ETSI GR CIM 018 V1.1.1 (2022-09)

cross-cutting Context Information Management (CIM); NGSI-LD; Enabling chain of trust from Content Sources to Content Consumers

Reference

DGR/CIM-0018

Keywords

blockchain, digital signature, distributed ledger, provenance, trust, verifiable registry

Contents

Intellectual Property Rights

Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) cross-cutting Context Information Management (CIM).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Executive summary

The present document focuses on the issue of trust and identity, scrutinizing the current landscape and pointing out how the current initiatives can be integrated into the NGSI-LD [i.18] ecosystem.

The goal is to design technical means for enabling a chain of trust from content sources to content consumers that helps endorse documents, by connecting or embedding verifiable credentials into NGSI-LD documents (defined in [i.18]).

These solutions will then enhance the current NGSI-LD standard [i.18].

Introduction

When focusing the issues of trust and identity, it is important to notice that trust is orthogonal to identity: while Self-Sovereign Identity (SSI) is about a digital identity that benefits the individual, trust is not about giving identifiers to assets, but about providing provenance to an asset, thus complementing each other.

Frameworks for provenance seek to delegate and transfer trust, in order to build authenticity solutions that are decentralized. In September 2021, the Coalition for Content Provenance and Authenticity (C2PA) released its content provenance specifications. C2PA leverages DID (a form of SSI) and Verifiable Credentials from W3C®.

The W3C Verifiable Credentials and the C2PA initiative, although still in their infancy, aim at filling a gap, creating coordinated efforts towards standardization of technical specifications for provenance that robustly link content to producers.

The present document includes material copied from or derived from "Decentralized Identifiers (DIDs) v1.0" [i.10], available at https://www.w3.org/TR/did-core/ and "Verifiable Credentials Data Model v1.1" [i.11], available at https://www.w3.org/TR/vc-data-model/. Copyright© 2022 W3C® (MIT, ERCIM, Keio, Beihang).

It also uses material form the "C2PA Technical Specifications" [i.12]) (licensed under a Creative Commons Attribution 4.0 International License) and from the W3C® "Data Integrity 1.0" [i.1] Specification, published by the Credentials Community Group under the W3C® Community Final Specification Agreement (FSA).

The present document discusses possible approaches about how such specifications should be completed and integrated with distributed registry/ledger technologies having precise requirements, for the purpose of integration with the NGSI-LD ecosystem [i.18].

1 Scope

The present document focuses on the issue of trust and identity, scrutinizing the current landscape, analysing the existing requirements, and pointing out how such current initiatives and requirements can be integrated into the NGSI-LD ecosystem.

The approach revolves around two goals:

- A first goal is decentralization.
- A second goal is not only the verification of the credential, but to have means to distinguish which of its claims can be considered authoritative and which cannot, or, equivalently, to discover information that can be used to evaluate the risk of accepting the claims in the VC.

Thus, the present document examines solutions to verify integrity, and to precisely evaluate attribution and authenticity of NGSI-LD Context Information throughout its lifecycle. The goal is to design technical means for enabling a chain of trust from content sources to content consumers that helps endorse documents, by connecting or embedding verifiable credentials into NGSI-LD documents.

These solutions will then enhance the current NGSI-LD standard (ETSI GS CIM 009) [i.18].

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] W3C® Final Community Group Report 22 July 2022: "Data Integrity 1.0".

NOTE: Available at https://www.w3.org/community/reports/credentials/CG-FINAL-data-integrity-20220722.

[i.2] W3C® Recommendation 3 May 2022: "BBS+ Signatures 2020".

NOTE: Available at https://w3c-ccg.github.io/ldp-bbs2020/.

[i.3] W3C® Recommendation 16 July 2020: "JSON-LD 1.1 Framing".

NOTE: Available at https://www.w3.org/TR/json-ld11-framing/.

[i.4] IETF draft-sporny-hashlink-07 expired on November 2021: "Cryptographic Hyperlinks ".

NOTE: Available at https://www.ietf.org/archive/id/draft-sporny-hashlink-07.txt.

[i.5] InterPlanetary File System (IPFS).

NOTE: Available at https://ipfs.tech.

[i.6] IETF RFC 7515: "JSON Web Signature (JWS)".

NOTE: Available at https://tools.ietf.org/html/rfc7515.

[i.7] IETF RFC 7516: "JSON Web Encryption (JWE)".

NOTE: Available at https://tools.ietf.org/html/rfc7516.

[i.8] IETF RFC 7519: "JSON Web Token (JWT)".

NOTE: Available at https://tools.ietf.org/html/rfc7519.

[i.9] IETF RFC 7518: "JSON Web Algorithms (JWA)".

NOTE: Available at https://tools.ietf.org/html/rfc7518.

[i.10] W3C® Recommendation 19 July 2022: "Decentralized Identifiers (DIDs) v1.0".

NOTE: Available at https://www.w3.org/TR/did-core/.

[i.11] W3C® Recommendation 03 March 2022: "Verifiable Credentials Data Model v1.1".

NOTE: Available at https://www.w3.org/TR/vc-data-model/.

[i.12] C2PA Specifications: "C2PA Technical Specification".

NOTE: Available at https://c2pa.org/specifications/specifications/1.0/specs/C2PA_Specification.html.

[i.13] IETF RFC 8785: "JSON Canonicalization Scheme (JCS)".

NOTE: Available at https://tools.ietf.org/html/rfc8785.

[i.14] IETF draft-jordan-jws-ct-01: "JWS Clear Text JSON Signature Option (JWS/CT)".

NOTE: Available at https://datatracker.ietf.org/doc/html/draft-jordan-jws-ct-01.

[i.15] W3C® Recommendation 25 February 2014: "RDF 1.1 Concepts and Abstract Syntax".

NOTE: Available at https://www.w3.org/TR/rdf11-concepts/.

[i.16] W3C® Working Group Note 07 November 2013: "RDF 1.1 JSON Alternate Serialization (RDF/JSON)".

NOTE: Available at https://www.w3.org/TR/rdf-json/.

[i.17] W3C® Draft Community Group Report 13 April 2021: "RDF Dataset Canonicalization".

NOTE: Available at https://json-ld.github.io/rdf-dataset-canonicalization/spec.

[i.18] ETSI GS CIM 009: "Context Information Management (CIM); NGSI-LD API".

[i.19] ETSI EN 319 412-1: "Electronic Signatures and Infrastructures (ESI); Certificate Profiles; Part 1: Overview and common data structures".

[i.20] ISO 3166-1: "Codes for the representation of names of countries and their subdivisions - Part 1: Country codes".

[i.21] Payments Services Directive (EU) 2015/2366 of the European Parliament and of the Council of 25 November 2015 on payment services in the internal market, amending Directives 2002/65/EC, 2009/110/EC and 2013/36/EU and Regulation (EU) No 1093/2010, and repealing Directive 2007/64/EC.

[i.22] ETSI TS 119 495: "Electronic Signatures and Infrastructures (ESI); Sector Specific Requirements; Certificate Profiles and TSP Policy Requirements for Open Banking".

[i.23] ISO 17442-1:2020: "Financial services - Legal entity identifier (LEI) - Part 1: Assignment".

NOTE: Available at https://www.iso.org/standard/78829.html.

[i.24] JSON-LD Signatures implementation.

NOTE: Available at https://github.com/digitalbazaar/jsonld-signatures.

[i.25] jsonld-signatures-bbs implementation.

NOTE: Available at https://github.com/mattrglobal/jsonld-signatures-bbs.

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

assertion: data structure which represents a statement asserted by an actor concerning the asset

atomic entity: signed NGSI-LD Entity with only one attribute

Base64url: modification of the main Base64 standard, the purpose of which is the ability to use the encoding result as filename or URL address

BBS+ signature: digital signature algorithm by Boneh, Boyen, and Shachum

BLS signature: Boneh–Lynn–Shacham (BLS) cryptographic signature scheme

CL signature: signature scheme developed by Jan Camenisch and Anna Lysyanskay

claim: assertion made about a subject

controller document: set of data that specifies one or more relationships between a controller and a set of data, such as a set of public cryptographic keys

derivation process: process that transforms NGSI-LD attributes in Sealed Attributes

DID controller: entity that has the capability to make changes to a DID document

DID document: set of data describing the DID subject

DID method: definition of how a specific DID method scheme is implemented

DID resolution: process that takes as its input a DID and a set of resolution options and returns a DID document

hard binding: one or more cryptographic hashes that uniquely identifies either the entire asset or a portion thereof

holder: role an entity might perform by possessing one or more verifiable credentials and generating presentations from them

issuer: role an entity can perform by asserting claims about one or more subjects, creating a verifiable credential

manifest: set of information about the provenance of an asset based on the combination of one or more assertions (including content bindings), a single claim, and a claim signature

NGSI-LD Attribute: reference to both an NGSI-LD Property and to an NGSI-LD Relationship

NGSI-LD Context Broker: architectural component that implements all the NGSI-LD interfaces

NGSI-LD Context Consumer: agent that uses the query and subscription functionality of NGSI-LD to retrieve context information

NGSI-LD Context Producer: agent that uses the NGSI-LD context provision and/or registration functionality to provide or announce the availability of its context information to an NGSI-LD Context Broker

NGSI-LD Context Source: source of context information which implements the NGSI-LD consumption and subscription (and possibly provision) interfaces defined by the present document

NGSI-LD Entity: informational representative of something that is supposed to exist in the real world, physically or conceptually

NGSI-LD Property: description instance which associates a main characteristic, i.e. an **NGSI-LD Value**, to either an NGSI-LD Entity, an NGSI-LD Relationship or another NGSI-LD Property and that uses the special *hasValue* property to define its target value

NGSI-LD Relationship: description of a directed link between a subject which is either an NGSI-LD Entity, an NGSI-LD Property or another NGSI-LD Relationship on one hand, and an object, which is an NGSI-LD Entity, on the other hand, and which uses the special *hasObject* property to define its target object

recreation process: opposite process of the Derivation Process

Sealed Attribute: NGSI-LD attribute structure with "ngsildproof" property

soft binding: content identifier that is either:

a) not statistically unique, such as a fingerprint; or

b) embedded as a watermark in the identified digital content

verification method: set of parameters that can be used together with a process to independently verify a proof

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BBS	Boneh, Boyen, and Shachum
BLS	Boneh-Lynn-Shacham
BMFF	Base Media File Format
CBOR	Concise Binary Object Representation
CID	Content IDentifier
CL	Jan Camenisch and Anna Lysyanskay
DID	Decentralized IDentifier
DoS	Denial of Service
GDPR	General Data Protection Regulation
HMAC	Hash-based Message Authentication Code
HTTP	Hypertext Transfer Protocol
ID	Identifier
I-JSON	Internet JSON
IPFS	InterPlanetary File System
IRI	Internationalized Resource Identifier
JCS	JSON Canonicalization Scheme
JOSE	JSON Object Signing and Encryption
JSON	JavaScript Object Notation
JSON-LD	JSON-Linked Data
JUMBF	JPEG Universal Metadata Box Format
JWA	JSON Web Algorithms
JWE	JSON Web Encryption
JWS	JSON Web Signature
JWS/CT	JWS "Clear Text"
JWT	JSON Web Token
MAC	Message Authentication Code
NGSI	Next Generation Service Interfaces
NGSI-LD	NGSI-Linked Data
PIA	Privacy Impact Assessment

PKI	Public Key Infrastructure
RDF	Resource Description Format
SSI	Self-Sovereign Identity
URI	Uniform Resource Identifier
URL	Universal Resource Locator
UTF	Unicode (or Universal Coded Character Set) Transformation Format
UTF-8	Unicode Transformation Format, 8 bit
VC	Verifiable Credential
ZKP	Zero Knowledge Proof

4 Data integrity and provenance

4.1 Goals

This clause is focused on data integrity problem, specifically to JSON-LD documents and NGSI-LD Entities.

The preferred solution, in both literature and industry, to achieve data integrity is the implementation of a digital signature system, where, in this case, a digest file of the NGSI-LD Entity, cryptographically encoded with the signer private key, bound with it, will guarantee the non-corruption of data and the association to a specific private key.

Thus, the first goal of the present document is to design a standard to guarantee the integrity of an NGSI-LD Entity.

In order to do this, it is important to create guidelines that ensure data accuracy and consistency over the Entity's entire life-cycle.

4.2 Scenarios and Use Cases

A generic scenario can be the signing process of a generic JSON-LD document.

In this case, a typical scenario is the generation of an NGSI-LD Entity from a content creator (Producer or Source) sent through multiple Context Brokers to a Client.

In this scenario, where a typical NGSI-LD Entity will contain Attributes (Property or Relationship), it is important to guarantee that these values will not be altered through all its cycles, so that a client without contact creators can be sure of its integrity.

4.3 Architectures and Existing Specifications

4.3.1 W3C data integrity model

4.3.1.0 Introduction

One already implemented solution can be the W3C Data integrity [i.1], which is not yet a W3C standard.

With this, it is possible to create a data integrity proof, which is a set of attributes that represent a digital proof and all parameters required to verify it.

A data integrity proof is containing the following attributes:

- type: which indicates the specific type of digital signature used
- proofPurpose: a parameter that ensures that the digital proof is used for the reason it was created for
- verificationMethod: a set of parameters required to independently verify the proof
- created: date and time of the proof generation
- proofValue: the value of the encoded hash file

Here is a proof example that uses the JcsSignature2020 proof type:

```
"proof": {
  "type": "JcsSignature2020",
  "created": "2020-11-05T19:23:24Z",
  "verificationMethod":"https://di.example/issuer#z6MkjLrk3gKS2nnkeWcmcxiZPGskmesDpuwRBorgHxUXfxnG",
  "proofPurpose": "assertionMethod",
  "proofValue":"zQeVbY4oey5q2M3XKaxup3tmzN4DRFTLVqpLMweBrSxMY2xHX5XT…"
}
```

More optional attributes can be implemented in the data integrity proof in order to add further information or limitation (`domain, challenge`).

At the moment the following signature suites are contemplated in W3C Data integrity specification [i.1]: eddsa-2022, nist-ecdsa-2022, koblitz-ecdsa-2022, rsa-2022, pgp-2022, bbs-2022, eascdsa-2022, ibsa-2022, and jws-2022.

4.3.1.1 Controller Document

The verification process is possible through the access of the Controller Document, a set of data that specifies the relationship between a controller, the entity who can change the controller document, and other data sets such as a public cryptographic key.

Whoever wants to verify the data integrity proof needs to ensure that a verification method is bound to a specific controller, by going from the verification method attribute in the proof to the controller document, ensuring that this also contains the same verification method and the same proof purpose.

The `verificationMethod` property in the Controller Document is optional, but if present it has to be a set of verification method's map.

Each verification method has to include the following properties:

- `id:` a string that conforms to the URL syntax.
- `type:` which indicates the specific type of verification method used.
- `controller`*:* a string that conforms to the URL syntax.

The `controller` value for a verification method is not necessarily a controller. Controllers are expressed using the controller property at the highest level of the controller document.

The verification method could be filled with other verification material for example the cryptographic public key, accordingly with the verification method type.

Examples of supported verification material are `publicKeyJwk` and `publicKeyMultibase`, representing respectively the JSON Web Key and the multibase encoded public key.

Here is a Controller Document example:

```
{
  "@context": [
     "https://w3id.org/security/suites/ed25519-2020/v1",
     "https://w3id.org/security/v2"
  ],
  "id": "https://example.org/controllerDocument",
  "verificationMethod":{
    "id":https://example.org/controllerDocument.json#z6MksnCpNjGV",
    "type": "Ed25519VerificationKey2020",
    "controller": "https://example.org/controllerInfo.json",
    "publicKeyMultibase": "z6MksnCpNjGV"
  },
  "assertionMethod": {
    "id":https://example.org/controllerDocument.json#z6MksnCpNjGV",
    "type": "Ed25519VerificationKey2020",
    "controller": "https://example.org/controllerDocument.json",
    "publicKeyMultibase": "z6MksnCpNjGVkL87mnaC2sbBbAZ"
  }
}
```

The verification method could be included by reference using an URL and its properties will need to be retrieved from elsewhere in the Controller Document or from another Controller Document. This is done by dereferencing the URL and searching the resulting resource for a verification method map with an id property whose value matches the URL.

The Controller Document can express also verification relationship between the controller and a verification method, enabling this to be used for different purposes.

Contemplated verification relationships are: Authentication, Assertion, keyAgreement, capabilityInvocation, capabilityDelegation.

The W3C Data integrity specification [i.1] suggests that any verification relation should be registered in the Data Integrity Specification Registries, which is not already defined.

At the actual state of the data integrity specification no other proof types that are not data integrity signatures are implemented.

4.3.1.2 Selective disclosure and aggregation

A secondary problem of the digital signature of a piece of data is that, in order to verify it, the entire document needs to be shared.

In many scenarios it can be useful or even necessary to choose what information can be shared without renouncing the integrity granted by the proof.

Also, in order to recreate provenance of data, it is commonly used the possibility to merge single data fragments without losing information of the original signatures.

Data integrity is a prerequisite, not only required for the final data, coming from multiple merging processes, but also required for every single original data fragment.

One possible solution can be the signature aggregation mechanism.

This property will guarantee the possibility to create an aggregated signature from every single message signature which can be verified by an aggregated public key created from all signer's public keys.

This process does not require the aggregator's signature, so whoever performs the aggregation does not enter inside the trust model.

The overhaul data integrity is guaranteed because only the aggregated public key made out of all original signers can verify the aggregated signature.

Additional information about the signer's identity, or its public key, can be added to the original fragments and signed in order to, subsequently, prove which fragment was signed by whom.

In addition, with Data Integrity Specification it is possible to utilize signature suites that allow the implementation of a selective disclosure, with Zero Knowledge proof technology and signature aggregation, such as BBS signature suite or Cl signature suite.

4.3.1.3 W3C BBS+ Signature suite

The BBS signature was invented by Dan Boneh, Ben Lynn and Hovav Shacham.

It uses elliptic curve pairings and guarantees the following benefits:

- is very simple to use (aside the extreme complexity of elliptic curve pairing)
- is deterministic and verifiably
- is able to provide signatures aggregations
- Signatures and keys can be 32 bytes long, but not at the same time (BLS12-381 48 bytes)

The W3C BBS+ Signatures 2020 specification [i.2], which is not a W3C standard, describes the utilization of BBS+ signatures to provide the capability of zero-knowledge proof disclosures.

Using traditional digital signatures, starting with a message and the issuer private key, it is possible to create a signature.

On verification, starting with the original message and issuer public key, it is possible to establish if the signature is valid or not.

Signature validation tells if the message is untampered with and it gives origin authenticity, in the sense that it is possible to know who produced that signature.

Traditional signature scheme can be represented as in Figure 4.3.1.3-1.

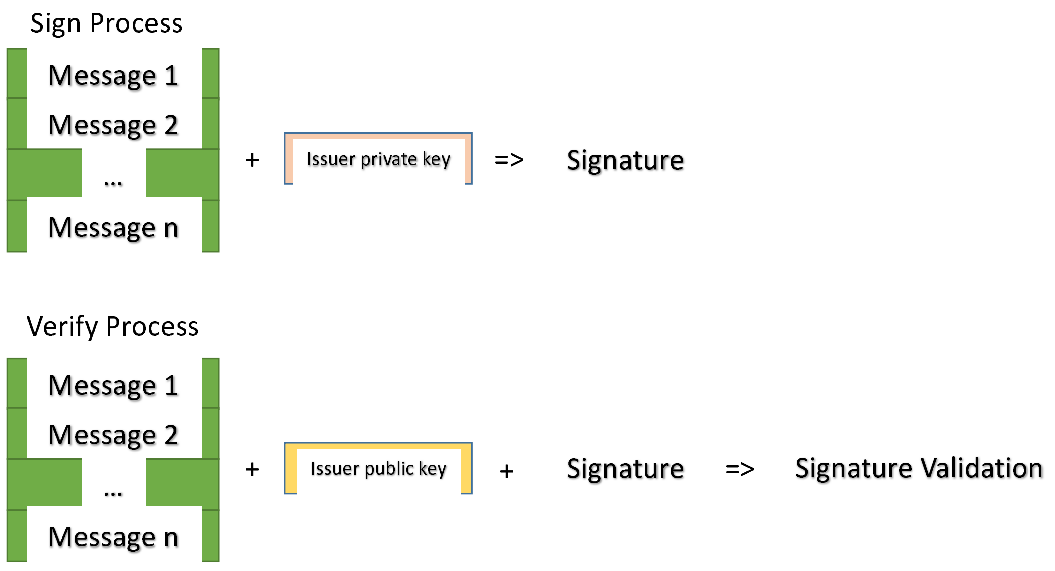


Figure 4.3.1.3-1: Traditional signature scheme

On the other side, BBS+ signature is a multi-message digital signature scheme, represented as in Figure 4.3.1.3-2.

