

---

# IP6: Blockchain Transactionmanager

Bachelorthesis

Faustina Bruno, Jurij Maïkoff

**Studiengang:**

- iCompetence
- Informatik

**Betreuer:**

- Markus Knecht
- Daniel Kröni

**Experte:**

- Konrad Durrer

**Auftraggeber:**

Fachhochschule Nordwestschweiz  
FHNW Campus Brugg-Windisch  
Bahnhofstrasse 6  
5210 Windisch



2019-10-01

## Abstract

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Problemstellung . . . . .	1
1.2	Ziel . . . . .	1
1.3	Methodik . . . . .	1
1.4	Strukturierung des Berichts . . . . .	2
<b>2</b>	<b>Theoretische Grundlagen</b>	<b>3</b>
2.1	Anwendungsbereich . . . . .	3
2.2	Komponenten . . . . .	3
2.2.1	Ethereum Blockchain . . . . .	4
2.2.2	Smart Contracts . . . . .	4
2.2.3	Transaktionen . . . . .	5
2.2.4	Gas . . . . .	6
2.2.5	Account . . . . .	7
2.2.6	Blockchain Wallet . . . . .	8
2.2.7	Denial of Service (DoS) Attacken . . . . .	9
2.3	Ethereum Client . . . . .	10
2.3.1	Parity . . . . .	10
2.3.2	Geprüfte Alternativen . . . . .	15
2.4	Lösungsansätze . . . . .	15
2.4.1	Architektur . . . . .	15
2.4.2	Evaluation der Architektur . . . . .	21
2.4.3	DoS-Algorithmus . . . . .	24
2.4.4	Evaluation DoS-Algorithmus . . . . .	26
2.4.5	Konfiguration des Algorithmus . . . . .	29
<b>3</b>	<b>Praktischer Teil</b>	<b>31</b>
3.1	Parity . . . . .	31
3.1.1	Konfiguration der Blockchain . . . . .	31
3.1.2	Docker . . . . .	35
3.1.3	Name Registry . . . . .	35

3.1.4	Certifier . . . . .	35
3.2	Externes Programm . . . . .	37
3.2.1	Wrapperklassen . . . . .	38
3.2.2	Überwachung von Transaktionen . . . . .	38
3.2.3	DoS Algorithmus . . . . .	38
3.2.4	Initialisierung . . . . .	38
3.2.5	Konfiguration . . . . .	38
<b>4</b>	<b>Fazit</b>	<b>40</b>
<b>5</b>	<b>Quellenverzeichnis</b>	<b>41</b>
<b>6</b>	<b>Anhang</b>	<b>45</b>
6.1	Glossar . . . . .	45
6.2	Entwicklungsumgebung . . . . .	45
6.2.1	Blockchain . . . . .	46
6.2.2	Wallet . . . . .	46
6.2.3	Smart Contracts . . . . .	46
6.2.4	Docker . . . . .	47
6.3	Weitere Lösungsansätze . . . . .	47
6.3.1	Super Smart Wallet . . . . .	47
6.4	Abnahmekriterien . . . . .	49
6.5	Abnahme Tests Report . . . . .	53
6.5.1	Abnahme Test 1 . . . . .	53
6.5.2	Abnahme Test 2 . . . . .	53
6.5.3	Abnahme Test 3 . . . . .	53
6.5.4	Abnahme Test 4 . . . . .	53
6.5.5	Abnahme Test 5 . . . . .	53
6.5.6	Abnahme Test 6 . . . . .	53
6.5.7	Abnahme Test 7 . . . . .	53
6.5.8	Abnahme Test 8 . . . . .	53
6.5.9	Abnahme Test 9 . . . . .	53
6.6	Registry . . . . .	53
6.6.1	ABI . . . . .	53
6.6.2	Owned.sol . . . . .	54
6.6.3	Registry.sol . . . . .	55
6.6.4	SimpleRegistry.sol . . . . .	56
6.6.5	Java-Wrapper für SimpleRegistry . . . . .	63

6.7	Certifier . . . . .	83
6.7.1	Certifier.sol . . . . .	83
6.7.2	Owned.sol . . . . .	84
6.7.3	SimpleCertifier.sol . . . . .	85
6.7.4	Java-Wrapper für SimpleCertifier . . . . .	86
<b>7</b>	<b>Ehrlichkeitserklärung</b>	<b>95</b>

# 1 Einleitung

Dieses Kapitel liefert eine ausführliche Zusammenfassung der Bachelorthesis.

## 1.1 Problemstellung

Die Aufgabe beinhaltet ein Blockchain Netzwerk [1] für die Fachhochschule Nordwest Schweiz[2] (FHNW) zur Verfügung zu stellen, welches von den Studierenden zu Testzwecken genutzt werden kann. Blockchains verfügen über verschiedene Mechanismen, um sich gegen Attacks abzusichern. Eine davon ist eine Gebühr auf jeder Transaktion, der sogenannte Gas Price 2.2.4 [3]. Dadurch können Denial of Service (DoS) Attacks 2.2.7 [4], bei denen das Netzwerk mit unzähligen Transaktionen geflutet wird, effizient bekämpft werden. Der Angreifer kann die Attacke nicht aufrecht erhalten, da ihm die finanziellen Mittel zwangsläufig ausgehen. Obwohl dieser Schutzmechanismus auf einer öffentlichen Blockchain sehr effizient und elegant ist, eignet er sich nicht für eine Lernumgebung. Hier sollen Anwender die Möglichkeit haben, Transaktionen ohne anfallende Gebühren ausführen zu können. Dadurch wird jedoch die Blockchain anfällig für DoS Attacks.

## 1.2 Ziel

Das Ziel der Arbeit ist es ein Test Blockchain Netzwerk aufzubauen, welches für eine definierte Gruppe von Benutzern gratis Transaktionen erlaubt und trotzdem ein Schutzmechanismus gegen DoS Attacks hat.

## 1.3 Methodik

//TODO Kapitel besprechen und beschreiben Hier wird beschrieben wie und was gemacht wurde

!!muss besprochen überarbeitet werden Wir haben zu Beginn Meilensteine und grössere Arbeitspakete definiert. Die kleineren Arbeitspakete wurden nach neugewonnen Wissen und Arbeitsstand definiert.

Durch die erarbeiteten Lösungsansätze, der Evaluation und die Besprechung nach der Zwischenpräsentation, wurden die Meilensteine geändert und die Planung anders gestaltet.

Agiles Vorgehen, -> mit neuem Wissen weiter geplant

//TODO möglicher Text besprechen und überarbeiten Zu Beginn wurde ein provisorischer Projekt Plan mit möglichen Arbeitspaketen und Meilensteine definiert. Da die Thematik komplett unbekannt war, wurde auf ein agiles Vorgehen gesetzt, um neue Erkenntnisse in die Planung einfließen zu lassen. Nach der Einlese- und Probierphase, wurden Lösungskonzepte konzipiert, evaluiert und an der Zwischenpräsentation dem Experten und den Betreuern präsentiert. Hier wurde das weitere Vorgehen besprochen und die neuen Meilensteine definiert. Die Arbeitspakete werden alle zwei Wochen definiert.

## 1.4 Strukturierung des Berichts

Der Bericht ist in einen theoretischen und praktischen Teil gegliedert. Gemachte Literaturstudien, geprüfte Tools, der aktuelle Stand der Ethereum Blockchain, sowie die konzipierten Lösungsansätze und deren Evaluation werden im theoretischen Teil behandelt. Im praktischen Teil wird beschrieben, wie das gewonnene Wissen umgesetzt wird. Es wird auf die implementierte Lösung und deren Vor- und Nachteile eingegangen. Geprüfte Alternativen und deren Argumente sind ebenfalls enthalten. Das Fazit bildet den Abschluss des eigentlichen Berichts. Im Anhang ist eine Beschreibung der Entwicklungsumgebung, die Installationsanleitung und verwendeter Code zu finden.

## 2 Theoretische Grundlagen

Dieses Kapitel befasst sich nebst dem Kontext der Arbeit, mit den gemachten Literaturrecherchen, welche für die Erarbeitung der Lösungsansätze nötig sind. Weiter wird der Anwendungsbereich der Lösung behandelt.

### 2.1 Anwendungsbereich

Die FHNW möchte zu Ausbildungszwecken eine eigene Ethereum Blockchain betreiben. Die Blockchain soll die selbe Funktionalität wie die öffentliche Ethereum Blockchain vorweisen. Sie soll den Studenten die Möglichkeit bieten, in einer sicheren Umgebung Erfahrungen zu sammeln und Wissen zu gewinnen. Obwohl eine öffentliche Blockchain für jedermann frei zugänglich ist, sind fast alle Aktionen mit Kosten verbunden. Die Kosten sind ein fixer Bestandteil einer Blockchain. So fallen zum Beispiel bei jeder Transaktionen Gebühren an. Diese ermöglichen nicht nur deren Verarbeitung, sondern garantieren auch Schutz vor Attacken.

Im Gegensatz zu einer öffentlichen Blockchain, sind Transaktionsgebühren in einer Lernumgebung nicht praktikabel. Die Studenten sollen gratis mit der Blockchain agieren können, ohne dass der Betrieb oder die Sicherheit der Blockchain kompromittiert werden.

Die FHNW bietet die kostenlose Verarbeitung von Transaktionen zu Verfügung. Damit sichert sie den Betrieb der Blockchain. Die Implementation von gratis Transaktionen und einem Schutzmechanismus wird in diesem Bericht behandelt.

### 2.2 Komponenten

//TODO Spellcheck

Die folgenden Abschnitte behandeln die gemachten Literaturrecherchen. Für jedes Thema sind die gewonnen Erkenntnisse aufgeführt. Dabei ist nebst einem grundsätzlichen Verständnis für die Materie immer der Schutz vor einer Denial of Service (DoS) Attacke im Fokus.



### 2.2.1 Ethereum Blockchain

Eine Blockchain ist eine kontinuierlich erweiterbare Liste von Datensätzen, „Blöcke“ genannt, die mittels kryptographischer Verfahren miteinander verkettet sind. Jeder Block enthält dabei typischerweise einen kryptographisch sicheren Hash (Streuwert) des vorhergehenden Blocks, einen Zeitstempel und Transaktionsdaten[1].

Ein speziell erwähnenswerter Block, ist der sogenannte Genesisblock[5]. Dieser ist der erste Block in einer Blockchain. Der Genesisblock ist eine JSON Datei mit allen nötigen Parametern und Einstellungen um eine Blockchain zu starten.

Blockchains sind auf einem peer-to-peer (P2P) Netzwerk[6] aufgebaut. Ein Computer der Teil von diesem Netzwerk ist, wird Node genannt. Jeder Node hat eine identische Kopie der Historie aller Transaktionen. Es gibt keinen zentralen Server der angegriffen werden kann. Das erhöht die Sicherheit der Blockchain.

Es muss davon ausgegangen werden, dass es Nodes gibt, die versuchen die Daten der Blockchain zu verfälschen. Dem wird mit der Verwendung von diversen Consensus Algorithmen[7] entgegengewirkt. Die Consensus Algorithmen stellen sicher, dass die Transaktionen auf der Blockchain valide und authentisch sind.

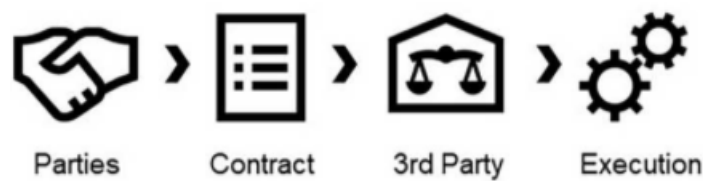
Im Gegensatz zur Bitcoin[8] kann bei Ethereum[9] auch Code in der Chain gespeichert werden, sogenannte Smart Contracts, siehe 2.2.2.

Ethereum verfügt über eine eigene Kryptowährung, den Ether (ETH).

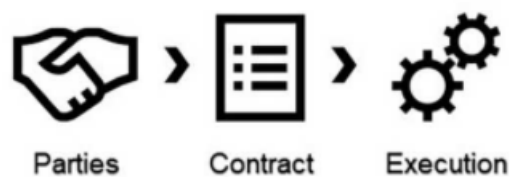
### 2.2.2 Smart Contracts

Der Begriff Smart Contract, wurde von Nick Szabo[10] in den frühen 1990 Jahren zum erten Mal verwendet. Es handelt sich um ein Stück Code, das auf der Blockchain liegt. Es können Vertragsbedingungen als Code geschrieben werden. Sobald die Bedingungen erfüllt sind, führt sich der Smart Contract selbst aus.

Der Code kann von allen Teilnehmern der Blockchain inspiziert werden. Da er dezentral auf der Blockchain gespeichert ist, kann er auch nicht nachträglich manipuliert werden. Das schafft Sicherheit für die beteiligten Parteien.



**Abbildung 2.1:** Ein traditioneller Vertrag[11]



**Abbildung 2.2:** Ein Smart Contract[11]

Der grosse Vorteil von Smart Contracts ist, dass keine third parties benötigt werden, das ist auf den Bildern 2.1 und 2.2 dargestellt. Der Code kontrolliert die Transaktionen, welche Nachverfolgbar und irreversibel sind. Bei einem traditionellen Vertrag werden diese durch third parties kontrolliert und meistens auch ausgeführt.

Sobald ein Smart Contract auf Ethereum deployed ist, verfügt er über eine Adresse, siehe Abschnitt 2.2.5.1. Mit dieser, kann auf die Funktionen des Smart Contracts zugegriffen werden.

### 2.2.2.1 Decentralized application (DApp)

Eine DApp ist eine Applikation (App), deren backend Code dezentral auf einem peer-to-peer Netzwerk läuft, zum Beispiel die Ethereum Blockchain. Der frontend Code kann in einer beliebigen Sprache geschrieben werden, sofern Aufrufe an das Backend möglich sind.

DApp's für die Ethereum Blockchain werden mit Smart Contracts realisiert. Das prominenteste Beispiel einer DApp ist CryptoKitties[12]. Die Benutzer können mit digitale Katzen handeln und züchten.

### 2.2.3 Transaktionen

Um mit der Blockchain zu interagieren, werden Transaktionen benötigt. Sie erlauben es Daten in der Blockchain zu erstellen oder anzupassen. Eine Transaktion verfügt über folgende Felder:

**From** Der Sender der Transaktion. Wird mit einer 20 Byte langen Adresse, siehe Abschnitt 2.2.5.1, dargestellt.

**To** Der Empfänger der Transaktion. Wird ebenfalls mit einer 20 Byte langen Adresse dargestellt. Falls es sich um ein Deployment von einem Smart Contract handelt, wird dieses Feld leer gelassen.

**Value** Mit diesem Feld wird angegeben, wieviel Wei[13] übertragen werden soll. Der Betrag wird von „From“ nach „To“ übertragen.

**Data/Input** Dieses Feld wird hauptsächlich für die Interaktion mit Smart Contracts, siehe Abschnitt 2.2.2, verwendet. Wenn ein Smart Contract deployed werden soll, wird in diesem Feld der dessen Bytecode[14] übertragen. Bei Funktionsaufrufen auf einen Smart Contract wird die Funktionssignatur und die codierten Parameter mitgegeben. Bei reinen Kontoübertragungen wird das Feld leer gelassen.

**Gas Price** Gibt an, welcher Preis pro Einheit Gas man gewillt ist zu zahlen. Mehr dazu im Abschnitt 2.2.4

**Gas Limit** Definiert die maximale Anzahl Gas Einheiten, die für diese Transaktion verwendet werden können, siehe Abschnitt 2.2.4 [15]

Damit eine Transaktion in die Blockchain aufgenommen werden kann, muss sie signiert[16] sein. Dies kann beim Benutzer offline gemacht werden. Die signierte Transaktion wird dann an die Blockchain übermittelt.

Die Übermittlung der Transaktionen wird mittels Remote procedure call(RPC)[17] gemacht.

## 2.2.4 Gas

Mit Gas[3] ist in der Ethereum Blockchain eine spezielle Währung gemeint. Mit ihr werden Transaktionskosten gezahlt. Jede Aktion in der Blockchain kostet eine bestimmte Menge an Gas (Gas Cost). Somit ist die benötigte Menge an Gas proportional zur benötigten Rechenleistung. So wird sichergestellt, dass die anfallenden Kosten einer Interaktion gerecht verrechnet werden. Die anfallenden Gas Kosten werden in Ether gezahlt. Für die Berechnung der Transaktionskosten wird der Preis pro Einheit Gas (Gas Price) verwendet. Dieser kann vom Sender selbst bestimmt werden. Ein zu tief gewählter Gas Price hat zur Folge, dass die Transaktion nicht in die Blockchain aufgenommen wird, da es sich für einen Miner, siehe Abschnitt ??, nicht lohnt, diese zu verarbeiten. Ein hoher Gas Price stellt zwar sicher, dass die Transaktion schnell verarbeitet wird, kann aber hohe Gebühren generieren.

$$TX = gasCost * gasPrice$$

Die Transaktionskosten werden nicht direkt in Ether berechnet, da dieser starken Kursschwankungen unterworfen sein kann. Die Kosten für Rechenleistung, also Elektrizität, sind hingegen stabiler Natur. Daher sind Gas und Ether separiert.

Ein weiterer Parameter ist Gas Limit. Mit diesem Parameter wird bestimmt, was die maximale Gas Cost ist, die man für eine Transaktion bereitstellen möchte. Es wird aber nur so viel verrechnet, wie auch

wirklich benötigt wird, der Rest wird einem wieder gutgeschrieben. Falls die Transaktionskosten höher als das gesetzte Gas Limit ausfallen, wird die Ausführung der Transaktion abgebrochen. Alle gemachten Änderungen auf der Chain werden rückgängig gemacht. Die Transaktion wird als „fehlgeschlagene Transaktion“ in die Blockchain aufgenommen. Das Gas wird nicht zurückerstattet, da die Miner bereits Rechenleistung erbracht haben.

## 2.2.5 Account

Um mit Ethereum interagieren zu können, wird ein Account benötigt. Es gibt zwei Arten von Accounts, solche von Benutzern und jene von Smart Contracts. Ein Account ermöglicht es einem Benutzer oder Smart Contract, Transaktionen zu empfangen und zu senden.

### 2.2.5.1 Benutzer Account

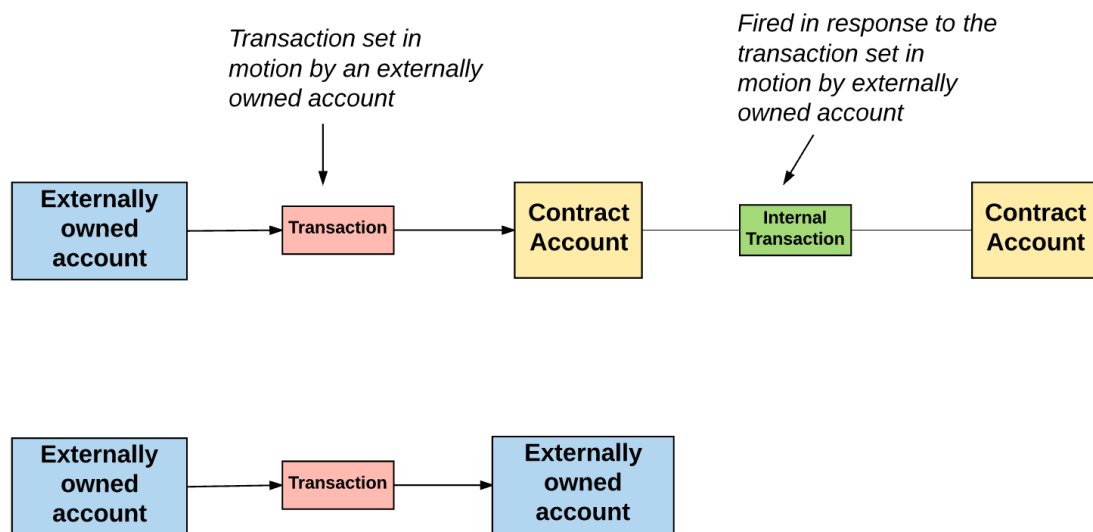
Der Account eines Benutzers besteht aus Adresse, öffentlichen und geheimen Schlüssel. Diese Art von Accounts haben keine Assoziation mit Code. Sie werden von Benutzer verwendet um mit der Blockchain zu interagieren.

**Geheimer Schlüssel** Der geheime Schlüssel ist ein 256 Bit lange zufällig generierte Zahl. Er definiert einen Account und wird verwendet um Transaktionen zu signieren. Daher ist es von grösster Wichtigkeit, dass ein geheimer Schlüssel sicher gelagert wird. Wenn er verloren geht, gibt es keine Möglichkeit mehr auf diesen Account zuzugreifen.

**Öffentlicher Schlüssel** Der öffentliche Schlüssel wird aus dem geheimen Schlüssel abgeleitet. Für die Generierung wird Keccak[18] verwendet, ein „Elliptical Curve Digital Signature Algorithm“[19]. Der öffentliche Schlüssel wird verwendet um die Signatur einer Transaktion zu verifizieren.

**Adresse** Die Adresse wird aus dem öffentlichen Schlüssel abgeleitet. Es wird SHA3[20] verwendet um einen 32 Byte langen String zu bilden. Von diesem bilden die letzten 20 Bytes, also 40 Zeichen, die Adresse von einem Account. Die Adresse wird bei Transaktionen oder Interaktionen mit einem Smart Contract verwendet.

**Contract Accounts** Contract Accounts sind durch ihren Code definiert. Sie können keine Transaktionen initiieren, sondern reagieren nur auf zuvor eingegangene. Das wird auf der Abbildung 2.3 dargestellt. Ein Benutzer Accounts wird als „Externally owned account“ bezeichnet.



**Abbildung 2.3:** TX zwischen Accounts

Im Gegensatz zu einem Benutzer Account hat ein Contract Account keine Verwendung für einen geheimen oder öffentlichen Schlüssel. Es wird nur eine Adresse benötigt. Analog zu einem Benutzer Account, wird diese benötigt, um Transaktionen an diesen Smart Contract zu senden.

Sobald ein Smart Contract deployed wird, wird eine Adresse generiert. Verwendet wird die Adresse und Anzahl getätigte Transaktionen (nonce[21]) des Benutzer Accounts, der das Deployment vornimmt.[22]

### 2.2.6 Blockchain Wallet

Eine Blockchain Wallet, kurz Wallet, ist ein digitales Portmonaie. Der Benutzer hinterlegt in der Wallet seinen geheimen Schlüssel, siehe 2.2.5.1. Dadurch erhält er eine grafische Oberfläche für die Verwaltung seines Accounts. Nebst dem aktuellen Kontostand, wird meistens noch die Transaktionshistorie angezeigt.

In der Wallet können mehrere Accounts verwaltet werden. So muss sich der Benutzer nicht selbst um die sichere Aufbewahrung der geheimen Schlüssel kümmern. Bei den meisten Wallets ist es möglich verschiedene Währungen zu verwalten.

Es existieren zwei unterschiedliche Arten von Wallets, Hot und Cold Wallets:

**Hot Wallet** Ein Stück Software, welches die geheimen Schlüssel verwaltet.

Es existieren drei unterschiedliche Typen, Destkop, Web und Mobile Wallets. [23], [24], [25]

**Cold Wallet** Der geheime Schlüssel wird in einem Stück Hardware gespeichert. Dadurch können die

geheimen Schlüssel offline gelagert werden. Das erhöht die Sicherheit der Wallet, da Angriffe aus dem Internet ausgeschlossen werden können. [23], [24], [25]

### 2.2.6.1 Smart Wallet

Smart Wallets basieren auf Smart Contracts. Der Benutzer ist der Besitzer der Smart Contracts und somit der Wallet. Die Verwendung von Smart Contract bei der Implementierung der Wallet ermöglicht mehr Benutzerfreundlichkeit ohne die Sicherheit zu kompromittieren. [26], [27], [28] //TODO ..

