FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Trading Securities on the Blockchain

Design and Implementation of a Prototypical System to Trade Securities Utilizing Smart Contracts

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering and Internet Computing

eingereicht von

Johannes Brottrager
Matrikelnummer 01326320

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Thomas Grechenig
Mitwirkung: Dominik Schmelz

Wien, 15. Oktober 2019 ______________________ ______________________
(Unterschrift Verfasser/In) (Unterschrift Betreuung)

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Trading Securities on the Blockchain

Design and Implementation of a Prototypical System to Trade Securities Utilizing Smart Contracts

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Johannes Brottrager
Registration Number 01326320

elaborated at the
Institute of Information Systems Engineering
Research Group for Industrial Software
to the Faculty of Informatics
at TU Wien

Advisor: Thomas Grechenig
Assistance: Dominik Schmelz

Vienna, October 15, 2019

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Statement by Author

Johannes Brottrager
Grangasse 1/26, 1150 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

I hereby declare that I am the sole author of this thesis, that I have completely indicated all sources and help used, and that all parts of this work – including tables, maps and figures – if taken from other works or from the internet, whether copied literally or by sense, have been labelled including a citation of the source.

(Place, Date) (Signature of Author)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Acknowledgements

First of all, I would like to thank my thesis advisors Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Grechenig and Dipl.-Ing. Dominik Schmelz for assisting me during the whole process, continuously reviewing the thesis and providing valuable input for my work, as well as all secondary reviewers for their very valuable comments on my thesis.

Of course, I would also like to thank all of my family and friends who accompanied me during the writing of this thesis.

Likewise, I am especially grateful to my parents for their support and for enabling my studies.

Last, but not least, I can't thank you enough, Jasmin, for supporting me during my thesis and my whole studies, enduring late nights full of work and stressful weeks of imminent deadlines!

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Kurzfassung

Die Einführung von Smart Contracts durch Kryptowährungssysteme wie Ethereum, die die dezentrale Berechnung von Programmen auf einer Blockchain ermöglichen, bietet viele Anwendungsmöglichkeiten und Einsparpotenziale im Bankenumfeld. Es gibt jedoch nur wenige konkrete Implementierungen und wissenschaftliche Arbeiten, die sich mit dem Potenzial der Tokenisierung von Wertpapieren zum Handel auf einer Blockchain befassen. Die vorliegende Arbeit zeigt deshalb auf, wie ein System, das es Retailbanken ermöglicht, konventionelle Wertpapiere auf einen Blockchain-Ledger zu bringen, entworfen und gebaut werden kann.

Zu diesem Zweck werden andere Systeme, die eine ähnliche Funktionalität anbieten, auf ihre Verwendbarkeit in diesem Bereich getestet und analysiert, sowie vielversprechende Standards für Wertpapiere auf einer Blockchain überprüft. Ein Prototyp wird auf der Grundlage der ERC-1400-Gruppe von Token-Standards, Anforderungen aus bestehenden Projekten und rechtlichen Voraussetzungen erstellt. Der smart-contract-basierte Prototyp repräsentiert konventionelle Wertpapiere auf einer Ethereum-Blockchain und bietet Funktionen wie Know-Your-Customer (KYC) und Anti Money Laundering (AML) Einschränkungen, Dividendenzahlungen und Aktionärsabstimmungen. Er ist konfigurierbar, kann während des Betriebs aktualisiert werden und stellt somit eine erweiterbare Basis für reale Anwendungen dar.

Der Prototyp demonstriert das Potenzial sowie die möglichen Probleme eines derartigen Systems. Durch seine Defizite werden etwaige Forschungfelder für zukünftige Studien, wie Datenschutzbedenken und die Rechtskonformität eines solchen Verfahrens, aufgezeigt.

Letztlich soll diese Arbeit weitere Projekte und Forschungsarbeiten auf diesem Gebiet anregen, um die Effizienz von Bankprozessen zu steigern.

Schlüsselwörter

Wertpapiere, Aktien, Blockchain, Ethereum, Trading, Token, Tokens, ERC-1400, Solidity, Smart Contract

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

The recent inception of smart contracts in cryptocurrency systems like Ethereum, enabling the decentralized computation of programs on a blockchain, offers a lot of potential applications and savings in a banking context. But there are few concrete implementations and scientific works dealing with the potentials of tokenizing securities to be traded on a blockchain. This thesis explores, how a system, that enables retail banks to bring conventional securities on a blockchain ledger, can be designed and built in order to capitalize on this potential.

To this end, other systems, that could potentially fulfill a similar role are reviewed and analyzed for their usability in this scope and checked for promising standards for securities on a blockchain. A prototype of such a system is created based on the ERC-1400 group of token standards, requirements elicited from existing projects and legal needs. The smart contract prototype represents conventional securities on an Ethereum blockchain and includes features like Know-Your-Customer (KYC) and Anti Money Laundering (AML) restrictions, dividend payment, and shareholder voting. It is configurable and upgradeable while deployed, making it an extendable base for real-world applications.

The prototype shows the potential, as well as the issues such a system needs to address. Its shortcomings show possible directions for future research, such as privacy concerns and legal compliance of such operations.

Finally, this thesis should inspire further work and research in this field, to help improve on the efficiency of banking processes.

Keywords

Securities, Stocks, Blockchain, Ethereum, Trading, Token, Tokens, ERC-1400, Solidity, Smart Contract

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Contents

1 Introduction **1**
1.1 Problem Formulation . . . 1
1.2 Motivation . . . 1
1.3 Aim of the Thesis . . . 3
1.4 Structure . . . 4

2 Securities, Blockchains and Smart Contracts **5**
2.1 Securities . . . 5
2.1.1 Definition and Types of Securities . . . 5
2.1.2 Legal Definition . . . 5
2.1.3 Conventional Trading of Securities . . . 6
2.2 Basics of Blockchain Technology . . . 7
2.2.1 Blockchain and Distributed Ledger Technology . . . 8
2.2.2 Byzantine Generals Problem . . . 8
2.2.3 Origins of Blockchain Technology . . . 9
2.3 Basics of Ethereum . . . 9
2.3.1 Blockchain and Accounts . . . 10
2.3.2 Transactions and Messages . . . 10
2.4 Smart Contracts and Solidity . . . 11
2.4.1 Properties of Smart Contracts . . . 11
2.4.2 Oracles . . . 12
2.4.3 Smart Contract Languages . . . 12
2.4.4 Basics of Solidity . . . 13
2.4.5 Attacks on Smart Contracts and Defense Mechanisms . . . 16
2.4.6 DApps . . . 18

3 State of the Art of Securities Tokenization **19**
3.1 Search Methodology and Related Works . . . 19
3.2 Smart Contracts and DApps . . . 20
3.2.1 Wagers . . . 21
3.2.2 Notary Uses and Intellectual Property . . . 22
3.2.3 Gaming . . . 22
3.2.4 Wallets and Identity Management . . . 22
3.2.5 Supply Chain Management . . . 23
3.2.6 Libraries . . . 23
3.2.7 Financial Services Contracts . . . 23
3.3 Asset and Securities Tokenization . . . 24
3.3.1 Tokenization of Tangible Assets . . . 25
3.3.2 Security Tokens . . . 27
3.3.3 Tokenized Securities . . . 32

4 Implementation of the Prototype **36**
4.1 Concrete Problem Description . . . 36
4.2 Requirements . . . 36

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4.2.1 Smart Contracts System 36
4.2.2 Web Client 39
4.3 Smart Contract System 39
4.3.1 Project Setup, Frameworks and Testing 40
4.3.2 Overview of the System 42
4.3.3 ERC1594 Contract 43
4.3.4 Controllers and Verifiers 47
4.3.5 Libraries 48
4.3.6 DividendToken 49
4.3.7 VotingToken 50
4.3.8 Role Based Access 52
4.3.9 Upgradeability and Maintainabilty 54
4.3.10 Discovery/Registry 57
4.3.11 Putting it All Together – Deployment 58
4.3.12 General Problems 58
4.3.13 Potential Extensions 61
4.4 Angular Client 62
4.4.1 Project Setup and Frameworks 63
4.4.2 Overview and Walkthrough of the System 64
4.4.3 Problems and Solutions 67

5 Validation of the Prototype **70**
5.1 Validation of the Requirements 70
5.1.1 Smart Contracts System 70
5.1.2 Web Client 72
5.2 Quality Characteristics of the Implementation 73
5.2.1 Functional Suitability 73
5.2.2 Performance Efficiency 73
5.2.3 Compatibility 74
5.2.4 Usability 74
5.2.5 Reliability 74
5.2.6 Security 75
5.2.7 Maintainability 76
5.2.8 Portability 76
5.3 Results of the Static Code Analysis 76

6 Discussion **77**

7 Conclusion and Future Work **79**
7.1 Conclusion 79
7.2 Future Work 79

Scientific References and Bills **81**

Whitepaper and Other References **85**

A Appendix **95**

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

2.1 Exemplary chains of intermediary participants from a retail investor to the CSD . . . 7

3.1 Overview of the Maecenas Art platform. 26
3.2 Overview of the DS Protocol 31

4.1 High level overview of the complete system showing deployment boundaries, roles, and exemplary use cases 40
4.2 Schematic Overview of the system in form of a class diagram 43
4.3 Overview of all roles present in the access control system, the contracts that use them, the methods that are guarded by the roles, and how all contracts interact. 53
4.4 View of the initial screen of the Registry component, showing the List of deployed Contracts that is used to navigate the system 64
4.5 Editing view on the Registry, that is only shown when the current user in MetaMask is an Orchestrator in the System 65
4.6 Full view of a KYCVerifier, with the current user being a KYCListManager, also showcasing the address bar. 66
4.7 Partial view of controller, showing the list of verifiers and some setter methods . . . 66
4.8 Overview of all VotingToken functionalities by an user that is an orchestrator as well as issuer, i.e. has all functionalities available. 67
4.9 Initializing a transaction 67
4.10 MetaMask showing the transaction that is about to be signed for the user to confirm . 68
4.11 Success message popping up after the transaction is mined 68

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

List of Listings

2.1 ExampleCoin, a very basic implementation of a subcurrency for Ethereum, written in Solidity 13
2.2 The Application Binary Interface of the previously shown ExampleCoin smart contract 2.1 15

4.1 A small excerpt of a JavaScript Test for the DividendToken, showcasing the mocha teststyle and checks for revert 42

A.1 The ERC1594 contract, the base token contract for our security token. 95
A.2 The DividendToken contract, the dividend-aware token subcontract. 100
A.3 The VotingToken contract, implementing voting funtionality, lowest in the inheritance hierarchy. 104
A.4 Unstructured Proxy, the source of upgradeability in our system. 110
A.5 Initializable contract, inherited by all upgradeable contracts that need to be upgraded. 112
A.6 Output of Solium Styleguide after removing critical Errors 113
A.7 Output of slither code analysis after removing critical Errors. The remaining errors are false positives or unavoidable when using proxies. 116

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

List of Abbreviations

ABI Application-Binary Interface

AML Anti Money Laundering

CCPH Central Counterparty Clearing House

CSD Central Securities Depository

CSDR Central Securities Depositories Regulation

DAO Decentral Autonomous Organisation

DASP Decentralized Application Security Project

EVM Ethereum Virtual Machine

ICO Initial Coin Offering

KYC Know-Your-Customer

LLL Lisp like Language

MiFID II Directive on Markets in Financial Instruments

REIT Real Estate Investment Trust

SCL Smart contract language

STO Security Token Offering

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1 Introduction

1.1 Problem Formulation

In the years since the publication of the bitcoin white paper by Satoshi Nakamoto [127] and the subsequent emergence of other cryptocurrencies based on distributed ledger technology utilizing a blockchain, there have been a great deal of inventive applications of investing on the blockchain, most notably the advent of so-called Initial Coin Offerings (ICOs) as a means of securing venture capital.

As of now, there are not many approaches dealing with conventional investing utilizing the blockchain. But there may be a lot to be gained from such an approach where code is law and therefore, investing regulations can be baked into immutable smart code on the blockchain, leading to savings in regulatory expenses. The question this thesis aims to answer is, how a system that enables this must be designed and how it could be implemented prototypically.

Building an exchange to trade assets on the blockchain will be out of scope for this work, as such systems exist in abundance, and there is nothing new to be learned in such an endeavor. In addition, the finer juridical hurdles of such a system won't be extensively covered as this work is supposed to be technical in nature.

The framework is to be created on top of the Ethereum blockchain, as this platform enables the creation of smart contracts for decentralized computation, as well as being widely accepted with a large user base and the second-highest transaction count per day, after bitcoin [89]. It needs to enable traditional security providers to pass out collateralized tokens, that can in turn be traded on the blockchain. To comply with local Know-Your-Customer (KYC) regulations, it must be technically ensured, that those who engage in trading must be known to the issuer of the security. While not all legal requirements can be followed in a prototype, a basic implementation of Anti Money Laundering (AML) rules should also be included in the system. What other requirements such a system needs to fulfill will be answered using a literature review, as well as a retrospective analysis of the prototype.

To sum it all up, the goal of the thesis can be condensed to the following research question:

> What properties must a system for the secure and transparent management of securities on the blockchain feature, and how can a prototype, that fulfills these requirements, be implemented?

1.2 Motivation

The practice of acquiring venture capital by issuing tokens is a very popular method of financing for startups in the cryptocurrency space. Therefore, it does not come by surprise that there are plenty of solutions that help to tokenize the securities of a single company, such as the protocols harbor [86] and polymath [154]. Some companies provide private implementations of similar protocols as a service for companies aiming to tokenize their assets, which include investor communication and voting rights, such as KoreConX [107] or Securitize [171]. However, the projects of the last

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

two can hardly be used as a model for this thesis, as they are largely closed-source and lack public documentation by papers.

Other current projects in the asset tokenization space aim to lift other tangible assets, like intellectual property, real estate or even art to the blockchain, such as TrustToken [192] or Maecenas [118].

Taking all these organizations into account, there are as of now no established projects that bring conventional securities to the blockchain without requiring the original owner of the security to actively devote resources to this effort, as all of these projects do. Such a project is an important step to support further adoption of blockchain technology by enabling current providers of products like investment funds or managed funds to utilize the blockchain. This will ideally result in more transparency and less overhead costs, and bring investment opportunities in companies that don't go through the hassle of utilizing one of the aforementioned approaches to tokenize their securities, to the blockchain.

Although the security tokenization field is somewhat sparsely populated in the commercial space, it is even less explored in the scientific community. Papers like "The Impact and Potential of Blockchain on the Securities Transaction Lifecycle" [119] published by the SWIFT Institute[1] already acknowledged in 2016 that such an application of the blockchain might lead to considerable savings in the securities settlement process, but concrete applications are still rare.

A literature review using existing systematic mapping studies and a small systematic search as described in the methodology of the state of the art chapter 3.1 has found that those papers that do relate to this topic, can be broadly separated into two categories:

1. Discussions on the impact of securities that are traded on the blockchain on **financial law** and what legal problems might arise, such as the Articles [68] and [165].

2. Works that address what **principles** such a system would need to follow such as the master thesis [197] by Eric Wall or the **theoretical design** of such a system, such as the paper "Trading Stocks on Blocks" [138].

The practical industry applications that were found and cited before also each come with a corresponding (white-)paper to illustrate their respective applications, which can be analyzed to find gaps in their application fields:

Firstly, **Harbor** [14] with their "R-Token" standard, which aims to create a system to transfer "crypto-securities" that are already persisted in the blockchain. It is supposed to be a first step into the Harbor ecosystem, but as of yet, there are no other whitepapers on the rest of the system published. **Securitize** on the other hand offers a complete system for the lifecycle management of a digital security for the company that created the security [37]. This paper describes all services of the system, but mostly only superficially and abstract. **Polymath** [110] also aims to provide a method to help issuers of financial products to launch security tokens on the blockchain. It is significantly better documented and more open than Securitize, using public code repositories and standards to its advantage.

But as said before, all of those practical applications of bringing securities to the blockchain still do not address issuing from the perspective of an actor, that wants to issue multiple securities without requiring action by the company that issues the security. This means, that there is still a lot to be explored scientifically, especially concerning a practical implementation of such a system.

[1] The Society for Worldwide Interbank Financial Telecommunication is the provider for the current de-facto standard protocols for global financial transactions.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1.3 Aim of the Thesis

As defined in the research question, the goal of this thesis is to define properties of and requirements for a system that can be used to issue and manage securities on a blockchain via the implementation of a prototypical smart contract system. This endeavor will come across multiple problems, that have to be addressed in the development of the prototype, or the thesis itself.

The overarching process of this master thesis consists of first giving an overview of the current state of the art of asset tokenization and also more general overarching topics such as other applications of smart contract systems to show which kinds of tasks in general can be accomplished by such systems. The conclusions and lessons learned by the found papers and applications, as well as basic legal requirements will be used to establish the necessary properties and requirements that the proposed system must feature.

The next step is the development of the prototype. The experiences gained during the development will be used to further enhance the requirements for such a project in general to create an iterative process alternating between developing and researching. The finished prototype will be validated according to the set requirements, and missing features or those that were not attainable in a realistic time frame will be highlighted and discussed.

The prototype implementation will be accomplished via the development of an ERC-20[2] compliant smart contract in the contract-oriented language Solidity[3] on the blockchain platform Ethereum. This contract will enable security issuers to issue tokens for the dispended securities. Other issues that should be addressed by the prototype are:

KYC The prototype must feature the possibility of restricting access to the system to only users who passed a KYC process to comply with legal requirements and prevent misuse.

AML Challenges of implementing AML features should be identified and a base for providing such features within the system should be created.

Blacklisting It might be necessary to blacklist specific addresses from some securities. Different methods of blacklisting sets of addresses should be implemented.

Special Features of Securities Different types of securities offer different interaction methods for issuers and holders. The most common interactions should be represented by the prototype.

Expandability and Upgradeability The created system should be created in a way, that it can be expanded in the future, possibly even upgraded while being deployed on the network without any data loss.

Access Restrictions The different methods of interaction with the system must be access restricted to entities that are entitled to use them.

In addition, a basic web interface that enables end-users to easily interact with the smart contracts will be created.

The smart contract development will be supported by the truffle suite[4], which consists of the development and testing environment truffle, Ganache, which enables easily creating a personal test

[2] The ERC-20 [12] is a standard for token creation on the Ethereum platform, that enables tokens to be received and spent using regular Ethereum wallets.

[3] Solidity is a Javascript-like language for defining smart contracts on the Ethereum blockchain. It will be used due to its high community support and due to being the de-facto standard in smart contract development.

[4] truffleframework.com (visited on 05/10/2019)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

blockchain that helps manual testing during development and drizzle, which facilitates frontend creation for smart contracts. For static code analysis, the linting tool Solium[5] will be used, which aims to enforce secure coding guidelines in solidity contracts. For testing purposes, the contract will be deployed to a local testing blockchain created by Ganache [189], where it can be safely tested without spending real ether.

To sum it up, the end product of the thesis will be the requirements for a security token system, a functional prototype for such a system, and the lessons learned from the development of the prototype.

1.4 Structure

Chapter 2 creates the underlying knowledge base that is needed for an academic reader as a basis for comprehending the following chapters. Both the basics of financial terms and technical fundamentals will be covered.

The next chapter, 3 will give an overview of relevant projects from academic and industrial origins starting from general smart contract applications, over approaches to asset tokenization to concrete takes on security equity collateralization.

The concrete requirements for the implementation of a blockchain platform for securities will be listed an discussed in section 4.2.

In sections 4.3 and 4.4 the actual design and implementation of the solution will be described.

In chapter 5, the solution presented in the previous chapter will be analyzed according to the requirements from chapter 4.2 and other quality metrics, potential shortcomings will be shown as well as theoretical possibilities to correct them.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[5] github.com/duaraghav8/Solium (visited on 05/10/2019)

2 Securities, Blockchains and Smart Contracts

This chapter informs the reader about the financial and technical fundamentals that this thesis builds upon. This includes the definition of the different types of securities and the legal definition that is to be used in this thesis.

2.1 Securities

2.1.1 Definition and Types of Securities

The term security in the financial context describes a fungible[1], tradable asset that holds some kind of value. Securities can be generally divided into three classes:

Equities Equity securities represent partial ownership of a company or a similar entity. Most commonly, this means owning shares of the capital stock. In the case of common stock, this also comes with proportional control of the company. Although stock ownership does not directly imply regular payments, some issuers pay out dividends to stockholders. When this is not the case, shareholders aim to realize profit through capital gains when they sell the security again.

Debt Debt securities such as bonds represent borrowed money, that must be repaid, regardless of the performance of the issuer on the stock market. The duration, interest rate and other terms of the loan are defined at the beginning of the loan. Just as equity securities, debt securities can also be sold by the holder of the loan.

Derivatives Derivative Securities are more complex assets that derive their current value from an underlying asset, such as the performance of another security, an index, or an interest rate, and can be used for various purposes, such as hedging against a capital loss of an asset or betting on the future price of an asset.

2.1.2 Legal Definition

As a legal term, "security" is defined nonconformably in different jurisdictions. In some it describes all possible tradable financial assets, as in the definition given above, others like the UK's legal system define it only as a defined subset of financial assets. For the purposes of this thesis, the legal framework that is provided by the European Union will be used. In Article 4(1)(44) of the Directive on Markets in Financial Instruments (MiFID II) [60] 'transferable securities' are defined as follows:

> (44) 'transferable securities' means those classes of securities which are negotiable on the capital market, with the exception of instruments of payment, such as:

[1] Fungibility means that two individual assets of a certain type can be exchanged for each other and have the same value, e.g. as denominations in fiat money.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

(a) shares in companies and other securities equivalent to shares in companies, partnerships or other entities, and depositary receipts in respect of shares;
(b) bonds or other forms of securitized debt, including depositary receipts in respect of such securities;
(c) any other securities giving the right to acquire or sell any such transferable securities or giving rise to a cash settlement determined by reference to transferable securities, currencies, interest rates or yields, commodities or other indices or measures;

2.1.3 Conventional Trading of Securities

The industry that deals with the storage and management of securities developed out of the need for investors to store their securities, originally represented by paper certificates, in a safe yet accessible place. Originally, this was done by so-called custodian banks, which then also started to provide other services in addition to safekeeping, most prominently:

settlement the delivery (buying or selling) of securities in exchange for money.

asset servicing assisting the holder of a security with exercising the rights that come with that security, such as the collection of interests or voting at shareholders meetings.

While an involved shareholder could do these things individually, it is much easier for most customers to rely on a custodian bank for these services.

As the financial market evolved, and the trade of securities became higher in volume and more complex, leading to an impracticality in physically swapping security certificates, regulatory bodies called for the establishment of so-called Central Securities Depository (CSD)s. These are institutions in which securities for all custodians in a country are immobilized, to eliminate the need for physical delivery. Instead, a change in ownership can be realized by a change in the books of the issuing CSD [24].

Theoretically, this should mean that with a national quasi-monopoly CSD there should be no need for further actors or custodian banks apart from the investors themselves. But this is not always possible because an investor or another market participant might either be unfit for the membership criteria of the CSD[2], or the participant might just not want to go through the hassle of registering with a CSD and use an intermediator for this reason [155] [24].

This leads to chains of Brokers, Custodians and Retail investors leading from the original investor to the actual CSD where each participant adds time and overhead costs to a trade, which is illustrated very descriptively in Figure 2.1. Even more complexity is introduced in cross border security trading, as multiple CSDs might have to get involved.

Another complicating factor is that for most trades[3], because no trading party wants to risk a defaulting illiquid counterparty, there are central parties assuming this risk. These institutions are called Central Counterparty Clearing Houses (CCPHs), which collect information about the trading parties to assess the respective risk of a default and reserve appropriate margins from the trading parties to alleviate a potential default while collecting additional fees from the market participants [44].

[2] Membership criteria are financial and operational capabilities, and sufficient regulation, as CSDs strive to minimize interruptions of their function

[3] As of 2012, all electronic trades are mandated to be cleared by a Central Counterparty due to EU regulation No 648/2012 [62]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek, Your knowledge hub, WIEN

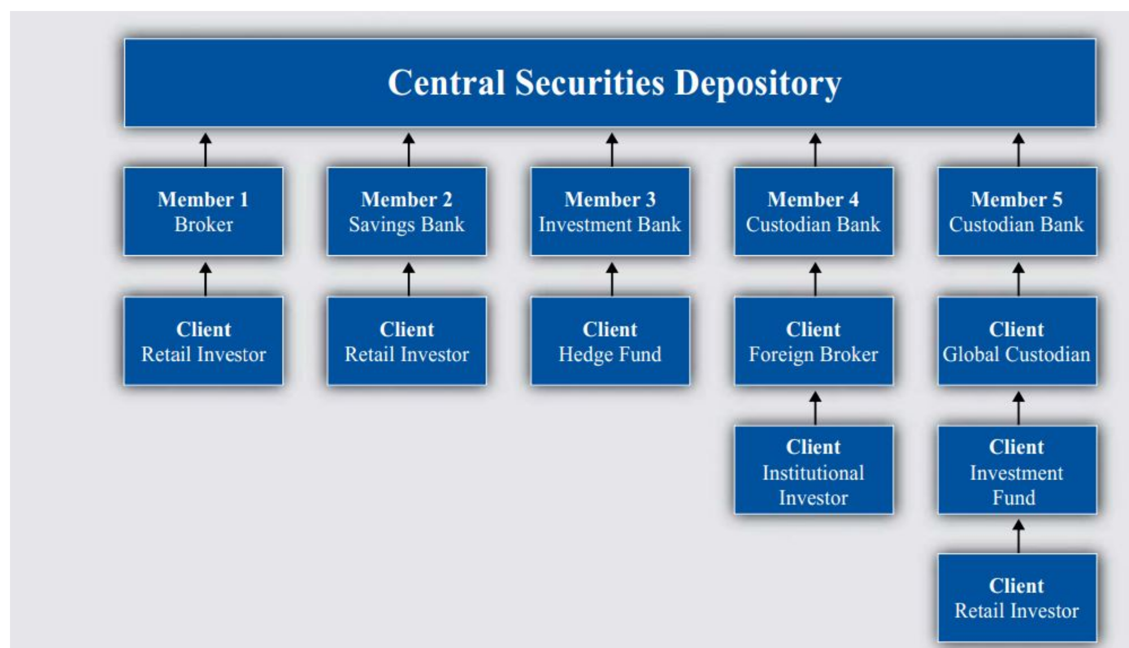

Figure 2.1: Exemplary chains of intermediary participants from a retail investor to the CSD
Source: The Securities Custody Industry by Chan et al. [24]

An additional cost factor is that in order to initiate a trade, the seller and the buyer of a security first have to discover each other. This discovery and matching is typically done via a broker as a middleman or an exchange where sellers can list their sale prices and buyers list their buying prices. This introduces another entity that collects fees into the process.

As for the time needed for a settlement in the current securities trading system, according to the Central Securities Depositories Regulation (CSDR) [65] of the European Union, the settlement times for security transactions between CSDs of member states have recently been reduced from T+3 to T+2[4], which still is a long time in a fast paced, digitized world.

As for the maintainability of the current system, it can be said, that it is very cumbersome to adapt and improve, as evidenced by the implementation of Target-2 Securities, a project by the European Central Bank. It aimed to streamline intranational trading of securities and to provide a single technical platform [57] and has been ongoing from 2008 on until 2017 from project start to the last implementation in national CSDs of participating member states [45].

Considering the convoluted list of actors, and the operational costs and latency that comes with their multitude, it is evident that there is room for improvement. This could potentially be reached by utilizing distributed ledger technologies, as proposed by multiple scientific publications [23], [149] and [165]. How this technology might support an alternative to the current system will be discussed in the following sections.

2.2 Basics of Blockchain Technology

After discussing the problem field in the previous section, this section aims to inform the reader about the basics of the intended solution space.

[4] T+x means that between the payment (transaction) and the actual transfer of ownership of the security (settlement) there may be up to x working days.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

2.2.1 Blockchain and Distributed Ledger Technology

So what does the buzzword "blockchain technology" encompass, and what possible applications arise from it?

A *blockchain* is a perpetually expanding list of blocks, which are linked by cryptographical hashes of previous blocks, i.e. each block contains a hash of the previous block. This ensures that no part of the chain can be altered without invalidating the following hash, and in turn the whole chain. This, combined with the fact that a blockchain is typically shared in a peer-to-peer network with a consensus algorithm so that every participant has a copy of the whole chain at all times creates a tamper-resistant distributed database. Such distributed databases are called *distributed ledgers*[5].

Although there have been other applications of distributed ledgers, this technology has certainly seen a significant boost with the publication of the Bitcoin whitepaper by one or multiple persons under the pseudonym Satoshi Nakamoto [127]. This whitepaper is the base for the popular peer-to-peer electronic cash system Bitcoin. It uses a distributed blockchain to save the state of a digital currency system, where no trust in a central entity is needed. One problem that occurs with distributed databases, especially when participants are incentivized to alter the state of the system for their benefit, that is solved by the Bitcoin system is the so-called *"Byzantine Generals Problem"*.

2.2.2 Byzantine Generals Problem

The Byzantine Generals Problem was first formulated in 1982 by Lamport et al. [112]. It states that

> "... several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can communicate with one another only by messenger. After observing the enemy, they must decide upon a common plan of action. However, some of the generals may be traitors, trying to prevent the loyal generals from reaching an agreement. The generals must have an algorithm to guarantee that (a) all loyal generals decide upon the same plan of action (...) and (b) a small number of traitors cannot cause the loyal generals to adopt a bad plan."

A distributed ledger that manages currency faces an analogous problem. Each actor has to decide whether it trusts messages from other participants. In most current implementations of a Blockchain system, such as Bitcoin, this problem is solved by a concept called Proof-of-Work.

Proof-of-Work describes a control for a system to prevent spam or false messages by requiring the sender of a message to do some verifiable work. This work must be adjustably hard, but very easy to verify. In the context of Bitcoin, this is done by searching for a value (called *nonce*), that in combination with a timestamp and the transaction data a client wants to send, results in a certain number of zero bits at the beginning of the resulting hash. The required number of zero bits can be modified to adjust the difficulty of the hash puzzle. It also solves the problem of finding a majority for the consensus by equating computing power with voting rights [127].

How Proof-of-Work solves the Byzantine Generals Problem was addressed by Satoshi Nakamoto by an analogy in a mailing list [128]. In essence, it is postulated, that a plan is broadcast, and each general, upon receiving the plan, computes a difficult to solve Proof-of-Work hash puzzle. The first general to find the solution broadcasts the plan including the solution, and all generals that

[5] The terms "blockchain" and "distributed ledger" are frequently misused by articles. While most blockchains are stored in a distributed way, using a consensus algorithm, not all distributed ledgers are blockchains.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

receive this message will include the solution into their computations. Once a sufficient number of consecutive solutions has been reached, each general can be sure, that a majority of the other generals have seen and agreed to the plan.

2.2.3 Origins of Blockchain Technology

But the concept of a chain of blocks containing hashed information itself is not as novel, as the recent boom would suggest. For example, it was already utilized in 1991 by Stuart Haber and Scott Stornetta, in a method to timestamp documents by hashing them and printing the hash in a newspaper [83]. Between this and the publication of the Bitcoin whitepaper, there were also some other applications of cryptographically secure chains of blocks, like the creation of a schema to proof the authenticity and Completeness of log files by Stefan Konst in [105]. Some basic ideas for a privacy-preserving, decentralized currency have also already been published by Wei Dai in 1998 in a cryptography mailing list, which was also referenced in the original Bitcoin whitepaper [31].

But only with the Bitcoin whitepaper [127], the first major application of blockchain technology [82] did it gain publicity. This inspired other applications of this technology beyond cryptocurrencies, utilizing the blockchain as a decentralized ledger for e-voting, digital identity management, healthcare, and many other fields [167].

The popularity and success of Bitcoin gave rise to other implementations of cryptocurrencies, including so-called second-generation [163] blockchains, with Ethereum [19] leading the way. These blockchains are characterized by the possibility of distributed on-chain computation in the form of smart contracts. Other examples of such platforms include the so-called “Chinese Ethereum”, Neo [130], that enables developers to write smart contracts in multiple programming languages (eg. C#, Python, Java, and Javascript), or RootStock, built on top of the Bitcoin platform to enhance it with smart contracts by computing them on a sidechain [114].

2.3 Basics of Ethereum

Now how can this technology be used to facilitate the trading of securities?

For this, we turn to the mentioned Ethereum Platform. It was originally conceived by Vitalik Buterin and Gavin Wood as an alternative to Bitcoin to enable developers to build decentralized applications and explore the possibilities of decentralized computing [201]. Buterin initially proposed an expansion to the capabilities of Mastercoin [198], an abstraction layer to the bitcoin protocol, aimed to provide additional features to the barebones scripting possibilities of Bitcoin. But this was rejected by the Mastercoin Team, due to the changes being too radical for their plans, which prompted Buterin to create Ethereum [4].[6]

Ethereum is a distributed, decentral peer-to-peer Network, that uses a blockchain as data storage. As in the Bitcoin system, participants may validate transactions for a reward using a proof of work algorithm. This reward is paid in ether, the currency of the Ethereum platform, similar to the bitcoin currency on the Bitcoin platform. But this platform also provides a virtual distributed machine to execute code, so-called “Smart Contracts”. The execution of this code is paid for with ether, so that miners executing the code get reimbursed for their CPU time. The value of ether has had an extreme all-time-high of over 1400 $ as of January 2018 [103] before plummeting to around 100 $ and has since (as of Sept 2019) settled at about 200$. [7]

[6] A very interesting and exhaustive writeup on the (pre)history of Ethereum by Vitalik Buterin is [17].

[7] The Prices were taken from coinmarketcap.com in September 2019.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

2.3.1 Blockchain and Accounts

In the Bitcoin blockchain, the state of all unspent units of bitcoin and their respective owners are stored. This can be modeled as a finite state machine, in which each transaction causes a state transition, in which the ownership of the coins is altered. Through propagation via the network and the consensus algorithm, eventually, all nodes will agree on the state of the blockchain, until the next block is added to the chain.

Ethereum, on the other hand, does not only track the state of the currency allocations, but all kinds of information in a key-value store. To be exact, this key-value store contains all the *accounts* that make up the state of the Ethereum blockchain. Both the actual accounts of users and deployed smart contracts are stored as accounts. Each account consists of four fields: [19]

nonce This is a simple counter of all sent transactions of this account (or more precisely, a counter of all sent CREATE opcodes), which is increased with each sent transaction. This helps to ensure that transactions of an account are not worked into the blockchain in the wrong order, and, more importantly, it prevents replay attacks where a single transaction of the account would be resent by an attacker to drain the account. (This nonce is not to be confused with the proof-of-work nonce at the start of a block, which is changed by miners in order to find a block with a matching hash value during the mining process.)

balance The number of ether this account currently possesses.

contract code If the account is a deployed smart contract, this is where the code that is executed when the smart contract is activated by a transaction, is stored.

storage If the account is a contract account, this is where all the variables that are stored permanently, are persisted.

As mentioned before, there are two types of accounts: *Externally owned accounts* or user accounts, and *contract accounts*. Externally owned accounts are controlled by private keys. Whoever knows the private keys of an externally owned account can create and sign transactions from this account. These also contain no contract code or storage. Contract accounts on the other hand, are controlled by their code and can not initiate transactions on their own. For smart contract code to be executed, the contract account has to receive a message including enough ether funds (called “gas” in this context) to pay for its execution. When a contract account is activated, it can execute calculations, access its storage and send messages to other accounts, in turn activating further contract accounts [91].

2.3.2 Transactions and Messages

In the above section, both terms *transaction* and *message* have been used to describe interactions between contract accounts and externally owned accounts. But these two terms do not describe an identical concept. Transactions can only be issued by an externally owned accounts and can simply transfer an amount of ether, execute a function of a smart contract, or deploy a new smart contract. They contain:

- The recipient of the transaction
- A signature identifying the sender
- The amount of ether that is being sent from the sender to the recipient
- An optional data field that can be used as inputs for the smart contract code

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- The STARTGAS value
- The GASPRICE value

The last two values are an integral part of the anti denial of service strategy of Ethereum. In order to prevent malign or accidental infinite loops or blown up storage requirements in the execution of smart contract code, each transaction must have a predefined limit on how many computational steps it may trigger and must provide the funds to pay for them. This limit is the STARTGAS value. Usually, each low-level step in the execution of smart contract code costs 1 unit of gas, with some exceptions. In addition, each byte of transaction data is reflected by a cost of 5 gas. The GASPRICE value expresses how much ether the sender of the transaction is willing to pay per unit of gas. A higher gas price will usually mean that a transaction is mined faster, as the gas the sender is paying is distributed to the miner of the block of the transaction, an miners will try to include the highest paying transactions in their blocks, to maximize their profits [19].

Messages, on the other hand, are sent by a smart contract and not an externally owned account. Whenever an activated contract needs to call another contract to use its methods, it sends a message to the contract in question. This means, that while transactions are persisted in the blockchain, messages between contracts only exist inside of the Ethereum execution environment when the call is performed. Another important fact to bear in mind is that all messages of a smart contract have to be paid for by the gas allowance that started the execution of the initial contract.

2.4 Smart Contracts and Solidity

In the previous sections the term “Smart Contract” was mentioned multiple times. What exactly are smart contracts and in what environment are they executed?

2.4.1 Properties of Smart Contracts

Historically speaking, this term has been first used by Nick Szabo [181] to describe an implementation of a real-world contract, i.e. it consists of promises between parties, that are enforced programmatically. This definition has changed somewhat since then. In the context of Ethereum, a smart contract is an immutable program that runs on the Ethereum Virtual Machine (EVM) as decentralized computation in the Ethereum network. The code execution is part of the transaction validation progress, i.e. smart contract code never runs on the EVM without being called by a transaction. This means, from the perspective of a mining node, it receives unconfirmed transactions from the peer-to-peer network, which are then validated. The validation consists of checking if the signature of the sender of the transaction is valid and if the spending account owns enough ether for this transaction. In addition, all smart contract methods that are called by the transaction are executed, as long as the steps are less than what was specified in the STARTGAS field. Once enough transactions to fill a block are validated (and their code is executed), the mining node starts to try for a hash on the block, which would finish the block creation.

Another important property of a smart contract is that the contract code is immutable. Once the smart contract is deployed, its code cannot be changed, as it is stored on all full nodes in the network. This means, that the only possibility to upgrade an existing smart contract is to create a new smart contract. In addition, the author of the smart contract does not have any special rights in it. But it is possible to manually program such special rights in the code. One of those special rights is the deletion of a smart contract, which can be used in cases where a smart contract should be replaced by a newer version. In this case, the author has to send a SELFDESTRUCT Opcode to the smart contract, which removes it from the state of the blockchain. Such a transaction has negative transaction costs to incentivize publishers of smart contracts to remove unused contracts.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.4.2 Oracles

So how can these immutable pieces of code, that are only executed within the context of the deterministic EVM gain any information from the outside world to decide depending on external events, or to create a source of true randomness[8]?

The answer to this question are oracles. An oracle is an implementation pattern, in which outside data can be published to the blockchain, without introducing indeterminism. The most common way to achieve this is by publishing an event to the blockchain, that can be listened for by outside services. These services then can in turn emit events including the queried data to the blockchain, or call a function of the requesting contract with the off chain information.

Of course, this raises some concerns. A single oracle instance is very centralized, introducing a single point of failure and going against the general decentralized idea of distributed ledgers. In addition, the authenticity of the data must be guaranteed in a way. But solutions to these problems are available: One solution is that oracles can also be constructed in a decentralized way, implementing a consensus algorithm between multiple contracts. Authenticity, on the other hand, may be guaranteed by digitally signed proofs or hardware-based security.

Many users of oracles do not implement these themselves, but rather fall back on oraclizing services, such as provable.xyz in order to provide external data to their smart contracts.

2.4.3 Smart Contract Languages

The code execution during the transaction validation takes place in the EVM on each mining node. The EVM executes its own stack-based low-level programming language, which is commonly referred to as "EVM code". There are multiple high-level programming languages designed for smart contract creation, that compile to this binary code. They all aim to provide a secure way to write bug-free smart contracts, as bugs in smart contracts can easily be disastrous, especially due to their immutability. Some of the currently existing Smart contract languages (SCLs) are:

Solidity Easily the most popular SCL for development in the Ethereum system, Solidity is an imperative, object-oriented language with a similar notation to javascript. It was developed by Gavin Wood as one of the "original" languages for the Ethereum platform. Its syntax aims to be easy to learn for developers that have experience with Java or other object-oriented languages. Features like inheritance, user-defined types and external libraries facilitate the creation of complex smart contracts with relative ease [52].

LLL The declarative "Lisp like Language" was the very first language compiling to EVM bytecode that was developed. It is minimalistic and assembler-like, more or less just a small layer of abstraction over bare EVM code. As many standard high-level language features are missing from LLL, the developer is forced to deal with the highly resource-constrained nature of the EVM, which in turn helps to produce very clean and efficient EVM code, especially compared to the Solidity compiler [46]. As of now, there are no new features being developed in LLL, as the focus has shifted to Solidity [51].

Serpent A python like imperative language, that was also developed in the early stages of Ethereum development. It compiles to EVM code via LLL and enables low-level opcode programming as well as high-level interfaces. But as the development on Serpent has been halted and

[8] The reason why the EVM cannot simply allow REST requests or something similar is that the code needs to be deterministic and possible to be computed in a lot of distributed nodes at once, which could also create problems for the data source, if many nodes tried to check for a result at once. In addition, this would also introduce indeterminism if the data sources response changed from one request to another.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

numerous flaws have been found in the language [207], it is not recommended to use Serpent any more. For low-level optimizations, LLL is still the best choice, for a python-like language, Serpent has been replaced by Vyper.

Vyper Vyper can be regarded as the successor of Serpent in a way, as it also uses a python-like imperative programming approach. It is designed with a focus on simplicity and auditability, this means, while some complex smart contracts might require more effort to implement, human readability and security through auditability is supposed to be easier to achieve [53].

Flint Besides the aforementioned popular languages, there are also a plethora of other smart contract languages being developed or recently published, expanding the diversity of languages to choose from for smart contract development. One of those is the language Flint, which was developed by Franklin Schrans [169]. It concentrates on security by design, aiming to prevent smart contract developers from making critical security related errors. As it is a very new language, that is still being developed, there is hardly any adoption, but it nevertheless serves as an example of the possibilities of smart contract language development.

A nonexhaustive curated list of additional smart-contract languages can be found in a Github repository[9] by researcher Sergei Tikhomirov.

2.4.4 Basics of Solidity

As mentioned above, Solidity is the most popular language for smart contracts. It is the de-facto standard for development on Ethereum, as it is also prominently featured on the website of the Ethereum Foundation. Therefore, it is also the most scientifically scrutinized and explored language of the stated SCLs, with the largest community support. This and its high-level features that enable the development of complex smart contracts are the reasons why it was chosen as the language to be used in the development of the securities trading system that was undertaken in the course of this thesis. At the time of writing, the current version of Solidity is 0.5.10. As the leading zero in the version number indicates, the Ethereum Foundation considers Solidity to be in its initial development phase, despite its widespread adoption, as it is still rapidly evolving, as can be witnessed on its github page[10].

