WHITEPAPER

Analysing the SSI World

A description of the current state of the Self Sovereign Identity (SSI) movement as a technological pillar of Gaia-X

Peer Review Version 1.1

October 2023

Gefördert durch:

Bundesministerium
für Wirtschaft
und Klimaschutz

aufgrund eines Beschlusses
des Deutschen Bundestages

www.gxfs.de

Analysing the SSI World

This Whitepaper was initialled written by Mirko Mollik (formerly TrustCerts GmbH, now Fraunhofer FIT) and Robin Klemens (Institute for Internet Security) in August 2022. Leading up to the publication of the Whitepaper in August 2023, a lot of new developments in the SSI ecosystem have led the publisher to pursue a peer review of the paper. The peer review was conducted in October 2023. Please refer to the imprint for a full list of peer reviewers.

Executive Summary

The analysis at hand describes the current state of the global Self Sovereign Identity (SSI) movement. The report covers core concepts like Decentralized Identifiers (DID), Verifiable Credentials (VCs) with their different flavors of implementations and schemas as the basis for VCs.

Further, the document discusses different transport protocols in SSI and highlights DIDComm messaging v1 and v2 as the most used protocol and OpenID for Verifiable Credentials as an evolving alternative. Digital signatures are one of the corner pieces of SSI, and thus the authors analyze modern vs. old crypto algorithms concerning crypto agility, functionality and provide an outlook regarding quantum computing. In SSI, the personally identifiable data is stored in wallets. The authors examine different wallet concepts (edge, cloud, hybrid) and compare a list of multiple SSI wallet vendors.

The analysis also investigates three different types of mechanisms to revoke VCs. The second last chapter is about the chain of trust, and it sets certificate chaining (e.g., X.509) into perspective with credential chaining. The section further compares PKIs with the Web of Trust and decentralized key management systems(DKMS). The analysis ends with differentiation of blockchain vs. distributed ledger technologies vs. centralized approaches.

The analysis shows that SSI is still a novel paradigm with many concepts and issues around composability, integration, and interoperability. Unfortunately, there is no solution that fits for all use cases, but at the same time, many people, institutions, and corporations are involved in further developing and defining the ecosystem.

Table of Content

Decentralised Identifiers

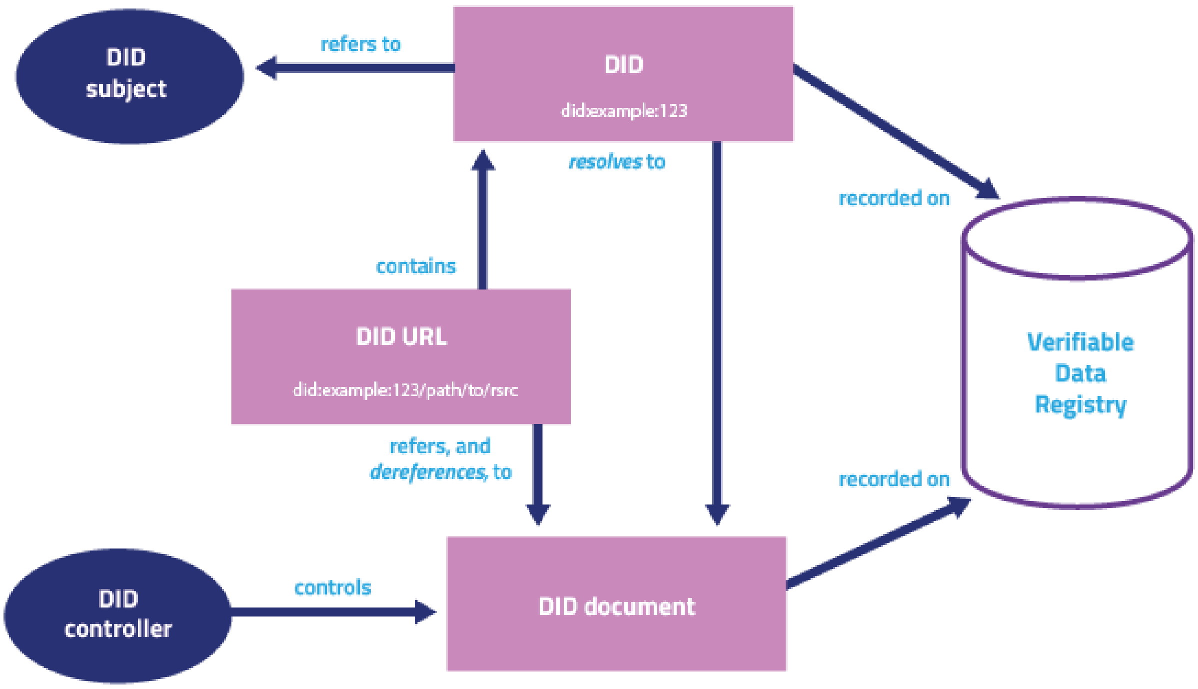
Digital identifiers help to identify different subjects like persons, objects, organisations, or abstract things. With the help of cryptographic algorithms, a subject can prove the ownership of information. There is no need for a centralised registry or party in the ecosystem to avoid a single point of failure or control. A universal resource identifier (URI) is the identifier of the resource, a similar approach when requesting resources via a URL from the world wide web.

The required resources to prove the ownership are stored in an associated DID document like the public keys or a service endpoint to get more information about the subject [DID-core]:

```
{
 "@context": [
  "https://www.w3.org/ns/did/v1",
  "https://w3id.org/security/suites/ed25519-2020/v1"
 ]
 "id": "did:example:123456789abcdefghi",
 "authentication": [{
  // used to authenticate as did
  "id": "did:example:123456789abcdefghi#keys-1",
  "type": "Ed25519VerificationKey2020",
  "controller": "did:example:123456789abcdefghi",
  "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
 }]
}
```

DID Methods

The DID method defines the kind how to resolve a DID document by getting information from a verifiable data registry. Since the DID core specification is written to support multiple verifiable data registries, the DID method is the required connector between the data registry and the requested DID document.



Source: https://www.w3.org/TR/did-core/#architecture-overview

The W3C specification allows the implementation of multiple methods to deal with DIDs. The only requirement is a unique identifier; the rest is optional or can be expanded by new attributes. When writing a new method, the author needs to define the basic operations to create, read, update, and delete a DID. Not all methods have all operations, like a DID can be designed to be immutable, so no update or delete operation is required and therefore not implemented.

To query one of the DID documents, the client must contact a verifiable data registry. In some rare cases, this is not necessary; when using the did:key method, all required information is already encoded in the identifier.

Resolving DID documents

Today there are over 180 different DID methods out there for generating a DID document. [DID-SPEC-REG]
Since DIDs should be decentralised and distributed to remove the single point of control, the implementation of the verifiable data registry has no technical regulations like allowed transport protocols (HTTP, WebSocket, ZeroMQ, gRPC, etc.) or the kind of API (REST, GraphQL, etc.).

To simplify the handling of DID documents, the open source project DID resolver[UNI-RES] tries to define a generalised way to query DID documents. A service (e.g. like a docker image including a REST API) must be implemented to publish a resolver for a DID method. This allows the authors to use any programming language inside the docker container that does not affect the usage of the container. It can easily be deployed on a server and used by other services, e.g., in a microservice architecture. The downside of this approach is that the resolver cannot be run on all edge devices like smartphones since they do not support docker.

So the outsourcing of the DID document assembling process to another service reduces the complexity of the agent side needing the resources but also moves a lot of trust to the resolver. The trust that is generated by, e.g., querying multiple nodes from a blockchain network or validating multiple signatures cannot be passed to the agent since it cannot validate it by reproducing the steps. If it would do, there is no need to outsource the assembling and validation logic to another service. The person controlling the universal resolver can decide which information will be sent in response. Depending on the DID method, it is possible to insert another public key or service endpoint to perform a man-in-the-middle attack. There are different approaches to guarantee the integrity of the results like with a state proof, where the end receiver can validate the received information. But this isn't a standardised way so the trust and security level are not equal between the DID methods and sometimes not inside the same DID method when using different resolvers.

The type of the request can be grouped into three different approaches:

Standalone: did:key [DID-KEY] and did:peer [DID-PEER] work without any external requirements. This means that there is no source of truth outsourced to another system. The identifier of the DID is bound to the public key that is derived from the private key. So the owner of the private key can prove that he/she is the owner of the DID. This simplicity has the downside that updating or deletion of the DID is not possible since they are immutable. Did:key is used for natural persons since no information must be published by a provider with the risk of violating the General Data Protection Regulation (GDPR). Did:peer is used for direct communication like credential exchanges but not for issuers since the key handling is too limited. The feature of a self-attested identifier and the sovereign approach to share the own identifier on demand comes with the tradeoff that there is no discovery service to look up the public key of someone to start an encrypted message channel.

Published: Methods like did:web [DID-WEB] or did:keri [DID-KERI] are mutable methods that are allowed to be updated or deleted. This allows the usage by issuers because a compromised key can be revoked. But since the DID document has to be published publicly, it is not designed to be used by holders that are natural persons because of GDPR. In the case of did:web the single point of failure is given when the website is down or not reachable so nobody is able to fetch the document. Other approaches like did:keri allow redundancy, by providing multiple witnesses to fetch the data. One of the witnesses could be a DLT based system, or a centralised web server.

DLT: Using a blockchain or DLT as a distributed and decentralized network allows the transaction storage to be used as a verifiable data registry. Methods like did:indy [DID-INDY], did:trust [DID-TRUST] or did:ebsi [DID-EBSI] are storing the changes of a DID document or the whole document inside a transaction that is stored on multiple servers. These approaches follow a hierarchical order because they are managed in a permissioned system. There are also ways to use permissionless systems and then manage the DIDs, for example when deploying a smart contract on the public Ethereum network where only authorized entities are allowed to add new entries to the contract.

DID Resources

To verify a credential, you need different type of resources, but right now they are queried in different format:

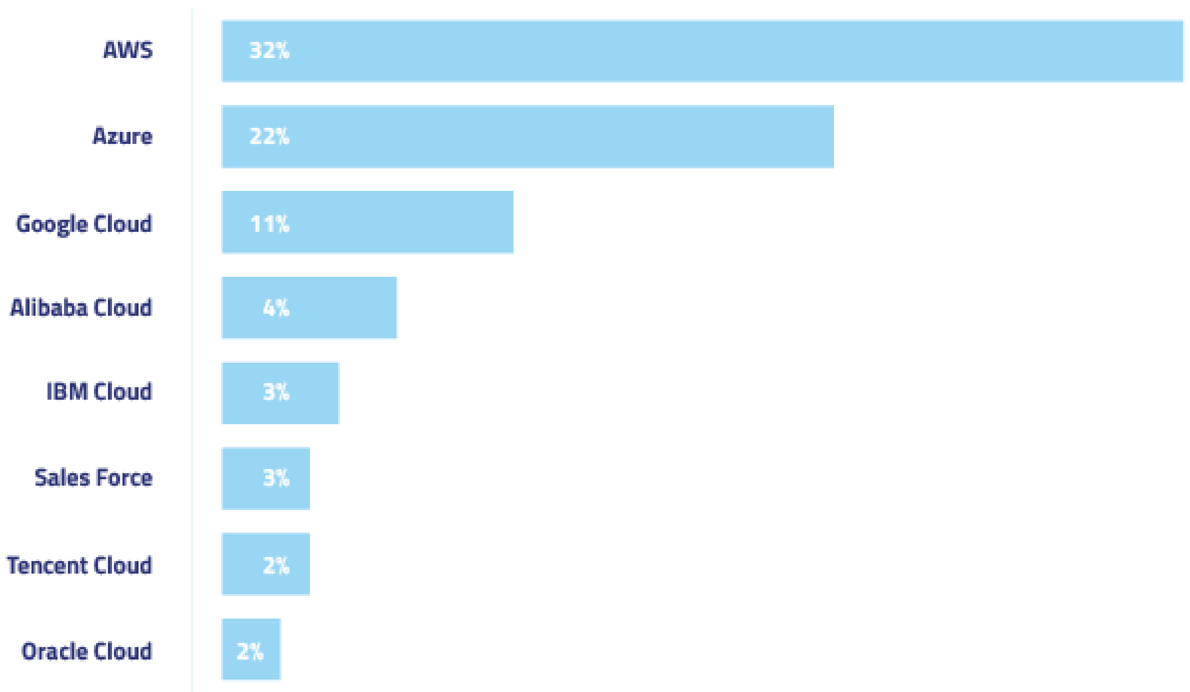
- a DID document giving information about the public key of the issuer to verify the signature
- a schema for structured data, hosted on schema.org as a JSON-LD object
- a revocation list published on a blockchain

These different types require the support of multiple file formats. The usage of a DID could be possible since all these resources are addressed by a uniform resource identifier. The main DID core specification only focuses on how the ownership of a DID can be validated. But the definition of the core properties only requires the unique identifier in a DID document:

Property	Required?	Value constraints
id	yes	A string that conforms to the rules in 3.1 DID Syntax.
alsoKnownAs	no	A set of strings that conform to the rules of [RFC3986] for URIs.
controller	no	A string or a set of strings that conform to the rules in 3.1 DID Syntax.
verificationMethod	no	A set of Verification Method maps that conform to the rules in Verification Method properties.
authentication	no	A set of either Verification Method maps that conform to the rules in Verification Method properties) or strings that conform to the rules in 3.2 DID URL Syntax.
assertionMethod	no	
keyAgreement	no	
capabilityInvocation	no	
capabilityDelegation	no	
service	no	A set of Service Endpoint maps that conform to the rules in Service properties.