### 2.2.7 Denial of Service (DoS) Attacken

//TODO ergänzen

Bei einer DoS Attacke versucht der Angreifer einen Service mit Anfragen zu überlasten. Die Überlastung schränkt die Verfügbarkeit stark ein oder macht den Service sogar gänzlich un verfügbar für legitime Anfragen.

Zurzeit sind Blockchains noch relativ langsam bei der Verarbeitung von Transaktionen. Ethereum kann ungefähr 15 Transaktionen pro Sekunde abarbeiten.[29] Dadurch ist ein möglicher Angriffsvektor, die Blockchain mit einer sehr hohen Zahl Transaktionen zu fluten.

Ein anderer Angriffsvektor, sind Transaktionen mit einem sehr hohen Bedarf an Rechenleistung. Hier wird Code auf der Blockchain aufgerufen, dessen Verarbeitung sehr lange dauert. Beide Vorgehen haben zur Folge, dass Benutzer sehr lange auf die Ausführung ihrer Transaktionen warten müssen. Blockchains schützen sich vor diesem Angriff mit einer Transaktionsgebühr. Diese werden durch Angebot und Nachfrage bestimmt. Das heisst, wenn es viele Transaktionen gibt, steigt der Bedarf an deren Verarbeitung und es kann davon ausgegangen werden, dass auch die Transaktionsgebühren steigen. Das bedeutet, dass bei einer DoS Attacke die Transaktionsgebühren tendenziell steigen. Um sicherzustellen, dass seine Transaktionen weiterhin zuverlässig in die Blockchain aufgenommen werden, muss der Angreifer seinen Gas Price kontinuierlich erhöhen.

Ein DoS Angriff auf eine Blockchain wird dadurch zu einem sehr kostspieligen Unterfangen. Die hohen Kosten schrecken die meisten Angreifer ab und sind somit ein sehr effizienter Schutzmechanismus.[30]

#### 2.2.7.1 DoS Attacke an der FHNW

Auf der Blockchain der FHNW existiert eine privilegierte Benutzergruppe. Diese dürfen gratis Transaktionen ausführen. Diese Gruppe von Benutzer ist eine potentielle Bedrohung. Ohne Transaktionskosten hat die Blockchain keinen Schutzmechanismus gegen eine DoS Attacke.

Aus diesem Grund muss das Verhalten der privilegierten Accounts überwacht werden. Falls einer dieser Accounts eine DoS Attacke einleitet, muss das frühst möglich erkannt und unterbunden werden können.

## 2.3 Ethereum Client

Für die Betreuung von einem Ethereum Node ist ein Client nötig. Dieser muss das Ethereum Protokoll[31] implementieren. Das Protokoll definiert die minimal Anforderungen an den Clienten. Das erlaubt, dass der Client in verschiedenen Sprachen, von verschiedenen Teams, realisiert werden kann. Nebst der verwendeten Programmiersprache, unterscheiden sich die Clienten bei implementierten Zusatzfunktionen, die im Protokoll nicht spezifiziert sind. Die populärsten Clients sind Go Ethereum (GETH)[32], Parity[33], Aleth[34] und Trinity[35]. Die Clients wurden auf die Zusatzfunktionalität untersucht, für eine definierte Gruppe von Accounts gratis Transaktionen zu ermöglichen.

### 2.3.1 Parity

Geschrieben in Rust[36], ist es der zweit populärste Client nach Geth[32]. Verfügbar ist Parity für Windows, macOS und Linux. Die Entwicklung ist noch nicht abgeschlossen und es wird regelmässig eine neue Version vorgestellt.

Konfiguriert wird das Programm mittels Konfigurationsdateien. Interaktion zur Laufzeit ist über die Kommandozeile möglich.

Parity ist der einzige Client, der es erlaubt, einer definierten Gruppe von Benutzern gratis Transaktionen zu erlauben.

Die Verwaltung der privilegierten Accounts geschieht mittels eines Smart Contracts. Die Accounts sind in einer Liste, der sogenannten Whitelist, gespeichert.

Für die Verwendung der Whitelist sind zwei Smart Contracts nötig, die Name Registry[37] und der Service Transaction Checker[38]. Diese sind in den folgenden Abschnitten erklärt.

#### 2.3.1.1 Name Registry

In Parity wird die Name Registry verwendet, um eine Accountadresse in eine lesbare Form zu übersetzen. Smart Contracts können für eine Gebühr von einem Ether registriert werden. Dabei wird die Adresse des Smart Contracts zusammen mit dem gewählten Namen registriert. Das erlaubt das Referenzieren von Smart Contracts, ohne dass hart kodierte Adressen verwendet werden müssen. Dieses System ist analog zu einem DNS Lookup[39].

Die Name Registry ist in Parity standardmässig immer unter der selben Adresse zu finden. Um eine Whitelist verwenden zu können, muss der zuständige Smart Contract, siehe 2.3.1.2, bei der Name Registry registriert werden.

Nachfolgenden sind die involvierten Methoden und Modifier[40] der Name Registry aufgeführt und erklärt. Der vollständige Code ist im Anhang unter 6.6 zu finden.

```
1 struct Entry {
2     address owner;
3     address reverse;
4     bool deleted;
5     mapping (string => bytes32) data;
6 }
7
8 mapping (bytes32 => Entry) entries;
```

In der Map `entries` sind alle registrierten Accounts festgehalten. Pro Eintrag wird der Besitzer (`owner`), die Adresse (`address`), ein Flag ob der Eintrag gelöscht ist (`deleted`) und dessen Daten (`data`) gespeichert.

Die Map `entries` ist die zentrale Datenstruktur der Name Registry. Änderungen daran sind daher durch Modifiers eingeschränkt.

```
1 modifier whenUnreserved(bytes32 _name) {
2     require(!entries[_name].deleted && entries[_name].owner == 0);
3     _;
4 }
```

Stellt sicher, dass ein Eintrag zu einem Namen (`_name`) nicht bereits existiert oder zu einem früheren Zeitpunkt gelöscht worden ist. Es wird also geprüft, ob die gewünschte Position in der Map `entries` noch frei ist und somit reserviert werden kann.

```
1 modifier onlyOwnerOf(bytes32 _name) {
2     require(entries[_name].owner == msg.sender);
3     _;
4 }
```

Der Besitzer einer Nachricht wird mit dem Besitzer eines Eintrags unter dem Namen `_name` in `entries` verglichen. Nur wenn dieser identisch ist, dürfen Änderungen an einem existierenden Eintrag vorgenommen werden.

```
1 modifier whenEntryRaw(bytes32 _name) {
2     require(
3         !entries[_name].deleted &&
4         entries[_name].owner != address(0)
5     );
6     _;
7 }
```



Prüft ob der Eintrag für Namen `_name` nicht gelöscht ist und über einen gültigen Besitzer verfügt. Mit `!= address(0)` wird der geprüft ob sich um mehr als einen uninitialisierten Account handelt.

```
1      uint public fee = 1 ether;
2
3      modifier whenFeePaid {
4          require(msg.value >= fee);
5          _;
6      }
```

Auf Zeile 1 ist die Höhe der Gebühr (`fee`) definiert. Ab Zeile 3 folgt ein Modifier. Dieser überprüft, ob der Betrag in der Transaktion gross genug ist um die Gebühr von Zeile 1 zu bezahlen.

```
1      function reserve(bytes32 _name)
2          external
3          payable
4          whenUnreserved(_name)
5          whenFeePaid
6          returns (bool success)
7      {
8          entries[_name].owner = msg.sender;
9          emit Reserved(_name, msg.sender);
10         return true;
11     }
```

Mit der Methode `reserve` kann ein Eintrag in der Liste `entries` für den Namen `_name` reserviert werden. Durch die Verwendung von `external` auf Zeile 2, kann die Methode von anderen Accounts aufgerufen werden.

Der Modifier `payable` erlaubt es, Ether an die Methode zu senden. Auf Zeile 4 wird überprüft, ob der Eintrag in `entries` noch frei ist. Schliesslich wird geprüft ob der Transaktion genügend Ether mitgegeben wird um die Gebühr zu begleichen.

Wenn alle Prüfungen erfolgreich sind, wird in `entries` eine neuer Eintrag erstellt. Als Besitzer des Eintrags wird der Sender der Transaktion gesetzt. Auf Zeile 9 wird die erfolgreiche Reservierung ans Netzwerk gesendet.

```
1      function setAddress(bytes32 _name, string _key, address _value)
2          external
3          whenEntryRaw(_name)
4          onlyOwnerOf(_name)
5          returns (bool success)
6      {
7          entries[_name].data[_key] = bytes32(_value);
8          emit DataChanged(_name, _key, _key);
9          return true;
10     }
```

Mit dieser Methode wird ein reservierter Eintrag in `entries` befüllt. Als erster Parameter wird der

Name des Eintags (`_name`) übergeben. Dieser muss identisch zum verwendeten Namen in der Methode `reserve` sein. Mit dem Parameter `_key` wird der Zugriff auf die innere Map `data` verwaltet. Mit `_value` wird die zu registrierende Adresse übergeben.

Auch diese Methode muss von Aussen aufgerufen werden können, daher `external` auf Zeile 2. Wenn die Bedingungen von `whenEntryRaw` und `onlyOwnerOf` auf Zeile 3 und 4 erfüllt sind, wird die eigentliche Registrierung vorgenommen.

In der Map `data` wird die Adresse (`_value`) an der Position `_key` gespeichert.

Die Änderung der Daten wird auf Zeile 9 ans Netzwerk gesendet.

### 2.3.1.2 Certifier

Als Standard werden alle Transaktionen mit einem Gas Price von 0 verworfen. Das heisst, diese Transaktionen werden bereits beim Node zurückgewiesen und erreichen nie die Blockchain. Dieses Verhalten kann geändert werden. Mit der Registrierung des Certifiers bei der Name Registry. Beim Start von Parity wird geprüft ob der Eintrag in `entries` vorhanden ist. Sofern vorhanden, werden Transaktionen mit einem Gas Price von 0 nicht mehr direkt abgewiesen, sondern es wird geprüft ob der Absender zertifiziert ist. Transaktionen von zertifizierten Accounts werden selbst mit einem Gas Price von 0 in die Blockchain aufgenommen. Gratis Transaktionen von unzertifizierten Benutzern werden weiterhin abgewiesen.

In diesem Abschnitt sind besonders wichtige Abschnitte des SimpleCertifiers aufgeführt und erklärt. Der gesamte Code ist im Anhang unter 6.7 zu finden.

```
1 struct Certification {
2     bool active;
3 }
4
5 mapping (address => Certification) certs;
```

Die zentrale Datenstruktur des Certifiers, die Whitelist. In der Liste `certs` sind zertifizierte Accounts gespeichert.

```
1 address public delegate = msg.sender;
2
3 modifier onlyDelegate {
4     require(msg.sender == delegate);
5     _;
6 }
```

Auf Zeile 1 wird der Besitzer (`msg.sender`) des Smart Contracts gespeichert und der Variabel `delegate` zugewiesen. Mit dem Modifier wird geprüft ob es sich beim Absender der aktuellen Anfrage um den Besitzer des Smart Contracts handelt.

```
1  function certify(address _who)
2      external
3      onlyDelegate
4  {
5      certs[_who].active = true;
6      emit Confirmed(_who);
7  }
```

Mit dieser Methode wird ein Account registriert. Als Parameter wird die zu registrierende Adresse (`_who`) angegeben. Mit `external` auf Zeile 2 ist die Methode von Aussen aufrufbar.

Zeile 3 stellt sicher, dass nur der Besitzer des Certifiers einen Account registrieren kann. Ist diese Prüfung erfolgreich, wird der Account `_who` der Liste `certs` hinzugefügt. Der Account ist nun für gratis Transaktionen berechtigt.

Der Event wird auf Zeile 6 an das Netzwerk gesendet.

```
1  function certified(address _who)
2      external
3      view
4      returns (bool)
5  {
6      return certs[_who].active;
7  }
```

Mit der Methode `certified` kann jederzeit überprüft werden, ob ein Account (`_who`) zertifiziert ist. Mit `view` auf Zeile 3 ist deklariert, dass es sich um eine Abfrage ohne weitere Komputationskosten handelt. Solche Abfragen sind daher mit keinen Transaktionskosten verbunden.

```
1  function revoke(address _who)
2      external
3      onlyDelegate
4      onlyCertified(_who)
5  {
6      certs[_who].active = false;
7      emit Revoked(_who);
8  }
```

Die Methode `revoke` entfernt einen zertifizierten Account (`_who`) von der Whitelist. Auf Zeile 3 wird wiederum sichergestellt, dass nur der Besitzer des Certifiers Änderungen vornehmen kann. Weiter wird auf Zeile 4 verifiziert, dass der Account `_who` in der Whitelist `certs` registriert ist.

Sind alle Bedingungen erfüllt, wird der Account von der Whitelist entfernt. Der Event wird auf Zeile 7 an die Blockchain gesendet.

### 2.3.2 Geprüfte Alternativen

Die Clients Geth, Aleth und Trinity sind ebenfalls evaluiert worden. Bei diesen Clients ist keine Möglichkeit vorhanden, bestimmte Accounts für gratis Transaktionen zu privilegieren. Daher sind sie zu diesem Zeitpunkt nicht für die FHNW geeignet.

## 2.4 Lösungsansätze

//TODO Spellcheck über ganze Seite

//TODO Erläuterungen zu Flow Charts

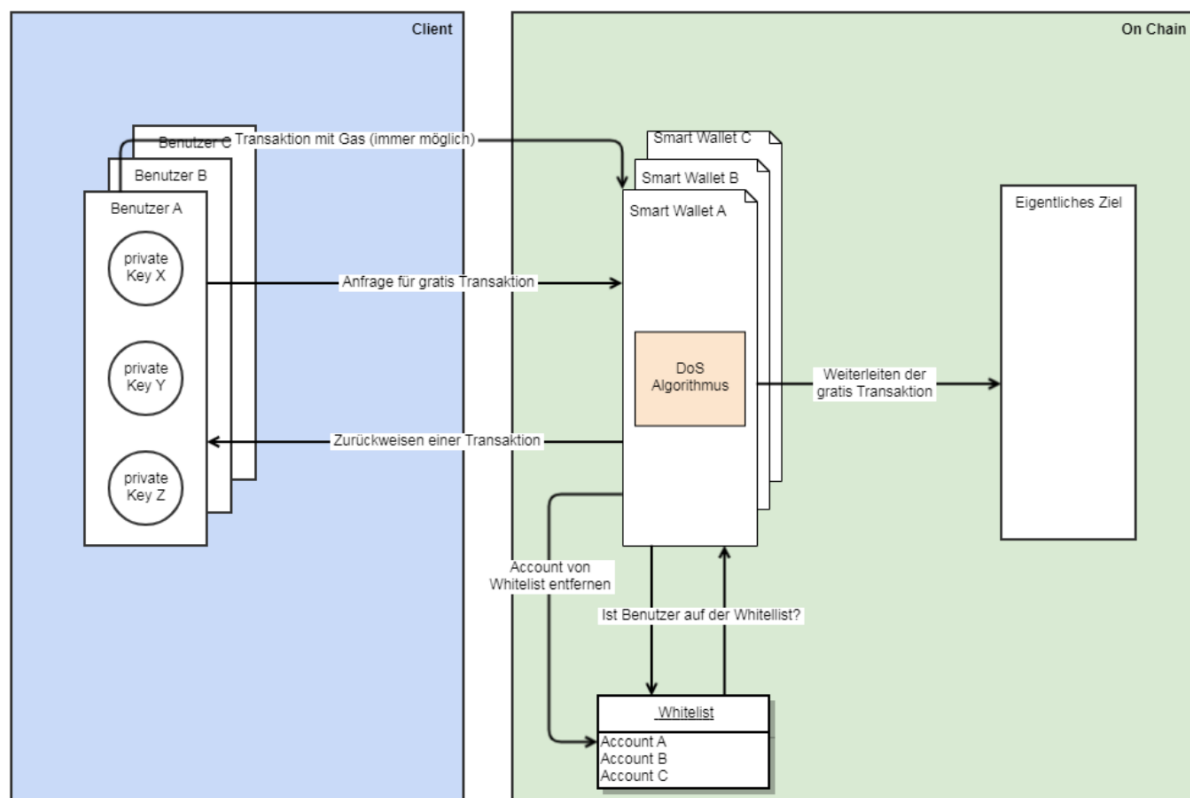
In diesem Kapitel werden die erarbeiteten Lösungsansätze vorgestellt. Die Stärken und Schwächen von jedem Lösungsansatz (LA) werden analysiert und dokumentiert. Mit der vorgenommenen Analyse wird ein Favorit bestimmt. Dieser wird weiterverfolgt und implementiert.

### 2.4.1 Architektur

Die erarbeiteten Architektur-Lösungsansätze (ALA) werden in diesem Abschnitt behandelt.

#### 2.4.1.1 ALA 1: Smart Wallet

Es wird selbst eine Smart Wallet entwickelt. Diese benötigt die volle Funktionalität einer herkömmlichen Wallet. Zusätzlich ist ein Schutzmechanismus gegen DoS Attacken implementiert. Wie in Abbildung ?? ersichtlich, wird für jeden Benutzer eine Smart Wallet deployed. Dies wird von der FHNW übernommen. So fallen für die Benutzer keine Transaktionsgebühren an. Wie unter 2.3.1 beschrieben, wird für die Betreuung der Blockchain der Client Parity mit einer Withelist verwendet.



**Abbildung 2.4:** Architektur mit Smart Wallet

Es muss sichergestellt werden, dass ein Benutzer auf seine Smart Wallet zugreifen kann, unabhängig davon ob er gratis Transaktionen tätigen darf oder nicht. Dies ist in der Abbildung 2.4 dargestellt.

Wie in 2.3.1 beschrieben, prüft Parity bei einer gratis Transaktion nur, ob sich der Account in der Whitelist befindet. Das bedeutet, dass mit einem whitelisted Account auch gratis Transaktionen getätigt werden können, die nicht an die Smart Wallet gerichtet sind. Somit kann der Benutzer den DoS Schutzmechanismus umgehen. Deswegen muss ein Weg gefunden werden, der den Benutzer zwingt Transaktionen über die Smart Wallet abzuwickeln.

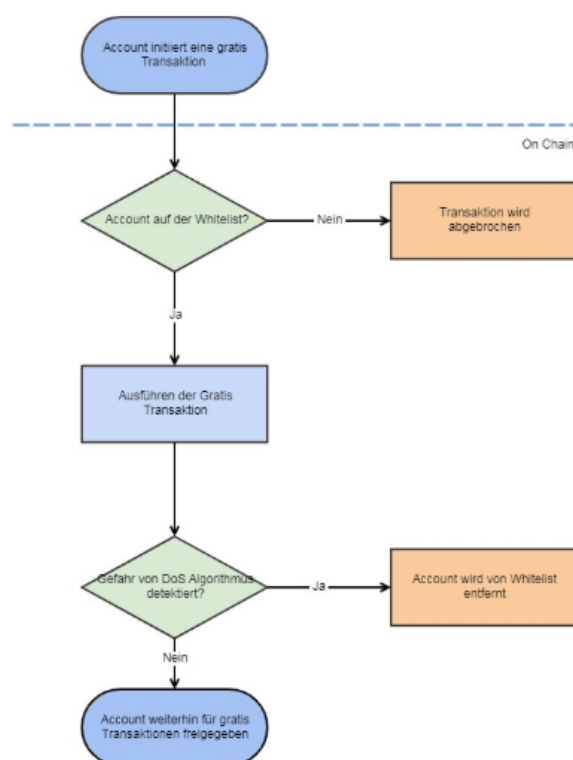
Eine Möglichkeit ist Parity selbst zu erweitern. Anstelle einer Liste mit Accounts, muss eine Liste von Verbindungen geführt werden. So kann definiert werden, dass nur eine Transaktion auf die Smart Wallet gratis ist.

**Pro** Dieser Ansatz besticht durch die Tatsache, dass alles auf der Blockchain läuft. Somit werden grundlegende Prinzipien, wie Dezentralität und Integrität, einer Blockchain bewahrt.

**Contra** Die Machbarkeit des Ansatzes ist unklar. Um diesen Ansatz umzusetzen, muss der Blockchain Client, Parity, erweitert werden. Es ist unklar, wie weitreichend die Anpassungen an Parity sind. Zusätzlich wird eine zusätzliche Programmiersprache, Rust[36], benötigt.

Ein weiterer Nachteil ist, dass bei einer Änderung am DoS Schutzalgorithmus eine neue Smart Wallet für jeden Account deployed werden muss. Das bedingt, dass die Whitelist ebenfalls mit den neuen Accounts aktualisiert wird.

**Prozessworkflow** In der Abbildung 2.5 ist der Prozessablauf für eine gratis Transaktion dargestellt.

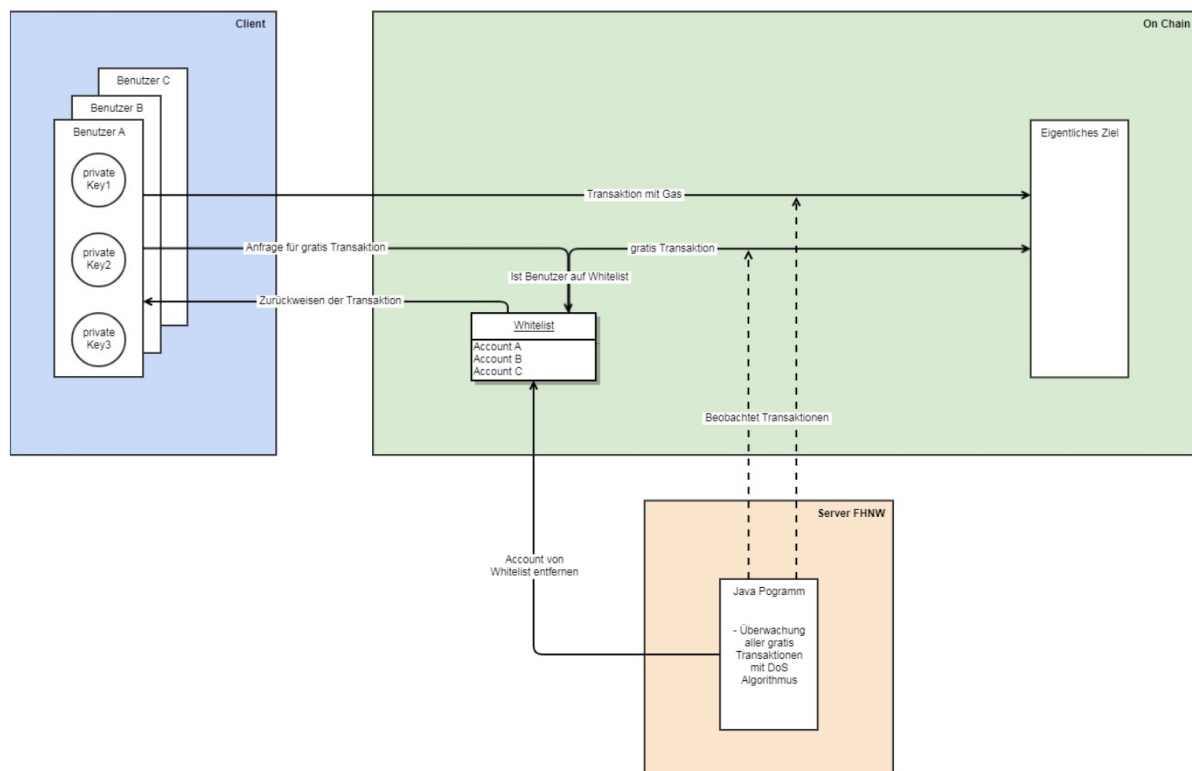


**Abbildung 2.5:** Flowchart für Smart Wallet

//TODO weitere Erläuterung?