In order to explain the very basics of the Language, and how such a smart contract might look like, the code listing 2.1 exemplifies a very basic implementation of a subcurrency resp. token in Ethereum. It allows the distribution of ExampleCoin tokens by the owner (= the initial deployer) of the contract to other Ethereum addresses. The current distribution of the coins is stored in a mapping from addresses to integers. Every address that has coins according to this mapping, can send as many coins as it possesses to another address via the send method. On a call to the send method, an event is emitted to the blockchain, to alert clients to a change of state in the currency ownership.

```
1 pragma solidity ^0.5.0;
2
3 contract ExampleCoin {
4     // declaration of storage variables
5     address payable public owner;
6     mapping (address => uint) public balances;
7
8     // Events can be watched for by clients, enabling monitoring of a smart
      ↪ contract without much resource usage.
```

[9] github.com/s-tikhomirov/smart-contract-languages (visited on 05/10/2019)
[10] github.com/ethereum/solidity (visited on 05/10/2019)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
9      event Sent(address from, address to, uint amount);
10
11     // The constructor, that is only called once on deployment, sets the
       ↪ owner and the initial balance.
12     constructor(uint supply) public {
13         balances[msg.sender] = supply;
14         owner = msg.sender;
15     }
16
17     function destroy() public {
18         require(msg.sender == owner);
19         selfdestruct(owner);
20     }
21
22     function send(address receiver, uint amount) public {
23         require(amount <= balances[msg.sender], "Insufficient balance.");
24         balances[msg.sender] -= amount;
25         balances[receiver] += amount;
26         emit Sent(msg.sender, receiver, amount);
27     }
28 }
```

Listing 2.1: ExampleCoin, a very basic implementation of a subcurrency for Ethereum, written in Solidity

The first line informs the compiler that this code is intended for all versions of solidity from 0.5.0 up to but excluding 0.6.0, working under the assumption, that potentially breaking changes are only introduced in such a major version change. This is indicated by the "^" sign. Alternatively, versions can be formulated explicitly using ">", "<", and "=" signs.

After the declaration of the contract itself come two variable declarations. They are storage variables, that are persisted even if the contract is not currently being executed. The public keyword makes the variables accessible from external sources, i.e. getter methods are generated, so that other contracts can access these variables[11]. The address type of the owner variable is a 160-bit value that does not allow any arithmetic operations. It is used for storing addresses of accounts or keypairs. The keyword "payable" indicates, that it is possible to pay to an address or function, and is necessary to allow the contract to send funds to the address stored in "owner". The keyword "mapping" creates a map from addresses to unsigned integers storing the balance of each possible address.

The next line declares an event, a solidity object that is used to indicate the occurrence of an action that can be easily watched on the blockchain. This is especially useful for light clients or DApps that want to reflect live updates in their user interface, without using many resources. In our example, a client UI could watch for these events and represent a live state of the coin distribution of our Examplecoin.

The next element in the listing is the constructor, which is only called once, during the deployment of the contract. During this, the initial supply of coins for this method is set as the balance for the creator, who can, in turn, distribute these coins. This method also stores the address of the contract creator, utilizing "msg" which is a property of the transaction that started the current execution of a smart contract. This is important to enable the creator of the smart contracts to have special rights, as otherwise, he would be considered the same as any other user. This pattern is often used for failsafe-mechanisms, where the contract can be shut down in case of an error or if it is obsolete.

[11] It is not correct to assume, that in turn, variables without the public keyword will be hidden from the general public. As the storage is public on the blockchain, they are not kept secret this way, but just not easily accessible by other smart contracts.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The next method utilizes exactly this previously set owner field. The keyword “require” forces a check before the execution of the method continues. If the sender of the transaction is the owner, the smart contract will be destroyed, resulting in loss of all the Examplecoins. All ether funds that might have been sent to the contract (by accident, as this contract is not designed to utilize its ether funds) will be sent to the address stored in the “owner” variable.

Finally, the send method enables a caller to send a certain amount of ExampleCoins to a receiver. As before, the require keyword is used to ensure that the sender actually has a large enough balance in our balances ledger. If this is not the case, an error message is relayed to the sender. In addition, if a require call evaluates to false, all changes are reverted. If it passes, the balances are adjusted accordingly and a Sent event is emitted, as described above [4] [52].

Another property of Ethereum worth mentioning in combination with smart contract programming is the Contract Application-Binary Interface (ABI). In general, an ABI is a low-level interface between two binary program modules in machine code.

In the Ethereum context, the ABI is the way to communicate with a deployed smart contract. As these bytecode deployments do not have any contextual information, a user of the contract, be it an end user or another smart contract, does not know how to address it, or what functions it possesses. The ABI is therefore described by a .json file, that outlines all functions with their arguments and return types, public variables, and other information, that users of the smart contract have to be aware of to use it.

The following listing 2.2 shows the abridged ABI description of the previously discussed Example-Coin smart contract.

```
1  [
2      //variable definitions
3      {
4        "name": "owner",
5        "constant": true,
6        "inputs": [],
7        "outputs": [
8          {
9            "name": "",
10           "type": "address"
11         }
12       ],
13       "payable": false,
14       "stateMutability": "view",
15       "type": "function"
16     },
17     {"name": "balances" ...},
18
19     //definition of the "sent" event
20     {"name": "Sent",
21       "type": "event",...},
22
23     //function definitions
24     {"type": "constructor"...},
25     {"name": "destroy"...},
26     {
27       "name": "send",
28       "constant": false,
29       "inputs": [
30         {
31           "name": "receiver",
32           "type": "address"
33         },
34         {
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
35            "name": "amount",
36            "type": "uint256"
37          }
38        ],
39        "outputs": [],
40        "payable": false,
41        "stateMutability": "nonpayable",
42        "type": "function"
43      }
44    ]
```

Listing 2.2: The Application Binary Interface of the previously shown ExampleCoin smart contract 2.1

A brief overview of this ABI will be given by reference to the "owner" variable and the "send" method. A complete explanation of the ABI and its rules in general would go beyond the scope of this work. Such an explanation can be found in the solidity documentation [52].

As the above snippet shows, both (public) variables and methods are encoded by the exact same fields, as such public variables are treated as getter methods without an input. They are described by their name, the boolean "constant", that describes whether the function leaves the storage of the contract unchanged, two arrays of inputs and outputs with their respective names and types, another boolean describing whether the function is payable, i.e. whether ether can be sent to this function or not. After this, there are two more fields remaining. One is the `type` field, describing the category of the function with the possible values of `function`, `constructor`, or `fallback` for the unnamed default function. The other is the `stateMutability` field, with the following possible values:

pure the function does not read the blockchain state

view the function does not change the blockchain state

nonpayable the function does not accept ether

payable the function accepts ether

Events are also described in a similar fashion in the ABI description. To sum it up, to work with another contract, a smart contract has to have access to its ABI at compile time, or else it will not be able to use its functions.

2.4.5 Attacks on Smart Contracts and Defense Mechanisms

Naturally, code that potentially controls a lot of monetary value is a prime target for persons with the malicious intent of exploiting bugs or weaknesses, and therefore has to be scrutinized meticulously for errors. In the history of smart contracts on Ethereum, this has not always been the case and therefore there are multiple examples of attacks on smart contracts that led to a loss of ether.

Attacks on smart contracts often utilize common vulnerabilities. The most common vulnerabilities have been listed by the Decentralized Application Security Project (DASP) Top 10 [129]. The DASP is a project started by the NCC Group based on the widely known OWASP Foundations Top 10 web vulnerabilities [144]. Other publications that describe vulnerabilities are [117], [35] and [7]. The following is an overview of some of the vulnerabilities and mitigation techniques described in these and other publications, as well as some prominent hacks utilizing them.

Reentrancy or call to the unknown The reentrancy attack is arguably the most popular exploit in Ethereum history, as well as the most impactful one. In general, a reentrancy vulnerability

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

occurs, when a contract sends ether to another (untrusted) contract, which in turn may make calls to the sending contract before the original execution is complete. This can lead to an unexpected state in the execution if exploited by a specifically designed smart contract. This was the case in the so-called DAO Hack, which led to a hard Fork and the subsequent split of the Ethereum blockchain in two parallel projects, Ethereum, and Ethereum Classic. "The DAO" was a smart contract that acted as a crowdfunding venture capital fund, which was vulnerable to a reentrancy attack. 50 million dollars worth of ether were stolen in an attack on the 16th of June, 2016. This theft was reverted by a hard fork of the Ethereum blockchain, which led to the split of the projects [8]. Such attacks can be mitigated by using the builtin `transfer()` function instead of the `send()` function, which has a gas stipend that is to small for the called contract to execute a reentrancy call. Another technique is to ensure that all state changes are finished before external calls are executed. This is a part of the Checks-Effects-Interactions Pattern [175] that guards against reentrancy attacks by first checking conditions, then applying state changes and then calling external contracts last [166].

Access Control or Default Visibility An access control vulnerability occurs when access to functions or variables is not restricted as intended, e.g. by omitting the `private` modifier or by using the deprecated `tx.origin` property to validate the callers, which may be obfuscated by sending a transaction through another smart contract. This vulnerability was utilized in two attacks on the parity multisig wallets, a part of a popular wallet software. Both attacks, one of them supposedly accidental, utilized an initialization method outside of the constructor, which could be called from any user, that made the caller the owner of the contract. In the first hack, this led to the theft of the funds of some wallets, in the second hack, a core library smart contract was deleted, leading to the immobilization of all funds of the multisig wallets [147], [146], [129]. This could have been prevented by initializing the library contract with the library keyword, which would have made it stateless and non-self-destructible, and making the initializing functions private.

Integer Over- and Underflows Similar to issues in conventional software engineering, integers might be over- or underflown. This can be mitigated by opting to use safe standard libraries instead of the standard implementations of arithmetic operators.

Unchecked Call Returns As mentioned before, calls and payments between contracts can be made by different methods: `call()`, `callcode()`, `delegatecall()` and `send()`. These methods have in common, that on an error they do not throw an exception, as many other functions would do, but simply return false. If a programmer does not anticipate and check these return values, this might lead to an unexpected state which can be exploited.

Denial of Services If an Ethereum contract that is used by other contracts becomes unusable in any way, this can lead to devastating results, as funds may be locked inside other smart contracts as described in the Parity library deletion [146]. This is more of a symptom than a specific vulnerability and therefore has no specific mitigation technique.

Entropy Illusion As the Ethereum State Machine is deterministic as a whole, true randomness cannot be achieved from within the Blockchain [4]. In addition, if the randomness is derived from the timestamp of the block, miners may have an incentive to manipulate the timestamp, controlling the randomness. To get truly random input, outside sources, not controllable by miners, have to be used.

Race Conditions As every transaction on the blockchain is public, even if its not persisted yet, it is easily imaginable that some actors might utilize information (like, the solution of a puzzle that a smart contract might reward) in transactions that wait in the mining pool to their

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

advantage, e.g. by cutting in front of these transactions via a higher gas price. A possible solution to these kinds of problems are commit-reveal schemes, in which the transaction is split in two. The first transaction only commits a hash of the actual value e.g. the solution of the puzzle, and a second transaction presents the actual solution, as soon as the original solver is fixed.

2.4.6 DApps

After this extensive description of the features of Solidity and the Ethereum platform one thing has yet to be mentioned: How can a not technically inclined user interact with smart contracts? The answer to this are Decentralized Apps or DApps, often also written as ÐApps[12]. DApps are part of the long-term vision for Ethereum of the Ethereum Foundation, which hopes to establish a whole new, distributed version of the World Wide Web called web3[13], based on the so-called Web 2.0, that describes the web of user-generated content.

A DApp at its core is at least a deployed smart contract representing the business logic of the application and a conventional web interface. Communication between the user of this web interface and the smart contract can be accomplished by browser extensions like metamask [124] that enable direct communication between the browser and the Ethereum network. Of course, a complex DApp will most likely consist of multiple interoperating smart contracts that act more or less autonomous. But as such business logic and especially storage requirements get larger, it may become uneconomic to store all of the business data on chain, as smart contract storage is comparably expensive. To this end, DApp developers can utilize decentralized storage solutions like Ethereums own Swarm [50], which aims to provide peer-to-peer decentralized storage for DApp development. To further improve on the decentralization aspects, a DApp can also utilize decentralized domain name services like the Ethereum Name Service that maps human readable addresses to contract addresses [47].

While currently, a lot of smart contract applications adorn themselves with the title of a DApp, only a few are truly and completely distributed from a puristic point of view. While some developers might purposefully choose to keep some aspects of their applications centralized for business reasons, overall the number of true DApps may rise as the technology matures.

[12] The Ð character in the term ÐApp is the Latin character "ETH", hinting at Ethereum.
[13] This also explains the name for the JavaScript library for the interaction with smart contracts, *web3.js*.

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

3 State of the Art of Securities Tokenization

This chapter aims to give an overview of currently existing tokenization solutions, as well as broader applications of smart contracts and DApps. For this, scientific sources, as well as industrial projects, are consulted.

An initial scoping review of the existing industrial applications and scientific works in the problem space of securities tokenization did not produce many results of scientific contexts. Therefore, systematic mapping studies of general blockchain applications[1] were used as a springboard for the state of the art overview. Systematic literature reviews with specifications on subtopics of blockchain research such as supply chain management or security aspects of blockchains were not considered.

3.1 Search Methodology and Related Works

One of the earliest most cited systematic literature reviews according to citation counts and metrics on the citation databases Scopus and Google Scholar is the systematic review "Where Is Current Research on Blockchain Technology?" by Jesse Yli-Huumo et al. which was published in 2016 [205]. As evidenced by the results of this study, in earlier days of blockchain research, most publications focused on the bitcoin system and its challenges[2], rather than other applications of blockchain technology such as smart contracts. The study also identifies the lack of research in contexts outside of the bitcoin system as a major research gap and recommends further research with other systems and smart contracts. Another shortcoming of published research according to this study is the lack of high-quality publications in journal-level publication channels. Of course, due to the age of this publication, these conclusions might potentially not hold validity anymore but are nonetheless an indicator for the state of the art at the time.

A more recent general blockchain literature review published in 2019[3] by Casino et al. [22] lists multiple papers that deal with on blockchain applications for financial assets[4], and could therefore be interesting for the purposes of this thesis:

- Fanning and Centers [68] shortly mention the ongoing efforts of different banks in asset tokenization without going into any detail.
- Peters and Panayi [149] discuss different potential applications of blockchain technologies in a banking context. Among others, the possible usages of the technology in the trade settlement process are described in theory.

[1] General blockchain applications studies were used as there are no fitting systematic literature reviews more specific to the topic of this thesis, such as reviews of the smart contract state of the art.
[2] Over 80% of the papers reviewed in [205] focused on the bitcoin system.
[3] The formal systematic review in this study was conducted in April of 2018.
[4] Another systematic literature review that was also considered, but did not feature interesting papers for this thesis due to its focus on finding the most cited papers rather than sorting them by application area is the review by Xu et al. [204]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

- Nijeholt et al [136] describe a factoring[5] framework utilizing distributed ledger technology.
- Paech [145] very thoroughly describes the juridical challenges that blockchain technology faces when applied to existing financial frameworks. He argues that currently available blockchain systems do not have the capacity to truly comply with existing legal frameworks and will either have to be adapted or newly created with legal issues in mind specifically.
- The listed Report by Oliver Wyman and Euroclear [139] also describes the general potential of blockchain technologies in capital markets without concrete examples.
- The last paper that was identified as dealing with financial assets by Casino et al. is by Wu and Liang [203]. It describes the implementation of a prototype for an interbank trading application for the China Foreign Exchange Trade System. The main focus of the paper is however on the existing situation and potential future additions rather than the actual prototype.

As this list shows, it still holds, that while there certainly are academic works in the general problem space of blockchain applications in finance, concrete implementations and especially works that focus on smart contracts and asset tokenization are rare. Reasons for this lack of concrete projects in scientifically peer-reviewed papers might be that due to the rapid developments in this problem space some might aim for faster publication or try to monetize their projects.

Apart from the search guided by systematic mapping studies, a more specific systematic search was also undertaken to find scientific works that have a similar research direction as this thesis to use as a basis for the state of the art chapter. The search engines Google Scholar[6], Scopus[7], the IEEE digital library[8] as well as the CatalogPlus library system[9] were used. Search terms that were used as a starting point are "blockchain" and alternatively "smart contracts" combined with "securities", "banking" and "trading" by the respective conjunctions for each search engine ("AND", "+"), as well as the search terms "asset tokenization" and "securities tokenization". Some of the results that were deemed as potentially interesting were also used as a basis for further searches (snowballing effect). As the results were, similarly as observed in the analysis of the systematic mapping studies, mostly focused on potential applications of the technology without presenting practical approaches, the searches were augmented with keywords like "application", "prototype" and "proof of concept".

Considering the results of these searches and related searches for applications of the technology in regular search engines, it was decided to split the state of the art section into two main sections. First of all, a very broad overview of currently available smart contracts and DApps and their respective fields of application is given. Then, approaches to asset tokenization developed in science and industry are explored, leading up to potentially existing solutions to security equities collateralized on blockchains, narrowing down the state of the art overview to the exact topic of the thesis.

3.2 Smart Contracts and DApps

This subsection aims to give an overview of state-of-the-art smart contracts and DApps, to show what kinds of tasks can be accomplished by such systems. To this end, smart contract systems and DApps are grouped in broad categories loosely based on the taxonomy in the Empirical analysis of

5 Factoring is a financial instrument that is used to insure small businesses against late or missed payments by customers.
6 scholar.google.com
7 scopus.com
8 ieeexplore.ieee.org
9 ub.tuwien.ac.at/catalogplus/utw_info_catalogplus.html

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

smart contracts by Bartoletti et al. [10] as well as the categorization in the Ethereum whitepaper [19].

3.2.1 Wagers

Systems in this category include all kinds of applications, where two or more parties make a wager about some type of event. If this event can be emitted to the blockchain via a centralized or decentralized oracle, a smart contract can be used to make fair, transparent bets.

In their simplest form, these kinds of smart contracts are simple gambling platforms. A bet is placed on the outcome of an event, and the winnings are paid out according to the result.

A more complex take on wagers in smart contract systems are insurance systems. As insurance is, at its core, just a bet between the insured and the insurer that an adverse event won't occur, they can be easily regulated using a smart contract, that pays out automatically, once a certain event is published by a premeditated oracle.

An example of such an insurance system that is especially well supported and thought through is the *etherisc* protocol. This aims to enable the creation of customized insurances for all kinds of events. The general process works as follows: The insureds select an insurance and pay the respective premiums. Part of these premiums is used to pay for sovereign workers providing the products and oracles etc. These premiums are put in the risk pool inside a smart contract, from which eventual insurance claims are paid out automatically when the event that was insured against occurs. For cases in which the specific risk pool for one insurance product depletes, there is the possibility for democratic reinsurance in a pool filled by sovereign investors that stake tokens for a fixed amount of time and get reimbursed at the end of their staking [54]. There are multiple products already built using this protocol, such as crop insurance and social insurance. Etheriscs automatic flight delay insurance is even already licensed according to their website. In this use case, flight delays are grabbed from airlines APIs and the reward is instantly paid out.

Another complex version of wagers in smart contracts is the implementation of *Prediction markets*. They combine the aspects of betting on an event, insuring to hedge risks and a probability of the event happening derived from the willingness of the market participants to bet on it. In general, this works by letting anyone create an event prediction by offering a collateral token. This event prediction creates a token for each possible outcome (e.g. two tokens for a binary choice). These tokens can, in turn, be traded via a specifically designed automated market maker, where the event creator earns a fee on each trade. The trading ratio between the outcome tokens is the current probability estimation by the collective of the market participants. After the event has taken place, the tokens representing the false outcomes are worth nothing, the correct tokens are worth 100% of the initial collateral token. This incentivizes actors that believe they know more about the event than the rest of the market participants as indicated by the current market valuation to buy the tokens they deem undervalued, thus driving the market price for these tokens up, precising the estimation of the prediction market further.

Two of the most prominent protocols that implement prediction markets in Ethereum are Augur [150] and Gnosis [106]. Augur has a lead in popularity according to the current stats on stateofthedapps.com[10] [178]. But neither of those has really taken off yet, as they are still under development. One of the major differences between these two systems is how the result of an event is verified: While gnosis uses predefined oracles, which makes resolvement faster, Augur uses a decentralized approach letting all non-betting market participants decide the result of an event, making it more decentralized but slower.

[10] As seen on 05/10/2019

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Some issues that these applications of smart contracts face:

Regulation Most of these applications could be regulated as gambling by authorities, making legal access to these markets difficult.

User bases As many applications, the latter two need large user bases to be able to function properly, which they do not possess as of now.

Scaling (This applies mainly to gambling and prediction markets, not so much insurances) Fees on the Ethereum blockchain are comparably high and might dissuade small transactions like small sports bets.

3.2.2 Notary Uses and Intellectual Property

This types of smart contracts deal with timestamping and proofing ownership of intellectual property. Of course, this application is not unique to smart contracts, as such timestamping services could also easily be implemented using bitcoin, or even plain newspapers[11]. But these services become inadequate when automated royalty payments or other advanced features are needed.

There are multiple different approaches to such a system, from simply proofing a copyright on a digital asset such as monegraph does [100], to complete music distributing systems, as described by Gheorghe et. al in [76]. Some of those feature micro metering, micro monetizing and dynamic pricing, others try to capture the whole existing ecosystem of existing music production, by integrating publishers, labels, other performers etc. in a coherent system. Such systems can allow a content creator to be directly paid every time the content is used, as the payment could be directly made on the underlying blockchain.

Another extremely ambitious project in this space is Qravity [160]. It claims to provide a system that facilitates the creation of all kinds of creative ventures. It especially focuses on enabling project management tasks and cooperation between multiple content producers to create a common product, including the precise percentual monetization of the product according to the workload of each content producer.

Although there are many ideas and projects in the field of intellectual property as presented, there has yet to be a project to capture a significant market share [76].

3.2.3 Gaming

Gaming is another big driver of smart contract development. Most notably, the popular cryptokitties[12] smart contract first popularized the notion of a game on a smart contract base. It uses non-fungible tokens, that represent cats with unique features that can be interbred in order to create new tokens/cats. Shortly after its release, this smart contract alone accounted for 25% of transactions on the Ethereum network leading to significant abnormal loads on the network [30]. There are also a lot of other games, some of them similar, some focusing more on gambling, but for the purposes of this overview, this short excursus shall suffice.

3.2.4 Wallets and Identity Management

Another pretty standard application of smart contracts is the implementation of complex wallets. Normally in the context of Ethereum, a wallet describes client-side software that handles the private

[11] As it was done already in 1991 by Haber and Stornetta by publishing hashes of digital documents in the New York Times [83].

[12] cryptokitties.co (visited on 05/10/2019)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

keys of the user. But for advanced features of a wallet such as multiple participants, a smart contract can be used. Such contracts can for example act as trust funds regulating their output, or be used to create safe two-factor authenticating wallets for multiple participants, as demonstrated by Homoliak et al. in [90].

A related application of smart contracts is identity management. As an (almost) immutable ledger, distributed blockchains seem to be predestined for identity management. An example of innovation in this sector is the proposed Ethereum standard ERC-725 by Fabian Vogelsteller [194] that provides a standard interface for a smart contract that controls the identity of a user. Users can implement their custom smart contracts, that can, in turn, be utilized by DApps supporting the ERC-725. Using the additional standard ERC-735, claims can be added to describe the properties of the entity holding the identity.

3.2.5 Supply Chain Management

A combination of IoT devices tightly integrated with a supply chain utilizing smart contracts on a blockchain has the potential to highly impact supply chain management according to industry professionals interviewed in a study by Korpela et al. [109].

There are numerous projects using such an approach in industrial as well as research settings. A very interesting example is modum[13]. It presents supply chain management with integrated sensors to ensure environmental parameters (e.g. humidity, temperature) for pharmaceutical products in transport. The parameters are recorded by IoT sensors and in turn immediately evaluated by a smart contract, making eventual violations of limit values visible to controlling parties [15].

Another approach is the provision of provenance data of products for consumers, as the startup provenance[14] does. It aims to provide complete immutable data on the provenance of a product so that a consumer can trace back the exact origin of e.g. a can of tuna [159].

3.2.6 Libraries

Another category of smart contracts deployed on the Ethereum blockchain that cannot be ignored is libraries. These types of contracts implement common operations like secure mathematical operators to be utilized by other smart contracts.

While not that scientifically interesting on their own, they are an important basis of smart contract development, and have to be especially considered in the face of security implications, as was explained in the Vulnerabilities section under Access Control (see 2.4.5).

3.2.7 Financial Services Contracts

This section arguably deals with applications that are a natural fit for Ethereum, as the possibility of token creation and the codification of financial contracts lend itself to these types of operations.

Utility and Security Tokens A very general use-case scenario for smart contracts is the issuing of tokens by a startup as a way to raise initial funds. This process is called an *Initial Coin Offering* (ICO - analogous to the Initial Public Offering, the stock market launch of a business) for so-called *utility tokens* and Security Token Offering (STO) for *security tokens*. Utility tokens can, in turn, be exchanged for services in the related DApp of the Startup after they have been launched and therefore constitute the delayed purchase of a service. Security

[13] modum.io (visited on 05/10/2019)
[14] provenance.org (visited on 05/10/2019)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Tokens, on the other hand, are similar to securities and partly constitute a kind of ownership in the issuing startup. This, in turn, comes with all the regulatory issues that securities face, as those tokens will most likely be treated as such under MiFID II [60] in Europe [94]. The practice of security tokens will be further explained in detail in section 3.3.2.

DAOs Decentral Autonomous Organisations (DAOs) are leaderless organizations, whose governance rules are encoded in a smart contract, and can be collectively controlled by holders of corresponding tokens. One of the most widely known DAOs was "The Dao", a decentralized crowdfunding organization, whose smart contract was hacked and drained of some of its capital, as described in section 2.4.5. But there are also many other DAOs running on the Ethereum platform, such as the MakerDAO, which governs a decentralized stablecoin, the DAI, that aims to overcome the centralization issues of other stablecoins such as Tether [120]. Another popular DAO system is DaoStack. DaoStacks intent is to create a collaboration and governance system for DAOs, to facilitate the development of such organizations and improve collaboration between those [32].

Banking applications Smart contracts do not only have a place as a replacement for traditional banking institutes, but can also be used by them to their advantages. Application fields may include Clearing and Settlement [29], KYC, mortgages or bonds [70]. An example for a "real-life" application of smart contract is the 150 Million Euro loan that was arranged between the Spanish bank BBVA and the Porsche Holding GmbH in 2018. The bank aims to use the Ethereum blockchain to streamline their loan processes and minimize costs [101].

Another application of smart contracts is the IIN Network by the bank J.P.Morgan. It is a network of 344 banks,[15] that aims to facilitate the resolvement of "stuck" cross-border payments by providing information in between the banks, utilizing a permissioned ledger on the Ethereum Fork Quorum [97].

From loans over clearing to cross-border payments, it seems only logical for the next step to be the recording of security ownership via smart contracts. In how far this is already being scientifically explored will be discussed in section 3.3.

Of course, not all smart contracts and DApps that currently exist or will be developed will be able to fit in the categories presented above, but they give an overview over some of the application domains of smart contracts and what the possibilities in the current state of the art are.

An view of most of the currently live DApps can be found at stateofthedapps.com (visited on 05/10/2019). It gives live usage statistics as well as development activity overviews, to enable users to gauge the usage and activity of a DApp.

But it also has to be conceded that while the wide range of displayed fields may make smart contracts seem like a jack of all trades, one could argue, that some projects utilize the media hype generated by blockchain technology as a crowd puller. In these cases, distribution for distributions sake might be used, without really profiting off it in a technological way. Future implementers also have to be wary of this potential fallacy.

3.3 Asset and Securities Tokenization

Tokenization describes the process of creating tokens as a representation for real, tangible assets. In this section, the different approaches to tokenization in the current state of the art are explored. This is achieved by explaining the respective concept, then presenting innovative implementations of the

[15] At the time of writing according to jpmorgan.com/global/treasury-services/IIN (visited on 05/10/2019)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

concept, and discussing interesting features and approaches, that may be important in creating the securities tokenization system of this thesis.

First of all, the tokenization of asset classes other than ownership in corporations is explored. Then, systems dealing with security tokens will be shown, and finally, the current implementations for tokenization of securities will be analyzed, narrowing down to the exact use case scenario that is to be approached in this thesis.

In news reports and also in scientific publications, the terms "security token" and "tokenized security" are often used interchangeably. But from a conceptual approach, they are two quite different things. Security tokens are tokens that represent ownership in a company that can only be owned via possession of such tokens. Tokenized securities, on the other hand, are traditional preexisting securities, that are represented, and distributed by tokens [2].

3.3.1 Tokenization of Tangible Assets

This section takes a look at the possibilities for tokenization of tangible assets other than equity. This means it especially deals with assets that are normally nonfungible i.e. not easily divisible and not easily acquirable and storable for low volume investors. Examples for such assets are real estate properties, fine art or more unconventional assets like luxury cars.

Tokenization makes investing in these kinds of assets way more efficient and accessible, in turn increasing liquidity by enabling more market participants to join. In addition, middlemen can be eliminated, transfer of ownership is immutably stored and the whole process is transparent for all sides. [196]

One example of such a tokenization platform is Maecenas. It is a perfect illustration of an effort to break up markets previously tightly controlled by few players and democratize them. Specifically, it deals with the fine art investing market. An estimated $65 billion of fine art is traded each year, of which 80% is captured by the market-leading auction houses Sotheby's and Christie's. This oligopolization enables them to take a cut of up to 25% from buyers and an additional cut from sellers [118]. As currently, even existing fine art funds require large entry fees, it is virtually impossible to invest in this market without spending a large amount of money.

Maecenas aims to break up this monopoly and make buying art accessible even in smaller quantities. To this end, a platform utilizing smart contracts and a custom ERC20-token (the "ART" token) on Ethereum is built. On this platform, users can buy shares of art pieces, split in a predefined number of shares. These shares themselves are also tokens, that can be resold and traded freely. The piece of art itself stays in an audited central vault, or with the original owner, who can only sell up to 49% of the artwork, so that a nominal primary owner remains. It can also be leased to galleries or museums, generating income for the token holders. An overview of the system can be seen in 3.1.

A very important aspect of creating an asset token are legal issues, in order to bind the tokens to the actual asset in a juristically claimable way. Maecenas met this requirement by creating a special purpose vehicle (SPV), a separate legal entity with the purpose of protecting the rights of the token owners, e.g. should the company Maecenas go out of business [118]. As the founder of the company put it: "Tokens representing fractions of assets are worthless without a legal contract that gives those tokens bearer rights over the underlying assets." [75]

Apart from creating tokens, which in itself is not that interesting from a scientific standpoint, the Maecenas system also implements a compelling concept that could potentially be used for the trade of some kinds of securities on the blockchain. This concept is the so-called "dutch auction" which is also used in the sale of US treasury securities [26]. In this process, each buyer may submit secret bids on how many shares to what share price the bidder would buy. After the bidding process is over, the bids are revealed. The highest price, that sells all shares is then instated as share price. I.e.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

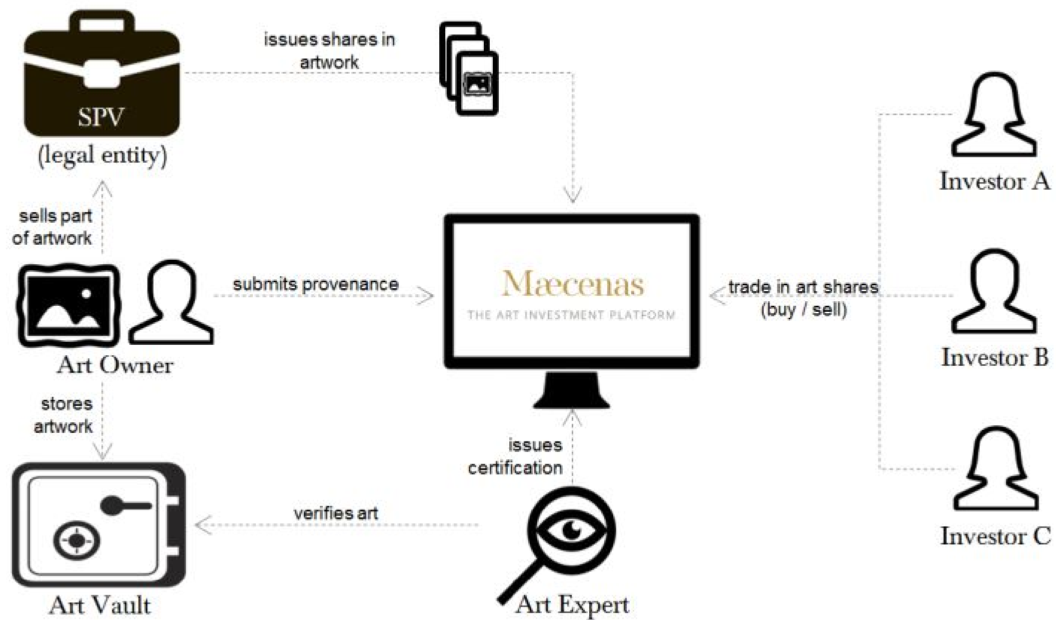

Figure 3.1: Overview of the Maecenas Art platform.
Source: Maecenas White Paper [118]

everyone that set a higher share price, also only pays the universal share price, but they get their shares first. Blockchain technology lends itself to this process, as it is easily possible to implement this type of auction by enabling users to bid by publishing hashes of their bids, which can be compared with the biddings after the end of the auction[16].

The exact functionality can be analyzed using the smart contract code for the dutch auction [20]. All hashes of the bids are stored in the smart contract until the auction ends, at which point the final share price is calculated, and all losing bids tokens are returned to their respective owners. An interesting aspect is that resource intensive operations are batched in order to prevent out-of-gas exceptions.

Of course, this project also has shortcomings. It claims, that the ART Token mitigates currency risks, but as it in itself is a tradeable token with a potentially fluctuating price, sellers are exposed to the currency risk while the auction runs. Another point of criticism is, that at least the smart contract for the dutch auction process that was found during the review of this project was designed in a very centralized way, as all writing methods are designed to be called from the owner of the auction only. This means, that all bids have to go through the owner of the contract, partly negating the perks of decentralization and restricting users.

While this makes development and possibly the interaction with the system for users easier by sending them through a centralized web application, it forces the users to trust the application more than would be necessary for a truly decentralized solution. But this does not negatively affect the principal goal of the system, namely to tokenize pieces of fine art in a divisible way, and some lessons can be learned from the smart contract implementation.

[16] But it is not implemented exactly by direct user interaction in Maecenas, as will be discussed in the next paragraphs.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

This and other sealed bid auctioning systems on blockchains have also been the issue of research, such as the proposal for a partly privacy preserving sealed auction contract on the Ethereum platform by Galal et al [72].

Another example of an asset tokenization platform is Propy[17]. It seeks to improve cross border liquidity for realty investment. Their plan is especially to combat differing standards in land registry records between countries by creating a platform to connect investors with brokers. This also doubles as an asset store for real estate property, that initially mirrors the local land registries to provide easy cross country trading. Their ambitious and quite possibly unattainable goal is to establish their blockchain registry as the official ledger of record for some jurisdictions [156].

Due to this enthusiastic goals, the combined smart contract software suite produced by propy is also quite large and well thought through [157]. Another positive aspect of the project is, that it aims to be fully decentralized in the future, contrary to Maecenas. It is also more actively maintained, considering recent and recurrent Github commits.

Problems of this project are again mostly of legal nature, as the adoption of their system as the main official land registry many different countries seems quite far fetched. Another issue is that a lot of the steps currently have to be done off chain (like the purchase agreement signing, the title report, etc.), as can be seen when reviewing contracts like the MetaDeedContract [158].

In general, the real estate tokenization market has quite a lot of contenders, such as Bitrent [13], which aims to foster investments in real estate projects during construction or Alt.Estate [3], which, in a similar fashion to Maecenas, tries to make the property investment market more liquid by enabling the trade of microshares of estates. Other competitors in this market include Atlant [180], Blocksquare [151], and Deedcoin [34].

To sum it up, legal problems make up the lion's share of the concerns regarding asset tokenization, and projects that deal with asset tokenization must find a way to deal with those.

3.3.2 Security Tokens

The next logical step towards tokenizing securities is the practice of issuing security tokens in Security Token Offerings (STOs). As mentioned before, the semantics in this area are often mixed up. For the matters of this section, the term Security Tokens in the narrow sense will be defined as tokens emitted in a Security Token Offering, that constitute shared ownership in a company, that has not emitted any traditional securities, as defined in the infobox in section 3.3. But especially considering other sources in media and literature, it has to be kept in mind, that the term "Security Token" can also be a superset of both security tokens and tokenized securities, and some sources even define tokenized assets as security tokens.

In general, current projects in this category aim to provide founders with a lawful way to initiate STOs without the hassle of dealing with implementation details and legal compliance themselves. The general upsides of security tokens as well as tokenized securities over equities using a traditional ledger are:

(Near) instant settlement A trade of security tokens is just as easy as an Ethereum transaction, which according to ethgasstation.info currently[18] has a duration of under five minutes for medium transfer costs of 1.7 cents, or under two minutes for transfer costs of 3.3 cents. In comparison to this, the transfer time of traditional securities are between one full working

[17] propy.com (visited on 05/10/2019)
[18] Visited on 05/10/2019.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

day and a whole week, depending on border crossings and jurisdictions [23], resp. two days for Target-2 securities within EU member states [65].

Fractional ownership Just as asset tokenization enables fractional ownership of other tangible assets, security tokens facilitate ownership in smaller fractions as possible with traditional securities. This enables investors to truly diversify their stock holdings by holding infinitesimally small amounts of STOs of many companies instead of e.g. investing in a synthetic abstraction of the intended holding [42].

Higher liquidity Through permitting faster trades and bringing in more investors with smaller holdings, the liquidity of the asset can be improved [123].

Possibility included compliance As security tokens are based on smart contracts, it is possible to include compliance measures direct in the smart contracts code, potentially further saving costs. In addition, as all transactions are persisted in an immutable ledger, legal revision tasks are inherently supported [123].

Possible reductions in cost and trust by removing intermediaries Most proponents of security tokens argue, that by removing intermediaries, administrative costs can be saved. In addition, this removes the requirement of trusting another counterparty [123].

Downsides and potential complications are that currently, the legal status of security tokens is not 100% clear in all jurisdictions. In addition, while compliance can included within security tokens, it might be a considerable effort to do so. And of course, as it is always the case with new technology, issuers are exposed to the risk that the technology is not adopted and therefore, the liquidity and market price of their assets may fall greatly [186].

Polymath

One of the most popular and well-documented platforms that implement and facilitate security tokens is polymath[19]. It enables the creation of security tokens in STOs on the Ethereum chain. One of the most interesting aspects of polymath is its expandability.

The core functionality of the polymath platform revolves around the ST-20 token. These tokens include, among other features, the possibility to restrict how they can be traded in order to enable issuers to comply with securities laws, as well as KYC and AML features. Coming from the implementation of the ST-20 token, the polymath team has also developed an Ethereum token standard proposal, that is implemented by the ST-20 and can also be implemented by other projects. This standard, which was created in cooperation with the Ethereum developer Fabian Vogelsteller is the ERC-1400 standard [41].

Now, how are expandability and modularity brought to this token? To explain this, a quick overview of the four-layered architecture of polymath as demonstrated on their tech blog [179].

Layer 1 - Ethereum The polymath system relies on the Ethereum blockchain as basic infrastructure to store the state of the security tokens and the smart contracts that make up their functionalities.

Layer 2 - ST-20 Token The basis for all security tokens that are created with polymath, enables the token creator to implement trading restrictions.

[19] polymath.network (visited on 05/10/2019)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Layer 3 - Polymath Token Studio This DApp that can be used to create ST-20 tokens. Here, security providers can create and customize their tokens and define transaction rules. In addition, the DApp offers direct inclusion of external providers that might be needed in an STO launch, such as KYC providers, custody services or law firms.

Layer 4 - Marketplace This is where the modularity of polymath really shines. The marketplace is a collection of all modules that can be added to ST-20 tokens, not only produced by polymath, but also by third-party developers. These modules can be dynamically added and removed from tokens, and can, for example, be used for different methods of dividend payment, voting, transfer restrictions and so on.

The aim to help third-party providers develop modules for the polymath platform also motivates the team to maintain a well documented open code base, which proves beneficial for the analysis in this thesis.

The platform is also continually being updated, first with version 2.0 that rolled out on November 22nd, 2018. In this release, many problems, that are relevant to the tokenization of securities, have been addressed. First and foremost, the possibility to create STOs pegged to USD via a stablecoin was created, as risk-averse issuers want to mitigate the currency risk associated with purely raising funds in cryptocurrencies. The concrete implementation can be seen in the file USDTieredSTO.sol on their public Github repository [153]. It uses an individually definable stablecoin as tether, as well as definable oracles to calculate the current exchange rate in all transfers. Other adaptations include features for juridical compliance, such as support for tax withholding and forced transfers that enable issuers of tokens to force token holder to transfer them, e.g. in order to comply with juristic rulings. Especially the latter might raise some concerns from investors as it practically gives issuers absolute control over their created tokens [164]. Version 3.0, released on July 29th, 2019 also brings in-place upgradeability of contracts[20]. This means, that from now on when new logic needs to be deployed to the smart contracts of the system, data does not need to be migrated. This is achieved via proxy contracts that hold all the data, while the logic is stored in separate contracts, that can be switched if needed [38]. In addition, this version brings full compatibility with the ERC-1400 token standard.

The economics of the polymath platform itself are managed via a custom ERC-20 Token, the "Poly" Token. Issuers pay registration fees to the company behind polymath, as well as their fees to third party service providers, like KYC providers. Smart contract developers that contribute modules to the marketplace are also remunerated in Poly tokens.

Next to the forced transfer issues, other possible criticisms of the platform include the possibility of price fluctuation of the Poly tokens. If the tokens price is too low, polymath loses this source of revenue, if it is too high, STOs might become too expensive for users. Granted, currently neither possibility seems threatening, as $59 million where raised in the initial token sale, and on the other hand, the price for poly tokens currently hovers at about 2 cent[21]. This sets the price for a token launch at about €5 [152]. Another worrying circumstance of the project is, that there is currently no officially endorsed whitepaper. The original whitepaper [110] can be found, but it is not presented on the polymath website itself. This may just be due to the fact that the whitepaper is just not relevant enough anymore, due to the evolution of the product, but in any case an official updated version would be beneficial for interested parties and researchers.

[20] The basis of this upgradeability can be seen in the following commit: [79]

[21] According to coinmarcetcap.com on the 4th of October, 2019.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

But regardless of these points of criticism, polymath is a remarkable project, with many interesting approaches to common problems in this problem space, such as the implementation of a common interface for securities, upgradeability of smart contracts and tethering to stable coins.

Securitize

While polymath is one of the most scientifically interesting projects in this problem space due to its openness, there are also other projects aiming to provide security token offerings for companies trying to raise funds. One of those is Securitize[22]. In comparison to polymath, securitize aims to be more of an all in one solution with dividend payments, voting rights, etc., without such a big focus on external developers or providers. It is especially geared towards companies that want to start a security token offering on their own, without going through another platform. This is enabled by their offer of a white-label platform, that can be seamlessly integrated into the website of the issuing company.

Securitize was initially implemented as a possibility for the STO of the first tokenized Venture Capital Fund, SPiCE [125]. The token of the SPiCE VC also serves as the first proof-of-concept for the securitize system.

The technical basis for the securitize system is the "DS Protocol" (Digital Securities Protocol) which was described in a whitepaper in 2018 [37]. This paper lays out the envisioned ecosystem in great detail. It consists of three core elements: The DS Tokens, the actual security tokens that implement a typical ERC-20 interface. DS Apps, that manage lifecycle events, such as voting or dividend issuing. And finally, DS Services which make up the basic infrastructure of the system. The DS services are:

The Trust Service Authorizes actors outside of the system, such as exchanges or new DS Apps.

The Registry Service Stores hashed versions of KYC information on chain so that these can be verified by the system.

The Compliance Service Handles the compliance requirements of the token trading, as such is called before each transfer of a DS Token, and can be defined individually by token issuers.

The Comms Service Communicates events of the digital securities lifecycle such as liquidity or governance events to token holders.

An overview of the system and its components can be seen in the architecture diagram in figure 3.2.

Another interesting feature that is described in the whitepaper is the so-called Ready For Exchange off-chain API. This API is supposed to enable real-time information gathering for external exchanges regarding user eligibility and KYC information. But although this API should be open sourced according to the whitepaper, no mention of it can be found anywhere on securitizes site or blog posts, so it might not actually be planned in this manner anymore. This is also one of the major flaws of the securitize system from a scientific viewpoint: No code of the core system is published anywhere. This makes gauging the activity of development and the maturity of the codebase difficult.

To sum it up, it can be said that while the whitepaper extensively covers the architecture of the system, the implementation of the system itself remains a mystery, due to being closed source, and while there are interesting ideas in the paper, none can be truly assessed without further information.

[22] securitize.io (visited on 05/10/2019)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

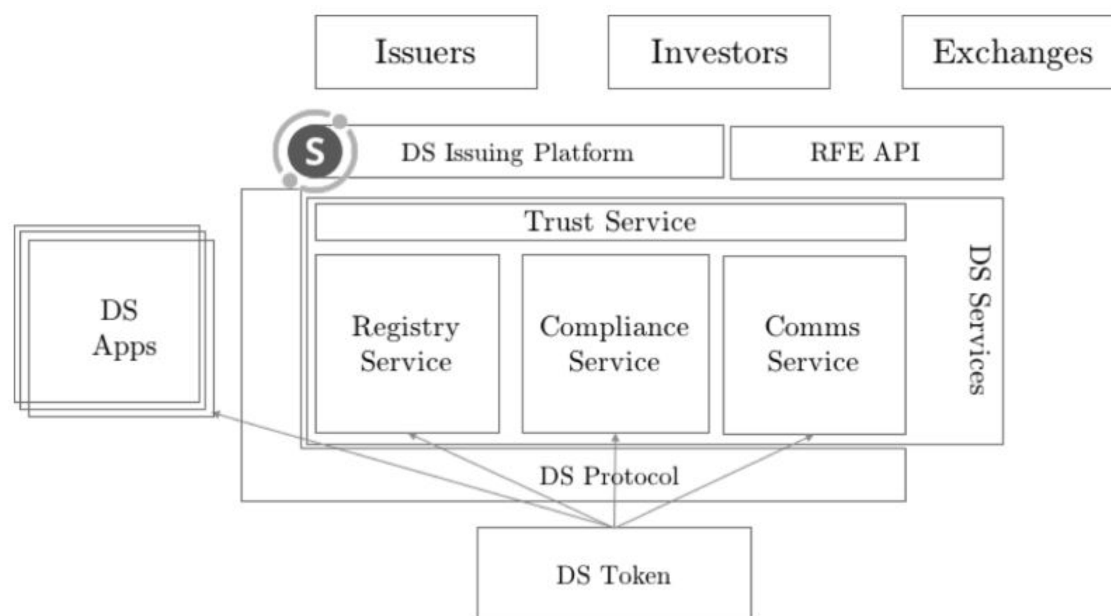

Figure 3.2: Overview of the DS Protocol
Source: DS Protocol Whitepaper [37]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Harbor

The third system that will be evaluated as part of the security tokens is harbor and it's R-Token (Regulated Token). The general architecture is similar to the other described solutions, an R-Token uses a fixed service registry to query for a regulator service, that decides whether a transaction is allowed or not. However, one significant difference is, that in harbor, the regulator service is very closely monitored and controlled by an off chain Trade Controller. Because Harbor relies on a whitelisting approach, each vetted investor and the corresponding permissions must be mapped and decided off-chain. This takes a lot of responsibility out of the system and is one of the reasons why at least the published version on Github [84] is considerably less complex than e.g. the polymath implementation. But this reliance on a trade controller has a rationale behind it: One major other difference between harbor and the projects discussed before is, that harbor is specialized on private equities, i.e. equities of companies that are not listed on any public exchange, or other illiquid assets like private Real Estate Investment Trust (REIT)s [185] or in their newest version even shares of sports teams [206].

But still, after an analysis of the code on Github, it remains that the system has substantially fewer features than the others that were analyzed. While it could well be that there is more development going on behind the scenes, it stands to reason that harbor relies more on its specification on private securities than on technical excellence to gain customers.

Summing it up, the field of security tokens is very much active and populated in the fintech scene, with many different companies. These projects, while numerous, in general all depend on the business model of offering STOs and related services to companies that search for funds. But apart from these projects and their whitepapers, there is little to none scientific work being done in this field, possibly because it is overshadowed by the more popular method of ICOs, as well as due to its relative recency.

All of the analyzed approaches have strong centralized elements with control over the environment being handed to either the issuer of a token or the company behind the system. While this may be worrying from a puristic decentralization favoring point of view, it may simply be nigh impossible

at this stage of legislation to implement a fully decentralized STO platform. Especially, since a legal party representing the platform is often needed, and the legal status of fully decentralized autonomous organizations is not clear in most jurisdictions [95]. So it just seems logical, that a certain amount of central control needs to be maintained in such systems, especially considering the possibility of upgrading contracts. Of course, this approach carries the risk that companies have to ask themselves, whether there is real value in their project from decentralization or if the blockchain label is just attached to them as a marketing pseudo-concept, that is not truly implemented [111].

Another issue that runs through all of these projects is, that to issue security tokens, a general standard of security tokens must be agreed upon. So far, the lowest common denominator is the ERC-20 standard for general tokens, but as this interface describes a general purpose token, and therefore lacks security specific functionality, it seems unlikely that this will be the standard for security tokens in the long run. Therefore, a dedicated standard for security tokens is needed. The STO platforms also recognized this problem, and this resulted in each platform creating their own standard including ST20 from Polymath, R-token from Harbor, DS Token from Securitize, and many others that were not described in this section.

One standard, that might stand a chance of becoming the norm is the previously mentioned ERC-1400 family of standards, that represent different smart contract interfaces and libraries to interact with them [41], as it is supported by multiple different companies, including polymath, who took part in developing it, and securitize. It is very feature rich, but at the same time, this complexity is also a point of criticism. Of course, only time will tell, which standard, if any, will be established.

3.3.3 Tokenized Securities

Tokenized securities are in a way the combination of the two above approaches and the core field of this thesis. The term describes securities, that may already exist in the traditional legal framework, that are mapped to tokens, and from this point on, the legal ledger on who is the owner of the security matches with the owner of the corresponding token. As Security Token Frameworks might theoretically also be used to tokenize traditional securities, and due to the newness of this field, the borders between these two approaches are often blurred.

In general, the pros of tokenized securities are shared with those of security tokens above namely improved liquidity, faster transfers and the possibility for overhead reductions [123].

One advantage of tokenizing securities over security token offerings is that the former deals with securities, that are already embedded in the existing legal framework, and therefore have more clearly defined boundaries. Of course this in turn also means, that tokenized securities have to deal with the existing legal rules, and there is no imminent possibility of legislation, that may be softer than the existing legal scaffolding.

So what kind of security tokenization options are currently on the market?

Overall, security tokenization systems can be divided into three categories, each of which will be presented here by means of an example.

- Self controlled tokenization
- Pseudo securities trading
- Large scale secondary tokenization for retailers

Self-controlled Tokenization

Self-controlled tokenization projects have the largest overlap with those described in the previous section. These projects enable the owners of companies to tokenize their assets, but not necessarily

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

in the context of STOs, but also at a later stage in the lifecycle of a company. One project in this category is Tokeny, another end to end tokenizing platform and their security token standard T-Rex [187], which is reasonably implemented, but does not bring any technical innovation to the table, that hasn't been analyzed in this thesis via other projects. Another project is the closed-source, ledger agnostic Securrency system, which also places special focus on managing cross chain investor identity [67] [148]. But one of the most interesting projects in this space, that will be analyzed further, is neufund[23] [134].

Neufund is a Germany based investment platform, that is owned by its customers, where startups and established companies can issue tokenized securities. In a legal sense, the securities are held by a "Transparent Trustee", a legal entity specific to this purpose, as long as there is no stable legal framework for tokenized securities. In German and EU jurisdictions, there is an additional "nominee", a natural person that represents the interests of the shareholders of tokenized securities towards the issuer. There have also already recently been successful equity tokenizations on neufund, like the tokenization of the company brille24.de[24].

One major difference between this project and others like polymath is that there is not so much focus on a security token standard, as the tokens can only be traded if the issuers explicitly permit it [108]. But looking at the code, there seem to be no safeguards in place that would prevent free trading of the equity tokens. The EquityToken.sol code as well as the underlying BasicSnapshotToken.sol in their public Github have a distinct lack of any preset checkers for eligibility of trading, as it was implemented in other systems [135]. For initial investors into the neufund system itself, the identity and eligibility is checked via the IdentityRegistry [132], but not so much for any trades of the tokenized securities in the system. The only possibility of regulating such checks is shifted to the issuers, that may subclass custom EquityTokenControllers to limit transfers [131]. This raises the question of how the trading of equities, arguably one of the most important features of such a platform, is planned to be implemented and regulated.

The whitepaper, however, [134] claims, that KYC will be mandatory for all trades, so it seems that the second-hand trading of tokenized securities is simply not possible yet. This also accords with announcements by neufund, that they are just planning on partnering with exchanges in order to enable trading of the created equity tokens [133] [11]. This means that it remains to be seen if and how trading restrictions will be implemented in this system.

Another decision that could be criticized, is that in neufunds announcements and in the whitepaper, there is more of a focus on the internal token economy of neufund token owners, than on the actual investment platform, and equity token functionality. This could, of course, be explained by their need to attract investors in their system, but it nonetheless raises the question of how much emphasis is placed on the core product of the investment platform and the tokens, and whether enough effort is behind this element of the project.

Regardless of the criticisms of this project, there are also many interesting approaches and positive qualities. As evidenced by the citations above, the source code of the project is public on Github, and it is also currently being very actively maintained and developed. While it uses no public token standard tailored for securities, another compelling standard is used, namely the ERC-233 standard. This standard extends the ERC-20 token standard with functionality that prevents funds from being sent to contracts that can not deal with being transferred tokens, which would be lost in the process [36].

[23] neufund.org (visited on 05/10/2019)
[24] blog.neufund.org/successful-eto-thank-you-to-our-community-ae8d97ea6c95 (visited on 05/10/2019)

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

While these kinds of projects are not exactly what this thesis aims to implement, there are nonetheless some lessons to be learned from these, especially regarding the implementation of such platforms and approaches to legal problems.

Pseudo Securities Trading

These projects boast the ability to trade securities on a blockchain, but rely on the established infrastructure of other exchanges in order to truly execute the trades. Of course, one could argue, that this is merely an intermediate step owed to the current laws, but nonetheless, it seems that for some, the blockchain is merely a marketing tool.

One of these projects is currency.com. It claims to be first true platform to trade traditional securities on the blockchain, but is basically just a frontend for capital.com, stationed in Belarus, according to the whitepaper due to the progressive securities laws [183]. The whitepaper does not really give any clear statement on technological innovations, just that the securities will be managed via a blockchain. There is also no public repository that could be analyzed for innovation, so in general, the development of this project is quite obscured.

A very similar project is dx.exchange which offers digital stocks, or tokens, based on actual shares bought and held by its partner MPS MarketPlace Securities.

While these kinds of projects are of no direct scientific importance, they serve as a reminder that the use of distributed ledger technology also has to be questioned, and that not all projects in this space are what they claim to be.

Large-Scale Secondary Tokenization

This is the exact core issue that this thesis addresses. It aims to provide a system to tokenize existing securities by entities other than the actual issuer of the security, so that they can be tokenized without requiring the issuers to step in. Currently, there are very few projects that openly pursue this functionality. But of course, as this topic is mainly interesting for retail banks, it is entirely possible, that there are numerous projects being developed and prototyped out of public view by banks, or by companies being financed by banks, as such a venture would not be immediately be marketed to the public, but to banking executives.

One of the few projects in this problem space that can be found is by the company Fineqia, who bought into Nivaura in 2018 for a joint creation of an equity and bonds tokenization platform. But as there is very little public information available about this venture, it can not really be said, whether this endeavor has any merit [71] [92].

But as this application of distributed ledger technology could potentially make traditional exchanges and clearing houses redundant, there also exist plans by these large industry players, that could potentially develop to a stage where they would enable large scale tokenization of securities. For example, according an article by the Nasdaq stock market [9], the stock exchange aims to utilize distributed ledger technology for private equities tracking and payment processing. Another exchange that utilizes blockchain technology is the Australian stock exchange ASX, which has started work on a replacement for their legacy clearing and settlement system. However, this system seems to be extremely centralized, allowing only ASX themselves to persist transactions, and in addition, it stores the actual transaction data off chain, and validates those on the actual blockchain, therefore not using the technology to its full potential [115] [6].

As evidenced by the low number of projects, and the even lower availability of any kind of public code, there is a lot to be discovered in this specific problem space, especially from a scientific point of view.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

All in all, it can be said that there are plenty of projects in the asset tokenization space, some reasonably successful, but so far, no clear market leader has emerged. Some of the innovators have brilliant ideas and approaches, some seem more like quick cash grabs, so all projects have to be taken with a grain of salt, and analyzed profusely. Another issue that becomes apparent looking over the presented approaches is that there are no purely academic projects in this list, due to the absence of relevant papers. This indicates either a lack of scientific interest or that in these problem spaces, industry and especially startups are the front runners of innovation because of the opportunities this technology presents.

Nonetheless, the potential of tokenizing assets is tremendous and could reshape many aspects of the financial world ([113]) by:

- increasing liquidity via a broader investor base and fractional ownership
- increasing transaction speed via automation and smart contracts
- removing middlemen like clearing houses
- increasing transparency, in the cases of a public ledger

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

4 Implementation of the Prototype

4.1 Concrete Problem Description

As concluded in the state of the art section, while there is work being done in this field, it is rarely executed in an academic manner, and there are some gaps to be filled, especially in the field of large scale secondary tokenization. If there is no possibility for retail banks to adopt distributed ledger technology to offer conventional securities at a large scale, users might be hesitant to utilize this technology due to the lack of options of securities to buy, hurting adoption rate and the general progress of technology development.

This thesis aims to close this gap in the asset tokenization space, by creating requirements for such a system and analysing what other properties it might exhibit. To this end, a prototype for a system, that can be used to trade securities of existing companies on the blockchain, including considerations of possible legal constraints, is developed.

To fulfill this ambition, the system needs to enable creation and trading of security tokens, that correspond to established token standards, while also enabling the deployers of the tokens to comply with local laws, i.e. they must be expandable and adaptable to different laws in different jurisdictions and changes to the legal framework.

4.2 Requirements

This section aims to provide requirements for a system as described above, that can be used as a basis for the development as well as for validation of the developed system. These requirements are not intended to become a full-fledged specification book, and are therefore less in-depth than some formal specifications of software products, but rather focus on providing an overview of the required features of the system, functional and nonfunctional alike.

Nonetheless, the requirements must conform to the SMART criteria, a common guide for finding objectives, that are most commonly defined as **S**pecific, **M**easurable, **A**chievable, **R**elevant, and **T**ime-bound[1] [121]. The keywords "must", "may", "shall", etc. are to be interpreted according to the standard RFC2119 [16].

The requirements are split on the one hand architectonically between the two main components, the smart contracts system and the frontend client, and on the other hand between functional and nonfunctional requirements. In addition, the functional requirements for the smart contracts system are split between requirements with technical reasoning and those with regulatory reasoning.

4.2.1 Smart Contracts System

The smart contract system is a group of linked smart contracts on a blockchain, that fulfill all core functionalities of the project. It can either be accessed via a dedicated client, as described in the next subsection, or as a standalone system.

[1] Time restrictions are not explicitly defined in the requirements, but rather implied by dependencies and the end date of the project.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Technical functional Requirements:

Deployability The system must be deployable to an Ethereum blockchain. For this, it must be written in a language that is compileable to EVM bytecode, such as Solidity, LLL, Serpent, or others that were described in the smart contract languages subsection 2.4.3 [4].

Token Creation The system must allow creation of transferable tokens. It must be possible to create additional token contracts, that differ from other tokens in order to represent different securities. All tokens must be separately configurable and tradeable individually.

ERC20 The system must implement at least the ERC-20 Standard [12], that regulates how general tokens can be interacted with and behave. This is a necessity for almost any token, so that standard exchanges and wallet software can interact with it, as it is the most commonly used standard [27].

Further Token Standards The system may implement further specialized standards or interfaces to enhance interoperability and build upon previous works. Especially general standards for security tokens are interesting in this context. To this end, existing standards shall be analyzed, to decide whether there are any that fit the needs of this project to capitalize on possible interoperability and broader adoption of the system [4]. Some of these standards have already been described in the security token subsection 3.3.2 of the state of the art chapter.

Expandable Rules The restrictions and general rules of the tokens must be expandable, by configuration as well as by expanding the smart contract code [182]. This means that on the one hand, there must be methods that can be used by operators without programming knowledge to restrict as well as relax restrictions on trading. On the other hand, there must be a prepared way to change the system to adapt to unanticipated legal changes by reprogramming parts of it.

Incentivizing The system must be able to incentivize its operator transparently so that there is a payoff for running the system [193]. Specifically, there must be a method (e.g. the transfer methods) where the user pays a small percentage or fee for the usage of this service, that can be collected by the maintainers of the system.

Dividends The system must be able to create tokens, that manage and pay out dividends. Users must be able to pay ether to the Contract, that is in turn distributed to all token holders according to their current holdings (which must not be distorted by trades after distribution). These kinds of tokens correspond to preferred stock [74].

Voting The system must be able to create tokens, that manage ballots and votes. Authorized users must be able to create ballots, that can be voted upon by token holders. These must be able to vote according to their holdings at the time of ballot creation. These tokens correspond to common stock [25].

Discovery The contracts of the system should be discoverable by addressing a single contract or a domain name service, to enable clients to interact with all contracts without storing their addresses, and still having a trusted source for these addresses. This is especially needed in combination with the “Upgradeability” requirement to find the current versions of contracts [1].

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Regulatory functional Requirements:

KYC The system must provide means to identify customers and restrict trading access to only those that are identified and allowed to trade. At least a list of identifiers of known customers is to be maintained. This is a minimal requirement to be able to comply with laws like the MiFID II regulation [60] which requires financial institutions to collect data on their customers to be able to report to regulators. This also implies that the customers must be able to be authenticated by the system.

Blacklists The tokens must offer a blacklisting feature to disallow specific groups of persons, identified by their address, from trading a certain token, such as insiders in a company, enabling the system to comply with laws such as the market abuse directive (EU Directive 2014/57) [58].

Manual Authorization The system should be able to process actions that are manually authorized by a user, overriding the internal ruleset, enabling for example the compliance with court orders [162]. This requirement depends on "Role Based Access" as a user must have the correct role to authorize actions.

Information The system should be able to store relevant information for each stock, in order to be compliant with customer information regulations like EU Regulation 2014/1286 [64]. This means, that every token contract should have a method that provides a link to a document containing information about the security. Such a method could also be used to provide the ex-ante costs information required by the MiFID II regulation [60].

AML The system must be able to deal with anti money laundering requirements by storing the moved volumes of securities for each customer, and and disallowing transfers over a defined limit. This is required by regulations like the the EU AML Directive 2015/849, Article 11 [61][2].

Nonfunctional Requirements:

Upgradeability The contracts in the system must be upgradeable to newer versions to enable maintainability and to future proof the system [66], contrary to the standard immutability of code in the Ethereum system. The process of upgrading should be possible without loss of storage, i.e. the balances of the token etc. should not be lost.

Gas efficiency and transaction costs The system must not waste gas, i.e. it must be optimized for gas consumption (e.g. splitting up costly operations), so that the system is cheaper than conventional trading systems. This is to be verified by calculating the gas fee for a standard[3] and a fast transaction on the most complex form of the token, and comparing it to the average flat trading fee of discount online brokers, which currently lies between $5 and $30, according to investopedia.com [73].

Secure Libraries The system must use secure libraries to prevent exploitable bugs (over-, underflows, etc.), going along with the recommendations of the solidity team, who suggest the openZeppelin SafeMath library [176].

[2] It is important to note here, that as the aim of this thesis is to implement a prototype to showcase technical possibilities, this law acts as a stand in for similar laws, and is naturally not fully respected just by implementing this requirement.

[3] As of now (05/10/2019), the recommended gas price for a standard transaction is 2 GWei, while that for a fast transaction is 9 GWei, according to ethgasstation.info. Historically, most transactions have been around 18 GWei gas price according to etherscan.io [55], so this number is also to be compared.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Role Based Access In order to deal with different roles within the system, role based access control must be implemented [184], differentiating between levels of permission of users by categorizing them in different roles. There must be at least roles for the deployer, issuers for each token to represent the issuing company, and managers for the white- and blacklists. The system may include additional roles, if necessary.

4.2.2 Web Client

As mentioned before, the web client is not the main focus of the project, therefore its requirements will be kept rather short and concise to establish a baseline of functionality for a minimal web client for this thesis.

Functional Requirements:

Authentication The frontend must enable KYC-aware login (i.e. no arbitrary account creation, all new wallets must be cleared by a KYC component before an account can be created [60], [99]). Only customers, who are whitelisted by the KYC component of the smart contract system may access the functionalities regarding token transfers.

Completeness of smart contract mapping The frontend must enable users to interact with all functionalities of the smart contract system.

Role based access The frontend must be aware of the role-based access protocol in the smart contract system and allow actions accordingly. Only bearers of a role may see the UI elements regarding that role, to correctly reflect the smart contract system requirement "Role Based Access".

Views The frontend must feature views for backend workers such as issuers, as well as customer views including their respective options of interaction with the smart contract system, again in accordance with the "Role Based Access" requirement.

Wallet applications The frontend may be compatible with multiple wallet applications/plugins, so that different clients can utilize this frontend, but it must support at least one such application [4].

User Information The backend of the web client may store user information ([99]) to preserve privacy and provide KYC - if the User Interface needs any additional contextual information that is not stored on the blockchain, it is to be stored in the backend.

Nonfunctional Requirements:

Encryption If any user information is stored in the backend, it should be encrypted as recommended by the EU GDPR [21], [63].

Two-factor Authentication The application may enable two-factor authentication for additional security [33].

4.3 Smart Contract System

The complete system consists of two components. The first part is the smart contract system itself, consisting of multiple deployed and linked smart contracts on an Ethereum blockchain.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

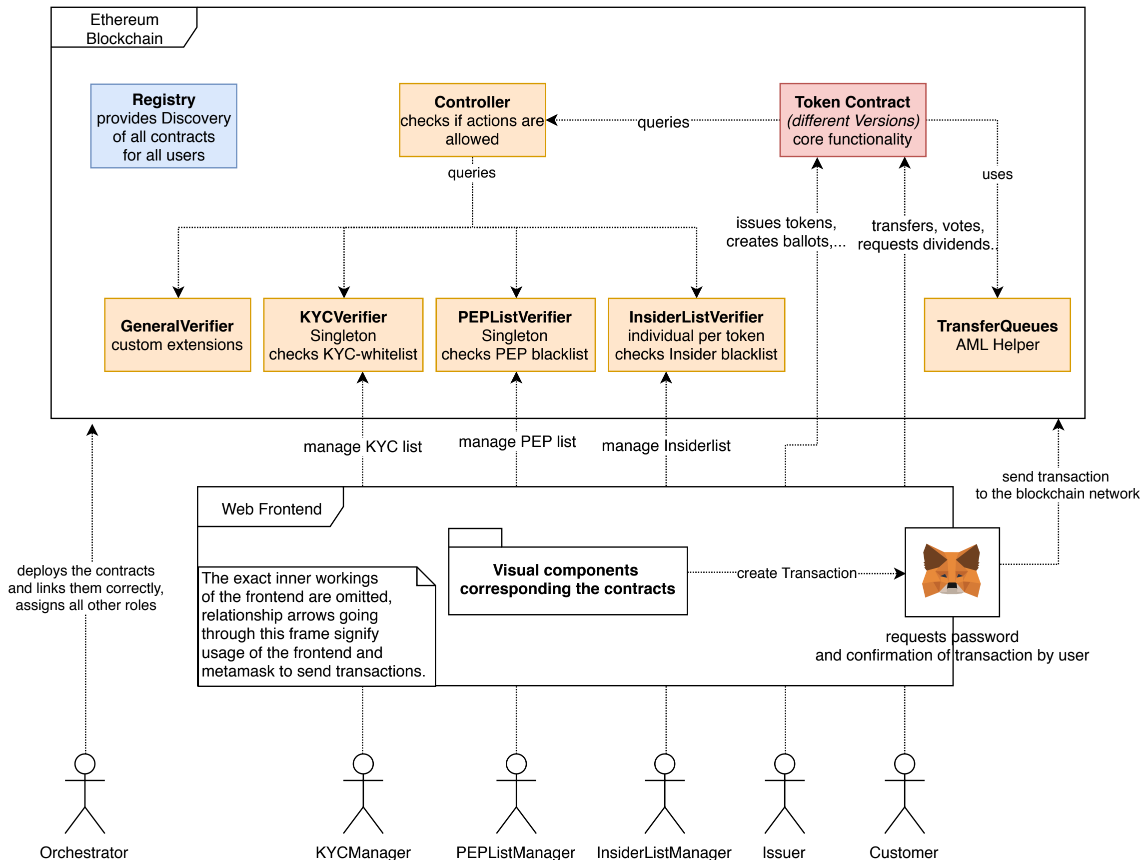

Figure 4.1: High level overview of the complete system showing deployment boundaries, roles, and exemplary use cases

The second part, which will be described in section 4.4 is the web-based client application, which is intended to provide a convenient method for interaction with the smart contracts system, as well as showcase its functions. As this was not the main focus of the thesis and is not as feature rich as the smart contract system, it will be described more briefly, primarily highlighting general problems and solutions that come up when implementing a client frontend for such a system.

Now for the smart contract system: It is the actual core of the system, and works as a standalone framework, while the client is primarily used for demonstration purposes, and could be replaced by any other UI component, that can communicate with the deployed contracts. The framework alone can also just be interacted with by sending transactions manually, utilizing any wallet software.

In figure 4.1, a high level abstraction of both systems and how they interact is shown, setting the context for the following detailed description of the components.

4.3.1 Project Setup, Frameworks and Testing

As mentioned before, the project was developed utilizing the truffle suite [189]. The suite is a combination of three different tools that aim to facilitate DApp development:

Truffle is the core of the suite, it provides a development and compilation environment, as well as automated testing and automated deployment of smart contracts.

Ganache is the so-called “one-click blockchain”, an application that starts a personal, local blockchain instance on a developers computer. This blockchain can be used to deploy and

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

test smart contracts during development without the need to spend real ether or connecting to a public test chain.

Drizzle is a redux-based collection of libraries, that ease front-end development for smart contracts.

Of these three, Truffle and Ganache are used for the development of the system, enabling deployment and testing and fulfilling the requirement “Deployability”.

Truffle requires a specific project setup, that is created either by calling the `init` command on the truffle command line interface, or by utilizing the `unbox` command to start with one of the predefined truffle boxes [188], that contain a base project to start from. The base directory structure of a truffle project contains:

contracts/ Contains all the solidity contracts of the project.

migrations/ Contains all scripts that are used for deployments, i.e. JavaScript files that contain instructions on how the contracts are to be deployed (sequence, parameters, linking between contracts...).

test/ Contains test files that test the deployed contracts. The test files can be JavaScript files as well as solidity files, depending on developer preference and goal of the test.

truffle-config.js Configuration file for truffle, contains preferences for the blockchain that should be connected to, the version of the solidity compiler that will be used for compilation and other configurations.

Tests are executed using truffles automated testing framework. While solidity tests focus on testing single contracts in-depth in bare-to-the-metal scenarios, JavaScript tests deal with the big picture, modeling complex scenarios and testing contract interactions [190]. A partly fitting and commonly mentioned analogy to conventional software engineering is that while solidity tests correspond to unit tests, JavaScript tests correspond to integration tests. But this is not to say, that JavaScript tests can not test single contracts, who can be tested just as well. In this project, testing is done purely with JavaScript tests, on the one hand, because they offer more functionality than pure Solidity tests, on the other hand, to reduce complexity by having only one type of test file.

JavaScript Tests are structured like Mocha Tests [126], and similar to Mochas `describe()` function, there is the `contract()` function, that provides truffles clean-room features. This means, that before each run of a `contract()` function, truffle provides fresh contracts using snapshotting. In addition, accounts are created and provided as a parameter, as can be seen in listing 4.1. This is an abridged example of JavaScript tests in the project. The `beforeEach()` function is called before each contract and contains the specific setup for the test. Afterward, the test function within the describe functions, that provides output for the description of the test, is executed. In this case, it simply calls `distributeDividends` and expects a revert. This expectRevert function stems from the openzeppelin-test-helpers package [140], a library of test helpers for JavaScript tests of solidity[4].

Another valuable method used for testing is a general mocking contract, that can be used to simulate any other contract similar to mocking frameworks in other programming languages (with fewer features of course), that was first published by gnosis [78]. The time-sensitive methods like AML constraints and ballot timing were tested by utilizing the web3 command `evm_increaseTime` to increase the time on the blockchain during the execution of the tests.

[4] This is the only drawback of JavaScript tests in comparison to Solidity tests that had an impact during development, as checks for reverts are supported natively there.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
1
2  contract('DividendToken', function ([deployer, initialHolder, distributer,
   ↪ recipient, anotherAccount]) {
3
4      beforeEach(async function () {
5
6          {...}
7
8          this.token = await DividendToken.new();
9
10         {...}
11     });
12
13
14     describe('distributeDividends', function () {
15         describe('when no tokens were issued yet', function () {
16             it('reverts', async function () {
17
18                 await expectRevert(this.token.distributeDividends({from:
                   ↪ distributer, value: new BN(web3.utils.toWei('2', 'ether')
                   ↪ )}),
19                     'Currently, no tokens exist.'
20                 );
21             });
22         });
23     });
24
25     {...}
26 }
```

