Workgroup: TBD
Internet-Draft:
draft-steele-cose-merkle-tree-proofs-00
Published: 14 March 2023
Intended Status: Standards Track
Expires: 15 September 2023
Authors: O. Steele Transmute
H. Birkholz Fraunhofer SIT
M. Riechert Microsoft
A. Delignat-Lavaud Microsoft
C. Fournet Microsoft

Concise Encoding of Signed Merkle Tree Proofs

Abstract

This specification describes three CBOR data structures for primary use in COSE envelopes. A format for Merkle Tree Root Signatures with metadata, a format for Inclusions Paths, and a format for disclosure of a single hadh tree leaf payload (Merkle Tree Proofs).

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 15 September 2023.

Copyright Notice

Table of Contents

1. Introduction

Merkle proofs are verifiable data structures that support secure data storage, through their ability to protect the integrity of batches of documents or collections of statements.

Merkle proofs can be used to prove a document is in a database (proof of existence), or that a smaller set of statements are contained in a large set of statements (proof of disclosure).

A merkle proof is a path from a leaf to a root in a merkle tree.

Merkle trees are constructed from simple operations such as concatenation and digest via a cryptographic hash function.

The simple design and valuable cryptographic properties of merkle trees have been leveraged in many network and database applications.

Differences in the representation of a merkle tree, merkle leaf and merkle inclusion proof can increase the burden for implementers, and create interoperability challenges.

This document describes the three data structures necessary to use merkle proofs with COSE envelopes.

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. Terminology

Leaf Bytes: A merkle tree leaf is labelled with the cryptographic hash of a sequence of bytes. These bytes may be structured as a combination of Payload and Extra Data.

Merkle Tree: A Merkle tree is a tree where every leaf is labelled with the cryptographic hash of a sequence of bytes and every node that is not a leaf is labeled with the cryptographic hash of the labels of its child nodes.

Merkle Tree Root: A Merkle tree root is the root node of a tree which represents the cryptographic hash that commits to all leaves in the tree.

Merkle Tree Algorithm: A Merkle tree algorithm specifies how nodes in the tree must be hashed to compute the root node.

Payload and Extra Data: A payload is data bound to in a Merkle tree leaf. The Merkle tree algorithm determines how a payload together with extra data is bound to a leaf. The simplest case is that the payload is the leaf itself without extra data.

Inclusion Path: An inclusion path confirms that a value is a leaf of a Merkle tree known only by its root hash (and tree size, possibly).

Signed Merkle Tree Proof: A signed Merkle tree proof is the combination of signed Merkle tree root hash, inclusion path, extra data, and payload.

3. CBOR Merkle Structures

This section describes representations of merkle tree structures in CBOR.

Some of the structures such as the construction of a merkle tree leaf, or an inclusion proof from a leaf to a merkle root, might have several different representations.

Some differences in representations are necessary to support efficient verification of proofs and compatibility with deployed tree algorithms used in specific implementations.

3.1. Signed Merkle Tree Root

A Merkle tree root is signed with COSE_Sign1, creating a Signed Merkle Tree Root:

```
SMTR = THIS.COSE.profile .and COSE_Sign1_Tagged
```

Protected header parameters:

- *alg (label: 1): **REQUIRED**. Signature algorithm. Value type: int / tstr.
- *tree alg (label: TBD): **REQUIRED**. Merkle tree algorithm. Value type: int / tstr.
- *tree size (label: TBD): **OPTIONAL**. Merkle tree size as the number of leaves. Value type: uint.

A COSE profile of this specification may add further header parameters, for example to identify the signer.

Payload: Merkle tree root hash bytes according to tree alg (i.e., header params tell you what the alg id is here)

Note: The payload is just a byte string representing the Merkle tree root hash (and not some wrapper structure) so that it can be detached (see defintion of payload in https://www.rfc-editor.org/rfc/rfc9052#section-4.1) and easily re-computed from an inclusion path and leaf bytes. This allows to design other structures that force re-computation and prevent faulty implementations (forgetting to match a computed root with one embedded in a signature).

One example of a Signed Merkle Tree Proof is a "transparent signed statement" or "claim" as defined in [I-D.ietf-scitt-architecture].

3.2. Inclusion Paths

[RFC6962] defines a merkle audit path for a leaf in a merkle tree as the shortest list of additional nodes in the merkle tree required to compute the merkle root for that tree.

[RFC9162] changed the term from "merkle audit path" to "merkle inclusion proof".

We prefer to use the term "inclusion path" to avoid confusion with Signed Merkle Tree Proof.

If the tree size and leaf index is known, then a compact inclusion path variant can be used:

```
IndexAwareInclusionPath = #6.1234([
    leaf_index: int
    hashes: [+ bstr]
])
```

Otherwise, the direction for each path step must be included:

FIXME bit vector: 0 right, 1 left, so no bit labels

```
IndexUnawareInclusionPath = #6.1235([
    hashes: [+ bstr]
    left: uint  ; bit vector
])
```

For some tree algorithms, the direction is derived from the hashes themselves and both the index and direction can be left out in the path:

```
; TODO: find a better name for this
UndirectionalInclusionPath = #6.1236([+ bstr])

InclusionPath = IndexAwareInclusionPath / IndexUnawareInclusionPath / Un
```

Note: Including the tree size and leaf index may not be appropriate in certain privacy-focused applications as an attacker may be able to derive private information from them.