#### 2.4.1.2 ALA 2: Externes Programm für die Verwaltung der Whitelist

Bei diesem Ansatz wird auf die Entwicklung einer Smart Wallet verzichtet. Stattdessen wird der Schutzmechanismus gegen DoS Attacken in einem externen Programm implementiert, dargestellt in Abbildung 2.6.



**Abbildung 2.6:** Externes Programm für die Verwaltung der Whitelist

Es wird auch für diesen Ansatz die Whitelist von Parity verwendet, siehe 2.3.1.

Im externen Programm werden alle gratis Transaktionen analysiert, die das Blockchain Netzwerk erreichen. Das Programm verfügt über einen eigenen Benutzer Account, siehe 2.2.5. Dieser ist berechtigt, die Whitelist zu manipulieren. Dadurch kann bei einer identifizierten Attacke, der angreifende Account automatisch von der Whitelist gelöscht werden.

Transaktionen für die ein Transaktionsgebühren gezahlt werden sind immer möglich. Diese werden vom externen Programm auch nicht überwacht. Die anfallenden Gebühren sind Schutz genug.

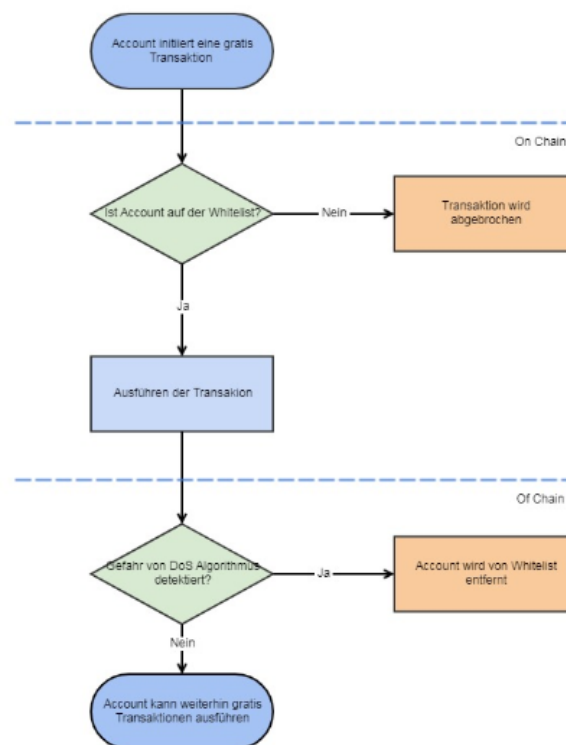
**Pro** Dieser Ansatz ist sicher umsetzbar in der zur Verfügung stehenden Zeit.

Falls eine Anpassung des DoS Schutzalgorithmus nötig ist, muss nur das externe Programm neu deployed werden. Eine aktualisierung der Whitelist ist nicht nötig.

**Contra** Es wird das Hauptprinzip, Dezentralität, einer Blockchain verletzt. Das externe Programm ist eine zentrale Autorität, die von der FHNW kontrolliert wird. Durch das externen Programm kommt eine weitere Komponente dazu. Diese muss ebenfalls administriert werden.

**Prozessworkflow** //TODO Flowchart falsch.. gibt keine Smart wallet, Transaktion kommt immer durch Java wenn auf white list, da java nur passiv mithört

Auf dem Flowchart 2.7 dargestellt ist, kann ein Benutzer mit einem whitelisted Account direkt gratis Transaktionen ausführen.



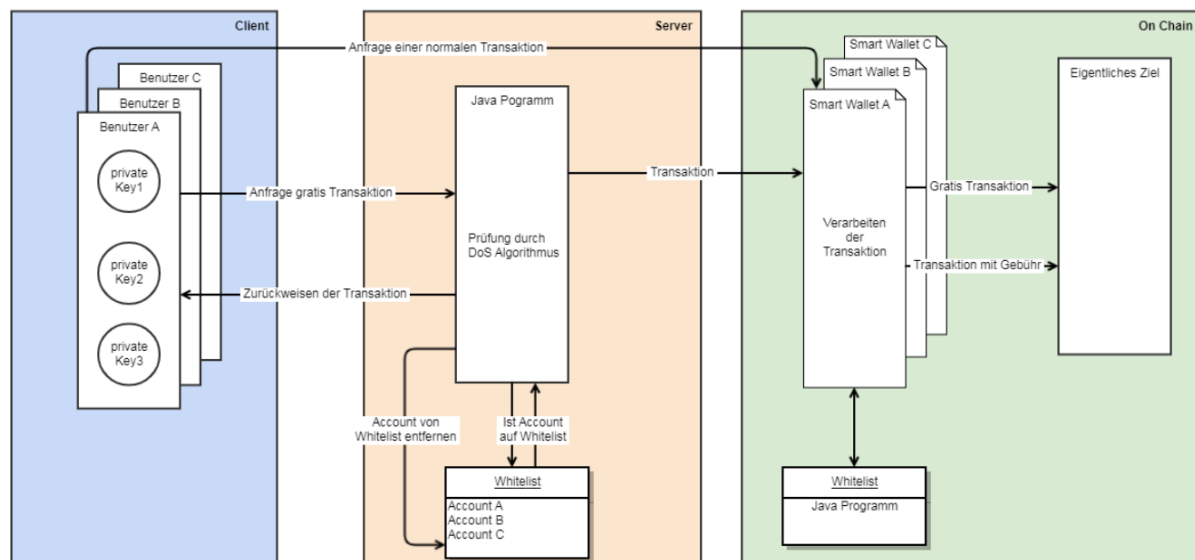
**Abbildung 2.7:** Flowchart externes Programm für die Verwaltung der Whitelist

### 2.4.1.3 ALA 3: Externes Programm mit Whitelist

Wie in Abbildung 2.8 illustriert, ist der Blockchain ein externes Programm vorgelagert. Das Programm verwaltet eine eigene Whitelist mit Accounts. Diese sind für gratis Transaktionen berechtigt. Weiter beinhaltet es den DoS Schutzalgorithmus. Dieser prüft ob der Account auf der Whitelist ist und ob die Transaktion die Schutzrichtlinien verletzt. Falls ein Account die Sicherheitsrichtlinien verletzt, wird dieser vom Algorithmus aus der eigenen Whitelist gelöscht.

Sofern keine Richtlinien verletzt werden, wird die Transaktion ins Data-Feld, siehe 2.2.3, einer neuen Transaktion gepackt. Das ist nötig, um die Transaktionsinformationen (wie z.B. Sender Identität) zu präservieren. Die neue erstellte Transaktion wird vom Programm an die Smart Wallet gesendet.





**Abbildung 2.8:** Externes Programm mit Whitelist

Weiter wird eine Smart Wallet entwickelt. Diese ist nötig, um die verschachtelten Transaktionen des Programms zu verarbeiten. Aus dem Data-Feld wird die eigentliche Transaktion extrahiert und abgesetzt.

Jeder Benutzer besitzt eine eigene Smart Wallet um die Sender Identität für jeden Benutzer einmalig zu halten.

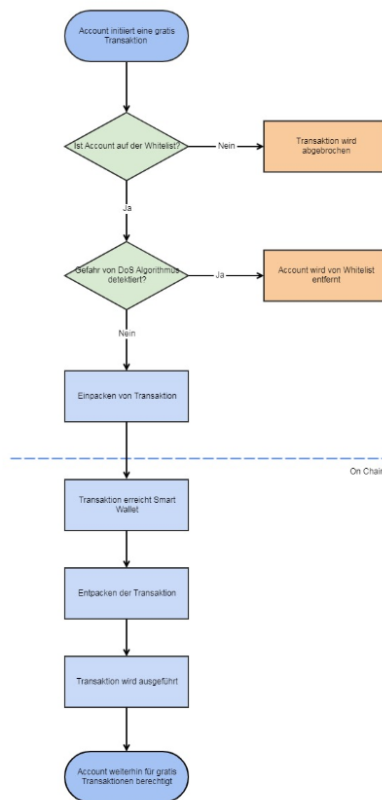
Auf der im Abschnitt 2.3.1 beschriebenen Whitelist ist nur der Account des externen Programmes aufgelistet. So ist sichergestellt, dass nur Transaktionen die vom Programm weitergeleitet werden, kostenfrei durchgeführt werden können. Der Benutzer kann immer mit kostenpflichtigen Transaktionen auf die Smart Wallet zugreifen. Dies ist insbesondere wichtig, falls das Programm nicht aufrufbar ist, wenn z.B. der Server ausfällt.

**Pro** Dieser Ansatz ist in der gegebenen Zeit umsetzbar. Falls eine Anpassung des DoS Schutzalgorithmus nötig ist, muss nur das externe Programm neu deployed werden. Eine aktualisierung der Whitelist ist nicht nötig.

**Contra** Es wird das Hauptprinzip, Dezentralität, einer Blockchain verletzt. Das externe Programm ist eine zentrale Autorität, die von der FHNW kontrolliert wird. Durch das externen Programm kommt eine weitere Komponente dazu. Diese muss ebenfalls administriert werden.

Dieser Ansatz bietet keine Vorteile im Vergleich zum LA 2, ist aber mit der Verschachtelung von Transaktionen komplexer.

**Prozessworkflow** //Todo flowchart falsch, zuerst Java dann richtige smart wallet



**Abbildung 2.9:** Flowchart externes Programm mit Whitelist

Die Abbildung 2.9 zeigt, dass alle gratis Transaktionen in erster Instanz von einem Programm geprüft werden. Falls keine Richtlinien verletzt werden, wird die Transaktion im Data-Feld einer neu generierten Transaktion an die Smart Wallet übermittelt.

## 2.4.2 Evaluation der Architektur

Die erarbeiteten Lösungsansätze werden gegeneinander verglichen. Um zu bestimmen, welcher Ansatz weiter verfolgt wird, wurden folgende Kriterien definiert:

**Machbarkeit (MK)** Bewertet die Machbarkeit des Ansatzes. Das berücksichtigt den gegebenen Zeitrahmen und die Komplexität des Ansatzes.

Da dieses Projekt im gegebenen Zeitrahmen abgeschlossen werden muss, ist es das wichtigste Kriterium. Daher wird es auch mit der höchsten Gewichtung versehen.

Gewichtung 3

**Blockchainprinzipien (BCP)** Gibt an ob die Prinzipien einer Blockchain berücksichtigt werden. Wie Dezentralität, Trust und Security

Die Einhaltung der Prinzipien ist wichtig, aber für die FHNW nicht zwingend. Daher eine mittlere Gewichtung.

Gewichtung 2

**Betrieb (BT)** Bewertet den administrativen Aufwand im Betrieb und die Möglichkeit zur Automatisierung. Das umfasst Deployment Smart Contracts, Anpassungen der Whitelist und Betreuung von zusätzlichen Servern.

Wird mit einer mittleren Gewichtung versehen. Ein zu hoher administrativer Aufwand ist nicht praktikabel.

Gewichtung 2

Jeder ALA wird auf diese drei Kriterien untersucht. Pro Kriterium können zwischen 3 und 1 Punkt erreicht werden, wobei 3 das Maximum ist. Die erreichten Punkte werden mit der entsprechenden Gewichtung multipliziert. Für die Evaluation, werden alle Punkte zusammengezählt. Der Ansatz mit den meisten Punkten wird weiterverfolgt.

**Tabelle 2.1:** Evaluation Lösungsansätze

	MK	BCP	BT	Total
Gewichtung	3	2	2	
ALA 1	1	3	2	13
ALA 2	3	2	2	17
ALA 3	2	1	3	12

#### 2.4.2.1 ALA 1: Smart Wallet

Wir haben diesen Ansatz als sehr komplex eingestuft. Für die Anpassung von Parity muss eine zusätzliche Programmiersprache verwendet werden. Es ist nicht klar, wie weitreichend die nötigen Anpassungen sind. Zusätzlich muss eine Smart Wallet entwickelt werden.

Dieser Ansatz ist komplett dezentral und in die Blockchain integriert. Daher maximale Punktzahl bei Blockchain Prinzipien.

Falls ein Anpassung am DoS Algorithmus nötig ist, muss jede Smart Wallet neu deployed werden. Das bedingt, dass die Whitelist ebenfalls aktualisiert wird. Die Adressen aller bestehenden Smart Wallets müssen ersetzt werden. Alle Studierenden müssen informiert werden, dass sie für ihre Smart Wallet

eine neue Adresse verwenden müssen. Die Automatisierung dieser Prozesse wird als komplex aber machbar eingeschätzt. Daher sind bei Betrieb 2 Punkte gesetzt.

#### **2.4.2.2 ALA 2: Externes Programm für die Verwaltung der Whitelist**

Die Entwicklung eines externen Programmes, welches getätigte Transaktionen der Blockchain prüft, ist in der gegebenen Zeit sicher realisierbar. Daher erhält der ALA für Machbarkeit die volle Punktzahl.

Mit der Verwendung von einem externen Programm, wird eine zentrale Autorität verwendet. Diese ist nicht dezentral und wird von der FHNW administriert. Da das Programm die Transaktionshistorie der Blockchain überwacht und nur bei einer DoS Attacke aktiv ist, wird 2 Punkte für Blockchainprinzipien gegeben.

Falls eine Anpassung am DoS Algorithmus nötig ist, muss das externe Programm neu deployed werden. Es benötigt keine Anpassungen an der Blockchain selbst. Für die Verwaltung der Whitelist, braucht das Programm eine Funktion, um Accounts zur Whitelist hinzuzufügen. Diese Funktion kann einfach erweitert werden, um eine Liste von Accounts zur Whitelist hinzuzufügen. Dadurch ist das hinzufügen von neuen Accounts für eine Klasse einfach automatisierbar.

Für die Betreuung des externen Programms wird ein zusätzlicher Server benötigt. Das bedeutet einen Mehraufwand für die FHNW.

Da der ALA einfach zu Automatisieren ist, sind für Betrieb 2 Punkte gesetzt worden.

#### **2.4.2.3 ALA 3: Externes Programm mit Whitelist**

Bei diesem ALA muss eine Smart Wallet und ein externes Programm entwickelt werden. Transaktionen werden im externen Programm verpackt und müssen von der Smart Wallet wieder entpackt werden. Somit liegt die Machbarkeit zwischen dem von ALA 1 und ALA 2. Daher werden 2 Punkte für Machbarkeit vergeben.

Mit der Verwendung von einem externen Programm, wird eine zentrale Autorität verwendet. Diese ist nicht dezentral und wird von der FHNW administriert. Im Gegensatz zu ALA 2, hat dieses Programm eine sehr viel zentralere Rolle. Das Programm interagiert nicht nur bei einer DoS Attacke mit der Blockchain, sondern ständig. Jede Transaktion wird an das Programm übermittelt und dort verarbeitet. Da die zentrale Autorität im Vergleich zu ALA 2 viel aktiver ist, ist für Blockchainprinzipien 1 Punkt vergeben worden.

Für die Betreuung des externen Programms ist ein zusätzlicher Server nötig.

Änderungen an der Smart Wallet bedingen ein erneutes Deployment.

In der Whitelist der Blockchain ist nur der Account des externen Programmes hinterlegt. Das Programm führt eine eigenen List von Accounts, die für gratis Transaktionen berechtigt sind.

Das externe Programm hat eine sehr zentrale Rolle, da es die Whitelist und den DoS Schutzalgorithmus enthält. Die Automatisierung wird daher als einfach eingestuft, da das externe Programm mit Java geschrieben wird und somit sehr viel zugänglicher ist. Daher sind bei Betrieb 3 Punkte vergeben worden.

#### 2.4.2.4 Resultat Evaluation

Durch die hohe Gewichtung von Machbarkeit, erzielt ALA 2 die meisten Punkte. Im weiteren Verlauf des Projektes wird daher ALA 2 umgesetzt.

Für die Realisierung des externen Programmes ist die Programmiersprache der Wahl Java. Java ist den Teammitgliedern bereits bekannt. Mit der Bibliothek Web3j sind Interaktionen mit der Blockchain effizient und intuitiv.

Im Anhang ist unter 6.3 ein weiterer Lösungsansatz aufgelistet. Dieser ist sehr früh in der Evaluierung als nicht realisierbar eingestuft worden und ist hier deshalb nicht aufgeführt.

### 2.4.3 DoS-Algorithmus

//TODO Spellcheck

In diesem Abschnitt sind die Komponenten des DoS-Algorithmus aufgeführt. Verschiedene Parameter und deren Verwendung werden untersucht. Die Behandlung von Accounts, die vom Algorithmus als Gefahr identifiziert werden, wird ebenfalls evaluiert.

#### 2.4.3.1 Parameter

Um zu bewerten, ob ein Account eine Gefahr für die Blockchain darstellt, braucht ein Algorithmus Parameter. Diese werden durch die Überwachung von getätigten gratis Transaktionen gesammelt. Dabei muss jeweils pro Account entschieden werden, ob ein Verhalten eine Gefahr darstellt. Nachfolgend sind mögliche Parameter für die Beurteilung von Accounts aufgeführt.

**Sender** Dieser Parameter ist zwingend nötig um eine gratis Transaktionen mit einem Account zu verknüpfen.

**Empfänger** Eine Transaktion wird immer an eine Adresse gesendet. Hierbei kann es sich sowohl um einen Benutzeraccount oder einen Smart Contract handeln.

**Reset-Intervall** Alle Interaktionen auf der Blockchain müssen relativ zu einem Zeitintervall bewertet werden. Hier werden zwei unterschiedliche Ansätze untersucht:

**Allgemeines Intervall** Gratis Transaktionen werden für alle Accounts im selben Zeitintervall betrachtet. Der Zeitpunkt ist relativ zum Programmstart. Beispielsweise ist als Intervall eine Stunde gesetzt und der Programmstart erfolgt um 8:00 UCT. Dadurch sind gratis Transaktionen die um 08:59 UTC gemacht werden, um 09:01 UTC nicht mehr relevant für die Beurteilung. Das hat zur Folge, dass Benutzer alle zulässigen Aktionen direkt vor und noch einmal, nach Ablauf eines Intervalls ausführen können.

**Individuelles Intervall** Das Intervall ist relativ zum Zeitpunkt einer getätigten gratis Transaktionen. Bei einer Prüfung wird untersucht, wie viele gratis Transaktionen der betroffene Account im vergangenen Zeitintervall, gerechnet ab dem Zeitpunkt der Prüfung, getätigt hat. Mit den selben Startparametern wie im oben aufgeführten Beispiel, ist eine um 08:59 UTC getätigte gratis Transaktion bis 09:59 relevant.

**Anzahl getätigte Transaktionen** Pro Account wird verfolgt, wie viele gratis Transaktionen pro Zeitintervall gemacht werden. Hier werden die Transaktionen unabhängig von Typ oder verursachten Komputationskosten auf der Blockchain gezählt.

**Anzahl verbrauchtes Gas** Pro Account wird verfolgt, wie viel Gas pro Zeitintervall auf der Blockchain durch dessen gratis Transaktionen verbraucht wird. Im Gegensatz zum oben genannten Parameter, werden hier die verursachten Komputationskosten auf der Blockchain berücksichtigt.

#### 2.4.3.2 Wiederaufnahme auf die Whitelist

Falls die Prüfung durch den Algorithmus positiv ausfällt, wird der betreffende Account von der Whitelist gelöscht. In diesem Abschnitt sind mögliche Vorgehensweisen aufgeführt, um einen Account nach der Löschung automatisch wieder zur Whitelist hinzuzufügen.

**Fixer Zeitpunkt für alle** Es wird ein fixer Zeitpunkt definiert, bei dem alle Accounts zurückgesetzt werden. Das heisst das Kontingent wird bei allen Accounts wieder auf den konfigurierten Wert gesetzt. Von der Whitelist gelöschte Accounts werden dieser wieder hinzugefügt. Zum Beispiel könnte als Zeitpunkt Montag 8:00 UTC definiert werden.

**Nach Zeitintervall** Ein Account wird für eine definierte Dauer von der Whitelist gelöscht. Die Zeit wird ab der Löschung von der Whitelist gemessen. Dadurch werden bei einem Vergehen alle Accounts gleich lange von gratis Transaktionen ausgeschlossen.

**Inkrementierendes Zeitintervall** Wie lange ein Account von der Whitelist entfernt wird, ist abhängig von der Anzahl bereits begangener Verstöße.

Beispiel:

# Verstöße	Dauer Sperrung
1	0.50
2	1.00
3	3.00
4	12.00
5	60.00
6	360.00

In der oben aufgeführten Tabelle ist ersichtlich, dass die Dauer der Sperrung proportional zu den Verstößen ist.

#### 2.4.3.3 Benutzermanagement

Bei der Verwaltung von Accounts geht es darum, wie die vorhergehenden Parameter und Intervalle auf die Accounts angewendet werden. Es werden drei Mögliche Ansätze betrachtet.

**Kein Benutzermanagement** Die Parameter werden global konfiguriert und gelten für alle Accounts. Eine Differenzierung von Accounts ist somit nicht möglich.

**Parameter über Gruppen konfigurierbar** Die Parameter sind über Gruppen konfiguriert. Jedem Account wird eine Gruppe zugewiesen, dieser erbt die Parameter der Gruppe. So lassen sich Strukturen der Schule, wie Studenten, Dozenten und Klassen einfach abbilden.

**Parameter pro Account konfigurierbar** Die Parameter sind bei jedem Account individuell konfigurierbar.

#### 2.4.4 Evaluation DoS-Algorithmus

In diesem Abschnitt werden die Komponenten des Algorithmus evaluiert.

#### 2.4.4.1 Parameter

Die aufgeführten Parameter werden auf ihre Relevanz für die Erkennung einer DoS Attacke geprüft.

**Sender** Ist zwingend nötig um eine Transaktion einem Account zuweisen zu können.

**Empfänger** Dieser kann von Sender frei gewählt werden. Es wird auch kein Einverständnis des Empfängers für eine Transaktion benötigt. Jeder Benutzer ist weiter in der Lage, selbst neue Accounts zu erstellen und diese als Empfänger zu verwenden. Der Parameter hat somit keine Aussagekraft und wird nicht verwendet.

**Reset-Intervall** Wir haben uns für die Implementierung eines allgemeinen Intervalls entschieden. Der Ansatz ist bedeutend einfacher umzusetzen als ein individuelles Intervall und kann daher sicher in der gegebenen Zeit realisiert werden. Am Ende des Intervalls, werden die Zähler für alle Parameter pro Account zurückgesetzt.

Die Auswirkung des genannten Nachteils beim allgemeinen Intervall ist stark von dessen Länge abhängig. Je kürzer das Intervall gewählt wird, umso kleiner sind die möglichen Folgen.

**Anzahl gratis Transaktionen** Dieser Parameter wird verwendet. Er ermöglicht es eine DoS Attacke zu identifizieren, welche die Beeinträchtigung der Blockchain mittels einer grossen Zahl von gratis Transaktionen erreichen will.

**Anzahl verbrauchtes Gas** Wie unter 2.2.7 erwähnt, können Transaktionen mit einem sehr hohen Gas-Bedarf für eine DoS-Attacke verwendet werden. Da beim Angreifer mit der Verwendung von gratis Transaktionen keine Mehrkosten anfallen, ist dieser Angriff sehr naheliegend. Daher wird dieser Parameter ebenfalls verwendet.

#### 2.4.4.2 Wiederaufnahme auf die Whitelist

Ein fixer Zeitpunkt ist sehr einfach umzusetzen. Allerdings werden dadurch die Accounts nicht mehr gleich behandelt. Wie lange ein Account keine gratis Transaktionen mehr tätigen kann, ist abhängig davon, zu welchem Zeitpunkt er von der Whitelist gelöscht wird. Wenn der gesetzte Zeitpunkt dem Benutzer bekannt ist, kann das System missbraucht werden. Wird ein DoS Angriff kurz vor dem Resetzeitpunkt ausgeführt, hat es praktisch keine Folgen für den Benutzer. Sein Account wird zwar von der Whitelist entfernt, aber mit dem entsprechendem Zeitmanagement gleich wieder entsperrt.



