CESR
Composable Event Streaming Representation
for
CESR/KERI/ACDC/SPAC-TSP Protocol Stack
and other
Cryptographically Agile
and
Cryptographically Heavy Protocols

Composable cryptographic material primitives and groups in dual text and binary streaming protocol design

Samuel M. Smith Ph.D.
Sam@keri.one

2024/04/17

Resources

Documentation:

ToIP Public Review Specs

https://github.com/trustoverip/tswg-cesr-specification
https://github.com/trustoverip/tswg-keri-specification
https://github.com/trustoverip/tswg-acdc-specification
https://github.com/trustoverip/tswg-did-method-webs-specification

ToIP Public Draft Specs

https://github.com/trustoverip/tswg-tsp-specification

Community: (meetings, open source code, IETF internet drafts)

https://github.com/WebOfTrust
https://github.com/WebOfTrust/keri
https://keri.one/keri-resources/

Protocol Formats

Internet Protocols

Binary

UDP, TCP, DNS, RTP, RTCP, NTP, SNMP, BGP, BGMP, ARP, IGMP, RIP, PING, WebRTC

Text *(line based)*

Syslog, SMTP, POP, Telnet, NNTP, RTSP, IRC

Text *(header framed)*

HTTP, SIP

Inflection Point in Protocol Design *(dual representation)*

XML, JSON (text self describing map)

MGPK, CBOR (binary self describing map)

Cryptographic Protocols

JSON or Hybrid JSON/MGPK

RAET, Secure Scuttlebutt, DIDCom

JSON/JOSE/JWT Family

Binary

RLPx (Ethereum)

Bitcoin

CBOR/COSE/CWT Family

Flexible Concatenable Binary Crypto Material

Noise (Signal)

Flexible Composable Concatenable Text/Binary Crypto Material

CESR Composable Event Streaming Representation

Protocol Formats

Truly secure protocols are both cryptographically agile and cryptographically primitive-heavy.

High cryptographic usability and performance drive enhanced security.

It avoids the false dilemma of trading down security for better developer usability and performance.

Binary vs Text Based Protocol Features

Binary Format

Advantages:

Compact, efficient, and performant

Disadvantages:

Difficult to develop, test, debug (over the wire), prove compliance, and extremely difficult to fix and version

Requires custom tooling especially over the wire debug, difficult to understand, and *hard to gain adoption*.

Text Format (especially self-describing field map based)

Advantages:

Easy to develop, test, debug (especially over the wire), and prove compliance, and extremely easy to fix and version

Requires little custom tooling especially over the wire debug, easy to understand and *easy to gain adoption.*

Disadvantages:

Verbose, Inefficient, Non-performant

Hybrid Both Text & Binary Formats (self describing map, Text=JSON and Binary=MGPK or CBOR)

Advantages:

Zero cost to switch between text and binary. Text for development and adoption. Binary for production use.

Easy to develop, test, debug (especially over the wire), and prove compliance when text

Requires little custom tooling especially over the wire debug, easy to understand

Extremely easy to fix and version

Easy to adopt as text with no additional barrier to binary adoption

Fairly compact when binary, Fairly efficient when binary, fairly performant when binary

Composable Concatenable (Text/Binary) Event Streaming Representation (CESR)

Hybrid Flexible Concatenated Compact Text Base64 & Binary Formats

Advantages:

Composable *(any concatenated block in one format may be converted as a block to the other without loss, round trippable)*

Zero cost to switch between text and binary. Text for development and adoption. Binary for production use.

Flexible concatenation of heterogenous crypto material that preserves byte/char boundaries between primitives

Pipelined parsing and processing

Stream or Datagram

Fully qualified self-framing derivation codes for primitives.

Fully qualified self-framing count codes for groups of primitives or groups of groups.

Fully qualified self-framing count codes for pipelining groups.

More compact text and binary than hybrid text and binary self-describing map based formats

Fairly Easy to develop, test, debug (especially over the wire), and prove compliance when text

Requires little custom tooling, especially over-the-wire debugging of text

Fairly easy to understand when text

Archivable audit-compliant, legally defensible text format

Fairly easy to adopt as text with no additional barrier to binary adoption

Compact when binary, Efficient when binary, Performant when binary

Disadvantages:

Almost as compact, efficient, and performant as non-composable tuned binary. More complex parsing.

Three native domains and formats

Raw Domain (text code, raw binary) (cryptographic operations)

Raw binary representation that crypto libraries use

Text Domain

fully qualified text, name-space able, streamable, archivable, envelopeable

Cryptographic material: Cryptonyms (identifiers), public keys, digests, signatures etc.

Includes any textual use of cryptographic material, Documents, VCs, Archives, Audit logs etc.

