TODO Working Group
Internet-Draft
Intended status: Informational
Expires: 7 October 2022

S. Smith
ProSapien LLC

5 April 2022

Authentic Chained Data Containers (ACDC)
draft-ssmith-acdc-00

Abstract

An authentic chained data container (ACDC) [ACDC_ID][ACDC_WP][VCEnh] is an IETF [IETF] internet draft focused specification being incubated at the ToIP (Trust over IP) foundation [TOIP][ACDC_TF]. An ACDC is a variant of the W3C Verifiable Credential (VC) specification [W3C_VC]. The W3C VC specification depends on the W3C DID (Decentralized IDentifier) specification [W3C_DID]. A major use case for the ACDC specification is to provide GLEIF vLEIs (verifiable Legal Entity Identifiers) [vLEI][GLEIF_vLEI][GLEIF_KERI]. GLEIF is the Global Legal Entity Identifier Foundation [GLEIF]. ACDCs are dependent on a suite of related IETF focused standards associated with the KERI (Key Event Receipt Infrastructure) [KERI_ID][KERI] specification. These include CESR [CESR_ID], SAID [SAID_ID], PTEL [PTEL_ID], CESR-Proof [Proof_ID], IPEX [IPEX_ID], did:keri [DIDK_ID], and OOBI [OOBI_ID]. Some of the major distinguishing features of ACDCs include normative support for chaining, use of composable JSON Schema [JSch][JSchCp], multiple serialization formats, namely, JSON [JSON][RFC4627], CBOR [CBOR][RFC8949], MGPK [MGPK], and CESR [CESR_ID], support for Ricardian contracts [RC], support for chain-link confidentiality [CLC], a well defined security model derived from KERI [KERI][KERI_ID], _compact_ formats for resource constrained applications, simple _partial disclosure_ mechanisms and simple _selective disclosure_ mechanisms.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 7 October 2022.

Copyright Notice

Table of Contents

1. Introduction

The primary purpose of the ACDC protocol is to provide granular provenanced proof-of-authorship (authenticity) of their contained data via a tree or chain of linked ACDCs (technically a directed acyclic graph or DAG). Similar to the concept of a chain-of-custody, ACDCs provide a verifiable chain of proof-of-authorship of the contained data. With a little additional syntactic sugar, this primary facility of chained (treed) proof-of-authorship (authenticity) is extensible to a chained (treed) verifiable authentic proof-of-authority (proof-of-authorship-of-authority). A proof-of-authority may be used to provide verifiable authorizations or permissions or rights or credentials. A chained (treed) proof-of-authority enables delegation of authority and delegated authorizations.

The dictionary definition of *_credential_* is _evidence of authority, status, rights, entitlement to privileges, or the like_. Appropriately structured ACDCs may be used as credentials when their semantics provide verifiable evidence of authority. Chained ACDCs may provide delegated credentials.

Chains of ACDCs that merely provide proof-of-authorship (authenticity) of data may be appended to chains of ACDCs that provide proof-of-authority (delegation) to enable verifiable delegated authorized authorship of data. This is a vital facility for authentic data supply chains. Furthermore, any physical supply chain may be measured, monitored, regulated, audited, and/or archived by a data supply chain acting as a digital twin [Twin]. Therefore ACDCs provide the critical enabling facility for an authentic data economy and by association an authentic real (twinned) economy.

ACDCs act as securely attributed (authentic) fragments of a distributed _property graph_ (PG) [PGM][Dots]. Thus they may be used to construct knowledge graphs expressed as property graphs [KG]. ACDCs enable securely-attributed and privacy-protecting knowledge graphs.

The ACDC specification (including its partial and selective disclosure mechanisms) leverages two primary cryptographic operations namely digests and digital signatures [Hash][DSig]. These operations

when used in an ACDC MUST have a security level, cryptographic strength, or entropy of approximately 128 bits [Level]. (See the appendix for a discussion of cryptographic strength and security)

An important property of high-strength cryptographic digests is that a verifiable cryptographic commitment (such as a digital signature) to the digest of some data is equivalent to a commitment to the data itself. ACDCs leverage this property to enable compact chains of ACDCs that anchor data via digests. The data _contained_ in an ACDC may therefore be merely its equivalent anchoring digest. The anchored data is thereby equivalently authenticated or authorized by the chain of ACDCs.

2. ACDC Fields

An ACDC may be abstractly modeled as a nested key: value mapping. To avoid confusion with the cryptographic use of the term _key_ we instead use the term _field_ to refer to a mapping pair and the terms _field label_ and _field value_ for each member of a pair. These pairs can be represented by two tuples e.g (label, value). We qualify this terminology when necessary by using the term _field map_ to reference such a mapping. _Field maps_ may be nested where a given

field value is itself a reference to another _field map_. We call this nested set of fields a _nested field map_ or simply a _nested map_ for short. A _field_ may be represented by a framing code or block delimited serialization. In a block delimited serialization, such as JSON, each _field map_ is represented by an object block with block delimiters such as {} [RFC8259][JSON][RFC4627]. Given this equivalence, we may also use the term _block_ or _nested block_ as synonymous with _field map_ or _nested field map_. In many programming languages, a field map is implemented as a dictionary or hash table in order to enable performant asynchronous lookup of a _field value_ from its _field label_. Reproducible serialization of _field maps_ requires a canonical ordering of those fields. One such canonical ordering is called insertion or field creation order. A list of (field, value) pairs provides an ordered representation of any field map. Most programming languages now support ordered dictionaries or hash tables that provide reproducible iteration over a list of ordered field (label, value) pairs where the ordering is the insertion or field creation order. This enables reproducible round trip serialization/deserialization of _field maps_. ACDCs depend on insertion ordered field maps for canonical serialization/deserialization. ACDCs support multiple serialization types, namely JSON, CBOR, MGPK, and CESR but for the sake of simplicity, we will only use JSON herein for examples [RFC8259][JSON]. The basic set of normative field labels in ACDC field maps is defined in the following table.

2.1. Field Label Table

Label	Title	Description
v	Version String	Regexable format: ACDCvvSSSShhhhhh_ that provides protocol type, version, serialization type, size, and terminator.
d	Digest (SAID)	Self-referential fully qualified cryptographic digest of enclosing map.
i	Identifier (AID)	Semantics are determined by the context of its enclosing map.

u	UUID	Random Universally Unique IDentifier as fully qualified high entropy pseudo-random string, a salted nonce.
ri	Registry Identifier (AID)	Issuance and/or revocation, transfer, or retraction registry for ACDC.
s	Schema	Either the SAID of a JSON Schema block or the block itself.
a	Attribute	Either the SAID of a block of attributes or the block itself.
A	Attribute Aggregate	Either the Aggregate of a selectively disclosable block of attributes or the block itself.
e	Edge	Either the SAID of a block of edges or the block itself.
r	Rule	Either the SAID a block of rules or the block itself.
n	Node	SAID of another ACDC as the terminating point of a directed edge that connects the encapsulating ACDC node to the specified ACDC node as a fragment of a distributed property graph (PG).
l	Legal Language	Text of Ricardian contract clause.

Table 1

2.2. Compact Labels

The primary field labels are compact in that they use only one or two characters. ACDCs are meant to support resource-constrained

applications such as supply chain or IoT (Internet of Things) applications. Compact labels better support resource-constrained applications in general. With compact labels, the over-the-wire verifiable signed serialization consumes a minimum amount of bandwidth. Nevertheless, without loss of generality, a one-to-one normative semantic overlay using more verbose expressive field labels may be applied to the normative compact labels after verification of the over-the-wire serialization. This approach better supports bandwidth and storage constraints on transmission while not precluding any later semantic post-processing. This is a well-known design pattern for resource-constrained applications.

2.3. Version String Field

The version string, v, field MUST be the first field in any top-level ACDC field map. It provides a regular expression target for determining the serialization format and size (character count) of a serialized ACDC. A stream-parser may use the version string to extract and deserialize (deterministically) any serialized ACDC in a stream of serialized ACDCs. Each ACDC in a stream may use a different serialization type.

The format of the version string is ACDCvvSSSShhhhhh_. The first four characters ACDC indicate the enclosing field map serialization. The next two characters, vv provide the lowercase hexadecimal notation for the major and minor version numbers of the version of the ACDC specification used for the serialization. The first v provides the major version number and the second v provides the minor version number. For example, 01 indicates major version 0 and minor version 1 or in dotted-decimal notation 0.1. Likewise 1c indicates major version 1 and minor version decimal 12 or in dotted-decimal notation 1.12. The next four characters SSSS indicate the serialization type in uppercase. The four supported serialization types are JSON, CBOR, MGPK, and CESR for the JSON, CBOR, MessagePack, and CESR serialization standards respectively [JSON][RFC4627][CBOR][RFC8949][MGPK][CESR_ID]. The next six characters provide in lowercase hexadecimal notation the total number of characters in the serialization of the ACDC. The maximum length of a given ACDC is thereby constrained to be _2^24 = 16,777,216_ characters in length. The final character - is the version string

terminator. This enables later versions of ACDC to change the total

version string size and thereby enable versioned changes to the composition of the fields in the version string while preserving deterministic regular expression extractability of the version string. Although a given ACDC serialization type may have a field map delimiter or framing code characters that appear before (i.e. prefix) the version string field in a serialization, the set of possible prefixes is sufficiently constrained by the allowed serialization protocols to guarantee that a regular expression can determine unambiguously the start of any ordered field map serialization that includes the version string as the first field value. Given the version string, a parser may then determine the end of the serialization so that it can extract the full ACDC from the stream without first deserializing it. This enables performant stream parsing and off-loading of ACDC streams that include any or all of the supported serialization types.

2.4. AID (Autonomic IDentifier) Fields

Some fields, such as the i and ri fields, MUST each have an AID (Autonomic IDentifier) as its value. An AID is a fully qualified Self-Certifying IDentifier (SCID) that follows the KERI protocol [KERI][KERI_ID]. A SCID is derived from one or more (public, private) key pairs using asymmetric or public-key cryptography to create verifiable digital signatures [DSig]. Each AID has a set of one or more controllers who each control a private key. By virtue of their private key(s), the set of controllers may make statements on behalf of the associated AID that is backed by uniquely verifiable commitments via digital signatures on those statements. Any entity may then verify those signatures using the associated set of public keys. No shared or trusted relationship between the controllers and verifiers is required. The verifiable key state for AIDs is established with the KERI protocol [KERI][KERI_ID]. The use of AIDS enables ACDCs to be used in a portable but securely attributable, fully decentralized manner in an ecosystem that spans trust domains.

2.4.1. Namespaced AIDs

Because KERI is agnostic about the namespace for any particular AID, different namespace standards may be used to express KERI AIDs within AID fields in an ACDC. The examples below use the W3C DID namespace specification with the did:keri method [DIDK_ID]. But the examples would have the same validity from a KERI perspective if some other supported namespace was used or no namespace was used at all. The latter case consists of a bare KERI AID (identifier prefix).

2.5. SAID (Self-Addressing IDentifier) Fields

Some fields in ACDCs may have for their value either a _field map_ or a SAID. A SAID follows the SAID protocol [SAID_ID]. Essentially a SAID is a Self-Addressing IDentifier (self-referential content addressable). A SAID is a special type of cryptographic digest of its encapsulating _field map_ (block). The encapsulating block of a SAID is called a SAD (Self-Addressed Data). Using a SAID as a _field value_ enables a more compact but secure representation of the associated block (SAD) from which the SAID is derived. Any nested field map that includes a SAID field (i.e. is, therefore, a SAD) may be compacted into its SAID. The uncompacted blocks for each associated SAID may be attached or cached to optimize bandwidth and availability without decreasing security.

Several top-level ACDC fields may have for their value either a serialized _field map_ or the SAID of that _field map_. Each SAID provides a stable universal cryptographically verifiable and agile reference to its encapsulating block (serialized _field map_). Specifically, the value of top-level s, a, e, and r fields may be replaced by the SAID of their associated _field map_. When replaced by their SAID, these top-level sections are in _compact_ form.

Recall that a cryptographic commitment (such as a digital signature or cryptographic digest) on a given digest with sufficient cryptographic strength including collision resistance [HCR][QCHC] is equivalent to a commitment to the block from which the given digest was derived. Specifically, a digital signature on a SAID makes a verifiable cryptographic non-repudiable commitment that is equivalent to a commitment on the full serialization of the associated block from which the SAID was derived. This enables reasoning about ACDCs in whole or in part via their SAIDS in a fully interoperable, verifiable, compact, and secure manner. This also supports the well-known bow-tie model of Ricardian Contracts [RC]. This includes reasoning about the whole ACDC given by its top-level SAID, d, field as well as reasoning about any nested sections using their SAIDS.

2.6. Selectively Disclosable Attribute Aggregate Field

The top-level selectively-disclosable attribute aggregate section, A, field value is an aggregate of cryptographic commitments used to make a commitment to a set (bundle) of selectively-disclosable attributes. The value of the attribute aggregate, A, field depends on the type of selective disclosure mechanism employed. For example, the aggregate value could be the cryptographic digest of the concatenation of an ordered set of cryptographic digests, a Merkle tree root digest of an

ordered set of cryptographic digests, or a cryptographic accumulator.

2.7. UUID (Universally Unique IDentifier) Fields

The purpose of the UUID, u, field in any block is to provide sufficient entropy to the SAID, d, field of the associated block to make computationally infeasible any brute force attacks on that block that attempt to discover the block contents from the schema and the SAID. The UUID, u, field may be considered a salty nonce [Salt]. Without the entropy provided the UUID, u, field, an adversary may be able to reconstruct the block contents merely from the SAID of the block and the schema of the block using a rainbow or dictionary attack on the set of field values allowed by the schema [RB][DRB]. The effective security level, entropy, or cryptographic strength of the schema-compliant field values may be much less than the cryptographic strength of the SAID digest. Another way of saying this is that the cardinality of the power set of all combinations of allowed field values may be much less than the cryptographic strength of the SAID. Thus an adversary could successfully discover via brute force the exact block by creating digests of all the elements of the power set which may be small enough to be computationally feasible instead of inverting the SAID itself. Sufficient entropy in the u field ensures that the cardinality of the power set allowed by the schema is at least as great as the entropy of the SAID digest algorithm itself.

A UUID, u field may optionally appear in any block (field map) at any level of an ACDC. Whenever a block in an ACDC includes a UUID, u, field then it's associated SAID, d, field makes a blinded commitment to the contents of that block. The UUID, u, field is the blinding factor. This makes that block securely partially-disclosable or even selectively-disclosable notwithstanding disclosure of the associated schema of the block. The block contents can only be discovered given disclosure of the included UUID field. Likewise when a UUID, u, field appears at the top level of an ACDC then that top-level SAID, d, field makes a blinded commitment to the contents of the whole ACDC itself. Thus the whole ACDC, not merely some block within the ACDC, may be disclosed in a privacy-preserving (correlation minimizing) manner.

2.8. Full, Partial, and Selective Disclosure

The difference between *_partial disclosure_* and *_selective disclosure_* of a given field map is determined by the correlatability of the disclosed field(s) after *_full disclosure_* of the detailed field value with respect to its enclosing block (map or array of fields). A _partially disclosable_ field becomes correlatable after _full disclosure_. Whereas a _selectively disclosable_ field may be excluded from the _full disclosure_ of any other _selectively disclosable_ fields in the _selectively

disclosable_ block (array). After such _selective disclosure_, the selectively disclosed fields are not correlatable to the so-far undisclosed but selectively disclosable fields in that block.

When used in the context of _selective disclosure_, _full disclosure_ means detailed disclosure of the selectively disclosed attributes not detailed disclosure of all selectively disclosable attributes. Whereas when used in the context of _partial disclosure_, _full disclosure_ means detailed disclosure of the field map that was so far only partially disclosed.