Mit einem Zeitintervall werden alle Accounts gleich lange von der Whitelist gelöscht. Dieser Ansatz bietet daher mehr Fairness als ein fixer Zeitpunkt.

Je öfter mit einem Account gegen die Regeln verstossen wird, desto kleiner ist die Wahrscheinlichkeit, dass es sich um Versehen handelt. Daher kann davon ausgegangen werden, dass ein Wiederholungstäter aktiv versucht, die Blockchain zu schädigen. Mit einem inkrementierenden Intervall werden diese Accounts gezielt und härter bestraft als bei den anderen Ansätzen.

Einmalige Verstöße die versehentlich auftreten werden in einer Lernumgebung als wahrscheinlich eingeschätzt. Mit diesem System werden solche Versehen sehr milde bestraft.

Wir haben uns entschieden, eine Kombination aus einem fixen Zeitpunkt und einem inkrementierenden Intervall zu verwenden. Dieser Ansatz ist in der gegebenen Zeit realisierbar und bietet nebst einem effizienten Schutz auch eine Toleranz für einmalige Verstöße.

Die Dauer einer Suspendierung von der Whitelist kann mit dem Parameter „Revoke-Faktor“ konfiguriert werden. Als Basis wird das Reset-Intervall verwendet.

$$t = resInter * revFak * v$$

Wobei  $t$  die Dauer der Suspendierung,  $resInter$  das Reset-Intervall,  $revFak$  der Revoke-Faktor und  $v$  die Anzahl bereits begangener Verstöße abbilden.

Anbei Beispiel mit einem Reset-Intervall von fünf Minuten, einem Revoke-Faktor von 3 und der daraus resultierenden Suspendierung von der Whitelist in Minuten:

resInter	revFak	Verstöße	Suspendierung (min)
5	3	1	15
5	3	2	30
5	3	3	45
5	3	4	60
5	3	5	75
5	3	6	90

#### 2.4.4.3 Benutzermanagement

Es besteht der Bedarf, dass Accounts von Dozenten toleranter behandelt werden als solche von Studenten. Daher muss ein Benutzermanagement implementiert werden.

Ein gruppenbasiertes Benutzermanagement ist intuitiv und effizient, da vorhandene Strukturen der FHWN, wie Klassen oder Dozenten, abgebildet werden können. Die Implementation wird jedoch als

sehr komplex eingeschätzt. Die Realisierbarkeit in der gegebenen Zeit ist fraglich. Der Ansatz wird daher nicht implementiert.

Das lässt nur die Möglichkeit, jeden Account einzeln zu konfigurieren. Es wird erwartet, dass für die Mehrheit der Accounts kein Bedarf an individuellen Parametern besteht. Um diesen Umstand gerecht zu werden, werden Standardparameter angeboten. Diese werden verwendet, für die Parameter nicht explizit definiert werden. So kann die Mehrheit der Accounts über Standardparameter und Ausnahmen individuell konfiguriert werden.

Um zu verhindern, dass das externe Programm angreifbar wird, kann das Reset-Intervall nur global definiert werden. Bei einem individuellen Reset-Intervall müsste für jeden Verstoss einer neuer Thread im Programm gestartet werden. Dadurch würde das Programm selbst anfällig für eine DoS Attacke.

## 2.4.5 Konfiguration des Algorithmus

Um dem Betreiber die Möglichkeit zu geben, den Algorithmus an seine Bedürfnisse anzupassen, können die Parameter und Zeitintervalle, siehe 2.4.4, konfiguriert werden. Die Konfiguration wird mit einer Textdatei vorgenommen. Für alle Parameter müssen natürliche Zahlen verwendet werden. Folgende Parameter können pro Account gesetzt werden:

### Gratis Transaktionen

```
1 Definiert die maximale Anzahl gratis Transaktionen die pro Reset-Intervall getätigt werden können.
```

### Gratis Gas

```
1 Definiert die maximale Menge an Gas die mit gratis Transaktionen innerhalb eines Reset-Intervalls verbraucht werden können.
```

Wenn für einen Account individuelle Schwellenwerte für Transaktionen und Gas definiert werden, müssen immer beide Parameter gesetzt werden.

Folgende Parameter gelten für alle Accounts:

**Reset-Intervall** Einheit ist Minuten, definiert die Länge des Reset-Intervalls.

**Revoke-Intervall** Anzahl der Reset-Intervalls, für die ein Account bei einer positiven Prüfung durch den Algorithmus von der Whitelist gelöscht wird.

**Standardwert gratis Transaktionen** Gilt für Accounts die ohne Parameter erfasst werden. Definiert die maximale Anzahl gratis Transaktionen die pro Reset-Intervall getätigt werden können.

**Standardwert gratis Gas** Gilt für Accounts die ohne Parameter erfasst werden. Definiert die maximale Menge an Gas die mit gratis Transaktionen innerhalb eines Reset-Intervalls verbraucht werden können.

Bei der Konfiguration sollten die Abhängigkeiten zwischen den Parametern geachtet werden. Verfügbares Gas, Anzahl Transaktionen und das Reset-Intervall sollten immer zusammen konfiguriert werden.

## 3 Praktischer Teil

Dieses Kapitel beschreibt, wie die gewonnen theoretischen Grundlagen umgesetzt sind. Die realisierte Lösung wird kritisch hinterfragt und anderen Lösungsansätzen gegenübergestellt.

### 3.1 Parity

In diesem Abschnitt ist beschrieben, wie die Blockchain konfiguriert ist. Als Client wird die stable Version[41] von Parity verwendet.

#### 3.1.1 Konfiguration der Blockchain

Parity wird mit der Konsole gestartet. Der Benutzer hat hier die Möglichkeit, gewisse Parameter an Parity zu übergeben. Eine einfache Konfiguration ist somit möglich. Für kompliziertere Konfigurationen, wird die Verwendung von einer Konfigurationsdatei empfohlen, diese ist im nächsten Abschnitt 3.1.1.1 beschrieben.

Die hier gezeigte Konfiguration ist für die Entwicklung verwendet worden. Hierbei ist es wichtig, dass Aktionen möglichst schnell auf der Blockchain sichtbar sind. Aus diesem Grund wurde auf einen Mining-Algorithmus verzichtet. Für einen produktiven Betrieb sollte die Konfiguration auf die eigenen Bedürfnisse geprüft und gegebenenfalls angepasst werden.

##### 3.1.1.1 Config.toml

Für die Konfiguration der Blockchain wird eine Konfigurationsdatei verwendet. Diese hat das Dateiformat .toml[42].

```
1 [parity]
2 chain = "/home/parity/.local/share/io.parity.ethereum/genesis/
   instant_seal.json"
3 base_path = "/home/parity/"
4
5 [rpc]
6 cors = ["all"]
```

```
7 apis = ["net", "private", "parity", "personal", "web3", "eth"]
8
9 [mining]
10 min_gas_price = 10000000000
11 refuse_service_transactions = false
12 tx_queue_no_unfamiliar_locals = true
13 reseal_on_txs = "all"
14 reseal_min_period = 0
15 reseal_max_period = 6000
16
17 [misc]
18 unsafe_expose = true
```

Der oben aufgeführte Codeblock ist in Sektionen gegliedert. Diese sind durch einen Namen in eckigen Klammern definiert. Innerhalb einer Sektion existieren bestimmte Schlüssel mit einem Wert. Jede Sektion ist in den folgenden Abschnitten erklärt.

**Parity** In dieser Sektion sind die grundlegenden Eigenschaften der Blockchain definiert. Dazu gehören Genesisblock und der Speicherort.

**Zeile 2** Der zu verwendende Genesisblock. Es wird der Pfad zu der entsprechenden JSON Datei[43] angegeben.

**Zeile 3** Mit „base\_path“ wird angegeben, wo die Blockchain abgespeichert werden soll. Hier wird das gewünschte Verzeichnis angegeben.

**RPC** Diese Sektion definiert, wie die Blockchain erreichbar ist.

**Zeile 6** „cors“ steht für Cross-Origin Requests. Dieser Parameter wird benötigt, um die Interaktion von Remix[44] oder Metamask[45] mit der Blockchain zu ermöglichen.

**Zeile 7** Hier sind die API's definiert, welche über HTTP zur Verfügung gestellt werden.

**Mining** Diese Sektion regelt das Verhalten beim Mining von Blocks.

**Zeile 10** Der minimale Gas-Preis der gezahlt werden muss, damit eine Transaktion in einen Block aufgenommen wird. Der Preis ist in WEI angegeben. Um sicherstellen, dass nur die definierte Benutzergruppe gratis Transaktionen tätigen kann, muss dieser Wert grösser als Null sein.

**Zeile 11** Service Transaktionen haben einen Gas-Preis von Null. Wird hier „true“ gesetzt, können keine gratis Transaktionen getätigt werden, unabhängig davon, ob eine Whitelist vorhanden ist oder nicht.

**Zeile 12** Dieser Parameter wird benötigt, dass Transaktionen die mittels RPC an Parity übermittelt werden, nicht als lokal betrachtet werden. Das ist sehr wichtig, da lokale Transaktionen standar-

mässig auch über einen Gas-Preis von Null verfügen dürfen. So wird sichergestellt, dass nur die definierte Benutzergruppe gratis Transaktionen tätigen darf.

**Zeile 13** Durch die Einstellung „tx\_queue\_no\_unfamiliar\_locals = true“ werden alle eingehenden Transaktionen behandelt, als ob fremd, also nicht lokal, behandelt. Standardmässig, werden aber nur lokale Transaktionen verarbeitet. Daher muss hier explizit definiert werden, dass alle Transaktionen verarbeitet werden.

**Zeile 14** Gibt an, wieviele Milisekunden im Minimum zwischen der Kreation von Blöcken liegen müssen.

**Zeile 15** Definiert die maximale Zeitspanne in Millisekunden zwischen der Kreation von Blöcken. Nach Ablauf dieser Zeit wird automatisch ein Block generiert. Dieser kann leer sein.

**Misc** In dieser Sektion sind Parameter, die sonst nirgends reinpassen.

**Zeile 18** Wird für die Interaktion mit Remix und Metamask benötigt.

### 3.1.1.2 Blockchainspezifikation

Mit dieser Datei wird die Blockchain definiert. Sie enthält nebst der Spezifikation den Genesis Block. Weiter können Benutzeraccounts und Smart Contracts definiert werden. Diese können verwendet werden, sobald die Blockchain gestartet ist.

[illegible]

[illegible]

Oben aufgeführt ist die Blockchainspezifikation. Im folgenden Abschnitt ist diese Zeilenweise erläutert.

**Zeile 2** Name der Blockchain

**Zeile 3 - 7** Der Abschnitt `engine` definiert, wie die Blöcke verarbeitet werden.

**Zeile: 4** Mit `instantSeal` wird angegeben, dass kein Miningalgorithmus verwendet wird. Die Blöcke, sofern valide, werden sofort in die Blockchain aufgenommen.

**Zeile 5** Die Engine InstantSeal braucht keine weiteren Parameter. Falls ein anderer Algorithmus verwendet wird, kann dieser hier konfiguriert werden.

**Zeile 8 - 15** Im Abschnitt `params` sind die generellen Parameter für die Blockchain aufgeführt.

**Zeile 9** Die verwendete Netzwerk ID. Die grossen Netzwerke haben eine definierte ID. Falls einem bestehenden Netzwerk beigetreten werden soll, muss diese korrekt gewählt werden. Der Wert 11 ist keinem Netzwerk zugeordnet, daher kann dieser für ein privates Netzwerk genutzt werden.

**Zeile 10** Der `registrar` hat als Wert die Adresse der Name Registry, siehe 2.3.1.1. Da bereits beim ersten Start von Parity die Adresse der Name Registry hinterlegt werden muss, findet das Deployment direkt in der Blockchainspezifikation statt.

**Zeile 11** Die maximale Grösse eines Smart Contracts welcher in mit einer Transaktion deployed wird.

**Zeile 12** Spezifiziert die maximale Anzahl Bytes, welche im Feld `extra_data` des Headers eines Blockes mitgegeben werden kann.

**Zeile 13** Definiert den minimalen Gasbetrag, der bei einer Transaktion mitgegeben werden muss.

**Zeile 14** Schränkt die Schwankungen der Gas Limite zwischen Blöcken ein.

**Zeile 16 - 22** Mit dem Abschnitt `genesis` ist der Genesis Block, also der erste Block, der Blockchain definiert.

**Zeile 17 - 19** Hier kann weiter definiert werden, wie Blöcke verarbeitet werden sollen. Da für dieses Projekt valide Blöcke sofort in die Blockchain eingefügt werden, sind keine weiteren Einstellungen nötig.

**Zeile 20** Gibt die Schwierigkeit des Genesis Blocks an. Da als Engine InstantSeal verwendet wird, hat dieser Parameter keinen Einfluss.

**Zeile 21** Gibt an, was die Gaslimite des Genesis Blockes ist. Da die Gaslimite für Blöcke dynamisch berechnet wird, hat dieser Wert einen Einfluss auf zukünftige Gaslimiten.

**Zeile 23 - 26** Dieser Abschnitt erlaubt es, Accounts zu definieren. Diese können für Benutzer oder Smart Contracts sein. Jeder Account wird mit einer Adresse und einem Guthaben initialisiert.

Bei einem Account für einen Smart Contract, wird zusätzlich dessen Bytecode angegeben.

**Zeile 24** Hier ist die SimpleRegistry, siehe Abschnitt 2.3.1 und 2.3.1.1, definiert. Der erste Parameter ist die Adresse, unter welcher der Smart Contract erreichbar sein soll. Das Guthaben wird mit einem Ether initialisiert. Der Wert für `constructor` ist der Bytecode des kompilierten Smart Contracts. Dieser ist aufgrund seiner Grösse durch einen Platzhalter ersetzt worden.

**Zeile 25** Definition von einem Benutzeraccount. Der erste Parameter ist die Adresse. Dem Account kann ein beliebiges Guthaben zugewiesen werden.

### 3.1.2 Docker

Um eine möglichst realitätsnahe Entwicklungsumgebung zu erhalten, wird Docker[46] für die Betreuung der Blockchain verwendet. Mehr Details zur Verwendung von Docker sind im Anhang unter 6.2.4 vorhanden.

### 3.1.3 Name Registry

Die Name Registry wird standardmässig in Parity verwendet. Die zur Verfügung gestellte Implementation der Name Registry ist SimpleRegistry genannt. Der vollständige Code ist im Anhang unter 6.6 aufgeführt.

### 3.1.4 Certifier

Parity stellt eine Implementation des Certifiers zu Verfügung, den SimpleCertifier. Der vollständige Code ist im Anhang unter 6.7 aufgeführt.

Sobald der Certifier bei der Name Registry registriert ist, können Accounts definiert werden, die gratis Transaktionen tätigen können.

#### 3.1.4.1 Deployment und Registrierung

Für eine Erfolgreiche Verwendung des Certifiers, die Name Registry in der Blockchainspezifikation definiert sein. Sobald Parity gestartet ist, kann mit dem Deployment des Certifiers begonnen werden. Hierfür wird Java und die Bibliothek Web3j[47] verwendet.

Um in Java mit Smart Contracts auf der Blockchain interagieren zu können, wird eine Wrapperklasse des Smart Contracts benötigt. Für dessen Generierung wird das Web3j Command Line Tool (`web3j-cli`)[48] und der Solidity Compiler (`solc`)[49] verwendet. Die Wrapper für die SimpleRegistry und den



SimpleCertifier sind im Anhang unter 6.6.5 und 6.7.4 zu finden. Der Bytecode ist bei beiden Wrapper nicht enthalten. Dieser kann jederzeit mit solc generiert werden[50].

Um einen Smart Contract auszurollen wird eine Instanz der generierten Wrapperklasse genutzt. Es wird die Methode `deploy` der Wrapperklasse genutzt.

```
1 private Web3j web3j = Web3j.build(new HttpService("http://jurijnas.  
  myqnapcloud.com:8545/"));  
2 private TransactionManager transactionManager = new  
  RawTransactionManager(web3j, Credentials.create(privateKey));  
3  
4 private SimpleCertifier simpleCertifier;  
5 try {  
6     simpleCertifier = SimpleCertifier.deploy(web3j, transactionManager,  
        new DefaultGasProvider()).send();  
7 } catch (Exception e) {  
8     e.printStackTrace();  
9 }  
10  
11 String simpleCertifierAddress = simpleCertifier.getContractAddress();
```

Die Verbindung zu einem Node wird mit einer Instanz von `Web3j` auf Zeile 1 definiert. Auf der zweiten Zeile wird ein `TransactionManager` instanziiert. Dieser definiert, wie und mit welchem Account auf die Ethereumblockchain verbunden wird.

Auf Zeile 6 findet das eigentliche Deployment statt. Nebst dem `Web3j` und dem `Transactionmanager` wird ein `ContractGasProvider` benötigt. Dieser definiert den Gas Price und die Gas Limite. Mit dem `DefaultGasProvicer` werden Standardwerte verwendet. Durch das Deployment erhalten wir eine Instanz des `SimpleCertifiers`. Diese kann nun verwendet werden um weitere Aktionen auf der Blockchain auszuführen.

Auf Zeile 11 wird die Adresse des Smart Contracts in eine Variable gespeichert.

Um den Certifier bei der Name Registry registrieren zu können, muss von der Name Registry ebenfalls eine Instanz erstellt werden. Auch hier wird die Wrapperklasse verwendet.

```
1 private Web3j web3j = Web3j.build(new HttpService("http://jurijnas.  
  myqnapcloud.com:8545/"));  
2 private TransactionManager transactionManager = new  
  RawTransactionManager(web3j, Credentials.create(privateKey));  
3  
4 private SimpleRegistry simpleRegistry;  
5 try {  
6     simpleRegistry = SimpleRegistry.load(simpleRegistryAddress, web3j,  
        transactionManager, new DefaultGasProvider());  
7 } catch (Exception e) {  
8     e.printStackTrace();  
9 }
```

Um eine Instanz von einem bereits platzierten Smart Contract zu erhalten, wird die Methode `load` verwendet. Als erster Argument wird die Adresse der Name Registry mitgegeben. Analog zum vorherigem Beispiel wird die Verbindung zur Blockchain mit `Web3j`, einem `Transactionmanager` und einem `DefaultGasProvider` definiert. Der Rückgabewert ist eine Instanz der `SimpleRegistry`.

Mit den zur Verfügung stehenden Instanzen, kann die Registrierung des Certifiers bei der Name Registry gemacht werden.

```
1 private static BigInteger REGISTRATION_FEE = BigInteger.valueOf
    (1000000000000000000L);
2
3 String str_hash = "6
    d3815e6a4f3c7fcec92b83d73dda2754a69c601f07723ec5a2274bd6e81e155";
4 private byte[] hash = new BigInteger(str_hash, 16).toByteArray();
5
6 try {
7     simpleRegistry.reserve(hash, REGISTRATION_FEE).send();
8     simpleRegistry.setAddress(hash, "A", simpleCertifier.
        getContractAddress()).send();
9 } catch (Exception e) {
10     e.printStackTrace();
11 }
```

Für die Registrierung wird eine Gebühr von einem Ether erhoben. Dafür wird auf Zeile 1 eine Variabel vom Typ `BigInteger` instanziiert.

Der auf Zeile 3 definierte String `str_hash` ist der sha3-Hash für den String `service_transaction_checker`. Dieser wird auf Zeile 4 in ein Byte-Array umgewandelt. Diese Variabel hält den Namen, unter welchem der Certifier bei der Name Registry registriert wird. Die Verwendung des Strings `service_transaction_checker` und sein Umwandlung sind in Parity hart kodiert und können nicht angepasst werden.

Auf Zeile 7 wird die Reservierung bei der Name Registry vorgenommen. Hier wird der Name und die anfallende Gebühr von einem Ether gesendet.

Auf Zeile 8 wird die Registrierung abgeschlossen. In der Name Registry wird die Bindung zwischen Namen und Adresse erstellt. Als erster Parameter wird der Name übergeben. Das zweite Argument ist der Zugriffsschlüssel in der Map. Auch dieser ist von Parity vorgegeben, es muss zwingend `"A"` übergeben werden. Das dritte Argument ist die Adresse des Certifiers. Diese wird von dessen Instanz abgerufen.

## 3.2 Externes Programm

In diesem Kapitel ist die Implementierung des externen Programms zur Überwachung der Whitelist in Parity beschrieben. Anhand von Codeausschnitten ist die Funktionsweise von einzelnen Komponenten

näher erklärt.

### 3.2.1 Wrapperklassen

Für die Interaktion mit Smart Contracts werden generierte Wrapperklassen verwendet. Es ist je eine Wrapperklasse für die Name Registry und den Certifier vorhanden. Für dessen Generierung und Verwendung siehe 3.1.4.1.

### 3.2.2 Überwachung von Transaktionen

Um die Transaktionen auf der Blockchain zu Observieren wird ein Filter[51] von Web3j verwendet. Dieser erlaubt es uns, eine [Subscription](#) zu erstellen. Diese läuft asynchron in einem eigenen Thread. Jede getätigte Transaktion wird von der [Subscription](#) erfasst. Es werden jedoch nur Transaktionen genauer untersucht, die einen Gas Preis von Null aufweisen.

//TODO

### 3.2.3 DoS Algorithmus

//TODO

### 3.2.4 Initialisierung

Deployment und Registrierung

### 3.2.5 Konfiguration

Die Konfiguration des Programmes findet mit einer Textdatei statt. In dieser werden alle Accounts angegeben, welche auf die Whitelist sollen. Weiter werden die Schwellenwerte für den DoS Algorithmus angegeben.

Die Konfigurationsdatei wird zeilenweise interpretiert. Das Einlesen der Datei hat keine Fehlertoleranz. Daher muss die hier beschriebene Struktur stets eingehalten werden. Wie unter 2.4.5 erläutert, gibt es Einstellungen pro Account und solche die global gelten. In der ersten Zeile der Datei werden alle globalen Parameter in folgender Reihenfolge aufgelistet:

1. Reset-Intervall
2. Revoke-Intervall

3. Standard Wert für gratis Transaktionen
4. Standard Wert für gratis Gas

Es müssen alle vier Parameter angegeben und mit einem ; separiert werden.

Alle nachfolgenden Zeilen enthalten jeweils eine Accountadresse. Zusätzlich kann die Anzahl gratis Transaktionen und die Menge gratis Gas pro Account definiert werden. Hier muss beachtet werden, dass es nicht möglich ist, nur einen der fakultativen Parameter anzugeben. Es müssen beide angegeben werden.

Ab Zeile 2 müssen die Parameter folgendermassen angegeben werden:

1. Accountadresse
2. [Anzal gratis Transaktionen]
3. [Menge an gratis Gas]

## 4 Fazit

### 4.0.0.1 Dokumentation

Parity wird stetig weiterentwickelt. Die letzte Minorversion[52] ist im April 2019 veröffentlicht worden. Obwohl es sich um eine Minorversion handelt, hat es Änderungen in der Code-Syntax. Daher verhält sich das Update eher wie eine neue Majorversion[52]. Das hat zur Folge, dass praktisch alle gefundenen Tutorials nicht mehr gültig sind.