Listing 4.1: A small excerpt of a JavaScript Test for the DividendToken, showcasing the mocha teststyle and checks for revert

In the project, all important functionalities are tested by JavaScript tests of this kind. As the interaction and the manual deployment of such a smart contract system are time-consuming, these tests are also a very viable alternative to manual developer tests during development.

4.3.2 Overview of the System

In Figure 4.2, a class-based overview of the whole system is presented. The core token contracts that all extend each other are presented in red, each adding functionality to the functional set of its parent. The reason for this split is to enable the creation of tokens with different levels of feature-richness, so that tokens, that may not need voting functionality are not burdened by its calculations, saving gas on their execution. In yellow, there are all external Interfaces that are defined by standards of the Ethereum community. On the right hand, there are abstractions of multiple contracts, that are depicted as packages in order to stay within the boundaries of the schematic overview. The contents of these packages are illustrated in Figure 4.2. There is the Role package[5], that contains all contracts pertaining roles, some of which are extended by the token contracts or some controlling contracts, the controlling package, which is used by the ERC1594 contract to check if an action is allowed by the verifiers, the TransferQueues contract for AML constraints, and lastly, a package for general use libraries.

A crucial part of the system that is not shown in this view is the registry and the proxies that are interposed in between all contracts, as the focus of this diagram is the programmatic hierarchy (as it is a class diagram) rather than the deployment structure (as in a deployment diagram). To make

[5] The role contracts are also described more in-depth in Fig. 4.3.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

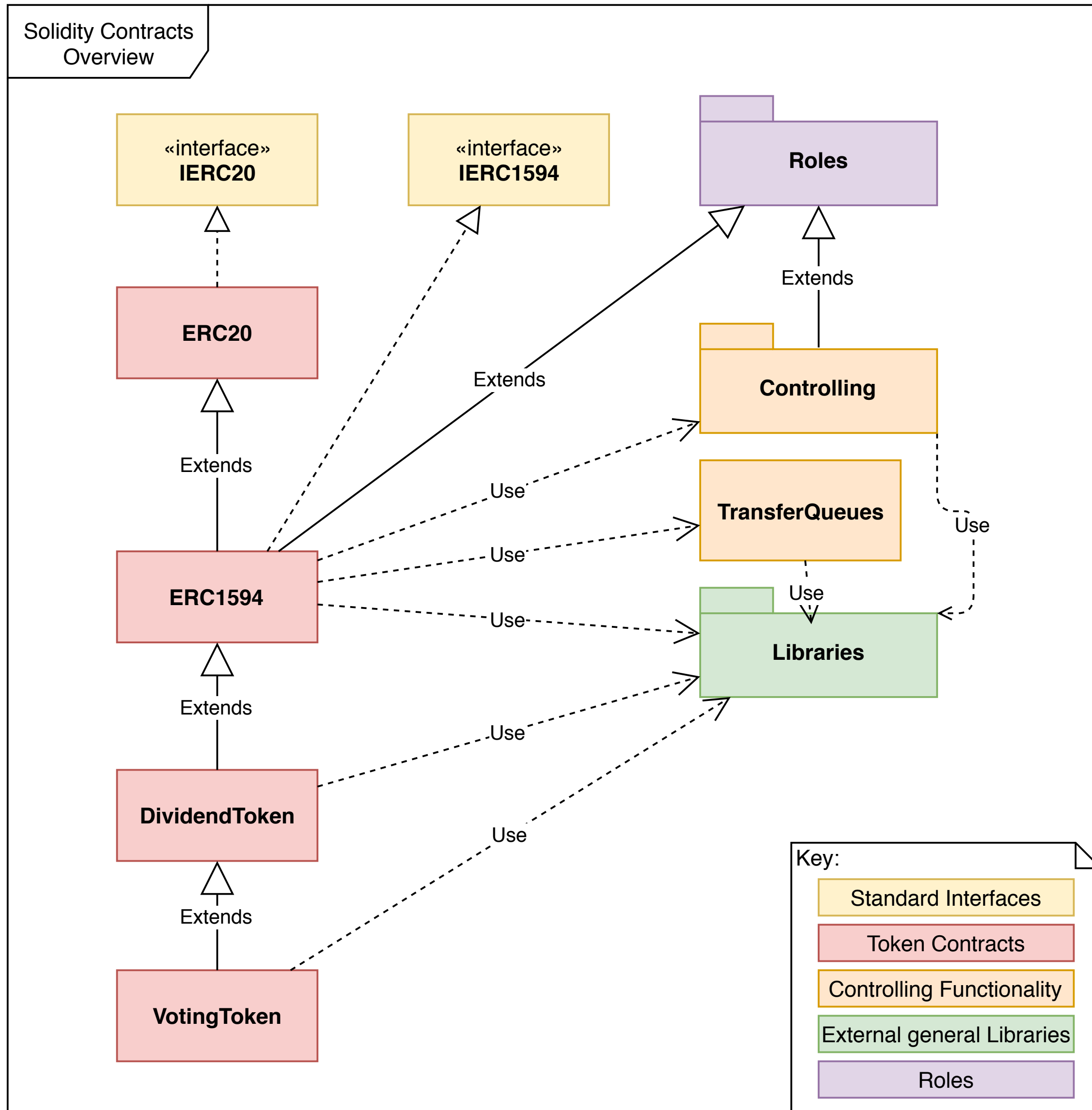

Figure 4.2: Schematic Overview of the system in form of a class diagram

the system upgradeable, each contract is hidden behind a proxy, so that the logic can be easily be switched. The registry indexes all proxies by their name to aid in the discovery of the contracts. These two features are discussed in the subsections 4.3.9 and 4.3.10.

In the following sections, each component of the system will be described, starting with a description of the general problem the component aims to solve, over possible solutions and choices that were made during design up until closeup looks in the implementation of the segment. The full code of the project can be examined online on github.com/jobrot/secblocks.

4.3.3 ERC1594 Contract

As it can already be seen in the overview, this contract is arguably the core of the whole system. It is the central token, as well as the minimal contract (including its dependencies) that must be deployed in order for the system to function.

The first and foremost decision in the design of this contract is the question of which further standards for security tokens are to be adhered to, in accordance with the requirement "Further

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Token Standards". Is there a de-facto global established standard, or are there many competing ones?

Of the projects that were discussed in the state of the art chapter, only those discussed in the security token chapter 3.3.2 are security-focused and matured enough to potentially produce a meaningful standard for security tokens. All of them at least try to introduce a standard for security tokens:

R-token Harbor, as discussed in 3.3.2, created a token standard called the Regulated Token or R-Token for short, described in their whitepaper [14]. The standard is relatively simple and can be implemented extremely fast. It only requires the implementation of a Regulatorservice with a `check` method, that checks if a transfer is allowed or not. This simplicity of course also has the drawback that some features must be cut out in favor of it. Therefore, there is no proper separation of concerns, at least in its basic implementation, although this could be changed. But a more concerning issue is that it does not support restrictions on minting or burning, other than the standard `onlyOwner` restrictions. A positive unique characteristic compared to the other standards is, that upgradeability is built-in, as the checkers can be switched via the registry component. But this does not include eternal storage of any kind, i.e. once the checkers are switched, data must be migrated manually, if needed. To sum it up, this is rather a token, that can be reused, than a standard. A real standard would have to leave the implementer more freedom than simply configuring a single function, to put it simply. The code for the standard can be found on harbors GitHub [85].

DS Token Standard The company Securitize has also produced their own token standard, the DS Token, as described in their 2018 whitepaper [37]. This token is very feature-rich, and, as described in the initial discussion of the token, focused on industrial applications. For a quick overview, refer to the graphical outline of the system in Figure 3.2. In the initial publications, there was not much of a standard present, that could have been adopted by other implementers, however recently, with the release of the 3.0 version of the Securitize Platform on May 13th of 2019 [170], parts of the system were open-sourced. In this press release, securitize also claims to have the highest adoption rate amongst industry players. But checking the GitHub repository, it becomes clear that what was actually open-sourced are only the interfaces of the system, that are needed for developers to interact with their proprietary system, i.e. the bare minimum for any third parties to interact with the DS-Token. This means, that the DS-Token still can not be implemented by any other entity than Securitize itself, and can therefore hardly be called a standard.

ERC-1400 Another contender is the token standard proposed by a team around Polymath. It was created based on polymaths ST-20 token [110], out of the need for a generally usable standard for security tokens [41]. This token standard is the first and only one on this list that has a real claim to being a comprehensive potential standard for security tokens. It is worked upon and supported by multiple companies [41], has extensive documentation, and is geared towards being a standard first, and not only as an afterthought after the development of a token. It also encompasses vastly more features than the other standards presented here, including minting and redeeming, tranching, arbitrary data validation and many more.

As evidenced by the descriptions above, the only reasonable choice for a standard, barring developing one anew, is the ERC-1400 group of standards.

Due to the aforementioned feature richness, the large community support as evidenced by GitHub activity, and the comprehensive, sophisticated interface definitions, it was decided to adhere to the this standard for the development of the prototype in this thesis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

But the fact, that the ERC-1400 group contains so many features is not purely positive, as it might be, that not all of them are necessary for the purposes of this thesis. But the developers of the token standard have considered this during development and split the actual standard up into multiple substandards, which have to be analyzed individually to decide which of those shall be implemented.

ERC-1594 This is the core of the security token standard. It is an extension to the general token standard ERC-20, that includes on-chain transfer restrictions, as well as the possibility to inject off-chain data for additional validation. Besides, there are error checker methods, as a failure in this context can be more complex than in a simple ERC-20 token, where insufficient balance or not enough approved funds are the main causes of error. This checker method can be called by clients in advance to a transfer to test if a transfer is allowed. The standard also provides issuing and redemption of tokens.

This must be implemented to fulfill the minimum of standards of the ERC-1400 suite. It also covers all essential functions for the implementation of a security token, therefore it was decided to be included in the implementation.

ERC-1410 This extends the core token standard to describe partially fungible[6] tokens. Specifically, this standard enables adding metadata to a token, making it distinguishable from other tokens. The concrete use case for this is to create batches of securities that can be treated differently, e.g. to differentiate between types of securities, if this is envisaged by the issuer and/or the respective legal framework. For example, this could be used to separate investors from the first offering round of the security from secondary offering investors or to differentiate between common or preferred stock.

The differentiation between types of securities can also be done by inheritance, which, while requiring more effort also enables different functionality between token types. Also, abandoning fungibility of tokens comes with its own problems, like splitting up the market and decreasing liquidity. Therefore it was decided to forego implementation of this standard, and implementing types of securities by inheritance.

ERC-1643 This standard extends the token with management of associated documents. This is important for security tokens due to Regulations such as MiFID II [60] that enforce distributors of securities to inform the buyers via standardized informational documents for the bought securities. The standard enables documents to be attached to tokens, but it does not in any way enforce that the customer agrees to having read the documents, which would be a useful addition to ERC-1643 [39].

As this substandard does not add that many interesting features to the system and would be quite costly to implement in the user interface, it was decided to refrain from implementing it.

ERC-1644 This standard specifies controllable tokens. This means, that issuers or other entities that are empowered to do so by issuers can force transfers of tokens. This is useful for instances, where a transfer is required by law, e.g. for dispossession or if the original holder of the secret key died to distribute the inheritance [40].

While this concept has its merits, it was not implemented to increase trust in the system, as such a possibility of forced transfers goes against the basic decentralized goals of blockchain technology. Of course, it is possible, that such a functionality is mandatory in certain

[6] Fungibility means that two individual assets of a certain type can be exchanged for each other and have the same value, e.g. as denominations in fiat money.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

jurisdictions, and it may be even prudent to implement it, considering the examples given above, but at least for the initial prototype, it will be left out, as the system also could easily be extended to include this functionality.

As elaborated above, it was concluded, that for the implementation of this thesis, the core functionality that is described in ERC-1594 shall suffice. Firstly, because most of the technically interesting parts lie within this contract, secondly because the implementation will also concentrate on other features that are not covered by this standard, which would not be possible to implement within the time constraints of this thesis if the whole standard was implemented. Additionally, due to the intercompatibility of the substandards, it would be no problem to expand the system, if necessary.

The concrete implementation of the ERC-1594 contract inherits from the ERC-20 standard and therefore also needs to contain an implementation for ERC-20 tokens. As there is no de facto standard implementation of the ERC-20 token as of yet, the implementation by openzeppelin [143] was used. It was also slightly amended to fit the implementation by changing visibilities of variables etc. The inclusion of the ERC-20 token functionalities fulfills the "ERC20" requirement.

The ERC1594 contract (see code A.1) was the starting point of the implementation and is arguably also the contract that was revised the most. The base that was used is the reference implementation by the SecurityTokenStandard Team [172]. It provides all methods that are defined by the standard. The contract stores a reference to a Controller, as well as to a Transferqueues Contract, that is used to verify actions by users and check if the AML component allows transfers. To this end, each function calls its respective verify method of the controller, (e.g. `verifyalltransfer`), that reverts if any of the verifiers fails.

The fact, that this contract needs to implement the ERC-20 Interface, and inherits from an ERC-20 instance also presents a problem: If the standard transfer function from the ERC20 contract is kept unchanged, it could be accessed, bypassing all checkers from the actual ERC1594 contract. But these functions are to be kept to stay ERC-20 compatible. In order to achieve this, a new method `transferWithData` was created in the ERC20 contract, that mirrors the ERC1594 method, and is overwritten by it. This function is called by the standard `transfer` function so that no transfer can slip by the checks imposed by subcontracts. It also ensures, that inheriting contracts beyond ERC1594 only have to extend `transferWithData` with their own checks, without having to think about extending any other function, as it would have been the case if we opted to overwrite the `transfer` function. An analogous fix was applied to transferFrom.

As said before, another responsibility of the ERC1594 contract is checking the AML constraints to fulfill the requirement "AML". As the EU AML Directive 2015/849, Article 11 (b) (i) [61] rules, transactions over €15000, or multiple transactions that appear to be linked and are over this threshold are to be treated with additional due diligence[7]. What exactly this due diligence entails, is not the goal of this AML component, but rather to create the technical framework that enables the detection and prevention of such transactions.

To check this, transactions and their amount have to be saved for a set amount of time. This poses multiple interesting technical problems:

- retaining transfers to sum them up
- ordering the transfers by date
- cleaning up old saved transfers in order to save space

[7] As mentioned in the requirements, this is mostly aimed at simply enabling such kind of checks, preparing the system for the implementation of concreter AML checks.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

- staying efficient gas-wise
- timing on the blockchain

The complete AML process is not regulated by the ERC1594 token, but rather extracted to a separate contract to save contract size. However, some of the logic, as well as the constants needed for the calculations, still reside within the token contract.

The retaining of the transfers is solved by simply storing them in an array, but this alone is no solution, as an ever-expanding array is unsustainable. To be able to delete and keep order, a FIFO-Queue[8] is needed. This is implemented in the TransferQueues by an array with a moving start index, by storing an integer that describes the startindex of the array, and perpetually freeing the storage in the first elements, and pushing the index forward. The TransferQueues contract also offers a convenience method to sum up all current transfer amounts of the queue.

The management of the queue is done in the token contract. Every time the AML constraints are checked, the start of the queue is examined, and all elements that are older than the predefined retention time allows, are dequeued from the queue.

The concrete check just controls, if the maximum number of tokens that can be traded within the retention time is overstepped or not, and reverts if this is the case.

Of course, this is a simplification of how the AML checking process would be executed, as the number of shares does not equal monetary value. But this could easily be implemented, if the deployer of the system were to create or link an oracle or a smart contract operated by an exchange, that could be queried by the token contract in order to check the current share price, so that the AML limit corresponds to the current share price, and checks if the transfers are over €15000, as the EU regulation rules.

But as said before, this component focuses on enabling the token from a technical point of view, without defining the concrete juridical parameters for real-world use, to leave the system configurable.

Using a deployment script and the truffle system, multiple instances of this contracts can be deployed to a blockchain, each identified by a name, which constitutes fulfillment of requirement "Token Creation".

4.3.4 Controllers and Verifiers

In this section, the controller contracts, that check whether participants of a transaction are allowed to execute it, are described.

