HOCHSCHULE
MITTWEIDA
University of Applied Sciences

MASTER-THESIS

Herr
Christoph Menzer

Interoperabilität zwischen DID-Methoden, Wallets, Agents und Verifiable-Credentials

2022

HOCHSCHULE
MITTWEIDA
University of Applied Sciences

MASTER'S THESIS

Mr.
Christoph Menzer

Interoperability of DID methods, wallets, agents, and Verifiable Credentials

2022

Fakultät **Angewandte Computer- und Biowissenschaften**

MASTER-THESIS

Interoperabilität zwischen DID-Methoden, Wallets, Agents und Verifiable-Credentials

Autor:
Christoph Menzer

Studiengang:
Blockchain & Distributed Ledger Technologies

Seminargruppe:
BC1Bw1-M

Erstprüfer:
Prof. Dr.-Ing. Andreas Ittner

Zweitprüfer:
Dipl.-Volkswirt Mario Oettler

Mittweida, 2022

Bibliografische Angaben

Menzer, Christoph: Interoperabilität zwischen DID-Methoden, Wallets, Agents und Verifiable-Credentials, 66 Seiten, 15 Abbildungen, Hochschule Mittweida, University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften

Master-Thesis, 2022

Referat

DID-Methoden, Wallets, Agents und Verifiable-Credentials sind grundlegende Begriffe im Kontext von Self-Sovereign-Identity (SSI) und stellvertretend für neuartige Methoden der Identitätsverwaltung im Internet. Es werden gegenwärtig Entwürfe von Standards und Spezifikationen unterschiedlicher Gruppen und Gremien forciert, die dem Paradigma von SSI gerecht werden wollen. Aus der Vielzahl technologischer Ansätze, die bereits entstanden sind, werden einige wichtige näher betrachtet und hinsichtlich ihrer Interoperabilität untersucht. Ausgangspunkt ist dabei der Trust-over-IP-Stack, wie er von gleichnamiger Organisation (Trust-over-IP-Foundation) vorangetrieben wird. Dabei spielen weitere Normungsgremien eine Rolle, wie z. B. die Decentralized-Identity-Foundation (DIF) oder das World-Wide-Web-Consortium (W3C). Gegenstand der Untersuchung ist der aktuelle Stand der Technik und dessen Implikationen hinsichtlich ihrer Interoperabilität, Portabilität sowie dem angestrebten Ziel der Dezentralisierung. Dabei stehen insbesondere die beiden Entwürfe zu den Standards der Decentralized-Identifiers und des Verifiable-Credentials-Data-Models im Mittelpunkt. Es werden aber auch weitere Spezifikationen betrachtet, die diese ergänzen und für derartige Identitätsverwaltungssysteme von Bedeutung sind.

I. Inhaltsverzeichnis

II. Abbildungs- und Tabellenverzeichnis

Abbildungen

Tabellen

1 Einleitung

Die Organisationen *Decentralized-Identity-Foundation (DIF)* und das *World-Wide-Web-Consortium (W3C)* unternehmen zurzeit gemeinsam mit einigen Unternehmen Anstrengungen, einen neuen Standard für den Umgang mit Digitalen Identitäten zu etablieren, der ein Paradigma für *Dezentralisierte Identitäten (Decentralized-Identity*) einführt. Darüber hinaus wird derzeit in der Community das Konzept der digitalen Selbst-souveränen Identität (*Self-Sovereign-Identity, SSI*) diskutiert und von einigen Unternehmen aktiv beworben. SSI stellt weitere Forderungen, die vor allem den Umgang mit persönlichen Daten und die Interoperabilität zwischen verschiedenen Identitätssystemen verbessern sollen.

Das W3C führt in einer offiziellen Liste mehr als 80 verschiedene technische Lösungen auf, denen der W3C-Standard für *Decentralized-Identifier (DID)* zugrunde liegt (Steele u. a. 2021; Sporny, Longley, Sabadello u. a. 2021). Darüber hinaus existieren einige weitere Vorschläge und Standardisierungsbestrebungen, die sich der Interoperabilität unterschiedlicher Implementierungen annehmen. Hierbei spielen u. a. die Begriffe DID-Methoden, Wallets, Agents und Verifiable-Credentials (VCs) eine wesentliche Rolle. Zur Förderung von interoperablen Lösungen gibt es zudem eine eigene Arbeitsgruppe innerhalb der DIF, die *Interop-Working-Group* (*DIF - Interoperability* 2021).

Die *Trust-Over-IP-Foundation* unter dem Dach der *Linux-Foundation* erweitert den technischen Fokus um Aspekte hinsichtlich Vertrauen und Regulierung (Governance) zwischen bzw. von technischen Systemen sowie den Akteuren, die für den Betrieb, die Entwicklung und ggf. der Zertifizierung dieser (Software-) Systeme verantwortlich sind. Das daraus abgeleitete Schichtenmodell ordnet den technischen Aspekten die jeweiligen Nicht-technischen zu und dient als Grundlage dieser Arbeit (*Introducing the Trust Over IP Foundation* 2020; Davie u. a. 2019).

Aus der Vielzahl von Lösungen und Spezifikationen, die sich im Kontext von SSI bisher herausgebildet haben und sich auf unterschiedlichen Schichten ansiedeln, sollen die aktuell wichtigsten identifiziert und hinsichtlich ihrer Bedeutung für Interoperabilität untersucht werden. Aus diesen Erkenntnissen soll eine Kategorisierung erarbeitet werden, die den jeweiligen Lösungen oder Spezifikationen das zugrundeliegende Problem und dessen Schicht zuordnet. Darüber hinaus sollen nicht-technische Aspekte betrachtet und einbezogen werden, die hinsichtlich Regulierung und Semantik einen Einfluss auf die Interoperabilität haben. Ziel ist es, einen umfassenden Überblick über die aktuell wichtigsten Entwicklungen in der SSI-Community zu geben und diese anhand ihrer jeweiligen Eigenschaften zu klassifizieren.

Hierfür werden zunächst die theoretischen Grundlagen erörtert, die zum Verständnis der weiteren Betrachtungen dienen. Neben dem Schichtenmodell der Trust-Over-IP-

Foundation sollen außerdem weitere Schichtenmodelle ausgemacht und verglichen werden. Des weiteren sollen eventuell vorhandene Parallelen zu den heute bereits existierenden Technologien gezogen und die neuartigen SSI-Lösungen dem Status quo gegenüber gestellt werden.

2 Grundlagen

2.1 Self-Sovereign-Identity

Self-Sovereign-Identity (SSI) beschreibt ein Modell für eine gemeinsame Infrastruktur für digitale Identitäten, die resistent gegenüber Zensur und einseitiger Einflussnahme ist. Es zeichnet sich weiterhin durch einen möglichst hohen Schutz der persönlichen Daten sowie eine breite Interoperabilität aus. Der Benutzer soll dabei zu jeder Zeit die Kontrolle über seine digitale(n) Identität(en) behalten und selbst darüber entscheiden können, welche Informationen er mit wem teilt. Persönliche Daten sind dabei so zu speichern, dass ausschließlich Berechtigte an diese Informationen gelangen.

Über eine allgemeingültige Definition herrscht derweil jedoch kein Konsens. Christopher Allen, Mitglied der *Credentials-Community-Group* (W3C) sowie Co-Autor des *TLS-Security-Standard*, stellt in seinem Blog-Beitrag „The Path to Self-Sovereign Identity“ zehn Prinzipien auf, die technische Systeme für die Idee einer Self-Sovereign-Identity erfüllen sollten (Allen 2016):

Existence (Existenz) jeder Nutzer hat eine unabhängige Existenz, d.h eine SSI basiert auf einer realen/physischen Persönlichkeit und kann nicht ausschließlich in der digitalen Welt existieren.

Control (Kontrolle) jeder Nutzer sollte die alleinige Kontrolle über seine Identität haben.

Access (Zugriff) der Nutzer sollte zu jeder Zeit uneingeschränkten Zugriff zu seinen Daten haben. Das heißt allerdings nicht, dass der Nutzer seine Daten jederzeit bearbeiten kann.

Transparency (Transparenz) die Algorithmen, über die Identitäten verwaltet werden, müssen frei und quelloffen und weitestgehend unabhängig von speziellen Systemarchitekturen sein. Ähnliches gilt für die Netzwerke, die zur Identitätsverwaltung genutzt werden.

Persistence (Persistenz) Identitäten sollten für immer, oder wenigstens so lange, wie der Nutzer es wünscht, gültig sein.

Portability (Portabilität) Identitäten sollten nicht an ein bestimmtes Netzwerk gebunden sein.

Interoperability (Interoperabilität) Identitäten sollten wenn möglich überall, d.h. über Landesgrenzen und Grenzen digitaler Systeme hinweg, nutzbar sein.

Consent (Zustimmung) jede Freigabe von persönlichen Informationen gegenüber Dritten erfordert die Zustimmung des Nutzers.

Minimalization (Minimalismus) es sollte möglich sein, nur Informationen freizugeben, die unbedingt notwendig sind. Typisches Beispiel ist der Nachweis eines bestimmten Alters. Wenn nur gefragt ist, ob man älter als 18 Jahre alt ist, sollte auch nur diese Information zur Verfügung gestellt werden und nicht etwa das genaue Alter oder gar das Geburtsdatum.

Protection (Schutz) die Freiheiten und Rechte eines Nutzers wiegen höher als die Interessen des Identitätsnetzwerkes und sollten geschützt werden.

2.2 Decentralized-Identifier

Decentralized-Identifiers (DIDs) beschreiben einen neuartigen Typ von Identifiers. Sie sollen verifizierbar und „selbst-souverän“ sein. Sie sind unter der vollen Kontrolle des *DID-Subjects* (Controller) und dabei nicht auf zentrale Vergabestellen, Identity-Providers (IdPs) oder Zertifizierungsstellen (Certificate-Authority, CA) angewiesen (vgl. Sporny, Longley, Sabadello u. a. 2021). Ein DID enthält die Bezeichnung eines Schemas, die Bezeichnung einer DID-Methode und den methodenspezifischen Identifier selbst. Die Bezeichner werden durch einen Doppelpunkt getrennt: did:example:123456789abcdefghi.

Das Schema kennzeichnet, dass es sich um einen DID handelt. Die *DID-Methode* kennzeichnet die Art und Weise, wie ein Identifier erzeugt und verwaltet werden kann. Das W3C führt hierfür eine inoffizielle Liste von DID-Methoden und deren Spezifikationen, die aktuell schon mehr als 80 Einträge umfasst (Steele u. a. 2021). Die Spezifikation einer DID-Methode enthält Angaben zu allen wichtigen *DID-Operationen*, wie das Erstellen, Lesen, Aktualisieren und Löschen von DIDs (CRUD[1]).

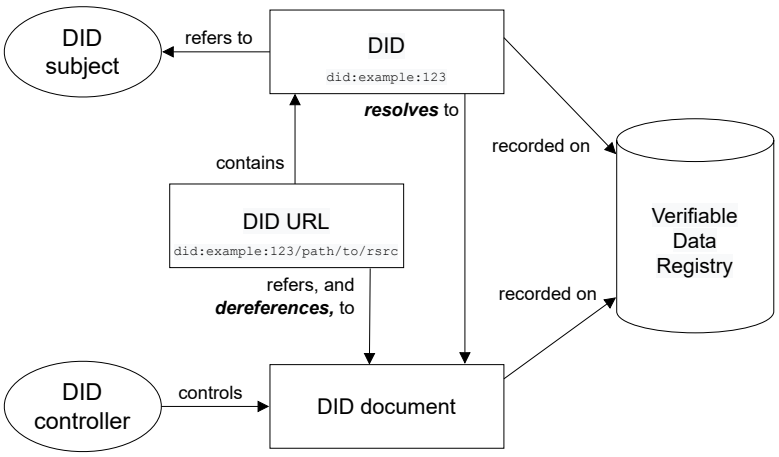

Abbildung 2.1: Grundlegende Komponenten der DID-Architektur (Sporny, Longley, Sabadello u. a. 2021)

[1] Create, Read, Update und Delete bzw. Deactivate (CRUD), auch CDUR oder RUDI (Insert anstelle von Create).

Ein DID wird durch einen *DID-Resolver* in ein *DID-Document* aufgelöst. Dieses enthält zusätzliche Informationen zu einem DID, beispielsweise weitere Öffentliche Schlüssel, Authentifizierungs- und Autorisierungsmethoden oder sogenannte *Service-Endpoints* (vgl. Abbildung 2.1). Letztere sind Angaben zu Diensten oder Speicherorten, die mit dem Identifier verknüpft sein können. Ein DID-Document enthält niemals persönliche Informationen (*Personally Identifiable Information, PII*).

Die DID-Methoden-Spezifikation enthält ebenso alle Informationen, die ein DID-Resolver zur Auflösung eines DIDs benötigt. Das Minimum, was eine DID-Methode beschreiben muss, ist die Art und Weise der Erzeugung des Identifiers. Dies trifft z. B. für die spezielle DID-Methode *Peer-DID* (did:peer) zu, die sich an kein spezielles Netzwerk bindet und für kurzzeitige und bidirektionale Verbindungen entwickelt wird (Deventer u. a. 2021). Die meisten DID-Methoden spezifizieren zusätzlich noch eine Routing-Information, die angibt, in welchem Netzwerk der Identifier zu finden ist (*Verifiable-Data-Registry*). Hier lassen sich dann weitere Informationen, wie z. B. das DID-Document, durch einen DID-Resolver abrufen. Dieser kann dabei lokal oder entfernt ausgeführt werden und seine Informationen aus einer Blockchain, einem Content-Addressable-Storage (CAS) oder einem anderen beliebigen API beziehen.

2.3 Wallets

Als *Wallet* wird jene Software-Komponente bezeichnet, die zum einen eine sichere Verwahrung von Privaten Schlüssel sicherstellt und zum anderen für das Erstellen und den Transport von digital signierten Datenstrukturen zuständig ist, z. B. von Transaktionen in einer Blockchain. Da an ein DID häufig kryptografisches Material geknüpft ist, besitzen auch sie Private Schlüssel, die im Kontext von DIDs ebenfalls in Wallets gespeichert werden. Anstatt (oder zusätzlich zu) Transaktionen, erstellen und signieren sie Verifiable-Credentials (VCs) und Verifiable-Presentations (VPs).

Wallets können grundsätzlich an verschiedenen Orten ausgeführt werden; lokal auf den Endgeräten oder auch entfernt bei einem Online-Dienst. Allerdings gewährleistet nur der Besitz des Privaten Schlüssels die volle Kontrolle über einen DID. Es bedarf zudem weiterer Maßnahmen, die die Privaten Schlüssel im Falle des Verlustes oder Diebstahls wiederherstellen oder zumindest in irgendeiner Form sperren. Einige weit verbreitete (Teil-)Lösungen für den Umgang mit Privaten Schlüsselns sind *HD-Wallets* und *Mnemonics* (Antonopoulos 2015, S. 82 ff.).

Um einen kompromittierten Öffentlichen Schlüssel bzw. Identifier für ungültig zu erklären, oder auf einen neuen Öffentlichen Schlüssel bzw. Identifier zu verweisen, wird eine entsprechende Information in einer Verifiable-Data-Registry hinterlegt. Diesen Vorgang bezeichnet man als *Key-Rotation*. Lösungen hierfür sind beispielsweise *Key-Event-Receipt-Infrastructure (KERI), Peer-DID*, *Sidetree* u.v.m. (Smith 2019; Deventer u. a.

2021; Buchner, Steele und Ronda 2022). Auch Verfahren wie *Secret-Sharing* (z. B. von Shamir) sind denkbare Lösungsansätze, um dem Verlust oder einer Kompromittierung entgegen zu wirken.

2.4 Agents

Ein Agent ist eine Software und beinhaltet Funktionen zur Kommunikation, z. B. für den Austausch von Credentials oder dessen Überprüfung zur Korrektheit mittels Verifizierung von digitalen Signaturen (Buchner 2019). Wie ein Agent arbeitet, wird dabei durch Protokolle festgelegt. So regeln Kommunikationsprotokolle die Interaktionen mit anderen Agents und kryptografische Protokolle bestimmen, wann eine Signatur korrekt ist. Des Weiteren besitzen Agents zumeist Wallet-Funktionalitäten, das heißt, sie dienen auch der Verwaltung kryptografischer Schlüssel. Ein Wallet ist somit Teil eines Agents. Die Verwendung von Agents und damit auch deren Entwicklung geht maßgeblich von der Community im Umfeld von Sovrin bzw. Hyperledger-Aries aus.

2.5 Verifiable-Credentials

Verifiable-Credentials (VCs) sind Datenstrukturen, die nach dem Prinzip der Claim-basierten Identität Aussagen (Claims) über eine Person oder eine Sache treffen und kryptografisch überprüfbar sind. Der aktuell wichtigste Entwurf eines W3C-Standards stellt der von Sporny, Longley und Chadwick (2021) dar. Die Claims werden als einfache Statements mit einem Identifier und einigen Metadaten verbunden und von einem Aussteller (*Issuer*) signiert. Als Issuer kann grundsätzlich jeder auftreten, z. B. eine (vermeintlich) vertrauenswürdige Stelle oder auch die betreffende Entität selbst. Die letztgenannte Art von VCs werden als *Self-issued-Credentials* bezeichnet und sind beispielsweise zur Delegation eines eigenen Credentials an einen anderen Identifier nützlich. Was als vertrauenswürdige Stelle anzusehen ist oder wie Vertrauen zwischen den Teilnehmern hergestellt wird, ist nicht Teil des Entwurfs.

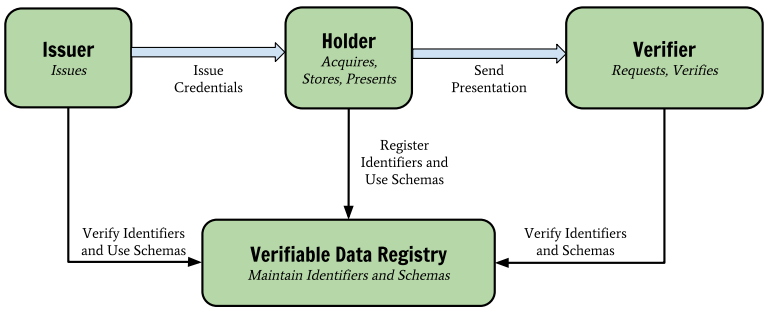

Abbildung 2.2: Das Ökosystem von Verifiable-Credentials

Das Modell des W3C sieht drei Rollen für die Teilnehmer vor, von denen grundsätzlich

Jede von Jedem eingenommen werden kann (vgl. Abbildung 2.2). Der Issuer stellt dabei die kritischste Position dar. Es bedarf weiterer Maßnahmen, die die Glaubwürdigkeit der Issuer sicherstellt.

Die Credentials werden auf ein *Credential-Subject* bzw. dessen Identifier ausgestellt, z. B. einen DID. Der Besitzer des korrespondierenden Privaten Schlüssels wird als *Holder* bezeichnet. Dieser kann aus einem oder mehreren VCs eine *Verifiable-Presentation (VP)* generieren, in denen er durch eine Signatur nachweist, dass er den oder die Identifier eines oder mehrerer VCs kontrolliert. Die Aggregation dieser Informationen drückt typischerweise einen Aspekt einer Person, Organisation oder Einheit aus. Die VP sendet er anschließend an einen *Verifier*, der die Vertrauenswürdigkeit des VCs kryptografisch validieren kann. In einer *Verifiable-Data-Registry* werden ggf. weitere Informationen zum Identifier, zu Credential-Schemas oder zu Credential-Sperrlisten (*Revocation*) veröffentlicht.

2.6 Universal-/App-Links

Universal- oder *App-Links* sind in mobilen Betriebssystemen ein Weg, bestimmte Daten mittels URL von einer App abzufragen. Für Apple's iOS sind sie unter dem Namen Universal-Links dokumentiert, bei Google bezeichnet man sie als Android App Links (*Allowing Apps* 2021; *Handling Android App Links* 2020). Es handelt sich dabei um eine Form von *Deep Links*, die in den Betriebssystemen speziell behandelt werden (vgl. Abbildung 2.3). Während der Installation einer App wird ein *Handler* für einen bestimmten URL im Betriebssystem erstellt, der die App eindeutig identifiziert. Um den Besitz eines URL nachzuweisen, wird auf ein bestimmtes maschinenlesbares Dokument geprüft, das unter der Domain des URL bereitgestellt werden muss. Nachdem diese Prüfung erfolgreich war, werden Anfragen an diesen URL ohne weitere Interaktion des Nutzers direkt an die installierte App weitergereicht (1). Die Bindung an eine Domain stellt die Eindeutigkeit und teilweise auch die Authentizität sicher.

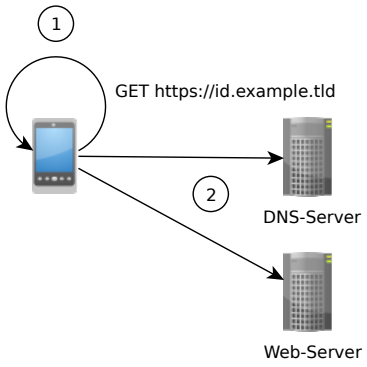

Abbildung 2.3: Universal-/App-Links (eigene Darstellung)

Erfolgt eine Anfrage an den URL und es ist keine entsprechende App installiert, wird

er wie gewöhnlich im Browser geöffnet. Die Anfrage endet also nicht in einem Fehlerzustand, sondern kann vom eigenen Webserver beliebig beantwortet werden (2). Der Anwender bekommt beispielsweise einen Hinweis zur Installation der entsprechenden App oder wird zu einer Online-Anwendung umgeleitet.

Die Anfrage selbst wird im URL als Parameter codiert und per HTTP(S) übertragen: https :// id . example.tld?Request (*HTTP-Request*). Die Antwort darauf liefert z.B. die angefragten Daten oder auch eine Fehlerinformation zurück (*HTTP-Response*).

3 Stand der Technik

3.1 Schichtenmodelle

Ausgangspunkt dieser Arbeit als auch der des Whitepapers der Trust-Over-IP-Foundation ist das Dokument „0289: The Trust Over IP Stack“ in den *Requests-for-Comments (RFC)* von *Hyperledger-Aries* (vgl. *Introducing the Trust Over IP Foundation* 2020, S. 1; Davie u. a. 2019; vgl. Preukschat u. a. 2021, S. 89 ff.). Hierin wird eine Architektur beschrieben, die Vertrauen sowohl auf der Maschinenebene mittels Kryptographie, als auch auf der menschlichen Ebene im Sinne der Geschäftsfähigkeit, Gesetzgebung und sozialer Aspekte, schaffen soll. Als Referenz wird hierbei auf das TCP/IP-Modell sowie den SSL/TLS-Standard Bezug genommen und die Notwendigkeit mit dem Fehlen eines *Identity-Layers* im Internet begründet. Diese Lücke soll nun mittels *Trust-over-IP (ToIP)* geschlossen werden, indem eine standardisierte Netzwerkarchitektur für vertrauensvolle Netzwerkverbindungen geschaffen wird.

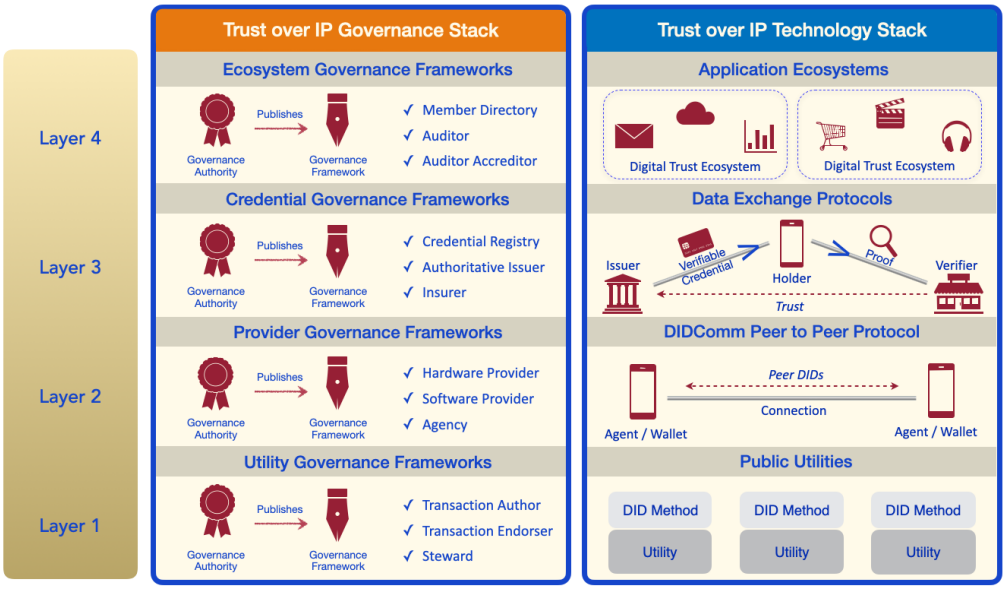

Abbildung 3.1: Das Trust-over-IP-Schichtenmodell (Davie u. a. 2019)

Das Schichtenmodell von ToIP besteht aus vier Schichten. Jede dieser Schichten unterteilt sich in organisatorische und technologische Aspekte: auf der einen Seite der *ToIP-Governance-Stack* und auf der Anderen der *ToIP-Technology-Stack* (vgl. Abbildung 3.1). Der Grund für diese Trennung besteht darin, dass Vertrauen in der digitalen Welt nicht allein durch Technologie hergestellt werden kann. Es benötigt immer auch Menschen, die die dafür notwendigen technischen Geräte betreiben; sowie Regeln (Governance), nach denen sie handeln. Es wird darauf hingewiesen, dass das Schichtenmodell keine konkreten Richtlinien für eine Governance-Vereinbarung aufstellt, sondern

lediglich als Rahmen (Framework) zum Entwurf und zur Implementierung solcher Richtlinien genutzt werden kann.