Usable in over the wire streaming for development and debug.

```
DKrJxkcR9m5u1xs33F5pxRJP6T7hJEbhpHrUtlDdhh0
```

Binary Domain (fully qualified binary) (streaming binary)

Fully qualified binary representation of cryptographic material for efficient over the wire streaming

Round Trippable Closed loop Transformation

Fully qualified means prepended derivation code.

In Text domain, readability is enhanced if pre-pended derivation code is stable (left aligned) and is not changed by post-pended crypto material value.

In Text domain, readability is enhanced if post-pended crypto material is stable (right aligned) and is not changed by pre-pended code

Text domain, T, is fully qualified Base64. Binary domain, S, is fully qualified base2 equivalent (conversion) of T.

Composability Property: transformation (round trip) between T and S of concatenated primitives does not cross primitive boundaries. Separable parseability is preserved.

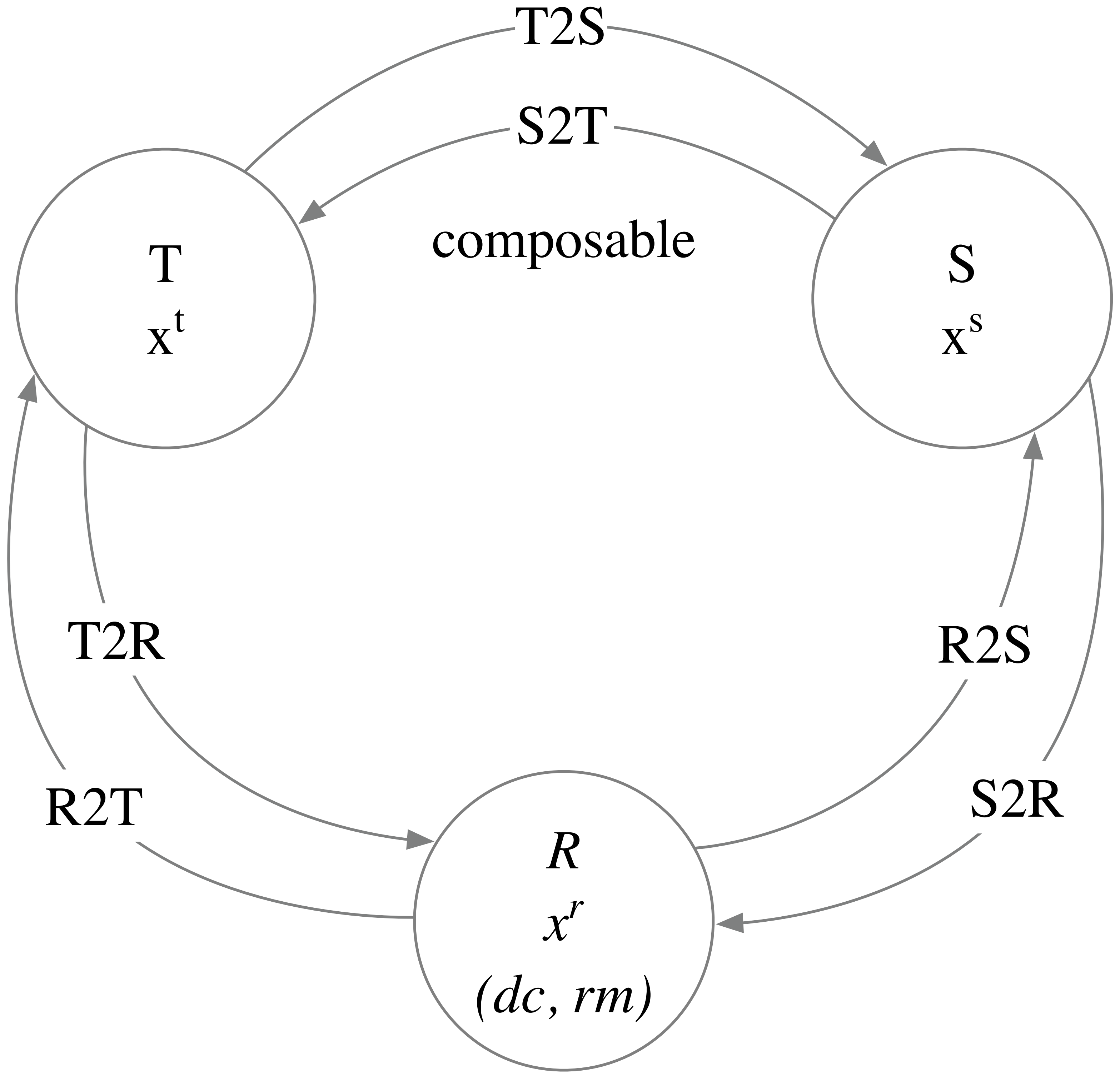
Normally composability requires pad characters (pad bytes) on each Base64 (Base2) primitive.

