Investigation of the utilization of blockchain technology as an identity system

Master's thesis for the Master degree
Master of Science in Media Technology
at the Faculty of Information, Media and Electrical Engineering
at the Technische Hochschule Köln

submitted by:	Attila Aldemir
matriculation number:	xxx
first supervisor:	Prof. Dr.-Ing. Luigi Lo Iacono
second supervisor:	Prof. Dr. Klaus-Dieter Ruelberg

Cologne, 31 August 2020

Untersuchung zum Einsatz der Blockchain-Technologie als ID System

Masterarbeit zur Erlangung des Master-Grades
Master of Science im Studiengang Medientechnologie
an der Fakultät für Informations-, Medien- und Elektrotechnik
der Technischen Hochschule Köln

vorgelegt von:	Attila Aldemir
Matrikel-Nr.:	xxx
Erstgutachter:	Prof. Dr.-Ing. Luigi Lo Iacono
Zweitgutachter:	Prof. Dr. Klaus-Dieter Ruelberg

Köln, den 31. August 2020

Abstract

Title: Investigation of the utilization of blockchain technology as an identity system

Abstract: Since the advent of Bitcoin, blockchain technology has evolved and moved into new fields. This includes the field of identity management, which eventually led to the development of a new identity management model. This model is called self-sovereign identity and promises users full control over their digital identity. In the context of this thesis the self-sovereign identity model shall be analyzed in more detail and an integration of the Blobaa project into it shall be developed. Blobaa is an open source project and investigates authentication mechanisms based on blockchain infrastructures.

Keywords: Blockchain, Self-Sovereign Identity, Decentralized Identifiers

Kurzbeschreibung

Titel: Untersuchung zum Einsatz der Blockchain-Technologie als ID System

Zusammenfassung: Seit dem Aufkommen von Bitcoin hat sich die Blockchain-Technologie weiterentwickelt und ist in neue Felder vorgedrungen. Hierzu zählt auch das Feld des Identitätsmanagements, was schließlich zum Entwickeln eines neuen Identitätsmanagementmodells geführt hat. Dieses Modell nennt sich Self-Sovereign Identity und verspricht dem Benutzer volle Kontrolle über seine digitale Identität. Im Rahmen dieser Arbeit soll das Self-Sovereign Identity Modell näher beleuchte und auf Basis dessen eine Integration des Blobaa Projektes in Dieses entwickelt werden. Blobaa ist ein quelloffenes Projekt und erforscht Authentifizierungsmechanismen auf Basis von Blockchain-Infrastrukturen.

Stichwörter: Blockchain, Self-Sovereign Identity, Decentralized Identifiers

Contents

1 Introduction

With occurrence of the Bitcoin whitepaper [1] in 2009, a new technology has been introduced to solve the problem of agreeing on the state of a distributed database at a particular time in a decentralized way. This technology is called blockchain and Bitcoin is the first application using this technology to realize a peer to peer electronic cash system. It was the starting point for a new technology area called distributed ledger technology (DLT) [2]. Blockchain concepts and implementations have been developed either as general purpose utilities [3] [4] [5] or for specific fields like fintech [1] [6], energy [7], supply chain [8] and art [9]. Another promising field for the blockchain technology is digital identity management. Due to the fact that blockchains are distributed databases, they are capable of serving as public key stores in a public key infrastructure (PKI). This brings the benefit of relying on a decentralized, transparent and tamper proof infrastructure and is called a decentralized public key infrastructure (DPKI) [10]. Based on this approach, a new identity model called self-sovereign identity (SSI) has been proposed and is still in development. It was first described in the blog post *The Path to Self-Sovereign Identity* [11] by Christopher Allen. The idea is to give full control of an entity's digital identity to that entity. The identity holder can self-sovereignly create and control ones identity, without being forced to rely on intermediates or central authorities. This includes to decide which personal data is shared and used with whom [12].

One approach for **blo**ckchain **ba**sed **a**uthentication that utilizes the Ardor blockchain [4] as a DPKI is the Blobaa project [13]. In its current state it provides a protocol, called attestation protocol, that allows the creation of PKI-like systems in which chains of trust can be build and managed via blockchain addresses. After a trust chain is built, users are able to sign verifiable data that can later be verified in terms of integrity and authenticity. The motivation of this thesis is to expand the Blobaa project with a new protocol, so that it is capable of acting in the SSI ecosystem. To do so, a new DID method is specified and evaluated. A DID method is a specification for managing decentralized identifiers (DIDs) on top of a specific DPKI.

The thesis is structured in the following way. Section two gives an introduction into the blockchain technology in general and describes the Ardor blockchain architecture in more detail. Section three discusses digital identity and provides an overview of existing identity models including the new SSI approach. Section four provides a detailed illustration of SSI, its current state and the underlying Trust over IP stack. Section five discusses the new *bba* DID method and its implementations. Section six presents a summary including considerations and an outlook.

2 Blockchain

In its core, a blockchain is a distributed database, called a ledger, based on peer to peer architecture. Each peer holds its own copy of the database and is called a node. What makes a blockchain special is the mechanism of agreeing on the state of the ledger and of tracking this agreement in form of a verifiable history. In its purest form this is done in a decentralized way where no central entities are involved. Being independent to central entities and tracking states in a verifiable history to prevent so called double spending attacks, an attack where an attacker manipulates the ledger history to benefit from a state change [1], makes it suitable for representing and transfering values over an insecure network like the internet.

2.1 Blockchain Components

Even though Bitcoin is only the first blockchain implementation and there are many different implementations existing build for different use cases with different characteristics like Ethereum, as a blockchain for smart contracts [3], Ardor with its multi-chain architecture [4] or Sovrin [14], entirely build for SSI, a blockchain can be described as a combination of the following seven components.

1. **Asymmetric cryptography**, used to bind transactions to an entity.
2. **Cryptographic hash function**, used to create digital fingerprints and as a random source in consensus algorithms.
3. **Peer to peer protocol**, used to exchange data between nodes in a decentralized way.
4. **Transaction**, used to change the state of the ledger.
5. **Block**, used to create a history of transactions.
6. **Incentive structure**, used to incentivize entities to include transactions into blocks
7. **Consensus mechanism**, used to create a consent history.

Asymmetric Cryptography Asymmetric Cryptography, also known as public-key cryptography, is a technology that uses a pair of related keys for encryption and signatures. Each key pair consists of a public and a private key. The public key should be publicly available and functions as an identifier of a key pair. The private key must be kept secret and is used to decrypt and/or sign messages. If for example Alice wants to send the message m encrypted to Bob, she would use Bobs public key $pubkey_b$ to create the encrypted message c_{mb} (also called cipher text) [15].

$$c_{mb} = enc(pubkey_b, m) \tag{1}$$

Only Bob would then be able to reverse the encryption process to decrypt c_{mb} with his private key $prvkey_b$.

$$m = dec(privkey_b, c_{mb}) \tag{2}$$

If Alice wants to prove to Bob that the message m was created or at least reviewed by Alice, she would create a signature s_a with her private key $prvkey_a$.

$$s_{ma} = sign(prvkey_a, m) \tag{3}$$

Bob would then be able to verify the signature s_{ma} with the help of Alice‘s public key $pubkey_a$ and the message m.

$$true/false = verify(pubkey_a, m, s_{ma}) \tag{4}$$

In a blockchain, asymmetric cryptography is used to bind transactions and blocks to entities (the private key holders). Every transaction and block contains a signature created by the issuing entity to cryptographically bind this exact transaction or block to that entity. If a transaction / block is part of the blockchain, it is cryptographically verifiable and undeniable that a specific transaction / block has been created by an entity that controlled the corresponding key pair at the time of creation. Public keys are also used to derive so called blockchain addresses to identify a key pair. This is done by either using the public key as it is or performing a cryptographic transformation (e.g. in form of a hash function) [16].

Cryptographic Hash Function A cryptographic hash function (hereafter, simply hash function) maps bit strings of arbitrary finite length to strings of fixed length in an irreversible way [15]. An input message m of arbitrary length is mapped to an output hash h with a finite, hash function dependent length. For example the widely-used SHA-256 hash function [17] maps the input message m to a 256 bits long hash h_{m256}.

$$h_{m256} = hash_{256}(m) \tag{5}$$

A hash function has four mayor characteristics. It is cheap to calculate h_m from m but very costly (practically impossible) to calculate m from h_m, which is why it is called a one way function. It is deterministic in mapping an input to an output which means that a specific hash function, given the same input m, always produces the exact same hash h_m. It is nondeterministic in mapping different inputs to different outputs which means that no correlation exists between inputs to the corresponding outputs. Every input produces an entirely unpredictable (but always the same) output even if there is just one bit difference in two inputs. A hash function is also collision resistant. Even though the existence of collision is guaranteed since a infinite number of inputs is mapped to a finit number of outputs, the output range must be wide enough to practically prevent collisions [15].

Hash functions in blockchains are used for two tasks. The first task is to create digital fingerprints of transactions and blocks so that they can be referenced in later transactions

/ blocks. The second task is located within the consensus mechanism. Due to the unpredictable output of a hash function, a hash function is used as a random generator to create random bits based on verifiable inputs.

Peer to Peer Protocol Since every node has its own copy of the ledger, there needs to be a protocol to connect nodes to form a network of peers. This is the task of the peer to peer protocol. It establishes connections to other nodes and synchronizes the ledger on the data level so that the current ledger state is present on every node.

Transaction A ledger's state is always changed with the help of a transaction. A transaction embeds the state-changing use case specific data with additional verification information like a signature of the submitting entity and a timestamp. The shape of the use case specific data as payloads are defined by the specific blockchain type. As an example, Bitcoin as an electronic cash system includes hashes of existing unspend transactions (a transaction that is unreferenced) as references to create a verifiable chain of transaction. By parsing the transaction chain from the newest to the oldest transaction, every node is able to verify and calculate the ledger state (which public key has how many bitcoins assigned) at any point in time [1]. Ethereum-based transactions on the other hand can carry payloads that contain machine-readable turing complete instructions for the Ethereum Virtual Machine (EVM) [18]. Each node runs its own EVM and executes the instructions inside a transaction. Since the transactions are ordered chronologically and shared across all nodes, each node computes the same state. Ardor, as a third example, embeds its child chains into this payload and secures the child chains with the Ardor parent chain [19].

Block A block is a bundle of transactions with additional meta data as shown in Figure 1. It includes the transactions `tx` as its payload along with meta data for indexing and

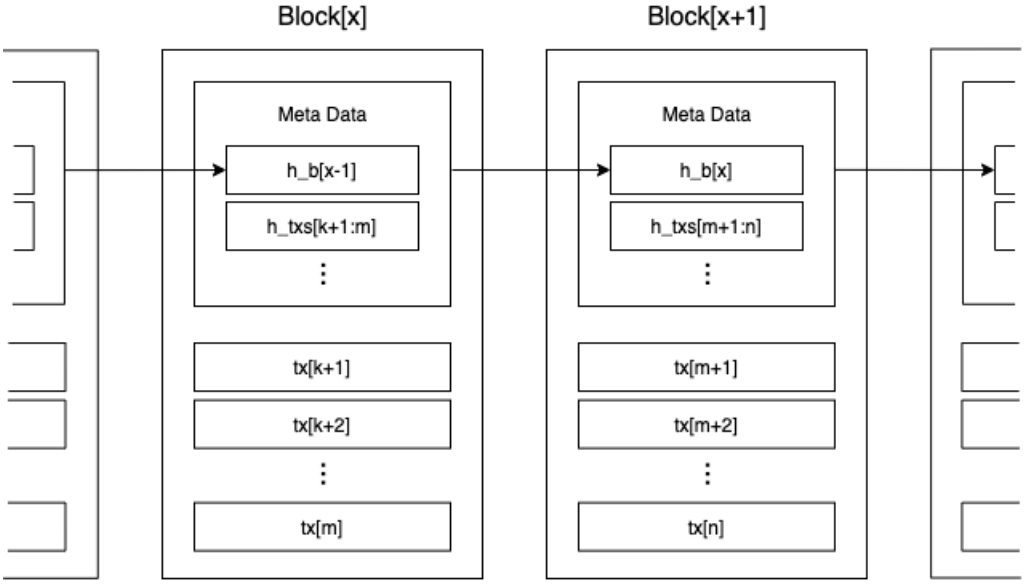

Figure 1: Chain of blocks

verification. A block also includes the hash of the previous block `h_b`, a hash that represents all combined block transactions `h_txs` (in most cases in form of a merkle root [1]) and other

meta data like a timestamp, a block index called block height, that increases with every block, and consensus mechanism related information. Adding the block fingerprint in form of a block hash into the next block creates a chain of blocks in which every previous block is cryptographically linked to the current one. This is where the name blockchain originates from. Every node verifies every block it receives and rejects invalid blocks so that only cryptographically verified blocks are included into the chain of blocks.

Incentive Structure A transaction is only accepted as part of the ledger if it is embedded into a block. Since validating unconfirmed transactions (transactions that are not included in a block) and bundling them into blocks require work, there has to be an incentive structure in place that encourages entities to perform these tasks. There are many incentive structures invented such as Bitcoin's deflational coin distribution where each miner (an entity that participates in a proof of work block generation race) is allowed to create a defined number of Bitcoins for each created accepted block (which is why it is called mining). The number of Bitcoins halves every 210000 blocks (roughly four years) until about 21 million Bitcoins are mined [20]. In Ethereum, a miner is rewarded two freshly-generated Ether (Ethereum's native currency) without a deflation mechanism. It is therefore inflational since no maximum amount of Ether is defined. Ardor does not reward a forger (an entity that participates in a proof of stake block generation race) with freshly-generated coins. In Ardor, the whole coin supply of roughly 1 billion Ardor (Ardor's native currency) is pre-forged and included into the genesis block (the very first block) [21]. It uses a fee mechanism where a transaction issuer is requested to add a fee in its transaction to create an incentive for the block generator to include that transaction into a block. The fee is then accounted to the block creator. A fee mechanism is also implemented in Bitcoin and Ethereum. Fees, aside from directly incentivizing a block creator to include a transaction into a block, also act as spam protection since adding transactions to the ledger involves monetary costs.

Consensus Mechanism Since there is an incentive to bundle transactions into blocks and there is no central authority to determine which entity is eligible to create the next block, a consensus mechanism with a defined set of rules that every node agrees on has to be in place. This is necessary to authorize entities to create the next valid block and therefore to collect the incentives. The two most popular consensus mechanisms are proof of work (PoW) and proof of stake (PoS). They differ in the way they authorize an entity to add the next block to the chain of blocks. Even though variations of each consensus mechanism exists, the core idea stays the same.

In a PoW-based blockchain, each miner tries to calculate a number that is below a certain threshold. This threshold is called the difficulty target and is dynamically adapted to react to environmental changes and to guarantee an average block time. A block time is the time between two consecutive blocks and can vary since the mining process (the process to generate new valid blocks) includes randomness. In Bitcoin, the average block time is set to ten minutes so that a new block is expected to occur every ten minutes on average. Since the number to be found also represents the fingerprint of the generated block (the block

hash), a hash function is required to create that number. Because there is no correlation between hash functions inputs and outputs, the only chance to create a hash lower than the difficulty target is to combine the block related data with a random number and to use this combination as the hash function input. If the resulting hash is greater than the threshold, one has to start over again and trying to find a hash below the threshold with a different random number (also called a nonce). This process is called mining. If a miner eventually finds a nonce that leads to a number below the difficulty target, the miner propagates its block to the network and starts mining the next block. If a miner receives valid blocks that are newer (in terms of block height) than the block one is currently mining, the miner stops the mining process and starts mining the next block based on the newer blocks. Since there is no other way than frequently trying random numbers to eventually generate a valid block, a valid block proofs that the miner performed the mining process and therefore put work into it in form of hashing. This is why this mechanism is called proof of work. The probability of mining the next block increases with the work (measured in hashes per second) a miner is able to perform [16]. Variations of the PoW mechanism are mainly located in the hash algorithm used to find the block hash. Bitcoin uses the SHA-256 hash function [1], whereas Ethereum uses a specialized hash function called Ethash [22].

In a PoS based blockchain, the probability of generation the next block is based on a forger's stake instead of work capacity. A stake is the amount of currency a forger is willing to "risk" to create a new valid block. Instead of requiring work, a PoS mechanism calculates the next block generator based on that stake. The calculation varies for each PoS variation [23] [24] [25] [26] but always considers the stake a forger has. If a forger acts honestly, one is rewarded by the incentive structure. In some variations, cheating means loosing some portion of ones stake. As an example PoS implementation, the PoS mechanism used by Ardor uses a threshold value, called target value T, that is calculated for each forger individually using the following equation [26].

$$T = T_b * S * B_e \tag{6}$$

The target value is based on the forging stake B_e (B_e stands for effective balance) a forger invests, a base target value T_b which is used to adjust the block time to realise an average block time of 60 seconds in the main network and ten seconds in the test network, and the time in seconds since the last block has been forged. This threshold changes individually after each second for each forger, since the effective balance (the stake) differs for each block generator. To be selected as the new block generator, ones hit target must be below the target value. A forger with a higher stake is more likely to be chosen as the next block generator since ones target value increases faster compared to a forger with a smaller stake. The hit target is calculated based on meta data of the previous block and a hash function similar to the PoW mechanism explained above. The difference is that a forger only creates one fix hit target and therefore performs the hash function only once. Instead of trying to find a value that is below a fixed threshold, one creates one random value (random in the sense of unpredictable outcome based on defined rules) and waits until ones moving target

value is greater than that value.

2.2 Blockchain Types

Based on the components explained above, blockchains with different participation types have evolved. A blockchain can be categorized based on two different types, access type and permission type [27]. The access type indicates the accessability of a blockchain. A blockchain can be publicly accessible so that no restrictions are in place that hinders an entity to run an own node to participate in the network. A private blockchain on the other hand is not publicly available and restricts an entity by means of artificial restrictions to access that blockchain. This type is used in consortia- or industry-wide blockchains. Beside the access type, a blockchain can also be categorized based on the permission type. It can be permissionless, which means that there is no restriction in participating in the block creation process, or permissioned, which means that there are requirements to be met to participate in the process. A commonly used consensus mechanism for a permissioned blockchain is called Proof of Authority (PoA) [28]. It restricts the creation of blocks to a fixed set of nodes.

2.3 Blockchain Characteristics

In principle, a blockchain has the four key characteristics listed below [29]. These characteristics vary with the blockchain type and implementation. A public permissionless blockchain for example can be considered more decentralized than a private permissioned one and a private blockchain can be considered to increase privacy compared to a publicly available blockchain. Arguments exists that state that a properly implemented PoS consensus can be more decentralized than a PoW mechanism [29].

1. **Decentralized**. Since a blockchain network is not controlled by a central entity and consensus over the ledger state is reached in a decentralized fashion, there is no single point of failure. This makes it very costly for an attacker to manipulate the ledger or to successfully perform a denial of service attack.

2. **Transparency**. Every node in a blockchain network has access to every transaction.

3. **Immutable**. Once included into a blockchain, manipulating data is very costly and can be considered unpractical with sufficient decentralization.

4. **Secure and Privat**. Since every transaction is signed by an entity's key pair, the origin of issued data can be verified and validated. An entity is only identifiable by its public key(s) and is therefore pseudonym.

