Remote ATtestation procedureS
Internet-Draft
Intended status: Informational
Expires: 10 August 2026

D. Condrey
Writerslogic Inc

6 February 2026

Proof of Process: Worked Examples
draft-condrey-rats-pop-examples-00

Abstract

This document provides worked examples demonstrating the Proof of Process Evidence format and Attestation Results. Examples include minimal Evidence packets, multi-checkpoint scenarios, jitter seal verification, VDF causality chains, and salt mode configurations. This companion document supplements the main Proof of Process specification with practical reference implementations.

Table of Contents

1. Introduction

This document provides worked examples of Proof of Process (PoP) Evidence packets and Attestation Results as defined in [I-D.condrey-rats-pop]. All examples use CBOR diagnostic notation [RFC8949] Section G, with comments (/ ... /) to annotate fields. Integer keys are used as defined in the companion CDDL schema.

These examples are informative. The normative schema is defined in the CDDL schema document. Implementers SHOULD validate their implementations against test vectors derived from these examples.

1.1. Requirements Language

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

2. Minimal Evidence Packet (Basic Tier)

This example demonstrates a minimal Basic tier Evidence packet with a single checkpoint. The document is approximately 500 characters, representing a short paragraph written in a single authoring session.

Scenario: An author writes a brief memo over approximately 3 minutes. The Attesting Environment captures one checkpoint at the end of the session.

```
1347571280({
  1: 1,
  2: "https://writerslogic.com/rats/eat/profile/pop/1.0",
  3: h'550e8400e29b41d4a716446655440000',
  4: 1(1706745600),

  5: {
    1: {
      1: 1,
      2: h'e3b0c44298fc1c149afbf4c8996fb924
          27ae41e4649b934ca495991b7852b855'
    },
    2: "memo.txt",
    3: 487,
    4: 478,
    5: 0
  },

  6: [
    {
      1: 0,
      2: h'6ba7b8109dad11d180b400c04fd430c8',
      3: 1(1706745780),

      4: {
        1: 1,
        2: h'e3b0c44298fc1c149afbf4c8996fb924
            27ae41e4649b934ca495991b7852b855'
      },
```

```
  5: 478,
  6: 87,

  7: {
    1: 478, 2: 12, 3: 47, 4: 8,
    5: 3, 6: 0, 7: 0, 8: 14, 9: 180.0
  },

  8: {
    1: 1,
    2: h'00000000000000000000000000000000
        00000000000000000000000000000000'
  },

  9: {
    1: 1,
    2: h'a7ffc6f8bf1ed76651c14756a061d662
        f580ff4de43b49fa82d80a4b80f8434a'
  },

  10: {
    1: 1,
    2: {1: 1, 2: 8500000},
    3: h'9f86d081884c7d659a2feaa0c55ad015
        a3bf4f1b2b0b822cd15d6c15b0f00a08',
    4: h'2c624232cdd221771294dfbb310aca00
        0a0df6ac8b66b696d90ef06fdefb64a3',
    5: h'',
    6: 180.0,
    7: 1530000000,
    8: {
      1: 8500000,
      2: 1(1706745600),
      3: h'deadbeef...',
      4: h'cafebabe...',
      5: "MacBook Pro M3"
    }
  },

  11: {
    1: {
      1: 1,
      2: h'7d865e959b2466918c9863afca942d0f
          b89d7c9ac0c99bafc3749504ded97730'
    },
    2: (1, 2, 3),
    3: {
      1: 423,
```

```
            2: [
              {1: 0, 2: 50, 3: 12},
              {1: 50, 2: 100, 3: 89},
              {1: 100, 2: 200, 3: 156},
              {1: 200, 2: 500, 3: 98},
              {1: 500, 2: 1000, 3: 34},
              {1: 1000, 2: 2000, 3: 18},
              {1: 2000, 2: 5000, 3: 12},
              {1: 5000, 2: 4294967295, 3: 4}
            ],
            3: 2.78,
            4: []
          },
          4: h'b94d27b9934d3e08a52e52d7da7dabfa
              c484efe37a5380ee9088f7ace2efcde9'
        },

        12: h'73475cb40a568e8da8a045ced110137e
            159f890ac4da883b6b17dc651b3a8049'
      }
    ]
  })
```

Key observations:

* The Basic tier contains only the required checkpoint chain. No optional sections (presence, forensics, etc.) are included.

