

# Proof of Personhood tokens on the Ethereum blockchain

Hugo Roussel

School of Computer and Communication Sciences

Decentralized and Distributed Systems lab

Bachelor Project

December 2017

**Responsible**  
Prof. Bryan Ford  
EPFL / DEDIS

**Supervisor**  
Linus Gasser  
EPFL / DEDIS

## 0.1 Introduction

Internet is often a trade-off between anonymity and accountability. Imagine an internet forum where users wish to share their political views anonymously but still wish to avoid "trolls", members that don't add value to the discussion. It's always possible to block an anonymous account but the "troll" can create a new one easily. This is not a viable solution. The forum could block certain IP addresses but once again it suffices for the troll to use a VPN or another connection. Another solution would be to ask each new member to provide an ID and verify their identity, however the discussion isn't anonymous anymore.

We will explain in the following paper a way to reconcile free speech and responsibility and one implementation of it : Proof of Personhood tokens and the advantages of porting such a system on a public blockchain.

## 0.2 Proof of Personhood tokens

### 0.2.1 What are Proof of Personhood tokens?

“Accountable anonymous credentials“

— Proof-of-Personhood: Redemocratizing  
Permissionless Cryptocurrencies

Proof of personhood tokens associate one person with one token. Binding a physical identity with a digital identity. The results can be achieved by combining pseudonym parties and cryptographic tools.

### 0.2.2 Pseudonym parties

The idea behind a pseudonym party is to verify persons and not identities. A party goes as follows : attendees met at designed location. Each attendee deposits a public key and once done, the key is added to the set of key of other participants. The person then is marked using permanent marker to insure he doesn't deposit twice. Once each attendee had the chance to register, the party ends and the organizers release the final statement of the party : the keyset of all participants along with details of the party. For transparency purposes the party can be recorded, in that case the hash of the tape is added to the final statement.

### 0.2.3 Current implementation

The current DEDIS implementation of PoP tokens is implemented using Golang on the Skipchain. The Skipchain is a blockchain created at EPFL by the DEDIS laboratory. The service provides commands for organizers and attendees to link to the skipchain for interacting with it

### 0.2.4 Limitations

At the moment skipchain is a private blockchain, meaning that only certified nodes can be added to it. The motivation of the project was to port the service to a public blockchain where ultimately it could be useful to more users.

## 0.3 Ethereum

### 0.3.1 Ethereum, Solidity and smart contracts

Ethereum is an open-source public blockchain which can perform decentralized computations on the ethereum virtual machine. The EVM is a worldwide computer than anyone can use in exchange for a small fee payed in the crypto-currency of the platform, ether. Scripts running on the EVM are called "smartcontracts". They must be written in Solidity, a statically-typed programming language inspired in its syntax by Javascript. Solidity is compiled to bytecode that is then pushed to the EVM.

### 0.3.2 Why use Ethereum?

The value of a network is proportional to the square  
of the number of connected users of the system

— Metacalfe's law

The ethereum platform quickly gained in popularity after its launch. The ecosystem has many tools to interact with the blockchain and to create decentralized applications. It is currently the network with the most nodes making it more secure and less centralized. A public blockchain also provides a greater transparency as every actor can verify who interact and how with the contract. This can be done using a block explorer.

