Technische Universität Berlin

Documentation

IOSL: Distributed Identity Management in the Blockchain

Authors:

May 13, 2017

Contents

1 Introduction

1.1 Background

1.1.1 SONIC Project

Today's communication happens mostly through Online Social Network (OSN), but with their proliferation, some problems have arisen. The main issue is related to the fact that OSN are built in a closed an proprietary manner (think of Facebook, Twitter or LinkedIn) and don't allow a smooth communication among them. Therefore, the user is forced to create different accounts, and build different networks within for every platform. All this accounts and information correspond to the same user, but are heavily segregated from one another. This segregation is caused by the platform which aim to bind the user with a lock-in effect.

The SONIC project has been proposed by the team working for the Telekom Innovation Laboratories [clarify this] to overcome this problem and facilitate a seamless connectivity between different OSN [reference 1] allowing the migration of accounts between different platforms.

The project is supported by a distributed and domain-independent ID management architecture where a GSLS (Global Social Lookup System) is employed to map Global ID to URL of a social profile. This mapping is possible thanks to datasets called Social Records which are digitally signed.

1.1.2 Blockchain

1.1.3 Identity Management

2 Related Work

2.1 Directory Service

Also known as name service, maps the names of network resources to their respective network addresses. Each resource on the network is considered an object by the directory server. Information about a particular resource is stored as a collection of attributes associated with that resource or object. One of the most representing example is the DNS, which offers internet domain names translation to ip addresses.

2.2 DHT and P2P Networks

A distributed hash table (DHT) is a class of a decentralised distributed system that provides a lookup service similar to a hash table: (key, value) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key

2.2.1 DHT characteristics

Autonomy and decentralization: multiple nodes, no central coordinator;

Fault tolerance: reliable, no single point of failure (like Napster's central server) (Byzantine fault tolerance);

Scalability: the system scales. This means it's working efficiently regardless the workload;

Load balancing (optional);

Data integrity (optional).

2.2.2 Functioning Example

Once these components are in place, a typical use of the DHT for storage and retrieval might proceed as follows. Suppose the keyspace is the set of 160-bit strings. To index a file with given filename and data in the DHT, the SHA-1 hash of filename is generated, producing a 160-bit key k, and a message put(k, data) is sent to any node participating in the DHT. The message is forwarded from node to node through the overlay network until it reaches the single node responsible for key k as specified by the keyspace partitioning. That node then stores the key and the data. Any other client can then retrieve the contents of the file by again

hashing filename to produce k and asking any DHT node to find the data associated with k with a message get(k). The message will again be routed through the overlay to the node responsible for k, which will reply with the stored data.

2.2.3 The Network

Each node maintains a set of links to other nodes (its neighbors or routing table). Together, these links form the overlay network. A node picks its neighbors according to a certain structure, called the network's topology.

Aside from routing, there exist many algorithms that exploit the structure of the overlay network for sending a message to all nodes, or a subset of nodes, in a DHT. These algorithms are used by applications to do overlay multicast, range queries, or to collect statistic. (Flooding).

Implementations Considered

Kademlia

TomP2P

2.3 Blockchain Implementations

Bitcoin

Namecoin

Hyperledger

Ehtereum

2.4 Blockstack Use Case

2.5 How to Store Social Records

2.5.1 Solution 1

Store the Social Record directly in the data field of a transaction. Then the problems are:

How do I retrieve the social record?

Every time i update the social record, the transaction address changes

Storing an entire social record costs more gas than storing only its hash.

How should the social record be stored? Only one contract with all the Social Records inside? But then other problems arise like: the gas cost per each operation, the necessity to build a data structure inside the contract losing all the good properties provided by the blockchain. Then the data field space is only about 44KB.

If you use one transaction per Social Record, then you have to search through the entire blockchain, looking inside the transaction tree starting from the merkel root per each block. Not that efficient.

2.5.2 Solution 2

This solution is inspired by Blockstack architecture and requires to create a DHT to map the GlobalID to the address of the transaction contained inside the blockchain.

The transaction holds in the data field the hash of the Social Record. In this manner, the blockchain has the role of validation infrastructure rather than storage system. The actual data is stored in the distributed hash table. This solution could be vulnerable in case a malicious node tries to restore an old version of the social record by injecting the wrong address in the DHT table and causing a wrong mapping.

Another vulnerability could be in the consistency of the data, some node could be inconsistent id the DHT doesn't get updated. Therefore, the version control should be handled by the blockchain. Most of the vulnerabilities depend on the fact that a transaction stored in the blockchain has a different address every time we update the information contained within the Social Record.

2.5.3 Solution 3

The third solution is based on the second one but requires each user to hold an Ethereum Wallet. In such design each transaction contains both the Wallet public key and the hash of the Social Record within the data field. Every time the Social Record gets updated, its hash changes and the new transaction will have a different address.

The idea is to use the wallet public key as id, and map the GlobalID with the Wallet ID which will remain always the same. Searching inside the blockchain looking for the ID, the ledger will return the latest version. Compared to the second solution, the former provide version check through the blockchain rather than DHT.

2.6 Security

2.6.1 Elliptic Curves

ECC is the next generation of public key cryptography, and based on currently understood mathematics, it provides a significantly more secure foundation than first-generation public key cryptography systems like RSA.

With ECC, you can use smaller keys to get the same levels of security. Small keys are important, especially in a world where more and more cryptography is done on less powerful devices like mobile phones. While multiplying two prime numbers together is easier than factoring the product into its component parts, when the prime numbers start to get very long, even just the multiplication step can take some time on a low powered device. While you could likely continue to

keep RSA secure by increasing the key length, that comes with a cost of slower cryptographic performance on the client. ECC appears to offer a better tradeoff: high security with short, fast keys.

3 Concept and Design

4 Implementation

5 Evaluation

6 Conclusion

A Code