SCITT
Internet-Draft
Intended status: Standards Track
Expires: 4 October 2026

H. Birkholz
Fraunhofer SIT
A. Delignat-Lavaud
C. Fournet
A. Chamayou
Microsoft Research

2 April 2026

CCF Profile for COSE Receipts
draft-ietf-scitt-receipts-ccf-profile-01

Abstract

This document defines a new verifiable data structure (VDS) type for COSE Receipts and inclusion proof specifically designed for append-only logs produced by the Confidential Consortium Framework (CCF) to provide stronger tamper-evidence guarantees.

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 4 October 2026.

Table of Contents

1. Introduction 2
1.1. Requirements Notation 3
2. Description of the Confidential Consortium Framework Ledger Verifiable Data Structure 3
2.1. Merkle Tree Shape 3
2.2. Transaction Components 4
3. CCF Inclusion Proofs 5
3.1. CCF Inclusion Proof Signature 6
3.2. Inclusion Proof Verification Algorithm 6
4. Usage in COSE Receipts 7
5. Privacy Considerations 8
6. Security Considerations 8
6.1. Trusted Execution Environments 8
6.2. Operators 9
7. IANA Considerations 9
7.1. Additions to Existing Registries 9
7.1.1. Tree Algorithms 9
8. Normative References 9
Authors' Addresses 10

1. Introduction

The COSE Receipts document [I-D.ietf-cose-merkle-tree-proofs] defines a common framework for expressing different types of proofs about verifiable data structures (VDS), providing a standardized way to convey trust relevant evidence. For instance, inclusion proofs guarantee to a verifier that a given serializable element is recorded at a given state of the VDS, while consistency proofs are used to establish that an inclusion proof is still consistent with the new state of the VDS at a later time.

In this document, we define a new type of VDS and inclusion proof associated with an application of the Confidential Consortium Framework (CCF) ledger that implements the SCITT Architecture defined in [I-D.ietf-wg-scitt-architecture]. This VDS carries indexed transaction information in a binary Merkle Tree, where new transactions are appended to the right, so that the binary

decomposition of the index of a transaction can be interpreted as the position in the tree if 0 represents the left branch and 1 the right branch. Compared to [RFC9162], the leaves of CCF trees carry additional internal information for the following purposes:

1. To bind the full details of the transaction executed, which is a super-set of what is exposed in the proof and captures internal information details useful for detailed system audit, but not for application purposes.

2. To allow the distributed system executing the application logic in Trusted Execution Environments (TEE) to persist signatures to storage early. Receipt production in only enabled once transactions are fully committed by the consensus protocol.

1.1. Requirements Notation

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. Description of the Confidential Consortium Framework Ledger Verifiable Data Structure

This documents extends the VDS registry of [I-D.ietf-cose-merkle-tree-proofs] with the following value:

Name	Value	Description	Reference
CCF_LEDGER_SHA256	TBD_1 (requested assignment 2)	Historical transaction ledgers, such as the CCF ledger	RFCthis

Table 1: Verifiable Data Structure Algorithms

2.1. Merkle Tree Shape

A CCF ledger is a binary Merkle Tree constructed from a hash function H, which is defined from the log type. For instance, the hash function for CCF_LEDGER_SHA256 is SHA256, whose HASH_SIZE is 32 bytes.

The Merkle tree encodes an ordered list of n transactions T_n = {T[0], T[1], ..., T[n-1]}. We define the Merkle Tree Hash (MTH) function, which takes as input a list of serialized transactions (as byte strings), and outputs a single HASH_SIZE byte string called the Merkle root hash, by induction on the list.

This function is defined as follows:

The hash of an empty list is the hash of an empty string:

MTH({}) = HASH().

The hash of a list with one entry (also known as a leaf hash) is:

MTH({d[0]}) = HASH(d[0]).

For n > 1, let k be the largest power of two smaller than n (i.e., k < n <= 2k). The Merkle Tree Hash of an n-element list D_n is then defined recursively as:

MTH(D_n) = HASH(MTH(D[0:k]) || MTH(D[k:n])),

where:

* || denotes concatenation

* : denotes concatenation of lists

* D[k1:k2] = D'_(k2-k1) denotes the list {d'[0] = d[k1], d'[1] = d[k1+1], ..., d'[k2-k1-1] = d[k2-1]} of length (k2 - k1).

