# IP6: Blockchain Transactionmanager

Bachelorthesis

Faustina Bruno, Jurij Maïkoff

**Studiengang:**
- iCompetence
- Informatik

**Betreuer:**
- Markus Knecht
- Daniel Kröni

**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

# 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 test Zwecken genutzt werden kann. Blockchains verfügen über verschiedene Mechanismen, um sich gegen Attacken abzusichern. Eine davon ist eine Gebühr auf jeder Transaktion, der sogenannte Gas Price 2.2.4 [3]. Dadurch können Denial of Service (DoS) Attacken 2.2.8 [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 Attacken.

## 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 Attacken hat.

## 1.3 Methodik

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.

## 1.4 Strukturierung des Berichts

Der Bericht ist in einen theoretischen und praktischen Teil gegliedert. Gemachte Literaturstudiuen, geprüfte Tools und der aktuelle Stand der Ethereum Blockchain 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 und verwendeter Code zu finden.

# 2 Theoretische Grundlagen

Dieses Kapitel befasst sich nebst dem Kontext der Arbeit, mit den gemachteten 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 kompromitiert 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

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 Servie (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]

Blockchains sind auf einem peer-to-peer (P2P) Netzwerk[5] 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 Consenus Algorithmen[6] entgegen gewirkt. Die Consensus Algorithemen stellen sicher, dass die Transaktionen auf der Blockchain valide und authentisch sind.

Im Gegensatz zur Bitcoin[7] kann bei Ethereum[8] 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[9] 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.

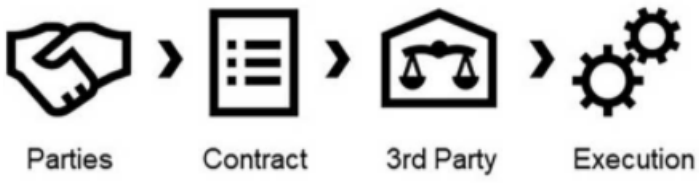



**Abbildung 2.1:** Ein traditioneller Vertag[10]


**Abbildung 2.2:** Ein Smart Contract[10]

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 2.2.5.3. Mit dieser, kann auf die Funktionen des Smart Contracts zugegriffen werden.

#### 2.2.2.1 Decentralized application (DApp)

Eine DApp ist 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. Das prominenteste Beispiel einer DApp ist CryptoKitties[11], beider die Benutzer digitale Katzen handeln und züchten können.

### 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 2.2.5.3, 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[12] 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, wir in diesem Feld der dessen Bytecode[13] 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 gwillt 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

### 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 Berchnung 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 2.2.7, 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 zurück erstattet, da die Miner bereits Rechenleistung erbracht haben.

### 2.2.5 Identitäten

Um mit Ethereum interagieren zu können, wird eine Identität benötigt. Diese besteht aus einem öffentlichen und einem geheimen Schlüssel.