* The genesis checkpoint has prev-hash of 32 zero bytes.

* The VDF proves at least 180 seconds elapsed (1.53B iterations at calibrated 8.5M iterations/second).

* Jitter binding shows 423 timing samples with realistic distribution peaking in the 100-200ms range.

* Estimated entropy is 2.78 bits, below the recommended 32-bit threshold for Standard tier but acceptable for Basic tier.

3. Multi-Checkpoint Evidence (Standard Tier)

This example demonstrates a Standard tier Evidence packet with three checkpoints showing document evolution. The document grows from 100 to 500 to 1200 characters across the checkpoints, representing a typical drafting process with revisions.

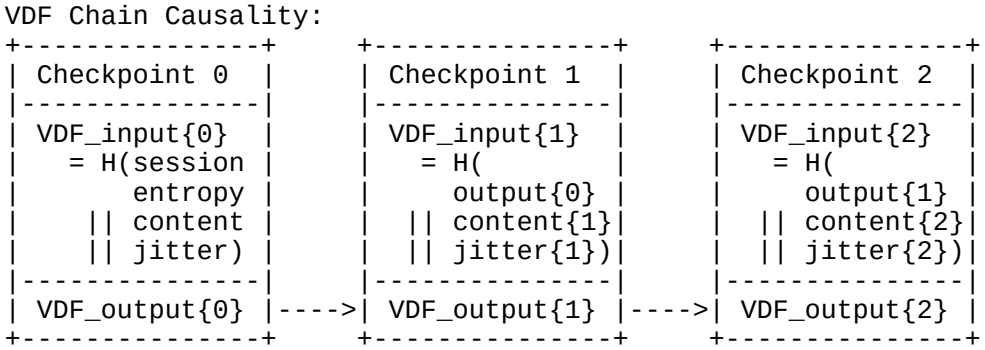
Scenario: An author writes a short essay over 45 minutes with two natural breaks where checkpoints are captured. The evidence includes presence challenges.

```
1347571280({
  1: 1,
  2: "https://writerslogic.com/rats/eat/profile/pop/1.0",
  3: h'123e4567e89b12d3a456426614174000',
  4: 1(1706832000),

  5: {
    1: {1: 1, 2: h'abcd1234...'},
    3: 1247,
    4: 1203,
    5: 0
  },

  6: [
    {
      1: 0,
      2: h'a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6',
      3: 1(1706832600),
      4: {1: 1, 2: h'1111aaaa...'},
      5: 103,
      6: 19,
      7: {
        1: 103, 2: 8, 3: 22, 4: 5,
        5: 2, 6: 0, 7: 0, 8: 6, 9: 600.0
      },
      8: {1: 1, 2: h'00000000...'},
      9: {1: 1, 2: h'2222bbbb...'},
      10: {
        1: 1,
        2: {1: 1, 2: 8500000},
        3: h'input0...',
        4: h'output0...',
        5: h'',
        6: 600.0,
        7: 5100000000,
        8: {
          1: 8500000, 2: 1(1706832000),
          3: h'sig...', 4: h'nonce...'
        }
      },
      11: {
        1: {1: 1, 2: h'jitter0...'},
        2: (1, 2),
        3: {1: 156, 2: (...), 3: 2.45},
```

```
        4: h'mac0...'
      },
      12: h'chainmac0...'
    },

    {
      1: 1,
      2: h'b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7',
      3: 1(1706833800),
      4: {1: 1, 2: h'3333cccc...'},
      5: 512,
      6: 94,
      7: {
        1: 423, 2: 14, 3: 87, 4: 12,
        5: 6, 6: 0, 7: 0, 8: 23, 9: 1200.0
      },
      8: {1: 1, 2: h'2222bbbb...'},
      9: {1: 1, 2: h'4444dddd...'},
      10: {
        1: 1,
        2: {1: 1, 2: 8500000},
        3: h'H(output0 || content-hash{1} || jitter-commitment{1})',
        4: h'output1...',
        5: h'',
        6: 1200.0,
        7: 10200000000
      },
      11: {
        1: {1: 1, 2: h'jitter1...'},
        2: (1, 2, 3),
        3: {1: 298, 2: (...), 3: 2.67},
        4: h'mac1...'
      },
      12: h'chainmac1...'
    },

    {
      1: 2,
      2: h'c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8',
      3: 1(1706835000),
      4: {1: 1, 2: h'abcd1234...'},
      5: 1203,
      6: 218,
      7: {
        1: 712, 2: 21, 3: 134, 4: 18,
        5: 8, 6: 0, 7: 0, 8: 45, 9: 1200.0
      },
      8: {1: 1, 2: h'4444dddd...'},
```

```
        9: {1: 1, 2: h'5555eeee...'},
        10: {
          1: 1,
          2: {1: 1, 2: 8500000},
          3: h'H(output1 || content-hash{2} || jitter-commitment{2})',
          4: h'output2...',
          5: h'',
          6: 1200.0,
          7: 10200000000
        },
        11: {
          1: {1: 1, 2: h'jitter2...'},
          2: (1, 2, 3, 4),
          3: {1: 387, 2: (...), 3: 2.89},
          4: h'mac2...'
        },
        12: h'chainmac2...'
      }
    ],

    10: {
      1: [
        {
          1: 1(1706832900),
          2: 1,
          3: 847,
          4: true,
          5: h'challenge-nonce-1'
        },
        {
          1: 1(1706834400),
          2: 2,
          3: 2134,
          4: true,
          5: h'challenge-nonce-2'
        }
      ],
      2: {
        1: 2,
        2: 2,
        3: 2,
        4: 1490
      }
    }
  })
```