2.2. Transaction Components

Each leaf in a CCF ledger carries the following components:

```
ccf-leaf = [
  ; Byte string of size HASH_SIZE(32)
  internal-transaction-hash: bstr .size 32

  ; Text string of at most 1024 bytes
  internal-evidence: tstr .size (1..1024)

  ; Byte string of size HASH_SIZE(32)
  data-hash: bstr .size 32
]
```

The internal-transaction-hash and internal-evidence byte strings are internal to the CCF implementation. They can be safely ignored by receipt Verifiers, but they commit the transparency service (TS) to the whole tree contents and may be used for additional, CCF-specific auditing.

internal-transaction-hash is a hash over the complete entry in the [CCF-Ledger-Format], and internal-evidence is a revealable [CCF-Commit-Evidence] value that allows early persistence of ledger entries before distributed consensus can be established. This mechanism is useful to implement high-throughput transparency applications in Trusted Execution Environments (TEEs) that only provide a limited amount of memory, while maintaining high availability afforded by distributed consensus.

data-hash summarizes the application data included in the ledger at this transaction, which is a Signed Statement as defined by [I-D.ietf-wg-scitt-architecture].

3. CCF Inclusion Proofs

CCF inclusion proofs consist of a list of digests tagged with a single left-or-right bit.

```
ccf-proof-element = [
  ; Position of the element
  left: bool

  ; Hash of the proof element: byte string of size HASH_SIZE(32)
  hash: bstr .size 32
]

ccf-inclusion-proof = bstr .cbor {
  &(leaf: 1) => ccf-leaf
  &(path: 2) => [+ ccf-proof-element]
}
```

Unlike some other tree algorithms, the index of the element in the tree is not explicit in the inclusion proof, but the list of left-or-right bits can be treated as the binary decomposition of the index, from the least significant (leaf) to the most significant (root).

3.1. CCF Inclusion Proof Signature

The proof signature for a CCF inclusion proof is a COSE signature (encoded with the COSE_Sign1 CBOR type) which includes the following additional requirements for protected and unprotected headers. Please note that there may be additional header parameters defined by the application.

The protected header parameters for the CCF inclusion proof signature MUST include the following:

* verifiable-data-structure: int/tstr. This header MUST be set to the verifiable data structure algorithm identifier for ccf-ledger (TBD_1).

* label: int. This header MUST be set to the value of the inclusion proof type in the IANA registry of Verifiable Data Structure Proof Type (-1).

The unprotected header for a CCF inclusion proof signature MUST include the following:

* inclusion-proof: bstr .cbor ccf-inclusion-proof. This contains the serialized CCF inclusion proof, as defined above.

The payload of the signature is the CCF ledger Merkle root digest, and MUST be detached in order to force verifiers to recompute the root from the inclusion proof in the unprotected header. This provides a safeguard against implementation errors that use the payload of the signature but do not recompute the root from the inclusion proof.

3.2. Inclusion Proof Verification Algorithm

CCF uses the following algorithm to verify an inclusion receipt:

```
compute_root(proof):
  h := proof.leaf.internal-transaction-hash
       || HASH(proof.leaf.internal-evidence)
       || proof.leaf.data-hash

  for [left, hash] in proof:
      h := HASH(hash + h) if left
           HASH(h + hash) else
  return h

verify_inclusion_receipt(inclusion_receipt):
  let label = INCLUSION_PROOF_LABEL
  assert(label in inclusion_receipt.unprotected_header)
  let proof = inclusion_receipt.unprotected_header[label]
  assert(inclusion_receipt.payload == nil)
  let payload = compute_root(proof)

  # Use the Merkle Root as the detached payload
  return verify_cose(inclusion_receipt, payload)
```

A description can also be found at [CCF-Receipt-Verification].

4. Usage in COSE Receipts

A COSE Receipt with a CCF inclusion proof is described by the following CDDL definition:

```
protected-header-map = {
  &(alg: 1) => int
  &(vds: 395) => 2
  * cose-label => cose-value
}
```

* alg (label: 1): REQUIRED. Signature algorithm identifier. Value type: int.