#### 2.2.5.1 Geheimer Schlüssel

#### 2.2.5.2 Öffentlicher Schlüssel

#### 2.2.5.3 Adresse

### 2.2.6 Wallets und Smart Wallets

### 2.2.7 Mining

### 2.2.8 Denial of Service (DOS) Attacken

### 2.2.9 Algorithmen

## 2.3 Lösungsansätze

//TODO Spellcheck über ganze Seite

### 2.3.1 Lösungsansatz 1: Super Smart Wallet

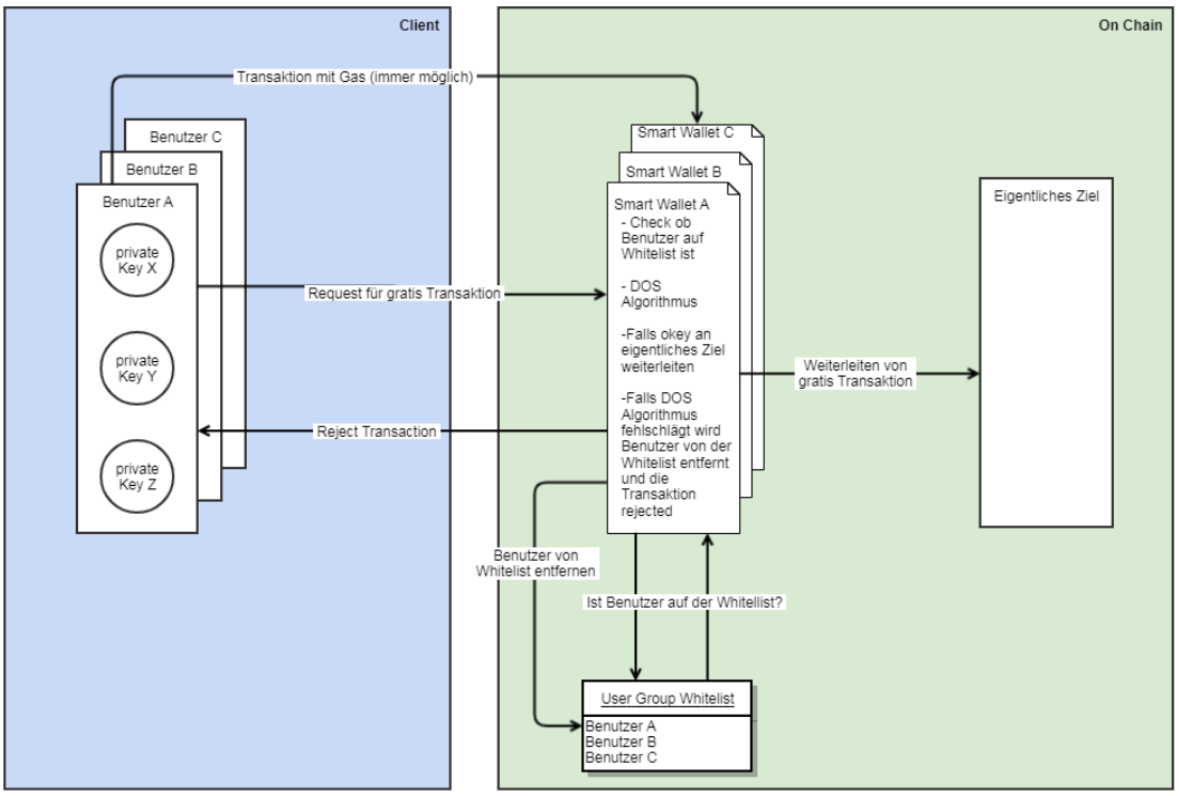



**Abbildung 2.3:** Lösungsansatz1

#### 2.3.1.1 Hauptlösungsansätze

Es wird eine komplett eigene Smart Wallet erstellt, die den DoS Schutzalgorithmus verwaltet. Jeder Benutzer erhält eine eigene Smart Wallet die von den Admins deployt wird. Die Whitelist wird von der Blockchain verwaltet. Auf einer Whitelist sind alle Benutzer aufgelistet die berechtigt sind gratis Transaktionen durchzuführen. Diese Liste wird von den Admins gepfleft. Der Sicherheitsalgorithmus prüft ob der Benutzer die Gratistransaktions Richtlinien verletzt. Falls der Benutzer die Sicherheitsrichtlinien verletzt, wird er vom Algorithmus aus der Whitelist gelöscht. Der Benutzer gelangt nur wieder in die Whitelist, wenn ein Admin ihn hinzufügt.

Die bezahlten Transaktionen laufen auch über die SmartWallet, um mit der gleichen Sender Identität Transaktionen an das eigentliche Ziel zu verschicken.

Hier besteht das Problem, dass auch gratis Transaktionen geschickt werden können, ohne über die Smart Wallet zu gehen. Somit kann der Benutzer den DoS Schutzalgorithmus umgehen. Deswegen

muss ein Weg gefunden werden, den den Benutzer zwingt über die Smart Wallet Transaktionen zu schicken (z.B. Whitelist wo Sender und Empfänger geführt wird).

#### 2.3.1.2 Pro

- Alles auf der Blockchain
- Dezentral
- Elegant
- Ein System

#### 2.3.1.3 Contra

- Machbarkeit unsicher
- Komplex
- Nach dem Anpassen vom Schutz Algorithmus in der Smart Wallet, muss eine neue Smart Wallet deployed werden. Die alte bleibt bestehen sofern die Blocchain nicht resetted wird