VDF entanglement across checkpoints:



To backdate checkpoint 1, adversary must:
(1) Compute content that hashes to content-hash{1}
(2) Generate jitter that commits to jitter-commitment{1}
(3) Recompute VDF_output{1} from new VDF_input{1}
(4) Recompute VDF_output{2} (depends on output{1})
(5) Complete steps 3-4 before external anchor confirms state

4. Jitter Seal Verification Example

This example shows the complete jitter seal verification process, including histogram data, entropy calculation, and binding MAC computation.

4.1. Sample Histogram Data

```
   jitter-binding = {
     1: {
       1: 1,
       2: h'b94d27b9934d3e08a52e52d7da7dabfa
           c484efe37a5380ee9088f7ace2efcde9'
     },

     2: (1, 2, 3),

     3: {
       1: 842,
       2: [
         {1: 0,    2: 50,   3: 24},
         {1: 50,   2: 100,  3: 178},
         {1: 100,  2: 200,  3: 312},
         {1: 200,  2: 500,  3: 196},
         {1: 500,  2: 1000, 3: 68},
         {1: 1000, 2: 2000, 3: 36},
         {1: 2000, 2: 5000, 3: 22},
         {1: 5000, 2: 4294967295, 3: 6}
       ],
       3: 2.54,
       4: []
     },

     4: h'73475cb40a568e8da8a045ced110137e
         159f890ac4da883b6b17dc651b3a8049',

     5: (87, 134, 112, 98, 203, 156, 89, 167, 1243, 78, ...)
   }
```