## 5 Quellenverzeichnis

- [1] „Blockchain - Wikipedia“, 2019. [Online]. Verfügbar unter: <https://en.wikipedia.org/wiki/Blockchain>.
- [2] „University of Applied Sciences and Arts Northwestern Switzerland“, 2019. [Online]. Verfügbar unter: <https://www.fhnw.ch/>.
- [3] M. Inc., „What is Gas | MyEtherWallet Knowledge Base“, 2018. [Online]. Verfügbar unter: <https://kb.myetherwallet.com/en/transactions/what-is-gas/>.
- [4] „Denial-of-service attack - Wikipedia“, 2019. [Online]. Verfügbar unter: [https://en.wikipedia.org/wiki/Denial-of-service\\_attack](https://en.wikipedia.org/wiki/Denial-of-service_attack).
- [5] „Genesis block - Bitcoin Wiki“, 2019. [Online]. Verfügbar unter: [https://en.bitcoin.it/wiki/Genesis\\_block](https://en.bitcoin.it/wiki/Genesis_block).
- [6] „Peer-to-peer - Wikipedia“, 2019. [Online]. Verfügbar unter: <https://en.wikipedia.org/wiki/Peer-to-peer>.
- [7] „What Is a Blockchain Consensus Algorithmen | Binance Academy“, 2019. [Online]. Verfügbar unter: <https://www.binance.vision/blockchain/what-is-a-blockchain-consensus-algorithm>.
- [8] „Bitcoin - Wikipedia“, 2019. [Online]. Verfügbar unter: <https://en.wikipedia.org/wiki/Bitcoin>.
- [9] Ethereum, „Home | Ethereum“, 2019. [Online]. Verfügbar unter: <https://www.ethereum.org/>.
- [10] „Nick Szabo - Wikipedia“, 2019. [Online]. Verfügbar unter: [https://en.wikipedia.org/wiki/Nick\\_Szabo](https://en.wikipedia.org/wiki/Nick_Szabo).
- [11] „Smart Contracts for Alpiq | ETH Zürich“, 2019. [Online]. Verfügbar unter: <https://ethz.ch/en/industry-and-society/industry-relations/industry-news/2019/04/smart-contract-for-alpiq.html>.
- [12] „CryptoKitties | Collect and breed digital cats!“, 2019. [Online]. Verfügbar unter: <https://www.cryptokitties.co/>.
- [13] „Wei“, 2019. [Online]. Verfügbar unter: <https://www.investopedia.com/terms/w/wei.asp>.
- [14] S. Fontaine, „Understanding Bytecode on Ethereum - Authereum - Medium“, 2019. [Online]. Verfügbar unter: <https://medium.com/authereum/bytecode-and-init-code-and-runtime-code-oh-my-7bcd89065904>.

- [15] K. Tam, „Transactions in Ethereum - KC Tam - Medium“, 2019. [Online]. Verfügbar unter: <https://medium.com/@kctheservant/transactions-in-ethereum-e85a73068f74>.
- [16] Y. Riady, „Signing and Verifying Ethereum Signatures - Yos Riady“, 2019. [Online]. Verfügbar unter: <https://yos.io/2018/11/16/ethereum-signatures/>.
- [17] „Remote procedure call - Wikipedia“, 2020. [Online]. Verfügbar unter: [https://en.wikipedia.org/wiki/Remote\\_procedure\\_call](https://en.wikipedia.org/wiki/Remote_procedure_call).
- [18] „<https://keccak.team/>“, 2019. [Online]. Verfügbar unter: [Keccak%20Team](https://keccak.team/).
- [19] „Elliptic Curve Digital Signature Algorithm - Wikipedia“, 2019. [Online]. Verfügbar unter: [https://en.wikipedia.org/wiki/Elliptic\\_Curve\\_Digital\\_Signature\\_Algorithm](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm).
- [20] „SHA-3 - Wikipedia“, 2019. [Online]. Verfügbar unter: <https://en.wikipedia.org/wiki/SHA-3>.
- [21] „Ethereum Series - Understanding Nonce - The Startup - Medium“, 2019. [Online]. Verfügbar unter: <https://medium.com/swlh/ethereum-series-understanding-nonce-3858194b39bf>.
- [22] N. Jabes, „Nu Jabe’s answer to What is an Ethereum contract address? - Quora“, 2019. [Online]. Verfügbar unter: <https://www.quora.com/What-is-an-Ethereum-contract-address/answer/Nu-Jabes>.
- [23] „Crypto Wallet Types Explained | Binance Academy“, 2019. [Online]. Verfügbar unter: <https://www.binance.vision/blockchain/crypto-wallet-types-explained>.
- [24] M. Wachal, „What is a blockchain wallet? - SoftwareMill Tech Blog“, 2019. [Online]. Verfügbar unter: <https://blog.softwaremill.com/what-is-a-blockchain-wallet-bbb30fbf97f8>.
- [25] StellaBelle, „Cold Wallet Vs. Hot Wallet: What’s The Difference?“, 2019. [Online]. Verfügbar unter: <https://medium.com/@stellabelle/cold-wallet-vs-hot-wallet-whats-the-difference-a00d872aa6b1>.
- [26] M. Wright, „So many mobile wallets, so little differentiation - Argent - Medium“, 2019. [Online]. Verfügbar unter: <https://medium.com/argenthq/recap-on-why-smart-contract-wallets-are-the-future-7d6725a38532>.
- [27] E. Conner, „smart Wallets are Here - Gnosis“, 2019. [Online]. Verfügbar unter: <https://blog.gnosis.pm/smart-wallets-are-here-121d44519cae>.
- [28] D. Labs, „Why Dapper is a smart contract wallet - Dapper Labs - Medium“, 2019. [Online]. Verfügbar unter: <https://medium.com/dapperlabs/why-dapper-is-a-smart-contract-wallet-ef44cc51cfa5>.
- [29] „Crypto Bites: Chat with Ethereum founder Vitalik Buterin“, 2019. [Online]. Verfügbar unter: [https://www.youtube.com/watch?v=u-i\\_mTwL-FI&feature=emb\\_logo](https://www.youtube.com/watch?v=u-i_mTwL-FI&feature=emb_logo).
- [30] R. Greene und M. N. Johnstone, „An investigation into a denial of service attack on an ethereum network“, 2018. [Online]. Verfügbar unter: <https://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1219&context=ism>.

- [31] „ethereum/yellowpaper: The Yellow Paper: Ethereum’s formal specification“, 2019. [Online]. Verfügbar unter: <https://github.com/ethereum/yellowpaper>.
- [32] go-ethereum, „Go Ethereum“, 2019. [Online]. Verfügbar unter: <https://geth.ethereum.org/>.
- [33] P. Technologies, „Blockchain Infrastructure for the Decentralised Web | Parity Technologies“, 2019. [Online]. Verfügbar unter: <https://www.parity.io>.
- [34] „https://github.com/ethereum/aleth“, 2019. [Online]. Verfügbar unter: <https://github.com/ethereum/aleth>.
- [35] „ethereum/trinity: The Trinity client for the Ethereum network“, 2019. [Online]. Verfügbar unter: <https://github.com/ethereum/trinity>.
- [36] „Rust Programming Language“, 2019. [Online]. Verfügbar unter: <https://www.rust-lang.org/>.
- [37] „Parity Name Registry - Parity Tech Documentation“, 2020. [Online]. Verfügbar unter: <https://wiki.parity.io/Parity-name-registry.html>.
- [38] „Permissioning - Parity Tech Documentation“, 2020. [Online]. Verfügbar unter: <https://wiki.parity.io/Permissioning#how-it-works-3>.
- [39] „Domain Name System - Wikipedia“, 2020. [Online]. Verfügbar unter: [https://en.wikipedia.org/wiki/Domain\\_Name\\_System](https://en.wikipedia.org/wiki/Domain_Name_System).
- [40] „Common Patterns - Solidity 0.4.24 documentation“, 2020. [Online]. Verfügbar unter: <https://solidity.readthedocs.io/en/v0.4.24/common-patterns.html#restricting-access>.
- [41] P. Technologies, „Releases - paritytech/parity-ethereum“, 2020. [Online]. Verfügbar unter: <https://github.com/paritytech/parity-ethereum/releases>.
- [42] „TOML - Wikipedia“, 2020. [Online]. Verfügbar unter: <https://en.wikipedia.org/wiki/TOML>.
- [43] „JSON - Wikipedia“, 2020. [Online]. Verfügbar unter: <https://en.wikipedia.org/wiki/JSON>.
- [44] „Remix - Ethereum IDE“, 2020. [Online]. Verfügbar unter: <https://remix.ethereum.org/>.
- [45] MetaMask, „MetaMask“, 2019. [Online]. Verfügbar unter: <https://metamask.io/>.
- [46] „Empowering App Development for Developers | Docker“, 2020. [Online]. Verfügbar unter: <https://www.docker.com/>.
- [47] „Web3j SDK - Where Java meets the Blockchain“, 2020. [Online]. Verfügbar unter: <https://www.web3labs.com/web3j>.
- [48] „Releases - web3j/web3j“, 2020. [Online]. Verfügbar unter: <https://github.com/web3j/web3j/releases>.
- [49] „Installing the Solidity Compiler – Solidity 0.6.4 documentation“, 2020. [Online]. Verfügbar unter: <https://solidity.readthedocs.io/en/develop/installing-solidity.html>.



- [50] „Generate a Java Wrapper from your Smart Contract - Kauri“, 2020. [Online]. Verfügbar unter: <https://kauri.io/generate-a-java-wrapper-from-your-smart-contract/84475132317d4d6a84a2c42eb9348e4b/a>.
- [51] „Getting Starte - web3j latest documentation“, 2020. [Online]. Verfügbar unter: [https://web3j.readthedocs.io/en/latest/getting\\_started.html#filters](https://web3j.readthedocs.io/en/latest/getting_started.html#filters).
- [52] „Software versioning“, 2020. [Online]. Verfügbar unter: [https://en.wikipedia.org/wiki/Software\\_versioning](https://en.wikipedia.org/wiki/Software_versioning).
- [53] T. B. G. 2019, „Sweet Tools for Smart Contracts“, 2019. [Online]. Verfügbar unter: <https://www.truffle-suite.com/>.
- [54] uPort, „uPort“, 2019. [Online]. Verfügbar unter: <https://www.uport.me/>.
- [55] A. Wallet, „Atomic Cryptocurrency Wallet“, 2019. [Online]. Verfügbar unter: <https://atomicwallet.io/>.
- [56] E. M. Inc., „Crypte Wallet - Send, Receive & Exchange Cryptocurrency | Exodus“, 2019. [Online]. Verfügbar unter: <https://www.exodus.io>.
- [57] MyEtherWallet, „MyEtherWallet | MEW“, 2019. [Online]. Verfügbar unter: <https://www.myetherwallet.com/>.
- [58] Solidity, „Solidity - Solidity 0.5.11 documentation“, 2019. [Online]. Verfügbar unter: <https://solidity.readthedocs.io/en/v0.5.11/>.
- [59] „Vyper-Vyper documentation“, 2019. [Online]. Verfügbar unter: <https://vyper.readthedocs.io/en/v0.1.0-beta.13/#>.

## 6 Anhang

### 6.1 Glossar

---

Begriff	Bedeutung
---------	-----------

---

---

### 6.2 Entwicklungsumgebung

In diesem Abschnitt wird die geplante Testumgebung und deren Verwendung beschrieben.

## 6.2.1 Blockchain

Es wird eine Test-Blockchain aufgesetzt. Diese wird benötigt, um geschriebenen Code zu testen und analysieren.

Als Blockchain wird Ethereum[9] verwendet. In den nachfolgenden Absätzen werden mögliche Tools besprochen, die für den Aufbau von einer Testumgebung genutzt werden können.

### 6.2.1.1 Client

In der Arbeit wird evaluiert ob Geth[32] als Client den Ansprüchen genügt oder ob ein anderer Client (z.B. Parity[33], Aleth[34], etc.) zum Einsatz kommt.

**Trufflesuite** Trufflesuite[53] wird verwendet, um eine simulierte Blockchain aufzusetzen. Diese kann für die Einarbeitung in die Materie genutzt werden.

## 6.2.2 Wallet

Wallets werden für die Verwaltung von Benutzerkonten und deren Transaktionen benötigt. Zu den möglichen Wallets gehören z.B.:

- uPort[54]
- Metamask[45]
- Atomic Wallet [55]
- Exodus[56]

Es wird davon ausgegangen, dass keine Wallet alle Bedürfnisse abdecken kann, daher wird die gewählte Wallet im Zuge dieses Projekts erweitert. Für Ethereum existiert ein offizieller Service um eine eigene Wallet zu erstellen: MyEtherWallet[57]

## 6.2.3 Smart Contracts

Smart Contracts werden benötigt, um zu bestimmen, wer auf einer Blockchain gratis Transaktionen ausführen kann. Sobald eigene Smart Contracts entwickelt werden, kann die Testumgebung genutzt werden, um diese zu testen.

**Programmiersprache** Für die Entwicklung von Smart Contracts werden folgende zwei Sprachen evaluiert:

- Solidity[58]
- Vyper[59]

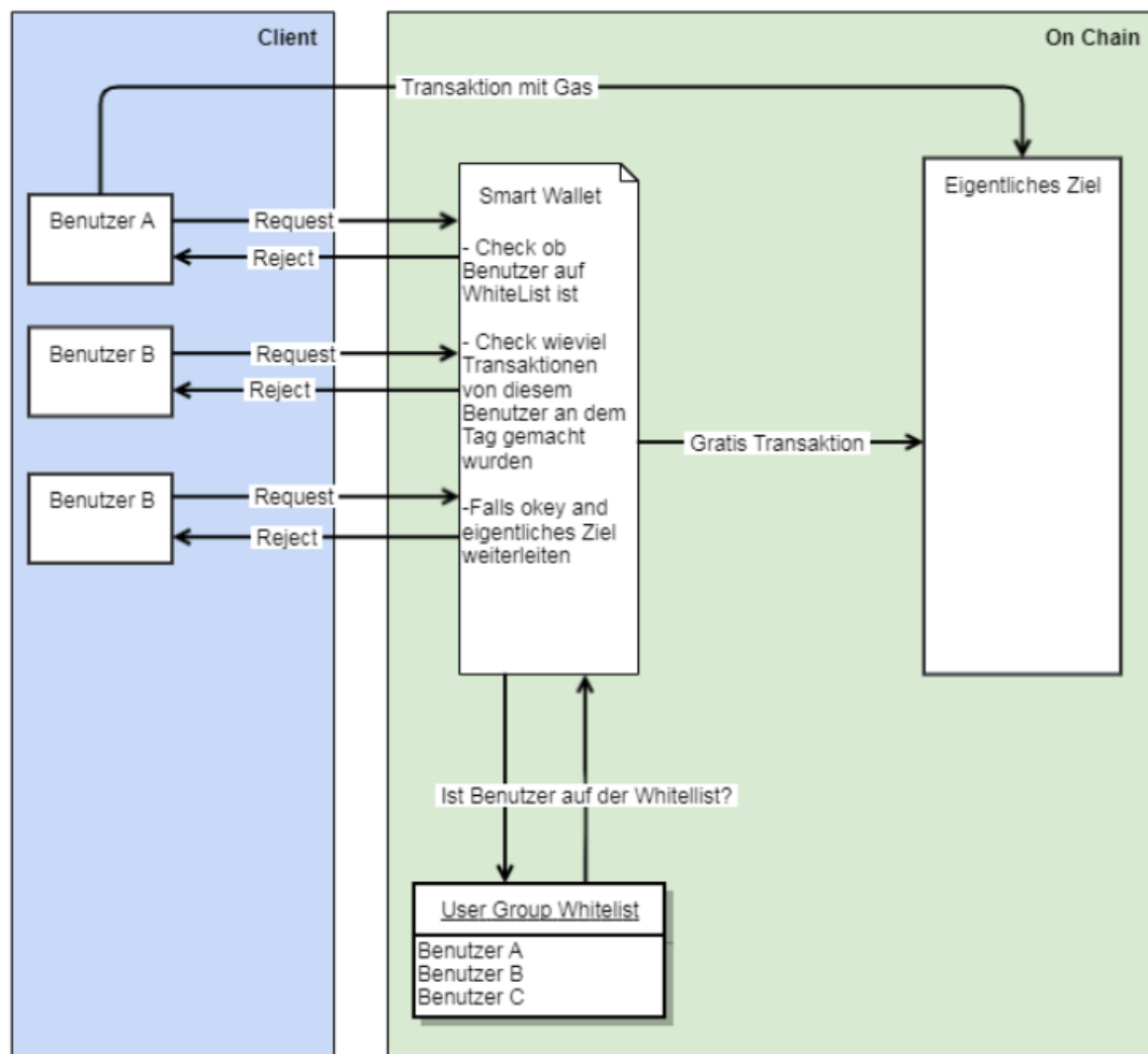
#### 6.2.4 Docker

```
docker run -ti -p 8545:8545 -p 8546:8546 -p 30303:30303 -p 30303:30303/u -v ~/.local/share/io.parity.ethereum/docker/:  
parity/parity:stable --config /home/parity/.local/share/io.parity.ethereum/docker.toml --jsonrpc-  
interface all
```

### 6.3 Weitere Lösungsansätze

#### 6.3.1 Super Smart Wallet

Es wird eine zentrale Smart Wallet entwickelt. Im Gegensatz zu LA 1, 2.4.1.1, wird nicht für jeden Benutzer eine Smart Wallet deployed, sondern nur eine einzige. Diese kann von allen Benutzern der Blockchain genutzt werden. Bei diesem Ansatz wird mit der in Absatz 2.3.1 beschriebenen Whitelist gearbeitet.



**Abbildung 6.1:** Super Smart Wallet

Die Smart Wallet verwaltet die Whitelist und den Schutzmechanismus gegen DoS Attacken. Das ist auf Abbildung 6.1 ersichtlich. Wird eine DoS Attacke identifiziert, wird der entsprechende Account aus der Whitelist gelöscht.

### 6.3.1.1 Pro

Es existiert nur eine einzige Smart Wallet. Das Deployment ist somit weniger aufwändig. Falls eine Änderung am Code gemacht nötig ist, muss nur eine Smart Wallet neu deployed werden.

### 6.3.1.2 Contra

Bei diesem Ansatz ist die Machbarkeit unklar. Parity muss umgeschrieben werden, da nicht die Sendidentität der Smart Wallet genutzt werden muss, sondern die des Benutzeraccounts. Ebenfalls muss die Whitelist-Funktionalität von Parity angepasst werden, analog zu LA 1.

## 6.4 Abnahmekriterien

In diesem Kapitel werden alle Abnahmekriterien des Blockchain Transaktions Managers aufgelistet und Kategorisiert. Es wird zwischen funktionalen und nicht-funktionales Kriterien unterschieden. //TODO Text

Nr.	Titel	Beschreibung
1.	Bezahlte Transaktionen für alle	Jeder gültige Account kann Transaktionen mit Gas Price durchführen
2.	Gratis Transaktionen für Whitelist	Ein Account der für die Whitelist zertifiziert ist, kann Transaktionen mit Gas Price „0“ durchführen
3.	Account aus Liste für Whitelist zertifizieren	Alle Account die auf der Liste stehen sind für die Whitelist zertifiziert

Nr.	Titel	Beschreibung
4.	Account aus Liste und Whitelist entfernen	Wenn ein Account gelöscht wird, wird er von der Whitelist wie auch von der Account Liste entfernt
5.	Account nach Transaktionen sperren	Ein Account der zu viele Transaktionen betätigt hat, wird für eine Zeitspanne gesperrt
6.	Account nach GasUsed sperren	Ein Account der zu viel Gas für seine Transaktionen benutzt hat wird gesperrt
7.	Gesperrte Account entsperren	Ein gesperrter Account wird nach einer gesetzten Zeitspanne wieder entsperrt
8.	Account manuell sperren	Ein Account kann manuell gesperrt werden

Nr.	Titel	Beschreibung
9.	Zeitintervall setzten	Es muss ein Zeitintervall gesetzt. Dieses gilt für die Zeitspanne wie lange Limiten und Sperrungen gelten.
10.	Sperrzeit	Die Sperrzeit ist Intervall Abhängig und gilt für alle gleich
11.	Anzahl Sperrungen speichern	Die Anzahl Sperrungen eines Accounts werden in einem File gespeichert
12.	Sperrzeit Abhängig von Anzahl Sperrungen	Die Sperrzeit wird höher, je öfter der Account gesperrt wurde
13.	Counter resettet	Der Counter aller Accounts wird nach Zeitperiode wieder auf 0 gesetzt
14.	Transaktionslimite pro User	Die Anzahl Transaktionen bis ein Account gesperrt wird, kann für jeden Account individuell eingestellt werden



Nr.	Titel	Beschreibung
15.	GasUsed Limite pro User	Die Anzahl Gas Used bis ein Account gesperrt wird, kann für jeden Account individuell eingestellt werden
16.	Default Werte	Es können default Werte für max Transaktionen und max Gas Used gesetzt werden
17.	Certifier nur von Owner deployen	Der Certifier kann nur vom Owner registriert werden
18.	Certifier nur einmal deploybar	Der Certifier kann nur einmal deplyt/regis-triert werden
19.	Owner Account	Der Certifier Owner Account kann nicht gesperrt werden
20.		
21.		

