Authentic Chained Data Containers (ACDC)

Create technical specifications in markdown. Based on the original Spec-Up, extended with Terminology tooling

---

CONTENTS

Authentic Chained Data Containers (ACDC)

Specification Status: v1.1

DOI

https://doi.org/10.5281/zenodo.18792085

Latest Draft:

https://github.com/trustoverip/kswg-acdc-specification

Author:

- Samuel Smith, Prosapien
- Philip Feairheller, GLEIF

Editors:

- Kevin Griffin, GLEIF

Contributors:

- Samuel Smith, Prosapien
- Philip Feairheller, HealthKERI
- Kevin Griffin, GLEIF
- Nuttawut Kongsuwan
- Henk van Cann, Blockchainbird
- Kor Dwarshuis, Blockchainbird

Participate:

GitHub repo

Commit history

Introduction

One 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. This could enable an “authentic” web where all data on the web has verifiable proof-of-authorship.

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 different types of verifiable authorizations such as, entitlements, permissions, rights, or credentials. A chained (treed) proof-of-authority enables delegation of authority and hence delegated authorizations. These proofs of authorship and/or authority provide provenance of an ACDC itself and, by association, any data that is so conveyed.

To elaborate, the dictionary definition of credential is “evidence of authority, status, rights, entitlement to privileges, etc.” 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 also 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 [54]. 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) [56] [57]. Thus, ACDCs may be used to construct knowledge graphs expressed as property graphs [58]. ACDCs enable securely-attributed and privacy-protecting knowledge graphs. Semantically modulated verifiable provenanceable graphs enable authenticatable, delegable, attenuable, and aggregable authorizations and attributions.

The ACDC specification (including its disclosure mechanisms) leverages two primary cryptographic operations, namely digests and digital signatures [48] [52]. These operations, when used in an ACDC, MUST have a security level, cryptographic strength, or entropy of approximately 128 bits (nominally) [53]. (See Annex A 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. The digest enables confidentiality because secure attribution of the commitment to the digest may be verified without disclosing the digested data. Later, confidential disclosure of the digested data can be verified against the digest. ACDCs leverage this property to enable compact chains of ACDCs that commit to (anchor or seal) data via digests. The data actually contained in an ACDC, therefore, may be merely its digest. The digested data may thereby be equivalently but more compactly and confidentially authenticated and authorized by the chain of ACDCs.

There are several different variants of ACDCs. These enable different types of disclosure mechanisms that provide differing levels of protection from exploitation and enable functional privacy with provisional authentication. A notable feature of ACDCs is support for Contractually Protected Disclosure that provides more comprehensive privacy protection than mere Selective Disclosure alone might provide.

Status of This Memo

Information about the current status of this document, any errata, and how to provide feedback on it, may be obtained at https://github.com/trustoverip/kswg-acdc-specification.

Copyright Notice

Terms of Use

WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND WHETHER OR NOT THE OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Scope

Implementation design of a protocol-based data container specification that enables secure attribution of data within the container to a cryptographically derived identifier using the KERI and CESR protocols. This makes the containers authenticatable. Containers may be chained together using cryptographic digests to form a verifiable extensible graph data structure. This makes the containers chainable. There is no reliance on trusted third parties. The application scope includes any electronically transmitted information. The implementation dependency scope assumes CESR and KERI.

Normative references

The normative documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

- ISO/IEC 7498-1:1994 Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model, including Requirement Levels (IETF RFC-2119).

See Bibliography - Normative Section

Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at https://www.iso.org/obp
- IEC Electropedia: available at http://www.electropedia.org/

⇨ **Attribute** (*attribute*)

Property	Value
Original Term	attribute
Link	https://trustoverip.github.io/kerisuite-glossary/#term:attribute
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a top-level field map within an ACDC that provides a property of an entity that is inherent or assigned to the entity.

Also see: attribute

⇨ **Authentic Chained Data Container** (ACDC, *authentic-chained-data-container*)

Property	Value
Original Term	authentic-chained-data-container
Link	https://trustoverip.github.io/kerisuite-glossary/#term:authentic-chained-data-container
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a directed acyclic graph with properties to provide a verifiable chain of proof-of-authorship. See the full specification

Source: Dr. S.Smith, 2024

Explained briefly, an ACDC or authentic-data-container proves digital data consistency and authenticity in one go. An ACDC cryptographically secures commitment to the data contained, and its identifiers are self-addressing, which means they point to themselves and are also contained in the data.

More in extended KERI glossary

⇨ **Autonomic Identifier** (AID, *autonomic-identifier*)

Property	Value
Original Term	autonomic-identifier
Link	https://trustoverip.github.io/kerisuite-glossary/#term:autonomic-identifier
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a self-managing cryptonymous identifier that must be self-certifying (self-authenticating) and must be encoded in CESR as a qualified Cryptographic primitive.

Source: Dr. S.Smith, 2024

An identifier that is self-certifying-identifier and self-sovereign-identity (or *self-managing*).

More in extended KERI glossary

⇨ **Chain-link Confidential Disclosure** (*chain-link-confidential-disclosure*)

Property	Value
Original Term	chain-link-confidential-disclosure
Link	https://trustoverip.github.io/kerisuite-glossary/#term:chain-link-confidential-disclosure
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

contractual restrictions and liability imposed on a recipient of a disclosed ACDC that contractually link the obligations to protect the disclosure of the information contained within the ACDC to all subsequent recipients as the information moves downstream. The Chain-link Confidential Disclosure provides a mechanism for protecting against un-permissioned exploitation of the data disclosed via an ACDC.

Source: Dr. S.Smith

More in extended KERI glossary

Compact Disclosure (*compact-disclosure*)

Property	Value
Original Term	compact-disclosure
Link	https://trustoverip.github.io/kerisuite-glossary/#term:compact-disclosure
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a disclosure of an ACDC that discloses only the SAID(s) of some or all of its field maps. Both Partial and Selective Disclosure rely on Compact Disclosure.

Source: Dr. S. Smith

More in extended KERI glossary

Contractually Protected Disclosure (*contractually-protected-disclosure*)

Property	Value
Original Term	contractually-protected-disclosure
Link	https://trustoverip.github.io/kerisuite-glossary/#term:contractually-protected-disclosure
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a discloser of an ACDC that leverages a Graduated Disclosure so that contractual protections can be put into place to minimize the leakage of information that can be correlated. A Contractually Protected Disclosure partially or selectively reveals the information contained within the ACDC in the initial interaction with the recipient and discloses further information only after the recipient agrees to the terms established by the discloser. More information may be progressively revealed as the recipient agrees to additional terms.

Source: Dr. S. Smith

More in extended KERI glossary

Controller (*controller*)

Property	Value
Original Term	controller
Link	https://trustoverip.github.io/kerisuite-glossary/#term:controller
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

an entity that can cryptographically prove the control authority over an AID and make changes on the associated KEL. A controller of a multi-sig AID may consist of multiple controlling entities.

Source: Dr. S.Smith, 2024

More in extended KERI glossary

⇨ **Disclosee** (*disclosee*)

Property	Value
Original Term	disclosee
Link	https://trustoverip.github.io/kerisuite-glossary/#term:disclosee
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a role of an entity that is a recipient to which an ACDC is disclosed. A Disclosee may or may not be the Issuee of the disclosed ACDC.

Source: Dr. S. Smith

More in extended KERI glossary

⇨ **Discloser** (*discloser*)

Property	Value
Original Term	discloser
Link	https://trustoverip.github.io/kerisuite-glossary/#term:discloser
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a role of an entity that discloses an authentic-chained-data-container. A Discloser may or may not be the Issuer of the disclosed ACDC.

Source: Dr. S. Smith

More in extended KERI glossary

Duplicity (*duplicity*)

Property	Value
Original Term	duplicity
Link	https://trustoverip.github.io/kerisuite-glossary/#term:duplicity
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

the existence of more than one version of a Verifiable key-event-log for a given AID .

Source: Dr. S.Smith, 2024

More in extended KERI glossary

Edge (*edge*)

Property	Value
Original Term	edge
Link	https://trustoverip.github.io/kerisuite-glossary/#term:edge
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a top-level field map within an ACDC that provides edges that connect to other ACDCs, forming a labeled property graph (LPG).

Source: Dr. S. Smith

More in extended KERI glossary

Framing Code (*framing-code*)

Property	Value
Original Term	framing-code
Link	https://trustoverip.github.io/kerisuite-glossary/#term:framing-code
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a code that delineates a number of characters or bytes, as appropriate, that can be extracted atomically from a stream.

Source: Dr. S. Smith

Full Disclosure (*full-disclosure*)

Property	Value
Original Term	full-disclosure
Link	https://trustoverip.github.io/kerisuite-glossary/#term:full-disclosure
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a disclosure of an ACDC that discloses the full details of some or all of its field maps. In the context of selective-disclosure, Full Disclosure means detailed disclosure of the selectively disclosed attributes, not the detailed disclosure of all selectively disclosable attributes. In the context of partial-disclosure, Full Disclosure means detailed disclosure of the field map that was so far only partially disclosed.

Source: Dr. S. Smith

More in extended KERI glossary

Graduated Disclosure (*graduated-disclosure*)

Property	Value
Original Term	graduated-disclosure
Link	https://trustoverip.github.io/kerisuite-glossary/#term:graduated-disclosure
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a disclosure of an ACDC that does not reveal its entire content in the initial interaction with the recipient and, instead, partially or selectively reveals only the information contained within the ACDC necessary to further a transaction with the recipient. A Graduated disclosure may involve multiple steps where more information is progressively revealed as the recipient satisfies the conditions set by the discloser. compact-disclosure, partial-disclosure, selective-disclosure, and full-disclosure are all Graduated disclosure mechanisms.

Source: Dr. S. Smith

More in extended KERI glossary

Inception event (*inception-event*)

Property	Value
Original Term	inception-event
Link	https://trustoverip.github.io/kerisuite-glossary/#term:inception-event
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

an establishment-event that provides the incepting information needed to derive an AID and establish its initial Key state.

Source Sam Smith

More in extended KERI glossary

Information theoretic security (ITPS, *information-theoretic-security*)

Property	Value
Original Term	information-theoretic-security
Link	https://trustoverip.github.io/kerisuite-glossary/#term:information-theoretic-security
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

the highest level of cryptographic security with respect to a cryptographic secret (seed, salt, or private key).

Source: Dr. S.Smith

Interaction Event (*interaction-event*)

Property	Value
Original Term	interaction-event
Link	https://trustoverip.github.io/kerisuite-glossary/#term:interaction-event
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

Non-establishment Event that anchors external data to the key-state as established by the most recent prior establishment event.

Source Sam Smith

More in extended KERI glossary

Issuee (*issuee*)

Property	Value
Original Term	issuee
Link	https://trustoverip.github.io/kerisuite-glossary/#term:issuee
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a role of an entity to which the claims of an ACDC are asserted.

Source: Dr. S. Smith

More in extended KERI glossary

Issuer (*issuer*)

Property	Value
Original Term	issuer
Link	https://trustoverip.github.io/kerisuite-glossary/#term:issuer
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a role of an entity that asserts claims and creates an ACDC from these claims.

Source: Dr. S. Smith

More in extended KERI glossary

Key-state (*key-state*)

Property	Value
Original Term	key-state
Link	https://trustoverip.github.io/kerisuite-glossary/#term:key-state
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a set of currently authoritative keypairs for an AID and any other information necessary to secure or establish control authority over an AID. This includes current keys, prior next key digests, current thresholds, prior next thresholds, witnesses, witness thresholds, and configurations. A key-state of an AID is first established through an inception event and may be altered by subsequent rotation events.

Source: Dr. S.Smith

Also see validator

Operator (*operator*)

Property	Value
Original Term	operator
Link	https://trustoverip.github.io/kerisuite-glossary/#term:operator
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

an optional field map in the Edge section that enables expression of the edge logic on edge subgraph as either a unary operator on the edge itself or an m-ary operator on the edge group.

Source: Dr. S.Smith

More in extended KERI glossary

⇨ **Partial Disclosure** (*partial-disclosure*)

Property	Value
Original Term	partial-disclosure
Link	https://trustoverip.github.io/kerisuite-glossary/#term:partial-disclosure
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a disclosure of an ACDC that partially discloses its field maps using Compact Disclosure. The Compact Disclosure provides a cryptographically equivalent commitment to the yet-to-be-disclosed content, and the later exchange of the uncompacted content is verifiable to an earlier Partial Disclosure. Unlike Selective disclosure, a partially disclosable field becomes correlatable to its encompassing block after its Full Disclosure.

Source: Dr. S. Smith

More in extended KERI glossary

⇨ **Percolated discovery** (*percolated-discovery*)

Property	Value
Original Term	percolated-discovery
Link	https://trustoverip.github.io/kerisuite-glossary/#term:percolated-discovery
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a discovery mechanism for information associated with an AID or a SAID, which is based on Invasion Percolation Theory. Once an entity has discovered such information, it may in turn share what it discovers with other entities. Since the information so discovered is end-verifiable, the percolation mechanism and percolating intermediaries do not need to be trusted.

Source: Dr. S. Smith

percolated-information-discovery

More in extended KERI glossary

Perfect security (*perfect-security*)

Property	Value
Original Term	perfect-security
Link	https://trustoverip.github.io/kerisuite-glossary/#term:perfect-security
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a special case of Information theoretic security ITPS

Source: Dr. S.Smith

Primitive (*primitive*)

Property	Value
Original Term	primitive
Link	https://trustoverip.github.io/ctwg-general-glossary/#term:primitive
Owner	trustoverip
Repo	ctwg-general-glossary
Commit hash	73604cf54c9f238b0d239daecc413a045f61c75e

a serialization of a unitary value. All Primitives in KERI must be expressed in composable-event-streaming-representation.

Source: Dr. S.Smith

⇨ **Rotation event** (*rotation-event*)

Property	Value
Original Term	rotation-event
Link	https://trustoverip.github.io/kerisuite-glossary/#term:rotation-event
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

an Establishment Event that provides the information needed to change the Key state, which includes a change to the set of authoritative keypairs for an AID.

Source: Dr. S.Smith

More in extended KERI glossary

⇨ **Rules** (*rules*)

Property	Value
Original Term	rules
Link	https://trustoverip.github.io/kerisuite-glossary/#term:rules
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a top-level field map within an ACDC that provides a legal language as a Ricardian Contract, which is both human and machine-readable and referenceable by a cryptographic digest.

Source: Dr. S. Smith

More in extended KERI glossary

Schema (*schema*)

Property	Value
Original Term	schema
Link	https://trustoverip.github.io/kerisuite-glossary/#term:schema
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

the said of a JSON schema that is used to issue and verify an ACDC.

Source: Dr. S.Smith

More in extended KERI glossary

Selective Disclosure (*selective-disclosure*)

Property	Value
Original Term	selective-disclosure
Link	https://trustoverip.github.io/kerisuite-glossary/#term:selective-disclosure
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a disclosure of an ACDC that selectively discloses its attributes using Compact Disclosure. The set of selectively disclosable attributes is provided as an array of blinded blocks where each attribute in the set has its own dedicated blinded block. Unlike Partial Disclosure, the selectively disclosed fields are not correlatable to the so far undisclosed but selectively disclosable fields in the same encompassing block.

Source: Dr. S. Smith

More in extended KERI glossary

⇨ **Self-addressing Identifier** (SAID, *self-addressing-identifier*)

Property	Value
Original Term	self-addressing-identifier
Link	https://trustoverip.github.io/kerisuite-glossary/#term:self-addressing-identifier
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

any identifier that is deterministically generated out of the content, or a digest of the content.

Source: Dr. S. Smtih

More in extended KERI glossary

⇨ **SEMVER** (*semver*)

Property	Value
Original Term	semver
Link	https://trustoverip.github.io/kerisuite-glossary/#term:semver
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

Semantic Versioning Specification 2.0. See also (https://semver.org)
[https://semver.org]

Stream (*stream*)

Property	Value
Original Term	stream
Link	https://trustoverip.github.io/kerisuite-glossary/#term:stream
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a CESR Stream is any set of concatenated Primitives, concatenated groups of Primitives, or hierarchically composed groups of primitive s.

Source: Dr. S. Smith

More in extended KERI glossary

Targeted ACDC (*targeted-acdc*)

Property	Value
Original Term	targeted-acdc
Link	https://trustoverip.github.io/kerisuite-glossary/#term:targeted-acdc
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

an ACDC with the presence of the Issuee field in the attribute or attribute aggregate sections.

Source: Dr. S.Smith

untargeted-acdc

More in extended KERI glossary

Unpermissioned correlation (*unpermissioned-correlation*)

Property	Value
Original Term	unpermissioned-correlation
Link	https://trustoverip.github.io/kerisuite-glossary/#term:unpermissioned-correlation
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

a correlation established between two or more disclosed ACDCs whereby the discloser of the ACDCs does not permit the disclosee to establish such a correlation.

Source: Dr. S. Smith

More in extended KERI glossary

Untargeted ACDC (*untargeted-acdc*)

Property	Value
Original Term	untargeted-acdc
Link	https://trustoverip.github.io/kerisuite-glossary/#term:untargeted-acdc
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

an ACDC without the presence of the Issuee field in the attribute or attribute aggregate sections.

Source: Dr. S. Smith

targeted-acdc

More in extended KERI glossary

Validator (*validator*)

Property	Value
Original Term	validator
Link	https://trustoverip.github.io/ctwg-general-glossary/#term:validator
Owner	trustoverip
Repo	ctwg-general-glossary
Commit hash	73604cf54c9f238b0d239daecc413a045f61c75e

any entity or agent that evaluates whether or not a given signed statement as attributed to an identifier is valid at the time of its issuance.

Source: Dr. S. Smith

Verifiable data registry (*verifiable-data-registry*)

Property	Value
Original Term	verifiable-data-registry
Link	https://trustoverip.github.io/kerisuite-glossary/#term:verifiable-data-registry
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

A role a system might perform by mediating issuance and verification of ACDCs. See verifiable data registry.

verifier

Property	Value
Original Term	verifier
Link	https://trustoverip.github.io/ctwg-main-glossary/#term:verifier
Owner	trustoverip
Repo	ctwg-main-glossary
Commit hash	00608bb6ce7f7e6285f202b580548d61ee716621

A role an agent performs to perform verification of one or more proofs of the claims in a digital credential or other verifiable data.

See also: relying party; issuer, holder.

Mental model: W3C Verifiable Credentials Data Model Roles & Information Flows

Supporting definitions:

W3C VC: A role an entity performs by receiving one or more verifiable credentials, optionally inside a verifiable presentation for processing. Other specifications might refer to this concept as a relying party.

eSSIF-Lab: a component that implements the capability to request peer agents to present (provide) data from credentials (of a specified kind, issued by specified parties), and to verify such responses (check structure, signatures, dates), according to its principal's verifier policy.

NIST The entity that verifies the authenticity of a digital signature using the public key.

weight

Property	Value
Original Term	weight
Link	https://trustoverip.github.io/kerisuite-glossary/#term:weight
Owner	trustoverip
Repo	kerisuite-glossary
Commit hash	ab23372198bed5cda8bc0ac0289cead7a06d3320

an optional field map in the Edge section that provides edge weight property that enables directed weighted edges and operators that use weights.

Source: Dr. S.Smith

More in extended KERI glossary

ACDC Structure

Ordered Nested Field Maps

An ACDC may be modeled abstractly as a nested `key: value` mapping. To avoid confusion with the cryptographic use of the term key, the term field is used instead 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)`. This terminology can be qualified, 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. A nested set of fields is called 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 `{}` [8] [7] [9]. Given this equivalence, the term block or nested block also may be used 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 without having to reorder them using lexicographic (alphabetical) order. ACDCs MUST use 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, JSON only will be used for examples [8] [7]. The basic set of normative field labels in ACDC field maps is defined in the following table.

Top-Level Fields

The following table defines the top-level fields in an ACDC and their order of appearance. Some fields are OPTIONAL but all fields that appear MUST appear in a defined order. This section defines which fields are REQUIRED and which are OPTIONAL.

Label	Title	Description
`v`	Version String	Regexable format: `ACDCMmmGggKKKKSSSS.` that provides protocol type, version, CESR genus version, serialization type, size, and terminator
`t`	Message Type	Three-character Message type
`d`	Message Digest SAID ([3])	Self-referential fully qualified cryptographic digest of enclosing map
`u`	UUID	Random Universally Unique Identifier as fully qualified high entropy pseudo-random string, a salty nonce.
`i`	Issuer AID (AID)	Autonomic Identifier whose control authority is established via KERI verifiable Key State.
`rd`	Registry Digest SAID ([3])	Issuance and/or revocation, transfer, or retraction registry for ACDC
`s`	Schema	Either the SAID [3] of a JSON Schema block or the block itself.
`a`	Attribute	Either the SAID [3] 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.

Label	Title	Description
`e`	Edge	Either the SAID [3] of a block of Edges or the block itself.
`r`	Rule	Either the SAID [3] a block of Rules or the block itself.

Field Ordering

When present, the top-level fields MUST appear in the following order: `[v, t, d, u, i, rd, s, a, A, e, r]`.

Required Fields

The following fields are REQUIRED `[v, d, i, s]` i.e. they MUST appear in any ACDC (not to be confused with other message types in the ACDC protocol).

Other Reserved Fields

The following table defines non-top-level fields whose labels MUST be reserved. These MAY appear at other levels besides the top-level of an ACDC.

Label	Title	Description
`d`	Digest SAID ([3])	Self-referential fully qualified cryptographic digest of enclosing map.
`u`	UUID	Random Universally Unique Identifier as fully qualified high entropy pseudo-random string, a salty nonce.
`i`	Identifier AID (AID)	Context-dependent AID as determined by its enclosing map such as Issuee identifier.
`rd`	Registry Digest SAID ([3])	Issuance and/or revocation, transfer, retraction, or usage registry for ACDC when not at top-level
`dt`	Datetime	Context-dependent ISO datetime string
`n`	Node	SAID [3] of another ACDC as the terminating point (vertex) of a directed Edge that connects the encapsulating ACDC node to the specified ACDC node as a fragment of a distributed property graph (PG).
`o`	Operator	Either unary operator on Edge or m-ary operator on edge-group in Edge section. Enables expressing of Edge logic on Edge subgraph.

Label	Title	Description
`w`	Weight	Edge weight property that enables default property for directed weighted edges and operators on directed weighted edges.
`l`	Legal Language	Text of Ricardian contract clause.
`dt`	Datetime	Context-dependent ISO datetime string

Compact Labels

The primary field labels are compact in that they MUST use only one or at most 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.

Version String Field

The Version String, `v`, field MUST be the first field in any top-level ACDC field map encoded in JSON, CBOR, or MGPK [RFC4627] [12] [13] [14]. It provides a regular expression target for determining a serialized field map's serialization format and size (character count), constituting an ACDC message body. A Stream parser SHOULD use the Version String to extract and deserialize (deterministically) any serialized Stream of ACDC Message bodies. Each ACDC Message body in a Stream MAY use a different serialization type. The format for the Version String field value is defined in the CESR specification [1].

The protocol field, `PPPP` value in the Version String MUST be `ACDC` for the ACDC protocol. The protocol version field, `Mmm`, MUST encode the current major `M` and minor `mm` version of the ACDC protocol [1] used by the associated message. The CESR genus version field `Ggg` MUST encode the major `G` and minor `gg` version of the CESR protocol used to encode the associated message [1].

Legacy Version String Field Format

Compliant ACDC version 2.XX implementations MUST support the old ACDC version 1.x Version String format to properly verify Message bodies created with 1.x format events. The old version 1.X Version String format is defined in the CESR specification [1]. The protocol field, `PPPP` value in the version string MUST be `ACDC` for the ACDC protocol. The version field, `vv`, MUST encode the old version of the ACDC protocol [1].

Message Type Field

The presence of the message type field, labeled `t`, is optional for messages of type `acm` with non-CESR-native serialization kinds. It MUST be present in any `acm` message that uses a CESR serialization kind, namely JSON, CBOR, and MGPK,. It MUST be present in messages with any other message type, regardless of the kind of serialization. A message without a message type, `acm`, field is assumed to be of type `acm`. Therefore, the `acm` message type is effectively the default message type for ACDC protocol messages. The protocol type in either the version string of non-CESR-native serialization kinds or the version field for native CESR serialization kind MUST be `ACDC`. To elaborate, the message type field MUST appear in all native CESR messages; it MAY only not appear in non-CESR-native serialization kinds of `acm` type messages. See the Annex ACDC Protocol Message Types#acdc-protocol-message-types).

Self-addressing Identifier (SAID) Fields

Some fields in ACDCs MAY have for their value either a field map or a SAID. A SAID follows the SAID protocol [3]. 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.

More specifically, special top-level ACDC fields MUST have for their value either a serialized field map or the SAID of that field map. Recall that each SAID provides a stable, universal, cryptographically verifiable, and agile reference to its encapsulating block (serialized field map). The special top-level SAID fields in ACDC include the `d`, and `rd` fields. Moreover, the value of top-level `s`, `a`, `e`, and `r` fields MAY be replaced by the SAID of their associated field map. The values of the `s`, `a`, `e`, and `r` fields provide normatively defined sections of an ACDC. When replaced by their SAID, these top-level field values represent their associated sections in *compact* form.

A cryptographic commitment (such as a digital signature or a cryptographic digest) on some other given cryptographic digest, all with sufficient cryptographic strength, including collision resistance [34] [35], of the digest(s), is equivalent to a commitment to thea actual serialized block on which the given digest was derived (computed). Specifically, a digi-

tal 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. Moreover, a digest of a data block that, in turn, contains digests of yet other data blocks, makes a compact, hierarchical, verifiable cryptographic commitment to the digested data blocks. The combination of non-repudiable commitments and hierarchical commitments 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 43. This includes reasoning about the whole ACDC given by its top-level SAID, `d`, field, as well as reasoning about any of its special nested sections using the section SAIDs.

Registry SAID Field

When present at the top-level, the registry SAID, `rd` field value is the SAID of the initializing event for a given transaction event log (TEL) registry that maintains a dynamic state for the ACDC, such as issuance and revocation state. Typically, this field appears at the top level, but in some applications, such as bulk-issued bulk-issued ACDCs bulk-issued, it may appear nested inside the Attribute, `a`, or aggregate, `A`, section field maps attr and aggr. This nested appearance better facilitates contractually protected disclosure of the registry SAID. When the registry SAID, `rd` field is used at the top level for the Issuer's registry, a registry SAID, `rd` field that appears nested in the Attributed, `a`, or Aggregate, `A`, section MAY be used for some other registry, such as an application-specific or Issuer-specific registry.

Universally Unique Identifier (UUID) 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 [29]. Without the entropy provided by 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 [30] [31]. The resultant 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 as a cryptographic digest. Thus, an adversary could successfully discover the exact block by creating digests of all the elements of the power set, which may be small enough to be computationally feasible, rather than inverting the SAID itself. Sufficient entropy in the `u` field, however, 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, its 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 along with the block SAID. With an embedded UUID field value that contains sufficient cryptographic entropy, the block contents can only be discovered if the included UUID field is explicitly disclosed. 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 correlation-minimizing (what some call privacy-preserving) manner.

Autonomic IDentifier (AID) Fields

Some fields, such as the `i`, Issuer identifier field, MUST each have an AID as its value. An AID is a fully qualified Self-certifying Identifier (SCID) that follows the KERI protocol [2]. An AID MUST be derived from one or more `(public, private)` key pairs using asymmetric or public-key cryptography to create verifiable digital signatures [52]. Each AID SHOULD have set of one or more Controllers who each control a private key. By virtue of their private key(s), the Controllers MAY make statements on behalf of the associated AID backed by uniquely verifiable commitments via digital signatures on those statements. Any entity then MAY 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 MUST be established with the KERI protocol [2]. 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.

Datetime, `dt` Fields

The datetime, `dt` field value, if any, MUST be the ISO-8601 datetime string with microseconds and UTC offset as per IETF RFC-3339. This datetime is relative to the clock of the issuer. Attributes typically include one or more date time fields. In a given field map (block) the primary datetime will use the label, `dt`. Typically, this is the datetime of the issuance of the ACDC. Other datetime fields MAY include an expiration datetime or the like.

An example datetime string in this format is as follows:

```
2020-08-22T17:50:09.988921+00:00
```

Partially Disclosable Attribute Section Field

The top-level Attribute section `a`, field value MAY have as its value a nested field map. Each level of nesting MAY be fully expanded or represented by its SAID. When present, the `a` field value provides the so-called payload data of the ACDC. The `a` field syntax is described in more detail below. An ACDC MUST not have both an `a` field and an `A` field (see next section) when it has either.

Selectively Disclosable Aggregate Section Field

The top-level selectively disclosable 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 Aggregate, `A`, field depends on the type of Selective Disclosure mechanism employed. The baseline standard approach is that the Aggregate field value is a cryptographic digest of the concatenation of an ordered set of the SAIDs of blinded Attribute sub-blocks. Other approaches to forming an aggregate could be a Merkle tree root digest of an ordered set of cryptographic digests, or a cryptographic accumulator. The standard Selective Disclosure mechanism, i.e. a digest of the concatenation of the SAIDs of blinded attribute sub-blocks is described in detail in the Selective Disclosure section. When present, the `A` field value provides the so-called payload data of the ACDC. The `A` field syntax is described in more detail below. An ACDC MUST not have both a non-empty `a` field value and a non-empty `A` field value (see next section) when it has either.

Partially Disclosable Edge Section Field

The top-level Edge section `e` field value makes a cryptographically verifiable commitment to other ACDCs via references to their SAIDs. The `e` field syntax is described in more detail below.

Partially Disclosable Rule Section field

The top-level Rule section `r`, field value provides both human and machine readable legal language that MAY be associated with the ACDC. This is a type of Ricardian contract. The `r` field syntax is described in more detail below.

ACDC Variants

There are several variants of ACDCs determined by the presence/absence of certain fields and/or the value of those fields when used in combination. The primary ACDC variants are public, private, metadata, and bespoke. A given variant MAY be Targeted (Untargeted).

All the variants have two alternate forms, compact and non-compact. In the compact form of any variant, the values of the top-level fields for the Schema, Attribute, Aggregate, Edge, and Rule sections are the SAIDs (digests) of the corresponding expanded (non-compact) form of each section SAID. When in a non-compact form then one or more of the section fields may be partially or fully expanded. Additional variants arise from the presence or absence of different fields inside the Attribute or Attribute aggregate section.

At the top level, the presence (absence), of the UUID, `u`, field produces two additional variant combinations. These are private (public), respectively. In addition, a present but empty UUID, `u`, field produces a *Private Metadata* variant. Furthermore, a given variant MAY be either *Targeted* or *Untargeted* based on the presence of the Issuee field in the Attribute or Attribute aggregate sections. Similarly, any variant with an Attribute section MAY have nested sub-blocks within the Attribute section that are either compact or non-compact. This enables nested Partial Disclosure. The type of disclosure a given variant supports MAY be dependent on how the different sections appear in the ACDC.

An overview of each variant is explained below.

Most compact form SAID

As defined in the Schema section below, each ACDC variant is defined by a composable JSON Schema. The primary Operator for such composition is the `oneOf` operator. The use of the `oneOf` operator for an ACDC schema enables the use of any combination of compacted/uncompacted top-level sections as well as nested blocks. When compact, any section or nested block MAY be represented merely by its SAID [10] [39].

However, for the sake of verifiability, each section or block that is compactable MUST have only one SAID regardless of how many different variants its `oneOf` compositions allow. Therefore, there MUST be one and only one unambiguous way to compute the SAID of a compactifiable section or block with compactifiable nested blocks. This is called the “most compact form” SAID computation algorithm. The basis of the algorithm is that the SAID of a given block is verifiable against that block in expanded form, but that SAID can be embedded in an enclosing block in compact form. The SAID of the enclosing block makes a verifiable commitment to the given block via the appearance of only the SAID of the given block. The SAID of the “most compact form” can be computed by a depth-first search. First, compute the SAIDs of the expanded leaf nodes of the tree, then compact the leaves, then compute the SAIDs of their enclosing blocks, then compact those, and so on until the trunk has been reached.

To verify the SAID, just reverse the process. First, expand a given block, verify its SAID, expand its enclosed blocks, verify their SAIDs, and so on as deep as needed or until the leaves are reached. The main advantage of “most compact form” SAID computation is

that it enables verification of portions of a set of nested compactifiable subblocks against their SAIDs without requiring that the whole tree of subblocks be exposed. This is essential to Graduated Disclosure.

The algorithm for computing the “most compact form" of a block for computing its SAID is as follows.

- A SAIDed block is any block that has a SAID field `d` when in block-level expanded form. Any SAIDed block is compactifiable into the compact form consisting of a string field whose value is the SAID of the block-level expanded form and whose label is the block label. This compact form MUST appear as the first variant in the `oneOf` subschema list for its labeled field. In block-level expanded form, all fields within a block with non-SAIDed subblock values are fully expanded, whereas all fields with SAIDed subblocks are represented in compact form. The latter requires first representing each SAIDed subblock in block-level expanded form, computing its SAID, and then using that SAID to represent it in compact form. This process is recursively applied to multiple levels of SAIDed subblocks
- A SAIDed block that does not have any nested SAIDed subblocks as fields is a leaf block. Its SAID MUST be computed on the full expanded representation of the block. It MAY then be represented in the compact form but using the SAID computed on its fully expanded form. This enables its enclosing block to be represented in blocklevel expanded form. To clarify, because the leaf has no SAIDed subblocks, its block-level expanded form is fully expanded, including any fields with subblocks. This is a concern when it has subblocks whose schema has `oneOf` composition variants that omit any details. This is the case for a simple-compact edge or a simple-compact rule (see below under the Edge and Rule sections), which are each a nested (non-SAIDed) subblock with a non-SAID-based compact `oneOf` variant. These are special cases, and the most detailed variant of the subblock MUST be the fully expanded form. To elaborate, when a non-Saided sublock has variants given by `oneof` composition in its schema, then the most fully expanded `oneof` variant is used to compute the SAID of its enclosing SAIDed subblock.
- The computation of the SAID of any SAIDed block is as follows.
    - Fully expand any field within the block whose value is a Non-SAIDed sub-block.
    - Compact any field within the block whose value is a leaf block with its SAID computed on its fully expanded form.
    - Compact any non-leaf SAIDed block with its SAID computed on its block-level expanded form. The SAID of any non-leaf SAIDed subblock is recursively computed. Once all SAIDed subblocks have been represented in compact form, the SAID of the SAIDed block is then computed on the resultant representation.

To elaborate, the algorithm recursively compacts a nested set of SAIDed subblocks by doing a depth-first search of all SAIDed subblock fields in a given block, where each branch of the search terminates when it reaches a leaf subblock, whereupon it computes the SAID of the leaf, compacts it, and then continues the depth-first search until all the SAIDed subblock fields are compacted and then computes the SAID of that block and then ascends the tree, compacting subblocks and computing block SAIDs until it reaches the top-level block that is the ACDC itself.

Thereby, there is one and only one “most compact form” SAID for any given ACDC, as well as one and only one “most compact form” SAID for each of the ACDC top-level sections, regardless of all the different variants allowed by the `oneOf` compositions of any SAIDed subblocks. This “most compact form,” SAID, is what is used to reference an ACDC as the node value of an Edge or to reference a section.

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`. A special case is an ACDC with an Aggregate section `A` field. Although the compact aggregate value is not computed using the most compact SAID algorithm, the most compact form of the ACDC has the aggregate value as the value of the Aggregate section field. It effectively acts like a SAID for the purpose of compacting the ACDC. To clarify, an Aggregate section uses its own algorightm for compact and un-compact (expanded) forms. Its compact form is used in the compact form of is its enclosing ACDC.

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 [30] [31]. 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 SHOULD be considered a public (non-confidential) ACDC.

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 could provide a secure cryptographic digest that blinds the contents of the ACDC [48]. 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

[30] [31]. Therefore, the top-level, UUID, `u`, field could 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 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. This 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, including a UUID, `u`, field in a block also enables Selective Disclosure mechanisms described later in the section on Selective Disclosure.

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 contractually protected exchange without leaking correlation to the actual ACDC to be disclosed. A Disclosee MAY verify 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 provenanced Edges, and the Rules (terms-of-use). The top-level Attribute section, `a`, field value, or the top-level Attribute aggregate, `A` field value of a metadata ACDC, MAY be empty or missing 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 can verify the Issuer, the Schema, the provenance of the Edges, and the terms and conditions in the 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 Rules 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 cryptographic commitments from the Discloser are attached, not commitments from the Issuer. This prevents the Issuer commitments from being used as a point of correlation until after the Disclosee has

agreed to the terms in the rule section. Likewise, when a Contractually Protected Disclosure is used, the Issuer's commitments are not disclosed to the Disclosee until after the Disclosee has agreed to the terms. The Disclosee is protected from a forged Discloser because, ultimately, verification of the disclosed ACDC will fail if the Discloser does not eventually provide verifiable Issuer commitments. Nonetheless, should the potential Disclosee not agree to the terms of the disclosure expressed in the Rules section, then the Issuer's commitments are not leaked.

In the case of a bulk-issued metadata ACDC ACDC , the optional registry SAID, `rd` field at the top-level could be empty until after contractual protection is in place. Alternatively, the registry SAID could be provided by a nested registry SAID, `rd` field inside the Attribute `a`, or Aggregate `A` section. In this case, the top-level `rd` field would be missing or empty (when the message is a top-level fixed field type).

Top-level ACDC Sections

Schema Section

Type-is-schema

Notably, no top-level field types exist in an ACDC. This is because the Schema , `s`, field itself is the type field for the ACDC and its parts. 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 used by ACDCs is JSON Schema 2020-12 [10] [11]. JSON Schema supports composable schema (subschema), conditional Schema (subschema), and regular expressions in the Schema. Composability enables a Validator to ask and answer complex questions about the type of payload elements, including optional elements, while maintaining isolation between payload information and type (structure) information about the payload [39] [40] [41] [42]. A static but composed schema allows a verifiably immutable set of variants. Although the set is immutable, the variants enable graduated but secure disclosure. ACDC's use of JSON Schema MUST be in accordance with the ACDC-defined profile as defined herein. The exceptions are defined below.

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 [41] [42]. The top-level id, `$id`, field value in a JSON Schema provides a unique identifier of the Schema instance. In a non-ACDC schema, the value of the id, `$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. This field value MUST be 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.

The digest algorithm employed for generating Schema SAIDs MUST have an approximate cryptographic strength of 128 bits. The [3] MUST be generated in compliance with the ToIP SAID internet draft specification and MUST be encoded using CESR. The CESR encoding indicates the type of cryptographic digest used to generate the SAID.

When an id, `$id`, field appears in a subschema, it indicates a bundled subschema called a schema resource [41] [42]. The value of the id, `$id`, field in any ACDC bundled subschema resource MUST include the SAID of that subschema using one of the formats described below. The subschema so bundled MUST be verifiable against its referenced and embedded SAID value. This ensures secure bundling.

Static (Immutable) Schema

For security reasons, the full Schema of an ACDC MUST be completely self-contained and statically fixed (immutable) for that ACDC. This means that dynamic Schema references or dynamic Schema generation mechanisms MUST NOT be used, i.e are not 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, which is called 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 type of attack is called a semantic malleability attack. This may be considered as a type of transaction malleability attack [55].

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

To elaborate, the serialization of a static schema SHOULD be self-contained. A compact commitment to the detailed static Schema is 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 Schemas MUST be SADs. In other words, the Schema MUST therefore be SAIDified. The associated detailed static Schema (uncompacted SAD) is bound cryptographically and verifiably to its SAID.

The JSON Schema specification allows complex Schema references that MAY include non-local URI references [41] [42] [15] [16]. 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.

ACDC Sub-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 [15] [16].

This does not preclude the use of remotely cached SAIDified Schema resources because those resources are end-verifiable to their embedded SAID references. To clarify, a SAIDified Schema resource is itself a SAD 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 an SAID, is verifiable to the underlying resource when that resource is an SAD. This applies to JSON Schema as a whole as well as bundled subschema resources.

The value of the top-level id field, `$id`, MUST be a bare SAID. A bare SAID is simply the serialized SAID value using the appropriate CESR primitive encoding. Because only a bare SAID may be used to refer to a SAIDified Schema, only JSON Schema validators that support bare SAID references are compatible. By default, many, if not all, JSON Schema Validators support bare strings (non-URIs) for the Base URI provided by the top-level `$id` field value.

The value of any relative URI references for schema as provided by a `$ref` keyword MAY be resolved against a virtual base URI constructed from top-level `$id` field bare SAID value. A virtual base URI may employ one of the following schemes:

- 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 media-type of `schema+json`.
- The KERI OOBI specification provides a URL syntax that references a SAD resource by its SAID at the service endpoint indicated by that URL [4]. Such remote OOBI URLs are also safe because the provided SAD resource is verifiable against the

SAID in the OOBI URL. Therefore, OOBI URLs MAY be used as non-local URI references for JSON Schema [4] [15] [16].

