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

D. Condrey
Writerslogic Inc

6 February 2026

Proof of Process: VDF Proof Aggregation Extension
draft-condrey-rats-pop-aggregation-00

Abstract

This document defines optional mechanisms for aggregating Verifiable Delay Function (VDF) proofs in Proof of Process Evidence packets. Aggregation enables O(1) or O(log n) verification of entire checkpoint chains that would otherwise require O(n) sequential VDF recomputation. This extension supports Merkle tree aggregation and SNARK-based proof compression for high-volume verification scenarios.

Status of This Memo

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

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

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

This Internet-Draft will expire on 10 August 2026.

Copyright Notice

Table of Contents

1. Introduction

This document defines optional mechanisms for aggregating VDF proofs to reduce verification cost. Aggregation enables O(1) or O(log n) verification of entire checkpoint chains that would otherwise require O(n) sequential VDF recomputation.

This extension is defined as a companion to the main Proof of Process specification [I-D.condrey-rats-pop].

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. Motivation

The iterated-sha256 VDF provides strong temporal guarantees but requires verifiers to recompute the entire hash chain. For a document with 1000 checkpoints, each with 10 million iterations, full verification requires 10 billion hash operations.

While this verification cost is acceptable for high-stakes scenarios, it creates barriers to adoption:

* Publishers processing thousands of submissions cannot afford O(n) verification per document.

* Mobile devices and web browsers may lack computational resources for full verification.

* Real-time verification scenarios require sub-second response.

VDF aggregation addresses these challenges by providing efficiently-verifiable proofs that attest to the correctness of the underlying VDF chain.

3. Aggregation Proof Structure

The VDF aggregate proof is an optional extension to the vdf-proof structure, identified by integer key 9.

```
; VDF aggregate proof extension
; Key 9 in vdf-proof (optional)
vdf-aggregate-proof = {
    1 => uint,                      ; checkpoints-covered
    2 => aggregation-method,        ; method
    3 => bstr,                      ; aggregate-proof
    ? 4 => aggregate-metadata,      ; metadata
}

aggregation-method = &(
    merkle-vdf-tree: 1,             ; Merkle tree over VDF outputs
    snark-groth16: 16,              ; Groth16 SNARK proof
    snark-plonk: 17,                ; PLONK SNARK proof
    stark: 18,                      ; STARK proof
    recursive-snark: 19,            ; Recursive SNARK composition
)

aggregate-metadata = {
    ? 1 => tstr,                    ; prover-version
    ? 2 => uint,                    ; proof-generation-time-ms
    ? 3 => uint,                    ; proof-size-bytes
    ? 4 => tstr,                    ; verification-key-id
    ? 5 => bstr,                    ; verification-key
}
```

4. Merkle VDF Tree Aggregation

The simplest aggregation method constructs a Merkle tree over VDF inputs and outputs, enabling selective verification with O(log n) proof size.

4.1. Tree Construction

For N checkpoints with VDF proofs:

Leaf{i} = H(VDF_input{i} || VDF_output{i} || iterations{i})

Internal nodes computed as standard Merkle tree:
Node{parent} = H(Node{left} || Node{right})

Root = final tree root

Aggregate proof contains:
- Root hash
- Total iterations across all checkpoints
- Optional: Merkle proofs for sampled checkpoints

4.2. Verification Procedure

Merkle aggregation supports three verification modes:

Full verification: Recompute all VDFs and verify Merkle root matches. O(n) time.

Sampled verification: Randomly select k checkpoints, verify their VDFs, and verify Merkle inclusion proofs. O(k * VDF_iterations/n + k * log n). Provides probabilistic assurance.

Root-only verification: Trust the prover, verify only the Merkle root signature. O(1) time. Requires trusted aggregation service.

The verification mode SHOULD be documented in the Attestation Result.

4.3. Merkle Aggregate CDDL

```
; Merkle VDF tree proof structure
merkle-vdf-proof = {
    1 => hash-value,                ; root-hash
    2 => uint,                      ; total-iterations
    3 => uint,                      ; checkpoint-count
    ? 4 => [+ merkle-sample],       ; sampled-proofs
    ? 5 => cose-signature,          ; aggregator-signature
}

merkle-sample = {
    1 => uint,                      ; checkpoint-index
    2 => [+ hash-value],            ; merkle-path
    3 => bool,                      ; vdf-verified (by aggregator)
}
```

5. SNARK-Based Aggregation

For constant-time verification, SNARK (Succinct Non-interactive ARgument of Knowledge) proofs can attest to the correctness of the entire VDF chain.