2.4 Ardor Blockchain

Ardor is a proof of stake based blockchain and a redesign of the Nxt blockchain [26]. It is developed and maintained by the Jelurida Swiss SA [30] and introduces a novel approach for securing dedicated, use case specific blockchains (called child chains) via the Ardor blockchain (the parent chain). The parent chain has very limited functionality in terms of

transaction types, since its main focus is to secure the child chains. In Ardor, multiple predefined transaction types exist, each used to implement a specific feature. In contrast to the Ethereum blockchain, an Ardor transaction cannot carry machine readable instructions. Child chains on the other hand can implement each feature the Ardor platform provides. Figure 2 illustrates Ardors multi-chain architecture. Child chain blocks are embedded

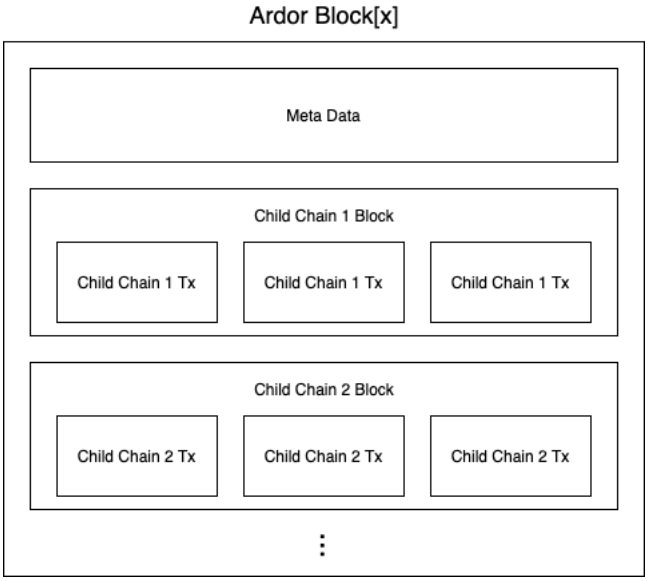

Figure 2: Ardor muli-chain architecture

into parent chain blocks in form of parent chain transactions. Each child chain block itself carries child chain transactions of various types. It is like a hierarchy of blockchains in which one blockchain (child chain) is embedded into transactions of another blockchain (parent chain). This architecture requires a new entity role, called bundler. Since each child chain provides its own currency and transaction fees are paid in chain native currencies, a bundler has the task to bundle child chain transactions into child chain blocks (similar to a forger bundling parent chain transactions into a parent chain block). As a reward, a bundler collects the child chain transaction fees, but must pay the parent chain transaction fee (the fee for the child chain block transaction). This creates a market for child chain to parent chain conversion. Since every participant is able to act as a bundler, this architecture also allows entities to sponsor child chain transaction fees for others [4] [21].

Another characteristic of Ardor is the representation of a blockchain address. Public keys are converted into human-readable address strings using the Reed-Solomon encoding [31]. Reed-Solomon is an error correction algorithm that allows up to four typing errors to be detected or two typing errors to be corrected [32]. Since the Reed-Solomon address space is smaller than the public key space from which it is derived, a transaction signed by the public key of the new address must be issued to bind that address to the public key. An example Reed-Solomon address is `ARDOR-YTHA-PLBJ-7TYT-CC6FX`. In Ardor, Reed-Solomon addresses are also called accounts.

2.5 Considerations

Despite its advantages, being decentralized and immutable has also some drawbacks. Since there has to be a consensus mechanism in place to agree on the ledger state and to create a verifiable, decentralized mechanism to update the state, a write operation to a blockchain is relatively slow compared to an ordinary data base. In addition to that, an entity needs to incentivize a block generator to include its transaction to the ledger in form of fees. A blockchain network also consumes more bandwidth than a distributed database due to its underlying peer to peer protocol.

3 Identity Management Models

An identity management model is an architecture to secure and manage digital identities [33]. It contains identity management systems which are typically used to manage two aspects of digital identity: authentication and attributes [34].

Authentication management Authentication management is the process of identifying an entity in the form of a digital identifier (e.g. an e-mail address). In general, this includes some form of proof (such as a password) to ensure that the entity presenting the identifier is indeed the entity the identifier represents [34].

Attribute management Attribute management, on the other hand, refers to the management of attributes of an entity such as address, birthday or credit card number [34]. The combination of identifier associated attributes creates an increasingly accurate image of a digital identity. According to the Bundesdruckerei, a digital identity refers to all procedures in which a person, object or process uses certain attributes to authenticate itself online [35]. Since an entity is capable of controlling multiple digital identities with different attributes, a digital identity is always a subset of an entity's *real world* identity.

Traditional identity management architectures can be categorized into three major models based on the identity provider (IDP) controlling party: centralized, federated and user-centric [33] [36] [37] [38] [34]. An IDP is the entity in an identity management system that holds and provides user information. It is the entity a service provider trusts when it comes to user authentication and authorization.

3.1 Centralized Identity Model

In the centralized identity model, each service provider (SP) controls its own IDP and self-manages its user's digital identities for its services (S). The most popular variant of this model is the isolated identity model [37]. In this model a service provider provides only one service, as shown in Figure 3. To authenticate, a user first creates an account

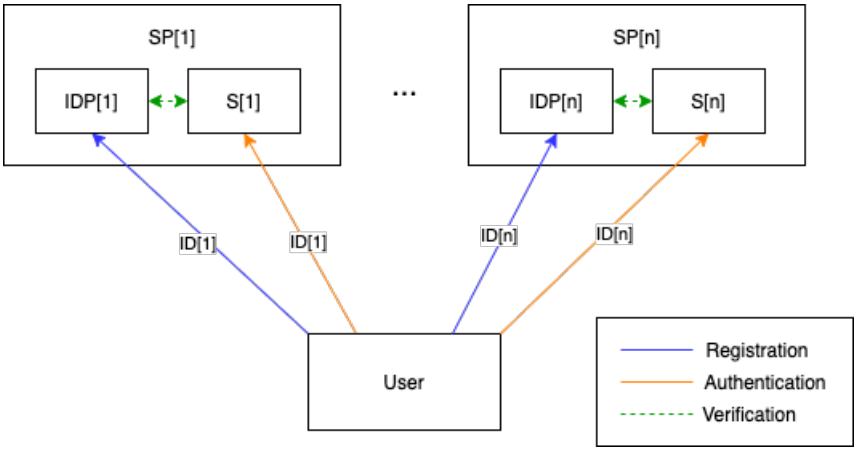

Figure 3: Isolated identity management

with the IDP of the SP that offers the service the user wants to use. If the user then

wants to use this service, one authenticates oneself with the previously created account. A common way to prove that one is the owner of an account is to provide a previously created shared secret (e.g. a password). A service then verifies the authenticity of an account by communicating with the IDP of its SP.

Although this model is probably the easiest to implement, it burdens the user the task of password management. This leads very often to simple passwords or reuse of the same password for different services [34], which reduces usability and security [37]. To lighten the burden of managing many different identities, some service providers that offer multiple services provide single sign-on mechanisms (SSO), as shown in Figure 4. A user can register

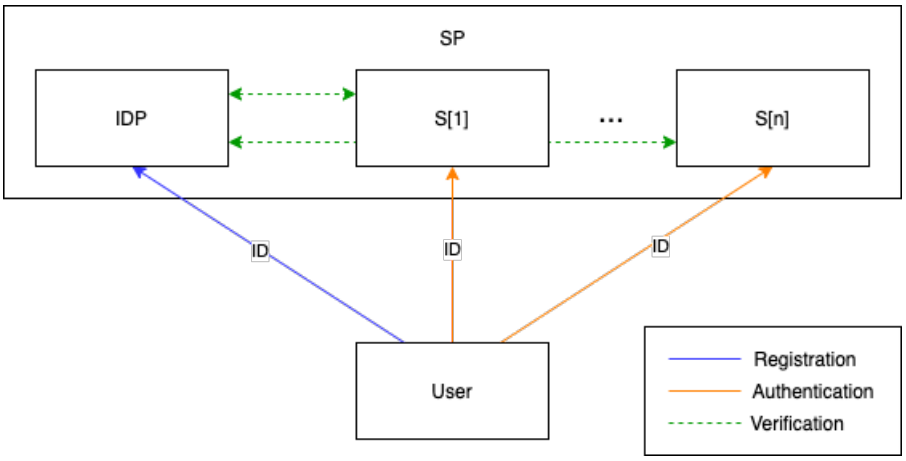

Figure 4: Centralized identity management

an identity with the IDP of a SP and use this identity to authenticate to multiple services provided by that SP. An example is Microsoft's Microsoft Account [39]. After registration, a user will be able to access several services such as Skype [40], OneDrive [41] or Outlook [42]. Another very similar example is a Google Account [43], which can be used to access services like Gmail [44], Google Docs [45], or Google Play [46]. This eliminates the need to manage multiple identities for one service provider.

The federated identity model goes one step further.

3.2 Federated Identity Model

The federated identity model allows a user to use one identity across multiple service providers. A user registers an account with the IDP of one service provider within a federation and can then use that identity to authenticate to all services within this federation, as shown in Figure 5. With this model, it is possible to implement SSO across multiple SPs and further reduce the number of accounts a user has to manage. Examples of this identity management model are logins via Facebook, LinkedIn or Google in the consumer sector or the eIDAS regulation [47] as a government backed association.

Three protocols have been developed to standardize the authentication workflow between service providers [12]. The first and oldest protocol is the Security Assertion Markup Language (SAML) [48]. However, due to its complexity it is not very popular [34]. Another approach and widely used for API authorization is OAuth [49]. The third protocol, based

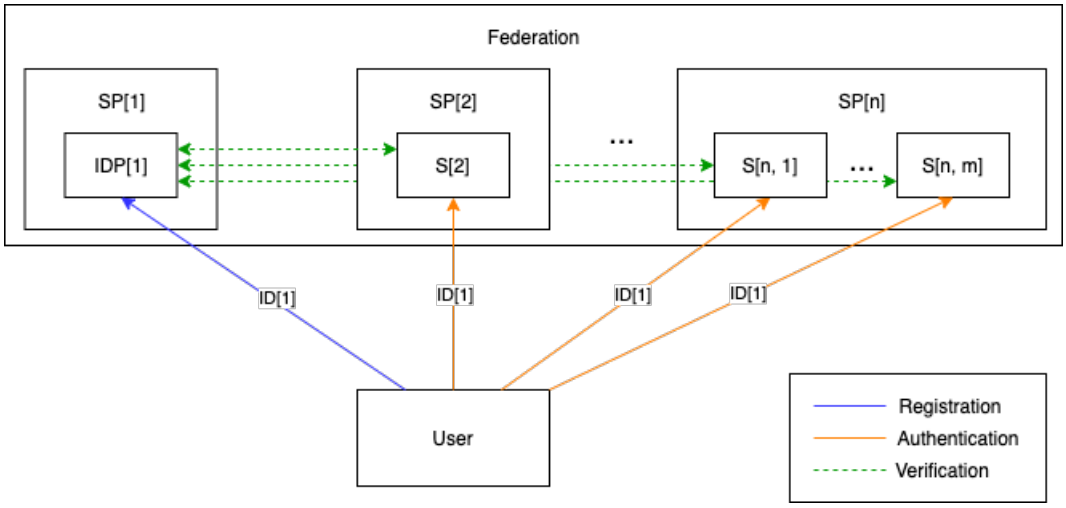

Figure 5: Federated identity management

on OAuth to facilitate user authentication, is OpenID Connect [50].

Although this model makes it even easier for a user to manage credentials, it does have some drawbacks. A user must entirely trust the IDP since it stands between the user and the service to which he wants to authenticate. This leads to the ability of monitoring a user's login activities across multiple websites and the ability of impersonating the user. Another problem is that large IDPs represent some of the biggest honeypots for cybercrime ever created [12]. If a hacker has succeeded in collecting user credentials, one is able to impersonate users not only in one, but in many services.

The user centric identity model tries to solve the IDP trust problem.

3.3 User-Centric Identity Model

The idea of a user centric identity model is to put the user into the center of one's identity managent. The user controls which IDP one wants to use or even provides an own IDP as shown in Figure 6. This means that a service must trust an IDP even if it is not in

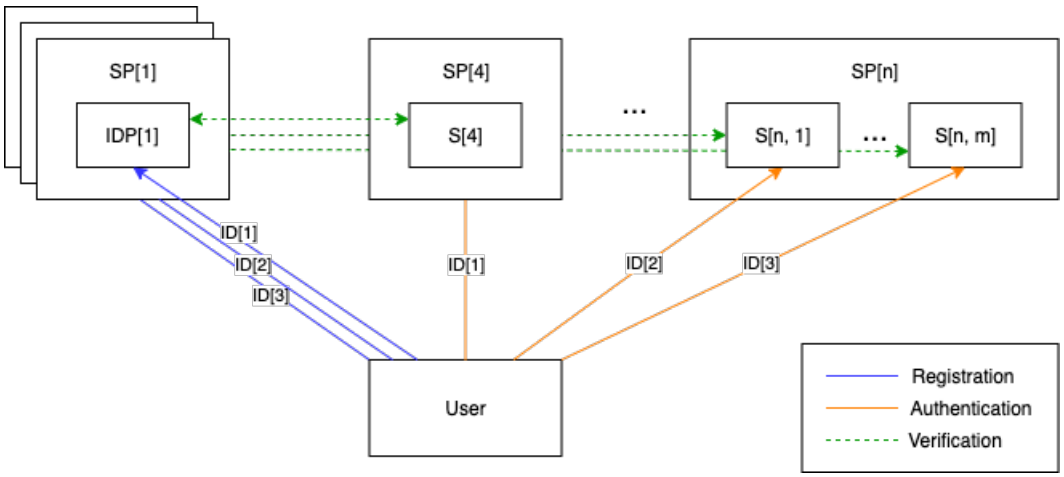

Figure 6: User-Centric identity management

the same federation [51]. An advantage of this model is the possibility to provide self-

controlled IDPs where user credentials are in control of the user. For example, a user could run a self-hosted OpenID Connect provider [52] and use that IDP to authenticate to services. And indeed, a prototype has been implemented that runs an IDP-based on OpenID Connect in a mobile phone [34]. Even though this approach solves the problem of trusting a foreign IDP (from a user's perspective), it takes some technical know-how to run ones own IDP. It can be assumed that the casual internet user is more likely to use an OpenID Connect implementation from a public website as a login for another [11].

Based on blockchain, a new identity model appeared: the SSI model [12].

3.4 Self-Sovereign Identity Model

The key idea of the SSI model is to move IDPs to blockchains, so that a user is able to self-sovereignly create globally unique identifiers that are controlled by that entity without needing to rely on centralized third parties, as shown in Figure 7 [12]. Another architectural

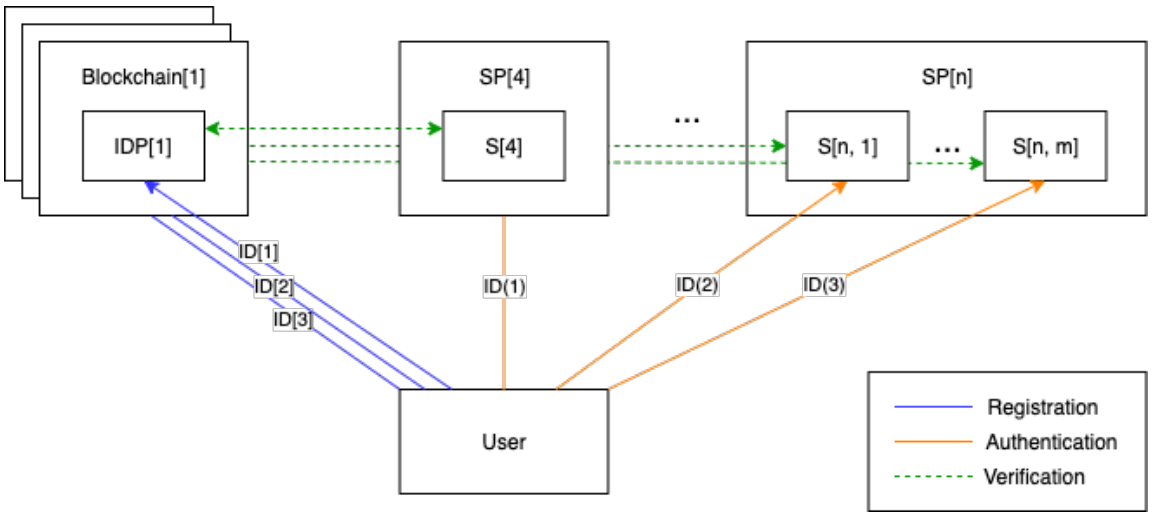

Figure 7: Self-Sovereign identity management

difference is the separation of attribute and identifier management. IDPs in this model only manage identifiers. Attributes are self-managed by the user and can be stored at any place a user wants. If a service needs user attributes for authentication, the user self-sovereignly forwards the requested attributes to that service. Since attributes are linked to identifiers, a service is able to verify received attributes and link them to the corresponding identifier.

The next section discusses the SSI model in more detail.

4 Self-Sovereign Identity

As mentioned in the previous section, the SSI model is a new emerging approach to establish and verify digital identity without having to rely on a central trusted third party. SSI is still in its infancy and in the state of being defined and specified. Although not finalized, it has already gotten traction from different initiatives. Microsoft, as one example, implemented a first prototype to integrate the SSI standard into its identity management system [53]. The global legal identifier foundation (GLEIF), a global non-profit foundation for trusted legal entity information [54], created a SSI based proof of concept (PoC) for verified data exchange in financial and commercial transactions [55]. A project named lissi (an acronym for let's initiate self-sovereign identity), an initiative to build an open identity ecosystem in Germany, released a first prototype recently [56]. The European Union (EU) started to define a European framework for self-sovereign identity, called ESSIF, to create a EU wide infrastructure for governmental SSI solutions [57]. It also released a paper on how to legally bind SSI identifier with its regulation for electronic identification, authentication and trust services (eIDAS) [58]. The EU is not the only governmental entity started to investigate on how to facilitated SSI. In Europe, multiple states started to investigate SSI on a nation-wide base too, like Finland [59], Spain [60], Austria [61] and the Netherlands [62]. In North America, Canada tries to adopt the SSI model within its public sector [12]. Another field where the SSI model piloted is e-mobility. The Deutsche Telekom piloted the project Blockchainscooter which is the first use case for a decentralized, open mobility ecosystem based on SSI, called Xride [63]. SSI also got traction in the academic field. A paper, published in IEEE Access journal, presents the first formal and comprehensive step toward an academic investigation of SSI [64]. Another paper proposed an architecture to integrating SSI solutions into web applications to forster their adoption [65]. A third paper investigates the potential of SSI in the context of industrial internet of things (IoT) [66]. These are just three examples of scientific research in the SSI field [67] [68] [69]. An attempt to collect the existing specifications and definitions to create the first complete in depth description of the SSI model is currently made by Alex Preukschat and Drummond Reed in form of a live book [12]. A live book is a format in which a reader is able to read parts of a book while it is being written. The final release is being estimated early 2021. The version 3 is the guideline for this chapter.

The concept of SSI is that one simply exists and therefore has a digital identity. One self-sovereignly builds, manages and uses ones digital identity and is no longer bound to identity authorities [70]. This does not mean that no trusted third parties (TTPs) exist, but instead of relying on their infrastructure to provide user identities, they exist to provide trust in claims an identity holder makes about oneself. The infrastructure itself is provided by the SSI environment and relies on decentralized, permissionless networks that must not be under control of single entities. These networks together constitute a universal identity layer [71] and can be considered as a decentralized IDP that allows digital identities under full control of the associated subject [70].

In the following, a brief overview of the SSI building blocks is provided and a more detailed explanation can be found inside the description of the Trust over IP stack [72].

4.1 SSI Building Blocks

The seven building blocks of SSI are [12]:

1. **Blockchains**, used to serve as decentralized IDPs
2. **Decentralized identifiers**, used to identify entities.
3. **Verifiable credentials**, used to hold entity attributes.
4. **Issuers, holders, and verifiers**, presenting the roles of the "trust triangle" [72].
5. **Digital wallets**, used to store credentials.
6. **Digital agents**, used to act on behalf of an entity inside the SSI ecosystem.
7. **Governance frameworks**, used to set the rules for how SSI infrastructure will be used.

Blockchains are used to provide the infrastructure for SSI identifier, so called **decentralized identifier** (DID) [73]. In the SSI model, identifiers are treated in a special way and managed separately from other identity related attributes. A DID is based on asymmetric cryptography and can be used to digitally sign so called **verifiable credentials** (VC) [74]. A VC is used to bundle verifiable claims and attributes about an entity (the subject). With these three concepts, trust triangles between a VC **issuer**, a VC **holder** and a VC **verifier** can be build to port the way trust is distributed in the physical world to the digital world [72], as shown in Figure 8.

