aplicação desenvolvida, ou seja, o agente do utilizador, o **Docker** e o **Docker Compose** [111], para implementar os vários nós da *blockchain*, e o **Git** [112], necessário para efetuar os clones do repositório do agente desenvolvido nesta dissertação e do repositório com a instância da *blockchain* [62], desenvolvido pela Verifiable Organizations Network [63], para a máquina local. O **Node** deverá estar numa das versões mais recentes (>12) para que não ocorram problemas de compatibilidade em algumas das bibliotecas utilizadas pelos servidores que constituem o agente do utilizador. Adicionalmente é necessário instalar algumas dependências de modo a que seja possível utilizar o Indy SDK [55]. A figura 119 apresenta o conjunto de comandos que deverão ser executados para instalar essas dependências.

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys CE7709D068DB5E88
sudo add-apt-repository "deb https://repo.sovrin.org/sdk/deb (xenial|bionic) stable"
sudo apt-get update
sudo apt-get install -y libindy
```

Figura 119: Instalação das dependências do Indy SDK

Se a execução do primeiro comando falhar, o comando deverá ser ligeiramente alterado, tal como mostra a figura 120. Relativamente ao segundo comando, se a distribuição for o Ubuntu 16.04 deverá ser utilizado o "xenial" enquanto que para o Ubuntu 18.04 deverá ser utilizado o "bionic".

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys CE7709D068DB5E88
```

Figura 120: Comando alternativo para adquirir a chave

Integração do sistema

Tendo sido cumpridos todos os pré requisitos, deverá ser criada uma pasta vazia numa localização à escolha. Dentro desta pasta deverão ser efetuados clones de dois repositórios Git, o repositório do agente do utilizador desenvolvido nesta dissertação (*https://github.com/Ricardo1597/SSI-EdgeAgent*) e o repositório com a implementação de uma *blockchain* baseada nos projetos Indy Node [58] e Indy Plenum [59] que foi desenvolvida pela Verifiable Organizations Network [63] (https://github.com/bcgov/von-network.git). Após efetuados os clones de ambos os repositórios, a pasta deverá conter os projetos "SSI-EdgeAgent" e "von-network".

Dentro da pasta "von-network" deverão ser executados os comandos apresentados na figura 121 para iniciar a *blockchain*. O comando "*./manage build*" é utilizado para construir uma imagem docker designada por "von-network-base", que será utilizada na criação dos vários *containers* necessários, isto é, os quatro nós da *lbockchain* e o servidor que monitoriza essa mesma *blockchain*. Após a construção da imagem deverá ser executado o comando "*./manage start*" para inicializar os vários *containers*.

```
./manage build
./manage start
```

Figura 121: Inicialização da *blockchain*

É provável que seja necessário utilizar "*sudo*" antes de executar comandos do Docker. Para remover esta necessidade basta apenas criar um grupo para o Docker e adicionar o utilizador a esse grupo tal como mostra a figura 122.

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

Figura 122: Adição do utilizador ao grupo do Docker

Através do comando "*docker container ls*" podemos confirmar que estão efetivamente a correr quatro nós, cada um no seu *container* ("von_node1_1", "von_node2_1", "von_node3_1" e "von_node4_1"). Podemos também verificar que é apresentado um *container* adicional, "von_webserver_1". Este *container* é constituído por um servidor que permite consultar o estado dos nós, as transações efetuadas nos vários *ledgers* de uma rede Indy (*domain*, *pool* e *config*) e as transações de génese utilizadas para a instância da rede implementada. Após a inicialização do *container* do servidor é possível consultar as informações fornecidas pelo mesmo através da porta 9000 do *localhost*.

Para remover os *containers* criados é apenas necessário executar o comando "*./manage stop*" dentro da pasta "von-network".

Depois de os vários nós da *blockchain* estarem em execução, resta apenas executar três agentes de utilizador, um para o *issuer*, outro para o *holder* e um último para o *verifier*, que se irão conectar à *blockchain* criada. Para tal, dentro da pasta "SSI-EdgeAgent" deverão ser executados os comandos apresentados na figura 123. O primeiro comando irá instalar todos os pacotes do Node necessários tanto para o servidor de *backend*, com toda a lógica do agente do utilizador, como para o servidor ReactJs, com a interface desse mesmo agente. Os comandos seguintes irão apenas inicializar os dois servidores (*frontend* e *backend*) que constituem cada um dos três agentes.

```
sudo ./installPackages.sh
./startAgent1.sh
./startAgent2.sh
./startAgent3.sh
```

Figura 123: Inicialização do agente do utilizador

Após a execução destes comandos, é possível aceder aos agentes três agentes através das portas 3000, 3001 e 3002 do *localhost*.

A.5.2 *Máquina virtual*

Com o objetivo de facilitar a configuração do ambiente no qual a aplicação será inserida, é também fornecido o *snapshot* de uma máquina virtual que já apresenta todos os componentes do sistema corretamente configurados. Este *snapshot* está disponível em *https://drive.google.com/drive/folders/1rw-qHBIq_3GvLo26TfxyywsbFhmgEL7u?usp=sharing*. As credenciais de acesso à máquina virtual são:

- Nome do utilizador: ssi-poc
- Senha: SSI-Poc-2021

Relativamente à aplicação foram criados três utilizadores: a Alice, o IRN e a CNE, isto é, as entidades utilizadas no caso de uso apresentado na secção A.4. Seguem-se as credenciais de acesso de cada um destes utilizadores:

- Alice:
  - Nome do utilizador: Alice
  - Senha: 123456
- CNE:
  - Nome do utilizador: cne.gov.pt
  - Senha: 123456
- IRN:
  - Nome do utilizador: irn.gov.pt
  - Senha: 123456