Integration of Ethereum & IPFS

Yoann Raucoules

ARHS Spikeseed

yoann.raucoules@arhs-spikeseed.com

July 25, 2017

Abstract

The present document provides an overview of the integration of Ethereum & IPFS in the context of the FutureTrust project. Those two technologies have been used in order to implement a Global Trust Service Status List (gTSL) in a decentralized and secure manner. The objective of using Ethereum & IPFS is to store the whole data regarding the gTSL in a distributed network. In this paper, we present an overview of each technology, the design of the adopted solution and the version control designed on top of IPFS which is used to keep track of the gTSL history.

I. Ethereum overview

Ethereum is an open blockchain platform that lets anyone build and use decentralized applications that run on blockchain technology. A blockchain is a distributed computing architecture where every network node executes and records the same transactions, which are grouped into blocks. Only one block can be added at a time, and every block contains a mathematical proof that verifies that it follows in sequence from the previous block. In this way, the blockchain's "distributed database" is kept in consensus across the whole network. Individual user interactions with the ledger (transactions) are secured by strong cryptography.

The particularity of the Ethereum blockchain is that it has been designed in order to enable users to write and run smart contracts. A smart contract describes a computer code that can facilitate the exchange of money, content, property, shares, or anything of value on the blockchain. When running on the blockchain, a smart contract becomes like a self-operating computer program that is automatically executed when specific conditions are met. Because smart contracts run on the blockchain, they run exactly as programmed without any possibility of censorship, downtime, fraud or third party interference. Ethereum's core innovation, the Ethereum Virtual Machine (EVM) is a Turing complete software that runs on the Ethereum network. It enables anyone to run any smart contracts, regardless of the programming language given enough time and memory. The most used language used on the Ethereum blockchain is Solidity which once compiled can be run by the EVM.

The great advantage of running applications on a blockchain is to benefit from its properties:

1. Immutability – A third party cannot make any changes to data.

2. Corruption & tamper proof – Apps are based on a network formed around the principle of consensus, making censorship impossible.

3. Secure – With no central point of failure and secured using cryptography, applications are well protected against hacking attacks and fraudulent activities.

4. Zero downtime – Apps never go down and can never be switched off.

II. IPFS overview

The InterPlanetary File System (IPFS) is a peer-to-peer distributed file system that seeks to connect all computing devices with the same system of files. It presents a new platform for writing and deploying applications, and a new system for distributing and versioning large data. In other words, IPFS provides a high throughput content-addressed block storage model, with content-addressed hyper links. This forms a generalized Merkle DAG, a data structure upon which one can build versioned file systems, blockchains, and even a Permanent Web. IPFS has no single point of failure, no nodes are privileged and nodes do not need to trust each other. IPFS nodes store IPFS objects in local storage. Nodes connect to each other and transfer objects. These objects represent files and other data structures.

The IPFS Protocol is divided into a stack of sub-protocols responsible for different functionalities:

1. Identities – manage node identity generation and verification.
2. Network – manages connections to other peers, uses various underlying network protocols.
3. Routing – maintains information to locate specific peers and objects. Responds to both local and remote queries.
4. Exchange – a novel block exchange protocol (BitSwap) that governs efficient block distribution. Modelled as a market, weakly incentivizes data replication.
5. Objects – a Merkle DAG of content-addressed immutable objects with links. Used to represent arbitrary datastructures, e.g. file hierarchies and communication systems.
6. Files – a file system hierarchy inspired by Git.
7. Naming – a self-certifying mutable name system.

These subsystems are not independent; they are integrated and leverage blended properties.

III. Design of the solution

In this part, we describe the way to integrate Ethereum and IPFS, and the purpose to use those technologies together.

i. The advantage of coupling the two technologies

A simple way to store the gTSL data is to keep them in a blockchain, this avoiding the need of a decentralized file system. Although a blockchain has a lot of advantages, it also have some drawbacks. One of them is that the deployment of a smart contract within the Ethereum blockchain has to be paid. Furthermore, the size of the smart contract (in terms of operations and memory) is directly linked to the price. Thus, the price follows the amount of data that are stored. A way to reduce the cost is to use a distributed network in order to store the gTSL data and use a blockchain in order to maintain the state of the data. In IPFS, all data are addressed by their hash, making their state easily maintainable. Hence, for each stored gTSL data, the blockchain should only maintain a hash corresponding to the current state of the data. This last solution reducing considerably our costs resulting to our use of the blockchain.

ii. The way to integrate the two technologies

As explained in the previous part, only the hash of the data is stored in the blockchain. All the actual data are stored in IPFS, which is called the *data layer*. A hash is the address which identifies a particular Trust Service List (TSL). Hence, when a TSL is stored in IPFS, a hash of its data is automatically computed. As expected, this hash is a unique identifier of the TSL. This hash is then stored in a smart contract in which it is attached

to the unique non-mutable identifier of the TSL (which is a country code according to [ETSI TS 119 612]). This non-mutable identifier can then be used by end users when they intent to find the hash of a TSL that they need. In contrast to the non mutable identifier of the TSL, the hash of a TSL depends on the data it contains, this making it a mutable identifier of its TSL. Hence, a TSL can be modified by a member state, and its hash must then be updated. As the blockchain saves the current state of the TSL and updating the TSL corresponds to modify the state of the contract, the blockchain is called the *state layer*. The smart contract maintains the state of each TSL and allows the users to add, update, remove or fetch a TSL. Note that an authorisation is required for modifying the TSL but reading is publicly available.

An abstraction of the smart contract's implementation is given in Listing 1:

Listing 1: *Ledger Contract*

```
contract Ledger {

  // Structures
  struct Tsl {
    bytes32 code;
    bytes hash;
  }

  // Attributes
  mapping(bytes32 => Tsl) public gtsl;

  function addTsl
    (bytes32 _code, bytes _hash)
    public
  {
    gtsl[_code] = Tsl({
      code: _code,
      hash: _hash
    });
  }

  function updateTsl
    (bytes32 _code, bytes _hash)
    public
  {
    gtsl[_code].hash = _hash;
  }

  function removeTsl
    (bytes32 _code)
    public
  {
    delete gtsl[_code];
  }

}
```

Note: A *get* function is generated by the compiler for every public attribute.

IV. Version control on top of IPFS

As specified in the gTSL's requirements, a history of each TSL must be kept. To achieve this, the idea is to maintain a linked-list of all the versions for each TSL. This linked-list is stored in IPFS and each node (so-called a *commit*) of the list can be viewed as a JSON object, defined as following :

Listing 2: *Commit Object*

```
{
 "dataAddress": "QmXXXXXXXXXXXXXX",
 "parent": "QmYYYYYYYYYYYYYY",
 "author": "Ada Lovelace ada@lov.cc",
 "date": "Mon July 24 15:19:12",
 "signature": {"signature info"}
}
```

1. dataAddress – is the IPFS address (hash) pointing to the data of the TSL for the current version. This hash corresponds to the new address of the TSL.

2. parent – is the IPFS address (hash) pointing to the previous version (commit object) of the TSL.

3. author – identifies the user who updated the TSL.

4. date – represents the time-stamp of the commit.

Instead of saving the hash of the data in the blockchain, it is the hash of the commit object which is saved. This allows the user to retrieve: the data of the TSL; the user who

created it; the time at which it was created; and the previous version. This layer on top of the data containing meta-data of the version is called *meta-data layer*.

V. Results

This following diagram summarizes the version control and the integration of Ethereum & IPFS:

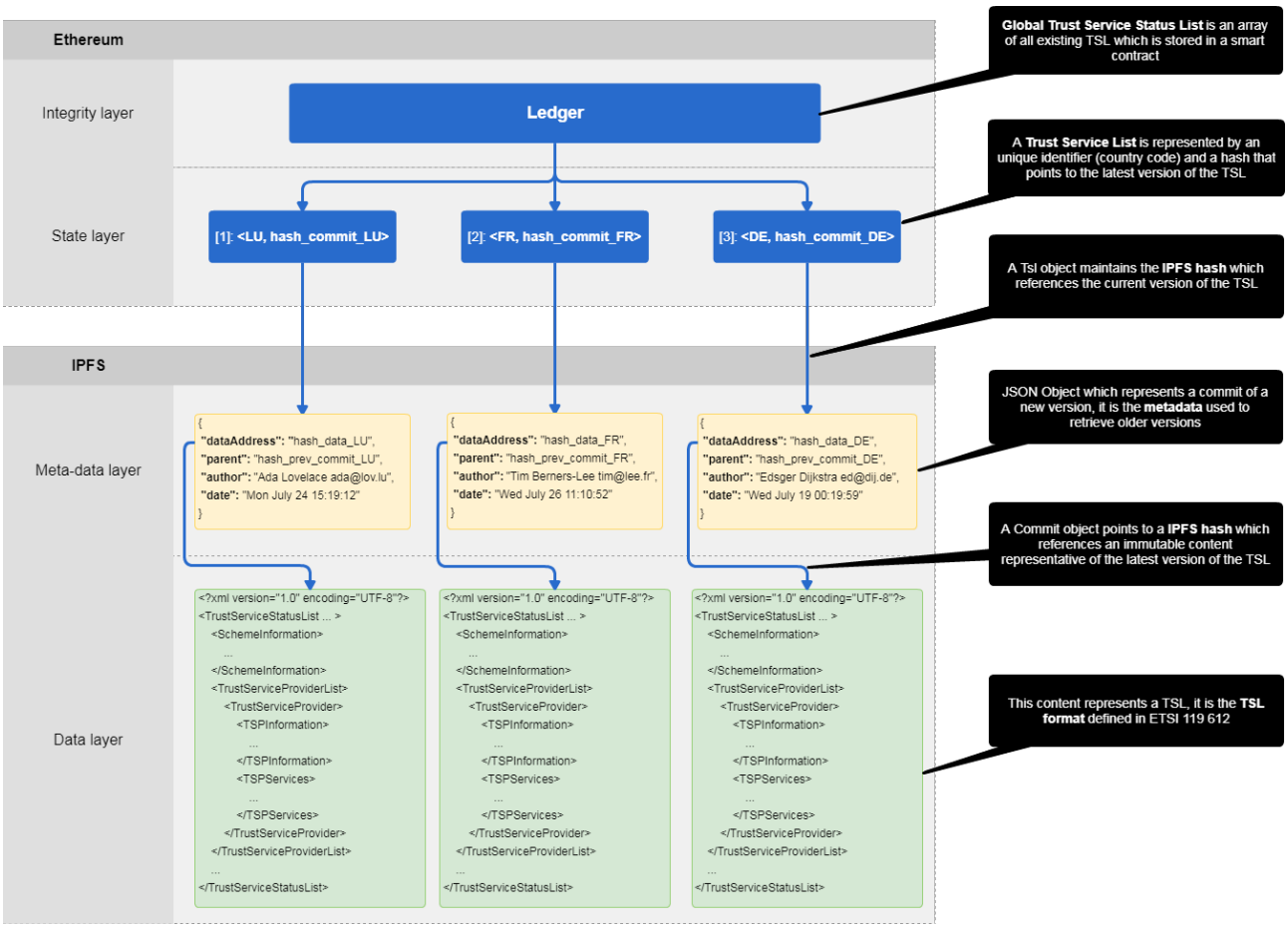

Figure 1: *Summary diagram*

References

[ETSI TS 119 612] Electronic Signatures and Infrastructures (ESI); Trusted Lists.

[Ethereum White Paper] A Next-Generation Smart Contract and Decentralized Application Platform.

[IPFS White Paper] Juan Benet (2014). IPFS - Content Addressed, Versioned, P2P File System.