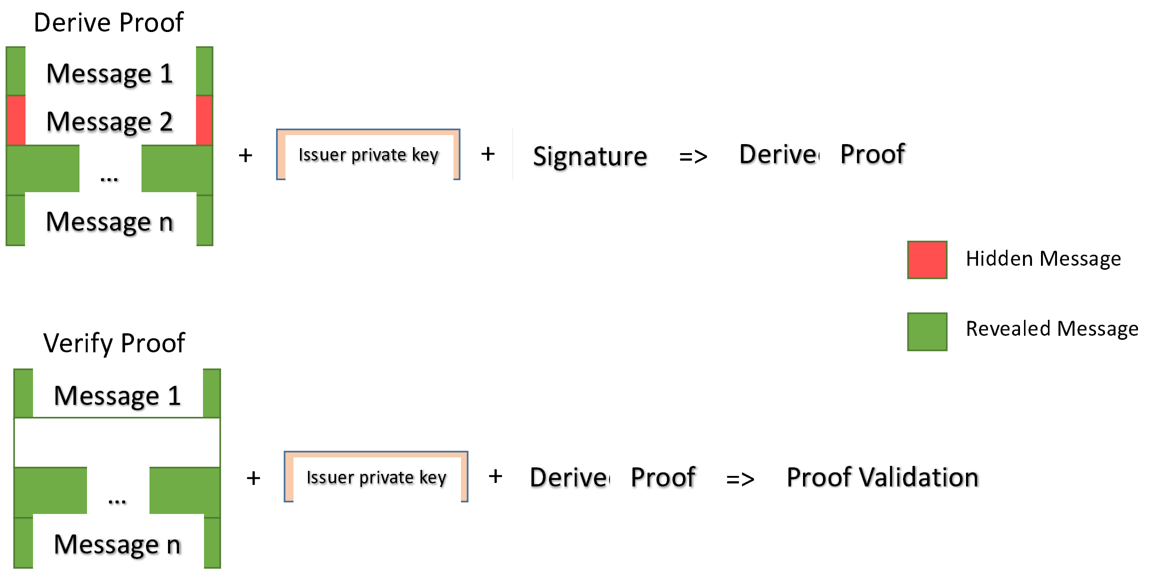


Figure 4.3.1.3-2: BBS signature scheme

Instead of a single message on input it is possible to have many of them. A group of messages are signed with the issuer's private key, and can be verified in groups as always.

In addition to traditional digital signatures it is possible to create what is called a derived proof.

Choosing one or more messages, from the original group, that has to be revealed, with issuer's public key and the original signature it is possible to create a derived proof.

On verification, a verifier using the sets of messages revealed with the issuer's public key and the derived proof can verify if the proof is valid or not.

In this scenario there is the same integrity for all messages, in addition to the possibility of sensitive disclosure of information.

The BLS Signature Proof is a proof of knowledge of the original BLS signature. This means that the information of the original signature is not shared, but only that its existence and knowledge is proved.

The signature aggregation property is guaranteed using the BLS signature.

An aggregate signature is a shorter representation of n signatures provided by different users on different messages.

The linearity property of elliptic curve pairing guarantees the possibility to verify in batch a group of messages following the schema of Figure 4.3.1.3-3.

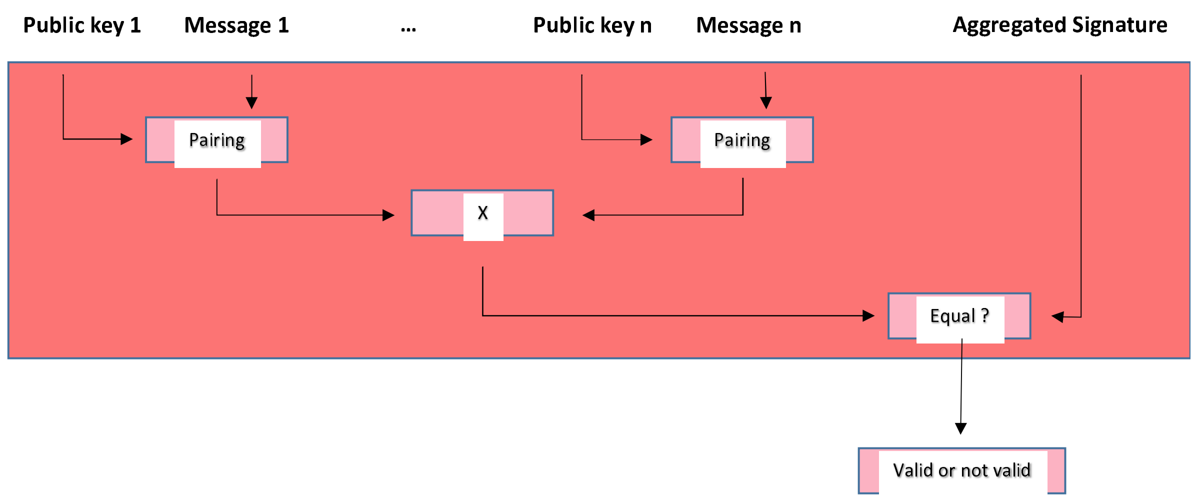


Figure 4.3.1.3-3: BBS multi-signed message Verification scheme

4.3.1.4 Bls12381G2Key2020

Is a Key Pair Standard based on elliptic curve pairing or bilinear map.

This proof has to contain, over the already defined Data integrity proof properties, a new property:

`requiredRevealStatements:` which has to be an array of unsigned integers representing the indices of the statements in the canonical form that has always to be revealed in a derived proof.

Here is an example of Bls Signature 2020 generated with Bls12381G2Key2020:

```
"proof": {
        "type": "BbsBlsSignature2020",
        "created": "2020-04-25",
        "verificationMethod": "did:example:489398593#test",
        "proofPurpose": "assertionMethod",
        "proofValue":"F9uMuJzNBqj4j+HPTvWjUN/MNoe6KRH0818WkvDn2Sf7kg1P17YpN",
        "requiredRevealStatements": [ 4, 5 ]
}
```

The main characteristic of a BBS signed document is the possibility to create a derived document (revealed document), containing revealed statements from the original document and a derived proof.

Implementing in JSON-LD the derived document is created via JSON-LD frame [i.3].

This is possible due to BBS proof of knowledge linked data proof which is a proof that is derived from a BbsBlsSignature2020 linked data proof, where a subset of the original statements is revealed.

A derived proof has to contain a type attribute that has a type equal to BbsSignatureProof2020.

A derived proof, over the already defined Data integrity proof properties, has to contain a nonce attribute.

This is a proof of knowledge of the original signature, not the signature itself.

It can be verified, proving the knowledge of the original signature, validating all the information shared.

Here is an example of Bls Signature Proof 2020:

```
"proof": {
        "type": "BbsBlsSignatureProof2020",
        "created": "2020-04-25",
        "verificationMethod": "did:example:489398593#test",
        "proofPurpose": "assertionMethod",
         "proofValue":"kTTbA3pmDa6Qia/JkOnIXDLmoBz3vsi7L5t3DWySI/VLmBqle …",
        "nonce": "6i3dTz5yFfWJ8zgsamuyZa4yAHPm75tUOOXddR6krCvCYk77sbCOuEV …"
}
```

4.3.1.5 BLS use cases

During the past years, BLS signatures were introduced in several blockchains, in order to ensure cryptographically that a specific validator has actually verified a particular transaction.

This method allows validators to sign messages. The resulting signatures can then be aggregated and verified at scale. This enables a full Proof-of-Stake system, with a massive number of validators, to be more efficient, since it enables to compress all transaction signatures in a block with one signature, in order to improve performance.

4.3.2 Integrity of linked content

4.3.2.0 Introduction

The solutions described so far can provide and guarantee the integrity of data. This means that every information stored inside a data piece is tamper-evident.

But what about all referred data that lies outside of the protection of the proof?

Starting with linked data contexts, a lot of information is usually expressed via URL. Digital signatures provide only the integrity of URLs but not the linked content. For this reason, Content integrity protection is required.

To achieve that, two possible solutions can be implemented: Hashlinks or IPFS links.

4.3.2.1 Hashlinks

Hashlinks are defined in the IETF Hashlink draft specification [i.4].

The hashlink data model is a simple expression of a cryptographic hash of the resource, one or more URLs, and a content type. The resource hash is the mechanism that enables content integrity protection for the associated data stream. The hashlink specification contemplates two different ways to serialize a hashlink.

Following the recommended method called "hashlink URL", the URL itself is followed by the tree characters "`hl:`", followed by the resource hash.

The value of the resource hash can be generated by utilizing the following algorithm:

1) Generate the raw hash value by processing the resource data using the cryptographic hashing algorithm.

2) Generate the multihash value by encoding the raw hash using the Multihash Data Format (multihash).

3) Generate the multibase hash by encoding the multihash value using the Multibase Data Format (multibase).

4) Output the multibase hash as the resource hash.

Here is an example of hashlink:

```
http://example.org/hw.txt?hl=zQmWvQxTqbG2Z9HPJgG57jjwR154cKhbtJenbyYTWkjgF3
```

4.3.2.2 IPFS

IPFS or InterPlanetary File System [i.5] is a protocol and peer-to-peer network for storing and sharing data in a distributed file system.

IPFS is a decentralized file system able to guarantee security, privacy and resistance to censorship of data.

IPFS uses content-addressing to uniquely identify each file in a global namespace connecting all computing devices.

So, the request of a file is made not by a location address but by a hash.

Whenever a file is added to IPFS it is split into smaller chunks, cryptographically hashed, and given a unique fingerprint called a Content IDentifier (CID). This CID acts as a permanent record of the file as it exists at that point in time.

When other nodes look up the file, they ask their peer nodes who is storing the content referenced by the file's CID. When they view or download the file, they cache a copy and become another provider of the content until their cache is cleared.

A node can pin content in order to keep (and provide) it forever, or discard content it has not used in a while to save space. This means each node in the network stores only content it is interested in, plus some indexing information that helps figure out which node is storing what.

If a new version of the file is added to IPFS, its cryptographic hash is different, and so it gets a new CID. This means files stored on IPFS are resistant to tampering and censorship.

Any changes to a file do not overwrite the original, and common chunks across files can be reused in order to minimize storage costs.

Here is an example of an ipfs link:

```
"ipfs:/ipfs/QmXfrS3pHerg44zzK6QKQj6JDk8H6cMtQS7pdXbohwNQfK/image"
```

An IPFS link can be used for a decentralized file in order to guarantee the content integrity protection.

4.3.3 JWS: JSON Web Signature

A remarkable mention goes to JWS (JSON Web Signature) [i.6], which is a compact signature format.

JWS represents digitally signed or MACed content using JSON data structures and base64url encoding.

JWS together with JWE (JSON Web Encryption) [i.7] are concrete implementations of JWT (JSON Web Token) [i.8].

JWS has two serializations:

- Compact Serialization
- JWS JSON Serialization

In its compact serialization, which is the most common one, JWS consists of three parts:

- JOSE Header
- JWS Payload
- JWS Signature

All of them are base64url-encoded for transmission, and typically represented as the concatenation of the encoded strings in that order, with the three strings being separated by period ('.') characters.

```
BASE64URL(UTF8(JWS Header)) ||'.' || BASE64URL(JWS Payload) || '.' || BASE64URL(JWS Signature)
```

The JOSE (JSON Object Signing and Encryption) Header is a JSON object containing the parameters describing the cryptographic operations and parameters employed, and is composed of a set of Header Parameters.

JOSE Header members are the union of two values:

- JWS Protected Header
- JWS Unprotected Header

JWS Protected Header contains the Header Parameters that are integrity protected by the JWS Signature, while JWS Unprotected Header contains Header Parameters that are not.

In JWS Compact Serialization, only the JWS Protected Header is present in the JOSE Header.

In the JWS JSON Serialization the members of the JOSE Header are the union of the members of the JWS Protected Header and the JWS Unprotected Header.

So, in JWS JSON Serialization there is another JOSE Header element which is not base64url encoded.

The JOSE Header contains information about the type of content to be signed and information about the algorithm to apply.

Allowed cryptographic algorithms and identifiers for JWS are described in the separate JSON Web Algorithms (JWA) specification [i.9].

An example of a JWS Protected Header is:

```
{
  "typ":"JWT",
  "alg":"HS256"
}
```

which declares that the encoded object is a JSON Web Token and the JWS Protected Header and the JWS Payload are secured using the HMAC SHA-256.

Subsequently the Header will be encoded as BASE64URL(UTF8(JWS Protected Header)) giving the value:

```
eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
```

Following the example, a JWS Payload will be:

```
{
  "iss":"joe",
  "exp":1300819380,
  "http://example.com/is_root":true
}
```

that encoded will be:

```
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ
```

At the end the JWS Signature will be created computing the HMAC of the JWS Signing Input ASCII(BASE64URL(UTF8(JWS Protected Header)) || '.' || BASE64URL(JWS Payload)) with the HMAC SHA-256 algorithm using a key and base64url-encoding it.

This will give the value:

```
dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

Concatenating these three values in the order with period ('.') characters between the parts generates a complete JWS representation using the JWS Compact Serialization.

```
eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9
leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ.dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

An example of JWS JSON Serialization Representation including a JWS Unprotected Header will be:

```
{
  "payload":"eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leG
  FtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ",
  "protected":"eyJhbGciOiJFUzI1NiJ9",
  "header":{
    "kid":"2010-12-29"
  },
  "signature":"DtEhU3ljbEg8L38VWAfUAqOyKAM6-Xx-F4GawxaepmXFCgfTjDxw5djxLa8ISlSApmWQxfKTUJqPP3-
   Kg6NU1Q"
}
```

In this serialization the JWS is a JSON object and, unlike the JWS Compact Serialization, it enables multiple digital signatures and/or MACs to be applied to the same content.

4.4 Status of the Specifications

W3C Data integrity and W3C BBS+ Signatures 2020 are experimental specifications.

They are not a W3C Standard nor are they on the W3C Standards Track.

A JavaScript library of the Linked Data Signatures specification for JSON-LD, for Node.js and browsers, called JSON-LD Signatures (jsonld-signatures) [i.24], has been implemented by the digitalbazaar group (https://github.com/digitalbazaar/jsonld-signatures).

This library utilizes the ed25519-signature-2020 suite.

An additional JavaScript library extends the JSON-LD Signatures library, adding the possibility to use the BBS+ signature suite, called jsonld-signatures-bbs [i.25], has been implemented by the mattrglobal group (https://github.com/mattrglobal/jsonld-signatures-bbs).

JWS [i.6], JWT [i.8] and JWE [i.7] are all internet Standards.

Many JavaScript libraries are present for JWT [i.6], JWS [i.8] and JWE [i.7] specifications.

5 Decentralized Identifiers

5.1 Introduction

As already seen in clause 4, using a digital signature, it is possible to ensure that a data piece is signed by the owner of that particular private key.

But how to know whether this process is trustworthy?

This issue is very important nowadays in digital credentials systems.

5.2 Goals

The second goal of the present document is to report about existing and on-going specifications that can be used and enhanced to guarantee the trustworthiness, privacy and authenticity of all subjects involved during an NGSI-LD Entity life-cycle.

5.3 Scenarios and Use Cases

In everyday life, even pre-digital, standardized identities and physical credentials are granted by third party trustworthy institutions, and the power to create or revoke them does not belong to subjects.

In the digital world, identifiers are increasingly important, because all that is possible to do with data or what is connected to other entities relies on them.

Of course, global digital identifiers are already present.

Nowadays e-mails, passwords and usernames are typically used to create accounts in different organizations, who store directly or through a third party all information.

This system is centralized because all information is stored in single organizations on the web.

All these identifiers are not persistent, not resolvable to get more information nor cryptographically verifiable.

The main goal is the creation of digital identifiers that an entity can own independently of any organization or institution.

5.4 Architectures and Existing Specifications

5.4.0 Introduction

A possible solution can be the implementation of W3C DID or decentralized identifiers [i.10].

Following the W3C Decentralized identifiers (DIDs) v1.0 data model, DIDs are a new type of globally unique identifier (URI) that does not require a centralized registration authority because control of the identifier can be proved, using cryptography.

DIDs move the control point to the subject of the digital identifier.

Every subject can have multiple DIDs and each one will give a lifetime encrypted private channel to other entities.

The main goals of DID are:

- **Decentralization:** they eliminate the requirement of centralized authorities or single point failure in identifier management, including the registration of globally unique identifiers, public verification keys, services and other information.
- **Persistence:** once created, a DID is permanently assigned to a subject.
- **Control:** they give entities the power to directly control their digital identifiers.
- **Simplicity:** creation of DIDs is easy and cheap.
- **Cryptographically verification:** enables DID controllers to provide cryptographic proof when interacting with other entities.
- **Discoverability:** make it possible for entities to discover basic set of information on the subject.

A DID is a simple text string consisting of three parts:

1) the "did:" URI scheme identifier;
2) the identifier for the DID method;
3) the DID method-specific identifier.

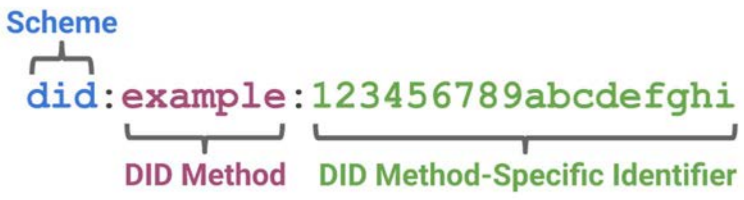

Scheme
did:example:123456789abcdefghi
DID Method
DID Method-Specific Identifier

Figure 5.4.0-1: DID Structure (from W3C Decentralized Identifiers [i.10])

The term method is used for different approaches and/or implementations.

Different methods can have different approaches, based on distributed ledgers or where DID documents are stored on specialized sites (e.g. GitHub).

The W3C DID specification registries, is a registry maintained at W3C by the working group for DID methods, DID document properties, key Types, service types.

At the moment 112 DID Methods are currently registered.

The DID method-specific identifier is a unique identifier generated and defined by the DID method specification.

It defines how to read and write a DID and its DID document.

5.4.1 DID architecture

The major components of Decentralized Identifier architecture are shown below.

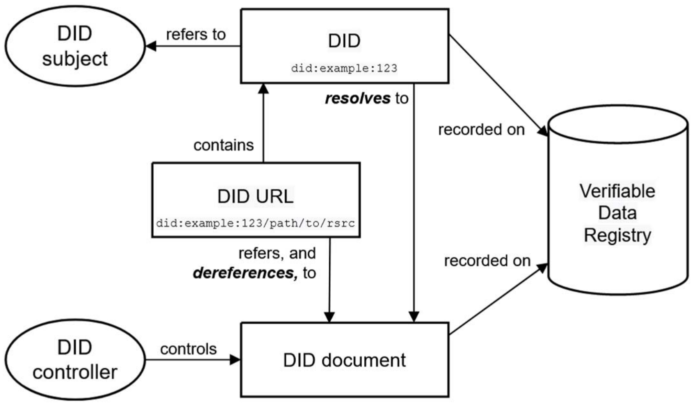


Figure 5.4.1-1: DID architecture and relationship of the basic components (from W3C Decentralized Identifiers [i.10])

A DID document is an information resource on the web, containing a reference to the DID controller, the entity who can make changes on the DID document.