Die erste Schicht wird auch als *Public-Utilities* bezeichnet, deren Hauptaufgabe in der Unterstützung der DID-Architektur liegt (vgl. Absatz 2.2). Dies sind vor allem verteilte Systeme wie Blockchains oder Distributed-Ledgers, die sich nach Aussage der Autoren durch ihre hohe Verfügbarkeit und kryptografische Überprüfbarkeit für einen „strengen Root-of-Trust“ eignen (vgl. Davie u. a. 2019). In der DID-Spezifikation werden diese Systeme als *Verifiable-Data-Registry* bezeichnet (vgl. Sporny, Longley, Sabadello u. a. 2021). Anhand der DID-Methode lässt sich eine konkrete Verifiable-Data-Registry eindeutig identifizieren. Neben Blockchains und Distributed-Ledgers sind in den DID-Methoden-Spezifikationen auch andere Systeme dokumentiert, die als Verifiable-Data-Registry dienen können. Beispiele hierfür sind verteilte Dateisysteme wie das *Inter-Planetary-File-System (IPFS)* oder Peer-to-Peer-Netzwerke wie *Git*, *KERI*[2] oder *Peer-DID* (vgl. Davie u. a. 2019; vgl. Preukschat u. a. 2021, S. 92, 95 ff.).

Public-Utilities können ein beliebiges Governance-Modell wählen und ihre Anforderungen an das jeweilige Geschäftsmodell, den Rechtsrahmen und die technische Architektur anpassen. Konformität nach dem ToIP-Governance-Stack erfordert, dass „Interoperabilität und transitives Vertrauen“ sichergestellt sind.[3] Dazu muss klar erkennbar sein, wer die Governance-Richtlinien verantwortet, wo sie einsehbar sind und wer die teilnehmenden Knoten oder Betreiber sind. Außerdem müssen die Knoten und/oder Dienstendpunkte eindeutig aufzufinden sowie Sicherheits-, Datenschutz- und andere Betriebsrichtlinien frei zugänglich sein (vgl. Davie u. a. 2019).

Governance-Frameworks, die dem ToIP-Governance-Stack entsprechen, enthalten Standardrollen für alle Arten von Governance-Autoritäten des entsprechenden Public-Utilities. Zum Beispiel umfassen die Rollen, die derzeit von Public-Permissioned-Utilities unterstützt werden (basierend auf *Hyperledger-Indy*), sogenannte *Transaction-Authors*, *Transaction-Endorsers* und *Stewards*. Für andere Arten von Public-Utilities können ggf. andere Rollen festgelegt werden.

Neben der Verwaltung von DIDs und deren DID-Documents können Public-Utilities zudem zur Verwaltung weiterer Informationen genutzt werden, die höhere Schichten betreffen. Als Beispiele werden hierzu *Schemata* und *Definitionen* für Credentials, *Revocation-Registries* (vgl. Zertifikatssperrlisten) oder auch *Agent-Authorization-Policies* (Sperrlisten für Agents) genannt. Nach Ansicht der Autoren wird Interoperabilität durch den W3C-DID-Standard in Verbindung mit den Aries-RFCs, die die vorgenannten Datenstrukturen definieren, gewährleistet (vgl. Davie u. a. 2019; vgl. Preukschat u. a. 2021, S. 94).

[2] Key-Event-Receipt-Infrastructure (KERI), vgl. Abschnitt 3.2.

[3] Transitives Vertrauen im Sinne einer transitiven Relation bedeutet: wenn A vertaut B und B vertraut C, dann vertaut A auch C.

Die zweite Schicht definiert sich vor allem über eine Reihe von Standards des *DIDComm-Messagings* (vgl. Davie u. a. 2019). Damit soll eine sichere Kommunikation zwischen zwei Agents gewährleistet werden, die von Protokollen der dritten Schicht genutzt werden kann. Es existieren zwei Linien von Spezifikationen, *DIDComm-V1* und *DIDComm-V2*, die sich in ihren Konzepten sehr ähnlich sind, sich aber in einigen Details unterscheiden. Erstere entstammt dem Umfeld von Hyperledger-Aries und ist in mehreren Aries-RFCs spezifiziert (vgl. Hardman 2019). Die Zweite wird aktuell durch eine Arbeitsgruppe der Decentralized-Identity-Foundation (DIF) weiterentwickelt (Curren u. a. 2021).

Für die gegenseitige Authentisierung der Verbindungen nutzt DIDComm in vielen Fällen das Prinzip der *Pairwise-Pseudonymous-Peer-DIDs*, wie es von Deventer u. a. (2021) beschrieben wird. Für jede Verbindung werden neue DIDs berechnet, die zusammen mit den DID-Documents über das *DID-Exchange-Protocol* ausgetauscht werden und fortan eine sichere und private Verbindung ermöglichen. Die meisten DIDs werden aus Schlüsselpaaren gebildet, deren Private Schlüssel im lokalen *Key-Management-System (KMS)* gespeichert und verwaltet werden. Ein KMS wird auch als *Wallet* bezeichnet und ist Teil eines Agents (vgl. Davie u. a. 2019).

Das Prinzip der Pairwise-Pseudonymous-Peer-DIDs ist grundsätzlich nicht auf die Public-Utilities der ersten Schicht angewiesen. In den meisten Fällen wird dies auch gar nicht benötigt und kann hinsichtlich des Datenschutzes und der Datensicherheit sogar problematisch sein. Als allgemeine Regel nennen die Autoren zwei Akteure, für die eine Registrierung in Schicht eins notwendig Ist:

- *Credential-Issuers* (Schicht 3) und
- *Governance-Authorities* (alle Schichten).

Ebenfalls der Schicht zwei zuzuordnen ist die Idee eines *Secure-Data-Stores*. Dabei handelt es sich um einen Speicher bzw. eine Datenbank, die Teil eines Agents ist und bestimmten Anforderungen unterliegt. So wird gefordert, dass die Kontrolle darüber ausschließlich beim DID-Controller liegt; also demjenigen, der über den entsprechenden Privaten Schlüssel in seinem KMS verfügt. Mit diesem Privaten Schlüssel sollen weiterhin sämtliche Inhalte in diesem Speicher verschlüsselt werden. Außerdem soll gewährleistet sein, dass der Nutzer bei Verwendung mehrerer Secure-Data-Stores eine automatisierte Synchronisierung zwischen diesen aktivieren kann. Um die Standardisierung dieser Bestrebungen bemüht sich die *Secure-Data-Storage-Working-Group* innerhalb der DIF (vgl. Davie u. a. 2019; *DIF - Secure Data Storage Working Group* 2021).

Agents und deren integrierte Wallets bzw. KMS sind zentrale Bestandteile einer Software über alle Schichten hinweg. Sie speichern und verwalten Private Schlüssel und signieren Datenstrukturen, wie z. B. Transaktionen, Verifiable-Credentials oder Verifiable-Presentations. Die Anforderungen an die Sicherheit können dabei ganz unterschiedlich

sein. Für höhere Sicherheit sind Hardware-basierte Lösungen verfügbar; bei geringeren Anforderungen können auch Software-basierte Lösungen mit Datei- bzw. Datenbankverschlüsselung bevorzugt werden. Über die Speicherung und Verwaltung von Privaten Schlüsseln hinaus geht die Aufgabe der Wiederherstellung jener, z. B. bei Verlust, Diebstahl oder Kompromittierung eines Endgerätes. Diese Problematik betrifft demnach ebenfalls alle Schichten. Einige wichtige Gedanken hierzu führen Reed u. a. (2019) auf.

Ein weiterer Aspekt der Schicht zwei ist der Zugang zu Agents. Es sind auch die Nutzer zu berücksichtigen, die keinen unmittelbaren Zugang zu geeigneten Endgeräten haben oder die über keinen eigenen Internetanschluss verfügen. Dieses Konzept wird als *Digital-Guardianship* bezeichnet und meint z. B. Dienste, die Agents in ihrer Infrastruktur für ihre Kunden bereitstellen. Darüber hinaus sind darunter auch Verantwortungsbeziehungen zu verstehen, die z. B. Minderjährige, Tiere oder Dinge betreffen und in Verifiable-Credentials ausgedrückt werden können (vgl. Davie u. a. 2019; vgl. Preukschat u. a. 2021, S. 233 f., 110 ff.).

Die Governance in dieser Schicht bezieht sich vor allem auf die Hersteller von Hard- und Software sowie die Betreiber von Infrastrukturdiensten. Insbesondere werden Anforderungen für Zertifizierungen und an Tests zur Interoperabilität definiert, die die Mindestanforderungen an Sicherheit, Datenschutz und Datensicherheit festlegen (vgl. Davie u. a. 2019).

Die Schichten eins und zwei stellen die technische Authentifikation und Kommunikation sicher. Dies wird auch als *Cryptographic-Trust* oder *Technical-Trust* bezeichnet. Dies ist jedoch nicht ausreichend, um den Kommunikationspartner tatsächlich zu identifizieren. Das *Verifiable-Credentials-Data-Model* in Schicht drei soll genau dieses Problem lösen (Sporny, Longley und Chadwick 2021). Die Datenstrukturen in diesem Modell, genauer *Verifiable-Credentials (VC)* und *Verifiable-Credentials (VP)*, beschreiben Eigenschaften aus der realen Welt und werden durch Dritte digital signiert, die die realen Eigenschaften prüfen und sie durch ihre Signatur kryptografisch überprüfbar bestätigen. Diese dritte Instanz wird als *Credential-Issuer*, oder auch einfach als *Issuer,* bezeichnet und soll den sogenannten *Human-Trust* herstellen (vgl. Davie u. a. 2019; vgl. Preukschat u. a. 2021, S. 104 ff.).

Agents tauschen VCs und VPs über verschiedene Protokolle der dritten Schicht aus, die ihrerseits das DIDComm-Protokoll in Schicht zwei verwenden können. Einige dieser Protokolle sind als Teil der *DIDComm-Suite* veröffentlicht (vgl. Davie u. a. 2019; Hardman 2019). Je nach Signaturverfahren können die Anforderungen an den *Request* und die *Response* eines solchen Protokolls sehr unterschiedlich sein, sodass für jedes Signaturverfahren jeweils eine eigene Variante benötigt wird. Das Ziel des ToIP-Technologie-Stacks ist die Standardisierung aller unterstützten Protokolle für den Austausch von Berechtigungsnachweisen, sodass jeder ToIP-kompatible Agent, jedes

Wallet und jeder Secure-Data-Store mit jedem anderen Agent, Wallet und Secure-Data-Store interoperieren kann (vgl. Davie u. a. 2019).

In Schicht drei ist Governance eine entscheidene Komponente für Interoperabilität. Es müssen Regeln festgelegt werden, wie die Prüfung realer Eigenschaften erfolgt und welchen Sicherheitsanforderungen diese genügen müssen, sowohl organisatorisch als auch technisch. Credential-Governance-Frameworks spezifizieren z. B.:

- Schema-Definitionen für Credentials.
- Regeln, die festlegen, wer als maßgeblicher Aussteller für diese Credentials dienen kann.
- Richtlinien, die diese Aussteller befolgen müssen, um diese Credentials auszustellen und zu widerrufen.
- Anwendbare Geschäftsmodelle, Haftungsregelungen und Versicherungsmodelle

und können Standardrollen definieren, die u. a.:

- *Authoritative-Issuers* (Aussteller, die von der Aufsichtsbehörde ermächtigt sind, bestimmte Arten von Berechtigungsnachweisen mit bestimmten Sicherheitsstufen auszustellen)
- *Credential-Registries* (alternative Inhaber von Credentials zur Unterstützung anderer Zwecke, z. B. öffentlich durchsuchbare Verzeichnisdienste)
- *Insurers* (versichern Aussteller, die gemäß den Bedingungen des Governance-Frameworks arbeiten)

umfassen. Schicht drei ermöglicht es, menschliches Vertrauen – in Form von überprüfbaren Aussagen über Entitäten, Attribute und Beziehungen – über das kryptografische Vertrauen der Schichten eins und zwei zu spannen.

In Schicht vier sind die Software- bzw. Anwendungs-Ökosysteme, die überprüfbare Nachweise anfordern und verarbeiten, um so die spezifischen Vertrauensmodelle und Richtlinien ihres eigenen digitalen Vertrauensökosystems zu erfüllen (vgl. ebd.). In dieser Schicht interagiert der Nutzer direkt mit einer Software und die Anwendungen greifen über den ToIP-Stack auf die unteren Schichten drei bis eins zu (vgl. TCP/IP-Stack).

> The ToIP stack simply defines the „tools and rules“ – technology and governance – for those applications to interoperate within digital trust ecosystems that provide the security, privacy, and data protection that their members expect (ebd.).

Darüber hinaus soll das Benutzererlebnis durch diese Standardisierungsbestrebungen weitgehend vereinheitlicht werden, um Nutzern den sicheren Umgang und bewusste Entscheidungen der Vertrauenswürdigkeit über die verschiedenen Anwendungen und

Ökosysteme hinweg zu ermöglichen. Als Analogie wird hierfür das Cockpit von Kraftfahrzeugen genannt. Die verschiedenen Fahrzeughersteller unterscheiden sich in ihren Produkten in vielen Details, aber die Anordnung und die Funktionsweise von z. B. Lenkrad, Gaspedal, Bremse oder Blinker ist weitestgehend gleich.

In der vierten Schicht kommen die Nutzer also direkt mit dem ToIP-Stack in Berührung. Insbesondere sind für sie die Vertrauens- und Grundsatzerklärungen im Governance-Framework des zugrundeliegenden Anwendungs-Ökosystems von Bedeutung, die für alle Schichten des ToIP-Stacks gelten. Speziell für die vierte Schicht kann ein Governance-Framework weitere Standardrollen definieren, wie z. B.

- *Member-Directories*, die sowohl von Menschen als auch von Maschinen durchsuchbare Auflistungen der öffentlichen DIDs und andere durchsuchbare Attribute der Teilnehmer beinhalten
- *Auditors*, die Teilnehmern im Ökosystem hinsichtlich der Einhaltung der Governance-Richtlinien prüfen können
- *Accreditors*, die Prüfer akkreditieren können, wenn ein Governance-Framework auf nationaler oder globaler Ebene skalieren muss.

Das Schichtenmodell erweitert die Vertrauensbeziehungen des Verifiable-Credentials-Data-Models zwischen Issuer, Holder und Verifier um eine weitere Instanz, der *Governance-Authority* (vgl. Abschnitt 2.5). Diese Instanz verantwortet und veröffentlicht das Governance-Framework, denen die Teilnehmer unterliegen; allem voran die Issuer. Das Vertrauensverhältnis zum Issuer wird also auch dadurch bestimmt, dass ein Vertrauensverhältnis zum Governance-Framework des entsprechenden Ökosystems existiert (vgl. Davie u. a. 2019; vgl. Preukschat u. a. 2021, S. 36f, 110 ff.). In Abbildung 3.2 wird dies durch das untere Vertrauensdreieck dargestellt.

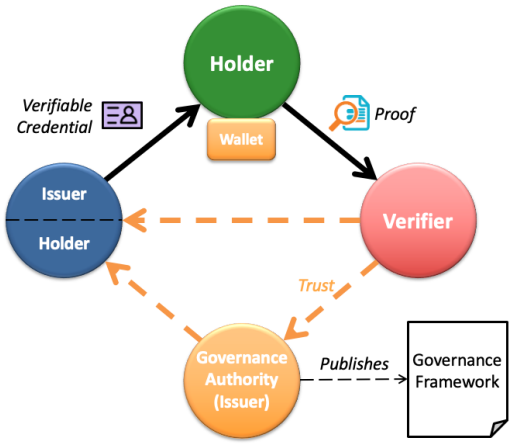

Abbildung 3.2: Vertrauen und die Rolle der Governance Authorities (Davie u. a. 2019)

Wie auch in der analogen Welt, in der nach einem vergleichbaren Governance-Modell Pässe, Führerscheine, Kreditkarten oder Krankenversicherungskarten ausgestellt wer-

den, können dieselben Instanzen, oder eigens für einen bestimmten Zweck Geschaffene, *digitale* Governance-Frameworks entwickeln und veröffentlichen – Privatunternehmen, Industriekonsortien, Finanznetzwerke oder auch Regierungen.

Der ToIP-Stack ist so konzipiert, dass er mit amtlichen Governance-Frameworks kompatibel ist. Als Beispiel hierfür wird das *Pan-Canadian Trust Framework (PCTF)* vom Authentication Council of Canada (DIACC) genannt. Auch andere lokale und regionale Regelungen sollen vereinbar sein (vgl. Davie u. a. 2019). In Europa ist insbesondere die eIDAS-Verordnung[4] des Europäischen Parlaments und des Rates relevant. Darüber hinaus soll es möglich sein, Verbindungen unter verschiedenen digitalen Vertrauensökosystemen herzustellen, also ein Vertrauensökosystem von Vertrauensökosystemen. Die Grenze des Vertrauens eines jeden Ökosystems ist durch das jeweilige Governance-Framework definiert, dem die Teilnehmer unterworfen sind. Diese Grenzen sollen nun genauso einfach überwunden werden können, wie im TCP/IP-Stack Netzwerkgrenzen überwunden werden können (vgl. Davie u. a. 2019).

Neben dem ToIP-Stack findet sich eine weitere Quelle zu einem Schichtenmodell. Das Ergebnis eines ersten Aufschlags während des 27. Internet Identity Workshops (IIW) dokumentiert ein Modell mit elf Schichten. Dieses wird aber mittlerweile als zu feingranular angesehen und nicht weiter verfolgt (Preukschat u. a. 2021, S. 89 f. vgl. Oliver Terbu 2019).

3.2 Standards und Spezifikationen

Die beiden dem W3C zur Standardisierung vorgeschlagenen Spezifikationen zu den Decentralized-Identifiers (DIDs) und Verifiable-Credentials (VCs) bilden die Grundlage zur Gestaltung von technischen System, die dem Paradigma von Self-Sovereign-Identity (SSI) folgen und dezentralisierte Identitäten erlauben sollen. Innerhalb des DID-Standards werden die unterschiedlichen DIDs in sogenannten DID-Methoden dokumentiert und in einer offiziellen Liste des W3C verwaltet (Steele u. a. 2021). Darüber hinaus existieren eine Reihe weiterer Spezifikationen und Vorschläge, die verschiedene Aufgaben auf den jeweiligen Schichten erfüllen sollen. Diese werden insbesondere bei der Decentralized-Identity-Foundation (DIF) gesammelt und der Öffentlichkeit zur Verfügung gestellt. Auch das W3C erfüllt eine solche Aufgabe. Weitere Vorschläge und Spezifikationen lassen sich im Umfeld von *Hyperledger-Indy* und *Hyperledger-Aries* sowie *Ethereum* ausmachen. Andere wiederum lassen sich keinem speziellen Umfeld zuordnen. Als Orientierung dient eine Übersicht der DIF, die aus einer Präsentation von Rouven Heck, Executive Director der DIF, des 30. Internet Identity Workshops (IIW) stammt (vgl. Abbildung 3.3). Einige der wichtigsten Spezifikationen, die in dieser Übersicht dargestellt sind, werden im Folgenden näher betrachtet.

[4] engl. *electronic Identification, Authentication and trust Services*, kurz eIDAS, Verordnung (EU) Nr. 910/2014 (2014).

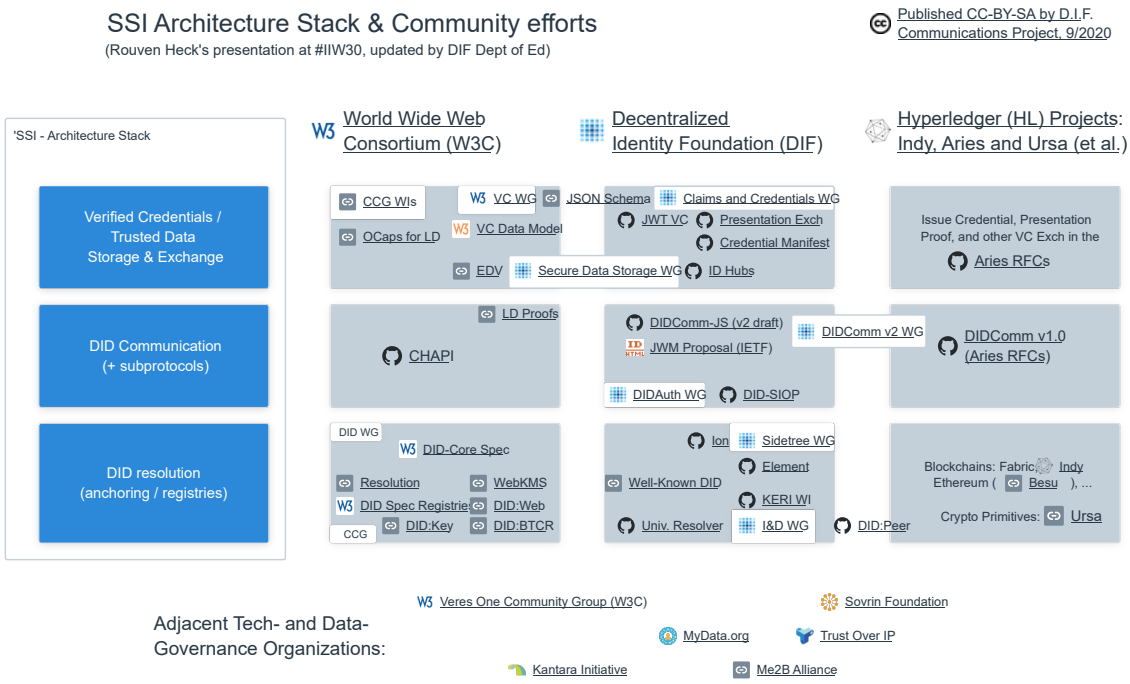

Abbildung 3.3: Übersicht von Rouven Heck, DIF (Heck 2020)

Die *BTCR-DID-Methode* (did:btcr, engl. Bitcoin Reference DID method) nutzt die Bitcoin-Blockchain als Public-Utility (Allen u. a. 2019). Anders als bei vielen anderen DID-Methoden bildet BTCR den Identifier nicht aus einem Schlüsselpaar, sondern nutzt ein Verfahren, das die Position einer Transaktion in der Blockchain codiert. Dabei handelt es sich um *Bech32-Encoded-Tx-Position-References (TxRef)*, die im BIP[5] 0136 beschrieben sind (Schnelli u. a. 2021). Eine TxRef codiert die Chain-ID, die Blockhöhe, den Index der Transaktion und optional den Index eines Outputs innerhalb einer Transaktion. Standardmäßig wird der Index 0 angenommen. Da der Identifier die Position einer Transaktion in der Blockchain beschreibt, muss zunächst eine Transaktion in einem Block bestätigt werden, bevor der Identifier genutzt werden kann. Ein DID-Resolver kann dann ein DID-Document erstellen, das den Öffentlichen Schlüssel dieser Transaktion beinhaltet. Eine Transaktion kann auch ein OP_RETURN-Feld enthalten; mit einem URL, der auf ein Dokument mit weiteren Attributen für das DID-Document verweist (Duffy u. a. 2018). Für die Transaktion werden zwei Schlüsselpaare benötigt: Eines, um die Transaktion zu signieren, und ein Anderes für die Adresse, die im Transaction Output verwendet wird (vgl. Wechselgeldadresse). Dies kann für eine sogenannte *Key-Rotation*, oder genauer *Key-Pre-Rotation*, genutzt werden (vgl. KERI). Das heißt, geht das Schlüsselpaar verloren, das die erste Transaktion signiert hat, dient das zweite Schlüsselpaar dazu, den *Unspent-Transaction-Output (UTXO)* zu entsperren („auszugeben“) und damit in einer zweiten Transaktion die nächste Adresse für die Wiederherstellung bzw. Key-Rotation im Transaction-Output festzulegen. Für diese Adresse wird im Moment der Wiederherstellung ein drittes Schlüsselpaar erzeugt. Sobald die zweite Transaktion den UTXO der ersten Transaktion referenziert, wird

[5] Bitcoin Improvement Proposal (BIP)

ein DID-Resolver das erste Schlüsselpaar für ungültig erklären und dem Transaction Output zur nächsten Transaktion folgen. Enthält diese keinen UTXO am Index 0 (oder dem angegebenen Index), wiederholt sich dieser Vorgang mehrfach. Der Öffentliche Schlüssel aus dem ersten Input der Transaktion mit dem UTXO wird nun für das DID-Document verwendet. Auf diese Weise lässt sich das Schlüsselmaterial für diesen Identifier austauschen (vgl. Abbildung 3.4). Die BTCR-DID-Methoden-Spezifikation ist seit 2019 in einem unfertigen Zustand, sodass davon ausgegangen werden kann, dass diese DID-Methode nicht weiter verfolgt wird (Allen u. a. 2019).

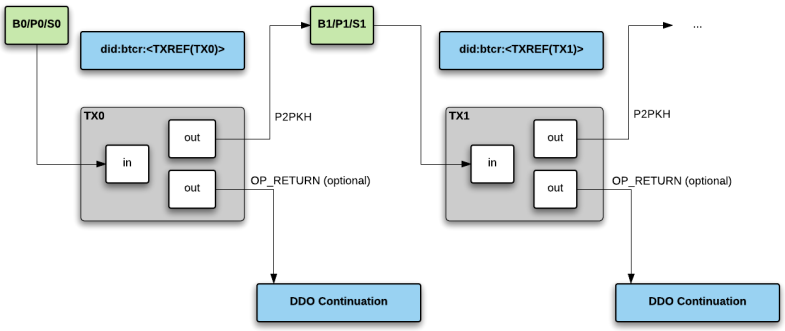

Abbildung 3.4: Das BTCR-Transaktions-Strukturdiagram (Allen u. a. 2019)

Die *Key-DID-Methode* (did:key) ist eine einfache DID-Methode, deren Identifier aus Schlüsselpaaren gebildet werden und für die kein Public-Utility vorgesehen ist (Longley, Zagidulin u. a. 2021). Das hat zur Folge, das von den üblichen DID-Operationen nur „Create“ und „Read“ unterstützt werden (vgl. Abschnitt 2.2). Es besteht keine Möglichkeit, den Öffentlichen Schlüssel eines Identifiers zu ändern oder für ungültig zu erklären (Key-Rotation). Die Key-DID-Methode und deren Identifier sind in erster Linie dafür gedacht, Endpunkte von einzelnen, kurzlebigen Verbindungen gegenseitig zu authentifizieren (vgl. Peer-DID). Bemerkenswert ist, dass im Identifier auch das Verfahren codiert ist, mit dem der Öffentliche Schlüssel vom Privaten Schlüssel abgeleitet wurde (*Multicodec* 2022).