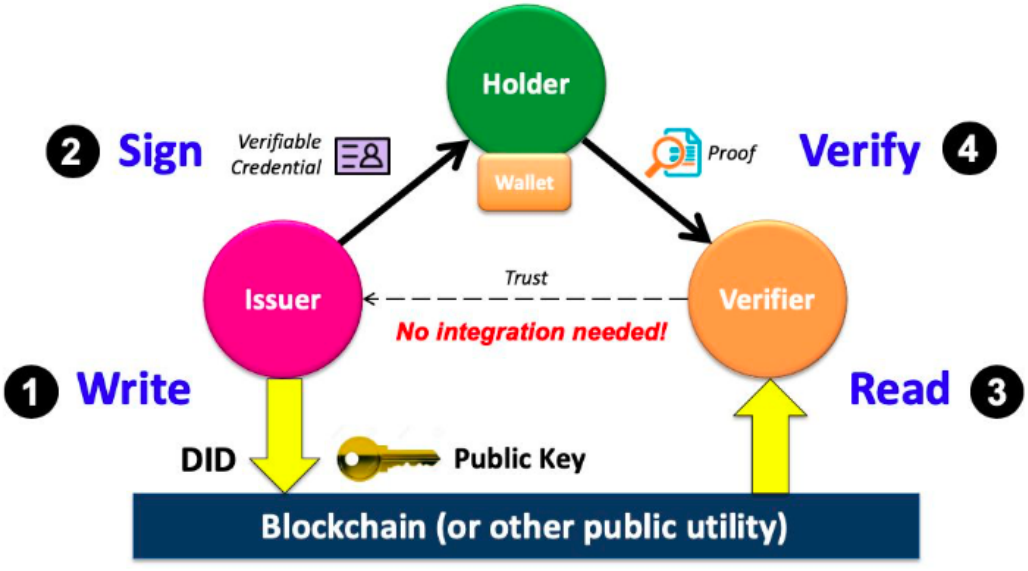

Figure 8: Verifiable credential trust triangle [72]

The workflow of the trust triangle can be best explained with an example. Let's say Alice, the VC holder (represented as the green circle in Figure 8), wants to buy a bottle of whisky for her father's birthday at a liqueur online shop, the VC verifier (the orange circle). In the physical world, Alice needs to prove her age by showing a trusted document with her

birth date as credential on it. The liqueur store employee would then first verify if her age is 18 or higher and then decide if he/she trusts the issuer of that document, an identity card for example. If he/she does, the employee sells the whisky to Alice and she is happy to have a present for her father. In the digital world with the use of VCs the workflow is nearly the same. The first and second step can be seen as a precondition in this scenario.

Step 1. DID Registration In the first step Alice and an ID agency (an entity authorized to create identity proofs) each register a DID on a SSI enabled blockchain including the public keys they are using for authentication.

Step 2. VC Issuing In the second step the ID agency, the VC issuer (pink circle), uses its private key to issue and sign a VC including Alice‘s birthday and DID and forwards it to Alice. Alice stores the VC in her own **digital wallet** and is now able to prove her age whenever she wants to. This is the key concept of the SSI model. Giving an individuum the opportunity to self-sovereignly control and share personal information without the compulsion of intermediates. A digital wallet is the digital equivalent of a physical wallet and is used to hold verifiable credentials on any computing device like smartphones, tablets and laptops [12]. It should be mentioned that an issuer doesn't actually have to issue the VC. The only thing that matters is that it is signed by the issuer to delegate its trust to the claim. It is also possible that the VC issuer and holder is the same entity. This is useful in scenarios where no TTP is needed.

Step 3. Issuer Credential Collection After choosing the whisky bottle at the online shop, Alice is requested to authenticate herself and prove her age within the checkout workflow. She authenticates herself with her DID and hands over a copy of her age proving VC via her **digital agent**. A digital agent is the software that lets an entity interact with digital wallets, obtain and present credentials, manage connections and exchange credentials with other agents. It is the assistant to interact within the SSI ecosystem [12]. After receiving the VC, the online shop gathers the public key of the ID agency‘s DID from the blockchain to prove the authenticity of the VC.

Step 4. VC Verification The last step is to actually verify the VC. To do so, the online shop first cryptographically verifies the authenticity of the VC by verifying its signature. If the signature is valid, the online shop can be sure that the VC is indeed created by the ID agency. The shop now has proven the VC and trusts the technical concepts of the SSI model. Since the VC also includes the verification signature created by the ID agency, no technical integration between VC issuer and VC verifier was needed to verify the VC. But one thing is missing which is not achievable via technical solutions and therefore called human trust [12]. The online shop needs to trust the ID agency and the processes and policies it has been used to verify Alice's age. It is the same problem as in the physical world. The liqueur shop employee needs to trust the creator of the identity card. He does that because he believes in the trustworthiness of the government and the process and policies involved to get that card. But how can this trust be transferred into the digital world so that the online shop is capable to automatically prove the trustworthiness of the

ID agency? The answer lays in the last SSI building block, the **governance framework**.

Governance frameworks, also known as trust frameworks, specify and standardize the policies and processes an issuer must follow to issue a specific VC. They also facilitate VCs issued by governance authorities to attest the trustworthiness of third parties operating under the policies and rules of that specific framework [12], as shown in Figure 9. These

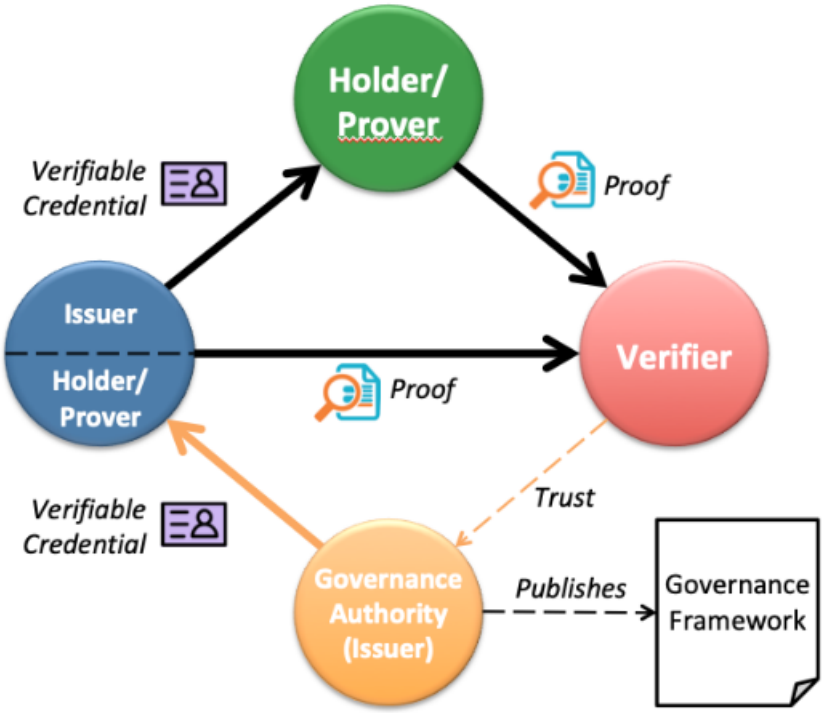

Figure 9: Trust triangle including governance framework [12]

VCs can be seen as digital representations of certificates and marks in the physical world and ensure that a VC issuer issues VCs according to specific rules and standards. A verifier in the end trusts the authorities of a governance framework that ensures that a specific VC has been issued according to well-defined and transparent policies and rules. In the liqueur shop example the governance authority could be a governmental institution that has defined the workflows for human age verification and attestation. It then audited and accredited the ID agency to perform legally binding age attestations by issuing a VC to the agency representing this decision. The governmental institution then functions as a an anchor of trust for the online shop. To verify the authenticity of Alice‘s age VC, the online shop first verifies the age VC and then the VC associated with the agency issued by governmental institution.

4.2 SSI Architecture

The architecture orchestrating the seven building blocks can be shown as a four layered dual stack illustrated in Figure 10. In this architecture, human trust is separated from technical trust which means that every of the for layers contain both governance and technical components to achieve user trust. Even though there is the need for governance components

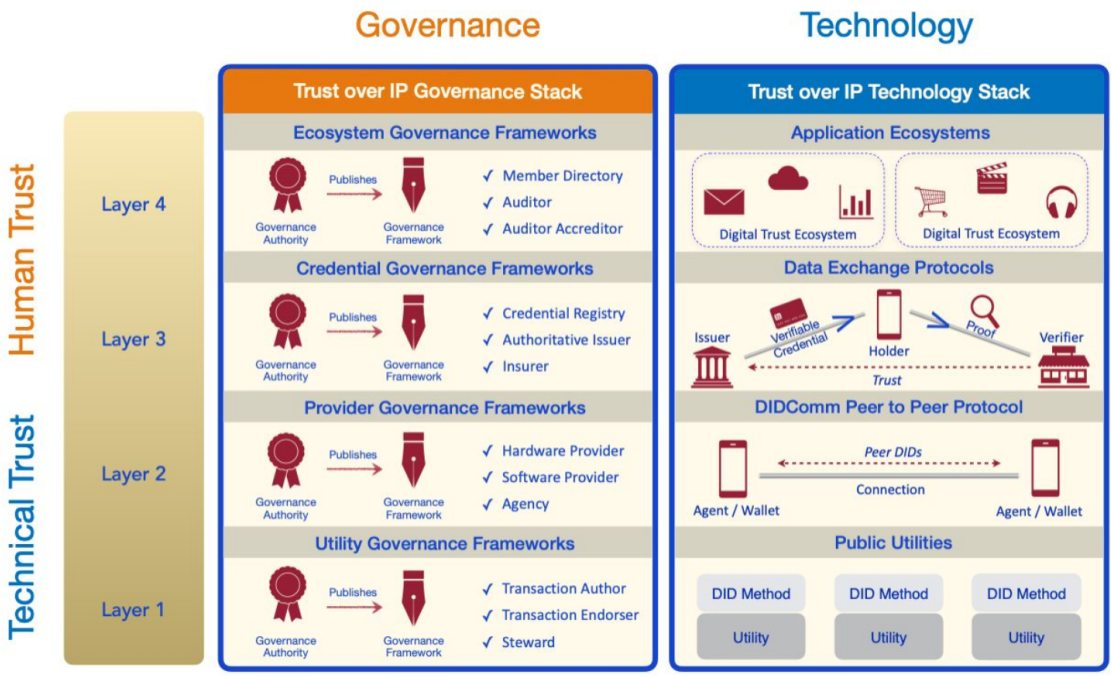

Figure 10: Trust over IP stack [72]

to provide trust in the SSI architecture in all four layers, the standards and policy models are still undefined. The recently created Trust over IP foundation [75] addresses this issue and started to define the models, templates, guidelines, and recommended best practices for the governance stack [72]. Since the governance framework is not in the scope of this thesis, the following parts of this chapter focus entirely on the technology stack.

The following sections discusses the details of each layer of the Trust over IP (ToIP) stack, building from the bottom up.

4.2.1 Technology Layer 1 - Public Utilities

The bottom of the technology stack is about identifiers and public keys. It serves as the root of trust in which DIDs are being registered and managed. This layer needs to guarantee that all stakeholders agree to the same truth about what an identifier references and how control of this identifier may be proved using cryptographic keys. It must also allow every party in the ecosystem to read and write data without reliance on or interference with central authorities. This is why blockchains with the properties of censorship resistance, immutability and tamper resistance serve so well as public utilities, also called verifiable data registries in the context of SSI [12] [73].

In the following, a closer description of DIDs is provided.

DID The specification [73], created by W3C and under active development at the time of writing, describes DIDs in the following way:

"Decentralized identifiers (DIDs) are a new type of identifier that enables verifiable, decentralized digital identity. A DID identifies any subject (e.g., a person, organization, thing, data model, abstract entity, etc.) that the controller of the DID decides that it identifies.

In contrast to typical, federated identifiers, DIDs have been designed so that they may be decoupled from centralized registries, identity providers, and certificate authorities. Specifically, while other parties might be used to help enable the discovery of information related to a DID, the design enables the controller of a DID to prove control over it without requiring permission from any other party. DIDs are URLs that associate a DID subject with a DID document allowing trustable interactions associated with that subject. Each DID document can express cryptographic material, verification methods, or service endpoints, which provide a set of mechanisms enabling a DID controller to prove control of the DID. Service endpoints enable trusted interactions associated with the DID subject. A DID document might contain semantics about the subject that it identifies. A DID document might contain the DID subject itself (e.g. a data model)."[73]

According to the specification, a DID is a unique decentralized identifier pointing to a specific DID document with information on how the DID controller can prove control over this DID. This is done by using asymmetric cryptography and therefore needs a PKI to publish and manage public keys.

DID Architecture In classic PKIs the problem usually exists that a relying party (RP) cannot easily say whether the controller of the public key is the one that the RP thinks it is. To solve this problem, TTPs issue certificates to trustfully bind an identifier to the controllers public key, a domain name to the public key of a web server for example. With this process, trust is delegated from the TTP to the controller and the RP relies on the trustworthiness of the TTP. This is even true in anarchical PKIs where trust is delegated throughout various trust paths. In both cases an RP needs to rely on TTPs to associate an identifier with a public key. A solution to circumvent the TTP is, instead of attaching a public to an identifier, to cryptographically create an identifier based on a public key. As an example, an identifier could be created by using the hash of a public key or issuing a blockchain transaction and using the transaction identifier. This makes the identifier self-certifying, since no TTP is needed to bind the public key to its identifier and the identifier to the controller. Every component is cryptographically bound and relies on the key material that only the controller controls. Even thought this eliminates human trust by means of relying on a TTP, it has one drawback. The identifier needs to change every time the public key rotates. A DID solves exactly this problem and provides a self-created, persistent, decentralized and verifiable identifier. Instead of creating a new identifier in the event of key rotation, a DID remains and only the DID document including the new public key is updated and signed by the previous key. This creates a resolvable chain of trust from the genisis to the current key as shown in Figure 11 and Figure 12. [12]

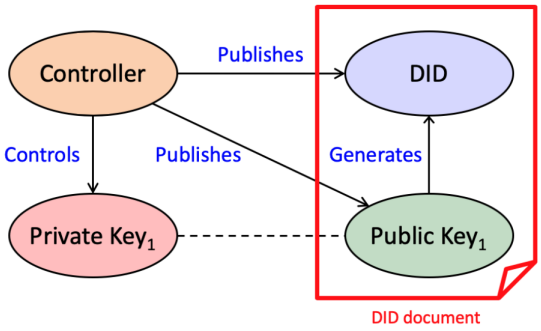

Figure 11: DID document [12]

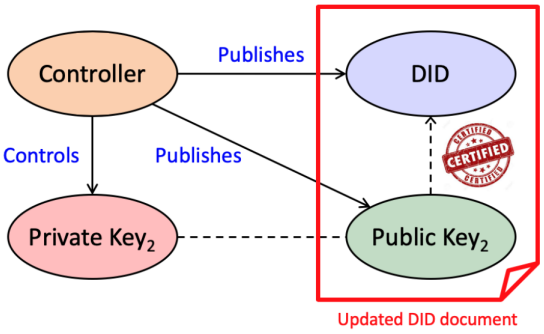

Figure 12: DID document update [12]

DID Layout A DID is a text string assembled by three parts [73]:

1. **URL scheme**. The scheme `"did"` identifies a DID as a decentralized identifier the same way the scheme *http* identifies a URL as a web URL.

2. **DID method name**. The DID method name identifies a specific DID method. A DID method is a definition of how a specific DID scheme must be implemented to work with a specific verifiable data registry. Being independent to a specific registry gains interoperability and independence in the way that a DID can be build on top of any verifiable data registry fulfilling the requirements of the DID methods specification.

3. **DID method specific identifier**. The method specific identifier is an identifier created by a specific DID method. It is the identifier derived from the genesis public key.

A DID can be expanded to a DID URL to identify a specific resources to be located. An example can be found in Listing 1.

```
1  /* DID in form of:
2     <prefix>:<method identifier>:<method specific identifier> */
3  did:example:123456789abcdefghi
4
5  /* DID URL including a DID query in form of:
6     <DID>?<parameter> */
7  did:example:123456789abcdefghi?version-time=2002-10-10T17:00:00Z
8
9  /* DID URL including a DID document fragment in form of:
10    <DID>#<fragment> */
11 did:example:123456789abcdefgh#vcs
```

Listing 1: Example DID

DID Documents Like web addresses in browsers, DIDs can be resolved to standardized data structures, the DID documents. Resolving DID documents is done by a so called DID resolver. Every DID has exactly one DID document associated. A DID document contains metadata about a DID subject. A DID subject is the entity that is identified by the DID and described by the DID document. In many cases the DID subject controls the

key material for the DID, which makes one also to the DID controller. Even though this is true for many cases, use cases exist where a DID controller differs from the DID subject. When a mother controls a DID which identifies her child, for example. The DID subject is the child but the DID controller (at least until the child comes of age) is the mother. This separation is another powerful feature of a DID. A DID document can theoretically contain any information about a DID subject like an email address or birth date. Even though this is possible, storing personal information in a publicly available DID document is problematic in terms of privacy. Because of that a DID document should only contain the minimum amount of machine-readable metadata to create a trustable interaction with the DID subject. To increase interoperability, the DID specification defines the following seven core properties [73].

1. **Context**. The context property (`@context`) is mandatory and adds semantic meaning to the properties inside a DID document. It contains links to specifications that contain machine readable information on how to interpret the represented data.

2. **Identifier**. The Identifier property (`id`) is also mandatory and identifies the DID subject. It contains the DID to which the DID document belongs.

3. **Verification Methods**. A verification method is a set of parameters that can be used together with a process or protocol to independently verify a proof [73]. It can stand alone in form of the `publicKey` property or/and be embedded in a verification relationship.

4. **Verification Relationships**. A verification relationship expresses the relationship between a DID subject and a verification method [73]. It expresses the context in which a method is used. A verification method can either be embedded in or referenced by a verification relationship. At the time of writing, five relationships are defined: `authentication`, used for authentication, `assertionMethod`, used to prove assertion statements about the DID subject, `keyAgreement`, used for key agreement protocols, `capabilityInvocation`, used to prove authorization of capabilities the DID subject is allowed to do, and `capabilityDelegation`, used to authorize capabilities to other DID subjects.

5. **Service Endpoints**. A service endpoint expresses a way for further communication with the DID subject and is represented in form of the `service` property.

6. **Created**. The `created` property indicated the creation date and time of a DID document.

7. **Updated**. The `updated` property indicates the last update of a DID document.

A typical DID document contains one or more public keys used to authenticate the DID subject and one or more services related to the DID subject. These information can then be used for concrete interactions via protocols supported by the services. An example DID document is shown in Listing 2 [12].

```
{
    "@context": "https://www.w3.org/ns/did/v1",
    "id": "did:example:123456789abcdefghi",
    "authentication": [{
        // used to authenticate as did:...fghi
        "id": "did:example:123456789abcdefghi#keys-1",
        "type": "Ed25519VerificationKey2018",
        "controller": "did:example:123456789abcdefghi",
        "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
    }],
    "service": [{
        // used to retrieve Verifiable Credentials associated with the DID
        "id":"did:example:123456789abcdefghi#vcs",
        "type": "VerifiableCredentialService",
        "serviceEndpoint": "https://example.com/vc/"
    }]
}
```

Listing 2: Example DID document [73]

A DID document uses the JSON-LD syntax to represent the DID subject related data. JSON-LD is an extension to the JSON syntax to add context to a data set [76]. It adds semantic to data sets so that an entity consuming a JSON-LD data set knows how to interpret these data. To do so, the `@context` property links to a specification that contains machine readable information on how to interpret the represented data. A DID document is not necessarily a static data set, it can be dynamically compiled by a DID resolver. It is more like a return value of a resolution request.