The DID controller may be the DID subject, the entity referred to by the DID.

The DID document also contains cryptographic data related to the DID subject like cryptographic keys and their utilization purposes.

A DID document may also contain a timestamp for audit history and a signature for integrity.

Every DID resolves to a specific DID document for example:

```
did:example:123456789abcdefghi
```

resolves to:

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

The DID document can only be serialized in JSON, JSON-LD and CBOR at the moment.

A DID document may also contain some other types of data (such as `alsoKnownAs`) related to the subject.

DID and DID document may be also used as a decentralized cryptographic keychain for various cryptographic applications.

DIDs are powerful identifiers by themselves, but they can also be used as the basis for constructing more advanced URLs rooted in a DID.

A DID URL enables an "identifier space" for additional resources associated with the DID. For example:

```
did:example:1234/
did:example:1234#keys-1
```

Decentralized identifiers represent one of the two pillars (together with verifiable credentials) of self-sovereign identity, a digital identity model that gives to the subject the control of its identifiers and of data sharing.

5.5 DIDs for juridical persons

5.5.1 Introduction

When the work on W3C Verifiable Credentials [i.11] and DIDs (Decentralized Identifiers) [i.10] started, there was a very strong focus on natural persons (versus juridical persons) and solving some severe problems that citizens have with the current identity systems. Actually, both concepts were somewhat conflated and mixed together.

However, when many initiatives explored the practical application to the real world of Verifiable Credentials, it became clear that those concepts could be used together but were really independent, so two specifications were created. As the current version of the W3C Verifiable Credentials Data Model (v1.1) [i.11] states (emphasis added):

> *"As of this publication, DIDs are a new type of identifier that* ***are not necessary for verifiable credentials to be useful****. Specifically, verifiable credentials do not depend on DIDs and DIDs do not depend on verifiable credentials. However, it is expected that many verifiable credentials will use DIDs and that software libraries implementing this specification will probably need to resolve DIDs. DID-based URLs are used for expressing identifiers associated with subjects, issuers, holders, credential status lists, cryptographic keys, and other machine-readable information associated with a verifiable credential."*

Based on the feedback from exploration of real use cases, the W3C Verifiable Credentials Working Group evolved the specification to enable wider coverage of many interesting use cases and taking into account compliance to current country/regional regulation and also industry-specific regulation.

However, the DID specification was not evolved in the same way, keeping its legacy of strong focus on natural persons and trying to avoid any type of central entity, including those whose mission is the protection of the citizen.

This lack of real-world feedback can be further appreciated in the fact that both W3C specifications are contradictory. The formal definition of DID [i.10] states that it is a "*globally unique* ***persistent*** *identifier*". However, the more mature Verifiable Credentials specification [i.11] says:

> *"If strong anti-correlation properties are a requirement in a verifiable credentials system, it is strongly advised that identifiers are either:*
>
> - ***Bound to a single origin***
> - ***Single-use***
> - ***Not used at all****, but instead replaced by short-lived, single-use bearer tokens."*

In some sense, the current (but outdated) DID specification shares the same spirit and ideology of Bitcoin and other cryptocurrencies, so a literal interpretation of the document makes it very difficult to enforce EU regulations like GDPR or Consumer Protection directives (for natural persons), or compliance to most legal requirements in the EU (for juridical persons).

This clause describes a real-world interpretation of DIDs which is better suited to the EU environment and at the same time provides the same advantages than the original DID in solving the major problems that citizens have with current identity systems.

The focus here is on legal persons, because DIDs for natural persons will probably be handled in the foreseen eIDAS2 and EU Digital Identity Wallet regulations.

5.5.2 Requirements on a DID for juridical persons

According to the current DID specification [i.10], it is a new type of identifier with 4 essential characteristics:

- **decentralized:** there should be no central issuing agency;
- **persistent:** the identifier should be inherently persistent, not requiring the continued operation of an underling organization;
- **cryptographically verifiable:** it should be possible to prove control of the identifier cryptographically;
- **resolvable:** it should be possible to discover metadata about the identifier.

For juridical persons in the EU and many other regions of the world, there is a fundamental problem with the first characteristic. Actually, that characteristic is undesirable in any advanced society that shares the EU values and where the citizen is the most important thing.

The EU regulatory environment requires that identities (and identifiers) of any juridical person be completely public and subject to public scrutiny, whether from regulators, consumer organizations, industry watchdogs or any other interested party. Some examples:

- Any of the 30 million businesses in the EU is required by law to register with the appropriate national or international body before being able to engage in any relevant activity. During the process, an identifier is assigned to the business. The registration process and the way to obtain the identifier varies depending on the industry regulation (e.g. banking, telco, health, etc.) and other factors.
- The identities of businesses are public and anybody can access all related information from the relevant registries. Even if in some EU countries access is not free, it is public. Information includes many details about the business, including identifier, ownership and place of establishment.
- Businesses are required to include those identifiers in any relevant transaction, whether they are electronic or offline. When citizens buy any product or service, they have the right to obtain documentation about the purchase including the identifier of the company.
- Any relevant process in the real economy (agrifood, health, manufacturing, transportation, etc.) imposes requirements on traceability of the supply chain, related among other with safety, health and consumer protection and the ability to react properly to emergencies. Information recorded and custodied by all businesses participating in the chain has to use the legally valid identifiers of each business.

From the above it follows that in the case of juridical persons, requiring an additional identifier that has no legal validity and is not recognized by any authority in the EU does not add any value and does not solve any real problem that businesses have in the EU.

Instead, it makes more sense to reuse the existing public identifiers which are compulsory for juridical persons and generate the DID deterministically from them.

5.6 Status of the Specifications

At the time of publication, there existed 103 experimental DID Method specifications, 32 experimental DID Method driver implementations, a test suite that determines whether or not a given implementation is conformant with the main W3C DID specification [i.10] and 46 implementations submitted to the conformance test suite.

6 Verifiable Credentials and Authenticity

6.1 Goals

Pursuing what has already been previously declared, in this clause the goal is the implementation of a standard that guarantees the trustworthiness, privacy and authenticity not only of all subjects involved, but of all their attributes and the overhaul of a NGSI-LD Entity during its life-cycle.

6.2 Scenarios and Use Cases

A typical scenario is the development of a digital credential for a person, e.g. a digital passport, or for a non-human entity, such as a picture taken by a camera bound with all information about its provenance.

This is also the standard case of an NGSI-LD Entity representing the person or the picture, which needs to have verifiable NGSI-LD Attributes representing the provenance information.

6.3 Architectures and Existing Specifications

6.3.1 W3C Verifiable Credentials

6.3.1.0 Introduction

W3C Verifiable Credentials specification [i.11] is developed by the W3C Verifiable Credentials Group which is a W3C (World Wide Web Consortium) working group, whose mission is to provide a data model, in order to express and exchange verifiable credentials on the Web.

Verifiable credentials are data objects consisting of claims made by the issuer attesting information about a subject that can be cryptographically verified.

They can represent physical credentials (passports, driver licenses) or entities that have no physical equivalent.

6.3.1.1 Advantages

The main advantage in Verifiable credentials, especially in comparison to their physical counterpart, is that they are digitally signed, which makes them tamper-resistant and instantaneously cryptographically verifiable.

Verifiable credentials are machine-readable, through the use of linked-data system.

Verifiable credentials are strongly authenticated and strictly bound to the subjects involved in it, especially with the DID implementation.

The important thing, however, is that Verifiable credentials are not tied to a specific blockchain technology, it can be used across any distributed ledger but it can also be used in traditional centralized databases.

Credentials can be self-organized and collected in a, so called, Verifiable presentation and shared with a verifier encoded in such a way that authorship of the data can be trusted after a process of cryptographic verification.

Another important property of Verifiable credentials is the selective disclosure, in association with Zero-knowledge proof implementation.

It is possible to create a subset of attributes from a credential and present them without the invalidation of the original credential.

6.3.1.2 Lifecycle

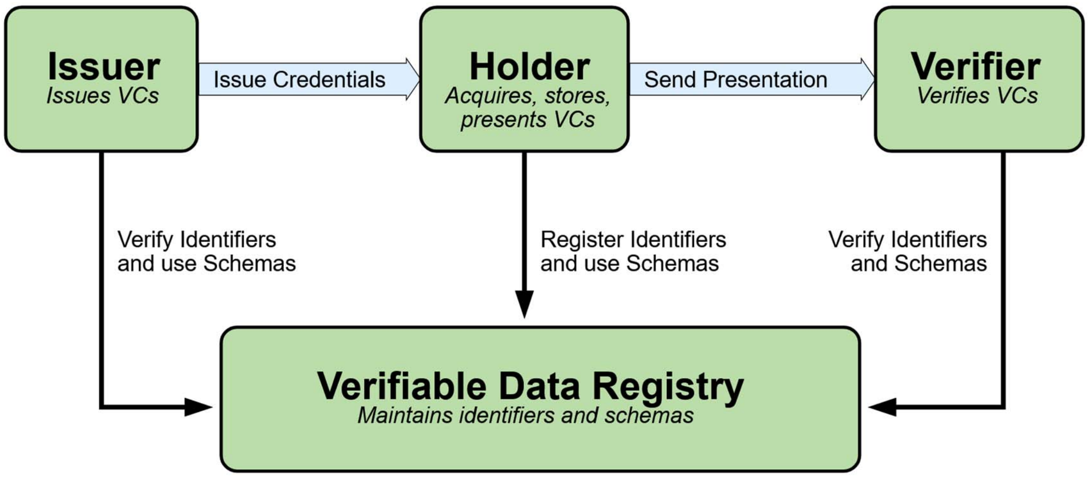

Issuer
Issues VCs
Issue Credentials
Holder
Acquires, stores, presents VCs
Send Presentation
Verifier
Verifies VCs
Verify Identifiers and use Schemas
Register Identifiers and use Schemas
Verify Identifiers and Schemas
Verifiable Data Registry
Maintains identifiers and schemas

Figure 6.3.1.2-1: Verifiable Credentials ecosystem (from W3C Verifiable Credentials [i.11])

The Lifecycle of Verifiable Credentials expressed in the Verifiable Credentials Data Model illustrates all actors involved.

- **Issuer:** Who creates and issues credentials
- **Holder:** Who receives, holds and shares credentials
- **Verifier:** Who verifies proofs from holders

In this ecosystem Verifier and Issuer do not need to interact directly, preventing time and cost expansion of verification.

The implementation of the trust model is possible through the Verifiable Data Registry, a centralized or decentralized database.

It mediates the creation and verification of identifiers, keys, schemas, and revocation registries.

All actors trust the verifiable data registry to be tamper-evident and to be a correct record of which data is controlled by which entities.

The lifecycle diagram also determines the "triangle of trust": since both the issuer and the verifier trust the reliability of the Verifiable Data Registry, the verifier trusts the issuer; hence the verifier trusts the credential it is issuing for the holder, that the holder is presenting. There needs be no direct interaction between the verifier and the issuer, in order for the verification process to be in place.

6.3.1.3 Credential Structure

The core of a verifiable credential structure is the expression of claims.

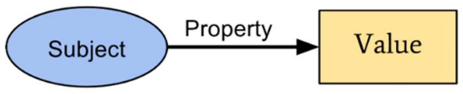


Figure 6.3.1.3-1: Claim structure (from W3C Verifiable Credentials [i.11])

A claim is a statement about a subject and is expressed using the subject-property-value relationship.

Different claims can be merged to express a graph of information about a subject.

A set of claims made by the same entity with metadata creates a Credential.

A credential digitally signed and bound with a proof is a Verifiable credential.

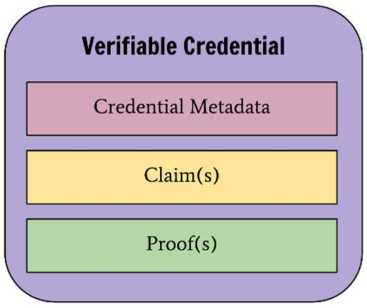


Figure 6.3.1.3-2: Verifiable Credentials structure (from W3C Verifiable Credentials [i.11])

A verifiable credential is normally composed of at least two information graphs.

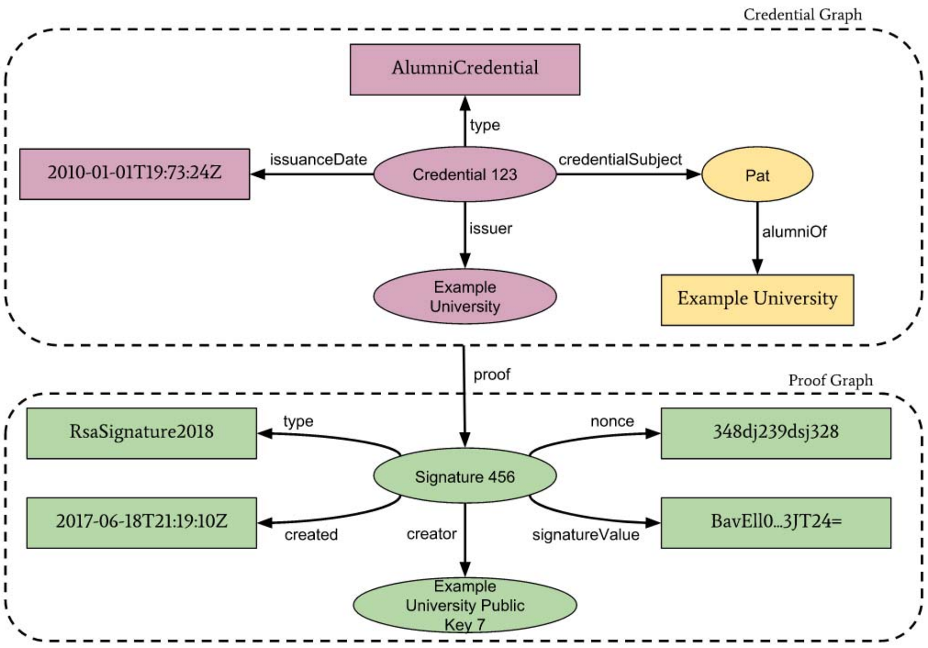


Figure 6.3.1.3-3: Verifiable Credentials graphs (from W3C Verifiable Credentials [i.11])

The first graph is the credential graph, which expresses credentials metadata and claims.

The second graph is the credential proof graph which expresses the digital proof.

Expressing an example of verifiable credential in JSON-LD format:

```
{
  "@context": ["https://www.w3.org/2018/credentials/v1",
               "https://www.w3.org/2018/credentials/examples/v1"],
  "id": "http://example.edu/credentials/1872",
  "type": ["VerifiableCredential", "AlumniCredential"],
  "issuer": "https://example.edu/issuers/565049",
  "issuanceDate": "2010-01-01T19:23:24Z",
  "credentialSubject": {
    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    "alumniOf": {
      "id": "did:example:c276e12ec21ebfeb1f712ebc6f1",
      "name": {
        "value": "Example University",
        "lang": "en"
      }
    }
  },
  "proof": {
   "type": "RsaSignature2018",
   "created": "2017-06-18T21:19:10Z",
   "proofPurpose": "assertionMethod",
   "verificationMethod": "https://example.edu/issuers/565049…",
   "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYj…"
  }
}
```

The credential metadata attributes are:

- `id`: an optional identifier for the credential that can be an HTTP-based URL or a DID.
- `type`: an array made by `VerifiableCredential` value and optional credential subtypes.
- `issuer`: an identifier for the issuing entity.
- `issuanceDate`: a date-time string representing the date and time the credential becomes valid.

The `credentialSubject` property contains information about one or more subjects and claims.

All statements about a subject and the subject id itself are nested in the `credentialSubject`.

The `id` property of a subject is optional.

The proof graph can be expressed following the W3C data integrity specification [i.1], as explained in clause 4.

Verifiable credentials are created in the left branch of the life cycle. They are signed by an issuer and stored by a holder, an entity that holds credentials and may be the subject itself.

6.3.1.4 Presentation Structure

On the right branch of the lifecycle diagram there is the usage of credentials in order to verify claims.

This operation is performed through the creation of a Verifiable Presentation by a holder.

A verifiable presentation is a document expressing data coming from one or more verifiable credentials.

Data in presentations is often about the same subject, but might have been issued by multiple issuers.

Verifiable presentations in their abstract form are expressed by a list of verifiable credentials, presentation metadata all signed with a proof.

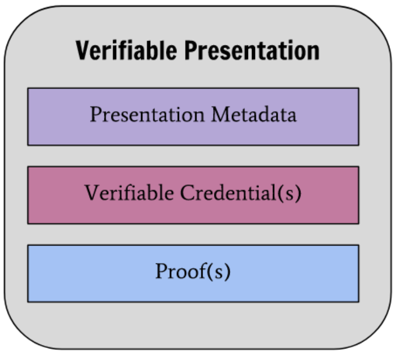


Figure 6.3.1.4-1: Verifiable Presentation structure (from W3C Verifiable Credentials [i.11])

A deeper look at the constitution of a verifiable presentation shows a graph structure, similar to the credential structure, nested inside a Presentation graph.

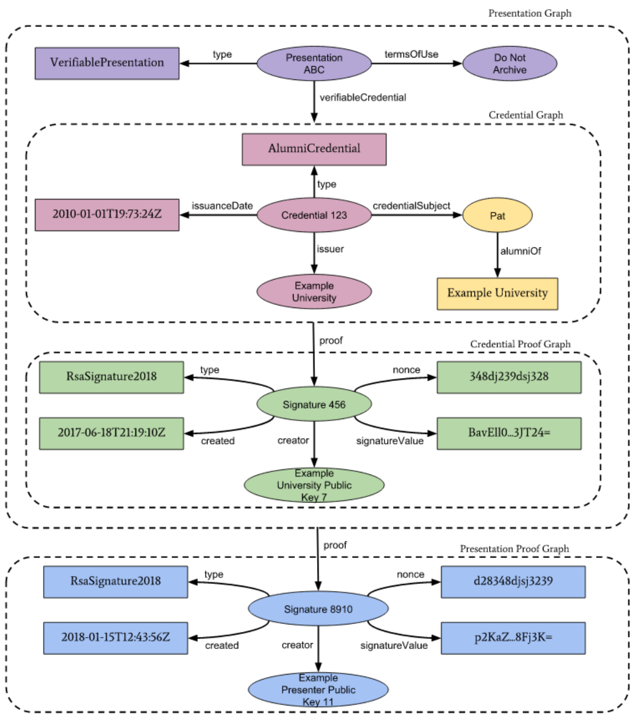


Figure 6.3.1.4-2: Verifiable Presentation graphs (from W3C Verifiable Credentials [i.11])

The presentation graph contains presentation metadata and the list of credentials.

Presentation metadata attributes are:

- `type`: an array made by `VerifiablePresentation` value and optional presentation subtypes.
- `verifiableCredential`*:* an array containing all verifiable credentials.

The presentation proof graph contains a proof signed by the holder, and as for verifiable credentials, it can be expressed following the W3C data integrity specification [i.1].