Die *Peer-DID-Methode* (did:peer, engl. Peer DID Method Specification) basiert auf Key-DID, unterstüzt darüber hinaus aber auch die DID-Operationen „Update“ und „Delete“ (Deventer u. a. 2021). Peer-DID verzichtet ebenfalls auf ein Public-Utility in Form eines Distributed-Ledgers oder anderen Peer-to-Peer-Systemen (P2P) wie Git oder IPFS. Stattdessen ist es ein Teil eines beliebigen (P2P-) Netzwerkes, das sich Nachrichten mit Fragmenten von DID-Documents, z.B. mittels DIDComm, austauscht. Die Kommunikation findet dabei jeweils nur zwischen den Knoten statt, die tatsächlich daran teilhaben wollen. Es werden drei Anwendungsklassen von DIDs unterschieden: *Anywise-DIDs* (Anzahl der Teilnehmer unbekannt), *Pairwise-DIDs* (genau zwei Teilnehmer) und *N-wise-DIDs* (bekannte Anzahl von Teilnehmern). Anywise-DIDs müssen grundsätzlich für alle Teilnehmer auflösbar sein, das heißt, sie müssen öffentlich gefunden werden können, z. B. auf Websites, in jeglicher Form eines Verzeichnisses, oder in einem Distributed-Ledger. Bei den anderen beiden genügt es, wenn sich die jeweiligen Partei-

en (*Peers*) gegenseitig finden können. Der methodenspezifische Identifier wird auf die gleiche Weise wie ein Key-DID gebildet. Allerdings fließt bei Peer-DID nicht nur der Öffentliche Schlüssel eines Schlüsselpaares allein ein, sondern jener als Teil eines DID-Documents. Da das DID-Document in die Berechnung des Identifiers einfließt, kann dieser selbst kein Teil des gleichen DID-Documents sein. Er ist zu diesem Zeitpunkt noch nicht bekannt. Ein DID-Resolver muss dieses Feld später einfügen, um ein konformes DID-Document zu erzeugen. Den Identifier kann er aus der ersten Version des DID-Documents (*Genesis document*) berechnen. DID-Operationen werden dann mittels signierter Datenstrukturen in Deltas aufgezeichnet und untereinander ausgetauscht (vgl. CRDT,[6] vgl. KERI). Auf diese Weise können weitere Schlüssel und Attribute zum DID-Document hinzugefügt und auch wieder entfernt werden, was sich auch für Key-Rotations eignet.

Die KERI-DID-Methode (did:keri) mit ihrer *Key-Event-Receipt-Infrastructure (KERI)* beschreibt ein System zur dezentralen Schlüsselverwaltung (KMS/DKMS) mittels Key-Pre-Rotation (Smith 2019; Smith u. a. 2021). Ähnlich wie bei der BTCR-DID-Methode wird auch hier ein zweites Schlüsselpaar erzeugt, das für Wiederherstellungszwecke bzw. eine Key-Rotation berechtigt ist. Unmittelbar nach der Erstellung des ersten Schlüsselpaares wird eine Key-Rotation durch den *Controller* selbst durchgeführt, bei der der Identifier des zweiten Schlüsselpaares in eine signierte Datenstruktur aufgenommen wird, die als *Inception-Event* bezeichnet und durch das erste Schlüsselpaar signiert wird. Der Identifier ist dabei eine Art *Commitment* des Öffentlichen Schlüssels, dessen Privater Schlüssel die nächste Key-Rotation durchführen darf bzw. welcher Private Schlüssel die Signatur für das nächste *Key-Event* erstellen muss. Mit jeder Key-Rotation wird ein Key-Event erstellt und darin ein neuer Identifier für die nächste Key-Rotation festgelegt. Jedes Key-Event enthält außerdem den Hash bzw. *Digest* des vorangegangenen Events und bildet damit eine sortierte, Hash-verkettete Liste, das *Key-Event-Log (KEL)*. Integrität und Authentizität sind durch die Signaturen sichergestellt (vgl. Blockchain). Die Key-Events werden anschließend regelmäßig mit anderen *Entities* (Peers) ausgetauscht. Dabei wird zwischen dem Online- und dem Offline-Fall unterschieden. Ersterer ist eine interaktive Verbindung zum eigentlichen Kommunikationspartner (pairwise, one-to-one); einem anderen Peer, zu dem die Key-Events übermittelt werden sollen. Dieser überprüft die Signaturen der Key-Events und speichert sie in einem KEL. In dieser Funktion wird der Peer als *Validator* bezeichnet. Eine zweite Funktion ist das Erstellen von *Key-Event-Receipts*. Dabei handelt es sich um Datenstrukturen, die ein Key-Event beinhalten und von diesem Peer in der Funktion als *Wittness* signiert werden. Die Key-Event-Receipts sendet der Peer zurück an den Sender der Key-Events und speichert sie selbst im *Key-Event-Receipt-Log (KERL)* ab. Auch der Sender, der inzwischen die Key-Event-Receipts empfangen hat, prüft und speichert sie in einem eigenen KERL ab (Smith 2019, S. 6 ff.). Nach dem gleichen Protokoll läuft auch in der Gegenrichtung der Austausch der Key-Events ab (vgl. Abbildung 3.5).

[6] Conflict-free Replicated Data Type (CRDT)

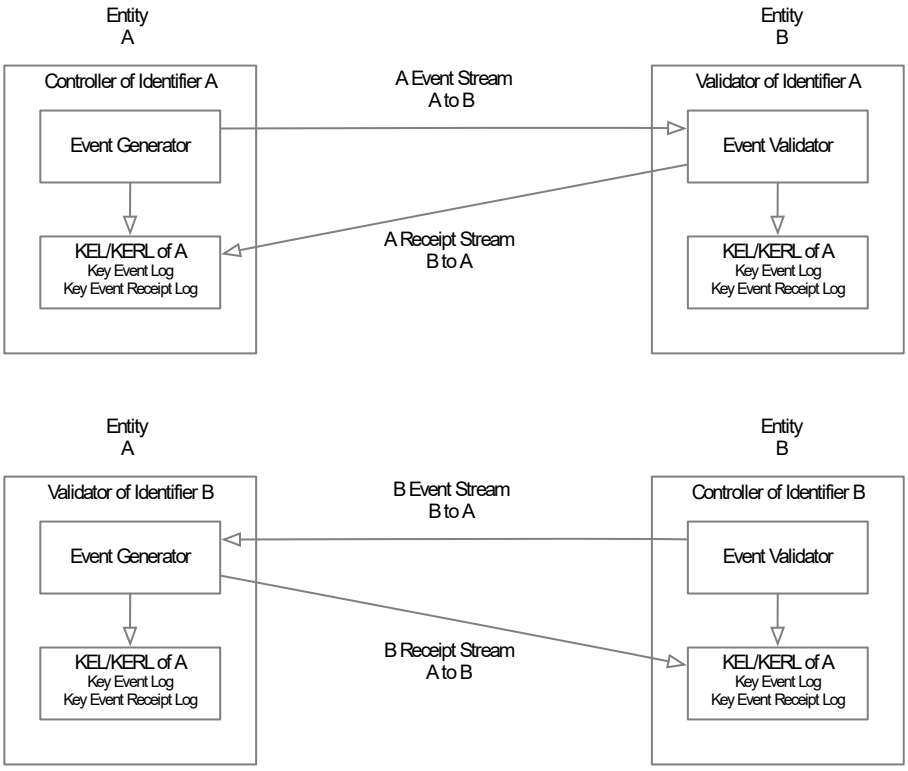

Abbildung 3.5: Online-bidirektionale Verbindungen in KERI (Smith 2019, S. 10)

Der Offline-Fall bedeutend, dass ein Validator vorübergehend keine Key-Events empfangen kann. Mögliche Ursachen können Netzwerkstörungen, Dienstausfälle oder Energiesparfunktionen auf mobilen Endgeräten sein. Um dennoch sicher zu gehen, dass ein Key-Event andere Peers unabhängig vom eigenen Online-Status erreicht, werden weitere Witnesses einbezogen (one-to-one, one-to-many/anywise). Die Bereitstellung derartiger Witnesses wird z. B. durch vertrauenswürdige Dienstleister unterstützt. Die Hauptaufgabe ist dabei die Überprüfung der Signaturen sowie der Integrität der Key-Events in den *Key-Event-Messages*. Beim Austausch der Nachrichten gilt das *First-Seen-Prinzip*, das heißt, ein Witness signiert nur das erste gültige Key-Event zu einem Key-Event-Receipt, das ihn zuerst erreicht. Eine besondere Art von Witnesses sind sogenannte *Judges*. Sie prüfen die Signaturen aller Key-Events und Key-Event-Receipts in einem KERL sowie dessen Vollständigkeit. Vollständig bedeutet, dass eine vorher vom Validator festgelegte Anzahl von Witnesses, sowie der Controller selbst, entsprechende Events signiert haben (Smith 2019, S. 10 ff.). Die Identifier basieren in KERI ebenfalls auf Schlüsselpaaren. Das Codierverfahren ist zu dem von *Multicodec* (2022) sehr ähnlich. Es werden ebenfalls mehrere Verfahren zur Erzeugung der Identifier und zur Signaturerstellung unterstützt und im Identifier gekennzeichnet (*KID0001* 2021).

Die *ETHR-DID-Methode* (did:ethr) nutzt einen *Smart-Contract* auf Basis von Ethereum als Public-Utility, der in EIP-/ERC-1056[7] als *Ethereum-Lightweight-Identity* dokumentiert ist (*ETHR-DID* 2021; Braendgaard u. a. 2018). Dieser Smart-Contract kann u. a.

[7] Ethereum Improvement Proposal / Ethereum Request for Comment, Nummer 1056

dazu genutzt werden, einfache Key-Rotations durchzuführen. Das heißt, dem Controller eines Identifiers bzw. einer Ethereum-Adresse ist es durch seine Signatur erlaubt, einen neuen Identifier im Smart-Contract festzulegen. Dieser Identifier repräsentiert dann ein neues Schlüsselpaar für den ursprünglichen Identifier. Ein Verifier nutzt eine andere Funktion, um den letzten Identifier für einen rotierten Identifier abzurufen. Existiert kein Eintrag für diesen, hat keine Rotation stattgefunden und die Funktion liefert den selben Identifier zurück. Gegen den abgerufenen Identifier vergleicht der Verifier anschließend einen Berechneten. Aus der Signatur einer Nachricht lässt sich mit dem in Ethereum verwendeten Signaturverfahren ECDSA[8] der Öffentliche Schlüssel berechnen. Die Signatur wird für diesen Öffentlichen Schlüssel geprüft und die Ethereum-Adresse berechnet, die gleichzeitig der Identifier ist. Das Verfahren zur Berechnung von Ethereum-Adressen ist bekannt. Stimmen abgerufener und berechneter Identifier überein, sind Integrität und Authentizität der Nachricht sichergestellt.

Identifier werden in ERC-1056 nicht generell im Smart-Contract gespeichert. Es werden keine Key-Pre-Rotations im Sinne derer von BTCR-DID oder KERI für jeden einzelnen Identifier durchgeführt. Daher sind sie ausdrücklich auch für eine Off-Chain-Nutzung geeignet, ähnlich zu der von Peer-DID oder Key-DID (*ETHR-DID* 2021). Der Smart-Contract implementiert des Weiteren eine Funktion, die eine bestimmte signierte Nachricht für eine Key-Rotation akzeptiert. Dieser Mechanismus kann zum einen dazu genutzt werden, die Kosten der Ethereum-Transaktion zu delegieren und zum anderen eine Key-Rotation ohne Kenntnis des Privaten Schlüssels durchzuführen, vorausgesetzt die signierte Nachricht befindet sich an einem sicheren und für den Eigentümer zugänglichen Ort. Dies ist annähernd eine Key-Pre-Rotation, hat aber neben den Vorteilen der Off-Chain-Nutzung auch einen Nachteil. Wird der Private Schlüssel gestohlen, könnte ein Angreifer eine Key-Rotation durchführen, noch bevor es der Controller bemerkt hat. Der Angreifer hat diesen Identifier somit erfolgreich in Besitz genommen. Der Controller hat keine Möglichkeit mehr, eines seiner Schlüsselpaare im Smart-Contract zu registrieren. Um diesem Problem zu begegnen, ist eine On-Chain-Interaktion notwendig. Für einen Identifier könnte beispielsweise die Adresse eines Multi-Signature-Wallets registriert werden. Dabei handelt es sich ebenfalls um Smart-Contracts, in denen N Adressen für eine repräsentative Adresse (die des Smart-Contracts) registriert werden können und die bestimmten Regeln unterliegen. So kann beispielsweise eine Regel lauten, dass ein Multi-Signature-Contract eine Transaktion an den ERC-1056-Contract nur dann auslöst, wenn dies durch M von N der festgelegten Adressen in signierten Transaktionen bestätigt wird. Auf diese Weise ist eine Key-Rotation für den Eigentümer auch im Falle eines Diebstahls oder Verlusts weiterhin möglich. Es gibt darüber hinaus weitere Funktionen des ERC-1056-Contracts, die das Hinzufügen und Entfernen von weiteren Attributen wie auch von Adressen für Delegationszwecke erlauben (DID-Operationen). Ein DID-Resolver kann diese Informationen später nutzen, um daraus ein gültiges DID-Document zu erstellen.

[8] Elliptic Curve Digital Signature Algorithm (ECDSA)

Der ERC-1056-Contract ist im Ethereum-Mainnet sowie einer Reihe von Testnetzwerken[9] *deployed*, kann darüber hinaus aber auch in privaten (z. B. Konsortien) Ethereum-Netzwerken deployt werden (*Ethereum DID Registry* 2022). Je nach Netzwerk, benutzt die ETHR-DID-Methode Suffixes mit dem Namen des Netzwerkes. Ist kein Suffix angegeben, wird das Mainnet angenommen. Der methodenspezifische Identifier ist die hexadezimale Repräsentation des Keccak-256-Hashes eines Öffentlichen Schlüssels, der über einer elliptischen Kurve, genauer einer Koblitz-Kurve mit 256 Bit (Secp256k1) nach *SEC 2* (2010), erzeugt wurde. Das ist die gleiche Kurve, die auch in Bitcoin und demnach der BTCR-DID-Methode verwendet wird (Antonopoulos 2015, S. 66). Es ist außerdem möglich, den Identifier mit einer Prüfsumme zu versehen, wie von Buterin (2020) beschrieben[10](*ETHR-DID* 2021). Hierbei ist die Groß- und Kleinschreibung zu beachten (Case-Sensitive). Als Signaturverfahren wird wie auch in Bitcoin nur ECDSA unterstützt (Antonopoulos und Wood 2018, S. 115).

Als Vorgängerversion des ERC-1056 kann der ERC-725 (did:erc725) angesehen werden (vgl. Braendgaard u. a. 2018; vgl. Vogelsteller u. a. 2017; vgl. Sabadello, Vogelsteller u. a. 2018). Darin werden für neue Identifier jeweils neue Smart-Contracts deployt, die als Stellvertreter oder Proxy dienen. Diese führen ihrerseits Smart-Contracts unter bestimmten Bedingungen aus oder deployen Neue. Es können jede Art von Token (nativ, ERC-20, etc.) sowie der Eigentümer der Contracts verwaltet werden. Der ERC-726 definiert lediglich eine abstrakte Standardschnittstelle zur Verwaltung von Identifiers auf Basis von Smart-Contracts. Softwarehersteller können diese dann in Smart-Contracts implementieren und jene wiederum für die Identitäten ihrer Anwendungen deployen. Der Verbrauch an *Gas*[11] und die damit verbundenen Kosten sind bei einer großen Anzahl an Deployments entsprechend hoch, woraufhin der leichtgewichtigere ERC-1056 vorgeschlagen wurde.

Die *Sovrin-DID-Methode* (did:sov, engl. Sovrin DID Method Specification, zukünftig evtl. did:indy:sovrin) beschreibt DIDs, die zur Verwendung mit dem *Sovrin-Netzwerk* vorgesehen sind (Lodder u. a. 2021). Das Sovrin-Netzwerk ist eine Instanz eines Distributed-Ledgers auf Basis von Hyperledger-Indy und unterliegt der Governance der *Sovrin-Foundation*. Hyperledger-Indy und Hyperledger-Aries bilden zusammen eine Full-Stack-Lösung, die die Schichten eins bis drei des ToIP-Stacks adressieren. Initiator des Sovrin-Projekts und der damit verbundenen Hyperledger-Projekte *Indy* und *Aries* ist das Unternehmen Evernym. Die ursprüngliche Codebasis von Sovrin wurde an die Sovrin-Foundation abgetreten, die diese wiederum weiter an die Linux-Foundation gegeben hat, wo sie unter dem Projektnamen Hyperledger-Indy weiterentwickelt wird (Tobin 2018). Der methodenspezifische Identifier kann auf verschiedene Weise gebildet werden. Es können weit verbreitete Methoden zur Erzeugung von UUIDs

[9] Adresse: 0xdca7ef03e98e0dc2b855be647c39abe984fcf21b; u. a. *Ropsten*, *Rinkeby*, *Goerli*, *Kovan* und Mainnet.

[10] Beispiel: did:ethr:ropsten:0x1D3e2E150b086D1086b496F3b8FAdf21d64ac5C7

[11] Gas ist eine Einheit in Ethereum, mit der die Komplexität für eine Berechnung, die in der Ethereum Virtual Maschine (EVM) ausgeführt werden soll, gemessen wird.

oder auch die ersten 16 Byte eines 256 Bit langen Öffentlichen Schlüssels, der über der elliptischen Kurve *Curve25519* berechnet wird (Ed25519), verwendet werden. Die Ableitung des Identifiers ist somit nicht dringend vom zugrundeliegenden Öffentlichen Schlüssel abhängig. Weitere unterstützte Arten von Schlüsseln können u. a. auf *RSA*, *Twisted-Edwards- / Montgomery-Curves*,[12] *BrainPool-Curves*,[13] *SEC2-Curves*[14] oder *Barreto-Naehrig-Curves*[15] basieren (Lodder u. a. 2021). Die Repräsentation bzw. Codierung des Identifiers erfolgt in Base58.

Hyperledger-Indy besteht aus einer Reihe von Tools und Bibliotheken für ein Distributed-Ledger, das speziell für Decentralized-Identities konzipiert ist (*Hyperledger Indy* 2019). Dies sind insbesondere *Indy-Node* und *Indy-Plenum*, SDKs und Tools für Clients, sowie gemeinsam genutzte Ressourcen für Kryptografie und zur Sammlung von RFCs. Indy-Node ist jene Softwarekomponente, die serverseitig als Teil eines Distributed-Ledger-Netzwerkes auf einem Netzwerkknoten (auch Peer oder Node) betrieben wird, um Transaktionen zu verarbeiten und das bzw. die Ledger zu speichern. Indy-Node verwaltet insgesamt drei Ledger. Das wichtigste ist das *Domain-Ledger*, das DIDs und deren DID-Documents, Schemas, Definitionen und Sperrlisten (Revocation-Registries) für Credentials, sowie *Agent-Authorisation-Policies* speichert. Letztere sind dafür gedacht, Geräte bzw. Agents im Falle eines Verlustes, Diebstahls oder einer Kompromittierung zu sperren. Die sogenannten *Public-DIDs*, die im Domain-Ledger gespeichert werden, sind in erster Linie für Issuer und Diensteanbieter gedacht (vgl. Hardman 2020). Wie auch bei der ETHR-DID-Methode werden DIDs in Sovrin bzw. Hyperledger-Indy nicht pauschal im Ledger gespeichert. Für Verbindungen zwischen zwei Agents werden jeweils eigene, private DIDs berechnet. Das Credential-Schema beschreibt, welche Aussagen (Claims) oder Attribute ein Credential enthält und die Schema-Definition bindet das Schema an einen Issuer, wobei ein Schema von mehreren Issuers verwendet werden kann. Die Revocation-Registry ist in Hyperledger-Indy eine Datenstruktur mit einem Verweis auf die Credential-Definition und einem Wert für einen *Cryptographic-Accumulator*, die ein Issuer im Ledger speichert und mit dessen Hilfe er Verifiable-Credentials zurückziehen kann. Die Revocation-Registry enthält außerdem einen Verweis und den Hash zu einem *Tails-File*, das auf andere Weise durch den Issuer veröffentlicht werden muss (Webserver o. ä.). Das Tails-File beinhaltet normalerweise eine sehr große Anzahl an Werten oder Hashes für den Cryptographic-Accumulator, typischerweise mehrere Hunderttausend bis zehn Millionen (vgl. ebd.). Die einzelnen Werte des Tails-Files ergeben zusammen den Wert des Accumulators (vgl. Abbildung 3.6). Jedem Credential wird während der Ausstellung ein Index des Tails-File zugeordnet und dieser zusammen mit einem *Witness* an

[12] m-221, e-222, ed1174, ed25519, ed383187, ed41417, e-382, m-383, ed448, e-521, m-511

[13] brainpoolp160r1, brainpoolp192r1, brainpoolp224r1, brainpoolp256r1, brainpoolp320r1, brainpoolp384r1, brainpoolp512r1.

[14] secp112r1, secp128r1, secp160r1, secp192r1, secp224r1, secp256r1, secp384r1, secp521r1, secp160k1, secp163k1, secp192k1, secp224k1, secp256k1, sect163r2, sect233r1, sect239r1, sect283r1, sect409r1, sect571r1.

[15] fp224bn, fp254bna, fp254bnb, fp256bn, fp384bn, fp512bn, fp638bn.

den Holder gesendet. Das Witness enthält alle Werte des Tails-Files, bis auf den am eigenen Index. Mit dem Wert am eigenen Index und dem Witness kann ein Holder (*Prover*) später gegenüber einem Verifier einen *Proof-of-Non-Revocation* in Form eines *Zero-knowledge-Proofs* erbringen.

Für einen Cryptographic-Accumulator können z. B. kryptografische, quasi-kommutative Hashfunktionen verwendet werden, für die u. a. gilt: $h(h(x,y_1),y_2) = h(h(x,y_2),y_1)$ (Benaloh u. a. 1994). Um ein Credential zurückzuziehen, entfernt der Issuer den Wert oder den Hash am entsprechenden Index im Tails-File und verändert damit auch den Wert des Accumulators (vgl. Abbildung 3.7). Einem Prover ist es nun nicht mehr möglich, einen Beweis darüber zu führen, dass das entsprechende Credential nicht zurückgezogen wurde. In Sovrin wird eine eigene, nach eigenen Aussagen verbesserte, Implementierung eines Cryptographic-Accumulators basierend auf den Ausführungen von Camenisch u. a. (2009) verwendet (vgl. Khovratovich und Law 2016, S. 2).

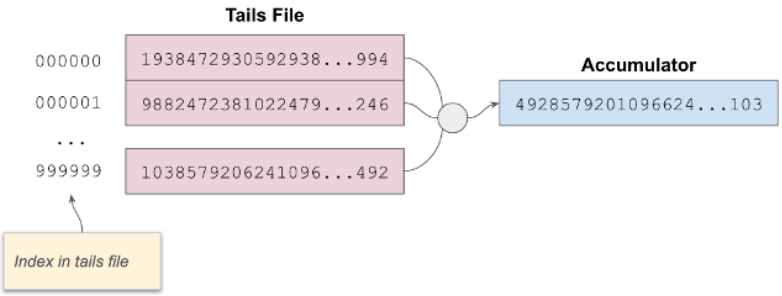

Abbildung 3.6: Aufbau eines Tails-File (Hardman 2020)

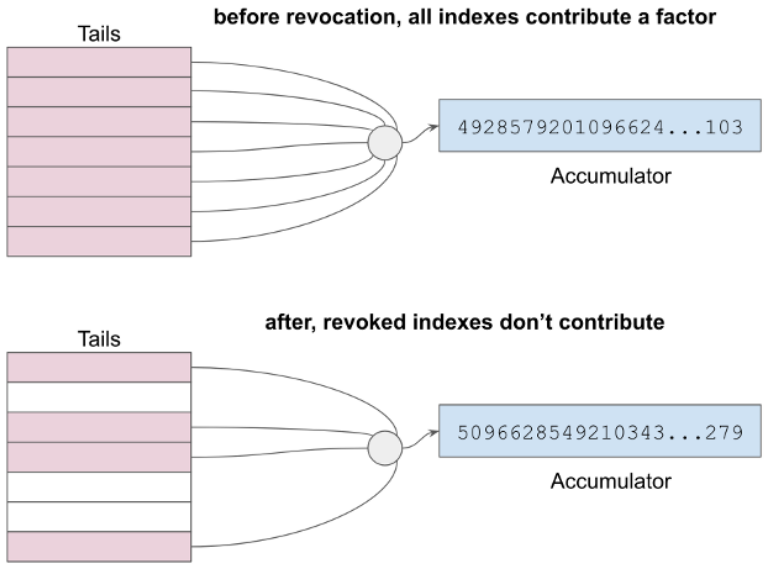

Abbildung 3.7: Tails-File und Accumulator vor und nach einer Revocation (Hardman 2020)

Die beiden anderen Ledger sind das *Pool-Ledger*, das Informationen zu den Nodes im Netzwerk enthält, und das *Config-Ledger*, mit Konfigurationsdaten zu diesen Nodes.

