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. 1, 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 (short: RDA) smart contract attempts to solve both problems, allowing for higher interest rates for the tenant and for immediate, digital and secure transactions between the parties.

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. 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 Improving rental deposit accounts

In this section we will give a short overview of how a smart RDA contract operates and then take a look at the distribution of incentives, obligations and risks for each of the three parties in a traditional rental deposit account contract and how these aspects change for the smart contract variant. It is assumed the reader is familiar on a basic level with the workings of smart contracts on the Ethereum blockchain.[4]

2.1 Smart RDA contract

A smart RDA contract can be created by anyone for a small transaction fee. To create it, one has to specify immutable Ethereum addresses for the three involved parties: Tenant, renter (landlord) and trustee; as well as a fixed trustee fee. The tenant then transfers the full deposit amount in DAI - a crypto stablecoin[5] - to the contract and starts the contract with another transaction. This locks the DAI security in the DAI savings rate contract (short: DSR) and begins to generate compounding interest on the locked capital. Details for the DSR contract and the interest rate are described in (see section). The trustee fee is paid by the contract to the trustee using aggregated interest. Figure 1 shows a more detailed, schematic view of how the smart RDA works. Note that it is not possible

[4]Reference to more information for smart contracts and addresses
[5]Reference to stablecoins

to interact with the deposit directly, every action is delegated by the RDA smart contract. Actions shown with a dotted line require the consent of at least two participants before they can be executed.

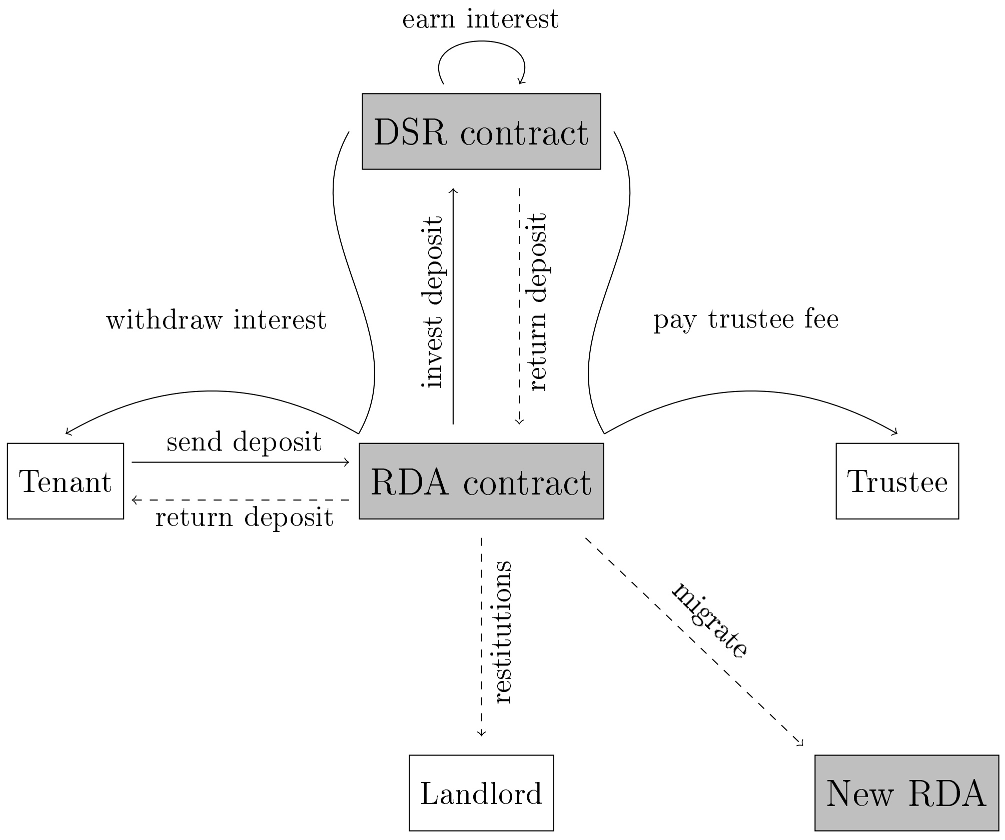

Figure 1: The smart RDA contract is in control

2.2 The trustee

The recommendation by Swiss Law (OR Art. 257e para. 1) is to store the deposit in the form of money or other securities on a (savings-) bank account. The reason for using a bank as the trustee is to minimize the counterparty risk. The bank assumes the role of a trustee and guarantees to safe-keep the deposit until it is eventually released to the tenant or renter. According to OR Art. 257e para 3., the release of the deposit can be requested in a form signed by the tenant and the renter or with a legal instruction by a court or debt enforcement office. During the lifetime of the deposit, the bank is in full control; they can invest it and keep the earnings of this investment. As noted in section 1, the interest passed on to the owner of the deposit is between 0% and 0.05% p.a.