Here is a simple verifiable presentation example:

```
{
  "@context": ["https://www.w3.org/2018/credentials/v1",
               "https://www.w3.org/2018/credentials/examples/v1"],
  "type": "VerifiablePresentation",
  "verifiableCredential": [{
   "@context": ["https://www.w3.org/2018/credentials/v1",
                "https://www.w3.org/2018/credentials/examples/v1"],
   "id": "http://example.edu/credentials/1872",
   "type": ["VerifiableCredential", "AlumniCredential"],
   "issuer": "https://example.edu/issuers/565049",
   "issuanceDate": "2010-01-01T19:23:24Z",
   "credentialSubject": {
     "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
     "alumniOf": {
       "id": "did:example:c276e12ec21ebfeb1f712ebc6f1",
       "name": {
         "value": "Example University",
         "lang": "en"
       }
     }
   },
   "proof": {
     "type": "RsaSignature2018",
     "created": "2017-06-18T21:19:10Z",
     "proofPurpose": "assertionMethod",
     "verificationMethod": "https://example.edu/issuers/...",
     "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImN…"
   }
  }],
  "proof": {
   "type": "RsaSignature2018",
   "created": "2018-09-14T21:19:10Z",
   "proofPurpose": "authentication",
   "verificationMethod": "did:example:ebfeb1f712ebc6f1c2…",
   "challenge": "1f44d55f-f161-4938-a659-f8026467f126",
   "domain": "4jt78h47fh47",
   "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOls…"
  }
}
```

Following what is already discussed in clause 4, it is also possible to implement a Zero Knowledge Proof mechanism.

Implementing a zero-knowledge proof, the holder can create derived credentials collected in a presentation in order to selectively disclose some information.

In order to do it, the issuer of the original Verifiable credential uses a signature suite that allows the derivation of a proof.

This allows the holder to present the information to a verifier in a privacy-enhancing manner, and prove the validity of the issuer's signature without revealing the values that were signed.

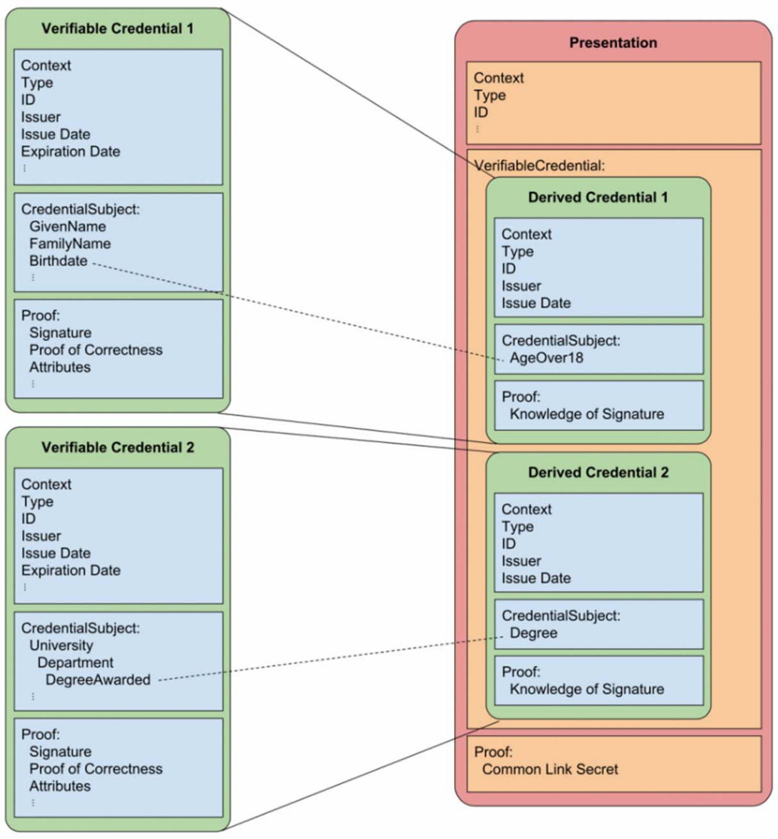


Figure 6.3.1.4-3: Selective Disclosure mechanism (from W3C Verifiable Credentials [i.11])

Closing the Credentials lifecycle, a holder can delete or transfer credentials, create presentations and present them to a verifier.

The Verifier cryptographically verifies the issuer's signatures, identifiers and schemas, checking in the verifiable data registry, without direct interactions with issuers.

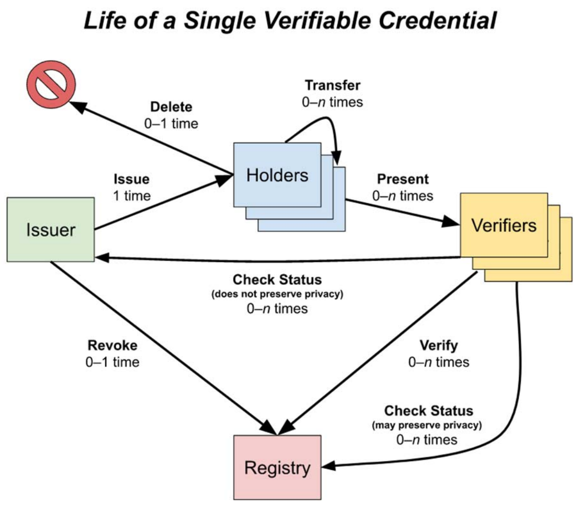


Figure 6.3.1.4-4: Verifiable Credentials lifecycle (from W3C Verifiable Credentials [i.11])

All entities trust the verifiable data registry to be tamper-evident and to be a correct record of which data is controlled by which entities.

Example verifiable data registries include trusted databases, decentralized databases, government ID databases, and distributed ledgers.

6.3.2 C2PA

6.3.2.0 Introduction

C2PA or Coalition for content Provenance and Authenticity is a Joint Development Foundation, founded in 2019 by Adobe with New York Times and Twitter, whose mission is to provide provenance and history for digital media, providing a tool for creators to claim authorship while empowering consumers to make informed decisions about what to trust.

C2PA, now formed by many important industry partners, is developing an open technical standard providing publishers, creators, and consumers the ability to trace the origin of different types of media.

Differently from W3C Verifiable Credentials, the C2PA specification [i.12] is a standard only for digital assets.

6.3.2.1 Architecture

Distinguishing the main actors in its model:

- **Content creators**, who wish to assert information about content they have produced in a way that can be trusted.
- **Content publishers**, who wish to have better information on which to make decisions about what content to trust.
- **Content consumers**, who wish to understand the process by which the content was created.
- **Vendors and implementers**, who wish to build software or hardware tools to create, persist, exchange, or consume C2PA provenance data.

In C2PA every statement about a digital asset, like creation or edit information, is called Assertion: they are bound to content, and they make up the provenance of a given asset and represent a series of trust signals.

All Assertions are grouped together and they form the Claim, which will be digitally signed by a hardware or software component called Claim generator, forming the Claim Signature.

Assertions, Claim and Claim Signature form the so-called C2PA Manifest.

C2PA also suggests the possibility to add W3C Verifiable Credentials of subjects involved in the manifest, in order to improve trustworthiness.

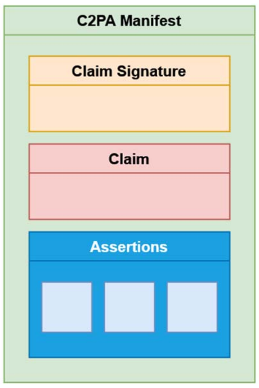


Figure 6.3.2.1-1: C2PA Manifest structure (from C2PA Technical Specifications [i.12])

Every time a new action is made on the digital asset a new manifest is generated and the set of all C2PA manifests are stored in the Manifest Store.

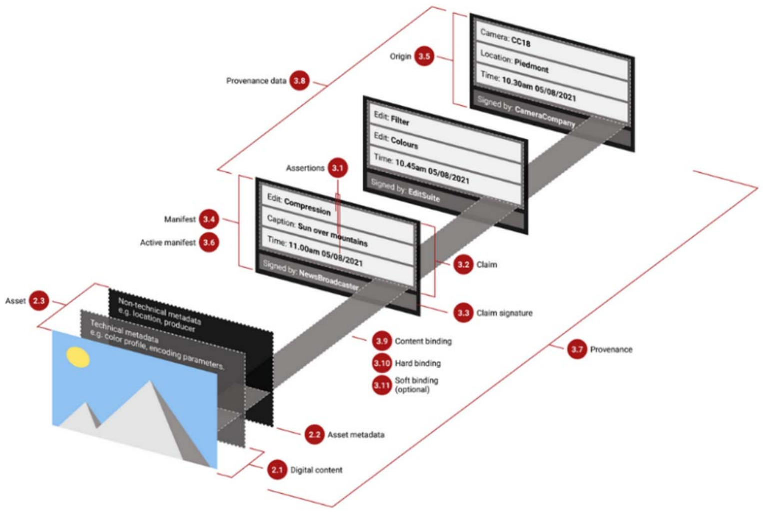


Figure 6.3.2.1-2: C2PA Provenance scheme (from C2PA Technical Specifications [i.12])

The last manifest is called Active Manifest and is the one with the set of content bindings that are able to be validated.

The authenticity of the digital asset is granted by the binding of manifests to the asset.

C2PA supports two types of bindings: hard bindings and soft bindings.

A hard binding (also known as a cryptographic binding) enables the validator to ensure that a manifest belongs to a specific asset that has not been modified.

The simplest type of hard binding that can be used to detect tampering is a cryptographic hashing algorithm, making Data Hash Assertions containing the hash of the asset.

A soft binding is computed from the digital content of an asset, rather than its raw bits. A soft binding is useful for identifying derived assets and asset renditions.

Soft bindings are described using soft binding assertions and a typical example could be a watermark or a fingerprint.

Following the manifest creation process as shown in Figure 6.3.2.1-3:

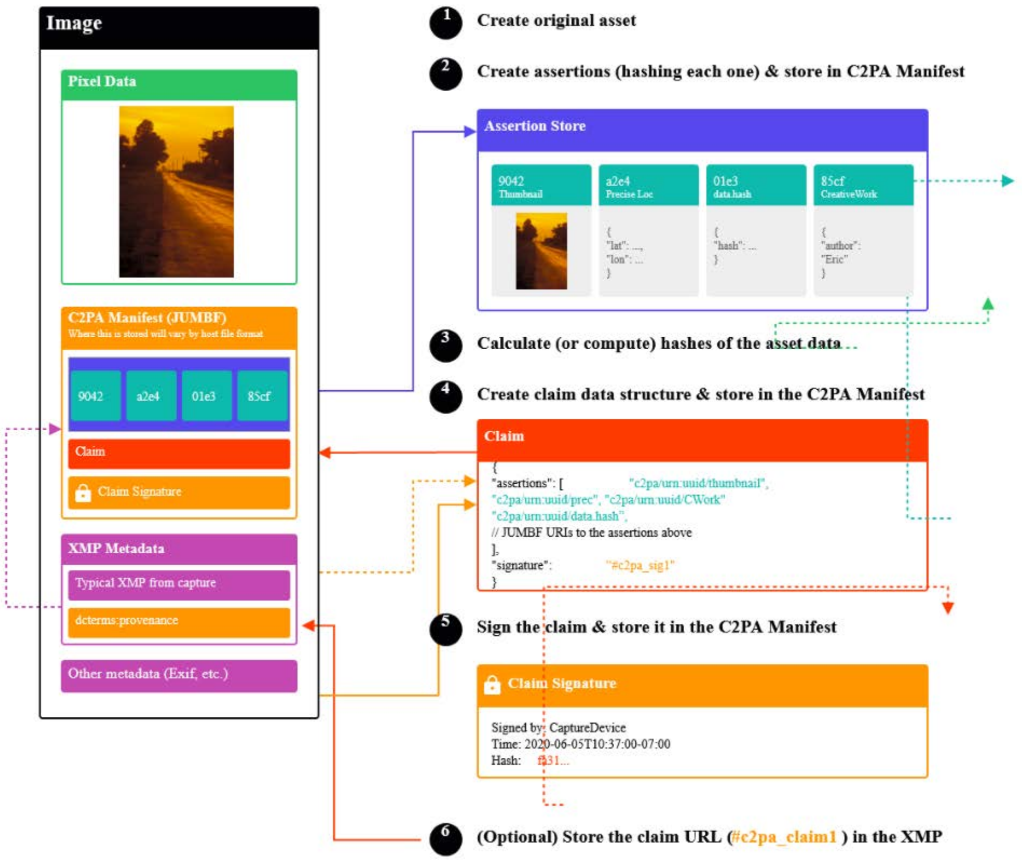


Image
Pixel Data
C2PA Manifest (JUMBF)
Where this is stored will vary by host file format
9042
a2e4
01e3
85cf
Claim
Claim Signature
XMP Metadata
Typical XMP from capture
dcterms:provenance
Other metadata (Exif, etc.)
1 Create original asset
2 Create assertions (hashing each one) & store in C2PA Manifest
Assertion Store
9042 Thumbnail
a2e4 Precise Loc
{ "lat": ..., "lon": ... }
01e3 data.hash
{ "hash": ... }
85cf CreativeWork
{ "author": "Eric" }
3 Calculate (or compute) hashes of the asset data
4 Create claim data structure & store in the C2PA Manifest
Claim
{ "assertions": ["c2pa/urn:uuid/thumbnail", "c2pa/urn:uuid/prec", "c2pa/urn:uuid/CWork" "c2pa/urn:uuid/data.hash", // JUMBF URIs to the assertions above], "signature": "#c2pa_sig1" }
5 Sign the claim & store it in the C2PA Manifest
Claim Signature
Signed by: CaptureDevice
Time: 2020-06-05T10:37:00-07:00
Hash: f431...
6 (Optional) Store the claim URL (#c2pa_claim1) in the XMP

Figure 6.3.2.1-3: C2PA Manifest creation process (from C2PA Technical Specifications [i.12])

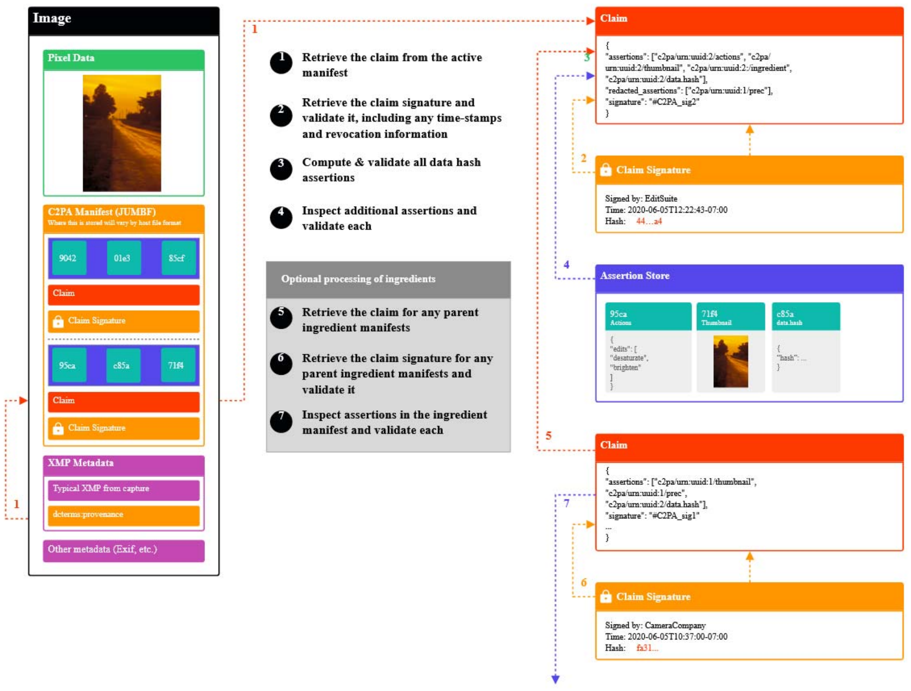
All assertions are created and labelled according to standard assertion types, encoded in the following possible structures:

- CBOR
- JSON
- JSON-LD
- JPEG Universal Metadata Box Format (JUMBF)
- ISO Base Media File Format (BMFF)

After that, they are hashed and stored in the Assertion Store, then the Claim is created and digitally signed.

Everything is stored, first in the Manifest, then in the Manifest store.

On the other side, following the validation process as shown in Figure 6.3.2.1-4:



1 Retrieve the claim from the active manifest

2 Retrieve the claim signature and validate it, including any time-stamps and revocation information

3 Compute & validate all data hash assertions

4 Inspect additional assertions and validate each

5 Retrieve the claim for any parent ingredient manifests

6 Retrieve the claim signature for any parent ingredient manifests and validate it

7 Inspect assertions in the ingredient manifest and validate each

```
{
"assertions": ["c2pa/urn:uuid:2/actions", "c2pa/
urn:uuid:2/thumbnail", "c2pa/urn:uuid:2:/ingredient",
"c2pa/urn:uuid:2/data.hash"],
"redacted_assertions": ["c2pa/urn:uuid:1/prec"],
"signature": "#C2PA_sig2"
}
```

Signed by: EditSuite
Time: 2020-06-05T12:22:43-07:00
Hash: 44…a4

```
{
"edits": [
"desaturate",
"brighten"
]
}
```

```
{
"hash": ...
}
```

```
{
"assertions": ["c2pa/urn:uuid:1/thumbnail",
"c2pa/urn:uuid:1/prec",
"c2pa/urn:uuid:2/data.hash"],
"signature": "#C2PA_sig1"
...
}
```

Signed by: CameraCompany
Time: 2020-06-05T10:37:00-07:00
Hash: fa31...

Figure 6.3.2.1-4: C2PA Manifest validation process (from C2PA Technical Specifications [i.12])

The active manifest is retrieved from the Manifest store, then the claim signature is validated, subsequently all data hashes of all assertions are computed and validated.

6.3.2.2 Trust Model

Following these two main processes the trust model is so defined:

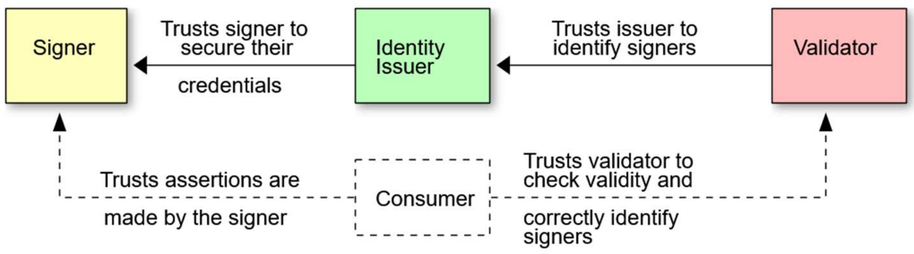

Signer
Trusts signer to secure their credentials
Identity Issuer
Trusts issuer to identify signers
Validator
Trusts assertions are made by the signer
Consumer
Trusts validator to check validity and correctly identify signers

Figure 6.3.2.2-1: C2PA Trust Model (from C2PA Technical Specifications [i.12])

The three entities (yellow, green and red), shown in Figure 6.3.2.2-1, form the trust model, which is concerned with trust in a signer's identity.

The identity is the means by which a cryptographic signing key is associated with an actor.

The signer's identity is ensured by credentials provided in the form of x.509 certificates.

As in the PKI system, the validator trusts the Certificate authority, who grants the signer's identity.

In this model the Consumer uses the identity of the signer, along with other trust signals, to decide whether the assertions made about an asset are true.

6.4 Status of the Specifications

W3C

W3C Verifiable Credentials [i.11] is a W3C standard and it has been adopted and implemented in many different areas, for example the Turkish Ministry of Foreign Affairs, in collaboration with the United Nations Development Programme, is piloting a Self-Sovereign Identity solution to optimize the process of issuing Work Permits to refugees.

A JavaScript library for issuing and verifying Verifiable Credentials, called Verifiable Credentials JS Library, has been implemented by the digitalbazaar group (https://github.com/digitalbazaar/vc-js).

C2PA

PyC2PA is a Python® implementation of C2PA specification (https://github.com/numbersprotocol/pyc2pa).