Addressing the resource via DID and not via URL allows for removing the single point of failure. Right now, the top three have the absolute majority when proving cloud infrastructure and the resources hosted on them:

The world's cloud infrastructure



Source: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/

There are more resources that must be queried, validating verifiable credentials from the self-sovereign identity view [CHEQD-RES]:

- **Schemas**
- **Revocation status list**
- **Visual representation of a credential**

All this information can also be stored inside a DID document and queried by a unique identifier.

The versioning aspect is already covered by the core specification that allows querying information by a timestamp (versionTime) or a versionId:

```
did:example:123?versionTime=2021-05-10T17:00:00Z
did:example:123?versionId=4
```

Most of the resources can be updated so it must be possible for a verifier to get the version of a DID document when it was used during the issuance process. [INDY-CC-2022-04-26].

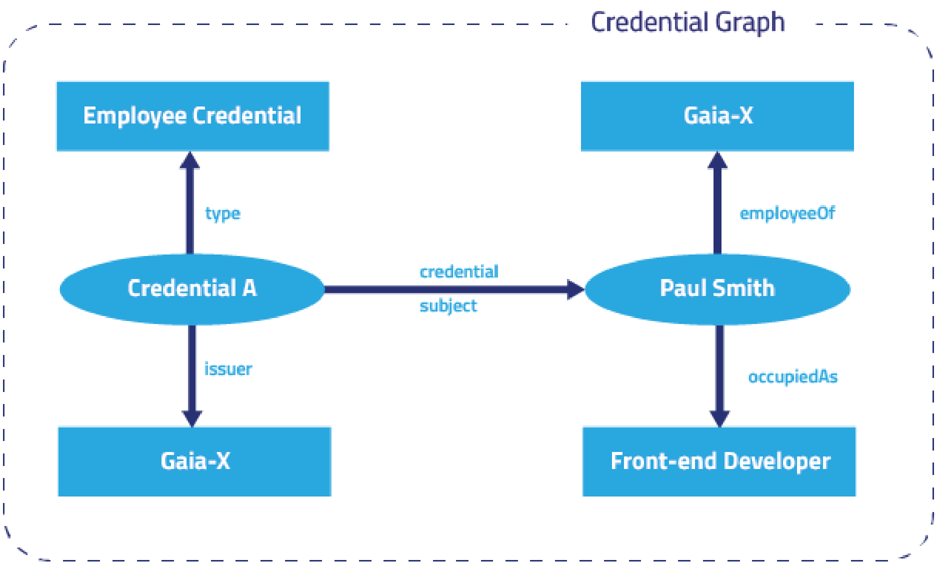
Critic

Some big players like Google, Apple, and Mozilla voted against the W3C recommendation status. They criticized that „DID-core is only useful with the use of ‚DID methods', which need their own specifications" [DID-Rec-Dec]. Right now, there are a lot of DID methods in the DID specification registries that are not compliant with the latest requirements anymore. E.g., the section for security and privacy considerations was defined after the first methods got accepted but were not updated to be compliant again.

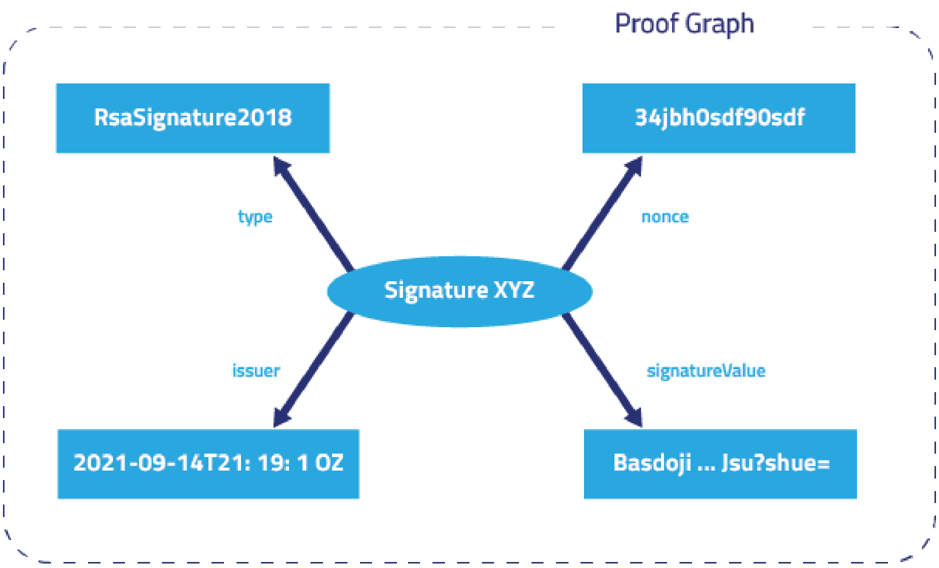
Verifiable Credentials

A Verifiable Credential (VC) is a set of one or more claims that are tamper resistant. A VC contains the proof(s) of one or more claims and might also include an identifier and metadata to describe properties of the VC, e.g., issuer, expiry date, and time.

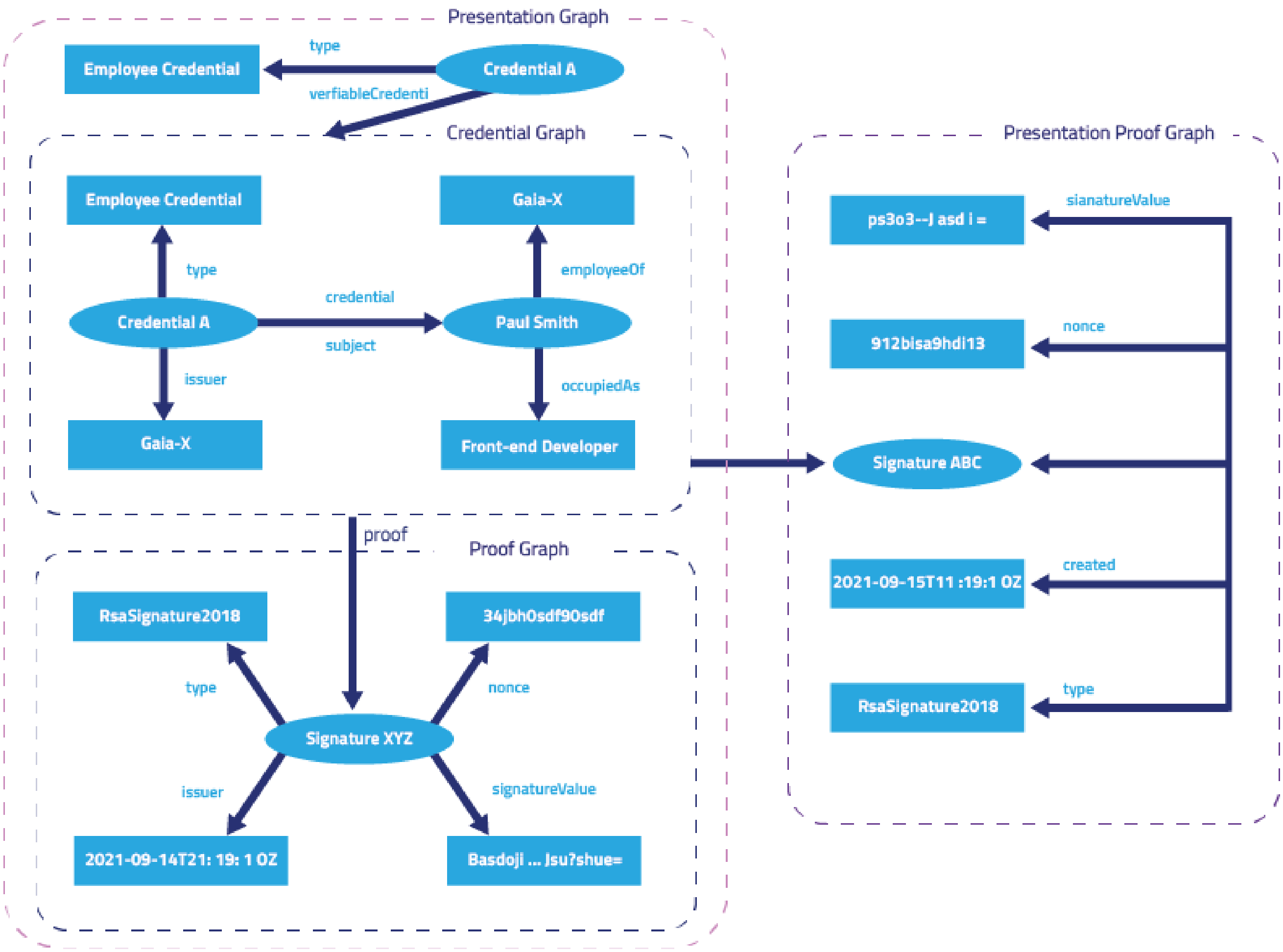
A VC can be separated into two sections:
The **Credential Graph** contains information about the attributes and dependencies of the credentials, as shown in the figure below:



The **Proof Graph** expresses the digital proof, which is usually a digital signature:



When verifying a VC, a presentation proof of a VC needs to be created. The **Presentation Graph** contains the Credential Graph and the Credential Proof Graph plus its presentation. And the **Presentation Proof Graph**, the proof of the presentation, is structured as the Credential Proof Graph but with the Presentation Graph as the subject.

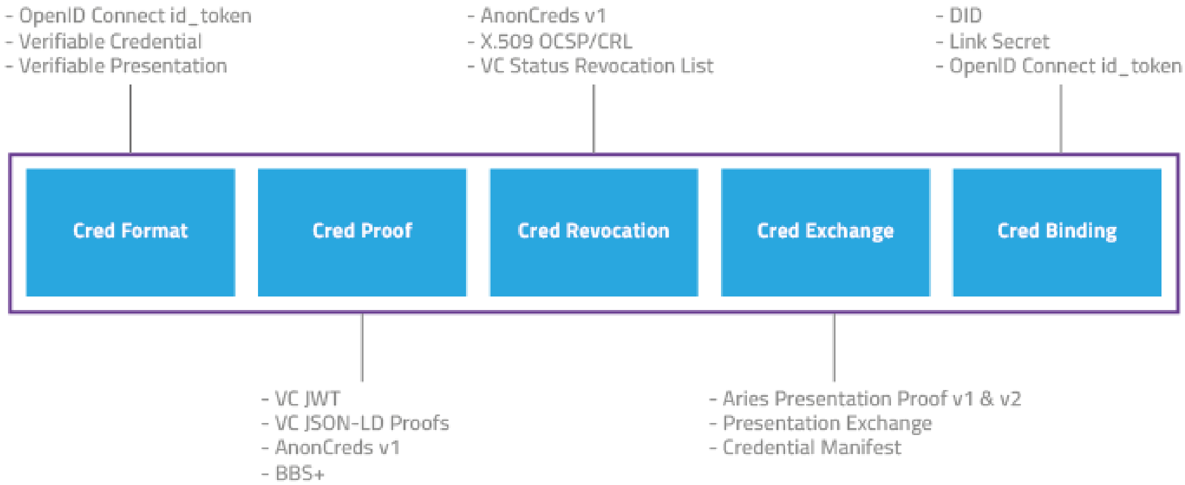


The Credential Layer

The Credential Layer is one of the main layers in SSI. It can be separated into the following five sections as proposed by Hakan Yildiz from TU Berlin in "Layers of SSI Interoperability" as part of his work in the DIF Interop WG [HAK-21] :

- **Credential Format describes the format of credentials that could either be an OpenID Connect id_token, or Verifiable Credential / Presentation according to the W3C standard.**
- **Credential Proof describes the digital proof or signature format.**
- **Credential Revocation describes the revocation implementation of the credential.**
- **Credential Exchange describes concepts for submitting proofs from a Holder to a Verifier.**
- **Credential Binding describes the linkage/binding of the holder or subject to a particular claim or a set of claims. This is required to determine the legitimacy of the presented proofs.**

The following graphic lists the different entities of the credential layer and assigns appropriate examples according to [HAK-21].



