Blockchain
Technical Tutorial

« *Write and deploy my first Dapp* »

Oleg Lodygensky - LAL - Juin 2018

https://www.lal.in2p3.fr

http://www.u-psud.fr

http://www.cnrs.fr

1. **Introduction**
2. Concepts
3. Credentials
4. Solidity
5. Tools

Our first Dapp

We'll write a roulette wheel dApp where users can bet.

Requirements:

- the wheel has 36 numbers
- the wheel runs at fixed intervals
- bet on a single number can make 35 times the stake
- bet on odd or even number can make 2 times the stake; 0 is not allowed

The smart contract must be able to:

- run the wheel at given interval
- receive bets
- send profits to winners

Acknowlegements

- The example provided by Ethereum France contains bugs, but is well documented:

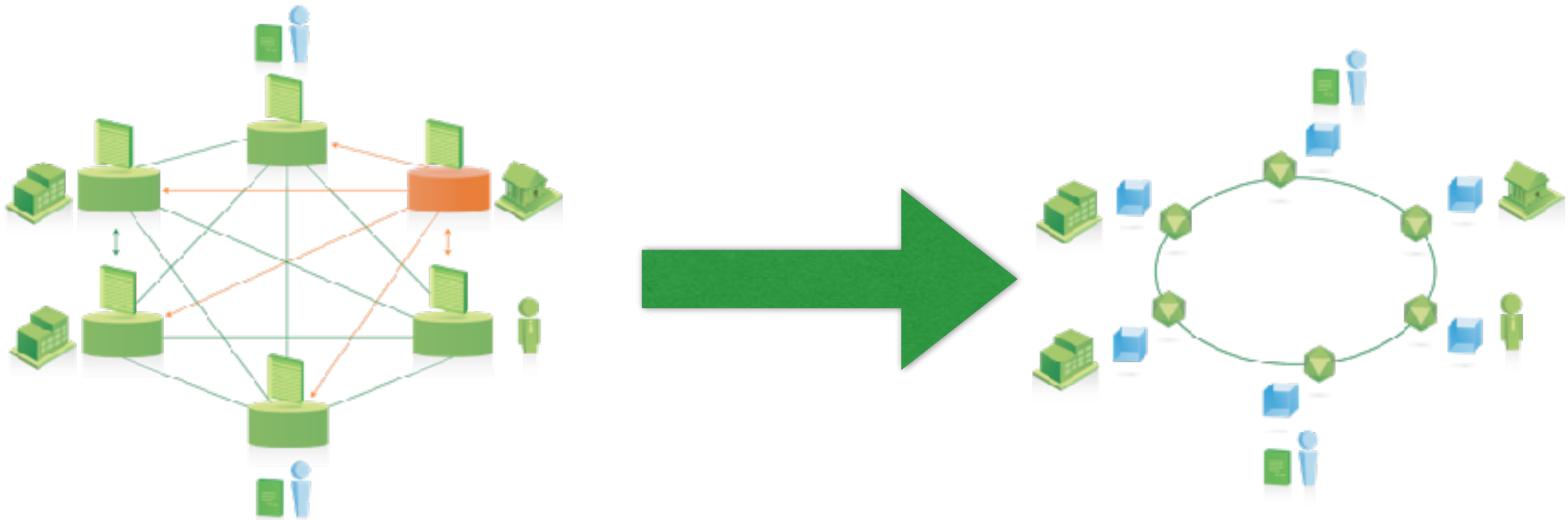
  https://www.ethereum-france.com/ecrire-une-dapp-pour-ethereum-1-smart-contract/

- Corrected contract available at
  https://gitlab.in2p3.fr/lodygens/blockchain-tutorial

- An image available on LAL Cloud: « blockchain »
  - Domain « stratuslab »
  - Project « stages »

1. Introduction
2. **Concepts**
3. Credentials
4. Solidity
5. Tools

Blockchain : une décentralisation totale



« *Degoogliser* » !