4.2. Entropy Calculation Walkthrough

Shannon entropy is calculated from the histogram distribution:

```
   Given histogram counts: (24, 178, 312, 196, 68, 36, 22, 6)
   Total samples: 842

   (1) Calculate probabilities p(i) = count(i) / total
     p{0} = 24/842  = 0.0285
     p{1} = 178/842 = 0.2114
     p{2} = 312/842 = 0.3705
     p{3} = 196/842 = 0.2328
     p{4} = 68/842  = 0.0808
     p{5} = 36/842  = 0.0428
     p{6} = 22/842  = 0.0261
     p{7} = 6/842   = 0.0071

   (2) Calculate Shannon entropy H = -sum(p{i} * log2(p{i}))
     H = -(0.0285 * log2(0.0285)   = 0.148
         + 0.2114 * log2(0.2114)   = 0.467
         + 0.3705 * log2(0.3705)   = 0.531
         + 0.2328 * log2(0.2328)   = 0.481
         + 0.0808 * log2(0.0808)   = 0.295
         + 0.0428 * log2(0.0428)   = 0.197
         + 0.0261 * log2(0.0261)   = 0.137
         + 0.0071 * log2(0.0071))  = 0.050

     H = 2.306 bits (per sample from 8-bucket distribution)

   (3) Estimated total entropy
     Total entropy bits = H * log2(sample_count)
     = 2.306 * log2(842)
     = 2.306 * 9.72
     = 22.4 bits (approximate)

     Reported: 2.54 bits (average per-sample entropy)
```

4.3. Binding MAC Computation

```
binding-mac = HMAC-SHA256(
    key = checkpoint-chain-key,
    message = entropy-commitment ||
              CBOR(sources) ||
              CBOR(summary) ||
              prev-checkpoint-hash
)
```

Where:

- checkpoint-chain-key = session-derived 256-bit key
- entropy-commitment = h'b94d27b9...' (32 bytes)
- CBOR(sources) = 83 01 02 03 (4 bytes: array of 3 uints)
- CBOR(summary) = A4 01 ... (variable length map)
- prev-checkpoint-hash = h'...' (32 bytes)

4.4. Verifier Checks

A Verifier performs the following checks on the jitter seal:

```
def verify_jitter_seal(jitter_binding, prev_hash, chain_key):
    # (1) Structural validation
    assert all_required_fields_present(jitter_binding)
    assert len(jitter_binding.sources) >= 1

    # (2) Recompute entropy-commitment (if raw-intervals disclosed)
    if jitter_binding.raw_intervals is not None:
        expected_commitment = sha256(
            concat_as_uint32_le(jitter_binding.raw_intervals)
        )
        commitment = jitter_binding.entropy_commitment
        assert commitment == expected_commitment

    # (3) Recompute histogram (if raw-intervals disclosed)
    if jitter_binding.raw_intervals is not None:
        computed_histogram = bucket_intervals(
            jitter_binding.raw_intervals,
            boundaries=(0, 50, 100, 200, 500, 1000, 2000, 5000)
        )
        assert histograms_consistent(
            computed_histogram,
            jitter_binding.summary.timing_histogram
        )

    # (4) Verify binding MAC
    expected_mac = hmac_sha256(
        key=chain_key,
        message=jitter_binding.entropy_commitment.value +
                cbor_encode(jitter_binding.sources) +
                cbor_encode(jitter_binding.summary) +
                prev_hash
    )
    assert jitter_binding.binding_mac == expected_mac

    # (5) Entropy threshold check
    assert jitter_binding.summary.entropy_bits >= MIN_THRESHOLD

    # (6) Sample count plausibility
    assert jitter_binding.summary.sample_count >= 10

    return VERIFIED
```

5. VDF Causality Example

This example demonstrates VDF input computation and why backdating requires recomputation of the entire subsequent chain.