KERI replaces pad characters with prepend derivation codes whose length preserves composability.

By comparison did:key, did:peer are neither stable nor composable.

In general, Multi-Codec is not stable nor composable except for serendipitous combinations of code and value

(dc, rm) -> R2T -> t -> T2S -> s -> S2R -> (dc, rm)
(dc, rm) -> R2S -> s -> S2T -> t -> T2R -> (dc, rm)



$$x^t$$

$$x^s$$

$$x^t = T\left(x^s\right)$$

$$x^s = S\left(x^t\right)$$

$$x^t + y^t = T\left(x^s + y^s\right)$$

$$x^s + y^s = S\left(x^t + y^t\right)$$

$$S\left(\sum_{i=0}^{N-1} x_i^t\right) = \sum_{i=0}^{N-1} x_i^s$$

$$T\left(\sum_{i=0}^{N-1} x_i^s\right) = \sum_{i=0}^{N-1} x_i^t$$

Why Composability

Verifiable Text Stream = Verifiable Binary Stream (no loss of verifiability in mass conversion)

Amount of cryptographic material in attachments may exceed cryptographic material in message bodies. Controller signatures, witness signatures, other receipt signatures, endorsement signatures.

Replay of KELs must replay messages (key events) plus attachments (signatures)

Want this replay to be compact, performant, and supported by bare metal protocols (TCP, UDP)

Verifiable Credential world is Text

Verifiable Authentic Data world human facing side is Text

Verifiable Digitally Signed Contract world is Text

Verifiable Audit Trail World is Text

Code Tables

3 classes of stable composable derivation codes:

Basic material primitives,

Indexed signature primitives or variable length values,

Grouping count codes.

CESR Code Table Schemes

Selector	Selector	Type Chars	Value Size Chars	Code Size	Lead Bytes	Pad Size	Format (Minimal Size)
[A-Z,a-z]		1*	0	1	0	1	$&&&
0		1	0	2	0	2	0$&&
1		3	0	4	0	0	1$$$&&&&
2		3	0	4	1	1	2$$$&&&&
3		3	0	4	2	2	3$$$&&&&
4		1	2	4	0	0	4$##&&&&
5		1	2	4	1	1	5$##&&&&
6		1	2	4	2	2	6$##&&&&
7		3	4	8	0	0	7$$$####&&&&
8		3	4	8	1	1	8$$$####&&&&
9		3	4	8	2	2	9$$$####&&&&
-	[A-Z,a-z]	1*	0	4	0	0	-$##
-	0	2	0	8	0	0	-0$$####
_	TBD	TBD	TBD	TBD	TBD	TBD	_

* selector character is also type character¶
$ means type code character from subset of Base64 [A-Z,a-z,0-9,-,_].
# means a Base64 digit as part of a base 64 integer that determines the number of following quadlets or triplets in the primitive or when part of a count code, the count of following primitives or groups of primitives.
& represents one or more Base64 value characters representing the converted raw binary value included lead bytes when applicable. The actual number of chars is determined by the prep-ended text code.

Parsing Size Tables

Size Table Selector

selector	hs
B	1
0	2
5	2

Parse Size Table

hard sized index	hs	ss	vs	fs	ls	ps
B	1	0	43*	44	0	1
0B	2	0	86*	88	0	2*
5A	2	2	#	#	1	1*

* size may be calculated from other sizes.
# size may be calculated from extracted code characters given by other sizes.¶

hs means hard size in chars.
ss means soft size in chars.
cs means code size where $cs = hs + ss$.
vs means value size in chars.
fs means full size in chars where $fs = hs + ss + vs$.
ls means lead size in bytes.
ps means pad size in chars.
rs means raw size in bytes of binary value.
bs means binary size in bytes where $bs = as + rs$.¶

Special Context Tables

Special Indexed Primitive Table

Selector	Select or	Type Chars	Index Chars	Code Size	Lead Bytes	Pad Size	Format
[A-Z,a-z]		1*	1	2	0	2	$#&&
0		1	2	4	0	0	0$##&&&&

* selector character is also type character¶

$ means type code character from subset of Base64 [A-Z,a-z,0-9,-,_].
# means a Base64 digit as part of a base 64 integer that determines the index.
& represents one or more Base64 value characters representing the converted raw binary value included lead bytes when applicable. The actual number of chars is determined by the prep-ended text code.