DID Resolution DID resolution, in contrast to DNS resolution for example, is not a specified protocol and DID method specific. Even proprietary verifiable data registries including proprietary DID resolver for proprietary DID methods are imaginable. However, such a system undermines the intention of a DID to be self controlled and TTP independent. It can be assumed that such systems would not be widely adopted. To gain interoperability and transparency, it is recommended to follow the DID method operation specification describes in the DID specification [73]. According to this specification, a DID method should support the four CRUD operations (create, read, update, deactivate) to manage a DID and be described in a publicly available document with a specified, detailed and complete enough description to independently implement a DID method using that description. As an addition, the description should contain sections describing security and privacy considerations. To gain an overview over all available DID methods following this specification, the World Wide Web Consortium (W3C) created an informal DID method registry with more than 50 DID methods currently listed [77]. One method listed in this registry has a unique position and is heavily used in layer 2, the Peer DID method.

Peer DID The Peer DID method [78] is a method to create blockchain-independent peer-to-peer DIDs (Peer DIDs) where DIDs are being created and exchanged directly between peers. It is a method to create pairwise and n-wise relationships between peers where

the number of parties is known, for example to create a relationship between two peers (pairwise). The verifiable data registry in this case is the digital wallet of each of the peers where each is the root of trust for the other. This has advantages in terms of scalability since there is no external party involved in the DID resolution. This also increases privacy because DID subject data like public keys or service endpoints are only shared between these peers.

4.2.2 Technology Layer 2 - Peer to Peer Protocol

Layer two is about establishing digital relationships between two or more parties in a secure and direct (peer-to-peer) way. This is the layer where digital wallets, holding DID subjects credentials, and digital agents, used to interact in the SSI ecosystem as a representative of a DID subject, life. They are used to establish trustworthy and secure relationships between peers. To do so, one protocol is in specification to act as the carrier of domain specific communication protocols.

DIDComm The DID communication protocol (DIDComm) utilities DIDs to create trustworthy, message-oriented, transport-agnostic, secure connections between agents. It was originally specified by the Hyperledger Aries project [79]. Since January 2020, development has continued under the Decentralised Identity Foundation [80] (DIF) to create a neutral and independent backbone for peer-to-peer connections [81]. The DIDComm protocol consists of two layers, as shown in Figure 13.

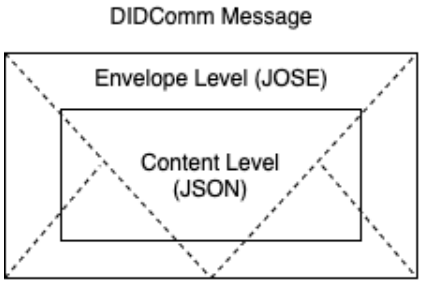

Figure 13: DIDComm Message Anatomy

Envelope Level The envelope level acts as a digital envelope for DIDComm messages analogous to envelops in a postal system for physical messages. It contains two main variations.

Encrypted Envelope

This format is used if a sender agent knows the DID of its receiver and wants to send a private message to it. The sender uses a public key presented in the receivers DID document to prepare and encrypt the message. It is like an opaque envelope in the physical world with the advantage that only the recipient is able to open it. Based on this envelope, three different sub-formats can be established.

1. **Anonymous Encrypted**. An anonymous encrypted envelope is the basic format explained above. It's like a physical postal message without any information about

the sender.

2. **Signed Encrypted**. A signed encrypted envelope, in contrast to an anonymous encrypted envelope, additionally contains an unencrypted signature from the sender, so that authentication can be performed by any party knowing the sender‘s DID. This is like a hand written signature on a physical envelope. It can be used for non-repudiation, where every sender-knowing party can authenticate that a specific message was created by the sender without being able to see the actual message.

3. **Authenticated Encrypted**. In an authenticated encrypted envelope the signature is also encrypted with the recipients public key so that only the recipient is able to encrypt the message and authenticate the origin. It is like a hand written signature on the inside of a physical envelope.

Signed Unencrypted Envelope

This format is used when the sender doesn't know the DID of a recipient. It is like a transparent physical envelope with the signature on it. It can be used to authenticate the message with its non-repudiable signature.

The envelope level utilizes the JSON Object Signing and Encryption (JOSE) suite [82] for encryption and signatures. JOSE provides a set of web technologies (JSON Web Signature (JWS), JSON Web Encryption (JWE), JSON Web Key (JWK), and JSON Web Algorithms (JWA)) to encrypt and/or sign content based on the JSON syntax.

Content Level The content level is about the actual message inside an envelope. It uses the JSON syntax and can contain any kind of information, the same way a letter can contain any information a sender wants to write on it. However, to gain interoperability and standardize message flows, three basic conventions are currently defined.

1. **Message Type**. The message type field is used to determine the context of a message and thus to process the content. It specifies the higher-level protocol the message is associated to. These protocols are designed for specific use cases to solve real world problems like issuing credentials [83], buying things or playing games [84].

2. **Message Id**. The message identifier is used to uniquely identify a message. It is a unique string (preferable a UUID [85] version 4) and can be used to globally identify a message by combining it with the senders DID.

3. **Decorators**. Decorates are optional chunks of data to convey meta data. They are useful for message tracing, message threading, timing measurements and other use case agnostic mechanisms.

An example DIDComm message content is shown in Listing 3.

```
{
    /* message type */
    "@type": "did:example:abcdefghi123456789;spec/tic-tac-toe/1.0/move",
    /* message id */
    "@id": "98fd8d72-80f6-4419-abc2-c65ea39d0f38",
    /* standardized thread decorator */
    "~thread": {
        "thid": "98fd8d72-80f6-4419-abc2-c65ea39d0f38",
    },
    /* use case specific protocol attribute */
    "me": "X",
    "moves": ["X:B2"]
}
```

Listing 3: Example DIDComm message content

The `@type` property represents the message type and contains a DID URL pointing to the type of the actual message (move). To do so, the URL also includes the use case specific protocol (tic-tac-toe) and the protocol version following the semantic versioning specification (1.0) [86]. To get the actual protocol specification, one needs to resolve the associated DID which then transforms to the concrete specification URL (e.g. https://example.com/spec/tic-tac-toe/1.0/move). Here, a DID is used to identify a digital object that can be referenced and is cryptographically verifiable. Even though using a DID as protocol specification reference is recommended, it is not required and it is also possible to use a concrete URL directly.

The `@id` property represents the unique message id.

The `~thread` property is a specific kind of decorator. It contains the message id of the first message in a message thread as reference. A message thread model is necessary in DIDComm, since a sender cannot determine the order and timing in which a response to a message arrives. One cannot even expect to get a response message via the same transport medium. A szenario is imaginable in which a sender sends a DIDComm message via HTTP and receives a response to that message via Bluetooth. To persist the communication state, this state is carried within the message in form of a thread decorator (a thread decorator defines multiple additional fields to persist the communication state [87]).

The attributes `me` and `moves` belong to the use case specific protocol. They contain the information used to solve the real world user problem of playing a round of Tic Tac Toe.

4.2.3 Technology Layer 3 - Data Exchange Protocol

Layer three is all about trusted data exchange. This is where the use case specific protocols carried by DIDComm life. Even though there are many use cases imaginable where data trust in the communication layer is sufficient, like playing a round of Tic Tac Toe with an anonymous opponent or anonymously logging into a website, many real world use cases require proofs of some kind of personal credentials like name, age and address or proof of rights and ownership. Proving credentials is where the trust triangle shown in Figure 8 with its issuer, holder and verifier roles comes into play. To standardize the way to express

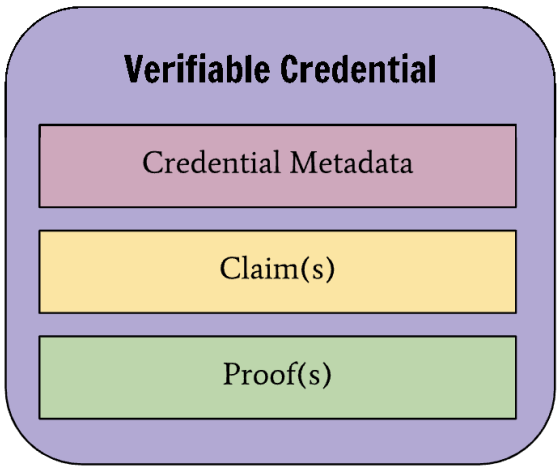

Figure 14: VC components [74]

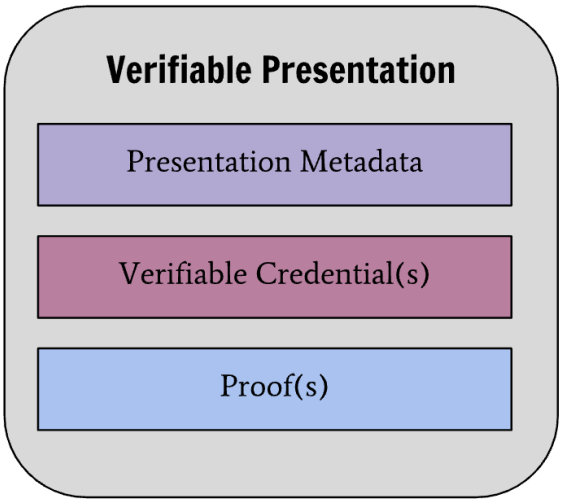

Figure 15: VP components [74]

credentials, a new data model, called verifiable credentials, has been specified by the W3C and is available in its first complete version [74].

Verifiable Credential VCs were created to provide a digital version of the credentials a person carries in a physical wallet. It allows one to prove ones identity "bottom-up" with a set of claims about an identifier rather than "top-down" by trying to define an identity and attach information to it. A VC gives a holder the ability to self-sovereignly share cryptographically provable information about a subject, the VC subject. In most cases, similar to the DID subject, a VC subject is the same entity as the VC holder. In some cases these roles are separated. A vaccination card for a cat for example could be represented as a VC in which the VC subject, the cat, is not the same entity as the cat owner, who manages the vaccination card VC for that cat, the VC holder [12].

It should be mentioned that sharing in the sense of transferring a VC from the holder to the verifier is not in the scope of a VC. This is a high level DIDComm protocol task and two solutions have been proposed by the Hyperledger Aries project to address this. These solutions define protocols for issuing credentials to a holder [83] and presenting credentials to a verifier [88] in a secure way.

The scope of the VC specification is to define a data model which is provable in terms of integrity and authenticity of VC subject related data within a VC. These data are called claims. The VC specification defines two data models to represent these claims, the verifiable credential and the verifiable presentation (VP) model. The VC model, as explained above and shown in Figure 14, holds one or more claims along with credential meta data and proofs for verification. The VP model, shown in Figure 15, is used to present VPs to a verifier. The advantage of presenting VCs in form of VPs is the ability to assemble one presentation based on multiple VCs. This is especially useful in scenarios where a VC holder holds multiple VCs issued by different VC issuers. With a VP, the VC holder is able to bundle a subset of ones personal data into a presentation. With this mechanism, multiple personas can be created that only reveal the minimum amount of data necessary for a given situation. Another important use case for VPs is to provide

information in a privacy focused way in form of zero-knowledge proofs. A zero-knowledge proof (ZKP) is a proof of some statement which reveals nothing other than the veracity of that statement [89]. With a ZKP based VP, a VC holder is able to minimize the disclosure of information even more. One can reveal only a subset of claims inside a VC in a verifiable way, called selective disclosure, or provide derived information based on a claim inside a VC. One could, for example, plausibly demonstrate that a VC subject is older than a specific age without revealing the birth date claim. Even though presenting a VC in form of a VP is beneficial in multiple ways, it is not a mandatory feature and a VC could also be presented directly [74].

Two syntax types are supported to describe a VC/VP, the JSON-LD syntax and JSON Web Tokens (JWT) [90]. These syntaxes are interchangeable and every VC/VP can be converted from one format into the other. In the following the JSON-LD syntax is being used to explain the fields of a VC and VP data model, since it is the recommended format. The most basic VC contains six pieces of information [74].

1. **Context**. The context property, similar to the type field of a DIDComm message and the context field in a DID document, is used to provide data inside a VC/VP semantic meaning. It contains at least one URL pointing to the (machine readable) description of how to process the data of a specific VC type. This makes VCs usable for many use cases, since everyone is able to create its domain specific VC. One only needs to point to the VC description inside a VC so that a verifier knows how to process the VC data. A basic context is provided by W3C and the URL pointing to it has to be included as the first item of the context array to indicate that the presented data object is a VC and supports at least the basic set of VC data fields.

2. **Type**. The type property contains a list of URLs indicating of which type a VC is. A verifier can parse the list and quickly determine if one is willing to accept and support a VC. Thanks to the JSON-LD context mechanism, the URL can be shortened and represented as a JSON-LD term, which makes it easier for developers to use. Like with the context array, the W3C defines a base VC type named "VerifiableCredential" that has to be included first.

3. **Issuer**. The issuer property holds the identifier of the VC issuer. Although not mandatory, in most cases this identifier includes a DID which can be resolved to a DID document to collect additional information to verify the data expressed in the VC.

4. **Issuance Date**. The issuance date indicates the date and time a VC becomes valid. It does not necessarily need to be the point in time a VC was signed by an VC issuer and can be set to a date and time in the future. The VC would then be invalid until this point in time is reached.

5. **Credential subject**. Despite the previous types all belonging to the credential meta data, the credential subject field contains the actual claims associated to a VC subject. It is a container for these claims which are specified by the domain specific context and type URLs.

6. **Proof**. the last required property is the proof property. It is, similar to the credential subject property, a container property and holds the signature created by the VC issuer along with additional information of how to prove that signature. Since no proof mechanism is defined here, a specification for proofs needs to be included by the context mechanism and provides the ability to integrate a wide range of cryptographic proof mechanisms.

An example VC with the minimum amount of properties is shown in Listing 4.

```
{
    /* meta data */
    "@context": [
        "https://www.w3.org/2018/credentials/v1",
        "https://www.w3.org/2018/credentials/examples/v1"
    ],
    "type": ["VerifiableCredential", "NameAndAddress"],
    "issuer": "did:example:123456789abcdefghi",
    "issuanceDate": "2020-07-06T19:73:24Z",
    /* claim */
    "credentialSubject": {
        "id": "did:example:abcdefghi123456789",
        "name": "Mr John Doe",
        "address": "10 Some Street, Anytown, ThisLocal, Country X"
    },
    /* proof */
    "proof": {
        "type": "RsaSignature2018",
        "created": "2018-06-18T21:19:10Z",
        "proofPurpose": "assertionMethod",
        "verificationMethod": "did:example:123456789abcdefghi#public-key-1"
    ,
        "jws": "eyJhbGciOiJQUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQi..."
    }
}
```

Listing 4: Example VC

Even though the data object in Listing 4 represents a valid VC, there are more properties defined in the basic credential definition to support a VC verifier in verifying a VC. The `id` property for example assigns a VC a unique identifier to identify and reference that exact VC, an `expirationDate` property defining the expiration date and time of a VC or the `credentialStatus` pointing to a VC status list.

A VP in its minimum form only requires four properties [74].

1. **Context**. The context property is again used to specify the context of VP. It has the exact same requirements as the context property of a VC

2. **Type**. The type property, similar to the type property of a VC, presents the type of a VP in form of a URL or JSON-LD term. The basic type is "VerifiablePresentation".

3. **Verifiable Credential**. The verifiable credential type holds the VCs to be presented. It holds the "payload" of a VP in the same way a VC holds its claims.

4. **Proof**. The last required property is again the proof property and functions in the same way as a VC proof. The differences here is the scope of proof. In contrast to a VC proof, a VP proof does not prove the authenticity of claim. It only proves that the VCs are assembled by a specific entity (which in most cases is the VC holder). In addition to that, a VP proof can be constructed in a privacy enhanced way by ZKPs.

An example VP containing the minimum amount of properties is illustrated in Listing 5

```
{
    /* meta data */
    "@context": [
        "https://www.w3.org/2018/credentials/v1",
    ],
    "type": ["VerifiablePresentation"],
    /* VCs */
    "verifiableCredential": [{...}],
    /* proof */
    "proof": {...}
}
```

Listing 5: Example VC

With the ability to create and manage digital credentials in a self-sovereign fassion, a user is now able to interact with applications in a self controlled way without needing to (technically) rely on TTPs.

4.2.4 Technology Layer 4 - Application Ecosystem

Applications using the ToIP stack to identify, authenticate and authorize users (where a user can be any kind of entity: human, legal, object or software) need common standards to interpret the presented VCs. This is true from a governance point of view to be able to make decisions about the trustworthiness of a VC, but also from a technical stand point. An application needs to know how to consume and issue VCs so that a user is able to use the same VC for different applications without friction. Applications processing the same types of VCs define an application ecosystem where data can be exchanged in a interoperable and frictionless way.

4.3 Summary

SSI defines a new identity management model to enable user autonomy in the way that users do not have to rely on a TTP to create, manage and use digital identity. It describes a model for a neutral infrastructure to self-sovereignly create and manage decentralized identifiers and verifiable credentials. DIDs are cryptographically provable and bound to a private key which only the user has control of. They are furthermore the basis to assemble digital identities "bottom up" by attaching personal information to it in form of VCs and are used to create direct secure communication channels between peers using a new type of communication protocol named DIDComm. Because DIDs are so essential and occur in each of the four layers of the Trust over IP stack defining the SSI architecture, they

are stored on publicly available utilities. This utilities are databases that act as so called decentralized public key infrastructures. Even though not mandatory, it is recommended to use blockchains as DPKIs to benefit from the neutral, (permissionless), highly available and tamper proof characteristics of these systems. VCs on the other hand are (in most cases) self stored data sets and issued by a trusted third party. The difference between a conventional IDP acting as TTP and a VC issuer lays in the way a verifier consumes the user identity related data. In the conventional model (centric, federated, user centric), the verifier reaches out to the IDP and therefore needs some kind of integration to establish a communication with it. The IDP on the other hand is able to collect meta data about a user identity and to deny access to that identity. Within the SSI model, the user self-sovereignly shares its identity data with the verifier and the verifier can decide if it accepts and trusts the data issued by the TTP. This has four advantages. First, there is no technical integration needed since no communication to the TTP occurs within a verification. Second, because there is no communication, the TTP cannot collect meta data of the user. Third, personal information are stored decentralized inside the wallets of each user so that no centralized honeypot of identity data exists. This makes it much harder for an attacker to collect a reasonable amount of identity data. Fourth, a user is able to granularly decide which information it wants to share with the verifier.

Even though SSI brings advantages to authentication, a second big field is authorization. Because VCs can contain any type of information (claims), they are also capable of carrying authorization claims in a secure and provable way. Alice, for example, could issue a claim to Bob which states that Bob is allowed to use her car from Monday to Wednesday. This VC could be created with two mobile apps (one for Alice and one for Bob) acting as their personal agents. The issued VC is then stored on Bob's mobile wallet. Alice's car agent, as an example of an object in SSI, could then prove this statement by providing a QR Code attached to the window. Whenever Bob wants to use Alice's car, he scans the code with his agent and Alice's car agent and Bob's mobile phone agent automatically start to communicate via Bluetooth and create a secure and persistent DIDComm connection. After the connection is established, Bob's agent presents the authorization VC (direct or in form of a VP) to the car agent which then proves the VC. To do so, it either has Alice‘s DID document stored locally in its memory or reaches out to the internet to resolve Alice‘s DID. If the VC is valid and it is a week day between Monday and Wednesday, the car opens the door and Bob is able to drive the car. In general, authorizing someone to act on behalf of another or to have the right to use or do something, opens many use cases to the SSI ecosystem. Since SSI is designed with machine-readability in mind and no IDP is technically needed for claim verification, another advantage of this model is workflow automation [12]. Use cases are imaginable where multiple claims issued by different issuer need to be verified to process a step in a business workflow. Think of the example above where Alice is a car rental company. In addition to the authorization claim, Bob is requested to present his driver license VC issued by the government. Alice's car agent can easily integrate this second step by supporting the standardized driver license VC and proving its validity by using the official driver license government DID. There is no need to get in contact with the government to support this feature. It only requires the driver

license VC specification and the official government DID.