5.1. VDF Input Computation for Checkpoint N

```
VDF_input{N} = SHA256(
    VDF_output{N-1} ||
    content-hash{N} ||
    jitter-commitment{N} ||
    uint32_le(sequence{N})
)

VDF_input{2} = SHA256(
    h'output1...'
 || h'abcd1234...'
 || h'jitter2...'
 || h'02000000'
)

VDF_output{2} = SHA256^10200000000(VDF_input{2})
```

5.2. Why Backdating Requires Recomputation

```
Adversary Goal: Insert a fake checkpoint between checkpoints 0 and 1

Attempt: Create fake checkpoint 0.5 with:
  - content-hash{0.5} = hash of backdated content
  - jitter-commitment{0.5} = fabricated timing data
  - sequence{0.5} = 1 (bumping original checkpoint 1 to sequence 2)

Problem: This changes VDF_input{1}:
  VDF_input{1}_original = H(VDF_output{0} || content{1} || jitter{1})
  VDF_input{1}_fake     = H(VDF_output{0.5} || content{1} || jitter{1})

Since VDF_output{0.5} != VDF_output{0}, the adversary must:

(1) Compute VDF_output{0.5} from VDF_input{0.5}
   Cost: ~1200 seconds (cannot parallelize)

(2) Recompute VDF_output{1} from new VDF_input{1}
   Cost: ~1200 seconds (cannot parallelize)

(3) Recompute VDF_output{2} from new VDF_input{2}
   Cost: ~1200 seconds (cannot parallelize)

Total minimum time: 3600 seconds = 1 hour
(Cannot be reduced by parallel computation)

If external anchor confirmed checkpoint 2 at T_anchor,
adversary must complete all recomputation before T_anchor.
Any attempt to backdate beyond anchor is impossible.
```

5.3. Calibration Cross-Check

```
   def verify_vdf_duration_claim(checkpoint, calibration):
       # Extract values
       iterations = checkpoint.vdf_proof.iterations
       claimed_duration = checkpoint.vdf_proof.claimed_duration
       calibration_rate = calibration.calibration_iterations

       # Compute minimum possible duration
       min_duration = iterations / calibration_rate

       # Allow 10% tolerance for measurement variance
       tolerance = 1.1

       # Claimed duration must be at least min_duration
       if claimed_duration < (min_duration / tolerance):
           return INVALID("Claimed duration impossibly short")

       # Claimed duration should not be excessively long
       if claimed_duration > (min_duration * 10):
           return WARNING("Claimed duration suspiciously long")

       return VALID

   # Example:
   # iterations = 10,200,000,000
   # calibration_rate = 8,500,000 / second
   # min_duration = 10.2B / 8.5M = 1200 seconds
   # claimed_duration = 1200 seconds: VALID
```

6. Absence Claim Example

This example demonstrates both chain-verifiable and monitoring-dependent absence claims, showing how verifiers prove claims from checkpoint data.

6.1. Chain-Verifiable Claim: max-single-delta-chars

Claim: No single checkpoint added more than 500 characters.

```
absence-claim = {
  1: 1,
  2: {1: 500},
  3: {
    1: 1,
    2: {
      1: (0, 2),
      2: 423
    }
  },
  4: {
    1: 1,
    2: []
  }
}
```

Verifier proof procedure:

```
def verify_max_single_delta_chars(evidence, threshold):
    # (1) Verify chain integrity
    assert verify_chain_hashes(evidence.checkpoints)
    assert verify_vdf_linkage(evidence.checkpoints)

    # (2) Extract max delta from checkpoint data
    max_chars_added = 0
    for checkpoint in evidence.checkpoints:
        delta = checkpoint.delta
        max_chars_added = max(max_chars_added, delta.chars_added)

    # (3) Compare against threshold
    if max_chars_added <= threshold:
        return PROVEN(
            observed=max_chars_added,
            threshold=threshold,
            confidence="proven"
        )
    else:
        return FAILED(
            observed=max_chars_added,
            threshold=threshold
        )
```