In principle, fulfilling KYC obligations when using a blockchain system is relatively straightforward: As every wallet is uniquely identified by its address, and its private key guarantees that only the owner of the wallet can sign transactions or approve transactions from this address, storing the addresses whose identities have been confirmed off-chain suffices as a basic KYC process. This is regulated in the KYCVerifier to satisfy requirement "KYC". This contract holds a mapping that defines whether an address is KYCd or not. This mapping can only be amended by a sender who has the role `KYCListManagerRole`. The functionality that regulates this access will be discussed in detail in 4.3.8.

Of course, whitelisting customers for each individual token contract is not practical, therefore the KYCVerifier is to be deployed as a single contract that is referred to by all token contracts.

[8] FIFO means First in - First out, describing the order in which elements that are added to a queue are removed.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

But simply verifying and whitelisting generally approved customers is of course not a solution that is fine-grained enough to fulfill all potential legal needs according to MiFID II [60]. Therefore, additional verifiers were created, that work in a similar fashion as the KYCVerifier: The PepListVerifier, that signifies politically exposed persons, whose trades have to be scrutinized additionally[9], and the InsiderListVerifier, which also consists of a blacklist of insiders of a company, that are not directly allowed to trade. While the PepListVerifier is to be implemented on a national/supranational level just as the KYCVerifier, the InsiderListVerifier is specific to a single company. Each verifier again possesses its own manager role that manages the list. These verifiers fulfill the blacklisting requirement “Blacklists”.

For the system to be easily expandable, partly in realization of requirement “Expandable Rules”, a general interface for all these Verifiers was also created.

But checking all these verifiers and providing appropriate error messages proved to be too large for the general token contract, therefore this functionality was extracted to a new contract called `Controller`. This contract is the only one that is needed by a token contract to check all verifiers, and offers convenience methods for all token methods like transfer, issue, etc., that check all verifiers that are registered to this controller. It also enforces the existence of at least one KYCVerifier, PepListVerifier and InsiderListVerifier per token contract, and optionally also contains an unlimited list of general verifiers conforming to a standardized IVerifier interface.

This setup enables a token to check all relevant verifiers with a single method call to the controller. The controller also provides helper methods that are used in the `canTransfer` methods, that do not revert, but rather provide a reason for potential failure. The setup can easily be amended or extended by implementing new verifiers and adding them to a controller, even if the controller is already deployed and running, making it flexible to deal with new requirements.

4.3.5 Libraries

This section gives a quick overview of the problems and overall rationale of including external libraries and dependencies within the system.

In general, libraries in solidity are contracts that are initialized with the keyword `library`. They are only deployed once and mostly contain simple operations that are reused often in the code. They are stateless and can be seen as implicit base contracts of all contracts that use them. This means, every contract that uses a library can use its functions, as if it were its own.

In this project, multiple libraries were used, primarily for mathematical conversions and safe operators.

Other external contracts that were used include functionality to check whether an address contains a contract or wallet, the Roles contract that provides the base for the role-based access functionality, and the standard IERC-20 interface and Ownable. All of these contracts are imported from the Openzeppelin library, which is an open-source project providing libraries and other contracts to be reused for smart contract development [142], thus conforming to requirement “Secure Libraries”.

While openzeppelin itself provides a node package that allows imports direct from their GitHub, it was decided to import the code manually as these imports are not completely matured yet, and this enables changes to suit the project.

[9] This additional scrutiny is defined as enhanced customer due diligence by Article 18 of the EU Directive 2015/849 [61]. It consist i.a. of analyzing the source of funds, wealth of the individual, and income of the individual, and checking for discrepancies amongst these.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

This copying of dependencies is certainly less than ideal, but there is currently no generally accepted alternative to this approach. The OpenZeppelin library obviously only provides contracts by OpenZeppelin and does not manage versions, so it can not substitute general package management. One promising project in this direction is EthPM, the Ethereum Package Management system [56], but this is not yet widely accepted.

4.3.6 DividendToken

The first subtoken of the ERC1594 token that was created to extend its functionality was the DividendToken (see code A.2), which is able to pay out dividends, fulfilling the “Dividends” Requirement. This token corresponds to preferred stock, which is one of the two big types of securities. Preferred stock represents ownership in a company without the right to vote, but with preferred treatment when paying out dividends [87]. In addition, should a company be liquidated, preferred stockholders also have a greater claim to a companies remaining assets.

Now how can dividends be calculated correctly in a token? The most obvious solution would, of course, be to create a method for distributing dividends, that iterates over all token holders, and sends them their share of the dividends, according to the current token distribution. But this solution encounters two grave errors: Firstly, token owners are always stored in mappings, that can not be iterated over, and secondly, even if these were stored in an array or a similar structure, such an iteration and send to multiple thousands of token holders would quickly run out of gas, and become unusable once enough token holders exist.

The main problem that has to be overcome when calculating dividends, is to store the information on how much dividends each share is entitled to, without disregarding transfers. Ignoring this could lead to attacks where a holder cashes in his dividends, transfers his tokens to another token holder, who in turn also cashes in, effectively using the same tokens twice to increase dividend gain.

A technique for dividend calculation that addresses these problems and was used in the implementation can be found in a proposal for a dividend-paying token standard developed by Roger Wu [202].

It uses the pattern of amortization of work to bypass the possibility of running out of gas, by shifting the calculation of dividends to the users of the tokens, away from the distribution operation.

The implementation adds three important state variables to control the dividends:

- magnifiedDividendPerShare
- withdrawnDividends
- magnifiedDividendCorrections

MagnifiedDividendPerShare is an unsigned integer, that describes how much Dividends each share is entitled to withdraw (over the entire lifetime of the contract). The magnification is added because solidity does not provide numbers with decimals yet, and this is the only possibility to calculate the dividends while including smaller amounts. The magnification of 2^{128} is chosen in this way, that it does not overflow on dividend payments up until 1 billion ETH, while still catching as small dividend payments as possible. The variable is updated each time ether is paid to the contract.

WithdrawnDividends is a mapping from addresses to unsigned integers simply storing the withdrawn dividends for each address, that is updated on each withdraw. This makes it so that the basic withdrawing simply consists of multiplying the current magnifiedDividendPerShare with the current balance of the owner dividing the result by the magnitude, and removing the already withdrawn Dividends.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

But as an observant reader might have noticed, this does not yet take into account the transfer of tokens, or the issuing of new tokens. This is what MagnifiedDividendCorrections is for: It is a mapping from addresses to integers, that stores the correction of dividends for each address. This means, if e.g. new tokens are minted, the corrections are reduced by the amount of tokens that were minted multiplied by the current magnified dividend per share. This means, that the issuing did not change the amount that the address that was issued to is entitled to, at least until the next distribution. Similarly, on a transfer of tokens, the correction is added to the sender's corrections and removed from the receiver. This enables dividend paying without out of gas errors, and with transfers taken into account.

Another option would have been to create snapshots at the time of dividend payout, that could then be queried to check the token balances at the time of distribution, but this would be overkill as the presented method makes the calculation much easier in terms of gas costs.

The token functions themselves override the functions from ERC1594, and in turn call their super functions so that all functionality is called. This makes it so, that the functions of the basic ERC20 contract also reflect this functionality, as they call the overridden functions.

The only shortcoming of the contract in its current form is, that it can only pay out dividends in ether. But it could easily be adapted to pay out any ERC-20 token that supports approval functions, this way the payouts could be undertaken directly using some kind of stablecoin.

4.3.7 VotingToken

The second subtoken of ERC1594 is the Voting token (see code A.3) which fulfills the requirement "Voting". It corresponds to common stock, which is the type of security most people refer to when they talk about stock [87]. Common stockholders are ranked after holders of preferred stock when it comes to liquidating the company, but in return get the right to vote.

For a token to be votable, it must be possible for eligible addresses to create a ballot with an identifying name, and multiple options to choose from. The ballot must also have a time limit, after which no more votes can be cast.

The general problems that come up in the creation of this token are quite similar to those of the Voting token before: it needs to be stored which address has how many votes (=tokens) at the point in time when a Ballot is created. But the approach that was used with the DividendToken does not work here, as multiple ballots need to track distinct voting capability, that is not canceled out by voting in other ballots. Therefore, it has to be solved differently.

In the previous section, the possibility of snapshotting a token was already mentioned but thrown out as overkill. This technique could potentially be used for a VotingToken. It was developed by the creators of Giveeth, an open-source Ethereum donation platform, in a token implementation called the Minime token [77]. It enables creating a clone of an ERC-20 token at a certain point in time, i.e. a completely separate token is created with all balances recorded as they were in the main token at the time of the split. These MiniMe tokens can then, in turn, be used for many different applications that any normal ERC-20 token would be able to do, in our case we could split a token to act as voting coupons, that can be burned for a specific ballot.

But creating a whole new token for each ballot would still be, while doable, overkill and quite costly for the VotingToken. Therefore, only the basics of the snapshotting features are used, to create historized balances for each user, that can be queried by in-contract ballot objects.

Concretely, this works by creating `Checkpoint` objects, that contain a block number and a balance. These checkpoints are stored as a map of addresses to arrays of checkpoints in `his-`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

`torizedBalances`. In addition, there is an array of checkpoints historizing the total supply of tokens, so that the token amounts can be put into context of the total supply.

In order to keep these balances up to date, every function that changes the token amounts includes a call to `updateValueAtNow` that adds a new checkpoint at the end of the array for the specified address. To check balances, a binary search is utilized, including a shortcut for the actual current value, to skip the binary search, especially as finding the last element would be a worst-case scenario performance-wise for binary search, being bound by $\mathcal{O}(\log n)$.

The voting ballots themselves can only be created by an issuer of the token. They contain their name, a list of options to choose from, counts for each of these options, a mapping on addresses that already voted, a cutoffBlockNumber, indicating the block number where the balances are relevant to the voting distribution, and the end date.

All names and options are limited to a maximum size of 32 bytes, on the one hand, to save storage space, and on the other hand to enforce short, actually writable options, as voters have to put in the names of the ballot and the option they want to choose.

Once a ballot is created, each address that held a number of tokens at the cutoffBlockNumber is entitled to vote with exactly that much votes. This is checked for in the `vote` function, as well as potential passing of the end date of the ballot.

The winner calculation is executed via the function `currentlyWinningOption`, which does not close the ballot or change the state in any way, and just checks for the winning option by iterating over all vote counts.

Contrary to the DividendToken, in the VotingToken, instead of expanding the ERC1594 functions and calling their respective super functions, the internal functions of the ERC20, that are called by the ERC1594 functions are overwritten. This is because in the DividendToken the way how balances are transferred is changed due to the snapshotting, and the normal balances variable does not apply anymore. This overwriting ensures that all other functions from parent contracts also calculate using the balances supplied by the VotingToken, if the contract is an instance of a VotingToken.

There are also other possible techniques of how a VotingToken could be developed, which were considered but later discarded during development. Those techniques are on the one hand to drop fungibility of the token and add a state to it, and on the other hand to escrow the tokens [104].

Dropping fungibility could be achieved by implementing the non-fungible token standard ERC721 [48]. While this would certainly be doable, it would open the whole system up to a host of new problems: securities could only be sold at a discrete amount, different tokens would be valued differently, adding confusion on exchanges, and of course, an adapter of some sort for ERC20 compatibility would be needed. In short, this option is not feasible in this system. The other option being the escrowing of tokens from the time of voting until would be very impractical for investors. It would either force ballots to take place during a very limited time, negating practicability advantages the remote voting process provides, or lock up funds for an extended amount of time, greatly decreasing liquidity, and utility for investors (e.g. by having ones securities locked up in an escrow during a major crash of the stock value). Therefore, this option was also out of the question, and the aforementioned technique of snapshotting was used.

Theoretically, it would also have been possible to fusion Dividends and Voting, as the snapshotting from the VotingToken could easily stand in for the dividend calculations. This would have saved contract size and made the implementation easier. But it was decided to split them, to provide a maximum of flexibility, as it is entirely possible that issuers may want to forego voting capability even in common stocks, and therefore choose to use DividendToken for common stock. In this case, the token would be unnecessary bloated and use an avoidable overhead in gas.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

This also implies, that if a company wants to have common as well as preferred stock, it must deploy two tokens. This enables dividend payments to the two types of tokens as it is commonly done with preferred stock getting a higher dividend quota than common stock.

4.3.8 Role Based Access

Apart from the business- and juristical-driven access controls for the "user" functionalities like transfer and redeem, that are regulated by the already discussed controller component, there is also the need for "backend" access control for all functions related to managing the contract and issuing, etc.

The simplest and most commonly chosen approach to access controls in general in solidity is the Ownable-Pattern. The creator of the contract is stored by a function checking msg.sender in the constructor. Then, a modifier is created, that only allows the owner to execute certain functions [10]. Of course, such a simplistic approach does not work here, as the system consists of multiple different functionalities that are supposed to be executed by multiple groups of users, e.g. the person that may create ballots should not necessarily also be able to remove controllers from a token.

Therefore, some kind of role-based access control is needed. The way this is commonly done in solidity is by creating contracts that represent a certain role, and provide functions to add and remove addresses to this role, as well as a checking modifier. The contracts that need to use the roles are then implemented as inheriting from the role contracts. For code reuse, the openzeppelin Roles library was used to simplify interacting with the role mapping. Note that this means, that each contract has its separate role storage, if an address has a certain role for one token, it does not mean that it also holds the role for another token.

Another possibility of implementing role-based access would be to implement a central contract, that either stores general roles for the whole system or mappings of contract addresses to roles to provide different roles for each contract. The pros of this would be that all access related information would be located centrally in a single contract and less storage would be needed in the actual token contracts, but on the other hand, as each check would be an external call, it would need more gas, and go against the general design philosophy of self-governing and data ownership of contracts. Therefore it was decided to go with inheriting from the role contracts.

But it has to be conceded, that the contract size restrictions of this approach did influence the development. Initially, there was an additional role, VotingOfficial for VotingToken, to distinguish between issuers and addresses that can create ballots, but this was scrapped to save space. This exemplifies the described drawback of this approach versus a centralized Role contract that can be queried: additional roles quickly eat away at contract size, meaning that bigger systems sooner or later need to outsource the access controls of their contracts. Contract size in general and the constraints that stem from it will be discussed in more detail in 4.3.12.

Diagram 4.3 shows, what exactly was implemented. It gives an overview of all roles that an address can occupy, and how they interact with each other. It also shows all methods that are controlled by their respective roles. The color coding is analogous to the previous schematic overview 4.2.

[10] Modifiers are properties of contracts, similar to functions. They are small pieces of code, that can be added to any function in the same contract by adding their identifier to the function head. This means, that the modifiers code is executed before the function code. Most commonly this is used to create access restrictions to functions by having a `require` check within the modifier, that reverts the function, if it is not fulfilled.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

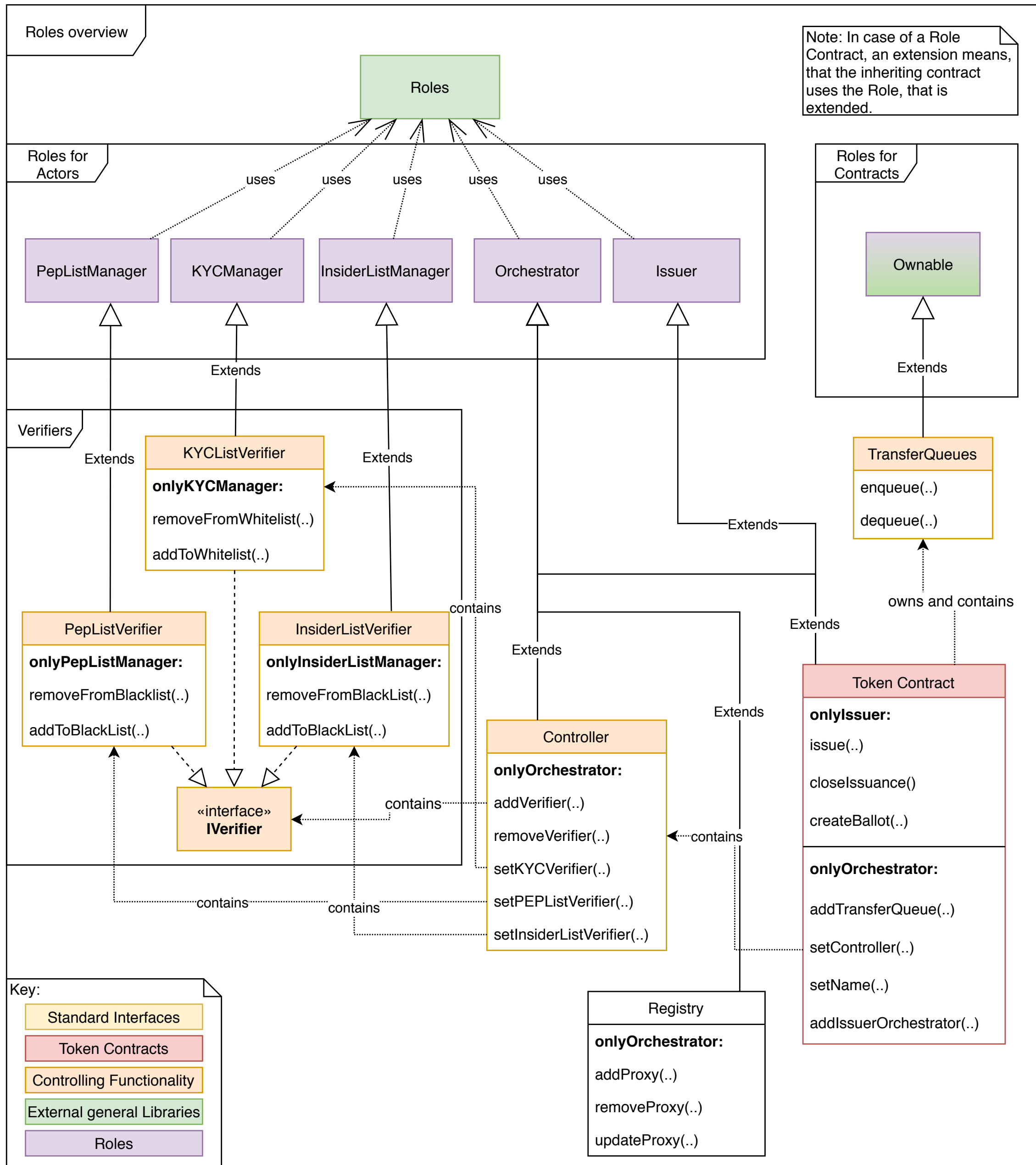

Figure 4.3: Overview of all roles present in the access control system, the contracts that use them, the methods that are guarded by the roles, and how all contracts interact.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

As is evident by the diagram, there are two types of roles in the system: Roles for actual actors, meaning users or operators that interact with the system, identified by their control over an address, and roles that can be taken by contracts, concretely only the Ownable role.

The Roles that are present in the system are:

ListManagers PepListManager, KYCManager, and InsiderListManager are functionally identical, they just pertain to other subcontracts. These roles enable eligible addresses to edit the white/blacklists in the respective verifier contracts. KYCManager might be held by bank clerks that are allowed to add new customers into the system. PepListManager might be a person working with the local government or overseeing the registration of new Users to check for politically exposed persons. The InsiderListManager is either also a high-ranking bank clerk or even a member of the respective issuing company, as publicly traded companies often are responsible for reporting depots of their employees.

Orchestrator Arguably the "highest-ranking" among the roles, the orchestrator is typically the address from which the smart contract system is deployed, that is responsible for connecting all contracts. For this, it has access to all functions setting the links to other contracts, such as the management of all verifier contracts in the Controller, and the setting of TransferQueues and the Controller inside the ERC1594 token contract. The Orchestrator of the token contract is also able to add an Issuer to the token contract, which is necessary in order to enable proxy deployment (in-depth reasoning in 4.3.9). The registry is also controlled by an orchestrator, who manages the list of registered contracts within the registry.

At this point, it has to be reiterated that while the Orchestrator role has the same name, contract code and most likely also represents the same user (the deployer) for these three contracts, the three orchestrators are nonetheless not identical from a technical point of view, as each of the contracts stores its orchestrators on its own.

Issuer The issuer is pretty straightforward, it simply enables an official of the issuing company to mint more of the tokens, so long as the issuance is not closed. In the case of a VotingToken, it also enables holders to create ballots that can be voted upon.

Ownable / Owner The Ownable role contract, creating the owner role has multiple unique characteristics compared to the others on this list. It is the classic standard in access control that was described above and therefore does not need the Roles library. The code is mostly used from the external library of openzeppelin itself, which is signified by the double color. It is only used in the TransferQueues contract, enabling the owner alone to manipulate the queue. As a TransferQueue always corresponds to a single token contract, and no manual interaction with it is required after deployment, the simpler Ownable, only enabling a single owner was used, instead of a more complex role. Ownership of TransferQueues is held initially by the deployer address, but passed on to the token contract during deployment, so that only this contract can add and remove elements in the Queue.

The implementation of this role system realizes the requirement "Role Based Access".

4.3.9 Upgradeability and Maintainabilty

Upgradeability and maintainability are important topics in all software products, but in the smart contract field, this aspect of software engineering is especially vigorously discussed.

To understand the problems that arise from including upgradeability in smart contracts, one has to consider one of the basic properties of smart contracts as discussed at the beginning of this thesis: Smart contracts are *immutable*. This means, once a smart contract is deployed, it cannot be changed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

But if this is the case, how can maintainability be achieved? And that is exactly where the problem lies: However maintainability, i.e. the ability to change a contract after deployment is achieved, it will break this basic property of smart contracts. Such a possibility may break public trust in the system and must, therefore, be scrutinized intensely before being put into action.

There are even developers, who argue that upgradeability in smart contracts does not only destroy trust but also introduces bugs, ultimately being a bug itself and therefore, should not be used at all in smart contract development [122].

A theoretical alternative to creating directly upgradeable smart contracts where code can be swapped to without the consent of the user are opt-in migrations. This means that to upgrade to a new version of the contract, the developer has to publish a new contract, that the users have to consciously switch to using.

But it is simply not viable to migrate such large swaths of data, as are anticipated in a large scale securities trading platform. Even if the the data migrations could be amortized by forcing the users to migrate their own data[11] when they switch to the new version, this still means that bug fixes can not be rolled out quickly and in a forced manner. This would be problematic e.g. in case of the uncovering of a critical bug. In addition, having multiple versions of a security token that should represent the same security at a time is very undesirable for issuers and market makers, as this restricts liquidity and might be unintuitive for users. For these reasons, a migration approach for potential upgrades is out of the question for our needs.

In addition, the argument that upgradeability removes trust in the smart contract code may be true for decentralized or one-man organizations, but if one is investing with a bank there has to be a certain customer relationship anyway. If the customer already trusts the bank enough to invest with them, he will also trust them not to change the smart contract of the token under his feet, especially since that would destroy the banks credibility and trust in all of their tokens.

So, as the alternatives are not really viable for this system, what are the general possibilities of making a smart contract upgradeable? The following list gives an overview of the conceptual approaches:

Simple proxy The most simplistic approach to upgradeability simply interposes a proxy contract between the user interface and the actual contract, that passes out the address to the actual current implementation. This implementation can then be switched out at will by the owners of the system. This has the obvious drawback, that all data of the contract is either lost on upgrade or must be manually migrated.

Simple data separation This is a simple improvement on the previous approach. All state variables from the actual token contract are extracted and stored in a separate data contract. This enables upgrades of the logic contract without losing state. If the structure of the data contract needs to be upgraded, the state is lost as with the simple proxy, so complete upgradeability is still not reached with this approach.

Complex Proxy Patterns These options rely on a more complex form of proxy: They store the data of the contract directly in the proxy, enabling an upgrade of the logic contract while the data is stored in the proxy contract. To do that, an adjusted version of delegatecall is used. As the delegatecall function does not return the results of the called function in solidity, this functionality has to be added using assembly code. The feature that enables this types of proxy, is that with a delegatecall execution, the method of the target contract is executed in the

[11] This migration in itself would open up the system to a host of new issues, namely in ensuring data integrity during migration, preventing users from changing their data or double spend attacks over the two versions, to name a few.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

storage context of the calling contract (this is the same method that is used for the execution of libraries). This means, that the called (logic) contract can access the state variables of the calling (proxy) contract, enabling the state to be stored in the proxy, while the logic contract can be upgraded. These three methods are described more in-depth by the ZeppelinOS team in a blog post [208]. The main problem that has to be solved here is that of managing the storage layout so that the state variables that are needed for the proxy (such as the address of the logic contract) are not overwritten by state variables of the logic contract when it is upgraded.

Inherited Storage is the simplest approach of the complex patterns. The proxy and the logic contract both inherit from a storage contract that enforces the variable locations for the initial version of the contract and the proxy variables. When an upgrade is executed, no storage is lost, as long as the upgraded contract still inherits from the storage contract.

Eternal storage This combines the previous approach with the pattern of an eternal storage contract: A separate storage contract is created, that basically consists of mappings of variable names to data types, enabling a contract to store any variable under an identifying name. This makes it so that an upgrade to the storage structure does not influence the memory allocation of the logic or proxy contract, as all variables are stored separately, in what is essentially a key-value storage [195]. (This pattern could also be used with the manner to the simple data separation method described above, without using assembly code, improving the simple data separation pattern and enabling data upgrades.) With assembly usage, the proxy itself is the eternal storage contract, enabling full upgradeability. The main drawback of this method is that variable definitions and access have some additional overhead and that structs are quite complex to model in this approach.

Unstructured storage proxy This pattern uses the storage layout of solidity to its advantage to eliminate the need for the logic contract to implement any proxy relevant storage contracts. Any variables needed by the proxy are stored in a pseudorandom location depending on a hash of a string. Due to the hashing algorithm, the collision possibility is negligible [208], enabling the proxy contract to store all the state of the logic contracts own variables. The drawback of this method is, that one has to be careful when upgrading, to not mix the structure of the state variables which would scramble the old data layout. But this is easily done by simply letting the upgraded versions inherit from the previous versions of the contract. This enables fully upgradeable contracts without the need for the logic contracts to implement additional interfaces.

Partially upgradeable systems This is not really a method on its own, but it can be combined with other approaches. With a partially upgradeable system, key elements of the system are left within a contract that can not be changed, i.e. the ability to pay out escrowed funds in an escrow contract. This helps to build trust by ensuring that no malign updates can tamper with this essential functionality.

Of course, there are many variations of these possibilities and methods that are combinations of the above approaches, but this list covers the most important concepts.

For the development of the system, complete upgradeability was desired in order to make the system completely future proof. This leaves either simple data separation with eternal storage, or one of the three complex proxy patterns as options.

As the inherited storage pattern would have required additional parent contracts for all of the contracts that are supposed to be proxied, and the eternal storage patterns would have added

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

additional overhead code for variable access to the system that is already hard-pressed for saving contract size, it was decided to go with the unstructured storage proxy pattern.

For further reading, this pattern is again explained in-depth by zeppelinOS [210] as it is also used in their smart contract development suite as a pattern for kernel upgrades, as described in the zeppelinOS[12] whitepaper [5].

Implementing the proxy (see code A.4) shines light on a further problem: Once a contract is replaced by its proxy, the proxy naturally has none of the state variables of the original logic contract. Of course, this would normally not be a problem, as the proxy setting would occur immediately at deployment time, but in case that a contract needs state to function properly, this raises a problem. Especially the role-based access does not work this way, as the deployed proxy would then have no owner/manager/ etc. and the deployer would have no way of adding one, due to access restrictions [13].

In order to combat this, the Initializable (see code A.5) contract is used. This contract stores whether it has already been initialized or not, and offers an `initialize(address initialManager)` function. This function can only be executed once and calls an abstract internal function `_initialize(address initialManager)` in the same contract, that must be extended by subcontracts. This enables subcontracts to set their managers/owners outside of the constructor. All contracts that are supposed to be proxied and have manager functionality, should inherit from this contract.

Now the obvious vulnerability that opens up here is that of a frontrunning attack: if an attacker knows in advance when a new version of the contract is going to be deployed, he might send timed transactions with high gas prices to call the initialize method first, to hijack the proxy contract. To prevent this, the `upgradeToInit(address newImplementation)` method in the proxy contract is utilized. It immediately calls the initialize method of the contract the proxy is created for, setting the creator of the proxy.

As only a single role can be initialized in this pattern, contracts that have multiple roles attached to them needed a “dominant” role that can add addresses to the other roles. In addition, as contracts that were initially being linked together in the constructor methods were of course also lost, a role to link the contracts up was also needed. These needs were filled by the creation of the Orchestrator role, which is typically filled by the deployer and maintainer of the system.

Now, the only potential problem with this approach is that theoretically there could be a storage collision between the proxy variables and the actual logic contract storage, but according to the zeppelinOS Team, this should be virtually impossible due to the storage layout of solidity [209].

The result of all this is, that every access to a contract, even from within the system, now goes through an intermediary proxy contract, that can be upgraded seamlessly by the deployer without loss of data, fulfilling the requirement “Upgradeability”.

4.3.10 Discovery/Registry

This is the only component that was not shown in the initial diagram, mostly because it would have added too much visual clutter. As the previous subsection explained, nearly all contracts are to be deployed behind proxies. This makes finding the correct contract to interact with an obscure task

[12] ZeppelinOS is a blockchain operating system for distributed applications, aimed towards enabling developers to easily develop using a secure library of existing functionalities.

[13] In addition, all boolean checker variables must be designed in a way, that they can be uninitialized (=false in case of boolean) in the beginning, such as the boolean variable describing if the issuance is open or closed in ERC1594.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

for the typical user. To combat this, a central registry was created to provide discovery services, which also fulfills the “Discovery” requirement.

In this registry, there is a list of all contracts in the system. Defined by their name, that can also be queried for. The registry is managed by the orchestrator. The naming of the contracts in the registry is also used in the frontend to provide proper views for each contract.

So while this component is not the most complex or interesting from a technical standpoint, it still fulfills a vital function in the overall system, enabling discovery and aiding in the visual representation of the system in the UI.

4.3.11 Putting it All Together – Deployment

Packing all the components and features that were described in the previous sections together, and deploying them to the blockchain is achieved utilizing truffle and its multiple helper functionalities for the deployment process.

To deploy contracts using truffle, JavaScript files, that contain instructions on what to deploy, with what arguments and in which order, need to be placed in the migrations folder. There is also the possibility for multiple consecutive deployments, or migrations, that are then stored in the deployed Migrations contract, in order to keep track of the current migration number. This enables updating the system (without any real upgradeability in contracts, just to refer to those contracts that were previously deployed), but won’t be used in this concrete case, as we use the custom upgradeability described above.

Therefore, there is only one really relevant migration file, the deploy contracts script. At first, all contracts are deployed normally. These are the logic contracts, that no actual state will be stored in. Afterwards, starting from the contracts that have no dependencies on any other contract (e.g. Verifiers) a proxy is created by calling the createProxy registry function, which also names the proxy accordingly. Then, upgradeToInit is called using the address of the logic contract. This makes it so, that every call to a proxy contract is responded to according to the logic of the logic contract. It also immediately initializes the contract with an owner resp. orchestrator. The contracts that have dependencies on other contracts get injected with them by method calls, as the deployer has the ability to do so due to his role as orchestrator.

In the end, ownership of the TransferQueues contract is transferred to the token contract, and all other linked contracts are also set, constituting a deployed system including a VotingToken.

For real-world use, to deploy another security token, the KYC- and PEPListControllers would not need to be redeployed, but just referenced by the address of their proxy, as would the registry, as these elements are global (at least within a jurisdiction).

This produces a finished, operative, proof of concept securities system.

4.3.12 General Problems

As it is to be expected when developing a project of this size, and even more so in a programming language that has not yet reached its stable version, multiple problems with the language and/or frameworks were encountered. The following is a list of problems that affected the development, including workarounds or fixes that were used, if there are any.

Misleading error messages On some errors pertaining to invalid storage access, the compilers error message just reports an invalid opcode, meaning that a function is wrong at an assembly level. Without proper experience, it is not trivial to find the underlying error, which in the concrete case was an array that was out of bounds. Of course, it is possible that for technical

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

reasons this can not be implemented differently, but a more informative error message for such common mistakes as an array out of bounds error would still be helpful.

Need to manually delete the build folder This is not an error with solidity itself, but rather an error with truffle [137], combined with misleading error messages. The concrete problem is, that truffle often does not do a full delete on recompilation by the migration. This error manifests itself either by redeploying older versions of the contract, which makes bug hunting nigh impossible or by throwing an error that claims that a function has an invalid number of parameters. The solution to this problem is simply to manually delete the build folder before the migration, to force a recompile. This problem was especially irritating during development, as it was not immediately recognized.

Infinite mappings This item is not an error per se, it is just an inherent property of developing smart contracts in solidity, that developers, who are not used to developing smart contracts have to get accustomed to. Namely, that one can not just iterate over all token holders or any other group of addresses that can grow freely, as these might grow to a size where a single transaction that iterates over all items can not be executed without going over the gas limit, making the function unexectuable. Working around this constraint requires either the possibility to batch requests or a workaround that does not require iterations in this format. This may prove unintuitive for developers new to the blockchain.

Currentness of information Most developers heavily rely on the internet when developing a product. Tutorials, public code repositories, and forum discussions are invaluable for development. But due to the ongoing, fast-paced development to solidity itself as well as to all related development tools, much information on the internet quickly becomes stale or outdated.

This is not to mean that this is an inherent fault to any of the solidity developers, it is just a fact that such problems come with fast release cycles. It just makes searching for correct information a bit more complicated.

Lacking ease of use functionality There is very little built-in functionality that one might be used to from more mature operating languages, especially for strings and lists. There is no string equality or string.length, as well as no list functionality beyond those of an array at all. But it might very well be, that such functionalities are never implemented in order to keep solidity lean.

Constraints on arguments In the current stable version of solidity, there is no possibility to pass a struct or a list of lists (for example a list of strings) to a function from an external call.

One solution for this would be to switch on the ExperimentalAbiEncoderV2 compiler feature by adding it as a pragma at the beginning of the solidity file. But as the name already says, this feature is experimental, and while multiple contracts use it in production code, it would not be a good idea to utilize such experimental features especially in a high stakes banking environment.

Case in point, just in March of 2019, a bug in the ExperimentalAbiEncoderV2 was discovered, that could lead to corrupt parameters being sent out from a contract using this encoder if a method accepted a struct directly to storage [177].

So this leaves only the possibility of working around sending lists of strings or structs by refactoring the functions so that this is not needed, or sending all contents of the structs separately.

Declaration of memory and Storage Memory and storage are two keywords in solidity, that signify, where a variable is to be stored: In the storage of the contract as a persistent state

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

variable, or just temporarily as a memory variable, that lives as long, as the current function is executed. The various constraints on these keywords sometimes force the developer to create awkward constructs to accomplish mundane things, such as having to create a storage variable of a list of names, to create a memory struct, that can then be pushed to storage (e.g. the createBallot Function in VotingToken (see code A.3).

Missing Language Features In addition to ease of use features as mentioned above, there are also multiple features missing, that would actively reduce mistakes and increase readability and reliability of solidity code. Most notably, the inclusion of over- and underflow secure calculations is missing, as the current state leads to every contract just using safemath libraries to alleviate the problem. Another sensible addition would be override annotations (could be easily done with a simple preprocessor, without altering the bytecode), especially for projects like this one and its multiple overriding functions.

Tool support Just as the language itself, the tools used, while brilliantly constructed, still often show some bugs, impeding productivity. Examples are frequent crashes of ganache, as well as the IntelliJ solidity integration not showing compile errors correctly.

Smart contract size Arguably the most serious challenge of solidity development that had to be overcome in this project was the reoccurring problem of smart contracts size. As the system's functionality mainly consists of the token contracts, that inherit from each other and each expand upon the previous one's functionality, the code for the single resulting contract was bound to go over the maximum size for a contract sooner or later. The limit itself was set to 24kB with EIP-140 [18]. As this limit impedes the development of complex DApps, it is often criticized and proposed to be removed, but lead developers like Vitalik Buterin stand firmly by this limit [80], arguing that it enforces clean modular code, therefore it has to be assumed, that it will stay this way.

To save contract size in the project, multiple steps were undertaken. First of all, Functionality that was originally within the token contracts, was removed, especially the Controller, and the AML queue functionality (although there is still some AML functionality present within the ERC1495 contract, that could theoretically be removed). The controller extraction alone saved a lot of space as it had multiple requires with large error messages attached. Additional measures that were taken are:

- Cutting of error messages to at most 32 bytes of length, to let them fit into a single block
- Marking functions that only need to be addressed from inside the contract internal, and those that only need to be addressed from the outside external
- Removing of the VotingTokenOfficialRole, as it was not really crucial to the system
- Consolidation of used libraries

In the end, this shrinking was doable, but nonetheless slowed the development effort down, and the limit should be reconsidered, as it hampers the development of large applications[14]. Additional possibilities of shrinking contract size can be found in this blogpost by polymath [81].

An additional problem, that comes up is that, once a contract is shrunken, this restricts the number of error messages that can be put into a contract, forcing less descriptive error

[14] Of course it can be argued, that every complex system should be able to be broken down so that it fits this requirement, and as we see, it is possible, but this does not mean that this is absolutely necessary, and a larger limit would still make development easier.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

messages upon the users or even forcing developers to completely forgo error messages. In the system of this thesis, this becomes apparent when trying to send more tokens than the address owns. The revert message for such an error is the message for an underflow coming from the SafeMath library (just as the standard implementation for ERC20 does it), which may be confusing for a user.

This is not to say, that these problems[15] are here to stay, or that they all are critical to be addressed, nonetheless they make developing solidity in its current form more complex than it needs to be.

4.3.13 Potential Extensions

This section lists features or improvements, that could be made to expand the system, but were not possible in the course of this thesis due to time constraints and/or their relative unimportance. The most interesting of them will be further expanded upon in the conclusion chapter (7).

Emergency stop pattern Next to the Checks-effects-interaction pattern[16] that was applied in the system, the token could also implement the emergency stop pattern, from the group of patterns described in this paper by Wöhrer et al. [200], thus enabling the token to be shut down in case of a critical failure. This would be as trivial as inheriting from openZeppelins pausableToken [141].

Checking of the data parameter For the matters of this prototype, the data parameter for transfers, etc. is mainly included to future proof the contract. If a concrete use case or requirement comes up, it should (and could easily) be expanded to be checked.

Batch operations A practical ease-of-use addition to the system for operators would be the possibility to add addresses to verifiers in bulk, to migrate a large number of existing clients.

Expirable whitelisting The KYC component could be outfitted with expiration dates for each customer, to force updating of the data, and checking if the customer should still be KYCd. Of course, this would need to be configurable for Verifiers.

External KYC providers Support for external identity providers/standards could be added as support for the local KYC-ing of customers. Of course, this pertains more to the frontend of the system, as the relevant changes would mostly have to be included there. Examples of such identity standard providers are civic [28] and uPort [116].

Privacy The current implementation of the system does not deal with privacy concerns in a special way. The users are identified by their addresses and could theoretically be deanonymized by analyzing transactions. In order to alleviate this, external protocols like the AZTEC protocol [199], which enables confidential transactions, or mixing services could be used.

Additional features of securities Some features of different security types have not yet been implemented, such as the call rights on preferred stock, which enables the investor to redeem the stock for a predefined price at a certain time [102]. Larger expansions would be to include complex options like puts and calls into the system, to fully map the actual stock market.

[15] The described problems were the ones that most impacted the work on this thesis, further issues with solidity can be found in [174].

[16] This is a commonly recommended pattern in solidity. It refers to first checking all preconditions for a function, then applying the local state changes, and finally calling external contracts [175]. Waiting for the result of external calls for checks can lead to a reentrancy problem, as explained in the attacks on smart contracts section 2.4.5.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Stable token To provide more stability, security and trust for investors, it would also be prudent, to add a stable token to the system. This token would fulfill two very important roles:

- Enabling the comparison of trades in the AML component with actual fiat value, to comply with AML laws.
- Enabling payout in quasi-fiat money for dividends.

Of course, for this, a fitting stable currency would need to be found, and for comparison to work, an exchange would be needed where current prices could be queried.

Binding Votes Giving issuers the option to make some votes binding. Of course, this is only possible with votes that refer directly to the blockchain, like amending of dividend payouts, etc.

Two-Factor Authentication As mentioned during the validation of the Two-factor Authentication requirement, there are currently no established methods of providing this mode within a blockchain context. Such methods could theoretically be established using multisig wallets or private chains. Considering the EU Directive 2015/2366 on payment services in the internal market [59], which necessitates payment service providers to implement strong customer authentication, it might even be mandatory to implement two-factor authentication, if a system as proposed here were to fall under the legal definition of a payment service provider.

Documents extension This extension is already envisaged by the ERC-1400 family of standards whose core is implemented in this system. It enables documents for customer information to be attached to a token and could be implemented by adhering to the ERC-1643 standard [39].

Controller extension This overlaps with another extension to the standard, ERC-1644 [40]. This deals with the problem, that private keys may be lost or that legal issues may force a move (e.g. if a token holder dies and the inheritance has to be divided, or if a majority of holders have accepted a tender offer and can force the rest into the deal). But as mentioned before, this may reduce trust in the system, although it might be ultimately unavoidable in order for the system to conform with legal requirements.

Additional VotingToken Version As mentioned in the implementation description, it would be possible to implement the functionality of the DividendToken using the already present functionality of the VotingToken. Therefore, it would be achievable to create a second version of the VotingToken, that solely relies on snapshots for voting as well as dividend calculation. This would save some gas costs on transfers.

4.4 Angular Client

With the implementation of the actual smart contracts system, the original core aim of this thesis is reached. But interacting with a smart contract ABI as the only access to the system, having to send each request manually through web3.js or a similar client interface appeals neither to the average end-user nor to bank clerks. Therefore, a basic user interface for all types of users to interact with the system is needed, on the one hand to be a prototype for an eventual production system, on the other hand also to showcase the actual functionality of the system.

An exchange-like trading overview, the current value of one's portfolio, and charts are out of scope for this system, as they are the responsibility of an exchange, and there are more than enough exchanges on the market. This client mainly aims to be the start of an administrative interface for orchestrators and issuers, while still providing investors and other participants with their respective functions.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

It still has to be kept in mind, that the frontend is in no way meant to be feature complete or fit for direct use in a production environment.

There are multiple options for the development of a client frontend for a smart contract system available. Due to the main interaction method with smart contracts being the JavaScript-based web3.js, most frontends for smart contracts are based on JavaScript libraries. Popular frameworks for such clients are:

drizzle Drizzle is a library within the truffle suite, that enables the connection of the users' provider handling (how the user can connect to the blockchain with the help of MetaMask, but more about this later). In addition, it provides objects for easy access to contract data and enables the user to quickly prototype bare-bones forms for contract interaction.

react Pure react frontend deployed by a node.js server, connected by web3.js.

angular Angular frontend deployed by a node.js server, connected by web3.js.

Of course, there are as many possibilities to connect to the blockchain, as there are frontend frameworks, but in the absence of any strong indications for one frontend over the others, it eventually decided to go with angular Angular, as there was not that much of a difference between the options, but this was the Framework where the most personal experience was present. While drizzle may have enabled faster prototyping, the user interface generated in a test run left a lot to be desired, which also contributed to this decision.

4.4.1 Project Setup and Frameworks

To scaffold such a frontend, one of the best ways to start is the usage of truffle boxes. Truffle boxes[17] are sets of boilerplate applications, that can be initialized directly using the truffle command-line interface. In this project, the angular-truffle-box by the Quintor Team [161] was used as a baseline.

The user interface is served by a node.js server, which has no other functions, as the complete state of the system is stored on the blockchain. This means, there is no need to communicate with the server from the client apart from fetching the frontend resources, all communication and interaction is done from the client directly to the blockchain.

As the typical browser does not have the capabilities to directly interact with the blockchain, a so-called provider is necessary. The de-facto standard for such providers is MetaMask [124]. Therefore, this was the only wallet implementation that was particularly regarded during development in order to comply with the "Wallet applications" requirement. MetaMask is a plugin, that allows a user to send transactions signed by private key to a blockchain via a browser. But the user does not have to initiate these transactions manually, as they can be provided by the website that the user is currently visiting. This is achieved by having the MetaMask plugin inject a web3.js instance into the website. The injected instance is configured to connect to the blockchain instance that the user has currently selected in the MetaMask plugin, be it the main Ethereum blockchain, a testnet, or a private local development blockchain. The private keys to the users' address are stored only locally and encrypted, true to the decentralized spirit of distributed ledger technology. Using them requires logging into the plugin via a user-chosen password.