#### 2.3.1.4 Prozessworkflow

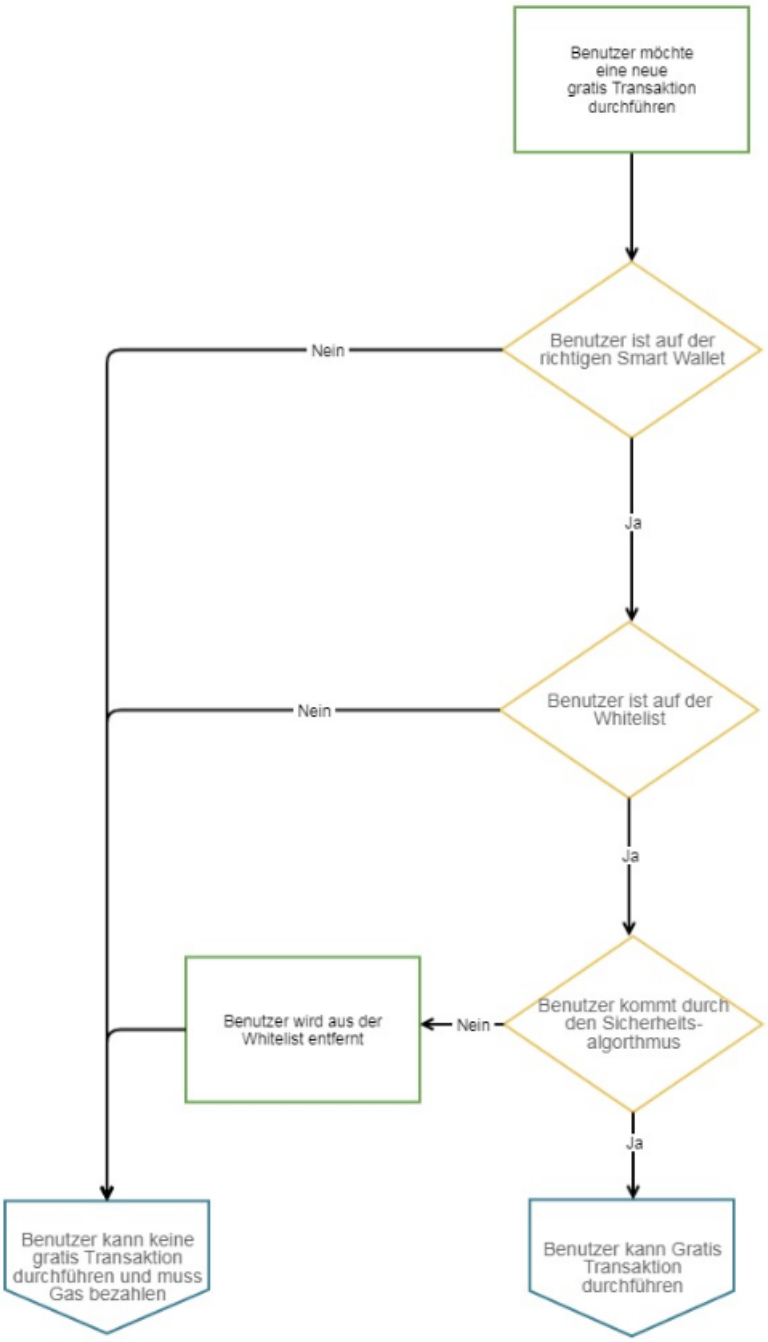



**Abbildung 2.4:** FlowchartLösungsansatz1

### 2.3.2 Lösungsansatz 2: Smart Wallet mit externen JavaProgramm nach Whitelist-Check

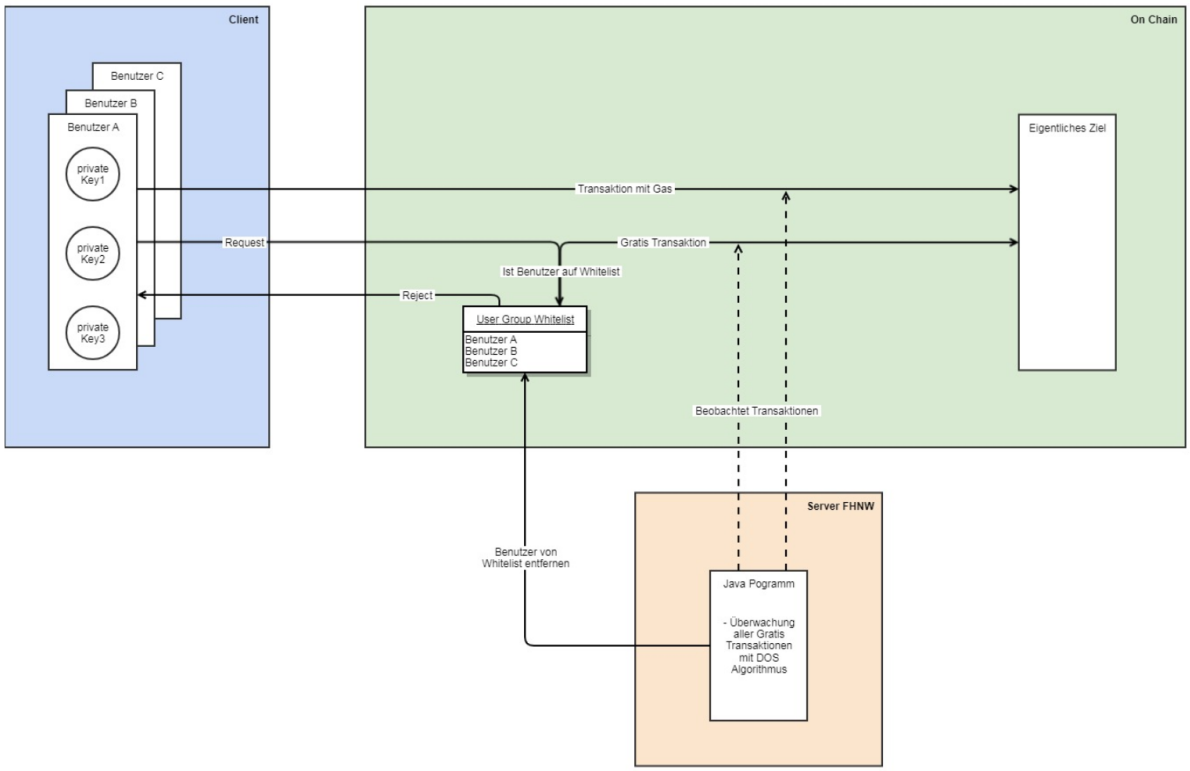



**Abbildung 2.5:** Lösungsansatz2

#### 2.3.2.1 Hauptlösungsansätze

Dies ist ein Lösungsansatz ohne Smart Wallet. Die Whitelist wird von der Blockchain selber verwaltet. Der DoS Schutzalgorithmus wird in einem externen Java Programm durchgeführt. Auf einer Whitelist sind alle Benutzer aufgelistet die berechtigt sind gratis Transaktionen durchzuführen. Diese Liste wird von den Admins gepfleft. Der externe Sicherheitsalgorithmus prüft nach dem Whitelist-Check ob der Benutzer die Gratistransaktion Richtlinien verletzt. Falls der Benutzer die Sicherheitsrichtlinien verletzt, wird der Benutzer vom Algorithmus aus der Whitelist gelöscht. Der Benutzer gelangt nur wieder in die Whitelist, wenn ein Admin ihn hinzufügt.

#### 2.3.2.2 Pro

- Sicher machbar

- Nach dem Anpassen vom Schutz Algorithmus in der Smart Wallet, muss keine neue Smart Wallet deployed werden.

