Merkle Tree Spec

Ann Weine

March 31, 2017

1 Data structure

type MerkleTreeElement with three **constructors**:

- MerkleTreeLeaf - the one for the leaf
- MerkleTreeHash - the one for the root that references a leaf (it has a hash of the corresponding leaf)
- MerkleTreeRoot - the one for the root.

The contrustructors have following **fields**:

- MerkleTreeRoot - hash, leftLeaf (reference to the left leaf), rightLeaf (reference to the right leaf, if it exists(NB: there are some cases when there is no right leaf, while there is the left root every time)), root (reference to the root, if it exists)
- MerkleTreeHashLeaf - leaf (reference to the corresponding leaf), hash (hash of the leaf), root (reference to the root)
- MerkleTreeLeaf - element, length (size of the element), root (reference to the root that corresponds to the leaf).

Data **types**: The following sub-structures are used

- for hashes - $uint8$ buffer with the size of $hash - function - output - size$
- for length - $uint32$

Sub-functions are used:

- hash function (that is used afterwards as hash(..). It takes a buffer of finite length and it length and returns the buffer of size $hash - function - output - size$. Hash function specification are out of the scope of the specification.
- concatenation function (that is used afterwards as '+'). It takes two buffers of size n each and returns the buffer of size 2n. The inner of the first buffer is placed on the $0..n-1$ addresses of buffer, the inner of the second buffer is placed on the $n..2n-1$ addresses of buffer respectively.

2 Methods over the structure

2.1 Exists

Checking the existance is possible by two possible ways. The first one is to go through all the elements. Second one could be somehow done by changing a structure. The need depends on the size of data structure.
On the proposed structure check of the existance could be done the following way:

to go through the all possible leafs until the hashed leaf. For the leaf check whether the value of the hash corresponds to the search. If yes, it means that element is found, otherwise one needs to go one level higher and continue the search.
As soon as element is found, proof system can return the path to the element as the set of hashes. The prover will be able to hash the search request and check the real presence of the element.

2.2 Insert

Due to certificate transparency we implement two different insertions. One of them is to insert the tree and the second one is insertion of a single leaf. Actually the idea of proof will not be changed depending on the method.
There is a possibility to add a proof to insertion process of the auditability (first proof). But for the current stage it is not important.
The addition is done the following way:
To the current root node it adds the new tree with new leafs. The new root is then computed.

2.2.1 Delete

Due to the fact that the structure is append-only, there is no need in delete implementation.

3 Properties

3.1 Correctness

We assume the data structure to be correct iff:

- for all MerkleTreeLeaf l
  - $l.root.hash = hash(l)$.
  - $l.root.leaf = l$
- for all MerkleTreeRoot l
  - $l.leftLeaf! = None$
  - $l.rightLeft = Some(MerkleTreeElementa)$ or $l.rightLeft = None$
  - $ifl.rightLeaf = None$ then $l.hash = hash(l.leftLeaf)$ else $l.hash = hash(l.leftLeaf + l.rightLeaf)$

4 Audit

4.1 Consistency

The property of consistency means that we have to prove that the previous tree is really exists in the new one and all the nodes were copied correctly.

4.2 Consistency II

To be able to check whether the leaf is presented in data structure, the system presents path to the leaf.
It means that on the side of system we do need to have a search. The data is presented the way it provides only set of hashes. It means that there is no way to leak additional data about the other leafs.
At the same time there is an attack in which a potential attacker could go through all the possible leaves to check the existence. This attack influences the metadata of structure. Nevertheless, it is out of the scope of the specification.