- The `did:` URI scheme (W3C DID) 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:webs` or `did:keri` [5] [61] and may return DID docs or DID doc metadata with SAIDified schema or service endpoints that return SAIDified schema or OOBIs that return SAIDified schema [15] [16] [4]. 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 value is its 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 [41] [42]. 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 using the `$ref` keyword 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 [4]. Recall that a commitment to an SAID with sufficient collision resistance makes an equivalent secure commitment to its encapsulating block SAD. Thus, a 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, non-malleable) by virtue of either being relative to the self-contained whole or being referenced by its SAID. The static schema, in whole or in part, 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).

Schema dialect

The Schema dialect for ACDC 1.0 MUST be JSON Schema 2020-12 and is indicated by the identifier `"https://json-schema.org/draft/2020-12/schema"` [10] [11]. 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 a dynamically down-

loadable 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 `$schema` string value SHOULD cause the validation to fail. The string MUST be treated simply as an identifier that communicates the intended dialect to be processed by the Schema validation tool. When provided, the top-level `$schema` field value for ACDC version 1.0 MUST be "https://json-schema.org/draft/2020-12/schema".

Schema Versioning

Each schema MUST have at the top level a version field with the field label `version`. The value of the `version` field MUST be a semantic version string in the dotted decimal notation of the form "major.minor.patch". For example, "1.2.3" has a major version number of 1, a minor version number of 2, and a patch version of 3. This value is informative. The `version` field value is not used in the JSON Schema validation against the ACDC. Therefore, a given ACDC SHOULD properly pass the JSON Schema validation process regardless of the value of its schema `version` field. SEMVER

Recall that the value of the Schema ID, `$id` field in an ACDC Schema is a SAID that provides an encoded agile cryptographic digest of the contents of the schema. Any change to the schema, including a change to its `version` field, results in a new SAID. Any copy of a schema that verifies against the same SAID given by the Schema ID, `$id` field value, SHOULD be assumed to be identical to any other copy that verifies to the same SAID by virtue of the strong collision resistance of the digest employed.

This gives rise to two meanings of the word "version" when applied to an ACDC's Schema. One is the version given by the value of its `$id` field, and the other is the version given by its `version` field. The version provided by the `$id` field is cryptographically verifiable. Whereas the version provided by the `version` field is not. Thus, the latter is an informative indication of the version, and the former is a normative determiner of the version. In this sense, the Schema ID, `$id` field value as a SAID provides a cryptographically verifiable version indicator independent of the version field value. To avoid confusion, any change to the Schema that changes the value of the `$id` MUST also be reflected in a correspondingly unique value of its `version` field. Business logic may depend on consistency between the `version` field value with respect to the `$id` field value. Notwithstanding the actual value of the `version` field, the `$id` field value is the normative determiner of the Schema's true, verifiable version.

The JSON Schema for an ACDC MAY use composition operators (see below). This allows extensibility in the Schema such that, in some cases, ACDCs with newer Schema versions may be backward (in)compatible with older schema versions. A new schema version, as given by the `$id` field value, is considered backward incompatible with respect to a previous version of a Schema when any instance of an ACDC that validates (in the JSON Schema sense of validation) against the previous version of the Schema

may not be guaranteed to validate against the new version. Because any change to the `version` field value results in a different `$id` field value, the backward compatibility rule also applies to changes in the `version` field value. To comply with the semantic versioning rules, a backward incompatible Schema MUST have a higher major version number in its `version` field value than any backward incompatible version.

One complication with schema versioning arises when an ACDC has Edges. As discussed below in the Edge section, an Edge block MAY have a Schema, `s` field, which indicates that the ACDC node the edge points to MUST validate against the schema indicated by the Edge's Schema, `s` field value. Consequently, the version of the Edge's schema `s` field value MAY differ from the version of the Schema `s` field value in the ACDC node pointed to by that edge. If both schemas validate, then the Edge's schema version is backward compatible with the ACDC's schema version (node pointed to by the Edge). This means the Edge's schema version MAY have a higher minor version number than the ACDC's schema version (node pointed to by the Edge). If the Edge's schema does not validate against the ACDC node pointed to by the edge, then the Edge's schema is backward incompatible and MUST have a higher major version number than the ACDC's schema (node pointed to by the edge). In this latter case, ACDCs issued under the old major version MUST either be revoked and reissued to comply with the new major version schema, or the Edge's schema MUST include a `oneOf` composition that accepts either old or new major versions. In general, this would lead to extending that list of `oneOf`s in that Edge's schema field value to include an entry for each major version change.

Schema Availability

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 ecosystem governance framework (EGF) may impose schema availability constraints, this 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. The Issuer of an ACDC is REQUIRED to satisfy any availability constraints on its Schema that MAY be imposed by the application domain or ecosystem.

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 [10] [39]. When used for the

top-level attribute, `a`, edge, `e`, or rule, `r`, section field values, the `oneOf` subschema composition operator provides both compact and uncompacted variants. The provided ACDC MUST validate against an allowed combination of the composed variants. The Validator determines what decomposed variants the provided ACDC MUST also validate against. Decomposed variants MAY be dependent on the type of Graduated Disclosure. Essentially, a composable schema is a verifiable bundle of metadata (composed) about content that can then be verifiably unbundled (decomposed) later. The Issuer makes a single verifiable commitment to the bundle (composed Schema), and a recipient may then safely unbundle (decompose) the schema to validate any of the Graduated disclosure variants allowed by the composition.

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 also MUST validate against one of its specified `oneOf` variants. Typically, it's SAID or a generic object.

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 SHOULD 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 usually are specified in the EGF 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 EGF. The EGF may be specified solely by the Issuer for the ACDCs it generates or be specified by some mutually agreed upon ecosystem 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 [[44]]. This provides a powerful mechanism for protecting against unpermissioned 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 is called 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 [10][39]. 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` subschema 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.

Attribute Section

Reserved field labels

The following field labels are reserved at all nested field map levels in the Attribute section of an ACDC.

Label	Title	Description
`d`	Digest (SAID)	Self-referential fully qualified cryptographic digest of enclosing map.
`u`	UUID	Random Universally Unique Identifier as fully qualified high entropy pseudo-random string, a salty nonce.
`i`	Identifier (AID)	Context-dependent AID as determined by its enclosing map such as Issuee identifier.
`rd`	Registry Digest ([3])	Issuance and/or revocation, transfer, or retraction registry for ACDC when not at top-level
`dt`	Datetime	Issuer's relative ISO datetime string

The types of values for the `d`, `u`, `i`, and `dt` fields are described above for ACDC reserved field labels.

The following field labels are reserved at the top-level only of the Attribute section of an ACDC.

Label	Title	Description
`cargo`	Cargo	Field value is embedded encapsulated data

The Cargo, `cargo` field is described below.

Cargo, `cargo` field

The cargo, `cargo` field value, embedded, encapsulated data. The cargo field establishes a convention for encapsulating data using the ACDC as a data container for its "cargo". The semantics of its cargo filed value is opaque to its ACDC as data container.

This enables an ACDC to convey some other data format, including other types of verifiable credentials or application-specific data types. These MUST be serializable using a serialization that is compatible with the serialization kind of the ACDC that conveys them.

Compact Attribute Section Schema

When the value of the Attribute section has been compacted into its SAID, its JSON Schema subschema is as follows:

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

Attribute Section Variants

Two variants, namely, Targeted (Untargeted), are defined respectively by the presence (absence) of an Issuee, `i` field at the top-level of the uncompacted Attribute section block.

Two other variants, namely private (public), are defined respectively by the presence (absence) of a UUID, `u`, field at the top-level of the uncompacted Attribute section block.

These four variants MAY appear in combination.

Targeted Attribute Section

When present at the top level of the Attribute section, the Issuee, `i`, field value is the ACDC's Issuee AID. This Issuee AID is a provably controllable identifier that is the Target. This makes the ACDC a Targeted ACDC. Targeted ACDCs MAY be used for many different purposes, these include but are not limited too: entitlements, authorizations, or delegations directed at the Target. In other words, a targeted ACDC provides a container for authentic data that MAY also be used as some form of entitlement, 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.

To elaborate, the definition of the term credential is "evidence of authority, status, rights, entitlement to privileges, or the like". 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 partially disclosable. Because the actual semantics of Issuee may depend on the use case, the semantically precise albeit less common terms of

Issuer and Issuee are used in this specification. In some use cases, for example, an Issuee MAY be called the Holder; in others, the Subject of the ACDC. But no matter the use case, the `i` field designates the Issuee AID, i.e., Target. The Issuee MUST be designated by the `i` field value at the top-level of the attribute `a`, field block.

The ACDC MUST be “issued by” an Issuer and MUST be “issued to” an Issuee. This precise terminology does not bias or color the role (function) an Issuee plays in using 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 MUST be an AID, by definition, there is a provable Controller of that AID. Therefore, the Issuee Controller MAY make non-repudiable verifiable commitments to an issuance either directly via ephemeral digital signatures on behalf of its AID or indirectly by anchoring the SAID of the issuance in its KEL (Key Event Long). Therefore, any use of the ACDC by the Issuee MAY be securely attributed to the Issuee.

Importantly, the presence of an Issuee, `i`, field means that the associated Issuee MAY 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 counterparty, 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 MAY impose liability waivers or other terms of use on that Issuee.

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

In some applications, such as bulk-issued ACDCs Bulk, the presence of registry SAID, `rd` field inside the attribute, `a`, section may better facilitate contractually protected disclosure of the bulk-issued registry.

Contractual issuance and presentation exchanges are described later in the IPEX protocol section IPEX.

Untargeted Attribute Section

Consider the case where the Issuee, `i`, field is absent at the top level of the Attribute section. In this case, the ACDC has an Issuer but no Issuee. Because there is no provably controllable AID as a Target, the issuance is therefore Untargeted. The data in the Attribute section MAY be considered an undirected verifiable attestation or observation

by the Issuer. In this case, the issuance is effectively addressed “to whom it may concern.” An Untargeted ACDC enables verifiable authorship by the Issuer of the data in the Attributes section, but there is no specified counterparty and no verifiable mechanism for delegation of authority. Consequently, the Rules section MAY provide only contractual obligations on implied counterparties while still expressing contractual obligations or waivers of obligations on the Issuer.

To elaborate, an Untargeted ACDC provides a container for authentic data only (not authentic data as an authorization or entitlement to a specified party). However, authentic data remains 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, nonrepudiable measurements and publish them as ACDCs. These MAY be chained together to provide provenance of a chain of custody of any data. Furthermore, 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 head of that chain. These provenanceable chains of 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 impostors. Such data supply chains are also useful as a verifiable digital twin of a physical supply chain [54].

Targeted private-attribute section example

As discussed above, the presence of the `i` field at the top level of the Attribute section block makes this a targeted Attribute section. The AID given by the `i` field value is the target entity called the Issuee. The semantics of the Issuee SHOULD be defined by the Credential Frameworks in an associated Ecosystem Goveranance Framework (EGF) for the ACDC.

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 [48]. 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 [30]] [31]. 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 SHOULD 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 [[44]].

Consider an example of an Attribute Section as defined by the following JSON Schema subschema,

```
"a":
{
    "description": "Attribute Section",
    "oneOf":
    [
        {
            "description": "Attribute Section SAID",
            "type": "string"
          },
          {
            "description": "Attribute Section Detail",
            "type": "object",
            "required":
            [
              "d",
              "u",
              "i",
              "score",
              "name"
            ],
            "properties":
            {
              "d":
              {
                "description": "Attribute Section SAID",
                "type": "string"
              },
              "u":
              {
                "description": "Attribute Section 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` subschema 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 compliant Full Disclosure. To clarify, using the JSON Schema `oneOf` composition Operator enables the composed subschema to validate against both the compact and un-compacted value of the private-attribute section, `a`, field.

The attribute section field, whose value is the uncompacted attribute section block, is given by the following Python dictionary:

```
"a":
{
        "d": "EIMMcLl1w2KW2J3AD3twaESJO4u_fDFCdlMHjouojU8C",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXcw",
        "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf",
        "score": 96,
        "name": "Zoe Doe"
}
```

The attribute section field whose value is the compacted SAID is given by,

```
"a": "EIMMcLl1w2KW2J3AD3twaESJO4u_fDFCdlMHjouojU8C"
```

Targeted Public-attribute Section Example

Consider the following JSON Schema subschema for targeted attribute section block with the subschema below,

```
  "a":
  {
    "description": "Attribute Section",
    "oneOf":
    [
      {
        "description": "Attribute Section SAID",
        "type": "string"
      },
      {
        "description": "Attribute Section 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
      }
    ]
  }
```

As described above in the Schema section of this specification, the `oneOf` subschema 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 compliant Full Disclosure. To clarify, using the JSON Schema `oneOf` composition Operator enables the composed subschema to validate against both the compact and un-compacted value of the public-attribute section, `a`, field.

The attribute section field, whose value is the uncompacted attribute section block is given by the following Python dictionary:

```
"a":
{
        "d": "ELNJxIInWN4WAih9MQ4vVDrMRYnmhToS9a0gqjLfctOO",
        "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf",
        "score": 96,
        "name": "Zoe Doe"
}
```

The attribute section field whose value is the compacted SAID is given by,

```
"a": "ELNJxIInWN4WAih9MQ4vVDrMRYnmhToS9a0gqjLfctOO"
```

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.

As discussed above, the presence of the `i` field at the top level of the Attribute section block makes this a targeted Attribute section. The AID given by the `i` field value is the Target or Issuee. The semantics of the issuance SHOULD be defined by the Credential Frameworks of the EGF.

Given the absence of a `u` field at the top level of the Attributes block, however, 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 [30] [31]. 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 potentially provides a fixed correlation point to the attribute block field values despite the non-disclosure of those field values via a Compact Attribute section. 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.

Nested Partially Disclosable Attribute Section Example

Suppose that the subschema for the Attribute section of an ACDC is as follows:

```
"a":
{
  "description": "Attribute Section",
  "oneOf":
  [
    {
      "description": "Attribute Section SAID",
      "type": "string"
    },
    {
      "description": "Attribute Section Detail",
      "type": "object",
      "required":
      [
        "d",
        "u",
        "i",
        "name"
        "gpa",
        "grades"
      ],
      "properties":
      {
        "d":
        {
          "description": "Attribute Section SAID",
          "type": "string"
        },
        "i":
        {
          "description": "Issuee AID",
          "type": "string"
        },
        "name":
        {
          "description": "Student Full Name",
          "type": "string"
```

```
        },
        "gpa":
        {
          "description": "Grade Point Average",
          "type": "number"
        },
        "grades":
        {
          "description": "Grades Block",
          "oneOf":
          [
            {
              "description": "Block SAID",
              "type": "string"
            },
            {
              "description": "Block detail",
              "type": "object",
              "required":
              [
                "d",
                "u",
                "history"
                "english",
                "math"
              ],
              "properties":
              {
                "d":
                {
                  "description": "Block SAID",
                  "type": "string"
                },
                "u":
                {
                  "description": "Block UUID",
                  "type": "string"
                },
                "history":
                {
                  "description": "History Grade",
                  "type": "number"
                },
                "english":
                {
                  "description": "English Grade",
                  "type": "number"
                },
                "math":
```

```
                {
                  "description": "Math Grade",
                  "type": "number"
                }
              },
              "additionalProperties": false
            }
          ]
        },
        "additionalProperties": false
      }
    }
  ]
}
```

Suppose that the uncompacted value of the Attribute section of an ACDC is as follows:

Attribute section:

```
"a":
{
    "d": "ELI2TuO6mLF0cR_0iU57EjYK4dExHIHdHxlRcAdO6x-U",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXcw",
    "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf",
    "name": "Zoe Doe",
    "gpa": 3.5,
    "grades":
    {
        "d": "EFQnBFeKAeS4DAWYoKDwWXOT4h2-XaGk7-w4-2N4ktXy",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXcx",
        "history": 3.5,
        "english": 4.0,
        "math": 3.0
    }
}
```

The Attribute section subschema includes a `oneOf` composition operator at the grades subblock. The `grades` subblock has both a block level SAID, `d` and UUID, `u` field. This means that the `grades` subblock detail can be hidden so that only the top-level fields in the Attribute section are disclosed. The following shows a compatible partially disclosed variant of the Attribute section.

Partially disclosed Attribute section:

```
"a":
{
    "d": "ELI2TuO6mLF0cR_0iU57EjYK4dExHIHdHxlRcAdO6x-U",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXcw",
```

```
    "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf",
    "name": "Zoe Doe",
    "gpa": 3.5,
    "grades": "EFQnBFeKAeS4DAWYoKDwWXOT4h2-XaGk7-w4-2N4ktXy"
}
```

Notice that the compact form of the `grades` subblock has as the field value of the `grades` field the value of the SAID, `d` field in the expanded version (see above). This means that when the subblock detail is provided, a Validator can verify it against the SAID provided in the compact (partially disclosed) form.

The fully compact section would be as follows:

```
"a": "ELI2TuO6mLF0cR_0iU57EjYK4dExHIHdHxlRcAdO6x-U"
```

Notice that the compact form of the `a` block has as the field value of the `a` field the value of the SAID, `d` field in the expanded `a` section (see above).

Aggregate Section

A selectively disclosable blinded Aggregate section appears at the top level using the field label `A`. This is distinct from the field label `a` for a partially disclosable Attribute section. This makes clear (unambiguous) the semantics of each respective section's associated Schema. This also clearly reflects the fact that the value of a compact variant of the selectively disclosable Aggregate section is an aggregate, AGID, (Aggregate ID) not a SAID. As described in more detail below, the top-level selectively disclosable Aggregate section, `A`, field value is an aggregate of cryptographic commitments that makes a commitment to an ordered set (list) of selectively disclosable blinded attribute blocks.

The Uncompacted (Expanded) Aggregate section value is a list of blindable attribute blocks, where each attribute block represents a field map.

Reserved field labels

The following field labels are reserved at all nested field map levels in the Aggregate section of an ACDC.

Label	Title	Description
`d`	Digest (SAID)	Self-referential fully qualified cryptographic digest of enclosing map.
`u`	UUID	Random Universally Unique Identifier as fully qualified high entropy pseudo-random string, a salty nonce.
`i`	Identifier (AID)	Context-dependent AID as determined by its enclosing map such as Issuee identifier.
`dt`	Datetime	Issuer's relative ISO datetime string

The types of values for the `d`, `u`, `i`, and `dt` fields are described above for ACDC reserved field labels.

Basic selective disclosure mechanism

The purpose of the Aggregate section is to provide what herein is called "selective disclosure". The verifiable credential community has defined "selective disclosure" with respect to verifiable credentials to mean disclosing some information contained in a verifiable credential in a way that does not leak other information contained in the verifiable credential. ACDCs employ several different mechanisms that fall under the general description of graduated disclosure, where some information in an ACDC is disclosed while other information is not disclosed. Because there are several mechanisms that may be employed by ACDCs, different nomenclature is used to distinguish those mechanisms from one another. For example, the Attribute section above refers to its graduate disclosure mechanism as "partial disclosure". The Aggregate section described here uses a different graduated disclosure mechanism that more closely aligns what other verifiable credentials protocols call "selective disclosure". Therefore nomenclature for the Aggregate section's type of graduated disclosure is "selective disclosure".

The use of the term "selective disclosure" in this context should not be confused with its application in the financial sector, particularly in relation to publicly traded companies that make selective disclosures about their operations.

The basic Selective Disclosure mechanism provided by the Aggregate section is based on a single aggregated blinded commitment to a list (array) 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 cryptographic commitment is asserted on the aggregated blinded commitment, then any disclosure of a given value belonging to any of the embedded blinded commitments in the list is also non-repudiable. This approach does not require any more complex cryptography than digests and digital signatures or anchors in TELS or KELs that in turn are digitally signed. This satisfies the KERI design ethos of “minimally sufficient means”. The primary drawback of this approach is verbosity. It trades ease, simplicity, and the adoptability of implementation for size. A more complex but less verbose approach would be to replace the list of blinded commitments with a Merkle tree of those commitments, where the Merkle tree root becomes the aggregated blinded commitment. This approach is beyond the scope of this version of the ACDC specification.

Together, the combination of sufficient cryptographic entropy of the blinding factors, collision resistance of the digests, and unforgeability of the non-repudiable digital commitments of the inclusion proof prevents a potential Disclosee or adversary from discovering in a computationally feasible way the values of any undisclosed blinded attribute block details merely from the schema of the block detail together with 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 the actual value details, which include the blinding factor.

Selectively disclosable Aggregate of attribute blocks

The selectively disclosable aggregate section of an ACDC, provides an ordered set of Attributes as an list (array) of blinded blocks. In addition to its attribute-specific field or fields, each blinded attribute block in the set has its own SAID, `d`, field and UUID, `u`, field. All fields in a given block MUST be disclosed together as a set. When a blinded attribute block has more than one attribute field, then each field in the block is not independently selectively disclosable.

Notably, because the field labels for a given block only appear within that blinded block, the field labels are also blinded. The order of appearance of elements in an ‘anyOf’ subschema for the Aggregate array is not correlated to the actual order of appearance of the associated block or blocks in the blinded array itself.

This prevents inference based on location in the blinded array. It also prevents the correlation between contextualized variants of a field label that appear in different, selectively disclosable blocks. For example, a given localized or internationalized variant that uses a different language or other contextually correlatable information in the field labels itself is

not leaked by the disclosure of another variant. Each of these localization/internatialization variants would get their own blinded attributed block with unique field labels. To clarify, knowledge of one variant via its disclosed block does not leak information about some other, so far undisclosed variant.

Blinded attribute array

Given that each blindable Attribute block's UUID, `u`, field has sufficient cryptographic entropy, then each blindable Attribute block's SAID, `d`, field provides a secure cryptographic digest of its contents that effectively blinds the Attribute block's attribute value(s) 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 value, 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 blindable 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.

In addition to the blindable Attribute blocks, the blinded attribute array includes as its zeroth element the aggregate value. This is called the AGID (Aggregate ID). This serves the same function of the SAID of a field map. It provides a universally unique content addressable self-referential identifier of a given blindable attribute list.

Given a total of `N` blindable attribute blocks plus the AGID, a given blinded Attributes list has `N+1` elements. Let a_i represent the SAID, `d`, field of the attribute block at zero-based index 'i'. More precisely, the set of elements is expressed as the ordered set,

$\{a_i \text{ for all } i \text{ in } \{0, \ldots, N\}\}$.

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

$[a_0, a_1, \ldots, a_N]$.

Computation of the AGID (Aggregate ID)

The zeroth entry in the blinded attribute list represents a cryptogrphic digest of a given type. The length of this string is defined by the type of digest. This string is populated with dummy characters, namely `#` equal to the length of the digest type when CESR encoded as qb64 (text domain). The remaining `N` elements in the list are populated with the SAID values (CESR encoded qb64) of each the `N` blindable attribute blocks. This results in a list with `N+1` elements. This list is then serialized as a list in whatever serialization kind is used by the enclosing ACDC (CESR, JSON, CBOR, MGPK). A digest of this serialization is computer and qb64 value of that digest replaces the dummied values in the zeroth element. This digest value in CESR qb64 is the AGID. Working examples of AGID computation are provided below.

Inclusion proof via aggregated list digest (AGID)

All the a_i in the list are serialized and then aggregated into a single aggregate digest, denoted the AGID, by computing the digest of their ordered serialization in a list. This is expressed as follows:

AGID = H(C(a_i for all i in {0, …, N})), where 'H' is the digest (hash) Operator and 'C' is the ordered serialization operator.

In compact form, the value of the selectively disclosable top-level Attribute section, `A`, field is set to the aggregated value 'AGID'. This aggregate 'AGID' makes a blinded cryptographic commitment to all SAIDS in the list,

[a_1, …, a_N].

Please note that the zeroth element [a_0] is a placeholder for the AGID itself.

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 48].

Moreover, the aggregate AGID makes a commitment to each element a_i for i=1…N SAID. A block's SAID itself makes a blinded commitment to its block's (SAD) attribute value(s) in detail. Because each block is itself blinded by its SAID and UUID, disclosure of any given a_i SAID 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 of a given blindable block consists of checking the list for a matching SAID 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 of a block's SAID.

To elaborate, given that aggregate value 'AGID' appears as the compact value of the top-level Attribute section, `A`, field, the Selective Disclosure proof of the Attribute at index 'j' may be provided 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 full list including the AGID at a_0 all `N` Attribute block SAID digests, i.e. [a_0, a_1, …, a_N] that includes a_j.
- The ACDC in compact form with selectively disclosable Attribute section, `A`, field value set to aggregate 'AGID'.
- The presence of the issuance seal digest in the Issuer's KEL bound to the ACDC top-level SAID, `d`, field either directly or indirectly through a TEL Registry entry.

In common practice, the actual, detailed disclosed Attribute block may only be 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 with only the AGID and/or the list of blinded Attribute SAIDS, [a_0, a_1, …, a_{N-1}]. does not provide any point of correlation to any of the blinded Attribute values themselves. The Attributes of block 'j' are hidden by its SAID at a_j\ and the list of Attribute digests [a_0, a_1, …, a_{N-1}] is hidden by the aggregate 'AGID'. The Partial Disclosure needed to enable graduate disclosure for 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 'AGID' from the provided list [a_0, a_1, …, a_{N-1}].
- confirming that the computed 'AGID' matches the value, 'AGID', of the selectively disclosable Attribute section, `A`, field value in the provided ACDC (also the zeroth element a_0 of the list.
- computing the top-level SAID, `d` field of the provided ACDC in the most compact form.
- 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.

A private selectively disclosable ACDC provides significant correlation minimization because a Discloser may use a metadata ACDC prior to acceptance by the Disclosee of the terms of the Chain Link Confidentiality expressed in the Rule section [[44]]. Thus, only malicious Disclosees who violate Chain Link Confidentiality may correlate between presentations of a given private, selectively disclosable ACDC. Nonetheless, the malicious Disclosees are not able to discover any undisclosed Attributes.

Composed Schema for selectively disclosable Aggregate section

Because the Aggregate section's selectively disclosable blindable Attribute blocks 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, 'AGID', 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. An additional layer of `oneOf` is used to enable disclosure of either the block SAID or the block detail.

```
{
      "A":
      {
        "description": "Selectively disclosable attribute
```

```
aggregate section",
        "oneOf":
        [
          {
            "description": "Aggregate Section AGID",
            "type": "string"
          },
          {
            "description": "Selectively disclosable attribute
details",
            "type": "array",
            "uniqueItems": True,
            "items":
            {
              "anyOf":
              [
                {
                    "description": "Issue Block",
                    "oneOf":
                    [
                        {
                            "description": "Issuee Block SAID",
                            "type": "string"
                        },
                        {
                          "description": "Issuee Block Detail",
                          "type": "object",
                          "required":
                          [
                            "d",
                            "u",
                            "i"
                          ],
                          "properties":
                          {
                            "d":
                            {
                              "description": "Block SAID",
                              "type": "string"
                            },
                            "u":
                            {
                              "description": "Block UUID",
                              "type": "string"
                            },
                            "i":
                            {
                              "description": "Issuee SAID",
                              "type": "string"
```

```
                    }
                },
                "additionalProperties": False
            }
        ]
    },
    {
        "description": "Score Block",
        "oneOf":
        [
            {
                "description": "Score Block SAID",
                "type": "string"
            },
            {
              "description": "Score Block Detail",
              "type": "object",
              "required":
              [
                "d",
                "u",
                "score"
              ],
              "properties":
              {
                "d":
                {
                  "description": "Block SAID",
                  "type": "string"
                },
                "u":
                {
                  "description": "Block UUID",
                  "type": "string"
                },
                "score":
                {
                  "description": "Score Value",
                  "type": "integer"
                }
              },
              "additionalProperties": False
            }
        ]
    },
    {
        "description": "Name Block",
        "oneOf":
        [
```

```
                        {
                            "description": "Name Block SAID",
                            "type": "string"
                        },
                        {
                          "description": "Name Block Detail",
                          "type": "object",
                          "required":
                          [
                            "d",
                            "u",
                            "name"
                          ],
                          "properties":
                          {
                            "d":
                            {
                              "description": "Block SAID",
                              "type": "string"
                            },
                            "u":
                            {
                              "description": "Block UUID",
                              "type": "string"
                            },
                            "name":
                            {
                              "description": "Name Value",
                              "type": "string"
                            }
                          },
                          "additionalProperties": False
                        }
                    ]
                }
            ]
        }
      }
    ],
    "additionalProperties": False
  }
}
```

JSON Serialization

Consider the following Aggregate section in uncompacted, fully disclosed form computed using JSON serialization as follows:

```
"A":
[
        "EN5d44fTNM0M4kmMMVrsH0HwMLRLyb6SoJEV0ogkLdXx",
        {
            "d": "EI2lwi1ZKrs-bDwgEreOhEh-W2O5xrOm5T-QCyMuX5V4",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcw",
            "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf"
        },
        {
            "d": "EC-vU19URXX8ztfWdp_j2HHr1lJsqtGa1YHtZrg6-GMR",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcx",
            "score": 96
        },
        {
            "d": "EKYLUIpDXNT0ujSdoNOT5pLp0okOKW3mAbg-M7K5OO_C",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcy",
            "name": "Zoe Doe"
        }
]
```

Note that the zeroth element in the list is the value of the AGID as `EN5d44fTNM0M4kmMMVrsH0HwMLRLyb6SoJEV0ogkLdXx`. Also note that the value of the `i` field is the Issuee of the ACDC as `ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf`.

In fully compact form, the Aggregate section is as follows:

```
"A": "EN5d44fTNM0M4kmMMVrsH0HwMLRLyb6SoJEV0ogkLdXx"
```

Notice that the value of the `A` field is the AGID.

In list form, i.e., compacted to a list with only the AGID and the block SAIDs, the Aggregate section is as follows:

```
"A":
[
        "EN5d44fTNM0M4kmMMVrsH0HwMLRLyb6SoJEV0ogkLdXx",
        "EI2lwi1ZKrs-bDwgEreOhEh-W2O5xrOm5T-QCyMuX5V4",
        "EC-vU19URXX8ztfWdp_j2HHr1lJsqtGa1YHtZrg6-GMR",
        "EKYLUIpDXNT0ujSdoNOT5pLp0okOKW3mAbg-M7K5OO_C"
]
```

When the serialization kind is JSON, the raw value used to compute the AGID with a dummied value for the AGID is as follows:

```
["############################################","EI2lwi1ZKrs-
bDwgEreOhEh-W2O5xrOm5T-QCyMuX5V4","EC-
```

```
vU19URXX8ztfWdp_j2HHr1lJsqtGa1YHtZrg6-
GMR","EKYLUIpDXNT0ujSdoNOT5pLp0okOKW3mAbg-M7K5OO_C"]
```

The Blake3-256 digest of this value encoded in CESR qb64 is the AGID above.

Lets suppose that selective disclosure of the Issuee block and the Score block is desired but without disclosing the Name block. This disclosure would be as follows:

```
"A":
[
        "EN5d44fTNM0M4kmMMVrsH0HwMLRLyb6SoJEV0ogkLdXx",
        {
            "d": "EI2lwi1ZKrs-bDwgEreOhEh-W2O5xrOm5T-QCyMuX5V4",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcw",
            "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf"
        },
        {
            "d": "EC-vU19URXX8ztfWdp_j2HHr1lJsqtGa1YHtZrg6-GMR",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcx",
            "score": 96
        },
        "EKYLUIpDXNT0ujSdoNOT5pLp0okOKW3mAbg-M7K5OO_C"
]
```

From this disclosure, the block SAIDS can be computed, and then the compact list form with only AGID and block SAIDs can be constructed, and then the AGID can be computed to verify the disclosure.

CESR Native Serialization

If instead, CESR native serialization is used, the fully uncompacted Aggregate section is as follows:

```
"A":
[
        "EEL7OTDzXjYoaDE8g8064thOpKdxsJWaG8DhRyOB58qW",
        {
            "d": "EPss9hsx7P5iYjWXNYJM5NiEu5EtPQHdGZ5K-qXK2p5E",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcw",
            "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf"
        },
        {
            "d": "EGoIcPap1swfLGRQzTaxf38HsLFuHehBCY5kUSDK8XGs",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcx",
            "score": 96
        },
        {
            "d": "ED50KTrvT5n20JFTsyZFvBJfH-bOAVP9xHFhtbI5nCN6",
```

```
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcy",
            "name": "Zoe Doe"
        }
]
```

The AGID is `EEL7OTDzXjYoaDE8g8064thOpKdxsJWaG8DhRyOB58qW`

In list form, i.e., compacted to a list with only the AGID and the block SAIDs, the Aggregate section is as follows:

```
"A":
[
        "EEL7OTDzXjYoaDE8g8064thOpKdxsJWaG8DhRyOB58qW",
        "EPss9hsx7P5iYjWXNYJM5NiEu5EtPQHdGZ5K-qXK2p5E",
        "EGoIcPap1swfLGRQzTaxf38HsLFuHehBCY5kUSDK8XGs",
        "ED50KTrvT5n20JFTsyZFvBJfH-bOAVP9xHFhtbI5nCN6"
]
```

When the serialization kind is CESR, the raw value used to compute the AGID with a dummied value for the AGID is as follows:

```
-JAs############################################EPss9hsx7P5iYjWXN
YJM5NiEu5EtPQHdGZ5K-qXK2p5EEGoIcPap1swfLGRQzTaxf38HsLFuHehBCY5kUS
DK8XGsED50KTrvT5n20JFTsyZFvBJfH-bOAVP9xHFhtbI5nCN6
```

The Blake3-256 digest of this value encoded in CESR qb64 is the AGID above.

Lets suppose that selective disclosure of the Issuee block and the Score block is desired but without disclosing the Name block. This disclosure would be as follows:

```
"A":
[
        "EEL7OTDzXjYoaDE8g8064thOpKdxsJWaG8DhRyOB58qW",
        {
            "d": "EPss9hsx7P5iYjWXNYJM5NiEu5EtPQHdGZ5K-qXK2p5E",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcw",
            "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf"
        },
        {
            "d": "EGoIcPap1swfLGRQzTaxf38HsLFuHehBCY5kUSDK8XGs",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcx",
            "score": 96
        },
        "ED50KTrvT5n20JFTsyZFvBJfH-bOAVP9xHFhtbI5nCN6"
    ]
```

From this disclosure, the block SAIDS can be computed, and then the compact list form with only AGID and block SAIDs can be constructed, and then the AGID can be computed to verify the disclosure.

Edge Section

The Edge Section, `e` field MAY appear as a top-level block within the ACDC and denotes the start of subsequent Edge-groups. The Edge Section has several reserved fields that MAY appear as top-level fields within its block.

The Edge Section itself MUST have a “oneOf” composition with only its SAID so that the Edge Section MUST be expressable in the most compact form as merely its SAID.

Block Types

There are two types of field maps or blocks that MAY appear as values of fields within the Edge Section, `e` field either at the top level of the Edge block itself or nested. These are Edges or Edge-groups. Edges MAY only appear as locally unique labeled (using non-reserved labels) blocks nested within an Edge-group. There are two exceptions for Edges, compact and simple compact form. In these two forms the Edge field value is not a block but a string. These exceptions are defined below.

Nested Edge-groups, when present, with one exception, MUST appear as locally unique labeled blocks nested within another Edge-group. The block type, Edge or Edge-group, is indicated by its corresponding labeled subschema. The exception is the top-level Edge-group, which is the Edge Section and is indicated by the Edge Section subschema. When present, it MUST appear at the top-level of the ACDC. The presence of an Edge Section is OPTIONAL, An Edge is indicated by the REQUIRED presence of a node, `n` field in the non-compact variant of its subschema. An Edge MUST contain a node, `n` field. An Edge-group MUST NOT have a node, `n` field.

ACDCs as secure graph fragments of a globally distributed property graph (PG)

A set of ACDCs as nodes connected by edges forms a labeled property graph (LPG) or property graph (PG) for short [56] [57] [58]. Property graphs have properties not only in the nodes but also in the edges. The properties of each node (ACDC) are provided essentially by its Attribute Section. The properties of each edge are provided by the combination of Edge blocks and Edge-group blocks. In this regard, the set of nested Edge-groups within a given top-level Edge Section, i.e., the `e` field block of a given ACDC, constitute a sub-graph of a super-graph of ACDCs where the nodes of the super-graph are ACDCs. The nodes of the sub-graph are the Edge-groups and Edges, not whole ACDCs. One of the attractive features of property graphs (PGs) is their support for different edge and node types, which enables nested sub-graphs that support the rich expres-

sion of complex logical or aggregative operations on groups of Edges and/or Edge-groups (as subnodes) within the top-level Edge Section, `e`, field of an ACDC (as supernode).

To clarify, the Edge Section of an ACDC forms a sub-graph where each Edge block is a leaf of a branch in that sub-graph that terminates at the value of its node, `n`, field. Its node, `n` field, points to some other ACDC that is itself a sub-graph. The head of each sub-graph is the top-level Edge-group, i.e. the Edge Section. Thus, an Edge block is a leaf with respect to the sub-graph formed by any nested Edge-group blocks in which the Edge appears.

Abstractly, an ACDC with one or more edges may be viewed as a fragment of a larger distributed property graph where each ACDC Edge Section is a sub-graph. Each node in this larger graph is universally uniquely identified by the SAID of its ACDC. Recall that a SAID is a cryptographic digest. The local labels on each Edge block or Edge-group block are not universally uniquely resolvable, so they are inappropriate as a verifiable hook for resolving the Edges in a globally distributed property graph. This requires resolving both nodes and Edges. To enable both the node and its connecting edge to be globally uniquely resolvable, each Edge's block MUST also have a SAID, `d`, field. Recall that a SAID is a cryptographic digest. Therefore, it will universally and uniquely identify an Edge with a given set of properties [48], including the node it points to.

Thus, a given ACDC with its Edges may be universally uniquely resolvable as a graph fragment of a larger graph. Moreover, because each ACDC with edges, i.e., a graph fragment, may be independently sourced and securely attributed, a distributed property graph can be securely assembled and verified fragment by fragment. To summarize, these features enable ACDCs to be used as securely attributed fragments of a globally distributed property graph (PG). This further enables such a property graph so assembled to serve as a global verifiable knowledge graph that crosses trust domains [56] [57] [58].

The universal uniqueness of the ACDC SAIDs (nodes) and their Edge SAIDs enable the simplified discovery of ACDCs via service endpoints. 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 or by the SAID of the Edge that points to the SAID of that ACDC [4]. 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 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 referenced ACDCs. That Issuee will then have everything it needs to make a successful disclosure to some other Disclosee. This is the essence of Percolated Discovery [4].

Edge-group

The reserved field labels for an Edge-group are detailed in the table below.

Label	Title	Description
`d`	Digest (SAID)	Optional, self-referential fully qualified cryptographic digest of enclosing Edge-group block.
`u`	UUID	Optional random Universally Unique Identifier as fully qualified high entropy pseudo-random string, a salty nonce.
`o`	Operator	Optional as an m-ary operator on the Edges in the Edge-group. Enables expression of the edge logic on edge subgraph.
`w`	Weight	Optional property for nested Edges or Edge-groups for weighted average `WAVG` operator. Enables expression weighted average on Edge or Edge-group sub-graph.

When present, the order of appearance of these fields is as follows: `[d, u, o, w]'.

An Edge-group MUST NOT have a node, `n`, field.

SAID, `d` field

The SAID, `d` field is optional but when it appears it MUST appear as the first field in the Edge-group block. The value of this field MUST be the SAID of its enclosing block.

UUID, `u` field

The UUID, `u` field is optional, but when it appears, it MUST appear as the second field in the Edge-group block following the SAID, `d`, field. The value of this field MUST be a cryptographic strength salty-nonce with approximately 128 bits of entropy. When present, the UUID, `u` field means that the block's SAID, `d` field value provides a secure cryptographic digest of the contents of the block [@Hash]. An adversary, when given both the block's subschema and its SAID, cannot discover the remaining contents of the block in a computationally feasible manner, such as a rainbow table attack [@RB][@DRB]. Therefore, the block's UUID, `u` field securely blinds the contents of the block via its SAID, `d` field notwithstanding knowledge of both the block's subschema and SAID. Moreover, a cryptographic commitment to that block's SAID, `d` field does not provide a fixed point of correlation to the block's field values themselves unless and until there has been a disclosure of those field values.

Operator, `o` field

The Operator, `o` field MUST appear immediately following the SAID, `d` field, and UUID, `u` field (when present) in the Edge-group block. The Operator field in an Edge-group block is an aggregating (m-ary) operator on all the nested labeled Edges or Edge-groups that appear in its block. This differs from the Operator, `o` field in an Edge block (see below).

The m-ary operators are defined in the table below:

M-ary Operator	Description	Default
`AND`	Logical AND of the validity of the Edge-group members. Edge-group is valid only if all members are valid.	Yes
`OR`	Logical OR of the validity of the Edge-group members. Edge-group is valid if one of the members is valid.	No
`NAND`	Logical NAND of the validity of the Edge-group members. Edge-group is valid only if not all members are valid.	No
`NOR`	Logical NOR of the validity of the Edge-group members. Edge-group is valid only if all members are invalid.	No
`AVG`	Arithmetic average of a given Edge-group member property. Averaged property is defined by the schema or EGF.	No
`WAVG`	Weighted arithmetic average of a given Edge-group member property. Weight is given by the `w` field. Averaged property is defined by the Schema or EGF.	No

When the Operator, `o`, field is missing in an Edge-group block, the default value for the Operator, `o`, field MUST be the `AND` Operator.

The actual logic for interpreting the validity of a set of chained or treed ACDCs is EGF-dependent. But as a default, at least at the time of issuance, a set of chained (treed) ACDCs can be interpreted as a provenance chain (tree) with the default logic explained below. ACDCs in a provenance chain or branch that have expiration dates or are dynamically revocable add a timeliness property to the validation logic in the event that the chain was unbroken at issuance but becomes broken later. The timeliness logic is EGF-dependent.

When the top-level Edge-group, the Edge Section includes only one Edge, the logic for evaluating the validity of the enclosing ACDC (near node) with respect to the validity of the connected far node ACDC pointed to by that edge MAY be considered a link in a provenance chain. When any node in a provenance chain is invalid, an Edge pointing to that node MAY also be invalid. If a node has an invalid Edge, then the node MAY also be invalid. To elaborate, a chain of nodes with Edges has a head and a tail. The Edges are directed from the head to the tail. Typically, in a given EGF (ecosystem governance framework), all links from the node at the head at one end of a chain to the tail at the other end MUST be valid in order for the node (head) to be valid. If any links between the head and the tail are broken (invalid), then the head is itself MUST be invalid. The unary operators (defined below) for edges enable modulation of the validation logic of a given Edge/node in a chain of ACDCs.

When the top-level Edge-group, the Edge Section includes more than one Edge directly or indirectly (as nested Edge(s) in nested Edge-group), then the logic for evaluating the validity of the enclosing ACDC (near node) with respect to the validity of all the connected far node ACDCs pointed to by those Edges MAY be a provenance tree. Instead of a single chain from the head to a single tail, there is a tree with the truck at the head and branches as chains of directed Edges that each point to a branch tail (leaf) node. To clarify, each branch is a chain from head to branch tail. Typically, in a given EGF (ecosystem governance framework), all branches from the head MUST be valid for the head node to be valid. The m-ary Operators (defined above), in combination with the unary Operators (defined below), allow modulation of the validation aggregation logic of groups of Edges/nodes at each branch in a tree of ACDCs.

Weight, `w` field

The Weight, `w` field, is OPTIONAL but when present, it MUST appear immediately following all of the SAID, `d` field, UUID, `u` field (when present), and Operator, `o` field (when present) in the Edge-group block. The Weight field enables weighted averages with Operators that perform some type of weighted average, such as the `WAVG` Operator. The top-level Edge-group MUST NOT have a weight, `w` field, because it is not a member of another Edge-group.

A Edge-group with a weight MAY provide an aggregate of weighted directed Edges. Weighted directed Edges MAY represent degrees of confidence or likelihood. PGs with weighted, directed Edges are commonly used for machine learning or reasoning under

uncertainty. The Weight, `w` field provides a reserved label that MAY be the primary weight on an Edge group to be applied by the Operator of its enclosing Edge-group. To elaborate, many aggregating Operators used for automated reasoning, such as the weighted average, `WAVG`, Operator, or ranking aggregation Operators, depend on each input's weight. To simplify the semantics for such Operators, the Weight, `w`, field MUST be the reserved field label for weighting. Other fields with other labels (labeled Edge-group properties) MAY provide other types of weights, but having a default label, namely `w`, simplifies the default definitions of weighted Operators.

Labeled nested edge and edge-group fields

Edge-groups and Edges nested within a given Edge-group MAY appear as labeled fields whose labels are not any of the reserved field labels for either Edge-groups or Edges, namely, `[d, u, n, s, o, w]'. Such labeled nested Edge or Edge-group fields MUST appear after all of any fields with a reserved field label.

Each nested Edge or Edge-group block within an Edge-group including the top-level Edge Section Edge-group MUST be labeled with a locally unique non-reserved field label that indicates the type of the nested block. To clarify, each nested block in every Edge-group MUST have its own field with its own local (to the ACDC) label. The field's value MAY be either an Edge sub-block or when in compact form, a string. The compact forms are defined below.

Note that each nested block MUST NOT include a type field. The type of each block is provided by that associated subschema in the Edge Section's subschema with a matching label. This is in accordance with the design principle of ACDCs succinctly expressed as "type-is-schema." This approach varies somewhat from other types of property graphs, which often do not have a Schema [56] [57] [58]. Because ACDCs MUST have a Schema, they SHOULD leverage it to provide property graph edge types with a cleaner separation of concerns. Notwithstanding this separation, an Edge block MAY include a constraint on the type of the ACDC to which that Edge points by including the SAID of the schema of the pointed-to ACDC as a property of that edge (see below).

A main distinguishing feature of a property graph (PG) including ACDCs is that both nodes and Edges MAY have a set of properties [56] [57] [58]. These MAY include modifiers that influence how the connected node is to be combined or place a constraint on the allowed type(s) of connected nodes. Each Edge's block provides the Edge's properties vis-a-vis a property graph. Additional properties MAY be inferred from the properties of an enclosing Edge-group's block. Each labeled Edge and Edge-group type MUST be defined by the subschema designated by its label.

Edge

An Edge MUST be represented as a block (field map) with two exceptions. These are: compact Edge form and simple compact Edge form. These are defined below. The Edge subschema is used to differentiate these two compact forms from the block form.

The reserved field labels within an Edge block are defined in the table below.

Label	Title	Description
d	Digest (SAID)	Optional, self-referential fully qualified cryptographic digest of enclosing Edge map.
u	UUID	Optional random Universally Unique Identifier as fully qualified high entropy pseudo-random string, a salty nonce.
n	Node	Required SAID of the far node ACDC as the terminating point of a directed edge that connects the Edge's encapsulating near node ACDC to the specified far node ACDC as a fragment of a distributed property graph (PG).
s	Schema	Optional SAID of the JSON Schema block of the far node ACDC.
o	Operator	Optional as either a unary operator on the Edge itself or an m-ary operator on the Edge-group in Edge section. Enables expression of the Edge logic on Edge subgraph.

Label	Title	Description
`w`	Weight	Optional edge weight property that enables default property for directed weighted edges and operators that use weights.

An Edge block MUST have a node, `n`, field. This differentiates an Edge block from an Edge-group block. The SAID, `d`, UUID, `u`, schema, `s`, operator, `o`, and weight, `w` fields are OPTIONAL. To clarify, each Edge block MUST have a node, `n`, field and MAY have any combination of SAID, `d`, UUID, `u`, schema, `s`, operator, `o`, or weight, `w` fields. When present the order of appearance of these fields MUST be as follows: `[d, u, n, s, o, w]'.

SAID, `d` field

The SAID, `d` field is optional but, when present, MUST appear as the first field in the Edge block. The value of this field MUST be the SAID of its enclosing block.

Compact edge

Given that an individual edge's property block includes a SAID, `d`, field, a compact representation of the edge's property block is provided by replacing it with its SAID. This is called a compact edge. The schema for that edge's label MUST indicate that the edge value is the edge block SAID by using a `oneOf` composition of the compact form and the expanded form. This is useful for compacting complex edges with many properties and then expanding them later. When the edge block also includes a UUID, `u` field, then compacting also hides the edge properties for later disclosure. A compact edge without a UUID, `u` field, is defined to be a public compact edge. A compact edge with a UUID, `u` field, is defined to be a private compact edge.

UUID, `u` field

The UUID, `u` field is optional, but when it appears, it MUST appear as the second field in the Edge block following the SAID, `d`, field. The value of this field MUST be a cryptographic strength salty-nonce with approximately 128 bits of entropy (nominally). When present, the UUID, `u` field means that the block's SAID, `d` field value provides a secure cryptographic digest of the contents of the block [48]. An adversary, when given both the block's subschema and its SAID, cannot discover the remaining contents of the block in a computationally feasible manner, such as a rainbow table attack [30] [31]. Therefore, the block's UUID, `u` field securely blinds the contents of the block via its SAID, `d` field notwithstanding knowledge of both the block's subschema and SAID. Moreover, a cryp-

tographic commitment to that block's SAID, `d` field does not provide a fixed point of correlation to the block's field values themselves unless and until there has been a disclosure of those field values.

The absence of the UUID, `u` field in an Edge block makes that edge a Public Edge. The presence of the UUID, `u` field in an Edge block makes that Edge a Private Edge. A Private Edge in compact form, i.e., a Compact Private Edge, enables a presenter of that ACDC to make a verifiable commitment to the ACDC attached to the Edge without disclosing any details of that ACDC, including the ACDC's SAID. Private ACDCs (nodes) and Private Edges MAY be combined to better protect the privacy of the information in a distributed property graph.

Node, `n` field

When an Edge block does not include a SAID, `d` field, then the node, `n` field MUST appear as the first field in the block.

The value of the required node, `n` field, is the SAID of the ACDC to which the Edge connects, i.e., the node, `n` field indicates, designates, references, links, or "points to" another ACDC called the far node. To clarify, the Edge is directed from the near node, i.e., the ACDC in which the Edge block resides, to the far node, which is the ACDC indicated by the value of node, `n` field of that block. The edges and nodes form a directed acyclic graph (DAG).

In order for a given Edge to be valid, at the very least, a Validator MUST confirm that the SAID of the provided far node ACDC matches the node, `n` field value given in the near node ACDC Edge block and MUST confirm that the provided far node ACDC satisfies its own schema. When the (near node) Edge block's schema, `s` field is present (see below), then the far node MUST also validate against the Schema indicated by the near node Edge block's Schema, `s` field value.

Schema, `s` field

When present, the Schema `s` field MUST appear immediately following the node `n` field in the Edge sub-block. The schema, `s` field provides an additional constraint on the far node ACDC. The far node ACDC MUST also validate against an Edge block's schema, `s` field when present. To clarify, the Validator, after validating that the provided far node ACDC indicated by the node, `n` field satisfies its (the far ACDC's) own Schema, MUST also confirm that far node ACDC passes Schema validation with respect to the Edge's `s` field value. This validation is simplified whenever the SAID of the far node ACDC Schema matches the SAID of the Edge's Schema, `s` field because only one Schema validation has to be performed. However, when the Schema SAIDs differ, two Schema validation runs MUST be performed.

The Edge Schema, `s` field enables a given Issuer of an ACDC to force forward compatibility constraints (i.e. minor schema version changes) on old ACDCs to which a new issuance is chained without having to add each old minor version to a list of `oneOf` compositions in the new Edge's Schema, `s` field value. As long as the new Edge's schema validates against the old minor version ACDC, no additional `oneof` is required in the new Edge's schema. This enables the old schema to be self-deprecated as it will automatically fall out of use once all the ACDCs issued under it have expired. Whereas, adding a `oneof` in the new Edge's schema effectively forces continued valid use of the old schema. This latter case is not self-deprecating.

Major version changes, in contrast, are, by definition, backward-breaking, so either old major version ACDCs must be revoked and reissued or the new Edge's schema value must include a `oneOf` composition that includes the old major versions. Refer to the discussion under the Schema section for information on Schema versioning.

Operator, `o` field

The Operator, `o` field MUST appear immediately following the SAID, `d` field, UUID, `u` field, node, `n` field, or schema, `s` field (when present) in the Edge block. The Operator, `o`, field value in an Edge block is a unary Operator on the Edge itself. When more than one unary Operator is applied to a given Edge, then the value of the Operator, `o`, field is a list of those unary Operators. When multiple unary Operators appear in the list, and there is a conflict between Operators, the latest Operator among the conflicting Operators in the list takes precedence. This differs from the Operator, `o` field in an Edge-group block (see above).

The unary operators are defined in the table below:

Unary Operator	Description	Default
`I2I`	Issuer-To-Issuee, The Issuer AID of this ACDC MUST be the Issuee AID of the node this Edge points to.	Yes
`NI2I`	Not-Issuer-To-Issuee, The Issuer AID of this ACDC MAY or MAY not be the Issuee AID of the node that this Edge points to.	No
`DI2I`	Delegated-Issuer-To-Issuee, The Issuer AID of this ACDC MUST be either the Issuee AID or a delegated AID of the Issuee AID of the node this Edge points to.	No
`NOT`	Logical NOT. The validity of the node this Edge points to is inverted. If valid, then not valid. If invalid, then valid.	No

When the Operator, `o`, field is missing or empty or is present but does not include any of the `I2I`, `NI2I` or `DI2I` Operators then:

- If the node pointed to by the Edge is a targeted ACDC, i.e., has an Issuee, then the `I2I` Operator MUST be appended to the Operator, `o`, field's effective list value.
- If the node pointed to by the Edge block is an Untargeted ACDC i.e., does not have an Issuee, then the `NI2I` Operator MUST be appended to the Operator, `o`, field's effective list value.

Many ACDC chains use Targeted ACDCs (i.e., have Issuees). A chain of Issuer-To-Issuee-To-Issuer Targeted ACDCs in which each Issuee becomes the Issuer of the next ACDC in the chain can be used to provide a chain-of-authority. A common use case of a chain-of-authority is a delegation chain for authorization.

The `I2I` unary operator, when present, means that the Issuer AID of the current ACDC in which the Edge resides MUST be the Issuee AID of the node to which the Edge points. Therefore, to be valid, the ACDC node pointed to by this Edge MUST be a Targeted ACDC. This is the default value when the Operator 'o' field value is missing or empty.

The `NI2I` unary Operator, when present, removes or nullifies any requirement expressed by the `I2I` Operator described above. In other words, any REQUIREMENT that the Issuer AID of the current ACDC in which the Edge resides MUST be the Issuee AID, if any, of the node the Edge points to is relaxed (not applicable). To clarify, when operative (present), the `NI2I` Operator means that both an Untargeted ACDC or Targeted ACDC, as the node pointed to by the Edge, MAY be valid even when Untargeted or if Targeted even when the Issuer of the ACDC in which the Edge appears is not the Issuee AID, of that node to which the Edge points.

The `DI2I` unary Operator, when present, expands the class of allowed Issuer AIDs of the node the Edge resides in to include not only the Issuee AID but also any delegated AIDs of the Issuee of the node to which the Edge points. Therefore, to be valid, the ACDC node pointed to by this Edge MUST be a Targeted ACDC.

The `NOT` unary Operator, when present, inverts the validation truthiness of the node pointed to by this Edge. If this Edge's far node ACDC is invalid, then the presence of the `NOT` operator makes this Edge valid. Conversely, if this Edge's far node ACDC is valid, then the presence of the `NOT` Operator makes this Edge invalid.

Weight, `w` field.

The Weight, `w` field MUST appear immediately following the SAID, `d` field, UUID, `u` field, Node, `n` field, Schema, `s` field, or Operator, `o` field (when present) in the Edge block. The Weight field enables weighted direct Edges. The weighted directed Edges within an enclosing Edge-group MAY be aggregated when that Edge-group's Operator performs some type of weighted average.

Weighted directed Edges may represent degrees of confidence or likelihood. PGs with weighted, directed Edges are commonly used for machine learning or reasoning under uncertainty. The Weight, `w` field provides a reserved label for the primary Weight on an Edge. To elaborate, many aggregating operators used for automated reasoning, such as the weighted average, `WAVG`, Operator, or ranking aggregation Operators, depend on each input's Weight. To simplify the semantics for such Operators, the Weight, `w` field is the reserved field label for weighting. Other fields with other labels (labeled Edge properties) could provide other types of weights, but having a default label, namely `w` simplifies the default definitions of weighted Operators.

Labeled property fields

Edge property fields appear as labeled fields whose labels are not any of the reserved field labels for either Edge-groups or Edges, namely, `[d, u, n, s, o, w].' Labeled property fields MUST appear after all of any fields with a reserved field label.

Simple compact edge

When an Edge sub-block has only one field, that is, its node, `n` field, i.e., it has no other properties, 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 Edge is, therefore, public. This enables the very compact expression of simple public Edges. The Schema for that Edge's label MUST indicate in its description that the Edge value is a Far Node SAID, not the Edge block SAID.

Summary

The Edge Section syntax enables the composable and extensible but efficient expression of aggregating (m-ary) and unary Operators, both singly and in combination, as applied to nestable groups of Edges. The intelligent defaults for the Operator, `o`, field, namely, `AND` for m-ary Operators and `I2I` for unary Operators for targeted ACDCs and `NI2I` for untargeted ACDCs, means that in many use cases, the Operator, `o`, field, does not even need to be present. This syntax is sufficiently general in scope to satisfy the contemplated use cases, including those with more advanced business logic.

Examples

The Edge section examples are included in the full ACDC examples in the Annex. This is because Edges refer to other ACDCs. To fully set up the Edge examples, it is necessary to understand the complete format of an ACDC.

Rule Section

The Rule Section, `r` field appears as a top-level block within the ACDC and denotes the start of subsequent Rule-groups. The purpose of the Rules section is to provide a set of rules or conditions as a Ricardian Contract [43]. The important features of a Ricardian contract are that it is both human and machine-readable and referenceable by a cryptographic digest. A JSON-encoded document or block, such as the Rules section block, is a practical example of both a human and machine-readable document. The Rules section's top-level SAID, `d` field provides the digest. This provision supports the bow-tie model of RC.

The Rules Section includes labeled nested blocks called Rules that provide the legal language (terms , conditions, definitions etc). The labeled clauses can be structured hierarchically, where each Rule, in turn, can include nested labeled Rules. The labels on the Rules MAY follow a structured naming or numbering convention. These provisions enable the Rules section to satisfy the features of an RC.

The Rule Section itself MUST have a “oneOf” composition with only its SAID so that the Rule Section MUST be expressable in the most compact form as merely its SAID.

Block Types

There are two types of field maps or blocks that MAY appear as values of fields within the Rule Section, `r` field either at the top level of the Rule block itself or nested. These are Rules or Rule-groups. Rules MAY only appear as locally unique labeled (using non-reserved labels) blocks nested within an Rule-Group. There are two exceptional forms for Rules, compact and simple compact form. In these two forms, the labeled Rule field value is not a block but a string. These exceptions are defined below.

Nested Rule-groups MAY only appear as locally unique labeled blocks nested within another Rule-group. The block type, Rule or Rule-group, is indicated by its corresponding labeled subschema, with the exception of the top-level Rule-group, which is the Rules Section and is indicated by the Rules Section subschema. A Rule-group is indicated by the presence of one or more non-reserved labeled fields whose value represents a nested Rule or Rule-Groups.

Rule Discovery

In compact form, the discovery of either the Rules section as a whole or a given Rule or Rule-group 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 rule). The discovery of a service endpoint URL that provides database access to a copy of the Rules section or to any of its Rules or Rule-groups may be bootstrapped via an OOBI that links the service endpoint URL to the SAID of the respective block [@OOBI_ID]. Alternatively, the Issuer may provide as an attachment at issuance a copy of the referenced contract associated with the whole rule section or any Rule. In either case, after a successful issuance exchange, the Issuee 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 will then have everything subsequently needed to make a successful presentation or disclosure to a Disclosee. This is the essence of Percolated Discovery.

Rule-group

The reserved field labels for Rule-groups are detailed in the table below.

Label	Title	Description
`d`	Digest (SAID)	Optional Self-referential fully qualified cryptographic digest of enclosing block.
`u`	UUID	Optional random Universally Unique Identifier as fully qualified high entropy pseudo-random string, a salty nonce.
`l`	Legal Language	Optional legal language for the Rule-group.

When present, the order of appearance of these fields is as follows: `[d, u, l]`.

A Rule-group MAY have a Legal, `l`, field and MAY have a SAID, `d` field. When the Rule-group has a SAID, `d` field it MAY also have a UUID, `u` field. A Rule-group MAY have one or more other labeled fields whose values represent nested Rules or Rule-groups. In this sense, a Rule-group is an intermediate node in a sub-graph of Rule-groups and Rules.

SAID, `d` field

The SAID, `d` field is optional but when it appears it MUST appear as the first field in the Rule-group block. The value of this field MUST be the SAID of its enclosing block. To elaborate, when the Rule-group is the top-level Rule Section its SAID is the same SAID used as the compacted value of the Rule Section, `r` field that appears at the top level of the ACDC. When not the top-level Rule-group, a given nested Rule-group's SAID, `d` field enables a verifiable globally unique reference to that nested Rule-group, not merely the whole contract as given by the Rule section's top-level SAID, `d`, field.

UUID, `u` field

The UUID, `u` field is optional, but when it appears, it MUST appear as the second field in the Rule-group block following the SAID, `d`, field. The value of this field MUST be a cryptographic strength salty-nonce with approximately 128 bits of entropy (nominally). When present, the UUID, `u` field means that the block's SAID, `d` field value provides a secure cryptographic digest of the contents of the block [@Hash]. An adversary, when given both the block's subschema and its SAID, cannot discover the remaining contents of the block in a computationally feasible manner, such as a rainbow table attack [@RB] [@DRB]. Therefore, the block's UUID, `u` field securely blinds the contents of the block

via its SAID, `d` field notwithstanding knowledge of both the block's subschema and SAID. Moreover, a cryptographic commitment to that block's SAID, `d` field does not provide a fixed point of correlation to the block's field values themselves unless and until there has been a disclosure of those field values.

Labeled nested rule and rule-group fields

Rules and Rule-group nested within a Rule-group appear as labeled fields whose labels are not any of the reserved field labels for a Rule-group, namely, `[d, u, l]`. Labeled nested Rule or Rule-group fields MUST appear after all of any fields with a reserved field label.

To elaborate, each nested Rule or Rule-group block MUST be labeled with a locally unique non-reserved field label that indicates the type of the nested block. To clarify, each nested block gets its own field with its own local (to the ACDC Rule Section) label. The field's value MAY be either the Rule or Rule-group block or, in compact form, a string. The compact forms are defined below.

Note that each nested block MUST NOT include a type field. The type of each block is provided by that associated nested subschema with a matching label. This is in accordance with the design principle of ACDCs succinctly expressed as "type-is-schema." This approach varies somewhat from other types of property graphs, which often do not have a Schema [56][57][58]. Because ACDCs have a Schema, they leverage it to provide property graph types with a cleaner separation of concerns.

A main distinguishing feature of a property graph (PG) including ACDCs is that both nodes and Edges MAY have a set of properties [56][57][58]. Each Rule-group's block MAY provide its additional properties vis-a-vis a property graph as labeled fields. Additional properties MAY be inferred from the properties of an enclosing Rule-group block. Each labeled Rule type MUST be defined by the subschema designated by its label.

Rule

The reserved field labels for a Rule block are detailed in the table below.

Label	Title	Description
`d`	Digest (SAID)	Optional self-referential fully qualified cryptographic digest of enclosing block.
`u`	UUID	Optional random Universally Unique Identifier as fully qualified high entropy pseudo-random string, a salty nonce.
`l`	Legal Language	The actual legal language for the clause.

When present, the order of appearance of these fields is as follows: `[d, u, l]`.

A Rule MUST have a Legal, `l`, field. And MAY have a SAID, `d` field. When the Rule has a SAID, `d` field, it MAY also have a UUID, `u` field. A Rule MUST NOT have any other fields. In this sense, a Rule is a terminal node in a sub-graph of Rule-groups and Rules.

SAID, `d` field

The SAID, `d` field is optional, but when it appears, it MUST appear as the first field in the Clause block. The value of this field MUST be the SAID of its enclosing block. A Rule's SAID enables a verifiable globally unique reference to that rule, not merely the whole contract as given by the Rule section's top-level SAID, `d`, field.

Compact Rule

Given that an individual Rule block includes a SAID, `d` field, a compact representation of the Rule's block is provided by replacing it with its SAID. This is called a compact Rule. The Schema for that clause's label MUST indicate that the clause field value is the clause block SAID by using a `oneOf` composition of the compact form and the expanded form. This may be useful for compacting lengthy clauses and then expanding them later. When the clause block also includes a UUID, `u` field, then compacting also hides the clause's legal language for later disclosure. A compact clause without a UUID, `u` field is defined to be a public compact clause. A compact clause with a UUID, `u` field is defined to be a private compact clause.

UUID, `u` field

The UUID, `u` field is optional, but when it appears, it MUST appear as the second field in the Rule Section block following the SAID, `d` field. The value of this field MUST be a cryptographic strength salty-nonce with approximately 128 bits of entropy. When present, the UUID, `u` field means that the block's SAID, `d` field value provides a secure cryptographic digest of the contents of the block [48]. An adversary, when given both the block's subschema and its SAID, cannot discover the remaining contents of the block in a computationally feasible manner, such as a rainbow table attack [30] [31]. Therefore, the block's UUID, `u` field securely blinds the contents of the block via its SAID, `d` field notwithstanding knowledge of both the block's subschema and SAID. Moreover, a cryptographic commitment to that block's SAID, `d` field does not provide a fixed point of correlation to the block's field values themselves unless and until there has been a disclosure of those field values.

When any Rule or Rule-group block, as opposed to the Rule Section as a whole, has both its own SAID, `d`, field and UUID, `u,` field then that block may be used in a confidential manner via partial disclosure. To clarify, a Rule or Rule-group block with both a SAID, `d`, and UUID, `u` fields, where that UUID has sufficiently high entropy, protects the compact form of that block from discovery via a rainbow table attack merely from its SAID and subschema [[30]] [[31]]. Therefore, such a Rule or Rule-group may be kept hidden until later disclosure. These are referred to as private or confidential Rules or Rule Groups.

When there is no benefit to a nested private (partially disclosable) rules or rule groups then the rule's or rule group's UUID field is not needed.

Legal, `l` Field

The legal language, `l`, field in each clause block provides the associated legal language as a string.

Simple Compact Rule

When a Rule block has only one field, that is, its legal, `l` field, i.e., it has no other properties, then the rule block MAY use an alternate simplified, compact form where the block is represented as a single rule field and that labeled rule field value is what would have been the value of the block's legal, `l` field when not in simple compact form. The Rule block (rule) is, therefore, public. This enables the very compact expression of simple public Rules. The Schema for that Rule's label MUST indicate in its description that the Rule value is Legal Language, not a SAID.

Rule Examples

Consider the following sub-schema for a partially disclosable Rule section:

```
{
    "description": "Rule Section",
    "oneOf":
```

```
    [
        {
            "description": "Rule Section SAID",
            "type": "string"
        },
        {
            "description": "Rule Section Detail",
            "type": "object",
            "required":
            [
                "d",
                "u",
                "disclaimers",
                "permittedUse"
            ],
            "properties":
            {
                "d":
                {
                    "description": "Rule Section SAID",
                    "type": "string"
                },
                "u":
                {
                    "description": "Rule Section UUID",
                    "type": "string"
                },
                "disclaimers":
                {
                    "description": "Rule Group",
                    "oneOf":
                    [
                        {
                            "description": "Rule Group SAID",
                            "type": "string"
                        },
                        {
                            "description": "Rule Group Detail",
                            "type": "object",
                            "required":
                            [
                                "d",
                                "u",
                                "l",
                                "warrantyDisclaimer",
                                "liabilityDisclaimer"
                            ],
                            "properties":
                            {
```

```
                                "warrantyDisclaimer":
                                {
                                    "oneOf":
                                    [
                                        {
                                            "description": "Rule
SAID",
                                            "type": "string"
                                        },
                                        {
                                            "description": "Rule
Detail",
                                            "type": "object",
                                            "required":
                                            [
                                                "d",
                                                "u",
                                                "l"
                                            ],
                                            "properties":
                                            {
                                                "d":
                                                {

"description": "Rule SAID",
                                                    "type":
"string"

                                                },
                                                "u":
                                                {

"description": "Rule UUID",
                                                    "type":
"string"

                                                },
                                                "l":
                                                {

"description": "Legal Language",
                                                    "type":
"string"

                                                }
                                            },

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

```
                                {
                                    "oneOf":
                                    [
                                        {
                                            "description": "Rule
SAID",
                                            "type": "string"
                                        },
                                        {
                                            "description": "Rule
Detail",
                                            "type": "object",
                                            "required":
                                            [
                                                "d",
                                                "u",
                                                "l"
                                            ],
                                            "properties":
                                            {
                                                "d":
                                                {

"description": "Rule SAID",
                                                    "type":
"string"
                                                },
                                                "u":
                                                {

"description": "Rule UUID",
                                                    "type":
"string"
                                                },
                                                "l":
                                                {

"description": "Legal Language",
                                                    "type":
"string"
                                                }
                                            },

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

```
                    ],
                },
                "permittedUse":
                {
                    "oneOf":
                    [
                        {
                            "description": "Rule SAID",
                            "type": "string"
                        },
                        {
                            "description": "Rule Detail",
                            "type": "object",
                            "required":
                            [
                                "d",
                                "u",
                                "l"
                            ],
                            "properties":
                            {
                                "d":
                                {
                                    "description": "Clause SAID",
                                    "type": "string"
                                },
                                "u":
                                {
                                    "description": "Clause UUID",
                                    "type": "string"
                                },
                                "l":
                                {
                                    "description": "Legal
Language",
                                    "type": "string"
                                }
                            },
                            "additionalProperties": false
                        }
                    ]
                }
            },
            "additionalProperties": false
        }
    ]
}
```

Also consider the following compatible nested partially disclosable rule section in fully disclosed uncompact form, as follows:

```
"r":
{
    "d": "EL7oXtsH1t7YqOOCS0fMhWfUKx1fHwiQ2u47fVba4lAA",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXcw",
    "disclaimers":
    {
        "d": "EIRP8ZLuMNb1I_Uk1GgnD3qZ_MAh6GaXV1JmzFKLebb3",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXcx",
        "l": "The person or legal entity identified by this ACD
C's Issuer AID (Issuer) makes the following disclaimers:",
        "warrantyDisclaimer":
        {
            "d": "EA84ClmyIMrSl5XaAWENAxTVZH25_YZGmu0WQm_VBBeV",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcy",
            "l": "Issuer provides this ACDC on an AS IS basis."
        },
        "liabilityDisclaimer":
        {
            "d": "ECENp0nXYDm_bLgr7TlJ8ns8I1QI2qzyqxoXnYG8B-ac",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcz",
            "l": "The Issuer SHALL NOT be liable for ANY damages
arising as a result of this credential."
        }
    },
    "permittedUse":
    {
        "d": "EIn94r7ax0PmalGUddjP3ElnU2Lzz92UFE1uIinoVeVs",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXc0",
        "l": "The Issuee (controller of the Issuee AID) MAY only
use this ACDC for non-commercial purposes."
    }
}
```

This rule section has a top-level rule group, that contains one nested rule group labeled “disclaimers” with two rules and one additional “permittedUse” rule. The SAID of this rule section, computed using the most compact SAID algorithm, is `EL7oXtsH1t7YqOOCS0fMhWfUKx1fHwiQ2u47fVba4lAA`. Notice that each rule group and each rule has its own SAID computed using the most compact SAID algorithm.

Consider a more compact, partially disclosed version of the rule section as follows:

```
"r":
{
    "d": "EL7oXtsH1t7YqOOCS0fMhWfUKx1fHwiQ2u47fVba4lAA",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXcw",
```

```
    "disclaimers": "EIRP8ZLuMNb1I_Uk1GgnD3qZ_MAh6GaXV1JmzFKLebb
3",
    "permittedUse": "EIn94r7ax0PmalGUddjP3ElnU2Lzz92UFE1uIinoVeV
s"
}
```

In this form, the “disclaimers” rule group is hidden but is verifiably referenced by its SAID. Likewise, the “permittedUse” rule is hidden but is verifiably referenced by its SAID.

The most compact form of the rule section is simply a field whose value is its SAID shown as follows:

```
"r": "EL7oXtsH1t7YqOOCS0fMhWfUKx1fHwiQ2u47fVba4lAA"
```

Consider a different variant of this Rule Section with the same rules, but uses the simple compact form. Its fully disclosed version is quite compact but does not enable nested graduated disclosure.

The schema for the non-partially disclosable version is as follows:

```
{
  "description": "Rule Section",
  "oneOf":
  [
    {
      "description": "Rule Section SAID",
      "type": "string"
    },
    {
      "description": "Rule Section Detail",
      "type": "object",
      "required":
      [
        "d",
        "disclaimers",
        "permittedUse"
      ],
      "properties":
      {
        "d":
        {
          "description": "Rule Section SAID",
          "type": "string"
        },
        "disclaimers":
        {
          "description": "Rule Group",
          "type": "object",
```

```
          "required":
          [
            "l",
            "warrantyDisclaimer",
            "liabilityDisclaimer"
          ],
          "properties":
          {
            "l":
            {
                "description": "Legal Language",
                "type": "string"
            },
            "warrantyDisclaimer":
            {
                "description": "Rule in Simple Compact Form",
                "type": "string"
            },
            "liabilityDisclaimer":
            {
                "description": "Rule in Simple Compact Form",
                "type": "string"
            }
          },
          "additionalProperties": false
        },
        "permittedUse":
        {
          "description": "Rule in Simple Compact Form",
          "type": "string"
        },
      },
      "additionalProperties": false
    }
  ]
}
```

Consider the following compatible rule section in fully disclosed form, as follows:

```
"r":
{
    "d": "EF9f-pCPJcgQclUu1OzzAfgyURW7iLPF2nwhuKYHFBlV",
    "disclaimers":
    {
        "l": "The person or legal entity identified by this ACD
C's Issuer AID (Issuer) makes the following disclaimers:",
        "warrantyDisclaimer": "Issuer provides this ACDC on an AS
IS basis.",
        "liabilityDisclaimer": "The Issuer SHALL NOT be liable fo
```

```
r ANY damages arising as a result of this credential."
    },
    "permittedUse": "The Issuee (controller of the Issuee AID) MA
Y only use this ACDC for non-commercial purposes."
}
```

The most compact form of the rule section is simply a field whose value is its SAID shown as follows:

```
"r": "EF9f-pCPJcgQclUu1OzzAfgyURW7iLPF2nwhuKYHFBlV"
```

Disclosure Mechanisms and Exploitation Protection

An important design goal of ACDCs is to support the sharing of provably authentic data while also protecting against the exploitation of that data. Often, the term privacy protection is used to describe similar properties. However, 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 unpermissioned exploitation of data. In this light, a given privacy protection mechanism may be employed to help protect against data exploitation, but only when it serves a more general purpose and not as an end in and of itself.

Binding to Key State at Time of ACDC State Change

To protect against later forgery in the event of a future compromise of the Issuer's signing keys, the Issuer must anchor an *issuance* proof digest seal to the ACDC in its KEL either directly or indirectly. This seal binds the signing Key State to the ACDC's *issuance*.

There are two cases for anchoring: direct and indirect.

With the direct case, the issuance proof digest seal is anchored in the KEL of the Issuer. No issuance/revocation Registry is used. This means the ACDC has only one state, that is, *issued* or, more specifically, *anchored*. The issuance proof seal digest is the SAID of the ACDC itself, or if bulk issued, the aggregate of the bulk issuance.

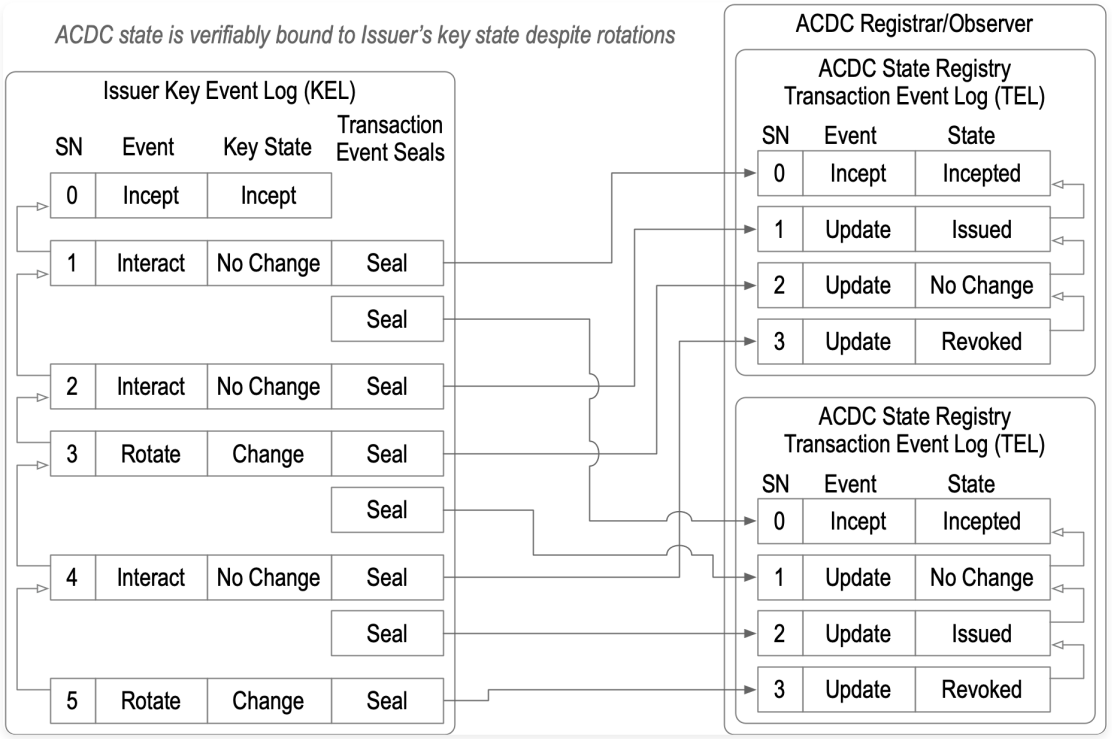
With the indirect case, the state of the ACDC is maintained by a Transaction Event LOG (TEL). That TEL could be an issuance/revocation Registry, for example, where the states of the ACDC are either *issued* or *revoked*. A TEL Registry has a registry inception event, `rip`. The SAID of this event MUST be anchored in the Issuer's KEL as the Registry proof seal. Update events in the Registry's TEL MUST also be anchored. An update event in the TEL MAY include the SAID of the ACDC, either blinded or unblinded. This binds the state of that ACDC to the Issuer's Key State at the time of the update event in the TEL.

In either case, this ACDC state proof seal makes a verifiable cryptographic binding between the state of the ACDC and the Key State of the Issuer at the time of ACDC state change. A verifiable presentation of the ACDC requires the validator to have knowledge of the ACDC state proof. This is provided via reference to the seal. This reference MAY be attached to the presentation or MAY be provided Out-Of-Band. In any case, given a reference to the seal, the validator can look up the seal in the Issuer's KEL and verify that the SAID of the Transaction Event is the SAID in the seal.

Likewise, because all the values in the sections of an ACDC are verifiably bound to the ACDC's SAID, then by extension, all the detail values in an ACDC are also verifiably bound to the Issuer's Key state. This means that ACDCs are not directly signed by the Issuer and are bound to the Issuer's Key State (either directly or indirectly), and the Issuer's Key State is signed. This enables the Key State of the Issuer to change independently of the ACDC state.

This is in contrast to other verifiable credential schemes, where the credentials are signed directly; in such schemes, a key rotation forces all the credentials signed with a given set of keys to be revoked; otherwise, a key compromise would enable the compromiser to issue verifiable credentials forged by the compromiser.

The approach of binding ACDC state to Issuer key state via anchored TEL sealsis shown in the following diagram:



Bound-ACDC-State

Figure: *Registrar Observer Components*

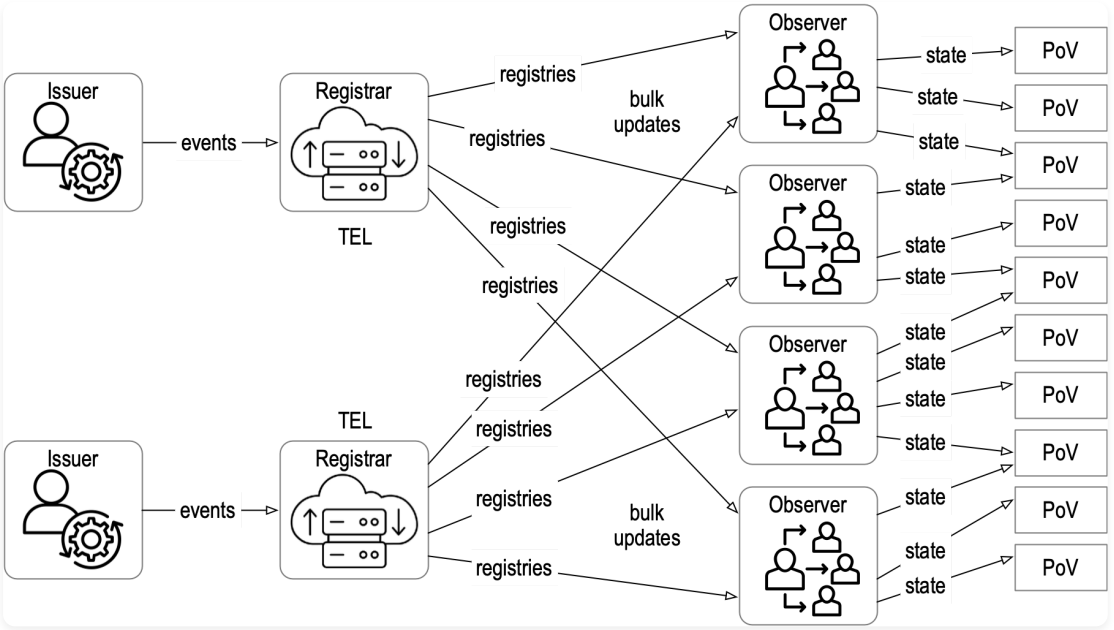
The requirement of an anchored ACDC state proof seal means that the forger must first successfully publish in the KEL of the Issuer an inclusion proof digest seal bound to a forged ACDC. The only way to publish an event in the Issuer's KEL is to verifiably sign the event, which means the forger must first compromise the issuer's private signing keys. This makes any forgery attempt detectable, and such an attempt makes the key compromise detectable.

To elaborate, the only way to successfully publish such a seal with compromised signing keys is in an Interaction Event in a KEL that has not yet recovered from that key compromise via a Rotation Event. This makes the forgery attempt both detectable and recoverable. In any event, the ACDC state proof seal ensures the detectability of any attempt at forgery using compromised keys for any ACDC state.

TEL Registrars and TEL Observers

The use of a Transaction Event Log (TEL) to manage the state of an ACDC may require two infrastructure components, namely, Registrars and Observers. A TEL Registrar is a computing component that operates under the auspices of the ACDC Issuer to maintain and publish a Registry of the ACDC state via a TEL. A TEL Observer, on the other hand, is a computing component that operates under the auspices of one or more Validators to cache the Registry, allowing Validators to validate the state of a given ACDC without exposing a point of validation (PoV). To clarify, an important feature of an Observer is that it can mask the usage of a given ACDC from the Issuer. The Observer maintains an updated cache of the Registry, reflecting state updates provided by the Registry. A validator queries its Observer, not the Registrar, at a point of validation (PoV) for an ACDC. A point of validation (PoV) occurs when an ACDC is presented to a Validator for validation. Whereas the interactions between the Observer and Registrar occur when there are ACDC state changes, not when there is a PoV of a given ACDC state. This protects against forced validator-to-issuer correlation of ACDC usage, i.e., no forced phone home validation.

To elaborate, an Observer starts up by downloading and caching the Registries from a Registrar. The Observer then updates its cache of current Registry state by either periodically polling the Registry for state updates or by subscribing to pushed state updates from the Registrar when available. Typically, because ACDC state changes are rare, optimized batch synchronization of state changes across multiple registries may be employed. Race conditions may be mitigated by implementing timed grace periods on revocations, whereby a revocation does not go into effect until a specified date and time after the state update. This enables synchronization batch windows to catch revocations in advance of PoV requests.



Registrar-Observers

Figure: *Registrar Observer Components*

Data Privacy

Information or data privacy is defined as the relationship between the collection and dissemination of data, technology, the public expectation of privacy, contextual information norms, and the legal and political issues surrounding them Information privacy. Data privacy is challenging since it attempts to allow the use of data by 2nd and 3rd parties while protecting personal (1st party) privacy preferences and personally identifiable information (PII). The fields of computer security, data security, and information security all design and use software, hardware, and human resources to address this issue. This definition is consistent with privacy viewed from the perspective of 1st party data rights and the role of 2nd parties in the three-party exploitation model defined below. The Trust over IP (ToIP) Foundation's architecture specification phrases privacy protection as answering the question: Privacy: will the expectations of each party with respect to the usage of shared information be honored by the other parties?

Trust over IP (ToIP) Technology Architecture Specification

Three-party Exploitation Model

Sustainable privacy is based on a three-party exploitation model Sustainable Privacy. Fundamentally, the goal is to protect the person (data subject) from exploitation via their personal data. In common usage, exploitation is selfishly taking advantage of someone to profit from them or otherwise benefit oneself. So, any unintended usage by any party is potentially exploitive. Intent is with respect to the person (data subject).

In this model, the 1st party is the person (data subject) of the original data. Their data is 1st party data. A 2nd party is the direct recipient of 1st party data as an intended recipient by the 1st party. A 3rd party is any other party who obtains or observes 1st party data but who is not the intended recipient.

There are two main avenues of exploitation of 1st party data. These are any 2nd party who uses the data in any way not intended by the 1st party and any 3rd party who uses 1st party data. To clarify, any unintended (unpermissioned) use of 1st party data by any 2nd party is naturally exploitive.

Moreover, because a 3rd party is defined as an unintended recipient, any use of 1st party data by a 3rd party is likewise, by definition, exploitive.

Furthermore, 1st party data may be conveyed by one 2nd party to another 2nd party (i.e. shared) in a non-exploitive manner when such conveyance and eventual use by the other 2nd party is intended (permitted) by the 1st party.

To elaborate, exploitation based on disclosure is characterized by 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).

Typically, protection from direct 3rd-party (Observer) exploitation without the collusion of the 2nd-party (Disclosee) of disclosed data may be provided by encrypting that data such that only the 2nd-party (Disclosee) may decrypt that data. Encryption is one effective mechanism for protecting the confidentiality of disclosed data from non-collusive 3rd-party observation. The detailed description of such mechanisms that are compatible with ACDCs is beyond the scope of this specification. See the Secure Privacy, Authenticity, Confidentiality protocol (SPAC) [[70] #SPAC] and the TSP protocol [[69] (#TSP)].

The primary mechanisms by which 2nd parties (Disclosees) erode the data privacy rights of disclosed data are as follows: • Exploitive use of 1st-party (Discloser) data by 2nd parties (Disclosees). • Sharing of 1st-party (Discloser) data by 2nd parties (Disclosees) with 3rd parties (Observers) either overtly (collusive) or inadvertently (leakage).

> ISSUE
>
> This model is diagrammed below. Need diagram here

Exploitation Protection Mechanisms

Least Disclosure

ACDCs provide several mechanisms designed to help protect against the exploitation of disclosed data. These are based on the principle of least disclosure as follows:

The system should disclose only the minimum amount of information about a given party needed to facilitate a transaction, and no more. [47]

Graduated Disclosure

Any given transaction may entail several disclosures that are iterative and incremental, such that one least disclosure facilitates another least disclosure, and so forth. The incremental interactive application of the principle of least disclosure is called Graduated Disclosure. The important insight is that one type of transaction enabled by a given least disclosure is one that specifically enables further disclosure. In other words, disclose enough to enable more disclosure, which in turn may enable even more disclosure. To clarify, Graduated Disclosure enables a potential Discloser to follow the principle of least disclosure by providing the least amount of information i.e., partial, incomplete, or uncorrelatable information needed to further a transaction. This progression of successive least disclosures to enable further disclosures forms a recursive loop of least disclosure-enabled transactions. In other words, the principle of least disclosure may be applied recursively as part of a Graduated Disclosure. A Contractually Protected Disclosure, for example, may result from the recursive application of least disclosure transactions.

There are several graduated disclosure mechanisms as follows:

- Compact Disclosure
- Metadata Disclosure
- Partial Disclosure
- Nested Partial Disclosure
- Full Disclosure
- Selective Disclosure
- Bulk-issued Instance Disclosure

As their names suggest, the Graduated Disclosure mechanisms disclose more or less of an ACDC. A short summary of each mechanism is provided immediately below. More detailed descriptions of each of the Graduated Disclosure mechanisms are either provided in the ensuing sections or in the Annex.

- Compact Disclosure of a block (field map) of data relies on the inclusion in that block of a cryptographic digest of the content (SAID) of that content. Disclosure of the SAID makes a verifiable commitment to its data that MAY be more fully disclosed lat-

er. The Schema for the block MUST include a `oneOf` composition Operator that validates against both the compact and full versions of the block.

- Metadata Disclosure happens with a Metadata ACDC is used to disclose any part of an ACDC. As defined above, a Metadata ACDC is indicated by the appearance of an empty, top-level UUID, `u`, field. Recall that 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`, the field of that yet-to-be disclosed ACDC.

- Partial Disclosure of a data block relies upon a cryptographic digest (SAID) of the content and a salty nonce (UUID) embedded in that content. The presence of the salty nonce means that disclosure of its digest (SAID) plus a Schema of that content is not enough to discover the actual content. The content remains blinded in spite of disclosure of its SAID until and unless the salty nonce (UUID) is also disclosed. The Schema for the block includes a `oneOf` composition Operator that validates against both the compact and full versions of the block.

- Nested Partial Disclosure of a tree of hierarchical data blocks relies on each nested block embedding both its digest (SAID) and a salty nonce (UUID). This allows the Partial Disclosure of different branches of the tree at different levels of nesting. The Schema for the block includes a `oneOf` composition Operator at each level of nesting that validates against both the compact and full versions of the nested block and any nesting levels above it in the tree.

- Full Disclosure is disclosure without hiding a given block's content behind SAIDs or salted SAIDs.

- Selective Disclosure of a set of data blocks relies on each element embedding its digest (said) and salty nonce (UUID) as partially disclosable elements. The Schema for such a set is unordered such that the disclosure of any element does not leak information about any other element. This requires a combination of an `anyOf` composition Operator at the set level and `oneOf` composition Operators for each element. Membership in the set can be verified against a set of SAIDs, one from each element. The salty nonce effectively blinds the element's contents when only its SAID is disclosed. The `anyOf` composition Operator is not order-dependent. This means that the selectively disclosable set can be provided as an ordered list of elements, yet one or more of its elements MAY be disclosed in any order so that the original order does not leak information.

- Bulk-issued Instance Disclosure relies on issuing multiple instances of a given ACDC, each a copy but with unique instance identifiers so that the disclosure of one instance is not correlatable to another via the instance identifiers.

All the Graduated Disclosure mechanisms MAY be used in combination.

A salient difference between Partial Disclosure and Selective Disclosure of a given block is the degree to which information about other fields is exposed in order to make Full Disclosure of its detailed field values. A partially disclosable block, when fully disclosed, exposes, at the very least, the labels of other fields in its enclosing block (a field map). Whereas a selectively disclosable block, when fully disclosed, does not expose any information about other yet-to-be-exposed fields, including their labels in its enclosing block (a field map array).

To clarify, when used in the context of Selective Disclosure, Full Disclosure means detailed disclosure of the selectively disclosed attributes in the element's block, not detailed disclosure of all selectively disclosable attributes in all elements. Whereas when used in the context of Partial Disclosure, Full Disclosure means detailed disclosure of at least the labels of other fields in the enclosing field map (block) that was so far only partially disclosed. Full Disclosure of a nested Partially disclosed block entails the Full Disclosure of the fields in the branch that extends down to the nested block and at least the disclosure of the labels of all the fields in the enclosing blocks of that branch.

Contractually Protected Disclosure

Graduated Disclosure enables a comprehensive protection mechanism called Contractually Protected Disclosure. There are two contractually protected disclosure mechanisms as follows:

- Chain-Link Confidentiality Disclosure
- Contingent Disclosure

In a Contractually Protected Disclosure, the potential Discloser first makes an offer using the least (Partial) Disclosure of some information about other information to be disclosed (Full Disclosure) contingent on the potential Disclosee first agreeing to the contractual terms provided in the offer. The contractual terms could, for example, limit the disclosure to third parties of the yet to be disclosed information. But those contractual terms MAY also include provisions that protect against liability or other concerns, not merely disclosure to third parties. The process by which such least disclosures progress to full disclosure is described in the IPEX (Issuance and Exchange Protocol) section below IPEX.

One special case of a Contractually protected disclosure is a Chain-Link Confidential disclosure [[44]]. Chain-Link Confidentiality imposes conditions and limitations on the further disclosure and/or use of the disclosed data. These MAY be specific terms of use or other consensual constraints. These terms MAY be applied to subsequent disclosures by the Disclosee that follow the data (hence chain-link). Another way of viewing Chain-link confidential disclosure is that the disclosed data has "strings attached." The chaining, in this case, is different from the chaining of ACDCs via their edges, i.e., a DAG of ACDCs. Chain-link confidentiality, in contrast, chains together a sequence of Disclosees. Each Disclosee in the sequence, in turn, is the Discloser to the next Disclosee. The terms-of-use of the original disclosure as applied to the original Disclosee MUST be applied by

each subsequent Discloser to each subsequent Disclosee via each of the subsequent disclosures. These terms of use are meant to contractually protect the data rights of the original Issuer or Issuee of the data being disclosed. These terms of use typically constrain disclosure to only approved parties, i.e., imbue the chain of disclosures with some degree of confidentiality.

Another special case of Contractually Protected Disclosure is Contingent Disclosure. In a Contingent Disclosure, some contingency is specified in the Rules section that places an obligation by some party to make a disclosure when the contingency is satisfied. This might be recourse given the breach of some other contract term. When that contingency is met, then the Contingent Disclosure MUST be made by the party whose responsibility it is to satisfy that disclosure obligation. The responsible party MAY be the Discloser, or it MAY be some other party, such as an escrow agent. The Contingent Disclosure clause MAY reference a cryptographic commitment to a private ACDC or private Attribute ACDC (Partial Disclosure) that satisfies via its Full Disclosure the Contingent Disclosure requirement. Contingent Disclosure MAY be used to limit the actual disclosure of personally identifying information (PII) to a just-in-time, need-to-know basis (i.e., upon the contingency) and not a priori. As long as the Discloser and Disclosee trust the escrow agent and the verifiability of the commitment, there is no need to disclose PII about the Discloser in order to enable a transaction, but merely an agreement to the terms of the contingency. This enables something called latent accountability. Recourse via Full Disclosure of PII is latent in the Contingent Disclosure but never realized (actualized) until the conditions of the contingency is satisfied. This minimizes inadvertent leakage while protecting both the Discloser and the Disclosee.

Issuance and Presentation Exchange (IPEX)

This section is non-normative. This merely outlines the Issuance and Presentation Exchange (IPEX) Protocol. A separate more detailed specification is being developed to more fully explicate this prototol. The IPEX protocol provides a uniform mechanism for the issuance and presentation of ACDCs ACDC in a securely attributable manner. A single protocol is able to work for both types of exchanges by recognizing that all exchanges (both issuance and presentation) MAY be modeled as the disclosure of information by a Discloser to a Disclosee. This specification is not exhaustive, it is only normative in the sense that it specifies the message types and routes that SHOULD be used in either issuance or presentation exchanges. It provides sufficient detail to show a likely mechanism for implementing the various Graduated Disclosure mechanisms described above. A given implementation of issuance and or presentation exchange is typically application and ecosystem-dependent and would, therefore, require additional specifications for issuance and/or presentation exchange tuned to those application and ecosystem contexts. This specification provides a baseline for other specifications.

The difference between exchange types is the information disclosed, not the mechanism for disclosure. Furthermore, the chaining mechanism of ACDCs and support for both Targeted and Untargeted ACDCs provide sufficient variability to accommodate the differences in applications or use cases without requiring a difference in the exchange protocol itself. This greatly simplifies the exchange protocol. This simplification has two primary advantages. The first is enhanced security. A well-delimited protocol can be designed and analyzed to minimize and mitigate attack mechanisms.

The second is convenience. A standard, simple protocol is easier to implement, support, update, understand, and adopt. The tooling is more consistent.

This IPEX IPEX protocol leverages important features of ACDCs and ancillary protocols such as CESR [1], SAIDs [3], and CESR-Path proofs (Proof-ID) as well as Ricardian Contracts ACDC [[43] and Graduated Disclosure (Metadata, Partial, Selective, Full) to enable Contractually Protected Disclosure. Contractually Protected Disclosure includes both Chain-Link Confidentialand Contingent Disclosure [[44]] ACDC .

IPEX Protocol Messages

The baseline exchange protocol is composed of a standard set of routed KERI exchange, `exn` messages. The REQUIRED routes MUST appear as values of the route, `r` field in the enclosing exchange message. The notional semantics of each route indicate how the message SHOULD be used in an exchange. The routes and semantics are delimited in the following table:

Discloser	Disclosee	Initiate	Contents	Description
	`apply`	Y	Schema or its SAID, Attribute field label list, Aggregate element label list, signature on `apply` or its SAID	defines wanted disclosure
`spurn`		N		rejects `apply`
`offer`		Y	Metadata ACDC or its SAID, Schema or its SAID, partial disclosure, Aggregate element label list, signature on `offer` or its SAID	proposes acceptable disclosure
	`spurn`	N		rejects `offer`
	`agree`	N	signature and/or anchored seal on `offer` or its SAID	accepts `offer`
`spurn`		N		rejects `agree`
`grant`		Y	Full or Selective Disclosure	discloses agreed to `offer`

Discloser	Disclosee	Initiate	Contents	Description
			ACDC, signature on `grant` or its SAID	
	`admit`	N	signature and/or anchored seal on `grant` or its SAID	confirms received `grant` disclosure

Commitments via SAID

All variants of an ACDC have various degrees of expansion compared to the compact variant. Therefore, an Issuer commitment via a signature (direct) or KEL anchored seal (indirect) to any variant of ACDC (metadata, compact, partial, nested partial, full, selective, etc.) makes a cryptographic commitment to the top-level section fields shared by all variants of that ACDC because the value of a top-level section field is either the SAD or the SAID of the SAD of the associated section. Both a SAD and its SAID, when signed or sealed, each provide a verifiable commitment to the SAD. In the former, the signature or seal verification is directly against the SAD itself. In the latter, the SAID as digest MUST first be verified against its SAD, and then the signature or seal on the SAID MAY be verified. This indirect verifiability (one or multiple levels of commitment via cryptographic digests) assumes that the cryptographic strength of the SAID digest is equivalent to the cryptographic strength of the signature used to sign it. To clarify, because all variants share the same top-level structure as the compact variant, a signature on any variant MAY be used to verify the Issuer's commitment to any other variant either directly or indirectly, in whole or in part, on a top-level section-by-section basis. This cross-variant Issuer commitment verifiability is an essential property that supports Graduated Disclosure by the Disclosee of any or all variants, whether Full, Compact, Metadata, Partial, Selective, etc.

To elaborate, the SAID of a given variant is useful even when it is not the SAID of the variant the Issuer signed because, during Graduated Disclosure, the Discloser MAY choose to sign or seal that given variant to fulfill a given step in an IPEX Graduated Disclosure transaction. The Discloser thereby can make a verifiable disclosure in a given step of the SAD of a given variant that fulfills a commitment made in a prior step via its signature or seal on merely the SAID of the SAD of the variant so disclosed.

For example, the metadata variant of an ACDC will have a different SAID than the compact variant because some of the top-level field values MAY be empty in the metadata variant. One can think of the metadata variant as a partial manifest that only includes those top-level sections that the Discloser is committing to disclose in order to induce the Disclosee to agree to the contractual terms of use when disclosed.

To elaborate, an IPEX transaction is between the Discloser and Disclosee, who both SHOULD make non-repudiable commitments to each other via signing or sealing variants of the ACDC to be disclosed. Typically, this means that the Discloser will eventually need to fulfill its commitment with proof of disclosure to the Disclosee. This proof MAY be satisfied either against the Discloser's signature or seal on the actual disclosed SAD or against the Discloser's signature or seal on the SAID of the actual disclosed SAD. In addition, the Disclosee SHOULD typically require proof of issuance via a non-repudiable signature or seal by the Issuer on a variant of the disclosed SAD that is verifiable (directly or indirectly) against the variant that is the disclosed SAD.

To summarize, when the Issuer commits to the composed Schema of an ACDC it is committing to all the variants so composed. As described above, the top-level field values in the compact variant enable verification against disclosure of any of the other Issuer committed variants because they all share the same top-level structure. This applies even to the metadata variant in spite of it only providing values for some top-level sections and not others. The verifiablity of a top-level section is separable.

Consequently, the IPEX protocol SHOULD specify how a validator does validation of any variant in a Graduated Disclosure. To restate, there are two proofs that a Discloser SHOULD provide. The first is proof of issuance (PoI), and the second is proof of disclosure (PoD). In the former, the Discloser provides the variant via its SAD that was actually signed or seal (as SAD or SAID of SAD) by the Issuer in order for the Disclosee to verify authentic issuance via the signature on that variant. In the latter, the Discloser discloses the Issuer-enabled (via Schema composition) variant that the Discloser offered to disclose as part of the Graduated Disclosure process.

IPEX Validation

The goal is to define a validation process (set of rules) that works for all variants of an ACDC and for all types of Graduated Disclosure of that ACDC.

For example, in the bulk issuance of an ACDC (see bulk issued ACDCs in the Annex Bulk), the Issuer only signs or seals the blinded SAID of the SAD, which is the compact variant of the ACDC, not the SAD itself. This enables a Discloser to make a proof of inclusion of the ACDC in a bulk issuance set by unblinding the signature on the blinded SAID without leaking correlation to anything but the blinded SAID itself. To clarify, the Disclosee can verify the commitment to the SAID via set inclusion without disclosing any

other information about the ACDC. Issuer signing or sealing of the SAID, not the SAD, also has the benefit of minimizing the computation of large numbers of bulk-issued commitments.

Issuer Commitment Rules

The Issuer MUST provide a signature or seal on the SAID of the most compact form variant defined by the Schema of the ACDC (see the “most compact form” algorithm above).

The different variants of an ACDC form a hash tree (using SAIDs). A commitment to the top-level SAID of the compact version of the ACDC is equivalent to a commitment to the hash tree root (trunk). This makes a verifiable commitment to all expansions of that tree. Different variants of an ACDC (SADs with SAIDs) correspond to different paths through the hash tree. The process of verifying a nested block SAD against its SAID is essentially verifying proof of inclusion of the branch of the hash tree that includes that nested block. This allows a single commitment (signature or seal) to provide PoI of the presentation of any Schema-authorized variants of the ACDC.

An Issuer SHOULD provide signatures or seals on the SAIDs of other variants and on the SADs of other variants.

To summarize:

Proof of Issuance (PoI) MAY be provided by disclosing the SAID of the most compact variant and the verifiable commitment (signature or seal) by the Issuer on that SAID.

Proof of Disclosure (PoD) MAY be provided by disclosing the SAD of the most compact variant and then recursively disclosing (expanding) the nested SADs of each of the blocks of the most compact variant as needed for the promised disclosure.

Thus, for any disclosed variant of an ACDC, the Disclosee MAY need only verify one PoI as defined above, and MAY need to verify a specific PoD for a given disclosed variant as defined above.

Disclosure-specific (Bespoke) Issued ACDCs

Chaining two or more ACDCs via edges enables disclosure-specific issuance of bespoke issued ACDCs. A given Discloser of an ACDC issued by some Issuer might want to augment the disclosure with additional contractual obligations or additional information sourced by the Discloser where those augmentations are specific to a given context, such as a specific Disclosee. A given Discloser issues its own bespoke ACDC referencing some other ACDC via an Edge. This means that the normal validation logic and tooling for a chained ACDC can be applied without complicating the presentation exchange logic. Furthermore, Attributes in other ACDCs pointed to by Edges in the bespoke ACDC MAY be addressed by Attributes in the bespoke ACDC using JSON Pointer or CESR-SAD-Path [6] proof references that are relative to the node SAID in the Edge Proof-ID .

For example, this approach enables the bespoke ACDC to identify (name) the Disclosee directly as the Issuee of the bespoke ACDC. This enables contractual legal language in the Rules section of the bespoke ACDC that references the Issuee of that ACDC as a named party. Signing the agreement to the offer of that bespoke ACDC consummates a contract between the named Issuer and the named Issuee. This approach means that custom or bespoke presentations do not need additional complexity or extensions. Extensibility comes from reusing the tooling for issuing ACDCs to issue a bespoke or disclosure-specific ACDC. When the only purpose of the bespoke ACDC is to augment the contractual obligations associated with the disclosure, then the Attribute section, `a`, field value of the bespoke ACDC MAY be empty, or it MAY include properties whose only purpose is to support the bespoke contractual language.

Similarly, this approach effectively enables a type of rich presentation or combined disclosure where multiple ACDCs MAY be referenced by edges in the bespoke ACDC that each contributes some Attribute(s) to the effective set of Attributes referenced in the bespoke ACDC. The bespoke ACDC enables the equivalent of a rich presentation without requiring any new tooling [59].

Example of a Bespoke Issued ACDC

Consider the following disclosure-specific ACDC. The Issuer is the Discloser, the Issuee is the Disclosee. The Rules section includes a context-specific anti-assimilation clause that limits the use of the information to a single one-time usage purpose, in this case, admittance to a restaurant. The ACDC includes an edge that references some other ACDC that may, for example, be a coupon or gift card. The Attribute section includes the date and place of admittance.


```
{
  "v":  "ACDC10JSON00011c_",
  "d":  "EBdXt3gIXOf2BBWNHdSXCJnFJL5OuQPyM5K0neuniccM",
  "i":  "EmkPreYpZfFk66jpf3uFv7vklXKhzBrAqjsKAn2EDIPM",
  "s":  "EGGeIZ8a8FWS7a646jrVPTzlSkUPqs4reAXRZOkogZ2A",
  "a":
  {
    "d": "EgveY4-9XgOcLxUderzwLIr9Bf7V_NHwY1lkFrn9y2PY",
    "i": "EpZfFk66jpf3uFv7vklXKhzBrAqjsKAn2EDIPmkPreYA",
    "date": "2022-08-22T17:50:09.988921+00:00",
    "place": "GoodFood Restaurant, 953 East Sheridan Ave, Cody WY
82414 USA"
  },
  "e":
  {
    "d": "EerzwLIr9Bf7V_NHwY1lkFrn9y2PgveY4-9XgOcLxUdY",
    "other":
    {
      "d": "E9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NHwY1lkFrn",
```

```
      "n": "EIl3MORH3dCdoFOLe71iheqcywJcnjtJtQIYPvAu6DZA"
    }
  },
  "r":
  {
    "d": "EwY1lkFrn9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NA",
    "Assimilation":
    {
      "d": "EXgOcLxUdYerzwLIr9Bf7V_NAwY1lkFrn9y2PgveY4-9",
      "l": "Issuee hereby explicitly and unambiguously agrees to
NOT assimilate, aggregate, correlate, or otherwise use in
combination with other information available to the Issuee, the
information, in whole or in part, referenced by this container or
any containers recursively referenced by the edge section, for
any purpose other than that expressly permitted by the Purpose
clause."
    },
    "Purpose":
    {
      "d": "EY1lkFrn9y2PgveY4-9XgOcLxUdYerzwLIr9Bf7V_NAw",
      "l": "One-time admittance of Issuer by Issuee to eat at
place on date as specified in Attribute section."
    }
  }
}
```

Informative examples of fully-featured variants of ACDCs can be found in Annex C.

Transaction event logs (TELs) as ACDC state registries

Overview

A Transaction Event Log (TEL) is a hash-chained data structure of sealed transaction events that can be used to track the transaction states (typically those associated with one or more ACDCs). Events in the TEL are sealed (anchored) in a Key Event Log (KEL) using seals. A seal can be as simple as the event's SAID (cryptographic strength digest). A transaction event seal MAY also include the transaction event's sequence number to make it easier to look up and verify. Because key events in a KEL are nonrepudiably signed by its Controller, the appearance of a transaction event seal provides a verifiable non-repudiable commitment to the transaction event by the KEL Controller. This makes TELs, which are thereby bound to KELs, also securely attributable to the KEL's controller. This provides verifiable but decorrelatable extensibility to KEL semantics. Any number of transaction event types can be constructed for different applications that may be securely attributed without complicating KEL semantics. The seals need no semantics beyond their secure attributability to the AID of the KEL controller. The semantics of the transaction event's state may be hidden by the transaction event SAID, which in turn

may be protected from rainbow table attack by a cryptographic strength UUID in the transaction event. Therefore, the transaction state, as given by a sequence of transaction events, can be either public or private, depending on how the transaction events are structured. Similar to ACDCs themselves, Graduated Disclosure mechanisms may be applied to transaction events.

Importantly, the process of sealing transaction events in a KEL binds the Key State at the sealing (anchoring) key event to the transaction state. This enables an extremely beneficial property of TELs; that is, the verifiability of transaction events in the TEL persists in spite of changes to Key States in the sealing KEL. In other words, the verifiability of transaction events persists in spite of changes in the Key State. To clarify, sealed transaction events remain verifiably bound to the Key State at the point of issuance of the sealing event. An example of a transaction state that benefits from this property is a TEL that tracks the issuance and revocation state of dynamically revocable ACDCs, i.e., a revocation registry.

The transaction events MUST be sealed (anchored or bound) in a KEL using transaction event seals, whose JSON representation MUST be as follows.

```
{
 "s": "3",
 "d": "ELvaU6Z-i0d8JJR2nmwyYAZAoTNZH3UfSVPzhzS6b5CM"
}
```

The CESR representation of the seal couple is given by the Count Code `-T##` or `--T#####`

Validating transaction events

As described above, a KEL can control a TEL by sealing (anchoring) transaction-specific events from the TEL inside key events in the KEL. The steps to this process are defined as follows:

1. Assign each TEL a universally unique identifier (such as the SAID of an associated ACDC)
2. Associate each TEL with the KEL's Controller AID that seals the transaction events. This AID is the Issuer of the transaction event.
3. Create the transaction-specific event with an event SAID.
4. Generate a seal that includes the event SAID.
5. Include the seal of that transaction event in a key event in the controlling KEL.
6. Have the sealing key event accepted (appended) into its KEL by the KEL Controller. This means at least signing the sealing key event and may include witnessing and or delegating the event. Such acceptance makes a verifiable, nonrepudiable commitment to the seal (digest) of the serialized transaction event.

Any Validator can cryptographically verify the authoritative state of the transaction by validating the presence of the seal in the associated KEL. The TEL events themselves do not have to be signed because the signed commitment to the event in the form of the digest in the seal in the KEL is cryptographically equivalent to signing the transaction event itself. Typically, the Validator is given a reference to the sealing event via a key event seal reference that includes the Issuer (KEL Controller) AID, the key event's sequence number, and its SAID. The Issuer of the transaction event can either directly attach the seal reference to the ACDC or can provide an API where transaction events or their seal references and their KEL seal references can be looked up by the SAID of the ACDC associated with the transaction event. Given a key event seal reference for a given transaction event, a Validator can then look up and verify the presence of the seal in the KEL.

Verifiable Container/Credential Registry

ACDCs may be rightly generically referred to as Verifiable Containers (VCs). Often, ACDCs are used as entitlements or credentials and, therefore, may also be rightly referred to as Verifiable Credentials (VCs). The abbreviation VC works in either case. An ACDC can more simply be denoted as a Container or a Credential. A Verifiable Container/Credential Registry (VCR) is a type of TEL. It is a form of a Verifiable Data Registry (VDR) that tracks the state of ACDCs issued by the Controller of the KEL. Without loss of specificity, in this section, a TEL serving the purpose of a VCR may be simply denoted as a Registry. A Registry that tracks the dynamic issuance and revocation state of an ACDC is called a Revocation Registry.

Registry Message Types and Fields

ACDC state registries (TELs) have three types of events. These types are shown in the following table:

Ilk	Name	Description
`rip`	Registry Inception	registry initialization
`bup`	Blindable Update	blindable transaction event state update
`upd`	Update	non-blindable transaction event state update

A Registry MAY be used in either a blinded or unblinded fashion or both, depending on the type of state update used. Even when a blindable state update is used, the Issuer could explicitly unblind the state by attaching the unblinded state to the event when publishing it. In contrast, an unblindable state update can never be used in a blinded fashion. Its state is always exposed. The advantage of an unblindable state update is its simplici-

ty. A given Registry could switch between using blindable and unblindable update messages as its state evolves. Consequently, a State-Registry is generic. It MAY be used in either a private (blinded) or public (unblinded) manner.

Top-level fields

The reserved field labels for the top level of a state registry (both blindable and unblindable) transaction event are as follows:

Label	Description
`v`	Version String
`t`	Message type
`d`	event block SAID
`u`	UUID salty nonce
`i`	Issuer AID
`rd`	Registry SAID
`n`	sequence number
`p`	prior event SAID
`dt`	Issuer relative ISO date/time string
`b`	blinded attribute block SAID
`ta`	transaction target ACDC SAID
`ts`	transaction state

Registry Inception event fields

The fields for the Registry-Inception, `rip` event , given by their labels, MUST appear in the following order, `[v, t, d, u, i, n, dt]`. All are required. The value of the Message type, `t` field MUST be `rip`. The value of the sequence number field, `s` MUST be the hex encoded string for the integer 0.

Blindable Update event fields

The fields for the Blindable-Update, `bup` event , given by their labels, MUST appear in the following order, `[v, t, d, rd, n, p, dt, b]`. All are required. The value of the Message type, `t` field MUST be `bup`.

Update event fields

The fields for the Update, `upd` event , given by their labels, MUST appear in the following order, `[v, t, d, rd, n, p, dt, ta, ts]`. All are required. The value of the Message type, `t` field MUST be `upd`.

Field descriptions

Version String, `v` field

The Version String, `v` field value uses the same format as a top-level ACDC message Version String. The protocol type MUST be `ACDC`.

Message type, `t` field

The Message type, `t` field value MUST be one of the Message types in the table above. The Message types do not leak any state information. The first Message in some types of Registries such as an issuance/revocation Registry.

SAID, `d` field

The SAID, `d` field value MUST be the SAID of its enclosing block. A transaction event's SAID enables a verifiable globally unique reference to that event.

UUID, `u` field

The UUID, `u` field value MUST be a cryptographic strength salty nonce with approximately 128 bits of entropy (nominally). The UUID, `u` field means that the block's SAID, `d` field value provides a secure cryptographic digest of the contents of the block [[48]]. An adversary, when given both the block's SAID and knowledge of all possible state values, cannot discover the actual state in a computationally feasible manner, such as a rainbow table attack [[30]] [[31]]. Therefore, the block's UUID, `u` field securely blinds the contents of the block via its SAID, `d` field, notwithstanding knowledge of both the block's structure, possible state values, and SAID. Moreover, a cryptographic commitment to that block's SAID, `d` field does not provide a fixed point of correlation to the block's state unless and until there has been a disclosure of that state.

Issuer, `i` field

The Issuer, `i` field value MUST be the AID of the Issuer. This removes any ambiguity about the semantics of a seal of a transaction event that appears in a KEL. When the KEL controller AID and Issuer AID are the same for a transaction event seal that appears in a given KEL, then the KEL Controller is making a commitment as Issuer to the transaction event. A transaction event seal that appears in a KEL with a different Controller AID is merely a nonrepudiable endorsement of the transaction state by some other party, not a duplicity-evident nonrepudiable commitment by the Issuer to the transaction state. This may appear to be redundant because the Issuer AID also appears in the ACDC. In a blinded state Registry, however, the ACDC SAID only appears in the blinded Attribute block likewise in a yet-to-be-disclosed private ACDC, the Issuer is also blinded, so a

Registry observer that hosts a copy of the Registry that is not also a Disclosee of either the expanded transaction event and/or the associated ADCD would not be able to confirm that commitment and may thereby be subject to a DDoS attack without the presence of the Issuer, `i` field in the Registy initialization event's public top-level fields.

Registry SAID, `rd` field

The Registry SAID, `rd` field value MUST be the value of the SAID, `d` field of the Registry Inception, `rip` event. Because the Issuer `i` field appears in the `rip` event, the Registry SAID, `rd` field value cryptographically binds the Registry to the Issuer AID. The Registry SAID enables a verifiable globally unique reference to the Registry (TEL). Update events MUST include the Registry SAID, `rd` field so that they can be verifiably associated with the Registry (TEL). The ACDC managed by the Registry MAY also includes a reference to the Registry in the ADCD's own Registry SAID, `rd` field, thereby providing secure discovery of the Registry when given the ACDC. Recall that an ACDC's `rd` field may appear at the top level of an ACDC or nested inside the ACDC's Attribute `a` section or its Aggregate `A` section. Nesting can be beneficial in some graduated disclosure contexts. When correlation minimization is more important than secure discovery, then the ACDC's `rd` field may be empty or missing. To clarify, when provided, the value of the Registry SAID, `rd` field in the associated ACDC is set to the same value as the corresponding Registry SAID, `rd` field value of the associated Registry events (TEL).

Sequence number, `n` field

The sequence number, `n` field value MUST be a hex-encoded string with no leading zeros of a zero-based strictly monotonically increasing integer. The first (zeroth) transaction event in a given Registry (TEL) MUST have a sequence number of 0 or `0` hex.

Prior event SAID, `p` field

The prior event SAID, `p` field value MUST be the SAID, `d` field value of the immediately prior event in the TEL. The prior, `p` field backward chains together the events in a given TEL.

Datetime, `dt` field

The datetime, `dt` field value MUST be the ISO-8601 datetime string with microseconds and UTC offset as per IETF RFC-3339. This MUST be the datetime of the issuance of the transaction event relative to the clock of the issuer. An example datetime string in this format is as follows:

```
2020-08-22T17:50:09.988921+00:00
```

Transaction ACDC SAID, `td` field

The transaction ACDC SAID, `td` field value is the SAID of the ACDC itself. It is the value of the top-level `d` field in the ACDC. This binds the ACDC to the TEL (Registry). Each event in the registry (TEL) is, in turn, bound to the key state of the issuer via an anchored

seal in the KEL of the Issuer. This hierarchical binding binds the key-state of the issuer to the TEL, which in turn is bound to the ACDC itself. This binding is a verifiable commitment by the issuer to its issuance of the ACDC that survives changes in the keystate of the Issuer.

Transaction state, `ts` field

The transaction state, `ts` field value MUST be a string from a small finite set of strings that delimit the possible values of the transaction state for the Registry. For example, the state values for an issuance/revocation registry may be `issued` or `revoked`.

Blinded attribute, `b` field

The blinded attribute, `b` field value MUST be the SAID of the blinded attribute block. This SAID of the blinded attribute block is also known as the BLID for blinding SAID. See below for a description of the blinded attribute block.

Blinded Attribute Block

The blinded attribute block corresponding to the blinded attribute `b` field value has the following fields:

Virtual Label	Description
`d`	BLID blinding SAID
`u`	UUID salty nonce blinding factor, random or HD generated
`td`	Transaction ACDC SAID field this is the value of the top-level `d` field in the ACDC
`ts`	Transaction state value string

The fields MUST appear in the following order: `[d, u, td, ts]`.

The field labels are virtual in that the labels never appear in the CESR serialization; they are a mnemonic for reference purposes. For example, the value of the Bliding SAID `d` field in the blinded attribute block appears as the value of the labeled BLID `b` field in the associated blindable state update, `bup`, message. Likewise, the `td` and `ts` field values have the same semantics as the `td` and `ts` field values in the non-blindable update, `upd` message.

The actual blinded attribute block is not part of the blindable update message. It MAY be provided as an attachment to some message, such as the associate ACDC, as part of a CESR serialization that MUST use one of the CESR group or count codes labeled

`BlindedStateQuadruples` with code format `-a##` or `BigBlindedStateQuadruples` with code format `--a######`. When provided as an attachment, multiple blinded attribute blocks MAY be included in a given group. A group consists of its count code followed by one or more Blinded attribute blocks, where each block is composed of a concatenation of the four fields serialized in order as CESR primitives that are appropriate for the BLID, UUID, ACDC SAID, and transaction state string fields in that block. An example of an attachment serialization with group code is provided below.

BLID, `d` field

The blinding SAID, BLID, `b` field value MUST be calculated as a cryptographic strength digest on the CESR serialization of the concatenation of block field values. The details of this calculation are provided later. A BLID is effectively a type of SAID. It is not calculated on the field map structure with a labeled field for a SAID, but uses instead a fixed field concatenation with a known place in that concatenation for its BLID (blinding SAID). A blinded attribute section's BLID enables a verifiable globally unique reference to that state contained in the block, but without necessarily disclosing that state.

The BLID, `d` field should typically use the Blake3 Digest code `E`, but any of the cryptographic strength digest codes of length 44 characters in the qb64 Text domain MAY be used. When the `td` field value is the empty placeholder, it uses the CESR `Empty` primitive code with value `1AAP`.

UUID, `u` field

When not empty, the UUID `u` field value MUST be a cryptographic strength salty nonce with approximately 128 bits of entropy (nominally). The UUID `u` field means that the block's SAID `d` field value provides a secure cryptographic digest of the contents of the block [[48]]. An adversary, when given both the block's SAID and knowledge of all possible state values, cannot discover the actual state in a computationally feasible manner, such as a rainbow table attack [[30]] [[31]]. Therefore, the block's UUID, `u` field securely blinds the contents of the block via its SAID, `d` field, notwithstanding knowledge of both the block's structure, possible state values, and SAID. Moreover, a cryptographic commitment to that block's SAID, `d` field does not provide a fixed point of correlation to the block's state unless and until there has been a disclosure of that state.

When used in public (unblinded) mode, the UUID, `u` field value MAY be the empty nonce value.

The UUID `u` field should use the `Salt_256` code for a 32-byte (256-bit) raw salty nonce value with approximately 256 bits of cryptographic strength. But any appropriate cryptographic digest may be used. Typically, this is some derivation process using a salt and a deterministic path based on the sequence number of the associated `bup` event message. When used in public unblinded mode, the UUID, `u` field value MAY be the CESR `Empty` primitive code with value `1AAP`.

When the UUID, `u`, is derived from a shared secret salt and a public path, such as the sequence number using a hierarchically deterministic derivation algorithm, and given that the possible state values are finite and small, then any holder of the shared secret can derive the state given the public information in the top-level fields of the transaction event. When the `u` field value is derived from a shared secret salt, the derivation algorithm MUST preserve the approximately 128 bits of cryptographic strength. This typically means a derived UUID `u` field value is 256 bits in length.

When the UUID is not derived with a hierarchically deterministic derivation algorithm, the Issuer and Issuee may need to interact for each transaction event update in order to exchange the shared secret salt.

Transaction ACDC SAID, `td` field

The transaction ACDC SAID, `td` field value is the SAID of the associated ACDC itself. It is the value of the top-level `d` field in the ACDC. This binds the ACDC to the TEL (Registry). The events in the registry are in turn, bound to the key state of the issuer via an anchored seal in the KEL of the issuer. This hierarchical binding binds the key-state of the issuer to the TEL, which in turn is bound to the ACDC itself. This binding is a verifiable commitment by the issuer to its issuance of the ACDC that survives changes in the keystate of the Issuer.

When the transaction ACDC SAID, `ts`, field value is a placeholder, it is indicated by the empty string, i.e., “”, which MUST be CESR encoded as the CESR `Empty` primitive code with value `1AAP`.

Transaction state, `ts` field

The transaction state, `ts` field value MUST be a string from a small finite set of strings that delimit the possible values of the transaction state for the Registry. For example, the state values for an issuance/revocation registry may be `issued` or `revoked`.

When the transaction state `ts` field value is a placeholder, it is indicated by the empty string, i.e., “”, which MUST be CESR encoded as the CESR `Empty` primitive code with value `1AAP`.

In general, the state `ts` field value may be encoded using any of the codes from the following table for string-ish encodings:

```
Empty: str= `1AAP` # Empty value for Nonce, UUID, state or relate
d fields
Tag1:  str = '0J'  # 1 B64 char tag with 1 pre pad
Tag2:  str = '0K'  # 2 B64 char tag
Tag3:  str = 'X'  # 3 B64 char tag
Tag4:  str = '1AAF'  # 4 B64 char tag
Tag5:  str = '0L'  # 5 B64 char tag with 1 pre pad
Tag6:  str = '0M'  # 6 B64 char tag
Tag7:  str = 'Y'  # 7 B64 char tag
```

```
Tag8:  str = '1AAN'  # 8 B64 char tag
Tag9:  str = '0N'  # 9 B64 char tag with 1 pre pad
Tag10: str = '0O'  # 10 B64 char tag
Tag11: str = 'Z'   # 11 B64 char tag
StrB64_L0:     str = '4A'  # String Base64 Only Leader Size 0
StrB64_L1:     str = '5A'  # String Base64 Only Leader Size 1
StrB64_L2:     str = '6A'  # String Base64 Only Leader Size 2
StrB64_Big_L0: str = '7AAA'  # String Base64 Only Big Leader Size
0
StrB64_Big_L1: str = '8AAA'  # String Base64 Only Big Leader Size
1
StrB64_Big_L2: str = '9AAA'  # String Base64 Only Big Leader Size
2
Label1:        str = 'V'  # Label1 1 bytes for label lead size 1
Label2:        str = 'W'  # Label2 2 bytes for label lead size 0
Bytes_L0:     str = '4B'  # Byte String lead size 0
Bytes_L1:     str = '5B'  # Byte String lead size 1
Bytes_L2:     str = '6B'  # Byte String lead size 2
Bytes_Big_L0: str = '7AAB'  # Byte String big lead size 0
Bytes_Big_L1: str = '8AAB'  # Byte String big lead size 1
Bytes_Big_L2: str = '9AAB'  # Byte String big lead size 2
```

Blinded State Disclosure

In some applications, it is desirable that the current state of an ACDC be hidden or blinded such that the only way for a potential Validator of the state to observe that state is when the Controller of some AID, when acting as Discloser, discloses the state at the time of presentation of that ACDC. This makes the associated TEL a blinded state registry. This is a type of private registry. To clarify, the Issuer designates some AID as the Discloser. Typically, for ACDCs with an Issuee, the Discloser is the Issuee, but the Issuer could designate any AID as the Discloser. Only the Discloser can unblind the state to a potential Disclosee.

Usually, a disclosure of a blinded state by Discloser to Disclosee is interactive. A Disclosee may only observe the state when first unblinded in an interactive exchange with the Discloser. After disclosure, the Discloser may then request that the Issuer update the state with a new blinding factor (the blind). The Disclosee cannot then observe the current state of the TEL without yet another disclosure interaction with the Discloser.

The blind is derived from a secret salt shared between the Issuer and the designated Discloser. The current blind is deterministically derived from this salt and the sequence number of the transaction event. This is used to blind the state of the event. To elaborate, the hierarchically deterministic derivation path for the blind is the sequence number of the TEL event, which, combined with the salt, produces a universally unique salty nonce (UUID) to act as the blind. The Issuer publishes the transaction event with a blinded state so that a Validator can independently verify the Issuer’s commitment to that state, but

without being able to determine the state via a transaction event seal in the Issuer's KEL with the SAID of that event. Only the Issuer can change the actual blinded state. Only the Issuer and Discloser have a copy of the secret salt, so only they can independently derive the current blind from the sequence number. Given the blind and a small finite number of possible values for the transaction state, the Discloser can verifiably discover and hence unblind the current transaction state from the published SAID of the current transaction event, its sequence number, and the shared secret salt.

The Issuer MAY provide an authenticated service endpoint for the Discloser to which the Discloser can make a signed request to update the blind. Each new event published by the Issuer in the Registry MUST increment the sequence number and hence the blinding factor, but MAY or MAY not change the actual blinded state. Because each updated event in the Registry has a new blinding factor, regardless of any actual change of state or not, an observer cannot correlate state to event updates.

In some cases, the Blindable-State-Registry may provide correlatable information, such as when the first real transaction state is always the same or the final state results in the registry's disuse. In those cases, the Issuer MAY choose to define an empty state and an empty ACDC SAID as known placeholder values. The first transaction state update event in the Registry can, therefore, be published before any real ACDC has been created, to which it may be later applied. Disuse of the Registry can be hidden by continuing to update the blind to the state without changing the final state value for some time after the ACDC has been revoked or abandoned.

Calculating the SAID of the serialized Blinded Attribute block

As mentioned above, the expanded attributed block is serialized as a concatenation of the CESR serializations of each of its field values. The BLID of such a serialization is computed on the QB64 TEXT domain representation. The BLID computation follows the SAID protocol, adapted for fixed-field representations. First, the size of the BLID field is derived from the type of digest to be used for its computation. For example, the number of characters of a BLAKE3-256 digest in the CESR Text domain is 44 characters. Its slot is replaced with `#` dummy characters, specifically forty-four `#` characters. The Text domain CESR serializations of the remaining fields are appended in order to form the string to be digested. The BLAKE3-256 32-byte raw digest of this dummied full serialization is then computed. This raw digest is serialized in the CESR Text domain (qualified Base64). This serialization is the BLID. Finally, the forty-four dummy characters are replaced with the 44 characters of the BLID. The resultant CESR serialized attribute block.

The BLID is the value BLID `b` field in the associated blindable update `bup` event message.

As mentioned previously, disclosure of the blinded attribute block can be provided by attaching its serialization via one of the CESR count (group) count codes labeled `BlindedStateQuadruples` with code format `-a##` or `BigBlindedStateQuadruples` with

code format `--a######`.

Blinded Attribute Block Placeholder Calculation Example

Suppose the UUID, `u` field value is encoded as `aG1lSjdJSNl7TiroPl67Uqzd5eFvzmr6b-PlL7Lh4ukv8`, the empty placeholder transaction ACDC said, `td` field value is encoded as `1AAP` (CESR coding for the empty value), and the transaction state, `ts` field is also encoded as `1AAP` for empty. The digest to be used to compute the BLID (Blinding SAID) is the BLAKE3-256 digest, which, when CESR encoded, has a length of 44 characters. This is filled with forty-four `#` dummy characters. The resulting length of the group content is 96 characters.

For the sake of clarity, the field values are shown in the table below:

Virtual Label	Value	Description
`d`	`############################################`	Dummied BLID (Blinding SAID)
`u`	`aG1lSjdJSNl7TiroPl67Uqzd5eFvzmr6bPlL7Lh4ukv8`	UUID salty nonce blinding factor, HD generated
`td`	`1AAP`	Transaction ACDC SAID field value, top-level `d`
`ts`	`1AAP`	Transaction state value string

The 96-character dummied serialization is as follows:

```
############################################aG1lSjdJSNl7TiroPl67Uqzd5eFvzmr6bPlL7Lh4ukv81AAP1AAP
```

The CESR encoded Blake3 digest of this string is `ECVr7QWEp_aqVQuz4yprRFXVxJ-9uWLx_d6oDinlHU6J`. This is substituted in the serialization above for the dummy characters to provide the final serialized expanded attribute block as follows:

```
ECVr7QWEp_aqVQuz4yprRFXVxJ-9uWLx_d6oDinlHU6JaG1lSjdJSNl7TiroPl67Uqzd5eFvzmr6bPlL7Lh4ukv81AAP1AAP
```

Broken out into fields, the values are provided in the table below:

Virtual Label	Value	Description
`d`	`ECVr7QWEp_aqVQuz4yprRFXVxJ-9uWLx_d6oDinlHU6J`	BLID (Blinding SAID)
`u`	`aG1lSjdJSNl7TiroPl67Uqzd5eFvzmr6bPlL7Lh4ukv8`	UUID salty nonce blinding factor, HD generated
`td`	`1AAP`	Transaction ACDC SAID field value, top-level `d`
`ts`	`1AAP`	Transaction state value string

In a presentation of the associated ACDC and/or TEL to a Disclosee, a discloser could attach this to verifiably unblind the blinded attribute SAID, `d` field in the associated blindable update, `bup` event.

An unblinded attachment of this blinded attribute block would be prefixed with the appropriate CESR count code, i.e., `BlindedStateQuadruples` with code format `-a##`, as follows:

```
-aAYECVr7QWEp_aqVQuz4yprRFXVxJ-9uWLx_d6oDinlHU6JaG1lSjdJSNl7TiroP
l67Uqzd5eFvzmr6bPlL7Lh4ukv81AAP1AAP
```

Blinded Attribute Block ACDC State Calculation Example

For example, suppose the UUID, `u` field value is encoded as `aLfCdNAnc-0P2Siruar-ZSajXiUWu5iU2VfQahvpNCyzB` the transaction ACDC said, `td` field value is encoded as `EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M`, and the transaction state, `ts` field value is the string “issued” encoded as `0Missued`. The digest to be used to compute the BLID is the BLAKE3-256 digest, which when CESR encoded has a length of 44 characters. This is filled with forty-four `#` dummy characters. The resulting length of the group content is 140 characters.

For the sake of clarity, the field values are shown in the table below:

Virtual Label	Value	Description
`d`	`############################################`	Dummied BLID (Blinding SAID)
`u`	`aLfCdNAnc-0P2SiruarZSajXiUWu5iU2VfQahvpNCyzB`	UUID salty nonce blinding factor, HD generated
`td`	`EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M`	Transaction ACDC SAID field value, top-level `d`
`ts`	`0Missued`	Transaction state value string

The 140-character dummied serialization is as follows:

```
############################################aLfCdNAnc-0P2SiruarZSajXiUWu5iU2VfQahvpNCyzBEMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M0Missued
```

The CESR encoded Blake3 digest of this string is `EOtWw6X_aoOJlkzNaLj23IC6MXHl7ZSYSWVulFW_Hr_t`. This is substituted in the serialization above for the dummy characters to provide the final serialized expanded attribute block as follows:

```
EOtWw6X_aoOJlkzNaLj23IC6MXHl7ZSYSWVulFW_Hr_taLfCdNAnc-0P2SiruarZSajXiUWu5iU2VfQahvpNCyzBEMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M0Missued
```

Broken out into fields, the values are provided in the table below:

Virtual Label	Value	Description
`d`	`EOtWw6X_aoOJlkzNaLj23IC6MXHl7ZSYSWVulFW_Hr_t`	Dummied BLID (Blinding SAID)
`u`	`aLfCdNAnc-0P2SiruarZSajXiUWu5iU2VfQahvpNCyzB`	UUID salty nonce blinding factor, HD generated
`td`	`EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M`	Transaction ACDC SAID field value, top-level `d`
`ts`	`0Missued`	Transaction state value string

In a presentation of the associated ACDC and/or TEL to a Disclosee, a discloser could attach this to verifiably unblind the blinded attribute SAID, `b` field in the associated blindable update, `bup` event.

An attachment of this blinded attribute block would be prefixed with the appropriate CESR count code, i.e., `BlindedStateQuadruples` with code format `-a##`, as follows:

```
-aAjEOtWw6X_aoOJlkzNaLj23IC6MXHl7ZSYSWVulFW_Hr_taLfCdNAnc-0P2SiruarZSajXiUWu5iU2VfQahvpNCyzBEMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M0Missued
```

Blinded State Registry Example

Consider a blindable state revocation registry for ACDCs operated in blinded (private) mode. The transaction state can be one of two values, `issued` or `revoked`. In this case, the placeholder value of the empty string for the transaction state, `ts` field, is also employed to decorrelate the initialization. The Issuer with AID, `ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf` first creates one among many placeholder Registries by issuing the following transaction event:

```
{
    "v": "ACDCCAACAAJSONAADa.",
    "t": "rip",
    "d": "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXcx",
    "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf",
    "n": "0",
```

```
    "dt": "2025-07-04T17:51:00.000000+00:00"
}
```

With respect to the event above, given that the UUID, `u` field value has sufficient cryptographic entropy, the SAID, `d` field provides a universally unique identifier for the Registry that can be referenced elsewhere. This value is derived from the Issuer AID, binding the Registry to the Issuer AID. When this Registry identifier value is included in the Registry SAID `rd` field of an ACDC, it provides secure discovery of the Registry given the ACDC by binding the Registry to the ACDC within the ACDC. This binding provides a point of correlation between the Registry and the ACDC, so it may not be appropriate for some applications. In this latter case, a presentation of the ACDC MAY attach the `rd` field value and a reference to the TEL event. The registry SAID, `rd` field value can then be used to look up the TEL event to provide the `td` field value, which is the SAID of the ACDC. Therefore, even without secure discovery, a given ACDC state can be cryptographically verified once discovery is provided.

To clarify, the registry identifier, REGID, is `ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ`. It is the value of the `d` field in the registry inception event above. It is computed as the SAID of that event. When provided in an ACDC, it is the value of the `rd` field.

The state is initialized with decorrelated placeholder values with the issuance of the following placeholder update event:

```
{
    "v": "ACDCCAACAAJSONAAEi.",
    "t": "bup",
    "d": "EPNwyvHp2XJsz9pSpXtHtcCmzw6bKSFc-nhGKTbso0Yg",
    "rd": "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ",
    "n": "1",
    "p": "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ",
    "dt": "2025-08-01T18:06:10.988921+00:00",
    "b": "ECVr7QWEp_aqVQuz4yprRFXVxJ-9uWLx_d6oDinlHU6J"
}
```

Notice in the event above that the registry SAID, `rd` field value matches the value of the SAID, `d` field in the Registry Inception, `rip` event. The value of the blinded attribute block BLID, `b` field is taken from the placeholder blinded attribute block above. Repeated here as follows:

Virtual Label	Value	Description
`d`	`ECVr7QWEp_aqVQuz4yprRFXVxJ-9uWLx_d6oDinlHU6J`	BLID (Blinding SAID)
`u`	`aG1lSjdJSNl7TiroPl67Uqzd5eFvzmr6bPlL7Lh4ukv8`	UUID salty nonce blinding factor, HD generated
`td`	`1AAP`	Transaction ACDC SAID field value, top-level `d`
`ts`	`1AAP`	Transaction state value string

Notice that the value of the BLID blinded attribute, `b` field in the transaction event, matches the value of the BLID `d` field in the expanded attribute block. Likewise, notice that the value of the transaction ACDC SAID, in the `td` field, is the CESR encoded value for empty, serving as a placeholder value. Furthermore, the value of the transaction state, `ts` field, is also the CESR encoded value for empty. This indicates that the transaction state does not yet correspond to a real ACDC. The blind for this placeholder attribute block may be updated any number of times prior to its first use as the true state of a real ACDC.

At some later time, the issuer issues a new blindable update event, `bup`, with a new blind, UUID `u` field value in the associate attribute block, derived from the shared Salt and the sequence number of the `bup` event. This makes the first use(s) of the registry uncorrelated with the eventual actual issuance of a real ACDC. In this case, the Issuer used a hierarchically deterministic algorithm to compute the UUID, `u` field in the blinded attribute block.

To do this, the Issuer and Discloser/Issuee must first have exchanged a shared secret salt from which the value of the blind, UUID, `u` field was derived. The shared secret salt MUST have approximately 128 bits of cryptographic entropy. The UUID field value is calculated using a hierarchically deterministic algorithm from the salt and the sequence number of the transaction event as the deterministic path. To preserve cryptographic entropy during derivation, the value of the UUID `u` field must be twice as long as the shared secret salt.

Suppose even later, the real ACDC is issued and is uniquely identified by its top-level SAID, `d`, field value, namely, `EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M`. The Issuer creates a new BLID blinded attribute, `b` field value with the ACDC SAID value for its `td` field with `issued` as the ACDC state for the `ts` field. The UUID, `u`, field uses

the hierarchically deterministic algorithm with the shared secret salt and the deterministic path set to the next sequence number. The Issuer issues a new blindable update event `bup` with its `b` field set to the SAID of the new blinded attribute block.

The Discloser can then download the published blinded update `bup` transaction event to get the sequence number `n` field value. With that value and the shared secret salt, the Discloser can regenerate the blind UUID and the `u` field value. The Discloser also knows the real ACDC that will be used for this Registry. Consequently, it knows that the value of the ACDC, SAID, `td` field MUST be either the empty string placeholder or the real ACDC SAID given above.

The Discloser can now compute the blinding SAID, BLID, `b` field value of the expanded Attribute block for all the combinations of the possible values for the `td` and `ts` fields. The `td` field possible values include either `1AAP` or `EMLjZLIMlfUOoKox_sDwQaJO-0wdo-GW0uNbmI28Wwc4M`. The possible `ts` field values include one of `1AAP`, `0Missued`, or `Yrevoked`. These latter three values correspond to the CESR Text domain encodings of the strings “”, “issued” and “revoked”. This gives a total of six combinations of possible field values. The Discloser tries each combination until it finds the one that matches the published transaction event blinded attribute, BLID, `b` field value. The Discloser can then verify if the published value is still a placeholder or the real initial state.

To elaborate, the value of the Issuer, `i` field of the corresponding issued ACDC will be the Issuer AID. When secure discovery is employed, the value of the registry SAID, `rd` field of that ACDC will be the registry SAID given by the value of the SAID, `d` field in the registry inception, `rip` event. The value of the top-level `d` field in the ACDC will be the same as the `td` field of the attribute block of the blindable update `bup` event that effectively “issues” the ACDC. These field values cryptographically bind the ACDC to the Registry and, in turn, bind the Registry to the ACDC.

Suppose the associated update event occurs at sequence number 2. The published blindable update transaction event is as follows:

```
{
    "v": "ACDCCAACAAJSONAAEi.",
    "t": "bup",
    "d": "EBdytzDC4dnatn-6mrCWLSGuM62LM0BgS31YnAg5NTeW",
    "rd": "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ",
    "n": "2",
    "p": "EPNwyvHp2XJsz9pSpXtHtcCmzw6bKSFc-nhGKTbso0Yg",
    "dt": "2020-08-02T12:00:20.000000+00:00",
    "b": "EOtWw6X_aoOJlkzNaLj23IC6MXHl7ZSYSWVulFW_Hr_t"
}
```

The value of the blinded attribute block BLID, `b` field, is taken from the issued blinded attribute block above. Repeated here as follows:

Virtual Label	Value	Description
`d`	`EOtWw6X_aoOJlkzNaLj23IC6MXHl7ZSYSWVulFW_Hr_t`	Dummied BLID (Blinding SAID)
`u`	`aLfCdNAnc-0P2SiruarZSajXiUWu5iU2VfQahvpNCyzB`	UUID salty nonce blinding factor HD generated
`td`	`EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M`	Transaction ACDC SAID field value, top-level `d`
`ts`	`0Missued`	Transaction state value string

Notice that the value of the blinded attribute, BLID, `b` field in the transaction event, matches the value of the BLID (Blinding SAID), `d` field in the expanded blinded attribute block. Further notice that the value of the `td` field is the same as the SAID (top-level `d`) field value of the issued ACDC. Finally, notice that in this case, the value of the transaction state, `ts` field, is `issued` (not the empty placeholder).

Suppose that the Discloser has been given the shared secret salt from which the value of the expanded attribute block's blind, UUID, `u` field was generated. The Discloser can then download the published transaction event to get the sequence number, `n` field value. With that value and the shared secret salt, the Discloser can regenerate the blind UUID, `u` field value. The Discloser also knows which ACDC it wishes to disclose, so it also has the ACDC, SAID, `d` field value. The Discloser can now compute the BLID, `b` field value of the expanded blinded attribute block for all the combinations of the possible values for the `td` and `ts` fields. For the `td` field these are either of the CESR serializations of the empty string placeholder value or the actual ACDC SAID. For the `ts` field these are one of the CESR serializations of the strings, "", "issued", or "revokes" e.g. `1AAP`, `0Missued`, or `Yrevoked`. This gives six combinations to try. The Discloser tries each one until it finds the one that matches the published transaction event blinded attribute, BLID, `b` field value. The Discloser can then disclose the matching expanded blinded block to the Disclosee, who can verify it against the published transaction event.

The Discloser can then instruct the Issuer to issue one or more updates with new blinding factors so that the initial Disclosee may no longer validate the state of the ACDC without another interactive disclosure by the Discloser.

Suppose, sometime later, a Validator requires that the Discloser provide continuing proof of issuance. In that case, the Discloser would disclose the current state of the Registry. Suppose it has been revoked. The Discloser may either refuse to disclose (with the associated consequences) or may only verifiably disclose the true state. The Discloser could continue to have the blind updated periodically. This would generate new blindable update, `bup` transaction events with new values for its blinded attribute, `b` field, but without changing either the `td` or `ts` field values. This decorrelates the time of revocation with respect to the latest event in the Registry.

At some time later, the issuer decides to revoke the issuance. The Issuer first creates a new BLID blinded attribute block with the `td` field set to the SAID of the ACDC and `revoked` as the value of the `ts` field. The UUID, `u`, field uses the hierarchically deterministic algorithm with the shared secret salt and the deterministic path set to the next sequence number, which in this case is `3`, to compute its value. The Issuer issues a new blindable update event `bup` with its `b` field set to the SAID of this new blinded attribute block.

The published blindable update transaction event is as follows:

```
{
    "v": "ACDCCAACAAJSONAAEi.",
    "t": "bup",
    "d": "EM8B1uDhWaJLfpIiEqgp-3EurGUcbfe7u2k5AarDl2XD",
    "rd": "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ",
    "n": "3",
    "p": "EBdytzDC4dnatn-6mrCWLSGuM62LM0BgS31YnAg5NTeW",
    "dt": "2020-08-03T12:00:20.000000+00:00",
    "b": "EPj3sZj8OOWTkTgAN5vzVYdANeoj3zxgEn5APb8fCRRN"
}
```

The value of the blinded attribute block BLID, `b` field, is the value of the SAID field in the associated blinded attribute block.

The serialized expanded attribute block as follows:

```
EPj3sZj8OOWTkTgAN5vzVYdANeoj3zxgEn5APb8fCRRNaGx7b16vGHVPT56tX30kY
OEzTwiVY4aabc4k9AawYyZGEMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc
4MYrevoked
```

Broken out into a table, the fields are as follows:

Virtual Label	Value	Description
`d`	`EPj3sZj8OOWTkTgAN5vzVYdANeoj3zxgEn5APb8fCRRN`	Dummied BLID (Blinding SAID)
`u`	`aGx7b16vGHVPT56tX30kYOEzTwiVY4aabc4k9AawYyZG`	UUID salty nonce blinding factor HD generated
`td`	`EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M`	Transaction ACDC SAID field value, top-level `d`
`ts`	`Yrevoked`	Transaction state value string

To unblind the Discloser would compute the attribute black and then prefix the appropriate CESR group code an attach the encoded group given below:

```
-aAjEPj3sZj8OOWTkTgAN5vzVYdANeoj3zxgEn5APb8fCRRNaGx7b16vGHVPT56tX30kYOEzTwiVY4aabc4k9AawYyZGEMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4MYrevoked
```

Public Unblinded Blindable Example

In this case, the issuer attaches the associated expanded blinded attribute block to any publication of a Blindable-Update `bup` event. As explained above, this uses either of the count codes `BlindedStateGroup` with code format `-a##` or `BigBlindedStateGroup` with code format `--a######` to attach one or more expanded blinded attribute blocks.

Public Non-Blindable State Update Registry Example

Consider a unblindable state revocation Registry for ACDCs operated in an unblinded (public) mode. The transaction state can be one of two values, `issued`, or `revoked`. The Issuer with AID, `EEDGM_DvZ9qFEAPf_FX08J3HX49ycrVvYVXe9isaP5SW` first creates one among many placeholder Registries by issuing the following registry inception `rip`, transaction event:

```
{
    "v": "ACDCCAACAAJSONAADa.",
    "t": "rip",
    "d": "EJl5EUxL23p_pqgN3IyM-pzru89Nb7NzOM8ijH644xSU",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXcz",
    "i": "EEDGM_DvZ9qFEAPf_FX08J3HX49ycrVvYVXe9isaP5SW",
```

```
    "n": "0",
    "dt": "2025-07-04T17:53:00.000000+00:00"
}
```

With respect to the event above, given that the UUID, `u` field value has sufficient cryptographic entropy, the SAID, `d` field provides a universally unique identifier for the Registry that can be referenced elsewhere, typically as the value of the `rd` field in an ACDC or as an attached reference to the presentation of an ACDC.

Sometime later, an ACDC is issued as indicated by its SAID, `d` field value, `EAU5dUws4ffM9jZjWs0QfXTnhJ1qk2u3IUhBwFVbFnt5`. The value of the Issuer, `i` field of that ACDC will be the same as the Issuer AID for the registry inception event. This binds the Issuer to both the ACDC and the registry. The SAID of the ACDC will be provided as the `td` field value in an update event. This binds the ACDC to the Registry.

When secure discovery is employed, the value of the registry SAID, `rd` field of that ACDC will be the value given by the value of the SAID, `d` field in the registry inception, `rip` event. This binds the Registry to the ACDC.

The state is initialized with the following (non-blindable) update event:

```
{
        "v": "ACDCCAACAAJSONAAEx.",
        "t": "upd",
        "d": "EJFxtbr9WioIkzTfVX4iC6Axxyg8jjKSX0ZrJgoNHiB-",
        "rd": "EJl5EUxL23p_pqgN3IyM-pzru89Nb7NzOM8ijH644xSU",
        "n": "1",
        "p": "EJl5EUxL23p_pqgN3IyM-pzru89Nb7NzOM8ijH644xSU",
        "dt": "2020-08-03T12:00:20.000000+00:00",
        "td": "EAU5dUws4ffM9jZjWs0QfXTnhJ1qk2u3IUhBwFVbFnt5",
        "ts": "issued"
}
```

Notice in the event above that the registry SAID, `rd` field value matches the value of the SAID, `d` field in the Registry Inception, `rip` event. Notice that the value of the `td` field matches the top-level SAID `d` field value of the issued ACDC (not shown). Also notice that the prior `p` field value is the same as the SAID `d` field value of the `rip` event and the sequence number `n` field is 1 more than the `rip` event sequence number. This cryptographically chain links the update event to the prior inception event as sequential events.

Sometime later, the ACDC is revoked with the publication by the Issuer of the following event:

```
{
        "v": "ACDCCAACAAJSONAAEy.",
```

```
        "t": "upd",
        "d": "EJQ-ezS6h0Oa0BIN_w4KjstdapfOfrwmVluxn1DR5Gja",
        "rd": "EJl5EUxL23p_pqgN3IyM-pzru89Nb7NzOM8ijH644xSU",
        "n": "2",
        "p": "EJFxtbr9WioIkzTfVX4iC6Axxyg8jjKSX0ZrJgoNHiB-",
        "dt": "2020-08-04T12:00:20.000000+00:00",
        "td": "EAU5dUws4ffM9jZjWs0QfXTnhJ1qk2u3IUhBwFVbFnt5",
        "ts": "revoked"
}
```

Notice that this event is chain-linked to the prior update event.

Bound Blinded Attribute Block

The bound blinded attribute block is a variant of the blinded attribute block described above. The bound blinded attribute block differs in that it adds two more fields. These two fields are labeled, respectively, the bound Issue key event sequence number, `bn`, and the bound Issuee key event SAID, `bd`. The associated Issuee key event is the key event in the KEL of the Issuee at the time of the publication of the associated blindable state update event, `bup`. Just like the blinded attribute block, the blinding said (BLID) of a bound blinded attribute block provides the value for the blinded attribute `b` field in the corresponding blindable update, `bup` event.

Essentially, the bound blinded attribute block binds to the publication time of a given registry's blindable update event, `bup`, in which the blid of the bound blinded attribute block appears to all of the following: the ACDC given by its `td` field value, the state of the ACDC given by its `ts` field value, and the key state of the ACDC's Issuee given by the combination of its `bn` and `bd` field values.

The purpose of the bound blinded attribute block is to better support chains of authority using delegated chained ACDCs. The bound Issuee key state establishes verifiably that a given Issuee in a delegation chain of ACDCs had an authorization to issue its own delegated ACDCs, which authorization had been issued but not yet been revoked at the time of anchoring its own delegated issuance in its own KEL. Similarly, other ACDC states may be bound to the Issuee key state.

The bound blinded attribute block has the following fields:

Virtual Label	Description
`d`	BLID blinding SAID
`u`	UUID salty nonce blinding factor, random or HD generated
`td`	Transaction ACDC SAID field, this is the value of the top-level `d` field in the ACDC
`ts`	Transaction state value string
`bn`	Bound Issuee key event sequence number field
`bd`	Bound Issuee key event SAID field

The fields MUST appear in the following order `[d, u, td, ts, bn, bd]`.

The field labels are virtual in that they never appear in the CESR serialization; they serve as a mnemonic for reference purposes.

The semantics of the first four fields is identical to the blinded attribute block described above and are therefore not repeated here.

The actual bound blinded attribute block is not part of the blindable update message. It MAY be provided as an attachment to some message, such as the associate ACDC, as part of a CESR serialization and MUST use one of the CESR group or count codes labeled `BoundStateSextuples` with code format `-b##` or `BigBoundStateSextuples` with code format `--b######`. When provided as an attachment, multiple blinded attribute blocks MAY be included in a given group. A group consists of its count code followed by one or more Blinded attribute blocks, where each block is composed of a concatenation of the six fields serialized in order as CESR primitives that are appropriate for the BLID, UUID, ACDC SAID, transaction state string, Issuee key state sequence number, and Issuee key state SAID fields in that block. An example of an attachment serialization with group code is provided below.

Bound Issuee Key Event Sequence Number, `bn` Field

The bound Issuee key event sequence number, `tn`, field value MUST be a CESR encoded non-negative integer. When not a placeholder, it MUST be the value of the `s` field in the current key event in the KEL of the Issuee AID's KEL at the time of publication of the associated blindable statue update, `bup`, event. This value is the sequence number of the key event. This field, in combination with the `bd` field, verifiably binds the ACDC state to the key state of the Issuee at the time the ACDC state is published.

When the bound Issuee key event sequence number, `bn`, field value is a placeholder, it is represented by the hex-encoded value of 0, i.e., “0”. It MUST be CESR-encoded as the CESR encoding for the integer value of 0.

In general, the bound Issue key state sequence number field value may be encoded using any of the codes from the following table for non-negative integers:

```
Short:   str = 'M'  # Short 2 byte b2 number
Long:    str = '0H'  # Long 4 byte b2 number
Tall:    str = 'R'  # Tall 5 byte b2 number
Big:     str = 'N'  # Big 8 byte b2 number
Large:   str = 'S'  # Large 11 byte b2 number
Great:   str = 'T'  # Great 14 byte b2 number
Huge:    str = '0A'  # Huge 16 byte b2 number (same as Salt_128)
Vast:    str = 'U'  # Vast 17 byte b2 number
```

Bound Issuee Key Event SAID, `bd` Field

When not a placeholder, the bound Issuee key event SAID, `bd`, field value MUST be the value of the `d` field in the current key event in the KEL of the Issuee AID’s KEL at the time of publication of the associated blindable statue update, `bup`, event. This value is the SAID of the key event. This field, in combination with the `bn` field, verifiably binds the ACDC state to the key state of the Issuee at the time the ACDC state is published.

When the bound Issuee key event SAID, `bd`, field value is a placeholder as indicated by the empty string, i.e., “”, it MUST be CESR encoded as the CESR `Empty` primitive code with value `1AAP`.

Bound Blinded Attribute Block Placeholder Calculation Example

Suppose the UUID, `u` field value is encoded as `aJxtoz6qVeJxPCZvP-qBJifRfIxP3itQB-VAAu7JJHxMa`, the empty placeholder transaction ACDC said, `td` field value is encoded as `1AAP` (CESR coding for the empty value), the transaction state, `ts` field is also encoded as `1AAP` for empty, the bound Issuee key event sequence number, `bn` field value is encoded as `MAAA` for 0, and the bound Issuee key event SAID, `bd` field value is encoded as `1AAP` (CESR coding for the empty value).

The digest to be used to compute the BLID (Blinding SAID) is the BLAKE3-256 digest, which, when CESR encoded, has a length of 44 characters. This is filled with forty-four `#` dummy characters. The resulting length of the group content is 104 characters.

For the sake of clarity, the field values are shown in the table below:

Virtual Label	Value	Description
`d`	`############################################`	Dummied BLID (Blinding SAID)
`u`	`aJxtoz6qVeJxPCZvP-qBJifRfIxP3itQBVAAu7JJHxMa`	UUID salty nonce blinding factor, HD generated
`td`	`1AAP`	Transaction ACDC SAID field value, top-level `d`
`ts`	`1AAP`	Transaction state value string
`bn`	`MAAA`	Bound Issuee key event sequence number field
`bd`	`1AAP`	Bound Issuee key event SAID field

The 104-character dummied serialization is as follows:

```
############################################aJxtoz6qVeJxPCZvP-qBJifRfIxP3itQBVAAu7JJHxMa1AAP1AAPMAAA1AAP
```

The CESR encoded Blake3 digest of this string is `EFaQ00QW-ZeoMxE9baWcpJbAFXrs5h0ya-wpKnHvMQ0c`. This is substituted in the serialization above for the dummy characters to provide the final serialized expanded attribute block as follows:

```
EFaQ00QW-ZeoMxE9baWcpJbAFXrs5h0ya-wpKnHvMQ0caJxtoz6qVeJxPCZvP-qBJifRfIxP3itQBVAAu7JJHxMa1AAP1AAPMAAA1AAP
```

Broken out into fields, the values are provided in the table below:

Virtual Label	Value	Description
`d`	`EFaQ00QW-ZeoMxE9baWcpJbAFXrs5h0ya-wpKnHvMQ0c`	BLID (Blinding SAID)
`u`	`aJxtoz6qVeJxPCZvP-qBJifRfIxP3itQBVAAu7JJHxMa`	UUID salty nonce blinding factor, HD generated
`td`	`1AAP`	Transaction ACDC SAID field value, top-level `d`
`ts`	`1AAP`	Transaction state value string
`bn`	`MAAA`	Bound Issuee key event sequence number field
`bd`	`1AAP`	Bound Issuee key event SAID field

In a presentation of the associated ACDC and/or TEL to a Disclosee, a discloser could attach this to verifiably unblind the blinded attribute SAID, `d` field in the associated blindable update, `bup` event.

An unblinded attachment of this blinded attribute block would be prefixed with the appropriate CESR count code, i.e., `BoundStateSextuples` with code format `-b##`, as follows:

```
-bAaEFaQ00QW-ZeoMxE9baWcpJbAFXrs5h0ya-wpKnHvMQ0caJxtoz6qVeJxPCZvP
-qBJifRfIxP3itQBVAAu7JJHxMa1AAP1AAPMAAA1AAP
```

Blinded Attribute Block ACDC State Calculation Example

For example, suppose the UUID, `u` field value is encoded as `aKNPEY4_60x6vUx2g5_5k-AoJTn0RDspR04Ql8ecNyTkO` the transaction ACDC said, `td` field value is encoded as `EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M`, the transaction state, `ts` field value is the string “issued” encoded as `0Missued`, the bound Issuee key event sequence number, `bn` field value is encoded as `MAAB` for 1, and the bound Issuee key event SAID, `bd` field value is encoded as `EJOnAKXGaSyJ_43kit0V806NNeGWS07lfjybB1UcfWsv`. The digest to be used for computing the BLID is the BLAKE3-256 digest, which, when CESR-encoded, has a length of 44 characters. This is filled with forty-four `#` dummy characters. The resulting length of the group content is 188 characters.

For the sake of clarity, the field values are shown in the table below:

Virtual Label	Value	Description
`d`	`############################################`	Dummied BLID (Blinding SAID)
`u`	`aKNPEY4_60x6vUx2g5_5kAoJTn0RDspR04Ql8ecNyTkO`	UUID salty nonce blinding factor, HD generated
`td`	`EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M`	Transaction ACDC SAID field value, top-level `d`
`ts`	`0Missued`	Transaction state value string
`bn`	`MAAB`	Bound Issuee key event sequence number field
`bd`	`EJOnAKXGaSyJ_43kit0V806NNeGWS07lfjybB1UcfWsv`	Bound Issuee key event SAID field

The 188-character dummied serialization is as follows:

```
############################################aKNPEY4_60x6vUx2g5_5kAoJTn0RDspR04Ql8ecNyTkOEMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M0MissuedMAABEJOnAKXGaSyJ_43kit0V806NNeGWS07lfjybB1UcfWsv
```

The CESR encoded Blake3 digest of this string is `EJAeKLEtVtMtt28IdAKJShyZHodEIZTHJHzaP21A_ZU4`. This is substituted in the serialization above for the dummy characters to provide the final serialized expanded attribute block as follows:

```
EJAeKLEtVtMtt28IdAKJShyZHodEIZTHJHzaP21A_ZU4aKNPEY4_60x6vUx2g5_5kAoJTn0RDspR04Ql8ecNyTkOEMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M0MissuedMAABEJOnAKXGaSyJ_43kit0V806NNeGWS07lfjybB1UcfWsv
```

Broken out into fields, the values are provided in the table below:

Virtual Label	Value	Description
`d`	`EJAeKLEtVtMtt28IdAKJShyZHodEIZTHJHzaP21A_ZU4`	BLID (Blinding SAID)
`u`	`aKNPEY4_60x6vUx2g5_5kAoJTn0RDspR04Ql8ecNyTkO`	UUID salty nonce blinding factor, HD generated
`td`	`EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M`	Transaction ACDC SAID field value, top-level `d`
`ts`	`0Missued`	Transaction state value string
`bn`	`MAAB`	Bound Issuee key event sequence number field
`bd`	`EJOnAKXGaSyJ_43kit0V806NNeGWS07lfjybB1UcfWsv`	Bound Issuee key event SAID field

In a presentation of the associated ACDC and/or TEL to a Disclosee, a discloser could attach this to verifiably unblind the blinded attribute SAID, `b` field in the associated blindable update, `bup` event.

An attachment of this blinded attribute block would be prefixed with the appropriate CESR count code, i.e., `BoundStateSextuples` with code format `-b##`, as follows:

```
-bAvEJAeKLEtVtMtt28IdAKJShyZHodEIZTHJHzaP21A_ZU4aKNPEY4_60x6vUx2g
5_5kAoJTn0RDspR04Ql8ecNyTkOEMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI2
8Wwc4M0MissuedMAABEJOnAKXGaSyJ_43kit0V806NNeGWS07lfjybB1UcfWsv
```

Bound Blinded State Registry Example

Consider a bound blindable state revocation registry for ACDCs operated in blinded (private) mode. This is identical to the blindable state revocation registry example above except that the bound blindable attribute blocks are used instead of merely blindable attribute blocks. As above, the transaction state can be one of two values, `issued` or `revoked`. In this case, the placeholder value of the empty string for the transaction state, `ts` field, is also employed to decorrelate the initialization. The Issuer with AID, `ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf` first creates one among many placeholder Registries by issuing the following transaction event:

```
{
    "v": "ACDCCAACAAJSONAADa.",
    "t": "rip",
    "d": "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXcx",
    "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf",
    "n": "0",
    "dt": "2025-07-04T17:51:00.000000+00:00"
}
```

Note that the registry identifier, REGID, is `ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ`. It is the value of the `d` field in the registry inception event above. It is computed as the SAID of that event. When provided in an ACDC, it is the value of the `rd` field.

The state is initialized with decorrelated placeholder values with the issuance of the following placeholder update event:

```
{
    "v": "ACDCCAACAAJSONAAEi.",
    "t": "bup",
    "d": "EOBVdcIL2rVEzBDpQvmBpsp3R52DsoKhTAAdvHqAz9yc",
    "rd": "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ",
    "n": "1",
    "p": "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ",
    "dt": "2025-08-01T18:06:10.988921+00:00",
    "b": "EFaQ00QW-ZeoMxE9baWcpJbAFXrs5h0ya-wpKnHvMQ0c"
}
```

Notice in the event above that the registry SAID, `rd` field value matches the value of the SAID, `d` field in the Registry Inception, `rip` event. The value of the blinded attribute block BLID, `b` field is taken from the placeholder bound blinded attribute block above. Repeated here as follows:

Virtual Label	Value	Description
`d`	`EFaQ00QW-ZeoMxE9baWcpJbAFXrs5h0ya-wpKnHvMQ0c`	BLID (Blinding SAID)
`u`	`aJxtoz6qVeJxPCZvP-qBJifRfIxP3itQBVAAu7JJHxMa`	UUID salty nonce blinding factor, HD generated
`td`	`1AAP`	Transaction ACDC SAID field value, top-level `d`
`ts`	`1AAP`	Transaction state value string
`bn`	`MAAA`	Bound Issuee key event sequence number field
`bd`	`1AAP`	Bound Issuee key event SAID field

Notice that the value of the BLID bound blinded attribute, `b` field in the transaction event, matches the value of the BLID, `d`, field in the expanded bound blinded attribute block. Likewise, notice that the value of the transaction ACDC SAID, in the `td` field, is the CESR encoded value for empty, serving as a placeholder value, as it the value of the transaction state, `ts` field, which is also the CESR encoded value for empty. Furthermore, the values of the bound Issuee key event sequence number, `bn`, and the bound Issuee key event SAID fields are the placeholder values of `MAAA` and `1AAP`. This indicates that the transaction state does not yet correspond to a real ACDC, nor does the bound Issuee key state yet correspond to an extant Issuee's key state. The blind for this placeholder attribute block may be updated any number of times prior to its first use as the true state of a real ACDC with a real Issuee.

At some later time, the issuer issues a new blindable update event, `bup`, with a new blind, UUID `u` field value in the associate attribute block, derived from the shared Salt and the sequence number of the `bup` event. This makes the first use(s) of the registry uncorrelated with the eventual actual issuance of a real ACDC. In this case, the Issuer used a hierarchically deterministic algorithm to compute the UUID, `u` field in the blinded attribute block.

Suppose even later, the real ACDC is issued and is uniquely identified by its top-level SAID, `d`, field value, namely, `EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M`. The Issuer creates a new BLID, bound blinded attribute, `b` field value, with the ACDC SAID

value for its `td` field, with `issued` as the ACDC state for the `ts` field, with the bound Issuee key event sequence number, `bn`, and key event SAID, `bd` field values taken from the Issuee AID's current key event. The UUID, `u`, field uses the hierarchically deterministic algorithm with the shared secret salt and the deterministic path set to the next sequence number. The Issuer issues a new bound blindable update event `bup` with its `b` field set to the SAID of the new bound blinded attribute block.

Suppose the associated update event occurs at sequence number 2 and the bound Issuee key event occurs at sequence number 1 in the Issuee's KEL. The published blindable update transaction event is as follows:

```
{
    "v": "ACDCCAACAAJSONAAEi.",
    "t": "bup",
    "d": "EGu4B78s6G_GVrzaoBw2a1vkFpB5tVo-wZ1OGsC9D_pK",
    "rd": "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ",
    "n": "2",
    "p": "EOBVdcIL2rVEzBDpQvmBpsp3R52DsoKhTAAdvHqAz9yc",
    "dt": "2020-08-02T12:00:20.000000+00:00",
    "b": "EJAeKLEtVtMtt28IdAKJShyZHodEIZTHJHzaP21A_ZU4"
}
```

The value of the bound blinded attribute block BLID, `b` field, is taken from the issued bound blinded attribute block above. Repeated here as follows:

Virtual Label	Value	Description
`d`	`EJAeKLEtVtMtt28IdAKJShyZHodEIZTHJHzaP21A_ZU4`	BLID (Blinding SAID)
`u`	`aKNPEY4_60x6vUx2g5_5kAoJTn0RDspR04Ql8ecNyTk0`	UUID salty nonce blinding factor, HD generated
`td`	`EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M`	Transaction ACDC SAID field value, top-level `d`
`ts`	`0Missued`	Transaction state value string
`bn`	`MAAB`	Bound Issuee key event sequence number field
`bd`	`EJOnAKXGaSyJ_43kit0V806NNeGWS07lfjybB1UcfWsv`	Bound Issuee key event SAID field

Notice that the value of the bound blinded attribute, BLID, `b` field in the transaction event, matches the value of the BLID (Blinding SAID), `d` field in the expanded bound blinded attribute block. Further notice that the value of the `td` field is the same as the SAID (top-level `d`) field value of the issued ACDC. Also, notice that in this case, the value of the transaction state, `ts` field, is `issued` (not the empty placeholder). Finally, notice, that the values of the bound Issuee key event sequence number and SAID field are not placeholders but correspond to a key event with sequence number == 1 (`s` field value), and SAID = `EJOnAKXGaSyJ_43kit0V806NNeGWS07lfjybB1UcfWsv` (`d` field value).

At some time later, the issuer decides to revoke the issuance. The Issuer first creates a new BLID, a bound blinded attribute block, with the `td` field set to the SAID of the ACDC and `revoked` as the value of the `ts` field with the bound Issuee key event sequence number, `bn`, and key event SAID, `bd` field values taken from the Issuee AID's current key event. In this case, the Issuee's key event is as sequence number == 8. The UUID, `u`, field uses a hierarchically deterministic algorithm with a shared secret salt and a deterministic path set to the next sequence number, which in this case is `3`, to compute its value. The Issuer issues a new blindable update event `bup` with its `b` field set to the SAID of this new blinded attribute block.

The published blindable update transaction event is as follows:

```
{
    "v": "ACDCCAACAAJSONAAEi.",
    "t": "bup",
    "d": "EDXGqM-V_sfj65Dk-lgMnpOZ9DYziqMaYkrju7NRKpFn",
    "rd": "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ",
    "n": "3",
    "p": "EGu4B78s6G_GVrzaoBw2a1vkFpB5tVo-wZ1OGsC9D_pK",
    "dt": "2020-08-03T12:00:20.000000+00:00",
    "b": "EIBNZ3t5rA_-PbBNmhtvtf0VgHBjVrE0fc-DO67f-wGv"
}
```

The value of the blinded attribute block BLID, `b` field, is the value of the SAID field in the associated blinded attribute block.

The serialized expanded attribute block is as follows:

```
EIBNZ3t5rA_-PbBNmhtvtf0VgHBjVrE0fc-DO67f-wGvaFudXE-d0b2owzZNBjd78
sx4kCTJx-RTP_Zd19HRUcVDEMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc
4MYrevokedMAAIEDeCPBTHAt75Acgi9PfEciHFnc1r2DKAno3s9_QIYrXk
```

Broken out into a table, the fields are as follows:

Virtual Label	Value	Description
`d`	`EIBNZ3t5rA_-PbBNmhtvtf0VgHBjVrE0fc-DO67f-wGv`	BLID (Blinding SAID)
`u`	`aFudXE-d0b2owzZNBjd78sx4kCTJx-RTP_Zd19HRUcVD`	UUID salty nonce blinding factor, HD generated
`td`	`EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M`	Transaction ACDC SAID field value, top-level `d`
`ts`	`Yrevoked`	Transaction state value string
`bn`	`MAAI`	Bound Issuee key event sequence number field
`bd`	`EDeCPBTHAt75Acgi9PfEciHFnc1r2DKAno3s9_QIYrXk`	Bound Issuee key event SAID field

To unblind the Discloser would compute the attribute black and then prefix the appropriate CESR group code, i.e., `BoundStateSextuples` with code format `-b##`, and attach the encoded group given below:

```
-bAvEIBNZ3t5rA_-PbBNmhtvtf0VgHBjVrE0fc-DO67f-wGvaFudXE-d0b2owzZNB
jd78sx4kCTJx-RTP_Zd19HRUcVDEMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI2
8Wwc4MYrevokedMAAIEDeCPBTHAt75Acgi9PfEciHFnc1r2DKAno3s9_QIYrXk
```

Annex

Performance and Scalability

The Compact Disclosure and distribute property graph fragment mechanisms in ACDC can be leveraged to enable high performance at scale. Simply using SAIDs and signed SAIDs of ACDCs in whole or in part enables compact but securely attributed and verifiable references to ACDCs to be employed anywhere performance is an issue. Only the SAID and its signature need be transmitted to verify secure attribution of the data represented by the SAID. Later receipt of the data MAY be verified against the SAID. The signature does not need to be re-verified because a signature on a SAID is making a unique (to within the cryptographic strength of the SAID) commitment to the data represented by the SAID. The actual detailed ACDC in whole or in part MAY then be cached or provided on-demand or just-in-time.

Hierarchical decomposition of data into a distributed verifiable property graph, where each ACDC is a distributed graph fragment, enables performant reuse of data or more compactly performant reuse of SAIDs and their signatures. The metadata and attribute sections of each ACDC provide a node in the graph and the Edge section of each ACDC provides the Edges to that node. Higher-up nodes in the graph with many lower-level nodes need only be transmitted, verified, and cached once per every node or leaf in the branch not redundantly re-transmitted and re-verified for each node or leaf as is the case for document-based Verifiable Credentials where the whole equivalent of the branched (graph) structure MUST be contained in one document. This truly enables the bow-tie model popularized by Ricardian Contracts, not merely for contracts, but for all data authenticated, authorized, referenced, or conveyed by ACDCs.

Cryptographic Strength and Security

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 para-

meter 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 conventional 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 show easily that 128 is a large enough N to make brute force attack computationally infeasible.

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 = $2^{\log_2 313536000}=2^{24.91}$ ~= 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.

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 [19]. A cryptosystem that has this level of security cannot be broken algorithmically even if the adversary has nearly unlimited computing power including quantum computing. The system 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 [20] or Vernum Cipher [21], and the other is Secret splitting [22], a type of secret sharing [23] that uses the same technique as an OTP.

Selective Disclosure

The verifiable credential community has defined “selective disclosure” with respect to verifiable credentials to mean disclosing some information contained in a verifiable credential in a way that does not leak other information contained in the verifiable credential. The use of the term “selective disclosure” in this context should not be confused with its application in the financial sector, particularly in relation to publicly traded companies that make selective disclosures about their operations.

ACDCs employ several different mechanisms that fall under the general description of graduated disclosure, where some information in an ACDC is disclosed while other information is not disclosed. Because there are several mechanisms that may be employed by ACDCs, different nomenclature is used to distinguish those mechanisms from one another. For example, the Attribute section refers to its graduate disclosure mechanism as “partial disclosure”. The Aggregate section, in contrast, refers to its graduated disclosure mechanism “selective disclosure”.

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 Rules 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.

The derivation the cryptographic aggregate used for Selective Disclosure 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.

As explained previously for the top-level Aggregate section, 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, inherently 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.

Tiered selective disclosure mechanisms

The ACDC chaining mechanism reduces the need for Selective Disclosure in some applications. Many non-ACDC verifiable credentials provide bundled credentials because there is no other way to associate the attributes in the bundle of credentials. 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.

Finally, in the case where the correlation of activity of an Issuee across contexts even when the ACDC used in those contexts is not correlatable may be addressed of a variant of bulk-issued ACDCs that have unique Issuee AIDs with an independent TEL Registry

per Issuee instance. This provides non-repudiable (recourse supporting) disclosure while protecting from the malicious correlation between 2nd parties and other 2nd and/or 3rd parties as to who (Issuee) is involved in a presentation.

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 [[49]]. 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 [[49]].

The Merkle tree needs to have appropriate second-pre-image attack protection of interior branch nodes [[50]] [[51]]. 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.

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 [[29]].

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) [[36]] [[37]] [[38]] [[60]]. An X25519 public key may be derived securely from an Ed25519 public key [[46]] [[60]]. Likewise, an X25519 private key may be derived securely from an Ed25519 private key [[46]] [[60]].

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 [[46]] [[45]]. The shared symmetric encryption key may be used to encrypt the secret salt or the shared symmetric encryption key itself may be used as 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 may be derived. Likewise, both compact and uncompacted versions of the ACDC then may be generated. The derivation path for the top-level UUID, `u`, field (for private ACDCs), is the string "0" and derivation path 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 of the ACDC. The Issuer also may provide references to the anchoring issuance proof seals. Everything else an Issuee (or recipient) needs to make a verifiable presentation/disclosure can be computed at the time of presentation/disclosure by the Issuee.

Bulk-issued Private ACDCs

The purpose of bulk issuance is to enable the Issuee to use ACDCs with unique SAIDs more efficiently to isolate and minimize correlation across different usage contexts. Each member of a set of bulk-issued ACDCs is essentially the same ACDC but with a unique SAID. This enables public commitments to each of the unique ACDC SAIDs without a 3rd party being able to correlate between them. A private ACDC may be effectively 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 3rd party uncorrelated copies while minimizing the associated data transfer and storage requirements involved in the issuance. Essentially, each copy (member) of a bulk-issued ACDC set shares a template that both the Issuer and Issuee use to generate on the fly 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.

The Issuer MAY make a cryptographically verifiable commitment to every member of the bulk-issued set by making a commitment to the bulk-issued aggregate value 'B' defined below. This commitment MAY be a seal in the Issuer's KEL or may be the value of the `td` field in a Registry TEL event. The commitment to the bulk aggregate `B` may be used to provide proof of issuance or proof of state of any member of the bulk-issued set. Because the aggregate value `B` could be a point of correlation to a given bulk-issued set, it should be disclosed only after Contractually Protected Disclosure is in place, i.e., no permissioned correlation. Thus, correlation would require a colluding 2nd-party who engages in unpermissioned correlation.

When a bulk-issued ACDC state is managed by a state registry, as implemented via a transaction event log (TEL), then each change of ACDC state requires anchoring a seal to the associated transaction event. Particularly, the registry SAID (REGID) is the SAID of the registry inception event. This is the value of the `rd` field in other events in the associated TEL. Update events in that TEL could in turn contain a commitment to the bulk-issued aggregate `B`.

In this case, both the Registry SAID (REGID) provided by the `rd` field value and the aggregate value `B` could be points of correlation to a given bulk-issued set; therefore, they should be disclosed only after Contractually Protected Disclosure is in place, i.e., no permissioned correlation. Thus, correlation would require a colluding 2nd-party who engages in unpermissioned correlation.

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 better control 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.

This protection may be further facilitated by not exposing the registry SAID, `rd` field at the top-level but hiding it within the Attribute `a`, or Aggregate, `A`, section field maps. This nested appearance may better facilitate the contractually protected disclosure of the registry SAID as a point of correlation.

This most basic form of bulk issuance is about protection from 3rd party correlation and protection from unpermissioned 2nd-party correlation through deterrence via contractually protected disclosure. 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 each ACDC to be more public. The SAID of a private ACDC does not expose the ACDC contents to an unpermissioned 3rd party. Unique SAIDs belonging

to bulk-issued ACDCs prevent 3rd 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 2nd 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 unpermissioned 3rd parties from making provable correlations, despite those SAIDs being public, unless they collude with malicious Verifiers (2nd parties).

In some applications, Chain Link Confidentiality is insufficient to deter unpermissioned correlation. Some Verifiers may be malicious, with sufficient incentives to overcome any counter-incentives imposed by the terms of the contractual Chain-link confidentiality. In these cases, more aggressive technological anti-correlation mechanisms such as independent TEL bulk-issued ACDCs may be useful. To elaborate, although Chain-link confidentiality terms of use may prohibit such malicious correlation, making such correlation more technically challenging may provide better protection than Chain-link confidentiality alone [44].

It is essential to note that a group of colluding malicious Verifiers may still establish a statistical correlation between presentations, despite technical barriers to cryptographically provable correlation. Cryptographically provable correlation is commonly referred to as unlinkability. This is a weak form of unlinkability as it does not prevent statistical correlation from contextual information. The latter is called contextual linkability. In general, there is no cryptographic mechanism that precludes statistical correlation among a set of colluding Verifiers, because they may make cryptographically unverifiable assertions about the information presented to them that can still be proven likely true using statistical correlation techniques. Linkability, due to the context of the disclosure itself, may defeat any unlinkability guarantees of a cryptographic technique. Thus, without contractually protected disclosure, contextual linkability, despite cryptographic unlinkability, may render the complexity of using advanced cryptographic mechanisms to provide so-called cryptographic *unlinkability* an exercise in diminishing returns.

Basic Bulk Issuance Procedure

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 [29].

As described above, there are several ways that the Issuer may share a secret salt securely. Given that the Issuer and Issuee (or recipient for Untargeted ACDCs) 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) [36][37][38]. An

X25519 public key may be securely derived from an Ed25519 public key [46][60]. Likewise, an X25519 private key may be securely derived from an Ed25519 private key [46][60].

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 [46][45]. The shared symmetric encryption key may be used to encrypt the secret salt, or the shared symmetric encryption key itself may be used as 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 for creating the Aggregate value of the set of elements in the selectively disclosable Aggregate Section, 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 can be further extended to enable bulk-issued Selective Disclosure ACDCs (i.e., those with an Aggregate Section instead of an Attribute Section) by using a similar hierarchical derivation path for the UUID field value in each of the selectively dis-

closable element blocks in its array of element attribute blocks. For example, the path `k/j` could be used to generate the UUID of aggregate element index `j` at bulk-issued ACDC index `k`.

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 and the salt.

The Issuer MUST anchor in its KEL an issuance proof digest seal of the set of bulk-issued ACDCs either directly or indirectly via an ACDC Registry (TEL). 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 a 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. This protects the Issuer, Issuees, and verifiers from impersonation fraud.

The issuance proof digest is a blinded aggregate generated from the blinded SAIDS of 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 not to enable provable correlation by a third party of the actual SAIDs in the bulk-issued set of ACDCs. Therefore, the actual SAIDs themselves must not be aggregated but only blinded commitments to those SAID. 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 unpermissioned provable way to the SAIDs of the other members.

The basic approach is to compute the blinded aggregate commitment 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 blinded 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 $_k$ 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. Blinding is performed by a digest of the concatenation of the binding factor, v_k, with the SAID, d_k.

As a side note, an XOR of the two elements could have been used. Given that the bit count of each is much greater than the number of XORed elements, then the XOR is not vulnerable to a birthday table attack [32] [31] [33]. However, in order to use an XOR, the two elements must be of the same length. Due to the cryptographic agility of CESR, different SAIDs may be of different lengths and therefore may require different-length blinding factors. Because concatenation is length-independent, it better supports crypto-agility.

The aggregation of blinded digests, `B`, is given by,

$B = H(C(b_k \text{ for all } k \text{ in } \{0, \ldots, M\text{-}1\}))$,

where `C` is the concatenation operator and `H` is the digest operator. This aggregate, `B`, provides the blinded issuance proof digest.

To summarize, the aggregate, `B`, makes a blinded cryptographic commitment to the ordered elements in the list $[b_0, b_1, \ldots, b_{M-1}]$. 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 [33][32][35] [34][48].

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, the full list of b_k elements can be disclosed safely without exposing the blinded bulk issued d_k SAID values.

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.

To protect against later forgery, in the event of a future compromise of the Issuer's signing keys, 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.

In the first case, where a bulk issued ACDC is registered using an issuance/revocation TEL, then the issuance proof seal in the KEL is the SAID of the registry inception event, `rip` for that TEL. The associated state update event in the TEL uses the aggregate value, `B`, as its `td` value. This binds the aggregate, `B`, to the issuance proof seal in the

Issuer's KEL through the TEL. A verifier can then verify the Issuer's commitment to a specific ACDC in the bulk issued set when given the registry state with field `td = B` and proof of inclusion of the ACDC SAID in `B`.

A proof of inclusion of an ACDC in a bulk-issued set requires disclosure of v_k, which could be contractually protected by only being disclosed after the Disclosee has accepted (agreed to) the terms of the rule section.

To elaborate, an ACDC that belongs to a registry would set the value of its `rd` field to the SAID of the registry inception event, `rip`. This provides a Verifier with a built-in secure discovery mechanism for the registry that governs the issuance/revocation state of a given ACDC. In a bulk-issued ACDC set, however, this registry SAID becomes a point of correlation for all ACDCs in its set. Consequently, a Discloser (Issuee) SHOULD NOT disclose the registry SAID until after contractual protection is in place, i.e., the Disclosee has agreed to the terms in the ACDC's Rule Section or not provide and `rd` field value in the ACDC at all. In this latter case, the `rd` field value MAY be attached to a presentation or provided out-of-band.

To elaborate, there are three methods that graduated disclosure of the registry SAID via the `rd` field of an ACDC can be performed to protect its disclosure until contractual protection is in place:

1. The first is that the Discloser (Issuee) uses a metadata ACDC where the top-level `rd` field is empty or missing until after contractual protection is in place.

2. The second is that the Issuer could structure the ACDC so that there is no `rd` value provided at the top level, but instead provide the `rd` field nested inside either the Attribute or Aggregate section of the ACDC. In this case, the Discloser could choose not to disclose these section details until after contractual protection is in place.

3. The third is that the Issuer of the bulk-issued set of ACDCs could choose not to provide a value for the `rd` field at all in the issued ACDC. Instead, this is communicated out-of-band to the Issuee (Discloser) who then communicates it out-of-band at the time of presentation to the Disclosee (Verifier). The advantage of this is that there is no point of correlation to the registry inside a fully disclosed variant of the ACDC. The disadvantage of this is that the Verifier has no built-in secure discovery mechanism for the registry. The Verifier can still securely verify the Issuer's commitment to any element of the bulk-issued set once provided with the unblinded inclusion proof of d_k in side `B` and a lookup of the state of the registry that uses `B` for `td`. A malicious Issuer, however, could provide more than one registry for a given set of bulk-issued ACDCs. So, duplicity of an Issuer with respect to its registry would be more difficult to detect.

Using any of the three methods, 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, $[b_0, b_1, \ldots, b_{M-1}]$ does not provide any point of correlation to the actual bulk-issued ACDC without disclosure of its v. 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 Rules section, i.e., contractual protection is in place.

Recall that the usual purpose of a TEL is to provide a verifiable data registry that enables dynamic revocation of an ACDC via a change of 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, `rd`, field is used to identify the public Registry, which usually provides a unique TEL entry for each ACDC. Each TEL entry is identified by the SAID of its inception event, `rip`. The associated ACDC is identified by its SAID, which is the value of the `td` field in the TEL update event. In the bulk issuance case, however, the state update event's `td` field value is `B`, the blinded aggregate of all the ACDCs in the set. A bulk-issued TEL SHOULD use a blinded, blindable state update, `bup` message. Otherwise, the `B` value is public and becomes a point of correlation to any third party that observes the TEL state updates. Using a blinded, `bup` update enables the Discloser (Issuee) to unblind one member of the bulk-issued set without leaking the other members of the set to 3rd parties or non-contractually protected 2nd parties.

To elaborate, using `B` as the `td` value in the blindable state update of the registry for a bulk-issued set of ACDCs enables all members of the bulk-issued set to share the same TEL without any 3rd party being able to discover which TEL any ACDC is using in an unpermissioned but cryptographically provable way. This is called 3rd party unlinkability. Moreover, a 2nd party may not discover in a permissioned way any other ACDCs from the bulk-issued set not specifically disclosed to that 2nd party. To determine to which TEL a specific bulk-issued ACDC belongs, the full inclusion proof must be disclosed, which can be contractually protected. This provides 2nd party unlinkability via contractual disincentives to link.

To further clarify, because it is blinded, the aggregate `B` is not a point of correlation to unpermissioned 3rd parties. When properly disclosed using contractually protected disclosure, the aggregate `B` is not a point of permissioned correlation between 2nd parties, but may be a point of unpermissioned correlation between 2nd parties. To remove `B` as a point of unpermissioned correlation between 2nd parties requires using independent TEL bulk-issued ACDCs described in the section so named below.

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 establishes a verifiable and binding connection between the issuance of all 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 member of a given bulk-issued set of ACDCs, at index `k`, by providing to the Disclosee four items of information that provide proof of inclusion. These are as follows:

1. The ACDC in compact form (at index `k`) where d_k is the value of its top-level SAID, `d`, field.
2. The blinding factor, v_k from which $b_k = H(v_k + d_k)$ may be computed.
3. The list of all blinded SAIDs, $[b_0, b_1, \ldots, b_{M-1}]$ that includes b_k.
4. Either a reference to the anchoring seal in the Issuer’s KEL that references the aggregate `B`, or a disclosure of the TEL state update event then references `B` and a reference to the seal in the Issuer’s KEL that references the TEL registry inception event SAID.

A Disclosee may then verify the disclosure by:

1. Computing d_j on the disclosed compact ACDC.
2. Computing $b_k = H(v_k + d_k)$.
3. Confirming that the computed b_k appears in the provided list $[b_0, b_1, \ldots, b_{M-1}]$.
4. Computing the aggregate `B` from the provided list $[b_0, b_1, \ldots, b_{M-1}]$.
5. 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 Seal that binds to a TEL registry whose state update events reference `B`. This provides proof of authorized issuance.

The last 3 steps culminate with verifying the anchoring seal, which also requires verifying the Key state of the Issuer at the time of issuance. This may require additional verification steps as per the KERI protocol.

The requirement of an anchored issuance proof seal of the aggregate `B` 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 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.

Inclusion proof via Merkle tree

The inclusion proof above of the concatenated aggregate value `B` 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 digest [49]. A Merkle tree can be much more efficient at inclusion proofs. Typically, the computation time to verify the inclusion proof is proportional to the log of the size of the Merkle tree, i.e., `O(log N)` where `N` is the size. Another beneficial property of Merkle trees is that a proof of inclusion does not disclose the other members of the tree. When the aggregate `B` is the Merkle root for a given bulk-issued set, then more efficient proofs may be provided. This enables the set to be much larger.

The Merkle tree needs to have appropriate second-pre-image attack protection of interior branch nodes [50] [51]. 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.

Independent AID Bulk Issued ACDCs

One potential point of provable but unpermissioned correlation among any group of colluding malicious Disclosees (2nd 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 (2nd parties). Thus, a common Issuee AID would be a point of correlation only for a group of colluding malicious verifiers. But in some cases, removing this unpermissioned 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. This removes any points of correlation derived from each AID and any information in that AID's KEL.

In this case, the bulk issuance process is augmented. The Issuee creates a set of unique Inception events that produce unique AIDs. One for each member of the bulk-issued set. The Issuee provides this set of unique AIDs to the Issuer who then can generate the associated ACDCs, Registries, and anchoring seals. The Issuee could use a hierarchical deterministic key chain algorithm to generate the public keys and next public key digests on the fly for each unique inception event. The path would use as a prefix the following concatenated elements: a bulk-issued path index `k/j`, where `k` is the bulk ACDC index and `j` is the key index, the bulk-issued shared secret salt, and a non-shared secret salt.

Independent Registry Bulk-Issued ACDCs

Recall that the purpose of using the aggregate `B` for a bulk-issued set from which the Registry identifier is derived is to enable a set of bulk-issued ACDCs to share a single public Registry and/or a single anchoring seal in the Issuer's KEL without enabling un-

permissioned correlation to any other members of the bulk set by virtue of the shared aggregate `B` used for either the TEL or anchoring seal in the KEL. When using a TEL as Registry, this enables the issuance/revocation/transfer state of all copies of a set of bulk-issued ACDCs to be provided by a single Registry, which minimizes the storage and compute requirements on the Registry while providing Selective Disclosure to prevent unpermissioned correlation via the public TEL Registry. When using an anchoring seal, it enables one seal to provide proof of inclusion in the bulk-issued aggregate `B`.

However, in some applications where Chain-link Confidentiality does not sufficiently deter malicious provable correlation by Disclosees (2nd party Verifiers), an Issuee may benefit from using ACDC with independent Registries (TELs) that are bulk-issued.

This is a concern when a set of bulk-issued ACDCs uses a Registry (TEL) because, despite using independent AIDs for each bulk-issued ACDC, the shared Registry identifier and shared transaction event seals could become points of correlation. In this case, the bulk issuance process MUST be augmented so that each uniquely identified copy of the ACDC with a unique Issuee AID also gets its own TEL entry. This is called *independent Registry bulk issuance*. This allows a given Issuee to use each ACDC in the set where malicious unpermissioned correlation among Validators would be problematic.

To clarify, Independent Registry bulk-issued ACDCs assume the use of independent AIDs. The combination of independent AIDs with independent Registries for a set of bulk-issued ACDCs removes points of correlation for colluding 2nd parties.

This approach can be further extended where a given ACDC is bulk-issued using a set of sets of bulk-issued ACDCs, where each set in the set of sets gets its own Registry with its own aggregate `B` for the ACDCs in its set. This extension is called *modular independent Registry bulk issuance*. Essentially, the total number of Registries may be fewer, as the total number of bulk-issued ACDCs is divided into smaller, more contextualized sets, as opposed to a single large bulk-issued set. This allows a given Issuee to use each set in the set of sets in a given context where malicious unpermissioned correlation among Validators using the same set is not problematic, while using a different set when malicious unpermissioned correlation among Validators between the sets would be problematic.

In either case, each Disclosee (Verifier) of a full presentation/disclosure of a given copy of the ACDC in a given context only receives proof of one uniquely identified TEL and cannot provably correlate the TEL state of one presentation to any other presentation in a given context because the Registry identifier for each given ACDC SAID or, when modularized, the aggregate `B` is different when used in another context. 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 unpermissioned 2nd and 3rd party exploitation via provable correlation of TEL identifiers even with colluding 2nd party Verifiers. They are limited to statistical correlation techniques.

To summarize the main benefit of this approach, despite its relatively larger storage and computational burden, is that in some applications, Chain-link Confidentiality does not sufficiently deter provable correlation due to unpermissioned malicious collusion. Therefore, completely independent bulk-issued ACDCs may be beneficial. Nonetheless, given the ease with which statistical methods can be used to correlate around any such protections against unprovable correlation (i.e., unlinkability), including independent Registry bulk-issued ACDCs, the added benefit may not be worth the added complexity and effort.

Independent Registry Transaction Event Seals Using Merkle Tree Roots

When using a TEL as an ACDC state Registry for a bulk-issued set, each event in the TEL MUST be bound with an anchoring seal in the Issuer's KEL. When using *independent registry bulk-issued ACDCs*, the total number of registries for a given set of bulk-issued ACDCs is equal to the number of members of the set. KERI key events seal types include a Merkle root seal or a typed seal. In either case, instead of a list of seals, one for each member of the bulk-issued set, all the seals could be anchored using a single seal, where that seal is the Merkle root of a Merkle tree that references the transaction events for all the independent Registry TELs. As mentioned above, the inclusion proof for a Merkle tree is much more efficient than the inclusion proof using a blinded list. But that is not the only advantage of using a Merkle tree root for anchoring the transaction event seals. Another useful property of Merkle trees is that one member of the tree may be disclosed via an inclusion proof without disclosing other members of the tree. This is especially true for sparse Merkle trees where the members are only stored in leaf nodes of the tree, not in intermediate nodes [61]. Sparse Merkle trees (SMT) were originally developed for the certificate transparency (CT) network [62][63][64][65]. Since then various optimized versions of SMT have been developed [66][67][68].

The efficient inclusion proofs of an SMT enable a given Issuer to amalgamate all Registry transaction event seals for all its bulk-issued ACDCs into a single SMT. This provides so-called *herd privacy* to the updates. A given seal in the KEL of the Issuer no longer provides a point of correlation to any other transaction event to any given registry. A given inclusion proof for one event from one Registry does not reveal inclusion for events in other Registries or correlate a given bulk-issued ACDC to any other bulk-issued ACDC using a different Registry. This amalgamation requires only one seal in the Issuer's KEL.

Using an optimized SMT algorithm enables an Issuer to provide efficient inclusion proofs for an amalgamated SMT of all transaction update events across all Registries for all bulk-issued ACDCs to all Issuees.

Essentially, the Issuer maintains a single SMT that includes all the event SAIDs from all transaction events across the Issuer's Registries. The values stored in the leaves of the tree are the SAIDs of each transaction event across all the Registries for all Issuees. When using blindable state update events in these Registries, then the ACDC is not viewable by a 3rd party. Each of these SAIDs is computed over a high-entropy UUID (salty Nonce). Therefore, they are not guessable. They can only be discovered when disclosed. Periodically, the Issuer anchors a bulk seal that is the current SMT root. This captures all events from all registries that have been updated since the last anchor. The Issuer's TEL Registrar also publishes to TEL Observers the incremental update to the SMT that resulted in the new SMT root. The Observer ban then efficiently maintains a verifiably synchronized copy of the SMT. A Validator can then query its TEL Observer for an inclusion proof of a TEL event at a point of validation (PoV) without the Registrar being able to track the PoV.

To provide proof of ACDC state, a Validator can request the latest transaction event update from its TEL Observer for a specific Registry (TEL) from a bulk-issued set as disclosed to it by a presentation from the Issuee. Only the Issuee and the Issuer know that the Registry comes from a bulk-issued set. Then, an inclusion proof for that event's SAID can be obtained from the Observer's SMT. This inclusion proof enables the Validator to verify that the Issuer committed to that event. This process does not provide a point of correlation with other Registries in the same bulk-issued set because the latest SMT root anchoring seal captures all transaction update events from all other bulk-issued sets that occurred within the same batch period. This provides herd privacy. The Issuee can then unblind the transaction event and disclose the associated ACDC.

To further strengthen herd privacy, when using blindable state Registries, the Issuer could randomly issue update events that do not change state, across all its registries, mixing updates over time and across each registry. This would ensure that enough updates are captured by each batch anchoring seal to guarantee a pre-specified level of herd privacy.

To elaborate, using blindable state update events in a Registry, the SAID of the ACDC is not guessable by any party and therefore cannot be linked to its Registry SAID (TEL) or to the SAID of any transaction event in that Registry (TEL). It needs to be disclosed to that party. To clarify, a party can't guess the ACDC SAID that corresponds to its corresponding Registry identifier (SAID) or transaction event SAID; it would first have to be disclosed to them. The disclosure to that 2nd party does not contain within itself any provable points of correlation to other ACDCs from the same bulk issued set.

Likewise, because the SMT provides herd privacy with respect to which Registries with their transaction event updates belong to which bulk-issued sets of ACDCS, a pair of malicious 2nd parties cannot compare inclusion proofs to determine if two ACDCs belong to the same bulk-issued set. Moreover, because each ACDC in a bulk-issued set gets its own Registry, the registry database does not provide a point of correlation between two different Registries that belong to the same bulk-issued set of ACDCs. Therefore, a given disclosure of a bulk-issued ACDC does not directly disclose any points of correlation between malicious 2nd parties.

The Merkle tree needs to have appropriate second-pre-image attack protection of interior branch nodes [50] [51]. 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.

Extensibility

The ACDC design leverages append-only verifiable data structures, named KELs and TELs, that have strong security properties that simplify end-verifiability and foster decentralization. Append-only verifiable data structures provide permission-less extensibility by Issuers, presenters, and/or Verifiers. By permission-less, it means that there is no shared governance over these data structures. This is completely decentralized and zero-trust. The fact that each ACDC has both a universally unique content-based identifier and a universally unique content-based Schema identifier that determines its type enables permission-less ACDC type Registries. Because Attributes, claims, and properties may be conveyed by verifiable graphs of ACDCs linked by their Edges, whose Edges may also have properties, ACDCs may be modeled as labeled property graphs. This enables extensibility of pre-existing already Issued ACDCs by appending additional attributes via application-specific ACDCs or bespoke ACDCs. In other words, custom Attributes or properties are appended via chaining using one or more custom ACDCs defined by a custom Schema (type-is-schema), without modifying pre-issued credential types in place.

In essence, an ACDC is essentially a verifiable property graph fragment of an extensible distributed property graph, where each node may be sourced by a different Issuer. This type of extensibility means there is no need for centralized permissioned name-space Registries to resolve name-space collisions of Attributes or properties. Each ACDC has a universally unique content address and a universally unique content addressable Schema (type-is-schema) as an ACDC type that defines the namespace. The function of an extensible Registry of ACDC types now becomes merely Schema discovery or schema blessing for a given context or ecosystem. The reach of such a Registry of ACDC types can be tailored by ecosystem participants to their desired level of interoperability. Versioning is also simplified because Edges are still verifiable as long as the new Schema version is backward compatible.

ACDC Protocol Message Types

CESR support for the ACDC protocol includes conveying sections of an ACDC as CESR-compatible messages (packets) with their own message types by section. These section messages can be part of a CESR stream or part of the attachment group to an ACDC. This is useful for sending the ACDC in its compact form and then attaching the section details as individual section message packets. This enables one to cache sections so that they do not have to be transmitted repeatedly or reuse sections that are the same for multiple ACDC instances, which is often the case for the schema and rule sections. CESR native ACDC messages may appear as either field maps or field fields at the top level. The difference is determined by the top-level universal count (group) code for the message. The combination of top-level count (group) code and message type determines the rules for the presence of fields and field labels.

The following section details the ACDC message types, including section message types.

Message Type Table

Ilk	Name	Description
		Registry TEL Message Types
rip	Registry Inception	Initialize blindable state ACDC Registry
upd	Update	Update transaction state of blindable state ACDC Registry
		ACDC Message
	ACDC	Top-level Field Map without Message type (ilk), `t` field, implied type is `acm`
acm	ACDC	Top-level Field Map with Message type (ilk), `t` field
act	ACDC	Top-level Fixed field with Attribute section and Message type (ilk), `t` field
acg	ACDC	Top-level Fixed field with Aggregate section and Message type (ilk), `t` field
		ACDC Section Message types
sch	Schema	Schema section Message
att	Attribute	Attribute section Message
agg	Aggregate	Aggregate attribute section Message
edg	Edge	Edge section Message

Ilk	Name	Description
rul	Rule	Rules section Message

Message Type Field

The presence of the message type field, labeled `t`, is optional for messages of type `acm` with non-CESR-native serialization kinds. It MUST be present in any `acm` message that uses a CESR serialization kind, namely JSON, CBOR, and MGPK,. It MUST be present in messages with any other message type, regardless of the kind of serialization. A message without a message type, `acm`, field is assumed to be of type `acm`. Therefore, the `acm` message type is effectively the default message type for ACDC protocol messages. The protocol type in either the version string of non-CESR-native serialization kinds or the version field for native CESR serialization kind MUST be `ACDC`. To elaborate, the message type field MUST appear in all native CESR messages; it MAY only not appear in non-CESR-native serialization kinds of `acm` type messages.

ACDC as a top-level field map in CESR native format

When an ACDC message of type `acm` in CESR native format, i.e. the serialization kind is `CESR`, appears as a top-level field map, it MUST use either of the CESR count codes, `-G##` or `--G#####` at the top-level. The top-level fields (labels and values) that appear MUST appear in the following order: `[ v, t, d, u, i, rd, s, a, A, e, r]`. The required fields for `acm` messages are `[v, t, d, i, s]`. The rules for the appearance of optional fields are the same as those defined above for the other serialization kinds. The Version field value is a CESR primitive that provides the protocol type and the version. It does not provide a serialization kind or length. This is already indicated by the count codes, `-G##` or `--G#####`.

Likewise, for all the other ACDC protocol message types, when the serialization kind is `CESR`, i.e. is in a native CESR message format then the appearance of top levels fields with optional fields is as described above for those specific messages.

ACDC as a top-level set of fixed fields in CESR native format

When an ACDC message of type `acm` in CESR native format, i.e. the serialization kind is `CESR`, appears as a top-level set of fixed fields, it MUST use either of the CESR count codes, `-F##` or `--F#####` at the top-level. The top-level field values (no labels) MUST appear in the following order: `[ v, t, d, u, i, rd, s, a, A, e, r]`. All fields are required but may have empty values. The value of either or both the `a` and `A` field MUST be empty. To clarify, both the `a` and `A` field values MUST not be non-empty, one or the other or both MUST be empty. Emptiness for field values that allow a field map count code as a value is indicated by a generic map count code with zero-length contents. Emptiness for field values that allow a list field count code as a value is indicated by a

generic list count code with zero-length contents. Emptiness for field values that require a CESR primitive is indicated by the `Null` CESR primitive code, `1AAK`. The Version field value is a CESR primitive that provides the protocol type and the version. It does not provide a serialization kind or length. This is already indicated by the count codes, `-F##` or `--F#####`.

Likewise, for all the other ACDC protocol message types, when the serialization kind is `CESR`, i.e. is in a native CESR message format then the appearance of top levels fields is required. There are no optional fields. Emptiness for field values that allow a field map count code as a value is indicated by a generic map count code with zero-length contents. Emptiness for field values that allow a list field count code as a value is indicated by a generic list count code with zero-length contents. Emptiness for field values that require a CESR primitive is indicated by the `Null` CESR primitive code, `1AAK`.

ACDC Message Fields

An ACDC can be represented internally, in computer memory, as a dictionary or hash map or equivalent data structure with labeled fields. We call this abstractly, a field map. One important feature of the fields maps used by ACDC is that they all MUST include a field with a SAID (self-addressing ID) using the SAID protocol SAID . We call a field map with a SAID field a self-addressed data structure or self-addressed dict. This is abbreviated with the acronym, SAD. To clarify, the SAD of an ACDC is a labeled field map which includes whose value is the SAID of that SAD. Field maps may be implemented differently in different computer software languages. For example, in Python it may be a `dict`. In Javascript it may be an object or a map. In contrast, The over-the-wire serialization could be native CESR using either a field map or fixed fields at the top level.

The following table defines the top-level fields in an ACDC and their order of appearance. For some message types, some fields are optional, but all fields that appear MUST appear in this order, `[v, t, d, u, i, s, a, A, e, r]`.

Label	Title	Description
`v`	Version String	Regexable format: `ACDCMmmGggKKKKSSSS.` that provides protocol type, version, CESR genus version, serialization type, size, and terminator.
`t`	Message Type	Three-character Message type
`d`	Message Digest SAID	Self-referential fully qualified cryptographic digest of enclosing map.
`u`	UUID	Random Universally Unique Identifier as fully qualified high entropy pseudo-random string, a salty nonce.
`i`	Issuer Identifier AID	AID whose control authority is established via KERI verifiable Key State.
`rd`	Registry Digest SAID	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.

Label	Title	Description
`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.

Message Type Field

The presence of the message type field, labeled `t`, is optional for messages of type `acm` with non-CESR-native serialization kinds. It MUST be present in any `acm` message that uses a CESR serialization kind, namely JSON, CBOR, and MGPK,. It MUST be present in messages with any other message type, regardless of the kind of serialization. A message without a message type, `acm`, field is assumed to be of type `acm`. Therefore, the `acm` message type is effectively the default message type for ACDC protocol messages. The protocol type in either the version string of non-CESR-native serialization kinds or the version field for native CESR serialization kind MUST be `ACDC`. To elaborate, the message type field MUST appear in all native CESR messages; it MAY only not appear in non-CESR-native serialization kinds of `acm` type messages.

ACDC of type `acm` as a top-level field map in CESR native format

When an ACDC message of type `acm` appears in CESR native format, i.e., the serialization kind is `CESR`. It MUST appear as a top-level field map that MUST use either of the CESR count codes, `-G##` or `--G#####` at the top-level. The type, `acm`, is a mnemonic for “ACdc field Map”. The top-level fields (labels and values) that appear MUST appear in the following order: `[ v, t, d, u, i, rd, s, a, A, e, r]`. The required fields for `acm` messages are `[v, d, i, s]`. The optional fields are: `[t, u, rd, a, A, e, r]`. Fields `[a, A]` are alternates. When field `a` appears then field `A` MUST NOT appear. Likewise, when field `A` appears, then field `a` MUST NOT appear. The Version field value is a CESR primitive that provides the protocol type, the protocol version number, and the CESR genus version number. It does not provide a serialization kind or length. This is already indicated by the count codes, `-G##` or `--G#####`.

Whenever the message type field does not appear in the field map for a message whose protocol type is `ACDC`, the message type `acm` is inferred.

ACDC of type `act` as a top-level set of fixed fields in CESR native format

An ACDC message of message type `act` in CESR native format, i.e., the serialization kind is `CESR`, MUST appear as a top-level set of fixed fields. Therefore, it MUST use either of the CESR count codes, `-F##` or `--F#####` at the top-level. The type, `act`, is a

mnemonic for “ACdc fixed field with aTtribute section”. The top-level field values (no labels) MUST appear in the following order: `[ v, t, d, u, i, rd, s, a, e, r]`. All fields are required, but the following fields, `[u, rd, a, e, r]` may have empty values. Emptiness for field values with labels `u` or `rd` is indicated by an empty string. In CESR, an empty string primitive is encoded as a variable-length bytes string primitive with zero length given by the primitive encoding `4BAA`. Emptiness for field values with labels `a`, `e`, or `r` is indicated by an empty field map. In CESR, an empty field map is encoded as a generic map group with empty contents given by the group code `-IAA`.

The Version field value is a CESR primitive that provides the protocol type and the version. It does not provide a serialization kind or length. This is already indicated by the count codes, `-F##` or `--F#####`.

ACDC of type `acg` as a top-level set of fixed fields in CESR native format

An ACDC message of message type `acg` in CESR native format, i.e. the serialization kind is `CESR`, MUST appear as a top-level set of fixed fields. Therefore, it MUST use either of the CESR count codes, `-F##` or `--F#####` at the top-level. The type, `actg`, is a mnemonic for “ACdc fixed field with aGgregate section”. The top-level field values (no labels) MUST appear in the following order: `[ v, t, d, u, i, rd, s, A, e, r]`. All fields are required, but the following fields, `[u, rd, a, e, r]` may have empty values. Emptiness for field values with labels `u` or `rd` is indicated by an empty string. In CESR, an empty string primitive is encoded as a variable-length bytes string primitive with zero length given by the primitive encoding `4BAA`. Emptiness for the field value of the field labeled `A` is indicated by an empty list. In CESR, an empty list is encoded as a generic list group with empty contents given by the group code `-JAA`. Emptiness for field values with labels `e` or `r` is indicated by an empty field map. In CESR, an empty field map is encoded as a generic map group with empty contents given by the group code `-IAA`.

The Version field value is a CESR primitive that provides the protocol type and the version. It does not provide a serialization kind or length. This is already indicated by the count codes, `-F##` or `--F#####`.

Compact Private ACDC with top-level field map of type `act` in CESR native format

Shown below is the labeled SAD as a Python dict as the internal representation (not over-the-wire).

The SAD of an ACDC is a labeled field map, such as an object in Javascript, or a dict in Python. The over-the-wire serialization could be native CESR using either a field map or fixed fields at the top level. The fixed field CESR native format could be especially compact. Shown below is the labeled SAD as a Python dict (not the over-the-wire JSON or CESR). The Message type, `t` field for ACDCs is an optional field for JSON, CBOR, MessagePack, and CESR field maps but is required for CESR fixed fields. This enables more than one type of CESR fixed field top-level ACDC CESR serialization that is unam-

biguously parseable. This seems to violate the schema-is-type convention in order to enable a parser to correctly parse a fixed field Message type. The message group count code determines if the ACDC is fixed field or a field map.

Python dict of compact ACDC with message type, `t` field.

```
{
  "v":  "ACDCCAAJSONAACD.",
  "t":  "act",
  "d":  "EBWNHdSXCJnFJL5OuQPyM5K0neuniccMBdXt3gIXOf2B",
  "u":  "0AHcgNghkDaG7OY1wjaDAE0q",
  "i":  "EAqjsKFk66jpf3uFv7An2EDIPMvklXKhmkPreYpZfzBr",
  "rd": "EMwsxUelUauaXtMxTfPAMPAI6FkekwlOjkggtymRy7x",
  "s":  "EAXRZOkogZ2A46jrVPTzlSkUPqGGeIZ8a8FWS7a6s4re",
  "a":  "EFrn9y2PYgveY4-9XgOcLxUderzwLIr9Bf7V_NHwY1lk",
  "e":  "ECdoFOLe71iheqcywJcnjtJtQIYPvAu6DZIl3MOARH3d",
  "r":  "EH3dCdoFOLBe71iheqcywJcnjtJtQIYPvAu6DZIl3MOR"
}
```

Compact Private ACDC with top-level field map in CESR native format

For clarity, the first column provides the equivalent label value for the other serialization kinds (JSON, CBOR, MGPK). The actual label is the CESR-encoded label in the second column.

Field Label	Label Value	Field or Count Value	Description
NA	NA	`-F##` or `-0F#####`	Count code for CESR native top-level fixed field signable message
`v`	`0J_v`	`0OACDCCAACAA`	Protocol Version primitive, ACDC 2.0 CESR 2.0
`t`	`0J_t`	`Xact`	Message type primitive
`d`	`0J_d`	`EGgbiglDXNE0GC4NQq-hiB5xhHKXBxkiojgBabiu_JCk`	SAID of ACDC packet
`u`	`0J_u`	`0AGC4NQq-hiB5xhHKXBxkiK`	UUID (salt) of ACDC packet
`i`	`0J_i`	`EBabiu_JCkE0GbiglDXNB5C4NQq-hiGgxhHKXBxkiojg`	AID of issuer of ACDC
`rd`	`0Krd`	`ECkE0GbiglDXNB5C4NQq-hiGgxhHKXBxkiojgBabiu_J`	SAID of revocation registry for ACDC
`s`	`0J_s`	`EDXNB5C4NQq-hiGgxhHKXBxkiojgBabiu_JCkE0Gbigl`	SAID of schema section of ACDC packet
`a`	`0J_a`	`EC4NQq-hiGgbiglDXNB5xhHKXBxkiojgBabiu_JCkE0G`	SAID of attribute section of ACDC packet

Field Label	Label Value	Field or Count Value	Description
`e`	`0J_e`	`EFXBxkiojgBabiu_JCkE0GC4NQq-hiGgbiglDXNB5xhH`	SAID of edge section of ACDC packet
`r`	`0J_r`	`EMiGgbiglDXNB5xhHFXBxkiojgBabiu_JCkE0GC4NQq_`	SAID of rule section of ACDC packet

Section Message Types

The section messages are meant to provide a way to send the exposed sections independently of the associated ACDC message. To elaborate, an ACDC itself has a message type of either `acm`, `act` or `acg`. Without loss of specifity, when referring to an ACDC message without specifying the message type, then one of the three types, `acm`, `act`, or `acg` is implied. Otherwise, the message type is specified. Whereas, in contradistinction, the ACDC section messages are always referred to as an “ACDC section message”, not merely an “ACDC message”, unless the message type of the section message is specified.

All designated fields are required in ACDC section messages. There are no optional fields. However, field values may be empty. Emptiness for field values that MUST have a string value is indicated by an empty string. In CESR, an empty string primitive is encoded as a variable-length bytes string primitive with zero length given by the primitive encoding `4BAA`. Emptiness for field values that MAY accept a field map is indicated by an empty field map. In CESR, an empty field map is encoded as a generic map group with empty contents given by the group code `-IAA`. Emptiness for field values that MAY accept a list is indicated by an empty list. In CESR, an empty list is encoded as a generic list group with empty contents given by the group code `-JAA`.

Section Message top-level fields

Label	Title	Description
`v`	Version String	Regexable format: `ACDCMmmGggKKKKSSSS.` that provides protocol type, version, CESR genus version, serialization type, size, and terminator
`t`	Message Type	Three-character Message type
`d`	Digest (SAID)	Self-referential fully qualified cryptographic digest of enclosing map.
`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 (effective SAID) 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.

Each section Message MUST have Version String, `v`, Message type, `t`, and SAID, `d` fields in that order. The value of the SAID, `d` field is the said of the Section Message itself not the SAID of the embedded section field value.

The remaining top-level field in each section message is the appropriate section field for the Message type, as follows:

- Schema, `s` field for the Schema, `sch` Message
- Attribute, `a` field for Atttribute, `att` Message
- Attribute aggregate, `A` field for the aggregate, `agg` Message
- Edge, `e` field for the Edge, `edg` Message
- Rule, `r` field for the Rule, `rul` Message
-

The embedded section block's SAID, `d` field in the respective section top-level field MUST match the section field value in the associated ACDC when in most compact form. This SAID is computed using the most compact SAID algorithm. To clarify, the said of the message itself as given by the top-level said, `d` field value is not computed using the most compact SAID algorithm, but the embedded said, `d` field of the top-level section field is computed using the most compact SAID algorithm.

For the following Messages, except for the Attribute aggregate variant, assume that the associated compact ACDC is as follows: Compact form with Attribute section:

```
{
  "v":  "ACDCCAAJSONAACD.",
  "t":  "acm",
  "d":  "EBWNHdSXCJnFJL5OuQPyM5K0neuniccMBdXt3gIXOf2B",
  "u":  "0AHcgNghkDaG7OY1wjaDAE0q",
  "i":  "EAqjsKFk66jpf3uFv7An2EDIPMvklXKhmkPreYpZfzBr",
  "rd": "EMwsxUelUauaXtMxTfPAMPAI6FkekwlOjkggtymRy7x",
  "s":  "EBdXt3gIXOf2BBWNHdSXCJnFJL5OuQPyM5K0neuniccM",
  "a":  "EFrn9y2PYgveY4-9XgOcLxUderzwLIr9Bf7V_NHwY1lk",
  "e":  "ECdoFOLe71iheqcywJcnjtJtQIYPvAu6DZIl3MOARH3d",
  "r":  "EH3dCdoFOLBe71iheqcywJcnjtJtQIYPvAu6DZIl3MOR"
}
```

For the Attribute aggregate variant below, assume that the associated compact ACDC is as follows:

Compact form with Attribute aggregate section:

```
{
  "v":  "ACDCCAAJSONAACD.",
  "t":  "acm",
  "d":  "EBWNHdSXCJnFJL5OuQPyM5K0neuniccMBdXt3gIXOf2B",
  "u":  "0AHcgNghkDaG7OY1wjaDAE0q",
  "i":  "EAqjsKFk66jpf3uFv7An2EDIPMvklXKhmkPreYpZfzBr",
  "rd": "EAMPAI6FkekwlOjkggtymRy7xMwsxUelUauaXtMxTfP",
  "s":  "EAXRZOkogZ2A46jrVPTzlSkUPqGGeIZ8a8FWS7a6s4re",
```

```
  "A":  "ELIr9Bf7V_NHwY1lkFrn9y2PYgveY4-9XgOcLxUderzw",
  "e":  "ECdoFOLe71iheqcywJcnjtJtQIYPvAu6DZIl3MOARH3d",
  "r":  "EH3dCdoFOLBe71iheqcywJcnjtJtQIYPvAu6DZIl3MOR"
}
```

Schema section Message

In a Schema Section Message, the Schema, `s` field value is the expanded Schema section from the associated ACDC. Notice that the value of the `$id` field within the Schema subblock matches the value of the Schema, `s` field in the associated ACDC above.

```
{
  "v": "ACDCCAAJSONAACD.",
  "t": "sch",
  "d":  "EBWNHdSXCJnFJL5OuQPyM5K0neuniccMBdXt3gIXOf2B",
  "s":
  {
    "$id": "EBdXt3gIXOf2BBWNHdSXCJnFJL5OuQPyM5K0neuniccM",
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "title": "Compact Public ACDC",
    "description": "Example JSON Schema for a Compact private
ACDC.",
    "credentialType": "CompactPublicACDCExample",
    "version": "1.0.0",
    "type": "object",
    "required":
    [
      "v",
      "d",
      "u",
      "i",
      "rd",
      "s",
      "a",
      "e",
      "r"
    ],
    "properties":
    {
      "v":
      {
        "description": "ACDC version string",
        "type": "string"
      },
      "d":
      {
        "description": "ACDC SAID",
        "type": "string"
      },
```

```
      "i":
      {
        "description": "Issuer AID",
        "type": "string"
      },
      "rd":
      {
        "description": "Registry SAID",
        "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
  }
}
```

Attribute Section Message

In a Attribute Section Message, the Attribute, `a` field value is the expanded Attribute section from the associated ACDC. Notice that the value of the `d` field within the Attribute subblock matches the value of the Attribute, `a` field in the associated ACDC above.

```
{
  "v": "ACDCCAAJSONAACD.",
  "t": "att",
  "d":  "EBWNHdSXCJnFJL5OuQPyM5K0neuniccMBdXt3gIXOf2B",
  "a":
  {
    "d": "EFrn9y2PYgveY4-9XgOcLxUderzwLIr9Bf7V_NHwY1lk",
    "i": "EKAn2EDIPmkPreYApZfFk66jpf3uFv7vklXKhzBrAqjs",
```

```
    "temp": 45,
    "lat": "N40.3433",
    "lon": "W111.7208"
  }
}
```

Aggregated Attribute Section Message

In an Aggregate Section Message, the Aggregate, `A` field value is the expanded attribute aggregate section from the associated ACDC. Notice that the value of the attribute Aggregate, `A` field in the associated ACDC above, is computed as the digest of concatenated digests of the elements of the selectively disclosable array (see Annex on Selective Disclosure).

```
{
  "v": "ACDCCAAJSONAACD.",
  "t": "agg",
  "d":  "EBWNHdSXCJnFJL5OuQPyM5K0neuniccMBdXt3gIXOf2B",
  "A":
  [
    {
      "d": "ErzwLIr9Bf7V_NHwY1lkFrn9y2PYgveY4-9XgOcLxUde",
      "u": "0AqHcgNghkDaG7OY1wjaDAE0",
      "i": "EpZfFk66jpf3uFv7vklXKhzBrAqjsKAn2EDIPmkPreYA"
    },
    {
      "d": "ELIr9Bf7V_NHwY1lkgveY4-Frn9y2PY9XgOcLxUderzw",
      "u": "0AG7OY1wjaDAE0qHcgNghkDa",
      "score": 96
    },
    {
      "d": "E9XgOcLxUderzwLIr9Bf7V_NHwY1lkFrn9y2PYgveY4-",
      "u": "0AghkDaG7OY1wjaDAE0qHcgN",
      "name": "Jane Doe"
    }
  ]
}
```

Edge Section Message

In an Edge Section Message, the Edge, `e` field value is the expanded Edge section from the associated ACDC. Notice that the value of the `d` field within the Edge subblock matches the value of the Edge, `e` field in the associated ACDC above.

```
{
  "v": "ACDCCAAJSONAACD.",
  "t": "edg",
  "d":  "EBWNHdSXCJnFJL5OuQPyM5K0neuniccMBdXt3gIXOf2B",
```

```
  "e":
  {
    "d": "ECdoFOLe71iheqcywJcnjtJtQIYPvAu6DZIl3MOARH3d",
    "poe": "ECJnFJL5OuQPyM5K0neuniccMBdXt3gIXOf2BBWNHdSX",
    "poa":
    {
      "o": "OR",
      "sewer": "EK0neuniccMBdXt3gIXOf2BBWNHdSXCJnFJL5OuQPyM5",
      "gas": "EBWNHdSXCJnFJL5OuQPyM5K0neuniccMBdXt3gIXOf2B"
    }
  }
}
```

Rule Section Message

In a Rule Section Message, the Rule, `r` field value is the expanded Rules section from the associated ACDC. Notice that the value of the `d` field within the Rule subblock matches the value of the Rule, `r` field in the associated ACDC above.

```
{
  "v": "ACDCCAAJSONAACD.",
  "t": "rul",
  "d":  "EBWNHdSXCJnFJL5OuQPyM5K0neuniccMBdXt3gIXOf2B",
  "r":
  {
    "d": "EH3dCdoFOLBe71iheqcywJcnjtJtQIYPvAu6DZIl3MOR",
    "disclaimers":
    {
      "l": "The person or legal entity identified by this ACDC's
Issuer AID (Issuer) makes the following disclaimers:",
      "warrantyDisclaimer": "Issuer provides this ACDC on an \"AS
IS\" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
express or implied, including, without limitation, any warranties
or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or
FITNESS FOR A PARTICULAR PURPOSE",
      "liabilityDisclaimer": "In no event and under no legal
theory, whether in tort (including negligence), contract, or
otherwise, unless required by applicable law (such as deliberate
and grossly negligent acts) or agreed to in writing, shall the
Issuer be liable for damages, including any direct, indirect,
special, incidental, or consequential damages of any character
arising as a result of this credential. "
    },
    "permittedUse": "The person or legal entity identified by the
ACDC's Issuee AID (Issuee) agrees to only use the information
contained in this ACDC for non-commercial purposes."
  }
}
```

Working ACDC Examples

Working Examples Setup

The examples were created with the keripy library. The code to generate the examples is provided via unit tests. These may be found in tests/spec/acdc.

A brief explanation of the setup code is provided here to help implementers who wish to reproduce the examples from scratch.

AIDs

The examples require an Issuer AID. This AID is created in accordance with the KERI protocol. This requires creating digital signing key-pairs whose public keys are used in an inception event to create the AID. The keripy library has a utility class Salter (found in keri.core.signing.Salter) that facilitates the creation of signing key pairs. For the examples, any needed signing key pairs can be recreated using a known non-random salt. The known non-random salt is merely for reproducibility. In a real-world application, the salt should be a high entropy secret. The salt used for the examples is the python byte string

```
b'acdcspecworkexam'
```

Using a Salter instance a set of key pairs may be created. These keypairs are instantiated as Signer class instances. The Signer class can be found in keri.core.signing.Sigher. The creation code is as follows:

```
salt = b'acdcspecworkexam'
salter = Salter(raw=salt)
signers = salter.signers(count=8,transferable=True, temp=True)
```

The Salter.signers method creates a count number of Signer instances and returns them in a list. Each Signer holds a key pair. The private key seed for each key pair is created using Argon2 to stretch a deterministic path that is based on the salt and a path that is the hex representation of the count offset for each Signer. For the zeroth signer this is as follows:

```
import pysodium
seed = pysodium.crypto_pwhash(outlen=32, passwd='0', salt=b'acdcs
pecworkexam', opslimit=1,
                                  memlimit=8192, alg=pysodium.crypt
o_pwhash_ALG_ARGON2ID13)
```

The seed becomes the private signing key for the keypair the public verification key is generated using Ed25519 as follows:

```
verkey, sigkey = pysodium.crypto_sign_seed_keypair(seed)
```

The verkey is used as the raw input to a Verfer (verifier) Class instance (found in keri.core.coring.Verfer) as follows:

```
verfer = Verfer(raw=verkey, code=MtrDex.Ed25519)
```

From this, the initial signing public verification key in CESR encoded qualified Base64 `Text` domain representation is as follows:

```
'DA8-J0EW88RMYqtUHQDqT4q2YH2iBFlW8HobHKV74yi_'
```

In order to create the Issuer AID, two other signing key pairs are needed. One other is the “next” rotating key pair. The signer at index 1 is used for this. From this, the next rotating public verification key in CESR encoded qualified Base64 `Text` domain representation is as follows:

```
'DLe4uewytqfqa4NB4AntNKBZ61I0TYcgMz-FSz1V9qeM'
```

Another one is a witness key pair. Witness AIDs are usually non-transferable and use a non-transferable CESR encoding. The associated key pairs may be generated as follows:

```
walt = b'acdcspecworkwits'
walter = Salter(raw=walt)
wigners = walter.signers(count=4,transferable=False, temp=True)
```

The signer (wigner) at index 0 is used for this. From this, the witness AID, (which is also its signing public verification key) in CESR encoded qualified Base64 `Text` domain representation is as follows:

```
'BKRaC6UsijUY1FRjExoAMc8WOHBDIfIKYnOlxWH8eOe8'
```

From these three cryptographic primitives we can create a Python dictionary with all the data needed to generate the inception event for the Issuer as follows:

```
 sad = \
 {
        'v': 'KERICAACAAJSONAAFb.',
        't': 'icp',
        'd': 'ECmiMVHTfZIjhA_rovnfx73T3G_FJzIQtzDn1meBVLAz',
        'i': 'ECmiMVHTfZIjhA_rovnfx73T3G_FJzIQtzDn1meBVLAz',
```

```
        's': '0',
        'kt': '1',
        'k': ['DA8-J0EW88RMYqtUHQDqT4q2YH2iBFlW8HobHKV74yi_'],
        'nt': '1',
        'n': ['DLe4uewytqfqa4NB4AntNKBZ61I0TYcgMz-FSz1V9qeM'],
        'bt': '1',
        'b': ['BKRaC6UsijUY1FRjExoAMc8WOHBDIfIKYnOlxWH8eOe8'],
        'c': [],
        'a': []
}
```

The JSON examples use a JSON serialization of the inception event to generate the Issuer AID. In python, this is performed as follows:

```
import json

raw = json.dumps(sad, separators=(",", ":"), ensure_ascii=False).
encode()
```

This results in the following raw JSON for the inception event.

```
(b'{"v":"KERICAACAAJSONAAFb.","t":"icp","d":"ECmiMVHTfZIjhA_rovnf
x73T3G_FJzIQtz'
b'Dn1meBVLAz","i":"ECmiMVHTfZIjhA_rovnfx73T3G_FJzIQtzDn1meBVLA
z","s":"0","kt":'
b'"1","k":["DA8-J0EW88RMYqtUHQDqT4q2YH2iBFlW8HobHKV74yi_"],"n
t":"1","n":["DLe4'
b'uewytqfqa4NB4AntNKBZ61I0TYcgMz-FSz1V9qeM"],"bt":"1","b":["BKRaC
6UsijUY1FRjEx'
b'oAMc8WOHBDIfIKYnOlxWH8eOe8"],"c":[],"a":[]}')
```

The calculation of the SAIDed fields in the inception event require knowledge of the KERI and CESR protocols for generating SAIDs on serialization of field maps. A utility function for generating an inception event may be found in keri.core.eventing.incept. The raw above was generated as follows:

```
from collections import namedtuple

issuerSigner = signers[0]
issuerVerKey = issuerSigner.verfer.qb64  # issuer's public verifi
cation key
assert issuerVerKey == 'DA8-J0EW88RMYqtUHQDqT4q2YH2iBFlW8HobHKV74
yi_'

issuerRotSigner = signers[1]
issuerRotVerKey = issuerRotSigner.verfer.qb64  # issuer's public
```

```
verification key
assert issuerRotVerKey == 'DLe4uewytqfqa4NB4AntNKBZ61I0TYcgMz-FSz
1V9qeM' # use in example

issuerWitSigner = wigners[0]
issuerWitVerKey = issuerWitSigner.verfer.qb64  # issuer's public
verification key
assert issuerWitVerKey == 'BKRaC6UsijUY1FRjExoAMc8WOHBDIfIKYnOlxW
H8eOe8' # use in example

Versionage = namedtuple("Versionage", "major minor")
Vrsn_2_0 = Versionage(major=2, minor=0)  # KERI Protocol Version
Specific

assert MtrDex.Blake3_256 == 'E'

keys = [issuerVerKey]  # initial signing keys
nkeys = [issuerRotVerKey]  # next (rotation) keys
wits = [issuerWitVerKey]  # witness aids (same as public verkey)
serder = incept(keys, code='E', ndigs=nkeys, wits=wits, version=V
rsn_2_0, kind='JSON')
```

The ACDC examples use the resultant Issuer AID, namely:

```
"ECmiMVHTfZIjhA_rovnfx73T3G_FJzIQtzDn1meBVLAz"
```

In some examples, this AID is given the name Amy has in Amy's AID.

Some examples also have an Issuee AID. This is generated similarly to the Issuer AID but using different keys. This produces the following inception event, sad dictionary, and raw.

```
sad = \
{
        'v': 'KERICAACAAJSONAAFb.',
        't': 'icp',
        'd': 'ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf',
        'i': 'ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf',
        's': '0',
        'kt': '1',
        'k': ['DLc6u76NIr0VqirQPC9k4WlOHw2mIG97Q3BV0UM3EjH5'],
        'nt': '1',
        'n': ['DC-QD-olCSce4Sf6qapV0vqP0lvROt5DWqMcUYxY6mFQ'],
        'bt': '1',
        'b': ['BOfVpWhuTlb5S2CJOVLSeUm9A0Xq7h0n6e9Qng3g4H2p'],
        'c': [],
```

```
        'a': []
    }
```

```
(b'{"v":"KERICAACAAJSONAAFb.","t":"icp","d":"ECWJZFBtllh99fESUOrB
vT3EtBujWtDKCm'
b'yzDAXWhYmf","i":"ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYm
f","s":"0","kt":'
b'"1","k":["DLc6u76NIr0VqirQPC9k4WlOHw2mIG97Q3BV0UM3EjH5"],"n
t":"1","n":["DC-Q'
b'D-olCSce4Sf6qapV0vqP0lvROt5DWqMcUYxY6mFQ"],"bt":"1","b":["BOfVp
WhuTlb5S2CJOV'
b'LSeUm9A0Xq7h0n6e9Qng3g4H2p"],"c":[],"a":[]}')
```

The process is summarized with the following code snippet.

```
from collections import namedtuple

issuerSigner = signers[2]
issuerVerKey = issuerSigner.verfer.qb64  # issuer's public verifi
cation key
assert issuerVerKey == 'DLc6u76NIr0VqirQPC9k4WlOHw2mIG97Q3BV0UM3E
jH5'

issuerRotSigner = signers[3]
issuerRotVerKey = issuerRotSigner.verfer.qb64  # issuer's public
verification key
assert issuerRotVerKey == 'DC-QD-olCSce4Sf6qapV0vqP0lvROt5DWqMcUY
xY6mFQ' # use in example

issuerWitSigner = wigners[1]
issuerWitVerKey = issuerWitSigner.verfer.qb64  # issuer's public
verification key
assert issuerWitVerKey == 'BOfVpWhuTlb5S2CJOVLSeUm9A0Xq7h0n6e9Qng
3g4H2p' # use in example

Versionage = namedtuple("Versionage", "major minor")
Vrsn_2_0 = Versionage(major=2, minor=0)  # KERI Protocol Version
Specific

assert MtrDex.Blake3_256 == 'E'

keys = [issuerVerKey]  # initial signing keys
nkeys = [issuerRotVerKey]  # next (rotation) keys
wits = [issuerWitVerKey]  # witness aids (same as public verkey)
serder = incept(keys, code='E', ndigs=nkeys, wits=wits, version=V
rsn_2_0, kind='JSON')
```

The ACDC examples use the resultant Issuee AID, namely:

```
"ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf"
```

In some examples, this AID is given the name Bob has in Bob's AID.

Similarly, an AID for Cal used in some examples can be created using signers[4] for the initial signing key pair, signers[5] for the rotating key pair, and wigners[2] for the witness AID. The resultant AID for Cal is:

```
"ECsGDKWAYtHBCkiDrzajkxs3Iw2g-dls3bLUsRP4yVdT"
```

Likewise, an AID for Deb used in some examples can be created using signers[6] for the initial signing key pair, signers[7] for the rotating key pair, and wigners[3] for the witness AID. The resultant AID for Deb is:

```
"EEDGM_DvZ9qFEAPf_FX08J3HX49ycrVvYVXe9isaP5SW"
```

These are summarized here for reference:

```
amy = "ECmiMVHTfZIjhA_rovnfx73T3G_FJzIQtzDn1meBVLAz"
bob = "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf"
cal = "ECsGDKWAYtHBCkiDrzajkxs3Iw2g-dls3bLUsRP4yVdT"
deb = "EEDGM_DvZ9qFEAPf_FX08J3HX49ycrVvYVXe9isaP5SW"
```

Registry SAIDs

ACDCs typically would reference a TEL that manages the issuance/revocation state of the ACDC. This requires a TEL registry identifier that is the SAID of a registry inception event, `rip`.

A registry is created for each of the AIDs for Amy, Bob, Cal and Deb as issuer of their respective registry as follows:

Registry Inception event for Amy

```
{
        "v": "ACDCCAACAAJSONAADa.",
        "t": "rip",
        "d": "EOMMCyztOvg970W0dZVJT2JIwlQ22DSeY7wtxNBBtpmX",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXcw",
        "i": "ECmiMVHTfZIjhA_rovnfx73T3G_FJzIQtzDn1meBVLAz",
        "n": "0",
        "dt": "2025-07-04T17:50:00.000000+00:00"
}
```

Registry SAID for Amy = `EOMMCyztOvg970W0dZVJT2JIwlQ22DSeY7wtxNBBtpmX`

Registry Inception event for Bob

```
{
    "v": "ACDCCAACAAJSONAADa.",
    "t": "rip",
    "d": "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXcx",
    "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf",
    "n": "0",
    "dt": "2025-07-04T17:51:00.000000+00:00"
}
```

Registry SAID for Bob = `ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ`

Registry Inception for Cal

```
{
        "v": "ACDCCAACAAJSONAADa.",
        "t": "rip",
        "d": "EPtolmh_NE2vC02oFc7FOiWkPcEiKUPWm5uu_Gv1JZDw",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXcy",
        "i": "ECsGDKWAYtHBCkiDrzajkxs3Iw2g-dls3bLUsRP4yVdT",
        "n": "0",
        "dt": "2025-07-04T17:52:00.000000+00:00"
}
```

Registry SAID for Cal = `EPtolmh_NE2vC02oFc7FOiWkPcEiKUPWm5uu_Gv1JZDw`

Registry Inception for Deb

```
{
        "v": "ACDCCAACAAJSONAADa.",
        "t": "rip",
        "d": "EJl5EUxL23p_pqgN3IyM-pzru89Nb7NzOM8ijH644xSU",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXcz",
        "i": "EEDGM_DvZ9qFEAPf_FX08J3HX49ycrVvYVXe9isaP5SW",
        "n": "0",
        "dt": "2025-07-04T17:53:00.000000+00:00"
}
```

Registry SAID for Deb = `EJl5EUxL23p_pqgN3IyM-pzru89Nb7NzOM8ijH644xSU`

The registry SAIDs (registry IDs) are summarized here:

```
regAmy = "EOMMCyztOvg970W0dZVJT2JIwlQ22DSeY7wtxNBBtpmX"
regBob = "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ"
```

```
regCal = "EPtolmh_NE2vC02oFc7FOiWkPcEiKUPWm5uu_Gv1JZDw"
regDeb = "EJl5EUxL23p_pqgN3IyM-pzru89Nb7NzOM8ijH644xSU"
```

UUIDs

Many of the examples include UUID, `u` fields with salty nonce values. For ease of reproducibility deterministic UUIDs are used. These come from the following set:

```
assert uuids == \
[
     '0ABhY2Rjc3BlY3dvcmtyYXcw',
     '0ABhY2Rjc3BlY3dvcmtyYXcx',
     '0ABhY2Rjc3BlY3dvcmtyYXcy',
     '0ABhY2Rjc3BlY3dvcmtyYXcz',
     '0ABhY2Rjc3BlY3dvcmtyYXc0',
     '0ABhY2Rjc3BlY3dvcmtyYXc1',
     '0ABhY2Rjc3BlY3dvcmtyYXc2',
     '0ABhY2Rjc3BlY3dvcmtyYXc3',
     '0ABhY2Rjc3BlY3dvcmtyYXc4',
     '0ABhY2Rjc3BlY3dvcmtyYXc5',
     '0ABhY2Rjc3BlY3dvcmtyYXdh',
     '0ABhY2Rjc3BlY3dvcmtyYXdi',
     '0ABhY2Rjc3BlY3dvcmtyYXdj',
     '0ABhY2Rjc3BlY3dvcmtyYXdk',
     '0ABhY2Rjc3BlY3dvcmtyYXdl',
     '0ABhY2Rjc3BlY3dvcmtyYXdm'
]
```

These may be generated with the following code snippet:

```
raws = [b'acdcspecworkraw' + b'%0x'%(i, ) for i in range(16)]
uuids = [Noncer(raw=raw).qb64 for raw in raws]
```

The Noncer class may be found in keri.core.coring.Noncer. Essentially, a Noncer instance can encode a byte string as a salty nonce in CESR format.

Complete ACDC Example: Private (partially disclosable) ACDC with Edges

Suppose for this example that Sunspot College issues a grade transcript via an ACDC to a student named Zoe Doe. This transcript has three edges. One edge links back to an accreditation credential issued to the college. The other two edges are examples of work product by the student to demonstrate skill proficiency. One is a credential issued by the student's major department endorsing a research report written by the student. This endorsement is in turn endorsed by the college by virtue of being linked via an edge in the

transcript ACDC. The other edge links to a credential issued by the student asserting the student's senior project. This is endorsed by the college by virtue of being linked via an edge in the transcript ACDC.

The AIDs as setup for this example have aliases for each AID, the aliases do not correspond to real names they are used for simplicity of reference. The college named Sunspot College uses the AID named Amy. The student named Zoe Doe uses the AID named Bob. The accreditation agency uses the AID named Cal. Finally, the department in the college uses the AID named Deb. In this example, the serialization kind used for all the messages is JSON.

The associated AIDs are summarized as follows:

```
amy = "ECmiMVHTfZIjhA_rovnfx73T3G_FJzIQtzDn1meBVLAz"
bob = "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf"
cal = "ECsGDKWAYtHBCkiDrzajkxs3Iw2g-dls3bLUsRP4yVdT"
deb = "EEDGM_DvZ9qFEAPf_FX08J3HX49ycrVvYVXe9isaP5SW"
```

Several other issuances must be created and issued before the transcript ACDC can be created and issued. These will be stepped through in turn.

Accreditation ACDC

Cal (a euphemism for the accreditation agency) creates a placeholder registry in order to issue a revokable ACDC to Amy (a euphemism for Sunspot College). This is created with a registration inception event message as follows:

```
{
    "v": "ACDCCAACAAJSONAADa.",
    "t": "rip",
    "d": "EPtolmh_NE2vC02oFc7FOiWkPcEiKUPWm5uu_Gv1JZDw",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXcy",
    "i": "ECsGDKWAYtHBCkiDrzajkxs3Iw2g-dls3bLUsRP4yVdT",
    "n": "0",
    "dt": "2025-07-04T17:52:00.000000+00:00"
}
```

The registration identifier is the SAID of this message, as follows:

```
regCal = "EPtolmh_NE2vC02oFc7FOiWkPcEiKUPWm5uu_Gv1JZDw"
```

Cal uses this to issue an accreditation ACDC. The full ACDC is as follows:

```
{
   "v": "ACDCCAACAAJSONAAKX.",
   "t": "acm",
```

```
    "d": "EIF7egPvC8ITbGRdM9G0kd6aPELDg-azMkAqT-7cMuAi",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXdh",
    "i": "ECsGDKWAYtHBCkiDrzajkxs3Iw2g-dls3bLUsRP4yVdT",
    "rd": "EPtolmh_NE2vC02oFc7FOiWkPcEiKUPWm5uu_Gv1JZDw",
    "s": "EK_iGlfdc7Q-qIGL-kqbDSD2z4fesT4dAQLEHGgH4lLG",
    "a":
    {
        "d": "EK799owRYyk8UPFWUmfsm5AJfJmU7jZGtZXJFbg2I0KL",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXc3",
        "i": "ECmiMVHTfZIjhA_rovnfx73T3G_FJzIQtzDn1meBVLAz",
        "name": "Sunspot College",
        "level": "gold"
    },
    "r":
    {
        "d": "EMZf9m0XYwqo4L8tnIDMZuX7YCZnMswS7Ta9j0CuYfjU",
        "l": "Issuer provides this ACDC on an AS IS basis. This AC
DC in whole or in part MUST NOT be shared with any other entity b
esides the intended recipient."
    }
}
```

Note that the value of the `rd` field is the same as the value of regCal. This binds the issued ACDC to a specific issuance/revocation state registry. Note also that the top-level `i` field value is Cal's AID. Also note that nested inside the attribute block with label `a` is the issuee field `i` (not to be confused with the top-level issuer). The issuee field value is Amy's AID. This binds Amy as the target of the ACDC.

This ACDC can be compacted into its most compact form. This is as follows:

```
{
    "v": "ACDCCAACAAJSONAAF3.",
    "t": "acm",
    "d": "EIF7egPvC8ITbGRdM9G0kd6aPELDg-azMkAqT-7cMuAi",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXdh",
    "i": "ECsGDKWAYtHBCkiDrzajkxs3Iw2g-dls3bLUsRP4yVdT",
    "rd": "EPtolmh_NE2vC02oFc7FOiWkPcEiKUPWm5uu_Gv1JZDw",
    "s": "EK_iGlfdc7Q-qIGL-kqbDSD2z4fesT4dAQLEHGgH4lLG",
    "a": "EK799owRYyk8UPFWUmfsm5AJfJmU7jZGtZXJFbg2I0KL",
    "r": "EMZf9m0XYwqo4L8tnIDMZuX7YCZnMswS7Ta9j0CuYfjU"
}
```

Note that the SAIDs of the sections are the same in both the uncompact and compact variants.

The schema for this ACDC is as follows:

```
{
        "$id": "EK_iGlfdc7Q-qIGL-kqbDSD2z4fesT4dAQLEHGgH4lLG",
        "$schema": "https://json-schema.org/draft/2020-
12/schema",
        "title": "Accreditation Schema",
        "description": "Accreditation JSON Schema for acm ACDC.",
        "credentialType": "Accreditation_ACDC_acm_message",
        "version": "2.0.0",
        "type": "object",
        "required": ["v", "d", "i", "s", "a", "r"],
        "properties":
        {
            "v": {"description": "ACDC version string", "type":
"string"},
            "t": {"description": "Message type", "type":
"string"},
            "d": {"description": "Message SAID", "type":
"string"},
            "u": {"description": "Message UUID", "type":
"string"},
            "i": {"description": "Issuer AID", "type": "string"},
            "rd": {"description": "Registry SAID", "type":
"string"},
            "s":
            {
                "description": "Schema Section",
                "oneOf":
                [
                    {"description": "Schema Section SAID",
"type": "string"},
                    {"description": "Schema Section Detail",
"type": "object"}
                ]
            },
            "a":
            {
                "description": "Attribute Section",
                "oneOf":
                [
                    {"description": "Attribute Section SAID",
"type": "string"},
                    {
                        "description": "Attribute Section
Detail",
                        "type": "object",
                        "required": ["d", "u", "i", "score",
"name"],
                        "properties":
                        {
```

```
                            "d": {"description": "Attribute
Section SAID", "type": "string"},
                            "u": {"description": "Attribute
Section UUID", "type": "string"},
                            "i": {"description": "Issuee AID",
"type": "string"},
                            "name": {"description": "Institution
Name", "type": "string"},
                            "level": {"description":
"Accreditation Level", "type": "string"}
                        },
                        "additionalProperties": false
                    }
                ]
            },
            "e":
            {
                "description": "Edge Section",
                "oneOf":
                [
                    {"description": "Edge Section SAID", "type":
"string"},
                    {"description": "Edge Section Detail",
"type": "object"}
                ]
            },
            "r":
            {
                "description": "Rule Section",
                "oneOf":
                [
                    {"description": "Rule Section SAID", "type":
"string"},
                    {
                        "description": "Rule Section Detail",
                        "type": "object",
                        "required": ["d", "l"],
                        "properties":
                        {
                            "d": {"description": "Rule Section
SAID", "type": "string"},
                            "l": {"description": "Legal
Language", "type": "string"}
                        },
                        "additionalProperties": false
                    }
                ]
            }
        },
```

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

Note that the SAID of the schema is provided by the `$id` field. Its field value matches the value of the `s` field in the ACDC itself.

Research Report ACDC

Deb (a euphemism for the department) creates a placeholder registry to issue a revocable ACDC that endorses a report authored by Bob. The associated registry was created with a registration inception event message as follows:

```
{
    "v": "ACDCCAACAAJSONAADa.",
    "t": "rip",
    "d": "EJl5EUxL23p_pqgN3IyM-pzru89Nb7NzOM8ijH644xSU",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXcz",
    "i": "EEDGM_DvZ9qFEAPf_FX08J3HX49ycrVvYVXe9isaP5SW",
    "n": "0",
    "dt": "2025-07-04T17:53:00.000000+00:00"
}
```

The registration identifier is the SAID of this message, as follows:

```
regDeb = "EJl5EUxL23p_pqgN3IyM-pzru89Nb7NzOM8ijH644xSU"
```

Deb uses this to issue a report endorsement credential as follows:

```
{
    "v": "ACDCCAACAAJSONAAK4.",
    "t": "acm",
    "d": "EAU5dUws4ffM9jZjWs0QfXTnhJ1qk2u3IUhBwFVbFnt5",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXdi",
    "i": "EEDGM_DvZ9qFEAPf_FX08J3HX49ycrVvYVXe9isaP5SW",
    "rd": "EJl5EUxL23p_pqgN3IyM-pzru89Nb7NzOM8ijH644xSU",
    "s": "EKMXqyMQmOy0RuEj1VgOK9aD4GYR0D8Dcj0kssQtcY4-",
    "a":
    {
        "d": "EFTqnoiGSf-D76W3geNxEudBI_wz81FIkIXjzsjFztI-",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXc4",
        "title": "Post Quantum Security",
        "name": "Zoe Doe",
        "report": "Impremention should prioritize cryptographic
agility over PQ."
    },
    "r":
    {
```

```
        "d": "EMZf9m0XYwqo4L8tnIDMZuX7YCZnMswS7Ta9j0CuYfjU",
        "l": "Issuer provides this ACDC on an AS IS basis. This A
CDC in whole or in part MUST NOT be shared with any other entity
besides the intended recipient."
    }
}
```

Note that the value of the `rd` field is the same as the value of regDeb. This binds the issued ACDC to a specific issuance/revocation state registry. Note also that the top-level `i` field value is Deb's AID. Also note there is no nested issuee field. This ACDC is therefore untargeted. The only association it to the student as named author via the `name` field.

This ACDC can be compacted into its most compact form. This is as follows:

```
{
    "v": "ACDCCAACAAJSONAAF3.",
    "t": "acm",
    "d": "EAU5dUws4ffM9jZjWs0QfXTnhJ1qk2u3IUhBwFVbFnt5",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXdi",
    "i": "EEDGM_DvZ9qFEAPf_FX08J3HX49ycrVvYVXe9isaP5SW",
    "rd": "EJl5EUxL23p_pqgN3IyM-pzru89Nb7NzOM8ijH644xSU",
    "s": "EKMXqyMQmOy0RuEj1VgOK9aD4GYR0D8Dcj0kssQtcY4-",
    "a": "EFTqnoiGSf-D76W3geNxEudBI_wz81FIkIXjzsjFztI-",
    "r": "EMZf9m0XYwqo4L8tnIDMZuX7YCZnMswS7Ta9j0CuYfjU"
}
```

Note that the SAIDs of the sections are the same in both the uncompact and compact variants.

The schema for this ACDC is as follows:

```
{
    "$id": "EKMXqyMQmOy0RuEj1VgOK9aD4GYR0D8Dcj0kssQtcY4-",
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "title": "Report Schema",
    "description": "Report JSON Schema for acm ACDC.",
    "credentialType": "Report_ACDC_acm_message",
    "version": "2.0.0",
    "type": "object",
    "required": ["v", "d", "i", "s", "a", "r"],
    "properties":
    {
        "v": {"description": "ACDC version string", "type":
"string"},
        "t": {"description": "Message type", "type": "string"},
        "d": {"description": "Message SAID", "type": "string"},
```

```
        "u": {"description": "Message UUID", "type": "string"},
        "i": {"description": "Issuer AID", "type": "string"},
        "rd": {"description": "Registry SAID", "type": "string"},
        "s":
        {
            "description": "Schema Section",
            "oneOf":
            [
                {"description": "Schema Section SAID", "type":
"string"},
                {"description": "Schema Section Detail", "type":
"object"}
            ]
        },
        "a":
         {
            "description": "Attribute Section",
            "oneOf":
            [
                { "description": "Attribute Section SAID",
"type": "string"},
                {
                    "description": "Attribute Section Detail",
                    "type": "object",
                    "required": [ "d", "u", "i", "title",
"author", "report"],
                    "properties":
                    {
                      "d": {"description": "Attribute Section
SAID", "type": "string"},
                      "u": {"description": "Attribute Section
UUID", "type": "string"},
                      "title": {"description": "Report Title",
"type": "string"},
                      "author": {"description": "Author Full
Name", "type": "string"},
                      "report": { "description": "Report Body",
"type": "string"}
                    },
                    "additionalProperties": False
                  }
            ]
        },
        "e":
        {
            "description": "Edge Section",
            "oneOf":
            [
                {"description": "Edge Section SAID", "type":
```

```
"string"},
                {"description": "Edge Section Detail", "type":
"object"}
            ]
        },
        "r":
        {
            "description": "Rule Section",
            "oneOf":
            [
                {"description": "Rule Section SAID", "type":
"string"},
                {
                    "description": "Rule Section Detail",
                    "type": "object",
                    "required": ["d", "l"],
                    "properties":
                    {
                        "d": {"description": "Rule Section SAID",
"type": "string"},
                        "l": {"description": "Legal Language",
"type": "string"}
                    },
                    "additionalProperties": False
                }
            ]
        }
    },
    "additionalProperties": False
}
```

Note that the SAID of the schema is provided by the `$id` field. Its field value matches the value of the `s` field in the ACDC itself.

Project Report ACDC

Bob (a euphemism for the student) creates a placeholder registry to issue a revocable ACDC that asserts a report authored by Bob. The associated registry was created with a registration inception event message as follows:

```
{
    "v": "ACDCCAACAAJSONAADa.",
    "t": "rip",
    "d": "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXcx",
    "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf",
    "n": "0",
```

```
    "dt": "2025-07-04T17:51:00.000000+00:00"
}
```

The registration identifier is the SAID of this message, as follows:

```
regBob = "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ"
```

Bob uses this to issue a report credential as follows:

```
{
    "v": "ACDCCAACAAJSONAAKt.",
    "t": "acm",
    "d": "EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXdj",
    "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf",
    "rd": "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ",
    "s": "EKMXqyMQmOy0RuEj1VgOK9aD4GYR0D8Dcj0kssQtcY4-",
    "a":
    {
        "d": "EIg1zAS3FfMMbQtLqARSwS3uGMttVbAPhKB71bjIPTs_",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXc5",
        "title": "PQ Proof of Concept",
        "name": "Zoe Doe",
        "report": "Demonstration of recovery from surprise quantu
m attack"
    },
    "r":
    {
        "d": "EMZf9m0XYwqo4L8tnIDMZuX7YCZnMswS7Ta9j0CuYfjU",
        "l": "Issuer provides this ACDC on an AS IS basis. This A
CDC in whole or in part MUST NOT be shared with any other entity
besides the intended recipient."
    }
}
```

Note that the value of the `rd` field is the same as the value of regBob. This binds the issued ACDC to a specific issuance/revocation state registry. Note also that the top-level `i` field value is Bob's AID. Also note there is no nested issuee field. This ACDC is therefore untargeted. The only association is to the student as named author via the `name` field.

This ACDC can be compacted into its most compact form. This is as follows:

```
{
    "v": "ACDCCAACAAJSONAAF3.",
    "t": "acm",
    "d": "EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4M",
```

```
    "u": "0ABhY2Rjc3BlY3dvcmtyYXdj",
    "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf",
    "rd": "ECOWJI9kAjpCFYJ7RenpJx2w66-GsGlhyKLO-Or3qOIQ",
    "s": "EKMXqyMQmOy0RuEj1VgOK9aD4GYR0D8Dcj0kssQtcY4-",
    "a": "EIg1zAS3FfMMbQtLqARSwS3uGMttVbAPhKB71bjIPTs_",
    "r": "EMZf9m0XYwqo4L8tnIDMZuX7YCZnMswS7Ta9j0CuYfjU"
}
```

Note that the SAIDs of the sections are the same in both the uncompact and compact variants.

This project report ACDC uses the same schema as the research report ACDC.

Transcript ACDC with Private Edges

Now that the dependent ACDCs have been issued, Amy (a euphemism for the Sunspot College) can issue the transcript ACDC to Bob.

Amy first creates a placeholder registry for maintaining the issue/revoke status of the transcript ACDC. The associated registry was created with a registration inception event message as follows:

```
{
    "v": "ACDCCAACAAJSONAADa.",
    "t": "rip",
    "d": "EOMMCyztOvg970W0dZVJT2JIwlQ22DSeY7wtxNBBtpmX",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXcw",
    "i": "ECmiMVHTfZIjhA_rovnfx73T3G_FJzIQtzDn1meBVLAz",
    "n": "0",
    "dt": "2025-07-04T17:50:00.000000+00:00"
}
```

The registration identifier is the SAID of this message, as follows:

```
regAmy = "EOMMCyztOvg970W0dZVJT2JIwlQ22DSeY7wtxNBBtpmX"
```

Amy uses this to issue a transcript ACDC to Bob as follows:

```
{
    "v": "ACDCCAACAAJSONAAXG.",
    "d": "ENeNWgCCNcOf1JbgKxUzREKpyK5kABYFd2QYUzEfwz9H",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXdk",
    "i": "ECmiMVHTfZIjhA_rovnfx73T3G_FJzIQtzDn1meBVLAz",
    "rd": "EOMMCyztOvg970W0dZVJT2JIwlQ22DSeY7wtxNBBtpmX",
    "s": "EABGAia_vH_zHCRLOK3Bm2xxujV5A8sYIJbypfSM_2Fh",
    "a":
    {
```

```
        "d": "ELI2TuO6mLF0cR_0iU57EjYK4dExHIHdHxlRcAdO6x-U",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXcw",
        "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf",
        "name": "Zoe Doe",
        "gpa": 3.5,
        "grades":
        {
            "d": "EFQnBFeKAeS4DAWYoKDwWXOT4h2-XaGk7-w4-2N4ktXy",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcx",
            "history": 3.5,
            "english": 4.0,
            "math": 3.0
        }
    },
    "e":
    {
        "d": "ECpmTyIIc1duvCeIceK19Sbd0uymklmwNTtwtmfjQnX0",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXcy",
        "accreditation":
        {
            "d": "EAFj8JaNEC3mdFNJKrXW8E03_k9qqb_xM9NjAPVHw-xJ",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcz",
            "n": "EIF7egPvC8ITbGRdM9G0kd6aPELDg-azMkAqT-7cMuAi",
            "s": "EK_iGlfdc7Q-qIGL-kqbDSD2z4fesT4dAQLEHGgH4lLG"
        },
        "reports":
        {
            "d": "EOObmbCppe1S-7vtLuy766_4-RcfrC7p4ciFtBxdexuz",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXc0",
            "o": "OR",
            "research":
            {
                "d": "EN9ngstOcFHqsjqf75JZFKtCRmW76NkeRrUSxTLoqqk
I",
                "u": "0ABhY2Rjc3BlY3dvcmtyYXc2",
                "n": "EAU5dUws4ffM9jZjWs0QfXTnhJ1qk2u3IUhBwFVbFnt
5",
                "o": "NI2I"
            },
            "project":
            {
                "d": "EFwHz5qJ4_8c7IefP7_zugX2eIgtoyY8Up_WZ3osXwk
I",
                "u": "0ABhY2Rjc3BlY3dvcmtyYXc1",
                "n": "EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Wwc4
M",
                "o": "NI2I"
            }
        }
```

```
    },
    "r":
    {
        "d": "EMZf9m0XYwqo4L8tnIDMZuX7YCZnMswS7Ta9j0CuYfjU",
        "l": "Issuer provides this ACDC on an AS IS basis. This A
CDC in whole or in part MUST NOT be shared with any other entity
besides the intended recipient."
    }
}
```

Note that the value of the `rd` field is the same as the value of regAmy. This binds the issued ACDC to a specific issuance/revocation state registry. Note also that the top-level `i` field value is Amy's AID. Also note that nested inside the attribute block with label `a` is the issuee field `i` (not to be confused with the top-level issuer). The issuee field value is Bob's AID. This binds Bob as the target of the ACDC.

Notable about this ACDC is the Edge Section. There are three edges, two of which belong to a nested Edge Group. The first Edge is to the accreditation ACDC. This is targeted and the value of the `n` field in its block is the accrediation ACDC SAID. The `s` field value is the schema SAID of the accreditation ACDC schema. Because the default M-ary operator is `AND` and `AND` is what is desired, no operator `o` field is needed for the top-level Edge Section group. Likewise the default operator for a target ACDC is `I2I` so no edge operator `o` field is required.

In the Edge group labeled `reports` are the other two edges; neither of these is targeted. This edge group uses an `OR` operator. This means that the Edge group is satisfied if either or both of its Edges are satisfied.

For the sake of example, each edge has a Unary, `NI2I` edge operator. Because the edges are in blocks, each with a SAID and UUID fields, the edges can be compacted for confidentiality.

This enables the graduated partial disclosure of the edges. The details of the linked (chained) credentials can be hidden by their block SAIDS so a disclosee can't see the linked ACDCs until the disclosse has to agreed to whatever terms are required by the discloser. Likewise for the rule section. This illustrates that not only attributes but edges and rules can be partially disclosable.

This ACDC can be compacted into its most compact form. This is as follows:

```
{
    "v": "ACDCCAACAAJSONAAGg.",
    "d": "ENeNWgCCNcOf1JbgKxUzREKpyK5kABYFd2QYUzEfwz9H",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXdk",
    "i": "ECmiMVHTfZIjhA_rovnfx73T3G_FJzIQtzDn1meBVLAz",
    "rd": "EOMMCyztOvg970W0dZVJT2JIwlQ22DSeY7wtxNBBtpmX",
    "s": "EABGAia_vH_zHCRLOK3Bm2xxujV5A8sYIJbypfSM_2Fh",
```

```
    "a": "ELI2TuO6mLF0cR_0iU57EjYK4dExHIHdHxlRcAdO6x-U",
    "e": "ECpmTyIIc1duvCeIceK19Sbd0uymklmwNTtwtmfjQnX0",
    "r": "EMZf9m0XYwqo4L8tnIDMZuX7YCZnMswS7Ta9j0CuYfjU"
}
```

Note that the SAIDs of the sections are the same in both the uncompact and compact varieties

The schema for this ACDC is as follows:

```
{
        "$id": "EABGAia_vH_zHCRLOK3Bm2xxujV5A8sYIJbypfSM_2Fh",
        "$schema": "https://json-schema.org/draft/2020-
12/schema",
        "title": "Transcript Schema",
        "description": "Transcript JSON Schema for acm ACDC.",
        "credentialType": "Transcript_ACDC_acm_message",
        "version": "2.0.0",
        "type": "object",
        "required": ["v", "d", "i", "s", "a", "r"],
        "properties":
        {
            "v": {"description": "ACDC version string", "type":
"string"},
            "t": {"description": "Message type", "type":
"string"},
            "d": {"description": "Message SAID", "type":
"string"},
            "u": {"description": "Message UUID", "type":
"string"},
            "i": {"description": "Issuer AID", "type": "string"},
            "rd": {"description": "Registry SAID", "type":
"string"},
            "s":
            {
                "description": "Schema Section",
                "oneOf":
                [
                    {"description": "Schema Section SAID",
"type": "string"},
                    {"description": "Schema Section Detail",
"type": "object"}
                ]
            },
            "a":
             {
                "description": "Attribute Section",
                "oneOf":
                [
```

```
                    { "description": "Attribute Section SAID",
"type": "string"},
                    {
                      "description": "Attribute Section Detail",
                      "type": "object",
                      "required": ["d", "u", "i", "name" "gpa",
"grades"],
                      "properties":
                      {
                        "d": {"description": "Attribute Section
SAID", "type": "string"},
                        "i": {"description": "Issuee AID",
"type": "string"},
                        "name": {"description": "Student Full
Name", "type": "string"},
                        "gpa": {"description": "Grade Point
Average", "type": "number"},
                        "grades":
                        {
                          "description": "Grades Block",
                          "oneOf":
                          [
                            {"description": "Block SAID", "type":
"string"},
                            {
                              "description": "Block detail",
                              "type": "object",
                              "required": ["d", "u", "history"
"english", "math"],
                              "properties":
                              {
                                "d": {"description": "Block
SAID", "type": "string"},
                                "u": {"description": "Block
UUID", "type": "string"},
                                "history": {"description":
"History Grade", "type": "number"},
                                "english": {"description":
"English Grade", "type": "number"},
                                "math": {"description": "Math
Grade", "type": "number"}
                              },
                              "additionalProperties": false
                            }
                          ]
                        },
                        "additionalProperties": false
                      }
                    }
```

```
                ]
            },
            "e":
            {
                "description": "Edge Section",
                "oneOf":
                [
                  { "description": "Edge Section SAID", "type":
"string"},
                  {
                    "description": "Edge Section Detail",
                    "type": "object",
                    "required": ["d", "u", "accreditation"
"reports"],
                    "properties":
                    {
                      "d": {"description": "Edge Section SAID",
"type": "string"},
                      "u": {"description": "Edge Section UUID",
"type": "string"},
                      "o": {"description": "Edge Section M-ary
Operator", "type": "string"},
                      "w": {"description": "Edge Section Weight",
"type": "number"},
                      "accreditation":
                      {
                        "description": "Accreditation Edge",
                        "oneOf":
                        [
                          {"description": "Edge SAID", "type":
"string"},
                          {
                            "description": "Edge Detail",
                            "type": "object",
                            "required": ["n"],
                            "properties":
                            {
                              "d": {"description": "Edge SAID",
"type": "string"},
                              "u": {"description": "Edge UUID",
"type": "string"},
                              "n": {"description": "Far Node
SAID", "type": "string"},
                              "s": {"description": "Far Node
Schema SAID", "type": "string"},
                              "o": {"description": "Edge Unary
Operator", "type": "string"},
                              "w": {"description": "Edge Weight",
"type": "number"},
```

```
                            },
                            "additionalProperties": false
                          }
                        ]
                      },
                      "reports":
                      {
                        "description": "Reports Edge Group",
                        "oneOf":
                        [
                          {"description": "Edge Group SAID",
"type": "string"},
                          {
                            "description": "Edge Group detail",
                            "type": "object",
                            "required": ["research", "project"],
                            "properties":
                            {
                              "d": {"description": "Block SAID",
"type": "string"},
                              "u": {"description": "Block UUID",
"type": "string"},
                              "s": {"description": "Far Node
Schema SAID", "type": "string"},
                              "o": {"description": "Edge Group M-
ary Operator", "type": "string"},
                              "w": {"description": "Edge Group
Weight", "type": "number"},
                              "research":
                              {
                                "description": "Research Edge",
                                "oneOf":
                                [
                                  {"description": "Edge SAID",
"type": "string"},
                                  {
                                    "description": "Edge Detail",
                                    "type": "object",
                                    "required": ["n"],
                                    "properties":
                                    {
                                      "d": {"description": "Edge
SAID", "type": "string"},
                                      "u": {"description": "Edge
UUID", "type": "string"},
                                      "n": {"description": "Far
Node SAID", "type": "string"},
                                      "s": {"description": "Far
Node Schema SAID", "type": "string"},
```

```
                                        "o": {"description": "Edge
Unary Operator", "type": "string"},
                                        "w": {"description": "Edge
Weight", "type": "number"},
                                    },
                                    "additionalProperties": false
                                }
                            ]
                        },
                        "project":
                        {
                            "description": "Project Edge",
                            "oneOf":
                            [
                                {"description": "Edge SAID",
"type": "string"},
                                {
                                    "description": "Edge Detail",
                                    "type": "object",
                                    "required": ["n"],
                                    "properties":
                                    {
                                        "d": {"description": "Edge
SAID", "type": "string"},
                                        "u": {"description": "Edge
UUID", "type": "string"},
                                        "n": {"description": "Far
Node SAID", "type": "string"},
                                        "s": {"description": "Far
Node Schema SAID", "type": "string"},
                                        "o": {"description": "Edge
Unary Operator", "type": "string"},
                                        "w": {"description": "Edge
Weight", "type": "number"},
                                    },
                                    "additionalProperties": false
                                }
                            ]
                        }
                    },
                    "additionalProperties": false
                }
            ]
        },
        "additionalProperties": false
    }
}
```

```
            "r":
            {
                "description": "Rule Section",
                "oneOf":
                [
                    {"description": "Rule Section SAID", "type":
"string"},
                    {
                        "description": "Rule Section Detail",
                        "type": "object",
                        "required": ["d", "l"],
                        "properties":
                        {
                            "d": {"description": "Rule Section
SAID", "type": "string"},
                            "l": {"description": "Legal
Language", "type": "string"}
                        },
                        "additionalProperties": false
                    }
                ]
            }
        },
        "additionalProperties": false
}
```

Transcript ACDC with Public Edges

Suppose there is no advantage to hidding the edges in the transcript ACDC. As a consequence, the Edge section can be greatly simplified. Instead of each block holding a SAID, UUID, as well as other fields, a given edge can be simplified to just the SAID of the linked ACDC. This requires that the default operators are adequate. In this case, it is assumed that there is no advantage to hiding the edges. Also the report edges do not actuall need the explicit unary, `NI2I` operators. This is the default for non-targeted ACDCs linked via an edge. All three edges are reexpressed in simple compact form.

The resulting ACDC is as follows:

```
{
    "v": "ACDCCAACAAJSONAAOD.",
    "d": "EBaEMTKi6ZtHXmkhxHUoGEEtG8JKelw3b0gv6cFTg6BN",
    "u": "0ABhY2Rjc3BlY3dvcmtyYXdl",
    "i": "ECmiMVHTfZIjhA_rovnfx73T3G_FJzIQtzDn1meBVLAz",
    "rd": "EOMMCyztOvg970W0dZVJT2JIwlQ22DSeY7wtxNBBtpmX",
    "s": "ECVhGE4yeuHZ8KEqWK-lx5O9xrfUg6wiDPkkxxQSjgfk",
    "a":
    {
```

```
        "d": "ELI2TuO6mLF0cR_0iU57EjYK4dExHIHdHxlRcAdO6x-U",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXcw",
        "i": "ECWJZFBtllh99fESUOrBvT3EtBujWtDKCmyzDAXWhYmf",
        "name": "Zoe Doe",
        "gpa": 3.5,
        "grades":
        {
            "d": "EFQnBFeKAeS4DAWYoKDwWXOT4h2-XaGk7-w4-2N4ktXy",
            "u": "0ABhY2Rjc3BlY3dvcmtyYXcx",
            "history": 3.5,
            "english": 4.0,
            "math": 3.0
        }
    },
    "e":
    {
        "d": "EEWx-E6Rexj3eORT-e2kLcAWVgviTqxwWvxS2LbNKuCh",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXcy",
        "accreditation": "EIF7egPvC8ITbGRdM9G0kd6aPELDg-azMkAqT-7
cMuAi",
        "reports":
        {
            "o": "OR",
            "research": "EAU5dUws4ffM9jZjWs0QfXTnhJ1qk2u3IUhBwFVb
Fnt5",
            "project": "EMLjZLIMlfUOoKox_sDwQaJO-0wdoGW0uNbmI28Ww
c4M"
        }
    },
    "r": "EMZf9m0XYwqo4L8tnIDMZuX7YCZnMswS7Ta9j0CuYfjU"
}
```

Note that the rule section, `r` field is represented by its SAID. It is the same rule section on the other ACDCs so its can be cached.

Notices that each of the edge values is the SAID of the far node ACDC not the SAID of the edge block. This is the simple edge format.

The most compact variant of this ACDC is as follows:

```
{
        "v": "ACDCCAACAAJSONAAGg.",
        "d": "EBaEMTKi6ZtHXmkhxHUoGEEtG8JKelw3b0gv6cFTg6BN",
        "u": "0ABhY2Rjc3BlY3dvcmtyYXdl",
        "i": "ECmiMVHTfZIjhA_rovnfx73T3G_FJzIQtzDn1meBVLAz",
        "rd": "EOMMCyztOvg970W0dZVJT2JIwlQ22DSeY7wtxNBBtpmX",
        "s": "ECVhGE4yeuHZ8KEqWK-lx5O9xrfUg6wiDPkkxxQSjgfk",
        "a": "ELI2TuO6mLF0cR_0iU57EjYK4dExHIHdHxlRcAdO6x-U",
        "e": "EEWx-E6Rexj3eORT-e2kLcAWVgviTqxwWvxS2LbNKuCh",
```

```
        "r": "EMZf9m0XYwqo4L8tnIDMZuX7YCZnMswS7Ta9j0CuYfjU"
    }
```

The ACDC's schema is as follows:

```
{
    "$id": "ECVhGE4yeuHZ8KEqWK-lx5O9xrfUg6wiDPkkxxQSjgfk",
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "title": "Transcript Schema",
    "description": "Transcript JSON Schema for acm ACDC.",
    "credentialType": "Transcript_ACDC_acm_message",
    "version": "2.0.0",
    "type": "object",
    "required": ["v", "d", "i", "s", "a", "r"],
    "properties":
    {
        "v": {"description": "ACDC version string", "type":
"string"},
        "t": {"description": "Message type", "type": "string"},
        "d": {"description": "Message SAID", "type": "string"},
        "u": {"description": "Message UUID", "type": "string"},
        "i": {"description": "Issuer AID", "type": "string"},
        "rd": {"description": "Registry SAID", "type": "string"},
        "s":
        {
            "description": "Schema Section",
            "oneOf":
            [
                {"description": "Schema Section SAID", "type":
"string"},
                {"description": "Schema Section Detail", "type":
"object"}
            ]
        },
        "a":
         {
            "description": "Attribute Section",
            "oneOf":
            [
                { "description": "Attribute Section SAID",
"type": "string"},
                {
                  "description": "Attribute Section Detail",
                  "type": "object",
                  "required": ["d", "u", "i", "name" "gpa",
"grades"],
                  "properties":
                  {
                    "d": {"description": "Attribute Section
```

```
SAID", "type": "string"},
                    "i": {"description": "Issuee AID", "type":
"string"},
                    "name": {"description": "Student Full Name",
"type": "string"},
                    "gpa": {"description": "Grade Point Average",
"type": "number"},
                    "grades":
                    {
                      "description": "Grades Block",
                      "oneOf":
                      [
                        {"description": "Block SAID", "type":
"string"},
                        {
                          "description": "Block detail",
                          "type": "object",
                          "required": ["d", "u", "history"
"english", "math"],
                          "properties":
                          {
                            "d": {"description": "Block SAID",
"type": "string"},
                            "u": {"description": "Block UUID",
"type": "string"},
                            "history": {"description": "History
Grade", "type": "number"},
                            "english": {"description": "English
Grade", "type": "number"},
                            "math": {"description": "Math Grade",
"type": "number"}
                          },
                          "additionalProperties": false
                        }
                      ]
                    },
                    "additionalProperties": false
                  }
                }
            ]
        },
        "e":
        {
            "description": "Edge Section",
            "oneOf":
            [
                { "description": "Edge Section SAID", "type":
"string"},
                {
```

```
                    "description": "Edge Section Detail",
                    "type": "object",
                    "required": ["d", "u", "accreditation"
"reports"],
                    "properties":
                    {
                        "d": {"description": "Edge Section SAID",
"type": "string"},
                        "u": {"description": "Edge Section UUID",
"type": "string"},
                        "o": {"description": "Edge Section M-ary
Operator", "type": "string"},
                        "w": {"description": "Edge Section
Weight", "type": "number"},
                        "accreditation": {"description": "Far
Node SAID", "type": "string"},
                        "reports":
                        {
                            "description": "Edge Group detail",
                            "type": "object",
                            "required": ["research", "project"],
                            "properties":
                            {
                                "d": {"description": "Block
SAID", "type": "string"},
                                "u": {"description": "Block
UUID", "type": "string"},
                                "s": {"description": "Far Node
Schema SAID", "type": "string"},
                                "o": {"description": "Edge Group
M-ary Operator", "type": "string"},
                                "w": {"description": "Edge Group
Weight", "type": "number"},
                                "research": {"description": "Far
Node SAID", "type": "string"},
                                "project": {"description": "Far
Node SAID", "type": "string"},
                            },
                            "additionalProperties": false
                        }
                    },
                    "additionalProperties": false
                }
            ]
        },
        "r":
        {
            "description": "Rule Section",
            "oneOf":
```

```
            [
                {"description": "Rule Section SAID", "type":
"string"},
                {
                    "description": "Rule Section Detail",
                    "type": "object",
                    "required": ["d", "l"],
                    "properties":
                    {
                        "d": {"description": "Rule Section SAID",
"type": "string"},
                        "l": {"description": "Legal Language",
"type": "string"}
                    },
                    "additionalProperties": false
                }
            ]
        }
    },
    "additionalProperties": false
}
```

Bibliography

Normative section

1. Composable Event Streaming Representation, CESR, https://github.com/trustoverip/kswg-acdc-specification

2. Key Event Receipt Infrastructure, KERI, https://github.com/trustoverip/kswg-keri-specification

3. Self-Addressing IDentifier, SAID, https://github.com/trustoverip/kswg-cesr-specification

4. Out-Of-Band-Introduction, OOBI, https://github.com/trustoverip/kswg-keri-specification

5. DIDK_ID, IETF DID-KERI Internet Draft, https://github.com/WebOfTrust/ietf-did-keri

8. RFC8259, JSON (JavaScript Object Notation), https://datatracker.ietf.org/doc/html/rfc8259

28. W3C_DID, W3C Decentralized Identifiers (DIDs) v1.0, https://w3c-ccg.github.io/did-spec/

61. ToIP did:webs Method Specification, https://trustoverip.github.io/tswg-did-method-webs-specification/

9. RFC4627, The application/json Media Type for JavaScript Object Notation (JSON), D. Crockford; 2006-07. Status: Informational.https://datatracker.ietf.org/doc/rfc4627/

71. IETF RFC-3339 Date and Time on the Internet: Timestamps DateTime. G. Klyne. 2002-07. Status: Standards Track

72. RFC4648 The Base16, Base32, and Base64 Data Encodings. S. Josefsson; 2006-10. Status: Proposed Standard.

73. IETF RFC-2119 Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. 1997-03. Status: Best Current Practice

74. ISO/IEC 7498-1:1994 Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model. June 1999. Introduction. Retrieved 26 August 2022.

Informative section

6. RFC6901, JavaScript Object Notation (JSON) Pointer, https://datatracker.ietf.org/doc/html/rfc6901

7. JSON, JavaScript Object Notation Delimeters, https://www.json.org/json-en.html

10. JSch, JSON Schema, https://json-schema.org

11. JSch_202012, JSON Schema 2020-12, https://json-schema.org/draft/2020-12/release-notes.html

12. CBOR, CBOR Mapping Object Codes, https://en.wikipedia.org/wiki/CBOR

13. RFC8949, Concise Binary Object Representation (CBOR), https://datatracker.ietf.org/doc/rfc8949/

14. MGPK, Msgpack Mapping Object Codes, https://github.com/msgpack/msgpack/blob/master/spec.md

15. RFC3986, Uniform Resource Identifier (URI): Generic Syntax, https://datatracker.ietf.org/doc/html/rfc3986

16. RFC8820, URI Design and Ownership, https://datatracker.ietf.org/doc/html/rfc8820

17. ACDC_TF, ACDC (Authentic Chained Data Container) Task Force, https://wiki.trustoverip.org/display/HOME/ACDC+(Authentic+Chained+Data+Container)+Task+Force

18. TOIP, Trust Over IP (ToIP) Foundation, https://trustoverip.org

19. ITPS, Information-Theoretic and Perfect Security, https://en.wikipedia.org/wiki/Information-theoretic_security

20. OTP, One-Time-Pad, https://en.wikipedia.org/wiki/One-time_pad

21.VCphr, Vernom Cipher (OTP) https://www.ciphermachinesandcryptology.com/en/one-timepad.htm

22. SSplt, Secret Splitting, https://www.ciphermachinesandcryptology.com/en/secretsplitting.htm

23. SShr, Secret Sharing, https://en.wikipedia.org/wiki/Secret_sharing

24. GLEIF, GLEIF (Global Legal Entity Identifier Foundation), https://www.gleif.org/en/

25. vLEI, vLEI (verifiable Legal Entity Identifier) Definition, https://github.com/WebOfTrust/vLEI

26. GLEIF_vLEI, GLEIF vLEI (verifiable Legal Entity Identifier), https://www.gleif.org/en/lei-solutions/gleifs-digital-strategy-for-the-lei/introducing-the-verifiable-lei-vlei

27. GLEIF_KERI, GLEIF with KERI Architecture, https://github.com/WebOfTrust/vLEI

29. Salt, Salts, Nonces, and Initial Values, https://medium.com/@fridakahsas/salt-nonces-and-ivs-whats-the-difference-d7a44724a447

30. RB, Rainbow Table, https://en.wikipedia.org/wiki/Rainbow_table

31. DRB, Dictionary Attacks, Rainbow Table Attacks and how Password Salting defends against them, https://www.commonlounge.com/discussion/2ee3f431a19e4deabe4aa30b43710aa7

32. BDay, Birthday Attack, https://en.wikipedia.org/wiki/Birthday_attack

33. BDC, Birthday Attacks, Collisions, And Password Strength, https://auth0.com/blog/birthday-attacks-collisions-and-password-strength/

34. HCR, Hash Collision Resistance, https://en.wikipedia.org/wiki/Collision_resistance

35. QCHC, Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?, https://cr.yp.to/hash/collisioncost-20090823.pdf

36. EdSC, The Provable Security of Ed25519: Theory and Practice Report, https://eprint.iacr.org/2020/823

37. PSEd, The Provable Security of Ed25519: Theory and Practice, https://ieeexplore.ieee.org/document/9519456?denied=

38. TMEd, Taming the many EdDSAs, https://eprint.iacr.org/2020/1244.pdf

39. JSchCp, Schema Composition in JSON Schema, https://json-schema.org/understanding-json-schema/reference/combining.html

40. JSchRE, Regular Expressions in JSON Schema, https://json-schema.org/understanding-json-schema/reference/regular_expressions.html

41/a>. JSchId, JSON Schema Identification, https://json-schema.org/understanding-json-schema/structuring.html#schema-identification

42. JSchCx, Complex JSON Schema Structuring, https://json-schema.org/understanding-json-schema/structuring.html#base-uri

43. RC, Ricardian Contract, https://en.wikipedia.org/wiki/Ricardian_contract

44. CLC, Chain-Link Confidentiality, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2045818

45. DHKE, Diffie-Hellman Key Exchange, https://www.infoworld.com/article/3647751/understand-diffie-hellman-key-exchange.html

46. KeyEx, Key Exchange, https://libsodium.gitbook.io/doc/key_exchange

47. IDSys, Identity System Essentials, https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/Identity-System-Essentials.pdf

48. Hash, Cryptographic Hash Function, https://en.wikipedia.org/wiki/Cryptographic_hash_function

49. Mrkl, Merkle Tree, https://en.wikipedia.org/wiki/Merkle_tree

50. TwoPI, Second Pre-image Attack on Merkle Trees, https://flawed.net.nz/2018/02/21/attacking-merkle-trees-with-a-second-preimage-attack/

51. MTSec, Merkle Tree Security, https://blog.enuma.io/update/2019/06/10/merkle-trees-not-that-simple.html

52. DSig, Digital Signature, https://en.wikipedia.org/wiki/Digital_signature

53. Level, Security Level, https://en.wikipedia.org/wiki/Security_level

54. Twin, Digital Twin, https://en.wikipedia.org/wiki/Digital_twin

55. TMal, Transaction Malleability, https://en.wikipedia.org/wiki/Transaction_malleability_problem

56. PGM, The Property Graph Database Model, http://ceur-ws.org/Vol-2100/paper26.pdf

57. Dots, Constructions from Dots and Lines, https://arxiv.org/pdf/1006.2361.pdf

58. KG, Knowledge Graphs, https://arxiv.org/pdf/2003.02320.pdf

59. Abuse, Alice Attempts to Abuse a Verifiable Credential, https://github.com/WebOfTrustInfo/rwot9-prague/blob/master/final-documents/alice-attempts-abuse-verifiable-credential.md

60. SKEM, On using the same key pair for Ed25519 and an X25519 based KEM, https://eprint.iacr.org/2021/509

61. Verifiable Data Structures, Sparse Merkle Tree, https://github.com/google/trillian/blob/master/docs/papers/VerifiableDataStructures.pdf

62. Certificate Transparency, Append Only Logs, https://www.researchgate.net/publication/274056469_Certificate_Transparency

63. IETF RFC-9162, Certificate Transparency, https://datatracker.ietf.org/doc/rfc9162/

64. Certificate Transparency Community, https://certificate.transparency.dev/community/

65. Trillion Sparse Merkle Tree, https://github.com/google/trillian?tab=readme-ov-file

66. Trillian Tessera, Tiled Transparency Logs, https://github.com/transparency-dev/tessera

67. Efficient Sparse Merkle Tree, https://eprint.iacr.org/2016/683.pdf

68. Optimized Compact Sparse Merkle Tree, https://github.com/nervosnetwork/sparse-merkle-tree/blob/master/SMT.md

69. ToIP Trust Spanning Protocol (TSP), https://trustoverip.github.io/tswg-tsp-specification/

70. Secure Privacy, Authenticity, Confidentiality (SPAC), https://github.com/Smith-SamuelM/Papers/blob/master/whitepapers/