The smart rental deposit account contract still requires a trustee to trigger a release of the deposit to the right party if the tenant and the renter are not in agreement. But contrary to the traditional bank deposit, the counterparty risk is greatly mitigated by using a smart contract since the control over the funds will always remain with the contract and not the trustee. For this reason, the role of the trustee can be assumed by any entity that both the tenant and the renter can agree upon: For example a notary or an institution that offers this service. A compensation has to be provided to the trustee to perform his services, but this compensation, called a trustee fee, can be in form of a one time fixed payment.

2.3 The tenant

With a smart rental deposit contract the locked assets of the tenant are securely invested (while remaining under full control of the contract) and the tenant is eligible for any resulting earnings. On the other hand, the tenant has to pay the fixed trustee fee and assume all of the newly arising risks linked to the underlying technology (see section X for detailed risks). As long as the interest rate of the DSR contract is sufficiently high, the renting period is sufficiently long (to offset the trustee fee) and the tenant values the risks of a system failure low enough, the smart deposit contract will provide a significant upgrade to the traditional system for the tenant.

2.4 The renter

Ideally, nothing would change for the renter. However, the additional risks that arise by using a new and developing technology can not be fully absorbed by the tenant. A small risk remains for the renter: The underlying asset losing some or all of its value while the tenant at the same time becomes insolvent. To compensate for this, we suggest to increase the total amount of the deposit. For example use four instead of three month's worth of rent as the deposit. Note that this is much less detrimental for the tenant than it would be in a traditional setting, as the locked assets will be invested at a profit.

2.5 Speed and ease of settlements

Interacting with a traditional rental deposit account poses significant transaction costs for all involved parties: Every action requires a written form with the signatures of the tenant and the landlord to be sent to the bank. For the smart RDA, all these actions can be performed instantly online with a very small transaction fee (a few cents)[6]. If multiple signatures are needed for an action, it can only be executed once the required parties have digitally agreed on the execution. The details for this process are described in the next section.

Authenticity of the signatures is guaranteed by the blockchain technology with the additional upside that any arbitrary document can be attached to an RDA and signed by the participants. This could for example be used to digitally sign the rental contract, eliminating the need for any physical documents and signatures in a tenant-renter relationship.

3 Multisig on Ethereum

Multisig is the process when more than one account needs to sign a transaction before it can be executed. The Ethereum blockchain does not natively implement any multisig functionality unlike for example Bitcoin[7]. Any implementation of multisig on Ethereum needs to be developed manually in a smart contract, which has been done by many different individ-

[6]Note about gas costs

[7]Bitcoin's P2SH scheme (BIP16) can generate an address from any number of public keys with an additional parameter for how many signatures from these keys are needed for a transaction. This is implemented directly in the settlement layer and provides maximum security.

uals and companies. However, these implementations are on the protocol layer, not the settlement layer and therefore much more open to mistakes, bugs and vulnerabilities. The famous parity hacks of 2017 - see (cite parity hacks) - were both caused by a careless implementation of the parity multisig wallet. It is therefore critical to build the multisig part of the RDA smart contract on a simple, understandable, verified and audited code base.

3.1 Different approaches to multisig

Signatures can be aggregated on-chain or off-chain and the content of the transaction can be stored on-chain or not. Which approach to use depends mostly on the use case of the contract.

3.2 Off-Chain implementation

This implementation is the closest to how Bitcoin implements their multisig: A transaction is created off-chain and then sequentially signed by the required addresses. The multi-signed transaction data is then sent to a smart contract where the signatures are verified and the transaction data executed. A standardized method to process these transactions is described in EIP191 and a minimalistic implementation by Christian Lundkvist can be found here: https://github.com/christianlundkvist/simple-multisig/blob/master/contracts/SimpleMultiSig.sol.

The main advantage of this approach is minimal mutable state on the blockchain. A transaction (and by extension assets) can't end up in a locked state where they are lost due to an invalid state of the contract. The complexity and room for bugs is also much smaller as the whole validation and execution is done in a single function call and the gas costs for the whole process are kept very low (only the account executing the transaction pays any gas).

The two main disadvantages are first, that low level non-transaction call data needs to be signed and it is not trivial, especially for more complicated transactions, for the signer to understand exactly what they are signing. And second, the signers need to coordinate off-chain in order to fully sign the transaction.