Example Simple Inception Key Event Message

```
{
        'v': 'KERICAACESRAAAA.',
        't': 'icp',
        'd': 'EEx4oxGYbNrd6nZsdGu2KdN4MSDGD5IWS7hXjST7r8ew',
        'i': 'DG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQ',
        's': '0',
        'kt': '1',
        'k': ['DG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQ'],
        'nt': '0',
        'n': [],
        'bt': '0',
        'b': [],
        'c': [],
        'a': []
}
```

Example Simple Inception Key Event Message

```
-FAtYKERICAAXicpEEx4oxGYbNrd6nZsdGu2KdN4MSDGD5IWS7hXjST7r8ewDG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4f
BJre3NGwTQMAAAMAAB-IALDG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQMAAA-IAAMAAA-IAA-IAA-IAA')
```

```
-FAt # Key Event Counter FixedMessageBodyGroup count=45 quadlets
  YKERICAA # 'v' version  Verser Tag7 proto=KERI vrsn=2.00
  Xicp # 't' message type Ilker Tag3 Ilk=icp
  EEx4oxGYbNrd6nZsdGu2KdN4MSDGD5IWS7hXjST7r8ew # 'd' SAID Diger Blake3_256
  DG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQ # 'i' AID Prefixer Ed25519
  MAAA # 's' Number Short sn=0
  MAAB # 'kt' Tholder signing threshold=1
  -IAL # 'k' Signing Key List Counter GenericListGroup count=11 quadlets
    DG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQ # key Verfer Ed25519
  MAAA # 'nt' Tholder rotation threshold=0
  -IAA # 'n' Rotation Key Digest List Counter GenericListGroup count=0 quadlets
  MAAA # 'bt' Tholder Backer (witness) threshold=0
  -IAA # 'b' Backer (witness)List Counter GenericListGroup count=0 quadlets
  -IAA # 'c' Config Trait List Counter GenericListGroup count=0 quadlets
  -IAA # 'a' Seal List Counter GenericListGroup count=0 quadlets
```

De-annotation

```
def denot(ams):
    """De-annotate CESR stream

    Returns:
        dms (bytes):  deannotation of input annotated CESR message stream

    Parameters:
        ams (str):  CESR annotated message stream text
    """
    dms = bytearray()  # deannotated message stream
    lines = ams.splitlines()
    for line in lines:
        line = line.strip()
        front, sep, back = line.partition('#') # finde comment if any
        front = front.strip()  # non-commented portion strip white space
        if front:
            dms.extend(front.encode())

    return bytes(dms)
```