### 0.3.3 A quick note on testnets

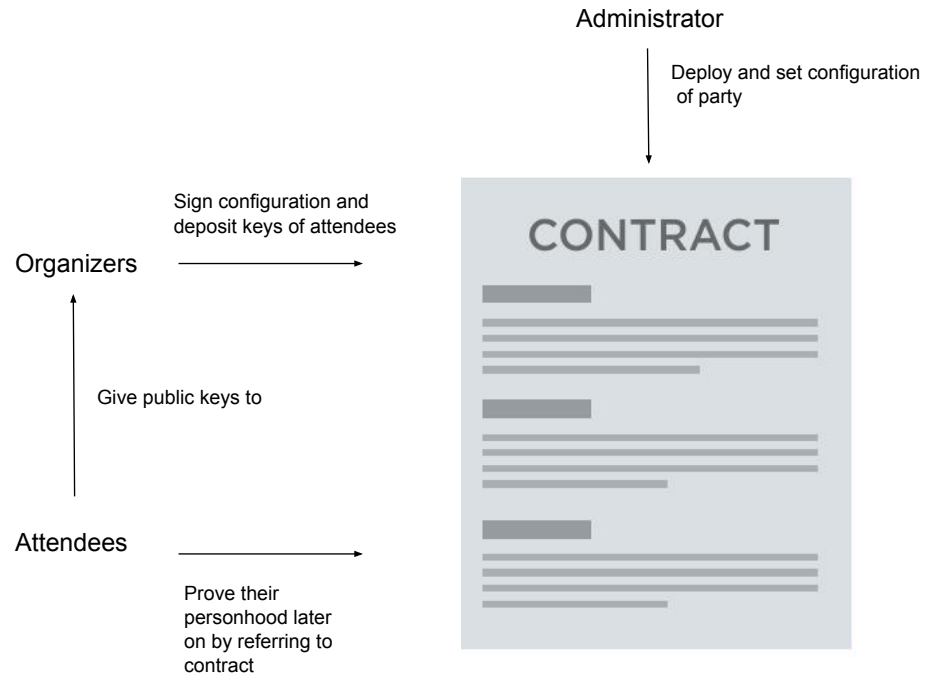
To help developers, the ethereum network provides multiples testnets with free test ethers. Each testnet is similar to the main network with minor differences. The testnet used in this project was the Rinkeby network.

## 0.4 Implementation of Popcontract

The popcontract is a smart-contract used to organize and store the informations of a pseudonym party. It is modeled by a finite state machine to improve security and reliability.

### 0.4.1 Abstract

Create a smart-contract that will store the information of the pseudonym party. At the end of the party, the contract is then locked, rendering it immutable. Each user can then refer to it knowing its genuinity. The second part of the project consisted in making the system interoperable with the current DEDIS implementation to have a clear and easy to use interface.



**Figure 1:** How actors interact with contract

### 0.4.2 Popcontract finite state machine

One of the ways to improve security of a smart contract is to transform it into a finite state machine (FSM). Thus at each state only certain functions can be called. This reduce the risk that some functions will be called maliciously. The contract is modeled by a FSM with only five states :

- initial state (contract was successefully deployed)
- configurationSet state (administrator provided the details of the party)

- configurationSigned state (organizers agree with the configuration)
- key-deposited state (at least one keyset was sent to contract)
- locked (party ended and final keyset was choosed)

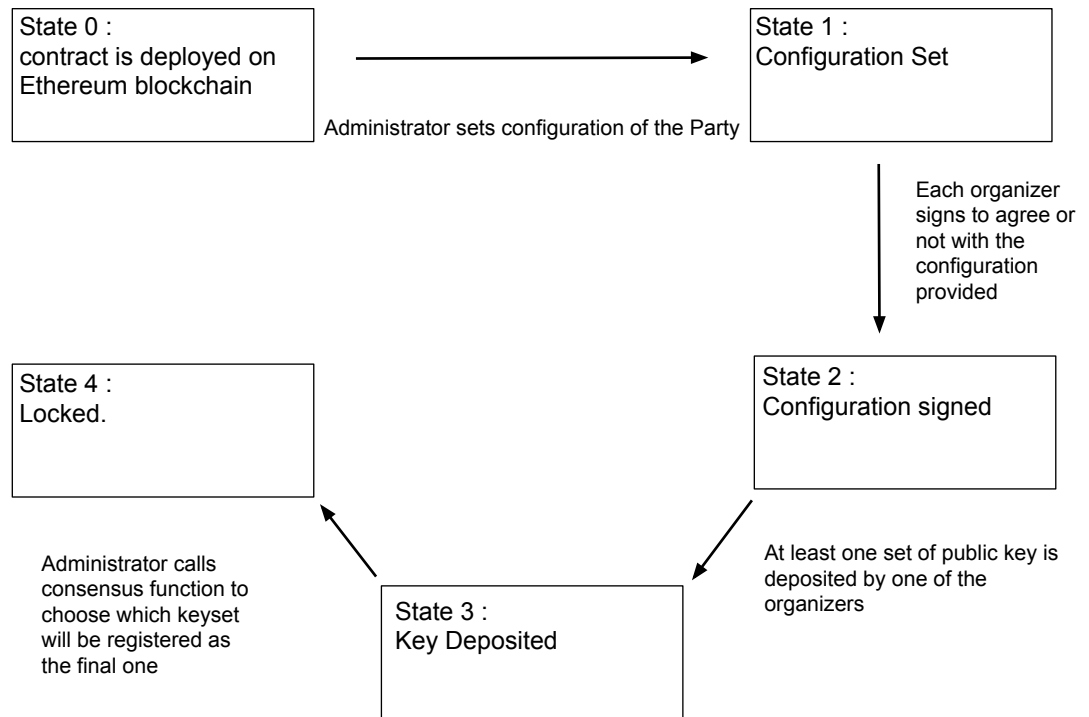


Figure 2: Popcontract FSM

### 0.4.3 Technical implementation

The project focused primarily on the implementation of pop tokens, and not in the verification service of attendee personhood using it.

#### Configuration of party

The configuration of the party is defined as follows : a name, a location, an array of public keys of organizers (in ethereum format) as well as a deadline. Passed the deadline, no function of the contract is callable, preventing it from being modified. The administrator sends a transaction signed with its private key with all the information above. The administrator is defined

as the person holding the private key used to deploy the contract initially. Of course the administrator must have enough ether/test-ether to pay the network fee.

### **Signing of configuration**

To prevent a single entity (the administrator) controlling the contract, all the organizers have to sign the configuration by sending a transaction signed with their private key to the contract. While the configuration is not fully signed by all organizers, no key can be deposited, and no other function can be called.

### **Deposit of public keys of attendees**

The organizers then send a set of public keys to the contract using a transaction signed with their private key including the data they wish to push. The current key format supported for attendees is ed25519 to permit interoperability with the DEDIS implementation.

### **Consensus and lock**

Once at least one keyset is added, the administrator can call a consensus function that will lock the contract and will choose which keyset will be considered as the reference/final keyset.

#### **0.4.4 How to interact with an Ethereum smartcontract**

A deployed smart contract is defined by its address and its ABI (Application Binary Interface). You can get the ABI of a contract using the Solidity compiler.

### **Creating and sending transactions**

A transaction is a call to the EVM to interact with a contract or an account. Technically a transaction should be from human to human and a message is human to contract or contract to contract. For clarity purposes we will not do the distinction and only use the term transaction. A transaction is constituted of the following fields :

- recipient address (contract or human)
- signature identifying the sender
- value field for transfer of ether. Here this field will always be ignored.
- gas limit, the maximum fee the sender is ready to pay
- gas price, decided by miners
- nonce, the number of transactions made by the account to avoid replay attacks

## 0.5. THEORETICAL AND PRACTICAL LIMITATIONS OF THE IMPLEMENTATION.7

- data field, for a contract this will be the compiled bytecode, for a contract function call, it will be the arguments of the function.

It's possible to create raw transactions through a node.js console attached to the network.

### Other ways

Many tools exist to interact with the ethereum blockchain, we will cite the most popular :

- Mist browser, official application that serves both as a wallet and a way to deploy smart-contract and interact with them. You will need to run a full node (download the whole blockchain) to use it or connect to another node.
- Myetherwallet.com website that serves as a wallet and way to interact with smart contract. The website is connected to a node and send transactions created on the website to the network.

### 0.4.5 Interoperability with current system

The current pop implementation at DEDIS uses a command line interface (CLI) to deploy and use the pop tokens. The goal was to keep the same commands if applicable and create new ones for new functions. The approach taken was to convert the popcontract into a library of go lang functions and then integrate those into the CLI app, thus making it possible to interact with the contract using go lang.

The conversion was done using a geth tool. Geth is a command line interface for running a full ethereum node implemented in go. The command take as input a solidity file and returns a new go file.

## 0.5 Theoretical and practical limitations of the implementation.

### 0.5.1 Theoretical limitations

#### Scalability

It might be possible to create large pseudonym parties but after a critical point it becomes highly unpractical for both attendees and organizers to register every member. This might be avoided by doing multiple different parties, which brings new issues. First off, the parties must held at the same time and sufficiently appart from each other to avoid attendees travelling and depositing multiple keys. Then arise a new issue, how to assure the legitimacy of each organizers of each party? How to assure that attendees don't pay others to use their tokens? How to avoid creation of fake parties?

### **0.5.2 Practical limitations**

## **0.6 Results**

### **0.6.1 Time to deploy**

The current contract takes on average one minute to deploy. However users don't need to wait until the contract is fully mined to interact with it. During tests, it was shown that sending all transactions with less than one second between them and in order, didn't caused it to malfunction. Each one was mined and activated correctly the contract afterwards.

Therefore it is possible to create fully fonctionnal PoP tokens in less than five minutes.

### **0.6.2 Price in ether**

The current price to deploy the contract is 0.12 eth which is at the current market price 84 USD (coinmarketcap.com the 16/12/2017). It also cost around 14 USD (0.02 eth) to call the heavier functions (configuration setting and key deposit). The price is arguably quite high. However, since the contract doesn't need value exchange between participants, it can be deployed and used on the testnet for 0 USD.

### **0.6.3 Attack vectors**

## **0.7 Possible improvements**

### **0.7.1 Create a friendlier interface**

One of the biggest issues with the current scheme is, its difficulty to use for non technical litterate users. It might be interesting to make an app/web-app symplifying the process for attendees and organizers. This could help micro-communities create their tokens with more ease.

### **0.7.2 Improve consensus function and ways to reach it**

The system could be optimized by adding an additionnal parameter in the contract to choose the way the final keyset is choosen. For the moment the contract only returns the last keyset added. There could be different ways to select the good keyset. One example could be voting, by organizers and/or attendees. Another would be to select the keyset with the least/more keys, or even creating a new keyset with all the different keys added. Optimally the organizers could sign beforehand the governance system. One of the issues concerning such functions is that computation on the EVM is very expensive and processing the keys is a very heavy process, particurlaly when the keyset becomes large.



### 0.7.3 Size optimization

If the entity wishing to verify the personhood of participants possess the set of keys, it might be possible to make the contract less expensive to deploy/ more efficient by only pushing hash of the data on the blockchain. For example for the function deposit key, the organizers could push only hashes of the keyset and the administrator could push only a hash of the configuration. It is not clear if the efficiency gain is worthwhile compared to the simplicity loss.

### 0.7.4 Technical improvements

It could be useful to add a way to choose the key format for attendees. For example adding an ethereum key format choice in the configuration of the contract, this would help attendees use the ethereum tools.

### 0.7.5 Decrease input parameters

For the moment the manual input of the nonce is not very practical. Moreover the gas price and gas limit are hardcoded into the app. Adding a function that would fetch each of those parameters would greatly increase practical usability.

### 0.7.6 Better way to handle private keys

The private key used by the administrator and the organizers is entered in clear and then stocked inside a .conf file. This is not very secure. It would be nice to have some more elegant solution.

## 0.8 How to create PoP tokens using the contract

### 0.8.1 Tools needed to deploy

#### Geth

Golang-ethereum, geth, is a command line interface to run a full ethereum node. The installation process can be done using the links provided in the source.

#### Golang

Golang is a compiled language created by Google. A link is given in the sources for installation.

[illegible]

## 0.8. HOW TO CREATE POP TOKENS USING THE CONTRACT 11

You should normally be able to see in your geth console :

As soon as the transaction is mined, your contract is deployed. You can verify it by checking the contract public address in a block explorer, or using the hash of the transaction. Here we can see that the above contract deployment was successful in etherscan.io :

[illegible]

Set the configuration of the contract

To set the configuration of the contract :

```
./popcontract org config "desc.toml"
```

Example of desc.toml file :

```

Name = "First party"
Location = "Nazareth"
NumberOfOrganizers = 1
OrganizersAddresses = ["0x286485b3026d5d817f1f444060516b439b13dd2b"]
Deadline = 60

```

Note : here the organizers and the administrator are the same person, so the contract is controlled by a single entity.

The console should display the hash of the transaction. Once again you can verify online that the transaction was successful.

### 0.8.3 How to interact with contract

Each organizers can save their own configuration(privatekey, path, nonce) using :

```
./popcontract org LinkOrganizers "your private key" "your geth path" "
```

To sign the configuration the organizers should run :

```
./popcontract org sign
```

The administrator can then start the key deposit by signing the whole configuration:

```
./popcontract org signAdmin
```

This step verify that all organizers signed the configuration.

To deposit new key set :

```
./popcontract org public "private key" "keyset"
```

Note that if a non organizer tries to invoke the function, the transaction will fail.

To reach consensus the administrator calls the following function :

```
./popcontract org final
```

## 0.9 What to do

Introduction of the subject and the problem. Goals of the project and motivation. Corresponding background. Step-by-step description of your design/implementation/evaluation/analysis. Theoretical and practical limitations of the project and its implementation. Results. Evaluation of the results and comparison with other approaches if applicable. Future work. Step-by-step guide for installation, including dependencies, and running of the final product.

# Contents

0.1	Introduction . . . . .	2
0.2	Proof of Personhood tokens . . . . .	2
0.2.1	What are Proof of Personhood tokens? . . . . .	2
0.2.2	Pseudonym parties . . . . .	2
0.2.3	Current implementation . . . . .	2
0.2.4	Limitations . . . . .	3
0.3	Ethereum . . . . .	3
0.3.1	Ethereum, Solidity and smart contracts . . . . .	3
0.3.2	Why use Ethereum? . . . . .	3
0.3.3	A quick note on testnets . . . . .	3
0.4	Implementation of Popcontract . . . . .	4
0.4.1	Abstract . . . . .	4
0.4.2	Popcontract finite state machine . . . . .	4
0.4.3	Technical implementation . . . . .	5
0.4.4	How to interact with an Ethereum smartcontract . . . . .	6
0.4.5	Interoperability with current system . . . . .	7
0.5	Theoretical and practical limitations of the implementation. . . . .	7
0.5.1	Theoretical limitations . . . . .	7
0.5.2	Practical limitations . . . . .	8
0.6	Results . . . . .	8
0.6.1	Time to deploy . . . . .	8
0.6.2	Price in ether . . . . .	8
0.6.3	Attack vectors . . . . .	8
0.7	Possible improvements . . . . .	8
0.7.1	Create a friendlier interface . . . . .	8
0.7.2	Improve consensus function and ways to reach it . . . . .	8
0.7.3	Size optimization . . . . .	9
0.7.4	Technical improvements . . . . .	9
0.7.5	Decrease input parameters . . . . .	9
0.7.6	Better way to handle private keys . . . . .	9
0.8	How to create PoP tokens using the contract . . . . .	9
0.8.1	Tools needed to deploy . . . . .	9
0.8.2	How to deploy a pop contract . . . . .	10
0.8.3	How to interact with contract . . . . .	12

0.9 What to do . . . . .	12
--------------------------	----