On the impossibility to forge illegitimate proofs of membership in Merkle (Patricia) Trees

Jerémie Albert and Serge Chaumette

2021/12/08

Jérémie Albert is co-founder at inBlocks, Serge Chaumette is Professor at the University of Bordeaux, researcher at LaBRI and advisor at inBlocks.

This paper is a draft version that may contain minor errors. Its should be only considered as such and any remark is welcome. Its goal is to prove the impossibility to forge illegitimate proofs of membership.

Contents

1 Introduction

inBlocks is a blockchain startup company founded in 2018 that created a SaaS platform designed to create digital assets. An asset, in a business perspective, is a promise to a future benefit. Depending on the business of our customer, this digital asset might be photographs, movies, sound, text document or structured key/value representations (like XML or JSON files). Whatever its format all these information can be considerered as binary data and in the following we do not assume any specific format regarding these digital assets; we consider them as binary data and we might refer to them simply as data.

The **digital asset creation** relates mainly on the capacity we have to demonstrate its existence. The data that can be used to demonstrate such existence is called a **proof of existence**. The keystone value of a proof of existence is related to its timestamping and the capacity for our platform to demonstrate, in a undeniable way, the veracity of this timestamp. To demonstrate the existence of a piece of data at a certain point in time we assume that the recording of its fingerprint (a small series of bits that is unique for any numeric item) in a secure and distributed ledger like Bitcoin or Ethereum is perfectly safe, meaning it cannot be forged. However, for real use case scenarios, the cost (in terms of euros or dollars) of such an operation is very high. Our goal at inBlocks is to be able to provide this mechanism to any type of data, even for those

whose value is unknown (and so might be very low). The solution we provide is a way for our customer to minimize this cost by aggregating the required operations by using **an open-source solution called Precedence (developed at inBlocks)**.

Precedence is an open-source software solution to create numerical assets. It is integrated to our platform to allow our customer to create data repository structured as blockchain. It has the same key characteritics like the impossibility to remove a record from an already existing block without having to recomputed all the blocks that refer to it (*i.e.* all the blocks that we created after it). The data structure that is used in Precedence is a a Merkle Patricia Tree. A MPT is a arborescent data structure designed to reduce the data size of the proof of existence of every data that compose it.

The goal of this paper is to demonstrate that the inBlocks software layers do not impact the security of the overall system. *i.e.* that the security is the same when using inBlocks platform and when using a public blockchain directly.

To achieve this goal we proceed as follows:

1. we give clean definitions of the different forms of trees that are used (Tree, Merkle Tree and Merkle Patricia Tree),
2. we then show that no proof of membership of a piece of data can be forged.

2 Tree

2.1 Definition

A Tree s the combination of a value (the value of its root) and of a number of children (Trees).

Definition 2.1 (Tree)

Let N be a set of values.
If $n \in N$ then $A = (root = n, children = \emptyset)$ is a Tree.
If $n \in N, a_1, a_2, ...a_n$ are trees then $A = (root = n, children = \{a1, a2, \ldots, an\})$ is a tree, and root is the root of this tree.

It should be noted that in this definition we do not consider the notion of node as can be found in the literature. The reason for that is because we are interested in values that are stored and the notion of node is thus not necessary in what follows.

Definition 2.2 (Children)

Let A be a Tree.
$Children(A) = A.children$ is the set of Children of A.

Definition 2.3 (Leaf)

A Tree with no child ($children = \emptyset$) is called a Leaf.

The Root of a Tree is defined by construction of this Tree, as follows:

Definition 2.4 (Root of a tree)

Let $A = (root, children)$ a tree.
$root(A) = A.root$ is the root of A.

Definition 2.5 (Sub Tree)

Let A be a Tree.
P is a Sub Tree of A iff P=A or P is a Sub Tree of a child of A.

Definition 2.6 (Father)

Let A be a Tree.
The Father of A in a Tree T is the Tree P such that P is a SubTree of T and $A \in Children(T)$.

Notation: we will write $Father(A \in T)$ to denote the Father of A in T and Father(A) when there is no ambiguity on T.

2.2 Operations on a Tree

The Brothers of a Tree are the Children of its Father minus itself.

Definition 2.7 (Brothers)

Let A be a Tree.
Brothers(A)=Children(Father(A))-A.

The Uncles of a Tree are the Brothers of its Father.

Definition 2.8 (Uncles)

Let A be a Tree.
Uncles(A)=Brothers(Father(A)).

The i-Uncles of a Tree are its Uncles at the i-th level.

Definition 2.9 (i-Uncles)

Let A be a Tree.
i-Uncles(A,1)=Uncles(A).
i-Uncles(A,i)=Uncles(Father(A),i-1).

The *i-Uncles of a Tree are its Uncles and the Uncles of its ancestors in the Tree up to the i^th level.

Definition 2.10 (*i-Uncles)

Let A be a Tree.
$\star i - Uncles(A, 1) = Uncles(A)$
$\star i - Uncles(A, i) = Uncles(A) \bigcup \star i - Uncles(Father(A), i - 1)$

Property 2.1

$\star i - Uncles(A, i) = \cup_{j=1}^{i} i - Uncles(A, j)$

3 Merkle Tree

A Merkle Tree is a tree built as follows. The value of a leaf is a hash. The value of a node is the hash of the concatenation of the values of its children.