Partial disclosure is an essential mechanism needed to support both performant exchange of information and chain-link confidentiality on exchanged information [CLC]. The exchange of only the SAID of a given field map is a type of _partial disclosure_. Another type of _partial disclosure_ is the disclosure of validatable metadata about a detailed field map e.g. the schema of a field map.

The SAID of a field map provides a _compact_ cryptographically equivalent commitment to the yet to be undisclosed field map details. A later exchange of the uncompacted field map detail provides _full disclosure_. Any later _full disclosure_ is verifiable to an earlier _partial disclosure_. Partial disclosure via compact SAIDs enables the scalable repeated verifiable exchange of SAID references to cached full disclosures. Multiple SAID references to cached fully disclosed field maps may be transmitted compactly without redundant retransmission of the full details each time a new reference is transmitted. Likewise, _partial disclosure_ via SAIDs also supports the bow-tie model of Ricardian contracts [RC]. Similarly, the schema of a field map is metadata about the structure of the field map this is validatable given the full disclosure of the field map. The details of_compact_ and/or confidential exchange mechanisms that

leverage partial disclosure are explained later.

Selective disclosure, on the other hand, is an essential mechanism needed to unbundle in a correlation minimizing way a single commitment by an Issuer to a bundle of fields (i.e. a nested array or list or tuple of fields) as a whole. This allows separating a "stew" (bundle) of "ingredients" (attributes) into its constituent "ingredients" (attributes) without correlating the constituents via the Issuer's commitment to the "stew" (bundle) as a whole.

3. Schema Section

3.1. Schema is Type

Notable is the fact that there are no top-level type fields in an ACDC. This is because the schema, s, field itself is the type field. ACDCs follow the design principle of separation of concerns between a data container's actual payload information and the type information of that container's payload. In this sense, type information is metadata, not data. The schema dialect is JSON Schema 2020-12 [JSch][JSch_202012]. JSON Schema support for composable schema (sub-schema), conditional schema (sub-schema), and regular expressions in schema enable a validator to ask and answer complex questions about the type of even optional payload elements while maintaining isolation between payload information and type (structure) information about the payload [JSchCp][JSchRE][JSchId][JSchCx]. ACDC's use of JSON Schema MUST be in accordance with the ACDC defined profile as defined herein. The exceptions are defined below.

3.2. Schema ID Field Label

The usual field label for SAID fields in ACDCs is d. In the case of the schema section, however, the field label for the SAID of the schema section is $id. This repurposes the schema id field label, $id as defined by JSON Schema [JSchId][JSchCx]. The top-level id, $id, field value in a JSON Schema provides a unique identifier of the schema instance. In a usual (non-ACDC) schema the value of the id,

$id, field is expressed as a URI. This is called the _Base URI_ of the schema. In an ACDC schema, however, the top-level id, $id, field value is repurposed. Its value MUST include the SAID of the schema. This ensures that the ACDC schema is static and verifiable to their SAIDS. A verifiably static schema satisfies one of the essential security properties of ACDCs as discussed below. There are several ACDC supported formats for the value of the top-level id, $id, field but all of the formats MUST include the SAID of the schema (see below). Correspondingly, the value of the top-level schema, s, field MUST be the SAID included in the schema's top-level $id field. The detailed schema is either attached or cached and maybe discovered via its SAIDified, id, $id, field value.

When an id, '$id', field appears in a sub-schema it indicates a bundled sub-schema called a schema resource [JSchId][JSchCx]. The value of the id, '$id', field in any ACDC bundled sub-schema resource MUST include the SAID of that sub-schema using one of the formats described below. The sub-schema so bundled MUST be verifiable against its referenced and embedded SAID value. This ensures secure bundling.

3.3. Static Schema

For security reasons, the full schema of an ACDC must be completely self-contained and statically fixed (immutable) for that ACDC. By this, we mean that no dynamic schema references or dynamic schema generation mechanisms are allowed.

Should an adversary successfully attack the source that provides the dynamic schema resource and change the result provided by that reference, then the schema validation on any ACDC that uses that dynamic schema reference may fail. Such an attack effectively revokes all the ACDCs that use that dynamic schema reference. We call this a *_schema revocation_* attack.

More insidiously, an attacker could shift the semantics of the dynamic schema in such a way that although the ACDC still passes its schema validation, the behavior of the downstream processing of that ACDC is changed by the semantic shift. This we call a *_semantic

malleability_* attack. It may be considered a new type of _transaction malleability_ attack [TMal].

To prevent both forms of attack, all schema must be static, i.e. schema MUST be SADs and therefore verifiable against their SAIDs.

To elaborate, the serialization of a static schema may be self-contained. A compact commitment to the detailed static schema may be provided by its SAID. In other words, the SAID of a static schema is a verifiable cryptographic identifier for its SAD. Therefore all ACDC compliant schema must be SADs. In other words, they MUST therefore be _SAIDified_. The associated detailed static schema (uncompacted SAD) is cryptographically bound and verifiable to its SAID.

The JSON Schema specification allows complex schema references that may include non-local URI references [JSchId][JSchCx]. These references may use the $id or $ref keywords. A relative URI reference provided by a $ref keyword is resolved against the _Base URI_ provided by the top-level $id field. When this top-level _Base URI_ is non-local then all relative $ref references are therefore also non-local. A non-local URI reference provided by a $ref keyword may be resolved without reference to the _Base URI_.

In general, schema indicated by non-local URI references ($id or $ref) MUST NOT be used because they are not cryptographically end-verifiable. The value of the underlying schema resource so referenced may change (mutate). To restate, a non-local URI schema resource is not end-verifiable to its URI reference because there is no cryptographic binding between URI and resource.

This does not preclude the use of remotely cached SAIDified schema resources because those resources are end-verifiable to their embedded SAID references. Said another way, a SAIDified schema resource is itself a SAD (Self-Address Data) referenced by its SAID. A URI that includes a SAID may be used to securely reference a remote or distributed SAIDified schema resource because that resource is fixed (immutable, nonmalleable) and verifiable to both the SAID in the reference and the embedded SAID in the resource so referenced. To elaborate, a non-local URI reference that includes an embedded cryptographic commitment such as a SAID is verifiable to the underlying resource when that resource is a SAD. This applies to

JSON Schema as a whole as well as bundled sub-schema resources.

There ACDC supported formats for the value of the top-level id, $id, field are as follows:

* Bare SAIDs may be used to refer to a SAIDified schema as long as the JSON schema validator supports bare SAID references. By default, many if not all JSON schema validators support bare strings (non-URIs) for the _Base URI_ provided by the top-level $id field value.

* The sad: URI scheme may be used to directly indicate a URI resource that safely returns a verifiable SAD. For example sad:SAID where _SAID_ is replaced with the actual SAID of a SAD that provides a verifiable non-local reference to JSON Schema as indicated by the mime-type of schema+json.

* The IETF KERI OOBI internet draft specification provides a URL syntax that references a SAD resource by its SAID at the service endpoint indicated by that URL [OOBI ID]. Such remote OOBI URLs are also safe because the provided SAD resource is verifiable against the SAID in the OOBI URL. Therefore OOBI URLs are also acceptable non-local URI references for JSON Schema.

* The did: URI scheme may be used safely to prefix non-local URI references that act to namespace SAIDs expressed as DID URIs or DID URLs. DID resolvers resolve DID URLs for a given DID method such as did:keri [DIDK ID] and may return DID docs or DID doc metadata with SAIDified schema or service endpoints that return SAIDified schema. A verifiable non-local reference in the form of DID URL that includes the schema SAID is resolved safely when it dereferences to the SAD of that SAID. For example, the resolution result returns an ACDC JSON Schema whose id, $id, field includes the SAID and returns a resource with JSON Schema mime-type of schema+json.

To clarify, ACDCs MUST NOT use complex JSON Schema references which allow *dynamically generated *schema resources to be obtained from online JSON Schema Libraries [JSchId][JSchCx]. The latter approach may be difficult or impossible to secure because a cryptographic

commitment to the base schema that includes complex schema (non-relative URI-based) references only commits to the non-relative URI reference and not to the actual schema resource which may change (is dynamic, mutable, malleable). To restate, this approach is insecure because a cryptographic commitment to a complex (non-relative URI-based) reference is NOT equivalent to a commitment to the detailed associated schema resource so referenced if it may change.

ACDCs MUST use static JSON Schema (i.e. _SAIDifiable_ schema). These may include internal relative references to other parts of a fully self-contained static (_SAIDified_) schema or references to static (_SAIDified_) external schema parts. As indicated above, these references may be bare SAIDs, DID URIs or URLs (did: scheme), SAD URIs (sad: scheme), or OOBI URLs. Recall that a commitment to a SAID with sufficient collision resistance makes an equivalent secure commitment to its encapsulating block SAD. Thus static schema may be either fully self-contained or distributed in parts but the value of any reference to a part must be verifiably static (immutable, nonmalleable) by virtue of either being relative to the self-contained whole or being referenced by its SAID. The static schema in whole or in parts may be attached to the ACDC itself or provided via a highly available cache or data store. To restate, this approach is securely end-verifiable (zero-trust) because a cryptographic commitment to the SAID of a SAIDified schema is equivalent to a commitment to the detailed associated schema itself (SAD).

3.4. Schema Dialect

The schema dialect for ACDC 1.0 is JSON Schema 2020-12 and is indicated by the identifier "https://json-schema.org/draft/2020-12/schema" [JSch][JSch_202012]. This is indicated in a JSON Schema via the value of the top-level $schema field. Although the value of $schema is expressed as a URI, de-referencing does not provide dynamically downloadable schema dialect validation code. This would be an attack vector. The validator MUST control the tooling code dialect used for schema validation and hence the tooling dialect version actually used. A mismatch between the supported tooling code dialect version and the $schema string value should cause the validation to fail. The string is simply an identifier that communicates the intended dialect to be processed by the schema validation tool. When provided, the top-level $schema field value for ACDC version 1.0 must be "https://json-schema.org/draft/2020-12/schema".

3.5. Schema Availablity

The composed detailed (uncompacted) (bundled) static schema for an ACDC may be cached or attached. But cached, and/or attached static schema is not to be confused with dynamic schema. Nonetheless, while securely verifiable, a remotely cached, _SAIDified_, schema resource may be unavailable. Availability is a separate concern. Unavailable does not mean insecure or unverifiable. ACDCs MUST be verifiable when available. Availability is typically solvable through redundancy. Although a given ACDC application domain or eco-system governance framework may impose schema availability constraints, the ACDC specification itself does not impose any specific availability requirements on Issuers other than schema caches SHOULD be sufficiently available for the intended application of their associated ACDCs. It's up to the Issuer of an ACDC to satisfy any availability constraints on its schema that may be imposed by the application domain or eco-system.

3.6. Composable JSON Schema

A composable JSON Schema enables the use of any combination of compacted/uncompacted attribute, edge, and rule sections in a provided ACDC. When compact, any one of these sections may be represented merely by its SAID [JSch][JSchCp]. When used for the top-level attribute, a, edge, e, or rule, r, section field values, the oneOf sub-schema composition operator provides both compact and uncompacted variants. The provided ACDC MUST validate against an allowed combination of the composed variants, either the compact SAID of a block or the full detailed (uncompacted) block for each section. The validator determines what decomposed variants the provided ACDC MUST also validate against. Decomposed variants may be dependent on the type of disclosure, partial, full, or selective.

Unlike the other compactifiable sections, it is impossible to define recursively the exact detailed schema as a variant of a oneOf composition operator contained in itself. Nonetheless, the provided schema, whether self-contained, attached, or cached MUST validate as a SAD against its provided SAID. It MUST also validate against one of its specified oneOf variants.

The compliance of the provided non-schema attribute, a, edge, e, and rule, r, sections MUST be enforced by validating against the composed schema. In contrast, the compliance of the provided composed schema for an expected ACDC type MUST be enforced by the validator. This is because it is not possible to enforce strict compliance of the schema by validating it against itself.

ACDC specific schema compliance requirements are usually specified in the eco-system governance framework for a given ACDC type. Because the SAID of a schema is a unique content-addressable identifier of the schema itself, compliance can be enforced by comparison to the allowed schema SAID in a well-known publication or registry of ACDC types for a given ecosystem governance framework (EGF). The EGF may be solely specified by the Issuer for the ACDCs it generates or be specified by some mutually agreed upon eco-system governance mechanism. Typically the business logic for making a decision about a presentation of an ACDC starts by specifying the SAID of the composed schema for the ACDC type that the business logic is expecting from the presentation. The verified SAID of the actually presented schema is then compared against the expected SAID. If they match then the actually presented ACDC may be validated against any desired decomposition of the expected (composed) schema.

To elaborate, a validator can confirm compliance of any non-schema section of the ACDC against its schema both before and after uncompacted disclosure of that section by using a composed base schema with oneOf pre-disclosure and a decomposed schema post-disclosure with the compact oneOf option removed. This capability provides a mechanism for secure schema validation of both compact and uncompacted variants that require the Issuer to only commit to the composed schema and not to all the different schema variants for each combination of a given compact/uncompacted section in an ACDC.

One of the most important features of ACDCs is support for Chain-Link Confidentiality [CLC]. This provides a powerful mechanism for protecting against un-permissioned exploitation of the data disclosed via an ACDC. Essentially an exchange of information compatible with chain-link confidentiality starts with an offer by the discloser to disclose confidential information to a potential disclosee. This offer includes sufficient metadata about the information to be disclosed such that the disclosee can agree to those terms. Specifically, the metadata includes both the schema of the information to be disclosed and the terms of use of that data once disclosed. Once the disclosee has accepted the terms then full disclosure is made. A full disclosure that happens after contractual acceptance of the terms of use we call _permissioned_ disclosure. The pre-acceptance disclosure of metadata is a form of partial

disclosure.

As is the case for compact (uncompacted) ACDC disclosure, Composable JSON Schema, enables the use of the same base schema for both the validation of the partial disclosure of the offer metadata prior to contract acceptance and validation of full or detailed disclosure after contract acceptance [JSch][JSchCp]. A cryptographic commitment to the base schema securely specifies the allowable semantics for

both partial and full disclosure. Decomposition of the base schema enables a validator to impose more specific semantics at later stages of the exchange process. Specifically, the oneOf sub-schema composition operator validates against either the compact SAID of a block or the full block. Decomposing the schema to remove the optional compact variant enables a validator to ensure complaint full disclosure. To clarify, a validator can confirm schema compliance both before and after detailed disclosure by using a composed base schema pre-disclosure and a decomposed schema post-disclosure with the undisclosed options removed. These features provide a mechanism for secure schema-validated contractually-bound partial (and/or selective) disclosure of confidential data via ACDCs.

4. ACDC Variants

There are several variants of ACDCs determined by the presence/ absence of certain fields and/or the value of those fields. At the top level, the presence (absence), of the UUID, u, field produces two variants. These are private (public) respectively. In addition, a present but empty UUID, u, field produces a private metadata variant.

4.1. Public ACDC

Given that there is no top-level UUID, u, field in an ACDC, then knowledge of both the schema of the ACDC and the top-level SAID, d, field may enable the discovery of the remaining contents of the ACDC via a rainbow table attack [RB][DRB]. Therefore, although the top-level, d, field is a cryptographic digest, it may not securely blind the contents of the ACDC when knowledge of the schema is available. The field values may be discoverable. Consequently, any cryptographic commitment to the top-level SAID, d, field may provide a fixed point of correlation potentially to the ACDC field values themselves in spite of non-disclosure of those field values. Thus an

ACDC without a top-level UUID, u, field must be considered a *_public_* (non-confidential) ACDC.