7 Trust Framework and the Verifiable Data Registry

7.1 Introduction

The W3C Verifiable Credentials ecosystem includes a Verifiable Data Registry in order to implement the trust model. The document [i.11] mentions examples including trusted databases, decentralized databases, government ID databases, and distributed ledgers. The trust model described in the W3C document is very general, so in this clause some specific requirements will be explored for the Verifiable Data Registry, imposed by some use cases and ecosystems found in NGSI-LD implementations.

Figure 7.1-1 describes a generalized ecosystem of producers and consumers of information, denoted in the diagram as Producers and End Users.

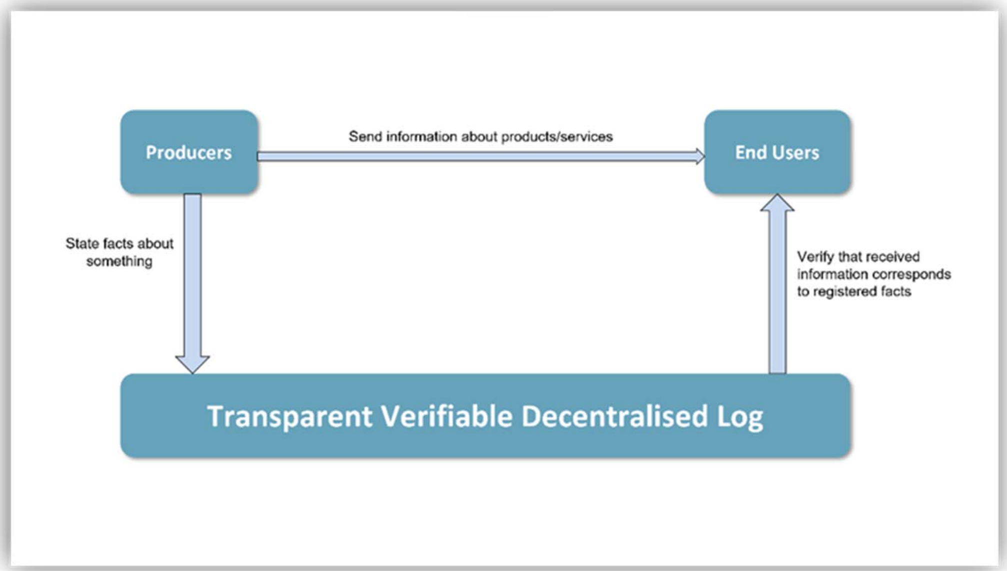

Producers
Send information about products/services
End Users
State facts about something
Verify that received information corresponds to registered facts
Transparent Verifiable Decentralised Log

Figure 7.1-1: Information ecosystem

Verifiable Credentials can be used in the path of information described by the arrow pointing directly from Producers to Consumers. The credentials alone prove that nobody has tampered with its contents since it was issued, because the credential is digitally signed by the issuer.

But in a wide and open ecosystem, it is difficult for the Consumer:

- To know the real-world identity of the issuer, because it is difficult to associate the public key that verifies the signature of the credential with the real-world identity of the issuer.

- To know that the issuer is really entitled to issue that specific type of credential. For example, an official Diploma can only be issued by a University, not by any other entity. Furthermore, a given University may not have the permission to issue some types of Diplomas, as authorized by the corresponding authority in the country.
- To prove that the credential was really issued no later than a given time, especially no later than the field "Issued at" included inside the credential.

This is the reason why a Verifiable Data Registry is needed, named Transparent Verifiable Decentralized Log in the diagram. The reason for that name will become apparent when some of the requirements for such component will be discussed.

7.2 Some requirements

7.2.0 Introduction

In a wide, open and inclusive ecosystem at the EU level, the Verifiable Data registry cannot be implemented by a centralized database operated and controlled by a single entity. The system has the following requirements.

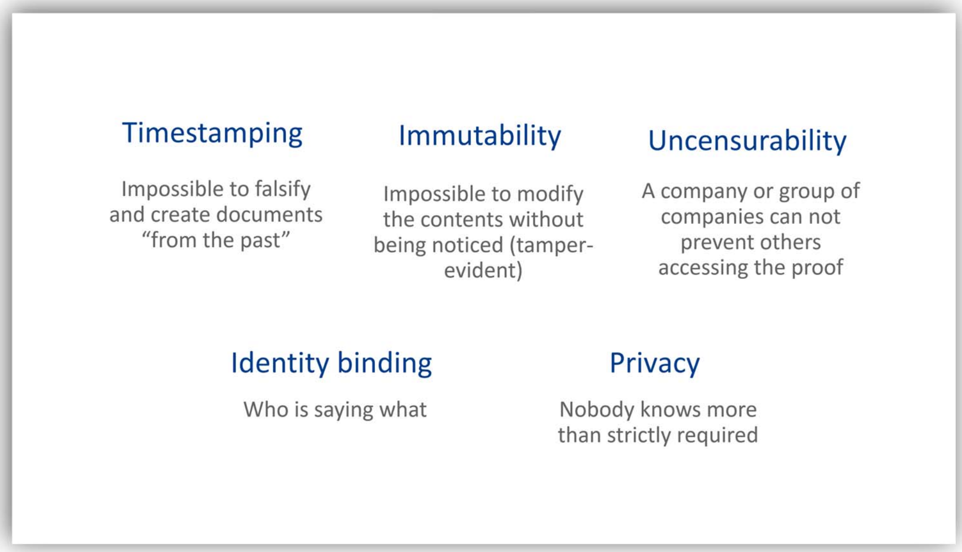

Timestamping
Impossible to falsify and create documents “from the past”
Immutability
Impossible to modify the contents without being noticed (tamper-evident)
Uncensurability
A company or group of companies can not prevent others accessing the proof
Identity binding
Who is saying what
Privacy
Nobody knows more than strictly required

Figure 7.2.0-1: Verifiable Data registry Requirements

7.2.1 Timestamping

Timestamping is needed to make impossible to create documents "from the past". For example, nobody should be able to create a Diploma today and pretend that it was issued several years ago. Today, it is not a problem to issue a credential attesting something from the past several years ago, as long as the issuance date in the credential is not backdated. The claims inside the credential can attest whatever is required, for example that a person received a given Diploma several years ago.

To achieve this requirement, timestamping is required. There are already established services for timestamping from (Q)TSPs (Trust Service Providers). But for many common services using Verifiable Credentials an easier and more generally available service could be provided by a Verifiable Data Registry.

NOTE: The term "notarization" is commonly used for this action, but it is wrong, because the term is coming from Anglo-Saxon cultures where notaries are very different from the Latin-Germanic notary functions in the EU and many other countries in the world. The term "timestamping" will be used.

The date of timestamping can be greater than the date in the field "Issued at" included inside the credential. For example, the credential is created and signed at one time, but timestamped the next day (maybe to batch the operation with other credentials). The real requirement is that nobody can create a credential and timestamp as if it happened in the past. In other words, nobody can create credentials from the past. The verifiers have to check that the field inside the credential "Issued at" is not later than the timestamp (at least by a small leeway to account for clock synchronization differences).

Also note that many credentials may not require timestamping, avoiding the overhead of the registration process. It all depends on the type of credential, the intended usage of the credential and the level of risk assumed. In many use cases the only thing that the verifier requires is that the holder of the credential can prove that at the time of usage of the credential, the credential was issued by the Issuer entity, and exactly when the credential was issued is not relevant. Obviously, this does not require timestamping, because if the holder can present a credential when performing login, it can do so only if the credential was issued before.

7.2.2 Immutability

It should be impossible to modify the data without being noticed. Even though the word "immutable" has been popularized with blockchain technology, in the real world there is nothing absolutely immutable but just more or less difficult to modify. What is really needed can be called "tamper-evident".

Digital signatures already provide the tamper-evidence property to the contents inside of the credential, but real-world use cases typically require some additional information (like metadata, identification of issuer, etc.) outside of the credential and registered in the Verifiable Data Registry. This requirement specifies the tamper-evident properties of the registry.

7.2.3 Un-censurability

In a wide, open and inclusive ecosystem at the EU level, it is not enough having a tamper-evident repository. It is required that no central entity or group of entities can prevent other participants from accessing data that was written in the repository and that can be critical to get the proofs needed to use the Verifiable Credentials.

If one entity could "censor" or prevent access to critical information to other entities, then it could perform a form of Denial of Service (DoS) attack, by preventing access to the information in the registry.

In other words, a registry is needed, such that once an entity has written something in it, nobody can avoid other participants from accessing the data. Coupled with the tamper-evidence property, it makes the registry very powerful.

7.2.4 Identity binding

Knowing who has said what is needed. The system has to provide a reliable and trusted way to link real-world-identities with the identifiers and public keys used in Verifiable Credentials and also with other data items registered in the Verifiable Data Registry.

Standard PKI infrastructure and x.509 certificates are not enough. As mentioned in clause 7.1, real use cases need to know additional information about the issuer. For example, in the case of Diplomas if the issuer is a public University, and if was authorized to issue a given type of diploma on a given year.

This information about the identity of a juridical person is registered in the Verifiable Data Registry in what are called Trusted Lists. But Trusted Lists are not limited to identity, and they are required for several other types of information.

Using as an example the EU Digital COVID Certificate, there are several Trusted Lists:

- List of health entities authorized to issue certificates, with their public key. In each country, Public Health Authority authorizes the corresponding issuers. There is no central entity authorizing issuers, the decision is delegated to the countries.

- List of accepted vaccines or test manufacturers.

- List of valid schemas for the data modelling.

7.2.5 Privacy

The fact that the registry is public does not necessarily mean that everybody knows everything about others. The global system should be designed to comply with the principle that nobody knows more than strictly required.

In other words, no data should be written to the public registry if it is not required and goes against this principle. What this means in practice is that no personal data about a natural person should ever be registered in the Verifiable Data Registry (or in any publicly accessible repository, in general).

And encryption does not solve the problem: encrypted personal data is considered personal data by the data protection authorities in the EU. A hash of personal data is in general also considered personal data, unless a Privacy Impact Assessment (PIA) has been performed in the system and has proven that the actual usage complies with the GDPR.

Actually, as mentioned before in clause 5.1, the strong legal requirements in the EU regarding GDPR and Consumer Protection imply that natural persons and juridical persons should be handled by the system as independent entities. The general W3C Verifiable Credential diagram could be modified as in Figure 7.2.5-1.

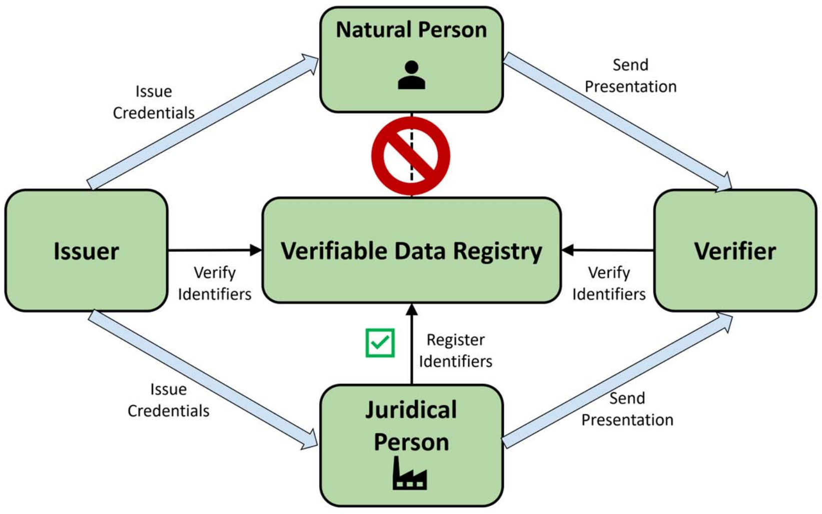


Figure 7.2.5-1: Modified W3C Verifiable Credential diagram

For natural persons, no information related to personal data should be ever written to the registry. This includes identifiers associated to the identity of the person (like DIDs), encrypted personal data, or hashes of personal data that could be used for correlation and tracking when combined with data existing outside of the registry.

For juridical persons the registry is used to register all public information about the entity, including DIDs, DID Documents and any additional information that may be useful in the ecosystem to increment the level of trust.

However, the system should enable privacy of commercial data that is not public and should not be available to competitors. This includes volumes of transactional activity, for example.

7.2.6 Conclusion

Summarizing the requirements described above, a wide, open and inclusive ecosystem in the EU requires a Verifiable Data Registry which:

- Provides timestamping for credentials and other facts asserted by participants
- Is immutable (tamper-evident) for all information that is written (including timestamping data)
- Is un-censorable (transparent), which means that anybody can read all information that was ever written
- Facilitates trusted ID binding
- Provides all the above with high levels of privacy

The timestamping and immutability requirements point to an append-only log system for the registry.

Trusted ID binding should be implemented by a trusted onboarding process that verifies the real-world identities of juridical persons and registers the information in the registry. Privacy can be achieved by registering only the minimal essential information required for achieving the desired level of trust.

Taking all the requirements together, it is possible to see that the required registry could be more properly called a Transparent Verifiable Decentralized Log.

The word "Decentralized" appears because it was controlled by a single entity it would be very difficult to ensure the un-censurability property.

There is not any universally accepted definition of "decentralization", but for the purposes of the registry Figure 7.2.6-1 will be used.

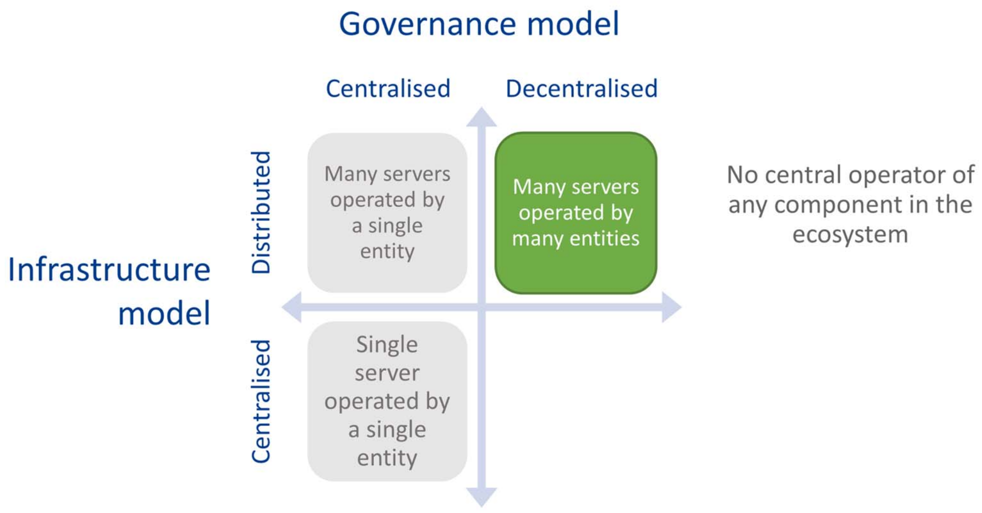

Governance model
Centralised
Decentralised
Infrastructure model
Distributed
Centralised
Many servers operated by a single entity
Many servers operated by many entities
No central operator of any component in the ecosystem
Single server operated by a single entity

Figure 7.2.6-1: Decentralization model

The registry could be implemented by either one centralized system or a distributed one with several machines in different geographies. This is the technical dimension, and it is clear from the requirements that a distributed system is needed.

But that is not enough: if all the machines are operated and controlled by the same entity, then that entity has too much power in the ecosystem and for example the property of un-censurability and transparency are difficult to enforce.

What is required is a distributed and Decentralized system, where the machines composing the system are not operated by a single entity. Actually, this principle should apply to all components of the system. Or in other words, there should not be any entity in the ecosystem that controls a disproportionate number of resources, whatever those resources are.

8 JSON Canonicalization Algorithms

8.1 Introduction

An important, distinguishing peculiarity of the provenance verification procedures within the NGSI-LD ecosystem, due to its data model, its federated nature and the JSON-LD serialization, is that every time an NGSI-LD Entity is shared, stored or created, it can be rearranged when it is subsequently serialized: for the purpose of provenance verification, signatures need to remain valid despite such re-arranging.

Due to such heterogeneous rearrangements of NGSI-LD Attributes, from different actors, during its lifecycle, it is imperative that any signature performed by context Creators (Producers or Sources) will not be invalidated, in order to not invalidating the data integrity, independently from the Signature system that will be implemented.

This issue is present because every time a broker receives an NGSI-LD Entity, it can store and eventually change its serialization format/arrangement of the NGSI-LD Attributes.

On request the Context Broker will recreate the entity in order to share it.

The same issue will also be present every time a merging operation is performed (clause 9).

Cryptographic operations like hashing and signing depend on the fact that the target data does not change during serialization, transport, or parsing.

There are two common ways of accomplishing this:

1) Converting the data into a format which has a simple and fixed representation
2) Creating a canonicalized version of the data

Both of these processes have their advantages and disadvantages.

The first approach does not need a complex canonicalization algorithm, but encapsulates the data in a message that is not in the original form.

The first approach is the one that has been adopted by JSON Web Signature (JWS) (clause 4).

The second approach, of course, requires the application of a canonicalization algorithm but it has the main advantage that data can be kept in its original form.

In other words, using canonicalization enables a JSON object to remain a JSON object even after being signed.

The second approach is the one that has been adopted by Linked Data Signature (clause 4).

Because it is imperative that a NGSI-LD Entity will not change its format during its lifecycle, **the second approach is the one that needs to be followed.**

8.2 Canonicalization algorithms and their status

8.2.0 Introduction

Canonicalization is the process of transforming an input dataset to a normalized dataset.

Any two input datasets that contain the same information, regardless of their arrangement, will be transformed into identical normalized dataset.

This process is sometimes also called normalization.

Different canonicalization algorithms exist and they are strictly dependent on the format of the data.

Because NGSI-LD can be serialized using JSON-LD, JSON canonicalization algorithms will be analysed.

8.2.1 JCS: JSON Canonicalization Scheme

8.2.1.0 Introduction

JSON Canonicalization Scheme [i.13] defines how to create a canonical representation of JSON data by building on the strict serialization methods for JSON primitives defined by ECMAScript, constraining JSON data to the Internet JSON (I-JSON) subset, and by using deterministic property sorting. The output from JCS is a "hashable" representation of JSON data that can be used by cryptographic methods.

The JSON Canonicalization Scheme can be synthesized:

- Possible whitespace between JSON elements will be ignored but not emitted.
- Serialization of primitive JSON data types using methods compatible with ECMAScript's `JSON.stringify().`
- Lexicographic sorting of JSON Object properties in a recursive process.
- **JSON Array data is also subject to canonicalization, but element order remains untouched.**

- Output is converted to UTF-8, for platform interoperable canonicalization.

Using JCS it is possible to enable signature creation schemes like the following:

- Create the data to be signed.
- Serialize the data using existing JSON tools.
- Externally canonicalize the serialized data and return canonicalized result data.
- Sign the canonicalized data.
- Add the resulting signature value to the original JSON data through a designated signature property.
- Serialize the completed and signed JSON object using existing JSON tools.

Signature Verification will follow this scheme:

- Parse the signed JSON data using existing JSON tools.
- Read and save the signature value from the designated signature property.
- Remove the signature property from the parsed JSON object.
- Serialize the remaining JSON data using existing JSON tools.
- Externally canonicalize the serialized data and return canonicalized result data.
- Verify that the canonicalized data matches the saved signature value using the algorithm and key used for creating the signature.

The JCS specification [i.13] is not an Internet Standards Track specification.

JCS-compatible serialization of JSON primitives is currently supported by most web browsers as well as by Node.js.

Because this algorithm does not support array elements sorting, since it was conceived for the JSON format, where order of the elements of an array is always preserved, it is not satisfactory for NGSI-LD, which is based on JSON-LD serialization.

8.2.1.1 JWS/CT (JWS "Clear Text")

It has already been said that JWS or JSON Web Signature does not utilize any canonicalization algorithm.

Although it is possible to extend the scope of the JSON Web Signature specification, combining the detached mode of JWS with the JSON Canonicalization Scheme (JCS).