Indy-Plenum ist ein Teil von Indy-Node und stellt die Komponenten für das Konsensverfahren zur Verfügung (*Plenum* 2020). Indy-Plenum implementiert einen Konsens-Algorithmus mit *Redundant-Byzantine-Fault-Tolerance (RBFT)* nach dem Vorbild von Aublin u. a. (2013). Dabei werden bei N teilnehmenden Nodes $(N-1)/3$ fehlerhafte Nodes toleriert.

Das *Hyperledger-Aries-Projekt* umfasst Softwarebibliotheken und Protokolle, die für Komponenten von Endanwendersoftware bzw. Agents entwickelt werden. Dabei ist vor allem die Spezifikation und Standardisierung der Kommunikation von Agents untereinander, wie auch die Ausdifferenzierung der Aufgaben und Abläufe dieser, Kernaufgabe des Projektes. Dafür gibt es verschiedenste Protokolle, die z. B. das Ausstellen (Issuer/Holder) oder Zeigen (Holder/Verifier) von Credentials beschreiben. Darüber hinaus werden Transportprotokolle, Nachrichtenformate und vieles weitere mehr zur Standardisierung vorgeschlagen (*Hyperledger-Aries-RFCs* 2022). Insbesondere *DIDComm* ist in diesem Projekt verwurzelt, wird nunmehr aber vornehmlich durch die DIF koordiniert (vgl. Abschnitt 3.1). Auch der ToIP-Stack wurde ursprünglich in einem Aries-RFC formuliert, sodass Hyperledger-Indy und Hyperlegder-Aries einschließlich Sovrin naturgemäß dem ToIP-Stack sehr nahe stehen. Bemerkenswert sind weiterhin dar Aries-RFC *0302: Aries Interop Profile* sowie die *Aries Protocol Test Suite*, die sich den Aspekten der Interoperabilität widmen (Curran u. a. 2021; *Aries Protocol Test Suite* 2021).

DIDComm-Messaging (DIDComm) ist ein Entwurf für ein Kommunikationsprotokoll, das sichere und private Verbindungen zwischen Agents in DID-basierten Systemen ermöglichen soll. (Curren u. a. 2021). Es arbeitet nachrichtenbasiert, asynchron sowie simplex[16] und lehnt sich eher an das Paradigma der E-Mail (SMTP) an. Findet der Nachrichtenaustausch aber schnell genug statt, lässt sich damit auch ein Interaktionsmodell konstruieren, das synchrone *Request-Response-Interaktionen* erlaubt, wie es heute im Web verbreitet ist. DIDComm ist selbst kein Transportprotokoll, sondern setzt auf Andere auf. Das können beispielsweise HTTPS, WebSocket, Bluetooth, SMTP oder NFC sein. DIDComm-Nachrichten lassen sich zudem *routen*, das heißt, sie können über mehrere Nodes und über unterschiedliche Transportprotokolle vermittelt werden. Es werden drei verschiedene Nachrichtentypen unterschieden: *Plaintext Messages*, *Signed Messages* und *Encrypted Messages*. Das Nachrichtenformat basiert auf *JSON-Web-Token (JWT)* und ist bei der IETF unter der Bezeichnung *JSON-Web-Message (JWM)* als Standardisierungsentwurf erfasst (Looker 2020; Jones, Bradley u. a. 2015b). Für signierte und verschlüsselte Nachrichten kommen *JSON-Web-Signature (JWS)* respektive *JSON-Web-Encryption (JWE)* zum Einsatz (Jones, Bradley u. a. 2015a; Jones und Hildebrand 2015). Die kryptografischen Algorithmen für JWS[17] und JWE[18] sind in

[16] Der Nachrichtenaustausch erfolgt zum Zeitpunkt t in nur einer Richtung.

[17] Ed25519, ES256, ES256K.

[18] Asymmetrisch: X25519 (Curve25519), P-384, P-256; Symmetrisch: XC20P, A256GCM, A256CBC-HS512; Key-Wrapping: ECDH-ES+A256KW (P-256, P-384, P521, X25519), ECDH-1PU+A256KW (P-256, P-384, P-521, X25519).

den *JSON-Web-Algorithms* definiert, wobei DIDComm nur eine Teilmenge davon unterstützt (Jones 2015a; Curren u. a. 2021).

JSON-Web-Tokens (JWT) sind Access Tokens, die normalerweise dazu verwendet werden, Behauptungen oder Ansprüche (*Claims*) zwischen zwei Parteien auszutauschen und zu verifizieren. Sie sind in RFC 7519 spezifiziert (Jones, Bradley u. a. 2015b). Typische Anwendungsgebiete sind beispielsweise der Austausch von Identitätsdaten zwischen einem Identity-Provider und einem Service-Provider oder der Implementierung von zustandslosen Sitzungen, wie sie z. B. bei *Representational-State-Transfer (REST)* gefordert werden. Vergleichbare Token sind *Simple-Web-Tokens (SWT)* und *Security-Assertion-Markup-Language-Tokens (SAML)*. Die Darstellung erfolgt in *JavaScript-Object-Notation (JSON)* nach RFC 8259 (Bray 2017) und wird in serialisierter Form mit Base64Url codiert, d.h. es wird eine Abwandlung der Base64-Kodierung benutzt, bei der die Zeichen „/“ und „+“ durch “_” (Unterstrich) und „-“ (Minus) ersetzt werden. Dadurch wird die Übertragung in einem URL ermöglicht. Die in einem JWT formulierten Claims sind wiederum Teil einer weiteren JSON-Datenstruktur, der *JSON-Web-Signature (JWS)* oder der *JSON-Web-Encryption (JWE)*. JWS und JWE ist in den RFCs 7515 und 7516 spezifiziert (Jones, Bradley u. a. 2015a; Jones und Hildebrand 2015). Welche Verfahren für Signaturen und zur Verschlüsselung verwendet werden können, lässt sich den *JSON-Web-Algorithms (JWA)* entnehmen (Jones 2015a). Die zugehörigen Schlüssel bzw. deren Repräsentation in einer Datenstruktur definiert RFC 7517: *JSON-Web-Key* (Jones 2015b).

Die Datenstruktur eines solchen Token besteht aus drei Teilen: den Metadaten (*header*, dem *JOSE*[19]*-Header*), den Nutzdaten (*payload*, dem *JWS Payload*) und einer Signatur (*signature*). Diese werden wie oben beschrieben codiert und konkateniert, wobei zur Trennung ein „.“ (Punkt) verwendet wird (header.payload.signature). Im Header werden der Token-Typ und die verwendeten Algorithmen für Signaturen und ggf. zur Verschlüsselung angekündigt. Der Token-Typ gibt den IANA-Medientyp (application / jwt) an und hat immer den Wert „JWT“. Bei verschachtelten JWTs, d.h. wenn der Payload ebenfalls ein JWT enthält, wird zusätzlich ein Feld für den *Content Type* benötigt. In diesem Fall ist auch hier der Wert „JWT“ anzugeben. Ansonsten enthält der Payload die eigentlichen Claims. Über diese beiden Teile wird anschließend eine Signatur erzeugt. Der JOSE-Header für eine JWS unterscheidet sich vom JOSE-Header für eine JWE durch ein Feld des Typs „enc“, das den Verschlüsselungsalgorithmus angibt. Wenn das Feld „enc“ existiert, handelt es sich um eine JWE, sonst um eine JWS. Die Signatur kann sowohl symmetrisch mittels *Keyed-Hash-Message-Authentication-Code (HMAC)* als auch asymmetrisch mittels Public-Key-Kryptografie erzeugt werden. Eine mögliche Verwendung dieser Token ist in Abbildung 3.8 dargestellt.

JWTs sind gegenüber *Man-in-the-Middle-*Angriffen nicht sicher, auch nicht bei Verwendung von JWE. Es ist daher darauf zu achten, dass eine Kanalverschlüsselung wie z.B.

[19] Javascript Object Signing and Encryption (JOSE)

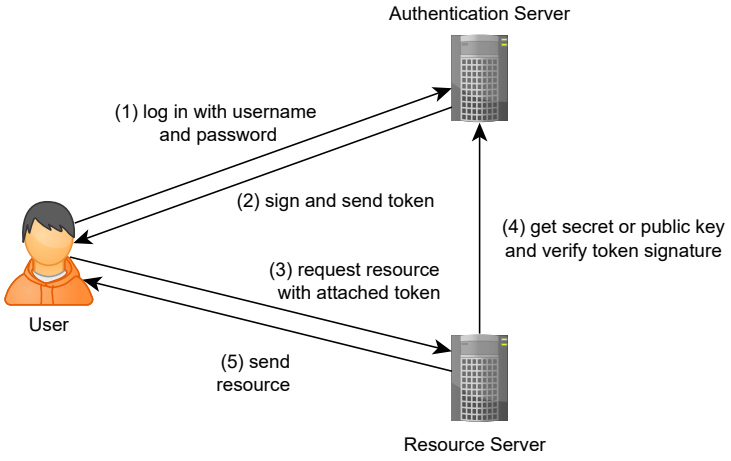

Abbildung 3.8: Verwendung von JWTs (eigene Darstellung)

Transport-Layer-Security (TLS) verwendet wird, damit das Token nicht von Dritten kopiert und wiederverwendet werden kann (*Replay-Attack*). Weiterhin muss im Falle von HMAC eine sichere Möglichkeit zum Austausch des Schlüssels (*secret* oder *key*) zwischen den beteiligten Servern gefunden werden. *Authentication-Server* und *Resource-Server* können sich aber auch auf ein und demselben Gerät befinden, sodass der eben genannte Punkt vernachlässigt werden kann. JWTs können auch Verifiable-Credentials (VCs) und Verifiable-Presentations (VPs) aufnehmen. Hierfür führen Sporny, Longley und Chadwick (2021) in § 6.3.1 zwei neue Schlüsselwörter als registrierte Bezeichnungen für JWT-Claims ein. Es werden daneben auch Regeln festgelegt, wie z. B. Claims von VCs oder VPs in die von JWTs zu übersetzen sind oder welche Arten von Proofs erlaubt sind. Eine Einschränkung gegenüber VCs ist, dass jeweils nur ein Credential-Subject verwendet werden darf.

Sidetree ist ein Protokoll, das sich als 2nd-Layer oder *Overlay-Network* für andere verteilte Systeme wie Blockchains, Distributed-Ledgers oder jene, die mit irgendeiner Form von Witnesses arbeiten, versteht (Buchner, Steele und Ronda 2022). Die Syntax der Identifier und das dazugehörige Datenmodell, die vom Protokoll verwendet werden, entsprechen denen der W3C-Spezifikation für DIDs. Implementierungen dieses Protokolls können als eigene DID-Methoden verfasst und in der W3C-DID-Method-Registry registriert werden. Dort ist auch festgelegt, wie die Identifier konkret zu erzeugen sind und welches konkrete Ledger zugrunde liegt. Bislang existieren für Sidetree zwei DID-Methoden, die zum einen auf Bitcoin (*ION*, did:ion) und zum anderen auf Ethereum (*Element*, did:elem) basieren, wobei Element offenbar nicht mehr weiter verfolgt wird, da dessen GitHub-Repository als archiviert gekennzeichnet ist und es keine Hinweise auf Nachfolgeprojekte gibt (vgl. *ION* 2021; vgl. *Element* 2020). Die Architektur von Sidetree-basierten DID-Methodenimplementierungen besteht aus Overlay-Netzwerken mit unabhängigen Sidetree-Nodes, die mit einem zugrundeliegenden dezentralisierten *Anchoring-System* (Distributed Ledger o. ä.) interagieren, um DID-Operationen unter

Verwendung deterministischer Protokollregeln zu replizieren. Durch die Verwendung des Sidetree-Protokolls lassen sich viele DID-Operationen in einer einzigen Layer-1-Transaktion unterbringen. Dies wird dadurch erreicht, dass mehrere DID-Operationen als Stapel bzw. *Batch* zusammengefasst werden und die Transaktion lediglich eine Referenz zu einem Batch-File enthält, das in einem Content-Addressable-Storage (CAS) wie z. B. IPFS gespeichert ist. Die Sidetree-Nodes schreiben also CAS-Verweise in ein Distributed-Ledger, die jeweils auf eine bestimmte Datei zeigen, das sogenannte *Core-Index-File*. Dieses enthält Informationen zum Erstellen, Wiederherstellen und Deaktivieren von DIDs sowie einen weiteren CAS-URI zu einer untergeordneten Datei, dem *Provisional-Index-File*. Dieses File enthält Daten zum Aktualisieren von DIDs und wiederum einen weiteren CAS-URI, der auf ein oder mehrere *Chunk-Files* verweist. Die Chunk-Files enthalten CRDT-Deltas, in denen die Zustandsänderungen des DID-Documents inkrementell aufgezeichnet werden. Sofern ein Batch an DID-Operationen in einer Transaktion auch jene zur Deaktivierung (Deactivate) oder zur Wiederherstellung (Recovery) einer DID enthält, muss sowohl das Core-Index-File als auch das Provisional-Index-File einen weiteren CAS-URI besitzen, der zu einem sogenannten *Core-Proof-File* respektive *Provisional-Proof-File* zeigt (s. Abbildung 3.9).

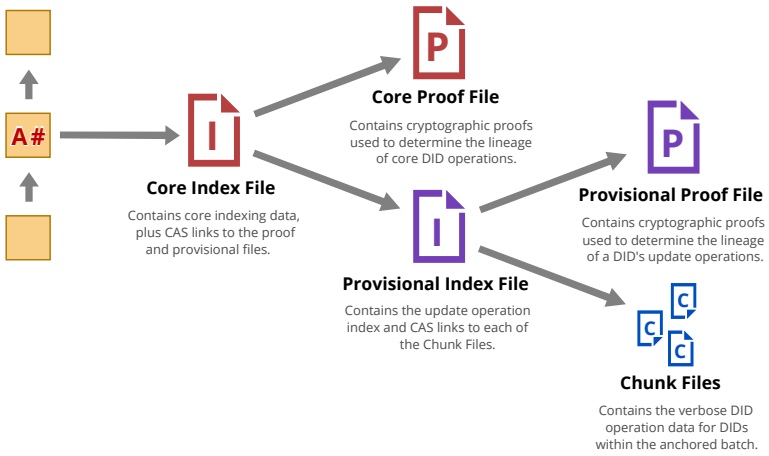

Abbildung 3.9: Datei-Struktur in Sidetree (Buchner, Steele und Ronda 2022)

Das Distributed-Ledger dient dazu, eine unveränderliche Historie von DID-Operationen zu ordnen, sodass alle beteiligten Sidetree-Nodes die genaue Abfolge von Änderungen an einem DID-Document reproduzieren können und eine konsistente Sicht auf diese bekommen, ohne dass ein zusätzlicher Konsensmechanismus erforderlich ist (vgl. Abbildung 3.10). Die methodenspezifischen Identifier werden bei DID-Methoden, die dem Sidetree-Protokoll folgen, grundsätzlich auf die gleiche Weise gebildet, wobei sich die konkreten Algorithmen unterscheiden können. Sie werden aus zwei verketteten JSON-Datenstrukturen berechnet, die den Ausgangszustand eines DID bzw. des zugehörigen DID-Documents repräsentieren und als *Create-Operation-Suffix-Data-Object* sowie *Create-Operation-Delta-Object* bezeichnet werden. Sie enthalten jeweils ein *Commitment* (Hash) zweier Öffentlicher Schlüssel, die zum Aktualisieren und zur Wiederherstellung des DID-Documents berechtigt sind (Key-Rotation). Sidetree unterscheidet

zwischen zwei Formen der resultierenden Identifier. Zur Erzeugung eines *Short-Form-DID-URI* wird auf das kanonisierte Create-Operation-Suffix-Data-Object ein methodenspezifischer Hash-Algorithmus angewendet. Bei vielen DID-Methoden vergeht jedoch zwischen der Erzeugung eines DID und der Verankerung, Weiterleitung und Verarbeitung der DID-Operation in den zugrundeliegenden Verankerungs- und Speichersystemen eine gewisse Zeit (die unbestimmt sein kann). Um diesem Umstand Rechnung zu tragen, führt Sidetree eine äquivalente Variante von Sidetree-basierten DIDs ein, die selbstzertifizierend und selbstauflösend sind und als *Long-Form-DID-URI* bezeichnet werden. Diese Variante ermöglicht es, dass DIDs sofort nach der Generierung aufgelöst werden können, indem die anfänglichen Zustandsdaten des DIDs in den Identifier selbst mit aufgenommen werden. Long-Form-DID-URIs bestehen aus der Short-Form-DID-URI und einem zusätzlichen, durch einen Doppelpunkt getrennten Segment, das am Ende des DIDs angehängt wird. Der Wert dieses abschließenden URI-Segments setzt sich aus den Create-Operation-Suffix-Daten sowie den Create-Operation-Delta-Daten zusammen, die durch ein methodenspezifisches Verfahren codiert sind (Base64Url o. ä.).

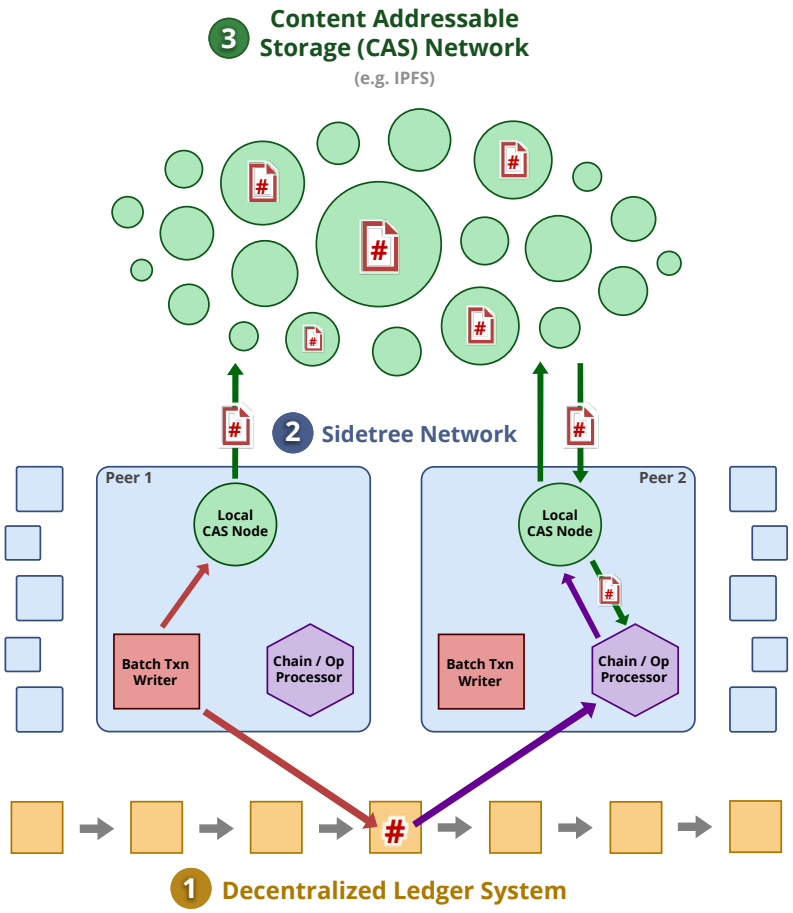

Abbildung 3.10: Architektur von Sidetree (Buchner, Steele und Ronda 2022)

Das *Credential-Handler-API (CHAPI)* ist ein Entwurf für ein standardisiertes API, das den Umgang mit Credentials in Browsern bzw. für Web-Anwendungen vereinheitlichen

soll (Longley und Sporny 2021). Es stellt eine Erweiterung der *Credential-Management-Level-1-Spezifikation* dar (West 2019). Der Schwerpunkt liegt dabei auf den Prozessen, die für eine Web-Anwendung zum Ausstellen und Zeigen von Verifiable-Credentials notwendig sind und Benutzern die Möglichkeit geben, vertrauenswürdige Web-Anwendungen von Drittanbietern als Handler für Credential-Anfragen und zur Speicherung von Credentials zu verwenden. Jene Web-Anwendungen werden in diesem Kontext als *Credential-Repository* bezeichnet. Ein *Credential-Handler* ist ein Event-Handler, der Credential-Request- und Credential-Storage-Events empfängt und verarbeitet. Das Credential-Handler-API ermöglicht es Websites, einen solchen Credential-Handler zu installieren, sodass andere Websites, die Anmeldedaten anfordern oder speichern, diesen nutzen können. Der Nutzer bekommt auf dieser anderen Website ein Auswahlfenster zu sehen, das die installierten und kompatiblen Credential-Handler anzeigt. Die Funktionen des Credential-Handler-API lassen sich auf verschiedene Weise in Web-Anwendungen bereitstellen. Eine Möglichkeit ist, die Events über einen *Service-Worker* zu empfangen (*Credential Handler API Demo* 2019). Mit dieser Browser-Funktion (API) lässt sich JavaScript im Hintergrund ausführen. Eine andere Möglichkeit ist ein sogenanntes *Credential-Handler-Polyfill*. Der Begriff Polyfill wurde von Remy Sharp in seinem Buch *Introducing HTML5* (2010) geprägt und bezeichnet einen nachladbaren Programmcode, meist ebenfalls JavaScript, der einen Browser um bisher nativ nicht unterstützte Funktionen erweitert.

Das *Self-Issued-OpenID-Connect-Provider-DID-Profile (SIOP-DID)* war eine Spezifikation, die OpenID-Connect (OIDC) um die Authentisierung mittels DIDs (DID-Auth) erweitern sollte und die aus der Arbeit einer Arbeitsgruppe innerhalb der DIF hervorgegangen ist (Oliver Terbu u. a. 2021). Sie gilt mittlerweile als veraltet und wird unter Federführung der OpenID-AB/Connect-Working-Group in zwei eigenständigen Spezifikationen weitergeführt: *Self-Issued-OpenID-Provider-v2* und *OpenID-Connect-for-Verifiable-Presentations* (Yasuda u. a. 2022; O. Terbu u. a. 2022). OpenID-Connect definiert Mechanismen, mit denen ein Nutzer einen *OpenID-Provider (OP)* nutzen kann, um Identitätsinformationen (wie Authentifizierungsdaten oder Claims) an eine *Relying-Party (RP)* weiterzugeben. Die erstgenannte Spezifikation erweitert OpenID-Connect um das Konzept eines Self-Issued-OpenID-Providers (Self-Issued-OP); ein OP, der unter der lokalen Kontrolle des Benutzers ausgeführt wird. Die Benutzer können sich mit selbst erstellten OPs authentifizieren und Claims direkt an die RPs übermitteln, ohne dabei auf einen Diensteanbieter angewiesen zu sein oder eine eigene OP-Infrastruktur betreiben zu müssen. Üblicherweise besteht in OpenID-Connect eine Vertrauensstellung zwischen einem gehosteten, von Dritten bereitgestellten OP und einer RP. Im Falle von SIOP gibt es dieses Vertrauen allerdings nicht, da jeder Nutzer einen eigenen, lokalen OP betreibt und dieser der RP normalerweise nicht bekannt ist. Das heißt, die übermittelten Identitätsdaten entsprechen einer Selbstauskunft und sind nicht überprüfbar. Um diese Lücke zu schließen, stellt die zweite Spezifikation Mechanismen zur Übermittlung von kryptographisch verifizierbaren Angaben, die von Drittanbietern ausgestellt wurden, bereit. Die in dieser Spezifikation definierten Erweiterungen bieten die Protokolländerungen,

die zur Unterstützung des SIOP-Modells in Verbindung mit Verifiable-Credentials und Verifiable-Presentations notwendig sind. Der Ausstellprozess von Verifiable-Credentials ist dabei kein Gegenstand dieser Spezifikationen. Es werden lediglich Datenstrukturen und Transportmöglichkeiten definiert, mit denen Claims bzw. Credentials von einer RP angefragt und durch einen OP beantwortet werden.

Presentation-Exchange (Version 1 und 2) beschreibt Datenstrukturen für den Austausch von Verifiable-Presentations zwischen einem Verifier und einem Holder (Buchner, Zundel und Riedel 2021; Buchner, Zundel, Riedel und Duffy 2021). Es werden zwei Datenstrukturen definiert: *Presentation-Definition* und *Presentation-Submission*. Erstere formuliert die Anforderungen eines Verifiers an eine Verifiable-Presentation, die ein Holder zurückliefern soll. Letztere drückt aus, wie die zurückgelieferten Identitätsdaten, die mit den in einer Presentation-Definition festgelegten Anforderungen übereinstimmen, bereitgestellt werden (z. B. das Format). Die Spezifikation ist so konzipiert, dass sie sowohl vom Claim-Format als auch vom Transportprotokoll unabhängig ist, sodass JSON-Web-Tokens (JWTs), Verifiable-Credentials (VCs), JWT-VCs oder jedes andere JSON-Claim-Format verwendet und diese über OpenID-Connect, DIDComm, Credential-Handler-API (CHAPI) oder jedes andere Transportprotokoll übermittelt werden kann. Die Spezifikation definiert selbst keine eigenen Transportprotokolle, Endpunkte (einer API o. ä.) oder andere Hilfsmittel, die zur Übermittlung dieser Datenstrukturen dienen können, sondern baut auf andere Spezifikation oder Projekte auf, die Presentation-Exchange in ihren Protokollen benutzen wollen.