Concretely, the workflow is the following: A user with the MetaMask extension navigates to a site that utilizes MetaMask as a provider, optionally enters data in a form provided by the site, clicks a button, that starts the website assembling and sending a transaction on the users' behalf.

[17] www.trufflesuite.com/boxes (visited on 05/10/2019)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

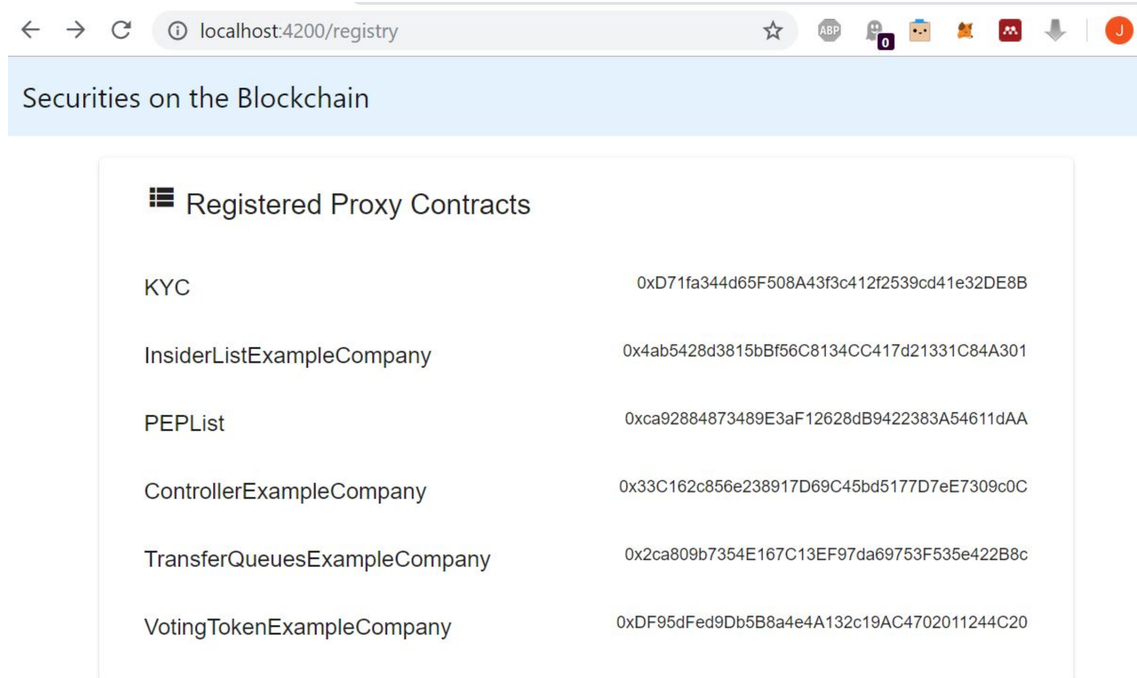

Figure 4.4: View of the initial screen of the Registry component, showing the List of deployed Contracts that is used to navigate the system

Of course, the transaction needs the users permission to be signed, so in order to ensure that no fraudulent transactions are executed without the users consent on his or her behalf, each transaction, that changes blockchain state, is shown to the user within the MetaMask plugin. It is shown which functions will be called, how much ether will be sent, and an estimate on the transaction fee that will occur. The user then has to manually accept this transaction, before it will be signed and sent by MetaMask.

Another crucial dependency for the quick development of smart contract frontends is truffle contract [191]. It is used to provide abstractions for contracts that can be interacted with in javascript code, utilizing promises and synchronized transactions.

4.4.2 Overview and Walkthrough of the System

The frontend system has a pretty simple structure: The architecture follows the standard Model-View-Controller pattern, that most angular applications adhere to. Each module corresponds to a single deployed contract. In cases where multiple files are inheriting from each other, that may or may not be all deployed, such as in the case of the tokens of the system, each individual logical contract corresponds to a component within the module.

Discovery of the token contracts is done via the registry component, which is the entry point for the system. When the registry is called, it connects to the registry contract on the blockchain[18] and queries it for the ids and addresses of the currently registered proxies.

All currently deployed contracts are shown on the user interface, presented in Figure 4.4. Each one is a clickable link, that refers to the view of the respective contract.

[18] Currently just to the last deployed one, of course, this address would need to be fixed once deployed for a real-world application.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Figure 4.5: Editing view on the Registry, that is only shown when the current user in MetaMask is an Orchestrator in the System

If the currently used address is an orchestrator in the system, i.e. it has the rights to edit the registry entries, appropriate forms to create, add and update proxies are shown. These can be used to edit the contents of the registry, as presented in Figure 4.5.

By a click on one of the elements of the list, the user is redirected to the respective contract. All of the verifiers follow the same pattern: they contain a public method checking whether a given address is white- or blacklisted, and two methods to add/remove addresses to the respective list, that is only visible for managers of the current address. An example is given by the means of the KYCVerifier in figure 4.6. This also showcases the definition of the URLs in the system: Every URL except for the registry is defined by `http://domain/contractName/address`. This means, that even without the registry if a user knows what contract he or she expects at a certain address, it can be viewed without a problem.

Likewise, the controller component also publicly shows all its verifiers as clickable list elements, as well as offering methods for setting new named as well as general verifiers, presented in figure 4.7.

The most complex and varied module is the token module. As mentioned before, it can be opened in three different modes, corresponding to the types of tokens in the system, by opening it either with the address prefixes “ERC1594”, or “DividendToken”, or “VotingToken”. Depending on this and the role of the current user, the component shows appropriate functionality, fulfilling both the “Role based access” and the “Views” requirement. In figure 4.8, all functionality is shown on collapsed panels, as a VotingToken inherits from both ERC1594 and DividendToken, all components are shown here. Showing all functionality in detail would go beyond the scope of this description, but the general process of signing a transaction will be shown at the example of transferring 10 of our tokens.

While constant functions do not require authorization from MetaMask, as they can be executed for free (provided the client has access to a local node or a similar execution environment), all other function calls need to be authorized via the MetaMask credentials, i.e. only KYC-whitelisted users can successfully send transactions, fulfilling the “Authentication” requirement. After the user clicks a button in the frontend (see Fig. 4.9), initializing the transaction, the MetaMask plugin pops up

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

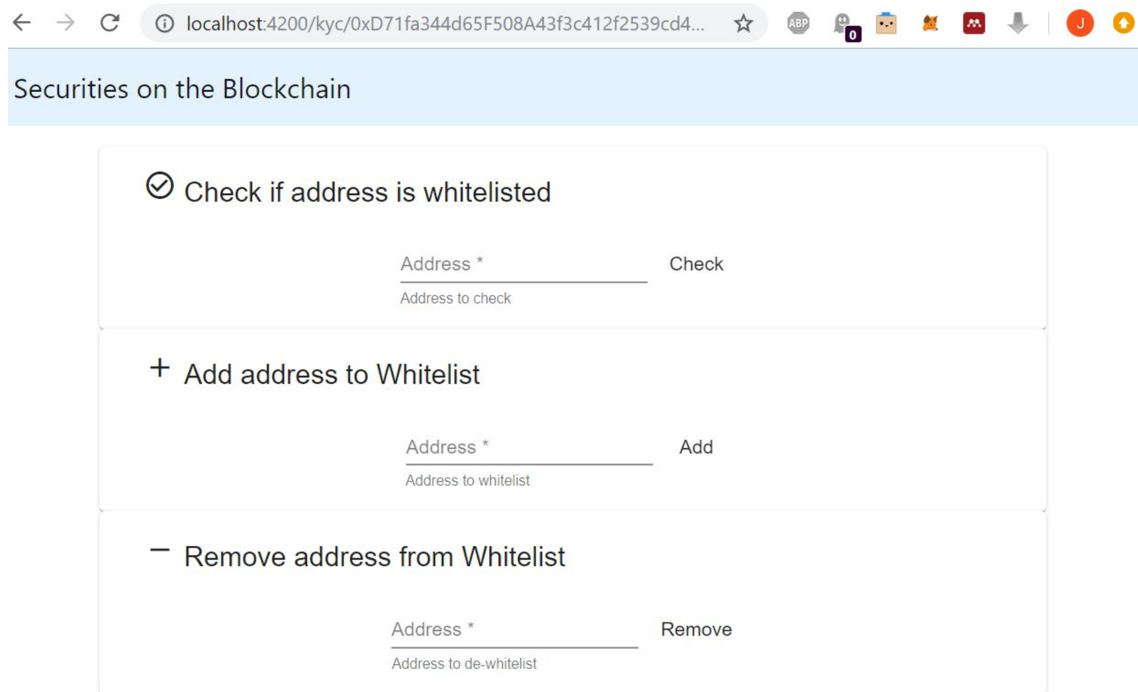

Figure 4.6: Full view of a KYCVerifier, with the current user being a KYCListManager, also showcasing the address bar.

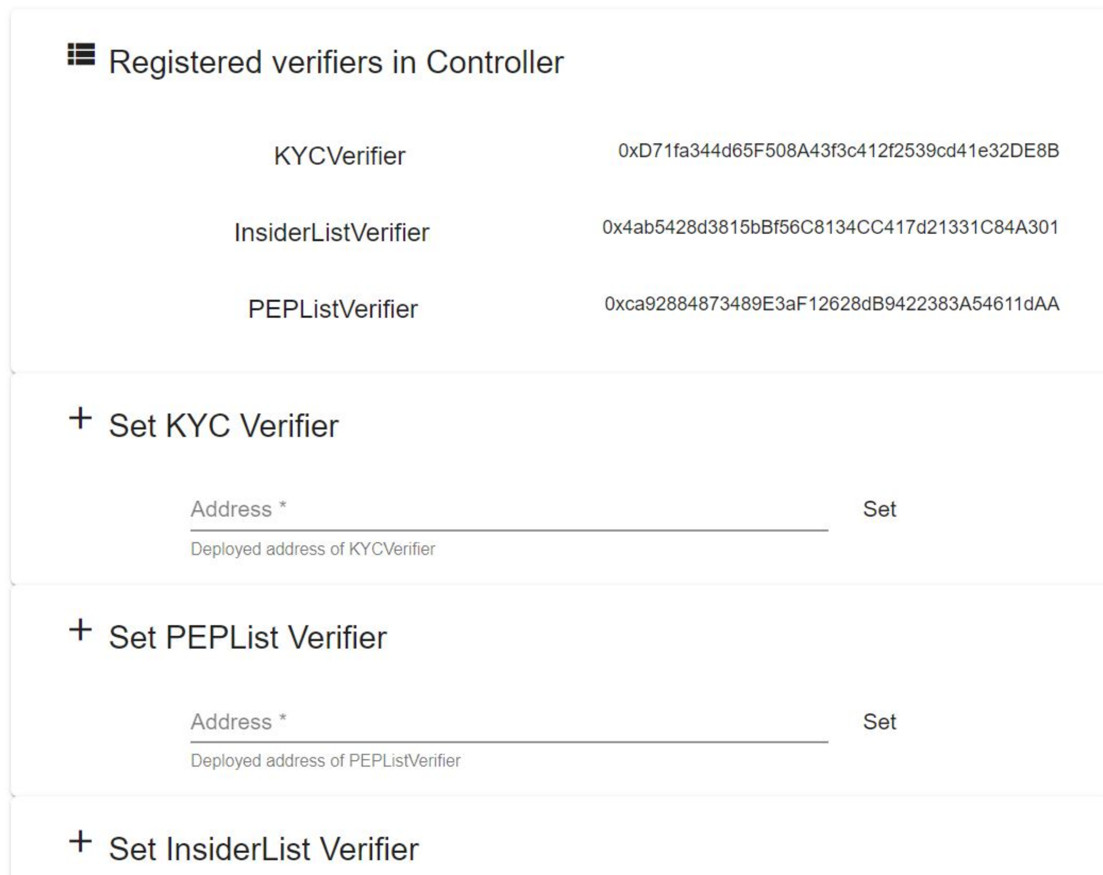

Figure 4.7: Partial view of controller, showing the list of verifiers and some setter methods

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

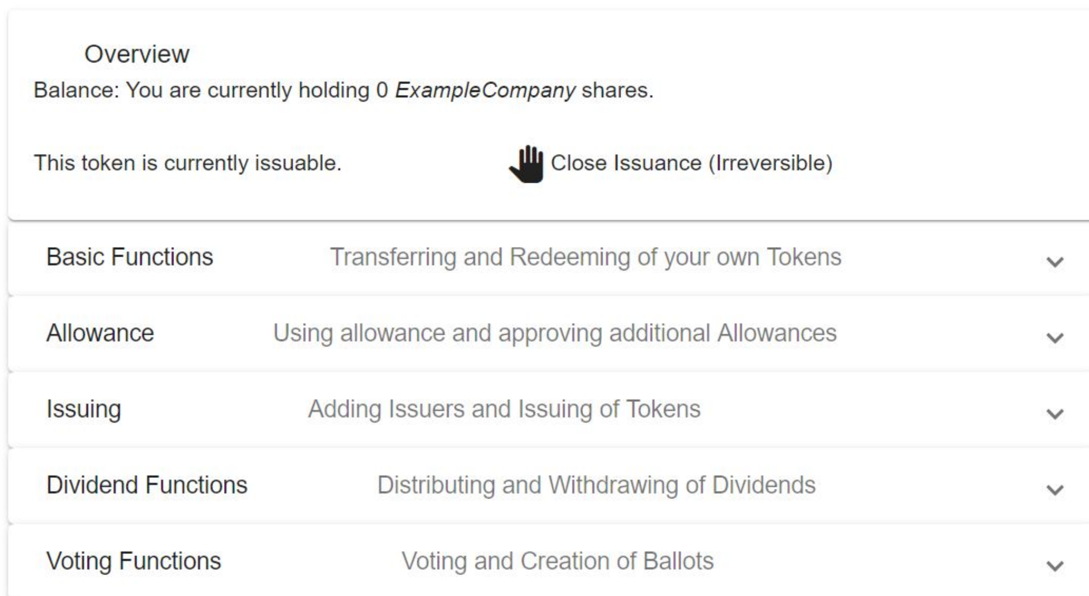

Figure 4.8: Overview of all VotingToken functionalities by an user that is an orchestrator as well as issuer, i.e. has all functionalities available.

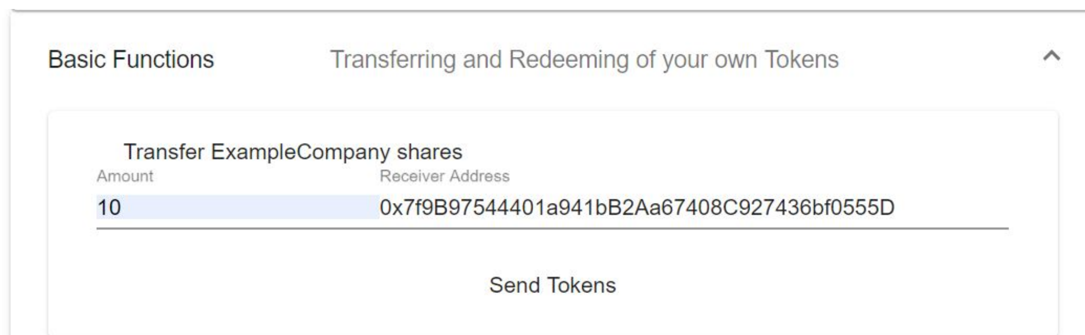

Figure 4.9: Initializing a transaction

showing all relevant information of the transaction (see Fig. 4.10) and requesting authorization. Once this is granted, the transaction is sent to the mining node. Once the transaction is mined and therefore confirmed, a snackbar element pops up, indicating success (see Fig. 4.11). In case the transaction does not succeed, the error returned by the transaction is shown.

4.4.3 Problems and Solutions

This chapter gives a short list of problems that were encountered during frontend development and describes how they were overcome.

Routing As already shown in Figure 4.6, the routing in the frontend is done by naming the type of component that the user wants to visit with the address of the contract on the blockchain as a parameter. The address is passed to the component, and it attempts to create a view using the contract at the respective address. This means, that a Contract can also be deliberately limited in the UI, if, for example, a DividendToken was opened as an ERC1594 token, only features of the ERC1594 token would be shown, but it would nonetheless be fully functional.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

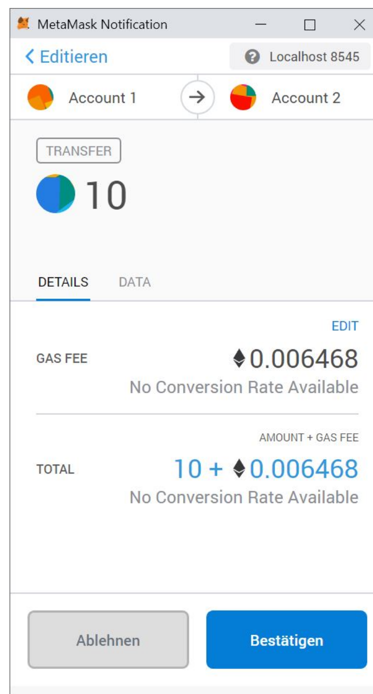

Figure 4.10: MetaMask showing the transaction that is about to be signed for the user to confirm

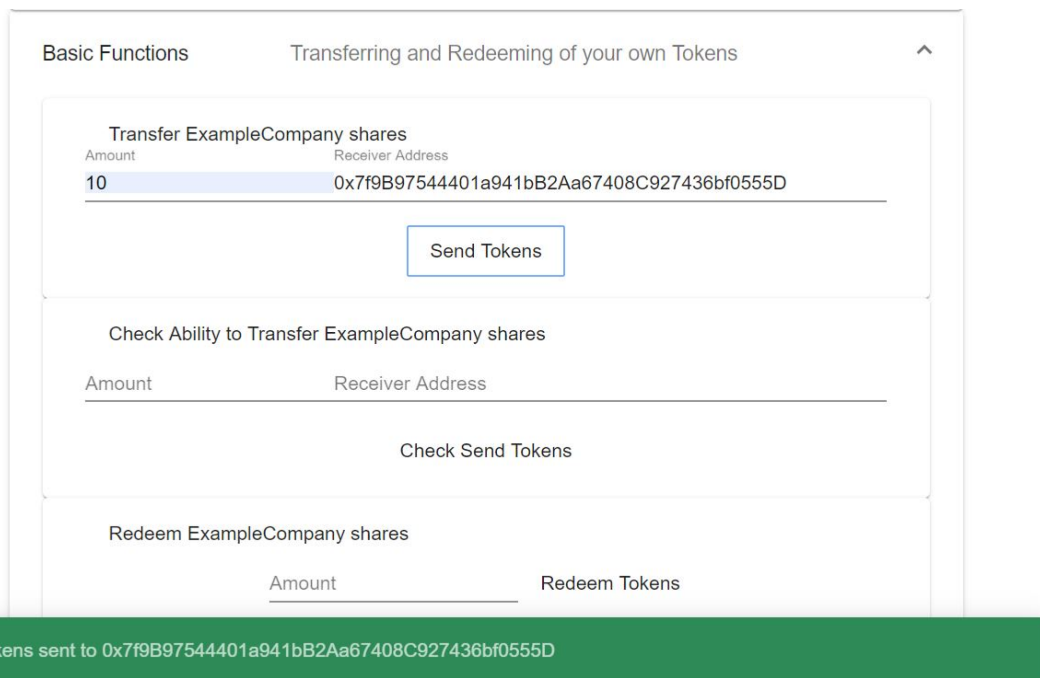

Figure 4.11: Success message popping up after the transaction is mined

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

This forces the registries naming convention to be adhered to strictly, because otherwise the contracts cannot be routed correctly.

This system makes for understandable addresses that also convey to the user exactly which contract he or she is currently interacting with.

Error messages As error messages on reverts are crucial to enable the user to understand mistakes, but are not conveniently provided by web3.js, they have to be parsed manually by searching the logs of the transaction that included the request for the relevant string that indicates a failure. This is not really a big problem, yet still a minor inconvenience that could be fixed in web3.js.

Role based access To comply with the role-based access restrictions of the smart contract system, each component that offers different levels of access requests the role of the current address of the user in the relevant contract and adjusts the view accordingly. In addition, the components are subscribed to updates on the current address by the provider, so that a change in accounts is immediately and dynamically reflected by the system.

This technique enables the system to function without the need for any server-side databases or credentials, reducing complexity and adhering to distributed principles.

Although the frontend is, as announced at the beginning of this section, not yet in a state to be published in a production environment, it still showcases all features of the underlying system, fulfilling the requirement "Completeness of smart contract mapping" and could be production-ready with some minor graphical tweaks and (a lot of) added ease of use functionality.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5 Validation of the Prototype

This chapter deals with the quality properties of the results and analyzes in how far the goals that were set at the beginning of the thesis and in the requirements could have been accomplished.

5.1 Validation of the Requirements

The following is a list of all requirements, each linked to their definition, in the same order as in the requirements chapter itself. It is described in short for each, whether it was achieved or not.

5.1.1 Smart Contracts System

Technical Functional Requirements:

Deployability The system is written in Solidity and deployable to Ethereum blockchains, as can be proved by executing the deployment script of the system.

Token Creation By deploying additional versions of the basic ERC1594 Token to the blockchain, more tokens can be produced. As each token can have an individual controller, they can be configured individually. They also can be distinguished by their address as well as their name property, fulfilling this requirement.

ERC20 The basic token contract implements the ERC-20 interface, therefore this requirement is fulfilled.

Further Token Standards The system adheres to the security token standards defined in ERC-1400, which was deemed to be the most useful, actively developed, and community-accepted standard. In particular, the substandard ERC-1594 was implemented, with the possibility to extend to implement the other substandards.

Expandable Rules The restrictions are expandable by operators of the system without changing smart contract code either by reconfiguring existing verifiers, or adding new ones. The smart contract code can also be amended using the upgradeability functionality, replacing old code to adapt to changes and expand the trading ruleset.

Incentivizing This requirement is not fulfilled. It was decided to forego this requirement as it could be very easily added, but does not necessarily belong in the base system, as a corporation deploying the system would first need to decide on the mode of monetization (which operations are to be made costly, which are free, is it a flat fee etc.).

Dividends Fulfilled by the implementation of the DividendToken (see description of the component and the linked code in 4.3.6), which implements this functionality.

Voting Fulfilled by the implementation of the VotingToken (see description of the component and the linked code in section 4.3.7), which implements this functionality.

Discovery The systems contracts can be discovered by querying the Registry contract, which enables users to interact with the system without saving all addresses.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Regulatory Functional Requirements:

KYC Trade restriction and identification (only addresses that are whitelisted by KYCManagers are able to trade, and each address is identified by its private key) are implemented in the system. However, this requirement still is only fulfilled partly, because due to privacy concerns and practicality, the real KYC data is to be stored outside of the system. I.e. a customer is identified by an address, but the mapping of an address to a name and other KYC information is to be done outside of the system, or by dedicated blockchain identity providers. Nonetheless, the basic KYC functionality is fulfilled by the KYCVerifier component, as can be proved by the relevant tests in the system.

Blacklists By the implementation of on-chain InsiderListVerifiers and PoliticallyExposedPerson-Verifiers, that can block trades from happening between blacklisted users, blacklisting is made possible. It is possible to blacklist on a national level using the PoliticallyExposedPersonVerifier, as well as on a company level with the InsiderListVerifier.

Manual Authorization This optional requirement was not fulfilled, as it would require too much time and bring not much additional functionality.

Information Not fulfilled due to time constraints. But as the system implements ERC-1594, it can be extended with its sibling standard ERC-1643 [39], that offers this functionality.

AML Mostly fulfilled, as trades are checked for large scale transfers or attempts to hide large scale transfers by splitting them up. Not completely fulfilled in so far as the system cannot access the current value of the moved stock in fiat money without an API to an exchange, and therefore can not set a monetary limit as required by AML laws. Of course, this is not to mean, that the system is ready for real-world application from an AML standpoint, but rather, that the technical base possibility for implementing AML restrictions tailored to the respective area of operation are created.

Additional features that could be implemented in the AML area are checks for additional other intervals and thresholds, as demanded by the respective jurisdiction.

A general verifier in the form of a blacklist could also be implemented to act as a flagging system for suspicious customers, as described in EU AML Directive 2015/849 Article 11 (e) and (f) [61].

As these features would add no interesting challenges from a technical standpoint and mostly be comprised of cookie-cutter solutions, none of them were added to the system.

Nonfunctional Requirements:

Upgradeability Through the use of unstructured storage proxies, the whole system is fully upgradeable, without loss of data (provided the upgrading contract inherits from the original contract).

Gas efficiency and transaction costs Gas saving measures have been applied as per static code analysis, and large operations have been broken up to amortize costs. The result is that a transaction on the VotingToken needs a gas stipend of 240000 gas[1], meaning at gas cost of respectively 3 GWei, 10 GWei and the historical average 18 GWei [55], the transaction costs are 0.00072, 0.0024 and 0.00432 Ether, which comes to around $0.12, $0.42, and $0.75 at

[1] This gas stipend rises about 1000 gas per transaction that is added within the week due to needing to iterate over the TranferQueue in the AML checks. Once a week has passed, the trades are removed, cancelling this growth.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the current valuation of Ethereum[2], easily beating out the costs for conventional transactions [73].

Secure Libraries All mathematical operations in the system are executed by safe libraries that prevent over- and underflows.

Role Based Access The roles system implements different roles for each single contract that needs differentiated access.

5.1.2 Web Client

Functional Requirements:

Authentication Although this is not explicitly addressed by the frontend, as there is no account creation there, it is implicitly solved because the frontend is aware of the role-based access protocol of the smart contract system.

Completeness of smart contract mapping All user-relevant functionalities of the smart contract system are mapped in the frontend component.

Views There are different views for each component and each role in the component.

Role based access As said before, the frontend is aware of the role-based access restrictions, and acts accordingly in showing transaction options only to entitled users.

Wallet applications As MetaMask is the de-facto standard in interacting with DApps, it was decided not to explicitly cater to alternatives. If the web3.js injection works the same way as with MetaMask, other providers are also supported.

User Information Optional requirement not fulfilled, as there was no reason for saving user data directly in the frontend.

Nonfunctional Requirements:

Encryption As no information is stored in the backend, this requirement does not apply.

Two-factor Authentication This optional requirement was not fulfilled, as without data outside of the blockchain, there is no need for additional two-factor authentication within the frontend system. Two-factor authentication within the Ethereum system, while it could be useful, does not yet have significant usable frameworks.

To sum it up, while the requirements all in all were set very optimistically, most of the obligatory ones could be fulfilled with the prototypical system, with clear paths of implementation for those that were not implemented, but still deemed a nice addition, and reasoning for those that were not implemented.

If one goes back even further in the analysis of the goals that were originally set for this thesis by the exposé, it is noticeable that several of the originally anticipated problems for this thesis were not really relevant during the prototype development or could not be addressed by the system. **Ownership** was not addressed explicitly because it can not be handled within the system and must be handled by the bank that uses the system, e.g. simply by the bank escrowing the security in

[2] Valuation as of October 2019.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the conventional system. **Whitelisting** was dealt with by the verifying components. **Auditing** and **Incentivizing** are no issues, as there is no marketplace of modules where developers can freely propose add-ons to the system, at least not within this prototype. **Currency Risk** does not have to be mitigated immediately as there is no intermediate token that is needed to buy the actual securities, whose value could fluctuate compared to fiat currency. The only risk in this regard is that of dividend payouts in ether, which can be mitigated by the inclusion of a stable token, as described in the future works section 7.2. **Oracles** are not needed for any aspect of the system but rather in the context of a future exchange or as a data input for future AML functionality, **Privacy** is addressed in so far as that no actual sensitive user data is stored on-chain, and potential ways on how to prevent deanonymization are brought forward. **KYC** is partially dealt with by the verifiers. Another issue that was initially planned but transpired to be way out of scope, is the creation of an autonomous index fund, which in itself could be the topic for a thesis[3].

The problem with this initial list of potential problems was, that they describe the edges of the system, and none represent the actual functionality of the system, as they were conceived before the system was even loosely prototyped. Therefore they are on a too high level, and some of them almost describe entire separate systems that could interact with the actual trading system. Nonetheless, they still serve as interesting thought experiments on what could theoretically still be added.

5.2 Quality Characteristics of the Implementation

This section will explore the quality characteristics of the system[4] by reference to the product quality model defined in the industry-standard ISO/IEC 25010 [93]. This does not aim to be a full-on formal evaluation but rather a concise informal overview of the respective aspects of the system to show in which sectors it fulfills its purpose, and in which some further work may be required, or rather, where the limitations of the system lie.

Of course, as the system is defined as well as constrained by its nature as a distributed blockchain application, some of its characteristics will be heavily influenced by this fact.

5.2.1 Functional Suitability

The functional suitability of the system was already discussed in the previous validation of the requirements section 5.1. In short, the core system fulfills most of the initially defined requirements, except for the implementation of securities information documents, incentivizing and manual authorization. The rationale for the disregard of each of these is described in the relevant subsections of the requirements section. The functional base for a smart contract trading system is therefore established, with special emphasis on compliance with standards and securities functionalities such as voting, dividend payments and trading restrictions.

5.2.2 Performance Efficiency

As the system is executed distributed on a blockchain, the main timing factor is not the actual execution time of the functions, but rather the time to mining. This mainly depends on the transaction fees the user is ready to pay. As resource utilization in Ethereum smart contract systems is inherently measured via the gas costs of a method, this serves as an excellent tool for this purpose. As elaborated in detail in the requirements validation section 5.1.1, with all the checks

[3] Case in point: A Thesis by Jakob Felix Schneider on the realization of an index fund of crypto assets [168]

[4] System meaning the smart contract system in this context. As the frontend is easily swapped and by no means essential to the system, and also not the main focus of the thesis, it is excluded here.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek, WIEN Your knowledge hub

and background calculations in the system, even for a fast transfer, the transfer fees come up at less than one dollar, which is much below costs in conventional systems with fees usually above $5 per trade for low cost providers [73][5].

5.2.3 Compatibility

How the compatibility of the system is assessed depends on the context that the interoperability is viewed from. From the standpoint of another smart contract system, the interoperability is good, as all public methods can be queried by other systems, the correct current contracts can be discovered, etc. Moreover, the fact that the system implements not only the ERC-20, but also the ERC-1594 standard, increases the potential for other applications to be able to interact with this system, as they can interact with a public, documented interface, that is used by multiple similar systems. However, from the standpoint of an actor outside of the blockchain where the system is deployed, interoperability is less than ideal due to the inherent properties of blockchain systems, as communication can only be done asynchronously via a blockchain client (e.g. web3.js) or a dedicated oracle.

5.2.4 Usability

The usability of the smart contracts system alone (without the frontend client, as per preset conditions) is less than stellar for users not acquainted with blockchain clients, as such systems do not have the capacity for many ease-of-use features and learnability helps. While it does have useful error messages on failure, it still requires a lot of specific knowledge to be operated correctly, as direct interaction with smart contracts requires using web3.js, supplying the ABI of the smart contract, and manually encoding values to send (as described in the Ethereum docs [49]).

Of course, in real-world use, the end-user would not interact with the system in this manner, but rather use a web interface as an intermediary. As the demo web interface developed in the course of this thesis is excluded from this evaluation, only general statements about such frontends can be made, criticizing the effort needed for interaction with web3 applications, like installing an browser extension or using a dedicated browser [96].

In sum, the usability of the system is mostly limited by its context as a distributed system on the blockchain.

5.2.5 Reliability

Reliability as defined by the mentioned ISO standard is composed of maturity, availability, fault tolerance and recoverability. As the system is new and has yet to see real world use, its maturity can be considered low. The availability of the system is high, as the computations are executed on the distributed ethereum virtual machine, which is not influenced by the constraints of a single machine breaking down. But the availability can not be considered perfect, as there have been historical instances of congestions of the system, slowing down transactions or at the very least hugely impacting gas prices, such as the 2017 cryptokitties craze [88].

As a caught error during the execution of the system leads to an revert, that resets the state of the system, recoverability for such events is no problem. However, in case that the system is in a

[5] Of course, as gas prices, as well as the value of ether compared to fiat currencies can fluctuate, and consequently may vary in the future. Therefore this can not be guaranteed due to the blockchain context of the system. In addition, it has to be conceded that the costs of maintaining such a system are more than merely the transaction costs, especially considering that the operator also aims to make a profit by providing the system. But as the base costs are so low, the point still stands that overall fees have the potential to be much lower than with conventional providers, even with a hefty profit margin.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

truly wrong state, e.g. if for some reason, the balances of the addresses are changed incorrectly, reestablishing the desired state of the system is very hard, if not impossible, barring an update of the smart contract code.

Fault tolerance is also somewhat spotty, as the return values of components are not sanity checked in other components, because execution time and code size is valuable in a solidity system, and therefore, there is no capacity for such checks.

In sum, while the system is reliable to some extent, dealing with potential faults leaves much to be desired.

5.2.6 Security

For the security analysis, each subcharacteristic according to the ISO-standard is analyzed seperately.

Confidentiality As the system is deployed on the public Ethereum blockchain, where all transactions are public, confidentiality is not given due to the constraints of this environment where all transactions are public. While users are pseudonymized, they could potentially be deanonymized, using attacks like those presented in [173]. But there are some extensions to the system that could potentially address this issue, as described in the privacy section in 4.3.13. Another possibility of providing confidentiality including confidential transactions would be the deployment on a private blockchain, which in turn would require a large rework of the system and negate some positive effects of public transactions like accountability.

Integrity Once data is written in a block secured by a blockchain, it can not be changed by an attacker that does not posses a majority of the computing power in the network without those changes being rejected by other miners. Likewise, contract execution is also controlled by other miners in the distributed execution of the contract code [19]. Therefore, the only feasible attack vector is some kind of man in the middle attack on the data integrity before it is entered into the blockchain system e.g. via malware installed on the clients computer. As this can not be controlled by the system, integrity can not be guaranteed for an untrusted client, but once data is in the system, its integrity is ensured[6].

Non-repudiation Non repudiation of a single address is given due to the blockchain environment, as transactions are recorded and their history can not be tampered with [19]. But theoretically, a user could just create a new address and claim that the original address was never his or hers to deny any action. This is where authenticity gets important.

Authenticity In this system, as all addresses need to be KYCed before being able to execute transfers, and all transfers are stored in the blockchain, the identity of the user corresponding to an address is known. Therefore, an identified user can not deny any actions initiated by the related address, except for claiming the theft of one's private key, that is used to sign transactions as proof of authenticity.

Accountability Again, as all state-changing actions are public and must be signed with an addresses private key, all transactions can be traced uniquely to the owner of the addresses private key.

[6] Barring errors and bugs within the system itself that might compromise integrity.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Of course, as security concerns are especially critical in banking systems, this small security assessment is not enough for the system to be considered production ready, and it would need to go through a thorough security evaluation before a go-live is practicable.

In sum, many of the securities issues are addressed or constrained by the properties of the underlying blockchain technology, and therefore the system exhibits the typical characteristics of Ethereum DApps in general.

5.2.7 Maintainability

Maintainability is often unattainable in such systems due to the immutability of the blockchain. But because of the added upgradeability in this system, it is not only made possible but even made relatively easy. Each contract can be switched out with another contract that extends the original contract.

In addition, the system is built in a modular fashion, where single components can be switched without issue. Because of this, the components of the system are also easily testable, as evidenced by the numerous testing suites created in the making of the system. Some of the modules are also potentially reusable, such as the verifiers, which could be easily queried by other types of tokens.

5.2.8 Portability

Portability, on the other hand, is quite poor, as the system can only be deployed to instances of Ethereum blockchains[7], hindering adaptability.

The only portings that would theoretically be possible, are those to other distributed ledgers that enable smart contracts, such as EOS, but still would most likely consist of bascially writing all contracts anew in the target system.

But it has to be conceded, that portability to contexts other than distributed ledgers is of no real use, as the system only makes sense in a distributed context.

5.3 Results of the Static Code Analysis

The smart contract system was also analyzed with the static analysis tools ethlint/solium [43] and slither [69]. While ethlint is a linter, that is mainly concerned with style guides and security issues as an afterthought, slither is a python based analysis tool, that includes reasoning for each error, that analyzes the code flow more thoroughly.

While ethlint mainly reported style errors, there were some potential problems found using slither, like unnecessary public (instead of external) methods, violations to the check-act-interaction pattern, and uninitialized variables. There were also some false positives, especially regarding the proxy implementation, as the static analyzer could of course not register what would happen in the delegation. All security-relevant reports that were not false positives were fixed. The reports of the two static analyzers after fixing all relevant issues can be found in the appendix, A.6, A.7.

[7] Or forks of the blockchain that also use solidity, like Quorum by JPMorgan Chase [98].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

6 Discussion

Every implementer of a system that utilizes blockchain technology in the context of recent hypes must face the question: "Does my system truly profit from the integration of blockchain technology, or could it be just as easily implemented without distributed ledger technology?"

In the context of this system, it is clearly the case that the technology is essential to the implementation, as it capitalizes on the often-cited potential for cost savings for blockchain applications in the banking sector [119].

But, that being said, where are the strengths of the implemented system and this thesis, and where is room for improvement?

First of all, this thesis is a novelty in the sense that it is the first scientific work[1] describing a system for trading securities on the blockchain catered to retail banks which, if such systems are to be adopted, is a likely way this can be achieved. However, it must be conceded that it is certainly possible, if not even highly likely, that similar projects are being developed behind closed doors in the R&D divisions in big retail banks.

Another strong point of the system is its feature richness. Between the plugin points for regulation, the modularity, voting functionality, dividend functionality, etc., most of the important functionalities for such a system are implemented in the prototype presented in this thesis.

The upgradeability of the system is also a big plus, as it is not yet standard in distributed apps and helps to remove potential bugs if any were to occur, increasing maintainability.

The adherence to community standards like ERC-20 and ERC-1594 makes it easy for the system to be interacted with using standard wallet and exchange solutions and also improves interoperability with possible other implementations.

Of course, the system also has some clear limitations and shortcomings that could be expanded upon:

The most severe limitation to the system with regards to being used in a production environment is that although it has the capabilities to comply with the most common banking laws, and can be extended in case of laws that were not considered, it is not clear whether that would be enough to be approved for a banking license in strict or conservative regulatory environments. Additionally, before the system could be used in production, those capabilities would of course also have to be realized fully with legal aid for the concrete jurisdiction it is to be deployed to. But as stated at the beginning of this thesis, such finer juristic matters are not explicitly considered here.

Although there is a lot of functionality implemented in the system, some issues could still be added, like the implementation of additional standards from the ERC-1400 family and the addition of a stable token. This will be discussed further in the following chapter on future work (see: 7). Speaking of functionality, the system would of course also need a more visually appealing customer-facing frontend if it were to be made publicly accessible, as the frontend presented here is geared towards demonstration purposes rather than real-world use.

[1] According to the literature review executed in the course of the thesis and our best knowledge.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Some purists might also criticize the system for implementing upgradeability: While it does help with maintainability, it removes the trustlessness, as the orchestrator might swap the code at will. But as this system is to used by retail banks, that escrow the securities in the conventional trading system, it requires trust no matter how it is implemented, making this point rather doubtful.

Another potential point of criticism for the thesis itself is, that it relatively often resorts to citing literature and projects produced by industry players, which might arguably not be held to the same academic rigor as peer-reviewed papers. As explained in the state of the art chapter 3, the reason for this is that the amount of scientific work done in the field of tokenized assets is plainly too low, and that many interesting, innovative projects are only backed by a whitepaper and blogposts of the implementers rather than a journal article.

As mentioned before, there is not really a similar system published, those that were found that share some general traits in so far as that they also constitute security tokens were already described in the relevant section in the state of the art chapter (see 3.3.2), namely Harbors R-Token, Securitizes DS-Token, and Polymaths ST-20 Token.

Compared with these systems there are of course several similarities. What they all share is the notion of some kind of verifier that is consulted during some interactions with the token contract, that can decide whether the interaction is allowed or not.

Considering Harbors R-Token, the similarities end here. As the R-Token system heavily relies on off-chain verification, and in itself is very simplistically designed, there is not much to be compared to here. Moving on to the other two systems, while not much can be said about the implementation of the DS-Token, as it is close-sourced, it can be deduced from analyzing the open-sourced interface methods that there are additional controlling methods, such as the seizure of tokens, as well as functions linking multiple wallet addresses together to a single investor. This functionality, while enabling some freedom for the investors to work with multiple wallets, potentially unnecessarily complicates things in the back-end. Apart from these points however, not that much can be said about the differences.

Now with the last token, the ST-20 token, there are a lot of similarities, which is understandable considering that they are based on the same standard, the ERC-1400 family of token standards, whose development was heavily supported by polymath. Therefore, the general token interface is quite similar. It also historizes balances, but not for voting or dividends, but just for data-keeping. Another similarity is that upgradeability is also implemented within polymaths system, although there it was opted for another approach, as all proxies have separate contracts and utilize an eternal storage solution, and are not handled by a one-size-fits-all approach as in our implementation. Areas, where the ST-20 token is ahead, are the deployment of new tokens directly from the blockchain, the usage of oracles and stablecoins to check the worth of tokens, and the implementation of a full marketplace, where tokens can be expanded with modules from third party developers . But just as the other security token implementations, the ST-20 has not got the necessary features for a security token that is to be deployed representing conventional securities, as this is not the focus of those systems, since they focus on helping companies raise capital themselves instead of aiming to bring more securities in general to the blockchain.

To conclude, although there is not a strong scientific base to draw from due to lack of formal works and literature in the area this thesis fulfilled its goal of creating a system that can serve as a base application for distributing security tokens on an Ethereum blockchain.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

7 Conclusion and Future Work

7.1 Conclusion

Blockchain technology has seen a huge surge of projects in recent years, fueled by hype, some with significant merit, some that are mere cash grabs by crowdsale. The potential of this technology for banking applications was soon discovered by mainstream studies [119]. An especially interesting aspect of this is the potential for savings in both costs and time within the global securities trade. As there is currently no solution aimed at bringing conventional securities to a blockchain environment in order to capitalize on this potentials, the research question for this thesis was posed as follows:

> What properties must a system for the secure and transparent management of securities on the blockchain feature, and how can a prototype, that fulfills these requirements, be implemented?

This question was answered by an extensive review of projects that tackle similar projects and existing literature (chapter 3), which resulted in the Requirements (section 4.2). These were in turn used to create the core of this thesis, a smart contract system that enables retail banks to create tradeable tokens corresponding to conventional securities.

The system consists of multiple smart contracts created in solidity, that are to be deployed on an Ethereum blockchain. Emphasis is laid on enabling the system to internally deal with standard checks that normally bog the processing of security trades down. To do this, the system only allows whitelisted addresses, has the capability of enforcing anti money laundering rules, and is able to blacklist insiders of a certain company for this special token, or politically exposed persons in general. The system is designed to be extensible, via configuration as well as via smart contract extension.

For such a system to be successful, it is also imperative to support commonly agreed upon, public standards. The basic token standard ERC-20 is a given, but the system also implements parts of the ERC-1400 suite of standards, as it was deemed the most comprehensive, and most widely accepted option [41]. This gives the system plenty of interoperability with other applications.

Furthermore, the system is also designed in an upgradeable way, meaning smart contract code can be switched out to fix bugs or apply updates, enabling better maintainability. This was achieved using the technique of unstructured storage [209].

A basic UI based on AngularJS was also created (see section 4.4) to demonstrate how clients can interact with the system, and to visualize the system to a wider audience.

The end result of the thesis is a system, that could be used as a base for a realistic real world application of blockchain technology, documented in scientific fashion, that hopefully might inspire future projects furthering adoption of blockchain technology in the banking sector.

7.2 Future Work

This thesis, while encompassing many desired features of a smart contract system for the trade of securities, still covers merely a small subsection of the potential functionalities of such a system.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Therefore, it can be expanded in many different avenues, depending on the legal situation as well as the market-dependent circumstances of the implementing bank. Some of those potential additions that were defined in the extensions subsection 4.3.13 will be elaborated here.

Privacy

To prevent deanonymization by an analysis of public transactions of the pseudonymized accounts in the system, additional measures have to be taken. A first step would be to utilize external protocols like the AZTEC protocol [199], which enables confidential transactions. For this, an AZTEC implementation would need to be attached to the actual token, enabling the conversion of our tokens into privacy-preserving tokens. However, whether this is really possible this easy without leaking any data to full nodes, e.g. via the historized balances in the VotingToken is an entirely different question, that could theoretically work as a research question for a further paper. It is also entirely possible, that the system in its current form including all features is not possible to be anonymized fully, in which case an update to relevant parts of the smart contract system would have to be executed.