Rational
To check the validity of a node, it is thus enough to have the root of the tree, the values of its brothers and of its uncles. This property will be detrailed later in this paper.

3.1 Definition

Let # be a one-way hash function defined over a set of values D. Let $H = \#(D)$ be a set of hash values.

Definition 3.1 (Merkle Tree)

If $n \in H$ then $A = (root = n, \emptyset)$ is a Merkle Tree.
If $\{a_1, a_2, ..., a_m\}$ is an ordered set of Merkle Trees then $A = (root = \#(root(a1).root(a2).....root(an)), children = \{a1, a2, \ldots, an\})$ is a Merkle Tree and root is the root of this tree.

The following property results from the construction of a Merkle Tree.

Property 3.1

A Merkle Tree is a Tree.

3.2 Operations on a Merkle Tree

We introduce the notion of a *Futur Merkle Tree* that will be useful to express attacks attempts that consist in replacing part(s) of the Tree with fake data.

Rational
This could be used to forge false proofs of membership in the blockchain, and we will thus use it to prove the impossibility of this attack. This will be detailed later in this paper.

Definition 3.2 (Future Merkle Tree)

A Future Merkle Tree is a Merkle Tree with a hole.
It is noted $T[]$.

Note that populating a Future Merkle Tree should be done with a Merkle Tree and also implies to recompute part of the tree. If not, the result is a Tree but not a Merkle Tree.

Definition 3.3 (Subtree substitution)

Let a_i be a Merkle Tree.
Let $MT = FMT[a_i]$ be a Merkle Tree.
$MT\{a_i \rightarrow a'_i\} = FMT[a'_i]$.

$MT\{a_i \rightarrow a'_i\}$ is the same tree as MT except a_i has been replaced by a'_i and consequently part of the tree has been recomputed.

3.3 Properties of a Merkle Tree

Property 3.2

Let $A = T[a_i]$ be a Merkle Tree.
$T[a'_i]$ (without recomputation) is a Merkle Tree iff $a'_i = a_i$.

The following properties states that is is not possible to change a subtree of a Merkle Tree without modifying its root.

Property 3.3 (One Sub Tree substitution instability)

Let a_i, a'_i be two distinct Merkle Trees.
Let $MT = FMT[a_i]$ be a Merkle Tree.
Then $root(FMT[a'_i]) \neq root(MT)$

Proof 3.1

The proof id build by induction on the depth of the tree.

- *for p=1*
  Let A be a Merkle Tree of depth 1, $A = (root, \emptyset) = FMT[(root, \emptyset))]$

  Let a be another Merkle Tree. If $Root(FMT[(root, \emptyset)]) = Root(a)$ *then we have two possibilities:*
  1. *either* $Root(a) = root$*, and this is possible if and only if* $a = (root, \emptyset)$ *which is not possible because we said the replacement Tree should be different from the original Tree*
  2. *or* $a = (root, a_1, ..., a_n)$ *with* $root = \#(\#_{i=1}^{n} a_i)$ *what is not possible / too costly by assumptions on* $\#$
- *for* $p > 1$ *(we consider a tree of depth* $p > 1$*)*
  Let $A = FMT[a_i]$ *be a tree of depth p,* $p > 1$
  Induction hypothesis: $\forall a'_i, Root(FMT[a'_i]) \neq root(A)$
- *for* $p + 1$
  Let $A = (\#(a_1.root, ..., a_n.root), a_1, ...a_n)$ *and let* a_i *be the tree where the substitution is done. This substitution is thus at depth* $\leq p$*.*
  By hypothesis $Root(a'_i) \neq root(a_i)$ *and then we have* $\#(a_1.root, ..., a'_i, ..., a_n.root) \neq \#(a_1.root, ..., a_i, ..., a_n.root)$ *since it would be too costly to build this equality by definition of* $\#$*. cqfd*

We now show that making several substitution is equivalent to making only one.

Property 3.4 (Multi Future Merkle Tree)

A Multi Future Merkle Tree can be written as follow $A = FMT2(FMT1[a_1])[a_2]$

Property 3.5 (Two Sub Trees substitution instability)

Let a_1, a'_1, a_2, a'_2 *be distinct Merkle Trees.*
Let $MT = FMT2(FMT1[a_1])[a_2]$ *be a Merkle Tree.*
Then $root(FMT[a'_1])[a'_2] \neq root(MT)$

Proof 3.2

Let us consider A the smallest common ancestor of the two holes.
Assume $A = (\#(a_1.root, ..., a_n.root), a_1, ..., a_j, ..., a_n)$ *and let* a_i *and* a_j *be the two sub trees where the holes have been substituted.*
By hypothesis $Root(a'_i) \neq root(a_i)$ *and* $Root(a'_j) \neq root(a_j)$ *and then we by induction we have* $\#(a_1.root, ..., a'_i, ..., a'_j, ..., a_n.root) \neq \#(a_1.root, ..., a_i, ..., a_j, ..., a_n.root)$ *since it would be too costly to build this equality by definition of* $\#$*. cqfd*

4 Merkle Patricia Tree

A Merkle Patricia Tree is a Merkle Tree that stores (key, value) pairs. It makes it possible to check the existence of a (key, value) pair in a Tree, without giving away any a additional information.

Property 4.1

A proof of membership of a pair (key, value) in a Merkle Patricia Tree cannot be forged.

Proof 4.1

This results from the fact that a Merkle Patricia Tree is a Tree and of the theorems of the previous section.