Inter-Wallet-Credential-Exchange (IWCE) beschreibt ebenfalls eine Möglichkeit, Verifiable-Presentations zwischen einem Verifier und einem Holder auszutauschen (Menzer u. a. 2021). Im Gegensatz zu Presentation-Exchange (2021) sind die Transportprotokolle sowie deren Sicherheitsimplikationen enger gefasst. Darüber hinaus werden organisatorische Aspekte und der Umgang mit Privaten Schlüsseln vorgeschrieben. IWCE fokussiert sich auf den Austausch von Verifiable-Credentials (VCs) bzw. Verifiable-Presentations (VPs) zwischen zwei verschiedenen Wallets unterschiedlicher Diensteanbieter oder Softwarehersteller. Es soll damit ein Austausch von Identitätsinformationen in einer Art und Weise erfolgen, bei der sich die eigene Digitale Identität durch die Gesamtheit vieler VCs unterschiedlichster Anwendungen von verschiedenen Herstellern und Diensten bildet. Es zielt dabei nicht auf eine rein technische Authentifizierung zwischen zwei Endpunkten ab, sondern ist Teil eines Prozesses, der heutzutage meist durch eine Registrierung von neuen Nutzern und dem Ausfüllen von (Web-) Formularen abgebildet ist. Es ist ausdrücklich keine Lösung für Single-Sign-On (SSO), sondern eine Möglichkeit für Dienste, überprüfbare Identitätsinformationen eines Nutzers zur Registrierung zu verwenden und Artefakte seiner Digitalen Identität in kryptografisch überprüfbarer Form zu erhalten. Jeder Dienst, der einem seiner Nutzer etwas in Form von VCs attestiert und damit als Issuer auftritt, bindet die VCs ausschließlich an das Schlüsselmaterial bzw. die Identifier der eigenen Anwendungen. Die ausgestellten VCs verwaltet der Nutzer ausschließlich in den

zugehörigen Anwendungen des jeweiligen Dienstes. Private Schlüssel, die in diesen Anwendungen gespeichert sind, verlassen diese niemals. Identitätsinformationen, die über die Anwendungs- bzw. Domänengrenzen hinweg übertragen werden sollen, sind ausnahmslos in Form von VPs auszudrücken. Durch die Verwendung von Universal- bzw. App-Links ist auch TLS bzw. HTTPS dringend für den Datenaustausch gefordert, wodurch sich zwei Anwendungen sicher gegenseitig authentifizieren können und, bei beidseitiger Nutzung dieser speziellen Links, die Kommunikation ausschließlich lokal auf dem Endgerät stattfindet. Diese Links, da es sich dennoch um gültige URLs handelt, können darüber hinaus ebenfalls in Verbindung mit Web-Anwendungen verwendet oder als QR-Code dargestellt werden.

Der *Universal-Resolver* ist eine Software, die die Auflösung von DIDs verschiedener DID-Methoden zu den entsprechenden DID-Documents erlaubt und damit eine Menge von *DID-Resolvern* vereint (*Universal Resolver* 2022). Die Implementierung folgt grundlegend den Spezifikationen von Sporny, Longley, Sabadello u. a. (2021) sowie Sabadello und Zagidulin (2022). Die Entwicklung der Software ist innerhalb der „Identifiers & Discovery Working Group“ der DIF organisiert. Die DIF stellt außerdem zwei Instanzen dieser Software zur Verfügung, die zum Testen verwendet werden kann.[20] Der Universal-Resolver kann sowohl lokal, als Teil einer Client-Software (Java-API), als auch entfernt auf einem Netzwerkknoten betrieben werden (REST-API). Die Implementierungen der einzelnen DID-Resolver werden als Treiber (Driver) bezeichnet und können intern unterschiedlich funktionieren. Einige Treiber rufen einfach nur einen entferntes Web-API auf, während andere direkten Zugang zu einem Full-Node der von ihnen verwendeten Blockchain haben (vgl. Abbildung 3.11). Die erste Variante ist vor allem für mobile Endgeräte von Nutzen, während die Zweite höhere Sicherheitsgarantien für das Ergebnis der Auflösung bietet (Sabadello 2017). Gegenwärtig werden 43 DID-Methoden unterstützt. Zusätzliche DID-Methoden lassen sich modular durch weitere Treiber hinzufügen (*Universal Resolver* 2022).

Als *Well-Known-DID-Configuration* wird eine Datenstruktur bezeichnet, die einen DID mit einem Domain-Namen verknüpft (Buchner, Steele und Looker 2021). Die DID-Configuration-Resource muss ein gültiges JSON-Objekt sein, das sogenannte *Domain-Linkage-Assertions* enthält, das heißt, kryptografisch überprüfbare Behauptungen, die beweisen, dass dieselbe Entität sowohl die enthaltenen DIDs als auch die Domain kontrolliert, unter der sich die DID-Configuration-Resource befindet. Diese muss außerdem unter einer Well-Known-URI nach RFC 8615 (Nottingham 2019) im Wurzelverzeichnis der jeweiligen Domain veröffentlicht sein.

Web-Authentication (WebAuthn) definiert ein API, das die Authentisierung mittels Public-Key-Kryptografie in Web-Anwendungen erlaubt und eine Erweiterung des Credential-Management-API darstellt (Hodges u. a. 2021; West 2019). Es ist eine Kernkomponente der *FIDO2-Protokollfamilie* und wird unter dem Dach des W3C zur

[20] https://resolver.identity.foundation/ (IBM Cloud); https://dev.uniresolver.io/ (AWS).

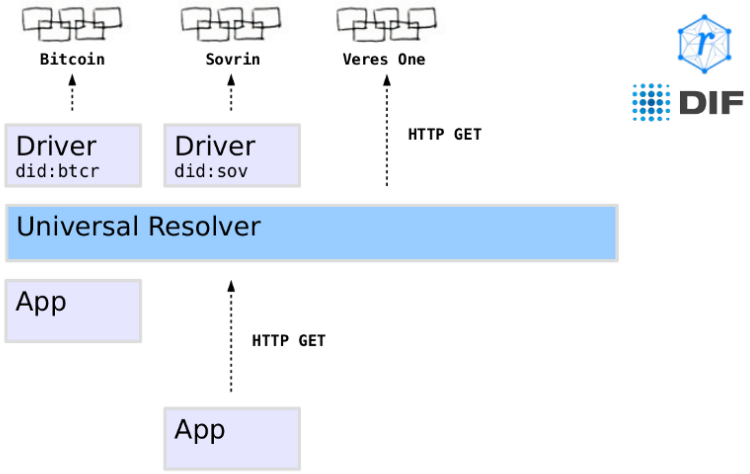

Abbildung 3.11: Architektur des Universal Resolvers (Sabadello 2017)

Standardisierung empfohlen. Bei WebAuthn nutzt ein Agent, z. B. ein Browser, einen konformen *WebAuthn-Authenticator*, der ein *Public-Key-Credential* erstellt und dieses sicher speichert, insbesondere den damit verbundenen Privaten Schlüssel. Der Agent sendet dieses Credential an eine *WebAuthn-Relying-Party*, die in diesem Kontext ein Webserver bzw. das Backend einer Web-Anwendung darstellt. Für jede Relying-Party (RP) wird ein eigenes Credential erstellt, sodass diese nicht mit anderen Web-Anwendungen korreliert werden können. Die jeweils einem Credential zugehörige RP wird als *Origin* bezeichnet. WebAuthn unterscheidet zwei Prozesse: die Registrierung (*Registration*) und die Authentisierung (*Authentication*). Bevor ein Public-Key-Credential zur Authentifizierung genutzt werden kann, muss es zunächst im Registrierungsprozess dem Backend bekannt gegeben werden. Der Login-Prozess muss bis zu diesem Zeitpunkt über einen anderen Weg erfolgen, z. B. mittels Benutzername und Passwort. Die Web-Anwendung fragt den Benutzer nach erfolgreicher Anmeldung, ob das gerade genutzte Gerät, bzw. der darauf ausgeführte Agent, WebAuthn als Anmeldemethode benutzen darf. Stimmt er zu, kann die Authentisierung durch den Agent dauerhaft automatisch und ohne weitere Eingabe eines Passwortes erfolgen. Agents sind in erster Linie Browser. Die meisten modernen Browser unterstützen WebAuthn bereits, u. a. Google Chrome, Mozilla Firefox, Microsoft Edge und Apple Safari (*W3C and FIDO Alliance Finalize Web Standard for Secure, Passwordless Logins* 2019). Diese greifen wiederum auf Implementierungen konformer Authenticators zurück, die in verschiedenen Umgebungen zur Ausführung kommen können. Jene, die sich auf dem gleichen Gerät befinden, werden als *Platform-Authenticators* bezeichnet und z. B. in einem Trusted-Execution-Environment (TEE), Trusted-Platform-Module (TPM) oder Secure-Element (SE) ausgeführt. Befindet sich ein Authenticator auf einem externen Gerät, spricht man von *Roaming-Authenticators*. Die Verbindung kann dabei über Universal-Serial-Bus (USB), Bluetooth-Low-Energy (BLE) oder Near-Field-Communications (NFC) hergestellt werden (Hodges u. a. 2021).

4 Interoperabilität

4.1 Definitionsversuch

Interoperabilität bezeichnet im Allgemeinen die „Fähigkeit unterschiedlicher Systeme, möglichst nahtlos zusammenzuarbeiten“ (*Duden* 2022). Der Begriff wird nicht nur in der Informations- und Telekommunikationstechnik (ICT) gebraucht, sondern spielt z. B. auch in der Medizintechnik, im Transport- und Verkehrswesen, der Automatisierungstechnik, dem Militär oder im E-Government eine Rolle. In Bezug auf ICT unterscheidet das *European-Telecommunications-Standards-Institute (ETSI)* vier Unterarten der Interoperabilität (Veer u. a. 2008, S. 5 f.):

- Technische Interoperabilität
- Syntaktische Interoperabilität
- Semantische Interoperabilität
- Organisatorische Interoperabilität

Technische Interoperabilität wird in der Regel mit Hardware-/Softwarekomponenten, technischen Systemen sowie Plattformen in Verbindung gebracht, die die Kommunikation zwischen Maschinen ermöglichen. Bei dieser Art von Interoperabilität geht es häufig um (Kommunikations-) Protokolle und die für den Betrieb dieser Protokolle erforderliche Infrastruktur.

Syntaktische Interoperabilität wird gewöhnlich mit Datenformaten in Verbindung gebracht. Die von Kommunikationsprotokollen übertragenen Nachrichten müssen eine genau definierte Syntax und Kodierung haben, damit sich einzelne (semantisch bewertbare) Informationseinheiten und Datenstrukturen in den übertragenen Nutzdaten identifizieren und zum Zwecke einer weiteren Verarbeitung extrahieren lassen.

Semantische Interoperabilität ist in der Regel mit der Bedeutung des Inhalts verbunden und betrifft eher die menschliche als die maschinelle Interpretation des Inhalts. Interoperabilität auf dieser Ebene bedeutet also, dass zwischen den Menschen ein gemeinsames Verständnis der Bedeutung der ausgetauschten Informationen besteht.

Organisatorische Interoperabilität ist die Fähigkeit von Organisationen, effektiv zu kommunizieren und (aussagekräftige) Daten (Informationen) zu übertragen, auch wenn sie eine Vielzahl verschiedener Informationssysteme über sehr unterschiedliche Infrastrukturen hinweg verwenden. Organisatorische Interoperabilität hängt von einer erfolgreichen technischen, syntaktischen und semantischen Interoperabilität ab (vgl. Abbildung 4.1).

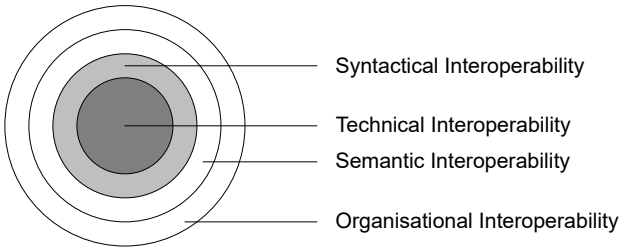

Abbildung 4.1: Stufen der Interoperabilität nach Veer u. a. (2008), S. 6

Eine interessante Herangehensweise ist die Näherung über die Symptome der Nicht-Interoperabilität. Veer und Wiles (2008, S. 6 ff.) definieren dies im Falle von zwei oder mehreren Entitäten, die miteinander kommunizieren, wie folgt:

> It (whatever it is) doesn't work (as expected).

Wichtig hierbei ist, dass ein System nicht nur nicht funktioniert, sondern dass es nicht wie erwartet oder wie beabsichtigt funktioniert. Einige Systeme funktionieren zwar so, wie es die Normungsgremien beabsichtigt haben, werden allerdings für Zwecke oder Aufgaben benutzt, für die sie weder konzipiert noch angedacht waren. Diese Systeme können Standard-konform sein, funktionieren aber dennoch nicht mit Anderen, die ebenfalls den Anspruch dieser Konformität erheben. Es gibt außerdem auch Fälle, in denen Systeme absichtlich nicht interoperabel sind, obwohl sie auf standardisierte Protokolle o. ä. setzen.

Die Gründe für fehlende oder unzureichende Interoperabilität von Implementierungen, die gemeinsam auf einen Standard oder eine Norm setzen, sind oft im Entstehungsprozess dieser selbst zu finden. Sie werden durch Beiträge zahlreicher Personen mit unterschiedlichem Hintergrund, verschiedenen Erwartungen und unterschiedlichen wirtschaftlichen Interessen erarbeitet (ebd., S. 9). In der Praxis stehen oft nicht genügend Ressourcen zur Verfügung, um diese verschiedenen Beiträge in ein konsistentes und kohärentes Ganzes zu bringen. In der Folge sind Spezifikationen oft unvollständig und wesentliche Aspekte für die Interoperabilität fehlen oder sind nur teilweise spezifiziert. Insbesondere Schnittstellen, die dafür entscheidend sind, sind unzureichend oder nicht klar genug definiert. Ein weiteres Problem ist eine zu hohe Anzahl von Optionen. Das heißt, ein Standard ist zu offen gestaltet oder die möglichen Optionen sind zu unscharf gehalten. Es kann beispielsweise sein, dass die Folgen unklar sind, wenn bestimmte Optionen nicht implementiert werden oder dass es zwischen verschiedenen Optionen zu Wiedersprüchen kommt. Standards sollten daher gut strukturiert sein und klare Grenzen zwischen dem ziehen, was standardisiert werden muss, und dem, was nicht standardisiert werden muss. Es gilt es zu vermeiden, verschiedene und ggf. konträre Konzepte zu vermischen oder ein und dieselbe Sache auf verschiedene Weise zu spezifizieren. Die Ausführlichkeit sollte des Weiteren auf ein Minimum reduziert werden; in einer Sprache, die nicht verwirrend und gut verständlich ist. Auch der Lebenszyklus

eines Standards sollte klar erkennbar sein. Eine fehlende Versionskontrolle und somit unklare Angaben darüber, welche Anforderungen (obligatorische und optionale) von einer bestimmten Version einer Spezifikation abgedeckt werden, sowie unzureichende Verfahren für Änderungsanträge, können sich negativ auf die Interoperabilität auswirken.

In sehr komplexen Systemen zeigen sich diese Probleme ganz besonders. Die Standardisierung dieser lässt sich selten in einem einzigen Werk zusammenfassen, sodass häufig eine größere Anzahl einzelner, kleinerer Teilspezifikationen zu einem Gesamtsystem führen. Die Standardisierung des Gesamtsystems bis ins kleinste Detail ist häufig wirtschaftlich nicht attraktiv oder zugunsten der Offenheit nicht gewollt. Das Wissen darüber, wie das System als Ganzes funktionieren soll, liegt häufig eher im Bewusstsein der Normungsgemeinschaft als in den Normen selbst. Die Problematik verschärft sich weiter, wenn die Standardisierung durch mehrere Normungsgremien erfolgt (z. B. W3C, DIF, IETF). Mögliche Ursachen der Nicht-Interoperabilität lassen sich dadurch noch schwerer nachvollziehen. Die Spezifikationen können von einer Vielzahl von Normungsgremien stammen, von denen jedes seine eigene Arbeitsweise hat und unterschiedliche Ansprüche an die Qualität der Spezifikationen stellt. Das ETSI bezeichnet dies als multiorganisatorische Normung (*Multi-Organizational Standardization*). Zusammengefasst sind die Hauptmerkmale, die zu einer unzureichenden Interoperabilität führen, die Folgenden (Veer u. a. 2008, S. 11):

Fehlender Überblick über das System Die Kombination mehrerer Normen und der darin enthaltenen Optionen, sofern sie nicht hinreichend spezifiziert und mit Querverweisen versehen sind, hindert einen Entwickler daran, einen klaren Überblick über das Gesamtsystem zu erhalten.

Verwendung von Normen über ihren ursprünglichen Zweck hinaus Es kommt immer wieder vor, dass Normen, die für einen bestimmten Zweck entwickelt wurden, für einen ganz Anderen verwendet werden. Gut durchdachte Normen sollten robust und flexibel genug sein, dieser (negativen) Entwicklung standzuhalten. In vielen Fällen führen Änderungen oder Ergänzungen der ursprünglichen Spezifikation jedoch zu Kompromissen bei der Interoperabilität, damit sie in einem zweckentfremdeten Kontext nutzbar sind. Dieses Risiko lässt sich verringern, wenn die Änderungen wohlüberlegt und gut geplant sind. Häufig werden diese allerdings in Eigenregie von den Anwendern einer Norm beschlossen, sodass die Wahrscheinlichkeit von Interoperabilitätsproblemen dadurch sehr hoch sein kann.

Unterschiedliche Qualität der Normen Jede Normungsorganisation hat ihre eigenen Regeln und ihre eigene Kultur in Bezug auf die Art und Weise, wie eine Norm verfasst und veröffentlicht wird. Das Niveau der technischen Qualität kann dabei sehr unterschiedlich sein. Dies lässt sich kaum vermeiden, ist aber für die Anwender dieser Normen problematisch.

4.2 DID-Methoden

Die in Abschnitt 3.2 vorgestellten DID-Methoden stellen eine Auswahl dar, die sich in ihren Lösungsansätzen in einem gewissen Maße unterscheiden. Viele der über 80 DID-Methoden ähneln sich sehr stark oder sind in ihren Konzepten schlicht Kopien von anderen DID-Methoden. Allen DID-Methoden ist jedoch gemein, dass sie die Konformität zur Spezifikation der Decentralized-Identifiers (Sporny, Longley, Sabadello u. a. 2021) für sich beanspruchen. Diese Spezifikation definiert allerdings lediglich einen neuen Typ eines Universal-Resource-Identifiers (URI) und der zugehörigen Architektur zur dezentralen Schlüsselverwaltung (DPKI[21]). Die Design-Ziele entsprechen weitestgehend denen, wie sie von Allen (2016) beschrieben werden (vgl. Abschnitt 2.1). Dies umfasst auch die Ziele der Dezentralität, Interoperabilität sowie der Portabilität, die im Folgenden näher untersucht werden.

Zunächst soll der Frage nachgegangen werden, was Konformität eines DID mit dieser Spezifikation bedeutet. Diese bezieht sich vor allem auf die Syntax des Identifiers (vgl. Abschnitt 2.2):

> A conforming DID is any concrete expression of the rules specified in § 3. Identifier which complies with relevant normative statements in that section. (Sporny, Longley, Sabadello u. a. 2021)

An die Syntax eines DID wird lediglich die Forderung gestellt, dass es sich um einen gültigen URI nach RFC 3986 handeln muss (vgl. Berners-Lee u. a. 2005).

> The generic DID scheme is a URI scheme conformant with [RFC3986]. (Sporny, Longley, Sabadello u. a. 2021)

Diese Syntax sieht die Bezeichnung einer DID-Methode vor, an die weitere Anforderungen gestellt werden. Bei diesen Bezeichnern handelt es sich ebenfalls um URIs. Diese müssen einzigartig sein und referenzieren auf weitere Spezifikationen, die sogenannten *DID-Methoden-Spezifikationen*. Die Einzigartigkeit dieses URI lässt sich allerdings nicht hundertprozentig sicherstellen, einzig eine Sammlung von DID-Methodenbezeichnern auf einer Website des W3C versucht hier Abhilfe zu schaffen (Steele u. a. 2021). Diese Sammlung (*DID-Specification-Registries*) verlinkt für jede DID-Methode die entsprechende Spezifikation.

Die Struktur von DID-Methoden-Spezifikationen wird von Sporny, Longley, Sabadello u. a. (2021) vorgeschrieben. Dazu gehört, wie die Autorisierung zur Ausführung der DID-Operationen erfolgt, einschließlich aller erforderlichen kryptografischen Prozesse; wie ein DID-Controller einen DID und das zugehörige DID-Document erzeugt; wie ein

[21] Decentralized Public Key Infrastructure

DID-Resolver einen DID verwendet, um ein DID-Document aufzulösen, einschließlich der Frage, wie der DID-Resolver die Authentizität der Antwort überprüfen kann; was eine Aktualisierung eines DID-Documents darstellt und wie ein DID-Controller dieses DID-Document aktualisieren, bzw. angeben kann, dass Aktualisierungen nicht möglich sind; und wie ein DID-Controller einen DID deaktivieren, bzw. angeben kann, dass eine Deaktivierung nicht möglich ist. Des Weiteren wird gefordert, dass bestimmte, konkret benannte Überlegungen zur Sicherheit und zum Datenschutz dokumentiert sein müssen. Wie dies allerdings technisch ausgestaltet ist, obliegt der Freiheit jeder einzelnen DID-Methode.

Einen großen Einfluss auf die Interoperabilität haben die Verfahren zur Erzeugung der methodenspezifischen Identifier (DID-Operation „Create“). Diese unterscheiden sich zum Teil beträchtlich voneinander (vgl. Abschnitt 3.2). Dabei kann zwischen zwei grundlegend verschiedenen Arten von Identifiers unterschieden werden. Es gibt solche, die einen mathematischen Zusammenhang zwischen einem Öffentlichen Schlüssel und dem resultierenden Identifier aufweisen, und solche, bei denen dies nicht der Fall ist. Sonderfälle der ersten Art sind die, bei denen der Öffentliche Schlüssel als Teil eines Zertifikates in die Berechnung des Identifiers einfließt. Für beide Varianten gilt aber, dass jeder zu jeder Zeit diesen Identifier berechnen kann, unter der Voraussetzung, dass der Öffentliche Schlüssel oder das Zertifikat zum Zeitpunkt der Kommunikation zugänglich sind. Es existieren auch Verfahren, die die Berechnung des Öffentlichen Schlüssels aus einer Signatur erlauben (z. B. ECDSA). Bei Identifiers, die diesen mathematischen Zusammenhang nicht aufweisen, muss es einen Mechanismus geben, der ihn an mindestens einen kryptografischen Schlüssel bindet; es sei denn, der Identifier ist der Öffentliche Schlüssel selbst (z. B. bei SSH[22] oder Key-DID). Diese sind aber aufgrund ihrer Länge zur Nutzung als Identifier oft nicht praktikabel. In bekannten Public-Key-Verfahren wie X.509 oder Pretty-Good-Privacy (PGP) werden hierfür Zertifikate verwendet. Die darin enthaltene Signatur bildet das Bindeglied zwischen einem Identifier und einem Öffentlichen Schlüssel. Ein derartiges Zertifikat kann sowohl vom Controller (self-signed/self-certifying) als auch von einem Dritten (Issuer/Certificate-Authority) signiert sein. Eine Ausnahme bildet die BTCR-DID-Methode, bei der der Identifier nicht berechnet oder durch Zertifikate an Schlüsselmaterial gebunden wird, sondern durch die Position einer Transaktion in einem Block der zugrundeliegenden Bitcoin-Blockchain bestimmt wird.

DID-Methoden unterscheiden sich nicht nur darin, was in die Berechnung eines Identifiers einfließt bzw. wie dieser an einen Öffentlichen Schlüssel gebunden wird, sondern auch in den Verfahren für jene Berechnung bzw. den Signaturverfahren, die in Zertifikaten verwendet werden. Für die Berechnung von Identifiers kommen meist Hashverfahren zum Einsatz, zum Teil auch verschiedene, die mehrfach hintereinander angewendet oder Teile davon in unterschiedlicher Weise zusammengesetzt werden. Da Hashfunktionen mit binären Daten arbeiten, ist des Weiteren eine entsprechende Codierung der

[22] Secure Shell (SSH)

resultierenden Bitfolge notwendig. Angefangen von einer simplen hexadezimalen Darstellung bis hin zu Verfahren wie Multicodec, die auch gleich die verwendeten Verfahren, die zu diesem Identifier geführt haben, codieren, ist die Vielfalt beachtlich hoch (vgl. Abschnitt 3.2). Nicht zuletzt ist die Berechnung der Öffentlichen Schlüssel selbst in den DID-Methoden unterschiedlich. Einige von ihnen beschränken sich auf ein einziges Verfahren, andere unterstützen Mehrere. Kryptografische Schlüssel bestehen meist aus mehreren Bestandteilen, z. B. dem eigentlichen Schlüssel (Exponent) und einem Modul im Falle von RSA,[23] oder den Koordinaten (x, y) eines Punktes und teils auch den verwendeten Kurvenparametern im Falle der Elliptische-Kurven-Kryptografie (*Elliptic-Curve-Cryptography*, *ECC*). Darüber hinaus ist deren Darstellung in serialisierter Form oder als Datenstruktur in den verschiedenen Verfahren unterschiedlich definiert, wie z. B. die in SEC 1 (Brown 2009, § 2.3) beschriebenen „Datentypen und Konvertierungen“ oder die eines JSON-Web-Keys (JWK).