- ✓ **Décentralisation** des services et des applications
- ✓ **Réappropriation** de ses données personnelles
- ✓ Tolérance aux **pannes**
- ✓ Résistance à la **censure**

Paradigmes

BD partagée et write once

Shared Ledger

Smart Contract

Transactions + Conditions de réalisation

Les transactions sont sûres, authentifiées et vérifiables

Privacy

Consensus

Tous les acteurs doivent accepter la transaction

Anonymity

Blockchain users are not **anonymous**
They are ***pseudonymous***

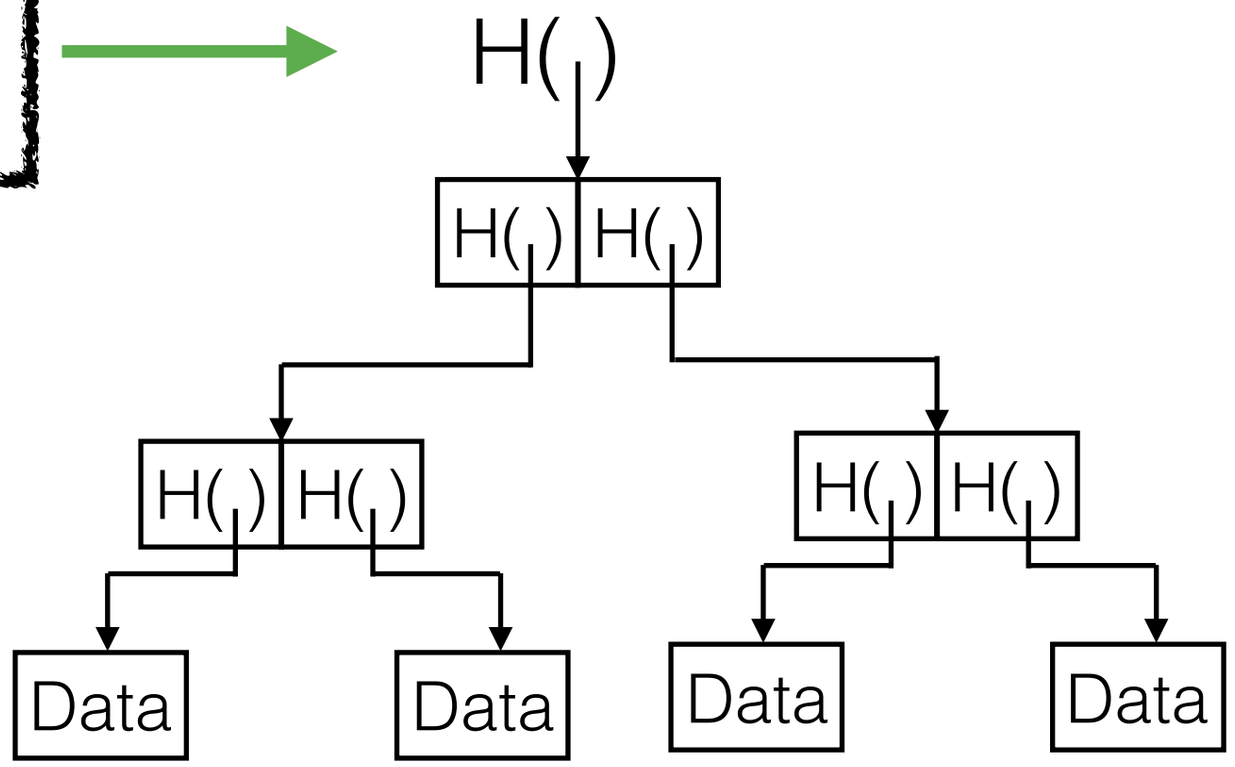
Anonymity and decentralisation are in conflict

Decentralization mainly achieved thanks to public traceability enforcing security

Merkel Tree

A Merkel Tree is a Binary Tree made with hash pointers.

Root Hash Pointer is enough to access all data

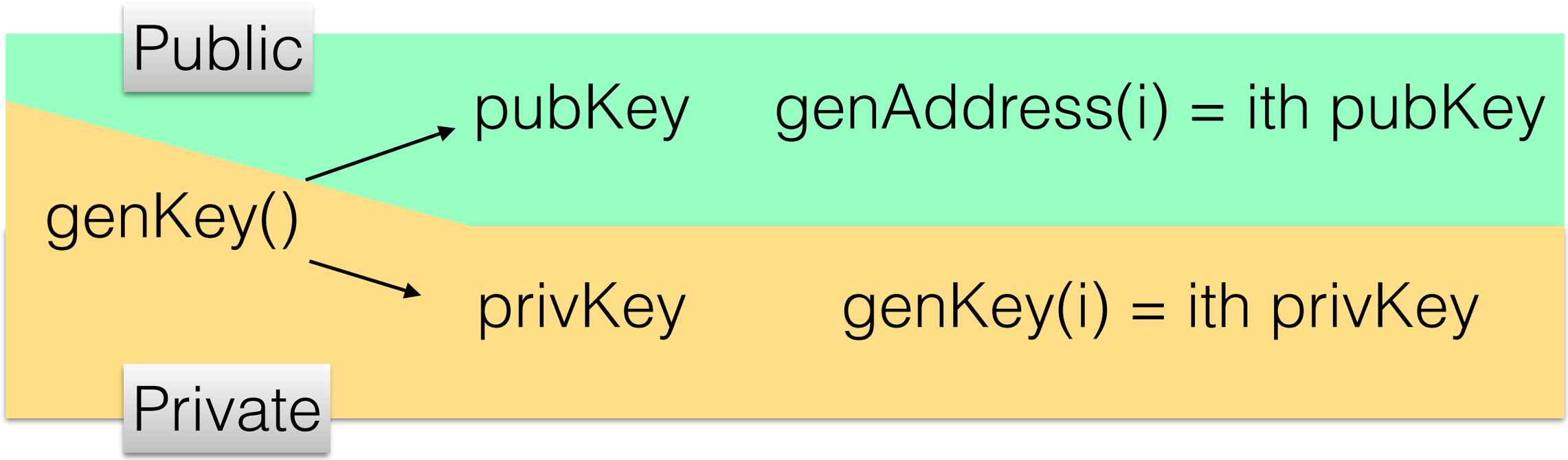


Membership is in $O(\log_n)$

Digital signature

ECDSA supports hierarchical key generation.

Public address and secret key can be generated independently of each other.



Coin

Coins are immutable:

- they can be created
- they must be digitally signed

They can’t be modified in any manner:

- no transfert
- no division
- no combination

Transactions

Transactions aim to:
- consume coins
- create new coins

Transaction is valid if:
- consumed coins
  - ✓ are valid
  - ✓ not already consumed
  - ✓ signed
- total value out = total value in

Validation is cryptographic only !

transID type=**payCoin**

consumed coins

92(5); 54(1) etc.

created coins

Num	Value	Recipient
0	…	…
1	…	…

coin signatures

sig[92(5)]; sig[54(1)] etc.

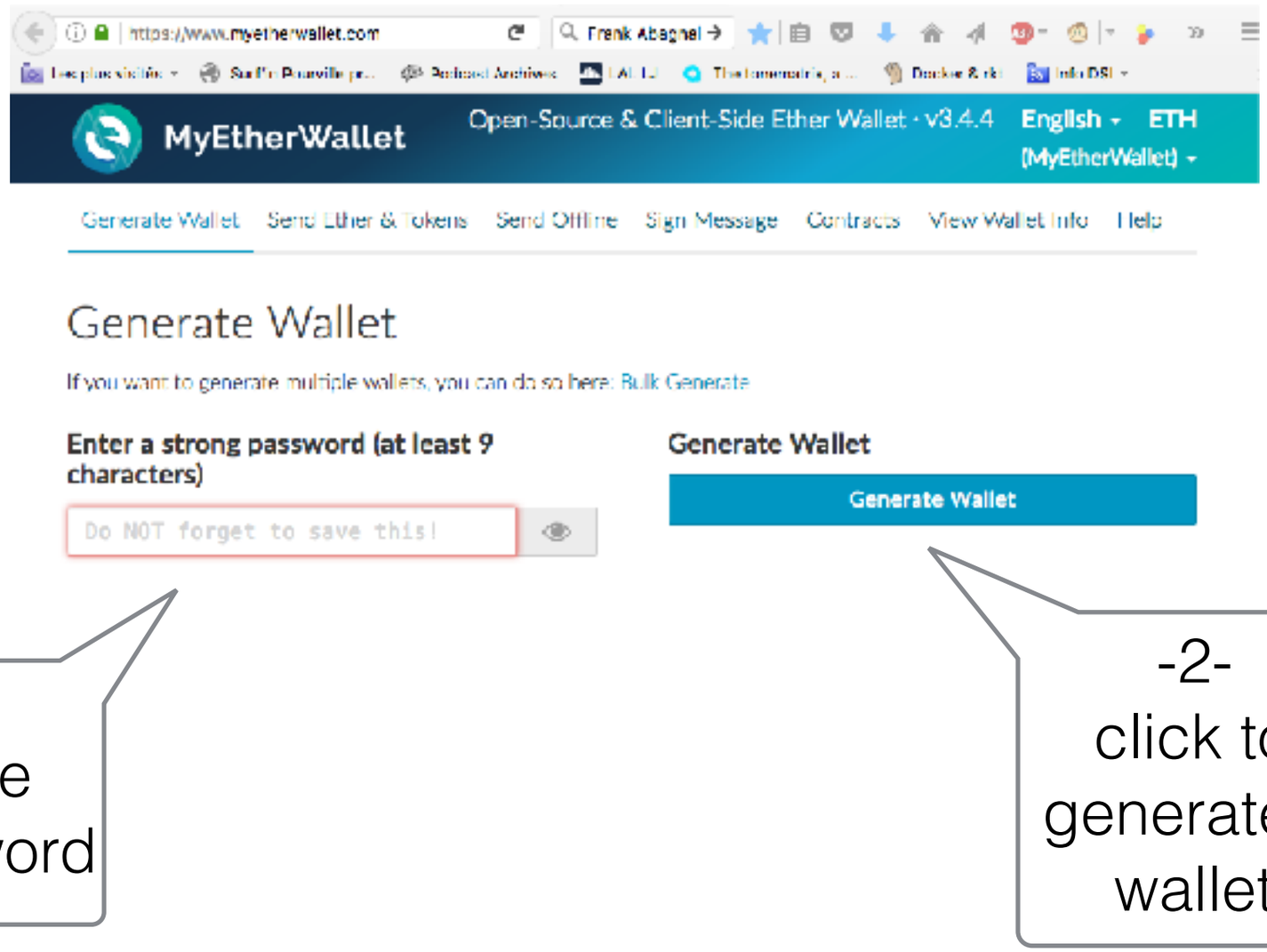
Transaction fees = input values - output values

1. Introduction
2. Concepts
3. **Credentials**
4. Solidity
5. Tools

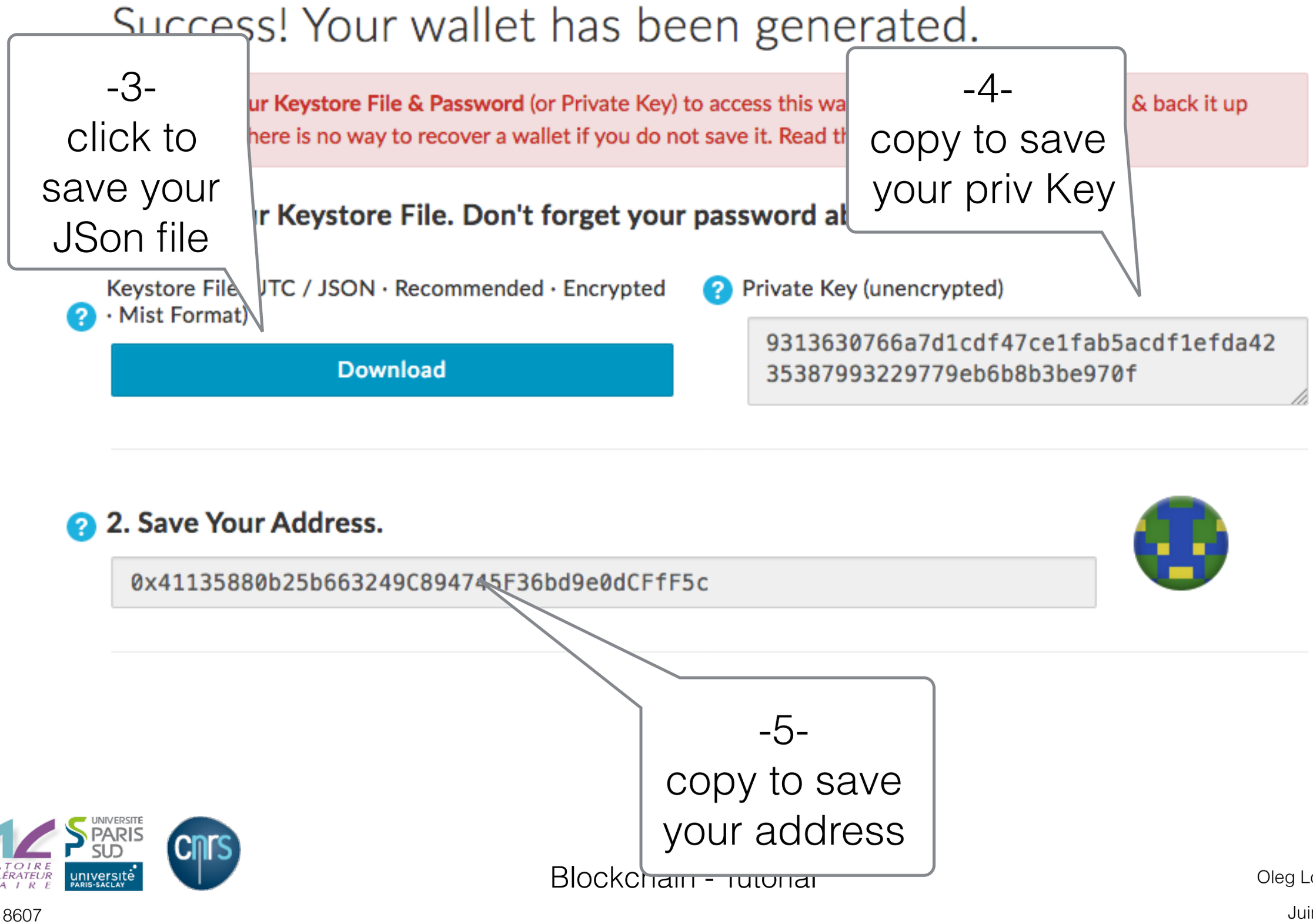
Credentials

We first create two testing users:

https://www.myetherwallet.com/

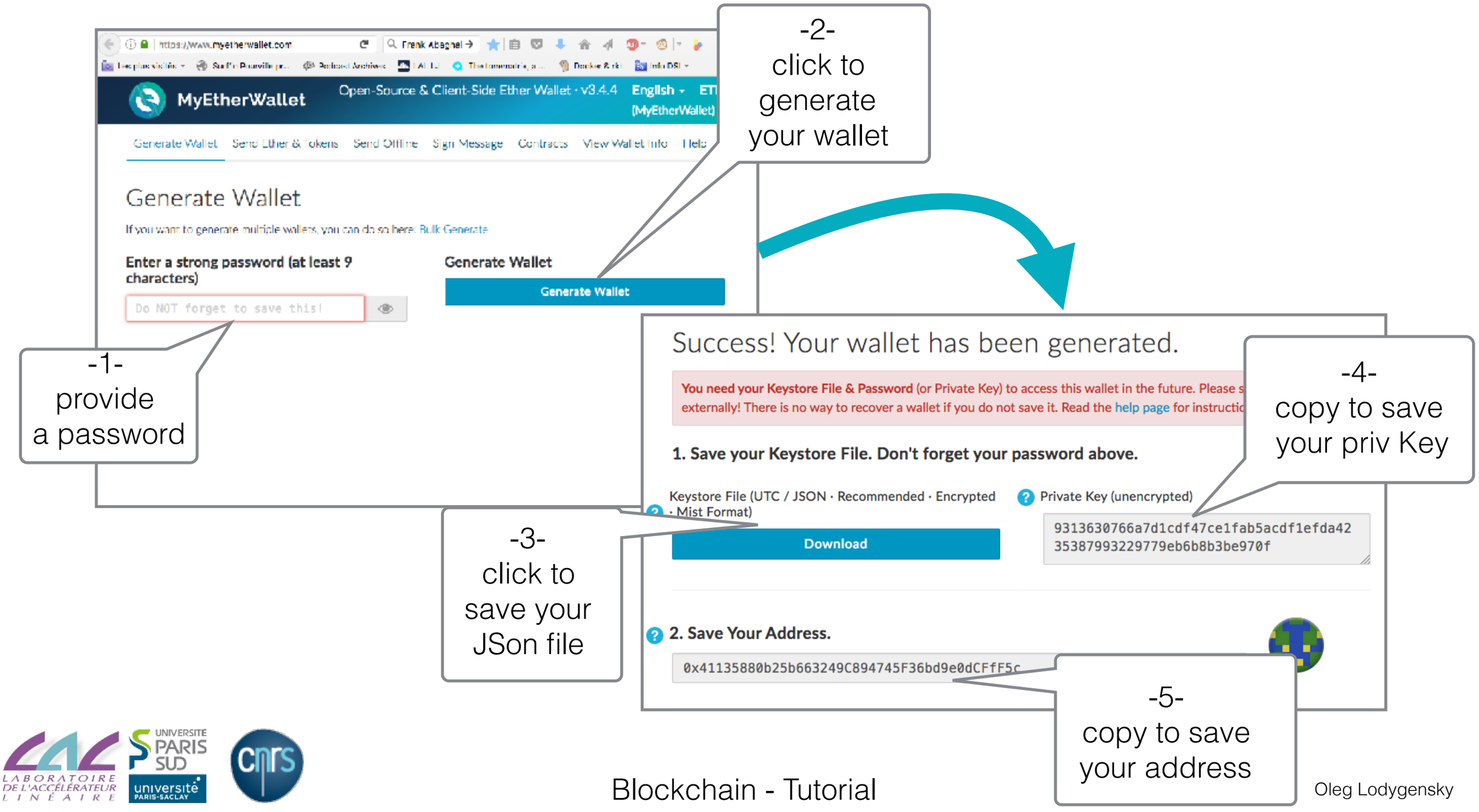


Credentials



Credentials

Again for the 2nd user



1. Introduction
2. Concepts
3. Credentials
4. **Solidity**
5. Tools

Variables

```
pragma solidity ^0.4.22;

contract Roulette {

    uint public lastRoundTimestamp;
    uint public nextRoundTimestamp;

    address public _bank; // creator address
    uint _interval;       // wheel interval

    enum BetType { Single, Odd, Even }

    struct Bet {
        BetType betType;
        address player;
        uint number;
        uint value;
    }

    Bet[] public bets;


}
```

Constructor

```
pragma solidity ^0.4.22;

contract Roulette {

 ...

   constructor(uint interval)  {
        _interval = interval;
        _bank = msg.sender;
        nextRoundTimestamp = now + _interval;
    }

 ...


}
```

Selfdestruct

Author should include a `kill()` method to be able to remove its contract from the blockchain

```
pragma solidity ^0.4.22;

contract Roulette {

 ...

    function kill() {
        if (msg.sender != _bank) revert();
        selfdestruct(_bank);
    }

 ...


}
```

Modifiers

```
pragma solidity ^0.4.22;

contract Roulette {

 ...

    modifier onlyOwner {
        if (msg.sender != _bank) revert();
        _;
    }

    modifier transactionMustContainEther() {
        if (msg.value == 0) revert();
        _;
    }

    modifier bankMustBeAbleToPayForBetType(BetType betType) {
        uint necessaryBalance = 0;
        for (uint i = 0; i < bets.length; i++) {
            necessaryBalance += getPayoutForType(bets[i].betType) * bets[i].value;
        }
        necessaryBalance += getPayoutForType(betType) * msg.value;
        if (necessaryBalance > _bank.balance) revert();
        _;
    }

 ...


}
```

Modifier is a « kind of » interface

Functions

Functions have no cost.

Views don't modify the state

```
ity ^0.4.22;

ette {


function getBetsCountAndValue() view returns(uint, uint) {
    uint value = 0;
    for (uint i = 0; i < bets.length; i++) {
        value += bets[i].value;
    }
    return (bets.length, value);
}
    }
    function getBankBalance() constant returns (uint) {
                return _bank.balance;
    }
    function getBank() constant returns (address) {
                return _bank;
    }
    function getPayoutForType(BetType betType) pure returns(uint) {
        if (betType == BetType.Single) return 35;
        if (betType == BetType.Even || betType == BetType.Odd) return 2;
        return 0;
    }

 ...


}
```

Pure functions don't even read the state.

Transactions

Transactions have a cost;
the are **payable**

```
pragma solidity ^0.4.22;

contract Roulette {

 ...

    function betSingle(uint number)
                payable
                transactionMustContainEther ()
                bankMustBeAbleToPayForBetType(BetType.Single) {

        if (number > 36) throw;

        bets.push(Bet({
            betType: BetType.Single,
            player: msg.sender,
            number: number,
            value: msg.value
        }));
        BetDone(BetType.Single, number, msg.value);
    }

 ...

}
```

msg.value is the price to pay for this transaction

First dApp

Summary

1. Introduction
2. Concepts
3. Solidity
4. Credentials
5. **Tools**

Tools

Tool	Description	Installation
Chrome	Google browser: https://www.google.com/chrome	Please follow instructions on web site
MetaMask	Chrome plugin Ethereum client: https://metamask.io	
Node.js	JavaScript Runtime: https://nodejs.org	
Npm	JavaScript package manager: https://github.com/npm/npm	
Web3.js	https://github.com/ethereum/web3.js	npm install -g web3
TestRpc	Node.js based Ethereum client for testing and development:	*npm install -g ethereumjs-testrpc*
Truffle	Ethereum development framework: http://truffleframework.com	*npm install -g truffle*
Solidity	Smart contract language: http://solidity.readthedocs.io	Nothing to be done
My Ether Wallet	Ethereum network client: https://www.myetherwallet.com/	

Docker image
zenika/truffle-with-testrpcdocker
contains everything

TestRPC

Launch a network and provide 10,000*Ether*
to our two users (1 *Ether* = 10E+18*weis*)

```
$> cd /root/blockchain-tutorial
$> testrpc --account=\
"0x31d7d10277e46af1c1b2ad604a642c4339a0ccd80157e630ec510d6b2c63b693, \
 100000000000000000000" \
          --account= \
"0xf4cfcd38ad4c221007e76fbe736c0853b74d959431c876b8970a1b89fcad168b,
 100000000000000000000"


Available Accounts
==================
…

Private Keys
==================
…

Listening on localhost:8545
```

Weis (crypto currency)

Priv Key

Node

You can check your network using **node.js**

```
$> npm install web3
$> node
> var Web3 = require('web3');
undefined
> var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
undefined

> web3.eth.accounts
[ '0xed1165286ad96f4d05bba688370896389d703439',
  '0xd81cf4ae3ddd9cf58e46fa8930ccbe2f2e7298ee' ]


> web3.fromWei(web3.eth.getBalance('0xed1165286ad96f4d05bba688370896389d703439'),
'ether')
{ [String: '10000'] s: 1, e: 4, c: [ 10000 ] }
```

testrpc must be running

Accounts

Ether (crypto currency)

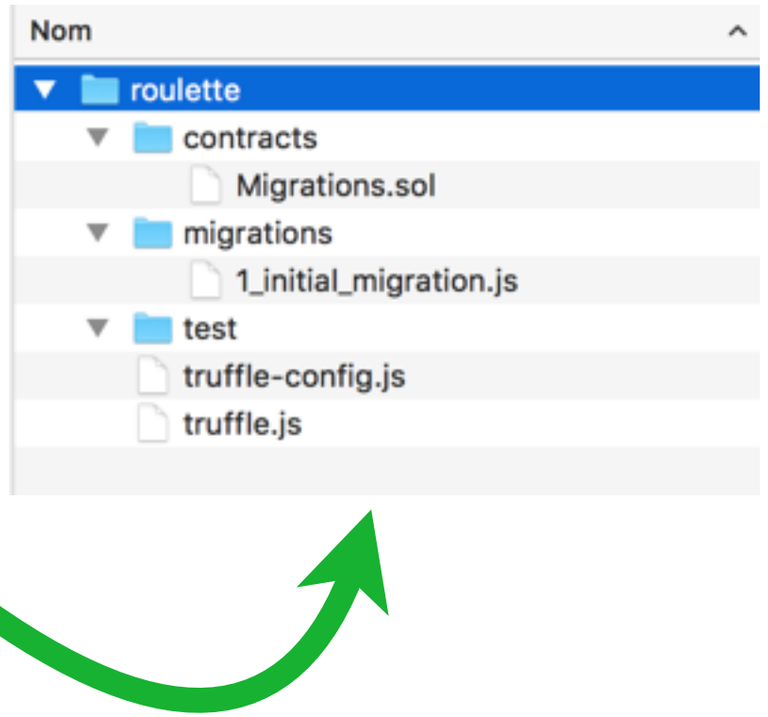
Truffle

Init

Truffle aims to compile and deploy (*migrate*) Smart contracts.

Initialize a new truffle project

```
$> mkdir roulette && cd $_ && truffle init
```



This creates a new file structure

- contracts : SmartContract codes
- migrations : migration scripts
- tests : test scripts

Truffle

You must write your Smart Contract:

- Solidity code : contracts/Roulette.sol
- Migration script : migrations/2_deploy_roulette.js

Available from
https://gitlab.in2p3.fr/lodygens/blockchain-tutorial

Truffle

Compile & Migrate

```
$> cd /root/blockchain-tutorial
$> truffle compile

$> truffle build

$> truffle migrate [--reset]
 Running migration: 1_initial_migration.js
  Deploying Migrations...
Saving successful migration to network...
Saving artifacts...
Running migration: 2_deploy_roulette2.js
  Deploying Roulette...
Saving successful migration to network...
Saving artifacts...

$> truffle serve
Serving app on port 8080...
Rebuilding...
Completed without errors on Fri Jan 13 2017 16:34:41 GMT+0100 (CET)
```

testrpc must be running

Truffle

Console

```
$> truffle console
truffle(development)> var contractFromRouletteDeployed = Roulette2.deployed();
undefined
truffle(development)> contractFromRouletteDeployed.then(function(instance) {return instance.betSingle(5,
{ from: web3.eth.accounts[0], value: 4})});
'0x029c47752f033dba08b6003927dbd062f66134caf7e4c06a1cb99780e29e6f1a'

truffle(development)> contractFromRouletteDeployed.then(function(instance) {return
instance.getBetsCountAndValue()});
[ { [String: '1'] s: 1, e: 0, c: [ 1 ] },
  { [String: '4'] s: 1, e: 0, c: [ 4 ] } ]

truffle(default)> contractFromRouletteDeployed.then(function(instance) {return instance.betSingle(5,
{ from: web3.eth.accounts[0], value: 4})});
'0x13265bc324030a6e4a59f1e5ec90f2f97db33adeac0d9fc03d0b79b0e882ba47'

truffle(development)> contractFromRouletteDeployed.then(function(instance) {return
instance.getBetsCountAndValue()});
[ { [String: '2'] s: 1, e: 0, c: [ 2 ] },
  { [String: '8'] s: 1, e: 0, c: [ 8 ] } ]

truffle(development)> contractFromRouletteDeployed.then(function(instance) {return instance.betSingle(5,
{ from: web3.eth.accounts[0], value: 4})});
‘0x0392bfc89899a4a0b0b41103e29096db452809589018b65727aee0edfa986b9f'

truffle(development)> contractFromRouletteDeployed.then(function(instance) {return
instance.getBetsCountAndValue()});
[ { [String: '3'] s: 1, e: 0, c: [ 3 ] },
  { [String: '12'] s: 1, e: 1, c: [ 12 ] } ]
```

References

- Ethereum France:
  - https://www.ethereum-france.com

- Mastering Bitcoin
  - https://bitcoin.fr/mastering-bitcoin-fr/

- Blockchain France
  - https://blockchainfrance.net/