4.2. Private ACDC

Given a top-level UUID, u, field, whose value has sufficient cryptographic entropy, then the top-level SAID, d, field of an ACDC may provide a secure cryptographic digest that blinds the contents of the ACDC [Hash]. An adversary when given both the schema of the ACDC and the top-level SAID, d, field, is not able to discover the remaining contents of the ACDC in a computationally feasible manner such as through a rainbow table attack [RB][DRB]. Therefore the top-level, UUID, u, field may be used to securely blind the contents of the ACDC notwithstanding knowledge of the schema and top-level, SAID, d, field. Moreover, a cryptographic commitment to that that top-

level SAID, d, field does not provide a fixed point of correlation to the other ACDC field values themselves unless and until there has been a disclosure of those field values. Thus an ACDC with a sufficiently high entropy top-level UUID, u, field may be considered a *_private_* (confidential) ACDC. enables a verifiable commitment to the top-level SAID of a private ACDC to be made prior to the disclosure of the details of the ACDC itself without leaking those contents. This is called _partial_ disclosure. Furthermore, the inclusion of a UUID, u, field in a block also enables _selective_ disclosure mechanisms described later in the section on selective disclosure.

4.3. Metadata ACDC

An empty, top-level UUID, u, field appearing in an ACDC indicates that the ACDC is a *_metadata_* ACDC. The purpose of a _metadata_ ACDC is to provide a mechanism for a _Discloser_ to make cryptographic commitments to the metadata of a yet to be disclosed private ACDC without providing any point of correlation to the actual top-level SAID, d, field of that yet to be disclosed ACDC. The top-level SAID, d, field, of the metadata ACDC, is cryptographically derived from an ACDC with an empty top-level UUID, u, field so its value will necessarily be different from that of an ACDC with a high entropy top-level UUID, u, field value. Nonetheless, the _Discloser_ may make a non-repudiable cryptographic commitment to the metadata SAID in order to initiate a chain-link confidentiality exchange

without leaking correlation to the actual ACDC to be disclosed [CLC]. A _Disclosee_ (verifier) may validate the other metadata information in the metadata ACDC before agreeing to any restrictions imposed by the future disclosure. The metadata includes the _Issuer_, the _schema_, the provenancing _edges_, and the _rules_ (terms-of-use). The top-level attribute section, a, field value of a _metadata_ ACDC may be empty so that its value is not correlatable across disclosures (presentations). Should the potential _Disclosee_ refuse to agree to the rules then the _Discloser_ has not leaked the SAID of the actual ACDC or the SAID of the attribute block that would have been disclosed.

Given the _metadata_ ACDC, the potential _Disclosee_ is able to verify the _Issuer_, the schema, the provenanced edges, and rules prior to agreeing to the rules. Similarly, an _Issuer_ may use a _metadata_ ACDC to get agreement to a contractual waiver expressed in the rule section with a potential _Issuee_ prior to issuance. Should the _Issuee_ refuse to accept the terms of the waiver then the _Issuer_ has not leaked the SAID of the actual ACDC that would have been issued nor the SAID of its attributes block nor the attribute values themselves.

When a _metadata_ ACDC is disclosed (presented) only the _Discloser's_ signature(s) is attached not the _Issuer's_ signature(s). This precludes the _Issuer's_ signature(s) from being used as a point of correlation until after the _Disclosee_ has agreed to the terms in the rule section. When chain-link confidentiality is used, the _Issuer's_ signatures are not disclosed to the _Disclosee_ until after the _Disclosee_ has agreed to keep them confidential. The _Disclosee_ is protected from forged _Discloser_ because ultimately verification of the disclosed ACDC will fail if the _Discloser_ does not eventually provide verifiable _Issuer's_ signatures. Nonetheless, should the potential _Disclosee_ not agree to the terms of the disclosure expressed in the rule section then the _Issuer's_ signature(s) is not leaked.

5. Unpermissioned Exploitation of Data

An important design goal of ACDCs is they support the sharing of provably authentic data while also protecting against the un-permissioned exploitation of that data. Often the term _privacy

protection_ is used to describe similar properties. But a narrow focus on "privacy protection" may lead to problematic design trade-offs. With ACDCs, the primary design goal is not _data privacy protection_ per se but the more general goal of protection from the *_un-permissioned exploitation of data_*. In this light, a _given privacy protection_ mechanism may be employed to help protect against _unpermissioned exploitation of data_ but only when it serves that more general-purpose and not as an end in and of itself. There are three primary mechanisms ACDCs use to protect against _unpermissioned exploitation of data_. These are:

* Chain-link Confidentiality [CLC]

* Partial Disclosure

* Selective Disclosure

5.1. Principle of Least Disclosure

ACDCs are designed to satisfy the principle of least disclosure.

> The system should disclose only the minimum amount of information about a given party needed to facilitate a transaction and no more. [IDSys]

For example, the _partial disclosure_ of portions of an ACDC to enable chain-link confidentiality of the subsequent full disclosure is an application of the principle of least disclosure. Likewise, unbundling only the necessary attributes from a bundled commitment

using _selective disclosure_ to enable a correlation minimizing disclosure from that bundle is an application of the principle of least disclosure.

5.2. Three Party Exploitation Model

Unpermission exploitation is characterized using a three-party model. The three parties are as follows:

* First-Party = _Discloser_ of data.

* Second-Party = _Disclosee_ of data received from First Party

(_Discloser_).

* Third-Party = _Observer_ of data disclosed by First Party (_Discloser_) to Second Party (_Disclosee_).

5.2.1. Second-Party (Disclosee) Exploitation

* implicit permissioned correlation.
  - no contractual restrictions on the use of disclosed data.
* explicit permissioned correlation.
  - use as permitted by contract
* explicit unpermissioned correlation with other second parties or third parties.
  - malicious use in violation of contract

5.2.2. Third-Party (Observer) Exploitation

* implicit permissioned correlation.
  - no contractual restrictions on use of observed data.
* explicit unpermissioned correlation via collusion with second parties.
  - malicious use in violation of second party contract

5.3. Chain-link Confidentiality Exchange

Chain-link confidentiality imposes contractual restrictions and liability on any Disclosee (Second-Party) [CLC]. The exchange provides a fair contract consummation mechanism. The steps in a

chain-link confidentiality exchange are as follows:

* _Discloser_ provides a non-repudiable _Offer_ with verifiable metadata (sufficient partial disclosure) which includes any terms or restrictions on use.

* _Disclosee_ verifies _Offer_ against composed schema and metadata adherence to desired data.

* _Disclosee_ provides non-repudiable _Accept_ of terms that are contingent on compliant disclosure.

* _Discloser_ provides non-repudiable _Disclosure_ with sufficient compliant detail.

* _Disclosee_ verifies _Disclosure_ using decomposed schema and adherence of disclosed data to _Offer_.

Disclosee may now engage in permissioned use and carries liability as a deterrent against unpermissioned use.

6. Compact ACDC

The top-level section field values of a compact ACDC are the SAIDs of each uncompacted top-level section. The section field labels are s, a, e, and r.

6.1. Compact Public ACDC

A fully compact public ACDC is shown below.

```
{
  "v":  "ACDC10JSON00011c_",
  "d":  "EBdXt3gIXOf2BBWNHdSXCJnFJL5OuQPyM5K0neuniccM",
  "i":  "did:keri:EmkPreYpZfFk66jpf3uFv7vklXKhzBrAqjsKAn2EDIPM",
  "ri": "did:keri:EymRy7xMwsxUelUauaXtMxTfPAMPAI6FkekwlOjkggt",
  "s":  "E46jrVPTzlSkUPqGGeIZ8a8FWS7a6s4reAXRZOkogZ2A",
  "a":  "EgveY4-9XgOcLxUderzwLIr9Bf7V_NHwY1lkFrn9y2PY",
  "e":  "ERH3dCdoFOLe71iheqcywJcnjtJtQIYPvAu6DZIl3MOA",
  "r":  "Ee71iheqcywJcnjtJtQIYPvAu6DZIl3MORH3dCdoFOLB"
}
```

6.2. Compact Private ACDC

A fully compact private ACDC is shown below.

```
{
  "v":  "ACDC10JSON00011c_",
  "d":  "EBdXt3gIXOf2BBWNHdSXCJnFJL5OuQPyM5K0neuniccM",
  "u":  "0ANghkDaG7OY1wjaDAE0qHcg",
  "i":  "did:keri:EmkPreYpZfFk66jpf3uFv7vklXKhzBrAqjsKAn2EDIPM",
  "ri": "did:keri:EymRy7xMwsxUelUauaXtMxTfPAMPAI6FkekwlOjkggt",
  "s":  "E46jrVPTzlSkUPqGGeIZ8a8FWS7a6s4reAXRZOkogZ2A",
  "a":  "EgveY4-9XgOcLxUderzwLIr9Bf7V_NHwY1lkFrn9y2PY",
  "e":  "ERH3dCdoFOLe71iheqcywJcnjtJtQIYPvAu6DZIl3MOA",
  "r":  "Ee71iheqcywJcnjtJtQIYPvAu6DZIl3MORH3dCdoFOLB"
}
```

6.2.1. Compact Private ACDC Schema

The schema for the compact private ACDC example above is provided below.

```
{
  "$id": "EN8i2i5ye0-xGS95pm5cg1j0GmFkarJe0zzsSrrf4XJY",
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Compact Private ACDC",
  "description": "Example JSON Schema for a Compact Private ACDC.",
  "credentialType": "CompactPrivateACDCExample",
  "type": "object",
  "required":
  [
    "v",
    "d",
    "u",
    "i",
    "ri",
    "s",
    "a",
    "e",
    "r"
  ],
  "properties":
  {
    "v":
    {
      "description": "ACDC version string",
      "type": "string"
    },
    "d":
```

```
      {
       "description": "ACDC SAID",
        "type": "string"
      },
      "u":
      {
       "description": "ACDC UUID",
        "type": "string"
      },
      "i":
      {
        "description": "Issuer AID",
        "type": "string"
      },
      "ri":
      {
        "description": "credential status registry AID",
        "type": "string"
      },
      "s": {
        "description": "schema SAID",
        "type": "string"
      },
      "a": {
        "description": "attribute SAID",
        "type": "string"
      },
      "e": {
        "description": "edge SAID",
        "type": "string"
      },
      "r": {
        "description": "rule SAID",
        "type": "string"
      },
    },
    "additionalProperties": false
  }
```

7. Attribute Section

The attribute section in the examples above has been compacted into

its SAID. The schema of the compacted attribute section is as follows,


```
{
  "a":
  {
    "description": "attribute section SAID",
    "type": "string"
  }
}
```

Two variants of an ACDC, namely, namely, *_private (public) attribute_* are defined respectively by the presence (absence) of a UUID, u, field in the uncompacted attribute section block.

Two other variants of an ACDC, namely, *_targeted (untargeted)_* are defined respectively by the presence (absence) of an issuee, i, field in the uncompacted attribute section block.

7.1. Public-Attribute ACDC

Suppose that the un-compacted value of the attribute section as denoted by the attribute section, a, field is as follows,

```
{
  "a":
  {
    "d": "EgveY4-9XgOcLxUderzwLIr9Bf7V_NHwY1lkFrn9y2PY",
    "i": "did:keri:EpZfFk66jpf3uFv7vklXKhzBrAqjsKAn2EDIPmkPreYA",
    "score": 96,
    "name": "Jane Doe"
  }
}
```

The SAID, d, field at the top level of the uncompacted attribute block is the same SAID used as the compacted value of the attribute section, a, field.

Given the absence of a u field at the top level of the attributes block, then knowledge of both SAID, d, field at the top level of an attributes block and the schema of the attributes block may enable the discovery of the remaining contents of the attributes block via a rainbow table attack [RB][DRB]. Therefore the SAID, d, field of the attributes block, although, a cryptographic digest, does not securely blind the contents of the attributes block given knowledge of the schema. It only provides compactness, not privacy. Moreover, any cryptographic commitment to that SAID, d, field provides a fixed point of correlation potentially to the attribute block field values themselves in spite of non-disclosure of those field values via a compact ACDC. Thus an ACDC without a UUID, u, field in its attributes block must be considered a *_public-attribute_* ACDC even when expressed in compact form.

7.2. Public Uncompacted Attribute Section Schema

The subschema for the public uncompacted attribute section is shown below,

```
{
  "a":
  {
    "description": "attribute section",
    "type": "object",
    "required":
    [
      "d",
      "i",
      "score",
      "name"
    ],
    "properties":
    {
      "d":
      {
        "description": "attribute SAID",
        "type": "string"
      },
      "i":
      {
```

```
          "description": "Issuee AID",
          "type": "string"
        },
        "score":
        {
          "description": "test score",
          "type": "integer"
        },
        "name":
        {
          "description": "test taker full name",
          "type": "string"
        }
      },
      "additionalProperties": false
    }
  }
```

7.3. Composed Schema for both Public Compact and Uncompacted Attribute Section Variants

Through the use of the JSON Schema oneOf composition operator the following composed schema will validate against both the compact and un-compacted value of the attribute section field.


```
  {
    "a":
    {
      "description": "attribute section",
      "oneOf":
      [
        {
          "description": "attribute SAID",
          "type": "string"
        },
        {
          "description": "uncompacted attribute section",
          "type": "object",
          "required":
```

```
          [
            "d",
            "i",
            "score",
            "name"
          ],
          "properties":
          {
            "d":
            {
              "description": "attribute SAID",
              "type": "string"
            },
            "i":
            {
              "description": "Issuee AID",
              "type": "string"
            },
            "score":
            {
              "description": "test score",
              "type": "integer"
            },
            "name":
            {
              "description": "test taker full name",
              "type": "string"
            }
          },
          "additionalProperties": false
        }
      ]
    }
  }
```


7.4. Private-Attribute ACDC

Consider the following form of an uncompacted private-attribute block,

```
{
  "a":
```

```
  {
    "d": "EgveY4-9XgOcLxUderzwLIr9Bf7V_NHwY1lkFrn9y2PY",
    "u": "0AwjaDAE0qHcgNghkDaG7OY1",
    "i": "did:keri:EpZfFk66jpf3uFv7vklXKhzBrAqjsKAn2EDIPmkPreYA",
    "score": 96,
    "name": "Jane Doe"
  }
}
```

Given the presence of a top-level UUID, u, field of the attribute block whose value has sufficient cryptographic entropy, then the top-level SAID, d, field of the attribute block provides a secure cryptographic digest of the contents of the attribute block [Hash]. An adversary when given both the schema of the attribute block and its SAID, d, field, is not able to discover the remaining contents of the attribute block in a computationally feasible manner such as a rainbow table attack [RB][DRB]. Therefore the attribute block's UUID, u, field in a compact ACDC enables its attribute block's SAID, d, field to securely blind the contents of the attribute block notwithstanding knowledge of the attribute block's schema and SAID, d field. Moreover, a cryptographic commitment to that attribute block's, SAID, d, field does not provide a fixed point of correlation to the attribute field values themselves unless and until there has been a disclosure of those field values.

To elaborate, when an ACDC includes a sufficiently high entropy UUID, u, field at the top level of its attributes block then the ACDC may be considered a *_private-attributes_* ACDC when expressed in compact form, that is, the attribute block is represented by its SAID, d, field and the value of its top-level attribute section, a, field is the value of the nested SAID, d, field from the uncompacted version of the attribute block. A verifiable commitment may be made to the compact form of the ACDC without leaking details of the attributes. Later disclosure of the uncompacted attribute block may be verified against its SAID, d, field that was provided in the compact form as the value of the top-level attribute section, a, field.

Because the _Issuee_ AID is nested in the attribute block as that block's top-level, issuee, i, field, a presentation exchange (disclosure) could be initiated on behalf of a different AID that has not yet been correlated to the _Issuee_ AID and then only correlated

to the Issuee AID after the _Disclosee_ has agreed to the chain-link confidentiality provisions in the rules section of the private-attributes ACDC [CLC].