Die Art und Weise der Berechnung von Öffentlichen Schlüsseln ist Teil der entsprechenden Signaturverfahren, die ein weiterer maßgeblicher Faktor in Bezug auf Interoperabilität darstellen und sich bis zur Schicht drei des ToIP-Stacks auswirken. Auch hier gilt für die DID-Methoden, dass sie vollkommene Freiheit in der Anzahl und Auswahl der unterstützten Verfahren haben und sie sich dahingehend erheblich unterscheiden. Die folgende Tabelle bietet eine Übersicht zu den verwendeten Verfahren einiger DID-Methoden:

Tabelle 4.1: Kryptografie-Verfahren und Parameter, Identifier und deren Codierungen von ausgewählten DID-Methoden

DID-Methode	Krypto-Systeme	EC-Parameter	Identifier	Codierung
BTCR	ECDSA	Secp256k1	TxRef	Bech32
Key-DID	N/A	N/A	Öffentlicher Schlüssel (N/A)	Multibase (Base58Btc) + Multicodec (N/A)
Peer-DID	N/A	N/A	SHA2-256	Multibase (Base58Btc) + Multicodec (SHA2-256)

[23] RSA-Kryptosystem, benannt nach Rivest, Shamir und Adleman.

DID-Methode	Krypto-Systeme	EC-Parameter	Identifier	Codierung
KERI	Ed25519, X25519, X448, Ed448, ECDSA	Curve25519, Curve448, secp256k1	SHA2-256, SHA2-512, SHA3-256, SHA3-512, Blake3-256, Blake3-512, Blake2b-256, Blake2b-512, Blake2s-256	KERI Derivation Code + Base64URL, KERI Derivation Code + Base2
ETHR	ECDSA	Secp256k1	Keccak-256	Hex-String (0x prefixed)

DID-Methode	Krypto-Systeme	EC-Parameter	Identifier	Codierung
Sovrin/Indy	Ed25519, ECDSA, RSA	Curve25519, m-221, e-222, ed1174, ed25519, ed383187, ed41417, e-382, m-383, ed448, e-521, m-511, brainpoolp160r1, brainpoolp192r1, brainpoolp224r1, brainpoolp256r1, brainpoolp320r1, brainpoolp384r1, brainpoolp512r1, secp112r1, secp128r1, secp160r1, secp192r1, secp224r1, secp256r1, secp384r1, secp521r1, secp160k1, secp163k1, secp192k1, secp224k1, secp256k1, sect163r2, sect233r1, sect239r1, sect283r1, sect409r1, sect571r1	UUID, erste 16 Byte des Öffentlichen Schlüssels	Base58
ION	ECDSA, Ed25519	secp256k1, Curve25519	SHA2-256	Multihash (SHA2-256) + Base64Url

Ein weiterer Punkt, in dem sich die DID-Methoden stark unterscheiden, sind die Mechanismen zur Ausführung der anderen DID-Operationen. Wie aus Abschnitt 3.2 her-

vorgeht, unterstützten nicht alle DID-Methoden auch alle der vier CRUD-Operationen.[24] Sofern die Operationen „Update“ und „Delete“ unterstützt werden, variieren sie in ihrer Umsetzung sehr stark. Für die Operation „Read“ legt der DID-Standard zumindest fest, dass als Antwort darauf ein DID-Document zurückzugeben ist. Dieser Vorgang wird als *DID-Resolution* bezeichnet und ist Aufgabe eines DID-Resolvers, der die methodenspezifische Operation zur Auflösung eines DIDs in ein DID-Document implementiert. Das heißt, für jede DID-Methode gibt es jeweils (mindestens) eine Implementierung eines DID-Resolvers. Eine Sammlung mehrerer dieser Implementierungen stellt der in Abschnitt 3.2 vorgestellte *Universal-Resolver* dar. Die Interoperabilität verbessert sich dadurch allerdings nicht wesentlich, unabhängig davon, ob die DID-Resolver einzeln oder als Sammlung in einem Softwarepaket verwendet werden (z. B. der Universal-Resolver). Auch wenn dadurch eine gewisse Abstraktion und Vereinheitlichung erreicht wird, in welcher Weise eine Anwendung an ein DID-Document gelangen kann, stellt letzteres lediglich Informationen zu den verwendeten Kryptografie-Verfahren bereit. Ein DID-Resolver prüft keine Signaturen, noch verändert er sie. Das bedeutet, eine Anwendung muss sämtliche Kryptografie-Verfahren aller DID-Methoden implementieren, die von ihr unterstützt werden sollen. Die schiere Anzahl an DID-Methoden und der damit verbundenen, teils gravierenden Unterschiede zur Ausführung von DID-Operationen werden dem Design-Ziel der Interoperabilität kaum gerecht. Zumal sich dies auf alle Schichten des ToIP-Stacks und somit auch bis in die Anwendungs-Ökosysteme hinein auswirkt. Verglichen mit dem TCP/IP-Stack würde das bedeuten, dass es neben den beiden heute üblichen Versionen des Internet-Protokolls (IP), IPv4 und IPv6, auch noch die Versionen IPv7 bis IPvX gibt, wobei X für die Anzahl der DID-Methoden steht. Bei gegenwärtig schon mehr als 80 spezifizierten DID-Methoden sollte dies einen kaum zu bewältigenden Aufwand darstellen. Erschwerend im Hinblick auf Interoperabilität kommt hinzu, dass das Verankerungssystem bzw. die Verifiable-Data-Registry einiger DID-Methoden, sofern sie eine vorsehen, zum Teil auch in höheren Schichten des ToIP-Stacks Anwendung finden, beispielsweise in Verbindung mit Verifiable-Credentials, deren Schemata oder anderer, methodenspezifischer Informationen. Dies ist in der Hinsicht problematisch, als dass dies weder vom DID-Standard, bzw. den DID-Methoden-Spezifikationen, noch vom Verifiable-Credentials-Data-Model erfasst wird (vgl. Abschnitt 4.3).

Allein die Tatsache, dass es mehrere DID-Methoden geben darf, widerspricht dem Gebot, die gleiche Sache nicht auf verschiedene Weise zu spezifizieren (vgl. Abschnitt 4.1). Genau das ist hierbei allerdings überwiegend der Fall. DID-Methoden spezifizieren in erster Linie Lösungen für eine ganz zentrale Aufgabe in KMS-Systemen: die Key-Rotation. Abgesehen von speziellen DID-Methoden wie Key-DID, die derartiges nicht adressieren, da sie nur für eine kurzzeitige, temporäre Nutzung vorgesehen sind („Wegwerf-DIDs“), unterstützen die meisten von ihnen entsprechende Mechanismen für Key-Rotation oder gar Key-Pre-Rotation. Zur Beschreibung dieser Mechanismen sieht

[24] Create, Read, Update und Delete bzw. Deactivate (CRUD), auch CDUR oder RUDI (Insert anstelle von Create).

der DID-Standard die DID-Operationen für entsprechende CRUD[25]-Vorgänge vor. Der Unterschied zwischen einer einfachen Key-Rotation und einer Key-Pre-Rotation liegt in der Autorisierung, jene Rotation durchführen zu dürfen. Mit ersterem ist es einem Inhaber eines Schlüsselpaares möglich, dieses durch ein neues Schlüsselpaar zu ersetzen. Voraussetzung ist dabei, dass der Zugriff auf den originären Privaten Schlüssel gewährleistet ist. Das heißt, im Falle des Verlustes oder Diebstahls eines Privaten Schlüssels ist diese Operation im Grunde nicht mehr möglich. Bei einem Diebstahl hängt es davon ab, wer die Key-Rotation zuerst ausführt – der eigentliche Eigentümer oder der Angreifer. Möchte man auch diesen Angriffsvektor ausschließen, muss ein zweites Schlüsselpaar zur „Wiederherstellung“ in irgendeiner Form von Verifiable-Data-Registry hinterlegt werden. Wobei sich auch hier die Frage stellt, wo dieses zweite Schlüsselpaar sicher gespeichert werden kann. Keine der betrachteten DID-Methoden bringt hierfür Lösungen hervor. Eine Variante, einen solchen Wiederherstellungsschlüssel anzumelden, stellt die Key-Pre-Rotation dar, wie sie z. B. in KERI oder der BTCR-DID-Methode vonstatten geht. Während der Erstellung und mit jeder Aktualisierung eines DIDs, und damit eines Schlüsselpaares, wird jeweils das nächste Schlüsselpaar angemeldet, das für DID-Operationen berechtigt ist. Vorteil an dieser Herangehensweise ist, dass der Wiederherstellungsschlüssel mit jeder DID-Operation ausgetauscht wird. Werden die Schlüsselpaare regelmäßig rotiert, vermindert das die Gefahr, dass ein Wiederherstellungsschlüssel allein durch eine lange Verwahrdauer kompromittiert werden kann. Auf der anderen Seite erhöht es möglicherweise die Gefahr der Kompromittierung, da sie regelmäßig an einen sicheren Ort übertragen werden müssen. Die ETHR-DID-Methode unterstützt unterdessen selbst keine Wiederherstellungsschlüssel oder Key-Pre-Rotation, zumindest nicht in dieser Form. Es ist aber dennoch möglich, etwas ähnliches durch die Registrierung mehrerer Schlüsselpaare für einen DID zu erreichen. Dafür lässt sich der zugrundeliegende Smart-Contract mit anderen Smart-Contracts kombinieren, die als *Multi-Signature-Wallet* agieren. Darin können Regeln definiert werden, dass z. B. M von N Schlüssel eine Transaktion signieren müssen, bevor eine Key-Rotation ausgeführt werden darf (vgl. Abschnitt 3.2). Eine andere Möglichkeit ist, die Schlüsselpaare in irgendeiner anderen Form von Datenstruktur oder direkt in den DID-Documents zu organisieren, die entweder öffentlich abrufbar sind oder zum Zeitpunkt der Kommunikation anderweitig übermittelt werden. Hierin kann die Autorisierung für bestimmte Teile dieser Datenstruktur, und damit für die DID-Operationen, festgelegt werden (z. B. Sidetree, Peer-DID). In der folgenden Tabelle sind die verfügbaren Operationen und Mechanismen für die betrachteten DID-Methoden noch einmal übersichtlich dargestellt:

Tabelle 4.2: Verfügbare DID-Operationen ausgewählter DID-Methoden

DID-Methode	Create	Read	Update	Deactivate	Key-Rotation	Key-Pre-Rotation
BTCR	x	x	x	x	-	x
Key-DID	x	x	-	-	-	-

[25] Create, Read, Update und Delete bzw. Deactivate (CRUD), auch CDUR oder RUDI (Insert anstelle von Create).

DID-Methode	Create	Read	Update	Deactivate	Key-Rotation	Key-Pre-Rotation
Peer-DID	x	x	x	x	x	x
KERI	x	x	x	x	x	x
ETHR	x	x	x	x	x	(x)
Sovrin/Indy	x	x	x	x	x	x
ION	x	x	x	x	x	x

Die DID-Methoden-Bezeichner stellen in vielen Fällen auch eine Routing-Information dar, zumindest bei den DID-Methoden, die eine Verifiable-Data-Registry vorsehen. Das bedeutet, dass ein DID zugleich an ein bestimmtes Netzwerk gebunden ist, das die relevanten Informationen für ein DID-Document vorhält. Da sich ebenso die methodenspezifischen Identifier und das zugrundeliegende Schlüsselmaterial unterscheiden, wird neben mangelhafter Interoperabilität auch das Design-Ziel der Portabilität nur bedingt erreicht. Ein DID, der einmal erzeugt und ggf. in einem Netzwerk bzw. einer Verifiable-Data-Registry registriert ist, lässt sich nicht in ein anderes Netzwerk übertragen. Je nachdem, wie diese Netzwerke und die darauf aufbauenden Ökosysteme organisiert sind bzw. wer sie betreibt, entspricht dies letztlich einem *Vendor-Lock-in*. Ein Verifiable-Credential, das auf einen DID für Netzwerk A ausgestellt ist, kann nachträglich nicht mehr auf einen DID für Netzwerk B geändert werden. Es muss entweder neu ausgestellt oder um ein abgeleitetes, selbst-zertifizierendes Verifiable-Credential ergänzt werden; wobei dies zum einen voraussetzt, dass die Anwendung eines Verifiers die Kryptografie-Verfahren der DID-Methoden beider Netzwerke beherrscht, und zum anderen, dass die Anwendung des Nutzers (Holder) einen DID aus Netzwerk B entweder selbst erzeugen und verwalten kann, oder aber diesen DID aus einer anderen Anwendung entgegennehmen und das abgeleitete sowie das originale Credential an diese zurücksenden kann.

Eine weitere, architektonische Schwäche stellen die DID-Resolver dar. Hier muss unterschieden werden, wo sie zur Ausführung kommen und wie sie eine Verbindung zu ihrem Zielnetzwerk herstellen. Grundsätzlich sollen sie sowohl lokal auf einem Endgerät als auch entfernt auf einem Server ausgeführt bzw. bereitgestellt werden können (z. B. HTTP-API). Darüber hinaus können sie selbst ein Knoten eines dezentrales Netzwerkes sein, z. B. einer Blockchain, als auch eine Verbindung zu einem dieser Knoten über ein entferntes (HTTP-)API herstellen (vgl. Abbildung 3.11). Die Daten fremder Knoten, die über ein solches API ausgeliefert werden, lassen sich kryptografisch nicht überprüfen. Das heißt, es kann nicht festgestellt werden, ob die Daten, die darüber ausgeliefert werden, tatsächlich den Daten entsprechen, die sich in diesem dezentralen Netzwerk befinden und über die netzwerkweiter Konsens besteht. Um hier Abhilfe zu schaffen, ist es üblich, mehrere dieser APIs gleichzeitig oder kurz nacheinander abzufragen und die zurückgelieferten Daten zu vergleichen. Ist der Zugang zum DID-Resolver ebenfalls nur über ein entferntes API möglich, hat auch dieser die Möglichkeit, die Daten zu manipulieren. Eine Anwendung müsste daher ebenfalls mehrere, voneinander unabhängige

DID-Resolver für jede Anfrage bemühen und die zurückgegebenen Daten wiederholt miteinander vergleichen, damit sie als vertrauenswürdig eingestuft werden können. Es kann jedoch nicht sichergestellt werden, dass dies von den Anwendungen entsprechend umgesetzt wird. Wird ein DID-Resolver entfernt ausgeführt, muss ein Client in irgendeiner Weise eine Verbindung zu ihm herstellen. Üblicherweise erfolgt dies über TCP/IP sowie HTTP oder auch HTTPS. Eine Möglichkeit ist es, die IP-Adresse eines Servers, der einen DID-Resolver ausführt, direkt im Quellcode einer Anwendung unterzubringen. Vor allem in Verbindung mit HTTPS werden aber häufig Domain-Namen verwendet, die als Identifier für X.509-Zertifikate verwendet werden. Das *Domain-Name-System (DNS)* sorgt im Internet dafür, dass Domain-Namen in IP-Adressen aufgelöst werden. X.509-Zertifikate werden von Zertifizierungsstellen (als Teil einer PKI[26]) ausgestellt und stellen die Vertrauenswürdigkeit mittels SSL/TLS[27] für HTTP-Verbindungen (HTTPS) sicher. Beide Systeme sind hierarchisch und zentralisiert. Sofern über derartige Systeme der Zugang zu dezentralen PKI-Systemen (DPKI) hergestellt wird, wie sie vom DID-Standard angestrebt werden, ist sowohl das Design-Ziel der Dezentralität verletzt, als auch insgesamt der Nutzen einer DPKI in Frage zu stellen. In Bezug auf Interoperabilität setzt eine solche Herangehensweise außerdem voraus, dass die Protokolle für DNS und HTTP/HTTPS ebenfalls in einer Anwendung implementiert sein müssen.

4.3 Verifiable-Credentials

Verifiable-Credentials (VCs) und Verifiable-Presentations (VPs) können DIDs verwenden, um bestimmte Entitäten in ihrer jeweiligen Rolle zu kennzeichnen und kryptografische Beweise über deren Kontrolle zu führen. Grundsätzlich sind auch andere Arten von Identifiers geeignet, solange die Eigentümerschaft mittels irgendeiner Form von PKI für diese Identifier gezeigt werden kann, beispielsweise Domain-Namen in Verbindung mit einem X.509-Zertifikat. Unterschiedliche Arten von Identifiers lassen sich zudem gleichzeitig in VCs und VPs benutzen. Dies gilt auch für verschiedene DID-Methoden. Um Technische Interoperabilität zu gewährleisten, ist die Implementierung der jeweiligen Mechanismen und Kryptografie-Verfahren für alle verwendeten Arten von Identifiers notwendig (vgl. Abschnitt 4.2). Für Identifier definiert das Verifiable-Credentials-Data-Model (Sporny, Longley und Chadwick 2021, § 4.2) u. a. ein optionales Attribut mit der Bezeichnung id, das in VCs und VPs sowie innerhalb von Objekten anderer Attribute dieser VCs und VPs vorkommen kann.

```
{
    "issuer": {
        "id": "did:example:76e12ec712ebc6f1c221ebfeb1f",
        "name": "Example Issuer"
    }
}
```

[26] Public Key Infrastructure (PKI)

[27] Secure Sockets Layer (SSL) / Transport Layer Security (TLS), wobei TLS der Nachfolger von SSL ist.

Enthält eines dieser Attribute keine weiteren Informationen als einen Identifier, kann dieser auch direkt als Wert für dieses Attribut verwendet werden.

```
{
    "issuer": "did:example:76e12ec712ebc6f1c221ebfeb1f"
}
```

Attribute innerhalb von VCs und VPs, die Identifier als Werte aufnehmen, sind insbesondere credentialSubject, issuer, holder, verificationMethod, credentialStatus und credentialSchema. Der VC-Standard legt weitestgehend fest, welche Datentypen und ggf. untergeordnete Attribute bei den vordefinierten Attributen erwartet werden. Genauso legt er fest, welche Attribute ein VC oder eine VP verpflichtend enthalten muss und welche optional sind. Für Attribute, in denen Objekte mit benutzerdefinierten Attributen verwendet werden dürfen, sieht der Standard eine Typisierung vor. Jedes VC und jede VP muss das Attribut type besitzen, das eine Liste von URIs als Bezeichner für bestimmte Typen von Credentials beinhaltet. Dass es sich um ein VC respektive eine VP handelt, wird durch die verbindlichen Bezeichner VerifiableCredential bzw. VerifiablePresentation ausgedrückt. Die Liste muss mindestens einen dieser Bezeichner enthalten. Mit weiteren Bezeichnern ist es möglich, den Grundtyp weiter zu präzisieren.

```
{
    "type": ["VerifiableCredential","UniversityDegreeCredential"]
}
```

Dieser Mechanismus erlaubt es Software-Entwicklern, diese Datenstrukturen in einen definierten Zustand zu bringen. Für Anwendungen, die diese konsumieren, ist dadurch klar erkennbar, welche Attribute sie erwarten können. Dies ist insbesondere für Interoperabilität wichtig. Ein Issuer dokumentiert die von ihm verwendeten Bezeichner bzw. Credential-Typen und stellt sie den Verifiers zur Verfügung, indem er sie z. B. auf der eigenen Website veröffentlicht.

VCs und VPs werden normalerweise als einfaches JSON repräsentiert, mit Ausnahme des Attributs @context. Dieses ist verpflichtend und ist in der JSON-LD-Spezifikation von Sporny, Longley, Kellogg u. a. (2020) definiert. Es muss eine geordnete Liste von URIs enthalten, wobei der erste URI auf https://www.w3.org/2018/credentials/v1 festgelegt ist. Dahinter verbirgt sich eine statische Datei, die nicht mehr verändert wird und weitere Informationen zu den Typen-Bezeichnern und den zugehörigen Attributen sowie deren Datentypen und Bedeutungen (Semantik) enthält (Sporny, Longley und Chadwick 2021, § 6.2). Es können weitere URIs in die Liste aufgenommen werden, um benutzerdefinierte Attribute näher zu beschreiben. Dies ist vor allem für das Attribut credentialSubject und dessen Wert relevant, das ein Objekt mit weiteren Attributen enthält, die über die Beschreibungen von VCs und VPs hinausgehen. Dies sind die eigentlichen Claims, die ein Credential ausdrückt.

Neben JSON und JSON-LD erlaubt der VC-Standard auch andere Repräsentationen wie z. B. XML,[28] YAML[29] oder CBOR.[30] Die Spezifikation stellt keine Anforderungen an die Unterstützung eines bestimmten Repräsentations- oder Serialisierungsformats. (Sporny, Longley und Chadwick 2021, § 6). Dies betrifft insbesondere die syntaktische Interoperabilität, wenn diese Anforderungen nicht gestellt werden. Sofern zwei Anwendungen unterschiedliche Formate verwenden und nur jeweils eines davon unterstützen, ist Interoperabilität nicht gegeben. Ähnlich verhält es sich mit der semantischen Interoperabilität. Die Typisierung und die Kontextinformationen erlauben es, dass die Bedeutungen und Datentypen in VCs und VPs klar gekennzeichnet werden können. Sie verweisen aber letztlich auf gewöhnliche Webseiten, die an Entwickler gerichtet sind. Zwei Anwendungen sind nur dann interoperabel, wenn sie Typ, Kontext und Datenformat gegenseitig verstehen. Dies bedeutet allerdings stets zusätzlichen Implementierungsaufwand.

Problematisch hinsichtlich Interoperabilität sind außerdem die Attribute credentialStatus und proof. Ersteres stellt Informationen über den aktuellen Status eines VCs bereit, z. B. ob es ausgesetzt oder widerrufen wurde (Revocation). Auch hier wird die Typisierung durch Identifier in Form eines URIs verwendet. Das Attribut enthält ein Objekt mit den Attributen id und type:

```
"credentialStatus": {
    "id": "https://example.edu/status/24",
    "type": "CredentialStatusList2017"
}
```

Der Identifier des Typs dokumentiert in diesem Beispiel (CredentialStatusList2017) eine einfache Datenstruktur, die ein Issuer auf seiner Website veröffentlichen kann (Sporny 2020). Das Attribut id ist hierbei ein Verweis auf eine dieser Datenstrukturen eines Issuers, nicht auf die Dokumentation jener. Letztere werden ähnlich wie die DID-Methoden auf einer eigenen Website des W3C gesammelt, die als *Registry* für verschiedene Widerrufsmechanismen dient (*Verifiable Credentials Extension Registry* 2021, § 3.2). Bislang ist hier nur der o. g. Typ gelistet und sowohl die Registry als auch die Dokumentation dieses Typs sind als inoffizieller Entwurf gekennzeichnet. Der Abruf der Statusliste erfolgt in diesem Beispiel mittels HTTP bzw. HTTPS (https://example.edu/status/24). Als weiteres Beispiel wäre es denkbar, einen *Smart-Contract* in einer Blockchain als Statusliste zu verwenden (Menzer u. a. 2021, § 4.4):

```
"credentialStatus": {
    "id": "0x06012c8cf97bead5deae237070f9587f8e7a266d",
    "type": "EthereumMainnetStatusList2021"
}
```

[28] Extensible Markup Language (XML)
[29] YAML Ain't Markup Language (YAML™)
[30] Concise Binary Object Representation (CBOR)

In diesem Fall würde für den Typ-Bezeichner (EthereumMainnetStatusList2021) dokumentiert sein, um welche Blockchain und um welches konkrete Netzwerk dieser Blockchain (Mainnet, Testnetzwerke) es sich handelt; und wie genau mit dem Smart-Contract interagiert werden kann. Das Attribut id gibt hierbei die Adresse des Smart-Contracts im jeweiligen Netzwerk an. Diese Art der Typisierung erlaubt es demnach, viele weitere Mechanismen für die Revocation innerhalb eines VCs zu verwenden (Content-Addressable-Storage, Cryptographic-Accumulator, etc.). Der Zugang zu den Speichersystemen und die Art und Weise der Verifikation kann sich dabei stark unterscheiden. Im Falle des Cryptographic-Accumulator wird beispielsweise nicht nur eine Liste oder ein Wert abzurufen. Die Verifikation erfolgt hierbei durch einen Zero-Knowledge-Proof, mit dem ein Holder einen Proof-of-Non-Revocation erbringen kann (vgl. Abschnitt 3.2).