Example Inception Key Event Message

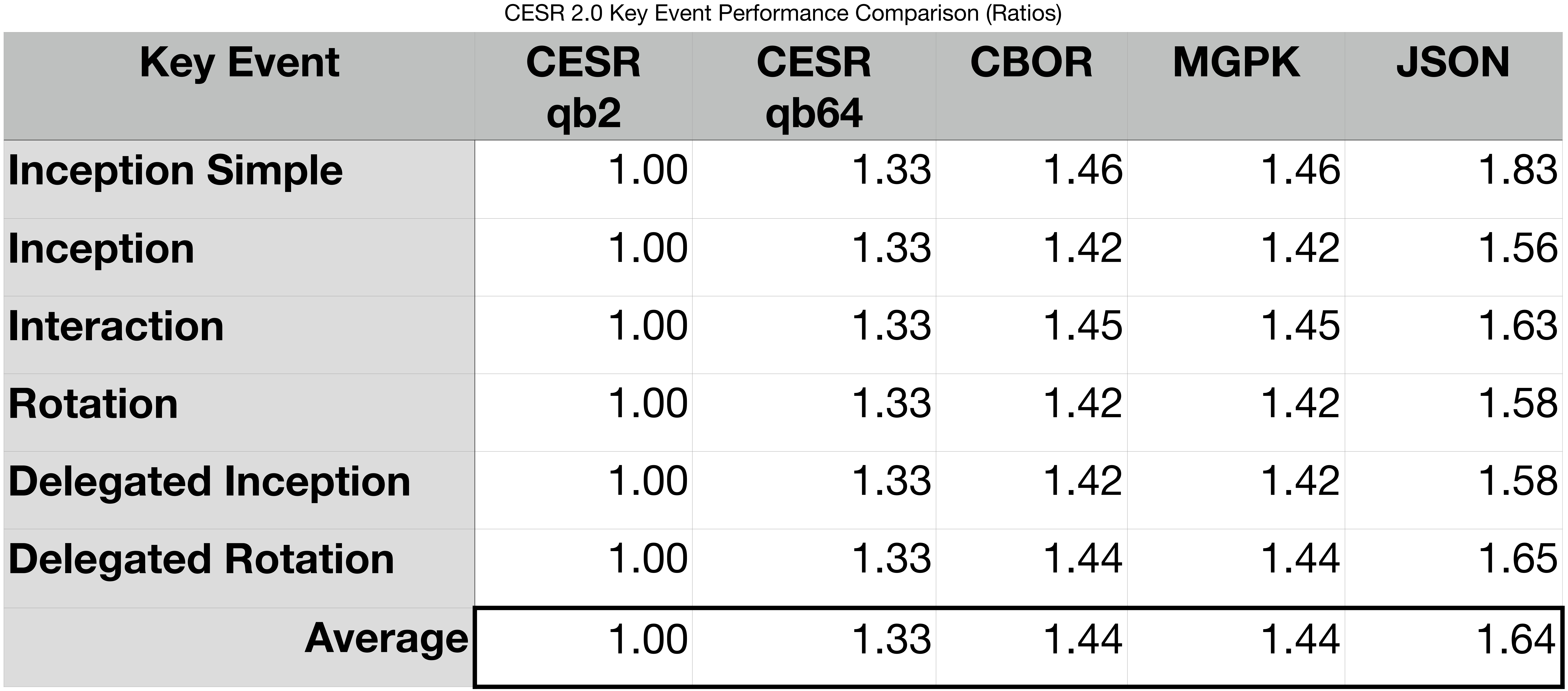
```
{
        'v': 'KERICAACESRAAAA.',
        't': 'icp',
        'd': 'EKIuA20I5q6IrgAHrX-gkAt4Og17Ebu5CDBrRvh8RToi',
        'i': 'EKIuA20I5q6IrgAHrX-gkAt4Og17Ebu5CDBrRvh8RToi',
        's': '0',
        'kt': '2',
        'k':
        [
            'DG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQ',
            'DK58m521o6nwgcluK8Mu2ULvScXM9kB1bSORrxNSS9cn',
            'DMOmBoddcrRHShSajb4d60S6RK34gXZ2WYbr3AiPY1M0'
        ],
        'nt': '2',
        'n':
        [
            'EB9O4V-zUteZJJFubu1h0xMtzt0wuGpLMVj1sKVsElA_',
            'EMrowWRk6u1imR32ZNHnTPUtc7uSAvrchIPN3I8S6vUG',
            'EEbufBpvagqe9kijKISOoQPYFEOpy22CZJGJqQZpZEyP'
        ],
        'bt': '3',
        'b':
        [
            'BG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQ',
            'BK58m521o6nwgcluK8Mu2ULvScXM9kB1bSORrxNSS9cn',
            'BMOmBoddcrRHShSajb4d60S6RK34gXZ2WYbr3AiPY1M0'
        ],
        'c': ['DND'],
        'a':
        [
            {
                'i': 'DG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQ',
                's': '0',
                'd': 'EB9O4V-zUteZJJFubu1h0xMtzt0wuGpLMVj1sKVsElA_'
            },
            {
                'i': 'DK58m521o6nwgcluK8Mu2ULvScXM9kB1bSORrxNSS9cn',
                's': '1',
                'd': 'EMrowWRk6u1imR32ZNHnTPUtc7uSAvrchIPN3I8S6vUG'
            },
            {
                's': 'f',
                'd': 'EEbufBpvagqe9kijKISOoQPYFEOpy22CZJGJqQZpZEyP'
            }
        ]
    }
```

```
-FDCYKERICAAXicpEKIuA20I5q6IrgAHrX-
gkAt4Og17Ebu5CDBrRvh8RToiEKIuA20I5q6IrgAHrX-
gkAt4Og17Ebu5CDBrRvh8RToiMAAAMAAC-
IAhDG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQDK58m521o6nwgcluK8Mu2U
LvScXM9kB1bSORrxNSS9cnDMOmBoddcrRHShSajb4d60S6RK34gXZ2WYbr3AiPY1M0MAA
C-IAhEB9O4V-
zUteZJJFubu1h0xMtzt0wuGpLMVj1sKVsElA_EMrowWRk6u1imR32ZNHnTPUtc7uSAvrc
hIPN3I8S6vUGEEbufBpvagqe9kijKISOoQPYFEOpy22CZJGJqQZpZEyPMAAD-
IAhBG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQBK58m521o6nwgcluK8Mu2U
LvScXM9kB1bSORrxNSS9cnBMOmBoddcrRHShSajb4d60S6RK34gXZ2WYbr3AiPY1M0-
IABXDND-IA8-
RAuDG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQMAAAEB9O4V-
zUteZJJFubu1h0xMtzt0wuGpLMVj1sKVsElA_DK58m521o6nwgcluK8Mu2ULvScXM9kB1
bSORrxNSS9cnMAABEMrowWRk6u1imR32ZNHnTPUtc7uSAvrchIPN3I8S6vUG-
QAMMAAPEEbufBpvagqe9kijKISOoQPYFEOpy22CZJGJqQZpZEyP
```