* vds (label: 395): REQUIRED. verifiable data structure algorithm identifier. Value type: int.

The unprotected header for an inclusion proof signature is described by the following CDDL definition:

```
inclusion-proof = ccf-inclusion-proof

inclusion-proofs = [ + inclusion-proof ]

verifiable-proofs = {
  &(inclusion-proof: -1) => inclusion-proofs
}

unprotected-header-map = {
  &(vdp: 396) => verifiable-proofs
  * cose-label => cose-value
}
```

5. Privacy Considerations

See the privacy considerations section of:

* [I-D.ietf-cose-merkle-tree-proofs]

6. Security Considerations

The security considerations of [I-D.ietf-cose-merkle-tree-proofs] apply.

6.1. Trusted Execution Environments

CCF networks of nodes rely on executing in TEEs to secure their function, in particular:

1. The evaluation of registration policies

2. The creation and usage of receipt signing keys

A compromise in the TEE platform used to execute the network may allow an attacker to produce invalid and divergent ledger branches. Clients can mitigate this risk in two ways: by regularly auditing the consistency of the CCF ledger; and by regularly fetching attestation information about the TEE instances, available in the ledger and from the network itself, and confirming that the nodes composing the network are running up-to-date, trusted platform components.

6.2. Operators

An operator has the ability to start successor networks with a distinct identity. The operator of a CCF network can recover the service by starting a successor network, for example a new CCF network with its own service identity, that endorses the ledger state of the previous instance. This provides service continuity after a catastrophic failure of a majority of the nodes. However, a malicious operator could exploit this mechanism and truncate the ledger's history by initializing the successor network from an earlier ledger prefix, thereby omitting some later entries. Clients can mitigate this risk by auditing the successor ledger and verifying that their latest known receipts from the prior service are included in the successor's ledger.

7. IANA Considerations

7.1. Additions to Existing Registries

7.1.1. Tree Algorithms

This document requests IANA to add the following new value to the 'COSE Verifiable Data Structures' registry:

* Name: CCF_LEDGER_SHA256

* Value: 2 (requested assignment)

* Description: Append-only logs that are integrity-protected by a Merkle Tree and signatures produced via Trusted Execution Environments containing a mix of public and confidential information, as specified by the Confidential Consortium Framework.

* Reference: This document

8. Normative References

[CCF] "Confidential Consortium Framework", n.d., <https://github.com/microsoft/ccf>.

[CCF-Commit-Evidence] "CCF Commit Evidence", n.d., <https://microsoft.github.io/CCF/main/use_apps/verify_tx.html#commit-evidence>.

[CCF-Ledger-Format]
"CCF Ledger Format", n.d., <https://microsoft.github.io/CCF/main/architecture/ledger.html>.

[CCF-Receipt-Verification]
"CCF Receipt Verification", n.d., <https://microsoft.github.io/CCF/main/use_apps/verify_tx.html#receipt-verification>.

[I-D.ietf-cose-merkle-tree-proofs]
Steele, O., Birkholz, H., Delignat-Lavaud, A., and C. Fournet, "COSE (CBOR Object Signing and Encryption) Receipts", Work in Progress, Internet-Draft, draft-ietf-cose-merkle-tree-proofs-18, 2 December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-merkle-tree-proofs-18>.

[I-D.ietf-wg-scitt-architecture]
"*** BROKEN REFERENCE ***".

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC9162] Laurie, B., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", RFC 9162, DOI 10.17487/RFC9162, December 2021, <https://www.rfc-editor.org/rfc/rfc9162>.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@ietf.contact

Antoine Delignat-Lavaud
Microsoft Research
21 Station Road
Cambridge
CB1 2FB
United Kingdom
Email: antdl@microsoft.com

Cedric Fournet
Microsoft Research
21 Station Road
Cambridge
CB1 2FB
United Kingdom
Email: fournet@microsoft.com

Amaury Chamayou
Microsoft Research
21 Station Road
Cambridge
CB1 2FB
United Kingdom
Email: amaury.chamayou@microsoft.com