7.4.1. Composed Schema for Both Compact and Uncompacted Private-Attribute ACDC

Through the use of the JSON Schema oneOf composition operator the following composed schema will validate against both the compact and un-compacted value of the private attribute section, a, field.

```
{
  "a":
  {
    "description": "attribute section",
    "oneOf":
    [
      {
        "description": "attribute SAID",
        "type": "string"
      },
      {
        "description": "uncompacted attribute section",
        "type": "object",
        "required":
        [
          "d",
          "u",
          "i",
          "score",
          "name"
        ],
        "properties":
        {
          "d":
          {
            "description": "attribute SAID",
            "type": "string"
          },
          "u":
          {
            "description": "attribute UUID",
            "type": "string"
          },
          "i":
          {
            "description": "Issuee AID",
            "type": "string"
```

```
            },
            "score":
            {
              "description": "test score",
              "type": "integer"
            },
            "name":
            {
              "description": "test taker full name",
              "type": "string"
            }
          },
          "additionalProperties": false,
        }
      ]
    }
  }
```

As described above in the Schema section of this specification, the oneOf sub-schema composition operator validates against either the compact SAID of a block or the full block. A validator can use a composed schema that has been committed to by the Issuer to securely confirm schema compliance both before and after detailed disclosure by using the fully composed base schema pre-disclosure and a specific decomposed variant post-disclosure. Decomposing the schema to remove the optional compact variant (i.e. removing the oneOf compact option) enables a validator to ensure complaint full disclosure.

7.5. Untargeted ACDC

Consider the case where the issuee, i, field is absent at the top level of the attribute block as shown below,

```
{
  "a":
  {
    "d": "EgveY4-9XgOcLxUderzwLIr9Bf7V_NHwY1lkFrn9y2PY",
    "temp": 45,
    "lat": "N40.3433",
    "lon": "W111.7208"
  }
}
```

This ACDC has an _Issuer_ but no _Issuee_. Therefore, there is no provably controllable _Target_ AID. This may be thought of as an undirected verifiable attestation or observation of the data in the attributes block by the _Issuer_. One could say that the attestation is addressed to "whom it may concern". It is therefore an

*_untargeted_* ACDC, or equivalently an _unissueed_ ACDC. An _untargeted_ ACDC enables verifiable authorship by the Issuer of the data in the attributes block but there is no specified counter-party and no verifiable mechanism for delegation of authority. Consequently, the rule section may only provide contractual obligations of implied counter-parties.

This form of an ACDC provides a container for authentic data only (not authentic data as authorization). But authentic data is still a very important use case. To clarify, an untargeted ACDC enables verifiable authorship of data. An observer such as a sensor that controls an AID may make verifiable non-repudiable measurements and publish them as ACDCs. These may be chained together to provide provenance for or a chain-of-custody of any data. These ACDCs could be used to provide a verifiable data supply chain for any compliance-regulated application. This provides a way to protect participants in a supply chain from imposters. Such data supply chains are also useful as a verifiable digital twin of a physical supply chain [Twin].

A hybrid chain of one or more targeted ACDCs ending in a chain of one or more untargeted ACDCs enables delegated authorized attestations at the tail of that chain. This may be very useful in many regulated supply chain applications such as verifiable authorized authentic datasheets for a given pharmaceutical.

7.6. Targeted ACDC

When present at the top level of the attribute section, the issuee, i, field value provides the AID of the _Issuee_ of the ACDC. This _Issuee_ AID is a provably controllable identifier that serves as the _Target_ AID. This makes the ACDC a *_targeted_* ACDC or equivalently an _issueed_ ACDC. Targeted ACDCs may be used for many different purposes such as an authorization or a delegation directed at the _Issuee_ AID, i.e. the _Target_. In other words, a _targeted ACDC_ provides a container for authentic data that may also be used

as some form of authorization such as a credential that is verifiably bound to the _Issuee_ as targeted by the _Issuer_. Furthermore, by virtue of the targeted _Issuee's_ provable control over its AID, the _targeted ACDC_ may be verifiably presented (disclosed) by the controller of the _Issuee_ AID.

For example, the definition of the term *_credential_* is _evidence of authority, status, rights, entitlement to privileges, or the like_. To elaborate, the presence of an attribute section top-level issuee, i, field enables the ACDC to be used as a verifiable credential given by the _Issuer_ to the _Issuee_.

One reason the issuee, i, field is nested into the attribute section, a, block is to enable the _Issuee_ AID to be private or partially or selectively disclosable. The _Issuee_ may also be called the _Holder_ or _Subject_ of the ACDC. But here we use the more semantically precise albeit less common terms of _Issuer_ and _Issuee_. The ACDC is issued from or by an _Issuer_ and is issued to or for an _Issuee_. This precise terminology does not bias or color the role (function) that an _Issuee_ plays in the use of an ACDC. What the presence of _Issuee_ AID does provide is a mechanism for control of the subsequent use of the ACDC once it has been issued. To elaborate, because the issuee, i, field value is an AID, by definition, there is a provable controller of that AID. Therefore that _Issuee_ controller may make non-repudiable commitments via digital signatures on behalf of its AID. Therefore subsequent use of the ACDC by the _Issuee_ may be securely attributed to the _Issuee_.

Importantly the presence of an issuee, i, field enables the associated _Issuee_ to make authoritative verifiable presentations or disclosures of the ACDC. A designated _Issuee_also better enables the initiation of presentation exchanges of the ACDC between that _Issuee_ as _Discloser_ and a _Disclosee_ (verifier).

In addition, because the _Issuee_ is a specified counter-party the _Issuer_ may engage in a contract with the _Issuee_ that the _Issuee_ agrees to by virtue of its non-repudiable signature on an offer of the ACDC prior to its issuance. This agreement may be a pre-condition to the issuance and thereby impose liability waivers or other terms of use on that _Issuee_.

Likewise, the presence of an issuee, i, field, enables the _Issuer_ to use the ACDC as a contractual vehicle for conveying an authorization to the _Issuee_. This enables verifiable delegation chains of authority because the _Issuee_ in one ACDC may become the _Issuer_ in some other ACDC. Thereby an _Issuer_ may delegate authority to an _Issuee_ who may then become a verifiably authorized _Issuer_ that then delegates that authority (or an attenuation of that authority) to some other verifiably authorized _Issuee_ and so forth.

8. Edge Section

In the compact ACDC examples above, the edge section has been compacted into merely the SAID of that section. Suppose that the un-compacted value of the edge section denoted by the top-level edge, e, field is as follows,


```
{
  "e":
  {
    "d": "EerzwLIr9Bf7V_NHwY1lkFrn9y2PgveY4-9XgOcLx,UdY",
    "boss":
    {
      "n": "EIl3MORH3dCdoFOLe71iheqcywJcnjtJtQIYPvAu6DZA"
    }
  }
}
```

The edge section's top-level SAID, d, field is the SAID of the edge block and is the same SAID used as the compacted value of the ACDC's top-level edge, e, field. Each edge in the edge section gets its field with its own local label. In the example above, the edge label is "boss". Note that each edge does NOT include a type field. The type of each edge is provided by the schema vis-a-vis the label of that edge. This is in accordance with the design principle of ACDCs that may be succinctly expressed as "schema is type". This approach varies somewhat from many property graphs which often do not have a schema [PGM][Dots][KG]. Because ACDCs have a schema for other reasons, however, they leverage that schema to provide edge types

with a cleaner separation of concerns.

Each edge sub-block has one required node, n, field. The value of the node, n, field is the SAID of the ACDC to which the edge connects.

A main distinguishing feature of a _property graph_ (PG) is that both nodes but edges may have a set of properties [PGM][Dots][KG]. These might include modifiers that influence how the connected node is to be used such as a weight. Weighted directed edges represent degrees of confidence or likelihood. These types of PGs are commonly used for machine learning or reasoning under uncertainty. The following example adds a weight property to the edge sub-block as indicated by the weight, w, field.

```
{
  "e":
  {
    "d": "EerzwLIr9Bf7V_NHwY1lkFrn9y2PgveY4-9XgOcLxUdY",
    "boss":
    {
      "n": "EIl3MORH3dCdoFOLe71iheqcywJcnjtJtQIYPvAu6DZA",
      "w": "high"
    }
  }
}
```


8.1. Globally Distributed Secure Graph Fragments

Abstractly, an ACDC with one or more edges may be a fragment of a distributed property graph. However, the local label does not enable the direct unique global resolution of a given edge including its properties other than a trivial edge with only one property, its node, n field. To enable an edge with additional properties to be globally uniquely resolvable, that edge's block may have a SAID, d, field. Because a SAID is a cryptographic digest it will universally and uniquely identify an edge with a given set of properties [Hash]. This allows ACDCs to be used as secure fragments of a globally distributed property graph (PG). This enables a property graph to serve as a global knowledge graph in a secure manner that crosses trust domains [PGM][Dots][KG]. This is shown below.

```
{
  "e":
  {
    "d": "EerzwLIr9Bf7V_NHwY1lkFrn9y2PgveY4-9XgOcLxUdY",
    "boss":
    {
      "d": "E9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NHwY1lkFrn",
      "n": "EIl3MORH3dCdoFOLe71iheqcywJcnjtJtQIYPvAu6DZA",
      "w": "high"
    }
  }
}
```

8.2. Compact Edge

Given that an individual edge's property block includes a SAID, d, field then a compact representation of the edge's property block is provided by replacing it with its SAID. This may be useful for complex edges with many properties. This is called a *_compact edge_*. This is shown as follows,

```
{
  "e":
  {
    "d": "EerzwLIr9Bf7V_NHwY1lkFrn9y2PgveY4-9XgOcLxUdY",
    "boss": "E9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NHwY1lkFrn",
  }
}
```


8.3. Private Edge

Each edge's properties may be blinded by its SAID, d, field (i.e. be private) if its properties block includes a UUID, u field. As with UUID, u, fields used elsewhere in ACDC, if the UUID, u, field value has sufficient entropy then the values of the properties of its enclosing block are not discoverable in a computationally feasible manner merely given the schema for the edge block and its SAID, d

field. This is called a *_private edge_*. When a private edge is provided in compact form then the edge detail is hidden and is partially disclosable. An uncompacted private edge is shown below.

```
{
  "e":
  {
    "d": "EerzwLIr9Bf7V_NHwY1lkFrn9y2PgveY4-9XgOcLxUdY",
    "boss":
    {
      "d": "E9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NHwY1lkFrn",
      "u":  "0AG7OY1wjaDAE0qHcgNghkDa",
      "n": "EIl3MORH3dCdoFOLe71iheqcywJcnjtJtQIYPvAu6DZA",
      "w": "high"
    }
  }
}
```

When an edge points to a _private_ ACDC, a _Discloser_ may choose to use a metadata version of that private ACDC when presenting the node, n, field of that edge prior to acceptance of the terms of disclosure. The _Disclosee_ can verify the metadata of the private node without the _Discloser_ exposing the actual node contents via the actual node SAID or other attributes.

Private ACDCs (nodes) and private edges may be used in combination to prevent an un-permissioned correlation of the distributed property graph.

8.4. Simple Compact Edge

When an edge sub-block has only one field that is its node, n, field then the edge block may use an alternate simplified compact form where the labeled edge field value is the value of its node, n, field. The schema for that particular edge label, in this case, "boss", will indicate that the edge value is a node SAID and not the edge sub-block SAID as would be the case for the normal compact form shown above. This alternate compact form is shown below.


```
{
```

```
  "e":
  {
    "d": "EerzwLIr9Bf7V_NHwY1lkFrn9y2PgveY4-9XgOcLxUdY",
    "boss": "EIl3MORH3dCdoFOLe71iheqcywJcnjtJtQIYPvAu6DZA"
  }
}
```

8.5. Node Discovery

In general, the discovery of the details of an ACDC referenced as a node, n field value, in an edge sub-block begins with the node SAID or the SAID of the associated edge sub-block. Because a SAID is a cryptographic digest with high collision resistance it provides a universally unique identifier to the referenced ACDC as a node. The Discovery of a service endpoint URL that provides database access to a copy of the ACDC may be bootstrapped via an OOBI (Out-Of-Band-Introduction) that links the service endpoint URL to the SAID of the ACDC [OOBI_ID]. Alternatively, the _Issuer_ may provide as an attachment at the time of issuance a copy of the referenced ACDC. In either case, after a successful exchange, the _Issuee_ or recipient of any ACDC will have either a copy or a means of obtaining a copy of any referenced ACDCs as nodes in the edge sections of all ACDCs so chained. That Issuee or recipient will then have everything it needs to make a successful disclosure to some other _Disclosee_. This is the essence of _percolated_ discovery.

9. Rule Section

In the compact ACDC examples above, the rule section has been compacted into merely the SAID of that section. Suppose that the un-compacted value of the rule section denoted by the top-level rule, r, field is as follows,

```
{
  "r":
  {
    "d": "EwY1lkFrn9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NA",
    "warrantyDisclaimer":
    {
      "l": "Issuer provides this credential on an \"AS IS\" BASIS, WITHOUT W
    },
    "liabilityDisclaimer":
    {
      "l": "In no event and under no legal theory, whether in tort (includin
    }
  }
}
```

The purpose of the rule section is to provide a Ricardian Contract [RC]. The important features of a Ricardian contract are that it be both human and machine-readable and referenceable by a cryptographic digest. A JSON encoded document or block such as the rule section block is a practical example of both a human and machine-readable document. The rule section's top-level SAID, d, field provides the digest. This provision supports the bow-tie model of Ricardian Contracts [RC]. Ricardian legal contracts may be hierarchically structured into sections and subsections with named or numbered clauses in each section. The labels on the clauses may follow such a hierarchical structure using nested maps or blocks. These provisions enable the rule section to satisfy the features of a Ricardian contract.

To elaborate, the rule section's top-level SAID, d, field is the SAID of that block and is the same SAID used as the compacted value of the rule section, r, field that appears at the top level of the ACDC. Each clause in the rule section gets its own field. Each clause also has its own local label.

The legal, l, field in each block provides the associated legal language.

Note there are no type fields in the rule section. The type of a contract and the type of each clause is provided by the schema vis-a-vis the label of that clause. This follows the ACDC design principle that may be succinctly expressed as "schema is type".

Each rule section clause may also have its own clause SAID, d, field. Clause SAIDs enable reference to individual clauses, not merely the whole contract as given by the rule section's top-level SAID, d, field.