But SSI also comes with drawbacks / risks in form of key management. Without proper mechanisms for key recovery and protection, a DID is lost without the chance of recovering in case of loosing the corresponding private key. Self-sovereignty is a two-edged sword. Nobody can impersonate you (except someone has access to your private key) or take your DID and therefore your identity away, but there is also no third party who is able to "recover your password". This is an analogy to the blockchain technology where you control your account with the generated private key, too. Progress has been made in this area to solve the "do not loose or compromise your key" problem like paper wallets [91], hardware wallets [92] or hierarchical deterministic (HD) wallets [93], which let you securely store your private key. This is another field where blockchain technologie has an impact on the SSI model. These wallets are different to the digital wallets used in the SSI model. They refer to the mechanism a key can be backed up. A paper wallet is a sheet of paper with key pair information on it. A hardware wallet stores private keys on a hardware device and an HD wallet is a software that creates private keys based on a master secrete.

The following section discusses a new DID method that utilizes the Ardor blockchain as a verifiable data registry.

5 BBA DID Method

In order to enable the Blobaa project to participate in the new SSI model described in the *Self-Sovereign Identity* section, a new DID method should be developed and evaluated. The method should be named *bba* as an abbreviation for **B**lo**b**a**a** and is discussed in this section.

First, the requirements that a DID method have to meet are discussed. This is the basis for the following *bba* DID method specification. Based on this specification, four implementations have been developed, which are presented in the *Implementations* section. In the last part of this section the *bba* DID method is evaluated.

5.1 DID Method Requirements

As mentioned in the *Self-Sovereign Identity* section, a DID specification conformal DID method must be described in a DID method specification with the following five requirements [73].

1. The specification must be specific, detailed, and complete enough for independent implementation.
2. The specification must contain a method-specific method scheme.
3. The specification must contain the method-specific CRUD operations.
4. The specification must contain a security considerations section.
5. The specification must contain a privacy considerations section.

5.1.1 Method Scheme

A DID method specification must define exactly one method-specific DID scheme with a unique method name among all known method names. It must be based on the generic DID scheme as represented in Listing 6 and further restrict the method-specific-id [73].

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

Listing 6: Generic DID method scheme [73]

A DID scheme is defined in the Augmented Backus-Naur Form (ABNF) [94] syntax. The generic DID scheme shown in Listing 6 can be translated as follows. The `did` string consists of four concatenated components. The first component is a fixed string set to *did:*. The second component is the `method-name` variable. It is itself defined as a concatenation of a minimum of one and up to unlimited `method-char` variables. The `method-char` variable is then defined as a character representing ether a digit (a number in the range of [0-9]) or a character encoded as ASCII [95] with the hexadecimal range of [61-7A] ([a-z]). The next component is again a fixed string consisting of the *:* character. It functions as a delimiter to

separate did string elements. The last component contains the method specific information and is called `method-specific-id`. It consists of at least one `idchar` and can itself contain did string delimiters to separate did method specific string elements. An `idchar` variable can consist of an alphabetic character [a-zA-Z], a digit, or one of the characters ., -, _ .

Even though not required, it is recommended that the method name is less than five characters in size [73]. A DID method specification must describe how to generate the `method-specific-id` component without the use of centralized registry. It is recommended that the method-specific-id is globally unique. If this is not feasible, it must be unique in combination with the other did string components [73].

5.1.2 Method Operations

A DID method specification must specify the CRUD (create, update, read, deactivate) operations for a specific DID method. It must also describe the authority of a party to carry out these operations [73].

Create It must be described how an entity creates a DID and its associated DID document on the verifiable data registry, including all cryptographic operations necessary to establish proof of control [73].

Read It must be described how an entity uses a DID to request a DID document from the verifiable data registry, including how the client can verify the authenticity of the response [73].

Update It must be described how an entity can update a DID, including all cryptographic operations necessary to establish proof of control. An update is any change that affects the DID [73].

Deactivate It must be described how an entity can deactivate a DID on the verifiable data registry, including all cryptographic operations necessary to establish proof of deactivation [73].

5.1.3 Security Considerations

The specification must consider all requirements mentioned in section 5 of the *Guidelines for Writing RFC Text on Security Considerations* [96] for the DID operations. At least the following attacks must be considered: eavesdropping, replay, message insertion, deletion, modification and man-in-the-middle, as well as potential denial of service attacks. It must further indicate which data are protected and what the protections are. In addition to that, it must discuss how endpoint authentication is achieved and by which policy mechanism DIDs are proven to be uniquely assigned [73].

5.1.4 Privacy Considerations

The specification must consider any subsection of section 5 of the *Privacy Considerations for Internet Protocols* [97] description that is applicable for the specified DID method.

The subsections to consider are: surveillance, stored data compromise, unsolicited traffic, misattribution, correlation, identification, secondary use, disclosure, exclusion [73].

In the following, the *bba* DID method is constructed based on the presented requirements.

5.2 DID Method Specification

The *bba* DID method aims to enable the Ardor blockchain to act as a verifiable data registry within the SSI ecosystem. It runs on the independent Ignis [98] child chain and utilizes Ardor's Account Properties feature [99] to manage DIDs and corresponding DID controllers. The Account Properties feature provides the possibility to tag an account with a small amount of data (160 characters). A DID controller is always represented in form of an Ardor account and is by default separated from the public keys (if present) embedded in a DID document. Think of a master key controlling the DID operations create, update and deactivate. A DID controller always corresponds to exactly one Ardor account, whereas one Ardor account can control multiple DIDs.

DID and DID document handling is decoupled so that multiple DID document storages can be defined and integrated to store DID document templates (DID documents without a DID reference). In the version presented here, the *bba* DID method defines only one storage type (Ardor Cloud Storage).

In the following, *bba* DID method compliant account properties are called DID attestations. An account property is *bba* DID method compliant if it aligns to the data model described in the *DID Attestation Data Fields* section and is self-set. A self-set account property is a property in which sender and receiver accounts are identical.

5.2.1 DID Attestation Data Fields

Since the *bba* method utilizes the Account Properties feature, most of its data fields are embedded in the *value* key/value pair of a property. In Ardor, an account property is represented as a JSON object with at least a *property* and a *value* key/value pair. An example attestation is shown in Figure 16.

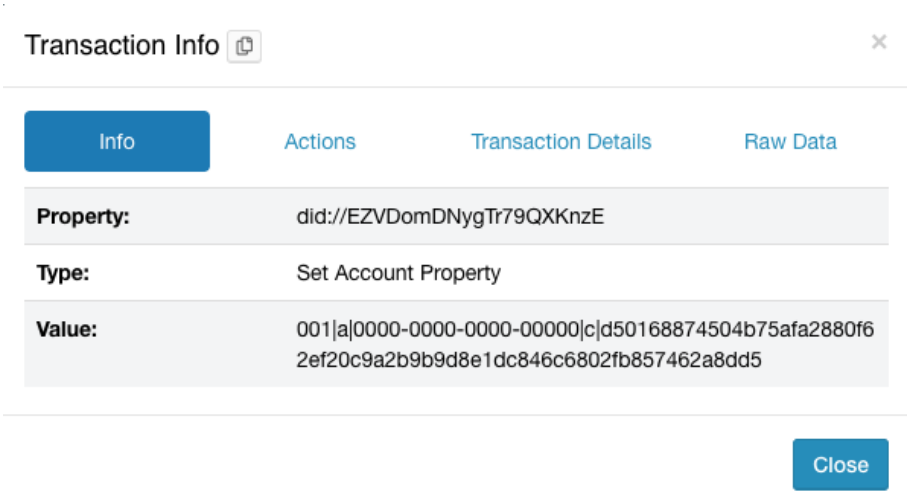

Figure 16: Example DID attestation

DID Id The only data field not embedded in the *value* key/value pair and used to determine a specific DID is the DID Id. It represents the value of the *property* key/value pair and is used to provide the ability to control multiple DIDs with one account. To indicate that a property is a DID attestation, the property value starts with the prefix `did://`, followed by a unique, case-sensitive, 20-character long alphanumeric Id. For example: `did://EZVDomDNygTr79QXKnzE`.

Data Field Concatenation All other data fields are embedded in the 160 characters long *value* key/value pair. They consist of one or more characters concatenated into a defined order and separated by a pipe delimiter [|], as illustrated in Figure 17.

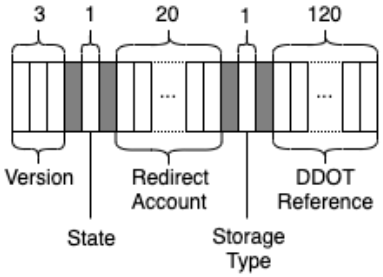

Figure 17: Character arrangement of DID attestation data fields

Version The version field points to the *bba* method version. It is a three-digit number starting with 001 and needs to be incremented whenever a protocol update occurs. The current version number is 001.

State The state data field indicates the current state of an attestation. There are three state types defined. An **active** state indicates that a DID is active and can be resolved. An **inactive** state states that a DID has been active in the past and is now inactive. It can no longer be resolved and cannot be reactivated. A **deprecated** state states that the DID controller is deprecated and the DID is now controlled by another controller. The new DID controller account is referenced in the redirect account data field. Table 1 shows the state characters used to represent the attestation states:

State	State Character	Brief
active	a	DID is active and resolvable
inactive	i	DID is inactive and cannot be resolved and reactivated
deprecated	d	DID controller is deprecated and a new controller is referenced via the redirect account field

Table 1: State type character overview

Redirect Account The redirect account data field is only used in case of a deprecated state. It refers to the controller account that has taken control of the DID. To save character space, this data field does not contain the complete Reed-Solomon account representation. Only the significant 20 characters without the prefix `ARDOR-` are present. To use the redirecting account, one needs to reconstruct the Reed Solomon representation later on. If the data field is not used, it has the dummy value `0000-0000-0000-0000000`.

Storage Type The storage type character indicates the storage mechanism used to store and retrieve a DID document template (DDOT). It must specify a mechanism to cryptographically prove that a DDOT has not been tampered with between storage and retrieval, define methods for storing and retrieving a DDOT, and define a reference that fits in the DDOT reference field.

Ardor Cloud Storage The Ardor Cloud Storage mechanism uses Ardor‘s Data Cloud feature [100] to store a stringified DDOT JSON in plain text within a transaction. Ardor's Data Cloud feature lets a user store user-defined data in a child chain. The transaction full hash value is then used as the DDOT reference and can later be used to retrieve the data cloud transaction and thus the DDOT object. Table 2 shows the storage type characters that are used to represent the storage types:

Storage Type	**Storage Type Character**	**Brief**
Ardor Cloud Storage	c	DDOT is stored in Ardors Data Cloud

Table 2: Storage types overview

DID Document Template Reference The DDOT reference field contains the DDOT reference used to retrieve the DID-related DDOT. It binds the DDOT cryptographically to the DID. It can consist of any character string not longer than 120 characters. The interpretation of these characters is defined by the storage mechanism.

Misc It should be mentioned that not all 160 characters are being used. Only 149 characters are occupied: 3 (version) + 1 (state) + 20 (redirect account) + 1 (storage type) + 120 (payload) + 4 (delimiter). The missing 11 characters are reserved for future method extensions.

5.2.2 Method-specific DID syntax

Listing 7 shows the ABNF scheme of the bba DID method:

```
1 bba-did        = "did:bba:" bba-identifier
2 bba-identifier = [ ardor-network ":" ] ardor-tx-hash
3 ardor-network  = "m" / "t"
4 ardor-tx-hash  = 64HEXDIG
```

Listing 7: ABNF scheme of bba DID method

bba DIDs can refer either to Ardor's test network or main network. If no network is specified, the main network is assumed. A *bba* DID contains the full hash of the first attestation transaction as a starting point and identifier. Due to the transaction hash as a unique identifier, it can be assumed that *bba* DIDs are globally unique. Listing 8 shows example *bba* DID strings.

```
/* mainnet */
did:bba:47ef0798566073ea302b8178943aaa83f227614d6f36a4d2bcd92993bbed6044
// or
did:bba:m:47ef0798566073ea302b8178943aaa83f227614d6f36a4d2bcd92993bbed6044

/* testnet */
did:bba:t:45e6df15dc0a7d91dcccd24fda3b52c3983a214fb0eed0938321c11ec99403cf
```

Listing 8: BBA DID examples

5.2.3 CRUD Operations

The *bba* method defines five operations to create, read, update (DDOT and DID controller) and deactivate a DID.

Create To create a DID, a DID document template is required. It contains the information of the DID document associated to the DID.

DID Document Template The difference between a resolved DID document and the DID document template stored with a storage mechanism is the associated DID. The DDOT does not contain any DID information about the associated DID. As an example, a valid DID document template is shown in Listing 9.

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/v1"
  ],
  "id": "",
  "authentication": [
    {
      "id": "#zAHd7ePaivnaJLK6feRrmrt1JJZb1t6EdeXrhH63hkn4zpAf3",
      "type": "RsaVerificationKey2018",
      "publicKeyPem": "-----BEGIN PUBLIC KEY-----\r\
    nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA5rdiNoPlx9PkJ3mCrRB5\r\
    nWckY5AArfMo5OI4TGP+nN74/tVY2xffyb9CZiJqpfNaYAXcXWJuX/brzYeMaa+sA\r\
    nbFkOMpY4LwHqYZmOrUWvpW/KcTOfvydOwzRjLjVHmWjHeCy5TdupU649r/YRYjKE\r\
    niPFh9RanXEbKeTDozyoEcrqdmW3onqFJ+U+b7kUd9ys0y5lf9F/mZmFrP+SZp0D6\r\
    nKgZC/jUR/ACaSv0jdb710BGROobvanTwXr7dLPVKZxbHAnlnftQ5+4Cjy5zxZO8o\r\n/
    KjKLSjPuO4l55Pth2oLPH7XT+PFUu/ejva1TcgpJooE96ODHLxm094dgVxFdvtS\r\
    neQIDAQAB\r\n-----END PUBLIC KEY-----\r\n"
    }
```

```
  ],
  "service": [
    {
      "id": "#openid",
      "type": "OpenIdConnectVersion1.0Service",
      "serviceEndpoint": "https://openid.example.com/"
    }
  ]
}
```

Listing 9: DID document template example

The read operation is later responsible for linking this template to the corresponding DID by inserting the missing DID information into the resolved DID document. The DID document illustrated in Listing 10 is created based on the DDOT shown in Listing 9 and the DID `did:bba:t:45e6df15dc0a7d91dcccd24fda3b52c3983a214fb0eed0938321c11ec99403cf`.

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/v1"
  ],
  "id": "did:bba:t:45
    e6df15dc0a7d91dcccd24fda3b52c3983a214fb0eed0938321c11ec99403cf",
  "authentication": [
    {
      "id": "did:bba:t:45
    e6df15dc0a7d91dcccd24fda3b52c3983a214fb0eed0938321c11ec99403cf#
    zAHd7ePaivnaJLK6feRrmrt1JJZb1t6EdeXrhH63hkn4zpAf3",
      "type": "RsaVerificationKey2018",
      "publicKeyPem": "-----BEGIN PUBLIC KEY-----\r\
    nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA5rdiNoPlx9PkJ3mCrRB5\r\
    nWckY5AArfMo5OI4TGP+nN74/tVY2xffyb9CZiJqpfNaYAXcXWJuX/brzYeMaa+sA\r\
    nbFkOMpY4LwHqYZmOrUWvpW/KcTOfvydOwzRjLjVHmWjHeCy5TdupU649r/YRYjKE\r\
    niPFh9RanXEbKeTDozyoEcrqdmW3onqFJ+U+b7kUd9ys0y5lf9F/mZmFrP+SZp0D6\r\
    nKgZC/jUR/ACaSv0jdb710BGROobvanTwXr7dLPVKZxbHAnlnftQ5+4Cjy5zxZO8o\r\n/
    KjKLSjPuO4l55Pth2oLPH7XT+PFUu/ejva1TcgpJooE96ODHLxmO94dgVxFdvtS\r\
    neQIDAQAB\r\n-----END PUBLIC KEY-----\r\n"
    }
  ],
  "service": [
    {
      "id": "did:bba:t:45
    e6df15dc0a7d91dcccd24fda3b52c3983a214fb0eed0938321c11ec99403cf#openid",
      "type": "OpenIdConnectVersion1.0Service",
      "serviceEndpoint": "https://openid.example.com/"
    }
  ]
}
```

Listing 10: DID document example

The mechanism to create a valid DID document template is outside the scope of this specification. A library assisting in the creation process is provided in form of the *did-*

document-ts library discussed in section *Implementations*. Separating the DDOT creation process from the DID creation process has the benefit of being independent to the evolution of the DID data model and lets the *bba* method being compatible with future data models and custom contexts. A DDOT creator must ensure that the related DID document conforms to the DID specification [73].

DID Creating and registering a *bba* DID involves multiple steps as shown in Figure 18.

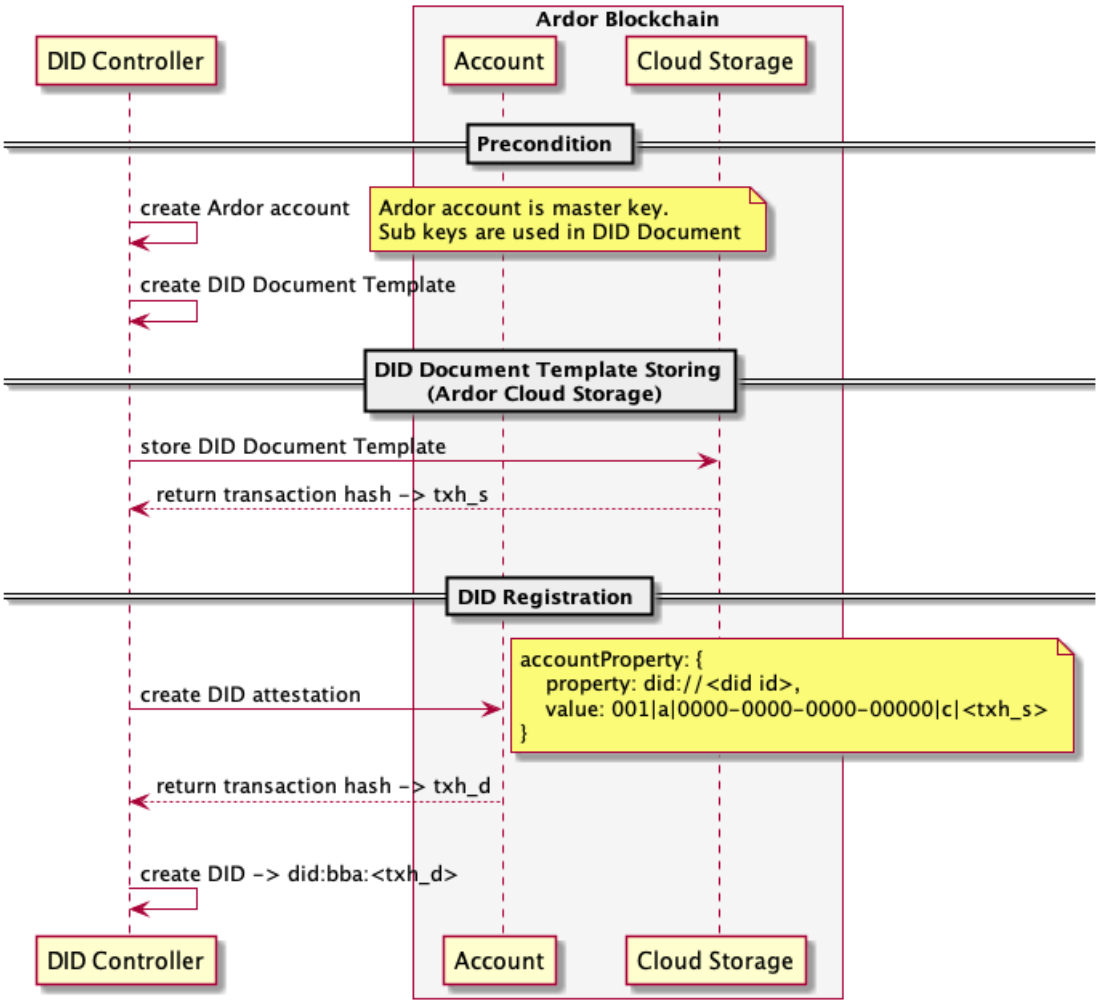

Figure 18: DID creation workflow

As a precondition it is assumed that the DID controller has created an Ardor account and has access to the DDOT one wants to link to the DID.

