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 **1**

1.1 Problemstellung 1
1.2 Ziel 1
1.3 Methodik 1
1.4 Strukturierung des Berichts 2

2 Theoretische Grundlagen **3**

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 7
2.2.7 Denial of Service (DoS) Attacken 8
2.2.8 Whitelist von Parity 9
2.3 Lösungsansätze 9
2.3.1 Lösungsansatz 1: Smart Wallet 10
2.3.2 Lösungsansatz 2: Smart Wallet mit externen JavaProgramm nach Whitelist-Check 13
2.3.3 Lösungsansatz 3: Smart Wallet mit externen JavaProgramm vor Whitelist-Check 16
2.3.4 Lösungsansatz 4: Super Smart Wallet 19
2.4 Evaluation der Lösungsansätze 20
2.4.1 Lösungsansatz 1: Smart Wallet 21
2.4.2 Lösungsansatz 2: Smart Wallet mit externen JavaProgramm nach Whitelist-Check 21
2.4.3 Lösungsansatz 3: Smart Wallet mit externen JavaProgramm vor Whitelist-Check 21
2.4.4 Lösungsansatz 4: Super Smart Wallet 22
2.5 Lösungsansätze 22
2.5.1 Lösungsansatz 1: Smart Wallet 22
2.5.2 Lösungsansatz 2: Smart Wallet mit externen JavaProgramm nach Whitelist-Check 26

2.5.3 Lösungsansatz 3: Smart Wallet mit externen JavaProgramm vor Whitelist-Check 29
2.5.4 Lösungsansatz 4: Super Smart Wallet 32
2.6 Evaluation der Lösungsansätze 33
2.6.1 Lösungsansatz 1: Smart Wallet 34
2.6.2 Lösungsansatz 2: Smart Wallet mit externen JavaProgramm nach Whitelist-Check 34
2.6.3 Lösungsansatz 3: Smart Wallet mit externen JavaProgramm vor Whitelist-Check 35
2.6.4 Lösungsansatz 4: Super Smart Wallet 35

3 Praktischer Teil 36

4 Fazit 37

5 Quellenverzeichnis 38

6 Anhang 41
6.1 Glossar 41
6.2 Entwicklungsumgebung 41
6.2.1 Blockchain 42
6.2.2 Wallet 42
6.2.3 Smart Contracts 42

7 Ehrlickeitserklärung 44

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 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 2.2.7 [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 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

//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 Lösungsansaätze, der Evaluation und die Besprechung nach der Zwischenpräsentation, wurden die Meilensteine geändert und die Planung anderst 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 Erkentnisse 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 Literaturstudiuen, geprüfte Tools, der aktuelle Stand der Ethereum Blockchain, sowie die konzipierten Lösungsansätze und derern 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 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

//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 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] entgegengewirkt. 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

//TODO Kommentar/Besprechen–> Nicht nur Vertargsbedingungen (Dies ist nur ein Anwendungsfall), sondern auch anderes Programm

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.

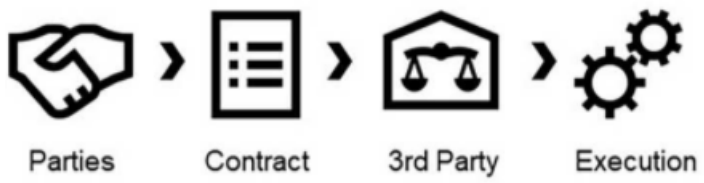

Abbildung 2.1: Ein traditioneller Vertag[10]

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.

//TODO evtl hier erwähnen, dass falls der Smart Contarct einen anderen aufruft, wird eine neue Transaktion geschickt, mit der Sender ID der Smart Contract Adresse und nicht der ursprünglichen ID.

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 [14]

Damit eine Transaktion in die Blockchain aufgenommen werden kann, muss sie signiert[15] sein. Dies kann offline beim Benutzer gemacht werden. Die signierte Transaktion wird dann an die Blockchain übermittelt.

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 **??**, 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. Dieser besteht aus einer Adresse, einem öffentlichen und einem geheimen Schlüssel. Es gibt zwei Arten von Accounts, solche von Benutzern und jene von Smart Contracts. Ein Account ermöglicht es einem Benutzer oder Smart Contract, Ether zu empfangen und zu senden.

//TODO Kommentar kann auch was anderes gesendet werden (Transaktionen für Interaktion mit Programm und nicht für Währung)

2.2.5.1 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.

2.2.5.2 Öffentlicher Schlüssel

Der öffentliche Schlüssel wird aus dem geheimen Schlüssel abgeleitet. Für die Generierung wird Keccak[16] verwendet, ein „Elliptical Curve Digital Signature Algorithm“[17]. Der öffentliche Schlüssel wird verwendet um die Signatur einer Transaktion zu verifizieren.

2.2.5.3 Adresse

Die Adresse wird aus dem öffentlichen Schlüssel abgeleitet. Es wird SHA3[18] 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.

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.

//TODO diese Typen kurz erklären oder nicht, weil nicht wichtig für unserer Arbeit?

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. [19], [20], [21]

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. [22], [23], [24] //TODO ..

2.2.7 Denial of Service (DoS) Attacken

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 unverfügbar für legitime Anfragen.

Zurzeit sind Blockchains noch relativ langsam bei der Verarbeitung von Transaktionen. Ethereum kann ungefähr 15 Transaktionen pro Sekunde abarbeiten.[25] Dadurch ist ein möglicher Angriffsvektor, die Blockchain mit Transaktionen zu fluten. Das würde dazu führen, dass Benutzer sehr lange auf 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 tedentiell steigen. Um sicherzustellen, dass seine Transaktionen weiterhin zuverlässig in die Blockchain aufgenommen werden, muss der Angfreifer 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.[26]

2.2.7.1 DoS Attacke identifizieren

Auf der Blockchain der FHNW existiert eine priviligierte 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 priviligierten Benutzer überwacht werden. Falls einer dieser Benutzer eine DoS Attacke einleitet, muss das frühst möglich erkannt und unterbunden werden können.

//TODO Möglich Vorgehensweisen //

Transaktionslimite pro Benutzer //TODO Benutzer oder Account? Jeder Benutzer darf nur eine gewisse Anzahl von gratis Transaktionen pro Zeiteinheit tätigen. Beim Überschreiten des Limits, wird er von der Whitelist gelöscht und muss die Transaktionsgebühr zahlen.

// TODO Muss wieder von der FHNW oder dem Algorithmus nach einem Tag ? hinzugefügt werden.

2.2.8 Whitelist von Parity

Der Client Parity hat eine Whitelist Funktionalität. Die Liste ist als Smart Contract geschrieben. Im Genesisblock[27] wird der Bytecode des Smart Contracts an der gewünschten Adresse hinterlegt. In der Liste können Accounts hinterlegt werden. Diese geniessen das Privileg, gratis Transaktionen tätigen zu dürfen. Dabei wird nur geprüft, ob der Sender einer Transaktion mit einem Gas Price von Null, sich in der Whitelist befindet. Ist er das, wird die Transaktion vom Node akzeptiert. Befindet sich der Account nicht in der Whitelist, wird die Transaktion vom Node abgelehnt. Die Whitelist wird initial von der FHNW mit Accounts befüllt. Die FHNW verfügt über einen Account, der berechtigt ist die Liste notfalls anzupassen. //TODO FHNW kann nur hinzufügen, jedoch nicht entfernen Ein weiterer Account, der die Liste anpassen kann, wird vom entwickelten Schutzmechanismus kontrolliert. So kann bei einer Bedrohung, der bösartige Account von der Liste entfernt werden.