An example rule section with clause SAIDs is provided below.

```
{
  "r":
  {
    "d": "EwY1lkFrn9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NA",
    "warrantyDisclaimer":
    {
      "d": "EXgOcLxUdYerzwLIr9Bf7V_NAwY1lkFrn9y2PgveY4-9",
      "l": "Issuer provides this credential on an \"AS IS\" BASIS, WITHOUT W
    },
    "liabilityDisclaimer":
    {
      "d": "EY1lkFrn9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NAw",
      "l": "In no event and under no legal theory, whether in tort (includin
    }
  }
}
```

9.1. Compact Clauses

The use of clause SAIDS enables a compact form of a set of clauses where each clause value is the SAID of the corresponding clause. For example,

```
{
  "r":
  {
    "d": "EwY1lkFrn9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NA",
    "warrantyDisclaimer":  "EXgOcLxUdYerzwLIr9Bf7V_NAwY1lkFrn9y2PgveY4-9",
    "liabilityDisclaimer": "EY1lkFrn9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NAw"
  }
}
```

9.2. Private Clause

The disclosure of some clauses may be pre-conditioned on acceptance of chain-link confidentiality. In this case, some clauses may benefit from partial disclosure. Thus clauses may be blinded by their SAID, d, field when the clause block includes a sufficiently high entropy UUID, u, field. The use of a clause UUID enables the

compact form of a clause to NOT be discoverable merely from the schema for the clause and its SAID via rainbow table attack [RB][DRB]. Therefore such a clause may be partially disclosable. These are called *_private clauses_*. A private clause example is shown below.


```
{
  "r":
  {
    "d": "EwY1lkFrn9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NA",
    "warrantyDisclaimer":
    {
      "d": "EXgOcLxUdYerzwLIr9Bf7V_NAwY1lkFrn9y2PgveY4-9",
      "u": "0AG7OY1wjaDAE0qHcgNghkDa",
      "l": "Issuer provides this credential on an \"AS IS\" BASIS, WITHOUT W
    },
    "liabilityDisclaimer":
    {
      "d": "EY1lkFrn9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NAw",
      "u": "0AHcgNghkDaG7OY1wjaDAE0q",
      "l": "In no event and under no legal theory, whether in tort (includin
    }
  }
}
```


9.3. Simple Compact Clause

An alternate simplified compact form uses the value of the legal, l, field as the value of the clause field label. The schema for a specific clause label will indicate that the field value, for a given clause label is the legal language itself and not the clause block's SAID, d, field as is the normal compact form shown above. This alternate simple compact form is shown below. In this form individual clauses are not compactifiable and are fully self-contained.


```
{
  "r":
```

```
    {
      "d": "EwY1lkFrn9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NA",
      "warrantyDisclaimer": "Issuer provides this credential on an \"AS IS\" B
      "liabilityDisclaimer": "In no event and under no legal theory, whether i
    }
  }
```

9.4. Clause Discovery

In compact form, the discovery of either the rule section as a whole or a given clause begins with the provided SAID. Because the SAID, d, field of any block is a cryptographic digest with high collision resistance it provides a universally unique identifier to the referenced block details (whole rule section or individual clause). The discovery of a service endpoint URL that provides database access to a copy of the rule section or to any of its clauses may be

bootstrapped via an OOBI (Out-Of-Band-Introduction) that links the service endpoint URL to the SAID of the respective block. Alternatively, the issuer may provide as an attachment at issuance a copy of the referenced contract associated with the whole rule section or any clause. In either case, after a successful issuance exchange, the Issuee or holder of any ACDC will have either a copy or a means of obtaining a copy of any referenced contracts in whole or in part of all ACDCs so issued. That Issuee or recipient will then have everything it needs to subsequently make a successful presentation or disclosure to a Disclosee. This is the essence of percolated discovery.

10. Informative Example of an ACDC

10.1. Public Compact Variant

```
{
  "v":  "ACDC10JSON00011c_",
  "d":  "EBdXt3gIXOf2BBWNHdSXCJnFJL5OuQPyM5K0neuniccM",
  "i":  "did:keri:EmkPreYpZfFk66jpf3uFv7vklXKhzBrAqjsKAn2EDIPM",
  "ri": "did:keri:EymRy7xMwsxUelUauaXtMxTfPAMPAI6FkekwlOjkggt",
  "s":  "E46jrVPTzlSkUPqGGeIZ8a8FWS7a6s4reAXRZOkogZ2A",
  "a":  "EgveY4-9XgOcLxUderzwLIr9Bf7V_NHwY1lkFrn9y2PY",
  "e":  "ERH3dCdoFOLe71iheqcywJcnjtJtQIYPvAu6DZIl3MOA",
  "r":  "Ee71iheqcywJcnjtJtQIYPvAu6DZIl3MORH3dCdoFOLB"
```

```
   }
```

10.2. Public Uncompacted Variant


```
   {
     "v":  "ACDC10JSON00011c_",
     "d":  "EBdXt3gIXOf2BBWNHdSXCJnFJL5OuQPyM5K0neuniccM",
     "i":  "did:keri:EmkPreYpZfFk66jpf3uFv7vklXKhzBrAqjsKAn2EDIPM",
     "ri": "did:keri:EymRy7xMwsxUelUauaXtMxTfPAMPAI6FkekwlOjkggt",
     "s":  "E46jrVPTzlSkUPqGGeIZ8a8FWS7a6s4reAXRZOkogZ2A",
     "a":
     {
       "d": "EgveY4-9XgOcLxUderzwLIr9Bf7V_NHwY1lkFrn9y2PY",
       "i": "did:keri:EpZfFk66jpf3uFv7vklXKhzBrAqjsKAn2EDIPmkPreYA",
       "score": 96,
       "name": "Jane Doe"
     },
     "e":
     {
       "d": "EerzwLIr9Bf7V_NHwY1lkFrn9y2PgveY4-9XgOcLxUdY",
       "boss":
       {
```

```
        "d": "E9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NHwY1lkFrn",
        "n": "EIl3MORH3dCdoFOLe71iheqcywJcnjtJtQIYPvAu6DZA",
        "w": "high"
      }
    },
    "r":
    {
      "d": "EwY1lkFrn9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NA",
      "warrantyDisclaimer":
      {
        "d": "EXgOcLxUdYerzwLIr9Bf7V_NAwY1lkFrn9y2PgveY4-9",
        "l": "Issuer provides this credential on an \"AS IS\" BASIS, WITHOUT W
      },
      "liabilityDisclaimer":
      {
        "d": "EY1lkFrn9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NAw",
        "l": "In no event and under no legal theory, whether in tort (includin
      }
    }
  }
```

10.3. Composed Schema that Supports both Public Compact and Uncompacted Variants

```
  {
    "$id": "EN8i2i5ye0-xGS95pm5cg1j0GmFkarJe0zzsSrrf4XJY",
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "title": "Public ACDC",
    "description": "Example JSON Schema Public ACDC.",
    "credentialType": "PublicACDCExample",
```



```
    "type": "object",
     "required":
    [
      "v",
      "d",
      "i",
      "ri",
      "s",
      "a",
      "e",
      "r"
```

```
    ],
    "properties":
    {
      "v":
      {
        "description": "ACDC version string",
        "type": "string"
      },
      "d":
      {
       "description": "ACDC SAID",
        "type": "string"
      },
      "i":
      {
        "description": "Issuer AID",
        "type": "string"
      },
      "ri":
      {
        "description": "credential status registry AID",
        "type": "string"
      },
      "s":
      {
        "description": "schema section",
        "oneOf":
        [
          {
            "description": "schema section SAID",
            "type": "string"
          },
          {
            "description": "schema detail",
            "type": "object"
          },
        ]
```



```
      },
      "a":
      {
        "description": "attribute section",
```

```
    "oneOf":
    [
      {
        "description": "attribute section SAID",
        "type": "string"
      },
      {
        "description": "attribute detail",
        "type": "object",
        "required":
        [
          "d",
          "i",
          "score",
          "name"
        ],
        "properties":
        {
          "d":
          {
            "description": "attribute section SAID",
            "type": "string"
          },
          "i":
          {
            "description": "Issuee AID",
            "type": "string"
          },
          "score":
          {
            "description": "test score",
            "type": "integer"
          },
          "name":
          {
            "description": "test taker full name",
            "type": "string"
          }
        },
        "additionalProperties": false,
      }
    ],
  },
  "e":
```

```
{
  "description": "edge section",
  "oneOf":
  [
    {
      "description": "edge section SAID",
      "type": "string"
    },
    {
      "description": "edge detail",
      "type": "object",
      "required":
      [
        "d",
        "boss"
      ],
      "properties":
      {
        "d":
        {
          "description": "edge section SAID",
          "type": "string"
        },
        "boss":
        {
          "description": "boss edge",
          "type": "object",
          "required":
          [
            "d",
            "n",
            "w"
          ],
          "properties":
          {
            "d":
            {
              "description": "edge SAID",
              "type": "string"
            },
            "n":
            {
              "description": "node SAID",
              "type": "string"
            },
            "w":
            {
```

```
                    "description": "edge weight",
```



```
                    "type": "string"
                  },
                  "additionalProperties": false
                },
              },
              "additionalProperties": false
            }
          ],
        },
        "r":
        {
          "description": "rule section",
          "oneOf":
          [
            {
              "description": "rule section SAID",
              "type": "string"
            },
            {
              "description": "rule detail",
              "type": "object",
              "required":
              [
                "d",
                "warrantyDisclaimer",
                "liabilityDisclaimer"
              ],
              "properties":
              {
                "d":
                {
                  "description": "edge section SAID",
                  "type": "string"
                },
                "warrantyDisclaimer":
                {
                  "description": "warranty disclaimer clause",
                  "type": "object",
                  "required":
                  [
```

```
                "d",
                "l"
              ],
              "properties":
              {
                "d":
                {
                  "description": "clause SAID",
```



```
                  "type": "string"
                },
                "l":
                {
                  "description": "legal language",
                  "type": "string"
                }
              },
              "additionalProperties": false
            },
            "liabilityDisclaimer":
            {
              "description": "liability disclaimer clause",
              "type": "object",
              "required":
              [
                "d",
                "l"
              ],
              "properties":
              {
                "d":
                {
                  "description": "clause SAID",
                  "type": "string"
                },
                "l":
                {
                  "description": "legal language",
                  "type": "string"
                }
              },
              "additionalProperties": false
```

```
              }
            },
            "additionalProperties": false
          }
        ]
      }
    },
    "additionalProperties": false
  }
```


11. Selective Disclosure

As explained previously, the primary difference between _partial disclosure_ and _selective disclosure_ is determined by the correlatability with respect to its encompassing block after _full disclosure_ of the detailed field value. A _partially disclosable_ field becomes correlatable to its encompassing block after its _full disclosure_. Whereas a _selectively disclosable_ field may be excluded from the _full disclosure_ of any other selectively disclosable fields in its encompassing block. After selective disclosure, the selectively disclosed fields are not correlatable to the so-far undisclosed but selectively disclosable fields in the same encompassing block. In this sense, _full disclosure_ means detailed disclosure of the selectively disclosed attributes not detailed disclosure of all selectively disclosable attributes.

Recall that _partial_ disclosure is an essential mechanism needed to support chain-link confidentiality [CLC]. The chain-link confidentiality exchange _offer_ requires _partial disclosure_, and _full disclosure_ only happens after _acceptance_ of the _offer_. _Selective_ disclosure, on the other hand, is an essential mechanism needed to unbundle in a correlation minimizing way a single commitment by an Issuer to a bundle of fields (i.e. a nested block or array of fields). This allows separating a "stew" of "ingredients" (attributes) into its constituent "ingredients" (attributes) without correlating the constituents via the stew.

ACDCs, as a standard, benefit from a minimally sufficient approach to selective disclosure that is simple enough to be universally implementable and adoptable. This does not preclude support for other more sophisticated but optional approaches. But the minimally sufficient approach should be universal so that at least one selective disclosure mechanism be made available in all ACDC implementations. To clarify, not all instances of an ACDC must employ the minimal selective disclosure mechanisms as described herein but all ACDC implementations must support any instance of an ACDC that employs the minimal selective disclosure mechanisms as described above.

The ACDC chaining mechanism reduces the need for selective disclosure in some applications. Many non-ACDC verifiable credentials provide bundled precisely because there is no other way to associate the attributes in the bundle. These bundled credentials could be refactored into a graph of ACDCs. Each of which is separately disclosable and verifiable thereby obviating the need for selective disclosure. Nonetheless, some applications require bundled attributes and therefore may benefit from the independent selective disclosure of bundled attributes. This is provided by *_selectively disclosable attribute_* ACDCs.

The use of a revocation registry is an example of a type of bundling, not of attributes in a credential, but uses of a credential in different contexts. Unbundling the usage contexts may be beneficial. This is provided by *_bulk-issued_* ACDCs.

In either case, the basic selective disclosure mechanism is comprised of a single aggregated blinded commitment to a list of blinded commitments to undisclosed values. Membership of any blinded

commitment to a value in the list of aggregated blinded commitments may be proven without leaking (disclosing) the unblinded value belonging to any other blinded commitment in the list. This enables provable selective disclosure of the unblinded values. When a non-repudiable digital signature is created on the aggregated blinded commitment then any disclosure of a given value belonging to a given blinded commitment in the list is also non-repudiable. This approach does not require any more complex cryptography than digests and digital signatures. This satisfies the design ethos of minimally sufficient means. The primary drawback of this approach is verbosity. It trades ease and simplicity and adoptability of implementation for size. Its verbosity may be mitigated by replacing the list of blinded commitments with a Merkle tree of those commitments where the Merkle tree root becomes the aggregated blinded commitment.

Given sufficient cryptographic entropy of the blinding factors, collision resistance of the digests, and unforgeability of the digital signatures, either inclusion proof format (list or Merkle tree digest) prevents a potential disclosee or adversary from discovering in a computationally feasible way the values of any undisclosed blinded value details from the combination of the schema of those value details and either the aggregated blinded commitment and/or the list of aggregated blinded commitments [Hash][HCR][QCHC][Mrkl][TwoPI][MTSec]. A potential disclosee or adversary would also need both the blinding factor and the actual value details.

Selective disclosure in combination with partial disclosure for chain-link confidentiality provides comprehensive correlation minimization because a discloser may use a non-disclosing metadata ACDC prior to acceptance by the disclosee of the terms of the chain-link confidentiality expressed in the rule section [CLC]. Thus only malicious disclosees who violate chain-link confidentiality may correlate between independent disclosures of the value details of distinct members in the list of aggregated blinded commitments. Nonetheless, they are not able to discover any as of yet undisclosed (unblinded) value details.

11.1. Selectively Disclosable Attribute ACDC

In a *_selectively disclosable attribute_* ACDC, the set of attributes is provided as an array of blinded blocks. Each attribute in the set has its own dedicated blinded block. Each block has its own SAID, d, field and UUID, u, field in addition to its attribute field or fields. When an attribute block has more than one attribute field then the set of fields in that block are not independently selectively disclosable but MUST be disclosed together as a set. Notable is that the field labels of the selectively disclosable attributes are also blinded because they only appear within the blinded block. This prevents un-permissioned correlation via contextualized variants of a field label that appear in a selectively disclosable block. For example, localized or internationalized variants where each variant's field label(s) each use a different language or some other context correlatable information in the field labels themselves.

A selectively-disclosable attribute section appears at the top level using the field label A. This is distinct from the field label a for a non-selectively-disclosable attribute section. This makes clear (unambiguous) the semantics of the attribute section's associated schema. This also clearly reflects the fact that the value of a compact variant of selectively-disclosable attribute section is an "aggregate" not a SAID. As described previously, the top-level selectively-disclosable attribute aggregate section, A, field value is an aggregate of cryptographic commitments used to make a commitment to a set (bundle) of selectively-disclosable attributes. The derivation of its value depends on the type of selective disclosure mechanism employed. For example, the aggregate value could be the cryptographic digest of the concatenation of an ordered set of cryptographic digests, a Merkle tree root digest of an ordered set of cryptographic digests, or a cryptographic accumulator.