The first step in the creation process is to store the DDOT with a supported storage mechanism. The Ardor Cloud Storage mechanism is used here. To store the DDOT, the DID controller must create a data cloud transaction, as explained in the Ardor Cloud Storage section, and store the transaction full hash `txh_d` to associate the DDOT with the DID. The next step is to register the DID. To do so, the DID controller creates a DID attestation for the Ardor account the controller controls with a newly generated DID Id and the storage transaction hash `txh_s` as a DDOT reference. The last step is to

create the DID string. The DID string can now be created by concatenating the DID prefix `did:bba:` with the network character on which the DID was registered and the full hash `txh_d` of the authentication transaction in the following way: `did:bba:<m or empty (without leading ':') for mainnet / t for testnet>:txh_d`. An example DID is `did:bba:t:0239684aef4c0d597b4ca5588f69327bed1fedfd576de35e5099c32807bb520e`.

Update Two update procedures are defined. One for updating a DID document template and one for updating a DID controller.

DID Document Template Update To change a DID document, the corresponding DDOT must be updated. This is done by using almost the same workflow as described in the *Create* section and illustrated in Figure 19. Again, it is assumed that a DID controller has created/is in possession of the new DDOT that should replace the current DDOT.

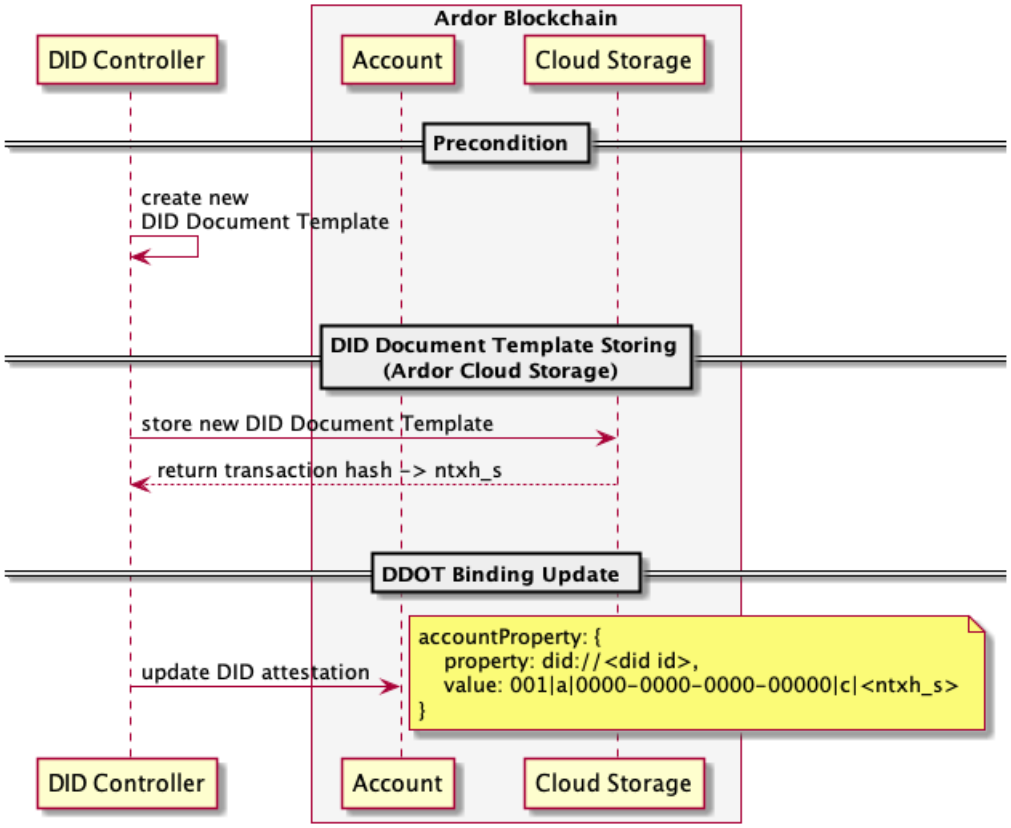

Figure 19: DDOT update workflow

The DID controller stores the new DDOT with one of the support storage mechanisms and updates the current attestation with the new DDOT reference.

DID Controller Update To change the DID controller, two attestations are required, as shown in Figure 20.

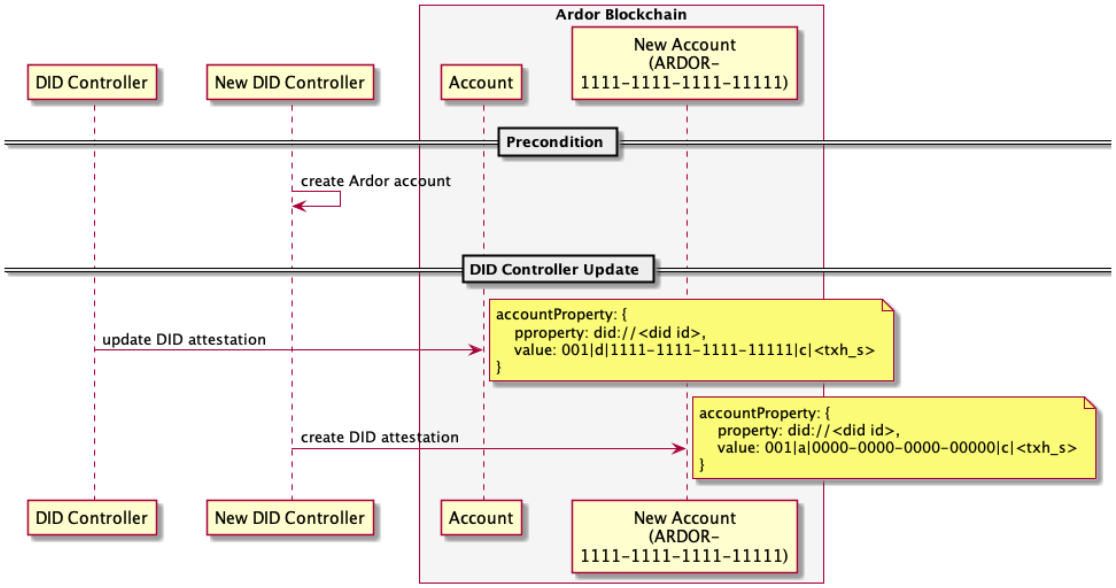

Figure 20: DID controller update workflow

First, the current DID controller updates its attestation with the Ardor account of the new DID controller. To do so, one replaces the redirect account with the 20 account characters of the new DID controller account and sets the state character to `d`. The second step is to *accept* the delegation of control by creating a confirmation with the new controller account in form of a DID attestation. This attestation must equal the pre-updated DID attestation of the current DID controller account.

Deactivate Deactivating a DID requires only one attestation update and is shown in Figure 21.

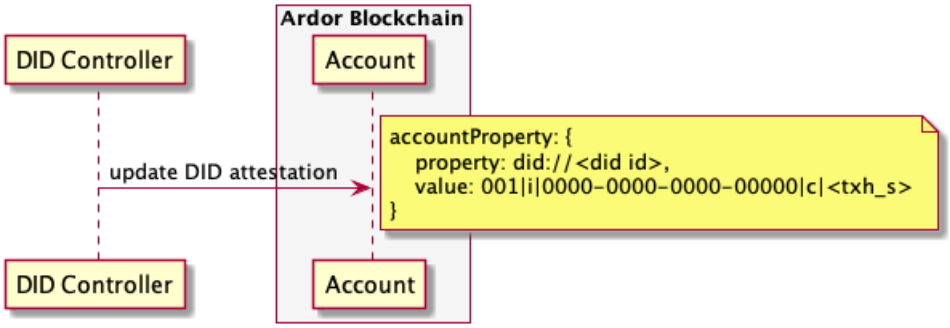

Figure 21: DID deactivation workflow

The DID controller only needs to set the state field of the DID attestation to `i`, which sets the DID to inactive. An inactive DID cannot be reactivated and is locked in this state.

Read (Resolve) The DID resolution process for solving *bba* DIDs mainly consists of three tasks, as shown in Figure 22.

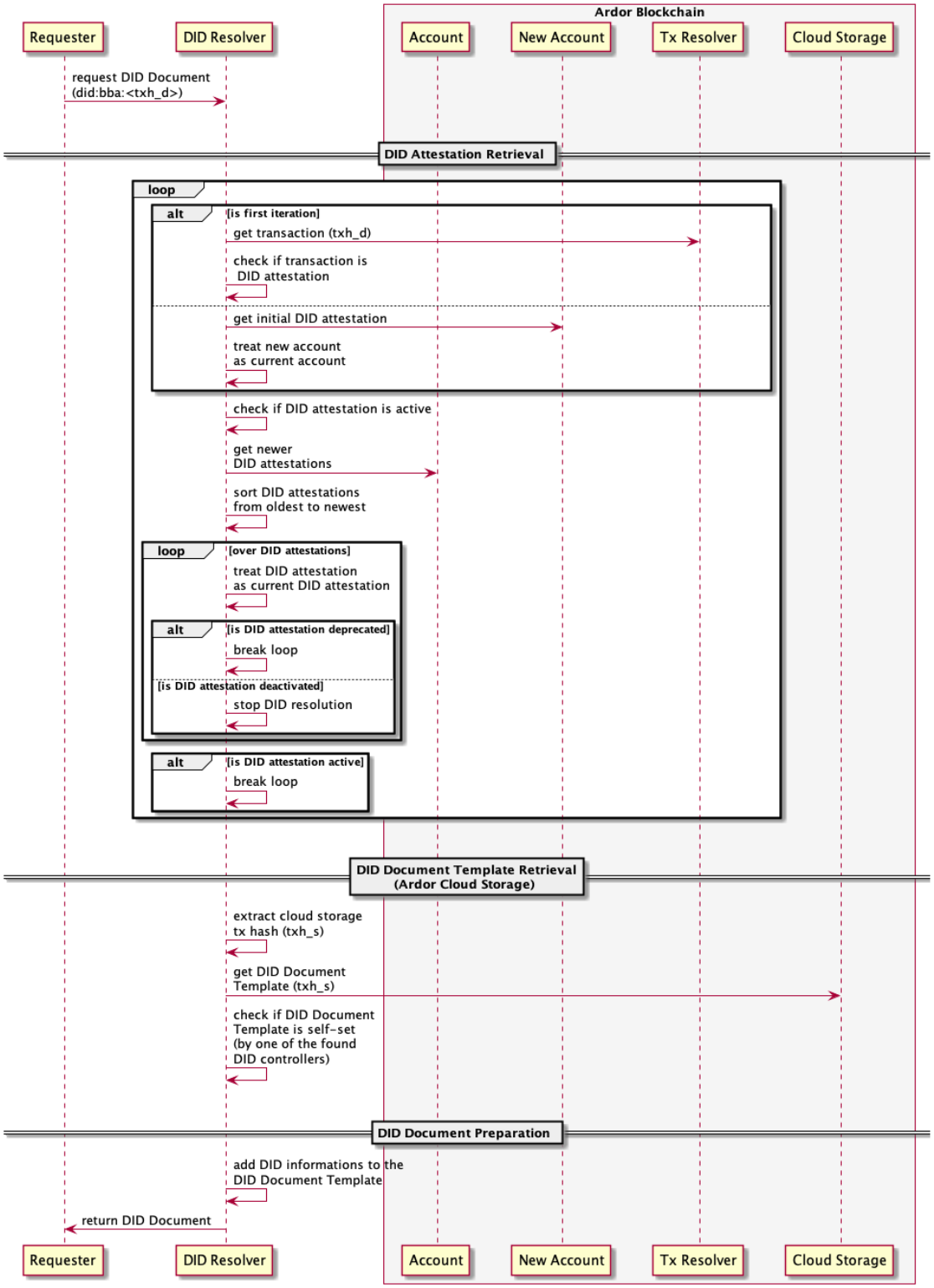

Figure 22: DID resolution workflow

The first task is to **retrieve the current DID attestation** and consists of the following 10 steps.

1. Retrieve the tranaction identified by the transaction full hash `txh_d` as part of the DID string

2. Check if the returned transaction represents a DID attestation

3. Continue with step 6.
4. Get the *delegation acceptance* DID attestation that has been created in the DID controller update process
5. Treat the new account as the current account
6. Check if the DID attestation state is active
7. Retrieve the DID attestations that have been set after the current DID attestation
8. Sort the found DID attestations from the oldest (closest to the current attestation) to the newest
9. Loop over the DID attestations sorted in step 8.
   (a) Treat the found attestation as the current attestation
   (b) Check if the DID attestation state is deprecated. If so, stop the loop.
   (c) Check if the DID attestation state is inactive. If so, stop the whole resolution process.

The second task is to **retrieve the DID document template**. This is done by determining the storage mechanism specified by the storage type field and retrieving the DDOT with the appropriate storage mechanism and the DDOT reference string. The last task is to **convert the DDOT into a DID document** by inserting the DID string information.

After successfully completing these three tasks, a *bba* DID resolver is able to return a verified DID document to the requester.

5.2.4 Security Considerations

The following security considerations must be taken into account.

Key Management A DID controller is always represented as an Ardor account and uses Ardor's native keys as its master key (which is based on the elliptic curve Curve25519 [101]). When an Ardor account is created, a mnemonic passphrase is created from which the private key is derived. The loss of this private key or its disclosure to other parties means the loss of control over the DID and therefore over the DID document.

Ardor provides mechanisms that can help to contain control in case of key compromise in the form of Phasing Transactions [102] as well as to restore keys in form of Shamir's Secret Sharing [103] or HD Wallets [104]. Phased transactions are transactions under certain conditions. To be recognized as a valid transaction, the relevant conditions must be met. A common use case is a transaction with multiple signatures, where several predefined accounts must approve the transaction. Losing or compromising a sub key's private key is less of a concern. As long as the DID controller is in possession of the master key, the controller can simply generate a new key and update the DDOT.

Authorization Because the *bba* DID method is composed of transactions issued in a specific order and with specific content, it cannot prevent a DID controller from acting maliciously and creating wrong statements about a DID. However, the validity and authority of DID operations is proven in the resolution process.

It should be noted that no sub key authenticity check is performed in the DID registration and DDOT binding process. This must be done either by a storage mechanism at DDOT storage time or dynamically by a DID consumer. The latter is preferred since it ensures that a DID controller is in possession of the private key at the time of interaction.

Usage Although not recommended and error-prone, a DID controller or resolver (actor) could use the official Ardor Web Wallet [105] to process the *bba* CRUD operations. This can be done by running an own node or by using an existing node trusted by an actor. The latter involves a compromise between security and usability. It lowers security by adding an additional intermediate (the node runner) between the actor and the blockchain, but increases usability since it does not require to operate a node. It can be assumed that this will be the common way.

In addition to the Web Wallet, an Ardor node provides various REST APIs [106] to interact with the blockchain. The reference implementation discussed in the *Implementations* section uses these APIs to provide an easy and error-preventing way for processing the CRUD operations. When using a remote node (a node that is only accessible via the Internet), the actor must ensure that the communication takes place over a secure TLS connection.

DID Id Collusion Since a DID attestation is retrieved using the DID Id, a DID Id collusion where two DIDs are represented with the same DID Id and collide during a DID controller update process would merge these two DIDs. From that moment on, these DID Ids would always return the same DID attestation and thus the same DID document. They would be inseparable from that point on.

This can be considered unlikely due to the wide range of 20^{62} (20 digits with 62 valid characters `[a-zA-Z0-9]`) possible DID Ids when using a suitable random number generator. Deliberately creating the same DID Id and attempting to force a DID collision would need to involve the new DID controller, since a DID controller update must be accepted by the new controller account.

Resolution Speed Since the DID attestation path is followed in a DID resolution process, the path length has a large influence on the resolution speed. It is expected that it is best practice to control as few DIDs as possible with one account and to use a DID controller account only for the purpose of controlling DIDs.

To speed up the resolution process, a Lightweight Contract [107] could be developed to process database operations (transaction lookups) directly on a node. This would save bandwidth and communication time. A Lightweight Contract is a piece of software stored in Ardor‘s Data Cloud and executed by a contract runner. Every node is able to set up

a contract runner and to run the software, since it is stored in the database within each node.

5.2.5 Privacy Considerations

All aspects described in the Privacy Considerations section within the DID specification are applicable to the *bba* DID method [73]. In addition to that, the following topics must be considered.

Account Tracing Since the resolution process starts with the DID creator account and passes through all accounts that had control over a DID, correlations between these accounts are observable and cannot be avoided. It is to be expected that whenever an account holder who controls a DID controller account updates the DID controller, the new and the old controller account holder have interacted in some way. This reveals an additional relationship between two or more accounts and could be used in combination with other account relationships (for example, payment transactions) to profile and/or identify an entity. This is also true at DID level when two DID attestations cross. A connection between those two DIDs is then no longer deniable.

Again, to avoid account tracing, it is recommended to control as few DIDs as possible with one account and to use a DID controller account only for the purpose of controlling a DID.

DDOT Accessability Using the Ardor Data Cloud as DDOT storage has the advantage of tamper resistance, but the disadvantage of data pruning. Data pruning is an Ardor mechanism to prevent blockchain bloat. It replaces prunable data after a configurable time by the corresponding hash in the node database. While account properties are non-prunable data, data from the Data Cloud can be pruned. This could lead to a denial of DDOT access.

There are two options to prevent this. One is to retrieve a DDOT from a known and trusted archival node (a node that does not prune its database and therefore does not prune DDOTs) or to operate an own archival node.

A slightly modified version of this specification is published within Blobaa‘s GitHub organization [108]. The next section discusses the implementations based on this specification.

5.3 Implementations

Four applications have been developed in the course of this master thesis.

1. **bba-did-method-handler-ts**. The reference implementation for the *bba* CRUD operations.
2. **did-document-ts**. A library to assist in DID document template creation.
3. **bba-did-ui**. A web user interface (UI) for bba CRUD operations.
4. **bba-did-driver**. A Universal Resolver [109] compatible DID driver.

All applications are open sourced, available within Blobaa's GitHub organization [110], and licensed under the permissive MIT License [111].

5.3.1 Reference Implementation

The *ba-did-method-handler-ts* reference implementation is a library that implements the five CRUD operations described in the *DID Method Specification* section. It is written in TypeScript [112], a superset of JavaScript [113]. TypeScript adds static types and type checking to the dynamically typed JavaScript language at development time. It compiles to JavaScript, so that every application written in TypeScript is translated into a corresponding JavaScript application. Since TypeScript is a superset, native JavaScript syntax is also valid. Using JavaScript has the benefit of being able to benefit from the JavaScript ecosystem, including frameworks for executing JavaScript in various environments. JavaScript can be executed in browsers, on operating systems [114], inside desktop applications [115] and mobile applications [116] [117]. Since the reference implementation has no UI dependencies and is suitable for running in browser and server environments (via Node.js), it is flexible enough to be integrated into frontend, backend, desktop and mobile applications. And indeed, the *bba-did-ui* and *bba-did-driver* implementations integrate the *ba-did-method-handler-ts* library to perform the required CRUD operations.

BBA method handler module The library exposes a bba method handler module as its application programming interface (API). It can be imported either as a pre-instantiated instance or as a class definition for self-instantiation as shown in Listing 11.

```
1  import { bbaMethodHandler, BBAMethodHandler, ResolveDIDParams } from "
       @blobaa/bba-did-method-handler-ts";
2
3
4  const moduleInstantiationExample = async (): Promise<void> => {
5
6      const params: ResolveDIDParams = {
7          did:"did:bba:t:45
       e6df15dc0a7d91dcccd24fda3b52c3983a214fb0eed0938321c11ec99403cf"
8      };
9
10     try {
11
12         /* default instance */
13         const response = await bbaMethodHandler.resolveDID("https://
       testardor.jelurida.com", params);
14         console.log(response);
15
16         /* custom instance */
17         const myBBAMethodHandler = new BBAMethodHandler();
18         const response = await myBBAMethodHandler.resolveDID("https://
       testardor.jelurida.com", params);
19         console.log(response);
20
21     } catch (e) { /* error handling */}
22 };
```

```
23
24 moduleInstantiationExample();
```

Listing 11: BBA method handler instance

The *bba* method handler module itself exposes five functions, one for each CRUD operation.