#### 2.3.2.3 Contra

- Mehrere Komponenten
- Zentrale Autorität

#### 2.3.2.4 Prozessworkflow

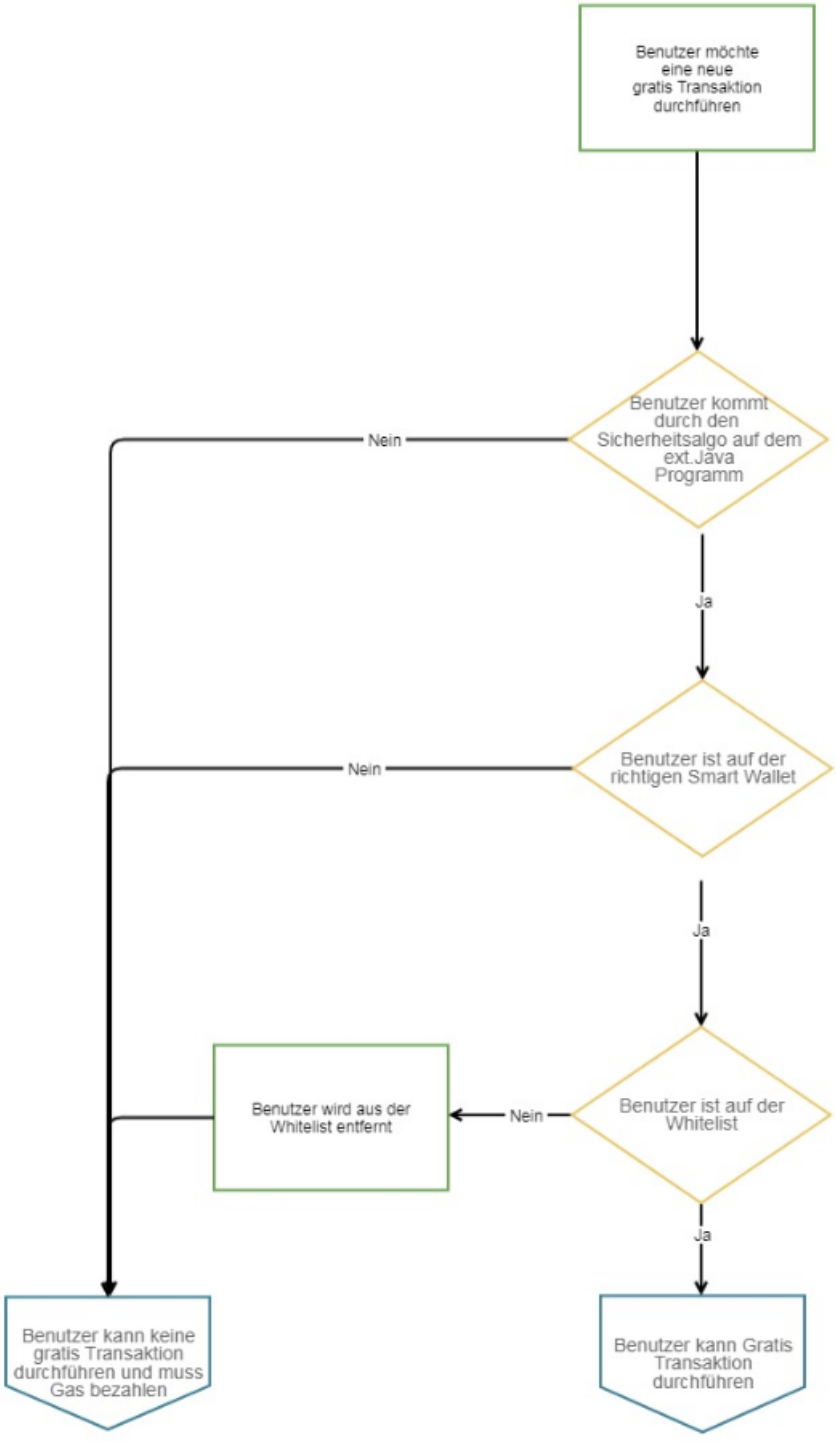



**Abbildung 2.6:** FlowchartLösungsansatz2

### 2.3.3 Lösungsansatz 3: Smart Wallet mit externen JavaProgramm vor Whitelist-Check

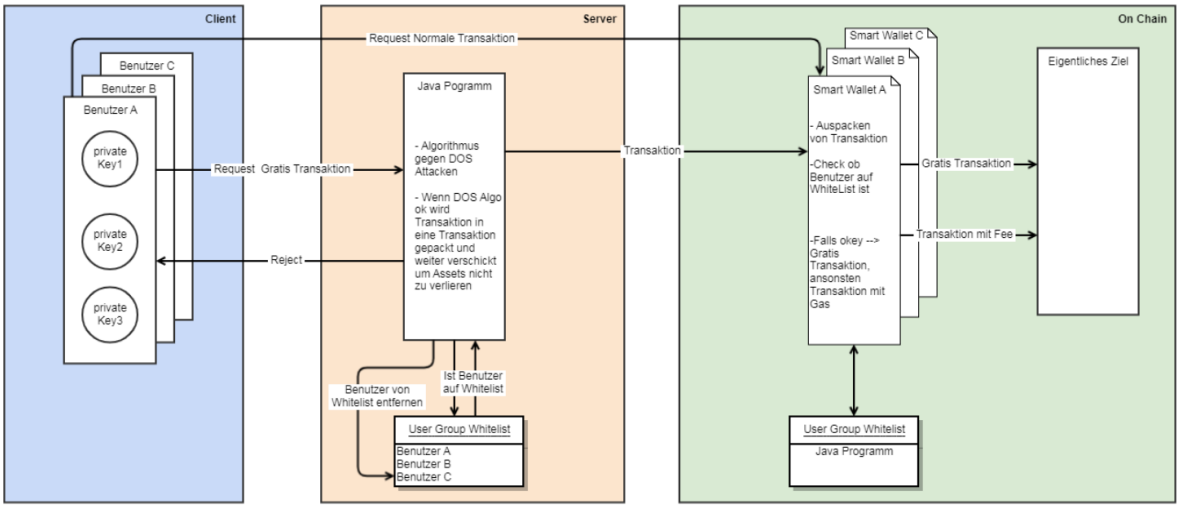



**Abbildung 2.7:** Lösungsansatz3

#### 2.3.3.1 Hauptlösungsansätze

Es wird ein Java Programm entwickelt, welches eine eigene Whitelist führt und den DoS Schutzalgorithmus beinhaltet. Dieser prüft ob der Benutzer auf der Whitelist ist und ob die Transaktion die Schutzrichtlinien nicht verletzt. Ist die Prüfung in Ordnung packt er die Transaktion in eine neue Transaktion ein, um die Transaktionsinformationen (wie z.B. Sender Identität) nicht zu verlieren und schickt die Transaktion an die Smart Wallet. Falls der Benutzer die Sicherheitsrichtlinien verletzt, wird er vom Algorithmus aus der Whitelist gelöscht. Jeder Benutzer besitzt eine eigene Smart Wallet um die Sender Identität für jeden Benutzer einmalig zu halten. Die Smart Wallet packt die Transaktion aus und schickt eine neue Transaktion mit den relevanten Informationen an das eigentliche Ziel. Auf einer Whitelist der Blockchain ist nur das Javaprogram aufgelistet, so dass nur die Transaktionen die vom Java Programm weitergeleitet wurden, kostenfrei durchgeführt werden können. Die kostenpflichtigen Transaktionen werden vom Benutzer auch an die Smart Wallet geschickt.

#### 2.3.3.2 Pro

- Komplett eigene Lösung
- Nach dem Anpassen vom Schutz Algorithmus in der Smart Wallet, muss keine neue Smart Wallet deployed werden.
- DOS Algorithmus blockt bevor Transaktion auf SmartWallet trifft

- Problem dass gratis Transaktionen nicht über die Smart Wallet geschickt werden, ist hier gelöst