The JWS/CT (JWS "Clear Text") specification [i.14] describes this method, enabling JSON objects to remain in the JSON format after being signed.

This introduces the advantage for data to be:

- stored in databases;
- passed through intermediaries;
- embedded in another JSON object;
- countersigned.

without losing the ability to (at any time) verify signatures.

JWS allows a JSON object to be signed but transformed into a string made by a header (JOSE Header), a payload (the original JSON) and a signature separated by the character ".".

For example, a JSON like:

```
{
  "statement": "Hello signed world!",
  "otherProperties": [2e+3, true]
}
```

will end-up like:

```
eyJhbGciOiJIUzI1NiIsImtpZCI6Im15a2V5In0.eyJvdGhlclByb3BlcnRpZXMiOlsyMDAwLHRydWVdLCJzdG
F0ZW1lbnQiOiJIZWxsbyBzaWduZWQgd29ybGQhIn0.FcE8h0GXJaOZ4Th3fNDBgcBE5HfEplOnS8GGtoSLU1K
```

The major benefits of text-based schemes (human readability), got lost in the process. In addition, the whole JSON structure was transformed into something entirely different.

Using JWS in "detached" mode it will end-up like:

```
{
  "statement": "Hello signed world!",
  "otherProperties": [2e+3, true],
  "signature": "eyJhbGciOiJIUzI1NiIsImtpZCI6Im15a2V5In0...5HfEplOnS8GGto
   SLU1KFcE8h0GXJaOZ4Th3fNDBgcBE"
}
```

without losing the benefits of text-based schemes, keeping existing security standards.

Giving as "JWS Payload" to the JWS signature process the JCS canonicalized, it is possible to unite these two realities.

Signing Operation scheme will be the following:

1) Create or parse the JSON object to be signed.
2) Use the result of the previous step as input to the canonicalizing process.
3) Use the result of the previous step as "JWS Payload" to the JWS signature process, using the compact serialization mode.
4) Add the resulting JWS string to the original JSON object through a designated signature property.

The validation scheme will be the following:

1) Parse the signed JSON object.
2) Read and save the JWS string from the designated signature property.
3) Remove the designated signature property from the parsed JSON object.
4) Apply the canonicalizing process on the remaining object.
5) Use the result of the previous step as "JWS Payload" to the JWS validation process.

JWS/CT specification [i.14] has been produced outside the IETF, is not an IETF standard, and does not have IETF consensus.

Even utilizing an internet standard like JWS in detached mode for sign, the JCS algorithm **does not support array elements sorting, since it was conceived for the JSON format, where order of the elements of an array is always preserved, so it is not satisfactory for NGSI-LD, which is based on JSON-LD serialization.**

8.2.2 RDF Dataset Canonicalization

Resource Description Framework (RDF) [i.15] is a framework for representing information on the Web.

The RDF data model is based on the idea of making statements about resources in expressions of the form subject-predicate-object, known as triples. An RDF triple along with a graph name is called a Quad.

A set of RDF triples forms a RDF graph.

RDF is an abstract model with several serialization formats like:

- Turtle
- N-Triples
- N-Quads.
- **JSON-LD**
- N3 or Notation3
- RDF/XML
- **RDF/JSON**

For JSON format the two main serializations are JSON-LD and RDF/JSON.

The W3C RDF 1.1 JSON Alternate Serialization (RDF/JSON) specification [i.16] defines a textual syntax for RDF called RDF/JSON that allows an RDF graph to be completely written in a form compatible with the JavaScript Object Notation.

On the other hand, JSON-LD is designed around the concept of a "context" to provide additional mappings from JSON to an RDF model.

It is important to note that, for the W3C Working Group, RDF/JSON serialization should not be used unless there is a specific reason to do so. Use of JSON-LD is recommended.

Considering a RDF dataset, it is crucial to define a canonicalization algorithm, not only for digital signing of graphs, but especially for comparing two graphs independently of serialization or format.

This eventuality matches exactly the characteristics of NGSI-LD.

The W3C RDF Dataset Canonicalization specification [i.17] outlines an algorithm for normalizing RDF datasets, called Universal RDF Dataset Normalization Algorithm 2015 (URDNA2015).

This is not a W3C Standard nor is it on the W3C Standards Track.

In order to better understand this algorithm some terms have to be defined:

- **Blank node:** is a node in a graph that is neither an IRI, nor a literal.
- **Blank node identifier:** is a string that begins with "`_:`" that is used as an identifier for a blank node.
- **Canonicalization state:** is a data structure containing information about:
  - **blank node to quads map:** a data structure that maps a blank node identifier to the quads in which they appear in the input dataset.
  - **hash to blank nodes map:** a data structure that maps a hash to a list of blank node identifiers.
  - **canonical issuer:** an identifier issuer, initialized with the prefix "`_:c14n`", for issuing canonical blank node identifiers.
- **Hash First Degree Quads algorithm:** an algorithm takes the canonicalization state and a reference blank node identifier as inputs, and returns the hash that results from passing the sorted, joined n-quads through the hash algorithm (SHA 256).

Considering a JSON or a JSON-LD as a serialization of an RDF Dataset, from now on called input dataset, this algorithm can be applied to it.

1) Create the canonicalization state.
2) For every quad in the input dataset, if a blank node that occurs in the quad, add a reference to the quad using the blank node identifier in the blank node to quads map.

3) Create a list of non-normalized blank node identifiers and populate it using the keys from the blank node to quads map.

4) Issue canonical identifiers for blank nodes.

5) For each blank node identifier in non-normalized identifiers, create a hash, according to the Hash First Degree Quads algorithm, and add hash and identifier to hash to blank nodes map, creating a new entry if necessary.

6) For each hash to identifier list mapping in hash to blank nodes map, lexicographically-sorted by hash, create a hash path list where each item will be a result of running the Hash N-Degree Quads algorithm.

7) For each result in the hash path list, lexicographically-sorted by the hash in result, create a hash path list where each item will be a result of running the Hash N-Degree Quads algorithm.

8) For each quad in the input dataset, create a copy of quad and replace any existing blank node identifiers using the canonical identifiers previously issued by the canonical issuer, and add quad copy to the normalized dataset.

9) Return the normalized dataset.

The W3C Data Integrity specification (clause 4) utilizes this canonicalization algorithm in order to create its linked data signature.

Thus, implementation of this model on an NGSI-LD Entity would solve two problems in one row: integrity and canonicalization.

9 Analysis and Comparison

9.1 Introduction

A first step to the implementation of a model for NGSI-LD entities distribution is a comparison with all the specifications described so far.

Analysing existing data models and performing a mapping process among actors and concepts will bring to a better comprehension of the strategy to adopt.

The main comparison in the present document is with the W3C Verifiable Credentials structure and an NGSI-LD Entity.

9.2 Mapping of actors and terms among different standards

Starting with actors:

Table 9.2-1: Actors Comparison

W3C VC	Subject	Issuer	Holder	Verifier
NGSI-LD	Entity Subject	Context Creator	Context Broker	Client

In this first comparison all four actors have a mutual representation but with many differences:

In an NGSI-LD the subject is represented by the entity and declared by the "id" value.

In a verifiable credential the subject is expressed in it, by the "subjectId" value, and its "id" value, if present, represents the credential itself.

- So, the first main difference is that the Verifiable credentials do not represent the subject, but represent a credential containing statements about one or more subjects.

It is important to notice that an NGSI-LD Entity can be created in order to represent a credential as a Verifiable Credentials.

Issuers and Context creators can be compared only because they are those who sign documents and they are trustworthy.

- The second difference is that Context Creators generate and eventually sign documents not because of an issuing request from a subject.

Holders and Context Brokers can be compared because they are both actors who possess and manage credentials/entities.

- The third difference is that Holders can collect credentials, create presentations and sign them. Context Brokers can merge, disclose and share entities without creating an equivalent presentation super-structure, and even if they can sign documents, they are not intended to be part of the trust model.

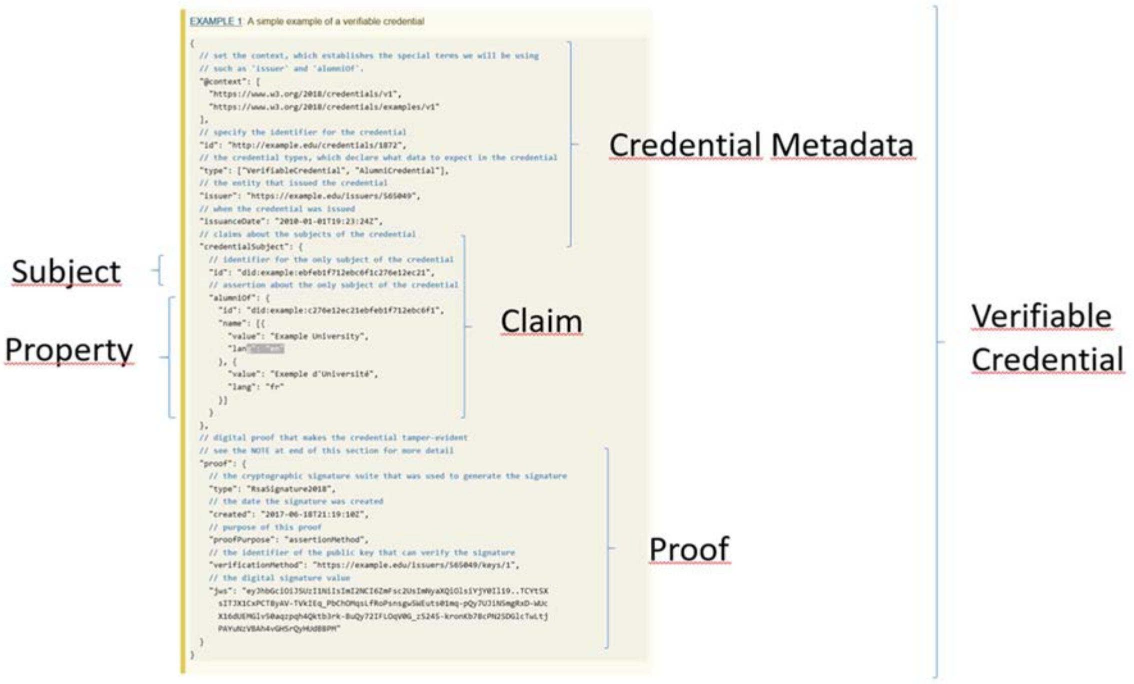
Verifiers and Clients can be compared because they basically check the integrity of data received.

- The fourth difference is that Verifiers just validate documents, while Clients are also who benefit and utilize those documents.

9.3 Mapping of Structures among different Standards

Continuing the comparison of the structures of these entities, in order to try to obtain a mapping between VC and NGSI-LD entities.

Considering a Credential serialized in JSON-LD:


EXAMPLE 1 A simple example of a verifiable credential

```
{
  // set the context, which establishes the special terms we will be using
  // such as 'issuer' and 'alumniOf'.
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1"
  ],
  // specify the identifier for the credential
  "id": "http://example.edu/credentials/1872",
  // the credential types, which declare what data to expect in the credential
  "type": ["VerifiableCredential", "AlumniCredential"],
  // the entity that issued the credential
  "issuer": "https://example.edu/issuers/565049",
  // when the credential was issued
  "issuanceDate": "2010-01-01T19:23:24Z",
  // claims about the subjects of the credential
  "credentialSubject": {
    // identifier for the only subject of the credential
    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    // assertion about the only subject of the credential
    "alumniOf": {
      "id": "did:example:c276e12ec21ebfeb1f712ebc6f1",
      "name": [{
        "value": "Example University",
        "lang": "en"
      }, {
        "value": "Exemple d'Université",
        "lang": "fr"
      }]
    }
  },
  // digital proof that makes the credential tamper-evident
  // see the NOTE at end of this section for more detail
  "proof": {
    // the cryptographic signature suite that was used to generate the signature
    "type": "RsaSignature2018",
    // the date the signature was created
    "created": "2017-06-18T21:19:10Z",
    // purpose of this proof
    "proofPurpose": "assertionMethod",
    // the identifier of the public key that can verify the signature
    "verificationMethod": "https://example.edu/issuers/565049/keys/1",
    // the digital signature value
    "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..TCYt5X
      sITJX1CxPCT8yAV-TVkIEq_PbChOMqsLfRoPsnsgw5WEuts01mq-pQy7UJiN5mgRxD-WUc
      X16dUEMGlv50aqzpqh4Qktb3rk-BuQy72IFLOqV0G_zS245-kronKb78cPN25DGlcTwLtj
      PAYuNzVBAh4vGHSrQyHUdBBPM"
  }
}
```

Figure 9.3-1: Verifiable Credential structure example

As shown in Figure 9.3-1 the Credentials are structured to be a collection of Credential metadata and a nested structure ("`credendiatlSubject`") containing information about subject and claims.

Although the logical equivalence between attributes and claims (both are used to indicate a statement about a subject, expressed using subject-property-value relationships) these two structures are different.

Credentials utilize types and subtypes properties to indicate the possible claims to be present, referring to schemes.

The NGSI-LD attribute type (Property or Relationship) is missing.

The second main structure in the Verifiable Credentials specification, Verifiable Presentation cannot be implemented in the NGSI-LD ecosystem.

Even though a NGSI-LD Entity can be created containing more NGSI-LD entities in it, the "`verifiableCredential`" array structure cannot be implemented, as is present in VC.

It also important to notice that a Verifiable Presentation is intended to be a collection of credentials singed by the Holder to be presented to a Verifier. On the other side Context Brokers are not supposed to sign documents because they are not part of the trust model.

For these reasons Verifiable Credential and Verifiable Presentation structures cannot be implemented in the NGSI-LD ecosystem.

10 Suggested Solution

10.1 Introduction

In the typical NGSI-LD context data lifecycle it is possible to distinguish three main phases:

1) Collection and merging of NGSI-LD Entities coming from Context data Creators (Context Producers and Context Sources), performed by the Context Broker.

2) Sharing and disclosing processes among federated context brokers.

3) Presentation of merged NGSI-LD Entities, as a result of a query, to Context consumers.

In order to guarantee data integrity (see clause 4), through the whole lifecycle, a cryptographic mechanism has to be implemented from creators to clients. Possible solutions and relative approaches will be here analysed.

10.2 Modular approach

10.2.0 Introduction

Considering a scenario where multiple different aggregation steps can happen, where different brokers can share and disclose part of an NGSI-LD Entity.

Considering also that among all actors in the NGSI-LD lifecycle: Creators, Brokers, Clients:

1) Only Creators can sign data

2) Not every broker can communicate with each creator for a future sign

a modular approach will fulfil all requirements.

Every NGSI-LD Entity can be seen as made of three parts:

1) The id and type part (head)

2) The attributes part (core)

3) And an optional proof part (tail)

Considering an NGSI-LD Entity as a chain and each attribute as a link, every manipulation process (merging, disclosing, aggregation) changes only its core, not the head, adding or removing links.

In order to have a modular approach every attribute has to be a self-signed "quantum" information.

NGSI-LD attributes are not fungible and cannot be separate from their subject; this means that every attribute has to contain information about the id and type of the entity which is referred to and, optionally, to its type. So, the attribute cannot be the atomic entity, because it has also to contain the head part information; it is a chain itself in disguise.

The atomic entity is the fragment, a signed NGSI-LD Entity with only one attribute:

```
{
  id
  type            head
  attribute       core
  proof           tail
}
```

This structure and its composition will be independent from the choice of signature mechanism.

For suggestion and example, the tail part will be called "proof", referring to the W3C Data Integrity Specification [i.1].

10.2.1 Algorithm

Starting with the definition of the Atomic Entity the following steps have to be followed:

1) Generation of the Atomic Entity.

2) The atomic entity will be transformed into a **Sealed Attribute**, through a **Derivation process**.

3) Sealed attributes will be merged in a single NGSI-LD Entity with the same id and type.

4) Depending on Client request, custom NGSI-LD Entity will be presented, sharing or disclosing attributes.

5) Original fragments are recreated following the **Recreation Process** and validated.

Step 1

An Atomic Entity is generated for each one of the original attributes of the NGSI-LD Entity.

Step 2

After the derivation process a Sealed Attribute will be:

```
attribute = {
    "type": "Property"
    "value": "value1"
    "ngsildproof" {
        "type": "Property"
        "entityIdSealed": "…"
        "entityTypeSealed": "…"
        "value": {
          "proof": {…}
        }
    }
}
```

A Sealed Attribute is an NGSI-LD attribute with the addition of "`ngsildproof`" key.

"`ngsildproof`" contains the following properties:

- `type`: NGSI-LD Property type
- `entityIdSealed`*:* optional, if missing fetch from Entity
- `entityTypeSealed`: optional, if missing fetch from Entity
- `value`: object containing the proof structure

The `value` object will contain the proof structure selected for the signature mechanism (for example W3C data integrity proof).

During the Derivation process, the proof object is extracted from the original fragment and inserted inside the "`ngsildproof`" structure nested inside the attribute itself (under "`value`" key).

Two more optional keys ("`entityIdSealed`", "`entityTypeSealed`") can be added to the "`ngsildproof`" structure, in order to keep information of id and type of the original atomic entity (as described in clause 10.2.0).

It is important to specify that the proof will not verify the new attribute structure, but only the original fragment.

Only the reconstruction of the original fragments of the atomic entity will allow checking the signature.

This different approach introduces a change of paradigm to the attribute structure of an NGSI-LD Entity.

Step 3

After the Derivation Process, the attribute can be considered as a link to be connected to every chain with the same head part through the merging process.

Links can be generated from creators or from the first Broker who receives it and then shared.

At the end of a merging process a Broker will have an entity like:

```
{
  "id": "…",
  "type": "…",
  "attribute1": {
    "type": "Property",
    "value": "…",
    "ngsildproof": {
      "type": "Property",
      "entityIdSealed": "…",
      "entityTypeSealed": "…",
      "value": {
        "proof": {…}
      }
    }
  },
  "attribute2": {
    "type": "Property",
    "value": "…",
    "ngsildproof": {
      "type": "Property",
      "entityIdSealed": "…",
      "entityTypeSealed": "",
      "value": {
        "proof": {…}
      }
    }
  },
  .
  .
  .
  "attribute_n": {
  }
}
```

Step 4

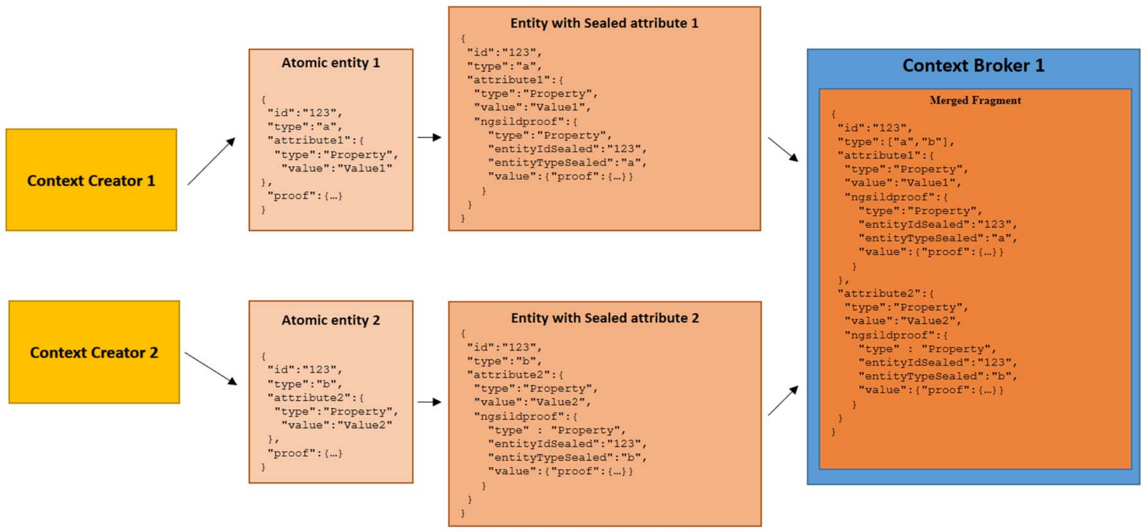
Following this recombination scheme merged entities will be shared among Context Brokers or presented to Clients.