Create The `createDID` function creates a *bba* DID. It has the function signature shown in Listing 12.

```
1  type CreateDIDParams = {
2      didDocumentTemplate: objectAny; // DDOT used for the DID document
3      passphrase: string; // DID controller passphrase
4      isTestnetDid?: boolean; // network type selector
5      fee?: number; // fees in IGNIS
6      feeNQT?: number; // fees in IGNIS NQT
7      // automatic fee calculation is used if no fee is specified
8  }
9
10 type CreateDIDResponse = {
11     did: string; // created DID
12     didDocument: objectAny; // DID document linked to the DID
13 }
14
15 async createDID(url: string, params: CreateDIDParams): Promise<
       CreateDIDResponse>
```

Listing 12: createDID function signature

The `objectAny` type is a custom type that represents a generic object. It ensures that only JavaScript objects are being used. The reference implementation also uses the automatic fee calculation mechanism if no fees are explicitly set.

Read (Resolve) The `resolveDID` function resolves a *bba* DID. It has the function signature shown in Listing 13.

```
1  type ResolveDIDParams = {
2      did: string; // DID to be resolved
3  }
4
5  type ResolveDIDResponse = {
6      did: string; // resolved DID
7      didDocument: objectAny; // resolved DID document
8  }
9
10 async resolveDID(url: string, params: ResolveDIDParams): Promise<
       ResolveDIDResponse>
```

Listing 13: resolveDID function signature

Update DID document template The `updateDIDDocument` function updates a *bba* DID. It has the function signature shown in Listing 14.

```
type UpdateDIDDocumentParams = {
    newDidDocumentTemplate: objectAny; // new DDOT
    passphrase: string; // DID controller passphrase
    did: string; // DID to be updated
    fee?: number; // fees in IGNIS
    feeNQT?: number; // fees in IGNIS NQT
}

type UpdateDIDDocumentResponse = {
    did: string; // updated DID
    newDidDocument: objectAny; // updated DID document
}

async updateDIDDocument(url: string, params: UpdateDIDDocumentParams):
    Promise<UpdateDIDDocumentResponse>
```

Listing 14: updateDIDDocument function signature

Update DID controller The `updateDIDController` function updates a *bba* DID controller. It has the function signature shown in Listing 15.

```
type UpdateDIDControllerParams = {
    passphrase: string; // current DID controller passphrase
    newPassphrase: string; // new DID controller passphrase
    did: string; // DID to be updated
    fee?: number; // fees in IGNIS
    feeNQT?: number; // fees in IGNIS NQT
}

type UpdateDIDControllerResponse = {
    did: string; // DID to be updated
    newControllerAccount: string; // new DID controller account
    oldControllerAccount: string; // old DID controller account
}

async updateDIDController(url: string, params: UpdateDIDControllerParams):
    Promise<UpdateDIDControllerResponse>
```

Listing 15: updateDIDController function signature

Deactivate The `deactivateDID` function deactivates a *bba* DID. It has the function signature shown in Listing 16.

```
type DeactivateDIDParams = {
    did: string; // DID to be deactivated
    passphrase: string; // DID controller passphrase
    fee?: number; // fees in IGNIS
    feeNQT?: number; // fees in IGNIS NQT
}

type DeactivateDIDResponse = {
    deactivatedDid: string; // deactivated DID
```

```
10 };
11
12 async deactivateDID(url: string, params: DeactivateDIDParams): Promise<
       DeactivateDIDResponse>
```

Listing 16: deactivateDID function signature

5.3.2 DID Document Library

The *did-document-ts* library assists in the creation process of DID documents and DID document templates. It is also written in TypeScript and its API consists of the three modules `DIDDocKey`, `DIDDocRelationship`, `DIDDocService`. These modules reflect the core properties of a DID document [73]. A fourth module to compose and create a document / document template is also provided in form of the `DIDDocument` module. All modules are exported as class definitions and need to be instantiated by the user.

DIDDocKey The `DIDDocKey` module represents the public key verification method [73]. In its core it uses Digital Bazaar's [118] *crypto-ld* library [119] for key generation. This has the advantage of being compatible with Digital Bazaar's SSI crypto suite. Since Digital Bazaar is one of the contributor of the DID specification, it can be assumed that being compatible with its crypto suite also means being compatible with the upcoming SSI standard tools. The `Ed25519VerificationKey2018` and `RsaVerificationKey2018` public key types [120] are implemented.

DIDDocRelationship The `DIDDocRelationship` module represents a verification relationship as described in the DID specification.

DIDDocService The `DIDDocService` module represents a service endpoint as described in the DID specification.

DIDDocument The `DIDDocument` module is used to create a DID document or DID document template. It combines all other modules into a DID specification conformal DID document or a DID document template that is compatible with the reference implementation.

Two examples of how to use this library to create a DID document and a DID document template is shown below.

Create DID Document Listing 17 shows an example DID document creation process.

```
1 import { DIDDocKey, DIDDocKeyType, DIDDocRelationship,
      DIDDocRelationshipType, DIDDocument } from "@blobaa/did-document-ts";
2
3
4 const didAlice = "did:bba:5
      ca5fb0b6c59f126f674eb504b7302c69ede9cf431d01dba07809314302e565f";
5
6
```

```
7  const createDIDDocument = async (): Promise<void> => {
8
9      /* create DID document public key  */
10     const key = new DIDDocKey({
11         did: didAlice,
12         keyType: DIDDocKeyType.RSA
13     });
14
15     await key.generate();
16     const publicKey = key.publish();
17
18
19     /* create verification relationships */
20     const authentication = new DIDDocRelationship({
21         relationshipType: DIDDocRelationshipType.AUTHENTICATION,
22         publicKeys: [ publicKey ]
23     });
24
25
26     /* create DID document */
27     const document = new DIDDocument({
28         did: didAlice,
29         relationships: [ authentication ]
30     });
31
32
33     /* publish DID document */
34     console.log(JSON.stringify(document.publish(), undefined, 4));
35     /*
36     {
37         "@context": [
38             "https://www.w3.org/ns/did/v1",
39             "https://w3id.org/security/v1"
40         ],
41         "id": "did:bba:5
    ca5fb0b6c59f126f674eb504b7302c69ede9cf431d01dba07809314302e565f",
42         "authentication": [
43             {
44                 "id": "did:bba:5
    ca5fb0b6c59f126f674eb504b7302c69ede9cf431d01dba07809314302e565f#
    zAHd7h9SzgFk8mf5MwKkWzVKyzGVmW6aJx8qTHe9wmxkE7yWc",
45                 "type": "RsaVerificationKey2018",
46                 "controller": "did:bba:5
    ca5fb0b6c59f126f674eb504b7302c69ede9cf431d01dba07809314302e565f",
47                 "publicKeyPem": "-----BEGIN PUBLIC KEY-----\r\
    nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0GKCpcCQA3n1WEmsdlS0\r\
    nCHjDeIAACnO0Z4CekXEpGUgbfRUPIYaNxwCQnWa4UXlZHojJrkZNASVUi2Q8sIBz\r\
    nTeLAmLaPPWGSPG3VS7Lta+94loFPSxU9OZ/Y5CLHW5xc/ll/SDrMRiDp6/j1vHuH\r\
    nWysShoMY1TWxHkfx8bnN3+OEogYTex85H7quRAKPmmD3iilj5e/zpjZBE5MnlaeC\r\
    nAnFHUQNnoUZyrf95MY22wkNVpCW4WwGXKDE2i/C5Qo/GRYbXH1VuTKNt8EoKcrDp\r\
    nUQy/jkrBH7uXrKwBNDKopeZNLdzXwOTemfxXrOl+VVeZB2x/Ph7MaZF90mxVZ43a\r\
    nUwIDAQAB\r\n-----END PUBLIC KEY-----\r\n"
48             }
49         ]
```

```
    }
    */
};

createDIDDocument();
```

Listing 17: DID document creation example

Create DID Document Template Listing 18 shows an example DID document template creation process.

```
import { DIDDocKey, DIDDocRelationship, DIDDocRelationshipType,
    DIDDocService, DIDDocument } from "@blobaa/did-document-ts";


const createDIDDocumentTemplate = async (): Promise<void> => {

    /* create DID document public keys  */
    const key1 = new DIDDocKey();
    const key2 = new DIDDocKey();

    await key1.generate();
    await key2.generate();

    const publicKey1 = key1.publish();
    const publicKey2 = key2.publish();


    /* create verification relationships */
    const authentication = new DIDDocRelationship({
        relationshipType: DIDDocRelationshipType.AUTHENTICATION,
        publicKeysAsRef: [ publicKey1 ], // referenced public keys
        publicKeys: [ publicKey2 ] // embedded public keys
    });


    /* create services */
    const myService = new DIDDocService({
        name: "mys",
        type: "MyService",
        serviceEndpoint: "https://my.domain.com/mys/",
        prop: "an additional custom property"
    });


    /* create DID document template */
    const document = new DIDDocument({
        contexts: [ "https://my-new.awesome-context.com/my/context" ],
        publicKeys: [ publicKey1 ],
        relationships: [ authentication ],
        services: [ myService ],
    });

```

```
42
43      /* publish DID document template */
44      console.log(JSON.stringify(document.publish(), undefined, 4));
45      /*
46      {
47          "@context": [
48              "https://www.w3.org/ns/did/v1",
49              "https://w3id.org/security/v1",
50              "https://my-new.awesome-context.com/my/context"
51          ],
52          "id": "",
53          "publicKey": [
54              {
55                  "id": "#z6MkoZy5Zb2RhHHJ6Ut4smTRWVzBc9HZGCNKUHxp5u7sNsrk",
56                  "type": "Ed25519VerificationKey2018",
57                  "publicKeyBase58": "
    A7i2yLmzMjnpyz3NCCVafQSBna1hrK7xnH3tFd9rTf5N"
58              }
59          ],
60          "authentication": [
61              "#z6MkoZy5Zb2RhHHJ6Ut4smTRWVzBc9HZGCNKUHxp5u7sNsrk",
62              {
63                  "id": "#z6MkhuK2biLQuNj4vUKZpqrWBCB4Nu9qoWnK3peNjJueWv5o",
64                  "type": "Ed25519VerificationKey2018",
65                  "publicKeyBase58": "4
    T3z1U5yZqEboyUs9GtfL6d4ZKszPdXxMojSu2wdbhJR"
66              }
67          ],
68          "service": [
69              {
70                  "id": "#mys",
71                  "type": "MyService",
72                  "serviceEndpoint": "https://my.domain.com/mys/",
73                  "prop": "an additional custom property"
74              }
75          ]
76      }
77      */
78  };
79
80  createDIDDocumentTemplate();
```

Listing 18: DID document template creation example

5.3.3 Web Interface

The *bba-did-ui* project contains the source code for the **w**eb **U**I for **b**ba **C**RUD **o**perations, called WUBCO. It wraps the *bba-did-method-handler-ts* and *did-document-ts* libraries into a single-page application (SPA) [121] to provide human friendly access to the *bba* DID method. A single-page application is a static website that dynamically manipulates the HTML document on the client side. It does not rely on an external server for HTML manipulation. This has the benefit of being able to be served from a static web server. A

static web server is an entity that serves the requested resources as-is without implementing logic for conditional content delivery [122]. There are free to use static web servers like GitHub Pages [123]. GitHub Pages serves static websites from within GitHub repositories. Since all implementations are open sourced and published via GitHub repositories, the WUBCO website is also served by GitHub Pages and available under `https://wubco.blobaa.dev`. Two Ardor testnet accounts are provided in the appendix to try it out. WUBCO is developed with Next.js [124], a framework for React [125] based SPAs.

5.3.4 DID Driver

The *bba-did-driver* project contains the source code for the `bba-did-driver` Docker image. It wraps the *bba-did-method-handler-ts* library into a Universal Resolver compatible DID driver [126]. Docker is a platform to abstract applications from operating systems. Applications are packaged with all its dependencies into so called containers that run within the docker environment [127]. This has the benefit of being platform independent as long as the docker environment is available. Docker images contain the instructions for docker containers to run a specific application. It is comparable to a class definition (an image) that needs to be instantiated (a container) to be used. The Universal Resolver project aims to provide a unified interface for DID method resolution. It is an open source project and is maintained by the DIF. It is open for contribution to expand the number of supported DID methods. A Guideline for DID driver integration is also provided [128]. The following requirements must be met in order to be accepted as a valid DID method specific DID driver.

1. The driver must be fully open source under a permissive license.
2. The driver image must be published on DockerHub with version tags.
3. The driver must be well-documented, tested and working.
4. The documentation should be clear enough to explain how to run the driver, how to troubleshoot it, and a process for updating the driver over time.

DockerHub is a platform for publishing and storing Docker images [129].

The following section discusses the evaluation of the *bba* DID method specification and implementations.

5.4 Evaluation

The following two evaluation types should be discussed.

1. **Concept Evaluation**. The evaluation of the *bba* DID method specification.
2. **Implementation Evaluation**. The evaluating of the *bba-did-method-handler-ts* reference implementation.

Concept Evaluation Since a DID method specification must fulfill the requirements described in section *DID Method Requirements* to be listed in the official DID method registry [77], the best way to evaluate the *bba* DID method concept is to request an integration into that registry. Each integration request goes through a review and approval process and is made in form of a pull request within the *Decentralized Identifier Core Registries* GitHub repository [130]. A pull request is a colaboration tool to propose changes to a git repository [131].

A pull request for the *bba* DID method [132] has been created and accepted so that the *bba* DID method is now listed within the official DID method registry.

Due to the fact that the DID specification is still under development, it cannot be ensured that the *bba* DID method aligns to the first stable version. Although this is true, it can be assumed that alignment can be achieved with reasonable effort. The reason for that is the separation of the DID document from the *bba* DID method specification. Being independent from the data types and properties of the DID document makes it independent from changes to these elements.

Implementation Evaluation Similar to the concept evaluation, an integration request [133] within the Universal Resolver repository has been made to prove the interoperability of the reference implementation in form of the *bba-did-driver* project. It was also accepted and the *bba* DID method is now resolvable with the Universal Resolver. Two *bba* DIDs can be found in the appendix to test out the Universal Resolver integration. Unfortunately, at the time of writing, a pull request for updating the Universal Resolver frontend [134] to show the integration of the *bba* DID method has not been merged. This leads to a the inconsistency of supporting the *bba* DID method without announcing it.

In addition to that, the *bba-did-ui* project shows that the reference implementation is capable of running in frontend (browser) and backend (Node.js) environments.

6 Summary

In the course of this thesis a new identity management model using the blockchain technology as the infrastructure for decentralize identifiers, called DIDs, has been described and discussed. This identity model is called self-sovereign identity (SSI) and is still in its infancy. In addition to that, the new DID method *bba* has been developed and evaluated to enable the Blobaa project to participate in the SSI model. The bba DID method has been accepted to be listed within the official DID method registry and to be integrated into the Universal Resolver project. An overview of the blockchain technology and how the SSI model differentiates from conventional identity management models was also given.

6.1 Considerations

Since the SSI model is still under development and some essential parts (like the DID) are not yet finally defined, the adoption of this identity model is not predictable. This is also true since identity providers must redefine their roles in this model. It is to be expected that identity provider still manager user identities in form of identity hubs [12]. Identity hubs are encrypted, cloud based data vaults for storing user credentials. Another crucial aspect is user experience. Agents and digital wallets must be easy to use without sacrificing security to achieve broad acceptance. A key part here is key management. A third aspect is the integration within services. Service providers must be willing to accept and handle self-managed SSI identities.

Even though the presented *bba* DID method is acknowledged as an officially supported DID method, additional tests have to be performed in a near-production environment to prove interoperability, security and reliability.

6.2 Outlook

Although still in development, the SSI model has already raised attention. This applies not only to the consumer and corporate sectors, but also to the public sector with various initiatives. Being driven by the public sector offers the advantage of not being dependent on the market alone, but also of being legally enforceable.

If the SSI model manages to gain adoption, there is a chance that the *bba* DID method will be used as one way to create and manage DIDs. This is especially true since the *bba* DID method is listed in the DID method registry and can therefore be assumed to be compatible with the SSI ecosystem.

Appendix

A BBA DIDs

A.1 Mainnet

`did:bba:47ef0798566073ea302b8178943aaa83f227614d6f36a4d2bcd92993bbed6044`

A.2 Testnet

`did:bba:t:45e6df15dc0a7d91dcccd24fda3b52c3983a214fb0eed0938321c11ec99403cf`

B Ardor Testnet Accounts

B.1 Account 1

Address:
`ARDOR-7EZ3-SM5A-2A9G-2BPNP`

Passphrase:
`ten opera segment ride apology arrow mushroom gadget invest ridge grid parent`

Initial Balance:
101 IGNIS

B.2 Account 2

Address:
`ARDOR-MS9Q-MH4E-GWBD-33RZ8`

Passphrase:
`place flower deliver kitchen clay window drum fluid size quality truck material`

Initial Balance:
101 IGNIS

C GitHub Repositories

C.1 BBA DID Method Specification

Name:
`bba-did-method-specification`

Version:

v0.1.1

Link:
`https://github.com/blobaa/bba-did-method-specification`

C.2 Reference Implementation

Name:
`bba-did-method-handler-ts`

Version:
v0.1.11

Link:
`https://github.com/blobaa/bba-did-method-handler-ts`

C.3 DID Document Library

Name:
`did-document-ts`

Version:
v0.1.6

Link:
`https://github.com/blobaa/did-document-ts`

C.4 Web Interface

Name:
`bba-did-ui`

Version:
v1.0

Link:
`https://github.com/blobaa/bba-did-ui`

C.5 DID Driver

Name:
`bba-did-drive`

Version:
v0.2.1

Link:
`https://github.com/blobaa/bba-did-driver`

List of Figures

List of Tables

Listings

References

[1] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. URL: https://bitcoin.org/bitcoin.pdf (visited on 08/17/2020).

[2] G. Suciu et al. "Comparative Analysis of Distributed Ledger Technologies". In: *2018 Global Wireless Summit (GWS)*. 2018, pp. 370–373.

[3] Ethereum Foundation. *Ethereum whitepaper*. URL: https://ethereum.org/en/whitepaper/ (visited on 08/17/2020).

[4] Jelurida Swiss SA. *Ardor whitepaper*. URL: https://www.jelurida.com/sites/default/files/JeluridaWhitepaper.pdf (visited on 08/27/2020).

[5] Block.one. *EOS whitepaper*. URL: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md (visited on 08/25/2020).

[6] Brad Chase and Ethan MacBrough. *Analysis of the XRP Ledger Consensus Protocol*. 2018. arXiv: 1802.07242 [cs.DC].

[7] U. Cali et al. "DLT / Blockchain in Transactive Energy Use Cases Segmentation and Standardization Framework". In: *2019 IEEE PES Transactive Energy Systems Conference (TESC)*. 2019, pp. 1–5.

[8] Siemens. *Trusted traceability*. URL: https://assets.new.siemens.com/siemens/assets/api/uuid:de496ba4-0081-48f5-965b-4963879b2d43/vrfb-b10033-00-7600sbblockchainfb-144.pdf (visited on 08/25/2020).

[9] Z. Wang et al. "ArtChain: Blockchain-Enabled Platform for Art Marketplace". In: *2019 IEEE International Conference on Blockchain (Blockchain)*. 2019, pp. 447–454.

[10] Rebooting the Web of Trust. *Decentralized-Public-Key-Infrastructure*. URL: https://www.weboftrust.info/downloads/dpki.pdf (visited on 06/11/2020).