## 6.5 Abnahme Tests Report

### 6.5.1 Abnahme Test 1

<b>AK Nr.:</b>	<b>Titel:</b>	<b>Testart:</b>
<b>Tester:</b>	<b>Datum:</b>	<b>Status</b>
<b>Vorbedingung:</b>		
<b>Ablauf:</b>		
<b>Erwünschtes Resultat:</b>		
<b>Tatsächliches Resultat</b>		

### 6.5.2 Abnahme Test 2

### 6.5.3 Abnahme Test 3

### 6.5.4 Abnahme Test 4

### 6.5.5 Abnahme Test 5

### 6.5.6 Abnahme Test 6

### 6.5.7 Abnahme Test 7

### 6.5.8 Abnahme Test 8

### 6.5.9 Abnahme Test 9

## 6.6 Registry

### 6.6.1 ABI

```
1  [  
2    {"constant":false,"inputs":[{"name":"_new","type":"address"}],"name  
3    ":"setOwner","outputs":[],"payable":false,"type":"function"},  
    {"constant":false,"inputs":[{"name":"_who","type":"address"}],"name  
      ":"certify","outputs":[],"payable":false,"type":"function"},
```

```

4      {"constant":true,"inputs":[{"name":"_who","type":"address"}, {"name":
      :"_field","type":"string"}], "name":"getAddress", "outputs":[{"name":
      :"" , "type":"address"}], "payable":false, "type":"function"},
5      {"constant":false,"inputs":[{"name":"_who","type":"address"}], "name":
      :":revoke", "outputs":[], "payable":false, "type":"function"},
6      {"constant":true,"inputs":[], "name":"owner", "outputs":[{"name":"","
      :":type":"address"}], "payable":false, "type":"function"},
7      {"constant":true,"inputs":[], "name":"delegate", "outputs":[{"name":":
      :":type":"address"}], "payable":false, "type":"function"},
8      {"constant":true,"inputs":[{"name":"_who","type":"address"}, {"name":
      :":_field","type":"string"}], "name":"getUint", "outputs":[{"name":
      :": "" , "type":"uint256"}], "payable":false, "type":"function"},
9      {"constant":false,"inputs":[{"name":"_new","type":"address"}], "name":
      :":setDelegate", "outputs":[], "payable":false, "type":"function"},
10     {"constant":true,"inputs":[{"name":"_who","type":"address"}], "name":
      :":certified", "outputs":[{"name":"","type":"bool"}], "payable":
      :false, "type":"function"},
11     {"constant":true,"inputs":[{"name":"_who","type":"address"}, {"name":
      :":_field","type":"string"}], "name":"get", "outputs":[{"name":"","
      :":type":"bytes32"}], "payable":false, "type":"function"}
12 ]

```

### 6.6.2 Owned.sol

```

1  /// The owned contract.
2  ///
3  /// Copyright 2016 Gavin Wood, Parity Technologies Ltd.
4  ///
5  /// Licensed under the Apache License, Version 2.0 (the "License");
6  /// you may not use this file except in compliance with the License.
7  /// You may obtain a copy of the License at
8  ///
9  ///     http://www.apache.org/licenses/LICENSE-2.0
10 ///
11 /// Unless required by applicable law or agreed to in writing, software
12 /// distributed under the License is distributed on an "AS IS" BASIS,
13 /// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
14 /// implied.
15 /// See the License for the specific language governing permissions and
16 /// limitations under the License.
17 pragma solidity ^0.4.24;
18
19
20 contract Owned {
21     event NewOwner(address indexed old, address indexed current);
22
23     address public owner = msg.sender;
24

```

```
25     modifier onlyOwner {
26         require(msg.sender == owner);
27         _;
28     }
29
30     function setOwner(address _new)
31         external
32         onlyOwner
33     {
34         emit NewOwner(owner, _new);
35         owner = _new;
36     }
37 }
```

### 6.6.3 Registry.sol

```
1  ///! The registry interface.
2  ///!
3  ///! Copyright 2016 Gavin Wood, Parity Technologies Ltd.
4  ///!
5  ///! Licensed under the Apache License, Version 2.0 (the "License");
6  ///! you may not use this file except in compliance with the License.
7  ///! You may obtain a copy of the License at
8  ///!
9  ///!     http://www.apache.org/licenses/LICENSE-2.0
10 ///!
11 ///! Unless required by applicable law or agreed to in writing, software
12 ///! distributed under the License is distributed on an "AS IS" BASIS,
13 ///! WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
14 ///! implied.
15 ///! See the License for the specific language governing permissions and
16 ///! limitations under the License.
17
18 pragma solidity ^0.4.24;
19
20 interface MetadataRegistry {
21     event DataChanged(bytes32 indexed name, string key, string plainKey
22         );
23
24     function getData(bytes32 _name, string _key)
25         external
26         view
27         returns (bytes32);
28
29     function getAddress(bytes32 _name, string _key)
30         external
31         view
32         returns (address);
```

```
32
33     function getUint(bytes32 _name, string _key)
34         external
35         view
36         returns (uint);
37 }
38
39
40 interface OwnerRegistry {
41     event Reserved(bytes32 indexed name, address indexed owner);
42     event Transferred(bytes32 indexed name, address indexed oldOwner,
43         address indexed newOwner);
44     event Dropped(bytes32 indexed name, address indexed owner);
45
46     function getOwner(bytes32 _name)
47         external
48         view
49         returns (address);
50 }
51
52 interface ReverseRegistry {
53     event ReverseConfirmed(string name, address indexed reverse);
54     event ReverseRemoved(string name, address indexed reverse);
55
56     function hasReverse(bytes32 _name)
57         external
58         view
59         returns (bool);
60
61     function getReverse(bytes32 _name)
62         external
63         view
64         returns (address);
65
66     function canReverse(address _data)
67         external
68         view
69         returns (bool);
70
71     function reverse(address _data)
72         external
73         view
74         returns (string);
75 }
```

#### 6.6.4 SimpleRegistry.sol

```
1 //! The simple registry contract.
```

```
2  ///
3  /// Copyright 2016 Gavin Wood, Parity Technologies Ltd.
4  ///
5  /// Licensed under the Apache License, Version 2.0 (the "License");
6  /// you may not use this file except in compliance with the License.
7  /// You may obtain a copy of the License at
8  ///
9  ///     http://www.apache.org/licenses/LICENSE-2.0
10 ///
11 /// Unless required by applicable law or agreed to in writing, software
12 /// distributed under the License is distributed on an "AS IS" BASIS,
13 /// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
14 /// implied.
15 /// See the License for the specific language governing permissions and
16 /// limitations under the License.
17
18 pragma solidity ^0.4.24;
19
20 import "./Owned.sol";
21 import "./Registry.sol";
22
23 contract SimpleRegistry is Owned, MetadataRegistry, OwnerRegistry,
24     ReverseRegistry {
25     struct Entry {
26         address owner;
27         address reverse;
28         bool deleted;
29         mapping (string => bytes32) data;
30     }
31
32     event Drained(uint amount);
33     event FeeChanged(uint amount);
34     event ReverseProposed(string name, address indexed reverse);
35
36     mapping (bytes32 => Entry) entries;
37     mapping (address => string) reverses;
38
39     uint public fee = 1 ether;
40
41     modifier whenUnreserved(bytes32 _name) {
42         require(!entries[_name].deleted && entries[_name].owner == 0);
43         _;
44     }
45
46     modifier onlyOwnerOf(bytes32 _name) {
47         require(entries[_name].owner == msg.sender);
48         _;
49     }
50
51     modifier whenProposed(string _name) {
```

```
51     require(entries[keccak256(bytes(_name))].reverse == msg.sender)
52     _;
53 }
54
55 modifier whenEntry(string _name) {
56     require(
57         !entries[keccak256(bytes(_name))].deleted &&
58         entries[keccak256(bytes(_name))].owner != address(0)
59     );
60     _;
61 }
62
63 modifier whenEntryRaw(bytes32 _name) {
64     require(
65         !entries[_name].deleted &&
66         entries[_name].owner != address(0)
67     );
68     _;
69 }
70
71 modifier whenFeePaid {
72     require(msg.value >= fee);
73     _;
74 }
75
76 // Reservation functions
77 function reserve(bytes32 _name)
78     external
79     payable
80     whenUnreserved(_name)
81     whenFeePaid
82     returns (bool success)
83 {
84     entries[_name].owner = msg.sender;
85     emit Reserved(_name, msg.sender);
86     return true;
87 }
88
89 function transfer(bytes32 _name, address _to)
90     external
91     whenEntryRaw(_name)
92     onlyOwnerOf(_name)
93     returns (bool success)
94 {
95     entries[_name].owner = _to;
96     emit Transferred(_name, msg.sender, _to);
97     return true;
98 }
99
100 function drop(bytes32 _name)
```

```
101     external
102     whenEntryRaw(_name)
103     onlyOwnerOf(_name)
104     returns (bool success)
105 {
106     if (keccak256(bytes(reverses[entries[_name].reverse])) == _name
107         ) {
108         emit ReverseRemoved(reverses[entries[_name].reverse],
109             entries[_name].reverse);
110         delete reverses[entries[_name].reverse];
111     }
112     entries[_name].deleted = true;
113     emit Dropped(_name, msg.sender);
114     return true;
115 }
116
117 // Data admin functions
118 function setData(bytes32 _name, string _key, bytes32 _value)
119     external
120     whenEntryRaw(_name)
121     onlyOwnerOf(_name)
122     returns (bool success)
123 {
124     entries[_name].data[_key] = _value;
125     emit DataChanged(_name, _key, _key);
126     return true;
127 }
128
129 function setAddress(bytes32 _name, string _key, address _value)
130     external
131     whenEntryRaw(_name)
132     onlyOwnerOf(_name)
133     returns (bool success)
134 {
135     entries[_name].data[_key] = bytes32(_value);
136     emit DataChanged(_name, _key, _key);
137     return true;
138 }
139
140 function setUint(bytes32 _name, string _key, uint _value)
141     external
142     whenEntryRaw(_name)
143     onlyOwnerOf(_name)
144     returns (bool success)
145 {
146     entries[_name].data[_key] = bytes32(_value);
147     emit DataChanged(_name, _key, _key);
148     return true;
149 }
150
151 // Reverse registration functions
```



```
150     function proposeReverse(string _name, address _who)
151         external
152         whenEntry(_name)
153         onlyOwnerOf(keccak256(bytes(_name)))
154         returns (bool success)
155     {
156         bytes32 sha3Name = keccak256(bytes(_name));
157         if (entries[sha3Name].reverse != 0 && keccak256(bytes(reverses[
158             entries[sha3Name].reverse])) == sha3Name) {
159             delete reverses[entries[sha3Name].reverse];
160             emit ReverseRemoved(_name, entries[sha3Name].reverse);
161         }
162         entries[sha3Name].reverse = _who;
163         emit ReverseProposed(_name, _who);
164         return true;
165     }
166     function confirmReverse(string _name)
167         external
168         whenEntry(_name)
169         whenProposed(_name)
170         returns (bool success)
171     {
172         reverses[msg.sender] = _name;
173         emit ReverseConfirmed(_name, msg.sender);
174         return true;
175     }
176
177     function confirmReverseAs(string _name, address _who)
178         external
179         whenEntry(_name)
180         onlyOwner
181         returns (bool success)
182     {
183         reverses[_who] = _name;
184         emit ReverseConfirmed(_name, _who);
185         return true;
186     }
187
188     function removeReverse()
189         external
190         whenEntry(reverses[msg.sender])
191     {
192         emit ReverseRemoved(reverses[msg.sender], msg.sender);
193         delete entries[keccak256(bytes(reverses[msg.sender]))].reverse;
194         delete reverses[msg.sender];
195     }
196
197     // Admin functions for the owner
198     function setFee(uint _amount)
199         external
```

```
200     onlyOwner
201     returns (bool)
202     {
203         fee = _amount;
204         emit FeeChanged(_amount);
205         return true;
206     }
207
208     function drain()
209         external
210         onlyOwner
211         returns (bool)
212     {
213         emit Drained(address(this).balance);
214         msg.sender.transfer(address(this).balance);
215         return true;
216     }
217
218     // MetadataRegistry views
219     function getData(bytes32 _name, string _key)
220         external
221         view
222         whenEntryRaw(_name)
223         returns (bytes32)
224     {
225         return entries[_name].data[_key];
226     }
227
228     function getAddress(bytes32 _name, string _key)
229         external
230         view
231         whenEntryRaw(_name)
232         returns (address)
233     {
234         return address(entries[_name].data[_key]);
235     }
236
237     function getUint(bytes32 _name, string _key)
238         external
239         view
240         whenEntryRaw(_name)
241         returns (uint)
242     {
243         return uint(entries[_name].data[_key]);
244     }
245
246     // OwnerRegistry views
247     function getOwner(bytes32 _name)
248         external
249         view
250         whenEntryRaw(_name)
```

```
251     returns (address)
252     {
253         return entries[_name].owner;
254     }
255
256     // ReversibleRegistry views
257     function hasReverse(bytes32 _name)
258         external
259         view
260         whenEntryRaw(_name)
261         returns (bool)
262     {
263         return entries[_name].reverse != 0;
264     }
265
266     function getReverse(bytes32 _name)
267         external
268         view
269         whenEntryRaw(_name)
270         returns (address)
271     {
272         return entries[_name].reverse;
273     }
274
275     function canReverse(address _data)
276         external
277         view
278         returns (bool)
279     {
280         return bytes(reverses[_data]).length != 0;
281     }
282
283     function reverse(address _data)
284         external
285         view
286         returns (string)
287     {
288         return reverses[_data];
289     }
290
291     function reserved(bytes32 _name)
292         external
293         view
294         whenEntryRaw(_name)
295         returns (bool)
296     {
297         return entries[_name].owner != 0;
298     }
299 }
```

### 6.6.5 Java-Wrapper für SimpleRegistry

```
1 package ch.brugg.fhnw.btm.contracts;
2
3 import io.reactivex.Flowable;
4 import org.web3j.abi.EventEncoder;
5 import org.web3j.abi.TypeReference;
6 import org.web3j.abi.datatypes.*;
7 import org.web3j.abi.datatypes.generated.Bytes32;
8 import org.web3j.abi.datatypes.generated.Uint256;
9 import org.web3j.crypto.Credentials;
10 import org.web3j.protocol.Web3j;
11 import org.web3j.protocol.core.DefaultBlockParameter;
12 import org.web3j.protocol.core.RemoteCall;
13 import org.web3j.protocol.core.RemoteFunctionCall;
14 import org.web3j.protocol.core.methods.request.EthFilter;
15 import org.web3j.protocol.core.methods.response.BaseEventResponse;
16 import org.web3j.protocol.core.methods.response.Log;
17 import org.web3j.protocol.core.methods.response.TransactionReceipt;
18 import org.web3j.tx.Contract;
19 import org.web3j.tx.TransactionManager;
20 import org.web3j.tx.gas.ContractGasProvider;
21
22 import java.math.BigInteger;
23 import java.util.ArrayList;
24 import java.util.Arrays;
25 import java.util.Collections;
26 import java.util.List;
27
28 /**
29  * <p>Auto generated code.
30  * <p><strong>Do not modify!</strong>
31  * <p>Please use the <a href="https://docs.web3j.io/command_line.html">
32  *   web3j command line tools</a>,
33  * or the org.web3j.codegen.SolidityFunctionWrapperGenerator in the
34  * <a href="https://github.com/web3j/web3j/tree/master/codegen">codegen
35  * module</a> to update.
36  *
37  * <p>Generated with web3j version 4.5.11.
38  *
39  * Generated with:
40  * web3j solidity generate -b .\src\main\resources\solidity\Registry\
41  *   out\SimpleRegistry.bin -a .\src\main\resources\solidity\Registry\
42  *   out\SimpleRegistry.abi -o .\src\main\java -p io.kauri.tutorials.
43  *   java_ethereum.contracts
44  */
45 @SuppressWarnings("rawtypes")
46 public class SimpleRegistry extends Contract {
47     public static final String BINARY = "Platzhalter für BinaryCode";
48
49     public static final String FUNC_CANREVERSE = "canReverse";
```

```
45
46     public static final String FUNC_SETOWNER = "setOwner";
47
48     public static final String FUNC_SETDATA = "setData";
49
50     public static final String FUNC_CONFIRMREVERSE = "confirmReverse";
51
52     public static final String FUNC_RESERVE = "reserve";
53
54     public static final String FUNC_DROP = "drop";
55
56     public static final String FUNC_GETADDRESS = "getAddress";
57
58     public static final String FUNC_SETFEE = "setFee";
59
60     public static final String FUNC_TRANSFER = "transfer";
61
62     public static final String FUNC_OWNER = "owner";
63
64     public static final String FUNC_GETDATA = "getData";
65
66     public static final String FUNC_RESERVED = "reserved";
67
68     public static final String FUNC_DRAIN = "drain";
69
70     public static final String FUNC_PROPOSEREVERSE = "proposeReverse";
71
72     public static final String FUNC_HASREVERSE = "hasReverse";
73
74     public static final String FUNC_GETUINT = "getUInt";
75
76     public static final String FUNC_FEE = "fee";
77
78     public static final String FUNC_GETOWNER = "getOwner";
79
80     public static final String FUNC_GETREVERSE = "getReverse";
81
82     public static final String FUNC_REVERSE = "reverse";
83
84     public static final String FUNC_SETUINT = "setUInt";
85
86     public static final String FUNC_CONFIRMREVERSEAS = "
    confirmReverseAs";
87
88     public static final String FUNC_REMOVEVERSE = "removeReverse";
89
90     public static final String FUNC_SETADDRESS = "setAddress";
91
92     public static final Event DRAINED_EVENT = new Event("Drained",
93         Arrays.<TypeReference<?>>asList(new TypeReference<UInt256
94             >() {}));
```

```
94     ;
95
96     public static final Event FEECHANGED_EVENT = new Event("FeeChanged"
97         ,
98         Arrays.<TypeReference<?>>asList(new TypeReference<Uint256
99             >() {}));
100     ;
101
102     public static final Event REVERSEPROPOSED_EVENT = new Event("
103         ReverseProposed",
104         Arrays.<TypeReference<?>>asList(new TypeReference<
105             Utf8String>() {}, new TypeReference<Address>(true) {}));
106     ;
107
108     public static final Event REVERSECONFIRMED_EVENT = new Event("
109         ReverseConfirmed",
110         Arrays.<TypeReference<?>>asList(new TypeReference<
111             Utf8String>() {}, new TypeReference<Address>(true) {}));
112     ;
113
114     public static final Event REVERSEREMOVED_EVENT = new Event("
115         ReverseRemoved",
116         Arrays.<TypeReference<?>>asList(new TypeReference<
117             Utf8String>() {}, new TypeReference<Address>(true) {}));
118     ;
119
120     public static final Event RESERVED_EVENT = new Event("Reserved",
121         Arrays.<TypeReference<?>>asList(new TypeReference<Bytes32>(
122             true) {}, new TypeReference<Address>(true) {}));
123     ;
124
125     public static final Event TRANSFERRED_EVENT = new Event("
126         Transferred",
127         Arrays.<TypeReference<?>>asList(new TypeReference<Bytes32>(
128             true) {}, new TypeReference<Address>(true) {}, new
129             TypeReference<Address>(true) {}));
130     ;
131
132     public static final Event DROPPED_EVENT = new Event("Dropped",
133         Arrays.<TypeReference<?>>asList(new TypeReference<Bytes32>(
134             true) {}, new TypeReference<Address>(true) {}));
135     ;
136
137     public static final Event DATACHANGED_EVENT = new Event("
138         DataChanged",
139         Arrays.<TypeReference<?>>asList(new TypeReference<Bytes32>(
140             true) {}, new TypeReference<Utf8String>() {}, new
141             TypeReference<Utf8String>() {}));
142     ;
143
144     public static final Event NEWOWNER_EVENT = new Event("NewOwner",
```

```

129         Arrays.<TypeReference<?>>asList(new TypeReference<Address>(
130             true) {}, new TypeReference<Address>(true) {}));
131     ;
132     @Deprecated
133     protected SimpleRegistry(String contractAddress, Web3j web3j,
134         Credentials credentials, BigInteger gasPrice, BigInteger
135         gasLimit) {
136         super(BINARY, contractAddress, web3j, credentials, gasPrice,
137             gasLimit);
138     }
139     protected SimpleRegistry(String contractAddress, Web3j web3j,
140         Credentials credentials, ContractGasProvider contractGasProvider
141         ) {
142         super(BINARY, contractAddress, web3j, credentials,
143             contractGasProvider);
144     }
145     @Deprecated
146     protected SimpleRegistry(String contractAddress, Web3j web3j,
147         TransactionManager transactionManager, BigInteger gasPrice,
148         BigInteger gasLimit) {
149         super(BINARY, contractAddress, web3j, transactionManager,
150             gasPrice, gasLimit);
151     }
152     protected SimpleRegistry(String contractAddress, Web3j web3j,
153         TransactionManager transactionManager, ContractGasProvider
154         contractGasProvider) {
155         super(BINARY, contractAddress, web3j, transactionManager,
156             contractGasProvider);
157     }
158     public RemoteFunctionCall<Boolean> canReverse(String _data) {
159         final Function function = new Function(FUNC_CANREVERSE,
160             Arrays.<Type>asList(new Address(160, _data)),
161             Arrays.<TypeReference<?>>asList(new TypeReference<Bool
162                 >() {}));
163         return executeRemoteCallSingleValueReturn(function, Boolean.
164             class);
165     }
166     public RemoteFunctionCall<TransactionReceipt> setOwner(String _new)
167     {
168         final Function function = new Function(
169             FUNC_SETOWNER,
170             Arrays.<Type>asList(new Address(160, _new)),
171             Collections.<TypeReference<?>>emptyList());
172         return executeRemoteCallTransaction(function);
173     }

```

```

164
165     public RemoteFunctionCall<TransactionReceipt> setData(byte[] _name,
166         String _key, byte[] _value) {
167         final Function function = new Function(
168             FUNC_SETDATA,
169             Arrays.<Type>asList(new Bytes32(_name),
170                 new Utf8String(_key),
171                 new Bytes32(_value)),
172             Collections.<TypeReference<?>>emptyList());
173         return executeRemoteCallTransaction(function);
174     }
175     public RemoteFunctionCall<TransactionReceipt> confirmReverse(String
176         _name) {
177         final Function function = new Function(
178             FUNC_CONFIRMREVERSE,
179             Arrays.<Type>asList(new Utf8String(_name)),
180             Collections.<TypeReference<?>>emptyList());
181         return executeRemoteCallTransaction(function);
182     }
183     public RemoteFunctionCall<TransactionReceipt> reserve(byte[] _name,
184         BigInteger weiValue) {
185         final Function function = new Function(
186             FUNC_RESERVE,
187             Arrays.<Type>asList(new Bytes32(_name)),
188             Collections.<TypeReference<?>>emptyList());
189         return executeRemoteCallTransaction(function, weiValue);
190     }
191     public RemoteFunctionCall<TransactionReceipt> drop(byte[] _name) {
192         final Function function = new Function(
193             FUNC_DROP,
194             Arrays.<Type>asList(new Bytes32(_name)),
195             Collections.<TypeReference<?>>emptyList());
196         return executeRemoteCallTransaction(function);
197     }
198
199     public RemoteFunctionCall<String> getAddress(byte[] _name, String
200         _key) {
201         final Function function = new Function(FUNC_GETADDRESS,
202             Arrays.<Type>asList(new Bytes32(_name),
203                 new Utf8String(_key)),
204             Arrays.<TypeReference<?>>asList(new TypeReference<
205                 Address>() {}));
206         return executeRemoteCallSingleValueReturn(function, String.
207             class);
208     }
209
210     public RemoteFunctionCall<TransactionReceipt> setFee(BigInteger
211         _amount) {