At the end of its lifecycle the Recreation process will generate all original Atomic Entities to validate them.

The Recreation process is the opposite of the derivation process, that takes as input Sealed Attributes and generates atomic entities as output.

Following the NGSI-LD lifecycle the Modular approach can be graphically represented.

The first phase is represented in Figure 10.2.1-1.



Atomic entity 1

```
{
 "id":"123",
 "type":"a",
 "attribute1":{
  "type":"Property",
   "value":"Value1"
 },
 "proof":{…}
}
```

Entity with Sealed attribute 1

```
{
 "id":"123",
 "type":"a",
 "attribute1":{
  "type":"Property",
  "value":"Value1",
  "ngsildproof":{
    "type":"Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"a",
    "value":{"proof":{…}}
   }
  }
}
```

Atomic entity 2

```
{
 "id":"123",
 "type":"b",
 "attribute2":{
  "type":"Property",
   "value":"Value2"
 },
 "proof":{…}
}
```

Entity with Sealed attribute 2

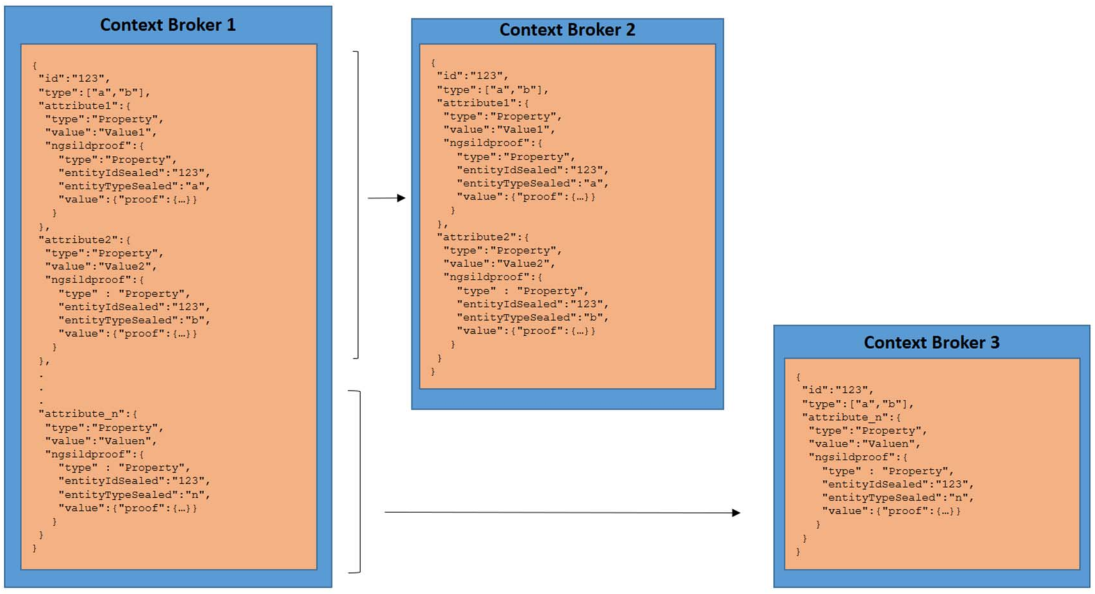
```
{
 "id":"123",
 "type":"b",
 "attribute2":{
  "type":"Property",
  "value":"Value2",
  "ngsildproof":{
    "type" : "Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"b",
    "value":{"proof":{…}}
   }
  }
}
```

Context Broker 1 – Merged Fragment

```
{
 "id":"123",
 "type":["a","b"],
 "attribute1":{
  "type":"Property",
  "value":"Value1",
  "ngsildproof":{
    "type":"Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"a",
    "value":{"proof":{…}}
   }
 },
 "attribute2":{
  "type":"Property",
  "value":"Value2",
  "ngsildproof":{
    "type" : "Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"b",
    "value":{"proof":{…}}
   }
  }
}
```

Figure 10.2.1-1: Phase 1 scheme

The second phase will be a recombination process of these derived attribute structures, for disclosing and sharing purposes among context Brokers. This phase can be represented as in Figure 10.2.1-2.



Context Broker 1

```
{
 "id":"123",
 "type":["a","b"],
 "attribute1":{
  "type":"Property",
  "value":"Value1",
  "ngsildproof":{
    "type":"Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"a",
    "value":{"proof":{…}}
   }
 },
 "attribute2":{
  "type":"Property",
  "value":"Value2",
  "ngsildproof":{
    "type" : "Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"b",
    "value":{"proof":{…}}
   }
 },
 .
 .
 .
 "attribute_n":{
  "type":"Property",
  "value":"Valuen",
  "ngsildproof":{
    "type" : "Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"n",
    "value":{"proof":{…}}
   }
  }
}
```

Context Broker 2

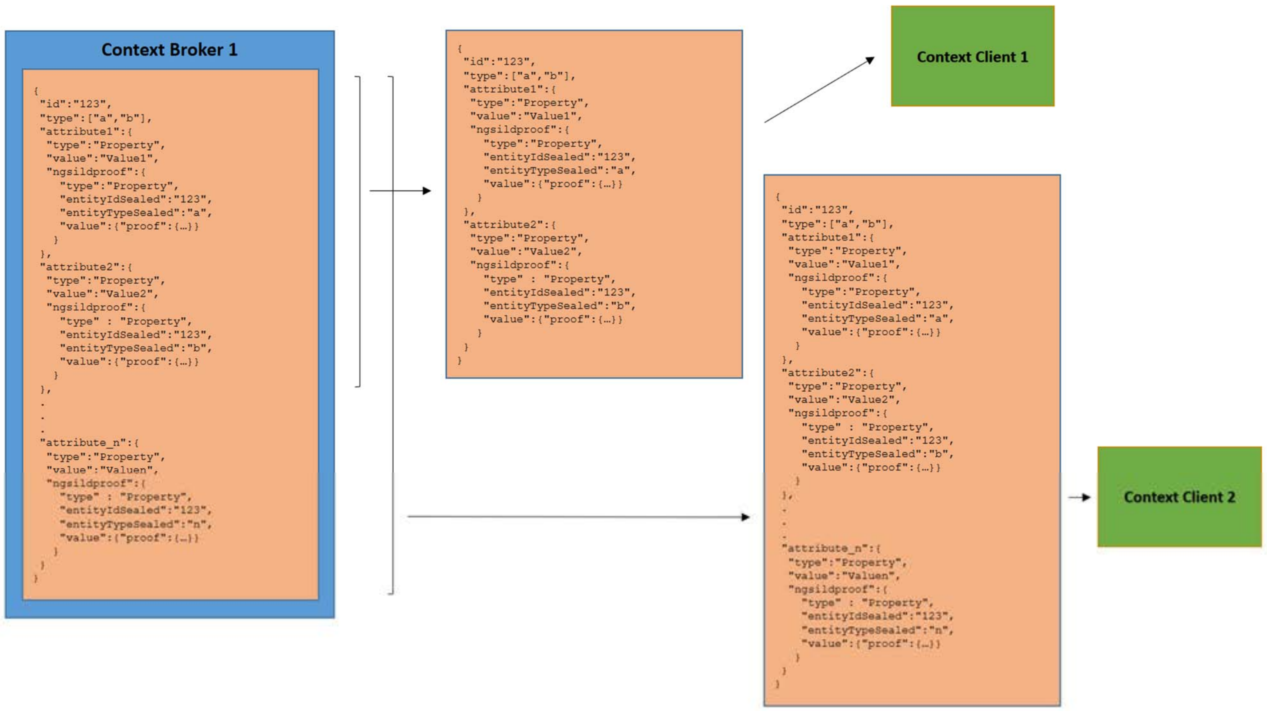
```
{
 "id":"123",
 "type":["a","b"],
 "attribute1":{
  "type":"Property",
  "value":"Value1",
  "ngsildproof":{
    "type":"Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"a",
    "value":{"proof":{…}}
   }
 },
 "attribute2":{
  "type":"Property",
  "value":"Value2",
  "ngsildproof":{
    "type" : "Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"b",
    "value":{"proof":{…}}
   }
  }
}
```

Context Broker 3

```
{
 "id":"123",
 "type":["a","b"],
 "attribute_n":{
  "type":"Property",
  "value":"Valuen",
  "ngsildproof":{
    "type" : "Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"n",
    "value":{"proof":{…}}
   }
  }
}
```

Figure 10.2.1-2: Phase 2 scheme

The third phase will be implemented, as the second one, by a recombination of attributes, depending on the client's request. Third phase can be represented as in Figure 10.2.1-3.



```
{
 "id":"123",
 "type":["a","b"],
 "attribute1":{
  "type":"Property",
  "value":"Value1",
  "ngsildproof":{
    "type":"Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"a",
    "value":{"proof":{…}}
   }
 },
 "attribute2":{
  "type":"Property",
  "value":"Value2",
  "ngsildproof":{
    "type" : "Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"b",
    "value":{"proof":{…}}
   }
 },
 .
 .
 .
 "attribute_n":{
  "type":"Property",
  "value":"Valuen",
  "ngsildproof":{
    "type" : "Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"n",
    "value":{"proof":{…}}
   }
 }
}
```

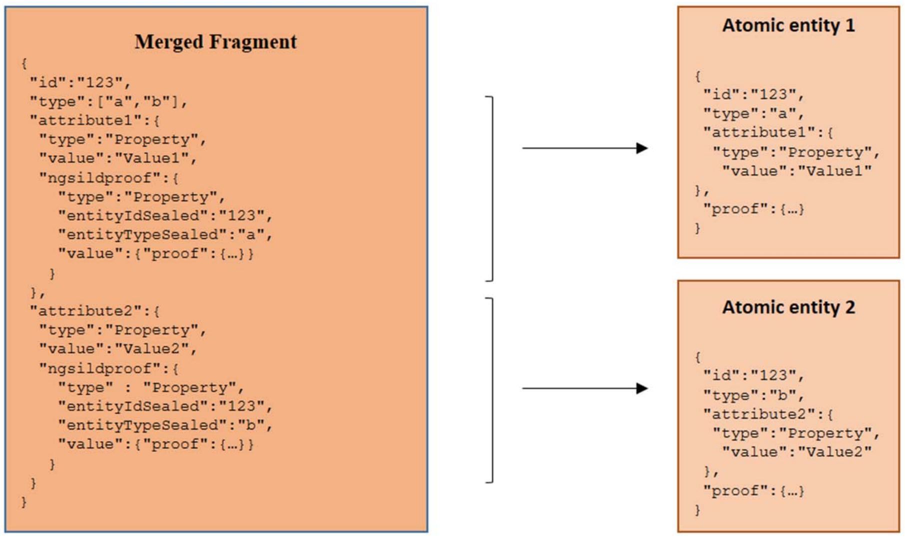
```
{
 "id":"123",
 "type":["a","b"],
 "attribute1":{
  "type":"Property",
  "value":"Value1",
  "ngsildproof":{
    "type":"Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"a",
    "value":{"proof":{…}}
   }
 },
 "attribute2":{
  "type":"Property",
  "value":"Value2",
  "ngsildproof":{
    "type" : "Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"b",
    "value":{"proof":{…}}
   }
 }
}
```

```
{
 "id":"123",
 "type":["a","b"],
 "attribute1":{
  "type":"Property",
  "value":"Value1",
  "ngsildproof":{
    "type":"Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"a",
    "value":{"proof":{…}}
   }
 },
 "attribute2":{
  "type":"Property",
  "value":"Value2",
  "ngsildproof":{
    "type" : "Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"b",
    "value":{"proof":{…}}
   }
 },
 .
 .
 .
 "attribute_n":{
  "type":"Property",
  "value":"Valuen",
  "ngsildproof":{
    "type" : "Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"n",
    "value":{"proof":{…}}
   }
 }
}
```

Figure 10.2.1-3: Phase 3 scheme

Step 5

The validation of all attributes has to be obtained through the Recreation Process and then with the verification checks, as in Figure 10.2.1-4.



```
{
 "id":"123",
 "type":["a","b"],
 "attribute1":{
  "type":"Property",
  "value":"Value1",
  "ngsildproof":{
    "type":"Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"a",
    "value":{"proof":{…}}
   }
 },
 "attribute2":{
  "type":"Property",
  "value":"Value2",
  "ngsildproof":{
    "type" : "Property",
    "entityIdSealed":"123",
    "entityTypeSealed":"b",
    "value":{"proof":{…}}
   }
 }
}
```

```
{
 "id":"123",
 "type":"a",
 "attribute1":{
  "type":"Property",
   "value":"Value1"
},
 "proof":{…}
}
```

```
{
 "id":"123",
 "type":"b",
 "attribute2":{
  "type":"Property",
   "value":"Value2"
 },
 "proof":{…}
}
```

Figure 10.2.1-4: Verification Process

An alternative solution can be the on-the-fly generation of an aggregated signature (through BBS+ signature suite utilization, for example) that can be verified in one batch with original fragments.

If a Context Broker cannot sign, the overall integrity of the entire entity cannot be guaranteed.

In this situation an arbitrary disclosure of attributes can be performed without control (i.e. if one broker decides to hide/remove some attributes, this goes unnoticed).

In this approach, signature aggregation and ZKP selective disclosure are useless, because these properties allow a one-way-only information reduction, so they could only be performed at the end of the life cycle, but the possibility of an intermediate Broker sharing (and possibly re-arranging) the same data, prevents it.

Furthermore, because of the "one attribute one signature" system, there is no need for bundle disclosure or aggregation.

Summarizing:

1) The atomic entity is the fragment, a signed NGSI-LD Entity with only one attribute.
2) The Derivation process will transform the atomic entity into a Sealed Attribute.
3) The Sealed Attribute structure will have a nested "ngsildproof" object containing information about the original atomic fragment and the original signature.
4) Attributes can be merged, shared and disclosed, through a combination of them as links of a chain.
5) The final presentation to the Client will be performed in the same way.
6) Original Atomic Entities are recreated following the inverse Derivation Process and validated.

This approach has the following pros and cons:

Pros:

1) Context information creators (or Context Providers on their behalf) sign every Atomic Entity only once. No further signing requests will be made.
2) Attribute-level granularity is achieved, as Sealed Attributes can be shared and removed easily, as links of a chain, within a federation.
3) Final presentation of a custom Entity, according to a Client's request, can be performed easily because of the one-Attribute-one-proof foundation.
4) No specific signature suites are required.

Cons:

1) Context information creators (or Context Providers on their behalf) have to sign every single Attribute that needs integrity and provenance. It is not possible to have one proof cover more than one Attribute.
2) A Sealed Attribute cannot be verified "as is" by the software libraries implementing the verification method and algorithms; it has to be reconverted into the original Atomic Entity.
3) Information about id and type of the Entity at the time the proof was created, and the proof itself, has to be stored in every single Sealed Attribute and kept at every step, causing some redundancy.
4) Verification requires one check for every single Attribute, because it is not possible to have one proof cover more than one Attribute.

10.2.2 Type changing and Multitype issue

During the merging process two issues can occur:

- Changing type
- Multi-typing

Different Context Creators can create NGSI-LD Entities referring to the same subject but with different Type values.

This can happen because the same subject, with the same "id" value, depending on the circumstances, can be described with different type values.

For example, a NGSI-LD Entity describing a car:

```
{
    "@context": ["…"],

    "id": "https://example.org/car1",
    "type": "Car",

    "color": {
      "type": "Property",
      "value": "Red"
    },
    "speed": {
      "type": "Property",
      "value": "45"
    },
    "proof": {…}
}
```

Can also be created or renamed as a vehicle by another Broker:

```
{
    "@context": ["…"],

    "id": "https://example.org/car1",
    "type": "Vehicle",

    "color": {
      "type": "Property",
      "value": "Red"
    },
    "proof": {…}
}
```

This brings to the first issue the change of type value for the same id.

Secondarily when it comes to merge these two entities, both types have to be collected and managed depending on how the final entity has to be presented, leading to the multitype issue.

Some NGSI-LD attributes may only belong to the subject of an entity with a specific type, so after the merging process they have to be disclosed or shared depending on the type of Entity to be presented.

Following the Modular approach, as already described, in order to collect information about possible type changing during a NGSI-LD Entity lifecycle, the introduction of "`entityTypeSealed`" key inside the Sealed Attribute structure could be a possible solution.

This value will keep track of the type value of the original entity which that attribute is related, and will be utilized in the Recreation process to create the original atomic entity with the correct Type value.

In order to resolve the multitype issue, the NGSI-LD Entity `type` is an array structure (see discussion about multi-typing in [i.18]), so all type values involved in the merging process can be collected.

10.2.3 Consideration about merged fragment integrity

As shown before, in clause 10.2.1, every single attribute content is tamper-evident because of the "one attribute one signature" system, but nothing can guarantee the overall integrity of one big, merged fragment, because a Context Broker could arbitrarily delete or hide attributes to Clients or other brokers.

This issue is caused by the absence of an actor who can sign the entire merged fragment with one signature covering the whole document.

One possible solution could be the integration in the trust model of a Verification Data Registry, similar to the W3C verifiable credential Data Model.

Like W3C VC, examples of verifiable data registries include trusted databases or decentralized databases, and more than one type can be present for different ecosystems, as described in clause 7.

All actors trust the verifiable data registry to be tamper-evident and to be a correct record of which fragment has been signed by which creators.

Whenever a Creator desires, it can register the signature process of a fragment and its attribute for its ID Entity, without revealing the value of that attribute.

Whenever a client or even a broker receives information about an entity, it can independently check in the verifiable data registry, in order to check if some attributes were hidden.

During the verification check, the attribute name, signer's identity, and signature date can be verified.

10.2.4 Attempts to update signed Attributes

Following the Modular approach, some implications in the NGSI-LD attribute structure and in the NGSI-LD API have to be analysed.

A decision is to be taken, whether the addition of sub-attributes to a sealed Attribute represents a modification that renders the signature invalid. The cautious approach is suggested: every modification of the signed sealed Attribute will invalidate the signature, including all sub-attributes.

Thus, the following procedures are recommended when updates of Entities from other context Brokers and context Consumers, via NGSI-LD API, are received:

1) Context Producers injecting signed Attributes should convey them with a datasetId already in place, thus the signed Attributes can be preserved whenever an external client tries to modify them.

2) If any client, including the original producer, uses the producer's datasetId to modify exactly that signed Attribute, the modify request can fail.

The preservation of the original signed Attribute keeps provenance information and guarantees the possibility to verify integrity of modified copies containing the same data.

The newly arriving values will thus have either the default datasetId or a datasetId different than the datasetId of a signed Attribute. New values of the same Attribute will live in the default Attribute instance, or will contain a datasetId value different from the one of the original Attribute, otherwise the request can fail, or the signature be removed.

In this scenario, a particular attack can be mounted by a malicious client, which could try to add a dummy `proof` block to an Attribute to make it appear as if it was signed (though it will not verify), together with a datasetId (maybe of another producer): hence, prior to failing the NGSI-LD API modify operation it is recommended that implementations verify that the new proof still verifies the original Attribute's signature, if they allow permission to modify it.

10.3 Call Back approach

10.3.0 Introduction

Another possible approach, on top of the "one attribute one signature", could be the implementation of aggregation and selective disclosure systems.

The atomic entity remains the original fragment with a single attribute and signature, provided by creators to brokers, described in clause 10.2.

This approach requires the utilization of BBS+ Signature Suite or every other suite with aggregation and selective disclosure properties (clause 4).

Distinguishing three different phases:

1) Original fragment distribution

2) Broker to broker sharing

3) Final custom entity to client presentation

this approach can be graphically represented as in Figure 10.3.0-1.