The many implementation possibilities for each section in the credential layer lead to the following issues:

- There are four different Signature approaches for VCs
    - JWT Signatures
    - LD Signatures
    - CL Signatures
    - BBS+ Signatures
- The Credential Revocation type depends on the Credential Proof type to be interoperable with each other
- The same applies to Cred Exchange and Cred Binding
- Creating one presentation proof of two credentials with different signatures (e.g., AnonCreds with CL-signatures and JSON-LD with BBS+)

Flavors of Credentials

In the following chapter, five different flavors/approaches of VCs are described. Each of the different implementations is categorized into the following five properties:

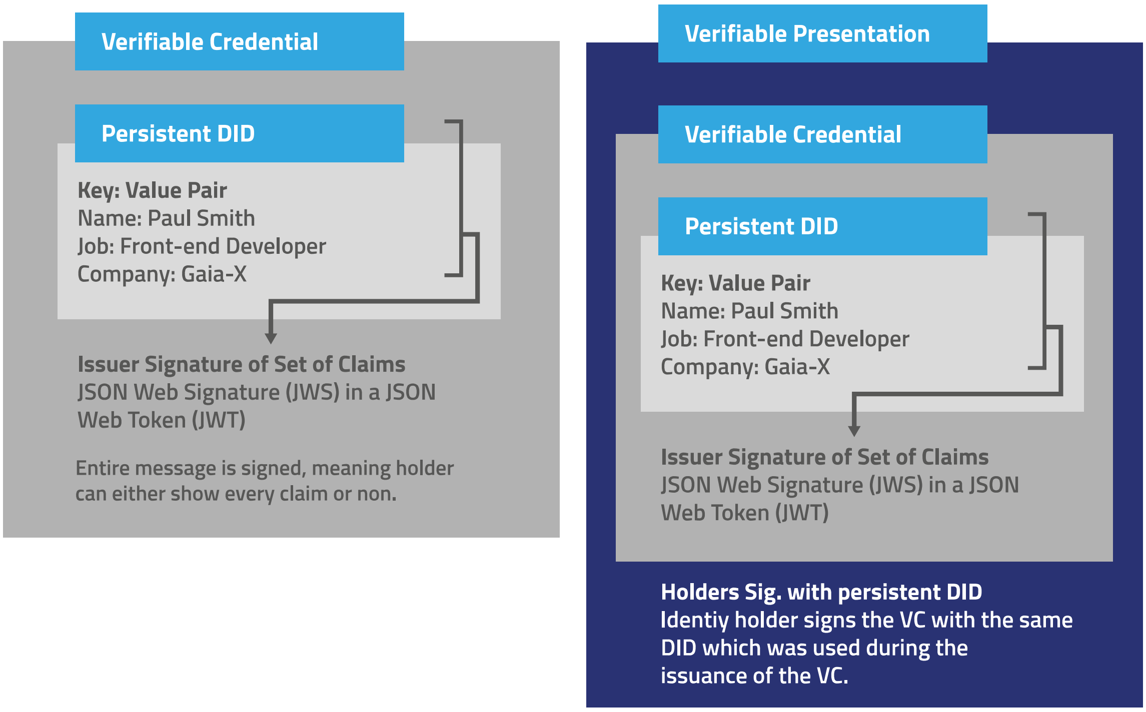
- **Privacy-Preserving** can be provided, when the credential implementation supports the selective disclosure of attributes and/or zero-knowledge proof capabilities.
- **Selective Disclosure** means one can hide some of the attributes of a credential, selectively revealing them for presentation as required.
- **Zero-Knowledge Proofs** are a novel approach in cryptography that lets one prove things are true without revealing the information.
- **Need to reveal persistent identifiers** is related to the binding of credentials. It describes the correlation risk when generating a proof request that contains either a persistent DID or a link secret that is only known to the holder and never needs to be revealed.
- **Semantic disambiguation** as provided by JSON-LD is established by referencing an RDF-defined schema via web-based open-data registries.

JSON JWT

The credential format JSON JWT is based on JSON Web Tokens [RFC-7519] that are secured by JSON Web Signatures [RFC-7515]. The issuers sign the entire message and attach the signature to the credential. Thereby, the holder can either reveal every claim or no claim of the credential. A persistent DID establishes the holder binding of the credential. When the holder generates presentation proof of the credentials, the identity holder signs the credential with the same DID used during the credential issuance, which leads to correlation risks.

Properties:

- Privacy-Preserving: No
- Selective Disclosure: No
- Zero-Knowledge Proof: No
- Need to reveal persistent identifiers: Yes
- Semantic disambiguation: No



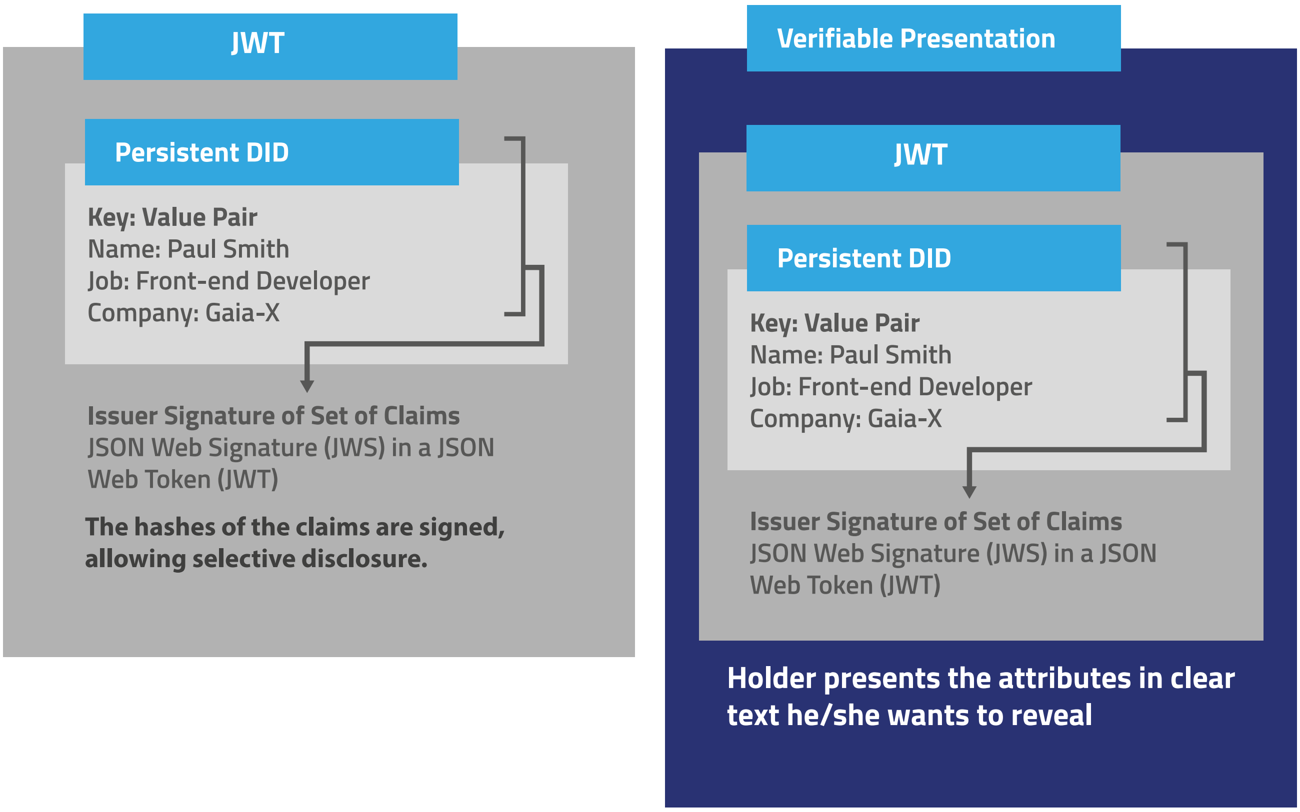
SD-JWT

Another approach for representing credentials is JSON Web Token (JWT). They are primarily used for authentication, authorisation, and information exchange in protocols like OAuth or OpenID Connect. JWTs and VCs are both containers for claims and their signatures.

The Selective Disclosure Json Web Token (SD-JWT) is very similar to the JSON JWT implementation. The existing approach was appended with a new functionality to present only a view attributes of the signed credential.[SD-JWT]

Properties:

- Privacy-Preserving: No
- Selective Disclosure: Yes
- Zero-Knowledge Proof: No
- Need to reveal persistent identifiers: Yes
- Semantic disambiguation: Yes

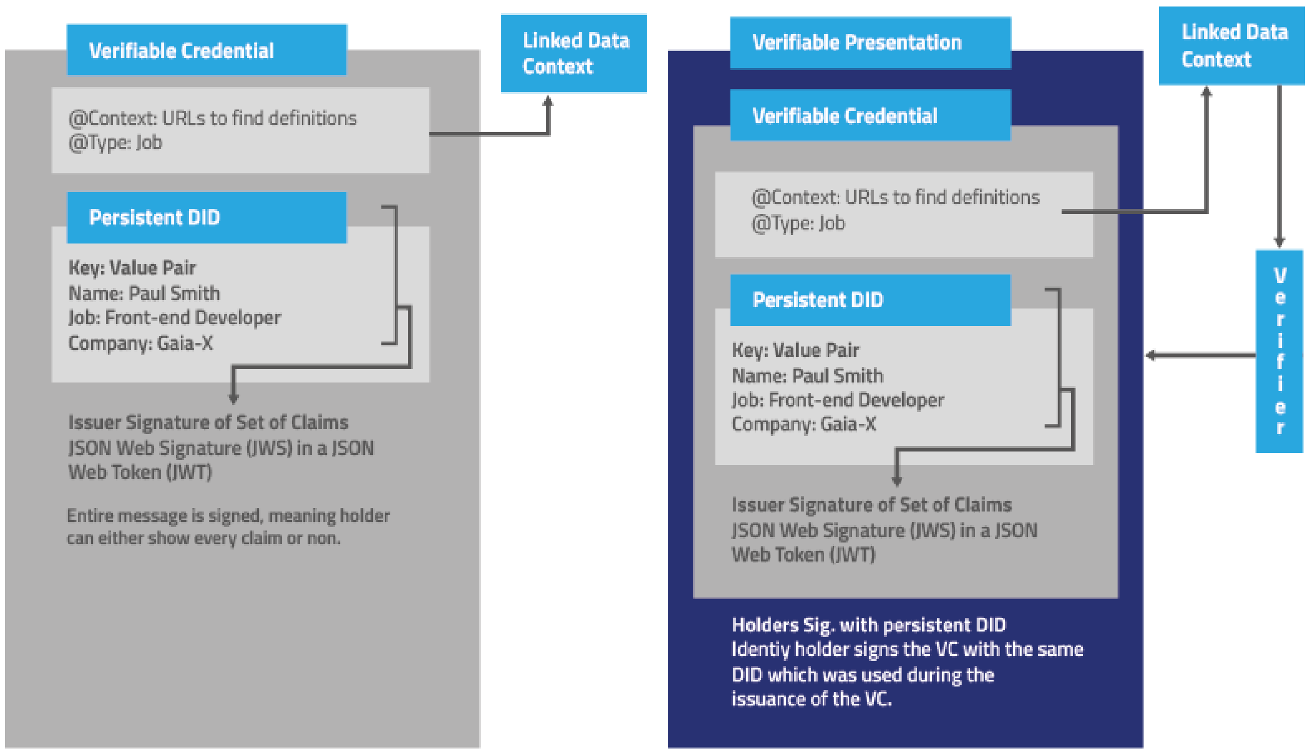


JSON-LD with LD Signatures

The credential format JSON-LD with LD Signature is based on JSON for Linking Data [JSON-LD] secured with Linked Data Signatures [LD-SIGNATURES]. The JSON-LD standard provides additional mappings from JSON to an RDF model and allows for semantic disambiguation in the context of verifiable credentials. The credential is extended by the context and type attributes compared to the JSON JWT format. The issuer signs the entire message, meaning the holder can either disclose every claim or none. A persistent DID establishes the holder binding of the credential. When the holder generates presentation proof of the credentials, the identity holder signs the credential with the same DID used during the credential issuance, which leads to correlation risks.