Das Attribut proof enthält das untergeordnete Attribut verificationMethod, das wiederum einen Identifier beinhaltet, und dient dazu, die Kontrolle über diesen Identifier nachzuweisen. Je nachdem, welche Art von Identifier verwendet wird, unterscheidet sich auch die Art und Weise, wie dieser Nachweis gezeigt werden kann. In VCs handelt es sich bei diesen Identifiers häufig um DIDs. Wie in Abschnitt 4.2 aufgezeigt wird, ist mit der Vielzahl an DID-Methoden eine noch größere Anzahl an kryptografischen Verfahren verbunden, mit denen die Kontrolle über einen DID bewiesen werden kann. Bei DIDs lassen sich die unterstützten Verfahren normalerweise direkt durch den DID-Methoden-Bezeichner (URI) und ggf. im methodenspezifischen Identifier erkennen. Letzteres ist dann notwendig, wenn eine DID-Methode mehrere Verfahren erlaubt, sodass dies in irgendeiner Weise im Identifier selbst codiert sein muss (Multicodec o. ä.). VCs und VPs unterstützen aber auch andere Arten von Identifiers, bei denen dies nicht zu erkennen ist. Beispielhaft seien hier URLs in Verbindung mit X.509-Zertifikaten genannt. Aus diesem Grund sieht das Attribut proof ebenfalls eine Typisierung vor, die das verwendete Verfahren für diesen Beweis kennzeichnet. Das untergeordnete Attribut type nutzt hierfür ebenfalls URIs, also eindeutige Bezeichner, die entsprechend dokumentiert werden. Eine rudimentäre Sammlung dieser Bezeichner findet sich in der *Verifiable Credentials Extension Registry* (2021), § 3.1. Einige andere sind in der *Linked Data Cryptographic Suite Registry* (2020) aufgelistet. Die Dokumentation der einzelnen Bezeichner und ihrer Kryptografie-Verfahren beschreibt zudem, welche untergeordneten Attribute innerhalb des Attributs proof erwartet werden können.

```
"issuer": "https://example.com/issuer/123#ovsDKYBjFemIy8DVhc-...
"proof": {
    "type": "JsonWebSignature2020",
    "created": "2019-12-11T03:50:55Z",
    "jws": "eyJhbGciOiJFZERTQSIsImI2NCI6ZmFsc2UsImNyaXQiOlsiY...
    "proofPurpose": "assertionMethod",
    "verificationMethod": "https://example.com/issuer/123#ovs...
}
```

In diesem Beispiel wird als Identifier ein URL benutzt. Damit der Beweis einem Identi-

fier eines Attributs id innerhalb eines VCs oder einer VP zugeordnet werden kann, wird der Identifier noch einmal im Attribut verificationMethod angegeben. Andere Verfahren können entsprechend auch andere Bezeichnungen und/oder Werte ihrer Attribute aufweisen:

```
"proof": {
    "type": "BbsBlsSignature2020",
    "created": "2020-04-25DE",
    "verificationMethod": "did:example:489398593DE#test",
    "proofValue": "F9uMuJzNBqj4j+HPTvWjUN/MNoe6KRH0818WkvDn2S...
    "proofPurpose": "assertionMethod",
    "requiredRevealStatements": [ 4, 5 ]
}
```

Der VC-Standard legt weder für die Revocation- noch für die Beweismechanismen konkrete Verfahren fest. Die Beweise werden mittels kryptografischer Signaturen erbracht und sind eng an das Schlüsselmaterial gekoppelt, das für einen Identifier verfügbar ist. Im Falle von DIDs werden die verfügbaren Öffentlichen Schlüssel eines einzelnen DIDs in einem zugehörigen DID-Document ausgedrückt. Demzufolge wirken sich die Identifier der Schicht eins des ToIP-Stacks unmittelbar auf VCs und VPs, bzw. deren Signaturen, und damit auf Schicht drei aus. Das Gebot, dieselbe Sache nicht auf verschiedene Weise zu spezifizieren, wird demnach auch hier nicht eingehalten. Gleiches gilt für die Revocation. Unterschiedliche Methoden oder Verfahren für bestimmte Dinge zu ermöglichen und diese durch eine Typisierung anzukündigen ist dennoch wichtig. Würde dies nicht berücksichtigt werden, gäbe es keine Möglichkeit mehr, jene Methoden im Laufe der Zeit modular durch Andere, Modernere zu ersetzen. Für Interoperabilität ist es jedoch wichtig, einen verpflichtenden Satz an Verfahren vorzuschreiben und Andere von vornherein auszuschließen, insbesondere die, die nach aktuellem Stand der Technik bereits als veraltet gelten oder bei denen gravierende Sicherheitsprobleme bekannt sind. Die Problematik verschärft sich weiter, wenn neben den VCs und VPs nach Sporny, Longley und Chadwick (2021) noch andere Arten Credentials verwendet werden, wie es bei Sovrin bzw. im Hyperledger-Indy und Hyperledger-Aries-Umfeld der Fall ist. Hier kommen außerdem sogenannte *AnonCreds* zum Einsatz, bei denen der Beweis über die enthaltenen Attribute in Form eines Zero-Knowledge-Proofs erbracht wird (vgl. Hardman 2021; Khovratovich und Lodder 2018).

4.4 Wallets und Agents

Wallets und Agents sind die Software-Komponenten, die sowohl DID-Methoden als auch Verifiable Credentials (VCs) und Verifiable Presentations (VPs) implementieren. Als Wallet wird häufig der Teil einer Software bezeichnet, der Private Schlüssel sicher speichert sowie Datenstrukturen wie VCs und VPs kryptografisch signiert und eine Komponente eines Agents ist. Letzterer wird meist mit Funktionalitäten für die Konnektivität

zwischen Entitäten in Verbindung gebracht (vgl. Abschnitte 2.3 und 2.4). Es gibt auch Auslegungen, bei denen der Begriff Wallet mit einer Anwendung als Ganzes bzw. mit einer App assoziiert wird, die wiederum die Funktionalitäten eines Agents beinhaltet (Preukschat u. a. 2021, S. 210). Letztlich ist beides Software, unabhängig davon, ob das eine Teil des anderen ist oder ob beide als Teil einer Anwendung gesehen werden. Für die Verarbeitung von VCs und VPs, deren Identifier meist durch DIDs repräsentiert werden, sowie deren Austausch über ein Netzwerk, sind beide Komponenten untrennbar miteinander verbunden. Aus den vorherigen Abschnitten geht hervor, dass DIDs und auch andersartige Identifier unmittelbaren Einfluss auf VCs und VPs bzw. deren Implementierung und damit auf eine Anwendung haben. Schicht eins des ToIP-Stacks, in denen die Identifier und ggf. deren Public-Utilities angesiedelt sind, wirken sich demnach auch auf die Schichten drei und vier aus (vgl. Abschnitt 3.1).

Gleiches gilt für die Schicht zwei des ToIP-Stacks, die vor allem durch das DIDComm-Messaging definiert ist. Wie der Name andeutet, sind DIDs elementar mit diesem Protokoll verbunden. Damit geht einher, dass eine Implementierung eines Agents ausgewählte DID-Methoden für DIDComm benutzt. Da die Nachrichten signiert und ggf. verschlüsselt sein können, ist das an die DIDs gekoppelte Schlüsselmaterial entsprechend unterschiedlich (vgl. Abschnitte 3.2 und 4.2). DIDComm ist zudem in zwei Versionen dokumentiert: DIDComm-V1 und DIDComm-V2 (vgl. Abschnitt 3.1). Zur Unterscheidung führt die aktuelle Spezifikation Profile ein, die sich über eindeutige Bezeichner nach dem Vorbild der IANA-Medientypen[31] identifizieren lassen. Für Abwärtskompatibilität sind bereits vier dieser Bezeichner definiert (Curren u. a. 2021, § 4.1):

`didcomm/aip1` die Datenstrukturen zur Verschlüsselung (*Envelope*) und Signaturverfahren, Nachrichtenformat und Routing-Algorithmen entsprechen dem Aries-Interop-Profile (AIP) der Version 1.0 (DIDComm-V1).

`didcomm/aip2;env=rfc19` Signaturverfahren, Nachrichtenformat und Routing-Algorithmen entsprechen dem AIP der Version 2.0. Die zur Verschlüsselung genutzten Datenstrukturen richten nach dem älteren Aries-RFC 0019 (DIDComm-V1).

`didcomm/aip2;env=rfc587` Signaturverfahren, Nachrichtenformat und Routing-Algorithmen entsprechen dem AIP der Version 2.0. Die zur Verschlüsselung genutzten Datenstrukturen richten nach dem neuerem Aries-RFC 0587 (DIDComm-V1).

`didcomm/v2` die Datenstrukturen zur Verschlüsselung (*Envelope*) und Signaturverfahren, Nachrichtenformat und Routing-Algorithmen entsprechen der aktuellen DIDComm-Spezifikation, die von der DIF vorangetrieben wird (DIDComm-V2).

Die Angabe dieser Bezeichner erfolgt in den DID-Documents der jeweiligen DIDs, genauer in der Beschreibung der Service-Endpoints. Damit kann ein Dienst zum Ausdruck

[31] Internet Media Type, auch MIME-Type (Multipurpose Internet Mail Extensions) oder Content-Type.

bringen, welche der Profile ein angegebener Endpunkt unterstützt. Handelt es sich bei diesem Endpunkt um einen Vermittler (*Mediator*), der die Nachrichten ggf. über weitere Vermittler weiterleitet (Routing), wird davon ausgegangen, dass alle nachgelagerten Service-Endpoints die angegebenen Profile beherrschen (Curren u. a. 2021, $ 4.1). Für vollständige Interoperabilität müssen Implementierungen folglich alle Profile für die unterschiedlichen Kommunikationswege unterstützen.

DIDComm soll darüber hinaus unabhängig von den zugrundeliegenden Transportprotokollen sein (vgl. Abschnitt 3.2). In den Anforderungen zu DIDComm werden eine ganze Reihe unterschiedlicher Transportmöglichkeiten genannt: HTTP(S), WebSockets, Bluetooth, Chat, Push-Benachrichtigungen, libp2p,[32] AMQP,[33] SMTP, NFC, Sneakernet[34] und Snail-Mail (ebd., § 1.2). Nähere Angaben zu konkreten Transportprotokollen, die auch Teil der eigentlichen Spezifikation sind, werden nur für HTTP(S) und WebSockets gemacht (ebd., § 10.3). Sie sind jedoch nicht verbindlich. Für eine DIDComm-Implementierung ist somit nicht sichergestellt, dass sie ein Transportprotokoll oder eine sonstige Art eines Endpunktes, die sie in einem DID-Document vorfindet, tatsächlich versteht. Wird nicht konkret festgelegt bzw. standardisiert, wie die Kommunikation mit einem Endpunkt abläuft oder wie z. B. eine DIDComm-Nachricht in einer SMTP-Nachricht (E-Mail) unterzubringen ist, kann Interoperabilität nicht gewährleistet werden.

DIDComm dient weiterhin als Grundlage für andere, darauf aufbauende Protokolle, die beispielsweise für den Austausch von VCs und VPs genutzt werden können. Derartige Protokolle werden gemeinhin der Schicht drei des ToIP-Stacks zugeordnet (vgl. 3.1), aber letztlich von einem Agent implementiert. Im Hyperledger-Aries-Umfeld sind dies:

- Aries-RFC 0453: Issue Credential Protocol 2.0 (vormals RFC 0036 Issue Credential v1.x / Indy HIPE PR #89),
- Aries-RFC 0454: Present Proof Protocol 2.0 (vormals RFC 0037: Present Proof Protocol 1.0 / Indy HIPE PR #89).

Mit diesen Protokollen gehen eine Reihe weiterer Spezifikationen einher, die beispielsweise zugehörige Datenstrukturen definieren bzw. ein Teil von DIDComm-V1 sind (Khateev, Klump u. a. 2021; Khateev und Curran 2021):

- Aries-RFC 0008: Message ID and Threading,
- Aries-RFC 0017: Attachments,
- Aries-RFC 0043: l10n (Locali[s|z]ation),
- Aries-RFC 0075: Payment Decorators,
- Aries-RFC 0317: Please ACK Decorator,

[32] Sammlung von Protokollen, Spezifikationen und Software-Bibliotheken zur Entwicklung von Peer-to-Peer-Anwendungen.

[33] Advanced Message Queuing Protocol (AMQP)

[34] Informeller Begriff für die Übertragung elektronischer Informationen über physische Medien (z. B. USB-Stick).

- Aries-RFC 0510: Presentation-Exchange Attachment format for requesting and presenting proofs,
- Aries-RFC 0511: Credential-Manifest Attachment format for requesting and presenting credentials,
- Aries-RFC 0592: Indy Attachment Formats for Requesting and Presenting Credentials,
- Aries-RFC 0593: JSON-LD Credential Attachment format for requesting and issuing credentials,
- Aries-RFC 0646: W3C Credential Exchange using BBS+ Signatures.

Aries-RFC 0510 basiert auf der Spezifikation *Presentation-Exchange* (2.0.0) der DIF (Aristy 2020). Andere Protokolle oder Spezifikationen für den Austausch von VCs bzw. VPs stellen das Credential-Handler-API (CHAPI), das Self-Issued-OpenID-Connect-Provider-DID-Profile (SIOP-DID/OIDC) oder auch Inter-Wallet-Credential-Exchange (IWCE) dar (vgl. Abschnitt 3.2). Diese nutzen für den Transport allerdings nicht DIDComm, sondern verwenden direkt HTTP bzw. HTTPS. Es gibt jedoch einen Vorschlag von Burlacu (2019), der es z. B. erlaubt, OIDC-Tokens mittels HTTP(S) innerhalb von DIDComm-Nachrichten zu übermitteln (Aristy 2020). Inwieweit es sinnvoll ist, ein synchrones Request-Response-Protokoll (duplex) wie HTTP(S) in DIDComm zu verpacken – ein Protokoll das überwiegend asynchron und simplex arbeitet – ist fraglich. Dauert die Übermittlung per DIDComm zu lange, sind häufige *Timeouts* der getunnelten HTTP(S)-Kommunikation die Folge. Zum einen lässt sich durch derartige Lösungen die Interoperabilität heterogener Systeme möglicherweise steigern, auf der anderen Seite kann die höhere Komplexität, die dadurch entsteht, wiederum zu einer Verringerung derer führen. Grundsätzlich sind Presentation-Exchange, CHAPI und IWCE untereinander nicht interoperabel. Interoperabilität kann nur dadurch entstehen, dass eine Anwendung alle diese Protokolle unabhängig voneinander implementiert (vgl. Abschnitt 4.2).

Ein weiterer wesentlicher Faktor hinsichtlich Interoperabilität zwischen Agents stellt die Semantik von VCs dar (vgl. Abschnitt 4.3). Eine Implementierung, die zwar mit VCs und VPs umgehen kann, versteht noch längst nicht die Bedeutung der eigentlichen Claims innerhalb eines VCs. Die Typ-Bezeichner der VCs (type) bringen diese Datenstrukturen lediglich in einen definierten Zustand. Zwei interoperable Agents müssen ihre Typen gegenseitig kennen, einschließlich der Bedeutung aller Attribute, die ein Credential enthält. Darüber hinaus muss ein Credential in der Regel in irgendeiner Weise auch in einer grafischen Benutzeroberfläche[35] dargestellt werden. Hierfür bedarf es weiterer Standardisierungsbemühungen, die sowohl bestimmte Credential-Typen, deren Darstellung als auch die Semantik der darin enthaltenen Attribute für die einzelnen Claims festlegen. Letztere werden beispielsweise durch Initiativen wie Schema.org gesammelt und veröffentlicht.

[35] Graphical User Interface (GUI)

5 Fazit

Die Identitätsverwaltung, wie sie heute üblich ist, ist gewiss nicht optimal. Auf der einen Seite sind die Nutzer mit einer großen Anzahl von Datensilos konfrontiert, da der Zugang zu jedem einzelnen Dienst normalerweise durch die Kombination eines Benutzernamens und einem Passwort geschützt ist. Dies führt zu einem hohen Verwaltungsaufwand dieser Zugangsdaten auf beiden Seiten und führt zu einer kaum beherrschbaren Streuung der eigenen persönlichen Daten. Auch für Diensteanbieter birgt dieser Ansatz ein gewisses Risiko und konfrontiert sie mit einer hohen Verantwortung, die Daten ihrer Kunden angemessen zu schützen. Ein weiteres bekanntes Problem in diesem Zusammenhang stellen schwache Passwörter dar, die teilweise seitens der Nutzer über mehrere Dienste hinweg genutzt werden. Auf der anderen Seite haben in den letzten Jahren zentralisierte Identitätsprovider (IdP) an Popularität gewonnen, die die Identitätsdaten ihrer Kunden auch für andere bzw. dritte Dienste bereitstellen und einen Login über mehrere Dienste hinweg erlauben. Bekannte Vertreter solcher Dienste sind Google, Facebook, GitHub oder LinkedIn. Letztlich ist dies aber nicht wesentlich besser, als dass ein Nutzer ein und dieselbe Kombination aus Benutzernamen und Passwort für mehrere Dienste verwendet. Werden diese Informationen gestohlen, erhält ein Angreifer augenblicklich Zugang zu allen verknüpften Diensten. Sofern es eine Möglichkeit gibt, die Kontrolle über das Konto des IdPs zurückzuerlangen, erhält der Nutzer zumindest schneller die Kontrolle über die verbundenen Konten anderer Dienste zurück. Dies kann grundsätzlich als Vorteil angesehen werden, der allerdings teuer erkauft wird: der zentralisierte IdP wickelt sämtliche Login-Vorgänge eines Nutzers ab. Er ist somit in der Lage, sämtliche Interaktionen für die Dienste, die der betreffende Nutzer verwendet, zu protokollieren. Eine weitere schwerwiegende Schwachstelle dieses Konzepts ist die Problematik des sogenannten *Single-Point-of-Failure*. Sind die Dienste des IdP gestört oder kompromittiert, ist der Nutzer augenblicklich handlungsunfähig.

Als Alternative zu diesen beiden Konzepten wird gegenwärtig das Konzept der Self-Sovereign-Identity (SSI) bzw. des Trust-over-IP-Stacks forciert. Als technologische Basis dienen die beiden Standards, bzw. deren Entwürfe, zu den Decentralized-Identifiers (DIDs) sowie den Verifiable-Credentials (VCs). Die Design-Ziele richten sich dabei u. a. nach den Ausführungen von Allen (2016). Der Schwerpunkt dieser Arbeit lag dabei insbesondere auf dem Design-Ziel der Interoperabilität, wobei dies auch eng mit denen der Portabilität und der Dezentralität in Zusammenhang steht; ganz besonders dann, wenn sich SSI als Alternative zu den heutigen, zentralisierten Lösungen versteht. Neben diesen beiden grundlegenden Standards haben sich viele weitere Spezifikationen herausgebildet, die bestimmte Probleme adressieren und in Abschnitt 3.2 kurz vorgestellt wurden. Auf den Begriff der Interoperabilität wurde in Abschnitt 4.1 näher eingegangen und die Schwierigkeiten während eines Standardisierungsprozesses aufgezeigt. Es wurde gezeigt, dass vor allem der DID-Standard mit seiner Offenheit gravierende

Mängel aufweist und in der Folge eine kaum überschaubare Anzahl an DID-Methoden beim W3C registriert worden sind. Dabei treffen die vom ETSI genannten Hauptprobleme auf diesen Standard zu. Die Vielzahl an kryptografischen Verfahren, Codierungen, Netzwerken usw. in Verbindung mit vielen weiteren Mechanismen, die darüber hinaus für ein Identitätsverwaltungssystem benötigt werden, erschweren den Überblick über das Gesamtsystem erheblich. Hinzu kommt, dass die verschiedenen Spezifikationen unter der Aufsicht unterschiedlicher Gremien stehen und sie zum Teil von Einem zu einem Anderen übergegangen sind (z. B. von Hyperledger bzw. der Linux-Foundation zur Decentralized-Identity-Foundation im Falle von DIDComm). Auch die Qualität der Spezifikationen weist große Unterschiede auf. Viele davon befinden sich noch im Status eines inoffiziellen Entwurfs, sind teils unvollständig oder überschneiden sich.

DID-Methoden sind de facto untereinander weder interoperabel noch portabel. Die Abhängigkeiten ziehen sich durch alle Schichten des ToIP-Stacks, angefangen von den DID-Operationen der DID-Methoden selbst (Schicht 1), über DIDComm (Schicht 2), den Verifiable-Credentials und Verifiable-Presentations (Schicht 3), bis hin zur eigentlichen Anwendung (Schicht 4). Für vollständige Interoperabilität müsste ein Entwickler letztlich mehr als 80 DID-Methoden in seiner Anwendung implementieren. In den Abschnitten 4.3 und 4.4 wurden zudem einige weitere Faktoren auf den verschiedenen Schichten aufgezeigt, die die Interoperabilität ebenfalls negativ beeinflussen. Insbesondere die Mechanismen zur Revocation von Credentials und die verwendbaren Kommunikations- und Austauschprotokollen der verschiedenen Lösungen sind problematisch. Solange keine Konsolidierung stattfindet, handelt es sich hierbei weitestgehend um Insellösungen. Auch die Mozilla-Foundation hat ihre Bedenken bereits in einer Mailing-List zum Ausdruck gebracht und plädiert dafür, den DID-Standard vom Status der *Recommendation* auf den Status des *Working-Drafts* zurückzusetzen (Çelik 2021).

Mit dem Verifiable-Credentials-Data-Model lassen sich Behauptungen (Claims) sehr einfach ausdrücken und mittels digitaler Signaturen in dezentraler Weise überprüfen. Vor allem JSON bzw. JSON-LD ist leicht verständlich und sowohl für Menschen als auch für Maschinen gut lesbar. Der VC-Standard legt allerdings nicht verbindlich fest, dass diese beiden Repräsentationen von einer Implementierung unterstützt werden müssen. Ein Entwickler kann für seine Anwendung auch entscheiden, dass er XML, YAML oder binäre Formate wie CBOR benutzt (vgl. 4.3). Hier bleibt abzuwarten, wie stark die Anzahl an verschiedenen Daten-Repräsentationen bzw. Formaten, Mechanismen zur Credential-Revocation sowie Protokollen für Austausch und Transport in den Implementierungen bzw. Spezifikationen weiter anwächst. Sofern dies überschaubar bleibt, kann zumindest die technische und syntaktische Interoperabilität gelingen. Die semantische Interoperabilität für tausende von Anwendungsfällen zu gewährleisten wird sich wahrscheinlich nicht umsetzen lassen.

Insgesamt sollte SSI und der ToIP-Stack kritisch betrachtet werden. Es zeigt sich, dass die angestrebten Ziele der Interoperabilität, Portabilität und Dezentralität in der Praxis

nur bedingt erreicht werden können. In der Hauptsache ist hierbei zu bemängeln, dass die beiden wesentlichen Standards der Decentralized-Identifiers und des Verifiable-Credentials-Data-Models die Mindestanforderungen einer konformen Implementierung nicht konkret spezifizieren. Was die Dezentralität anbetrifft, sind u. a. mit den DID-Resolvern architektonische Mängel zu verzeichnen. Dezentralität betrifft aber nicht nur die Technologie, sondern auch organisatorische Aspekte und die Stakeholder, die diese Technologien entwickeln und Instanzen derer betreiben.

Die Ausführungen von Davie u. a. (2019) enthalten zudem die bemerkenswerte Aussage, dass mit DIDs und ihren Verifiable-Data-Registries ein „strenger Root-of-Trust“ gegeben sei (vgl. Abschnitt 3.1). Einen DID in einer Blockchain o. ä. zu registrieren schafft jedoch noch kein Vertrauen (Trust). Die Berechtigung über die jeweiligen Schreibvorgänge bzw. DID-Operationen auf einem DID-Document liegen immer beim Holder. Dieser kann dort grundsätzlich alles mögliche hinein schreiben. Es können z. B. Service-Endpoints eingetragen werden, die einen Nutzer eines DIDs durch einen Domain-Namen oder eine E-Mail-Adresse identifizieren. Das eigentliche Vertrauen fußt dann aber mehr auf Systemen wie DNS oder X.509-Zertifizierungsstellen, die letztlich wieder zentralisierte und hierarchische Systeme darstellen. DIDs identifizieren letztlich nur Software-Endpunkte. Um auszudrücken, welches Individuum sich hinter einem Endpunkt verbirgt, sind grundsätzlich Verifiable-Credentials geeignet. Sie enthalten persönliche Informationen, die ein Dritter (Issuer) mit einer digitalen Signatur bestätigt. Daraus ergibt sich aber das gleiche Problem wie bei X.509: Wo bzw. wer ist der Root-of-Trust? Wer bestätigt, dass dieser Dritte derjenige ist, der er vorgibt zu sein; dass er entsprechende Aussagen über andere treffen überhaupt darf und welche Qualität seine Aussagen bezüglich der realen Welt haben? Es verbleiben an dieser Stelle nur die beiden bekannten Ansätze so etwas zu organisieren: in Form einer Hierarchie (z. B. X.509) oder in Form eines Web-of-Trust (WoT). Inwieweit neuartige Technologien wie Blockchains oder DLTs[36] hierzu einen Beitrag leisten können, bleibt abzuwarten. Wenn, dann läuft es darauf hinaus, dass für bestimmte Vorgänge ein erheblicher monetärer Betrag zu entrichten sein wird, um z. B. Sybil-Attacken[37] standzuhalten. Das heißt, das Erstellen eines Identifiers oder die gegenseitige Vertrauensbekundung durch Signaturen bzw. deren Veröffentlichung müsste so teuer sein, dass die Wahrscheinlichkeit von Missbrauch dadurch abnimmt und Reputation entstehen kann. Es müssten außerdem per Protokoll monetäre Anreize geschaffen werden, die einen vertrauenswürdigen Betrieb solcher Systeme gewährleisten, oder dies zumindest begünstigen. Bislang ist dies nur mit Proof-of-Work-basierten Blockchains wie Bitcoin o. ä. gelungen. Dass konsortial organisierte Blockchains bzw. DLTs wie Hyperledger-Indy bzw. Sovrin diese Probleme lösen können, darf angezweifelt werden.

Es ist außerdem die Frage zu stellen, inwieweit ein Holder (Nutzer) tatsächlich die Kon-

[36] Distributed Ledger Technology (DLT)

[37] Als Sybil-Attacke bezeichnet man in der Computersicherheit eine Attacke auf Peer-to-Peer-Netzwerke durch die Erstellung falscher Identitäten.

trolle über einen Identifier bzw. einen DID hat. Die wenigsten Nutzer werden eine Münze werfen, um ihre Privaten Schlüssel zu erzeugen. Sie sind in den meisten Fällen von der Vertrauenswürdigkeit einer Software abhängig, die sie zur Verwaltung ihrer digitalen Identität benutzen. Jene Software hat außerdem Zugriff auf die Privaten Schlüssel. Betrachtet man nun noch die Stakeholder, die diese SSI-Lösungen vorantreiben, wird es beinahe absurd. Dies sind u. a. Unternehmen wie Evernym, Digital Bazaar, Trinsic Technologies oder Jolocom, die Anwendern entsprechende Software für diese Systeme zur Verfügung stellen. Jene Software wird oft in Form einer App für mobile Betriebssysteme angeboten und als Produkt unter der Bezeichnung *Wallet* oder *Identity-Wallet* vertrieben. Das Ziel ist es, möglichst viele Identitätsdaten der Nutzer in einer dieser Apps zu vereinen, damit er seine Daten darin bequem verwalten kann. Aus Sicht des Nutzers sind die Anbieter dieser Apps jedoch genauso zentral wie die, die heutzutage in zentralisierten Identitätssystemen vorzufinden sind. Somit ließe sich folgender Schluss fassen: *Just another Google!*[38] Aus technologischer Sicht verhindert SSI weder die Problematik der Protokollierung noch die des Single-Point-of-Failure. Ersteres wird zumindest ein wenig durch den SSI-Gedanken entschärft. Es ist im Vergleich zu den heute im Einsatz befindlichen Protokollen (OAuth, OpenID-Connect, SAML, etc.) ein Fortschritt, wenn nicht jeder Login-Vorgang über die Server-Systeme des entsprechenden Anbieters abgewickelt wird, sondern lokal auf den Endgeräten stattfindet. Hundertprozentige Sicherheit gegen Protokollierung bietet das aber auch nicht. Der Hersteller eines Identity-Wallets ist immer in der Lage, die Vorgänge auch lokal zu protokollieren und an die eigenen Server-Systeme zu übermitteln. Es lässt sich aber immerhin messen und die serverseitige Kommunikation ist kein integraler Bestandteil des Protokolls selbst.