Stable Token

To correctly calculate the current worth of securities to comply with AML restrictions as well as for paying out dividends in a non-fluctuating currency, the addition of a stable token and an oracle for security prices is needed. The stable currency would need to be chosen wisely, with special consideration for availability and trust in the currency. This extension would also benefit from interfacing with exchanges in order to query current prices while still considering the potential dangers of race conditions and frontrunning posed by interfacing with an exchange.

Additional VotingToken Version and Refactoring

Next to implementing the functionality of the DividendToken using the already present functionality of the VotingToken, relying solely on snapshots for voting as well as dividend calculation as described previously, other refactorings could also be applied. Theoretically, it would be possible to outsource most of the dividend and voting functionality in external contracts, in order to alleviate the contract size problem. But if there is any merit to this idea is not sure, it is possible that this would lead to intolerable gas increases, or would not be feasible due to some unforeseen implementation details. It could nonetheless certainly be tried as an experiment.

Theoretical Works

Of course, next to the expansions regarding the practical applications of the system, there are also academic possibilities that have been uncovered in the course of this thesis. In general, the literature on blockchain topics is lacking, to say the least, and some work along the lines of a systematic mapping study on projects would have been helpful during writing of this thesis, as there is no real state of the art report published. Of course, this might also be due to the fast pacedness of this problem space, which would make extensive studies of this kind obsolete quickly, which could be part of the reason why no such study exists.

All in all, as evidenced by the numerous suggestions, there is a plethora of starting points for further work in this field.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Scientific References and Bills

[7] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. *A survey of attacks on Ethereum smart contracts*. Ed. by Matteo Maffei and Mark Ryan. Vol. 10204. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 164–186. ISBN: 978-3-662-54454-9. DOI: 10.1007/978-3-662-54455-6. URL: http://link.springer.com/10.1007/978-3-662-54455-6 (visited on 05/10/2019).

[10] Massimo Bartoletti and Livio Pompianu. „An empirical analysis of smart contracts: platforms, applications, and design patterns“. In: *International Conference on Financial Cryptography and Data Security* (Mar. 2017). DOI: 10.1007/978-3-319-70278-0. arXiv: 1703.06322. URL: http://dx.doi.org/10.1007/978-3-319-70278-0 (visited on 05/10/2019).

[15] Thomas Bocek et al. „Blockchains Everywhere -A Use Case of Blockchains in the Pharma Supply-Chain“. In: *IFIP/IEEE Symposium on Integrated Network and Service Management*. IEEE, 2017, pp. 772–777. ISBN: 9783901882890. URL: http://dl.ifip.org/db/conf/im/im2017exp/119.pdf (visited on 05/10/2019).

[22] Fran Casino, Thomas K Dasaklis, and Constantinos Patsakis. „A systematic literature review of blockchain-based applications: Current status, classification and open issues“. In: *Telematics and Informatics* 36 (Mar. 2019), pp. 55–81. ISSN: 07365853. DOI: 10.1016/j.tele.2018.11.006. URL: https://doi.org/10.1016/j.tele.2018.11.006 (visited on 05/10/2019).

[23] Joanna Diane Caytas. „Developing Blockchain Real-Time Clearing and Settlement in the EU, U.S., and Globally“. In: *Columbia Journal of European Law: Preliminary Reference (June 22, 2016)* (2016), pp. 1–11. URL: https://ssrn.com/abstract=2807675 (visited on 05/10/2019).

[27] Yan Chen. „Blockchain tokens and the potential democratization of entrepreneurship and innovation“. In: *Business Horizons* 61.4 (July 2018), pp. 567–575. ISSN: 00076813. DOI: 10.1016/j.bushor.2018.03.006. URL: https://doi.org/10.1016/j.bushor.2018.03.006 (visited on 05/10/2019).

[33] Emiliano De Cristofaro et al. „A Comparative Usability Study of Two-Factor Authentication“. In: *Network and Distributed System Security (NDSS) Symposium*. Sept. 2014. arXiv: 1309.5344. URL: https://arxiv.org/pdf/1309.5344.pdf (visited on 05/10/2019).

[35] Giuseppe Destefanis et al. „Smart contracts vulnerabilities: a call for blockchain software engineering?“ In: *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE, Mar. 2018, pp. 19–25. ISBN: 978-1-5386-5986-1. DOI: 10.1109/IWBOSE.2018.8327567. URL: http://ieeexplore.ieee.org/document/8327567/ (visited on 05/10/2019).

[58] European Parliament. *Directive 2014/57/EU of the European Parliament and of the Council of 16 April 2014 on criminal sanctions for market abuse (market abuse directive)*. 2014. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32014L0057 (visited on 05/10/2019).

[59] European Parliament and Council of the European Union. *EU Directive 2015/2366 on payment services in the internal market*. 2015. URL: http://data.europa.eu/eli/dir/2015/2366/oj (visited on 05/10/2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[60] European Parliament and Council of the European Union. *MIFID II, Directive 2014/65/EU of 15 May 2014*. 2014. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32014L0065 (visited on 05/10/2019).

[61] European Parliament and the Council. *DIRECTIVE (EU) 2015/849*. 2015. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:02015L0849-20180709 (visited on 05/10/2019).

[62] European Parliament and the Council. „European Market Infrastructure Regulation". In: *Official Journal of the European Union* 648/2012 (2012). URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32012R0648 (visited on 05/10/2019).

[63] European Parliament and the Council. *GDPR - Regulation (EU) 2016/679*. 2016. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:L:2016:119:FULL (visited on 05/10/2019).

[64] European Parliament and the Council. *Regulation (EU) No 1286/2014 of the European Parliament and of the Council of 26 November 2014 on key information documents for packaged retail and insurance-based investment products (PRIIPs)*. 2014. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32014R1286 (visited on 05/10/2019).

[65] European Parliament and the Council. *REGULATION (EU) No 909/2014 - CSDR*. 2014. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32014R0909 (visited on 05/10/2019).

[68] Kurt Fanning and David P. Centers. „Blockchain and Its Coming Impact on Financial Services". In: *Journal of Corporate Accounting & Finance* 27.5 (July 2016), pp. 53–57. ISSN: 10448136. DOI: 10.1002/jcaf.22179. URL: http://doi.wiley.com/10.1002/jcaf.22179 (visited on 05/10/2019).

[72] Hisham S Galal and Amr M Youssef. „Verifiable Sealed-Bid Auction on the Ethereum Blockchain". In: *International Conference on Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2018, pp. 265–278. URL: https://eprint.iacr.org/2018/704.pdf (visited on 05/10/2019).

[76] Paul Niculescu-mizil Gheorghe and Bogdan Ţigănoaia. „Blockchain and smart contracts in the music industry – streaming vs. downloading". In: *International Conference on Management and Industrial Engineering*. Bucharest, 2017, pp. 215–229. URL: https://search.proquest.com/docview/1990422971/fulltextPDF/3695ADF4C3994A7CPQ/ (visited on 05/10/2019).

[83] Stuart Haber and W Scott Stornetta. „How to Time-Stamp a Digital Document". In: *Advances in Cryptology-CRYPTO' 90*. Ed. by Alfred J Menezes and Scott A Vanstone. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 437–455. ISBN: 978-3-540-38424-3. URL: https://link.springer.com/content/pdf/10.1007/3-540-38424-3_32.pdf (visited on 05/10/2019).

[94] Olly Jackson. „US or Swiss approach for EU crypto regulation?" In: *International Financial Law Review* (2018), pp. 1–5. ISSN: 0262-6969. URL: https://search.proquest.com/openview/949af914d834628fa263a1e90897daf6 (visited on 05/10/2019).

[99] Gerti Kappel. *Web engineering : the discipline of systematic development of web applications*. John Wiley & Sons, 2006, p. 366. ISBN: 9780470015544.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[100] Elena Karafiloski and Anastas Mishev. „Blockchain solutions for big data challenges: A literature review“. In: *IEEE EUROCON 2017 -17th International Conference on Smart Technologies*. July. IEEE, July 2017, pp. 763–768. ISBN: 978-1-5090-3843-5. DOI: 10.1109/EUROCON.2017.8011213. URL: http://ieeexplore.ieee.org/document/8011213/ (visited on 05/10/2019).

[105] Stefan Konst. „Sichere Log-Dateien auf Grundlage kryptographisch verketteter Einträge“. MA thesis. Technische Universität Braunschweig, 2000. URL: https://publikationsserver.tu-braunschweig.de/servlets/MCRFileNodeServlet/dbbs_derivate_00043715/konst_seclog.pdf (visited on 05/10/2019).

[109] Kari Korpela, Jukka Hallikas, and Tomi Dahlberg. „Digital Supply Chain Transformation toward Blockchain Integration“. In: *Proceedings of the 50th Hawaii international conference on system sciences*. 2017. ISBN: 9780998133102. URL: http://hdl.handle.net/10125/41666 (visited on 05/10/2019).

[117] Loi Luu et al. „Making Smart Contracts Smarter“. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*. New York, New York, USA: ACM Press, 2016, pp. 254–269. DOI: 10.1145/2976749.2978309. URL: http://dl.acm.org/citation.cfm?doid=2976749.2978309 (visited on 05/10/2019).

[121] Mike Mannion and Barry Keepence. „SMART requirements“. In: *ACM SIGSOFT Software Engineering Notes* 20.2 (1995), pp. 42–47. ISSN: 01635948. DOI: 10.1145/224155.224157. URL: https://www.win.tue.nl/~wstomv/edu/2ip30/references/smart-requirements.pdf (visited on 05/10/2019).

[136] Hidde Lycklama Nijeholt, Joris Oudejans, and Zekeriya Erkin. „DecReg“. In: *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts - BCC '17*. New York, New York, USA: ACM Press, Apr. 2017, pp. 29–34. ISBN: 9781450349741. DOI: 10.1145/3055518.3055529. URL: http://dl.acm.org/citation.cfm?doid=3055518.3055529 (visited on 05/10/2019).

[138] Benedikt Notheisen, Magnus Gödde, and Christof Weinhardt. *Trading Stocks on Blocks - Engineering Decentralized Markets*. Ed. by Alexander Maedche, Jan vom Brocke, and Alan Hevner. Vol. 10243. Lecture Notes in Computer Science. Karlsruhe: Springer International Publishing, 2017, pp. 468–473. ISBN: 978-3-319-59143-8. DOI: 10.1007/978-3-319-59144-5. URL: http://link.springer.com/10.1007/978-3-319-59144-5 (visited on 05/10/2019).

[145] Philipp Paech. „The Governance of Blockchain Financial Networks“. In: *The Modern Law Review* 80.6 (2017), pp. 1073–1110. ISSN: 00267961. DOI: 10.1111/1468-2230.12303. URL: http://doi.wiley.com/10.1111/1468-2230.12303 (visited on 05/10/2019).

[149] Gareth William Peters and Efstathios Panayi. „Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money“. In: *Banking beyond banks and money*. Springer, 2016, pp. 239–278. DOI: 10.1007/978-3-319-42448-4_13. arXiv: 1511.05740. URL: http://arxiv.org/abs/1511.05740 (visited on 05/10/2019).

[162] Max Raskin. „The law and legality of smart contracts“. In: *Georgetown Law Technology Review* 305 (2017). URL: https://perma.cc/UC8L-KTW3 (visited on 05/10/2019).

[165] Reade Ryan and Mayme Donohue. „Securities on Blockchain“. In: *The Business Lawyer* 73.1 (2017), pp. 85–108. ISSN: 00076899. URL: https://www.huntonak.com/images/content/3/5/v2/35271/ABA-The-Business-Lawyer-Securities-on-Blockchain.pdf (visited on 05/10/2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

[167] Manuel Schlegel, Liudmila Zavolokina, and Gerhard Schwabe. „Blockchain Technologies from the Consumers' Perspective: What Is There and Why Should Who Care?" In: *Proceedings of the 51st Hawaii International Conference on System Sciences*. 2018. ISBN: 9780998133119. DOI: 10.24251/HICSS.2018.441. URL: http://hdl.handle.net/10125/50329 (visited on 05/10/2019).

[168] Jakob Felix Schneider. „Theoretische und praktische Smart Contracts - Realisierung eines Investmentfonds". MA thesis. Technische Universität Wien, 2018. URL: http://repositum.tuwien.ac.at/obvutwhs/download/pdf/2870072 (visited on 05/10/2019).

[169] Franklin Schrans, Susan Eisenbach, and Sophia Drossopoulou. „Writing safe smart contracts in Flint". In: *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming - Programming'18 Companion*. New York, USA: ACM Press, 2018, pp. 218–219. ISBN: 9781450355131. DOI: 10.1145/3191697.3213790. URL: http://dl.acm.org/citation.cfm?doid=3191697.3213790 (visited on 05/10/2019).

[173] Murali Siddharth. „Using Ethereum Transactions to deanonymize Senders". MA thesis. University of Illinois, 2018. URL: https://www.ideals.illinois.edu/bitstream/handle/2142/101616/MURALI-THESIS-2018.pdf?sequence=1 (visited on 05/10/2019).

[182] Utamchandani Tulsidas Tanash. „Smart Contracts From a Legal Perspective". MA thesis. University of Alicante, 2018. URL: https://rua.ua.es/dspace/bitstream/10045/78007/1/Smart_Contracts_from_a_Legal_Perspective_Utamchandani_Tulsidas_Tanash.pdf (visited on 05/10/2019).

[197] Eric Wall and Gustaf Malm. „Using Blockchain Technology and Smart Contracts to Create a Distributed Securities Depository". MA thesis. Faculty of Engineering, LTH, Lund University, 2016, pp. 1–90. URL: http://www.eit.lth.se/sprapport.php?uid=987 (visited on 05/10/2019).

[200] Maximilian Wöhrer and Uwe Zdun. „Smart Contracts: Security Patterns in the Ethereum Ecosystem and Solidity". In: *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE, 2018. ISBN: 9781538659861. URL: https://eprints.cs.univie.ac.at/5433/7/sanerws18iwbosemain-id1-p-380f58e-35576-preprint.pdf (visited on 05/10/2019).

[203] Tong Wu and Xiubo Liang. „Exploration and practice of inter-bank application based on blockchain". In: *2017 12th International Conference on Computer Science and Education (ICCSE)*. IEEE, Aug. 2017, pp. 219–224. ISBN: 978-1-5090-2508-4. DOI: 10.1109/ICCSE.2017.8085492. URL: http://ieeexplore.ieee.org/document/8085492/ (visited on 05/10/2019).

[204] Min Xu, Xingtong Chen, and Gang Kou. „A systematic review of blockchain". In: *Financial Innovation* 5.1 (2019). ISSN: 21994730. DOI: 10.1186/s40854-019-0147-z.

[205] Jesse Yli-Huumo et al. „Where Is Current Research on Blockchain Technology?—A Systematic Review". In: *PLOS ONE* 11.10 (Oct. 2016). Ed. by Houbing Song. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0163477. URL: https://dx.plos.org/10.1371/journal.pone.0163477 (visited on 05/10/2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Whitepaper and Other References

[1] Hassan Abdel-Rahman. *Upgradable Contracts in Solidity - Cardstack - Medium*. 2018. URL: https://medium.com/cardstack/upgradable-contracts-in-solidity-d5af87f0f913 (visited on 05/10/2019).

[2] Noelle Acheson. *Security Tokens vs. Tokenized Securities: It's More Than Semantics*. 2019. URL: https://www.coindesk.com/security-tokens-vs-tokenized-securities-its-more-than-semantics (visited on 05/10/2019).

[3] Alt.Estate Team. *Alt.Estate Whitepaper v 1.15*. Tech. rep. 2018. URL: https://alt.estate/upload/files/altestate_whitepaper.pdf (visited on 05/10/2019).

[4] Andreas M Antonopoulos and Gavin Wood. *Mastering ethereum: building smart contracts and dapps*. O'Reilly Media, 2018. URL: https://github.com/ethereumbook/ethereumbook (visited on 05/10/2019).

[5] Manuel Araoz et al. *zeppelin os: An open-source, decentralized platform of tools and services on top of the EVM to develop and manage smart contract applications securely*. 2017. URL: https://openzeppelin.com/assets/zeppelin_os_whitepaper.pdf (visited on 05/10/2019).

[6] ASX. *CHESS Replacement: New Scope and Implementation Plan*. 2018. URL: https://www.asx.com.au/documents/public-consultations/chess-replacement-new-scope-and-implementation-plan.pdf (visited on 05/10/2019).

[8] Emma Avon. *The DAO hack - what happened and what followed? | CoinCodex*. 2017. URL: https://coincodex.com/article/50/the-dao-hack-what-happened-and-what-followed/ (visited on 05/10/2019).

[9] Prableen Bajpai. *How Stock Exchanges Are Experimenting With Blockchain Technology*. 2017. URL: https://www.nasdaq.com/articles/how-stock-exchanges-are-experimenting-blockchain-technology-2017-06-12 (visited on 05/10/2019).

[11] Zoë Bernard. *Neufund launches global decentralized currency exchange in Malta - Business Insider Deutschland*. 2018. URL: https://www.businessinsider.de/neufund-binance-global-decentralized-currency-exchange-malta-2018-7 (visited on 05/10/2019).

[12] BitcoinWiki Team. *ERC-20 Token Standard*. 2019. URL: https://de.bitcoinwiki.org/wiki/ERC20 (visited on 05/10/2019).

[13] Bitrent Team. *BitRent Whitepaper*. 2017. URL: https://icorating.com/upload/whitepaper/VeXcr2hJElsaRbDUf9cLwktAUfC8HEUG4oRsA5G0.pdf (visited on 05/10/2019).

[14] Arisa Amano Bob Remeika and David Sacks. *The Regulated Token (R-Token) Standard*. 2018. URL: https://harbor.com/rtokenwhitepaper.pdf (visited on 05/10/2019).

[16] Scott Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. Tech. rep. Internet Requests for Comments, 1997. URL: https://www.ietf.org/rfc/rfc2119.txt (visited on 05/10/2019).

[17] Vitalik Buterin. *A Prehistory of the Ethereum Protocol*. 2017. URL: https://vitalik.ca/general/2017/09/14/prehistory.html (visited on 05/10/2019).

[18] Vitalik Buterin. *Contract code size limit · Issue #170 · ethereum/EIPs*. 2016. URL: https://github.com/ethereum/EIPs/issues/170 (visited on 05/10/2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[19] Vitalik Buterin. *Ethereum White Paper*. 2013. URL: https://github.com/ethereum/wiki/wiki/White-Paper (visited on 05/10/2019).

[20] Federico Cardoso. *Maecenas Dutch Auction Smart Contract*. 2018. URL: https://github.com/MaecenasArt/Dutch-Auction (visited on 05/10/2019).

[21] Ramona Carr. *GDPR Compliance and Encryption Requirements to Know About | Zettaset*. 2018. URL: https://www.zettaset.com/blog/gdpr-compliance-encryption-requirements/ (visited on 05/10/2019).

[24] Diana Chan et al. „The securities custody industry“. In: *European Central Bank - Occasional Paper Series* 68 (2007). URL: https://www.econstor.eu/bitstream/10419/154521/1/ecbop068.pdf (visited on 05/10/2019).

[25] James Chen. *Common Stock Definition*. 2019. URL: https://www.investopedia.com/terms/c/commonstock.asp (visited on 05/10/2019).

[26] James Chen. *Dutch Auction Definition*. 2017. URL: https://www.investopedia.com/terms/d/dutchauction.asp (visited on 05/10/2019).

[28] Civic Team. *Civic Whitepaper*. Tech. rep. 2018. URL: https://tokensale.civic.com/CivicTokenSaleWhitePaper.pdf (visited on 05/10/2019).

[29] David Creer et al. *Proving Ethereum for the Clearing Use Case Emerald Performance Testing Technical Paper*. Tech. rep. Royal Bank of Scotland, 2016. URL: https://cdn.relayto.com/media/files/lBFCJiHESx64mWhBAlzs_emeraldTechnicalPaper.pdf (visited on 05/10/2019).

[30] Crypto Kitties Team. *Crypto Kitties Technical Details*. 2019. URL: https://www.cryptokitties.co/Technical-details (visited on 05/10/2019).

[31] Wei Dai. *b-money*. 1998. URL: http://www.weidai.com/bmoney.txt (visited on 05/10/2019).

[32] DaoStack Team. *DaoStack - An Operating System for Collective Intelligence*. Tech. rep. 2018. URL: https://daostack.io/wp/DAOstack-White-Paper-en.pdf (visited on 05/10/2019).

[34] Deedcoin Team. *Deedcoin Whitepaper*. Tech. rep. 2018. URL: https://www.deedcoinlaunch.com/documents/Deedcoin%20White%20Paper.pdf (visited on 05/10/2019).

[36] Dexaran. *ERC223 token standard*. 2017. URL: https://github.com/ethereum/EIPs/issues/223 (visited on 05/10/2019).

[37] Carlos Domingo, Shay Finkelstein, and Jorge Serna. *DS Protocol - Securitize ' s Digital Ownership Architecture for Complete Lifecycle Management of Digital Securities*. 2018. URL: https://securitize.sfo2.digitaloceanspaces.com/whitepapers/DS-Protocolv1.0.pdf (visited on 05/10/2019).

[38] Adam Dossa. *Announcing Polymath Core v3.0 — "Poho" - Polymath Network*. 2019. URL: https://blog.polymath.network/announcing-polymath-core-v3-0-poho-a71df9cce65b (visited on 10/05/2019).

[39] Adam Dossa. *ERC-1643: Document Management Standard · Issue #1643 · ethereum/EIPs*. 2018. URL: https://github.com/ethereum/EIPs/issues/1643 (visited on 05/10/2019).

[40] Adam Dossa. *ERC-1644: Controller Token Operation Standard · Issue #1644 · ethereum/EIPs*. 2018. URL: https://github.com/ethereum/EIPs/issues/1644 (visited on 05/10/2019).

[41] Adam Dossa et al. *ERC 1400 - Token Standard*. 2018. URL: https://thesecuritytokenstandard.org/ (visited on 05/10/2019).

[42] Dr. Werner & Partner. *STO Security Token - Ausführlicher Guide*. 2018. URL: https://www.drwerner.com/de/sto-security-token-ausfuehrlicher-guide/ (visited on 05/10/2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[43] Raghav Dua. *duaraghav8/Ethlint: (Formerly Solium) Linter to identify and fix style and security issues in Solidity*. 2019. URL: https://github.com/duaraghav8/Ethlint (visited on 05/10/2019).

[44] André Ebner, Falko Fecht, and Alexander Schulz. „How central is central counterparty clearing? A deep dive into a European repo market during the crisis“. In: *Deutsche Bundesbank Discussion Paper 14/2016* (2016). URL: https://www.econstor.eu/bitstream/10419/142125/1/860791300.pdf (visited on 05/10/2019).

[45] ECB Europe. *Congratulations Europe – the final T2S migration wave has been completed*. 2017. URL: https://www.ecb.europa.eu/paym/intro/mip-online/2017/html/201709_article_T2S_migration_wave_completed.en.html (visited on 05/10/2019).

[46] Ben Edgington. *LLL_erc20*. 2017. URL: https://github.com/benjaminion/LLL_erc20 (visited on 05/10/2019).

[47] ENS Team. *Ethereum Name Service*. 2019. URL: https://docs.ens.domains/ (visited on 05/10/2019).

[48] ERC721 Team. *ERC-721 Token Standard*. 2018. URL: http://erc721.org/ (visited on 05/10/2019).

[49] Ethereum Community. *Accessing Contracts and Transactions — Ethereum Homestead 0.1 documentation*. 2019. URL: http://ethereum-homestead.readthedocs.io/en/latest/contracts-and-transactions/accessing-contracts-and-transactions.html (visited on 05/10/2019).

[50] Ethereum Team. *1. Introduction — Swarm 0.5 documentation*. 2019. URL: https://swarm-guide.readthedocs.io/en/latest/introduction.html (visited on 05/10/2019).

[51] Ethereum Team. *LLL Introduction — LLL Compiler Documentation 0.1 documentation*. 2019. URL: https://lll-docs.readthedocs.io/en/latest/lll_introduction.html (visited on 05/10/2019).

[52] Ethereum Team. *Solidity — Solidity 0.5.12 documentation*. 2019. URL: https://solidity.readthedocs.io/en/latest/ (visited on 05/10/2019).

[53] Ethereum Team. *Vyper — Vyper documentation*. 2019. URL: https://vyper.readthedocs.io/en/v0.1.0-beta.8/ (visited on 05/10/2019).

[54] Etherisc Team. *Etherisc White Paper*. 2018. URL: https://etherisc.com/files/etherisc_whitepaper_1.01_en.pdf (visited on 05/10/2019).

[55] Etherscan Team. *Ethereum Average Gas Price Chart*. 2019. URL: https://etherscan.io/chart/gasprice (visited on 05/10/2019).

[56] EthPM Team. *EthPM v2 - Ethereum Package Management*. 2019. URL: https://www.ethpm.com/ (visited on 05/10/2019).

[57] European Central Bank. *General Principles of T2s*. Tech. rep. 2011. URL: https://www.ecb.europa.eu/paym/target/t2s/profuse/shared/pdf/2011_t2s_general_principles.pdf? (visited on 05/10/2019).

[66] Maarten Everts and Frank Muller. *Will That Smart Contract Really Do What You Expect It To Do ?* Groningen, 2018. URL: https://nn8.nl/publications/pdfs/Everts2018WTS.pdf (visited on 05/10/2019).

[67] Luc Falempin. *investorID, the compliant identity for digital assets holders*. 2019. URL: https://medium.com/tokeny/investorid-the-compliant-identity-for-digital-assets-holders-b4c8a2409990 (visited on 05/10/2019).

[69] Josselin Feist. *crytic/slither: Static Analyzer for Solidity*. 2019. URL: https://github.com/crytic/slither (visited on 05/10/2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

[70] FinTech Network. *Smart Contracts – From Ethereum to Potential Banking Use Cases*. 2016. URL: http://blockchainapac.fintecnet.com/uploads/2/4/3/8/24384857/smart_contracts.pdf (visited on 05/10/2019).

[71] Tim Fries. *Fineqia Partners with Nivaura to Issue Tokenized Bonds in the United Kingdom - The Tokenist*. 2019. URL: https://thetokenist.io/fineqia-partners-with-nivaura-to-issue-tokenized-bonds-in-the-united-kingdom/ (visited on 05/10/2019).

[73] Akhilesh Ganti. *Brokerage Fee Definition*. 2019. URL: https://www.investopedia.com/terms/b/brokerage-fee.asp (visited on 05/10/2019).

[74] Akhilesh Ganti. *Preferred Stock Definition*. 2019. URL: https://www.investopedia.com/terms/p/preferredstock.asp (visited on 05/10/2019).

[75] Marcelo Garcia Casil. *Asset Tokenisation: Soft Tokens vs Hard Tokens – Maecenas – Medium*. 2017. URL: https://medium.com/maecenas/asset-tokenisation-soft-tokens-vs-hard-tokens-1ad3a8e39340 (visited on 05/10/2019).

[77] Giveeth Team. *Giveth/minime: Minimi Token. ERC20 compatible clonable token*. 2018. URL: https://github.com/Giveth/minime (visited on 05/10/2019).

[78] Gnosis Team. *gnosis/mock-contract: Simple Solidity contract to mock dependent contracts in truffle tests.* 2019. URL: https://github.com/gnosis/mock-contract (visited on 05/10/2019).

[79] Stephane Gosselin. *Polymath Upgradeability Commit*. 2018. URL: https://github.com/PolymathNetwork/polymath-core/commit/4bc9a9a9ba990cf667e0516fe1b1198fd0d6e4c2 (visited on 05/10/2019).

[80] Mudit Gupta. *Discussion - Removing or Increasing the Contract Size Limit - Primordial Soup - Fellowship of Ethereum Magicians*. 2019. URL: https://ethereum-magicians.org/t/removing-or-increasing-the-contract-size-limit/3045 (visited on 05/10/2019).

[81] Mudit Gupta. *Solidity tips and tricks to save gas and reduce bytecode size*. 2019. URL: https://blog.polymath.network/solidity-tips-and-tricks-to-save-gas-and-reduce-bytecode-size-c44580b218e6 (visited on 05/10/2019).

[82] Vinay Gupta. *A Brief History of Blockchain*. 2017. URL: https://hbr.org/2017/02/a-brief-history-of-blockchain (visited on 05/10/2019).

[84] Harbor Team. *Harborhq Github Repository*. 2018. URL: https://github.com/harborhq (visited on 05/10/2019).

[85] Harbor Team. *harborhq/r-token: Smart Contracts for applying regulatory compliance to tokenized securities issuance and trading*. 2018. URL: https://github.com/harborhq/r-token (visited on 05/10/2019).

[86] Harbor Team. *Website of the Harbor Platform*. 2019. URL: https://harbor.com (visited on 05/10/2019).

[87] Adam Hayes. *Understanding Preferred vs. Common Stock*. 2019. URL: https://www.investopedia.com/ask/answers/difference-between-preferred-stock-and-common-stock/ (visited on 05/10/2019).

[88] Alyssa Hertig. *Cat Fight? Ethereum Users Clash Over CryptoKitties - CoinDesk*. 2017. URL: https://www.coindesk.com/cat-fight-ethereum-users-clash-cryptokitties-congestion (visited on 05/10/2019).

[89] Garrick Hileman and Michel Rauchs. „2017 Global Cryptocurrency Benchmarking Study“. In: *SSRN Electronic Journal* (2017). ISSN: 1556-5068. DOI: 10.2139/ssrn.2965436. URL: http://www.ssrn.com/abstract=2965436 (visited on 05/10/2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[90] Ivan Homoliak et al. *An Air-Gapped 2-Factor Authentication for Smart-Contract Wallets*. Dec. 2018. arXiv: 1812.03598. URL: http://arxiv.org/abs/1812.03598 (visited on 05/10/2019).

[91] Kenneth Hu. *Ethereum account – Coinmonks – Medium*. 2018. URL: https://medium.com/coinmonks/ethereum-account-212feb9c4154 (visited on 05/10/2019).

[92] Samantha Hurst. *Fineqia Sets Up Blockchain and Cryptocurrency Investment Subsidiary in Malta and Is Accepted Into FCA Regulatory Sandbox*. 2018. URL: https://www.crowdfundinsider.com/2018/08/137881-fineqia-sets-up-blockchain-cryptocurrency-investment-subsidiary-in-malta-is-accepted-into-fca-regulatory-sandbox/ (visited on 05/10/2019).

[93] International Organization for Standardization. *ISO 25010*. 2011. URL: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010 (visited on 05/10/2019).

[95] Christoph Jentzsch. *Decentralized autonomous organization to automate governance*. Tech. rep. 2016. URL: https://lawofthelevel.lexblogplatformthree.com/wp-content/uploads/sites/187/2017/07/WhitePaper-1.pdf (visited on 05/10/2019).

[96] Raul Jordan. *The User Experience of Blockchain Applications - The Startup - Medium*. 2017. URL: https://medium.com/swlh/the-user-experience-of-blockchain-applications-ea7defc388df (visited on 05/10/2019).

[97] J.P. Morgan. *Interbank Information Network*. 2018. URL: https://www.jpmorgan.com/global/treasury-services/IIN (visited on 05/10/2019).

[98] JP Morgan Chase. *Quorum Whitepaper*. 2016. URL: https://github.com/jpmorganchase/quorum-docs/blob/master/Quorum%20Whitepaper%20v0.1.pdf (visited on 05/10/2019).

[101] Ulla Karppinen. *BBVA and Porsche Holding close an acquisition term loan using blockchain technology*. 2018. URL: https://www.bbva.com/en/bbva-and-porsche-holding-close-an-acquisition-term-loan-using-blockchain-technology/ (visited on 05/10/2019).

[102] Will Kenton. *Callable Preferred Stock*. 2018. URL: https://www.investopedia.com/terms/c/callablepreferredstock.asp (visited on 05/10/2019).

[103] Arjun Kharpal. *Ethereum hits a fresh record high and is up over 13,000% in a year*. 2018. URL: https://www.cnbc.com/2018/01/10/ethereum-price-hits-record-high-above-1400-up-17000-percent-in-a-year.html (visited on 05/10/2019).

[104] Roland Kofler. *How to Vote Safely with an ERC20 Token - validitylabs - Medium*. 2018. URL: https://medium.com/validitylabs/how-to-vote-safely-with-an-erc20-token-518adadbf923 (visited on 05/10/2019).

[106] Martin Köppelmann. *Gnosis Whitepaper*. 2017. URL: https://gnosis.io/pdf/gnosis-whitepaper.pdf (visited on 05/10/2019).

[107] KoreconX Team. *Website of KoreconX*. 2019. URL: http://koreconx.io (visited on 10/05/2019).

[108] Shani Koren. *Neufund Support - Equity Token Inquiry*. 2019. URL: https://support.neufund.org/support/solutions/articles/36000060377-how-will-i-be-able-to-trade-equity-tokens- (visited on 05/10/2019).

[110] Trevor Koverko and Chris Housser. *Polymath - The Securities Token Platform*. Tech. rep. 2017. URL: https://daks2k3a4ib2z.cloudfront.net/5a46fad33472ef00014b540a/5a46fad33472ef00014b54d0_Polymath%20Whitepaper.pdf (visited on 05/10/2019).

[111] Krypital Group. *From Github to See the Future of STO standards – Krypital Group – Medium*. 2019. URL: https://medium.com/krypital/from-github-to-see-the-future-of-sto-standards-1752b63f2c9a (visited on 05/10/2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

[112] Leslie Lamport, Robert Shostak, and Marshall Pease. „The Byzantine Generals Problem“. In: *ACM Transactions on Programming Languages and Systems* 4.3 (1982), pp. 382–401. ISSN: 01640925. DOI: 10.1145/357172.357176. arXiv: arXiv:1011.1669v3. URL: http://portal.acm.org/citation.cfm?doid=357172.357176 (visited on 05/10/2019).

[113] Patrick Laurent. „The tokenization of assets is disrupting the financial industry. Are you ready?“ In: *Deloitte Inside Magazine Issue 19* (2018). URL: https://www2.deloitte.com/content/dam/Deloitte/lu/Documents/financial-services/lu-tokenization-of-assets-disrupting-financial-industry.pdf (visited on 05/10/2019).

[114] Sergio Demian Lerner. *RSK: White Paper Overview*. 2015. URL: https://docs.rsk.co/RSK_White_Paper-Overview.pdf (visited on 05/10/2019).

[115] Jenny Leung. *ASX's Proposed Distributed Ledger and the Future of Clearing and Settlement*. 2018. URL: https://medium.com/bitfwd/asxs-proposed-distributed-ledger-and-the-future-of-clearing-and-settlement-1d401160a0fd (visited on 05/10/2019).

[116] Christian Lundkvist et al. *Uport: a platform for self-sovereign Identity*. Tech. rep. 2016. URL: http://blockchainlab.com/pdf/uPort_whitepaper_DRAFT20161020.pdf (visited on 05/10/2019).

[118] Maecenas Team. *Maecenas-the decentralized art gallery*. 2018. URL: https://icosbull.com/whitepapers/1483/Maecenas_whitepaper.pdf (visited on 05/10/2019).

[119] Michael Mainelli and Alistair Milne. „The Impact and Potential of Block chain on the Securities Transaction Lifecycle“. In: *Swift Institute Working Paper* 2015-007 (2016), p. 81. URL: https://www.swiftinstitute.org/wp-content/uploads/2016/05/The-Impact-and-Potential-of-Blockchain-on-the-Securities-Transaction-Lifecycle_Mainelli-and-Milne-FINAL-1.pdf (visited on 05/10/2019).

[120] Maker Team. *The Dai Stablecoin System Whitepaper*. Tech. rep. 2017. URL: https://makerdao.com/whitepaper/Dai-Whitepaper-Dec17-en.pdf (visited on 05/10/2019).

[122] Steve Marx. *Upgradeability Is a Bug - ConsenSys Diligence - Medium*. 2019. URL: https://medium.com/consensys-diligence/upgradeability-is-a-bug-dba0203152ce (visited on 05/10/2019).

[123] Stephen McKeon. *Liquidity is about market depth, not magic*. 2017. URL: https://hackernoon.com/liquidity-is-about-market-depth-not-magic-345d9f12f40b (visited on 05/10/2019).

[124] Metamask Team. *MetaMask Web Page*. 2019. URL: https://metamask.io/ (visited on 05/10/2019).

[125] Michel Dominique. *The missing standard in Security token standards — an overview*. 2019. URL: https://medium.com/@dominique_10399/the-missing-standard-in-security-token-standards-an-overview-f34733b82df7 (visited on 05/10/2019).

[126] Mocha Team. *Mocha - the fun, simple, flexible JavaScript test framework*. 2019. URL: https://mochajs.org/ (visited on 05/10/2019).

[127] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. URL: https://bitcoin.org/bitcoin.pdf (visited on 05/10/2019).

[128] Satoshi Nakamoto. *Re : Bitcoin P2P e-cash paper*. 2008. URL: http://users.encs.concordia.ca/~clark/biblio/bitcoin/Nakamoto%202008.pdf (visited on 05/10/2019).

[129] NCC Group. *DASP - TOP 10*. 2019. URL: https://dasp.co/ (visited on 05/10/2019).

[130] Neo Team and Da Hongfei. *NEO Whitepaper*. 2018. URL: https://docs.neo.org/docs/en-us/basic/whitepaper.html (visited on 05/10/2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[131] Neufund Team. *Neufund IEquityTokenController.sol*. 2018. URL: https://github.com/Neufund/platform-contracts/blob/master/contracts/Company/IEquityTokenController.sol (visited on 05/10/2019).

[132] Neufund Team. *Neufund IndentityRegistry.sol*. 2018. URL: https://github.com/Neufund/platform-contracts/blob/master/contracts/Identity/IdentityRegistry.sol (visited on 05/10/2019).

[133] Neufund Team. *Neufund partners with Blocktrade.com – Neufund*. 2018. URL: https://blog.neufund.org/neufund-partners-with-blocktrade-com-240d52804d82 (visited on 05/10/2019).

[134] Neufund Team. *Neufund Whitepaper v 2.0*. 2017. URL: https://neufund.org/cms_resources/whitepaper.pdf (visited on 05/10/2019).

[135] Neufund Team. *Neufunds EquityToken.sol*. 2018. URL: https://github.com/Neufund/platform-contracts/blob/master/contracts/Company/EquityToken.sol (visited on 05/10/2019).

[137] Janek Noest, Fabian Vogelsteller, and Other Commenters. *Error: Invalid number of arguments to Solidity function, when correct number of arguments is passed to a contract method · Issue #1043 · ethereum/web3.js*. 2017. URL: https://github.com/ethereum/web3.js/issues/1043 (visited on 05/10/2019).

[139] Oliver Wyman and Euroclear. *Blockchain in Capital Markets*. Tech. rep. 2016. URL: https://www.oliverwyman.com/content/dam/oliver-wyman/global/en/2016/feb/BlockChain-In-Capital-Markets.pdf (visited on 05/10/2019).

[140] OpenZeppelin Team. *OpenZeppelin/openzeppelin-test-helpers: JavaScript testing helpers for Ethereum smart contract development*. 2019. URL: https://github.com/OpenZeppelin/openzeppelin-test-helpers (visited on 05/10/2019).

[141] OpenZeppelin Team. *openzeppelin-solidity/Pausable.sol at master*. 2019. URL: https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/lifecycle/Pausable.sol (visited on 05/10/2019).

[142] Openzeppelin Team. *OpenZeppelin Library*. 2019. URL: https://openzeppelin.org/ (visited on 05/10/2019).

[143] Openzeppelin Team. *openzeppelin-eth/ERC20.sol at master*. 2019. URL: https://github.com/OpenZeppelin/openzeppelin-eth/blob/master/contracts/token/ERC20/ERC20.sol (visited on 05/10/2019).

[144] Owasp Foundation. *OWASP*. 2019. URL: https://www.owasp.org/index.php/Main_Page (visited on 05/10/2019).

[146] Parity Team. *A Postmortem on the Parity Multi-Sig Library Self-Destruct | Parity Technologies*. 2017. URL: https://www.parity.io/a-postmortem-on-the-parity-multi-sig-library-self-destruct/ (visited on 05/10/2019).

[147] Parity Team. *The Multi-sig Hack: A Postmortem | Parity Technologies*. 2017. URL: https://www.parity.io/the-multi-sig-hack-a-postmortem/ (visited on 05/10/2019).

[148] Stefan Perlebach. *Securrency: How to make Security Tokens compliant – STOcheck – Medium*. 2019. URL: https://medium.com/stocheck/securrency-how-to-make-security-tokens-compliant-665a3f90096 (visited on 05/10/2019).

[150] Jack Peterson et al. *Augur: a Decentralized Oracle and Prediction Market Platform*. 2018. URL: https://arxiv.org/pdf/1501.01042.pdf (visited on 05/10/2019).

[151] Denis Petrovcic and Viktor Brajak. *Blocksquare Whitepaper*. 2018. URL: https://blocksquare.io/assets/pdf/Blocksquare-Whitepaper.pdf (visited on 05/10/2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

[152] PolyMath Support. *Service Fees – PolyMath*. 2018. URL: https://polymath.zendesk.com/hc/en-us/articles/360007626613-Service-Fees (visited on 05/10/2019).

[153] Polymath Team. *USDTieredSTO.sol - Polymath Github*. 2018. URL: https://github.com/PolymathNetwork/polymath-core/blob/2ec70fe3e8adcbea7207a2ef1b3e234d17c70362/contracts/modules/STO/USDTieredSTO.sol (visited on 05/10/2019).

[154] Polymath Team. *Website of the Polymath Platform*. 2019. URL: https://polymath.network/ (visited on 05/10/2019).

[155] Randy Priem. „Distributed Ledger Technology for Securities Clearing and Settlement: Benefits, Risks, and Regulatory Implications“. In: *SSRN Electronic Journal* (2018). ISSN: 1556-5068. DOI: 10.2139/ssrn.3292815. URL: https://www.ssrn.com/abstract=3292815 (visited on 05/10/2019).

[156] Propy Team. *Propy*. Tech. rep. 2017. URL: https://whitepaper.io/document/288/propy-whitepaper (visited on 05/10/2019).

[157] Propy Team. *Propy Github Repository*. 2018. URL: https://github.com/Propy/ethereum (visited on 05/10/2019).

[158] Propy Team. *Propy MetaDeed Contract*. 2018. URL: https://github.com/Propy/ethereum/blob/develop/contracts/disposable/MetaDeedCalifornia.sol (visited on 05/10/2019).

[159] Provenance Team. *Provenance Whitepaper*. 2015. URL: https://www.provenance.org/whitepaper (visited on 05/10/2019).

[160] Qravity Team. *Qravity WhitePaper*. Tech. rep. 2018. URL: https://qravity.com/en/whitepaper/ (visited on 05/10/2019).

[161] Quintor Team. *Quintor/angular-truffle-box: Truffle Box for Angular is a quick-and-easy way to get your Dapp on the road with Truffle and Angular*. 2019. URL: https://github.com/Quintor/angular-truffle-box (visited on 05/10/2019).

[163] Nathan Reiff. *Blockchain Technology's Three Generations*. 2018. URL: https://www.investopedia.com/tech/blockchain-technologys-three-generations/ (visited on 05/10/2019).

[164] Pablo Ruiz. *Polymath Core v2.0.0 Release – PolymathNetwork*. 2018. URL: https://blog.polymath.network/polymath-core-v2-0-0-release-2d5b954c4c99 (visited on 05/10/2019).

[166] Vaibhav Saini. *HackPedia: 16 Solidity Hacks/Vulnerabilities, their Fixes and Real World Examples*. 2018. URL: https://hackernoon.com/hackpedia-16-solidity-hacks-vulnerabilities-their-fixes-and-real-world-examples-f3210eba5148 (visited on 05/10/2019).

[170] Securitize Team. *Securitize | Securitize platform version 3.0 goes live to accelerate...* 2019. URL: https://www.securitize.io/press/securitize-platform-version-3-0-goes-live-to-accelerate-digital-securities-adoption (visited on 05/10/2019).

[171] Securitize Team. *Website of Securitize*. 2019. URL: https://securitize.io/ (visited on 05/10/2019).