Properties:

- Privacy-Preserving: No
- Selective Disclosure: No
- Zero-Knowledge Proof: No
- Need to reveal persistent identifiers: Yes
- Semantic disambiguation: Yes

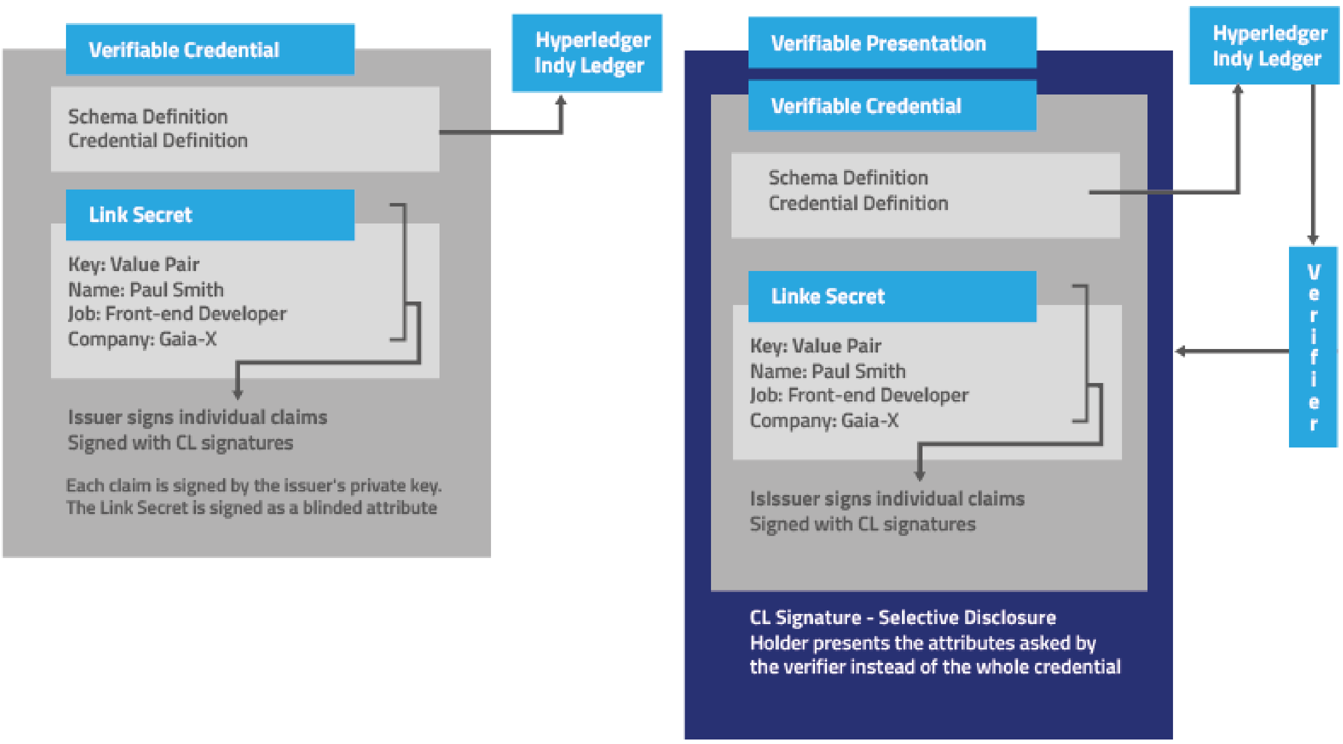


AnonCreds with CL Signatures

The concept AnonCreds (Anonymous Credentials) evolved from the Hyperledger Indy community in 2017 and is now in the process of becoming standardized [ANON-SPEC]. AnonCreds focuses on privacy and is based on Camenisch-Lysyanskaya Zero-Knowledge Proofs [CL-SIGNATURES]. AnonCreds credentials link to a schema and a credential definition publicly stored on a verifiable data registry like a Hyperledger Indy ledger. The issuer signs every claim individually, allowing for selective disclosure of the attributes. Using ZKP enhances AnonCreds with ZKP capabilities like proofing that the age is over 18 without revealing any additional information about the holder's date of birth. The link secret ensures holder binding and is signed as a blinded attribute removing the risk of correlation.

Properties:

- Privacy-Preserving: Yes
- Selective Disclosure: Yes
- Zero-Knowledge Proof: Yes
- Need to reveal persistent identifiers: No
- Semantic disambiguation: No

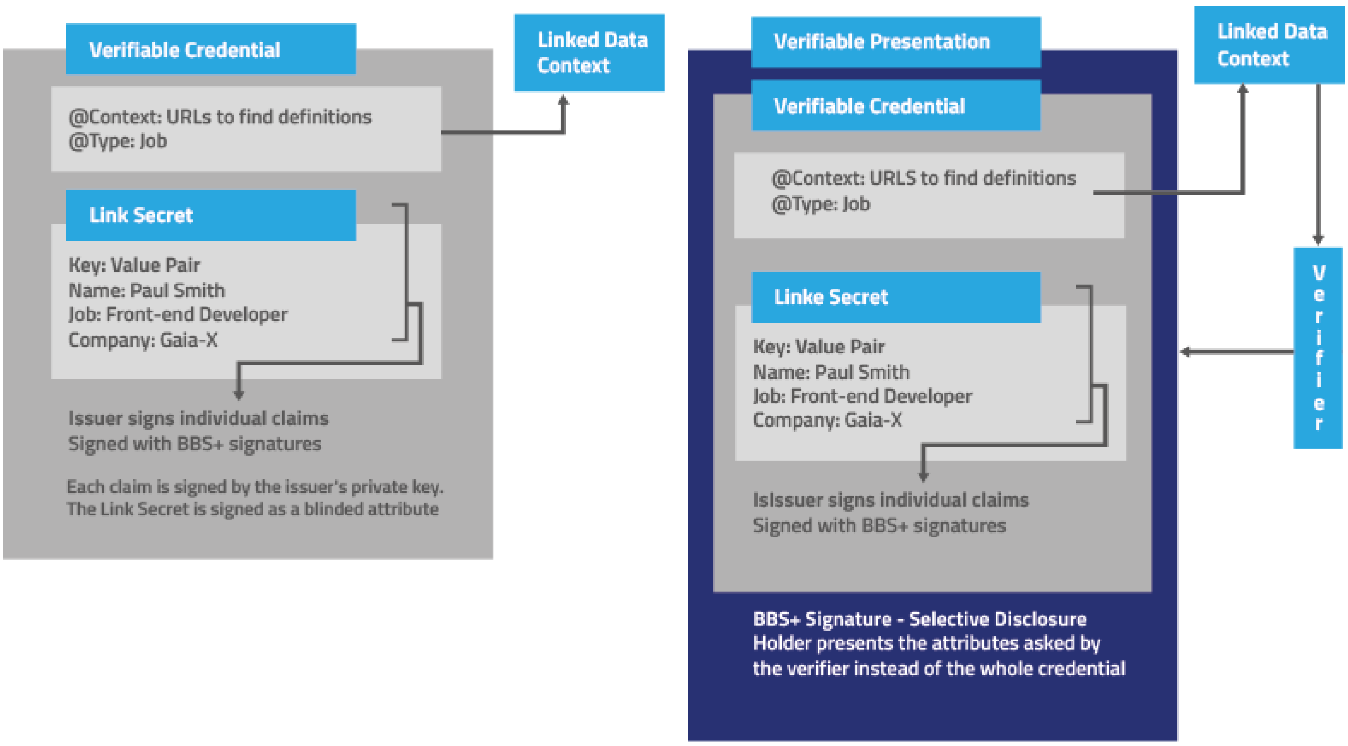


JSON-LD ZKP with BBS+ Signatures

The credential flavor JSON-LD ZKP with BBS+ 2020 Signatures [BBS+-Signatures] combines JSON for Linking Data [JSON-LD] with the ZKP cryptography of BBS+. It combines the concepts of JSON-LD with LD-Signatures (semantic disambiguation by adding the context and type attributes to the credential) with AnonCreds (credential binding via blinded linked secret, signing each claim individually, and ZKP capabilities). Compared to CL signatures, BBS+ are smaller and faster.

Properties:

- Privacy-Preserving: Yes
- Selective Disclosure: Yes
- Zero-Knowledge Proof: Yes
- Need to reveal persistent identifiers: No
- Semantic disambiguation: Yes



Mobile Driver's License

The Mobile Driver's License (mDL) standard is published under the ISO/IEC 18013-5 standard and describes an ISO-compliant mobile driving license. mDL is intends to: [INABTA-MDL]

- enable verifiers not affiliated with or associated with the issuing authority to gain access to and authenticate the information
- allow the holder of the driving license to decide what information to release to a verifier
- include the ability to update information frequently and authenticate information at a high confidence level.

In contrast to SSI, mDL doesn't cover how the issuer can share verifiable information with the holder and how the issuer can revoke verifiable information shared with the holder. However, mDL supports selective disclosure and allows the verifier to authenticate and interpret information.

The table below compares the mDL standard with SSI [PROC-mDL]. Especially the correlation risks would be needed to be addressed when dealing with personally identifiable data in the EU. In mDL, the driver's license is bound to an identifier that is revealed every time they present the credential. This puts the holder or subject at the risk of being tracked across services. Also, the issuer is involved in every verification, making it a single point of failure and providing complete visibility over all services that an mDL may use to authenticate.

	ISO/IEC 18013-5	SSI
Advantages	▪ Well specified protocols for device engagement and data retrieval ▪ Based on mature technologies which can be deployed at scale	▪ Non-correlating credentials (CL-Signatures & BBS+) ▪ Large open-source community
Disadvantages	▪ Correlating credentials ▪ Reliance on issuer infrastructure during proof verification ▪ Proprietary ecosystem of solutions	▪ Still early stages of developments but quickly evolving ▪ Openness of standards has led to divergent implementations

Comparison and Recap

The following table provides an overview of the capabilities of the credential flavors explained in this chapter.

Credential Type	JSON-JWT	SD-JWT	JSON-LD with LD Signature	Anoncreds-with CL-Signature	JSON-LD with BBS+ Signature	mDL
Privacy Preservin	No	No	No	Yes	Yes	No
Selective Disclosur	No	Yes	No	Yes	Yes	Yes
ZKP	No	No	No	Yes	Yes	No
Need of Persis-tent Identifier	Yes	Yes	Yes	No	No	Yes (device binding)
Semantic Disambiguation	No	No	Yes	No	Yes	Yes

Schemas

When dealing with credentials the content is machine-readable and designed to be processed automatically by a system without any interaction of a human being. But to understand the attributes or to perform special cryptographic algorithms schemas are required to structure the data. File formats like PDFs are widely used as a credential since it is very similar to classic paper handling. They can be protected with digital signatures to guarantee their integrity, but since they have no schema, it is very difficult to process them.

AnonCreds Schemas

To perform the zero knowledge-proofs with AnonCreds with the help of CL-signatures, the credential must be structured to calculate the signature. A schema includes a list of attributes that are used to build a credential [ANON-SPEC]:

```
{
  "attr_names": [
    "birthlocation",
    "facephoto",
    "expiry_date",
    "citizenship",
    "name",
    "birthdate",
    "firstname",
    "uuid"
  ],
  "name": "BasicIdentity",
  "version": "1.0.0"
}
```

Since the attr_names value is a list of strings, the definition of the schema is very limited: The values are all treated like strings and not like booleans or numbers. The focus when defining the schema was based on the requirements to perform the signature algorithm and not on a high-value schema that describes the input.
Another problem is the missing feature of nested objects. They allow to use of the same name for a key when there are not in the same object:

```
{
 "person": {
  "name": "Max"
 },
 "class": {
  "name": "Math"
 }
}
```

The structure must be flattened to be used with AnonCreds like:

```
{
 "person_name": "Max",
 "class_name": "Math"
}
```


To solve both problems the RFC 0119 introduced rich schema objects to use the power of JSON-LD [INDY-RFC-0119]. This milestone was moved to version two of AnonCreds and lost power since BBS+ with JSON-LD already covered some important requirements. [EVERNYM-BBS]

JSON-Schema

To validate JSON objects a JSON schema can be used to check for the correctness of the format, the syntax, the data types, and the structure. The following example shows the definition of an attribute. A description can help a human being to understand the meaning of the attribute.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/product.schema.json",
  "title": "Product",
  "description": "A product in the catalog",
  "type": "object"
}
```

This information is not directly stored inside the credential, instead they are referenced by a link. Some cryptographic algorithms like BBS+ signatures need structured data, so a definition of the schema is required to calculate the signatures.

JSON-LD

To reduce redundancy and support the definition of standards it is possible to use linked data to connect schemas. Without linked data, every schema must define all objects on its own. But when using links, a schema can point to already defined definitions like in the DID core specification:

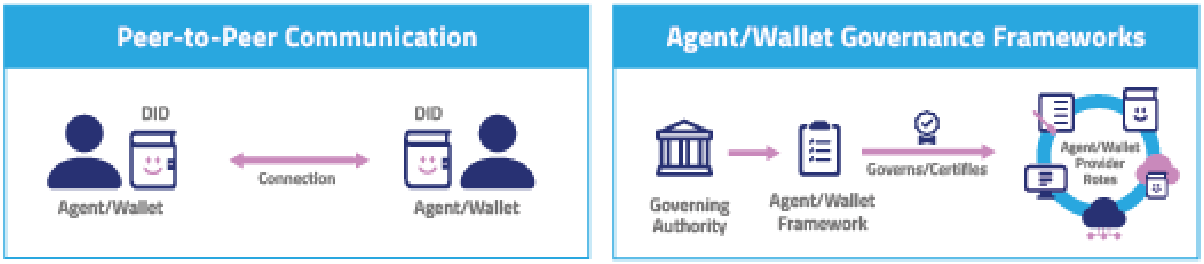
```
{
  "@context": {
    "@protected": true,
    "id": "@id",
    "type": "@type",

    "alsoKnownAs": {
      "@id": "https://www.w3.org/ns/activitystreams#alsoKnownAs",
      "@type": "@id"
    },
    "assertionMethod": {
      "@id": "https://w3id.org/security#assertionMethod",
      "@type": "@id",
      "@container": "@set"
    },
   ...
}
```

On the one hand, it reduces the size of a schema because of the references. On the other hand, it increases the number of different resources where some of which can be hosted by a third party. Caching can help to reduce the number of requests. But when one of the external resources or one of the sub-resources that must be loaded is not available and there is no alternative way to load it, the whole validation cannot be done.

Transport Protocols

Layer 2 of the Trust over IP (ToIP) stack is for the private digital wallets and agents needed by individuals, organizations, and digital "things" (or the digital twins of non-digital things) to accept, store, and exchange digital credentials over a standard peer-to-peer protocol such as DIDComm. [TOIP-V2.0]

Layer 2



DIDComm Messaging

The purpose of DIDComm Messaging is to provide a secure, private communication methodology built atop the decentralized design of DIDs. [DIF-DIDComm]. It works on top of any transport: HTTP, Bluetooth, SMTP, raw sockets, and sneakernet, for example.

DIDComm Messaging tells how to: [HARD-DIDComm]

- Use a DID to sign and encrypt messages for one or more other DIDs, each with multiple devices having different keys
- Declare and use a DID endpoint with standard semantics
- Route a message through untreated intermediaries with high privacy
- Verify the sender of a message
- Use standard message headers, and declare custom ones
- Declare/handle the schema of a message
- Attach data to messages by value or by reference
- Sequence messages into a coherent thread, even with unreliable delivery
- Detect and report errors
- Discover features of other parties
- Build protocols out of these primitives

DIDComm Messaging is does not cover how to: [HARD-DIDComm]

- Create or use wallets
- Work with credentials
- Associate a DID with a human (or other) identity
- Bind a remote party to a biometric
- Move messages over a transport
- Choose DID methods or key types of blockchains
- Properly maintain relationships
- Synchronize state across multiple agents

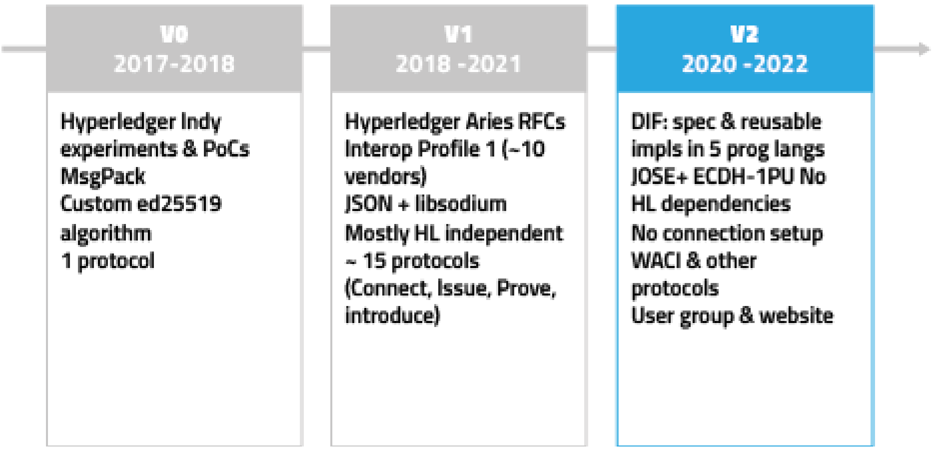
There are different libraries out there implementing the DIDComm v2 protocol [DIDCOM-LIBs].

DIDComm v1 vs. v2

- Formalization of methods used in V1
  - JWM based envelope
  - ECDH-1PU standardized form of AuthCrypt
- Both DID and key in each message
- Special Handling of Peer DIDs eliminated
- Message structure split between 'headers' and body.
- No AnonCrypt encryption method.
- No HL Indy dependencies anymore

DIDComm History

The first version of DIDComm was incubated in the Hyperledger Aries community and is referred to as Version v1. The DIDComm v2 spec, however, is more technology agnostic and incubated by the decentralized identity foundation (DIF). The following graphic provides an overview of the history of DIDComm messaging. DIDComm v2 is the current standard, but many projects still rely on v1.



OpenID for Verifiable Credentials (OID4VC)

The OpenID connect protocol is a widely adopted protocol used for federated identity with user-centric control. The privacy of this approach can be increased so the provider will not know with which service the end user is communicating. To follow this principle of self-sovereign identity, the OpenID for Verifiable Credential protocol was defined in the OpenID Foundation.

The core properties of OID4VC are:

- User-centric
- Easy to implement
- Universal: supports web, apps, and APIs (through integration with OAuth)
- Safety is systematically examined and scientifically proven (under certain circumstances)
- Interoperability through conformance testing [OID4VC-CERT]
- And OID4VC has a very active community

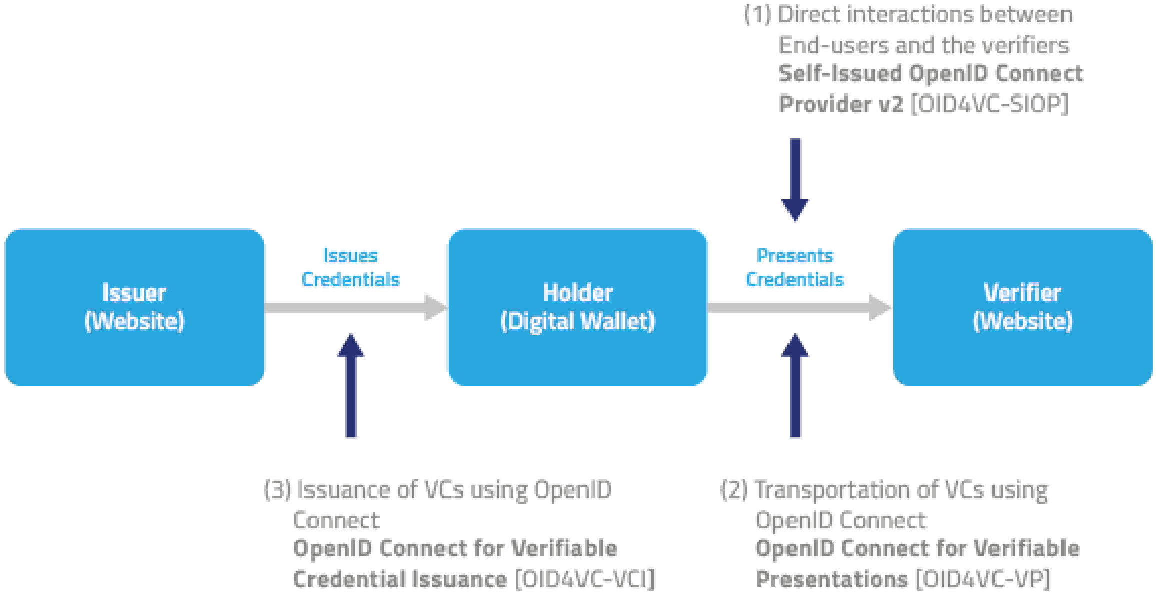
In total, OID4VC is working on three specifications that embed SSI in the OID4VC ecosystem.

OID4VC SSI Specs:

- Self-Issued OpenID Provider v2 [OID4VC-SIOP]
  - extends OpenID Connect with the concept of a Self-Issued OpenID Provider (Self-Issued OP), an OP controlled by the End-User.
- OpenID for Verifiable Presentations [OID4VC-VP]
  - defines a mechanism on top of OAuth 2.0 to allow presentation of claims in the form of verifiable credentials as part of the protocol flow
- OpenID for Verifiable Credential Issuance [OID4VC-VCI]
  - defines an API and corresponding OAuth-based authorization mechanisms for issuance of verifiable credentials

OID4VC connect is gaining more and more traction as a communication channel over DIDComm v2 and has been selected by the European Commission. This choice could be because OID4VC focuses on bridging Web2 and more federated identity models into SSI through the widely adapted and intensively tested OpenID Connect protocol.
Since there are already millions of OID4VC relying parties (RP) which may be able to access and issue VCs through the OID4VC SSI approach, this bridge could lead to a larger adoption vector. [CHEQD-SSI-TRENDS]

The following graphic provides an overview of the above-mentioned OID4VC SSI specs in the flow of issuing and presenting a credential.



 |

Cryptographic agility

The term cryptographic agility describes the ability to support multiple algorithms in parallel or in the future when defining a protocol or implementing software. Both aspects are relevant for the SSI world:

Security vs. Usability: there is not the perfect algorithm out there that is able to fulfill all requirements. It depends on the use case, for example when a human being is involved with sensitive data the privacy aspect is very important. On the other hand, dealing with internet of things devices they have limited resources so dealing with resource-intensive tasks can be impossible or it needs a huge amount of time.

Increased performance: Over the last decades, the power of computers has increased. This made some algorithms not secure anymore because brute-forcing all possible combinations was now possible in a shorter period of time or someone found a way to reduce the number of required steps to find the solution. To counter this, a more robust algorithm must replace the current one.

RSA, ECC, BBS+, CL

When exploring the realm of verifiable credentials, the focus often gravitates towards cryptographic methods that offer robust security, efficiency, and, crucially, privacy preservation. RSA, with its widespread adoption, sets a foundational benchmark, given its established presence in the digital trust ecosystem. ECC, on the other hand, stands out for its efficiency, especially in constrained environments, making it a prime choice for scalable systems. Shifting to more privacy-centric needs, the CL and BBS+ signature schemes are pivotal. Both inherently support zero-knowledge proofs, a cornerstone for ensuring user data remains undisclosed during verifications. In essence, these four algorithms collectively encapsulate the versatility and depth required for the multifaceted challenges posed by verifiable credentials.

	RSA	Elliptic Curve Cryptography (EEC Ell	Camenisch-Lysyanskaya (CL)	Boneh-Boyen-Short (BBS+)
Nature	Asymmetric encryption and digital signature scheme	Asymmetric encryption and digital signature scheme	Privacy-preserving signature scheme	Privacy-preserving signature scheme
Foundation	Relies on the difficulty of factoring large composite numbers	Uses properties of elliptic curves over finite fields	Allows proof of a signature on a committed value without revealing the value	Bilinear pairings offering strong privacy guarantees
Key Size	Typically long (1024, 2048, 3072 bits)	Shorter for equivalent security (e.g., 256-bit ECC ≈ 3072-bit RSA)	Varies but generally requires larger parameters than standard RSA for equivalent security	Typically efficient, leveraging the properties of pairings
Usage	Widely adopted for secure data transmission, SSL/TLS, and digital signatures	Growing in modern security protocols, mobile devices, and cryptocurrency	Anonymous credential systems and privacy-preserving protocols	Modern anonymous credential schemes and zero-knowledge proofs
Computational Efficiency	Can be intensive, especially with long key lengths	More efficient than RSA for the same security level	Depends on parameter choice but can be intensive.	Tends to be more efficient than CL, especially with optimized pairings
Mathematical Basis	Modular arithmetic, particularly exponentiation	Elliptic curves and algebraic structures	Typically built on RSA assumptions	Bilinear pairings over elliptic curves
Standardisation & Adoption	Standardized by multiple organizations, including NIST. RSA has been universally adopted in many digital systems, SSL/TLS protocols, and is supported by numerous software libraries.	Standardized by NIST and other organizations. ECC is increasingly adopted in IoT devices, modern communication protocols, and blockchain technologies, given its efficiency.	While not as universally standardized as RSA or ECC, CL signatures are recognized in the realm of privacy-enhancing technologies. They've been integrated into certain niche systems and academic projects focusing on user privacy.	BBS+ is recognized in the privacy-focused cryptographic community. While not as standardized as RSA or ECC in broader applications, it's increasingly adopted in modern systems requiring advanced privacy features, especially in the context of decentralized identities and blockchain systems.
Zero Knowledge Features	Traditional RSA doesn't directly support ZKPs. However, various cryptographic protocols built on RSA can be designed to provide zero-knowledge properties.	ECC forms the basis for many ZKP schemes, particularly in the realm of blockchain and privacy-preserving protocols. For instance, zk-SNARKs often utilize ECC.	The Camenisch-Lysyanskaya signature scheme inherently supports zero-knowledge proofs, allowing users to prove possession of a signature without revealing the actual data.	BBS+ signatures inherently allow for ZKPs, making them suitable for privacy-centric applications like anonymous credentials and certain blockchain protocols.

How to deal with Quantum Computing?

The power of a quantum computer is the huge amount of computing capacity to brute force the private key of a given public key. So, it is important to find algorithms that are efficient to use when the secret is known but very hard when it is not known. Right now, there is no proof that the algorithms above are already broken when using a quantum computer. But depending on the use case, some credentials need to be valid for over 70 years, and during that time, it is very probable that a machine can break the signature. Therefore, the term crypto agility is an important aspect that allows you to replace the current algorithm with a more up to date one that is more robust.

Algorithms like RS256 have a low capability in features like zero-knowledge proofs out of the box and low requirements. When using an RS256 with a JSON web signature for issuing verifiable credentials and the signature algorithm becomes insecure, it is easy to replace it. There is no complex requirement when validating the signature that was produced by the signing function.Other algorithms like BBS+ have special requirements for the new chosen algorithm like using paring firendly curves. It is not impossible to find a new algorithm that is quantum proofed and has the same benefits as the replaced one, but the chances are lower.

Recap

When choosing the correct signature algorithm, it depends a lot on the use case:

- When there are no attributes involved that should not be presented the selective disclosure capabilities are not required
- When the credential is only valid for a short period of time the security level of the signature can be relevant to it

Using different types of signature algorithms increases the amount of support for all stakeholders. All applications must support the algorithms, otherwise the credential cannot be validated on the local device.

SSI Wallets

“A wallet is portable. A wallet is worth safeguarding. Good wallets are organized so we can find things easily. A wallet has a physical location.” [ARIES-RFC-0050]

Introduction to SSI Wallets

"Wallet" has been a well-known term in Web3 for many years, but most of the time referenced with cryptocurrencies. In general, digital wallets store secrets. However, there are some key differences between SSI and cryptocurrency wallets:

- A SSI wallet can be compared with a physical wallet. It can hold more than a paper currency (or cryptocurrency), e.g., SSI wallets can store credentials directly in the wallet
- Crypto wallets don’t store the tokens/credentials in the wallet directly. They just manage the key of the owner and display the token/cryptocurrencies queried from the ledger.
- SSI wallets need to manage many more relationships (up to millions in the case of large organizations)

In SSI, users of wallets can be differentiated into three categories:

- Individual identity owners (e.g., German citizen that stores their digital ID in their wallet, no credential issuance)
- Institutional identity owners (e.g., a corporation that manages a wide range of credentials and relationships, issues & revokes credentials)
- Trusted hub (e.g., a public register that can be queried for business licenses, etc.)

Every SSI wallet should support the following functionalities as outlined in [ARIES-RFC-0050]:

- Speak protocols to exchange secure messages such as DIDComm and OID4VC
- Collect digital credentials and securely store them in your wallet
- Share verifiable proofs of your credentials
- Manage cryptographic keys
- Import/export your digital wallet to use with another wallet app provider.

Edge vs. Cloud Wallet

A wallet performs many jobs at the same time, as described in the section above. Most of these functions require the wallet to perform quite complex and heavy cryptographic operations.

Most smartphones nowadays provide enough computational resources to perform cryptographic operations, so most SSI Wallet vendors build mobile apps called edge wallets. The benefit of edge wallet is that it is a purer approach to empowering individuals to own, control, and manage their wallets and keys without relying on a third-party provider. In the case of an edge wallet, all credentials are only stored locally on the smartphone.
Cloud **wallets** offer an alternative to edge wallets. In the case of cloud wallets, the cryptographic operations are performed on a server in the cloud, which keeps the wallet implementation lean. Compared to the edge wallet, the holder of credentials that are managed in a cloud wallet can view the same wallet through a browser via a mobile app which is not possible with edge wallets. As the cloud wallet needs to always connect with a third-party server, it needs an internet connection for every operation. **Cloud wallets** are also known as custodial wallets.

A third approach can be described as a combination of edge and cloud wallets. It uses end-to-end encrypted cloud storage as the storage layer to synchronize data between mobile apps, desktop apps, and browser apps. The data is always encrypted in rest and transit during synchronizing between the apps and the cloud storage layer. The cryptographic operations, however, are always performed at the edge device. At the time of writing, none of the SSI wallet vendors listed in the next chapter are working on the hybrid wallet approach. It unites the strength of cloud and edge wallets. Sovereignty about the data and availability across multiple apps while maintaining offline capabilities.

For all implementations, the security of the storage layer can be increased by integrating a secure enclave that could either be hardware (HSM, TPM) or trusted execution environments (TEE) like Intel SGC, AMD SVE, or ARM Trustzone. When connected with a secure enclave, the secret never leaves the secure enclave, and access is managed by ACL. Thereby, secure enclaves can increase the security of wallets and may be interesting for some institutional identity owners.

Comparison of SSI Wallets

The following table lists 12 wallet solutions and differentiates between their type (edge or cloud), the supported networks (Hyperledger Indy (Sovrin, IDunion) or Ethereum networks (EBSI)), and the supported transport protocols (DIDComm v1/v2 or OID4VC for SSI).

Wallet	Vendor	Open-Source	Type	Network Support		Transport Protocol	
				HL Indy	Ethereum	DIDComm	OIDC
Connect.me	Evernym Inc.	no	Edge	yes	no	yes	no
esatus Wallet App	esatus AG	no	Edge	yes	no	yes	yes
GATACA Wallet	ataca España S.L.U.	?	Edge	no	yes	no info available	no info available
iGrant Wallet	LCubed AB	no	Edge	yes	yes	yes	no
Lissi Wallet	Lissi (main incu-bator GmbH)	no	Edge	yes	no	yes	no
Spherity Cloud Identity Wallet	Spherity GmbH	no	Cloud	yes	yes	yes	yes
Trinsic Wallet	Trinsic Tech-nologies	no	Edge	yes	no	yes	yes
walt.id	walt.id GmbH	yes	Cloud	no	yes	no	yes
PCM	GXFS (GAIA-X)	yes	Edge	yes	no	yes	partial
OCM	GXFS (GAIA-X)	yes	Cloud	yes	no	yes	partial
Vereign Wallet	Verign	yes	Edge	yes	no	yes	partial
Sphereon Wallet	Sphereon	yes	Edge	no	yes	yes	yes

In addition to the wallets listed in the table above, we want to mention TNO EASSI as a multi-technology SSI wallet gateway that enables organizations to issue and verify credentials on multiple SSI infrastructures, including esatus, Trinsic, Gataca, Walt.id, and others.

Revocation

Revocation Methods

There are three main revocation methods used in SSI, which differ in terms of security, scalability, and privacy. The next section describes the concepts, and the following chapter compares them.

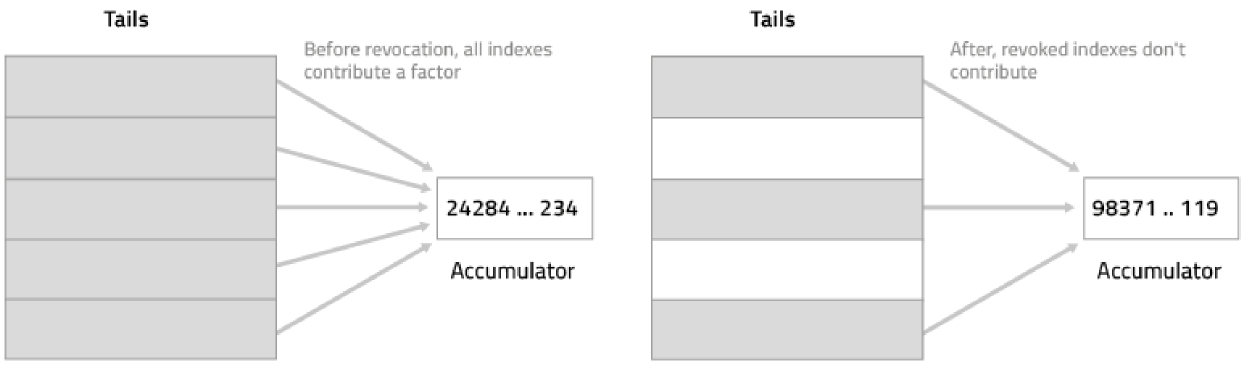
Cryptographic Accumulators

Cryptographic Accumulators are used in the credential revocation method that is described in "Indy Hype 0011: Credential Revocation" [INDY-HIPE-0011]. A Cryptographic Accumulator e can be calculated by multiplying a single factor such as a by the product of all the other factors (b * c * d). The product of all other factors is called a witness. In the example below, the accumulator e value is 210, and the value of the witness is 105.

a * b * c * d = e →2 * 3 * 5 * 7 = 210
a * (b * c * d) = e →2 * (105) = 210

In Hyperlegder Indy, for every credential definition based on a schema there and should have revocation capabilities, there also needs a Revocation Registry and an Accumulator (and witness delta) published to the ledger. The Revocation Registry references a tails file associated with an accumulator e and its factors. A tails file is a public file that needs to be accessible to every credential holder with reference to the credential definition. The tails file is a binary file containing an array of randomly generated factors that are very long numbers for an accumulator. Its content never changes.

To revoke a VC, issuers must update the delta witness on the ledger. The update just changes the answer of the math problem that either includes or excludes specific factors that correspond to VCs. Removing a factor from the math problem translates to revoking a credential. However, factors can be included again, which means reverting the revoked status.
Since the issuer is publishing an update to the public ledger, there is no need to contact each credential holder separately. In the process of verifying the proof of non-revocation, the latest delta witness is queried from the public ledger.
The following graphic illustrates the process of removing factors and updating the accumulator.



When requesting a presenting proof of a credential, the holder must create a proof of non-revocation and thereby must show that they can derive the accumulator's value for their credential. They do so by multiplying the factor they know (assigned to them during the time of credential issuance) with the witness (delta).

To sum up based on [INDY-HIPE-0011]:

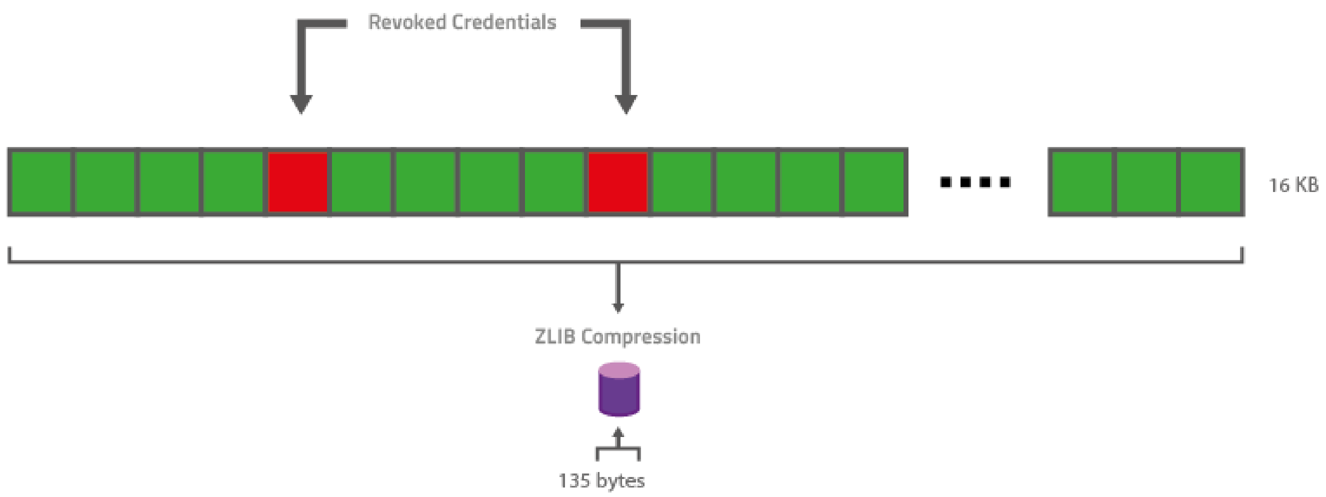
- Revocation based on cryptographic accumulators is privacy-preserving →They cannot be correlated by something like a credential ID or a tails index.
- Verification of proof of non-revocation is extremely easy and cheap.
- No tails files are needed by verifiers, and computation is trivial.
- Verifiers do not need to contact issuers or consult a revocation list to test revocation.

Bitstring

The Bitstring revocation method, as published in the draft specifications Status List 2021 [W3C-VC-STAT] and Revocation List 2020 [W3C-VC-REV], "describes a privacy-preserving, space-efficient, and high-performance mechanism for publishing status information such as suspension or revocation of Verifiable Credentials."

In the most basic explanation, the revocation status of VCs is expressed via a binary value in a long bitstring (bit list), where each bit represents the revocation status of one VC. If the binary value is 1 (one), the VC has been revoked; if the binary value is 0 (zero), the VC is still valid.

A significant advantage of the bitstring is its ability to compress the list with standard compression algorithms like GZIP because most list entries will still be 0 (zero). The default bitstring size is 16KB (131,072 entries). However, compressing the bitstring with GZIP or ZLIB when only a couple of VCs are revoked reduces the size to 135 bytes, as visualized in the figure below (source: [W3C-VC-STAT])



type (StatusList2021Entry), statusPurpose (revocation or suspension), statusListIndex (bit position of the status), and the statusListCredential (URL to a StatusList2021Credential).

```
{
 "@context": [
  "https://www.w3.org/2018/credentials/v1",
  "https://w3id.org/vc/status-list/2021/v1"
 ],
 "id": "https://example.com/credentials/23894672394",
 "type": ["VerifiableCredential"],
 "issuer": "did:example:12345",
 "issued": "2021-04-05T14:27:42Z",
 "credentialStatus": {
  "id": "https://example.com/credentials/status/3#94567"
  "type": "StatusList2021Entry",
  "statusPurpose": "revocation",
  "statusListIndex": "94567",
  "statusListCredential": "https://example.com/credentials/status/3"
 },
 "credentialSubject": {
  "id": "did:example:6789",
  "type": "Person"
 },
 "proof": { ... }
}
```


The **statusListCredential** attribute of the VC above references to a **StatusList2021Credential** that contains information about the general **type** (VerifiableCredential and StatusList2021Credential) of the VC and additional information listed in the **credentialSubject** as such as **type** (StatusList2021), **statusPurpose** (revocation or suspension), and the **encodedList** (GZIP-compressed, base-64 encoded bitstring values for the associated range of verifiable credential status values).


```
{
 "@context": [
  "https://www.w3.org/2018/credentials/v1",
  "https://w3id.org/vc/status-list/2021/v1"
 ],
 "id": "https://example.com/credentials/status/3",
 "type": ["VerifiableCredential", "StatusList2021Credential"],
 "issuer": "did:example:12345",
 "issued": "2021-04-05T14:27:40Z",
 "credentialSubject": {
  "id": "https://example.com/status/3#list",
  "type": "StatusList2021",
  "statusPurpose": "revocation",
  "encodedList": "H4sIAAAAAAAAA-3BMQEAAADCoPVPbQwfoAAAAAAAAAAAAAAAAAAAAIC3AYbSVKsAQAAA"
 },
 "proof": { ... }
}
```


Hash List

A hash list is a revocation list that contains the hash value of revoked credentials. Depending on the design decisions, there can be one or more hash lists per type of credential, issuer, network, etc. In theory, there could be one hash-list "to rule them all," which is not practical due to a single point of failure and the growing list size with every entry.

Implementing revocation via a hash list is not privacy preserving because everyone who is able to correlate a hash value to a specific VC can track whether the VC has been revoked or not.

The hash list is like a certificate revocation list (CRL) in classic PKI systems like X.509 certificate chaining.

Security vs. Scalability vs. Privacy

The usage of the correct revocation mechanism depends on the use case. When the privacy of the holder should be guaranteed at all costs, the accumulator approach seems to be the best choice. The high privacy comes with the price of high data storage and the size of the accumulator list must be chosen based on other conditions like the bandwidth to load it.

The Hash list approach scales the best when issuing credentials since the interaction is only required when revoking credentials. The other two approaches need to publish the list before even issuing a credential to reserve an index that could be used for revocation. But the tradeoff is the privacy approach could not only track the status of a credential but also brute force the revoked credentials without even knowing the correct one. In the middle of both approaches the Bitstring approach exists. It has a smaller size as the Accumulator approach is only traceable when the index is known to a verifier.

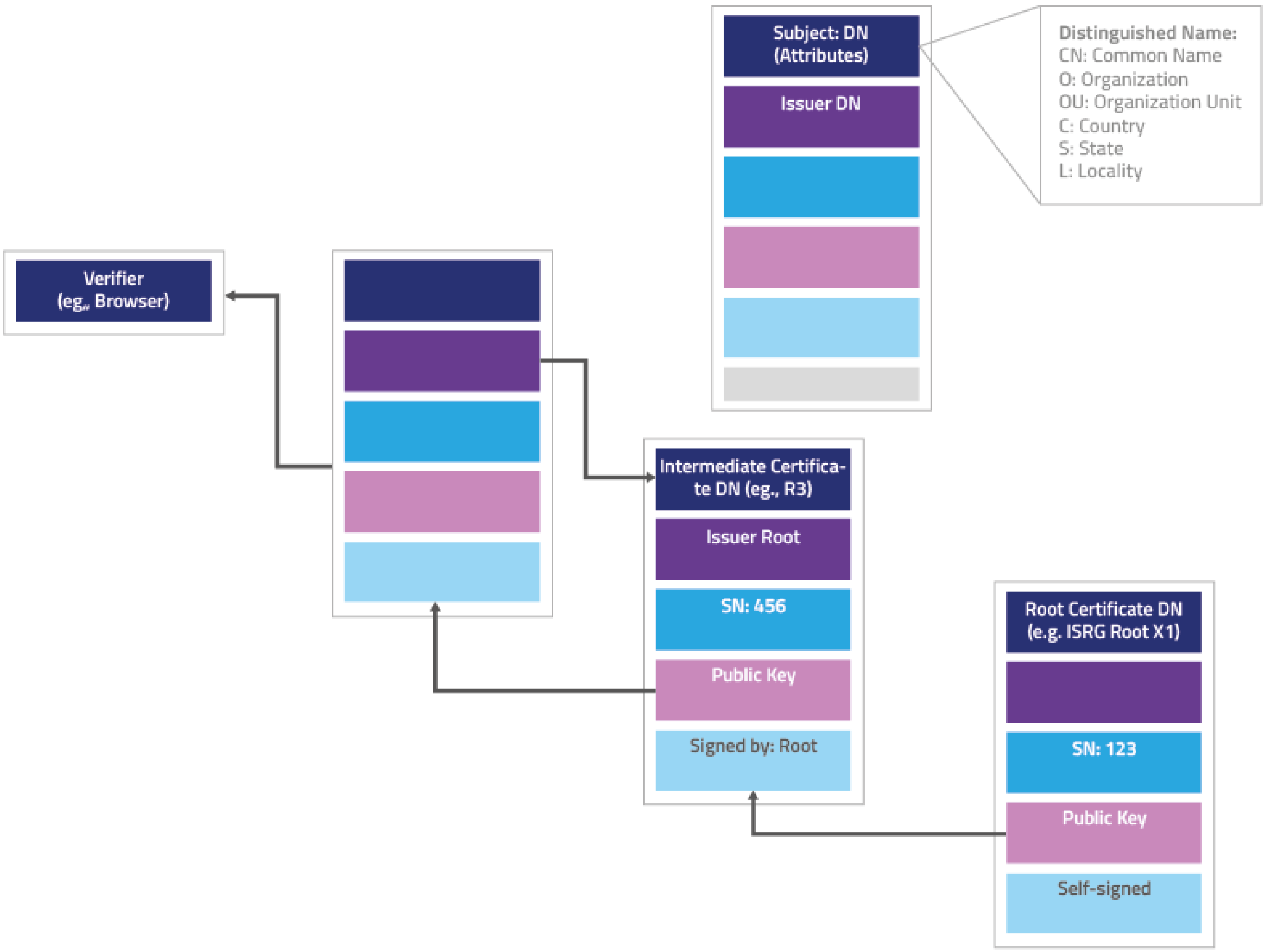
Chain of Trust

The chain of trust is well-known from the certificate chain and describes the capability to validate each component of hardware and software from the end entity up to the root/trust anchor.

Certificate Chaining - Status quo

Certificate chaining is a technical representation of the chain of trust and currency state of the art in Web 2. The goal of a certificate chain is to reach a point that can be trusted by the verifier. In certificate chaining, this is usually the root certificate authority (CA) which serves as the root of trust.
The root CA is identified by a public key certificate (the root certificate) which is self-signed. These self-signed certificates need to be recognized as such and are thereby a published list. The root CA certificates are then stored in the browser (e.g., Chrome Root Store) or the OS itself (e.g., Microsoft Trusted Root Program).
Nowadays, **X.509 certificates** are the industry standard and are used for TLS (HTTPS). The chain is established by the previous CA issuing and signing the subsequent X.509 certificate with its private key and thus establishing a cryptographically verifiable chain of trust until the root CA. Every certificate contains the necessary information to complete the chain of trust.

The figure below demonstrates the certificate chain from the https://gaia-x.eu/ domain.



Different Kinds of Trust Chains

There are different approaches to solve the chaining problem. A solution that fits perfectly for every use case does not exist since there needs to be a balance between privacy, scalability, and complexity.

Credential Chaining

Credential chaining describes an approach to establishing a chain of trust based on Verifiable Credentials. Credential chaining is still at an early stage and missing production-ready implementations. In the following, we describe four concepts of how credential chaining could work in the future.

DID Authorization Registries

The concept via DID authorization registries lists authorized DIDs in a public DID Document that is owned by the root attester. Credential Chaining with registries is based on the W3C DID-Core spec [DID-CORE], and the DID Document contains data about the Controller (who manages the register), LinkedData (a reference to the schema), and the Isser (who is authorized to issue a specific credential). Versioning of the DID Document allows for updating the registry entries and thus adding and removing issuer DID from the Issuers list.
The authorization registry is written to a verifiable data registry and can be accessed publicly in the process of verifying a chained credential. No additional system is required as the DID Document can be written to the same verifiable data registry as all other data. However, a new transaction type would need to be defined in the case of blockchain systems like
[Hyperledger Indy](#) or [Trustchain](#).

An example DID Document written to the Trustchain blockchain is described below:

```
{
 "@context": [
  "https://www.w3.org/ns/did/v1",
  "did:trust:tc:prod:ld:UFyM9YHakDJrfPVxLLyNke"
 ],
 "id": "did:trust:tc:prod:reg:77ahUUvBTLY4BVzNHdTZZS",
 "controller": ["did:trust:tc:prod:id:oRkTGFqoyktnVKn4txqVpv"],
 "linkedData": "did:trust:tc:prod:ld:AA8Z7mGtwGXYGHj86jLWBi",
 "issuers": [
  "did:trust:tc:prod:id:3PDmysyNMsJ3PxCHpn1DT5",
  "did:trust:tc:prod:id:31fvpFns3mrkvGmapLqMsG",
  "did:trust:tc:prod:id:3z5JtaqV27xZKE9Evi8uTB"
 ]
}
```

In the verification of the proof, the verifier needs to query the DID Document containing the trusted list of Issuers plus a reference to the schema (highlighted in bold in the VC below). The verifier then verifies if the issuer of the chained credential is listed in the list of authorized issuers that have been published by the root attester.

```
{
 "@context": [
  "https://www.w3.org/2018/credentials/v1",
  "did:trust:tc:prod:sch:AA8Z7mGtwGXYGHj86jLWBi"
 ],
 "id": "urn:uuid:3978344f-8596-4c3a-a978-8fcaba3903c5",
 "type": ["VerifiablePresentation", "CredentialManagerPresentation"],
 "issuer": "did:trust:tc:prod:id:3PDmysyNMsJ3PxCHpn1DT5",
 "verifiableCredential": [{ ... }],
 "proof": [{ ... }]
}
```


DID authorization registries are W3C DID-Core [DID-CORE] and W3C-VC-Data-Model [W3C-VC] compliant.

VC with Sub-Chain

The first concept of VCs with a sub-chain was published in the Aries RFC 0104: Chained Credentials proposed by Daniel Hardman and Lovesh Harchandani [ARIES-RFC-0104]. In this approach, the credential contains the data and signatures of the chain until the root attester.

The first entity of a VC with a sub-chain is called the root attester, like a traditional credential issuer with a public DIDthat is written to the verifiable data registry and, in the case of Hyperledger Indy, with a published credential definition.
All downstream participants in the provenance chain don't need to have a public DID or credential definition. The verifiable capabilities of the assertions don't depend on the sub issuer and rather depend on the robustness of the cryptographic algorithms used to secure the provenance chain starting with the root attester.

The chained credential concept described in [ARIES-RFC-0104] adds the following conventions to the core requirements of an ordinary VC:

- It contains a special field named schema
  - based64url-encoded representation of its own schema
  - provides the credential with stand-alone capabilities ad doesn't depend on schema or credential definition defined by an external authority
- It contains a special field named provenanceProofs
  - array of tuples
  - first member of the tuple is a list of field names
  - second member is an embedded W3C verifiable presentation that proves the provenance of the values listed in the field names
- For delegation, it is associated with a **trust framework** that describes the semantics of some scheme variants for a family of chained credentials
- Trust is based on an unbroken and cryptographically chain back to the public root attester

Proof of non-revocation and offline mode works the same way as they work for standard credentials.

The following sample credential shows how the schema and provanceProofs are embedded into the W3C-VC-Data-Model.

```
{
    "@context": ["https://w3.org/2018/credentials/v1", "https://github.com/hyperledger/aries-rfcs/tree/main/concepts/0104-delegatable-credentials"],
  "type": ["VerifiableCredential", "ProcenanceCredentials"],
  "schema": "WwogICJAY29udGV4dCIsIC8vSlN...(clipped for brevity)..ob2x",
  "provenanceProofs": [
   [["authorization"], {
     // proof that authorizes holder of the credential
   }
  ]],
}
```

Dynamic Chain Composition

Another approach to chain verifiable credentials is dynamic chain composition. In this concept, the verifier interactively requests the identity and authorization credentials of the credentials issuers until the verifier has queried all data until the root attester.

The credential presented to the verifier needs to contain two additional data fields in the credential subject that allow the verifier to complete the chain of trust:

- The field previousCredentialId contains the credential ID that authorizes the issuer to issue this credential
- The field rootAttester contains the public DID of the root attester

An example of a credential that implements the concept of dynamic chain composition is stated below:

```
{
 "@context": [
  "https://www.w3.org/2018/credentials/v1",
 ],
 "id": "http://example.edu/credentials/58473",
 "type": ["VerifiableCredential", "DynamicChainComposition"],
 "issuer": "https://example.com/issuers/111111",
 "issuanceDate": "2010-01-01T00:00:00Z",
 "credentialSubject": {
  "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
  "previousCredentialId": "did:example:c276e12ec21ebfeb1f712ebc6f1",
  "rootAttester": "https://root.com/issuers/000000",
 },
 "proof": { }
}
```

In the case of verification, the verifier contacts the issuer and requests proof of the authorizing credential with the previousCredentialId. The credential proof then contains information about the previousCredentialId and the rootAttester (which is the same along the chain). The verifier completes the chain by dynamically composing all the proofs needed until the verifier has reached the root attester.

The concept allows for low storage and computation requirements for the holder and transfers the computational effort together with the request calls to the chain of issuers to the verifier.

The concept is compliant with the W3C-VC-Data-Model [W3C-VC] specification.

Authentic Chained Data Container (ACDC)

ACDC is an IETF internet draft-focused specification being incubated at the Trust over IP (ToIP) foundation and builds on top of the ideas of Samuel Smith. An ACDC is a variant of a VC defined by W3C [SMITH-ACDC].
A major use case for the ACDC specification is to provide GLEIF - Global Legal Entity Identifier Foundation - verifiable Legal Entity Identifiers (vLEIs).
The main purpose of the ACDC protocol is to provide provenance proof of their contracted data via a tree of linked ACDCs. The protocol can be extended to a chained variable proof of authorship and thereby be used to authorize delegation [SMITH-ACDC].
The motivation of ACDC is to technically realize more complex use cases that W3C VCs cannot cover. ACDC aims to extend the W3C VC standard.

Trusted Issuer Registry

The trusted issuer registry is used in the EBSI blockchain to:

- Validate Trusted Issuers by their DID
- Verify Trusted Issuers public Information
- Verify if a Trusted Issuer is authorized to issue a given Verifiable Credential

The registry is built on a smart contract of the Ethereum based blockchain. The ledger is permissioned based so only authorized users can write to the smart contract or to deploy more registries. The single point of failure is removed due to the distributed infrastructure and every change to the smart contract is traceable for an audit at every time.

Scalability vs. Privacy vs. Complexity

The registry approaches are easy to implement and usable at low costs. However, they can lack scalability when interacting with the registry. Most blockchain based systems are limited in the amount of throughput when performing write transactions. Read transactions aren't a big issue since there is no consensus involved that is responsible for the synchronization operation.

The transparency approach makes it possible to define an allow list of all authorized issuers: "if the issuer is not listed here, it was not allowed to issue the credential" This approach is easier to validate instead of monitoring a deny list of banned issuers. This dependency on a list gives some security since you are unable to forget whom you give privileges to. If privileges are given out by a credential and you lost the link to whom I gave the credential, you must revoke everything and reissue it.

PKI vs. Web of Trust vs. DKMS

When managing public key material there are two approaches that were used in the past.
The most used approach is the usage of a public key infrastructure (PKI) where multiple certificate authorities build a chain of trust. Each certificate authority is responsible for hosting the issued certificates for verification and to publish the revocation registries. While this approach is easy to scale it has no automated synchronization built in. If a certificate authority goes offline and has no running backup system, the chain of trust is broken. Another negative aspect is possible hierarchy abuse. Since the list of all issued certificates is not public, a certificate authority is able to add a new keypair to an issuer to sign in the name of this organization. But this risk is low since such an action will have heavy consequences.

A more decentralized approach without any hierarchy is the web of trust. Trust is not defined by a structure from the top to the bottom, but by verifying the identity of other members. If member A knows member B and member B knows member C there is a trust connection between member A and member C. While the approach strengthens the self-control of the identity, it comes with a higher risk when losing it. Since there is no authorized member that can reset your access there is a chance of it getting locked out. Another negative aspect with the PGP-servers was that the list of email addresses was publicly available for hackers to do social engineering.

A combination of both approaches can be done with a decentralized key management system (DKMS). It is built on a distributed network to get the positive aspects of PGP with redundancy. But it has a hierarchical approach to prevent a lookout or to implement a business model when offered it as a service. The possible backdoor by hierarchical abuse is still there, but can be recognised as soon as possible since all changes to all identities are visible to everyone. Projects like TrustChain [TRUSTCHAIN] or Hyperledger Indy [HL-INDY] are designed to work this way.

Blockchain vs. DLT vs. Centralised

Distributed system can use different consensus algorithms to prevent the single point of failure:
Crash Fault Tolerance: a leader will perform all the actions and other servers will copy the new results without questioning them. When the leader stops working another one takes its place. This algorithm has a low complexity level and allows a huge throughput.

Byzantine Fault Tolerance: To prevent a single point of control a leader is suggesting the next block that should be stored and the majority must agree with it before finally persisting it. These types of algorithms are often used in blockchain systems to guarantee all nodes have the same ordered set of transactions defining the state. Since there are a lot of messages that must be exchanged the performance is lower in a CFT-based consensus or a System without any consensus.

A blockchain based verifiable data registry removes the single point of control if the network is set up correctly. In terms of a permissionless system like the bitcoin or main Ethereum network the majority of the miners decide which elements will be persisted. If anybody can control the majority of the nodes, it is still impossible to get control over another identity. A private key is needed for this action. But instead, the miners can deny new transactions when looking either into the transaction's body or checking the signature. Therefore, an issuer can be locked out from a system and has no chance to update his/her identity or revocation list. It is very unlikely that this scenario takes place, but nobody can give guarantees since technology is evolving. Another approach is to run the network in a permissioned way. This means only authorised members can write and everyone is able to read. For the SSI use case a limitation on read requests is not useful since the blockchain main purpose is to be a verifiable data registry. The authorisation is handled by a hierarchical approach of the members. In most cases the validators run the system with proof of authority consensus. They are one part of the trust since they guarantee the integrity of the information. The other half is given by the authorized members that identity issuers and allow them by publishing their DID on the ledger to issue credentials. This is important since a verifier never talks directly with the issuer to verify its identity but with the verifiable data registry. A blockchain can offer two relevant aspects:

Transparency: each change is visible for everyone. This means that even someone who is not participating in the consensus is able to verify all changes that happened to the registry. This is important when using a hierarchical approach instead of a full sovereign on. In permissioned systems like TrustChain or Hyperledger Indy a new member with a higher role can set a new public key inside a DID of another member. This action could be used for identity theft. But since the change is visible for everyone immediately, this kind of "backdoor" cannot be used unseen.

Redundancy: To validate a verifiable claim the public key and the revocation registry must be available all the time. If the issuer is shutting down its business, they will also shut down the system hosting these services. In the case of offering them via a blockchain, the other systems can host this information. No change or redirect is required since the identifiers of the DIDs and other resources are the same on all the servers. The high redundancy comes with complex algorithms that are new to the industry. Since distributed work or interaction wasn't so popular in the past the systems were designed as centralized systems. Robust and secure software for public key infrastructures has existed for many years and it's quite simple to host a webservice to publish a revocation list or a schema. There is no need to build a consortium, only to follow standards as the other centralized hosted systems do.

References

Abbreviation	Reference Source
[DID-SPEC-REG]	https://www.w3.org/TR/did-spec-registries/
[DID-PEER]	https://identity.foundation/peer-did-method-spec/
[DID-WEB]	https://w3c-ccg.github.io/did-method-web/
[DIF-DIDComm]	https://identity.foundation/DIDComm-messaging/spec/
[EVERNYM-BBS]	https://www.evernym.com/blog/bbs-verifiable-credentials/
[HAK-21]	https://github.com/decentralized-identity/interoperability/blob/master/assets/interop-mapping-version-by-Hakan-Yildiz(TUB).pdf
[HARD-DIDComm]	https://www.youtube.com/watch?v=TBxWgNmsnvU
[HL-INDY]	https://www.hyperledger.org/use/hyperledger-indy
[INABTA-MDL]	https://inatba.org/news/mobile-drivers-licence-mdl-self-sovereign-identity-ssi-comparison/
[INDY-CC-2022-04-26]	https://wiki.hyperledger.org/display/indy/2022-04-26+Indy+Contributors+Call
[INDY-HIPE-0011]	https://hyperledger-indy.readthedocs.io/projects/hipe/en/latest/text/0011-cred-revocation/README.html
[INDY-RFC-0119]	https://hyperledger-indy.readthedocs.io/projects/hipe/en/latest/text/0119-rich-schemas/README.html#use-of-json-ld
[JSON-LD]	https://json-ld.org/
[LD-SIGNATURES]	https://w3c-dvcg.github.io/ld-signatures/
[OID4VC-CERT]	https://openid.net/certification/
[OID4VC-CON]	https://openid.net/connect/
[OID4VC-SIOP]	https://openid.net/specs/openid-connect-self-issued-v2-1_0.html
[OID4VC-VCI]	https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html
[OID4VC-VP]	https://openid.net/specs/openid-4-verifiable-presentations-1_0.html#name-response
[MATTR-BBS]	https://github.com/mattrglobal/jsonld-signatures-bbs
[PROC-mDL]	https://www.procivis.ch/post/iso-iec-18013-5-vs-self-sovereign-identity-a-proposal-for-an-mdl-verifiable-credential
[RFC7515]	https://tools.ietf.org/html/rfc7515
[RFC7519]	https://tools.ietf.org/html/rfc7519
[SD-JWT]	https://datatracker.ietf.org/doc/draft-ietf-oauth-selective-disclosure-jwt/
[SMITH-ACDC]	https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/ACDC.web.pdf
[TOIP-V2.0]	https://trustoverip.org/wp-content/uploads/Introduction-to-ToIP-V2.0-2021-11-17.pdf
[TRUSTCHAIN]	https://github.com/trustcerts/trustchain-node
[UNI-RES]	https://github.com/decentralized-identity/universal-resolver/blob/main/docs/driver-development.md#how-to
[W3C-VC]	https://www.w3.org/TR/vc-data-model/
[W3C-VC-REV]	https://w3c-ccg.github.io/vc-status-rl-2020/
[W3C-VC-STAT]	https://w3c-ccg.github.io/vc-status-list-2021/

Imprint

eco – Association of the Internet Industry
Lichtstraße 43h
50825 Köln

fon: +49 221 – 7000 48 – 0
fax: +49 221 – 7000 48 – 111

E-Mail: info@eco.de
Web: https://www.eco.de

Vereinsregister Köln; Vereinsregisternummer: 14478
Umsatzsteueridentifikationsnummer: VAT-ID: DE 182676944

Point of Contact:

Andreas Weiss, Geschäftsführer eco - Association of the Internet Industry
E-Mail: andreas.weiss@eco.de
Address: Lichtstraße 43h, 50825 Köln
Website: www.gxfs.eu

Commissioned study author:

TrustCerts GmbH
Munscheidstraße 14
45886 Gelsenkirchen
Point of contact: Mirko Mollik
E-Mail: mirko.mollik@fit.fraunhofer.de
Website: www.trustcerts.de

Peer Reviewed by the following persons in October 2023:

- Mirko Mollik, Fraunhofer FIT
- Steffen Schwalm, Bitkom e. V.
- Berthold Maier, T-Syxstems
- Detlef Hühnlein, ecsec