6.2. Monitoring-Dependent Claim: max-paste-event-chars

Claim: No paste event inserted more than 200 characters.

```
absence-claim = {
  1: 16,
  2: {1: 200},
  3: {
    1: 2,
    2: {
      1: (0, 2),
      2: 0,
      3: 0.0
    }
  },
  4: {
    1: 2,
    2: (
      "Requires trust in clipboard monitoring",
      "Coverage fraction: 0.98"
    )
  },
  5: {
    1: 2,
    2: "macOS NSPasteboard notifications monitored continuously",
    3: true
  }
}
```

Trust chain for monitoring-dependent claim:

```
   Trust Chain Analysis:

   (1) AE Trust Target: os-reported-events (2)
      - Requires: OS correctly reports clipboard access
      - macOS: NSPasteboard change notifications
      - Trustworthiness: Moderate (depends on OS integrity)

   (2) Verification Status: true
      - Cross-reference: hardware-section contains SE attestation
      - SE attests: witnessd binary hash, measurement time
      - This increases confidence that AE was unmodified

   (3) Monitoring Coverage: 98%
      - monitoring-coverage.coverage-fraction = 0.98
      - 2% gap when app was backgrounded
      - Caveat: paste during gap would be undetected

   (4) Resulting Confidence: HIGH (level 2)
      - Not PROVEN (would require trustless verification)
      - HIGH because: HW attestation + high coverage + OS events

   Relying Party Decision:
     - For academic submission: HIGH confidence acceptable
     - For legal proceeding: May require additional corroboration
```

7. Attestation Result Example (.war)

This example shows a Verifier’s Attestation Result after appraising the multi-checkpoint Evidence packet from Section 3.

```
1463894560({
  1: 1,
  2: h’123e4567e89b12d3a456426614174000’,
  3: 1(1706840000),

  4: 2,

  5: 0.78,

  6: [
    {
      1: 6,
      2: true,
      3: "Sequence numbers 0,1,2 consecutive",
      4: 1
    },
    {
      1: 7,
```

```
        2: true,
        3: "All prev-hash values match prior checkpoint-hash",
        4: 1
      },
      {
        1: 4,
        2: true,
        3: "Total VDF time: 3000 seconds (threshold: 2700)",
        4: 1
      },
      {
        1: 8,
        2: true,
        3: "Cumulative entropy: 7.92 bits (threshold: 6.0)",
        4: 1
      },
      {
        1: 16,
        2: true,
        3: "No paste events detected (threshold: 500)",
        4: 2
      },
      {
        1: 100,
        2: true,
        3: "2/2 challenges passed, median response 1490ms",
        4: 2
      }
    ],

    7: 18(h'D28441A0A201260442313154...'),

    8: "WritersLogic Verification Service v2.1",

    9: {
      1: "2.1.0",
      2: "https://verify.writerslogic.com",
      4: "pop-standard-v1"
    },

    10: (
      "No hardware attestation in Evidence packet",
      "Monitoring coverage: 98%",
      "VDF calibration self-reported"
    )
  })
```

Key aspects of the Attestation Result:

* Verdict: likely-human (2)

  Based on: checkpoint chain integrity, realistic jitter distribution, presence challenges passed, no anomalies.

* Confidence: 0.78 (high range)

  Factors increasing: chain integrity proven, presence verified. Factors limiting: no hardware attestation, self-reported calibration.

* Caveats document limitations:

  Relying Parties can assess whether the caveats are acceptable for their use case. Academic submission may accept; legal proceeding may require additional evidence.

* Verifier signature enables trust chain:

  Relying Party trusts WritersLogic Verification Service. Signature proves Attestation Result came from that Verifier.

8. Salt Mode Examples

This example demonstrates the three salt modes and their verification flow differences. The same document is shown with each salt mode.