#### 2.3.3.3 Contra

- Mehrere Systeme
- Zentrale Autorität

#### 2.3.3.4 Prozessworkflow

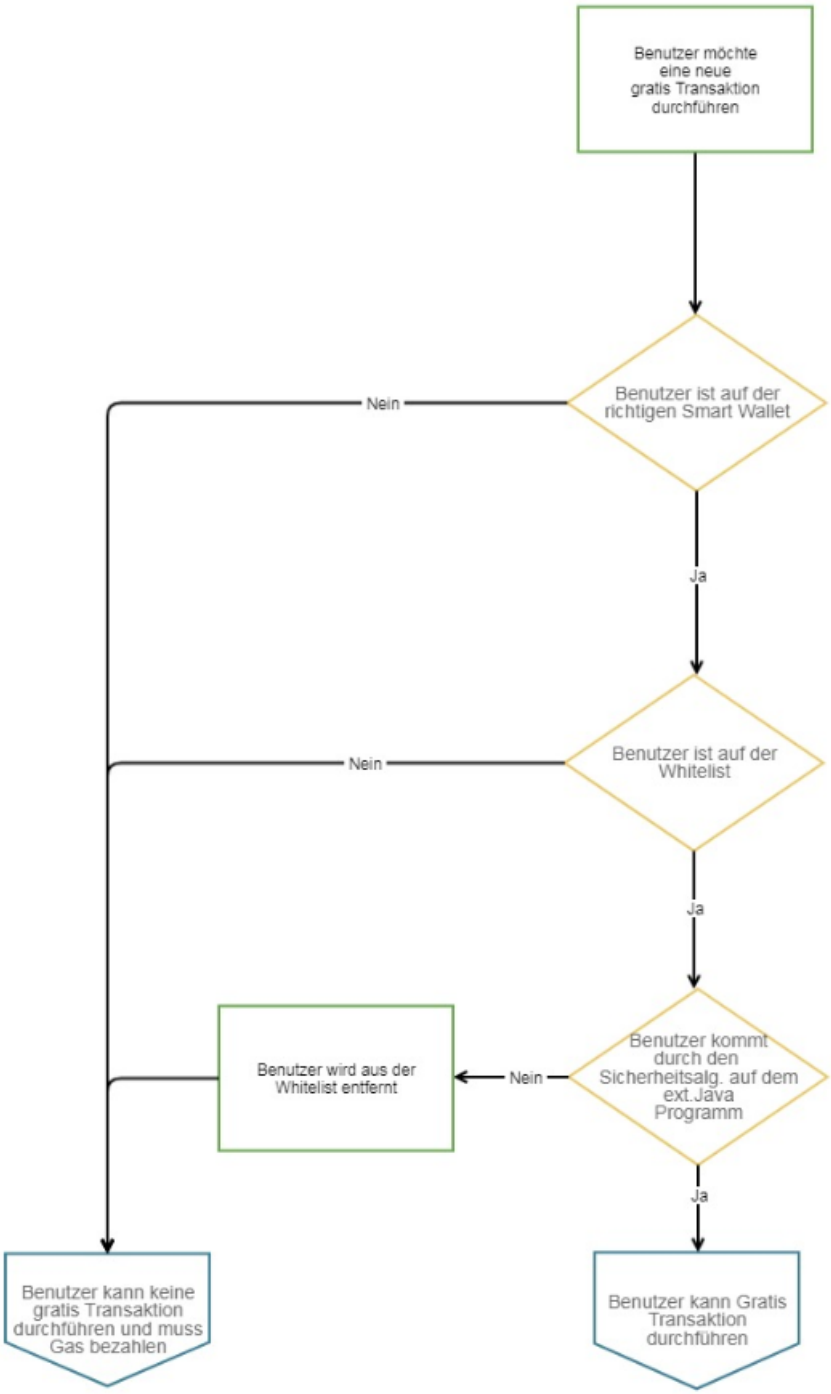
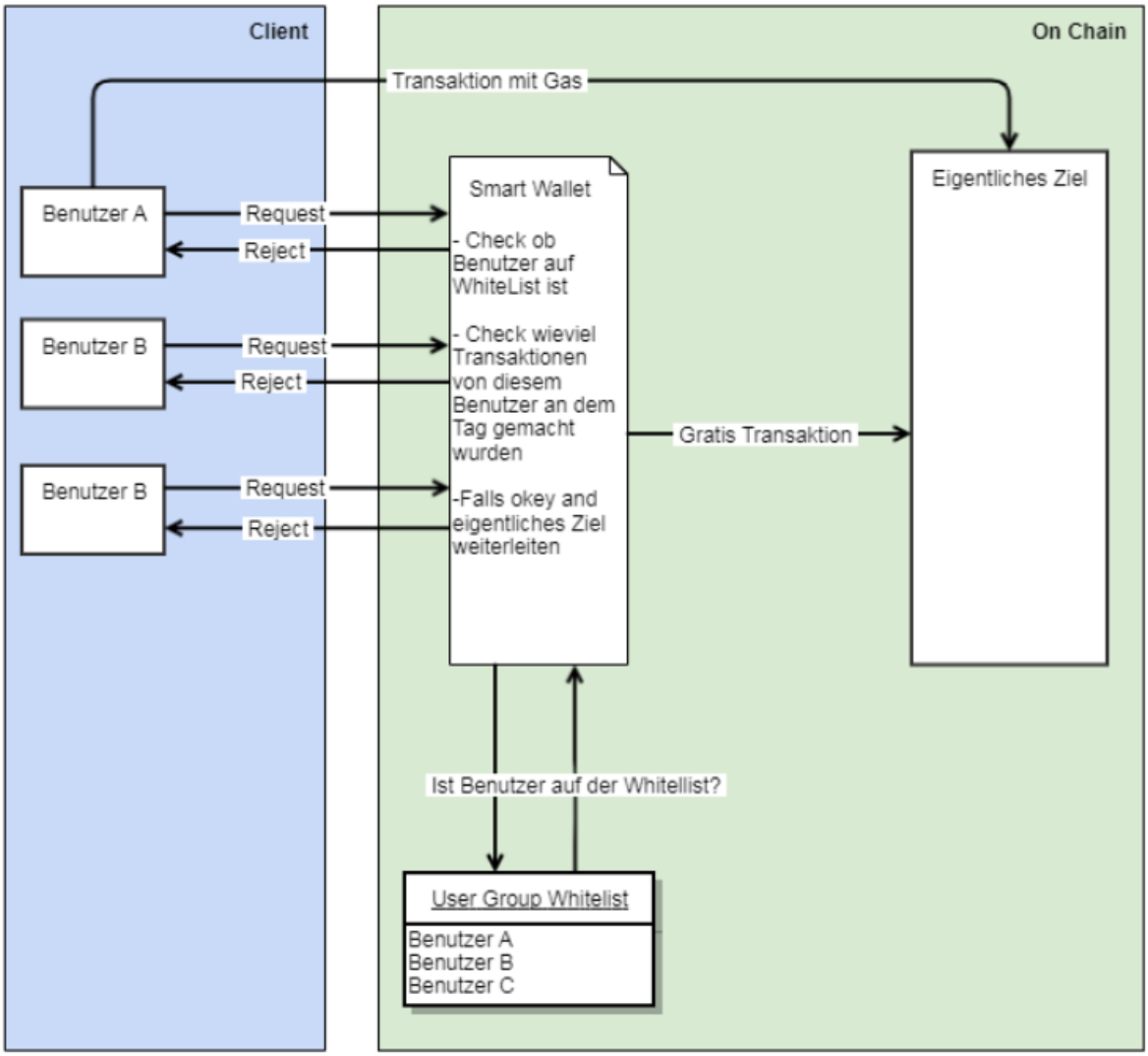



**Abbildung 2.8:** FlowchartLösungsansatz3

### 2.3.4 Lösungsansatz 4: Zentrale Smart Wallet für jeden Benutzer




//TODO Bild

#### 2.3.4.1 Hauptlösungsansätze

Dieser Lösungsansatz war der Es wird eine komplett eigene Smart Wallet erstellt, die sowohl die Whitelist wie auch den Schutzalgorithmus verwaltet. Alle Benutzer erhalten eine zentrale Smart Wallet die von den Admins deployt wird. Auf einer Whitelist sind alle Benutzer aufgelistet die berechtigt sind gratis Transaktionen durchzuführen. Diese Liste wird von den Admins gepfleft. Der Sicherheitsalgorithmus prüft ob der Benutzer die Gratistransaktions Richtlinien verletzt. Falls der Benutzer die Sicherheitsrichtlinien verletzt, wird er vom Algorithmus aus der Whitelist gelöscht. Der Benutzer gelangt nur wieder in die Whitelist, wenn ein Admin ihn hinzufügt.

#### 2.3.4.2 Pro

- Nur eine Smart Wallet muss deployed und betrieben werden
- Weniger administrativer Aufwand für Admins

#### 2.3.4.3 Contra

-Schwierig Sender ID für Transaktion zu setzten (überhaupt möglich?)

#### 2.3.4.4 Prozessworkflow

//TODO Bild

## 2.4 Evaluation der Lösungsansätze

Folgende Evaluationskriterien wurden bestummen

- Sichere Machbarkeit (hohe Priorität)
- Tiefe Komplexität (hohe Priorität)
- Komplexität bei Anpassungen (mittlere Priorität)
- Tiefer Waste bei Anpassungen (mittlere Priorität)
- Alles On Chain (tiefe Priorität)
- Wenig administrativer Aufwand (tiefe Priorität)
- Elegantheit der Lösung (tiefe Priorität)
- Normale Transaktion ( ?? Prio)
- Security ( ?? Prio ) //TODO Prioritäten besprechen