The _Issuer_ attribute block is absent from an uncompacted untargeted selectively disclosable ACDC as follows:


```
{
  "A":
  [
    {
      "d": "ELIr9Bf7V_NHwY1lkgveY4-Frn9y2PY9XgOcLxUderzw",
```

```
        "u": "0AG7OY1wjaDAE0qHcgNghkDa",
        "score": 96
      },
      {
        "d": "E9XgOcLxUderzwLIr9Bf7V_NHwY1lkFrn9y2PYgveY4-",
        "u": "0AghkDaG7OY1wjaDAE0qHcgN",
        "name": "Jane Doe"
      }
    ]
  }
```

The _Issuer_ attribute block is present in an uncompacted untargeted selectively disclosable ACDC as follows:

```
{
  "A":
  [
    {
      "d": "ErzwLIr9Bf7V_NHwY1lkFrn9y2PYgveY4-9XgOcLxUde",
      "u": "0AqHcgNghkDaG7OY1wjaDAE0",
      "i": "did:keri:EpZfFk66jpf3uFv7vklXKhzBrAqjsKAn2EDIPmkPreYA"
    },
    {
      "d": "ELIr9Bf7V_NHwY1lkgveY4-Frn9y2PY9XgOcLxUderzw",
      "u": "0AG7OY1wjaDAE0qHcgNghkDa",
      "score": 96
    },
    {
      "d": "E9XgOcLxUderzwLIr9Bf7V_NHwY1lkFrn9y2PYgveY4-",
      "u": "0AghkDaG7OY1wjaDAE0qHcgN",
      "name": "Jane Doe"
    }
  ]
}
```

11.1.1. Blinded Attribute Array

Given that each attribute block's UUID, u, field has sufficient cryptographic entropy, then each attribute block's SAID, d, field provides a secure cryptographic digest of its contents that effectively blinds the attribute value from discovery given only its Schema and SAID. To clarify, the adversary despite being given both the schema of the attribute block and its SAID, d, field, is not able

to discover the remaining contents of the attribute block in a computationally feasible manner such as a rainbow table attack [RB][DRB]. Therefore the UUID, u, field of each attribute block enables the associated SAID, d, field to securely blind the block's contents notwithstanding knowledge of the block's schema and that SAID, d, field. Moreover, a cryptographic commitment to that SAID, d, field does not provide a fixed point of correlation to the associated attribute (SAD) field values themselves unless and until there has been specific disclosure of those field values themselves.

Given a total of _N_ elements in the attributes array, let _a_i_ represent the SAID, d, field of the attribute at zero-based index _i_. More precisely the set of attributes is expressed as the ordered set,

_{a_i for all i in {0, ..., N-1}}_.

The ordered set of _a_i_ may be also expressed as a list, that is,

_[a_0, a_1, ..., a_(N-1)]_.

11.1.2. Composed Schema for Selectively Disclosable Attribute Section

Because the selectively-disclosable attributes are provided by an array (list), the uncompacted variant in the schema uses an array of items and the anyOf composition operator to allow one or more of the items to be disclosed without requiring all to be disclosed. Thus both the oneOf and anyOf composition operators are used. The oneOf is used to provide compact partial disclosure of the aggregate, _A_, as the value of the top-level selectively-disclosable attribute section, A, field in its compact variant and the nested anyOf operator is used to enable selective disclosure in the uncompacted selectively-disclosable variant.

```
{
  "A":
  {
    "description": "attribute section",
    "oneOf":
    [
      {
        "description": "attribute section SAID",
        "type": "string"
      },
      {
        "description": "attribute details",
        "type": "array",
        "uniqueItems": true,
```

```
            "items":
            {
              "anyOf":
              [
                {
                  "description": "issuer attribute",
                  "type": "object",
                  "properties":
                  "required":
                  [
                    "d",
                    "u",
                    "i"
                  ],
                  "properties":
                  {
                    "d":
                    {
                      "description": "attribute SAID",
                      "type": "string"
                    },
                    "u":
                    {
                      "description": "attribute UUID",
                      "type": "string"
                    },
                    "i":
                    {
                      "description": "issuer SAID",
                      "type": "string"
                    },
                  },
                  "additionalProperties": false
                },
                {
                  "description": "score attribute",
                  "type": "object",
                  "properties":
                  "required":
                  [
                    "d",
                    "u",
```

```
      "score"
    ],
    "properties":
    {
      "d":
      {
```



```
        "description": "attribute SAID",
        "type": "string"
      },
      "u":
      {
        "description": "attribute UUID",
        "type": "string"
      },
      "score":
      {
        "description": "score value",
        "type": "integer"
      },
    },
    "additionalProperties": false
  },
  {
    "description": "name attribute",
    "type": "object",
    "properties":
    "required":
    [
      "d",
      "u",
      "name"
    ],
    "properties":
    {
      "d":
      {
        "description": "attribute SAID",
        "type": "string"
      },
      "u":
      {
```

```
                    "description": "attribute UUID",
                    "type": "string"
                  },
                  "name":
                  {
                    "description": "name value",
                    "type": "string"
                  },
                },
                "additionalProperties": false
              }
            ]
          }
```



```
        }
      ]
      "additionalProperties": false
    }
  }
```

11.1.3. Inclusion Proof via Aggregated List Digest

All the _a_i_ in the list are aggregated into a single aggregate digest denoted _A_ by computing the digest of their ordered concatenation. This is expressed as follows:

_A = H(C(a_i for all i in {0, ..., N-1}))_ where _H_ is the digest (hash) operator and _C_ is the concatentation operator.

To be explicit, using the targeted example above, let _a_0_ denote the SAID of the _Issuee_ attribute, _a_1_ denote the SAID of the _score_ attribute, and _a_2_ denote the SAID of the _name_ attribute then the aggregated digest _A_ is computed as follows:

_A = H(C(a_0, a_1, a_2))_.

Equivalently using _+_ as the infix concatenation operator, we have,

_A = H(a_0 + a_1 + a_2)_

Given sufficient collision resistance of the digest operator, the digest of an ordered concatenation is not subject to a birthday

attack on its concatenated elements [BDC][BDay][QCHC][HCR][Hash].

In compact form, the value of the selectively-disclosable top-level attribute section, A, field is set to the aggregated value _A_. This aggregate _A_ makes a blinded cryptographic commitment to the all the ordered elements in the list,

_[a_0, a_1, ..., a_(N-1)]_.

Moreover because each _a_i_ element also makes a blinded commitment to its block's (SAD) attribute value(s), disclosure of any given _a_i_ element does not expose or disclose any discoverable information detail about either its own or another block's attribute value(s). Therefore one may safely disclose the full list of _a_i_ elements without exposing the blinded block attribute values.

Proof of inclusion in the list consists of checking the list for a matching value. A computationally efficient way to do this is to create a hash table or B-tree of the list and then check for inclusion via lookup in the hash table or B-tree.

To protect against later forgery given a later compromise of the signing keys of the Issuer, the issuer MUST anchor an issuance proof digest seal to the ACDC in its KEL. This seal binds the signing key state to the issuance. There are two cases. In the first case, an issuance/revocation registry is used. In the second case, an issuance/revocation registry is not used.

When the ACDC is registered using an issuance/revocation TEL (Transaction Event Log) then the issuance proof seal digest is the SAID of the issuance (inception) event in the ACDC's TEL entry. The issuance event in the TEL includes the SAID of the ACDC. This binds the ACDC to the issuance proof seal in the Issuer's KEL through the TEL entry.

When the ACDC is not registered using an issuance/revocation TEL then the issuance proof seal digest is the SAID of the ACDC itself.

In either case, this issuance proof seal makes a verifiable binding between the issuance of the ACDC and the key state of the Issuer at the time of issuance. Because aggregated value _A_ provided as the attribute section, A, field, value is bound to the SAID of the ACDC

which is also bound to the key state via the issuance proof seal, the attribute details of each attribute block are also bound to the key state.

The requirement of an anchored issuance proof seal means that the forger Must first successfully publish in the KEL of the issuer an inclusion proof digest seal bound to a forged ACDC. This makes any forgery attempt detectable. To elaborate, the only way to successfully publish such a seal is in a subsequent interaction event in a KEL that has not yet changed its key state via a rotation event. Whereas any KEL that has changed its key state via a rotation must be forked before the rotation. This makes the forgery attempt either both detectable and recoverable via rotation in any KEL that has not yet changed its key state or detectable as duplicity in any KEL that has changed its key state. In any event, the issuance proof seal ensures detectability of any later attempt at forgery using compromised keys.

Given that aggregate value _A_ appears as the compact value of the top-level attribute section, A, field, the selective disclosure of the attribute at index _j_ may be proven to the disclosee with four items of information. These are:

* The actual detailed disclosed attribute block itself (at index _j_) with all its fields.

* The list of all attribute block digests, _[a_0, a_1, ..., a_(N-1)]_ that includes _a_j_.

* The ACDC in compact form with selectively-disclosable attribute section, A, field value set to aggregate _A_.

* The signature(s), _s_, of the Issuee on the ACDC's top-level SAID, d, field.

The actual detailed disclosed attribute block is only disclosed after the disclosee has agreed to the terms of the rules section. Therefore, in the event the potential disclosee declines to accept the terms of disclosure, then a presentation of the compact version of the ACDC and/or the list of attribute digests, _[a_0, a_1, ...,

a_(N-1)]_. does not provide any point of correlation to any of the attribute values themselves. The attributes of block _j_ are hidden by _a_j_ and the list of attribute digests _[a_0, a_1, ..., a_(N-1)]_ is hidden by the aggregate _A_. The partial disclosure needed to enable chain-link confidentiality does not leak any of the selectively disclosable details.

The disclosee may then verify the disclosure by: * computing _a_j_ on the selectively disclosed attribute block details. * confirming that the computed _a_j_ appears in the provided list _[a_0, a_1, ..., a_(N-1)]_. * computing _A_ from the provided list _[a_0, a_1, ..., a_(N-1)]_. * confirming that the computed _A_ matches the value, _A_, of the selectively-disclosable attribute section, A, field value in the provided ACDC. * computing the top-level SAID, d, field of the provided ACDC. * confirming the presence of the issuance seal digest in the Issuer's KEL * confirming that the issuance seal digest in the Issuer's KEL is bound to the ACDC top-level SAID, d, field either directly or indirectly through a TEL registry entry. * verifying the provided signature(s) of the Issuee on the provided top-level SAID, d field value.

The last 3 steps that culminate with verifying the signature(s) require determining the key state of the Issuer at the time of issuance, this may require additional verification steps as per the KERI, PTEL, and CESR-Proof protocols.

A private selectively disclosable ACDC provides significant correlation minimization because a presenter may use a metadata ACDC prior to acceptance by the disclosee of the terms of the chain-link confidentiality expressed in the rule section [CLC]. Thus only malicious disclosees who violate chain-link confidentiality may correlate between presentations of a given private selectively disclosable ACDC. Nonetheless, they are not able to discover any undisclosed attributes.

11.1.4. Inclusion Proof via Merkle Tree Root Digest

The inclusion proof via aggregated list may be somewhat verbose when there are a large number of attribute blocks in the selectively disclosable attribute section. A more efficient approach is to create a Merkle tree of the attribute block digests and let the aggregate, _A_, be the Merkle tree root digest [Mrkl]. Specifically,

set the value of the top-level selectively-disclosable attribute section, A, field to the aggregate, _A_ whose value is the Merkle tree root digest [Mrkl].

The Merkle tree needs to have appropriate second-pre-image attack protection of interior branch nodes [TwoPI][MTSec]. The discloser then only needs to provide a subset of digests from the Merkle tree to prove that a given digest, _a_j_ contributed to the Merkle tree root digest, _A_. For ACDCs with a small number of attributes the added complexity of the Merkle tree approach may not be worth the savings in verbosity.

11.1.5. Hierarchical Derivation at Issuance of Selectively Disclosable Attribute ACDCs

The amount of data transferred between the Issuer and Issuee (or recipient in the case of an untargeted ACDC) at issuance of a selectively disclosable attribute ACDC may be minimized by using a hierarchical deterministic derivation function to derive the value of the UUDI, u, fields from a shared secret salt [Salt].

There are several ways that the Issuer may securely share that secret salt. Given that an Ed25519 key pair(s) controls each of the Issuer and Issuee AIDs, (or recipient AID in the case of an untargeted ACDC) a corresponding X15519 asymmetric encryption key pair(s) may be derived from each controlling Ed25519 key pair(s) [EdSC][PSEd][TMEd]. An X25519 public key may be derived from an Ed25519 public key. Likewise, an X25519 private key may be derived from an Ed25519 private key [KeyEx].

In an interactive approach, the Issuer derives a public asymmetric X25519 encryption key from the Issuee's published Ed25519 public key and the Issuee derives a public asymmetric X25519 encryption key from the Issuer's published Ed25519 public key. The two then interact via a Diffie-Hellman (DH) key exchange to create a shared symmetric encryption key [KeyEx][DHKE]. The shared symmetric encryption key may be used to encrypt the secret salt or the shared symmetric encryption key itself may be used has high entropy cryptographic material from which the secret salt may be derived.

In a non-interactive approach, the Issuer derives an X25519 asymmetric public encryption key from the Issuee's (recipient's) public Ed25519 public key. The Issuer then encrypts the secret salt with that public asymmetric encryption key and signs the encryption with the Issuer's private Ed25519 signing key. This is transmitted to the Issuee, who verifies the signature and decrypts the secret salt using the private X25519 decryption key derived from the Issuee's private Ed25519 key. This non-interactive approach is more scalable for AIDs that are controlled with a multi-sig group of signing keys. The Issuer can broadcast to all members of the Issuee's (or recipient's) multi-sig signing group individually asymmetrically encrypted and signed copies of the secret salt.

In addition to the secret salt, the Issuer provides to the Issuee (recipient) a template of the ACDC but with empty UUID, u, and SAID, d, fields in each block with such fields. Each UUID, u, field value is then derived from the shared salt with a path prefix that indexes a specific block. Given the UUID, u, field value, the SAID, d, field value may then be derived. Likewise, both compact and uncompacted versions of the ACDC may then be generated. The derivation path for the top-level UUID, u, field (for private ACDCS), is the string "0" and derivation path the the the zeroth indexed attribute in the attributes array is the string "0/0". Likewise, the next attribute's derivation path is the string "0/1" and so forth.

In addition to the shared salt and ACDC template, the Issuer also provides its signature(s) on its own generated compact version ACDC. The Issuer may also provide references to the anchoring issuance proof seals. Everything else an Issuee (recipient) needs to make a verifiable presentation/disclosure can be computed at the time of presentation/disclosure by the Issuee.

11.2. Bulk-Issued Private ACDCs

The purpose of bulk issuance is to enable the Issuee to use unique ACDC more efficiently SAIDs to isolate and minimize correlation across different usage contexts of essentially the same ACDC while allowing public commitments to the ACDC SAIDs. A private ACDC may be issued in bulk as a set. In its basic form, the only difference between each ACDC is the top-level SAID, _d_, and UUID, _u_ field values. To elaborate, bulk issuance enables the use of un-correlatable copies while minimizing the associated data transfer and storage requirements. Essentially each copy (member) of a bulk issued ACDC set shares a template that both the Issuer and Issuee use to generate a given ACDC in that set without requiring that the Issuer and Issuee exchange and store a unique copy of each member of the set independently. This minimizes the data transfer and storage requirements for both the Issuer and the Issuee.

An ACDC provenance chain is connected via references to the SAIDs given by the top-level SAID, d, fields of the ACDCs in that chain. A given ACDC thereby makes commitments to other ACDCs. Expressed another way, an ACDC may be a node in a directed graph of ACDCs. Each directed edge in that graph emanating from one ACDC includes a reference to the SAID of some other connected ACDC. These edges provide points of correlation to an ACDC via their SAID reference. Private bulk issued ACDCs enable the Issuee to control better the correlatability of presentations using different presentation strategies.

For example, the Issuee could use one copy of a bulk-issued private ACDC per presentation even to the same verifier. This strategy would consume the most copies. It is essentially a one-time-use ACDC strategy. Alternatively, the Issuee could use the same copy for all presentations to the same verifier and thereby only permit the verifier to correlate between presentations it received directly but not between other verifiers. This limits the consumption to one copy per verifier. In yet another alternative, the Issuee could use one copy for all presentations in a given context with a group of verifiers, thereby only permitting correlation among that group.

In this context, we are talking about permissioned correlation. Any verifier that has received a complete presentation of a private ACDC has access to all the fields disclosed by the presentation but the terms of the chain-link confidentiality agreement may forbid sharing those field values outside a given context. Thus an Issuee may use a combination of bulk issued ACDCs with chain-link confidentiality to control permissioned correlation of the contents of an ACDC while allowing the SAID of the ACDC to be more public. The SAID of a private ACDC does not expose the ACDC contents to an un-permissioned third party. Unique SAIDs belonging to bulk issued ACDCs prevent third parties from making a provable correlation between ACDCs via their SAIDs in spite of those SAIDs being public. This does not stop malicious verifiers (as second parties) from colluding and correlating against the disclosed fields but it does limit provable correlation to the information disclosed to a given group of malicious colluding verifiers. To restate unique SAIDs per copy of a set of private bulk issued ACDC prevent un-permissioned third parties from making provable correlations in spite of those SAIDs being public unless they collude with malicious verifiers (second parties).

In some applications, chain-link-confidentiality is insufficient to

deter un-permissioned correlation. Some verifiers may be malicious with sufficient malicious incentives to overcome whatever counter incentives the terms of the contractual chain-link confidentiality may impose. In these cases, more aggressive technological anti-correlation mechanisms such as bulk issued ACDCs may be useful. To

elaborate, in spite of the fact that chain-link confidentiality terms of use may forbid such malicious correlation, making such correlation more difficult technically may provide better protection than chain-link confidentiality alone [[41]].

It is important to note that any group of colluding malicious verifiers may always make a statistical correlation between presentations despite technical barriers to cryptographically provable correlation. In general, there is no cryptographic mechanism that precludes statistical correlation among a set of colluding verifiers because they may make cryptographically unverifiable or unprovable assertions about information presented to them that may be proven as likely true using merely statistical correlation techniques.

11.3. Basic Bulk Issuance

The amount of data transferred between the Issuer and Issuee (or recipient of an untargeted ACDC) at issuance of a set of bulk issued ACDCs may be minimized by using a hierarchical deterministic derivation function to derive the value of the UUID, u, fields from a shared secret salt [Salt].

As described above, there are several ways that the Issuer may securely share a secret salt. Given that the Issuer and Issuee (or recipient when untargeted) AIDs are each controlled by an Ed25519 key pair(s), a corresponding X15519 asymmetric encryption key pair(s) may be derived from the controlling Ed25519 key pair(s) [EdSC][PSEd][TMEd]. An X25519 public key may be derived from an Ed25519 public key. Likewise, an X25519 private key may be derived from an Ed25519 private key [KeyEx].

In an interactive approach, the Issuer derives a public asymmetric X25519 encryption key from the Issuee's published Ed25519 public key and the Issuee derives a public asymmetric X25519 encryption key from the Issuer's published Ed25519 public key. The two then interact via

a Diffie-Hellman (DH) key exchange to create a shared symmetric encryption key [KeyEx][DHKE]. The shared symmetric encryption key may be used to encrypt the secret salt or the shared symmetric encryption key itself may be used has high entropy cryptographic material from which the secret salt may be derived.

In a non-interactive approach, the Issuer derives an X25519 asymmetric public encryption key from the Issuee's (or recipient's) public Ed25519 public key. The Issuer then encrypts the secret salt with that public asymmetric encryption key and signs the encryption with the Issuer's private Ed25519 signing key. This is transmitted to the Issuee, who verifies the signature and decrypts the secret

salt using the private X25519 decryption key derived from the Issuee's private Ed25519 key. This non-interactive approach is more scalable for AIDs that are controlled with a multi-sig group of signing keys. The Issuer can broadcast to all members of the Issuee's (or recipient's) multi-sig signing group individually asymmetrically encrypted and signed copies of the secret salt.

In addition to the secret salt, the Issuer also provides a template of the private ACDC but with empty UUID, u, and SAID, d, fields at the top-level of each nested block with such fields. Each UUID, u, field value is then derived from the shared salt with a deterministic path prefix that indexes both its membership in the bulk issued set and its location in the ACDC. Given the UUID, u, field value, the associated SAID, d, field value may then be derived. Likewise, both full and compact versions of the ACDC may then be generated. This generation is analogous to that described in the section for selective disclosure ACDCs but extended to a set of private ACDCs.

The initial element in each deterministic derivation path is the string value of the bulk-issued member's copy index _k_, such as "0", "1", "2" etc. Specifically, if _k_ denotes the index of an ordered set of bulk issued private ACDCs of size _M_, the derivation path starts with the string _"k"_ where _k_ is replaced with the decimal or hexadecimal textual representation of the numeric index _k_. Furthermore, a bulk-issued private ACDC with a private attribute section uses _"k"_ to derive its top-level UUID and _"k/0"_ to derive its attribute section UUID. This hierarchical path is extended to any nested private attribute blocks. This approach is further extended to enable bulk issued selective disclosure ACDCs by using a

similar hierarchical derivation path for the UUID field value in each of the selectively disclosable blocks in the array of attributes. For example, the path _"k/j"_ is used to generate the UUID of attribute index _j_ at bulk-issued ACDC index _k_.

In addition to the shared salt and ACDC template, the Issuer also provides a list of signatures of SAIDs, one for each SAID of each copy of the associated compact bulk-issued ACDC. The Issuee (or recipient) can generate on-demand each compact or uncompacted ACDC from the template, the salt, and its index _k_. The Issuee does not need to store a copy of each bulk issued ACDC, merely the template, the salt, and the list of signatures.

The Issuer MUST also anchor in its KEL an issuance proof digest seal of the set of bulk issued ACDCs. The issuance proof digest seal makes a cryptographic commitment to the set of top-level SAIDS belonging to the bulk issued ACDCs. This protects against later forgery of ACDCs in the event the Issuer's signing keys become compromised. A later attempt at forgery requires a new event or new

version of an event that includes a new anchoring issuance proof digest seal that makes a cryptographic commitment to the set of newly forged ACDC SAIDS. This new anchoring event of the forgery is therefore detectable.

Similarly, to the process of generating a selective disclosure attribute ACDC, the issuance proof digest is an aggregate that is aggregated from all members in the bulk-issued set of ACDCs. The complication of this approach is that it must be done in such a way as to not enable provable correlation by a third party of the actual SAIDS of the bulk-issued set of ACDCs. Therefore the actual SAIDs must not be aggregated but blinded commitments to those SAIDs instead. With blinded commitments, knowledge of any or all members of such a set does not disclose the membership of any SAID unless and until it is unblinded. Recall that the purpose of bulk issuance is to allow the SAID of an ACDC in a bulk issued set to be used publicly without correlating it in an un-permissioned provable way to the SAIDs of the other members.

The basic approach is to compute the aggregate denoted, _B_, as the digest of the concatenation of a set of blinded digests of bulk issued ACDC SAIDS. Each ACDC SAID is first blinded via concatenation

to a UUID (salty nonce) and then the digest of that concatenation is concatenated with the other blinded SAID digests. Finally, a digest of that concatenation provides the aggregate.

Suppose there are _M_ ACDCs in a bulk issued set. Using zero-based indexing for each member of the bulk issued set of ACDCs, such that index _k_ satisfies _k in {0, ..., M-1}, let *d_k_ denote the top-level SAID of an ACDC in an ordered set of bulk-issued ACDCs. Let _v_k_ denote the UUID (salty nonce) or blinding factor that is used to blind that said. The blinding factor, _v_k_, is NOT the top-level UUID, u, field of the ACDC itself but an entirely different UUID used to blind the ACDC's SAID for the purpose of aggregation. The derivation path for _v_k_ from the shared secret salt is _"k."_ where _k_ is the index of the bulk-issued ACDC.

Let _c_k = v_k + d_k_, denote the blinding concatenation where _+_ is the infix concatenation operator.
Then the blinded digest, _b_k_, is given by,
_b_k = H(c_k) = H(v_k + d_k)_,
where _H_ is the digest operator.

The aggregation of blinded digests, _B_, is given by,
_B = H(C(b_k for all k in {0, ..., M-1}))_,
where _C_ is the concatenation operator and _H_ is the digest operator. This aggregate, _B_, provides the issuance proof digest.

The aggregate, _B_, makes a blinded cryptographic commitment to the ordered elements in the list _[b_0, b_1, ..., b_(M-1)]_. A commitment to _B_ is a commitment to all the _b_k_ and hence all the d_k.

Given sufficient collision resistance of the digest operator, the digest of an ordered concatenation is not subject to a birthday attack on its concatenated elements [BDC][BDay][QCHC][HCR][Hash].

Disclosure of any given _b_k_ element does not expose or disclose any discoverable information detail about either the SAID of its associated ACDC or any other ACDC's SAID. Therefore one may safely disclose the full list of _b_k_ elements without exposing the blinded bulk issued SAID values, d_k.

Proof of inclusion in the list of blinded digests consists of checking the list for a matching value. A computationally efficient way to do this is to create a hash table or B-tree of the list and then check for inclusion via lookup in the hash table or B-tree.

A proof of inclusion of an ACDC in a bulk-issued set requires disclosure of _v_k_ which is only disclosed after the disclosee has accepted (agreed to) the terms of the rule section. Therefore, in the event the _Disclosee_ declines to accept the terms of disclosure, then a presentation/disclosure of the compact version of the ACDC does not provide any point of correlation to any other SAID of any other ACDC from the bulk set that contributes to the aggregate _B_. In addition, because the other SAIDs are hidden by each _b_k_ inside the aggregate, _B_, even a presentation/disclosure of,
_[b_0, b_1, ..., b_(M-1)]_
does not provide any point of correlation to the actual bulk-issued ACDC without disclosure of its _v_k_. Indeed if the _Discloser_ uses a metadata version of the ACDC in its _offer_ then even its SAID is not disclosed until after acceptance of terms in the rule section.

To protect against later forgery given a later compromise of the signing keys of the Issuer, the issuer MUST anchor an issuance proof seal to the ACDC in its KEL. This seal binds the signing key state to the issuance. There are two cases. In the first case, an issuance/revocation registry is used. In the second case, an issuance/revocation registry is not used.

When the ACDC is registered using an issuance/revocation TEL (Transaction Event Log) then the issuance proof seal digest is the SAID of the issuance (inception) event in the ACDC's TEL entry. The issuance event in the TEL uses the aggregate value, _B_, as its identifier value. This binds the aggregate, _B_, to the issuance proof seal in the Issuer's KEL through the TEL.

Recall that the usual purpose of a TEL is to provide a verifiable data registry that enables dynamic revocation of an ACDC via a state of the TEL. A verifier checks the state at the time of use to check if the associated ACDC has been revoked. The Issuer controls the state of the TEL. The registry identifier, ri, field is used to identify the public registry which usually provides a unique TEL entry for each ACDC. Typically the identifier of each TEL entry is the SAID of the TEL's inception event which is a digest of the

event's contents which include the SAID of the ACDC. In the bulk issuance case, however, the TEL's inception event contents include the aggregate, _B_, instead of the SAID of a given ACDC. Recall that the goal is to generate an aggregate value that enables an Issuee to selectively disclose one ACDC in a bulk-issued set without leaking the other members of the set to un-permissioned parties (second or third). Using the aggregate, _B_ of blinded ACDC saids as the TEL registry entry identifier allows all members of the bulk-issued set to share the same TEL without any third party being able to discover which TEL any ACDC is using in an un-permissioned provable way. Moreover, a second party may not discover in an un-permissioned way any other ACDCs from the bulk-issued set not specifically disclosed to that second party. In order to prove to which TEL a specific bulk issued ACDC belongs, the full inclusion proof must be disclosed.

When the ACDC is not registered using an issuance/revocation TEL then the issuance proof seal digest is the aggregate, _B_, itself.

In either case, this issuance proof seal makes a verifiable binding between the issuance of all the ACDCs in the bulk issued set and the key state of the Issuer at the time of issuance.

A _Discloser_ may make a basic provable non-repudiable selective disclosure of a given bulk issued ACDC, at index _k_ by providing to the _Disclosee_ four items of information (proof of inclusion). These are as follows:

* The ACDC in compact form (at index _k_) where _d_k_ as the value of its top-level SAID, d, field.

* The blinding factor, _v_k_ from which _b_k = H(v_k + d_k)_ may be computed.

* The list of all blinded SAIDs, _[b_0, b_1, ..., b_(M-1)]_ that includes _b_k_.

* The signature(s), _s_k_, of the Issuee on the ACDC's top level SAID, _d_k_, field.

A _Disclosee_ may then verify the disclosure by:

* computing _d_j_ on the disclosed compact ACDC.

* computing _b_k = H(v_k + d_k)_

* confirming that the computed _b_k_ appears in the provided list _[b_0, b_1, ..., b_(M-1)]_.

* computing the aggregate _B_ from the provided list _[b_0, b_1, ..., b_(M-1)]_..

* confirming the presence of an issuance seal digest in the Issuer's KEL that makes a commitment to the aggregate, _B_, either directly or indirectly through a TEL registry entry.

* verifying the provided signature(s), _s_k_, of the Issuee on the provided top level SAID, _d_k_, field.

The last 3 steps that culminate with verifying the signature(s) require determining the key state of the Issuer at the time of issuance, this may require additional verification steps as per the KERI, PTEL, and CESR-Proof protocols.

The requirement of an anchored issuance proof seal means that the forger Must first successfully publish in the KEL of the issuer an inclusion proof digest seal bound to a set of forged bulk issued ACDCs. This makes any forgery attempt detectable. To elaborate, the only way to successfully publish such a seal is in a subsequent interaction event in a KEL that has not yet changed its key state via a rotation event. Whereas any KEL that has changed its key state via a rotation must be forked before the rotation. This makes the forgery attempt either both detectable and recoverable via rotation in any KEL that has not yet changed its key state or detectable as duplicity in any KEL that has changed its key state. In any event, the issuance proof seal makes any later attempt at forgery using compromised keys detectable.

11.3.1. Inclusion Proof via Merkle Tree

The inclusion proof via aggregated list may be somewhat verbose when there are a very large number of bulk issued ACDCs in a given set. A more efficient approach is to create a Merkle tree of the blinded SAID digests, _b_k_ and set the aggregate _B_ value as the Merkle tree root [Mrkl].

The Merkle tree needs to have appropriate second-pre-image attack protection of interior branch nodes [TwoPI][MTSec]. The discloser then only needs to provide a subset of digests from the Merkle tree to prove that a given digest, _b_k_ contributed to the Merkle tree

root digest. For a small numbered bulk issued set of ACDCs, the added complexity of the Merkle tree approach may not be worth the savings in verbosity.

11.3.2. Bulk Issuance of Private ACDCs with Unique Issuee AIDs

One potential point of provable but un-permissioned correlation among any group of colluding malicious _Disclosees_ (Second-Party verifiers) may arise when the same Issuee AID is used for presentation/disclosure to all _Disclosees_ in that group. Recall that the contents of private ACDCs are not disclosed except to permissioned _Disclosees_ (Second-Parties), thus a common _Issuee_ AID would only be a point of correlation for a group of colluding malicious verifiers. But in some cases removing this un-permissioned point of correlation may be desirable.

One solution to this problem is for the _Issuee_ to use a unique AID for the copy of a bulk issued ACDC presented to each _Disclosee_ in a given context. This requires that each ACDC copy in the bulk-issued set use a unique _Issuee_ AID. This would enable the _Issuee_ in a given context to minimize provable correlation by malicious _Disclosees_ against any given _Issuee_ AID. In this case, the bulk issuance process may be augmented to include the derivation of a unique Issuee AID in each copy of the bulk-issued ACDC by including in the inception event that defines a given Issuee's self-addressing AID, a digest seal derived from the shared salt and copy index _k_. The derivation path for the digest seal is _"k/0."_ where _k_ is the index of the ACDC. To clarify _"k/0."_ specifies the path to generate the UUID to be included in the inception event that generates the Issuee AID for the ACDC at index _k_. This can be generated on-demand by the _Issuee_. Each unique _Issuee_ AID would also need its own KEL. But generation and publication of the associated KEL can be delayed until the bulk-issued ACDC is actually used. This approach completely isolates a given _Issuee_ AID to a given context with respect to the use of a bulk-issued private ACDC. This protects against even the un-permissioned correlation among a group of malicious Disclosees (Second Parties) via the Issuee AID.

11.4. Independent TEL Bulk-Issued ACDCs

Recall that the purpose of using the aggregate _B_ for a bulk-issued set from which the TEL identifier is derived is to enable a set of bulk issued ACDCs to share a single public TEL that provides dynamic revocation but without enabling un-permissioned correlation to any other members of the bulk set by virtue of the shared TEL. This enables the issuance/revocation/transfer state of all copies of a set

of bulk-issued ACDCs to be provided by a single TEL which minimizes the storage and compute requirements on the TEL registry while

providing selective disclosure to prevent un-permissioned correlation via the public TEL.

However, in some applications where chain-link confidentiality does not sufficiently deter malicious provable correlation by Disclosees (Second-Party verifiers), an Issuee may benefit from using ACDC with independent TELs but that are still bulk-issued.

In this case, the bulk issuance process must be augmented so that each uniquely identified copy of the ACDC gets its own TEL entry in the registry. Each Disclosee (verifier) of a full presentation/ disclosure of a given copy of the ACDC only receives proof of one uniquely identified TEL and can NOT provably correlate the TEL state of one presentation to any other presentation because the ACDC SAID, the TEL identifier, and the signature of the issuer on the SAID of a given copy will all be different for each copy. There is therefore no point of provable correlation permissioned or otherwise.

The obvious drawbacks of this approach (independent unique TELs for each private ACDC) are that the size of the registry database increases as a multiple of the number of copies of each bulk-issued ACDC and every time an Issuer must change the TEL state of a given set of copies it must change the state of multiple TELs in the registry. This imposes both a storage and computation burden on the registry. The primary advantage of this approach, however, is that each copy of a private ACDC has a uniquely identified TEL. This minimizes un-permissioned Third-Party exploitation via provable correlation of TEL identifiers even with colluding Second-Party verifiers. They are limited to statistical correlation techniques.

In this case, the set of private ACDCs may or may not share the same Issuee AID because for all intents and purposes each copy appears to be a different ACDC even when issued to the same Issuee. Nonetheless, using unique Issuee AIDs may further reduce correlation by malicious Disclosees (Second-Party verifiers) beyond using independent TELs.

To summarize the main benefit of this approach, in spite of its storage and compute burden, is that in some applications chain-link

confidentiality does not sufficiently deter un-permissioned malicious collusion. Therefore completely independent bulk-issued ACDCs may be used.

12. Appendix: Cryptographic Strength and Security

12.1. Cryptographic Strength

For crypto-systems with _perfect-security_, the critical design parameter is the number of bits of entropy needed to resist any practical brute force attack. In other words, when a large random or pseudo-random number from a cryptographic strength pseudo-random number generator (CSPRNG) [CSPRNG] expressed as a string of characters is used as a seed or private key to a cryptosystem with _perfect-security_, the critical design parameter is determined by the amount of random entropy in that string needed to withstand a brute force attack. Any subsequent cryptographic operations must preserve that minimum level of cryptographic strength. In information theory [IThry][ITPS] the entropy of a message or string of characters is measured in bits. Another way of saying this is that the degree of randomness of a string of characters can be measured by the number of bits of entropy in that string. Assuming conventional non-quantum computers, the convention wisdom is that, for systems with information-theoretic or perfect security, the seed/key needs to have on the order of 128 bits (16 bytes, 32 hex characters) of entropy to practically withstand any brute force attack. A cryptographic quality random or pseudo-random number expressed as a string of characters will have essentially as many bits of entropy as the number of bits in the number. For other crypto-systems such as digital signatures that do not have perfect security, the size of the seed/key may need to be much larger than 128 bits in order to maintain 128 bits of cryptographic strength.

An N-bit long base-2 random number has 2^N different possible values. Given that no other information is available to an attacker with perfect security, the attacker may need to try every possible value before finding the correct one. Thus the number of attempts that the attacker would have to try maybe as much as 2^(N-1). Given available

computing power, one can easily show that 128 is a large enough N to make brute force attack computationally infeasible.

Let's suppose that the adversary has access to supercomputers. Current supercomputers can perform on the order of one quadrillion operations per second. Individual CPU cores can only perform about 4 billion operations per second, but a supercomputer will parallelly employ many cores. A quadrillion is approximately 2^50 = 1,125,899,906,842,624. Suppose somehow an adversary had control over one million (2^20 = 1,048,576) supercomputers which could be employed in parallel when mounting a brute force attack. The adversary could then try 2^50 * 2^20 = 2^70 values per second (assuming very conservatively that each try only took one operation). There are about 3600 * 24 * 365 = 313,536,000 = 2^(log_2313536000)=2^24.91 ~= 2^25 seconds in a year. Thus this set of a million super computers could try 2^(50+20+25) = 2^95 values per year. For a 128-bit random

number this means that the adversary would need on the order of 2^(128-95) = 2^33 = 8,589,934,592 years to find the right value. This assumes that the value of breaking the cryptosystem is worth the expense of that much computing power. Consequently, a cryptosystem with perfect security and 128 bits of cryptographic strength is computationally infeasible to break via brute force attack.

12.2. Information Theoretic Security and Perfect Security

The highest level of cryptographic security with respect to a cryptographic secret (seed, salt, or private key) is called _information-theoretic security_ [ITPS]. A cryptosystem that has this level of security cannot be broken algorithmically even if the adversary has nearly unlimited computing power including quantum computing. It must be broken by brute force if at all. Brute force means that in order to guarantee success the adversary must search for every combination of key or seed. A special case of _information-theoretic security_ is called _perfect-security_ [ITPS]. _Perfect-security_ means that the ciphertext provides no information about the key. There are two well-known cryptosystems that exhibit _perfect security_. The first is a _one-time-pad_ (OTP) or Vernum Cipher [OTP][VCphr], the other is _secret splitting_ [SSplt], a type of secret sharing [SShr] that uses the same technique as a _one-time-pad_.

13. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

* SAID - Self-Addressing Identifier - any identifier which is deterministacly generated out of the content, digest of the content

14. Security Considerations

TODO Security

15. IANA Considerations

This document has no IANA actions.

16. References

16.1. Normative References

[ACDC_ID] Smith, S., "IETF ACDC (Authentic Chained Data Containers) Internet Draft", 2022, <https://github.com/trustoverip/tswg-acdc-specification>.

[CBOR] "CBOR Mapping Object Codes", n.d., <https://en.wikipedia.org/wiki/CBOR>.

[CESR_ID] Smith, S., "IETF CESR (Composable Event Streaming Representation) Internet Draft", 2022, <https://github.com/WebOfTrust/ietf-cesr>.

[DIDK_ID] Feairheller, P., "IETF DID-KERI Internet Draft", 2022, <https://github.com/WebOfTrust/ietf-did-keri>.

[IPEX_ID] Feairheller, P., "IPEX (Issuance and Presentation EXchange) Internet Draft", 2022, <https://github.com/WebOfTrust/keripy/blob/master/ref/Peer2PeerCredentials.md>.

[JSch] "JSON Schema", n.d., <https://json-schema.org>.

[JSch_202012]
"JSON Schema 2020-12", n.d., <https://json-schema.org/draft/2020-12/release-notes.html>.

[JSON] "JavaScript Object Notation Delimeters", n.d., <https://www.json.org/json-en.html>.

[KERI_ID] Smith, S., "IETF KERI (Key Event Receipt Infrastructure) Internet Draft", 2022, <https://github.com/WebOfTrust/ietf-keri>.

[MGPK] "Msgpack Mapping Object Codes", n.d., <https://github.com/msgpack/msgpack/blob/master/spec.md>.

[OOBI_ID] Smith, S., "IETF OOBI Internet Draft", 2022, <https://github.com/WebOfTrust>.

[Proof_ID] Feairheller, P., "IETF CESR-Proof Internet Draft", 2022, <https://github.com/WebOfTrust/ietf-cesr-proof>.

[PTEL_ID] Feairheller, P., "IETF PTEL (Public Transaction Event Log) Internet Draft", 2022, <https://github.com/WebOfTrust/ietf-ptel>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC4627] "The application/json Media Type for JavaScript Object Notation (JSON)", n.d., <https://datatracker.ietf.org/doc/rfc4627/>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,

May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8259] "JSON (JavaScript Object Notation)", n.d., <https://datatracker.ietf.org/doc/html/rfc8259>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", 4 December 2020, <https://datatracker.ietf.org/doc/rfc8949/>.

[SAID_ID] Smith, S., "IETF SAID (Self-Addressing IDentifier) Internet Draft", 2022, <https://github.com/WebOfTrust/ietf-said>.

16.2. Informative References

[ACDC_TF] "ACDC (Authentic Chained Data Container) Task Force", n.d., <https://wiki.trustoverip.org/display/HOME/ACDC+%28Authentic+Chained+Data+Container%29+Task+Force>.

[ACDC_WP] "Authentic Chained Data Containers (ACDC) White Paper", n.d., <https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/ACDC.web.pdf>.

[BDay] "Birthday Attack", n.d., <https://en.wikipedia.org/wiki/Birthday_attack>.

[BDC] "Birthday Attacks, Collisions, And Password Strength", n.d., <https://auth0.com/blog/birthday-attacks-collisions-and-password-strength/>.

[CAcc] "Cryptographic Accumulator", n.d., <https://en.wikipedia.org/wiki/Accumulator_(cryptography)>.

[CLC] "Chain-Link Confidentiality", n.d., <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2045818>.

[CSPRNG] "Cryptographically-secure pseudorandom number generator (CSPRNG)", n.d., <https://en.wikipedia.org/wiki/Cryptographically-secure_pseudorandom_number_generator>.

[DHKE] "Diffie-Hellman Key Exchange", n.d., <https://www.infoworld.com/article/3647751/understand-diffie-hellman-key-exchange.html>.

[Dots] Rodriguez, M. and P. Neubauer, "Constructions from Dots and Lines", 2010, <https://arxiv.org/pdf/1006.2361.pdf>.

[DRB] "Dictionary Attacks, Rainbow Table Attacks and how Password Salting defends against them", n.d., <https://www.commonlounge.com/discussion/2ee3f431a19e4deabe4aa30b43710aa7>.

[DSig] "Digital Signature", n.d., <https://en.wikipedia.org/wiki/Digital_signature>.

[EdSC] "The Provable Security of Ed25519: Theory and Practice Report", n.d., <https://eprint.iacr.org/2020/823>.

[GLEIF] "GLEIF (Global Legal Entity Identifier Foundation)", n.d., <https://www.gleif.org/en/>.

[GLEIF_KERI] "GLEIF with KERI Architecture", n.d., <https://github.com/WebOfTrust/vLEI>.

[GLEIF_vLEI] "GLEIF vLEI (verifiable Legal Entity Identifier)", n.d., <https://www.gleif.org/en/lei-solutions/gleifs-digital-strategy-for-the-lei/introducing-the-verifiable-lei-vlei>.

[Hash] "Cryptographic Hash Function", n.d., <https://en.wikipedia.org/wiki/Cryptographic_hash_function>.

[HCR] "Hash Collision Resistance", n.d., <https://en.wikipedia.org/wiki/Collision_resistance>.

[IDSys] "Identity System Essentials", n.d., <https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/Identity-System-Essentials.pdf>.

[IETF] "IETF (Internet Engineering Task Force", n.d., <https://www.ietf.org>.

[IThry] "Information Theory", n.d., <https://en.wikipedia.org/wiki/Information_theory>.

[ITPS] "Information-Theoretic and Perfect Security", n.d., <https://en.wikipedia.org/wiki/Information-theoretic_security>.

[JSchCp] "Schema Composition in JSON Schema", n.d., <https://json-schema.org/understanding-json-schema/reference/combining.html>.

[JSchCx] "Complex JSON Schema Structuring", n.d., <https://json-schema.org/understanding-json-schema/structuring.html#base-uri>.

[JSchId] "JSON Schema Identification", n.d., <https://json-schema.org/understanding-json-schema/structuring.html#schema-identification>.

[JSchRE] "Regular Expressions in JSON Schema", n.d., <https://json-schema.org/understanding-json-schema/reference/regular_expressions.html>.

[KERI] Smith, S., "Key Event Receipt Infrastructure (KERI)", 2021, <https://arxiv.org/abs/1907.02143>.

[KeyEx] "Key Exchange", n.d., <https://libsodium.gitbook.io/doc/key_exchange>.

[KG] "Knowledge Graphs", n.d., <https://arxiv.org/pdf/2003.02320.pdf>.

[Level] "Security Level", n.d., <https://en.wikipedia.org/wiki/Security_level>.

[Mrkl] "Merkle Tree", n.d., <https://en.wikipedia.org/wiki/Merkle_tree>.

[MTSec] "Merkle Tree Security", n.d., <https://blog.enuma.io/update/2019/06/10/merkle-trees-not-that-simple.html>.

[OTP] "One-Time-Pad", n.d., <https://en.wikipedia.org/wiki/One-time_pad>.

[PGM] Angles, R., "The Property Graph Database Model", 2018, <http://ceur-ws.org/Vol-2100/paper26.pdf>.

[PSEd] Brendel, J., Cremers, C., Jackson, D., and M. Zhao, "The Provable Security of Ed25519: Theory and Practice", 2021 IEEE Symposium on Security and Privacy (SP) , 24 May 2021, <https://ieeexplore.ieee.org/document/9519456?denied=>.

[QCHC] "Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?", n.d., <https://cr.yp.to/hash/collisioncost-20090823.pdf>.

[RB] "Rainbow Table", n.d., <https://en.wikipedia.org/wiki/Rainbow_table>.

[RC] "Ricardian Contract", n.d., <https://en.wikipedia.org/wiki/Ricardian_contract>.

[Salt] "Salts, Nonces, and Initial Values", n.d., <https://medium.com/@fridakahsas/salt-nonces-and-ivs-whats-the-difference-d7a44724a447>.

[SShr] "Secret Sharing", n.d., <https://en.wikipedia.org/wiki/Secret_sharing>.

[SSplt] "Secret Splitting", n.d., <https://www.ciphermachinesandcryptology.com/en/secretsplitting.htm>.

[TMal] "Transaction Malleability", n.d., <https://en.wikipedia.org/wiki/Transaction_malleability_problem>.

[TMEd] "Taming the many EdDSAs", n.d., <https://eprint.iacr.org/2020/1244.pdf>.

[TOIP] "Trust Over IP (ToIP) Foundation", n.d., <https://trustoverip.org>.

[Twin] "Digital Twin", n.d., <https://en.wikipedia.org/wiki/Digital_twin>.

[TwoPI] "Second Pre-image Attack on Merkle Trees", n.d.,

<https://flawed.net.nz/2018/02/21/attacking-merkle-trees-with-a-second-preimage-attack/>.

[VCEnh] "VC Spec Enhancement Strategy Proposal", n.d., <https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/VC_Enhancement_Strategy.md>.

[VCphr] "Vernom Cipher (OTP)", n.d., <https://www.ciphermachinesandcryptology.com/en/onetimepad.htm>.

[vLEI] "vLEI (verifiable Legal Entity Identifier) Definition", n.d., <https://github.com/WebOfTrust/vLEI>.

[W3C_DID] "W3C Decentralized Identifiers (DIDs) v1.0", n.d., <https://w3c-ccg.github.io/did-spec/>.

[W3C_VC] "W3C Verifiable Credentials Data Model v1.1", n.d., <https://www.w3.org/TR/vc-data-model/>.

[XORA] "XORA (XORed Accumulator)", n.d., <https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/XORA.md>.

Acknowledgments

TODO acknowledge.

Author's Address

S. Smith
ProSapien LLC
Email: sam@prosapien.com