Die Nutzer, bzw. Menschen im Allgemeinen, sind der binären Sprache normalerweise nicht mächtig. Die Repräsentanten seiner digitalen Identität sind die Geräte, die sie nutzen. Es wird in Zukunft mehr darauf ankommen, wie sich diese Geräte untereinander synchronisieren können – im besten Fall nach den Prinzipien der Peer-to-Peer-Kommunikation. Um die Probleme zu lösen, die mit der Verwaltung Privater Schlüssel einhergehen, müssen zwangsläufig mehrere Geräte unter der Kontrolle eines Individuums stehen. Bei Verlust, Diebstahl oder Kompromittierung eines Gerätes bzw. des darauf gespeicherten Schlüssels kann er jeweils auf ein anderes Gerät zurückgreifen, um Ersteres zu sperren. Es lassen sich damit außerdem Szenarien für eine Zwei-Faktor-Authentifizierung abbilden. Es müsste ein Layer oder eine *Best-Practice* geschaffen werden, mit dem sich Gruppen von Geräten bilden lassen und bei denen die Kommunikation unter den Geräten bzw. Anwendungen möglichst direkt vonstatten geht. Die persönlichen Daten eines Nutzers sollten sich über mehrere Anwendungen und Stakeholder verteilen, die mit einem solchen Layer die darin enthaltenen Daten synchronisieren können.

Die Grundsätze der Datensparsamkeit, -Privatheit und -Sicherheit lassen sich aber durch Technologie allein nicht vollumfänglich sicherstellen. Die Technologie sollte je-

[38] Google ist hierbei als einer der Vertreter dieser Systeme zu sehen

doch so konzipiert sein, dass sich Missbrauch zumindest messen lässt. Ausschlaggebend sind daher weiterhin die Rahmenbedingungen des Gesetzgebers und dessen Möglichkeiten, die Umsetzung seiner Gesetze zu kontrollieren und durchzusetzen.

Literaturverzeichnis

Allen, Christopher (25. Apr. 2016). *The Path to Self-Sovereign Identity*. URL: http://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html (besucht am 27. 10. 2021).

Allen, Christopher u. a. (8. Aug. 2019). *BTCR DID Method*. URL: https://w3c-ccg.github.io/didm-btcr/ (besucht am 05. 01. 2022).

Allowing Apps (2021). *Allowing Apps and Websites to Link to Your Content. Apple Developer Documentation*. URL: https://developer.apple.com/documentation/uikit/inter-process_communication/allowing_apps_and_websites_to_link_to_your_content (besucht am 11. 11. 2021).

Antonopoulos, Andreas M. (2015). *Mastering Bitcoin*. First edition. Sebastopol CA: O'Reilly. 272 S. ISBN: 978-1-4493-7404-4.

Antonopoulos, Andreas M. und Gavin A. Wood (4. Dez. 2018). *Mastering Ethereum: Building Smart Contracts and DApps*. 1. Aufl. Farnham: O'Reilly UK Ltd. 384 S. ISBN: 978-1-4919-7194-9.

Aries Protocol Test Suite (19. Feb. 2021). Hyperledger. URL: https://github.com/hyperledger/aries-protocol-test-suite (besucht am 25. 01. 2022).

Aristy, George (21. Juli 2020). *Aries RFC 0510: Presentation-Exchange Attachment Format for Requesting and Presenting Proofs*. Hyperledger. URL: https://github.com/hyperledger/aries-rfcs/blob/663259fc56c21f454196e301f47264a3b98097f8/features/0510-dif-pres-exch-attach/README.md (besucht am 28. 02. 2022).

Aublin, Pierre-Louis, Sonia Ben Mokhtar und Vivien Quema (Juli 2013). „RBFT: Redundant Byzantine Fault Tolerance". In: *2013 IEEE 33rd International Conference on Distributed Computing Systems*. 2013 IEEE 33rd International Conference on Distributed Computing Systems (ICDCS). Philadelphia, PA, USA: IEEE, S. 297–306. ISBN: 978-0-7695-5000-8. DOI: 10.1109/ICDCS.2013.53. URL: http://ieeexplore.ieee.org/document/6681599/ (besucht am 20. 01. 2022).

Benaloh, Josh und Michael de Mare (1994). „One-Way Accumulators: A Decentralized Alternative to Digital Signatures". In: *Advances in Cryptology — EUROCRYPT '93*. Hrsg. von Tor Helleseth. Bearb. von Gerhard Goos und Juris Hartmanis. Bd. 765. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 274–285. ISBN: 978-3-540-48285-7. DOI: 10.1007/3-540-48285-7_24. URL: http://link.springer.com/10.1007/3-540-48285-7_24 (besucht am 22. 01. 2022).

Berners-Lee, Tim, Roy T. Fielding und Larry M. Masinter (Jan. 2005). *Uniform Resource Identifier (URI): Generic Syntax*. Request for Comments RFC 3986. Internet Engineering Task Force. 61 S. DOI: 10.17487/RFC3986. URL: https://datatracker.ietf.org/doc/rfc3986 (besucht am 10. 02. 2022).

Braendgaard, Pelle und Joel Torstensson (3. Mai 2018). *EIP-1056: Ethereum Lightweight Identity*. Ethereum Improvement Proposals. URL: https://eips.ethereum.org/EIPS/eip-1056 (besucht am 15. 01. 2022).

Bray, Tim (2017). *The JavaScript Object Notation (JSON) Data Interchange Format*. URL: https://tools.ietf.org/html/rfc8259 (besucht am 17. 11. 2020).

Brown, Daniel R. L. (21. Mai 2009). *SEC 1: Elliptic Curve Cryptography*. URL: https://www.secg.org/sec1-v2.pdf.

Buchner, Daniel (23. Juli 2019). *Rhythm and Melody: How Hubs and Agents Rock Together*. Decentralized Identity Foundation. URL: https://medium.com/decentralized-identity/rhythm-and-melody-how-hubs-and-agents-rock-together-ac2dd6bf8cf4 (besucht am 09. 02. 2022).

Buchner, Daniel, Orie Steele und Tobias Looker (2021). *Well Known DID Configuration*. URL: https://identity.foundation/specs/did-configuration/ (besucht am 11. 10. 2021).

Buchner, Daniel, Orie Steele und Troy Ronda (2022). *Sidetree*. DIF Sidetree Protocol. URL: https://identity.foundation/sidetree/spec/ (besucht am 11. 10. 2021).

Buchner, Daniel, Brent Zundel und Martin Riedel (2. Nov. 2021). *DIF Presentation Exchange v1.0.0*. URL: https://identity.foundation/presentation-exchange/spec/v1.0.0/ (besucht am 01. 02. 2022).

Buchner, Daniel, Brent Zundel, Martin Riedel und Kim Hamilton Duffy (16. Dez. 2021). *DIF Presentation Exchange 2.0.0*. URL: https://identity.foundation/presentation-exchange/ (besucht am 11. 10. 2021).

Burlacu, Filip (3. Dez. 2019). *Aries RFC 0335: HTTP Over DIDComm*. Hyperledger. URL: https://github.com/hyperledger/aries-rfcs/blob/663259fc56c21f454196e301f47264a3b98097f8/features/0335-http-over-didcomm/README.md (besucht am 01. 03. 2022).

Buterin, Vitalik (5. Dez. 2020). *EIP-55: Mixed-case Checksum Address Encoding*. Ethereum Foundation. URL: https://github.com/ethereum/EIPs/blob/c48da00d07d12b7903097773bc8f14876eeca535/EIPS/eip-55.md (besucht am 15. 01. 2022).

Camenisch, Jan, Markulf Kohlweiss und Claudio Soriente (2009). „An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials“. In: *Public Key Cryptography – PKC 2009*. Hrsg. von Stanisław Jarecki und Gene Tsudik.

Bd. 5443. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 481–500. ISBN: 978-3-642-00468-1. DOI: 10.1007/978-3-642-00468-1_27. URL: http://link.springer.com/10.1007/978-3-642-00468-1_27 (besucht am 23. 01. 2022).

Çelik, Tantek (1. Sep. 2021). *[Wbs] Response to 'Call for Review: Decentralized Identifiers (DIDs) v1.0 Is a W3C Proposed Recommendation' from Tantek Çelik via WBS Mailer on 2021-09-01 (Public-New-Work@w3.Org from September 2021)*. E-mail. URL: https://lists.w3.org/Archives/Public/public-new-work/2021Sep/0000.html (besucht am 02. 03. 2022).

Credential Handler API Demo (8. Okt. 2019). Digital Bazaar, Inc. URL: https://github.com/digitalbazaar/credential-handler-demo/blob/master/README.md (besucht am 30. 01. 2022).

Curran, Stephen und John Jordan (6. Jan. 2021). *Aries RFC 0302: Aries Interop Profile*. GitHub. URL: https://github.com/hyperledger/aries-rfcs/tree/main/concepts/0302-aries-interop-profile (besucht am 05. 01. 2022).

Curren, Sam, Tobias Looker und Oliver Terbu (22. Nov. 2021). *DIDComm Messaging Specification*. URL: https://identity.foundation/didcomm-messaging/spec/ (besucht am 11. 10. 2021).

Davie, Matthew u. a. (4. Nov. 2019). *Aries RFC 0289: The Trust Over IP Stack*. GitHub. URL: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0289-toip-stack/README.md (besucht am 21. 09. 2021).

Deventer, Oskar u. a. (12. Okt. 2021). *Peer DID Method Specification*. URL: https://identity.foundation/peer-did-method-spec/ (besucht am 11. 10. 2021).

DIF - Interoperability (2021). URL: https://identity.foundation/interop/ (besucht am 23. 09. 2021).

DIF - Secure Data Storage Working Group (2021). URL: https://identity.foundation/working-groups/secure-data-storage.html (besucht am 29. 12. 2021).

Duden (2022). *Duden | Interoperabilität | Rechtschreibung, Bedeutung, Definition, Herkunft*. URL: https://www.duden.de/rechtschreibung/Interoperabilitaet (besucht am 07. 02. 2022).

Duffy, Kim Hamilton, Christopher Allen und Ryan Grant (6. März 2018). *BTCR DID Resolver Specification*. Web of Trust Info. URL: https://github.com/WebOfTrustInfo/rwot6-santabarbara/blob/238be6d91a8929696bba90cefa7af1a67a1a3bbd/topics-and-advance-readings/btcr_did_resolver.md (besucht am 05. 01. 2022).

Element (6. Apr. 2020). *Element DID Method Specification*. Decentralized Identity Foundation. URL: https://github.com/decentralized-identity/element/blob/master/docs/did-method-spec/spec.md (besucht am 11. 10. 2021).

Ethereum DID Registry (11. Jan. 2022). Veramo core development. URL: https://github.com/uport-project/ethr-did-registry (besucht am 15. 01. 2022).

ETHR-DID (10. Nov. 2021). *ETHR DID Method Specification*. Decentralized Identity Foundation. URL: https://github.com/decentralized-identity/ethr-did-resolver/blob/b37e060f0c89dcdd445e282c4ceccc0a527e5bee/doc/did-method-spec.md (besucht am 13. 01. 2022).

Handling Android App Links (2020). *Handling Android App Links. Android Developers*. URL: https://developer.android.com/training/app-links (besucht am 11. 11. 2020).

Hardman, Daniel (21. Nov. 2019). *Aries RFC 0005: DID Communication*. Hyperledger. URL: https://github.com/hyperledger/aries-rfcs/blob/b40a77b05e11b0dcb7c94f24da597f1388220139/concepts/0005-didcomm/README.md (besucht am 28. 12. 2021).

– (29. Sep. 2020). *HIPE 0011: Credential Revocation - Hyperledger Indy HIPE Documentation*. URL: https://hyperledger-indy.readthedocs.io/projects/hipe/en/latest/text/0011-cred-revocation/README.html (besucht am 21. 01. 2022).

– (15. Apr. 2021). *Aries RFC 0592: Indy Attachment Formats for Requesting and Presenting Credentials*. GitHub. URL: https://github.com/hyperledger/aries-rfcs/tree/26344513082af4d76c77b8b4f5064e72d0a83b58/features/0592-indy-attachments (besucht am 25. 02. 2022).

Heck, Rouven (Sep. 2020). *SSI Architecture Stack & Community Efforts*. Decentralized Identity Foundation. URL: https://github.com/decentralized-identity/decentralized-identity.github.io/blob/f94e99e7bacb7a583aca5c8b06909b7b839b95a3/assets/ssi-architectural-stack--and--community-efforts-overview.pdf (besucht am 11. 10. 2021).

Hodges, Jeff u. a. (8. Apr. 2021). *Web Authentication: An API for Accessing Public Key Credentials - Level 2*. URL: https://www.w3.org/TR/webauthn-2/ (besucht am 02. 02. 2022).

Hyperledger Indy (28. März 2019). *Hyperledger Indy 1.0 Documentation*. URL: https://indy.readthedocs.io/en/latest/ (besucht am 19. 01. 2022).

Hyperledger-Aries-RFCs (12. Jan. 2022). Hyperledger. URL: https://github.com/hyperledger/aries-rfcs (besucht am 15. 01. 2022).

Introducing the Trust Over IP Foundation (5. Mai 2020). URL: https://trustoverip.org/wp-content/uploads/sites/98/2020/05/toip_introduction_050520.pdf (besucht am 21. 09. 2021).

ION (6. Dez. 2021). *ION: The Identity Overlay Network (ION)*. Decentralized Identity Foundation. URL: https://github.com/decentralized-identity/ion (besucht am 11.10.2021).

Jones, Michael (Mai 2015a). *JSON Web Algorithms (JWA)*. Request for Comments RFC 7518. Internet Engineering Task Force. 69 S. DOI: 10.17487/RFC7518. URL: https://datatracker.ietf.org/doc/rfc7518 (besucht am 26.01.2022).

– (Mai 2015b). *JSON Web Key (JWK)*. Request for Comments RFC 7517. Internet Engineering Task Force. 40 S. DOI: 10.17487/RFC7517. URL: https://datatracker.ietf.org/doc/rfc7517 (besucht am 13.02.2022).

Jones, Michael, John Bradley und Nat Sakimura (Mai 2015a). *JSON Web Signature (JWS)*. Request for Comments RFC 7515. Internet Engineering Task Force. 59 S. DOI: 10.17487/RFC7515. URL: https://datatracker.ietf.org/doc/rfc7515 (besucht am 26.01.2022).

– (Mai 2015b). *JSON Web Token (JWT)*. Request for Comments RFC 7519. Internet Engineering Task Force. 30 S. DOI: 10.17487/RFC7519. URL: https://datatracker.ietf.org/doc/rfc7519 (besucht am 26.01.2022).

Jones, Michael und Joe Hildebrand (Mai 2015). *JSON Web Encryption (JWE)*. Request for Comments RFC 7516. Internet Engineering Task Force. 51 S. DOI: 10.17487/RFC7516. URL: https://datatracker.ietf.org/doc/rfc7516 (besucht am 26.01.2022).

Khateev, Nikita und Stephen Curran (15. Apr. 2021). *Aries RFC 0454: Present Proof Protocol 2.0*. Hyperledger. URL: https://github.com/hyperledger/aries-rfcs/blob/663259fc56c21f454196e301f47264a3b98097f8/features/0454-present-proof-v2/README.md (besucht am 28.02.2022).

Khateev, Nikita, Stephen Klump und Stephen Curran (15. Apr. 2021). *Aries RFC 0453: Issue Credential Protocol 2.0*. Hyperledger. URL: https://github.com/hyperledger/aries-rfcs/blob/663259fc56c21f454196e301f47264a3b98097f8/features/0453-issue-credential-v2/README.md (besucht am 28.02.2022).

Khovratovich, Dmitry und Jason Law (Dez. 2016). *Sovrin: Digital Identities in the Blockchain Era*. URL: https://sovrin.org/wp-content/uploads/AnonCred-RWC.pdf.

Khovratovich, Dmitry und Michael Lodder (9. Feb. 2018). *Anonymous Credentials with Type-3 Revocation*. URL: https://github.com/hyperledger/indy-crypto/blob/master/libindy-crypto/docs/AnonCred.pdf.

KID0001 (15. Apr. 2021). *KID0001 - Prefixes, Derivation and Derivation Reference Tables*. Decentralized Identity Foundation. URL: https://github.com/decentralized-

identity / keri / blob / ce7bcf3152f07c9df128f1934f28296794bcf297 / kids / kid0001 . md (besucht am 13. 01. 2022).

Lawson, B. und R. Sharp (2010). *Introducing HTML5*. Voices That Matter. Pearson Education. ISBN: 978-0-321-71796-2. URL: https://books.google.de/books?id=hfH5m3lse7IC.

Linked Data Cryptographic Suite Registry (29. Dez. 2020). Unter Mitarb. von Manu Sporny, Drummond Reed und Orie Steele. URL: https://w3c-ccg.github.io/ld-cryptosuite-registry/ (besucht am 23. 02. 2022).

Lodder, Michael und Daniel Hardman (20. Aug. 2021). *Sovrin DID Method Specification*. URL: https://sovrin-foundation.github.io/sovrin/spec/did-method-spec-template.html (besucht am 17. 01. 2022).

Longley, Dave und Manu Sporny (23. Juni 2021). *Credential Handler API 1.0*. URL: https://w3c-ccg.github.io/credential-handler-api/ (besucht am 11. 10. 2021).

Longley, Dave, Dmitri Zagidulin und Manu Sporny (8. Dez. 2021). *The Did:Key Method v0.7*. URL: https://w3c-ccg.github.io/did-method-key/ (besucht am 06. 01. 2022).

Looker, Tobias (14. Jan. 2020). *JSON Web Message*. Internet Draft draft-looker-jwm-01. Internet Engineering Task Force. 21 S. URL: https://datatracker.ietf.org/doc/draft-looker-jwm-01 (besucht am 26. 01. 2022).

Menzer, Christoph und Sarah Otto (16. Aug. 2021). *Inter-Wallet Credential Exchange*. URL: https://b2cm.github.io/iwce/ (besucht am 01. 02. 2022).

Multicodec (2. Jan. 2022). Multiformats. URL: https://github.com/multiformats/multicodec/blob/e9ecf587558964715054a0afcc01f7ace220952c/README.md (besucht am 06. 01. 2022).

Nottingham, Mark (Mai 2019). *Well-Known Uniform Resource Identifiers (URIs)*. Request for Comments RFC 8615. Internet Engineering Task Force. 12 S. DOI: 10.17487/RFC8615. URL: https://datatracker.ietf.org/doc/rfc8615 (besucht am 02. 02. 2022).

Plenum (27. Juli 2020). *Plenum Byzantine Fault Tolerant Protocol*. Hyperledger. URL: https://github.com/hyperledger/indy-plenum (besucht am 20. 01. 2022).

Preukschat, Alexander und Drummond Reed (2021). *Self-Sovereign Identity: Decentralized Digital Identity and Verifiable Credentials*. Shelter Island: Manning. 466 S. ISBN: 978-1-61729-659-8.

Reed, Drummond u. a. (29. März 2019). *DKMS (Decentralized Key Management System) Design and Architecture V4*. Hyperledger. URL: https://github.com/hyperledger/

indy-hipe/blob/49fcd78883d38babe9c95a4e1d150969797cffa2/design/dkms/dkms-v4.md (besucht am 11. 10. 2021).

Sabadello, Markus (1. Nov. 2017). *A Universal Resolver for Self-Sovereign Identifiers*. Decentralized Identity Foundation. URL: https://medium.com/decentralized-identity/a-universal-resolver-for-self-sovereign-identifiers-48e6b4a5cc3c (besucht am 02. 02. 2022).

Sabadello, Markus, Fabian Vogelsteller und Peter Kolarov (27. Feb. 2018). *Did:Erc725 Method*. Web of Trust Info. URL: https://github.com/WebOfTrustInfo/rwot6-santabarbara/blob/238be6d91a8929696bba90cefa7af1a67a1a3bbd/topics-and-advance-readings/DID-Method-erc725.md (besucht am 17. 01. 2022).

Sabadello, Markus und Dmitri Zagidulin (2. Feb. 2022). *Decentralized Identifier Resolution (DID Resolution) v0.2*. URL: https://w3c-ccg.github.io/did-resolution/ (besucht am 02. 02. 2022).

Schnelli, Jonas und Daniel Pape (14. Mai 2021). *BIP 0136: Bech32 Encoded Tx Position References*. BIP 0136 - Bitcoin Wiki. URL: https://en.bitcoin.it/wiki/BIP_0136 (besucht am 05. 01. 2022).

SEC 2 (27. Jan. 2010). *SEC 2: Recommended Elliptic Curve Domain Parameters*. Unter Mitarb. von Daniel R. L. Brown. URL: https://www.secg.org/sec2-v2.pdf.

Smith, Samuel (3. Juli 2019). *Key Event Receipt Infrastructure (KERI)*. URL: https://www.researchgate.net/publication/334248478_Key_Event_Receipt_Infrastructure_KERI.

Smith, Samuel, Charles Cunningham und Phil Feairheller (10. Nov. 2021). *The Did:Keri Method v0.1*. URL: https://identity.foundation/keri/did_methods/ (besucht am 11. 01. 2022).

Sporny, Manu (29. Dez. 2020). *Credential Status List 2017*. URL: https://w3c-ccg.github.io/vc-csl2017/ (besucht am 22. 02. 2022).

Sporny, Manu, Dave Longley und David Chadwick (9. Nov. 2021). *Verifiable Credentials Data Model v1.1*. URL: https://www.w3.org/TR/vc-data-model/ (besucht am 30. 12. 2021).

Sporny, Manu, Dave Longley, Gregg Kellogg u. a. (16. Juli 2020). *JSON-LD 1.1*. URL: https://www.w3.org/TR/json-ld11/ (besucht am 22. 02. 2022).

Sporny, Manu, Dave Longley, Markus Sabadello u. a. (27. Juni 2021). *Decentralized Identifiers (DIDs) v1.0*. URL: https://www.w3.org/TR/did-core/ (besucht am 08. 09. 2021).

Steele, Orie und Manu Sporny (24. Juni 2021). *DID Specification Registries*. URL: https://www.w3.org/TR/did-spec-registries/ (besucht am 08. 09. 2021).

Terbu, O. u. a. (28. Jan. 2022). *OpenID Connect for Verifiable Presentations*. URL: https://openid.net/specs/openid-connect-4-verifiable-presentations-1_0.html (besucht am 01. 02. 2022).

Terbu, Oliver (27. Jan. 2019). *The Self-sovereign Identity Stack*. Decentralized Identity Foundation. URL: https://medium.com/decentralized-identity/the-self-sovereign-identity-stack-8a2cc95f2d45 (besucht am 11. 10. 2021).

Terbu, Oliver u. a. (22. Juli 2021). *Self-Issued OpenID Connect Provider DID Profile v0.1 (DEPRECATED)*. URL: https://identity.foundation/did-siop/ (besucht am 01. 02. 2022).

Tobin, Andrew (Sep. 2018). *Sovrin: What Goes on the Ledger?* URL: https://sovrin.org/wp-content/uploads/2018/10/What-Goes-On-The-Ledger.pdf.

Universal Resolver (26. Jan. 2022). Decentralized Identity Foundation. URL: https://github.com/decentralized-identity/universal-resolver (besucht am 02. 02. 2022).

Veer, Hans van der und Anthony Wiles (Apr. 2008). *Achieving Technical Interoperability - the ETSI Approach.* URL: https://www.etsi.org/images/files/ETSIWhitePapers/IOP%20whitepaper%20Edition%203%20final.pdf.

Verifiable Credentials Extension Registry (6. Dez. 2021). Unter Mitarb. von Manu Sporny und Daniel C. Burnett. URL: https://w3c-ccg.github.io/vc-extension-registry/ (besucht am 22. 02. 2022).

Verordnung (EU) Nr. 910/2014 (17. Sep. 2014). *Verordnung (EU) Nr. 910/2014 Des Europäischen Parlaments Und Des Rates Vom 23. Juli 2014 Über Elektronische Identifizierung Und Vertrauensdienste Für Elektronische Transaktionen Im Binnenmarkt Und Zur Aufhebung Der Richtlinie 1999/93/EG*. URL: https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:02014R0910-20140917&from=EN.

Vogelsteller, Fabian und Tyler Yasaka (2. Okt. 2017). *EIP-725: General Key-Value Store and Execution Standard*. Ethereum Improvement Proposals. URL: https://eips.ethereum.org/EIPS/eip-725 (besucht am 15. 01. 2022).

W3C and FIDO Alliance Finalize Web Standard for Secure, Passwordless Logins (4. März 2019). URL: https://www.w3.org/2019/03/pressrelease-webauthn-rec.html (besucht am 03. 02. 2022).

West, Mike (17. Jan. 2019). *Credential Management Level 1*. URL: https://www.w3.org/TR/credential-management-1/ (besucht am 30. 01. 2022).

Yasuda, K. und M. Jones (28. Jan. 2022). *Self-Issued OpenID Provider V2*. URL: https://openid.net/specs/openid-connect-self-issued-v2-1_0.html (besucht am 01. 02. 2022).

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 3. März 2022