Example Inception Key Event Message

```
-FDC # Key Event Counter FixedMessageBodyGroup count=194 quadlets
  YKERICAA # 'v' version  Verser Tag7 proto=KERI vrsn=2.00
  Xicp # 't' message type Ilker Tag3 Ilk=icp
  EKIuA20I5q6IrgAHrX-gkAt4Og17Ebu5CDBrRvh8RToi # 'd' SAID Diger Blake3_256
  EKIuA20I5q6IrgAHrX-gkAt4Og17Ebu5CDBrRvh8RToi # 'i' AID Prefixer Blake3_256
  MAAA # 's' Number Short sn=0
  MAAC # 'kt' Tholder signing threshold=2
  -IAh # 'k' Signing Key List Counter GenericListGroup count=33 quadlets
    DG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQ # key Verfer Ed25519
    DK58m521o6nwgcluK8Mu2ULvScXM9kB1bSORrxNSS9cn # key Verfer Ed25519
    DMOmBoddcrRHShSajb4d60S6RK34gXZ2WYbr3AiPY1M0 # key Verfer Ed25519
  MAAC # 'nt' Tholder rotation threshold=2
  -IAh # 'n' Rotation Key Digest List Counter GenericListGroup count=33 quadlets
    EB9O4V-zUteZJJFubu1h0xMtzt0wuGpLMVj1sKVsElA_ # key digest Diger Blake3_256
    EMrowWRk6u1imR32ZNHnTPUtc7uSAvrchIPN3I8S6vUG # key digest Diger Blake3_256
    EEbufBpvagqe9kijKISOoQPYFEOpy22CZJGJqQZpZEyP # key digest Diger Blake3_256
  MAAD # 'bt' Tholder Backer (witness) threshold=3
  -IAh # 'b' Backer (witness)List Counter GenericListGroup count=33 quadlets
    BG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQ # AID Prefixer Ed25519N
    BK58m521o6nwgcluK8Mu2ULvScXM9kB1bSORrxNSS9cn # AID Prefixer Ed25519N
    BMOmBoddcrRHShSajb4d60S6RK34gXZ2WYbr3AiPY1M0 # AID Prefixer Ed25519N
  -IAB # 'c' Config Trait List Counter GenericListGroup count=1 quadlets
    XDND # trait Traitor Tag3 trait=DND
  -IA8 # 'a' Seal List Counter GenericListGroup count=60 quadlets
    -RAu # Seal Counter SealSourceTriples count=46 quadlets
      DG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQMAAAEB9O4V-zUteZJJFubu1h0xMtzt0wuGpLMVj1sKVsElA_# seal Sealer SealEvent
        #  'i' = DG9XhvcVryHjoIGcj5nK4sAE3oslQHWi4fBJre3NGwTQ
        #  's' = 0
        #  'd' = EB9O4V-zUteZJJFubu1h0xMtzt0wuGpLMVj1sKVsElA_
      DK58m521o6nwgcluK8Mu2ULvScXM9kB1bSORrxNSS9cnMAABEMrowWRk6u1imR32ZNHnTPUtc7uSAvrchIPN3I8S6vUG# seal Sealer SealEvent
        #  'i' = DK58m521o6nwgcluK8Mu2ULvScXM9kB1bSORrxNSS9cn
        #  's' = 1
        #  'd' = EMrowWRk6u1imR32ZNHnTPUtc7uSAvrchIPN3I8S6vUG
    -QAM # Seal Counter SealSourceCouples count=12 quadlets
      MAAPEEbufBpvagqe9kijKISOoQPYFEOpy22CZJGJqQZpZEyP# seal Sealer SealTrans
        #  's' = f
        #  'd' = EEbufBpvagqe9kijKISOoQPYFEOpy22CZJGJqQZpZEyP
```

Size Comparison - Key Event Messages



CESR 2.0 Key Event Performance Comparison (Ratios)

Key Event	CESR qb2	CESR qb64	CBOR	MGPK	JSON
Inception Simple	1.00	1.33	1.46	1.46	1.83
Inception	1.00	1.33	1.42	1.42	1.56
Interaction	1.00	1.33	1.45	1.45	1.63
Rotation	1.00	1.33	1.42	1.42	1.58
Delegated Inception	1.00	1.33	1.42	1.42	1.58
Delegated Rotation	1.00	1.33	1.44	1.44	1.65
Average	1.00	1.33	1.44	1.44	1.64

Crypto-Agile Comparison JSON/JOSE/JWK CBOR/COSE

- Interior primitives inside messages are typically public keys and digests (Hashes).
- Crypto Agility for Cryptographic Primitives in JSON using JOSE and CBOR using COSE.