```



```
208     final Function function = new Function(
209         FUNC_SETFEE,
210         Arrays.<Type>asList(new Uint256(_amount)),
211         Collections.<TypeReference<?>>emptyList());
212     return executeRemoteCallTransaction(function);
213 }
214
215 public RemoteFunctionCall<TransactionReceipt> transfer(byte[] _name
, String _to) {
216     final Function function = new Function(
217         FUNC_TRANSFER,
218         Arrays.<Type>asList(new Bytes32(_name),
219             new Address(160, _to)),
220         Collections.<TypeReference<?>>emptyList());
221     return executeRemoteCallTransaction(function);
222 }
223
224 public RemoteFunctionCall<String> owner() {
225     final Function function = new Function(FUNC_OWNER,
226         Arrays.<Type>asList(),
227         Arrays.<TypeReference<?>>asList(new TypeReference<
228             Address>() {}));
229     return executeRemoteCallSingleValueReturn(function, String.
230         class);
231 }
232
233 public RemoteFunctionCall<byte[]> getData(byte[] _name, String _key
) {
234     final Function function = new Function(FUNC_GETDATA,
235         Arrays.<Type>asList(new Bytes32(_name),
236             new Utf8String(_key)),
237         Arrays.<TypeReference<?>>asList(new TypeReference<
238             Bytes32>() {}));
239     return executeRemoteCallSingleValueReturn(function, byte[]
240         .class);
241 }
242
243 public RemoteFunctionCall<Boolean> reserved(byte[] _name) {
244     final Function function = new Function(FUNC_RESERVED,
245         Arrays.<Type>asList(new Bytes32(_name)),
246         Arrays.<TypeReference<?>>asList(new TypeReference<Bool
247             >() {}));
248     return executeRemoteCallSingleValueReturn(function, Boolean.
249         class);
250 }
251
252 public RemoteFunctionCall<TransactionReceipt> drain() {
253     final Function function = new Function(
254         FUNC_DRAIN,
255         Arrays.<Type>asList(),
256         Collections.<TypeReference<?>>emptyList());
```

```
251         return executeRemoteCallTransaction(function);
252     }
253
254     public RemoteFunctionCall<TransactionReceipt> proposeReverse(String
        _name, String _who) {
255         final Function function = new Function(
256             FUNC_PROPOSEREVERSE,
257             Arrays.<Type>asList(new Utf8String(_name),
258                 new Address(160, _who)),
259             Collections.<TypeReference<?>>emptyList());
260         return executeRemoteCallTransaction(function);
261     }
262
263     public RemoteFunctionCall<Boolean> hasReverse(byte[] _name) {
264         final Function function = new Function(FUNC_HASREVERSE,
265             Arrays.<Type>asList(new Bytes32(_name)),
266             Arrays.<TypeReference<?>>asList(new TypeReference<Boolean>() {}));
267         return executeRemoteCallSingleValueReturn(function, Boolean.
            class);
268     }
269
270     public RemoteFunctionCall<BigInteger> getUint(byte[] _name, String
        _key) {
271         final Function function = new Function(FUNC_GETUINT,
272             Arrays.<Type>asList(new Bytes32(_name),
273                 new Utf8String(_key)),
274             Arrays.<TypeReference<?>>asList(new TypeReference<
                Uint256>() {}));
275         return executeRemoteCallSingleValueReturn(function, BigInteger.
            class);
276     }
277
278     public RemoteFunctionCall<BigInteger> fee() {
279         final Function function = new Function(FUNC_FEE,
280             Arrays.<Type>asList(),
281             Arrays.<TypeReference<?>>asList(new TypeReference<
                Uint256>() {}));
282         return executeRemoteCallSingleValueReturn(function, BigInteger.
            class);
283     }
284
285     public RemoteFunctionCall<String> getOwner(byte[] _name) {
286         final Function function = new Function(FUNC_GETOWNER,
287             Arrays.<Type>asList(new Bytes32(_name)),
288             Arrays.<TypeReference<?>>asList(new TypeReference<
                Address>() {}));
289         return executeRemoteCallSingleValueReturn(function, String.
            class);
290     }
291 }
```

```
292     public RemoteFunctionCall<String> getReverse(byte[] _name) {
293         final Function function = new Function(FUNC_GETREVERSE,
294             Arrays.<Type>asList(new Bytes32(_name)),
295             Arrays.<TypeReference<?>>asList(new TypeReference<
296                 Address>() {}));
297         return executeRemoteCallSingleValueReturn(function, String.
298             class);
299     }
300     public RemoteFunctionCall<String> reverse(String _data) {
301         final Function function = new Function(FUNC_REVERSE,
302             Arrays.<Type>asList(new Address(160, _data)),
303             Arrays.<TypeReference<?>>asList(new TypeReference<
304                 Utf8String>() {}));
305         return executeRemoteCallSingleValueReturn(function, String.
306             class);
307     }
308     public RemoteFunctionCall<TransactionReceipt> setUint(byte[] _name,
309         String _key, BigInteger _value) {
310         final Function function = new Function(
311             FUNC_SETUINT,
312             Arrays.<Type>asList(new Bytes32(_name),
313                 new Utf8String(_key),
314                 new Uint256(_value)),
315             Collections.<TypeReference<?>>emptyList());
316         return executeRemoteCallTransaction(function);
317     }
318     public RemoteFunctionCall<TransactionReceipt> confirmReverseAs(
319         String _name, String _who) {
320         final Function function = new Function(
321             FUNC_CONFIRMREVERSEAS,
322             Arrays.<Type>asList(new Utf8String(_name),
323                 new Address(160, _who)),
324             Collections.<TypeReference<?>>emptyList());
325         return executeRemoteCallTransaction(function);
326     }
327     public RemoteFunctionCall<TransactionReceipt> removeReverse() {
328         final Function function = new Function(
329             FUNC_REMOVEVERSE,
330             Arrays.<Type>asList(),
331             Collections.<TypeReference<?>>emptyList());
332         return executeRemoteCallTransaction(function);
333     }
334     public RemoteFunctionCall<TransactionReceipt> setAddress(byte[]
335         _name, String _key, String _value) {
336         final Function function = new Function(
337             FUNC_SETADDRESS,
```

```
336         Arrays.<Type>asList(new Bytes32(_name),
337         new Utf8String(_key),
338         new Address(160, _value)),
339         Collections.<TypeReference<?>>emptyList());
340     return executeRemoteCallTransaction(function);
341 }
342
343 public List<DrainedEventResponse> getDrainedEvents(
344     TransactionReceipt transactionReceipt) {
345     List<EventValuesWithLog> valueList =
346         extractEventParametersWithLog(DRAINED_EVENT,
347         transactionReceipt);
348     ArrayList<DrainedEventResponse> responses = new ArrayList<
349         DrainedEventResponse>(valueList.size());
350     for (EventValuesWithLog eventValues : valueList) {
351         DrainedEventResponse typedResponse = new
352         DrainedEventResponse();
353         typedResponse.log = eventValues.getLog();
354         typedResponse.amount = (BigInteger) eventValues.
355             getNonIndexedValues().get(0).getValue();
356         responses.add(typedResponse);
357     }
358     return responses;
359 }
360
361 public Flowable<DrainedEventResponse> drainedEventFlowable(
362     EthFilter filter) {
363     return web3j.ethLogFlowable(filter).map(new io.reactivex.
364         functions.Function<Log, DrainedEventResponse>() {
365         @Override
366         public DrainedEventResponse apply(Log log) {
367             EventValuesWithLog eventValues =
368                 extractEventParametersWithLog(DRAINED_EVENT, log);
369             DrainedEventResponse typedResponse = new
370                 DrainedEventResponse();
371             typedResponse.log = log;
372             typedResponse.amount = (BigInteger) eventValues.
373                 getNonIndexedValues().get(0).getValue();
374             return typedResponse;
375         }
376     });
377 }
378
379 public Flowable<DrainedEventResponse> drainedEventFlowable(
380     DefaultBlockParameter startBlock, DefaultBlockParameter endBlock
381 ) {
382     EthFilter filter = new EthFilter(startBlock, endBlock,
383         getContractAddress());
384     filter.addSingleTopic(EventEncoder.encode(DRAINED_EVENT));
385     return drainedEventFlowable(filter);
386 }
```

```
373
374     public List<FeeChangedEventResponse> getFeeChangedEvents(
375         TransactionReceipt transactionReceipt) {
376         List<EventValuesWithLog> valueList =
377             extractEventParametersWithLog(FEECHANGED_EVENT,
378                 transactionReceipt);
379         ArrayList<FeeChangedEventResponse> responses = new ArrayList<
380             FeeChangedEventResponse>(valueList.size());
381         for (EventValuesWithLog eventValues : valueList) {
382             FeeChangedEventResponse typedResponse = new
383                 FeeChangedEventResponse();
384             typedResponse.log = eventValues.getLog();
385             typedResponse.amount = (BigInteger) eventValues.
386                 getNonIndexedValues().get(0).getValue();
387             responses.add(typedResponse);
388         }
389         return responses;
390     }
391
392     public Flowable<FeeChangedEventResponse> feeChangedEventFlowable(
393         EthFilter filter) {
394         return web3j.ethLogFlowable(filter).map(new io.reactivex.
395             functions.Function<Log, FeeChangedEventResponse>() {
396                 @Override
397                 public FeeChangedEventResponse apply(Log log) {
398                     EventValuesWithLog eventValues =
399                         extractEventParametersWithLog(FEECHANGED_EVENT, log)
400                         ;
401                     FeeChangedEventResponse typedResponse = new
402                         FeeChangedEventResponse();
403                     typedResponse.log = log;
404                     typedResponse.amount = (BigInteger) eventValues.
405                         getNonIndexedValues().get(0).getValue();
406                     return typedResponse;
407                 }
408             });
409     }
410
411     public Flowable<FeeChangedEventResponse> feeChangedEventFlowable(
412         DefaultBlockParameter startBlock, DefaultBlockParameter endBlock
413     ) {
414         EthFilter filter = new EthFilter(startBlock, endBlock,
415             getContractAddress());
416         filter.addSingleTopic(EventEncoder.encode(FEECHANGED_EVENT));
417         return feeChangedEventFlowable(filter);
418     }
419
420     public List<ReverseProposedEventResponse> getReverseProposedEvents(
421         TransactionReceipt transactionReceipt) {
422         List<EventValuesWithLog> valueList =
423             extractEventParametersWithLog(REVERSEPROPOSED_EVENT,
```

```

transactionReceipt);
407     ArrayList<ReverseProposedEventResponse> responses = new
        ArrayList<ReverseProposedEventResponse>(valueList.size());
408     for (EventValuesWithLog eventValues : valueList) {
409         ReverseProposedEventResponse typedResponse = new
            ReverseProposedEventResponse();
410         typedResponse.log = eventValues.getLog();
411         typedResponse.reverse = (String) eventValues.
            getIndexedValues().get(0).getValue();
412         typedResponse.name = (String) eventValues.
            getNonIndexedValues().get(0).getValue();
413         responses.add(typedResponse);
414     }
415     return responses;
416 }
417
418 public Flowable<ReverseProposedEventResponse>
    reverseProposedEventFlowable(EthFilter filter) {
419     return web3j.ethLogFlowable(filter).map(new io.reactivex.
        functions.Function<Log, ReverseProposedEventResponse>() {
420         @Override
421         public ReverseProposedEventResponse apply(Log log) {
422             EventValuesWithLog eventValues =
                extractEventParametersWithLog(REVERSEPROPOSED_EVENT,
                    log);
423             ReverseProposedEventResponse typedResponse = new
                ReverseProposedEventResponse();
424             typedResponse.log = log;
425             typedResponse.reverse = (String) eventValues.
                getIndexedValues().get(0).getValue();
426             typedResponse.name = (String) eventValues.
                getNonIndexedValues().get(0).getValue();
427             return typedResponse;
428         }
429     });
430 }
431
432 public Flowable<ReverseProposedEventResponse>
    reverseProposedEventFlowable(DefaultBlockParameter startBlock,
        DefaultBlockParameter endBlock) {
433     EthFilter filter = new EthFilter(startBlock, endBlock,
        getContractAddress());
434     filter.addSingleTopic(EventEncoder.encode(REVERSEPROPOSED_EVENT
        ));
435     return reverseProposedEventFlowable(filter);
436 }
437
438 public List<ReverseConfirmedEventResponse>
    getReverseConfirmedEvents(TransactionReceipt transactionReceipt)
    {

```

```

439         List<EventValuesWithLog> valueList =
            extractEventParametersWithLog(REVERSECONFIRMED_EVENT,
                transactionReceipt);
440         ArrayList<ReverseConfirmedEventResponse> responses = new
            ArrayList<ReverseConfirmedEventResponse>(valueList.size());
441         for (EventValuesWithLog eventValues : valueList) {
442             ReverseConfirmedEventResponse typedResponse = new
                ReverseConfirmedEventResponse();
443             typedResponse.log = eventValues.getLog();
444             typedResponse.reverse = (String) eventValues.
                getIndexedValues().get(0).getValue();
445             typedResponse.name = (String) eventValues.
                getNonIndexedValues().get(0).getValue();
446             responses.add(typedResponse);
447         }
448         return responses;
449     }
450
451     public Flowable<ReverseConfirmedEventResponse>
        reverseConfirmedEventFlowable(EthFilter filter) {
452         return web3j.ethLogFlowable(filter).map(new io.reactivex.
            functions.Function<Log, ReverseConfirmedEventResponse>() {
453             @Override
454             public ReverseConfirmedEventResponse apply(Log log) {
455                 EventValuesWithLog eventValues =
                    extractEventParametersWithLog(REVERSECONFIRMED_EVENT
                        , log);
456                 ReverseConfirmedEventResponse typedResponse = new
                    ReverseConfirmedEventResponse();
457                 typedResponse.log = log;
458                 typedResponse.reverse = (String) eventValues.
                    getIndexedValues().get(0).getValue();
459                 typedResponse.name = (String) eventValues.
                    getNonIndexedValues().get(0).getValue();
460                 return typedResponse;
461             }
462         });
463     }
464
465     public Flowable<ReverseConfirmedEventResponse>
        reverseConfirmedEventFlowable(DefaultBlockParameter startBlock,
        DefaultBlockParameter endBlock) {
466         EthFilter filter = new EthFilter(startBlock, endBlock,
            getContractAddress());
467         filter.addSingleTopic(EventEncoder.encode(
            REVERSECONFIRMED_EVENT));
468         return reverseConfirmedEventFlowable(filter);
469     }
470
471     public List<ReverseRemovedEventResponse> getReverseRemovedEvents(
        TransactionReceipt transactionReceipt) {

```

```

472     List<EventValuesWithLog> valueList =
        extractEventParametersWithLog(REVERSEREMOVED_EVENT,
            transactionReceipt);
473     ArrayList<ReverseRemovedEventResponse> responses = new
        ArrayList<ReverseRemovedEventResponse>(valueList.size());
474     for (EventValuesWithLog eventValues : valueList) {
475         ReverseRemovedEventResponse typedResponse = new
            ReverseRemovedEventResponse();
476         typedResponse.log = eventValues.getLog();
477         typedResponse.reverse = (String) eventValues.
            getIndexedValues().get(0).getValue();
478         typedResponse.name = (String) eventValues.
            getNonIndexedValues().get(0).getValue();
479         responses.add(typedResponse);
480     }
481     return responses;
482 }
483
484 public Flowable<ReverseRemovedEventResponse>
    reverseRemovedEventFlowable(EthFilter filter) {
485     return web3j.ethLogFlowable(filter).map(new io.reactivex.
        functions.Function<Log, ReverseRemovedEventResponse>() {
486         @Override
487         public ReverseRemovedEventResponse apply(Log log) {
488             EventValuesWithLog eventValues =
                extractEventParametersWithLog(REVERSEREMOVED_EVENT,
                    log);
489             ReverseRemovedEventResponse typedResponse = new
                ReverseRemovedEventResponse();
490             typedResponse.log = log;
491             typedResponse.reverse = (String) eventValues.
                getIndexedValues().get(0).getValue();
492             typedResponse.name = (String) eventValues.
                getNonIndexedValues().get(0).getValue();
493             return typedResponse;
494         }
495     });
496 }
497
498 public Flowable<ReverseRemovedEventResponse>
    reverseRemovedEventFlowable(DefaultBlockParameter startBlock,
        DefaultBlockParameter endBlock) {
499     EthFilter filter = new EthFilter(startBlock, endBlock,
        getContractAddress());
500     filter.addSingleTopic(EventEncoder.encode(REVERSEREMOVED_EVENT)
        );
501     return reverseRemovedEventFlowable(filter);
502 }
503
504 public List<ReservedEventResponse> getReservedEvents(
    TransactionReceipt transactionReceipt) {

```



```

505     List<EventValuesWithLog> valueList =
        extractEventParametersWithLog(RESERVED_EVENT,
            transactionReceipt);
506     ArrayList<ReservedEventResponse> responses = new ArrayList<
        ReservedEventResponse>(valueList.size());
507     for (EventValuesWithLog eventValues : valueList) {
508         ReservedEventResponse typedResponse = new
            ReservedEventResponse();
509         typedResponse.log = eventValues.getLog();
510         typedResponse.name = (byte[]) eventValues.getIndexedValues
            ().get(0).getValue();
511         typedResponse.owner = (String) eventValues.getIndexedValues
            ().get(1).getValue();
512         responses.add(typedResponse);
513     }
514     return responses;
515 }
516
517 public Flowable<ReservedEventResponse> reservedEventFlowable(
    EthFilter filter) {
518     return web3j.ethLogFlowable(filter).map(new io.reactivex.
        functions.Function<Log, ReservedEventResponse>() {
519         @Override
520         public ReservedEventResponse apply(Log log) {
521             EventValuesWithLog eventValues =
                extractEventParametersWithLog(RESERVED_EVENT, log);
522             ReservedEventResponse typedResponse = new
                ReservedEventResponse();
523             typedResponse.log = log;
524             typedResponse.name = (byte[]) eventValues.
                getIndexedValues().get(0).getValue();
525             typedResponse.owner = (String) eventValues.
                getIndexedValues().get(1).getValue();
526             return typedResponse;
527         }
528     });
529 }
530
531 public Flowable<ReservedEventResponse> reservedEventFlowable(
    DefaultBlockParameter startBlock, DefaultBlockParameter endBlock
    ) {
532     EthFilter filter = new EthFilter(startBlock, endBlock,
        getContractAddress());
533     filter.addSingleTopic(EventEncoder.encode(RESERVED_EVENT));
534     return reservedEventFlowable(filter);
535 }
536
537 public List<TransferredEventResponse> getTransferredEvents(
    TransactionReceipt transactionReceipt) {
538     List<EventValuesWithLog> valueList =
        extractEventParametersWithLog(TRANSFERRED_EVENT,

```

```
transactionReceipt);
539     ArrayList<TransferredEventResponse> responses = new ArrayList<
        TransferredEventResponse>(valueList.size());
540     for (EventValuesWithLog eventValues : valueList) {
541         TransferredEventResponse typedResponse = new
            TransferredEventResponse();
542         typedResponse.log = eventValues.getLog();
543         typedResponse.name = (byte[]) eventValues.getIndexedValues
            ().get(0).getValue();
544         typedResponse.oldOwner = (String) eventValues.
            getIndexedValues().get(1).getValue();
545         typedResponse.newOwner = (String) eventValues.
            getIndexedValues().get(2).getValue();
546         responses.add(typedResponse);
547     }
548     return responses;
549 }
550
551 public Flowable<TransferredEventResponse> transferredEventFlowable(
    EthFilter filter) {
552     return web3j.ethLogFlowable(filter).map(new io.reactivex.
        functions.Function<Log, TransferredEventResponse>() {
553         @Override
554         public TransferredEventResponse apply(Log log) {
555             EventValuesWithLog eventValues =
                extractEventParametersWithLog(TRANSFERRED_EVENT, log
            );
556             TransferredEventResponse typedResponse = new
                TransferredEventResponse();
557             typedResponse.log = log;
558             typedResponse.name = (byte[]) eventValues.
                getIndexedValues().get(0).getValue();
559             typedResponse.oldOwner = (String) eventValues.
                getIndexedValues().get(1).getValue();
560             typedResponse.newOwner = (String) eventValues.
                getIndexedValues().get(2).getValue();
561             return typedResponse;
562         }
563     });
564 }
565
566 public Flowable<TransferredEventResponse> transferredEventFlowable(
    DefaultBlockParameter startBlock, DefaultBlockParameter endBlock
) {
567     EthFilter filter = new EthFilter(startBlock, endBlock,
        getContractAddress());
568     filter.addSingleTopic(EventEncoder.encode(TRANSFERRED_EVENT));
569     return transferredEventFlowable(filter);
570 }
571
```

```
572     public List<DroppedEventResponse> getDroppedEvents(  
573         TransactionReceipt transactionReceipt) {  
574         List<EventValuesWithLog> valueList =  
            extractEventParametersWithLog(DROPPED_EVENT,  
            transactionReceipt);  
574         ArrayList<DroppedEventResponse> responses = new ArrayList<  
            DroppedEventResponse>(valueList.size());  
575         for (EventValuesWithLog eventValues : valueList) {  
576             DroppedEventResponse typedResponse = new  
                DroppedEventResponse();  
577             typedResponse.log = eventValues.getLog();  
578             typedResponse.name = (byte[]) eventValues.getIndexValues  
                ().get(0).getValue();  
579             typedResponse.owner = (String) eventValues.getIndexValues  
                ().get(1).getValue();  
580             responses.add(typedResponse);  
581         }  
582         return responses;  
583     }  
584  
585     public Flowable<DroppedEventResponse> droppedEventFlowable(  
        EthFilter filter) {  
586         return web3j.ethLogFlowable(filter).map(new io.reactivex.  
            functions.Function<Log, DroppedEventResponse>() {  
587             @Override  
588             public DroppedEventResponse apply(Log log) {  
589                 EventValuesWithLog eventValues =  
                    extractEventParametersWithLog(DROPPED_EVENT, log);  
590                 DroppedEventResponse typedResponse = new  
                    DroppedEventResponse();  
591                 typedResponse.log = log;  
592                 typedResponse.name = (byte[]) eventValues.  
                    getIndexValues().get(0).getValue();  
593                 typedResponse.owner = (String) eventValues.  
                    getIndexValues().get(1).getValue();  
594                 return typedResponse;  
595             }  
596         });  
597     }  
598  
599     public Flowable<DroppedEventResponse> droppedEventFlowable(  
        DefaultBlockParameter startBlock, DefaultBlockParameter endBlock  
    ) {  
600         EthFilter filter = new EthFilter(startBlock, endBlock,  
            getContractAddress());  
601         filter.addSingleTopic(EventEncoder.encode(DROPPED_EVENT));  
602         return droppedEventFlowable(filter);  
603     }  
604  
605     public List<DataChangedEventResponse> getDataChangedEvents(  
        TransactionReceipt transactionReceipt) {
```

```

606         List<EventValuesWithLog> valueList =
            extractEventParametersWithLog(DATACHANGED_EVENT,
                transactionReceipt);
607         ArrayList<DataChangedEventResponse> responses = new ArrayList<
            DataChangedEventResponse>(valueList.size());
608         for (EventValuesWithLog eventValues : valueList) {
609             DataChangedEventResponse typedResponse = new
                DataChangedEventResponse();
610             typedResponse.log = eventValues.getLog();
611             typedResponse.name = (byte[]) eventValues.getIndexValues
                ().get(0).getValue();
612             typedResponse.key = (String) eventValues.
                getNonIndexedValues().get(0).getValue();
613             typedResponse.plainKey = (String) eventValues.
                getNonIndexedValues().get(1).getValue();
614             responses.add(typedResponse);
615         }
616         return responses;
617     }
618
619     public Flowable<DataChangedEventResponse> dataChangedEventFlowable(
        EthFilter filter) {
620         return web3j.ethLogFlowable(filter).map(new io.reactivex.
            functions.Function<Log, DataChangedEventResponse>() {
621             @Override
622             public DataChangedEventResponse apply(Log log) {
623                 EventValuesWithLog eventValues =
                    extractEventParametersWithLog(DATACHANGED_EVENT, log
                        );
624                 DataChangedEventResponse typedResponse = new
                    DataChangedEventResponse();
625                 typedResponse.log = log;
626                 typedResponse.name = (byte[]) eventValues.
                    getIndexValues().get(0).getValue();
627                 typedResponse.key = (String) eventValues.
                    getNonIndexedValues().get(0).getValue();
628                 typedResponse.plainKey = (String) eventValues.
                    getNonIndexedValues().get(1).getValue();
629                 return typedResponse;
630             }
631         });
632     }
633
634     public Flowable<DataChangedEventResponse> dataChangedEventFlowable(
        DefaultBlockParameter startBlock, DefaultBlockParameter endBlock
        ) {
635         EthFilter filter = new EthFilter(startBlock, endBlock,
            getContractAddress());
636         filter.addSingleTopic(EventEncoder.encode(DATACHANGED_EVENT));
637         return dataChangedEventFlowable(filter);
638     }