[172] SecurityTokenStandard Team. *SecurityTokenStandard/EIP-Spec: SecurityToken standard that helps to launch the compliance securities over the ethereum blockchain*. 2019. URL: https://github.com/SecurityTokenStandard/EIP-Spec (visited on 05/10/2019).

[174] Susmit Sil. *Challenges of Developing Smart Contracts Using Solidity*. 2018. URL: https://medium.com/@susmitsil/challenges-of-developing-smart-contracts-using-solidity-b09352aecc9 (visited on 05/10/2019).

[175] Solidity Team. *Security Considerations - Checks-Effects-Interactions Pattern — Solidity 0.5.10 documentation*. 2019. URL: https://solidity.readthedocs.io/en/v0.5.10/security-considerations.html (visited on 05/10/2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[176] Solidity Team. *Security Considerations — Solidity 0.5.10 documentation*. 2019. URL: https://solidity.readthedocs.io/en/v0.5.10/security-considerations.html (visited on 05/10/2019).

[177] Solidity Team. *Solidity Optimizer and ABIEncoderV2 Bug*. 2019. URL: https://blog.ethereum.org/2019/03/26/solidity-optimizer-and-abiencoderv2-bug/ (visited on 05/10/2019).

[178] StateOfTheDapps Team. *State of the Dapps Website*. 2019. URL: https://www.stateofthedapps.com/ (visited on 05/10/2019).

[179] Charles St.Lous. *Understanding the Layers of the Polymath Network – PolymathNetwork*. 2018. URL: https://blog.polymath.network/understanding-the-layers-of-the-polymath-network-b81c67e49572 (visited on 05/10/2019).

[180] Julian Svirsky, Neil Mohinani, and Denis Donin. *ATLANT Platform Whitepaper*. Tech. rep. 2017. URL: https://cryptorating.eu/whitepapers/ATLANT/Atlant_WP_publish.pdf (visited on 05/10/2019).

[181] Nick Szabo. *Smart Contracts: Building Blocks for Digital Markets*. 1996. URL: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html (visited on 05/10/2019).

[183] currency.com Team. *Currency just got real - Currency.com Whitepaper*. Tech. rep. 2018. URL: https://currency.com/static/Currencycom_WP_V2.pdf (visited on 05/10/2019).

[184] openZeppelin Team. *openzeppelin-contracts-docs/learn-about-access-control.md at master · OpenZeppelin/openzeppelin-contracts-docs*. 2019. URL: https://github.com/OpenZeppelin/openzeppelin-contracts-docs/blob/master/demo-docs/learn-about-access-control.md (visited on 05/10/2019).

[185] Tokenist Team. *Harbor Launches Securities Tokenization Platform, First REIT Tokenized*. 2018. URL: https://thetokenist.io/harbor-launches-securities-tokenization-platform-first-reit-tokenized/ (visited on 05/10/2019).

[186] Tokeny Team. *The Pros and Cons of Security Token Offerings (STOs) - TOKENY*. 2018. URL: https://tokeny.com/the-pros-and-cons-of-security-token-offerings/ (visited on 05/10/2019).

[187] Tokeny Team. *T-REX (Token for Regulated EXchanges)*. Tech. rep. 2018, tokeny.com. URL: https://tokeny.com/wp-content/uploads/2018/12/t-rex-whitepaper.pdf (visited on 05/10/2019).

[188] Truffle Team. *Boxes | Truffle Suite*. 2019. URL: https://www.trufflesuite.com/boxes (visited on 05/10/2019).

[189] Truffle Team. *Sweet Tools for Smart Contracts | Truffle Suite*. 2019. URL: https://www.trufflesuite.com/ (visited on 05/10/2019).

[190] Truffle Team. *Truffle | Testing Your Contracts | Documentation | Truffle Suite*. 2019. URL: https://www.trufflesuite.com/docs/truffle/testing/testing-your-contracts (visited on 05/10/2019).

[191] Truffle Team. *truffle-contract - npm*. 2019. URL: https://www.npmjs.com/package/truffle-contract (visited on 05/10/2019).

[192] TrustToken Team. *Website of Trusttoken*. 2019. URL: https://www.trusttoken.com/ (visited on 05/10/2019).

[193] Nico Vergauwen. *Programmable Incentives — A Blockchain proof of concept*. 2018. URL: https://hackernoon.com/programmable-incentives-a-blockchain-proof-of-concept-fa25c9d95a1a (visited on 05/10/2019).

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

[194] Vogelsteller Fabian. *ERC725 Alliance*. 2018. URL: https://erc725alliance.org/ (visited on 05/10/2019).

[195] Franz Volland. *Eternal Storage | solidity-patterns*. 2018. URL: https://fravoll.github.io/solidity-patterns/eternal_storage.html (visited on 05/10/2019).

[196] Maria Wachal. *Asset tokenization on blockchain will disrupt the asset management landscape*. 2018. URL: https://blog.softwaremill.com/asset-tokenization-on-blockchain-will-disrupt-the-asset-management-landscape-befbd71639b1 (visited on 05/10/2019).

[198] Jr Willet. *MasterCoin Complete Specification vs. 1.1 (Smart Property Edition)*. Tech. rep. 2013. URL: https://sites.google.com/site/2ndbtcwpaper/ (visited on 05/10/2019).

[199] Zachary J. Williamson. *The AZTEC Protocol Whitepaper*. 2018. URL: https://github.com/AztecProtocol/AZTEC/blob/master/AZTEC.pdf (visited on 05/10/2019).

[201] Gavin Wood. *ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER*. 2014. URL: http://gavwood.com/paper.pdf (visited on 05/10/2019).

[202] Roger Wu. *ERC-1726: Dividend-Paying Token Standard · Issue #1726 · ethereum/EIPs*. 2019. URL: https://github.com/ethereum/EIPs/issues/1726 (visited on 05/10/2019).

[206] Kevin Young. *Harbor Platform 2.0 – Tokenize Company Equity for Fundraising and Incentivizing Partners, Customers*. 2019. URL: https://medium.com/harborhq/harbor-platform-startup-equity-67b19c5b745e (visited on 05/10/2019).

[207] Zeppelin Solutions and Augur. *Serpent Compiler Audit*. 2017. URL: https://blog.openzeppelin.com/serpent-compiler-audit-3095d1257929/ (visited on 05/10/2019).

[208] ZeppelinOS Team. *Proxy Patterns - ZeppelinOS Blog*. 2018. URL: https://blog.zeppelinos.org/proxy-patterns/ (visited on 05/10/2019).

[209] ZeppelinOS Team. *Upgradeability using Unstructured Storage - ZeppelinOS Blog*. 2018. URL: https://blog.zeppelinos.org/upgradeability-using-unstructured-storage/ (visited on 05/10/2019).

[210] ZeppelinOS Team. *ZeppelinOS Upgrades Pattern · ZeppelinOS*. 2018. URL: https://docs.zeppelinos.org/docs/pattern.html (visited on 05/10/2019).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

A Appendix

The full code of the system can be examined on GitHub under github.com/jobrot/secblocks.

```
pragma solidity ^0.5.4;

import "../Interfaces/IERC1594.sol";
import "./ERC20.sol";
import "../Roles/IssuerRole.sol";
import "../AML/TransferQueues.sol";
import "../Controlling/Controller.sol";

/**
 * @title AML aware implementation of ERC1594 (Subset of ERC1400 https://
 ↪ github.com/ethereum/EIPs/issues/1411)
 * adapted from the standard implementation of the spec: https://github.com/
 ↪ SecurityTokenStandard/EIP-Spec
 */
contract ERC1594 is IERC1594, ERC20, IssuerRole, OrchestratorRole {
    // Variable which tells whether issuance is ON or OFF forever
    bool internal issuanceClosed = false;
    bool internal nameSet=false;

    bytes32 public name;

    // Variable that stores stores a mapping of the last transfers of the
    ↪ account
    // in order to comply with AML regulations
    // @dev maps each address to an array of dynamic length, that consists a
    ↪  struct of the timestamp and
    // the value of the outbound funds (counted in number of tokens, value
    ↪ must be determined on check
    TransferQueues queues;
    // The single controller that is to be queried before all token moving
    ↪ actions on the respective functions
    Controller controller;

    // Constant that defines how long the last Transfers of each sender are
    ↪ considered for AML checks
    uint constant TRANSFER_RETENTION_TIME = 604800; //604800 == 1 Week in
    ↪ Seconds
    // Constant that defines the maximum value that may be traded within the
    ↪  retention time
    // currently is used as token number, but should in future be
    ↪ implemented using a pricing oracle to represent euros
    uint constant SPEND_CEILING = 15000;

    // reenable constructor, if deployment without proxy is needed
    /*constructor(Controller _controller, TransferQueues _queues)  public {
    ↪ //The super contract is a modifier of sorts of the constructor
        queues = _queues;
        controller = _controller;
    }*/

    /**
     * @dev these three functions are the replacement for the constructor
     ↪ setters in proxy setups
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
42      */
43     function setTransferQueues(TransferQueues _queues) external
       ↪ onlyOrchestrator{
44         queues = _queues;
45     }
46     function setController(Controller _controller) external onlyOrchestrator
       ↪ {
47         controller = _controller;
48     }
49     function addIssuerOrchestrator(address issuer) external onlyOrchestrator
       ↪ {
50         _addIssuer(issuer);
51     }
52     function setName(bytes32 _name) external onlyOrchestrator{
53         require(!nameSet);
54         name = _name;
55         nameSet = true;
56     }
57
58     /**
59      * @notice Transfers tokens if allowed by AML constraints and controller
60      * @param _to address The address which you want to transfer to
61      * @param _value uint256 the amount of tokens to be transferred
62      * @param _data The `bytes _data` allows arbitrary data to be submitted
        ↪ alongside the transfer.
63      */
64     function transferWithData(address _to, uint256 _value, bytes memory
       ↪ _data) public {
65         super.transferWithData(_to, _value,_data);
66
67         controller.verifyAllTransfer(msg.sender, _to, _value, _data);
68         _checkAMLConstraints(msg.sender,_value);
69     }
70
71
72
73     /**
74      * @notice Transfers tokens if allowed by AML constraints and controller
        ↪ , but not from the
75      * sender itself, but from the _from address. The sender must have
        ↪ sufficient allowance (see ERC20 standard)
76      * @param _from address The address which you want to send tokens from
77      * @param _to address The address which you want to transfer to
78      * @param _value uint256 the amount of tokens to be transferred
79      * @param _data The `bytes _data` allows arbitrary data to be submitted
        ↪ alongside the transfer.
80      */
81     function transferFromWithData(address _from, address _to, uint256 _value
       ↪ , bytes memory  _data) public {
82         super.transferFromWithData(_from, _to, _value,_data);
83
84         controller.verifyAllTransferFrom(msg.sender, _from, _to, _value,
           ↪ _data);
85         _checkAMLConstraints(_from,_value);
86     }
87
88
89
90
91     /**
92      * @notice A security token issuer can specify that issuance has
        ↪ finished for the token
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
93      * (i.e. no new tokens can be minted or issued).
94      * @dev If a token returns FALSE for `isIssuable()` then it MUST always
        ↪ return FALSE in the future.
95      * If a token returns FALSE for `isIssuable()` then it MUST never allow
        ↪ additional tokens to be issued.
96      * @return bool true signifies the minting is allowed. false denotes the
        ↪  end of minting
97      */
98     function isIssuable() external view returns (bool) {
99         return !issuanceClosed;
100    }
101
102    /**
103     * @notice Increases the total token supply by adding tokens to
        ↪ _tokenHolder. Can only be called by the Issuer, and
104     * if the Issuance of the token is not closed. Checks with the
        ↪ controller, and emits an Issued event.
105     * @param _tokenHolder The account that will receive the created tokens
        ↪ (account should be whitelisted or KYCed).
106     * @param _value The amount of tokens need to be issued
107     * @param _data The `bytes _data` allows arbitrary data to be submitted
        ↪ alongside the transfer.
108     */
109    function issue(address _tokenHolder, uint256 _value, bytes memory _data)
       ↪  public onlyIssuer {
110        _mint(_tokenHolder, _value);
111
112        //There is no need to override the mint function here, as it is
           ↪ never accessed in the ERC20 contract and an internal function
113        require(!issuanceClosed, "Issuance is closed");
114        controller.verifyAllIssue(_tokenHolder,_value,_data);
115
116        //Future work: additional checks on issuing, but will most likely be
           ↪  primarily offchain, as there are no good
117        //one size fits all issuing checks
118        emit Issued(msg.sender, _tokenHolder, _value, _data);
119    }
120
121
122
123
124    /**
125     * @notice Closes the issuance forever, forbidding any more tokens to be
        ↪  minted as per ERC1594 definition.
126     */
127    function closeIssuance() external onlyIssuer {
128        issuanceClosed = true;
129    }
130    /**
131     * @notice Redeems (i.e. burns) tokens of the sender. Emits an Redeemed
        ↪ event which can be listened for, in order
132     * to give some off chain reward or reimbursement to the redeemer, if so
        ↪  defined by the token issuer
133     * @param _value The amount of tokens to be redeemed
134     * @param _data The `bytes _data` it can be used in the token contract
        ↪ to authenticate the redemption.
135     */
136    function redeem(uint256 _value, bytes memory  _data) public { //public
       ↪ instead of external so that subcontracts can call
137        _burn(msg.sender, _value);
138
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
139         //There is no need to override the burn function here, as it is
            ↪ never accessed in the ERC20 contract and an internal function
140         controller.verifyAllRedeem(msg.sender,_value,_data);
141         emit Redeemed(address(0), msg.sender, _value, _data);
142     }
143
144
145
146
147     /**
148      * @notice Redeems (see notice above) tokens of the allowance of
         ↪ _tokenholder.
149      * @dev analogous to `transferFrom`
150      * @param _tokenHolder The account whose tokens gets redeemed.
151      * @param _value The amount of tokens to be redeemed
152      * @param _data The `bytes _data` it can be used in the token contract
         ↪ to authenticate the redemption.
153      */
154     function redeemFrom(address _tokenHolder, uint256 _value, bytes memory
        ↪ _data) public {
155         _burnFrom(_tokenHolder, _value);
156
157         //There is no need to override the burn function here, as it is
            ↪ never accessed in the ERC20 contract and an internal function
158         controller.verifyAllRedeem(msg.sender,_value,_data);
159         emit Redeemed(msg.sender, _tokenHolder, _value, _data);
160     }
161
162     /**
163      * @notice Checks if a transfer can be executed. May be called by
         ↪ clients prior to executing a transaction, in order
164      * to prevent losses.
165      * Does not take AML into account, as this depends on timing of the
         ↪ transaction
166      * @param _to address The address which you want to transfer to
167      * @param _value uint256 the amount of tokens to be transferred
168      * @param _data The `bytes _data` allows arbitrary data to be submitted
         ↪ alongside the transfer.
169      * @return bool It signifies whether the transaction will be executed or
         ↪  not.
170      * @return byte Ethereum status code (ESC)
171      * @return bytes32 Application specific reason code
172      */
173     function canTransfer(address _to, uint256 _value, bytes calldata _data)
        ↪ external view returns (bool, byte, bytes32) {
174         bool can;
175         bytes32 reason;
176         (can, reason)=controller.checkAllTransfer(msg.sender,  _to,  _value,
            ↪   _data);
177         if (balanceOf(msg.sender) < _value) return (false, 0x54, bytes32(0))
            ↪ ;
178         else if (!can) return (false, 0x59, reason);
179         return (true, 0x51, bytes32(0));
180     }
181
182     /**
183      * @notice Checks if a transferFrom can be executed. May be called by
         ↪ clients prior to executing a transaction, in order
184      * to prevent losses.
185      * Does not take AML into account, as this depends on timing of the
         ↪ transaction
186      * @param _from address The address which you want to send tokens from
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
187         * @param _to address The address which you want to transfer to
188         * @param _value uint256 the amount of tokens to be transferred
189         * @param _data The `bytes _data` allows arbitrary data to be submitted
            ↪ alongside the transfer.
190         * @return bool It signifies whether the transaction will be executed or
            ↪  not.
191         * @return byte Ethereum status code (ESC)
192         * @return bytes32 Application specific reason code
193         */
194        function canTransferFrom(address _from, address _to, uint256 _value,
           ↪ bytes calldata _data) external view returns (bool, byte, bytes32) {
195            bool can;
196            bytes32 reason;
197            (can, reason)=controller.checkAllTransfer(msg.sender,  _to,  _value,
               ↪   _data);
198            if (_value > _allowed[_from][msg.sender]) return (false, 0x53,
               ↪ bytes32(0));
199            else if (balanceOf(_from) < _value) return (false, 0x54, bytes32(0))
               ↪ ;
200            else if (!can) return (false, 0x59, reason);
201            return (true, 0x51, bytes32(0));
202        }
203
204        /**
205         * @notice checks if the Anti Money Laundering constraints are satisfied
            ↪ , i.e. all outgoing transactions by _from
206         * within a timespan of TRANSFER_RETENTION_TIME do not exceed
            ↪ SPEND_CEILING.
207         */
208        function _checkAMLConstraints(address from, uint value) private{
209            //actually, according to the aml law, for transactions over the
               ↪ limit the senders only have to be authenticated,
210            //which would anyway be the case with the KYCVerifier
211            require(_updateTransferListAndCalculateSum(from,value) <
               ↪ SPEND_CEILING,"ERC1594: The transfer exceeds the allowed quota
               ↪ within the retention period.");
212        }
213
214        /**
215        * @dev Adds two numbers, return false on overflow, keeps transfer list
           ↪ ordered
216        */
217        function _updateTransferListAndCalculateSum(address _from, uint256
           ↪ _value) private returns(uint sum) {
218            uint sumOfTransfers=0;
219            uint timestamp;
220
221
222            while(!queues.empty(_from)){
223                (timestamp, )= queues.peek(_from);
224                if(timestamp <= now - TRANSFER_RETENTION_TIME){
225                    queues.dequeue(_from);
226                }
227                else break;
228            }
229
230            queues.enqueue(_from, now, _value);
231
232            return queues.sumOfTransfers(_from);
233
234        }
235
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
236 }
```

Listing A.1: The ERC1594 contract, the base token contract for our security token.

```
1  pragma solidity ^0.5.4;
2
3  import "../Openzeppelin/SafeMath.sol";
4  import "./ERC1594.sol";
5  import "../Libraries/UIntConverterLib.sol";
6  import "../Libraries/SafeMathInt.sol";
7
8  /**
9   * This is an adaptation of the standard implementation of the Dividend
    ↪ Paying Token Standard
10  * https://github.com/ethereum/EIPs/issues/1726
11  * @notice this Token corresponds to preferred Securities, that do not allow
    ↪  voting but gather more regular dividends
12 */
13 contract DividendToken is ERC1594 {
14     using SafeMath for uint;
15     using SafeMathInt for int;
16     using UIntConverterLib for uint;
17
18
19     // Event to inform holders of new Distribution of dividends
20     event DividendsDistributed(
21         address indexed from,
22         uint256 weiAmount
23     );
24
25     // Event to inform holders of new Distribution of dividends
26     event DividendWithdrawn(
27         address indexed to,
28         uint256 weiAmount
29     );
30
31
32     // With `magnitude`, we can properly distribute dividends even if the
       ↪ amount of received ether is small.
33     // For more discussion about choosing the value of `magnitude`,
34     //  see https://github.com/ethereum/EIPs/issues/1726#issuecomment
       ↪ -472352728
35     uint constant internal magnitude = 2**128;
36
37     uint internal magnifiedDividendPerShare;
38
39     // About dividendCorrection:
40     // If the token balance of a `_user` is never changed, the dividend of `
       ↪ _user` can be computed with:
41     //   `dividendOf(_user) = dividendPerShare * balanceOf(_user)`.
42     // When `balanceOf(_user)` is changed (via minting/burning/transferring
       ↪ tokens),
43     //   `dividendOf(_user)` should not be changed,
44     //   but the computed value of `dividendPerShare * balanceOf(_user)` is
       ↪ changed.
45     // To keep the `dividendOf(_user)` unchanged, we add a correction term:
46     //   `dividendOf(_user) = dividendPerShare * balanceOf(_user) +
       ↪ dividendCorrectionOf(_user)`,
47     //   where `dividendCorrectionOf(_user)` is updated whenever `balanceOf(
       ↪ _user)` is changed:
48     //   `dividendCorrectionOf(_user) = dividendPerShare * (old balanceOf(
       ↪ _user)) - (new balanceOf(_user))`.
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
49      // So now `dividendOf(_user)` returns the same value before and after `
        ↪ balanceOf(_user)` is changed.
50      mapping(address => int) internal magnifiedDividendCorrections;
51      mapping(address => uint) internal withdrawnDividends;
52
53
54      /*constructor(Controller _controller, TransferQueues _queues) ERC1594(
        ↪ _controller, _queues) public { //The super contract is a modifier of
        ↪ sorts of the constructor
55
56      }*/
57
58
59      /// @dev Distributes dividends whenever ether is paid to this contract.
60      function() external payable {
61          distributeDividends();
62      }
63      /**
64       * @notice Distributes ether to token holders as dividends.
65       * @dev It reverts if the total supply of tokens is 0.
66       * It emits the `DividendsDistributed` event if the amount of received
         ↪ ether is greater than 0.
67       * About undistributed ether:
68       *   In each distribution, there is a small amount of ether not
         ↪ distributed,
69       *     the magnified amount of which is
70       *     `(msg.value * magnitude) % totalSupply()`.
71       *   With a well-chosen `magnitude`, the amount of undistributed ether
72       *     (de-magnified) in a distribution can be less than 1 wei.
73       *   We can actually keep track of the undistributed ether in a
         ↪ distribution
74       *     and try to distribute it in the next distribution,
75       *     but keeping track of such data on-chain costs much more than
76       *     the saved ether, so we don't do that.
77      */
78      function distributeDividends() public payable {
79          require(totalSupply() > 0, "Currently, no tokens exist.");
80
81          if (msg.value > 0) {
82              magnifiedDividendPerShare = magnifiedDividendPerShare.add(
83                  (msg.value).mul(magnitude) / totalSupply()
84              );
85              emit DividendsDistributed(msg.sender, msg.value);
86          }
87      }
88      /**
89       * @notice Withdraws the ether distributed to the sender.
90       * @dev It emits a `DividendWithdrawn` event if the amount of withdrawn
         ↪ ether is greater than 0.
91      */
92      function withdrawDividend() external {
93          uint _withdrawableDividend = withdrawableDividendOf(msg.sender);
94          if (_withdrawableDividend > 0) {
95              withdrawnDividends[msg.sender] = withdrawnDividends[msg.sender].
                ↪ add(_withdrawableDividend);
96              emit DividendWithdrawn(msg.sender, _withdrawableDividend);
97              (msg.sender).transfer(_withdrawableDividend);
98          }
99      }
100     /**
101     * @notice View the amount of dividend in wei that an address can
        ↪ withdraw.
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
102      * @param _owner The address of a token holder.
103      * @return The amount of dividend in wei that `_owner` can withdraw.
104      */
105      function dividendOf(address _owner) external view returns(uint) {
106          return withdrawableDividendOf(_owner);
107      }
108
109      /**
110      * @notice View the amount of dividend in wei that an address can
         ↪ withdraw.
111      * @param _owner The address of a token holder.
112      * @return The amount of dividend in wei that `_owner` can withdraw.
113      */
114      function withdrawableDividendOf(address _owner) public view returns(uint
         ↪ ) {
115          return accumulativeDividendOf(_owner).sub(withdrawnDividends[_owner
             ↪ ]);
116      }
117
118      /**
119      * @notice View the amount of dividend in wei that an address has
         ↪ withdrawn.
120      * @param _owner The address of a token holder.
121      * @return The amount of dividend in wei that `_owner` has withdrawn.
122      */
123      function withdrawnDividendOf(address _owner) external view returns(uint)
         ↪  {
124          return withdrawnDividends[_owner];
125      }
126
127      /**
128       * @notice View the amount of dividend in wei that an address has earned
          ↪  in total.
129       * @dev accumulativeDividendOf(_owner) = withdrawableDividendOf(_owner)
          ↪ + withdrawnDividendOf(_owner)
130       * = (magnifiedDividendPerShare * balanceOf(_owner) +
          ↪ magnifiedDividendCorrections[_owner]) / magnitude
131       * @param _owner The address of a token holder.
132       * @return The amount of dividend in wei that `_owner` has earned in
          ↪ total.
133      */
134      function accumulativeDividendOf(address _owner) public view returns(uint
         ↪ ) {
135          return magnifiedDividendPerShare.mul(balanceOf(_owner)).toIntSafe()
136          .add(magnifiedDividendCorrections[_owner]).toUintSafe() / magnitude;
137      }
138
139      /**
140       * @dev transfers with regard to the dividend corrections, overrides
          ↪ super function
141       * @param _to The address to transfer to.
142       * @param _value The amount to be transferred.
143       * @param _data The `bytes _data` allows arbitrary data to be submitted
          ↪ alongside the transfer.
144       */
145      function transferWithData(address _to, uint256 _value, bytes memory
         ↪ _data) public {
146          int magCorrection = magnifiedDividendPerShare.mul(_value).toIntSafe
             ↪ ();
147          magnifiedDividendCorrections[msg.sender] =
             ↪ magnifiedDividendCorrections[msg.sender].add(magCorrection);
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
148            magnifiedDividendCorrections[_to] = magnifiedDividendCorrections[_to
               ↪ ].sub(magCorrection);
149
150            super.transferWithData( _to, _value, _data);
151        }
152
153        /**
154         * @dev transfers using the allowance for another address with regard to
           ↪  the dividend corrections,
155         * overrides super and calls it
156         * @param _from The address to transfer from.
157         * @param _to The address to transfer to.
158         * @param _value The amount to be transferred.
159         * @param _data The `bytes _data` allows arbitrary data to be submitted
           ↪ alongside the transfer.
160         */
161        function transferFromWithData(address _from, address _to, uint _value,
           ↪ bytes memory _data) public {
162            int magCorrection = magnifiedDividendPerShare.mul(_value).toIntSafe
               ↪ ();
163            magnifiedDividendCorrections[_from] = magnifiedDividendCorrections[
               ↪ _from].add(magCorrection);
164            magnifiedDividendCorrections[_to] = magnifiedDividendCorrections[_to
               ↪ ].sub(magCorrection);
165
166            super.transferFromWithData(_from, _to, _value, _data);
167        }
168
169        /**
170         * @dev function that issues tokens to an account.
171         * Updates magnifiedDividendCorrections to keep dividends unchanged.
172         * @param _tokenHolder The account that will receive the created tokens.
173         * @param _value The amount that will be created.
174         * @param _data The `bytes _data` allows arbitrary data to be submitted
           ↪ alongside the transfer.
175         */
176        function issue(address _tokenHolder, uint _value, bytes memory _data)
           ↪ public onlyIssuer {
177            magnifiedDividendCorrections[_tokenHolder] =
               ↪ magnifiedDividendCorrections[_tokenHolder]
178            .sub( (magnifiedDividendPerShare.mul(_value)).toIntSafe() );
179
180            super.issue(_tokenHolder, _value, _data);
181        }
182
183        /**
184         * @notice This function redeems an amount of the token of a msg.sender.
185         * @dev overrides and calls the super implementation
186         * @param _value The amount of tokens to be redeemed
187         * @param _data The `bytes _data` it can be used in the token contract
           ↪ to authenticate the redemption.
188         */
189        function redeem(uint _value, bytes memory _data) public {
190            magnifiedDividendCorrections[msg.sender] =
               ↪ magnifiedDividendCorrections[msg.sender]
191            .add( (magnifiedDividendPerShare.mul(_value)).toIntSafe() );
192
193            super.redeem(_value, _data);
194        }
195
196        /**
197         * @notice This function redeems an amount of the token of a msg.sender.
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
198         * @dev It is an analogy to `transferFrom`, and overrides the method in
            ↪ the super contract to hide it
199         * in order to enforce usage of this method that is aware of the
            ↪ dividend corrections
200         * @param _tokenHolder The account whose tokens gets redeemed.
201         * @param _value The amount of tokens to be redeemed
202         * @param _data The `bytes _data` that can be used in the super token
            ↪ contract to authenticate the redemption.
203         */
204        function redeemFrom(address _tokenHolder, uint _value, bytes memory
           ↪ _data) public {
205            magnifiedDividendCorrections[_tokenHolder] =
               ↪ magnifiedDividendCorrections[_tokenHolder]
206            .add( (magnifiedDividendPerShare.mul(_value)).toIntSafe() );
207
208            super.redeemFrom(_tokenHolder, _value, _data);
209        }
210
211    }
```

Listing A.2: The DividendToken contract, the dividend-aware token subcontract.

```
1   pragma solidity ^0.5.4;
2
3   import "./DividendToken.sol";
4
5
6   /**
7    * @dev Inspiration by MiniMeToken https://github.com/Giveth/minime/blob/
     ↪ master/contracts/MiniMeToken.sol by giveEth
8    * @notice this Token corresponds to common stock that enables dividends as
     ↪ well as voting
9   */
10  contract VotingToken is DividendToken {
11
12      event BallotCreated(
13          bytes32 ballotName
14      );
15
16      struct Checkpoint {
17          // `fromBlock` is the block number that the checkpoint was generated
18          uint128 fromBlock;
19          // `value` is the amount of tokens of the checked address at the
            ↪ specific block number
20          uint128 value;
21      }
22
23      /*
24       * @notice a ballot is a structure that corresponds to a single decision
         ↪  that can be
25       * voted upon by token holders, each with a weight corresponding to
         ↪ their
26       * owned tokens at the cutoff date
27       */
28      struct Ballot {
29          bytes32 name; // name of the current ballot / asked question
30          bytes32[] optionNames; //list of all voteable options
31          uint[] optionVoteCounts; //list of all resp. vote counts
32          mapping(address => bool) voted; // record on which addresses already
            ↪  voted
33          uint cutoffBlockNumber; // block number of the block, where the
            ↪ balances are counted for voting weight
34          uint endDate;
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
35      }
36
37      bytes32[] tempOptionNames;
38
39
40      /**
41        * @dev tracks the balance of each address over time via timestamped
          ↪ Checkpoints
42        * including the blocknumber
43        * intentionally
44      */
45      mapping(address => Checkpoint[]) private _historizedBalances;
46
47
48      // Tracks the history of the `totalSupply` of the token
49      Checkpoint[] totalSupplyHistory;
50
51      // all ballots that can be voted upon by token holders
52      Ballot[] public ballots;
53
54      // reenable constructor, if deployment without proxy is needed
55      /*constructor(Controller _controller, TransferQueues _queues)
        ↪ DividendToken(_controller, _queues) public {//The super contract is a
        ↪  modifier of sorts of the constructor
56
57      }*/
58
59      /*
60      * @info get all optionNames for a single ballot, indicated by its index
        ↪ in the public ballot array
61      * @param ballotIndex ballot to get optionNames from
62      * @returns all options of the ballot at the given index
63      */
64      function getOptionNames(uint ballotIndex) external view returns (bytes32
        ↪ [] memory){
65          return ballots[ballotIndex].optionNames;
66      }
67
68      /*
69      * @info get all voteCounts for a single ballot, indicated by its index
        ↪ in the public ballot array
70      * @param ballotIndex ballot to get voteCounts from
71      * @returns all voteCounts of the ballot at the given index,
        ↪ corresponding to the names from getOptionNames
72      */
73      function getOptionVoteCounts(uint ballotIndex) external view returns (
        ↪ uint256[] memory){
74          return ballots[ballotIndex].optionVoteCounts;
75      }
76
77      /**
78       * @dev Overwrites the ERC-20 function so that it will be used
79       * by all functions in super contracts to keep checkpoints up to date
80       * @param _from The address holding the tokens being transferred
81       * @param _to The address of the recipient
82       * @param _amount The amount of tokens to be transferred
83      */
84      function _transfer(address _from, address _to, uint _amount) internal {
85          //require(_amount != 0, "VotingToken: Empty Transfers are not
            ↪ allowed.");
86
87          // Do not allow transfer to or from 0x0
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
88          require((_to != address(0)) && (_from != address(0)), "Transfer to
            ↪ or from 0x");
89
90          uint previousBalanceSender = balanceOfAt(_from, block.number);
91
92          // update the balance array with the new value for the sender
93          updateValueAtNow(_historizedBalances[_from], previousBalanceSender.
            ↪ sub(_amount));
94
95          // update the balance array with the new value for the receiver
96          uint previousBalanceTo = balanceOfAt(_to, block.number);
97          updateValueAtNow(_historizedBalances[_to], previousBalanceTo.add(
            ↪ _amount));
98
99          emit Transfer(_from, _to, _amount);
100     }
101
102     /**
103      * @dev overrides the function from ERC20
104      * @param _owner The address that's balance is being requested
105      * @return The balance of `_owner` at the current block
106      */
107     function balanceOf(address _owner) public view returns (uint256) {
108         return balanceOfAt(_owner, block.number);
109     }
110
111
112     /**
113      * @dev creates a new ballot and appends it to 'ballots'
114      * caller must ensure that optionNames are unique, or else only the
         ↪ first appearance is used
115      * @param ballotName The name of the ballot resp. the asked Question
116      * @param optionNames List of possible choices / answers to the question
117     */
118     function createBallot(bytes32 ballotName, bytes32[] calldata optionNames
        ↪ , uint endDate) external onlyIssuer {
119         //Check the arguments for validity
120         require(ballotName[0] != 0, "BallotName must not be empty!");
121         require(optionNames.length > 0, "OptionNames must not be empty!");
122
123         delete tempOptionNames;
124
125         for (uint i = 0; i < optionNames.length; i++) {
126             tempOptionNames.push(optionNames[i]);
127         }
128
129         Ballot memory ballot = Ballot({name : ballotName, cutoffBlockNumber
            ↪ : block.number, endDate : endDate, optionNames : tempOptionNames,
            ↪  optionVoteCounts : new uint[](optionNames.length)});
130         ballots.push(ballot);
131
132         emit BallotCreated(ballotName);
133     }
134     /**
135       * @notice casts votes equal to the senders tokens at the cutoff time
          ↪ of the newest ballot designated
136       * by ballotName to the option designated by optionName
137       * @param ballotName Ballot to vote in
138       * @param optionName option to vote for
139       * @dev rolls back on unfound ballot or option, if sender already voted
          ↪ , or does not own tokens at cutoff
140     */
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
141     function vote(bytes32 ballotName, bytes32 optionName) external {
142         //Search through all Ballots backwards, so that the recent ones are
            ↪ found first
143         int found = - 1;
144
145         for (int i = UIntConverterLib.toIntSafe(ballots.length); i > 0; i--)
            ↪  {//could still be optimized for gas
146             if (ballots[SafeMathInt.toUintSafe(i - 1)].name == ballotName) {
147                 found = i - 1;
148                 break;
149             }
150         }
151         require(found >= 0, "VotingToken: Ballot not found!");
152
153         Ballot storage ballot = ballots[SafeMathInt.toUintSafe(found)];
154
155         //check if User had tokens at the time of the ballot start == right
            ↪ to vote
156         uint senderBalance = balanceOfAt(msg.sender, ballot.
            ↪ cutoffBlockNumber);
157         require(senderBalance > 0, "Sender held no tokens at cutoff");
158
159         //check if endDate has not passed
160         require(ballot.endDate > now, "Vote has ended.");
161
162         //check if User already voted
163         require(ballot.voted[msg.sender] == false, "Sender already voted");
164
165         //Search for the voted option in the ballot
166
167         for (int j = UIntConverterLib.toIntSafe(ballot.optionNames.length);
            ↪ j > 0; j--) {
168             if (ballot.optionNames[SafeMathInt.toUintSafe(j - 1)] ==
                ↪ optionName) {
169                 ballot.voted[msg.sender] = true;
170                 ballot.optionVoteCounts[SafeMathInt.toUintSafe(j - 1)] =
                    ↪ ballot.optionVoteCounts[SafeMathInt.toUintSafe(j - 1)].
                    ↪ add(senderBalance);
171                 return;
172             }
173         }
174         require(false, "Option does not exist in Ballot.");
175     }
176     /**
177      * @notice Computes the winning proposal taking all votes up until now
         ↪ into account, exact tallying can be gotten from
178      * the function getBallot(bytes32 ballotName)
179      * @dev does not change state or close the voting intentionally, is
         ↪ public so that everybody
180      * can look at the current winner, and to allow for maximum flexibility
         ↪ (company can decide
181      * a time when to decide the winners, as the time may change due to not
         ↪ all voters being on chain
182      * ATTENTION: does not deal with votes where two options are equal, this
         ↪  must be decided by the user via getBallot(bytes32 ballotName)
183      * @param ballotName ballot to be queried
184     * @return name of the winning option and resp. vote count, if it can be
        ↪ calculated, else returns error message and 0
185     */
186     function currentlyWinningOption(bytes32 ballotName) external view
        ↪ returns (bytes32 winningOptionName, uint winningOptionVoteCount){
187
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
188         Ballot memory ballot;
189         //Search through all Ballots backwards, so that the recent ones are
            ↪ found first
190         for (int i = UIntConverterLib.toIntSafe(ballots.length); i > 0; i--)
            ↪  {
191             if (ballots[SafeMathInt.toUintSafe(i - 1)].name == ballotName) {
192                 ballot = ballots[SafeMathInt.toUintSafe(i - 1)];
193             }
194         }
195         //require(ballot.name.length > 0, "Ballot not found!");
196         //This is because some clients apparently can not deal with requires
            ↪  on view functions
197         if (ballot.name.length <= 0) {
198             return (bytes32("Ballot not found!"), 0);
199         }
200         uint winningVoteCount = 0;
201         //bool invalid = false;
202         int winningOptionIndex = - 1;
203         for (uint p = 0; p < ballot.optionVoteCounts.length; p++) {
204             if (ballot.optionVoteCounts[p] > winningVoteCount) {
205                 //invalid = false;
206                 winningVoteCount = ballot.optionVoteCounts[p];
207                 winningOptionIndex = UIntConverterLib.toIntSafe(p);
208             }
209             /*if (ballot.optionVoteCounts[p] == winningVoteCount){
210                 invalid =true;
211             }*/
212         }
213 
214         //return (bytes32("At least two votes are tied!"), winningVoteCount)
            ↪ ;
215 
216         //require(winningOptionIndex != - 1,"No votes yet!");
217         if (winningOptionIndex == - 1) {
218             return (bytes32("No votes yet!"), 0);
219         }
220 
221         winningOptionName = ballot.optionNames[SafeMathInt.toUintSafe(
            ↪ winningOptionIndex)];
222         winningOptionVoteCount = ballot.optionVoteCounts[SafeMathInt.
            ↪ toUintSafe(winningOptionIndex)];
223     }
224 
225 
226     /**
227     * @dev Queries the balance of `_owner` at a specific `_blockNumber`
228     * @param _owner The address from which the balance will be retrieved
229     * @param _blockNumber The block number when the balance is queried
230     * @return The balance at `_blockNumber`
231     */
232     function balanceOfAt(address _owner, uint _blockNumber) public view
233     returns (uint) {
234 
235         return getValueAt(_historizedBalances[_owner], _blockNumber);
236 
237     }
238     /**
239      * @notice Total amount of tokens at a specific `_blockNumber`.
240      * @param _blockNumber The block number when the totalSupply is queried
241      * @return The total amount of tokens at `_blockNumber`
242     */
243     function totalSupplyAt(uint _blockNumber) public view returns (uint) {
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

```
244             return getValueAt(totalSupplyHistory, _blockNumber);
245         }
246 
247 
248         /**
249         * @dev overwrite erc20 function
250         * @notice mints tokens to _account, while keeping tabs on supply
251         * @param _account The address that will be assigned the new tokens
252         * @param _value The quantity of tokens generated
253         */
254         function _mint(address _account, uint256 _value) internal {
255             require(_account != address(0));
256             uint curTotalSupply = totalSupply();
257             uint previousBalanceTo = balanceOf(_account);
258             updateValueAtNow(totalSupplyHistory, curTotalSupply.add(_value));
259             updateValueAtNow(_historizedBalances[_account], previousBalanceTo.
                ↪ add(_value));
260             emit Transfer(address(0), _account, _value);
261         }
262 
263         /**
264         * @notice Burns `_amount` tokens from `_owner`
265         * @dev overwrites the erc20 implementation, is also used by _burnFrom
266         * @param _account The account whose tokens will be burnt.
267         * @param _value The amount that will be burnt.
268         */
269         function _burn(address _account, uint _value) internal {
270             //erc 20 checks
271             require(_account != address(0));
272             require(_value <= balanceOf(_account));
273 
274             uint curTotalSupply = totalSupply();
275             require(curTotalSupply >= _value);
276             uint previousBalanceFrom = balanceOf(_account);
277             require(previousBalanceFrom >= _value);
278             updateValueAtNow(totalSupplyHistory, curTotalSupply.sub(_value));
279             updateValueAtNow(_historizedBalances[_account], previousBalanceFrom.
                ↪ sub(_value));
280             emit Transfer(_account, address(0), _value);
281         }
282 
283         /// @dev get total number of tokens
284         function totalSupply() public view returns (uint) {
285             return totalSupplyAt(block.number);
286         }
287 
288 
289         /**
290         * @dev retrieve the number of tokens at a given block number
291         * @param checkpoints The history of values being queried
292         * @param _block The block number to retrieve the value at
293         * @return The number of tokens being queried
294         */
295         function getValueAt(Checkpoint[] storage checkpoints, uint _block
296         ) view internal returns (uint) {
297             if (checkpoints.length == 0) return 0;
298 
299             // Shortcut for the current value
300             if (_block >= checkpoints[checkpoints.length - 1].fromBlock)
301                 return checkpoints[checkpoints.length - 1].value;
302             if (_block < checkpoints[0].fromBlock) return 0;
303 
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
304         // Binary search of the value in the array
305         uint min = 0;
306         uint max = checkpoints.length - 1;
307         while (max > min) {
308             uint mid = (max.add(min).add(1)) / 2;
309             if (checkpoints[mid].fromBlock <= _block) {
310                 min = mid;
311             } else {
312                 max = mid - 1;
313             }
314         }
315         return checkpoints[min].value;
316     }
317     /**
318      * @dev update current value for balances or totalsupply
319      * @param checkpoints The history of data being updated
320      * @param _value The new number of tokens
321     */
322     function updateValueAtNow(Checkpoint[] storage checkpoints, uint _value)
        ↪  internal {
323         if ((checkpoints.length == 0)
324             || (checkpoints[checkpoints.length - 1].fromBlock < block.number
            ↪ )) {
325             Checkpoint storage newCheckPoint = checkpoints[checkpoints.
            ↪ length++];
326             newCheckPoint.fromBlock = uint128(block.number);
327             newCheckPoint.value = uint128(_value);
328         } else {
329             Checkpoint storage oldCheckPoint = checkpoints[checkpoints.
            ↪ length - 1];
330             oldCheckPoint.value = uint128(_value);
331         }
332     }
333
334 }
```

Listing A.3: The VotingToken contract, implementing voting funtionality, lowest in the inheritance hierarchy.

