Seminar Thesis

A solidity smart contract for rental deposit accounts

Matthias Nadler

Supervised by:
Prof. Dr. Fabian Schär
Credit Suisse Asset Management (Schweiz) Professor for
Distributed Ledger Technologies and Fintech
Center for Innovative Finance, University of Basel

Abstract

TEMP: Analyze the incentive structure of the current rental deposit account situation and devise a DAI powered smart contract that will invest the locked funds.

Keywords: Ethereum, smart contract, DAI, rental deposit account.
JEL: G21, G52, G19

Contents

University of Basel

Center for
Innovative Finance

Plagiatserklärung

Ich bezeuge mit meiner Unterschrift, dass meine Angaben über die bei der Abfassung meiner Arbeit benutzten Hilfsmittel sowie über die mir zuteil gewordene Hilfe in jeder Hinsicht der Wahrheit entsprechen und vollständig sind. Ich habe das Merkblatt zu Plagiat und Betrug vom 22. Februar 2011 gelesen und bin mir der Konsequenzen eines solchen Handelns bewusst.

Matthias Nadler

1 Introduction

In Switzerland, according to the code of obligations (*dt. Obligationenrecht, OR*) Art. 257e para. 3, a rental deposit needs to be placed in a (savings-) account at a bank, or in a deposit on the tenants name. The interest rate for a rental deposit account at one of Switzerland's largest three banks is between 0% (UBS, Credit Suisse) and 0.05% (Raiffeisen).[1]

Setup, changes and release or restitution related to the deposit account all have to be handled in paper forms and require the signatures of both the tenant and the renter. These are time consuming processes for all involved parties.

The presented rental deposit account smart contract attempts to solve both problems, allowing for higher interest rates and for immediate, digital transactions between the parties. All while keeping the same level of security and improving transparency.

1.1 Smart Contracts

Smart contracts are similar to traditional contracts, but they are written in a computer language and deployed on a system that is accessible to all contract parties. When interacting with a smart contract program or protocol, the code is able verify your instructions and enforce the resulting actions.

There are a few issues and questions that arise with the implementation of smart contracts. Most of these can be elegantly solved by deploying the smart contract on a public blockchain[2] with a Turing complete language[3].

[1] As of March 8th 2020, according to the official rates published by the banks.

[2] It is assumed that the reader is familiar with blockchain and accompanying terminology. For an in-depth introduction to the topic the author recommends Berentsen and Schär (2017) (in German) or Lewis (2018) (in English).

[3] A touring complete programming language has all the necessary instructions to solve any computational problem given enough resources. Bitcoin for example does not have a touring complete language, therefore no universal smart contracts can be deployed on this blockchain.

Availability A public blockchain is always online and accessible from everywhere. Each smart contract is deployed at a fixed address on the blockchain.

Authentication Blockchains use private key cryptography to guarantee ownership of accounts (addresses) and do all of the heavy lifting. Smart contracts need no additional authentication logic.

Immutability The most difficult feature to replicate without a public blockchain. The code of a deployed smart contract can never be changed. This means no one has the ability to alter the contract retroactively. Often times this also removes the need for a third party trustee and makes completely trust-less interactions possible.

Power to enforce Smart contracts, like any other address, can directly control digital assets built on top of the blockchain. Any coin or token sent to a contract can not be recovered unless it is transferred by the contract logic.

Transparency The full contract code and every past interaction with the smart contract is stored on the blockchain and publicly available.

Our rental deposit account will combine a smart contract as described above with traditional off-chain contracts to create a more efficient agreement between the involved parties.

1.2 Ethereum

Any blockchain that implements a Turing complete programming language can be used to develop smart contracts. Among these options, Ethereum is the most popular according to an ecosystem report by Electric Capital (2019): Of all open source crypto developers, 18% worked in the Ethereum ecosystem during the first half of 2019. This is around four times more than for the second most popular platform EOS.

More reasons to develop this project on Ethereum include: The University of Basel teaches Solidity - the Ethereum specific programming language - in their courses. Ethereum has the widest array of decentralized finance products, including a very successful stablecoin which is the cornerstone of our application. Finally, the shortcomings of Ethereum compared to its alternatives - fewer transactions per second and higher transaction costs - are mitigated since our contract will perform very few transactions over its lifetime.

2 Analysis of current incentive structure

Analyze current deposit account contracts and work out incentives for each party.

2.1 Tenant incentives

Discuss tenant incentives and areas to improve situation.

2.2 Landlord incentives

Discuss landlord incentives and areas to improve situation.

2.3 Trustee incentives

Discuss trustee (bank, notary, depositary, ...) incentives and areas to improve situation.

3 Multisig on Ethereum

Describe the problem of multisig on ethereum. And explore possible implementations.

3.1 Off-Chain implementation

Sign transactions off-chain and then send the signed data to the contract.
SWOT analysis

3.2 On-Chain implementation

Multiple approaches exist

3.2.1 Transactions as struct

SWOT analysis, multiple transaction types (GNOSIS Ltd. (2019))

3.2.2 Minimal state approach

SWOT analysis, work with signatures

3.3 Conclusion

Present selected approach with additional details.

4 Designing the smart contract

Work out the functionality of the smart contract in pseudo code.

4.1 Multisig design

How we do multisig

4.2 Deposit account functionality

Functions related to the deposit account management

5 Implementation

Software development philosophy.

5.1 Environment

Tools used

5.2 Testing

Testing approach and coverage

6 Discussion

Discussion of result and potential applications

References

Berentsen, A. and Schär, F. (2017), *Bitcoin, Blockchain und Kryptoassets*, 1. edition edn.

Electric Capital (2019), 'Developer Report', (August), 104.

GNOSIS Ltd. (2019), 'gnosis/MultiSigWallet: Allows multiple parties to agree on transactions before execution.'.
URL: *https://github.com/gnosis/MultiSigWallet*

Lewis, A. (2018), *The basics of bitcoins and blockchains : an introduction to cryptocurrencies and the technology that powers them*, Mango Publishing.
URL: *https://xsava.xyz/ebooks/The_Basics_of_Bitcoins_and_Blockchains.html*