```

```
639
640     public List<NewOwnerEventResponse> getNewOwnerEvents(
641         TransactionReceipt transactionReceipt) {
642         List<EventValuesWithLog> valueList =
643             extractEventParametersWithLog(NEWOWNER_EVENT,
644                 transactionReceipt);
645         ArrayList<NewOwnerEventResponse> responses = new ArrayList<
646             NewOwnerEventResponse>(valueList.size());
647         for (EventValuesWithLog eventValues : valueList) {
648             NewOwnerEventResponse typedResponse = new
649                 NewOwnerEventResponse();
650             typedResponse.log = eventValues.getLog();
651             typedResponse.old = (String) eventValues.getIndexedValues()
652                 .get(0).getValue();
653             typedResponse.current = (String) eventValues.
654                 getIndexedValues().get(1).getValue();
655             responses.add(typedResponse);
656         }
657         return responses;
658     }
659
660     public Flowable<NewOwnerEventResponse> newOwnerEventFlowable(
661         EthFilter filter) {
662         return web3j.ethLogFlowable(filter).map(new io.reactivex.
663             functions.Function<Log, NewOwnerEventResponse>() {
664                 @Override
665                 public NewOwnerEventResponse apply(Log log) {
666                     EventValuesWithLog eventValues =
667                         extractEventParametersWithLog(NEWOWNER_EVENT, log);
668                     NewOwnerEventResponse typedResponse = new
669                         NewOwnerEventResponse();
670                     typedResponse.log = log;
671                     typedResponse.old = (String) eventValues.
672                         getIndexedValues().get(0).getValue();
673                     typedResponse.current = (String) eventValues.
674                         getIndexedValues().get(1).getValue();
675                     return typedResponse;
676                 }
677             });
678     }
679
680     public Flowable<NewOwnerEventResponse> newOwnerEventFlowable(
681         DefaultBlockParameter startBlock, DefaultBlockParameter endBlock
682     ) {
683         EthFilter filter = new EthFilter(startBlock, endBlock,
684             getContractAddress());
685         filter.addSingleTopic(EventEncoder.encode(NEWOWNER_EVENT));
686         return newOwnerEventFlowable(filter);
687     }
688
689     @Deprecated
```

```
674     public static SimpleRegistry load(String contractAddress, Web3j
        web3j, Credentials credentials, BigInteger gasPrice, BigInteger
        gasLimit) {
675         return new SimpleRegistry(contractAddress, web3j, credentials,
            gasPrice, gasLimit);
676     }
677
678     @Deprecated
679     public static SimpleRegistry load(String contractAddress, Web3j
        web3j, TransactionManager transactionManager, BigInteger
        gasPrice, BigInteger gasLimit) {
680         return new SimpleRegistry(contractAddress, web3j,
            transactionManager, gasPrice, gasLimit);
681     }
682
683     public static SimpleRegistry load(String contractAddress, Web3j
        web3j, Credentials credentials, ContractGasProvider
        contractGasProvider) {
684         return new SimpleRegistry(contractAddress, web3j, credentials,
            contractGasProvider);
685     }
686
687     public static SimpleRegistry load(String contractAddress, Web3j
        web3j, TransactionManager transactionManager,
        ContractGasProvider contractGasProvider) {
688         return new SimpleRegistry(contractAddress, web3j,
            transactionManager, contractGasProvider);
689     }
690
691     public static RemoteCall<SimpleRegistry> deploy(Web3j web3j,
        Credentials credentials, ContractGasProvider contractGasProvider
        ) {
692         return deployRemoteCall(SimpleRegistry.class, web3j,
            credentials, contractGasProvider, BINARY, "");
693     }
694
695     @Deprecated
696     public static RemoteCall<SimpleRegistry> deploy(Web3j web3j,
        Credentials credentials, BigInteger gasPrice, BigInteger
        gasLimit) {
697         return deployRemoteCall(SimpleRegistry.class, web3j,
            credentials, gasPrice, gasLimit, BINARY, "");
698     }
699
700     public static RemoteCall<SimpleRegistry> deploy(Web3j web3j,
        TransactionManager transactionManager, ContractGasProvider
        contractGasProvider) {
701         return deployRemoteCall(SimpleRegistry.class, web3j,
            transactionManager, contractGasProvider, BINARY, "");
702     }
703
```

```
704  @Deprecated
705  public static RemoteCall<SimpleRegistry> deploy(Web3j web3j,
    TransactionManager transactionManager, BigInteger gasPrice,
    BigInteger gasLimit) {
706      return deployRemoteCall(SimpleRegistry.class, web3j,
    transactionManager, gasPrice, gasLimit, BINARY, "");
707  }
708
709  public static class DrainedEventResponse extends BaseEventResponse
    {
710      public BigInteger amount;
711  }
712
713  public static class FeeChangedEventResponse extends
    BaseEventResponse {
714      public BigInteger amount;
715  }
716
717  public static class ReverseProposedEventResponse extends
    BaseEventResponse {
718      public String reverse;
719
720      public String name;
721  }
722
723  public static class ReverseConfirmedEventResponse extends
    BaseEventResponse {
724      public String reverse;
725
726      public String name;
727  }
728
729  public static class ReverseRemovedEventResponse extends
    BaseEventResponse {
730      public String reverse;
731
732      public String name;
733  }
734
735  public static class ReservedEventResponse extends BaseEventResponse
    {
736      public byte[] name;
737
738      public String owner;
739  }
740
741  public static class TransferredEventResponse extends
    BaseEventResponse {
742      public byte[] name;
743
744      public String oldOwner;
```

```
745
746     public String newOwner;
747 }
748
749 public static class DroppedEventResponse extends BaseEventResponse
750 {
751     public byte[] name;
752     public String owner;
753 }
754
755 public static class DataChangedEventResponse extends
756 BaseEventResponse {
757     public byte[] name;
758     public String key;
759     public String plainKey;
760 }
761
762 public static class NewOwnerEventResponse extends BaseEventResponse
763 {
764     public String old;
765     public String current;
766 }
767 }
768 }
```

## 6.7 Certifier

### 6.7.1 Certifier.sol

```
1  /// Certifier contract, used by service transaction.
2  ///
3  /// Copyright 2016 Gavin Wood, Parity Technologies Ltd.
4  ///
5  /// Licensed under the Apache License, Version 2.0 (the "License");
6  /// you may not use this file except in compliance with the License.
7  /// You may obtain a copy of the License at
8  ///
9  ///     http://www.apache.org/licenses/LICENSE-2.0
10 ///
11 /// Unless required by applicable law or agreed to in writing, software
12 /// distributed under the License is distributed on an "AS IS" BASIS,
13 /// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
14 /// implied.
15 /// See the License for the specific language governing permissions and
16 /// limitations under the License.
```



```
16
17 pragma solidity ^0.4.24;
18
19
20 interface Certifier {
21     event Confirmed(address indexed who);
22     event Revoked(address indexed who);
23
24     function certified(address _who)
25         external
26         view
27         returns (bool);
28 }
```

### 6.7.2 Owned.sol

```
1  ///! The owned contract.
2  ///!
3  ///! Copyright 2016 Gavin Wood, Parity Technologies Ltd.
4  ///!
5  ///! Licensed under the Apache License, Version 2.0 (the "License");
6  ///! you may not use this file except in compliance with the License.
7  ///! You may obtain a copy of the License at
8  ///!
9  ///!     http://www.apache.org/licenses/LICENSE-2.0
10 ///!
11 ///! Unless required by applicable law or agreed to in writing, software
12 ///! distributed under the License is distributed on an "AS IS" BASIS,
13 ///! WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
14 ///! implied.
15 ///! See the License for the specific language governing permissions and
16 ///! limitations under the License.
17
18 pragma solidity ^0.4.24;
19
20 contract Owned {
21     event NewOwner(address indexed old, address indexed current);
22
23     address public owner = msg.sender;
24
25     modifier onlyOwner {
26         require(msg.sender == owner);
27         _;
28     }
29
30     function setOwner(address _new)
31         external
32         onlyOwner
```

```
33     {
34         emit NewOwner(owner, _new);
35         owner = _new;
36     }
37 }
```

### 6.7.3 SimpleCertifier.sol

```
1  ///! The SimpleCertifier contract, used by service transaction.
2  ///!
3  ///! Copyright 2016 Gavin Wood, Parity Technologies Ltd.
4  ///!
5  ///! Licensed under the Apache License, Version 2.0 (the "License");
6  ///! you may not use this file except in compliance with the License.
7  ///! You may obtain a copy of the License at
8  ///!
9  ///!     http://www.apache.org/licenses/LICENSE-2.0
10 ///!
11 ///! Unless required by applicable law or agreed to in writing, software
12 ///! distributed under the License is distributed on an "AS IS" BASIS,
13 ///! WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
14 ///! implied.
15 ///! See the License for the specific language governing permissions and
16 ///! limitations under the License.
17
18 pragma solidity ^0.4.24;
19
20 import "./Certifier.sol";
21 import "./Owned.sol";
22
23 contract SimpleCertifier is Owned, Certifier {
24     struct Certification {
25         bool active;
26     }
27
28     mapping (address => Certification) certs;
29
30     // So that the server posting puzzles doesn't have access to the
31     // ETH.
32     address public delegate = msg.sender;
33
34     modifier onlyDelegate {
35         require(msg.sender == delegate);
36     }
37
38     modifier onlyCertified(address _who) {
39         require(certs[_who].active);
```

```
40     _;  
41 }  
42  
43 function certify(address _who)  
44     external  
45     onlyDelegate  
46 {  
47     certs[_who].active = true;  
48     emit Confirmed(_who);  
49 }  
50  
51 function revoke(address _who)  
52     external  
53     onlyDelegate  
54     onlyCertified(_who)  
55 {  
56     certs[_who].active = false;  
57     emit Revoked(_who);  
58 }  
59  
60 function setDelegate(address _new)  
61     external  
62     onlyOwner  
63 {  
64     delegate = _new;  
65 }  
66  
67 function certified(address _who)  
68     external  
69     view  
70     returns (bool)  
71 {  
72     return certs[_who].active;  
73 }  
74 }
```

#### 6.7.4 Java-Wrapper für SimpleCertifier

```
1 package ch.brugg.fhnw.btm.contracts;  
2  
3 import io.reactivex.Flowable;  
4 import org.web3j.abi.EventEncoder;  
5 import org.web3j.abi.TypeReference;  
6 import org.web3j.abi.datatypes.*;  
7 import org.web3j.crypto.Credentials;  
8 import org.web3j.protocol.Web3j;  
9 import org.web3j.protocol.core.DefaultBlockParameter;  
10 import org.web3j.protocol.core.RemoteCall;  
11 import org.web3j.protocol.core.RemoteFunctionCall;
```

```
12 import org.web3j.protocol.core.methods.request.EthFilter;
13 import org.web3j.protocol.core.methods.response.BaseEventResponse;
14 import org.web3j.protocol.core.methods.response.Log;
15 import org.web3j.protocol.core.methods.response.TransactionReceipt;
16 import org.web3j.tx.Contract;
17 import org.web3j.tx.TransactionManager;
18 import org.web3j.tx.gas.ContractGasProvider;
19
20 import java.math.BigInteger;
21 import java.util.ArrayList;
22 import java.util.Arrays;
23 import java.util.Collections;
24 import java.util.List;
25
26 /**
27  * <p>Auto generated code.
28  * <p><strong>Do not modify!</strong>
29  * <p>Please use the <a href="https://docs.web3j.io/command_line.html">
    web3j command line tools</a>,
30  * or the org.web3j.codegen.SolidityFunctionWrapperGenerator in the
31  * <a href="https://github.com/web3j/web3j/tree/master/codegen">codegen
    module</a> to update.
32  *
33  * <p>Generated with web3j version 4.5.11.
34  *
35  * String used to generate this file:
36  * web3j solidity generate -b .\src\main\resources\solidity\Certifier\
    out\SimpleCertifier.bin -a .\src\main\resources\solidity\Certifier\
    out\SimpleCertifier.abi -o .\src\main\java -p io.kauri.tutorials.
    java_ethereum.contracts
37  */
38 @SuppressWarnings("rawtypes")
39 public class SimpleCertifier extends Contract {
40     public static final String BINARY = "Platzhalter für BinaryCode";
41
42     public static final String FUNC_SETOWNER = "setOwner";
43
44     public static final String FUNC_CERTIFY = "certify";
45
46     public static final String FUNC_REVOKE = "revoke";
47
48     public static final String FUNC_OWNER = "owner";
49
50     public static final String FUNC_DELEGATE = "delegate";
51
52     public static final String FUNC_SETDELEGATE = "setDelegate";
53
54     public static final String FUNC_CERTIFIED = "certified";
55
56     public static final Event CONFIRMED_EVENT = new Event("Confirmed",
```

```

57         Arrays.<TypeReference<?>>asList(new TypeReference<Address>(
58             true) {}));
59     ;
60     public static final Event REVOKED_EVENT = new Event("Revoked",
61         Arrays.<TypeReference<?>>asList(new TypeReference<Address>(
62             true) {}));
63     ;
64     public static final Event NEWOWNER_EVENT = new Event("NewOwner",
65         Arrays.<TypeReference<?>>asList(new TypeReference<Address>(
66             true) {}, new TypeReference<Address>(true) {}));
67     ;
68     @Deprecated
69     protected SimpleCertifier(String contractAddress, Web3j web3j,
70         Credentials credentials, BigInteger gasPrice, BigInteger
71         gasLimit) {
72         super(BINARY, contractAddress, web3j, credentials, gasPrice,
73             gasLimit);
74     }
75     protected SimpleCertifier(String contractAddress, Web3j web3j,
76         Credentials credentials, ContractGasProvider contractGasProvider
77         ) {
78         super(BINARY, contractAddress, web3j, credentials,
79             contractGasProvider);
80     }
81     @Deprecated
82     protected SimpleCertifier(String contractAddress, Web3j web3j,
83         TransactionManager transactionManager, BigInteger gasPrice,
84         BigInteger gasLimit) {
85         super(BINARY, contractAddress, web3j, transactionManager,
86             gasPrice, gasLimit);
87     }
88     protected SimpleCertifier(String contractAddress, Web3j web3j,
89         TransactionManager transactionManager, ContractGasProvider
90         contractGasProvider) {
91         super(BINARY, contractAddress, web3j, transactionManager,
92             contractGasProvider);
93     }
94     public RemoteFunctionCall<TransactionReceipt> setOwner(String _new)
95     {
96         final Function function = new Function(
97             FUNC_SETOWNER,
98             Arrays.<Type>asList(new Address(160, _new)),
99             Collections.<TypeReference<?>>emptyList());
100        return executeRemoteCallTransaction(function);

```

```
92     }
93
94     public RemoteFunctionCall<TransactionReceipt> certify(String _who)
95     {
96         final Function function = new Function(
97             FUNC_CERTIFY,
98             Arrays.<Type>asList(new Address(160, _who)),
99             Collections.<TypeReference<?>>emptyList());
100         return executeRemoteCallTransaction(function);
101     }
102
103     public RemoteFunctionCall<TransactionReceipt> revoke(String _who) {
104         final Function function = new Function(
105             FUNC_REVOKE,
106             Arrays.<Type>asList(new Address(160, _who)),
107             Collections.<TypeReference<?>>emptyList());
108         return executeRemoteCallTransaction(function);
109     }
110
111     public RemoteFunctionCall<String> owner() {
112         final Function function = new Function(FUNC_OWNER,
113             Arrays.<Type>asList(),
114             Arrays.<TypeReference<?>>asList(new TypeReference<
115                 Address>() {}));
116         return executeRemoteCallSingleValueReturn(function, String.
117             class);
118     }
119
120     public RemoteFunctionCall<String> delegate() {
121         final Function function = new Function(FUNC_DELEGATE,
122             Arrays.<Type>asList(),
123             Arrays.<TypeReference<?>>asList(new TypeReference<
124                 Address>() {}));
125         return executeRemoteCallSingleValueReturn(function, String.
126             class);
127     }
128
129     public RemoteFunctionCall<TransactionReceipt> setDelegate(String
130         _new) {
131         final Function function = new Function(
132             FUNC_SETDELEGATE,
133             Arrays.<Type>asList(new Address(160, _new)),
134             Collections.<TypeReference<?>>emptyList());
135         return executeRemoteCallTransaction(function);
136     }
137
138     public RemoteFunctionCall<Boolean> certified(String _who) {
139         final Function function = new Function(FUNC_CERTIFIED,
140             Arrays.<Type>asList(new Address(160, _who)),
141             Arrays.<TypeReference<?>>asList(new TypeReference<Bool
142                 >() {}));
```

```
136         return executeRemoteCallSingleValueReturn(function, Boolean.  
137             class);  
138     }  
139     public List<ConfirmedEventResponse> getConfirmedEvents(  
140         TransactionReceipt transactionReceipt) {  
141         List<EventValuesWithLog> valueList =  
142             extractEventParametersWithLog(CONFIRMED_EVENT,  
143             transactionReceipt);  
144         ArrayList<ConfirmedEventResponse> responses = new ArrayList<  
145             ConfirmedEventResponse>(valueList.size());  
146         for (EventValuesWithLog eventValues : valueList) {  
147             ConfirmedEventResponse typedResponse = new  
148                 ConfirmedEventResponse();  
149             typedResponse.log = eventValues.getLog();  
150             typedResponse.who = (String) eventValues.getIndexedValues()  
151                 .get(0).getValue();  
152             responses.add(typedResponse);  
153         }  
154         return responses;  
155     }  
156     public Flowable<ConfirmedEventResponse> confirmedEventFlowable(  
157         EthFilter filter) {  
158         return web3j.ethLogFlowable(filter).map(new io.reactivex.  
159             functions.Function<Log, ConfirmedEventResponse>() {  
160                 @Override  
161                 public ConfirmedEventResponse apply(Log log) {  
162                     EventValuesWithLog eventValues =  
163                         extractEventParametersWithLog(CONFIRMED_EVENT, log);  
164                     ConfirmedEventResponse typedResponse = new  
165                         ConfirmedEventResponse();  
166                     typedResponse.log = log;  
167                     typedResponse.who = (String) eventValues.  
168                         getIndexedValues().get(0).getValue();  
169                     return typedResponse;  
170                 }  
171             });  
172     }  
173     public Flowable<ConfirmedEventResponse> confirmedEventFlowable(  
174         DefaultBlockParameter startBlock, DefaultBlockParameter endBlock  
175     ) {  
176         EthFilter filter = new EthFilter(startBlock, endBlock,  
177             getContractAddress());  
178         filter.addSingleTopic(EventEncoder.encode(CONFIRMED_EVENT));  
179         return confirmedEventFlowable(filter);  
180     }  
181     public List<RevokedEventResponse> getRevokedEvents(  
182         TransactionReceipt transactionReceipt) {
```

```
171     List<EventValuesWithLog> valueList =
        extractEventParametersWithLog(REVOKED_EVENT,
            transactionReceipt);
172     ArrayList<RevokedEventResponse> responses = new ArrayList<
        RevokedEventResponse>(valueList.size());
173     for (EventValuesWithLog eventValues : valueList) {
174         RevokedEventResponse typedResponse = new
            RevokedEventResponse();
175         typedResponse.log = eventValues.getLog();
176         typedResponse.who = (String) eventValues.getIndexedValues()
            .get(0).getValue();
177         responses.add(typedResponse);
178     }
179     return responses;
180 }
181
182 public Flowable<RevokedEventResponse> revokedEventFlowable(
    EthFilter filter) {
183     return web3j.ethLogFlowable(filter).map(new io.reactivex.
        functions.Function<Log, RevokedEventResponse>() {
184         @Override
185         public RevokedEventResponse apply(Log log) {
186             EventValuesWithLog eventValues =
                extractEventParametersWithLog(REVOKED_EVENT, log);
187             RevokedEventResponse typedResponse = new
                RevokedEventResponse();
188             typedResponse.log = log;
189             typedResponse.who = (String) eventValues.
                getIndexedValues().get(0).getValue();
190             return typedResponse;
191         }
192     });
193 }
194
195 public Flowable<RevokedEventResponse> revokedEventFlowable(
    DefaultBlockParameter startBlock, DefaultBlockParameter endBlock
    ) {
196     EthFilter filter = new EthFilter(startBlock, endBlock,
        getContractAddress());
197     filter.addSingleTopic(EventEncoder.encode(REVOKED_EVENT));
198     return revokedEventFlowable(filter);
199 }
200
201 public List<NewOwnerEventResponse> getNewOwnerEvents(
    TransactionReceipt transactionReceipt) {
202     List<EventValuesWithLog> valueList =
        extractEventParametersWithLog(NEWOWNER_EVENT,
            transactionReceipt);
203     ArrayList<NewOwnerEventResponse> responses = new ArrayList<
        NewOwnerEventResponse>(valueList.size());
204     for (EventValuesWithLog eventValues : valueList) {
```



```

205         NewOwnerEventResponse typedResponse = new
            NewOwnerEventResponse();
206         typedResponse.log = eventValues.getLog();
207         typedResponse.old = (String) eventValues.getIndexedValues()
            .get(0).getValue();
208         typedResponse.current = (String) eventValues.
            getIndexedValues().get(1).getValue();
209         responses.add(typedResponse);
210     }
211     return responses;
212 }
213
214 public Flowable<NewOwnerEventResponse> newOwnerEventFlowable(
    EthFilter filter) {
215     return web3j.ethLogFlowable(filter).map(new io.reactivex.
        functions.Function<Log, NewOwnerEventResponse>() {
216         @Override
217         public NewOwnerEventResponse apply(Log log) {
218             EventValuesWithLog eventValues =
                extractEventParametersWithLog(NEWOWNER_EVENT, log);
219             NewOwnerEventResponse typedResponse = new
                NewOwnerEventResponse();
220             typedResponse.log = log;
221             typedResponse.old = (String) eventValues.
                getIndexedValues().get(0).getValue();
222             typedResponse.current = (String) eventValues.
                getIndexedValues().get(1).getValue();
223             return typedResponse;
224         }
225     });
226 }
227
228 public Flowable<NewOwnerEventResponse> newOwnerEventFlowable(
    DefaultBlockParameter startBlock, DefaultBlockParameter endBlock
    ) {
229     EthFilter filter = new EthFilter(startBlock, endBlock,
        getContractAddress());
230     filter.addSingleTopic(EventEncoder.encode(NEWOWNER_EVENT));
231     return newOwnerEventFlowable(filter);
232 }
233
234 @Deprecated
235 public static SimpleCertifier load(String contractAddress, Web3j
    web3j, Credentials credentials, BigInteger gasPrice, BigInteger
    gasLimit) {
236     return new SimpleCertifier(contractAddress, web3j, credentials,
        gasPrice, gasLimit);
237 }
238
239 @Deprecated

```

```
240     public static SimpleCertifier load(String contractAddress, Web3j
        web3j, TransactionManager transactionManager, BigInteger
        gasPrice, BigInteger gasLimit) {
241         return new SimpleCertifier(contractAddress, web3j,
            transactionManager, gasPrice, gasLimit);
242     }
243
244     public static SimpleCertifier load(String contractAddress, Web3j
        web3j, Credentials credentials, ContractGasProvider
        contractGasProvider) {
245         return new SimpleCertifier(contractAddress, web3j, credentials,
            contractGasProvider);
246     }
247
248     public static SimpleCertifier load(String contractAddress, Web3j
        web3j, TransactionManager transactionManager,
        ContractGasProvider contractGasProvider) {
249         return new SimpleCertifier(contractAddress, web3j,
            transactionManager, contractGasProvider);
250     }
251
252     public static RemoteCall<SimpleCertifier> deploy(Web3j web3j,
        Credentials credentials, ContractGasProvider contractGasProvider
        ) {
253         return deployRemoteCall(SimpleCertifier.class, web3j,
            credentials, contractGasProvider, BINARY, "");
254     }
255
256     @Deprecated
257     public static RemoteCall<SimpleCertifier> deploy(Web3j web3j,
        Credentials credentials, BigInteger gasPrice, BigInteger
        gasLimit) {
258         return deployRemoteCall(SimpleCertifier.class, web3j,
            credentials, gasPrice, gasLimit, BINARY, "");
259     }
260
261     public static RemoteCall<SimpleCertifier> deploy(Web3j web3j,
        TransactionManager transactionManager, ContractGasProvider
        contractGasProvider) {
262         return deployRemoteCall(SimpleCertifier.class, web3j,
            transactionManager, contractGasProvider, BINARY, "");
263     }
264
265     @Deprecated
266     public static RemoteCall<SimpleCertifier> deploy(Web3j web3j,
        TransactionManager transactionManager, BigInteger gasPrice,
        BigInteger gasLimit) {
267         return deployRemoteCall(SimpleCertifier.class, web3j,
            transactionManager, gasPrice, gasLimit, BINARY, "");
268     }
269
```

```
270     public static class ConfirmedEventResponse extends
      BaseEventResponse {
271         public String who;
272     }
273
274     public static class RevokedEventResponse extends BaseEventResponse
      {
275         public String who;
276     }
277
278     public static class NewOwnerEventResponse extends BaseEventResponse
      {
279         public String old;
280
281         public String current;
282     }
283
284
285 }
```

## 7 Ehrlichkeitserklärung

Die eingereichte Arbeit ist das Resultat unserer persönlichen, selbstständigen Beschäftigung mit dem Thema. Alle wörtlichen und sinngemässen Übernahmen aus anderen Werken sind als solche gekennzeichnet

Datum \_\_\_\_\_

Ort \_\_\_\_\_

Faustina Bruno \_\_\_\_\_

Serge Jurij Maïkoff \_\_\_\_\_