2.3 Lösungsansätze

//TODO Spellcheck über ganze Seite

2.3.1 Lösungsansatz 1: Smart Wallet

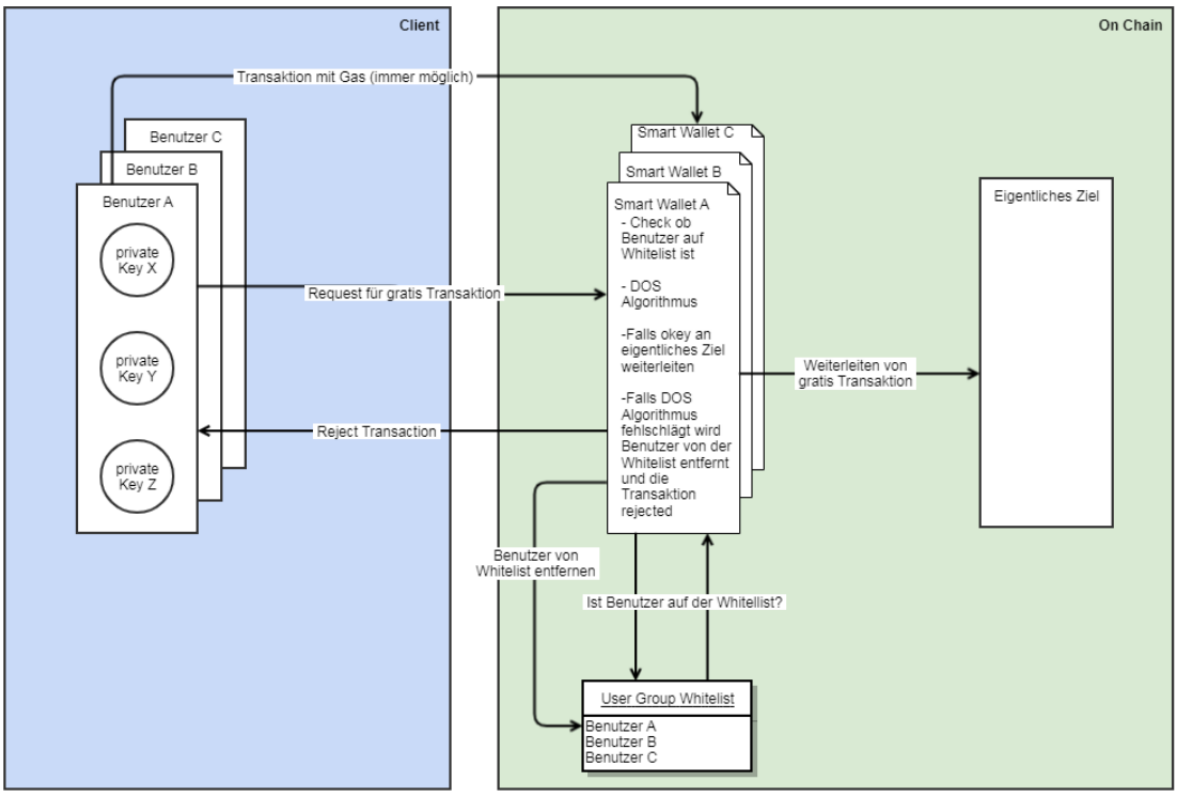

Abbildung 2.3: Lösungsansatz1

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, um für den Benutzer keine Anfangskosten für die Transaktion zu generieren. Die Whitelist wird von der Blockchain (Parity) verwaltet. Auf der Whitelist sind alle Benutzer aufgelistet die berechtigt sind gratis Transaktionen durchzuführen. Benutzer können in diese Liste von den Admins hinzugefügt werden. Gelöscht werden Sie nur von der Smart Wallet. 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. Es muss geprüft werden ob die Benutzer automatisiert wieder in die Liste hinzugefügt werden können.

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 Blockchain nicht resetted wird

2.3.1.4 Prozessworkflow

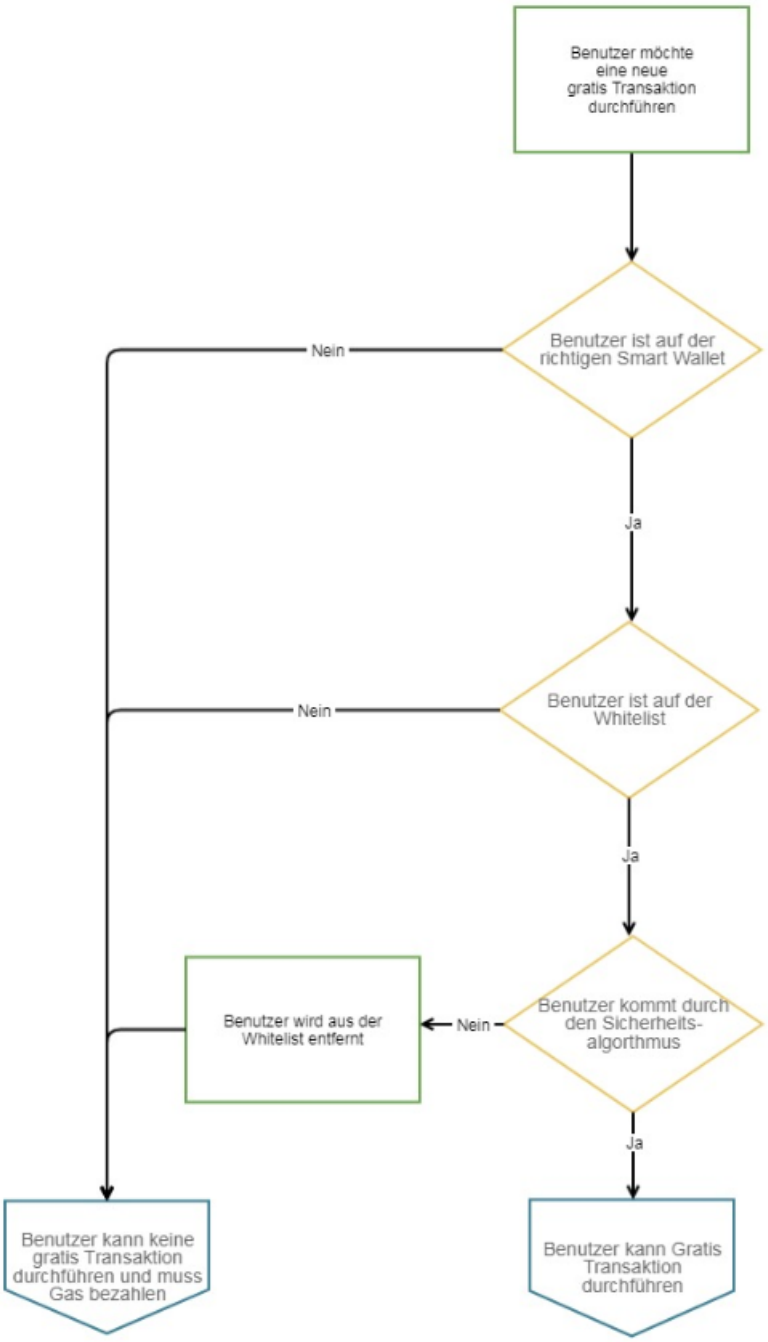

Abbildung 2.4: Flowchart Lösungsansatz 1

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

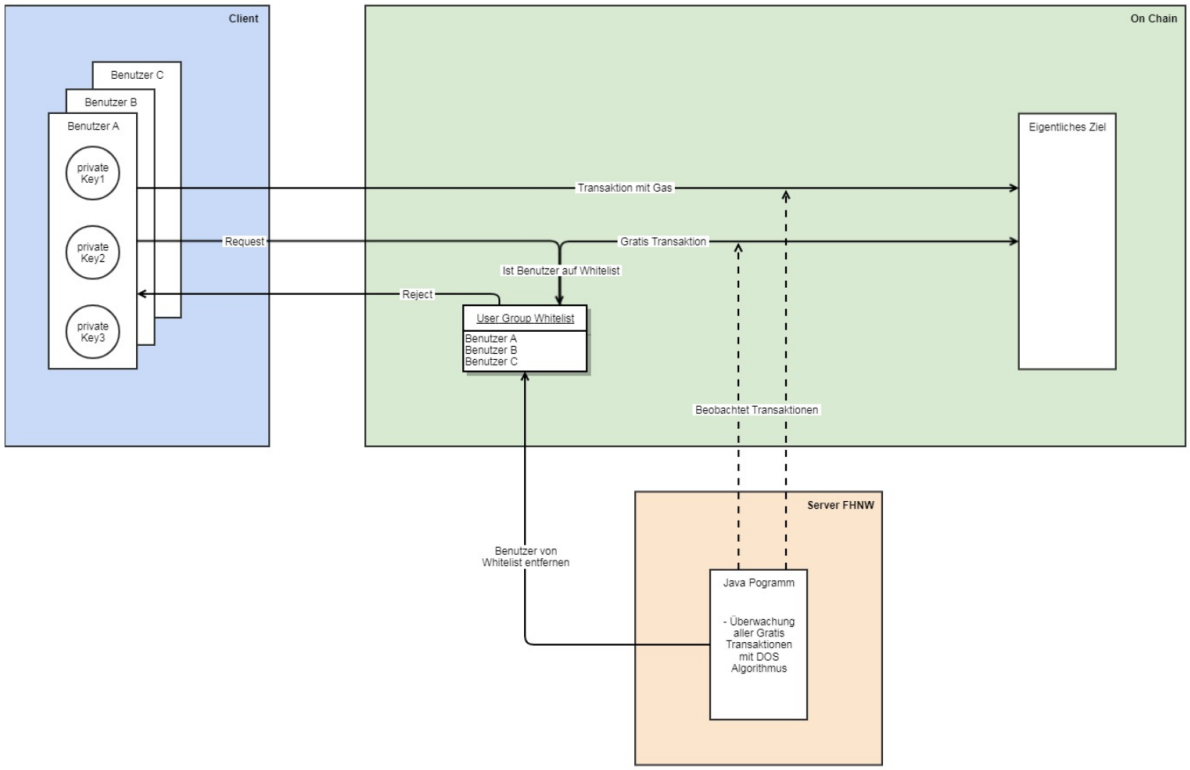

Abbildung 2.5: Lösungsansatz 2

2.3.2.1 Hauptlösungsansätze

Dies ist ein Lösungsansatz ohne Smart Wallet. Die Whitelist wird von der Blockchain (Parity) 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. Die Benutzer werden zu Beginn von den Admins in die Liste hinzugefügt. Gelöscht werden sie nur durch den Schutzalgorithmus. 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. Es muss geprüft werden ob die Benutzer automatisiert wieder in die Liste hinzugefügt werden können.

2.3.2.2 Pro

- Sicher machbar
- Nach dem Anpassen vom Schutz Algorithmus in der Smart Wallet, muss keine neue Smart Wallet deployed werden, oder die Blockchain resetted werden.

2.3.2.3 Contra

- Mehrere Komponenten
- Zentrale Autorität

2.3.2.4 Prozessworkflow

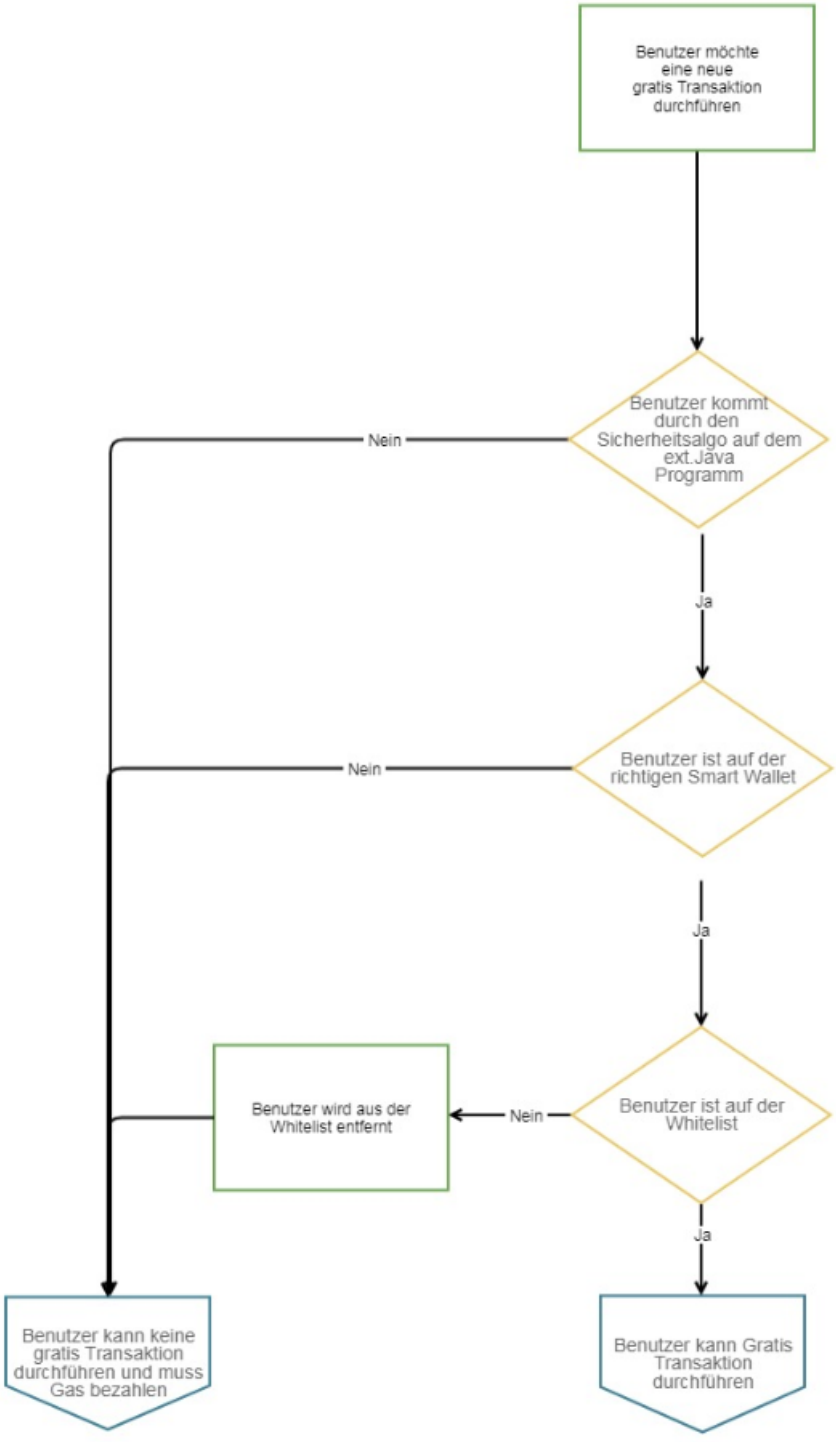

Abbildung 2.6: Flowchart Lösungsansatz 2

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

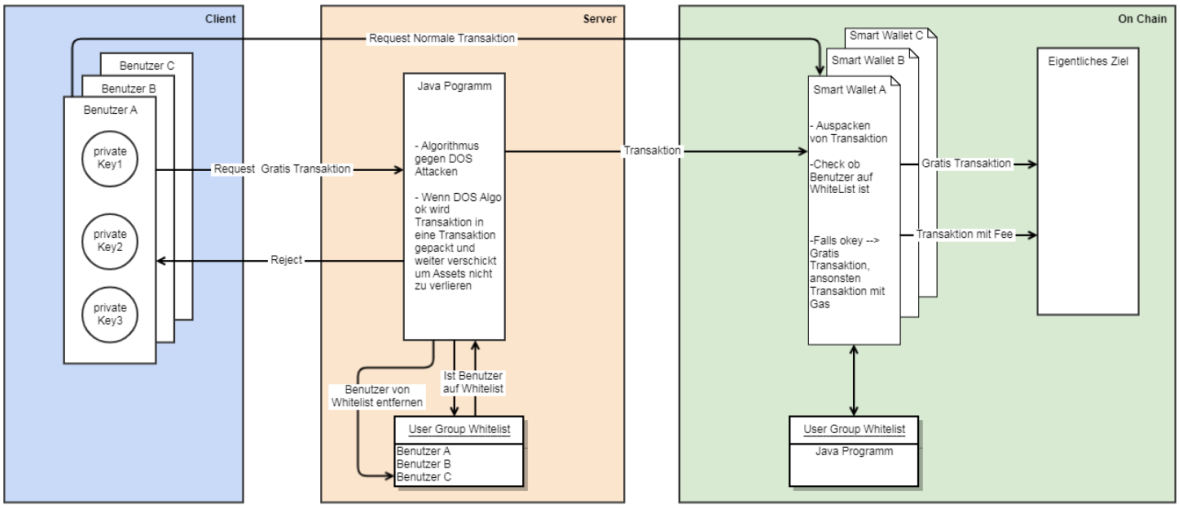

Abbildung 2.7: Lösungsansatz 3

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

- 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

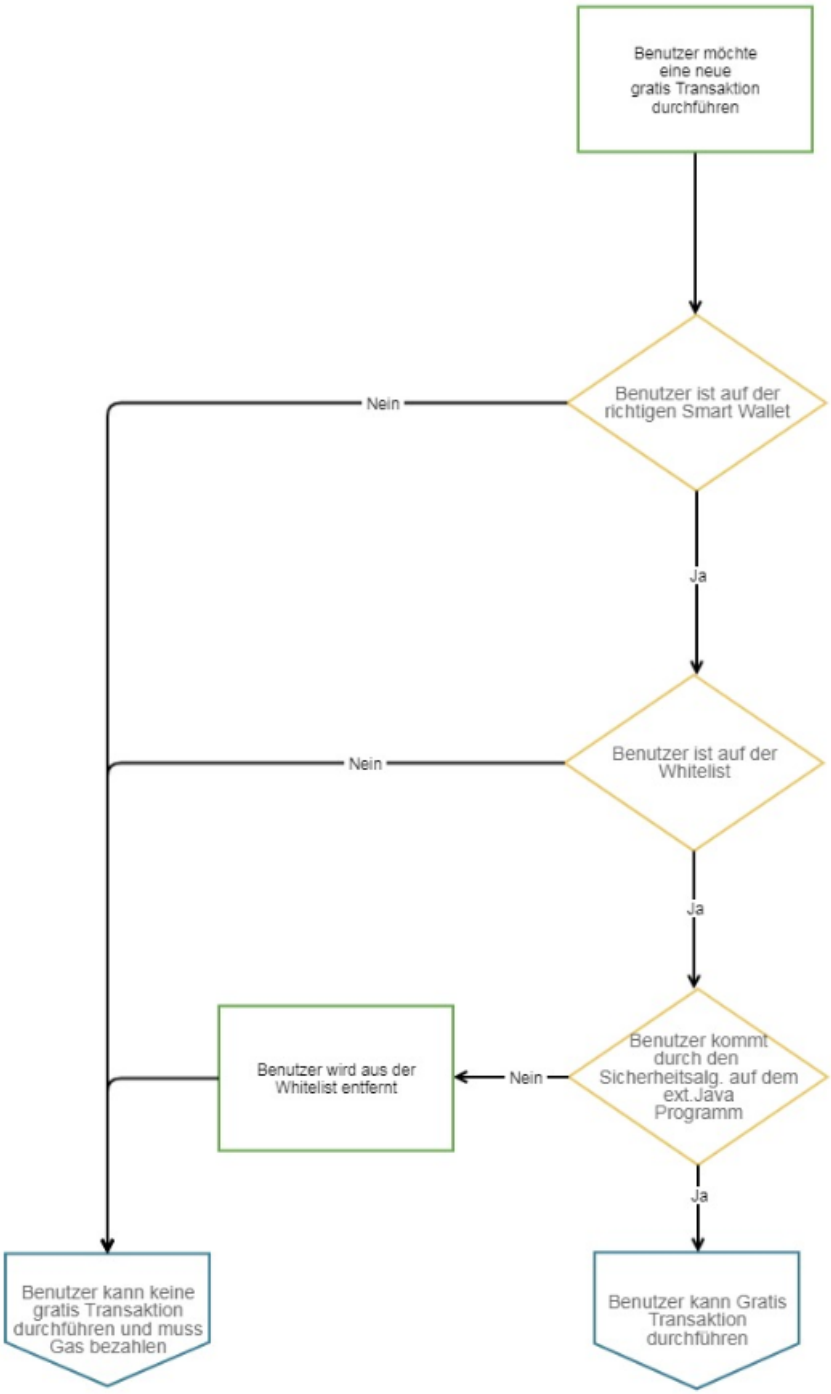

Abbildung 2.8: Flowchart Lösungsansatz 3

2.3.4 Lösungsansatz 4: Super Smart Wallet

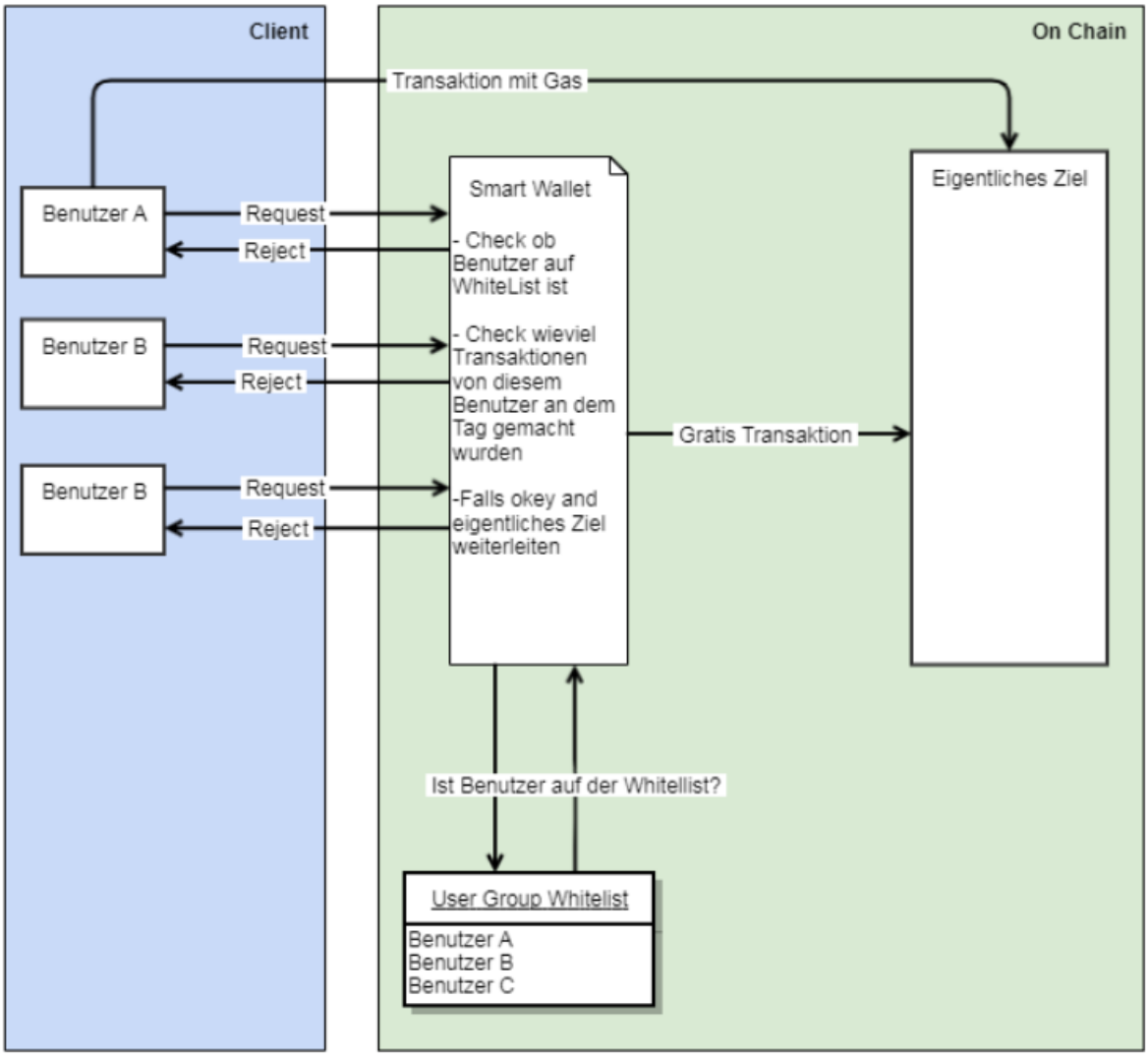

Abbildung 2.9: Lösungsansatz 4

2.3.4.1 Hauptlösungsansätze

Dieser Lösungsansatz war der erste konzipierte Lösungsansatz. 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. Der Benutzer wirdvon den Admins zu Beginn in diese Liste hinzugefügt. 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. Es muss geprüft werden ob die Benutzer automatisiert wieder in die Liste hinzugefügt werden können.

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?)

2.4 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	1	?	2	

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

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 zu Lösungsansatz 3. 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 zweiten Platz, zusammen mit Lösungsansatz 1. Da diese Lösung auch ein Javaprogramm wie Lösung 2 vorsieht, wird sie weniger prioritär wie Lösungsansatz 1 betrachtet.

2.4.4 Lösungsansatz 4: Super Smart Wallet

Dieser Lösungsansatz ist laut Evaluation auf dem letzten Platz. Dies war von Anfang an klar. Dieser Ansatz wurde jedoch aufgezeigt, da er das erste Lösungskonzept war.

2.5 Lösungsansätze

//TODO Spellcheck über ganze Seite //Benutzer != Account –> Auf der Whitelist sind Accounts !

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

2.5.1 Lösungsansatz 1: Smart Wallet

Es wird selbst eine Smart Wallet entwickelt. Diese benötigt die volle Funktionalität einer herkömlichen 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 2.2.8 beschrieben, wird für die Betreibung der Blockchain der Client Parity mit einer Withelist verwendet.

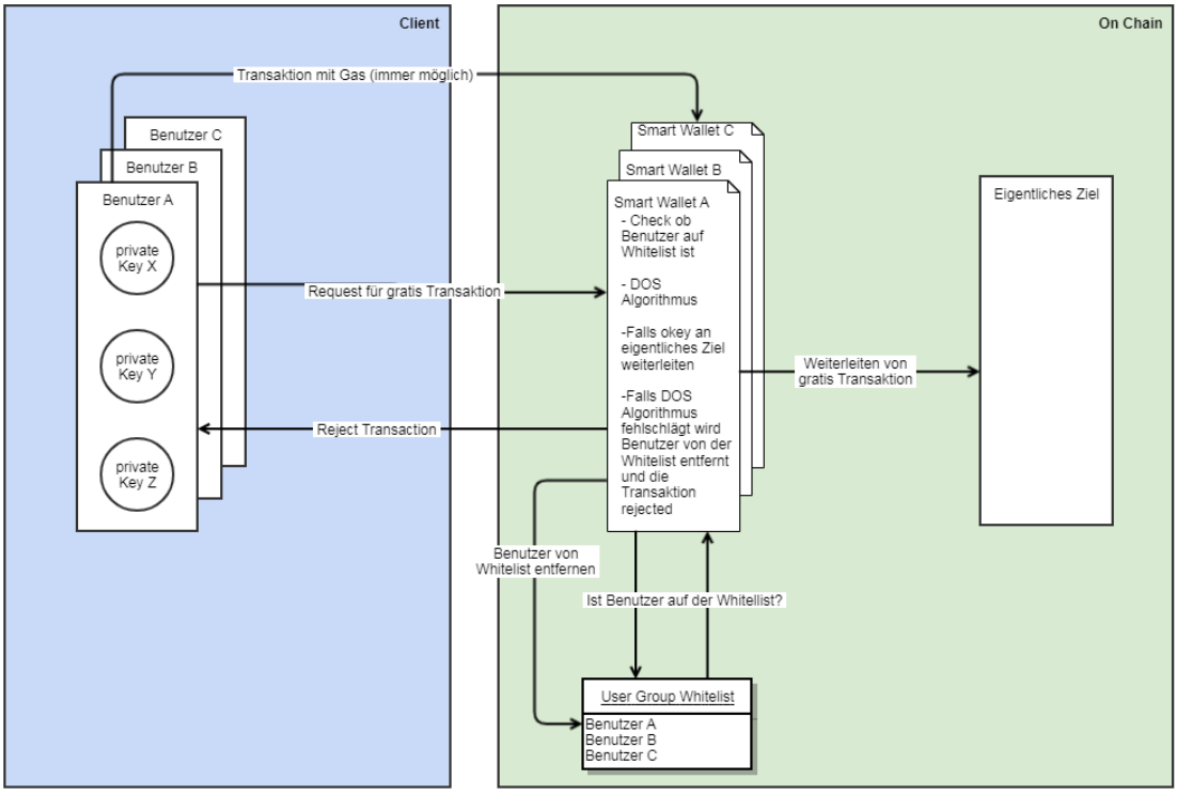

Abbildung 2.10: Lösungsansatz1

//TODO klären: Bezahlte Transaktionen dürfen überall hin, müssen nicht über wallet (Behauptung Jurij) //müssen nicht über die Wallet, aber nur so, haben bezahlte und gratis Transaktionen die gleiche Sender ID

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 2.10 dargestellt.

Wie in 2.2.8 beschrieben, prüft Parity bei einer gratis Transaktion nur, ob sich der Account in der Whitelist befindet. Das bedeuted, 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.

//TODO die verwaltung der Whitelist wird hier nicht erwähnt. Dass die FHNW Account iin die Liste aufnehmen kann, aber keine Löschen und was mit den gelöschten Accounts passiert. Muss nochmals

in sektion sec_whitelist kontrolliert werden, ob alles drin steht.

2.5.1.1 Pro

Als Vorteile des Lösungsansatzes 1 ist, dass die ganze Lösung auf der Blockchain läuft und somit dezentral ist. Diese Lösung ist die eleganteste Lösung, da sie die Prinzipien einer Blockchain einhält.

//TODO Text oder Auflistung

//vlt als text ausformulieren? –> Bei allen PRO und CONTRAS

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

2.5.1.2 Contra

Ein grosses Nachteil des Lösungsansatzes 1 ist, dass es unklar ist, ob er machbar ist. Um diesen Lösungsansatz umzusetzten, müsste der Blockchain Client Parity erweitert werden, um die Accounts zu zwingen über die Smart Wallet Transaktionen zu verschicken. Diese Erweiterungen müssten in einer weiteren Sprache entwickelt werden. Diese Änderungen macht den Lösungsansatz sehr komplex.t. Ein weiterer Nachteil ist, dass bei einer Änderung am DoS Schutzalgorithmus eine neue Smart Wallet für jeden Account deployed werden müsste. Des weiteren würde die alte Smart Wallet bestehen bleiben, ausser die Blockchain wird resetted.

//TODO Text oder Auflistung

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

2.5.1.3 Prozessworkflow

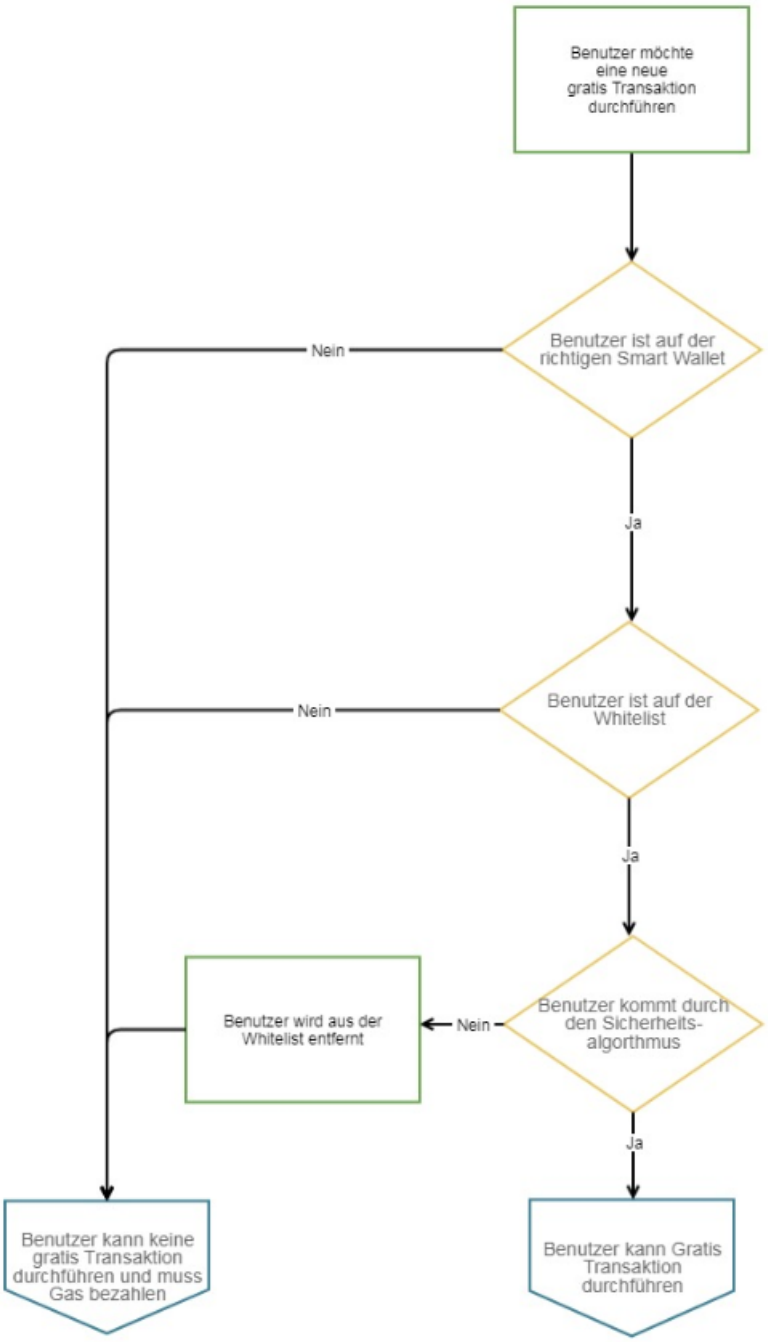

Abbildung 2.11: Flowchart Lösungsansatz 1

In der Abbildung 2.11 ist der Prozessablauf für eine gratis Transaktion dargestellt.

//TODO weitere Erläuterung?

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

Bei diesem Lösungsansatz wird auf die Entwicklung einer Smart Wallet verzichtet. Stattdessen wird der Schutzmechnismus gegen DoS Attacken mit einem Javaprogramm implementiert.

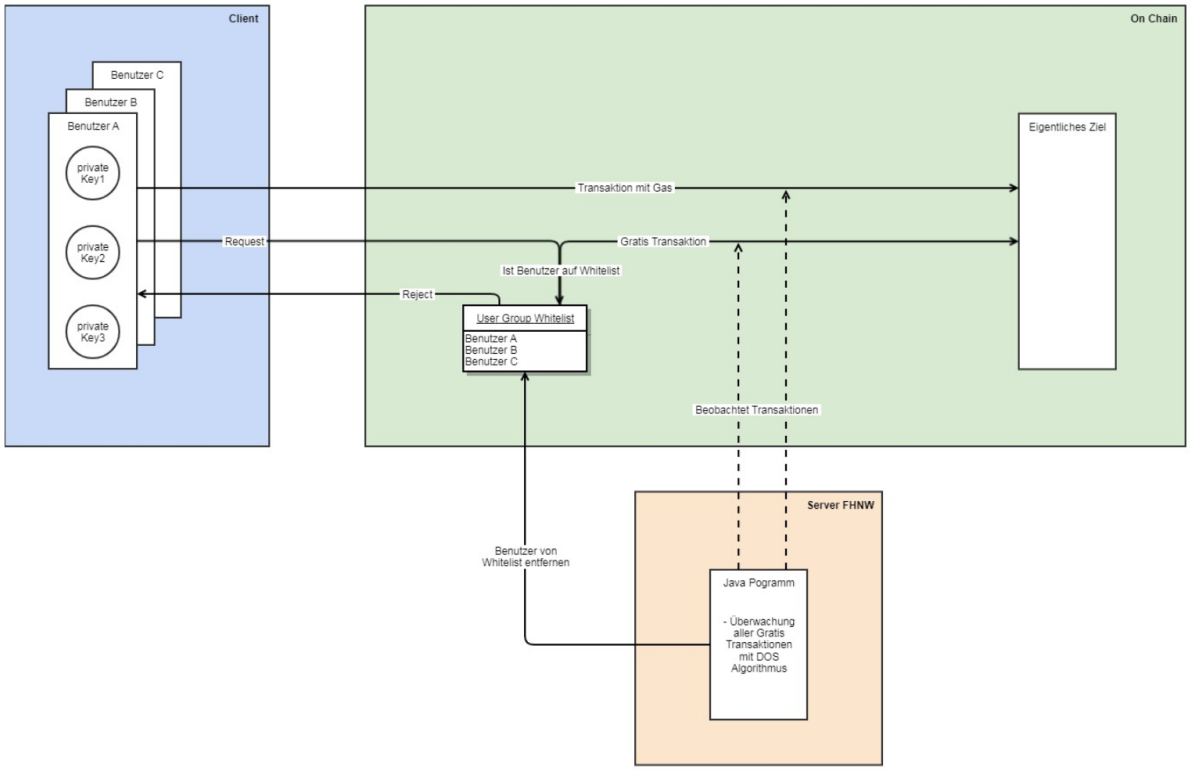

Abbildung 2.12: Lösungsansatz 2

Wie in Abbildung 2.12 ersichtlich ist, wird für diesen Lösungsansatz der DoS Schutzalgorithmus in einem externen Java Programm implementiert. Es wird auch für diesen Lösungsansatz die Whitelist von Parity verwendet, siehe 2.2.8. Der externe Sicherheitsalgorithmus überwacht getätigte gratis Transaktionen. Falls ein Account die Sicherheitsrichtlinien verletzt, wird dieser vom Algorithmus aus der Whitelist gelöscht.

//TODO wer verwaltet Whitelist? in Sec_whitelist kontrollieren

2.5.2.1 Pro

//TODO Text oder Auflistung

Ein klarer Vorteil des Lösungsansatzes 2 ist, dass er sicher umsetzbar ist. Die Elegantheit dieser Lösung besteht darin, dass bei einer Änderung des DoS Schutzalgorithmus nur das Java Programm neu deployed werden muss. Dies ist ein einmaliges Deployement und die Blockchain muss nicht resetted werden, um den alten Code zu entfernen.

- Sicher machbar
- Nach dem Anpassen vom Schutz Algorithmus in der Smart Wallet, muss keine neue Smart Wallet deployed werden, oder die Blockchain resetted werden.

2.5.2.2 Contra

Ein grosser Nachteil dieses Lösungsansatzes ist, dass durch die zentrale Autorität die Prinzipen der Blockchain verletzt werden. Somit wäre die Lösung nicht sehr elegant. Ein weiterer Nachteil ist, dass durch das Java Programm ein Server als Komponente hinzukommen würde und somit nicht alles auf einer Komponente läuft. Dies bringt weiteren administrativen Aufwand.

//TODO Auflistung oder Text

- Mehrere Komponenten
- Zentrale Autorität

2.5.2.3 Prozessworkflow

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

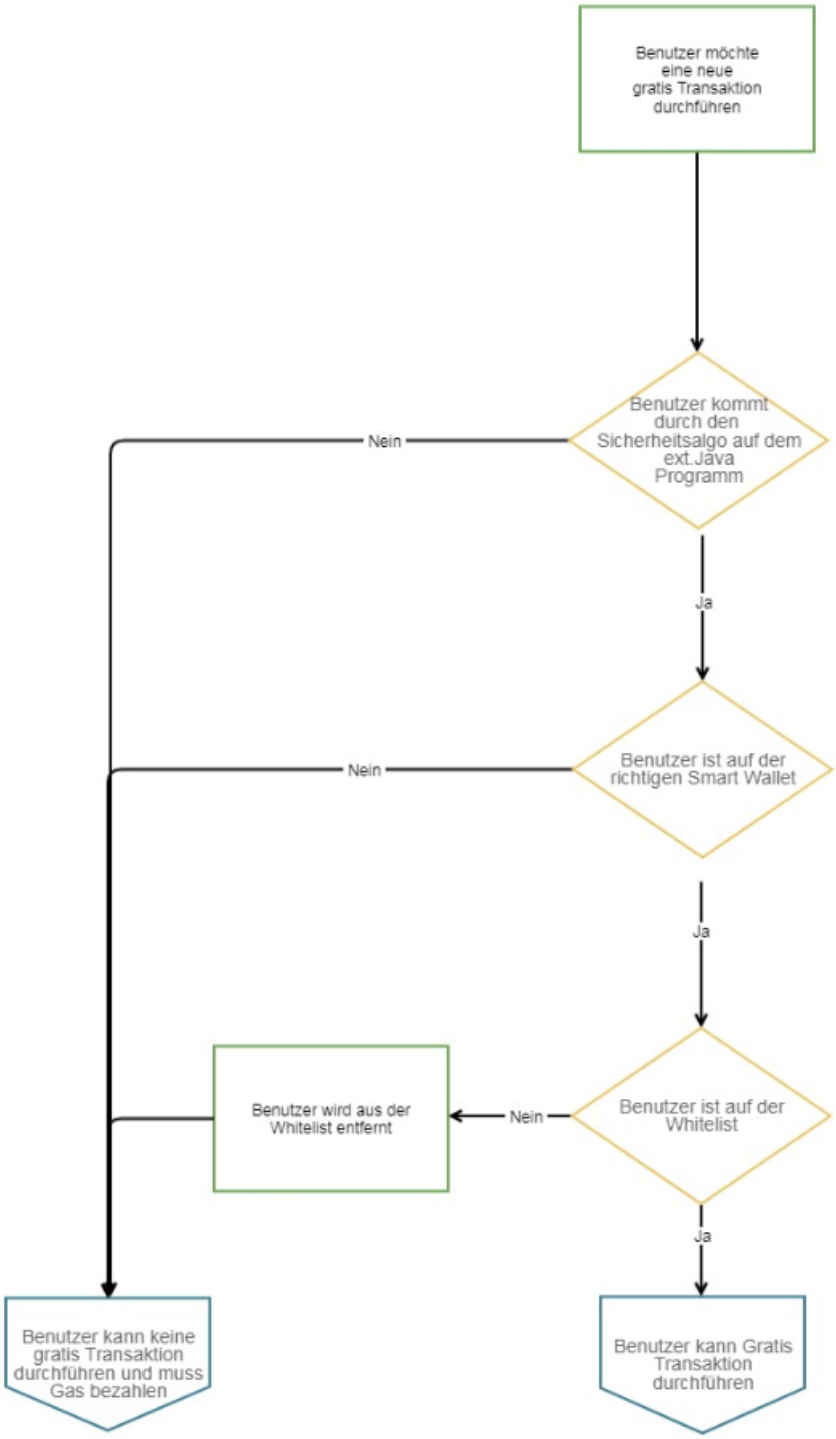

Abbildung 2.13: Flowchart Lösungsansatz 2

Auf dem Flowchart 2.13 dargestellt ist, kann ein Benutzer mit einem whitelisted Account direkt gratis

Transaktionen ausführen.

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

Wie in Abbildung 2.14 illustriert, ist der Blockchain ein Javaprogramm 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 der Benutzer die Sicherheitsrichtlinien verletzt, wird sein Account vom Algorithmus aus der eigenen Whitelist gelöscht. //TODO Benutzer oder Transaktion verletzt die Schutzrichtlinien?

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 Javaprogramm an die Smart Wallet gesendet.

//TODO wer verwaltet die Whitelist des Javaprogramms?

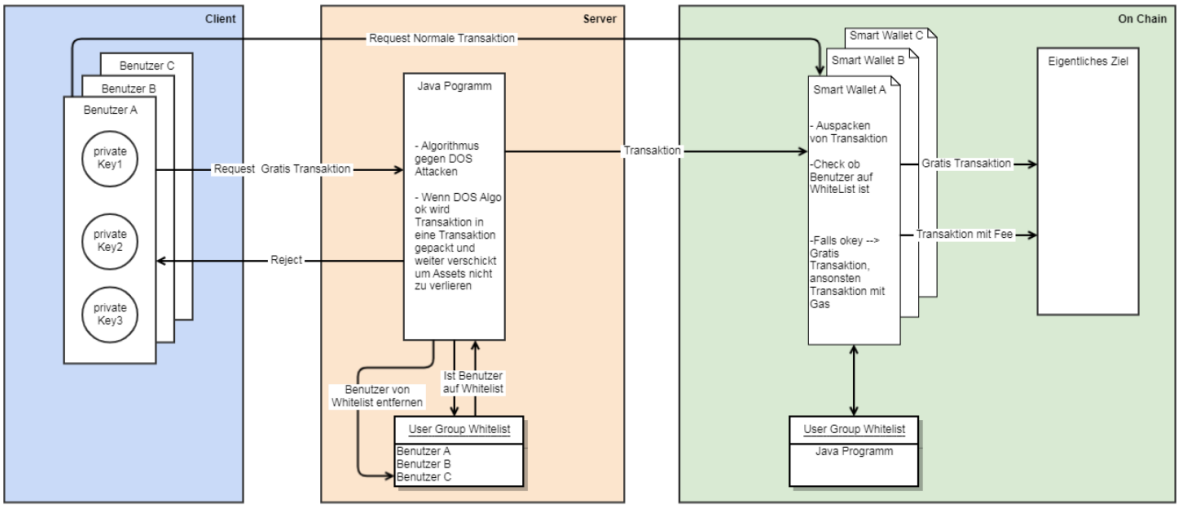

Abbildung 2.14: Lösungsansatz 3

Weiter wird eine Smart Wallet entwickelt. Diese ist nötig, um die verschachtelten Transaktionen des Javaprogramms 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 2.2.8 beschriebenen Whitelist ist nur der Account des Javaprogrammes aufgelistet. So ist sichergestellt, dass nur Transaktionen die vom Java Programm weitergeleitet werden,

kostenfrei durchgeführt werden können. Der Benutzer kann immer mit kostenpflichtigen Transaktionen auf die Smart Wallet zugreifen. Dies ist insbesindere wichtig, falls das Java Programm nicht aufrufbar ist, wenn z.B. der Server ausfällt.

2.5.3.1 Pro

Ein grosser Vorteil dieses Lösungsansatzes ist, dass alle gratis Transaktionen vom DoS Schutzalgorithmus geprüft werden, da sie alle über das Javaprogramm laufen müssen, weil dort die Whitelist der Accounts geführt wird. Ein weiterer grosser Vorteil ist, dass der DoS Schutzalgorithmus im Javaprogramm geändert werden kann und nur einmalig deployed werden muss und der alte nicht auf der Blockchain bestehen bleibt. Somit muss bei Änderungen die Blockchain nicht resetted werden. Das Besondere bei dieser Lösung ist, dass der DoS Schutzalgorithmus die unerwünschten Transaktionen blockt bevor sie auf die Blockchain treffen.

//TODO Text oder Auflistung

- 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

2.5.3.2 Contra

Durch die zentrale Autorität werden bei diesem Lösungsansatz die Prinzipien der Blockchain verletzt. Ein weiterer Nachteil wäre die weitere Komponente die durch das Java Programm und dessen zusätzlichen Server anfallen.

//TODO Text oder Auflistung

- Mehrere Systeme
- Zentrale Autorität

2.5.3.3 Prozessworkflow

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

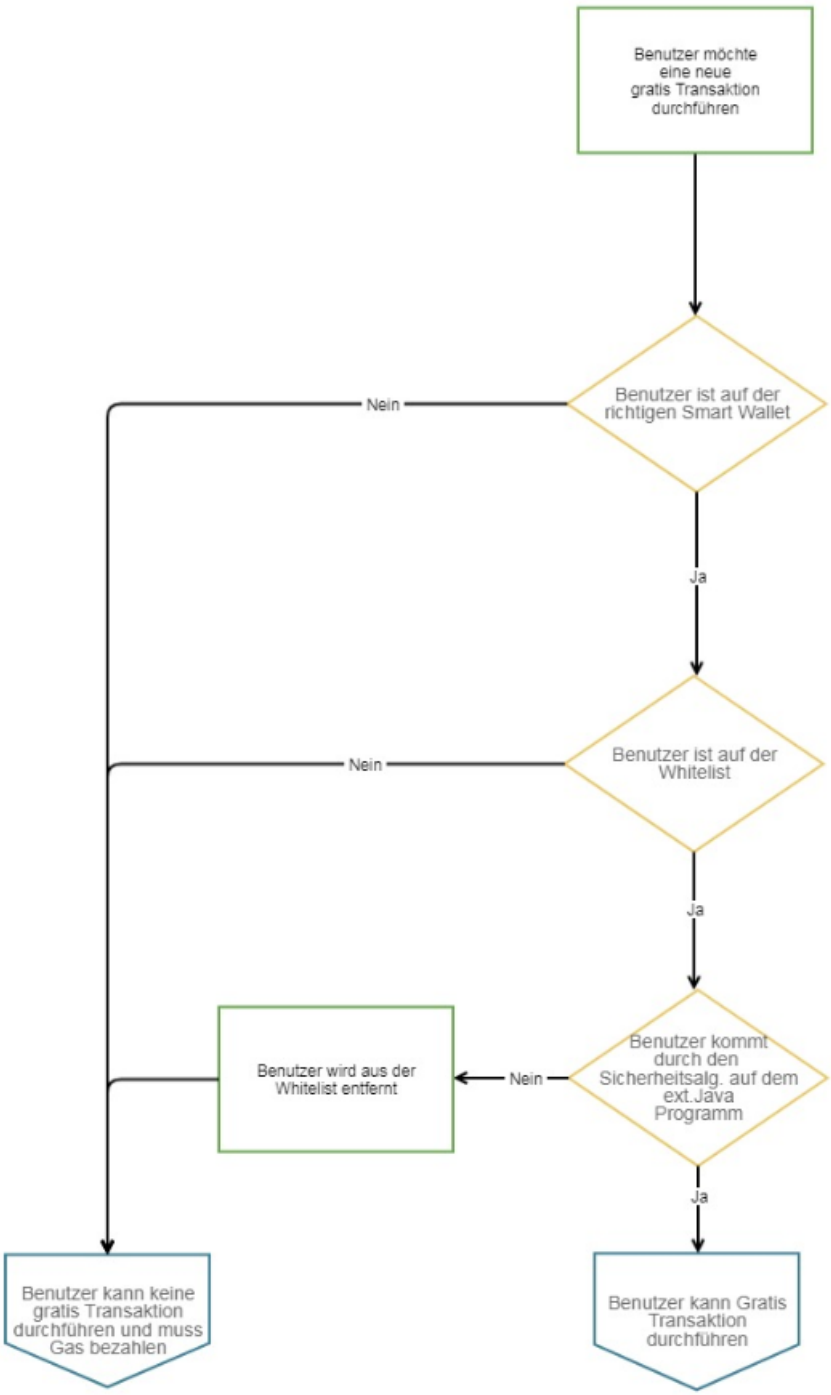

Abbildung 2.15: Flowchart Lösungsansatz 3

Die Abbildung 2.15 zeigt, dass alle gratis Transaktionen in erster Instanz von einem Javaprogramm geprüft werden. Falls keine Richtlinien verletzt werden, wird die Transaktion im Data-Feld einer neu

generierten Transaktion an die Smart Wallet übermittelt.

2.5.4 Lösungsansatz 4: Super Smart Wallet

Es wird eine zentrale Smart Wallet entwickelt. Im Gegensatz zu Lösungsansatz 1, 2.5.1, 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 2.2.8 beschriebenen Whitelist gearbeitet.

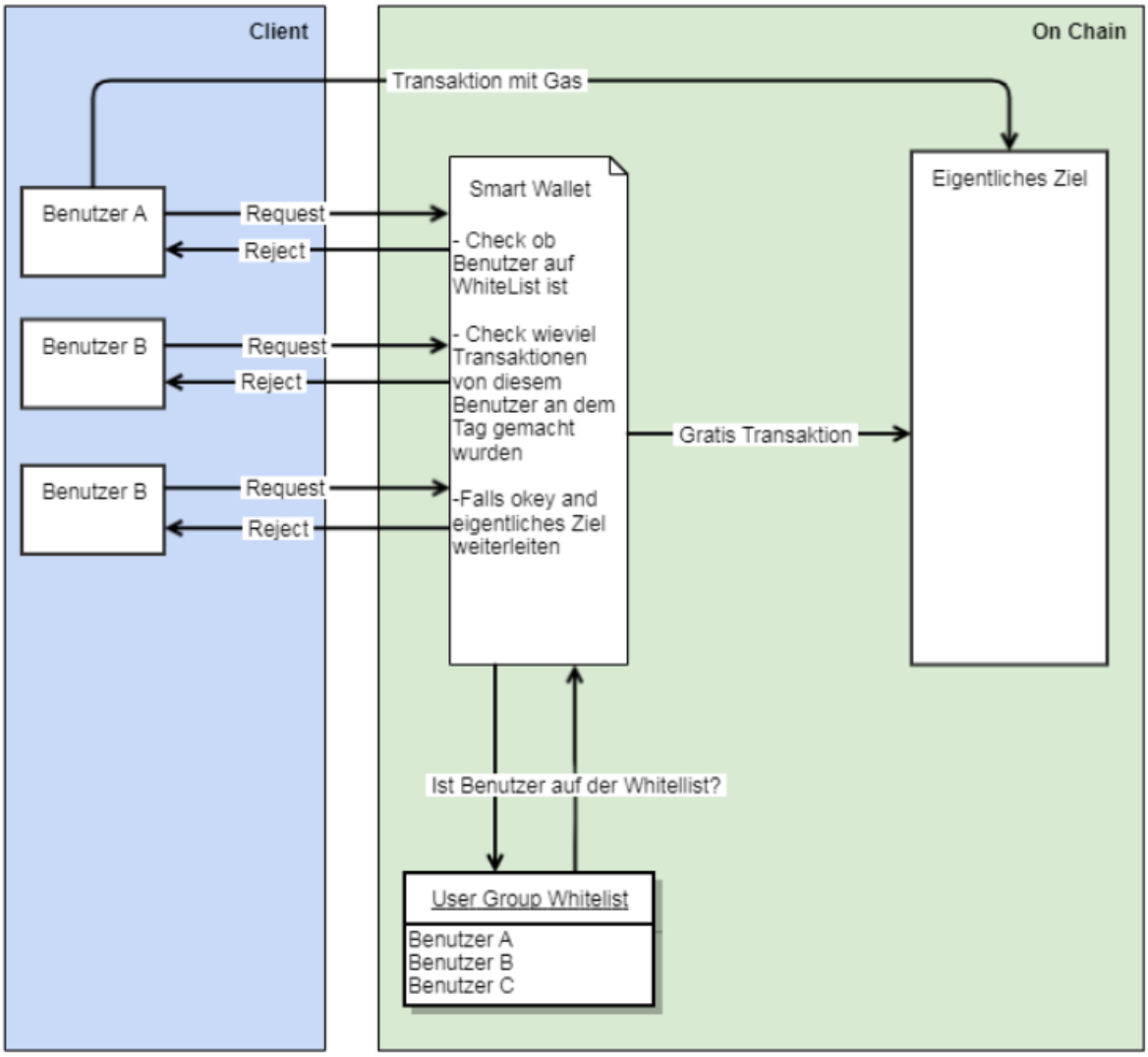

Abbildung 2.16: Lösungsansatz 4

Die Smart Wallet verwaltet die Whitelist und den Schutzmechanismus gegen DoS Attacken. Das ist auf Abbildung 2.16 ersichtlich.

//TODO nicht die Smart Wallet, sondern die Blockchain verwaltet die Whitelist?

2.5.4.1 Pro

Vorteil dieses Lösungsansatzes ist, dass nur eine Smart Wallet deployed werden muss und nicht für jeden Benutzer einzeln eine zur Verfügung gestellt werden muss. Dies bringt weniger administrativer Aufwand.

//TODO Text oder Auflistung

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

2.5.4.2 Contra

Bei diesem Lösungsansatz ist nicht klar, ob er umsetzbar ist. Die Blockchain muss umgeschrieben werden, da nicht die Senderidentität der Smart Wallet genutzt würde, sondern die des Accounts. Des weiteren ist es bei dieser Lösung möglich, gratis Transaktionen nicht über die Smart Wallet zu schicken und somit den DoS Schutzalgorithmus zu umgehen.

//TODO auflistung oder Text

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

2.6 Evaluation der Lösungsansätze

//TODO muss ausgeschrieben werden, das ist ein zentraler Punkt in unserer Arbeit.

Die Lösungsansätze wurden anhand folgender Evaluationskriterien bewertet:

- 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 2.2: Evaluation Lösungsansätze

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

2.6.1 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 zu Lösungsansatz 3. Falls die Kapazitäten ausreichen, wird sie somit auch implementiert.

2.6.2 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.

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

Laut Evaluation ist dieser Lösungsansatz auf dem zweiten Platz, zusammen mit Lösungsansatz 1. Da diese Lösung auch ein Javaprogramm wie Lösung 2 vorsieht, wird sie weniger prioritär wie Lösungsansatz 1 betrachtet.

2.6.4 Lösungsansatz 4: Super Smart Wallet

Dieser Lösungsansatz ist laut Evaluation auf dem letzten Platz. Dies war von Anfang an klar. Dieser Ansatz wurde jedoch aufgezeigt, da er das erste Lösungskonzept war.

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] K. Tam, „Transactions in Ethereum - KC Tam - Medium“, 2019. [Online]. Verfügbar unter: https://medium.com/@kctheservant/transactions-in-ethereum-e85a73068f74.

[15] Y. Riady, „Signing and Verifying Ethereum Signaturs - Yos Riady“, 2019. [Online]. Verfügbar unter: https://yos.io/2018/11/16/ethereum-signatures/.

[16] „https://keccak.team/“, 2019. [Online]. Verfügbar unter: Keccak%20Team.

[17] „Elliptic Curve Digital Signature Algorithm - Wikipedia“, 2019. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm.

[18] „SHA-3 - Wikipedia“, 2019. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/SHA-3.

[19] „Crypto Wallet Types Explained | Binance Academy“, 2019. [Online]. Verfügbar unter: https://www.binance.vision/blockchain/crypto-wallet-types-explained.

[20] M. Wachal, „What is a blockchain wallet? - SoftwareMill Tech Blog“, 2019. [Online]. Verfügbar unter: https://blog.softwaremill.com/what-is-a-blockchain-wallet-bbb30fbf97f8.

[21] 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.

[22] 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.

[23] E. Conner, „smart Wallets are Here - Gnosis“, 2019. [Online]. Verfügbar unter: https://blog.gnosis.pm/smart-wallets-are-here-121d44519cae.

[24] 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.

[25] „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.

[26] 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.

[27] „Genesis block - Bitcoin Wiki“, 2019. [Online]. Verfügbar unter: https://en.bitcoin.it/wiki/Genesis_block.

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

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

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

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

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

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

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

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

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

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

[38] „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[28] als Client den Ansprüchen genügt oder ob ein anderer Client (z.B. Parity[29], Aleth[30], etc.) zum Einsatz kommt.

Trufflesuite Trufflesuite[31] 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[32]
- Metamask[33]
- Atomic Wallet [34]
- Exodus[35]

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[36]

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[37]
- Vyper[38]

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 _____