3.3 On-Chain implementation

On-Chain multisig stores the transaction in some form - either as direct call data or as struct of parameters - as a state on the contract. This transaction can then be signed by calling a sign function on the contract, which collects the signatures in a mapping. Once the required signatures have been collected, the transaction can be executed. A very popular and audited implementation of this method is the GNOSIS Multisig wallet which currently holds around 350'000 ETH across its most significant instances (Aragon, Golem, Myterium and District0x).

The main advantages of this approach is a complete on-chain settlement with no additional coordination between the signers and a transparent log of confirmations and executions that is permanently stored on the blockchain.

Compared to an off-chain approach, more gas is needed to complete the settlement and the increased complexity of the code renders the contract more susceptible to undesired states and vulnerabilities.

3.4 Conclusion

Our propsed RDA smart contract will perform a very limited number of multisig transactions over its lifetime; we can therefore disregard the slightly higher costs needed for an on-chain settlement. The transparency and persistent logs in combination with a simple coordination mechanism are however very desirable. For these reasons we pursue an on-chain approach and decided to base our implementation of multisig heavily on the proven GNOSIS multisig wallet code. A more technical overview of our implementation will be presented in (reference section).

4 Risks

In this section we discuss the most urgent risks that arise when using an RDA smart contract. The risks can be divided into Counterparty risk, foreign exchange risk and Layer 1 - Layer 4 risks as described by (cite DeFi paper Schär). The following sections assume that the RDA smart contract works as intended; the risk of smart contract failure is covered in Layer 3 risks.

4.1 Counterparty risk

As counterparty risk we understand the likelihood that one of the involved parties does not perform their contractual obligations, due to malicious intent or otherwise. Most of the counterparty risk is mitigated by the smart contract nature of the RDA. Especially the funds are kept completely secure and can't be compromised. The obligations of the trustee - confirming multisig transactions when needed - need to be backed up by a written (off-chain) contract and there remains a risk that the trustee defaults on these obligations. However, the trustee has no personal gain from this behaviour and is contractually liable. It is still vital to chose a reputable entity for this task.

The tenant or landlord can not behave in a way that will compromise the smart contract as long as the trustee fulfils their obligation.

4.2 Foreign exchange risk

Currently the only widely available stablecoins on the Ethereum platform are pegged to the US-Dollar; therefore the rental deposit needs to be paid in US-Dollar. If the US-Dollar is not the local currency of the country where the RDA contract is settled, there exists a risk for the deposit to become undercollateralized if the US-Dollar is devalued. This risk can be mitigated with hedging or similar strategies if desired, otherwise the tenant and the landlord need to agree to absorb this risk. A further possibility would be to include a contract clause that will trigger a migration of the contract with re-collateralization should the exchange rate fall below a certain value.

4.3 Layer 1 risk

Settlement layer risks include a partial or total failure of the Ethereum protocol due to technical issues. This would most likely also default the entire rental deposit. The tenant is fully liable for this risk and needs to set up a new rental deposit on a different platform.

4.4 Layer 2 risk

Layer 2 includes the assets built on the Ethereum protocol; relevant for this smart contract is the stablecoin DAI and by extension the Maker-DAO ecosystem. MakerDAO is still very much a developing and experimental institution that brings the risk of a partial or complete failure. In most scenarios a total default can be prevented if the contract is migrated in time to a different platform on initiative of the trustee and the tenant. However, the tenant is responsible to re-collateralize the deposit. We try to keep the layer 2 risk minimal by not involving any other asset.

4.5 Layer 3 risk

The protocol layer risk is extensively described in Schär 2020, section 3. It consists of smart contract execution risk (coding errors and vulnerabilities), operational security (OpSec) risks and dependency risks (dependency on MakerDao and especially DSR). OpSec risks are negligible as there exists no admin key for the smart contract. Execution risk and dependency risk is absorbed by the tenant. One significant dependency risk is a low interest rate as defined by the DSR contract.

4.6 Layer 4 risk

The application layer risks arises if the parties decide to use the provided dApp https://smartdepos.it or a similar, external interface which has a chance to malfunction and submit faulty data to the RDA smart contract. Every participant bears this risk for themselves, but it is highest for the tenant as they need to send DAI transactions.

5 Functionality of the RDA smart contract

The RDA smart contract is split into three different modules which are outlined in this section.

5.1 DSR module: SavingDai.sol

interface to dsr

5.2 Core Contract with multisig: multisigRDA.sol

core functionality

5.3 Factory for RDA contracts: RDAregistry.sol

factory-child with lookup

6 Implementation

Software development philosophy.

6.1 Environment

Tools used

6.2 Testing

Testing approach and coverage

7 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*