JSON/JOSE/CBOR/COSE Family: JWT/JWS/JWE/JWK /CWT

JWT rfc-7519 https://datatracker.ietf.org/doc/html/rfc7519

JWS rfc-7515 https://datatracker.ietf.org/doc/rfc7515/

JWK rfc-7517 https://datatracker.ietf.org/doc/html/rfc7517

JOSE/COSE RFC-7520https://datatracker.ietf.org/doc/rfc7520/

CWT rfc-8392 https://datatracker.ietf.org/doc/html/rfc8392

COSE rfc-8152 https://datatracker.ietf.org/doc/html/rfc8152

CBOR rfc-8949 https://www.rfc-editor.org/rfc/rfc8949.html (Diagnostic Notation)

CBOR Stable Tag https://www.rfc-editor.org/rfc/rfc9277.html

CBOR Sequences (streams) rfc-8742 https://www.rfc-editor.org/rfc/rfc8742.html

- The JOSE/COSE family only provides normative support for a limited set of key public types.

ECDSA EDDSA RSA HMAC DES

https://connect2id.com/products/nimbus-jose-jwt/examples/jwk-generation

https://pycose.readthedocs.io/en/latest/pycose/keys/ec2.html

- Currently, JOSE and COSE do not provide normative support for Interior Hashes (not exterior HMACs as reputable signatures). But one could use a non-normative approach with similar data structure for digest algorithms or comparable size (256 bit) as Public keys.

JWK for Crypto Agility

Inception Example: 16 interior primitives of size of a key

Overhead of JWK:

```
djwk = {
        "kty" : "OKP",
        "crv" : "Ed25519",
        "x"   : "11qYAYKxCrfVS_7TyWQHOg7hcvPapiMlrwIaaPcHURo",
        "d"   : "nWGxne_9WmC6hEr0kuwsxERJxWl7MmkZcDusAxyuf2A",
        "use" : "sig",
        "kid" : "FdFYFzERwC2uCBB46pZQi4GG85LujR8obt-KWRBICVQ"
      }
```

Compact JSON =

```
{“kty":"OKP","crv":"Ed25519","x":"11qYAYKxCrfVS_7TyWQHOg7hcvPapiMlrwIaaPcHURo","d":"nWGxne_9WmC6hEr0kuwsxERJxWl7MmkZcDusAxyuf2A","use":"sig","kid":"FdFYFzERwC2uCBB46pZQi4GG85LujR8obt-KWRBICVQ"}
```

Size =193 bytes

Overhead per primitive = 193-44 =149 bytes

Overhead of using JWK vs CESR = 16*149 = 2384

Overhead ratio (JSON with JWK to JSON with CESR) = (2384+915) / 915 = 3.61

Overhead ratio (JSON with JWK to qb2 CESR) = (2384+915) / 585 = 5.64

COSE for Crypto Agility

Inception Example: 16 interior primitives of size of a key

Overhead of COSE Key:

```
{
        'KTY': 'EC2',
        'CURVE': 'P_256',
        'ALG': 'ES256',
        'D': unhexlify(b'57c92077664146e876760c9520d054aa93c3afb04e306705db6090308507b4d3')
 }
```

```
ck = \xa6\x01\x02\x03&\x01!X\xba\xc5\xb1\x1c\xad\x8f\x99\xf9\xc7+\x05\xcfK\x9e&\xd2D\xdc\x18\x9ftR(%Z!
\x9a\x86\xd6\xa0\x9e\xffX\x13\x8b\xf8-\xc1\xb6\xd5b\xbe\x0f\xa5J\xb7\x80J:d\xb6\xd7,\xcf\xedko\xb6\xed(\xbb\xfc\x11~#XW\xc9
wfAF\xe8vv\x0c\x95\xd0T\xaa\x93\xc3\xaf\xb0N0g\x05\xdb`\x900\x85\x07\xb4\xd3
```

len(ck) == 105
 105 - 36 == 69

Size = 105 bytes

Overhead per primitive = 105-36 = 69 bytes

Overhead of using COSE vs CESR = 16*69 = 1104

Overhead ratio (CBOR with COSE to CBOR with CESR) = (1104+829) / 829 = 2.33

Overhead ratio (CBOR with COSE to qb2 CESR) = (1104+829) / 585 = 3.30

TSP Example

https://github.com/WebOfTrust/keripy/discussions/612

Tritet Parsing

CESR Parser Tritet Table

Starting Tritet	Serialization	Character
0b000	Annotated CESR	/t, /n, /r
0b001	CESR *T* Domain Count (Group) Code	-
0b010	CESR *T* Domain Op Code	_
0b011	JSON	{
0b100	MGPK	
0b101	CBOR	
0b110	MGPK	
0b111	CESR *B* Domain	

Stream Parsing Rules

Stream start, cold restart, message end, group end:

Examine tritet (3 bits).

Each stream must start (restart) with one of 6 things:

- Framing count code in either Base64 or Binary.
- Framing opcode in either Base64 or Binary
- JSON encoded mapping.
- CBOR encoded Mapping.
- MGPK encoded mapping.
- Annotated Base64

A parser merely needs to examine the first tritet (3 bits) of the first byte of the stream start to determine which one of the 6 it is.