5.1. Circuit Definition

The SNARK circuit proves the following statement:

Public inputs:
- VDF_input{0} (genesis input)
- VDF_output{N-1} (final output)
- total_iterations
- checkpoint_count

Private inputs:
- All intermediate VDF_input{i} and VDF_output{i}
- All iteration counts per checkpoint

Circuit constraints:
For each checkpoint i in 0..N-1:
(1) VDF_output{i} = SHA256^iterations{i}(VDF_input{i})
(2) VDF_input{i+1} = H(VDF_output{i} || content-hash{i+1} || ...)
(3) sum(iterations{i}) = total_iterations

A valid SNARK proof demonstrates that there exist valid intermediate values satisfying all constraints, without revealing those values or requiring recomputation.

5.2. SNARK Verification

SNARK verification is constant-time regardless of checkpoint count:

```
def verify_snark_aggregate(
    proof: bytes,
    vdf_input_genesis: bytes,
    vdf_output_final: bytes,
    total_iterations: int,
    checkpoint_count: int,
    verification_key: bytes
) -> bool:
    public_inputs = encode_public_inputs(
        vdf_input_genesis,
        vdf_output_final,
        total_iterations,
        checkpoint_count
    )
    return snark_verify(verification_key, public_inputs, proof)
```

Verification complexity: O(1) for Groth16, O(log n) for PLONK with logarithmic verification.

5.3. Trust Assumptions

SNARK-based aggregation introduces additional trust assumptions:

* *Trusted setup (Groth16):* The verification key MUST be generated through a secure multi-party computation. A compromised setup allows forged proofs.

* *Cryptographic assumptions:* SNARK security relies on hardness of discrete logarithm and pairing assumptions.

* *Implementation correctness:* The circuit MUST correctly encode the VDF verification constraints.

For maximum assurance, implementations SHOULD support both SNARK verification (for efficiency) and full VDF recomputation (for trust-minimized verification).

5.4. SNARK Aggregate CDDL

```
; SNARK proof structure
snark-vdf-proof = {
    1 => snark-scheme,              ; scheme
    2 => bstr,                      ; proof-bytes
    3 => bstr,                      ; verification-key-id
    4 => [+ bstr],                  ; public-inputs (encoded)
    ? 5 => tstr,                    ; circuit-version
    ? 6 => bstr,                    ; setup-ceremony-hash
}

snark-scheme = &(
    groth16-bn254: 1,               ; Groth16 on BN254 curve
    groth16-bls12-381: 2,           ; Groth16 on BLS12-381
    plonk-bn254: 3,                 ; PLONK on BN254
    plonk-bls12-381: 4,             ; PLONK on BLS12-381
)
```

6. Verification Mode Selection

Verifiers SHOULD select verification modes based on the assurance level required by the Relying Party:

Mode	Complexity	Trust Required	Use Case
Full VDF recomputation	O(n * iterations)	None	Litigation, forensics
Merkle + full sample	O(k * iterations/n)	Statistical	Academic review
SNARK verification	O(1)	Setup ceremony	High-volume processing
Signed aggregate only	O(1)	Aggregator	Real-time display

Table 1: Verification Mode Comparison

Attestation Results MUST document which verification mode was used and any associated trust assumptions.

7. Aggregation Proof Example

```
vdf-proof = {
  1: 1,
  2: {1: 1, 2: 8500000},
  3: h'genesis-input...',
  4: h'final-output...',
  5: h'',
  6: 3600.0,
  7: 30600000000,
  8: { ... },

  9: {
    1: 150,
    2: 1,
    3: h'merkle-proof-bytes...',
    4: {
      1: "witnessd-aggregator-1.0",
      2: 45000,
      3: 2048,
    }
  }
}
```

8. Security Considerations

VDF aggregation introduces security trade-offs:

* *Merkle aggregation:* Security equivalent to underlying hash function. No additional cryptographic assumptions. Sampled verification provides probabilistic (not cryptographic) assurance proportional to sample size.

* *SNARK aggregation:* Security depends on discrete logarithm hardness in pairing-friendly groups. Groth16 requires trusted setup; PLONK uses universal setup. Post-quantum security is NOT provided.

* *Aggregator trust:* Signed aggregates require trust in the aggregation service. Verifiers SHOULD verify aggregator identity through established PKI.

Full security considerations for the Proof of Process format are specified in [I-D.condrey-rats-pop].

9. IANA Considerations

This document has no IANA actions. The aggregation extension uses key 9 within the vdf-proof structure as defined in the main architecture document.

10. References

10.1. 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>.

10.2. 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