TODO: Should leaf index be part of inclusion path (IndexAwareInclusionPath) or outside?

TODO: Define root computation algorithm for each inclusion path type

TODO: Do we need both inclusion path types? what properties does each type have?

TODO: Should the inclusion path be opaque (bstr) and fixed by the tree algorithm? It seems this is orthogonal and the choice of inclusion path type should be application-specific.

3.3. Signed Merkle Tree Proof

A signed Merkle tree proof is a CBOR array containing a signed tree root, an inclusion path, extra data for the tree algorithm, and the payload.

```
SignedMerkleTreeProof = [
  signed_tree_root: bstr .cbor SMTR  ; payload of COSE_Sign1_Tagged is d
  inclusion_path: bstr .cbor InclusionPath
  extra_data: bstr / nil
  payload: bstr
]
```

extra_data is an additional input to the tree algorithm and is used together with the payload to compute the leaf hash. A use case for this field is to implement blinding.

TODO: maybe rename extra_data

3.4. Signed Merkle Tree Multiproof

TODO: define a multi-leaf variant of a signed Merkle tree proof like in:

- *https://github.com/transmute-industries/merkle-proof
- *https://transmute-industries.github.io/merkle-disclosure-proof-2021/

TODO: consider using sparse multiproofs, see https://medium.com/@jgm.orinoco/understanding-sparse-merkle-multiproofs-9b9f049e8f08 and https://arxiv.org/pdf/2002.07648.pdf

4. Merkle Tree Algorithms

This document establishes a registry of Merkle tree algorithms with the following initial contents:

[FIXME] exploration table, what should go into -00?

Name	Label	Description
Reserved	0	
RFC9162_SHA256	1	RFC9162 with SHA-256

Table 1: Merke Tree Alogrithms

Each tree algorithm defines how to compute the root node from a sequence of leaves each represented by payload and extra data. Extra data is algorithm-specific and should be considered opaque.

4.1. RFC9162_SHA256

The RFC9162_SHA256 tree algorithm uses the Merkle tree definition from [RFC9162] with SHA-256 hash algorithm.

For n > 1 inputs, let k be the largest power of two smaller than n.

```
MTH({d(0)}) = SHA-256(0x00 || d(0))
MTH(D[n]) = SHA-256(0x01 || MTH(D[0:k]) || MTH(D[k:n]))
```

where d(0) is the payload. This algorithm takes no extra data.

5. Privacy Considerations

TBD

6. Security Considerations

TBD

7. IANA Considerations

7.1. Additions to Existing Registries

7.1.1. New Entries to the COSE Header Parameters Registry

IANA will be requested to register the new COSE Header parameters defined below in the "COSE Header Parameters" registry at some point.

7.2. New SCITT-Related Registries

IANA will be asked to add a new registry "TBD" to the list that appears at https://www.iana.org/assignments/.

The rest of this section defines the subregistries that are to be created within the new "TBD" registry.

7.2.1. Tree Algorithms

IANA will be asked to establish a registry of tree algorithm identifiers, named "Tree Algorithms", with the following registration procedures: TBD

The "Tree Algorithms" registry initially consists of:

Identifier	Tree Algorithm	Reference
TBD	TBD tree algorithm	This document

Table 2: Initial content of Tree Algorithms registry

The designated expert(s) should ensure that the proposed algorithm has a public specification and is suitable for use as [TBD].

8. References

8.1. Normative References

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

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://doi.org/10.17487/RFC6234>.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <https://doi.org/10.17487/RFC6962>.

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <https://doi.org/10.17487/RFC6979>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://doi.org/10.17487/RFC8032>.

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

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://doi.org/10.17487/RFC8949>.

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

8.2. Informative References

[I-D.ietf-cose-countersign]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Countersignatures", Work in Progress, Internet-Draft, draft-ietf-cose-countersign-10, 20 September 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-countersign-10>.

[I-D.ietf-scitt-architecture]
Birkholz, H., Delignat-Lavaud, A., Fournet, C., and Y. Deshpande, "An Architecture for Trustworthy and Transparent Digital Supply Chains", Work in Progress, Internet-Draft, draft-ietf-scitt-architecture-01, 13 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-scitt-architecture-01>.

Authors' Addresses

Orie Steele
Transmute

Email: orie@transmute.industries

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: henk.birkholz@sit.fraunhofer.de

Maik Riechert
Microsoft
United Kingdom

Email: Maik.Riechert@microsoft.com

Antoine Delignat-Lavaud
Microsoft
United Kingdom

Email: antdl@microsoft.com

Cedric Fournet
Microsoft
United Kingdom

Email: fournet@microsoft.com