Solution Design

If we draw attention to the code behind our idea we can divide it into two parts: backend (solidity) and frontend (html, javascript, metamask). The purpose of the frontend is to give a simple but hopefully not less secure interface to the user so he can easily access the blockchain. The backend consists of the wallet contract (managing the tokens) and an example implementation of how some of the transfers could work. It is still not fully realistic because the complexity would increase heavily if you make the model more realistic. The focus of the following chapter is on the backend part because the frontend is quite simple. Frontend is just about letting MetaMask call some prepared functions of certain contracts on the blockchain, and about listening to events of the contracts (and showing these in a stylish and simple way to the user).

We will start with the abstract model of what we tried to build. Then the main part is about the implementation (in solidity) and it's difficulties. The last part of the solution design will be about some problems with this implementation.

Idea

The main target of our implementation is, that it has to be secure. So we try to do it as simple as possible (then there are less sources of errors). Every user should be able to get a wallet where his tokens will be stored. Nobody else should have the right to take tokens out of that 'bag'. But there should be some automation for the transactions (e.g when a user produces for or buys energy from the connecting network at his home).

To create the connection between the real world and the blockchain we use the concept of SmartMeters. Any organization that got the trust of people in its environment could offer this service. SmartMeters should measure the produced and consumed energy of users and report it to the blockchain, where SmartContracts between the user and the connecting network will transfer tokens dependant on the individual implementation of these SmartContracts.

Our Model Implementation

Wallet

This contract is responsible for the ICO and it holds the balances of tokens. Therefore it should be really really simple. If there is a bug that can be exploited in this contract the whole concept couldn't work. We explain shortly the functions of this contract:

- When the wallet is constructed it will initialize the amount of coins on the market to 1e20. We didn't spend a lot research in how much this should be, so this could be changed. The owner is the creator of this contract. The owners address will get all the Ether that is given to this contract (the ether from the ICO).

- 'transfer' transfers an amount of tokens from the callers account to a destination account. It's quite simple. It only works if there are enough tokens on the callers account and if the transaction happened it will fire an event, there is no need for overflow check because there aren't enough tokens initialized to create an overflow. Frontend programms can listen on the fired events. In the

ethereum blockchain there is an extra section where fired events are listed, so everyone could check or listen to them.

- ‘buyTokens’ is the only payable function of this contract. It’s used for the ICO. For simplicity it just does a normal transfer of tokens from the wallets account to the callers account. The amount of tokens is equal to the amount of payed wei. Obviously the transaction gets cancelled if there aren’t enough tokens on the wallets account.

Model Scenario

The Wallet is the only part that will be the same for all transactions in different regions with different producers and consumers. The other three contracts just show a possible scenario how SmartContracts could do the transactions automatically.

SmartMeterBackend contains functions that can be called by the real (hardware) SmartMeter that measures electricity in and outflow of a producer/consumer. In our implementation it has just one function that reports energy production to the network operator.

The OperatorBackend will need some more explanation. There are functions (that are called by the owner, as example EWZ) that set the price for electricity from this network and the reward for feeding energy into it. Aside from that there is a list of trusted SmartMeterBackends. Only trusted SmartMeterBackends will be able to trigger a payout of tokens to the producer (they can call the rewardProduction function). Now, if a user wants to buy energy, he can call the function buyEnergy with a callback that sends tokens to to the operator if and only if the operator fires the event that reports the transaction as Event.

The User is an example contract of how a producer/consumer could implement his contracts. There is a function to ‘pay for energy’ and a function to transfer tokens to other users (as example if Bob wants to sell produced electricity to Alice). The energy supply will be done by the operators of the networks but if both operators (the one of Bob and the one of Alice) allow to pay for energy with tokens, it will be as if Bob sold his energy to Alice. The ‘pay for energy’ function calls the ‘buyEnergy’ function of the linked operator and sets up all required variables and functions for the transaction (it sends the callback function and sets the price, that the callback function will expect).

Problems and Possible Solutions

One problem could be the privacy. There may be users that don’t want, that everyone could know how much energy they consume and produce. This is a hard to solve problem because one of the main features of the blockchain is transparency. An approach for a solution could be, that the operator and user handle the transaction between them outside the blockchain (this requires trust from the user side). Still some transactions would be visible on the blockchain.

Another issue is, that in the implementation of the scenario (not in the implementation of the wallet) are still features, that could be exploited by the operator or the user. As example the operator can untrust a SmartMeter. Then the token transfer wouldn’t get triggered even if the consumer produces energy. This case isn’t that much a problem because all actions get logged in the blockchain and the trust in the operator would decrease much, if such things happen.

Another possible weakness of the system could be the SmartMeter itsself. A user could cheat if he manages to ‘hack’ the SmartMeter. A possible solution for this would be a SmartMeter that has some features that prevent hacking (as example it destroys its private key, when it notifies an attempt).