Correctness of the state hash

Lukasz Glen
lukasz.glen@imapp.pl, lukasz.glen@poczta.fm

Each ERC998 bundle can contain ERC20 tokens, ERC721 tokens and other ERC998 bundles. In particular a bundle can contain a bundle. The ERC998 contract prevents from circular dependencies. Thus bundles form a forest. A bundle has a state, which is a set of owned tokens and other bundles. Bundles are valuable objects and they are subjects to trade. So we need a protection from modifying a bundle just before a trade. A kind of state indicator that is easy to be checked, that is changed whenever the state of the bundle is changed or a state of any bundle in its subtree is changed. We propose something similar to Merkle tree but with the ability to track changes.

Let $I \in \mathbb{Z}_+$, $\mathbf{P} \subset \mathbb{Z}_+$, $\mathbf{R} \subset \mathbb{Z}_+$, such that sets $\{I\}, \mathbf{P}, \mathbf{R}$ are disjoint. The state S_i of i-th bundle, $i \in \mathbb{Z}_+$, is

$$S_i = \{S_{i,erc20}, S_{i,erc721}, S_{i,bundle}\}$$

where

$$S_{i,erc20}(p) \in \mathbb{N} \quad \text{for} \quad p \in \mathbf{P},$$

$$S_{i,erc721}(r) \subset \mathbb{Z}_+ \quad \text{for} \quad r \in \mathbf{R},$$

$$S_{i,bundle} \subset \mathbb{Z}_+.$$

The initial state is $S_{i,erc20}(p) = 0$, $S_{i,erc721}(r) = \emptyset$, $S_{i,bundle} = \emptyset$.

A state can be changed upon a transaction, a state of one bundle within one transaction. Within one transaction we perform one of the following operations.

1. $addERC20(i, p, v)$ provides a change $S_{i,erc20}(p) := S_{i,erc20}(p) + v$,

2. $removeERC20(i, p, v)$ provides a change $S_{i,erc20}(p) := S_{i,erc20}(p) - v$, assuming that output $S_{i,erc20}(p) \geq 0$,

3. $addERC721(i, r, a)$ provides a change $S_{i,erc721}(r) := S_{i,erc721}(r) \cup \{a\}$, assuming that input $a \notin S_{i,erc721}(r)$,

4. $removeERC721(i, r, a)$ provides a change $S_{i,erc721}(r) := S_{i,erc721}(r) \setminus \{a\}$, assuming that input $a \in S_{i,erc721}(r)$,

5. $addNode(i, j)$ provides a change $S_{i,bundle} := S_{i,bundle} \cup \{j\}$, assuming that input $j \notin S_{i,bundle}$ and no circular dependency is created,

6. $removeNode(i, j)$ provides a change $S_{i,bundle} := S_{i,bundle} \setminus \{j\}$, assuming that input $j \in S_{i,bundle}$.

The state hash H_i of i-th bundle is changed by a transaction. The initial value is

$$H_i = \text{keccak256}(I, i).$$

An operation within a transaction changes a state hash as follows. A state hash update occurs after a state update.

1. $addERC20(i, p, v)$ provides a change $H_i := \text{keccak256}(H_i, p, w)$, where $w = S_{i,erc20}(p)$,
2. $removeERC20(i, p, v)$ provides a change $H_i := \text{keccak256}(H_i, p, w)$, where $w = S_{i,erc20}(p)$,
3. $addERC721(i, r, a)$ provides a change $H_i := \text{keccak256}(H_i, r, a)$,
4. $removeERC721(i, r, a)$ provides a change $H_i := \text{keccak256}(H_i, r, a)$,
5. $addNode(i, j)$ provides a change $H_i := \text{keccak256}(H_i, I, H_j)$,
6. $removeNode(i, j)$ provides a change $H_i := \text{keccak256}(H_i, I, H_j)$.

Note that 1 and 2, 3 and 4, 5 and 6 have the same formula.

So if a state is changed then a state hash is updated. Additionally, if a state of i-th bundle is changed and $i \in S_{j,bundle}$ then within the same trasaction the state hash of j-th bundle is updated also, as follows. We perform the operation $updateNode(j, H_i)$.

- $updateNode(j, H_i)$ provides a change $H_j := \text{keccak256}(H_j, I, H_i)$, and if $j \in S_{k,bundle}$ then we perform recursively $updateNode(k, H_j)$ with output H_j.