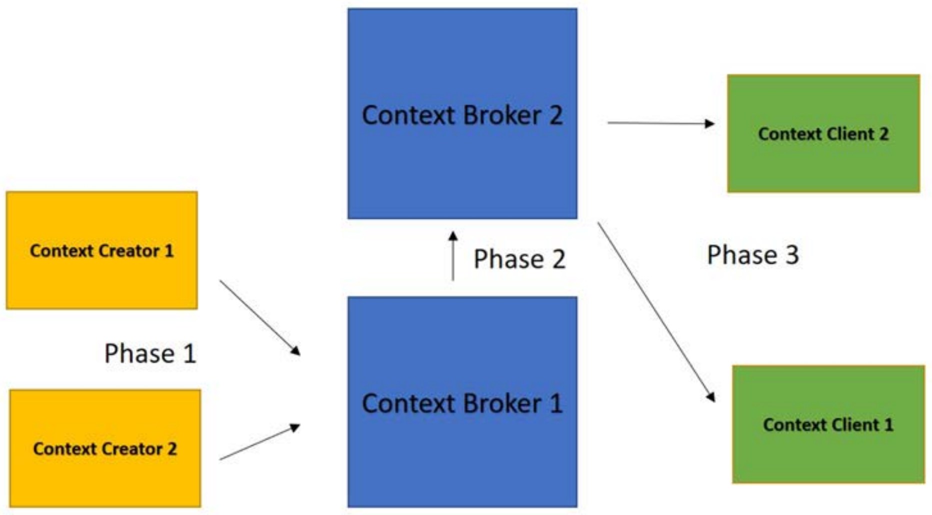


Figure 10.3.0-1: NGSI-LD Entity lifecycle

10.3.1 Algorithm

Starting with the definition of the atomic Entity the following steps have to be followed:

1) Generation of atomic entity
2) Generation of merged entity through the **Derivation Process**
3) Call back to all involved creators for a signature request
4) Signature aggregation
5) Generation of a Zero Knowledge Proof while sharing fragments among Context Broker
6) Repetition of steps 3 and 4
7) Final presentation to the Client of the requested entity through ZKP or the entire signed NGSI-LD Entity
8) Verification of the Aggregated signature and all attributes signatures

Analysing it by phase:

Phase 1

After the atomic entities generation, Context Broker will collect all of them and trough the Derivation Process (described in clause 10.2.1, steps 1 to 3) will generate a Merged Fragment.

This process will be independent from the choice of signature mechanism.

For suggestion and example, the tail part will be called "proof", referring to the W3C Data Integrity Specification [i.1].

Considering two fragments:

fragment 1

```
{
  "@context": ["https://w3id.org/security/bbs/v1"],
  "id": "https://example.org/car1",
  "type": "Car",
  "color": {
    "type": "Property",
    "value": "Red",
    "ngsildproof" {
      "type": "Property",
      "entityIdSealed": "https://example.org/car1"
      "entityTypeSealed": "Car",
      "value": {
        "proof": {…}
      }
    }
  },
  "speed": {
    "type": "Property",
    "value": "45"",
    "ngsildproof" {
      "type": "Property",
      "entityIdSealed": "https://example.org/car1"
      "entityTypeSealed": "Car",
      "value": {
        "proof": {…}
      }
    }
  }
}
```

fragment 2

```
 {
   "@context": ["https://w3id.org/security/bbs/v1"],
   "id": "https://example.org/car1",
   "type": "Car",
   "brand": {
     "type": "Property",
     "value": "BMW" ",
     "ngsildproof" {
       "type": "Property",
       "entityIdSealed": "https://example.org/car1"
       "entityTypeSealed": "Car",
       "value": {
         "proof": {…}
       }
     }
  }
}
```

The merged fragment has to be an entity equivalent in form to the original fragments.

```
{
  "@context": ["https://w3id.org/security/bbs/v1"],
  "id": "https://example.org/car1",
  "type": "Car",
  "color": {
    "type": "Property",
    "value": "Red"",
    "ngsildproof" {
      "type": "Property",
      "entityIdSealed": " https://example.org/car1"
      "entityTypeSealed": "Car",
      "value": {
        "proof": {…}
      }
    }
  },
  "speed": {
    "type": "Property",
    "value": "45"",
    "ngsildproof" {
      "type": "Property",
      "entityIdSealed": " https://example.org/car1"
      "entityTypeSealed": "Car",
        "value": {
          "proof": {…}
        }
     }
   },
   "brand": {
     "type": "Property",
     "value": "BMW" ",
     "ngsildproof" {
       "type": "Property",
       "entityIdSealed": " https://example.org/car1"
       "entityTypeSealed": "Car",
       "value": {
         "proof": {…}
       }
```

```
        }
    }
}
```

After the derivation of every attribute into the merged fragment, a Context Broker calls back every single Creator involved in its merged fragments for a second signature to the overall entity.

Thanks to the aggregation property of the BBS signature, if every creator signs the same merged fragment, all signatures and related public keys can be aggregated.

This will create a single signature that verifies the merged fragment directly and is guaranteed by all creators combined.

This will solve the overall integrity, inhibiting the possibility that hiding/removing attributes goes unnoticed (clause 10.2.1, step 5).

The process of call back has to be repeated every time a new original fragment is received (or after a certain time or a certain number of attributes received).

In order to ensure this call-back process, a broker has to send to every single creator the merged fragment or a derived document of the merged fragment revealing only the attributes related to that specific creator, with the original signature.

Subsequently, the merged fragment is signed with the creator's private keys.

All signatures are sent back to the broker, aggregated and bound to the merged fragment.

The aggregated fragment integrity is always guaranteed.

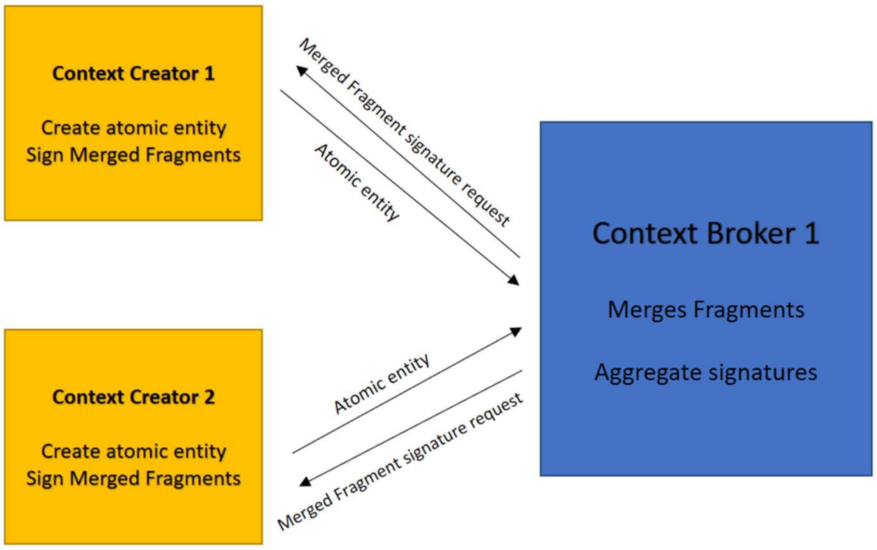


Figure 10.3.1-1: Phase 1 scheme

Phase 2

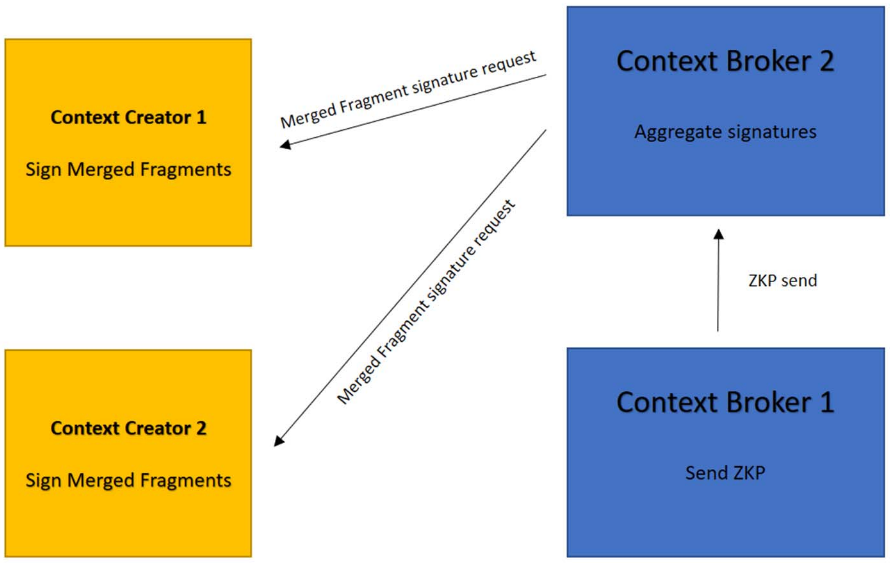


Figure 10.3.1-2: Phase 2 scheme

When two brokers want to share information, if they need to share the entire fragment they can actually do it, otherwise, if some attributes, only, have to be disclosed, a Proof of knowledge of the required attribute list will be sent.

Subsequently, all attributes' integrity can be verified.

The last broker has to repeat the call back system to obtain an aggregated signature from every creator involved in its merged fragments.

The last broker will use each attribute signature of every single creator to prove the integrity, and require a new overall signature.

Phase 3

The final phase is the presentation of results to a client.

If the client requires the entire merged fragment, the entire document can be sent, otherwise a proof of knowledge of the required attributes will be provided.

The integrity of every single attribute is still guaranteed by the underlying "one attribute one signature system".

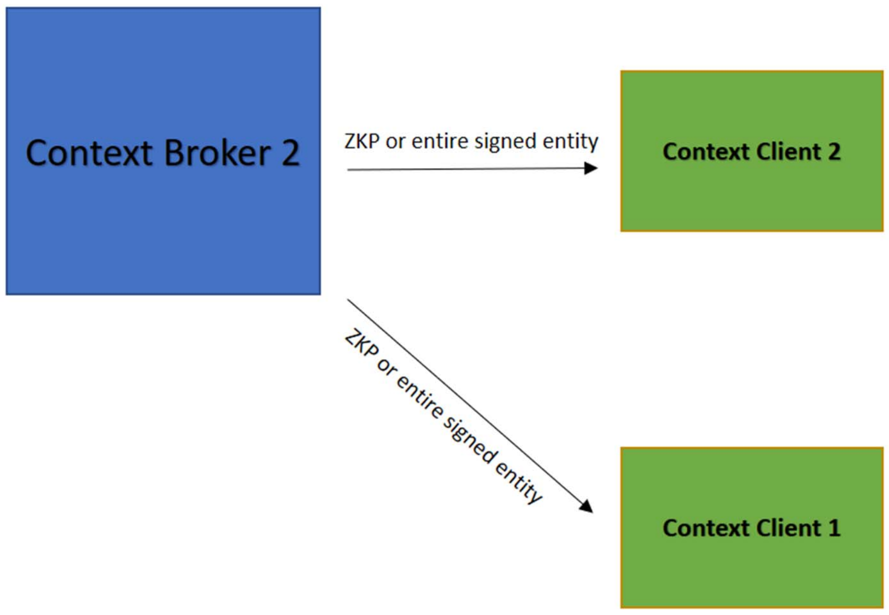


Figure 10.3.1-3: Phase 3 scheme

If no privacy disclosure policy is applied among creators and brokers this method can be implemented.

Pros:

1) This model guarantees the integrity of every attribute and every merged fragment, with the signature of every creator involved.
2) No further derivation process is required; every single merged fragment has its own proof that directly verifies it.
3) The zero-knowledge proof system implementation allows the possibility to generate a subset of verifiable merged fragments and the knowledge of a disclosure process.
4) Every Broker does not need to be trusted.
5) No independent verification system (like Verifiable data registry) is required.
6) Every process needs just one signature verification in order to prove its integrity.
7) No malicious merging process can be performed by any broker because it has to be performed every time a fragment is generated and sent back.

Cons:

1) Multiple signature call-backs are required.
2) If a privacy disclosure policy is applied among creators and brokers, at phases 1 and 2, a functional request of signature from brokers to creators has to be implemented.
3) This approach restricts the suite utilization to BBS+ signature suite, or every other suite that allows aggregation and selective disclosure.
4) This method requires that every broker can communicate with every creator involved in its own merged fragments.

10.3.2 Final Consideration about the Call Back approach

In almost all real-world use cases, due to the actual impossibility to satisfy all requirements of this approach, it is actually impracticable. Specifically:

- In the case when creators are Context Producers, they cannot be called back; they generate data and, optionally, sign it, and they send it to brokers, which cannot call them back.
- In the case when creators are Context Sources, they are registered to a broker and possibly can be called back, but due to the performance loss of multiple signing requests at every aggregation step occurring in the NGSI-LD federation, this approach still remains impracticable.

If a scenario, where all these limitations are not present, is possible, this approach could be a valid alternative.

10.4 Bridging DIDs and ETSI identifiers for juridical persons

A good candidate is the **Legal person Semantics Identifier** as defined in ETSI EN 319 412-1 [i.19], related to digital signatures, peer entity authentication, data authentication and data confidentiality. The standard identifier uses the following structure in the presented order:

- 3 character legal person identity type reference;
- 2 character ISO 3166-1 [i.20] country code;
- hyphen-minus "-" (0x2D (ASCII), U+002D (UTF-8)); and
- identifier (according to country and identity type reference).

The three initial characters have to have one of the following defined values:

1) "**VAT**" for identification based on a national value added tax identification number.
2) "**NTR**" for identification based on an identifier from a national trade register.
3) "**PSD**" for identification based on national authorization number of a payment service provider under Payments Services Directive (EU) 2015/2366 [i.21]. This needs to use the extended structure as defined in ETSI TS 119 495 [i.22], clause 5.2.1.
4) "**LEI**" for a global Legal Entity Identifier as specified in ISO 17442-1 [i.23]. The 2 character ISO 3166-1 [i.20] country code has to be set to 'XG'.
5) Two characters according to local definition within the specified country and name registration authority, identifying a national scheme that is considered appropriate for national and European level, followed by the character ":" (colon).

The ETSI Legal person Semantic Identifier covers any juridical entity that can receive a digital certificate in the EU, and can be used for any digital or physical transaction, making it very appropriate for using it as an identifier in a Verifiable Credential both as an issuer and as a subject.

In order to obtain the full benefits of DIDs, a DID can be deterministically generated from the ETSI Legal Person Identifier. For example:

```
did:elsi:VATFR-14348623562
```

could be the DID for ETSI, where 14348623562 is the EU Tax ID assigned by the French tax authority to ETSI, and recognized by all other EU countries (and most other countries in the world). The "elsi" method name stands for **E**(TSI) **L**(egal person) **S**(emantic) **I**(dentifier).

Regarding the remaining properties of a DID:

Persistent: the ETSI Legal person Semantic Identifier is assigned once and never changed until the legal person disappears.

Cryptographically verifiable: any Verifiable Credential signed by an entity with a digital certificate including its Legal person Semantic Identifier (as specified in ETSI EN 319 412-1 [i.19]) can be used to cryptographically verify the ownership (control) of the DID.

Resolvable: any ETSI DID registered in a blockchain can include a DID Document with all required public information about the legal person, including pointers to public records outside of the blockchain maintained by relevant authorities.

11 Prototyped Implementation of Digital Signatures

A prototype implementation of the Modular Approach has been developed in node.js, for test purposes.

For this implementation, the W3C Data integrity proof has been adopted as signature mechanism.

For this reason, the JavaScript library of the Linked Data Signatures specification for JSON-LD, for Node.js and browsers, called JSON-LD Signatures (jsonld-signatures) [i.24], implemented by the digitalbazaar group (https://github.com/digitalbazaar/jsonld-signatures), has been adopted.

The ed25519-signature-2020 suite has been utilized for signing NGSI-LD Entities.

Following the algorithm explained in clause 10.2.1, first of all, three NGSI-LD Entities with one single attribute (Atomic Entities) have been generated and signed:

1

```
{
  "@context": ["https://w3id.org/security/suites/ed25519-2020/v1"],
  "id": "https://example.org/car1",
  "type": "Car",
  "color": {
    "type": "Property",
    "value": "Red"
  },
  "proof": {…}
}
```

2

```
{
  "@context": ["https://w3id.org/security/suites/ed25519-2020/v1"],
  "id": "https://example.org/car1",
  "type": "Car",
  "speed": {
    "type": "Property",
    "value": "45"
  },
  "proof": {…}
}
```

3

```
{
  "@context": ["https://w3id.org/security/suites/ed25519-2020/v1"],
  "id": "https://example.org/car1",
  "type": "Car",
  "brand": {
    "type": "Property",
    "value": "BMW"
  },
  "proof": {…}
}
```

Then an implementation of the Derivation Process (as described in clause 10.2.1) will produce as output three entities with Sealed Attributes:

1

```
{
  "@context": ["https://w3id.org/security/suites/ed25519-2020/v1"],
  "id": "https://example.org/car1",
  "type": "Car",
  "color": {
    "type": "Property",
    "value": "Red",
    "ngsildproof": {
      "type": "Property",
      "entityIdSealed": "https://example.org/car1",
      "entityTypeSealed": "Car",
      "value": {
        "proof": {…}
      }
    }
  }
}
```

2

```
{
  "@context": ["https://w3id.org/security/suites/ed25519-2020/v1"],
  "id": "https://example.org/car1",
  "type": "Car",
  "speed": {
    "type": "Property",
    "value": "45",
    "ngsildproof": {
      "type": "Property",
      "entityIdSealed": "https://example.org/car1",
      "entityTypeSealed": "Car",
      "value": {
        "proof": {…}
      }
    }
  }
}
```

3

```
{
  "@context": ["https://w3id.org/security/suites/ed25519-2020/v1"],
  "id": "https://example.org/car1",
  "type": "Car",
  "brand": {
    "type": "Property",
    "value": "BMW",
    "ngsildproof": {
      "type": "Property",
      "entityIdSealed": "https://example.org/car1",
      "entityTypeSealed": "Car",
      "value": {
        "proof": {…}
      }
    }
  }
}
```

This will end the of the first phase of the NGSI-LD lifecycle, simulating the generation of three NGSI-LD attributes of the same Entity (same id), provided by three different context creators (with three different public keys).

After that, the merging algorithm has been implemented, simulating the context broker activity.

This will generate as output:

```
{
  "@context": ["https://w3id.org/security/suites/ed25519-2020/v1"],
  "id": "https://example.org/car1",
  "type": "Car",
  "color": {
    "type": "Property",
    "value": "Red",
    "ngsildproof": {
      "type": "Property",
      "entityIdSealed": "https://example.org/car1",
      "entityTypeSealed": "Car",
      "value": {
        "proof": {…}
      }
    }
  },
  "speed": {
    "type": "Property",
    "value": "45",
    "ngsildproof": {
      "type": "Property",
      "entityIdSealed": "https://example.org/car1",
      "entityTypeSealed": "Car",
      "value": {
        "proof": {…}
      }
    }
  },
  "brand": {
    "type": "Property",
    "value": "BMW",
    "ngsildproof": {
      "type": "Property",
      "entityIdSealed": "https://example.org/car1",
      "entityTypeSealed": "Car",
      "value": {
      "proof": {…}
      }
    }
  }
}
```

This entity has been produced separating all Sealed Attributes, coming from all entities in input, adding all entity types (if present) in a JSON array structure.

This will end the second phase.

At the end, simulating a Client's request, all original Entities have been recreated through the implementation of the Recreation process (opposite of the Derivation Process, see clause 10.2.1), using information contained in the `ngisldproof` object (`entityIdSealed, sentityTypeSealed`).

At the end all recreated Entities signatures have been verified.

History

Document history		
V1.1.1	September 2022	Publication