[11] Christopher Allen. *The Path to Self-Sovereign Identity*. URL: https://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html (visited on 05/23/2020).

[12] Alex Preukschat and Drummond Reed. *Self-Sovereign Identity MEAP Version 3*. Manning, 2020-05-14. ISBN: 9781617296598.

[13] Attila Aldemir. *Blobaa website*. URL: https://www.blobaa.dev (visited on 08/25/2020).

[14] Sovrin Foundation. *Sovrin whitepaper*. URL: https://sovrin.org/wp-content/uploads/Sovrin-Protocol-and-Token-White-Paper.pdf (visited on 08/17/2020).

[15] A. J. M. Paul C. van Oorschot Scott A. Vanstone. *Handbook of applied cryptography*. CRC Press, 1996. ISBN: 0-8493-8523-7.

[16] A. M. Antonopoulos. *Mastering Bitcoin*. O'Reilly Media, Inc., 2014. ISBN: 9781449374044.

[17] IETF. *US Secure Hash Algorithms (SHA and HMAC-SHA)*. URL: https://tools.ietf.org/html/rfc4634 (visited on 08/19/2020).

[18] Ethereum Foundation. *What is Ethereum? desription.* URL: https://github.com/ethereum/homestead-guide/blob/master/source/introduction/what-is-ethereum.rst (visited on 08/17/2020).

[19] Jelurida Swiss SA. *Ardor Blockchain Platform Design.* URL: https://www.jelurida.com/sites/default/files/ArdorPlatformDesign.pdf (visited on 08/19/2020).

[20] Bitcoin Community. *Bitcoin Wiki.* URL: https://en.bitcoin.it/wiki/Controlled_supply#Projected_Bitcoins_Short_Term (visited on 08/17/2020).

[21] Jelurida Swiss SA. *Ardor Source Code.* URL: https://bitbucket.org/Jelurida/ardor/src/master/ (visited on 08/17/2020).

[22] Ethereum Foundation. *Ethash wiki.* URL: https://eth.wiki/en/concepts/ethash/ethash (visited on 08/26/2020).

[23] Sunny King and Scott Nadal. "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake". In: *self-published paper, August* 19 (2012), p. 1.

[24] Iddo Bentov, Rafael Pass, and Elaine Shi. "Snow White: Provably Secure Proofs of Stake." In: *IACR Cryptol. ePrint Arch.* 2016 (2016), p. 919.

[25] Aggelos Kiayias et al. "Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol". In: *Advances in Cryptology – CRYPTO 2017*. Ed. by Jonathan Katz and Hovav Shacham. Cham: Springer International Publishing, 2017, pp. 357–388. ISBN: 978-3-319-63688-7.

[26] Jelurida Swiss SA. *Nxt Whitepaper.* URL: https://nxtdocs.jelurida.com/Nxt_Whitepaper (visited on 08/21/2020).

[27] V. Osmov et al. "On the Blockchain-Based General-Purpose Public Key Infrastructure". In: *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*. 2019, pp. 1–8.

[28] X. Liu et al. "MDP-Based Quantitative Analysis Framework for Proof of Authority". In: *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. 2019, pp. 227–236.

[29] C. T. Nguyen et al. "Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities". In: *IEEE Access* 7 (2019), pp. 85727–85745.

[30] Jelurida Swiss SA. *Jelurida webiste.* URL: https://www.jelurida.com (visited on 08/26/2020).

[31] Irving S. Reed and Xuemin Chen. "Reed-Solomon Codes". In: *Error-Control Coding for Data Networks*. Boston, MA: Springer US, 1999, pp. 233–284. ISBN: 978-1-4615-5005-1. DOI: 10.1007/978-1-4615-5005-1_6. URL: https://doi.org/10.1007/978-1-4615-5005-1_6.

[32] Jelurida Swiss SA. *Ardor Address Format Description.* URL: https://ardordocs.jelurida.com/RS_Address_Format (visited on 04/24/2020).

[33] I. Bourass et al. “Towards a new model of management and securing digital identities”. In: *2014 International Conference on Next Generation Networks and Services (NGNS)*. 2014, pp. 308–312.

[34] Luigi Lo Iacono, Nils Gruschka, and Peter Nehren. “Mobile Personal Identity Provider Based on OpenID Connect”. In: *Trust, Privacy and Security in Digital Business*. Ed. by Javier Lopez, Simone Fischer-Hübner, and Costas Lambrinoudakis. Cham: Springer International Publishing, 2017, pp. 19–31. ISBN: 978-3-319-64483-7.

[35] Bundesdruckerei GmbH. *Glossary*. URL: `https://www.bundesdruckerei.de/en/Glossary` (visited on 05/14/2020).

[36] H. L’Amrani et al. “Identity management systems: Laws of identity for models7 evaluation”. In: *2016 4th IEEE International Colloquium on Information Science and Technology (CiSt)*. 2016, pp. 736–740.

[37] A. JØsang and S. Pope. “User Centric Identity Management AusCERT Conference”. In: 2005.

[38] X. Zhu and Y. Badr. “A Survey on Blockchain-Based Identity Management Systems for the Internet of Things”. In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2018, pp. 1568–1573.

[39] Microsoft. *Microsoft Account*. URL: `https://account.microsoft.com` (visited on 05/29/2020).

[40] Microsoft. *Skype*. URL: `https://www.skype.com` (visited on 05/29/2020).

[41] Microsoft. *OneDrive*. URL: `https://www.microsoft.com/en/microsoft-365/onedrive/online-cloud-storage` (visited on 05/29/2020).

[42] Microsoft. *Outlook*. URL: `https://outlook.live.com` (visited on 05/29/2020).

[43] Google. *Google Account*. URL: `https://myaccount.google.com` (visited on 05/29/2020).

[44] Google. *GMail*. URL: `https://www.google.com/intl/en/gmail/about/` (visited on 05/29/2020).

[45] Google. *Google Docs*. URL: `https://www.google.com/intl/en/docs/about/` (visited on 05/29/2020).

[46] Google. *Google Play*. URL: `https://play.google.com` (visited on 05/29/2020).

[47] European union. *eIDAS regulation*. URL: `https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32014R0910` (visited on 08/27/2020).

[48] Hal Lockhart and B Campbell. “Security assertion markup language (SAML) V2. 0 technical overview”. In: *OASIS Committee Draft* 2 (2008), pp. 94–106.

[49] IETF. *The OAuth 2.0 Authorization Framework*. URL: `https://tools.ietf.org/html/rfc6749` (visited on 06/09/2020).

[50] The OpenID Foundation. *OpenID Connect*. URL: `https://openid.net/specs/openid-connect-core-1_0-final.html` (visited on 06/09/2020).

[51] D. Choi, S. Jin, and H. Yoon. "Trust Management for User-Centric Identity Management on the Internet". In: *2007 IEEE International Symposium on Consumer Electronics*. 2007, pp. 1–4.

[52] panva. *oidc-provider*. URL: https://github.com/panva/node-oidc-provider (visited on 08/26/2020).

[53] Microsoft. *Introducing Verifiable Credentials as a Service by Azure AD*. URL: https://didproject.azurewebsites.net/docs/overview.html (visited on 06/21/2020).

[54] Global Legal Entity Identifier Foundation. *GLEIF Website*. URL: https://www.gleif.org/en/ (visited on 06/22/2020).

[55] Consensys. *GLEIF and uPort Test Verified Data Exchange in Financial and Commercial Transactions*. URL: https://consensys.net/blog/press-release/gleif-uport-test-verified-data-exchange-in-financial-and-commercial-transactions/ (visited on 06/22/2020).

[56] Main incubator GmbH. *lissi website*. URL: https://lissi.id/start (visited on 06/22/2020).

[57] European Economic and Social Committee. *European self-sovereign identity framework*. URL: https://www.eesc.europa.eu/sites/default/files/files/1._panel_-_daniel_du_seuil.pdf (visited on 06/22/2020).

[58] European Economic and Social Committee. *eIDAS bridge*. URL: https://joinup.ec.europa.eu/sites/default/files/document/2020-04/SSI%20eIDAS%20Bridge%20-%20Use%20cases%20and%20Technical%20Specifications%20v1.pdf (visited on 06/29/2020).

[59] TrustNetFI. *FIndy website*. URL: https://www.findy.fi (visited on 06/22/2020).

[60] Alastria. *Alastria website*. URL: https://alastria.io/en/id-alastria (visited on 06/22/2020).

[61] Business Design. *Meine Sichere Id website*. URL: https://meinesichereid.org (visited on 06/22/2020).

[62] Dutch Blockchain Coalition. *Blockchain for Good vision document*. URL: https://dutchblockchaincoalition.org/uploads/pdf/Visiondocument-Blockchain-For-Good-EN.pdf (visited on 06/22/2020).

[63] Deutsche Telekom AG. *Blockchainscooter*. URL: https://laboratories.telekom.com/blockchain-scooter (visited on 06/22/2020).

[64] M. S. Ferdous, F. Chowdhury, and M. O. Alassafi. "In Search of Self-Sovereign Identity Leveraging Blockchain Technology". In: *IEEE Access* 7 (2019), pp. 103059–103079.

[65] A. Grüner, A. Mühle, and C. Meinel. "An Integration Architecture to Enable Service Providers for Self-sovereign Identity". In: *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*. 2019, pp. 1–5.

[66] P. C. Bartolomeu et al. “Self-Sovereign Identity: Use-cases, Technologies, and Challenges for Industrial IoT”. In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2019, pp. 1173–1180.

[67] Alexander Mühle et al. “A Survey on Essential Components of a Self-Sovereign Identity”. In: July 2018.

[68] D. W. Chadwick et al. “Improved Identity Management with Verifiable Credentials and FIDO”. In: *IEEE Communications Standards Magazine* 3.4 (2019), pp. 14–20.

[69] Lux, Zoltán András and Thatmann, Dirk and Zickau, Sebastian and Beierle, Felix. “Distributed-Ledger-based Authentication with Decentralized Identifiers and Verifiable Credentials”. In: *Proc. 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE, 2020.

[70] Q. Stokkink and J. Pouwelse. “Deployment of a Blockchain-Based Self-Sovereign Identity”. In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2018, pp. 1336–1342.

[71] Blockchain Bundesverband. *A position paper on blockchain enabled identity and the road ahead*. URL: `https://www.bundesblock.de/wp-content/uploads/2019/01/ssi-paper.pdf` (visited on 06/18/2020).

[72] Trust over IP Foundation. *Introducing the Trust over IP Foundation*. URL: `https://trustoverip.org/wp-content/uploads/sites/98/2020/05/toip_introduction_050520.pdf` (visited on 06/17/2020).

[73] W3C. *Decentralized Identifiers (DIDs) v1.0*. URL: `https://www.w3.org/TR/2020/WD-did-core-20200723/` (visited on 07/23/2020).

[74] W3C. *Verifiable Credentials Data Model 1.0*. URL: `https://www.w3.org/TR/vc-data-model` (visited on 06/18/2020).

[75] Trust over IP Foundation. *Trust over IP Foundation Website*. URL: `https://trustoverip.org` (visited on 06/17/2020).

[76] W3C. *JSON-LD 1.1*. URL: `https://www.w3.org/TR/json-ld11` (visited on 06/26/2020).

[77] W3C. *DID Specification Registries*. URL: `https://w3c.github.io/did-spec-registries` (visited on 08/30/2020).

[78] W3C. *Peer DID Method Specification*. URL: `https://identity.foundation/peer-did-method-spec/` (visited on 08/30/2020).

[79] Hyperledger Aries. *Aries RFC 0005: DID Communication*. URL: `https://github.com/hyperledger/aries-rfcs/tree/master/concepts/0005-didcomm` (visited on 06/26/2020).

[80] DIF. *DIF website*. URL: `https://identity.foundation` (visited on 08/27/2020).

[81] Decentralized Identity Foundation. *DIDComm Messaging*. URL: `https://github.com/decentralized-identity/didcomm-messaging` (visited on 08/31/2020).

[82] IETF. *Examples of Protecting Content Using JSON Object Signing and Encryption (JOSE)*. URL: https://www.rfc-editor.org/rfc/rfc7520.html (visited on 06/26/2020).

[83] Hyperledger Aries. *Issue Credential Protocol 1.0*. URL: https://github.com/hyperledger/aries-rfcs/blob/master/features/0036-issue-credential/README.md (visited on 06/28/2020).

[84] Hyperledger Aries. *Tic Tac Toe Protocol 1.0*. URL: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0003-protocols/tictactoe/README.md (visited on 06/28/2020).

[85] IETF. *A Universally Unique IDentifier (UUID) URN Namespace*. URL: https://tools.ietf.org/html/rfc4122 (visited on 06/28/2020).

[86] Tom Preston-Werner. *Semantic Versioning 2.0.0*. URL: https://semver.org (visited on 06/29/2020).

[87] Hyperledger Aries. *Message ID and Threading*. URL: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0008-message-id-and-threading/README.md (visited on 06/29/2020).

[88] Hyperledger Aries. *Present Proof Protocol 1.0*. URL: https://github.com/hyperledger/aries-rfcs/blob/master/features/0037-present-proof/README.md (visited on 07/02/2020).

[89] M. K. Ibrahem. "Modification of Diffie-Hellman key exchange algorithm for Zero knowledge proof". In: *2012 International Conference on Future Communication Networks*. 2012, pp. 147–152.

[90] IETF. *JSON Web Token (JWT)*. URL: https://tools.ietf.org/html/rfc7519 (visited on 07/03/2020).

[91] bitcoinpaperwallet.com. *Bitcoin Paper Wallet*. URL: https://bitcoinpaperwallet.com (visited on 07/06/2020).

[92] A. G. Khan et al. "Security Of Cryptocurrency Using Hardware Wallet And QR Code". In: *2019 International Conference on Innovative Computing (ICIC)*. 2019, pp. 1–10.

[93] Bitcoin Project. *Hierarchical Deterministic Wallets*. URL: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki (visited on 07/06/2020).

[94] IETF. *Augmented BNF for Syntax Specifications: ABNF*. URL: https://tools.ietf.org/html/rfc5234 (visited on 08/22/2020).

[95] IETF. *ASCII format for Network Interchange*. URL: https://tools.ietf.org/html/rfc20 (visited on 08/22/2020).

[96] IETF. *Guidelines for Writing RFC Text on Security Considerations*. URL: https://tools.ietf.org/html/rfc3552 (visited on 08/22/2020).

[97] IETF. *Privacy Considerations for Internet Protocols*. URL: https://tools.ietf.org/html/rfc6973 (visited on 08/22/2020).

[98] Jelurida Swiss SA. *Ignis webiste*. URL: https://www.jelurida.com/ignis (visited on 08/18/2020).

[99] Jelurida Swiss SA. *Ardor Account Properties*. URL: https://ardordocs.jelurida.com/Account_Properties (visited on 08/18/2020).

[100] Jelurida Swiss SA. *Ardor Data Cloud*. URL: https://ardordocs.jelurida.com/Data_Cloud (visited on 08/18/2020).

[101] Pascal Sasdrich and Tim Güneysu. "Efficient Elliptic-Curve Cryptography Using Curve25519 on Reconfigurable Devices". In: *Reconfigurable Computing: Architectures, Tools, and Applications*. Ed. by Diana Goehringer et al. Cham: Springer International Publishing, 2014, pp. 25–36. ISBN: 978-3-319-05960-0.

[102] Jelurida Swiss SA. *Ardor Phasing Transactions*. URL: https://ardordocs.jelurida.com/Phasing_Transactions (visited on 08/18/2020).

[103] Jelurida Swiss SA. *Ardor Shamir's Secret Sharing*. URL: https://ardordocs.jelurida.com/Secret_Sharing (visited on 08/18/2020).

[104] Jelurida Swiss SA. *Ardor HD Wallets*. URL: https://ardordocs.jelurida.com/From_Simple_Wallet_to_HD_Wallet (visited on 08/18/2020).

[105] Jelurida Swiss SA. *Ardor Web Wallet*. URL: https://ardor.jelurida.com/index.html (visited on 08/18/2020).

[106] Jelurida Swiss SA. *Ardor REST APIs*. URL: https://ardordocs.jelurida.com/API (visited on 08/18/2020).

[107] Jelurida Swiss SA. *Ardor Lightweight Contracts*. URL: https://ardordocs.jelurida.com/Lightweight_Contracts (visited on 08/18/2020).

[108] Attila Aldemir. *BBA DID method specification*. URL: https://github.com/blobaa/bba-did-method-specification (visited on 08/29/2020).

[109] DIF. *Universal Resolver website*. URL: https://dev.uniresolver.io (visited on 08/22/2020).

[110] Attila Aldemir. *Blobaa GitHub organization*. URL: https://github.com/blobaa (visited on 08/29/2020).

[111] Massachusetts Institute of Technology. *MIT License*. URL: https://opensource.org/licenses/MIT (visited on 08/29/2020).

[112] Microsoft. *TypeScript website*. URL: https://www.typescriptlang.org (visited on 08/22/2020).

[113] ECMA. *ECMA-262 specification*. URL: https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf (visited on 08/22/2020).

[114] OpenJS Foundation. *Node.js website*. URL: https://nodejs.org (visited on 08/22/2020).

[115] OpenJS Foundation. *Electron website*. URL: https://www.electronjs.org (visited on 08/22/2020).

[116] Facbook. *React Native website*. URL: https://reactnative.dev (visited on 08/22/2020).

[117] Ionic. *Ionic website*. URL: https://ionicframework.com (visited on 08/22/2020).

[118] Digital Bazaar. *Digital Bazaar website*. URL: https://digitalbazaar.com (visited on 08/23/2020).

[119] Digital Bazaar. *crypto-ld source code*. URL: https://github.com/digitalbazaar/crypto-ld/tree/v3.8.0 (visited on 08/23/2020).

[120] W3C Community Group. *Linked Data Cryptographic Suite Registry*. URL: https://w3c-ccg.github.io/ld-cryptosuite-registry/ (visited on 08/23/2020).

[121] S. Deshmukh, D. Mane, and A. Retawade. "Building a Single Page Application Web Front-end for E-Learning site". In: *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*. 2019, pp. 985–987.

[122] Mozilla. *What is a web server? website*. URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server (visited on 08/24/2020).

[123] GitHub. *GitHub Pages website*. URL: https://pages.github.com (visited on 08/24/2020).

[124] Vercel. *Next.js website*. URL: https://nextjs.org (visited on 08/24/2020).

[125] Facbook. *React website*. URL: https://reactjs.org (visited on 08/24/2020).

[126] DIF. *Universal Resolver — GitHub repository*. URL: https://github.com/decentralized-identity/universal-resolver (visited on 08/24/2020).

[127] Docker Inc. *What is a Container? website*. URL: https://www.docker.com/resources/what-container (visited on 08/24/2020).

[128] DIF. *Universal Resolver — Driver Development*. URL: https://github.com/decentralized-identity/universal-resolver/blob/master/docs/driver-development.md (visited on 08/24/2020).

[129] Docker Inc. *Docker Hub website*. URL: https://hub.docker.com (visited on 08/24/2020).

[130] W3C Decentralized Identifier Working Group. *Decentralized Identifier Core Registries*. URL: https://github.com/w3c/did-spec-registries (visited on 08/25/2020).

[131] GitHub. *About pull requests website*. URL: https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests (visited on 08/25/2020).

[132] Attila Aldemir. *bba DID method integration request*. URL: https://github.com/w3c/did-spec-registries/pull/103 (visited on 08/10/2020).

[133] Attila Aldemir. *bba DID driver integration request*. URL: https://github.com/decentralized-identity/universal-resolver/pull/137 (visited on 08/11/2020).

[134] Attila Aldemir. *Universal resolver frontend updated request*. URL: https://github.com/decentralized-identity/universal-resolver-frontend/pull/26 (visited on 08/25/2020).

Declaration

I declare on oath that I have written the submitted thesis independently and without the assistance of others, that I have not used any sources or aids other than those indicated, and that I have marked as such the passages taken from the sources used, either literally or in terms of content.

Place, Date

Legally binding signature