Note that if a state of i-th bundle is changed then a state hash of i-th bundle is updated and all its ancestors up to the root owner within one transaction.

Our goal is to prove that the state hash is a good indicator of the state in the broader meaning - the state of the bundle and all its descendants. Good means that the probability that two different states have the same state hash (a collision) is neglectible. $keccak256$ is not injective. This causes troubles when providing a mathematical proof. We make the following assumption. Note that the state hash is an expression containing nested $keccak256$. We assume that the probability of collision for such expressions is neglectible. Having this we interpret H_i as a formal expression (a string of characters) rather than evaluating this expression. Thus we need to prove that two different states cannot be associated with the same state hash.

We say that a state hash H is proper if there exists a sequence of transactions that H is a state hash of i-th bundle: H_i.

Fact. A proper H determines the bundle (determines i).

Proof. If the state hash is of the initial form

$$H = \text{keccak256}(I, i)$$

then i is given. If the state hash is of the form

$$H = \text{keccak256}(H', a, w)$$

then H and H' are of the same bundle. ■

The depth $D(H_i)$ is defined as: it is a number of keccak256 in the expression.

Fact. Assume a state hash H_i. Then

a) the state S_i is uniquely determined,

b) if $j \in S_{i,bundle}$ then H_j is uniquely determined.

Proof. We use mathematical induction.

The statements hold when $D(H_i) = 1$.

Say that for all $D(H) < N$ the statements hold and $D(H_i) = N$. Consider a transaction that changed H_i:

$$H_i = \text{keccak256}(H'_i, a, w)$$

and S'_i the state of i-th bundle before the transaction. Note, that S'_i is uniquely determined. Note also that it is enough to proof that the operation that caused transition $H'_i \to H_i$ is determined uniquely by H_i.

Consider $a \in P$. If $w > S'_{i,erc20}(a)$ then the last operation was

$$addERC20(i, a, w - S'_{i,erc20}(a))$$

and holds $S_{i,erc20}(a) = w$. If $w < S'_{i,erc20}(a)$ then the last operation was

$$removeERC20(i, a, S'_{i,erc20}(a) - w)$$

and holds $S_{i,erc20}(a) = w$.

Consider $a \in R$. If $w \notin S'_{i,erc721}(a)$ then the last operation was

$$addERC721(i, a, w)$$

and holds $S_{i,erc721}(a) = S'_{i,erc721}(a) \cup \{w\}$. If $w \in S'_{i,erc721}(a)$ then the last operation was

$$removeERC721(i, a, w)$$

and holds $S_{i,erc721}(a) = S'_{i,erc721}(a) \setminus \{w\}$.

Consider $a = I$. Then $w = H_j$. If $j \notin S'_{i,bundle}$ then the last operation was $addNode(i, j)$. So assume that $j \in S'_{i,bundle}$. The last operation can be $removeNode(i, j)$ or $updateNode(i, H_j)$. Let H'_j be the state hash of j-th bundle before the transaction. Note that $updateNode(i, H_j)$ implies that the state hash of j-th bundle changed within this transaction. So if $H'_j = H_j$ then the last operation was $removeNode(i, j)$, in the other case the last operation was $updateNode(i, H_j)$. ■

Remark. The fact above holds also if we admit operations $addERC20(i, p, 0)$ and $removeERC20(i, p, 0)$. The case is that these two does not change the state so the state hash cannot distinct them.

Applying the fact above recursively, we got the following.

Fact. The state hash determines the state of the bundle and states of all bundles in its subtree.

From the proof of the fact above, we got the following.

Fact. The state hash uniquely determines the sequence of transactions that affected the state of subtree.

Concluding. The state hash encodes not only the state but also the history of how this state was created. Although it has some limitations. The order of independent changes to children bundles cannot be determined by the state hash. For instance.

$$addERC20(2, r, 1), \quad addERC20(3, r, 1), \quad addNode(1, 2), \quad addNode(1, 3)$$

and

$$addERC20(3, r, 1), \quad addERC20(2, r, 1), \quad addNode(1, 2), \quad addNode(1, 3)$$

are sequences of transactions that yield the same H_1. It can be overcome by adding a global transaction counter, if there is enough practical reasoning.

Two final comments. 1. The state hash is similar to Merkle tree - a single hash determines the structure of all tree. But utilization is different. 2. The state hash is better than last modification time or changes counter. Those do not determine the state and this leaves some space for adversaries.