8.1. Unsalted Mode (Default)

```
document-ref = {
  1: {
    1: 1,
    2: h'e3b0c44298fc1c149afbf4c8996fb924
        27ae41e4649b934ca495991b7852b855'
  },
  2: "essay.txt",
  3: 4523,
  4: 4401,
  5: 0
}
```

Verification flow (unsalted):

```
   Verifier has: document content, Evidence packet

   (1) Compute document hash
     computed_hash = SHA256(document_content)
     = h'e3b0c442...'

   (2) Compare with Evidence
     assert computed_hash == document_ref.content_hash.value
     assert computed_hash == checkpoints{-1}.content_hash.value

   Result: VERIFIED
     - Anyone with the document can verify binding
     - No additional information needed
     - Document linkage is globally verifiable
```

8.2. Author-Salted Mode

```
   document-ref = {
     1: {
       1: 1,
       2: h'7f83b1657ff1fc53b92dc18148a1d65d
           fc2d4b1fa3d677284addd200126d9069'
     },
     2: "confidential-report.txt",
     3: 8934,
     4: 8712,
     5: 1,
     6: h'9f86d081884c7d659a2feaa0c55ad015
         a3bf4f1b2b0b822cd15d6c15b0f00a08'
   }

   Verification flow (author-salted):
```

```
   Verifier has: document content, Evidence packet
   Verifier needs: salt (provided out-of-band by author)

   (1) Author provides salt to chosen verifier
     received_salt = h'deadbeefcafebabe1234567890abcdef'

   (2) Verify salt matches commitment
     computed_commitment = SHA256(received_salt)
     assert computed_commitment == document_ref.salt_commitment

   (3) Compute salted document hash
     computed_hash = SHA256(received_salt || document_content)

   (4) Compare with Evidence
     assert computed_hash == document_ref.content_hash.value

   Result: VERIFIED
     - Only verifiers who received salt can verify
     - Author controls who can verify document binding
     - Useful for: unpublished manuscripts, confidential documents
```

8.3. Third-Party Escrowed Mode

```
   document-ref = {
     1: {
       1: 1,
       2: h'3a7bd3e2360a3d29eea436fcfb7e44c7
           35d117c42d1c1835420b6b9942dd4f1b'
     },
     3: 12045,
     4: 11823,
     5: 2,
     6: h'a591a6d40bf420404a011733cfb7b190
         d62c65bf0bcda32b57b277d9ad9f146e'
   }

   Verification flow (escrowed):
```

```
   Verifier has: document content, Evidence packet
   Verifier needs: salt (from escrow service)

   (1) Request salt from escrow
     - Verifier contacts escrow service
     - Escrow verifies release conditions are met:
       * Legal subpoena
       * Author consent
       * Time-based release
       * Dispute resolution trigger
     - Escrow provides salt if conditions satisfied

   (2-4) Same as author-salted verification

   Use cases:
     - Litigation discovery: salt released upon court order
     - Embargo periods: salt released after publication date
     - Dispute resolution: salt released if authorship contested
     - Dead man's switch: salt released if author inactive
```

8.4. Salt Mode Comparison

Aspect	Unsalted	Author-Salted	Escrowed
Who can verify	Anyone with doc	Author's choice	Conditions-based
Privacy	None (hash public)	High (author controls)	Medium (escrow policy)
Typical use	Published works	Unpublished drafts	Legal/regulatory
Salt storage	N/A	Author responsibility	Third party
Lost salt impact	N/A	Cannot verify	Escrow backup

Table 1

9. Security Considerations

This document provides examples for the Proof of Process format. Security considerations for the format itself are specified in the main architecture document [I-D.condrey-rats-pop].

Example data in this document is illustrative. Implementations MUST NOT use example values as actual cryptographic material. All hash values, keys, and signatures shown are placeholders.

10. IANA Considerations

This document has no IANA actions. All registrations are specified in the main architecture document.

11. References

11.1. Normative References

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

11.2. Informative References

[I-D.condrey-rats-pop] Condrey, D., "Proof of Process: An Evidence Framework for Digital Authorship Attestation", Work in Progress, Internet-Draft, draft-condrey-rats-pop-00, <https://datatracker.ietf.org/doc/html/draft-condrey-rats-pop-00>.

Author’s Address

David Condrey
Writerslogic Inc
United States
Email: david@writerslogic.com