**Tabelle 2.1:** Evaluation Lösungsansätze

| | Sichere Machbarkeit | Tiefe Komplexität | Komplexität bei Anpassungen | Tiefer Waste bei Anpassungen | Alles On Chain | Wenig administrativer Aufwand | Elegantheit der Lösung | Normale Transaktion | Security | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| Prio | 3 | 3 | 2 | 2 | 1 | 1 | ? | ? | | |

| | Sichere Machbarkeit | Tiefe Komplexität | Komplexität bei Anpassungen | Tiefer Waste bei Anpassungen | Alles On Chain | Wenig administrativer Aufwand | Eleganteheit der Lösung | Normale Transaktion | Security | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| Lösungsansatz 1 | 2 | 1 | 0 | 0 | 2 | 1 | 2 | 2 | ? | |
| Lösungsansatz 2 | 2 | 2 | 2 | 2 | 0 | 1 | 1 | 2 | ? | |
| Lösungsansatz 3 | 2 | 1 | 2 | 2 | 0 | 1 | 0 | 0 | ? | |
| Lösungsansatz 4 | 0 | 0 | 0 | 2 | 0 | 2 | 1 | 0 | ? | |

### 2.4.1 Lösungsansatz 1: Smart Wallet

//TODO nach Prio nochmals umschreiben Der Lösungsansatz 1 ist die eleganteste Lösung, jedoch laut Evaluation die zweit beste, zusammen mit dem Lösungsansatz 3. Da diese Lösung kein Java Programm wie Lösungsansatz 2 und 3 vorsieht ist sie prioritär. Falls die Kapazitäten ausreichen, wird sie somit auch implementiert.

### 2.4.2 Lösungsansatz 2: Smart Wallet mit externen JavaProgramm nach Whitelist-Check

Ergebnis der Evaluation zeigt, dass diese Lösung die beste nach den Kriterien ist. Deswegen wird diese Lösung als erstes implementiert.

### 2.4.3 Lösungsansatz 3: Smart Wallet mit externen JavaProgramm vor Whitelist-Check

Laut Evaluation ist dieser Lösungsansatz auf dem dritten Platz. Deswegen wird

### 2.4.4 Lösungsansatz 4: Super Smart Wallet

Dieser Lösungsansatz ist laut Evaluation auf dem letzten Platz. Dies war von Anfang an klar, er wurde aufgezeigt, da es die erste Idee war, für die Lösung.

# 3 Praktischer Teil

# 4 Fazit

# 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.

[5] „Peer-to-peer - Wikipedia“, 2019. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/Peer-to-peer.

[6] „What Is a Blockchain Consensus Algorithmen | Binance Academy“, 2019. [Online]. Verfügbar unter: https://www.binance.vision/blockchain/what-is-a-blockchain-consensus-algorithm.

[7] „Bitcoin - Wikipedia“, 2019. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/Bitcoin.

[8] Ethereum, „Home | Ethereum“, 2019. [Online]. Verfügbar unter: https://www.ethereum.org/.

[9] „Nick Szabo - Wikipedia“, 2019. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/Nick_Szabo.

[10] „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.

[11] „CryptoKitties | Collect and breed digital cats!“, 2019. [Online]. Verfügbar unter: https://www.cryptokitties.co/.

[12] „Wei“, 2019. [Online]. Verfügbar unter: https://www.investopedia.com/terms/w/wei.asp.

[13] 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.

[14] go-ehereum, „Go Ethereum“, 2019. [Online]. Verfügbar unter: https://geth.ethereum.org/.

[15] P. Technologies, „Blockchain Infrastructure for the Decentralised Web | Parity Technologies“, 2019. [Online]. Verfügbar unter: https://www.parity.io.

[16] „https://github.com/ethereum/aleth“, 2019. [Online]. Verfügbar unter: https://github.com/ethereum/aleth.

[17] T. B. G. 2019, „Sweet Tools for Smart Contracts“, 2019. [Online]. Verfügbar unter: https://www.trufflesuite.com/.

[18] uPort, „uPort“, 2019. [Online]. Verfügbar unter: https://www.uport.me/.

[19] MetaMask, „MetaMask“, 2019. [Online]. Verfügbar unter: https://metamask.io/.

[20] A. Wallet, „Atomic Cryptocurrency Wallet“, 2019. [Online]. Verfügbar unter: https://atomicwallet.io/.

[21] E. M. Inc., „Crypte Wallet - Send, Receive & Exchange Cryptocurrency | Exodus“, 2019. [Online]. Verfügbar unter: https://www.exodus.io.

[22] MyEtherWallet, „MyEtherWallet | MEW“, 2019. [Online]. Verfügbar unter: https://www.myetherwallet.com/.

[23] Solidity, „Solidity - Solidity 0.5.11 documentation“, 2019. [Online]. Verfügbar unter: https://solidity.readthedocs.io/en/v0.5.11/.

[24] „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[8] 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[14] als Client den Ansprüchen genügt oder ob ein anderer Client (z.B. Parity[15], Aleth[16], etc.) zum Einsatz kommt.

**Trufflesuite** Trufflesuite[17] 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[18]
- Metamask[19]
- Atomic Wallet [20]
- Exodus[21]

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 ertellen: MyEtherWallet[22]

### 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.

#### 6.2.3.1 Programmiersprache

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

- Solidity[23]
- Vyper[24]

# 7 Ehrlickeitserklärung

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 _____