```
1  pragma solidity ^0.5.4;
2
3  //Based on work by ZeppelinOS Team at https://github.com/zeppelinos/labs/
   ↪ tree/master/upgradeability_using_unstructured_storage
4  contract UnstructuredProxy {
5      bytes32 private constant ownerPosition = keccak256("secblocks.proxy.
       ↪ owner");
6      bytes32 private constant implementationPosition = keccak256("secblocks.
       ↪ proxy.implementation");
7
8      /**
9      * @dev the constructor sets the owner
10     */
11     constructor(address owner) public {
12         require(address(0) != owner, "The owner of a proxy must not be null"
           ↪ );
13         _setProxyOwner(owner);
14     }
15
16     /**
17     * @dev upgrades the contained implementation of the Contract and
       ↪ initializes the contract
18     * Attention! Upgraded contract must extend the original implementation
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
19      * and follow the same storage layout (no switching of variables etc.)
20      * Contract must be Initializable
21     */
22     function upgradeToInit(address newImplementation) external
       ↪ onlyProxyOwner {
23         address currentImplementation = implementation();
24         require(currentImplementation != newImplementation, "Upgrading
           ↪ address identical to current Address");
25         setImplementation(newImplementation);
26         bytes memory payload = abi.encodeWithSignature("initialize(address)"
           ↪ , msg.sender);
27
28         bool success;
29         bytes memory returndata;
30         (success, returndata) = address(this).call(payload);
31         require(success, "Initialization did not work, ensure that the
           ↪ Contract is Initializable");
32     }
33
34     /**
35     * @dev upgrades the contained implementation of the Contract without
       ↪ initialization
36     * Attention! Upgraded contract must extend the original implementation
37     * and follow the same storage layout (no switching of variables etc.)
38     */
39     function upgradeTo(address newImplementation) external onlyProxyOwner {
40         address currentImplementation = implementation();
41         require(currentImplementation != newImplementation, "Upgrading
           ↪ address identical to current Address");
42         setImplementation(newImplementation);
43     }
44
45     /**
46     * @dev returns position of the implementation
47     */
48     function implementation() public view returns (address impl) {
49         bytes32 position = implementationPosition;
50         assembly {impl := sload(position)}
51     }
52
53     /**
54     * @dev sets position of the implementation
55     */
56     function setImplementation(address newImplementation) internal {
57         bytes32 position = implementationPosition;
58         assembly {sstore(position, newImplementation)}
59     }
60
61     /**
62     * @dev sets new Proxy owner. ATTENTION! Removes the owner rights of the
       ↪ executing address!
63     */
64     function setProxyOwner(address newProxyOwner) external onlyProxyOwner {
65         require(newProxyOwner != address(0));
66         _setProxyOwner(newProxyOwner);
67     }
68
69     /**
70     * @dev sets position of the implementation
71     */
72     function proxyOwner() public view returns (address owner) {
73         bytes32 position = ownerPosition;
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

```
74            assembly {owner := sload(position)}
75        }
76
77
78        /**
79         * @dev Fallback function allowing to perform a delegatecall to the
           ↪ given implementation.
80        * This function will return whatever the implementation call returns
81        * Is marked external, to save gas.
82        */
83        function() payable external {
84            address _impl = implementation();
85            require(_impl != address(0));
86
87            assembly {
88                let ptr := mload(0x40) //always the next free pointer
89                calldatacopy(ptr, 0, calldatasize)
90                let result := delegatecall(gas, _impl, ptr, calldatasize, 0, 0)
91                let size := returndatasize
92                returndatacopy(ptr, 0, size)
93
94                switch result
95                case 0 {revert(ptr, size)}
96                default {return (ptr, size)}
97            }
98        }
99
100       /**
101       * @dev internal ownersetter
102       */
103       function _setProxyOwner(address newProxyOwner) internal {
104           bytes32 position = ownerPosition;
105           assembly {sstore(position, newProxyOwner)}
106       }
107
108       modifier onlyProxyOwner() {
109           require(msg.sender == proxyOwner());
110           _;
111       }
112   }
```

Listing A.4: Unstructured Proxy, the source of upgradeability in our system.

```
1   pragma solidity ^0.5.4;
2
3
4   /**
5    * @dev For Contracts to be able to be deployed via a proxy, the setting of
     ↪ the resp. managing address (e.g. owner)
6    * in the constructor must be circumvented, as all storage is lost when
     ↪ switching to the proxy, and the constructor
7    * is not called again.
8    */
9   contract Initializable {
10      bool internal initialized=false;
11
12      /**
13       * @dev To be called in the process of proxy creation to re- set an
         ↪ owner/manager/controller
14       */
15      function initialize(address initialManager) external{
16          require(!initialized, "Initializable is already initialized");
17          initialized = true;
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
18          _initialize(initialManager);
19      }
20
21      /**
22      * @dev To be extended by Initializable Contracts, who need to set their
        ↪ respective managers in this functions
23      */
24      function _initialize(address _initialManager) internal;
25
26  }
```

Listing A.5: Initializable contract, inherited by all upgradeable contracts that need to be upgraded.

```
1   solium -d .
2
3   contracts\AML\TransferQueues.sol
4     20:12     warning    Assignment operator must have exactly single space on
      ↪ both sides of it.    operator-whitespace
5     21:12     warning    Assignment operator must have exactly single space on
      ↪ both sides of it.    operator-whitespace
6     53:12     warning    Assignment operator must have exactly single space on
      ↪ both sides of it.    operator-whitespace
7
8   contracts\Controlling\Controller.sol
9     219:8     warning    Assignment operator must have exactly single space on
      ↪ both sides of it.    operator-whitespace
10    221:8     warning    Assignment operator must have exactly single space on
      ↪ both sides of it.    operator-whitespace
11    234:4     warning    Line exceeds the limit of 145 characters
      ↪                                      max-len
12
13  contracts\Controlling\InsiderListVerifier.sol
14    30:12      warning    Line contains trailing whitespace    no-trailing-
      ↪ whitespace
15    34:12      warning    Line contains trailing whitespace    no-trailing-
      ↪ whitespace
16    50:12      warning    Line contains trailing whitespace    no-trailing-
      ↪ whitespace
17    54:12      warning    Line contains trailing whitespace    no-trailing-
      ↪ whitespace
18    70:12      warning    Line contains trailing whitespace    no-trailing-
      ↪ whitespace
19    74:12      warning    Line contains trailing whitespace    no-trailing-
      ↪ whitespace
20    90:12      warning    Line contains trailing whitespace    no-trailing-
      ↪ whitespace
21    94:12      warning    Line contains trailing whitespace    no-trailing-
      ↪ whitespace
22    110:12     warning    Line contains trailing whitespace    no-trailing-
      ↪ whitespace
23    114:12     warning    Line contains trailing whitespace    no-trailing-
      ↪ whitespace
24
25  contracts\Controlling\KYCVerifier.sol
26    32:12      warning    Line contains trailing whitespace    no-trailing-
      ↪ whitespace
27    36:12      warning    Line contains trailing whitespace    no-trailing-
      ↪ whitespace
28    53:12      warning    Line contains trailing whitespace    no-trailing-
      ↪ whitespace
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
29    57:12      warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
30    73:12      warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
31    77:12      warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
32    93:12      warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
33    97:12      warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
34    113:12     warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
35    117:12     warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
36
37  contracts\Controlling\PEPListVerifier.sol
38    11:1       warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
39    28:12      warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
40    32:12      warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
41    66:12      warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
42    70:12      warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
43    86:12      warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
44    90:12      warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
45    106:12     warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
46    110:12     warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
47
48  contracts\Interfaces\IVerifier.sol
49    8:1      warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
50    14:1     warning    Line contains trailing whitespace    no-trailing-
    ↪ whitespace
51
52  contracts\Libraries\SafeMathInt.sol
53    5:8      warning    Provide an error message for require().    error-reason
54    12:8     warning    Provide an error message for require().    error-reason
55    17:8     warning    Provide an error message for require().    error-reason
56
57  contracts\Libraries\UIntConverterLib.sol
58    6:8     warning    Provide an error message for require().    error-reason
59
60  contracts\Mocks\MockContract.sol
61    300:12     error      Avoid using Inline Assembly.
    ↪                                            security/no-inline-assembly
    ↪ 317:8     error      Avoid using Inline Assembly.
    ↪                                            security/no-inline-assembly
    ↪ 327:23    warning    Visibility modifier "external" should come before
    ↪ other modifiers.    visibility-first
62    329:8      error      Avoid using Inline Assembly.
    ↪                                            security/no-inline-assembly
    ↪ 368:8     error      Avoid using Inline Assembly.
    ↪                                            security/no-inline-assembly
63  contracts\Openzeppelin\Address.sol
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
64    24:8    error    Avoid using Inline Assembly.    security/no-inline-
      ↪ assembly
65
66  contracts\Proxy\UnstructuredProxy.sol
67    29:46    warning    Avoid using low-level function 'call'.
      ↪                              security/no-low-level-calls
68    49:8     error      Avoid using Inline Assembly.
      ↪                                        security/no-inline-assembly
69    57:8     error      Avoid using Inline Assembly.
      ↪                                        security/no-inline-assembly
70    64:8     warning    Provide an error message for require().
      ↪                             error-reason
71    73:8     error      Avoid using Inline Assembly.
      ↪                                        security/no-inline-assembly
72    82:23    warning    Visibility modifier "external" should come before
      ↪ other modifiers.    visibility-first
73    84:8     warning    Provide an error message for require().
      ↪                             error-reason
74    86:8     error      Avoid using Inline Assembly.
      ↪                                        security/no-inline-assembly
75    104:8    error      Avoid using Inline Assembly.
      ↪                                        security/no-inline-assembly
76    108:8    warning    Provide an error message for require().
      ↪                             error-reason
77
78  contracts\Tokens\DividendToken.sol
79    150:31    warning    'undefined': The first argument must not be preceded
      ↪ by any whitespace or comments (only '(').    function-whitespace
80
81  contracts\Tokens\ERC1594.sol
82    54:8      warning    Provide an error message for require().
      ↪                                  error-reason
83    177:8     warning    Assignment operator must have exactly single space on
      ↪  both sides of it.    operator-whitespace
84    198:8     warning    Assignment operator must have exactly single space on
      ↪  both sides of it.    operator-whitespace
85    212:8     warning    Line exceeds the limit of 145 characters
      ↪                                   max-len
86    219:8     warning    Assignment operator must have exactly single space on
      ↪  both sides of it.    operator-whitespace
87    224:12    warning    Assignment operator must have exactly single space on
      ↪  both sides of it.    operator-whitespace
88    225:28    warning    Avoid using 'now' (alias to 'block.timestamp').
      ↪                         security/no-block-members
89    231:30    warning    Avoid using 'now' (alias to 'block.timestamp').
      ↪                         security/no-block-members
90
91  contracts\Tokens\ERC20.sol
92    185:37    warning    There should be no whitespace or comments between
      ↪ argument and the comma following it.    comma-whitespace
93
94  contracts\Tokens\VotingToken.sol
95    133:8     warning    Line exceeds the limit of 145 characters
      ↪                                              max-len
96    164:33    warning    Avoid using 'now' (alias to 'block.timestamp').
      ↪                                    security/no-block-members
97    251:8     warning    Provide an error message for require().
      ↪                                              error-reason
98    267:8     warning    Provide an error message for require().
      ↪                                              error-reason
99    268:8     warning    Provide an error message for require().
      ↪                                              error-reason
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
100   271:8     warning    Provide an error message for require().
      ↪                                                 error-reason
101   273:8     warning    Provide an error message for require().
      ↪                                                 error-reason
102   292:11    warning    Visibility modifier "internal" should come before
      ↪ other modifiers.                      visibility-first
103   319:36    warning    Operator "||" should be on the line where left side
      ↪ of the Binary expression ends.     operator-whitespace
```

Listing A.6: Output of Solium Styleguide after removing critical Errors

```
1  INFO:Detectors:
2  VotingToken._historizedBalances (Tokens\VotingToken.sol#45) is never
   ↪ initialized. It is used in:
3          - _transfer (Tokens\VotingToken.sol#84-104)
4          - _mint (Tokens\VotingToken.sol#250-257)
5          - _burn (Tokens\VotingToken.sol#265-277)
6          - balanceOfAt (Tokens\VotingToken.sol#228-233)
7  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#
   ↪ uninitialized-state-variables
8  INFO:Detectors:
9  Address.isContract (Openzeppelin\Address.sol#17-26) is declared view but
   ↪ contains assembly code
10 MockContract.uintToBytes (Mocks\MockContract.sol#315-318) is declared view
   ↪ but contains assembly code
11 UnstructuredProxy.implementation (Proxy\UnstructuredProxy.sol#47-50) is
   ↪ declared view but contains assembly code
12 UnstructuredProxy.proxyOwner (Proxy\UnstructuredProxy.sol#71-74) is declared
   ↪  view but contains assembly code
13 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#
   ↪ constant-functions-changing-the-state
14 INFO:Detectors:
15 Contract locking ether found in :
16         Contract MockContract has payable functions:
17          - fallback (Mocks\MockContract.sol#327-371)
18         But does not have a function to withdraw the ether
19 Contract locking ether found in :
20         Contract UnstructuredProxy has payable functions:
21          - fallback (Proxy\UnstructuredProxy.sol#82-97)
22         But does not have a function to withdraw the ether
23 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#
   ↪ contracts-that-lock-ether
24 INFO:Detectors:
25 ballot in VotingToken.currentlyWinningOption (Tokens\VotingToken.sol#192) is
   ↪  a local variable never initialazed
26 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#
   ↪ uninitialized-local-variables
27 INFO:Detectors:
28 Controller.verifyAllTransfer has external calls inside a loop: "verified =
   ↪ verifier.verifyTransfer(_from,_to,_value,_data)" (Controlling\Controller.
   ↪ sol#137)
29 Controller.verifyAllTransferFrom has external calls inside a loop: "verified
   ↪  = verifier.verifyTransferFrom(_from,_to,spender,_value,_data)" (
   ↪ Controlling\Controller.sol#155)
30 Controller.verifyAllRedeem has external calls inside a loop: "verified =
   ↪ verifier.verifyRedeem(_sender,_value,_data)" (Controlling\Controller.sol
   ↪ #173)
31 Controller.verifyAllRedeemFrom has external calls inside a loop: "verified =
   ↪  verifier.verifyRedeemFrom(_sender,_tokenholder,_value,_data)" (
   ↪ Controlling\Controller.sol#191)
32 Controller.verifyAllIssue has external calls inside a loop: "verified =
   ↪ verifier.verifyIssue(_tokenholder,_value,_data)" (Controlling\Controller.
   ↪ sol#209)
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
33 Controller.checkAllTransfer has external calls inside a loop: "verified =
   ↪ verifier.verifyTransfer(_from,_to,_value,_data)" (Controlling\Controller.
   ↪ sol#227)
34 Controller.checkAllTransferFrom has external calls inside a loop: "verified
   ↪ = verifier.verifyTransferFrom(_from,_to,spender,_value,_data)" (
   ↪ Controlling\Controller.sol#246)
35 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/
   ↪ _edit#calls-inside-a-loop
36 INFO:Detectors:
37 Address.isContract uses assembly (Openzeppelin\Address.sol#17-26)
38         - Openzeppelin\Address.sol#24
39 MockContract.useAllGas uses assembly (Mocks\MockContract.sol#297-305)
40         - Mocks\MockContract.sol#300-303
41 MockContract.uintToBytes uses assembly (Mocks\MockContract.sol#315-318)
42         - Mocks\MockContract.sol#317
43 MockContract.fallback uses assembly (Mocks\MockContract.sol#327-371)
44         - Mocks\MockContract.sol#329-331
45         - Mocks\MockContract.sol#368-370
46 UnstructuredProxy.implementation uses assembly (Proxy\UnstructuredProxy.sol
   ↪ #47-50)
47         - Proxy\UnstructuredProxy.sol#49
48 UnstructuredProxy.setImplementation uses assembly (Proxy\UnstructuredProxy.
   ↪ sol#55-58)
49         - Proxy\UnstructuredProxy.sol#57
50 UnstructuredProxy.proxyOwner uses assembly (Proxy\UnstructuredProxy.sol
   ↪ #71-74)
51         - Proxy\UnstructuredProxy.sol#73
52 UnstructuredProxy.fallback uses assembly (Proxy\UnstructuredProxy.sol#82-97)
53         - Proxy\UnstructuredProxy.sol#86-96
54 UnstructuredProxy._setProxyOwner uses assembly (Proxy\UnstructuredProxy.sol
   ↪ #102-105)
55         - Proxy\UnstructuredProxy.sol#104
56 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#
   ↪ assembly-usage
57 INFO:Detectors:
58 IERC20.allowance (Openzeppelin\IERC20.sol#34) should be declared external
59 ERC20.allowance (Tokens\ERC20.sol#41-43) should be declared external
60 ERC20.approve (Tokens\ERC20.sol#54-63) should be declared external
61 IERC20.approve (Openzeppelin\IERC20.sol#50) should be declared external
62 IERC20.transferFrom (Openzeppelin\IERC20.sol#61) should be declared external
63 ERC20.transferFrom (Tokens\ERC20.sol#108-111) should be declared external
64 ERC20.increaseAllowance (Tokens\ERC20.sol#144-147) should be declared
   ↪ external
65 ERC20.decreaseAllowance (Tokens\ERC20.sol#158-161) should be declared
   ↪ external
66 ERC20Mock.mint (Mocks\ERC20Mock.sol#8-10) should be declared external
67 ERC20Mock.burn (Mocks\ERC20Mock.sol#12-14) should be declared external
68 ERC20Mock.burnFrom (Mocks\ERC20Mock.sol#16-18) should be declared external
69 ERC20Mock.transferInternal (Mocks\ERC20Mock.sol#20-22) should be declared
   ↪ external
70 ERC20Mock.approveInternal (Mocks\ERC20Mock.sol#24-26) should be declared
   ↪ external
71 Migrations.setCompleted (Migrations.sol#15-17) should be declared external
72 Migrations.upgrade (Migrations.sol#19-22) should be declared external
73 MockContract.updateInvocationCount (Mocks\MockContract.sol#320-325) should
   ↪ be declared external
74 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#
   ↪ public-function-that-could-be-declared-as-external
75 INFO:Detectors:
76 Pragma version "^0.5.4" necessitates versions too recent to be trusted.
   ↪ Consider deploying with 0.5.3 (Openzeppelin\Address.sol#1)
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
77  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Controlling\Controller.sol#1)
78  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Tokens\DividendToken.sol#1)
79  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Tokens\ERC1594.sol#1)
80  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Mocks\ERC1594Mock.sol#1)
81  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Tokens\ERC20.sol#1)
82  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Mocks\ERC20Mock.sol#1)
83  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Mocks\GeneralVerifierMock.sol#1)
84  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Interfaces\IERC1594.sol#1)
85  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Openzeppelin\IERC20.sol#1)
86  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Proxy\Initializable.sol#1)
87  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Roles\InsiderListManagerRole.sol#1)
88  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Controlling\InsiderListVerifier.sol#1)
89  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Roles\IssuerRole.sol#1)
90  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Interfaces\IVerifier.sol#1)
91  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Roles\KYCListManagerRole.sol#1)
92  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Controlling\KYCVerifier.sol#1)
93  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Migrations.sol#1)
94  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Mocks\MockContract.sol#1)
95  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Roles\OrchestratorRole.sol#1)
96  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Openzeppelin\Ownable.sol#1)
97  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Roles\PEPListManagerRole.sol#1)
98  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Controlling\PEPListVerifier.sol#1)
99  Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Registry.sol#1)
100 Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Openzeppelin\Roles.sol#1)
101 Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Openzeppelin\SafeMath.sol#1)
102 Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Libraries\SafeMathInt.sol#1)
103 Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (AML\TransferQueues.sol#1)
104 Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Libraries\UIntConverterLib.sol#1)
105 Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Proxy\UnstructuredProxy.sol#1)
106 Pragma version "^0.5.4" necessitates versions too recent to be trusted.
    ↪ Consider deploying with 0.5.3 (Tokens\VotingToken.sol#1)
107 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#
    ↪ incorrect-versions-of-solidity
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
108 INFO:Detectors:
109 Low level call in MockContract.fallback (Mocks\MockContract.sol#327-371):
110         -(r) = address(this).call.gas(100000)(abi.encodeWithSignature(
            ↪ updateInvocationCount(bytes4,bytes),methodId,msg.data)) Mocks\
            ↪ MockContract.sol#365
111 Low level call in UnstructuredProxy.upgradeToInit (Proxy\UnstructuredProxy.
    ↪ sol#21-31):
112         -(success,returndata) = address(this).call(payload) Proxy\
            ↪ UnstructuredProxy.sol#29
113 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low
    ↪ -level-calls
114 INFO:Detectors:
115 Parameter '_verifier' of Controller.addVerifier (Controlling\Controller.sol
    ↪ #44) is not in mixedCase
116 Parameter '_verifier' of Controller.removeVerifier (Controlling\Controller.
    ↪ sol#55) is not in mixedCase
117 Parameter '_verifier' of Controller.setKYCVerifier (Controlling\Controller.
    ↪ sol#70) is not in mixedCase
118 Parameter '_verifier' of Controller.setPEPListVerifier (Controlling\
    ↪ Controller.sol#81) is not in mixedCase
119 Parameter '_verifier' of Controller.setInsiderListVerifier (Controlling\
    ↪ Controller.sol#92) is not in mixedCase
120 Parameter '_verifierToRemove' of Controller.remove (Controlling\Controller.
    ↪ sol#103) is not in mixedCase
121 Parameter '_from' of Controller.verifyAllTransfer (Controlling\Controller.
    ↪ sol#125) is not in mixedCase
122 Parameter '_to' of Controller.verifyAllTransfer (Controlling\Controller.sol
    ↪ #125) is not in mixedCase
123 Parameter '_value' of Controller.verifyAllTransfer (Controlling\Controller.
    ↪ sol#125) is not in mixedCase
124 Parameter '_data' of Controller.verifyAllTransfer (Controlling\Controller.
    ↪ sol#125) is not in mixedCase
125 Parameter '_from' of Controller.verifyAllTransferFrom (Controlling\
    ↪ Controller.sol#143) is not in mixedCase
126 Parameter '_to' of Controller.verifyAllTransferFrom (Controlling\Controller.
    ↪ sol#143) is not in mixedCase
127 Parameter '_value' of Controller.verifyAllTransferFrom (Controlling\
    ↪ Controller.sol#143) is not in mixedCase
128 Parameter '_data' of Controller.verifyAllTransferFrom (Controlling\
    ↪ Controller.sol#143) is not in mixedCase
129 Parameter '_sender' of Controller.verifyAllRedeem (Controlling\Controller.
    ↪ sol#161) is not in mixedCase
130 Parameter '_value' of Controller.verifyAllRedeem (Controlling\Controller.sol
    ↪ #161) is not in mixedCase
131 Parameter '_data' of Controller.verifyAllRedeem (Controlling\Controller.sol
    ↪ #161) is not in mixedCase
132 Parameter '_sender' of Controller.verifyAllRedeemFrom (Controlling\
    ↪ Controller.sol#179) is not in mixedCase
133 Parameter '_tokenholder' of Controller.verifyAllRedeemFrom (Controlling\
    ↪ Controller.sol#179) is not in mixedCase
134 Parameter '_value' of Controller.verifyAllRedeemFrom (Controlling\Controller
    ↪ .sol#179) is not in mixedCase
135 Parameter '_data' of Controller.verifyAllRedeemFrom (Controlling\Controller.
    ↪ sol#179) is not in mixedCase
136 Parameter '_tokenholder' of Controller.verifyAllIssue (Controlling\
    ↪ Controller.sol#197) is not in mixedCase
137 Parameter '_value' of Controller.verifyAllIssue (Controlling\Controller.sol
    ↪ #197) is not in mixedCase
138 Parameter '_data' of Controller.verifyAllIssue (Controlling\Controller.sol
    ↪ #197) is not in mixedCase
139 Parameter '_from' of Controller.checkAllTransfer (Controlling\Controller.sol
    ↪ #215) is not in mixedCase
```


TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
140 Parameter '_to' of Controller.checkAllTransfer (Controlling\Controller.sol
    ↪ #215) is not in mixedCase
141 Parameter '_value' of Controller.checkAllTransfer (Controlling\Controller.
    ↪ sol#215) is not in mixedCase
142 Parameter '_data' of Controller.checkAllTransfer (Controlling\Controller.sol
    ↪ #215) is not in mixedCase
143 Parameter '_from' of Controller.checkAllTransferFrom (Controlling\Controller
    ↪ .sol#234) is not in mixedCase
144 Parameter '_to' of Controller.checkAllTransferFrom (Controlling\Controller.
    ↪ sol#234) is not in mixedCase
145 Parameter '_value' of Controller.checkAllTransferFrom (Controlling\
    ↪ Controller.sol#234) is not in mixedCase
146 Parameter '_data' of Controller.checkAllTransferFrom (Controlling\Controller
    ↪ .sol#234) is not in mixedCase
147 Parameter '_to' of DividendToken.transferWithData (Tokens\DividendToken.sol
    ↪ #145) is not in mixedCase
148 Parameter '_value' of DividendToken.transferWithData (Tokens\DividendToken.
    ↪ sol#145) is not in mixedCase
149 Parameter '_data' of DividendToken.transferWithData (Tokens\DividendToken.
    ↪ sol#145) is not in mixedCase
150 Parameter '_from' of DividendToken.transferFromWithData (Tokens\
    ↪ DividendToken.sol#161) is not in mixedCase
151 Parameter '_to' of DividendToken.transferFromWithData (Tokens\DividendToken.
    ↪ sol#161) is not in mixedCase
152 Parameter '_value' of DividendToken.transferFromWithData (Tokens\
    ↪ DividendToken.sol#161) is not in mixedCase
153 Parameter '_data' of DividendToken.transferFromWithData (Tokens\
    ↪ DividendToken.sol#161) is not in mixedCase
154 Parameter '_tokenHolder' of DividendToken.issue (Tokens\DividendToken.sol
    ↪ #176) is not in mixedCase
155 Parameter '_value' of DividendToken.issue (Tokens\DividendToken.sol#176) is
    ↪ not in mixedCase
156 Parameter '_data' of DividendToken.issue (Tokens\DividendToken.sol#176) is
    ↪ not in mixedCase
157 Parameter '_value' of DividendToken.redeem (Tokens\DividendToken.sol#189) is
    ↪  not in mixedCase
158 Parameter '_data' of DividendToken.redeem (Tokens\DividendToken.sol#189) is
    ↪ not in mixedCase
159 Parameter '_tokenHolder' of DividendToken.redeemFrom (Tokens\DividendToken.
    ↪ sol#204) is not in mixedCase
160 Parameter '_value' of DividendToken.redeemFrom (Tokens\DividendToken.sol
    ↪ #204) is not in mixedCase
161 Parameter '_data' of DividendToken.redeemFrom (Tokens\DividendToken.sol#204)
    ↪  is not in mixedCase
162 Parameter '_owner' of DividendToken.dividendOf (Tokens\DividendToken.sol
    ↪ #105) is not in mixedCase
163 Parameter '_owner' of DividendToken.withdrawableDividendOf (Tokens\
    ↪ DividendToken.sol#114) is not in mixedCase
164 Parameter '_owner' of DividendToken.withdrawnDividendOf (Tokens\
    ↪ DividendToken.sol#123) is not in mixedCase
165 Parameter '_owner' of DividendToken.accumulativeDividendOf (Tokens\
    ↪ DividendToken.sol#134) is not in mixedCase
166 Constant 'DividendToken.magnitude' (Tokens\DividendToken.sol#35) is not in
    ↪ UPPER_CASE_WITH_UNDERSCORES
167 Parameter '_to' of ERC1594.transferWithData (Tokens\ERC1594.sol#65) is not
    ↪ in mixedCase
168 Parameter '_value' of ERC1594.transferWithData (Tokens\ERC1594.sol#65) is
    ↪ not in mixedCase
169 Parameter '_data' of ERC1594.transferWithData (Tokens\ERC1594.sol#65) is not
    ↪  in mixedCase
170 Parameter '_from' of ERC1594.transferFromWithData (Tokens\ERC1594.sol#82) is
    ↪  not in mixedCase
```

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
171 Parameter '_to' of ERC1594.transferFromWithData (Tokens\ERC1594.sol#82) is
    ↪ not in mixedCase
172 Parameter '_value' of ERC1594.transferFromWithData (Tokens\ERC1594.sol#82)
    ↪ is not in mixedCase
173 Parameter '_data' of ERC1594.transferFromWithData (Tokens\ERC1594.sol#82) is
    ↪  not in mixedCase
174 Parameter '_tokenHolder' of ERC1594.issue (Tokens\ERC1594.sol#110) is not in
    ↪  mixedCase
175 Parameter '_value' of ERC1594.issue (Tokens\ERC1594.sol#110) is not in
    ↪ mixedCase
176 Parameter '_data' of ERC1594.issue (Tokens\ERC1594.sol#110) is not in
    ↪ mixedCase
177 Parameter '_value' of ERC1594.redeem (Tokens\ERC1594.sol#137) is not in
    ↪ mixedCase
178 Parameter '_data' of ERC1594.redeem (Tokens\ERC1594.sol#137) is not in
    ↪ mixedCase
179 Parameter '_tokenHolder' of ERC1594.redeemFrom (Tokens\ERC1594.sol#155) is
    ↪ not in mixedCase
180 Parameter '_value' of ERC1594.redeemFrom (Tokens\ERC1594.sol#155) is not in
    ↪ mixedCase
181 Parameter '_data' of ERC1594.redeemFrom (Tokens\ERC1594.sol#155) is not in
    ↪ mixedCase
182 Parameter '_to' of ERC1594.canTransfer (Tokens\ERC1594.sol#174) is not in
    ↪ mixedCase
183 Parameter '_value' of ERC1594.canTransfer (Tokens\ERC1594.sol#174) is not in
    ↪  mixedCase
184 Parameter '_data' of ERC1594.canTransfer (Tokens\ERC1594.sol#174) is not in
    ↪ mixedCase
185 Parameter '_from' of ERC1594.canTransferFrom (Tokens\ERC1594.sol#195) is not
    ↪  in mixedCase
186 Parameter '_to' of ERC1594.canTransferFrom (Tokens\ERC1594.sol#195) is not
    ↪ in mixedCase
187 Parameter '_value' of ERC1594.canTransferFrom (Tokens\ERC1594.sol#195) is
    ↪ not in mixedCase
188 Parameter '_data' of ERC1594.canTransferFrom (Tokens\ERC1594.sol#195) is not
    ↪  in mixedCase
189 Parameter '_queues' of ERC1594.setTransferQueues (Tokens\ERC1594.sol#44) is
    ↪ not in mixedCase
190 Parameter '_controller' of ERC1594.setController (Tokens\ERC1594.sol#47) is
    ↪ not in mixedCase
191 Parameter '_name' of ERC1594.setName (Tokens\ERC1594.sol#53) is not in
    ↪ mixedCase
192 Function 'ERC1594._checkAMLConstraints' (Tokens\ERC1594.sol#209-213) is not
    ↪ in mixedCase
193 Function 'ERC1594._updateTransferListAndCalculateSum' (Tokens\ERC1594.sol
    ↪ #218-235) is not in mixedCase
194 Parameter '_from' of ERC1594._updateTransferListAndCalculateSum (Tokens\
    ↪ ERC1594.sol#218) is not in mixedCase
195 Parameter '_value' of ERC1594._updateTransferListAndCalculateSum (Tokens\
    ↪ ERC1594.sol#218) is not in mixedCase
196 Parameter '_owner' of ERC20.balanceOf (Tokens\ERC20.sol#31) is not in
    ↪ mixedCase
197 Function 'ERC20._approve' (Tokens\ERC20.sol#76-82) is not in mixedCase
198 Parameter '_to' of ERC20.transferWithData (Tokens\ERC20.sol#98) is not in
    ↪ mixedCase
199 Parameter '_value' of ERC20.transferWithData (Tokens\ERC20.sol#98) is not in
    ↪  mixedCase
200 Parameter '_from' of ERC20.transferFromWithData (Tokens\ERC20.sol#117) is
    ↪ not in mixedCase
201 Parameter '_to' of ERC20.transferFromWithData (Tokens\ERC20.sol#117) is not
    ↪ in mixedCase
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
202 Parameter '_value' of ERC20.transferFromWithData (Tokens\ERC20.sol#117) is
    ↪ not in mixedCase
203 Function 'ERC20._transferFrom' (Tokens\ERC20.sol#130-133) is not in
    ↪ mixedCase
204 Function 'ERC20._transfer' (Tokens\ERC20.sol#169-175) is not in mixedCase
205 Function 'ERC20._mint' (Tokens\ERC20.sol#184-189) is not in mixedCase
206 Function 'ERC20._burn' (Tokens\ERC20.sol#197-203) is not in mixedCase
207 Function 'ERC20._burnFrom' (Tokens\ERC20.sol#212-215) is not in mixedCase
208 Variable 'ERC20._allowed' (Tokens\ERC20.sol#15) is not in mixedCase
209 Function 'Initializable._initialize' (Proxy\Initializable.sol#24) is not in
    ↪ mixedCase
210 Function 'InsiderListManagerRole._initialize' (Roles\InsiderListManagerRole.
    ↪ sol#22-24) is not in mixedCase
211 Parameter '_initialManager' of InsiderListManagerRole._initialize (Roles\
    ↪ InsiderListManagerRole.sol#22) is not in mixedCase
212 Function 'InsiderListManagerRole._addInsiderListManager' (Roles\
    ↪ InsiderListManagerRole.sol#43-46) is not in mixedCase
213 Function 'InsiderListManagerRole._removeInsiderListManager' (Roles\
    ↪ InsiderListManagerRole.sol#48-51) is not in mixedCase
214 Parameter '_tokenHolder' of InsiderListVerifier.verifyIssue (Controlling\
    ↪ InsiderListVerifier.sol#25) is not in mixedCase
215 Parameter '_from' of InsiderListVerifier.verifyTransfer (Controlling\
    ↪ InsiderListVerifier.sol#45) is not in mixedCase
216 Parameter '_to' of InsiderListVerifier.verifyTransfer (Controlling\
    ↪ InsiderListVerifier.sol#45) is not in mixedCase
217 Parameter '_from' of InsiderListVerifier.verifyTransferFrom (Controlling\
    ↪ InsiderListVerifier.sol#65) is not in mixedCase
218 Parameter '_to' of InsiderListVerifier.verifyTransferFrom (Controlling\
    ↪ InsiderListVerifier.sol#65) is not in mixedCase
219 Parameter '_spender' of InsiderListVerifier.verifyTransferFrom (Controlling\
    ↪ InsiderListVerifier.sol#65) is not in mixedCase
220 Parameter '_sender' of InsiderListVerifier.verifyRedeem (Controlling\
    ↪ InsiderListVerifier.sol#85) is not in mixedCase
221 Parameter '_sender' of InsiderListVerifier.verifyRedeemFrom (Controlling\
    ↪ InsiderListVerifier.sol#105) is not in mixedCase
222 Parameter '_tokenHolder' of InsiderListVerifier.verifyRedeemFrom (
    ↪ Controlling\InsiderListVerifier.sol#105) is not in mixedCase
223 Parameter '_addr' of InsiderListVerifier.addAddressToBlacklist (Controlling\
    ↪ InsiderListVerifier.sol#122) is not in mixedCase
224 Parameter '_addr' of InsiderListVerifier.removeAddressFromBlacklist (
    ↪ Controlling\InsiderListVerifier.sol#131) is not in mixedCase
225 Function 'InsiderListVerifier._onBlacklist' (Controlling\InsiderListVerifier
    ↪ .sol#140-142) is not in mixedCase
226 Parameter '_addr' of InsiderListVerifier._onBlacklist (Controlling\
    ↪ InsiderListVerifier.sol#140) is not in mixedCase
227 Function 'IssuerRole._addIssuer' (Roles\IssuerRole.sol#37-40) is not in
    ↪ mixedCase
228 Function 'IssuerRole._removeIssuer' (Roles\IssuerRole.sol#42-45) is not in
    ↪ mixedCase
229 Function 'KYCListManagerRole._initialize' (Roles\KYCListManagerRole.sol
    ↪ #22-24) is not in mixedCase
230 Parameter '_initialManager' of KYCListManagerRole._initialize (Roles\
    ↪ KYCListManagerRole.sol#22) is not in mixedCase
231 Function 'KYCListManagerRole._addKYCListManager' (Roles\KYCListManagerRole.
    ↪ sol#43-46) is not in mixedCase
232 Function 'KYCListManagerRole._removeKYCListManager' (Roles\
    ↪ KYCListManagerRole.sol#48-51) is not in mixedCase
233 Parameter '_tokenHolder' of KYCVerifier.verifyIssue (Controlling\KYCVerifier
    ↪ .sol#27) is not in mixedCase
234 Parameter '_from' of KYCVerifier.verifyTransfer (Controlling\KYCVerifier.sol
    ↪ #47) is not in mixedCase
```

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
235 Parameter '_to' of KYCVerifier.verifyTransfer (Controlling\KYCVerifier.sol
    ↪ #47) is not in mixedCase
236 Parameter '_from' of KYCVerifier.verifyTransferFrom (Controlling\KYCVerifier
    ↪ .sol#68) is not in mixedCase
237 Parameter '_to' of KYCVerifier.verifyTransferFrom (Controlling\KYCVerifier.
    ↪ sol#68) is not in mixedCase
238 Parameter '_spender' of KYCVerifier.verifyTransferFrom (Controlling\
    ↪ KYCVerifier.sol#68) is not in mixedCase
239 Parameter '_sender' of KYCVerifier.verifyRedeem (Controlling\KYCVerifier.sol
    ↪ #88) is not in mixedCase
240 Parameter '_sender' of KYCVerifier.verifyRedeemFrom (Controlling\KYCVerifier
    ↪ .sol#108) is not in mixedCase
241 Parameter '_tokenHolder' of KYCVerifier.verifyRedeemFrom (Controlling\
    ↪ KYCVerifier.sol#108) is not in mixedCase
242 Parameter '_addr' of KYCVerifier.addAddressToWhitelist (Controlling\
    ↪ KYCVerifier.sol#125) is not in mixedCase
243 Parameter '_addr' of KYCVerifier.removeAddressFromWhitelist (Controlling\
    ↪ KYCVerifier.sol#134) is not in mixedCase
244 Function 'KYCVerifier._onWhitelist' (Controlling\KYCVerifier.sol#143-145) is
    ↪  not in mixedCase
245 Parameter '_addr' of KYCVerifier._onWhitelist (Controlling\KYCVerifier.sol
    ↪ #143) is not in mixedCase
246 Parameter 'new_address' of Migrations.upgrade (Migrations.sol#19) is not in
    ↪ mixedCase
247 Variable 'Migrations.last_completed_migration' (Migrations.sol#5) is not in
    ↪ mixedCase
248 Function 'MockContract._givenAnyReturn' (Mocks\MockContract.sol#125-128) is
    ↪ not in mixedCase
249 Function 'MockContract._givenCalldataReturn' (Mocks\MockContract.sol
    ↪ #161-165) is not in mixedCase
250 Function 'MockContract._givenMethodReturn' (Mocks\MockContract.sol#184-189)
    ↪ is not in mixedCase
251 Function 'OrchestratorRole._initialize' (Roles\OrchestratorRole.sol#21-23)
    ↪ is not in mixedCase
252 Parameter '_initialManager' of OrchestratorRole._initialize (Roles\
    ↪ OrchestratorRole.sol#21) is not in mixedCase
253 Function 'OrchestratorRole._addOrchestrator' (Roles\OrchestratorRole.sol
    ↪ #42-45) is not in mixedCase
254 Function 'OrchestratorRole._removeOrchestrator' (Roles\OrchestratorRole.sol
    ↪ #47-50) is not in mixedCase
255 Function 'Ownable._initialize' (Openzeppelin\Ownable.sol#30-33) is not in
    ↪ mixedCase
256 Parameter '_initialManager' of Ownable._initialize (Openzeppelin\Ownable.sol
    ↪ #30) is not in mixedCase
257 Function 'Ownable._transferOwnership' (Openzeppelin\Ownable.sol#80-84) is
    ↪ not in mixedCase
258 Function 'PEPListManagerRole._initialize' (Roles\PEPListManagerRole.sol
    ↪ #22-24) is not in mixedCase
259 Parameter '_initialManager' of PEPListManagerRole._initialize (Roles\
    ↪ PEPListManagerRole.sol#22) is not in mixedCase
260 Function 'PEPListManagerRole._addPEPListManager' (Roles\PEPListManagerRole.
    ↪ sol#43-46) is not in mixedCase
261 Function 'PEPListManagerRole._removePEPListManager' (Roles\
    ↪ PEPListManagerRole.sol#48-51) is not in mixedCase
262 Parameter '_tokenHolder' of PEPListVerifier.verifyIssue (Controlling\
    ↪ PEPListVerifier.sol#23) is not in mixedCase
263 Parameter '_from' of PEPListVerifier.verifyTransfer (Controlling\
    ↪ PEPListVerifier.sol#43) is not in mixedCase
264 Parameter '_to' of PEPListVerifier.verifyTransfer (Controlling\
    ↪ PEPListVerifier.sol#43) is not in mixedCase
265 Parameter '_from' of PEPListVerifier.verifyTransferFrom (Controlling\
    ↪ PEPListVerifier.sol#61) is not in mixedCase
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
266 Parameter '_to' of PEPListVerifier.verifyTransferFrom (Controlling\
    ↪ PEPListVerifier.sol#61) is not in mixedCase
267 Parameter '_spender' of PEPListVerifier.verifyTransferFrom (Controlling\
    ↪ PEPListVerifier.sol#61) is not in mixedCase
268 Parameter '_sender' of PEPListVerifier.verifyRedeem (Controlling\
    ↪ PEPListVerifier.sol#81) is not in mixedCase
269 Parameter '_sender' of PEPListVerifier.verifyRedeemFrom (Controlling\
    ↪ PEPListVerifier.sol#101) is not in mixedCase
270 Parameter '_tokenHolder' of PEPListVerifier.verifyRedeemFrom (Controlling\
    ↪ PEPListVerifier.sol#101) is not in mixedCase
271 Parameter '_addr' of PEPListVerifier.addAddressToBlacklist (Controlling\
    ↪ PEPListVerifier.sol#118) is not in mixedCase
272 Parameter '_addr' of PEPListVerifier.removeAddressFromBlacklist (Controlling
    ↪ \PEPListVerifier.sol#127) is not in mixedCase
273 Function 'PEPListVerifier._onBlacklist' (Controlling\PEPListVerifier.sol
    ↪ #136-138) is not in mixedCase
274 Parameter '_addr' of PEPListVerifier._onBlacklist (Controlling\
    ↪ PEPListVerifier.sol#136) is not in mixedCase
275 Function 'UnstructuredProxy._setProxyOwner' (Proxy\UnstructuredProxy.sol
    ↪ #102-105) is not in mixedCase
276 Constant 'UnstructuredProxy.ownerPosition' (Proxy\UnstructuredProxy.sol#4)
    ↪ is not in UPPER_CASE_WITH_UNDERSCORES
277 Constant 'UnstructuredProxy.implementationPosition' (Proxy\UnstructuredProxy
    ↪ .sol#5) is not in UPPER_CASE_WITH_UNDERSCORES
278 Parameter '_owner' of VotingToken.balanceOf (Tokens\VotingToken.sol#111) is
    ↪ not in mixedCase
279 Function 'VotingToken._transfer' (Tokens\VotingToken.sol#84-104) is not in
    ↪ mixedCase
280 Parameter '_from' of VotingToken._transfer (Tokens\VotingToken.sol#84) is
    ↪ not in mixedCase
281 Parameter '_to' of VotingToken._transfer (Tokens\VotingToken.sol#84) is not
    ↪ in mixedCase
282 Parameter '_amount' of VotingToken._transfer (Tokens\VotingToken.sol#84) is
    ↪ not in mixedCase
283 Function 'VotingToken._mint' (Tokens\VotingToken.sol#250-257) is not in
    ↪ mixedCase
284 Parameter '_account' of VotingToken._mint (Tokens\VotingToken.sol#250) is
    ↪ not in mixedCase
285 Parameter '_value' of VotingToken._mint (Tokens\VotingToken.sol#250) is not
    ↪ in mixedCase
286 Function 'VotingToken._burn' (Tokens\VotingToken.sol#265-277) is not in
    ↪ mixedCase
287 Parameter '_account' of VotingToken._burn (Tokens\VotingToken.sol#265) is
    ↪ not in mixedCase
288 Parameter '_value' of VotingToken._burn (Tokens\VotingToken.sol#265) is not
    ↪ in mixedCase
289 Parameter '_owner' of VotingToken.balanceOfAt (Tokens\VotingToken.sol#228)
    ↪ is not in mixedCase
290 Parameter '_blockNumber' of VotingToken.balanceOfAt (Tokens\VotingToken.sol
    ↪ #228) is not in mixedCase
291 Parameter '_blockNumber' of VotingToken.totalSupplyAt (Tokens\VotingToken.
    ↪ sol#239) is not in mixedCase
292 Parameter '_block' of VotingToken.getValueAt (Tokens\VotingToken.sol#291) is
    ↪  not in mixedCase
293 Parameter '_value' of VotingToken.updateValueAtNow (Tokens\VotingToken.sol
    ↪ #318) is not in mixedCase
294 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#
    ↪ conformance-to-solidity-naming-conventions
295 INFO:Detectors:
296 MockContract.fallback (Mocks\MockContract.sol#327-371) uses literals with
    ↪ too many digits:
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
297         - (r) = address(this).call.gas(100000)(abi.encodeWithSignature(
            ↪ updateInvocationCount(bytes4,bytes),methodId,msg.data))
298 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too
    ↪ -many-digits
299 INFO:Slither:. analyzed (32 contracts), 253 result(s) found
```

Listing A.7: Output of slither code analysis after removing critical Errors. The remaining errors are false positives or unavoidable when using proxies.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub