ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

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 responsability and one implementation of it : Proof of Persoonhood 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 achievied 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 attendees 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 fimed, 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

For 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 is 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 intereact 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 intereact 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 minors 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.

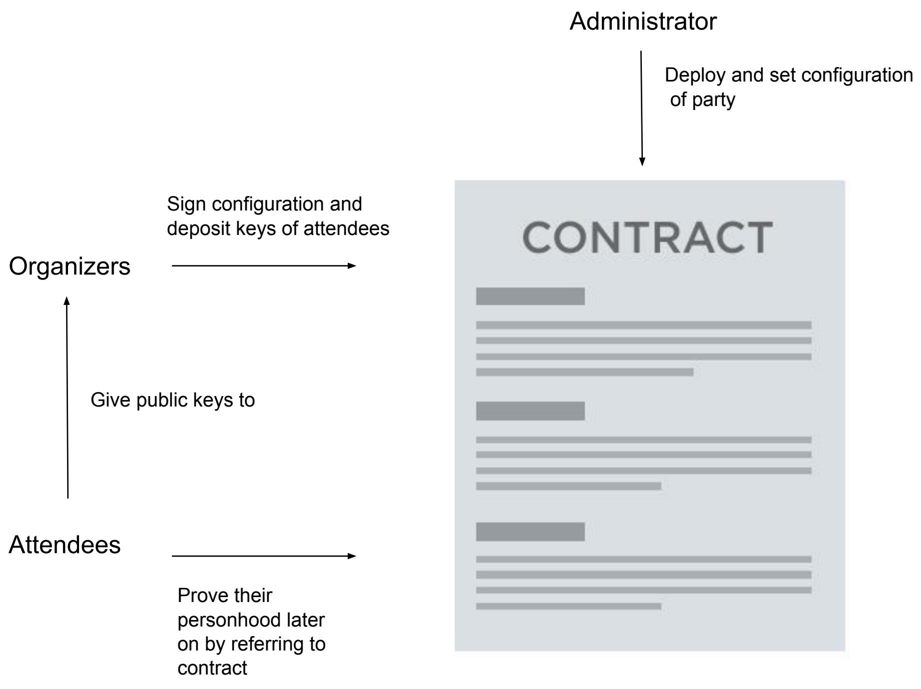

Administrator
Deploy and set configuration of party
Sign configuration and deposit keys of attendees
Organizers
CONTRACT
Give public keys to
Attendees
Prove their personhood later on by referring to contract

Figure 1: How actors intereact 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 state :
- initial state (contract was succefully 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)

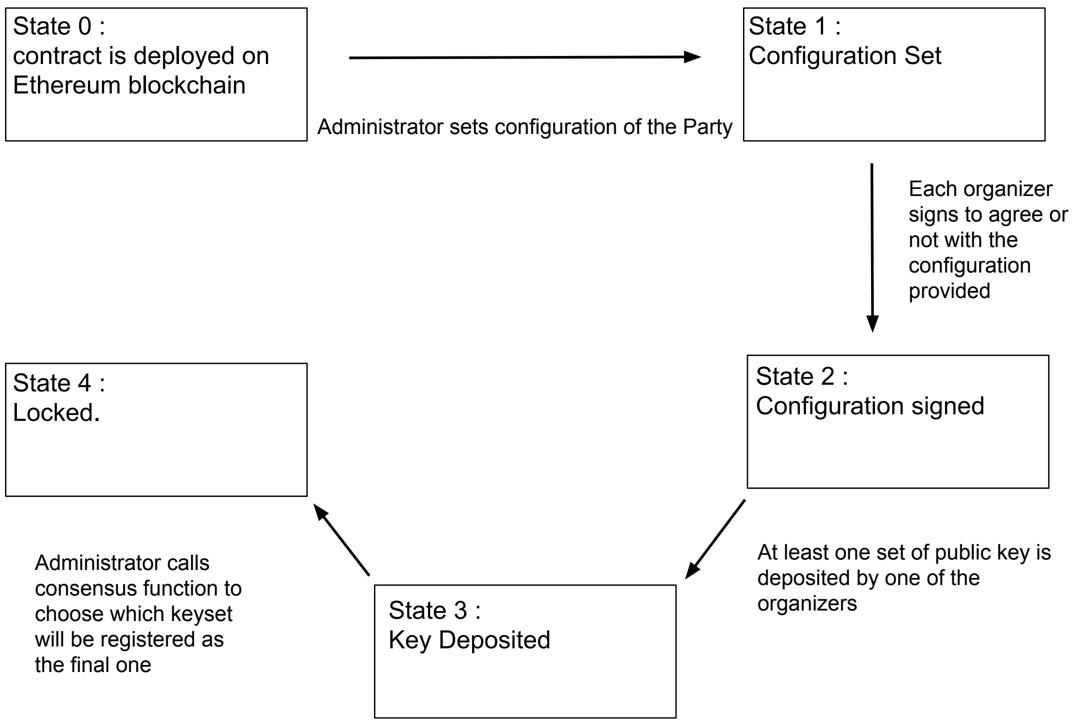

State 0 :
contract is deployed on Ethereum blockchain
Administrator sets configuration of the Party
State 1 :
Configuration Set
Each organizer signs to agree or not with the configuration provided
State 2 :
Configuration signed
At least one set of public key is deposited by one of the organizers
State 3 :
Key Deposited
Administrator calls consensus function to choose which keyset will be registered as the final one
State 4 :
Locked.

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 keys can be deposited, and no other functions can be called.

Deposit of public keys of attendees

The organizers then send a set of public key 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 it's address and it's 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 intereact 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 constitued 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

- 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 intereact 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 intereact 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 intereact 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 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 golang functions and then integrate those into the CLI app, thus making it possible to intereact with the contract using golang.
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

This scheme of authentication has many issues, particurlarly in its implementation on a large scale

0.5.2 Practical limitations

0.6 Results

0.7 Possible improvements

0.7.1 Create a friendlier interface

One of the biggest issue 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 choosed. 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 issue 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

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

If 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.

0.8.2 How to deploy a pop contract

Synchronize with the ethereum blockchain

First you have to run a full node. Depending on the network this will use between 20 and 60 Gb of hardrive space. To synchronize with rinkeby, run :

```
geth --rinkeby
```

Save the path to your geth.ipc file. Depending on your connection this will take a little while, so run it the first time during the night.

Clone the project directory

Clone the project directory using the github link in the sources.

Create an ethereum account

Create or use an already existing account. You will need your account ethereum private key and the nonce associated. As mentionned earlier, you can find your nonce by inputing your public address in a block explorer and incrementing the nonce of the last successful transaction. You can also finding it directly by running :

```
geth attach "your geth.ipc path"
```

In the console type :

```
eth.web3.getTransactionsCount("public address")
```

Deploy contract

Go in the cloned directory and run

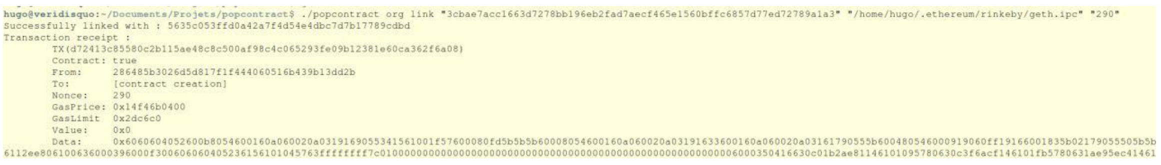
```
go build
```

This step should not be necessary, but will verify that your go installation and project import went well.

To deploy the contract run :

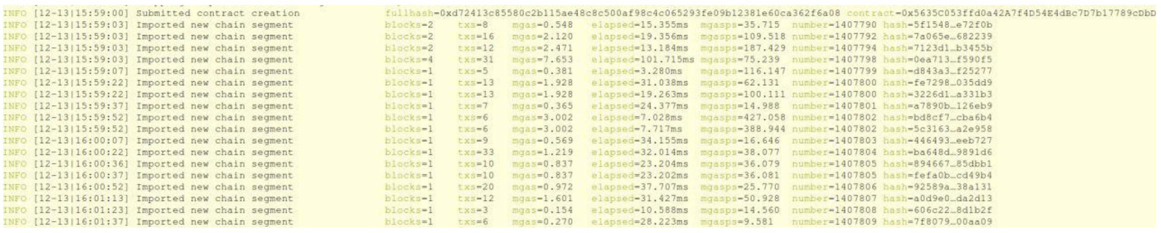
```
./popcontract org link "your private key" "your geth path" "nonce"
```

These parameters will be saved in a configuration file. The console should show something similar to that :



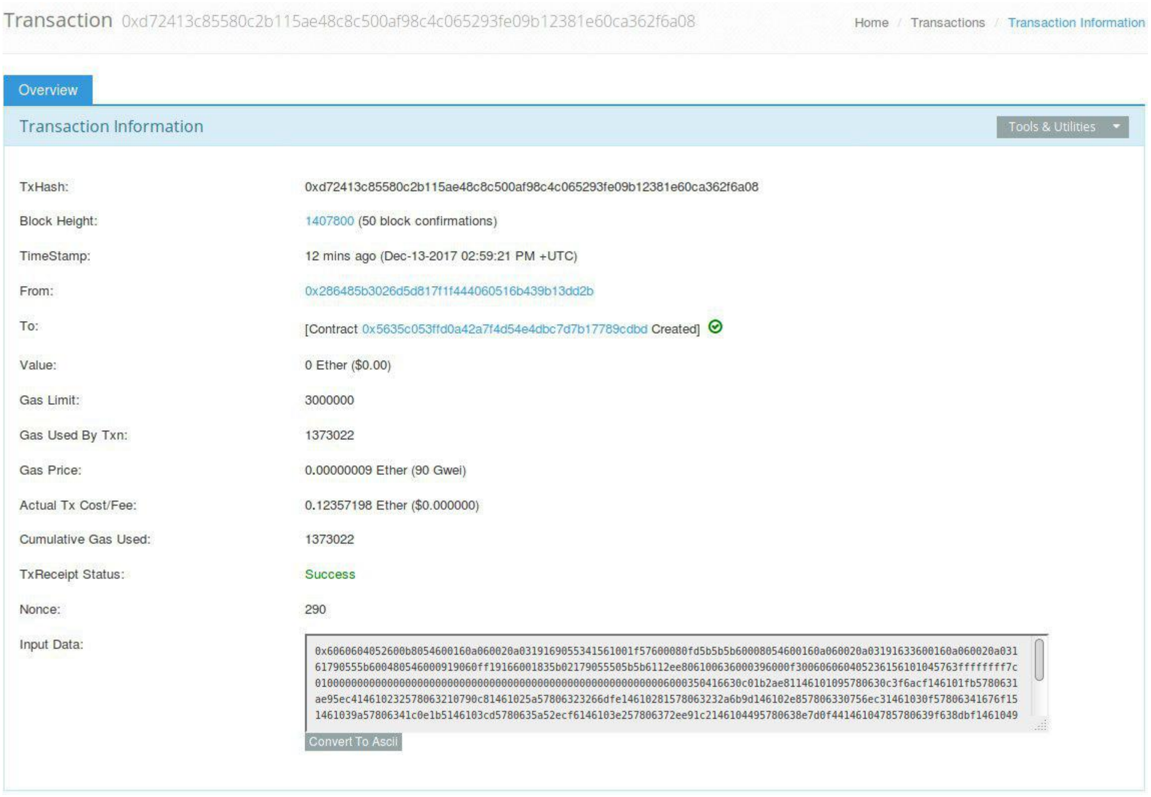
```
hugo@veridisquo:~/Documents/Projets/popcontract$ ./popcontract org link "3cbae7acc1663d7278bb196eb2fad7aecf465e1560bffc6857d77ed72789a1a3" "/home/hugo/.ethereum/rinkeby/geth.ipc" "290"
Successfully linked with : 5635c053ffd0a42a7f4d54e4dbc7d7b17789cdbd
Transaction receipt :
        TX(d72413c85580c2b115ae48c8c500af98c4c065293fe09b12381e60ca362f6a08)
        Contract: true
        From:     286485b3026d5d817f1f444060516b439b13dd2b
        To:       [contract creation]
        Nonce:    290
        GasPrice: 0x14f46b0400
        GasLimit  0x2dc6c0
        Value:    0x0
        Data:     0x6060604052600b8054600160a060020a0319169055341561001f57600080fd5b5b5b60008054600160a060020a03191633600160a060020a03161790555b600480546000919060ff19166001835b02179055505b5b
6112ee806100636000396000f300606060405236156101045763ffffffff7c010000000000000000000000000000000000000000000000000000000060003504166630c01b2ae81146101095780630c3f6acf146101fb5780631ae95ec41461
```

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



```
INFO [12-13|15:59:00] Submitted contract creation              fullhash=0xd72413c85580c2b115ae48c8c500af98c4c065293fe09b12381e60ca362f6a08 contract=0x5635C053ffd0a42A7f4D54E4dBc7D7b17789cDbD
INFO [12-13|15:59:03] Imported new chain segment               blocks=2  txs=8   mgas=0.548  elapsed=15.355ms  mgasps=35.715  number=1407790 hash=5f1548…e72f0b
INFO [12-13|15:59:03] Imported new chain segment               blocks=2  txs=16  mgas=2.120  elapsed=19.356ms  mgasps=109.518 number=1407792 hash=7a065e…682239
INFO [12-13|15:59:03] Imported new chain segment               blocks=2  txs=12  mgas=2.471  elapsed=13.184ms  mgasps=187.429 number=1407794 hash=7123d1…b3455b
INFO [12-13|15:59:03] Imported new chain segment               blocks=4  txs=31  mgas=7.653  elapsed=101.715ms mgasps=75.239  number=1407798 hash=0ea713…f590f5
INFO [12-13|15:59:07] Imported new chain segment               blocks=1  txs=5   mgas=0.381  elapsed=3.280ms   mgasps=116.147 number=1407799 hash=d843a3…f25277
INFO [12-13|15:59:22] Imported new chain segment               blocks=1  txs=13  mgas=1.928  elapsed=31.038ms  mgasps=62.131  number=1407800 hash=fe7298…035dd9
INFO [12-13|15:59:22] Imported new chain segment               blocks=1  txs=13  mgas=1.928  elapsed=19.263ms  mgasps=100.111 number=1407800 hash=3226d1…a331b3
INFO [12-13|15:59:37] Imported new chain segment               blocks=1  txs=7   mgas=0.365  elapsed=24.377ms  mgasps=14.988  number=1407801 hash=a7890b…126eb9
INFO [12-13|15:59:52] Imported new chain segment               blocks=1  txs=6   mgas=3.002  elapsed=7.028ms   mgasps=427.058 number=1407802 hash=bd8cf7…cba6b4
INFO [12-13|15:59:52] Imported new chain segment               blocks=1  txs=6   mgas=3.002  elapsed=7.717ms   mgasps=388.944 number=1407802 hash=5c3163…a2e958
INFO [12-13|16:00:07] Imported new chain segment               blocks=1  txs=9   mgas=0.569  elapsed=34.155ms  mgasps=16.646  number=1407803 hash=446493…eeb727
INFO [12-13|16:00:22] Imported new chain segment               blocks=1  txs=33  mgas=1.219  elapsed=32.014ms  mgasps=38.077  number=1407804 hash=ba648d…9891d6
INFO [12-13|16:00:36] Imported new chain segment               blocks=1  txs=10  mgas=0.837  elapsed=23.204ms  mgasps=36.079  number=1407805 hash=894667…85dbb1
INFO [12-13|16:00:37] Imported new chain segment               blocks=1  txs=10  mgas=0.837  elapsed=23.202ms  mgasps=36.081  number=1407805 hash=fefa0b…cd49b4
INFO [12-13|16:00:52] Imported new chain segment               blocks=1  txs=20  mgas=0.972  elapsed=37.707ms  mgasps=25.770  number=1407806 hash=92589a…38a131
INFO [12-13|16:01:13] Imported new chain segment               blocks=1  txs=12  mgas=1.601  elapsed=31.427ms  mgasps=50.928  number=1407807 hash=a0d9e0…da2d13
INFO [12-13|16:01:23] Imported new chain segment               blocks=1  txs=3   mgas=0.154  elapsed=10.588ms  mgasps=14.560  number=1407808 hash=606c22…8d1b2f
INFO [12-13|16:01:37] Imported new chain segment               blocks=1  txs=6   mgas=0.270  elapsed=28.223ms  mgasps=9.581   number=1407809 hash=7f8079…00aa09
```

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 succesful in etherscan.io :


Transaction 0xd72413c85580c2b115ae48c8c500af98c4c065293fe09b12381e60ca362f6a08

Home / Transactions / Transaction Information

Overview

Transaction Information

Tools & Utilities

TxHash:	0xd72413c85580c2b115ae48c8c500af98c4c065293fe09b12381e60ca362f6a08
Block Height:	1407800 (50 block confirmations)
TimeStamp:	12 mins ago (Dec-13-2017 02:59:21 PM +UTC)
From:	0x286485b3026d5d817f1f444060516b439b13dd2b
To:	[Contract 0x5635c053ffd0a42a7f4d54e4dbc7d7b17789cdbd Created]
Value:	0 Ether ($0.00)
Gas Limit:	3000000
Gas Used By Txn:	1373022
Gas Price:	0.00000009 Ether (90 Gwei)
Actual Tx Cost/Fee:	0.12357198 Ether ($0.000000)
Cumulative Gas Used:	1373022
TxReceipt Status:	Success
Nonce:	290
Input Data:	0x6060604052600b8054600160a060020a031916905534156100 1f57600080fd5b5b5b60008054600160a060020a03191633600160a060020a031 61790555b600480546000919060ff19166001835b02179055505b5b6112ee806100636000396000f300606060405236156101045763ffffffff7c 0100000000000000000000000000000000000000000000000000000000600035041663 0c01b2ae81146101095780630c3f6acf146101fb5780631 ae95ec41461023257806321079 0c81461025a57806323266dfe14610281578063232a6b9d146102e857806330756ec31461030f57806341676f15 1461039a57806341c0e1b5146103cd5780635a52ecf6146103e257806372ee91c2146104495780638e7d0f441461047857806 39f638dbf1461049

Convert To Ascii

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 succesful.

0.8.3 How to intereact 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
```

Note that the transaction will fail if there is a missing signature from one of the organizers.

To deposit new key set :

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

Note that if a non organizers 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 dependecies, and running of the final product.

Contents