When the first tritet indicates its JSON, CBOR, or MGPK, then the included version string provides the remaining and confirming information needed to fully parse the associated encoded message.

When the first tritet is a framing code then, the remainder of framing code itself will include the remaining information needed to parse the attached group.

The framing code may be in either Base64 or binary.

When the first trivet is annotated qb64 then parser de-annotates stream/file before parsing as above.

At the end of the stream start, the stream must resume with one of the 6 things, either a new JSON, CBOR, or MGPK encoded mapping or another of two types of framing codes expressed in either Base64 or binary or annotated qb64.

Archival Preservation

The ISO ISO 14641:2018 standard for the preservation of electronic documents, lists four important features of a legally defensible archive [33][34]. These are long-term preservation, data integrity, data security, and traceability.

A KERI Key Event Receipt Log (KERL) is already in a form that provides data integrity, data security, and traceability.

All that is needed is to ensure long-term preservation capability.

The standard formats for long-term document archival are by-in-large text based (with some exceptions) [30].

What this means is that a KERI event log stream in a native text format is inherently compatible with archival requirements. Indeed because a KERL text stream with composition operators only uses the Base64URL character set, that is,[A-Z,a-z,0-9,-,_]

`Annotated KELs`

Any of the other characters in the ASCII set may be used to loss-lessly annotate a KERI text event stream.

These include white space characters.

Primitives and groups of framed primitives within the text KERL could then be line delimited and white spaced and comments added using the “#” symbol for example.

A text parser could easily strip all non Base-64 characters, line delimiters and comments to reconstitute the KERL stream which could then be converted en mass for transmission.

The archivist never need access or convert to the raw binary format.

Legally Binding Digital Signatures (Contracts)

The relevant legislation for legally compliant electronic signatures are the USA Electronic Signatures in Global and National Commerce Act (ESIGN), the USA Uniform Eletronic Transactions Act (UETA) and the EU Regulation for Electronic Identification and Electronic Trust Services (eIDAS) [27][28][29].

The legislations have similar conditions at least those relevant to KERI.

When the set of legally defined conditions are met, a cryptographic digital signature based on asymmetric key pairs has legal standing.

Key pairs used to control a KERI self-certifying identifier may also be used to create legally binding electronic signatures on electronic documents.

Establishment of control authority is provided by cryptographically verifiable proof of key state, that is, a proof that a given set of key pairs are the authoritative ones.

This proof is expressed as a KERI KERL (Key Event Receipt Log) which is a hash chained signed data structure.

The most relevant condition (to KERI) that a legally binding signature must satisfy is proof or assurance that the entity creating the electronic signature is also the entity that is the controller of the associated private keys.

A KERI KEL may be used to support that proof or assurance.

An annotated text domain KEL could be attached to the signed legal document as an appendix or provided to an electronic signature notary.

This text based annotated KEL appendix could be archived with all the associated legal documents.

This greatly facilitates the broader adoption of electronic signatures and KERI.

This could also help reduce trust transaction costs for the authentic data economy.

Audit Trails

An audit log is used to provenance something.

An annotated text domain KERL could be used to support a securely attributed tamper evident archival audit trail [24][25][26].

Composability would allow archival in streaming text form while also enabling efficient transmission in streaming binary form.

Electronic Code of Federal Regulations: Electronic Signatures (E-CFR) regulation requires non-repudiable audit trial attribution[25]:

E-CFR

(e) Use of secure, computer-generated, time-stamped audit trails to independently record the date and time of operator entries and actions that create, modify, or delete electronic records. Record changes shall not obscure previously recorded information. Such audit trail documentation shall be retained for a period at least as long as that required for the subject electronic records and shall be available for agency review and copying.

(f) Use of operational system checks to enforce permitted sequencing of steps and events, as appropriate.

(g) Use of authority checks to ensure that only authorized individuals can use the system, electronically sign a record, access the operation or computer system input or output device, alter a record, or perform the operation at hand.

(j) The establishment of, and adherence to, written policies that hold individuals accountable and responsible for actions initiated under their electronic signatures, in order to deter record and signature falsification.

The Alliance for Telecommunications Industry Solutions (ATIS) provides various definitions for audit trail as follows [26]:

A set of records that collectively provide documentary evidence of processing used to aid in tracing from original transactions forward to related records and reports, and/or backwards from records and reports to their component source transactions.

KERI supports secure attribution via non-repudiable signatures on any data in such a compliance log.

The key events in the text KEL may be interspersed in the audit trail so that the timing of key events that rotate keys may be correlated to the audit log signed data. The audit trail may be viewed as an annotated KEL.

A text processor could extract the KERL from any archived audit trail and then convert it to streaming binary for compact transmission to a processor to perform the cryptographic verification operations.

This provides efficient scalable provenance.