Merkle Tree Hash Documentation Format

1. Document Overview

This document describes the Merkle Tree Hashing structure and details the processes, parameters, and verification steps required for its implementation.

2. Version Information

Version	Date	Author	Description
1.0	2024-06-18	Jane Doe	Initial version

3. Merkle Tree Overview

A Merkle tree is a binary tree in which every leaf node is a hash of a data block, and every non-leaf node is a hash of its child nodes. It enables efficient and secure verification of large datasets.

4. Hash Function

- **Algorithm:** SHA-256
- **Input encoding:** UTF-8 (unless specified otherwise)

5. Merkle Tree Construction

1. Hash each data item to form the leaf nodes of the tree.
2. Pair leaf nodes and hash each pair to form parent nodes.
3. If a layer has an odd number of nodes, duplicate the last node before hashing.
4. Repeat until a single root hash remains.

Example

```
Data Items: [D1, D2, D3, D4]
Step 1. Leaf Hashes:
  H1 = SHA256(D1)
  H2 = SHA256(D2)
  H3 = SHA256(D3)
  H4 = SHA256(D4)
Step 2. Internal Nodes:
  H12 = SHA256(H1 + H2)
  H34 = SHA256(H3 + H4)
Step 3. Root:
  Root = SHA256(H12 + H34)
```

6. Merkle Proof (Verification)

1. Provide the list of hashes (Merkle proof) required to verify the inclusion of a specific leaf.
2. Hash up the tree combining the proof nodes as per level direction (left/right pairing).
3. Check if the computed root equals the known Merkle root.

Example Merkle Proof Format

```
{
  "leaf": "D3",
  "leaf_hash": "H3",
  "proof": ["H4", "H12"],
  "root": "Root"
}
```

7. Document Fields

Field	Type	Description
version	string	Document versioning
hash_algorithm	string	Name of the hash function used
root_hash	string	Merkle root hash value
leaf_hashes	array	Hashes of original data items
proof	array	Merkle proof hashes for verification

8. Sample JSON Format

```
{
  "version": "1.0",
  "hash_algorithm": "SHA-256",
  "data": ["D1", "D2", "D3", "D4"],
  "leaf_hashes": ["H1", "H2", "H3", "H4"],
  "root_hash": "Root",
  "proof": ["H4", "H12"]
}
```

Important Notes

- Ensure the hash algorithm is clearly specified and consistently applied.
- Duplicating the last node for odd layers ensures a balanced tree.
- Merkle proof should include all necessary hashes for verification up to the root.
- All data and hashes should be encoded/serialized in a standard way for interoperability.
- Document changes and version updates clearly within the format.