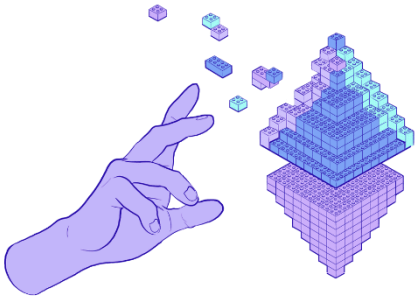
Fernfachhochschule Schweiz
Mitglied der SUPSI

Fernfachhochschule Schweiz
SemA Inf FS22
BSc Informatik

Dezentralisierte Smart-Contracts

Ausbruch aus der Blockchain mittels Oracles



Seminararbeit im Studiengang Informatik

von

Luca Hostettler

Eingereicht bei:

Prof. Dr. Beatrice Paoli,
Dr. Urs-Martin Künzi
Departement Informatik

Zürich, 13.07.2022

Zusammenfassung

Die vorliegende Seminararbeit soll einen Überblick über Blockchain, Smart-Contracts und Oracles geben. Mit der Blockchain kann ein Smart Contract als Adresse beschrieben werden. Als Ergebnis der Dateneingabe und der Integration von Geschäftslogik kann eine Ausgabe generiert werden. In den meisten Fällen werden Smart Contracts verwendet, um wiederkehrende Transaktionen auf einer Blockchain durchzuführen, wie beispielsweise die Überweisung von Geld.

Smart Contracts können Daten von der Blockchain anfordern, um eine Transaktion durchzuführen. Sie können feststellen, ob ein Debit Konto über ausreichende Mittel verfügt oder gewisse Vorbedingungen erfüllt sind. Die Nodes überprüfen dann die im Block enthaltenen Transaktionen (Miner). Wie alle anderen Transaktionen auf der Blockchain müssen auch die Smart Contracts deterministisch sein. Dieselbe Ausgabe muss zu verschiedenen Zeiten mit derselben Eingabe an mehrere Nodes gesendet werden.

Ebenso ist es notwendig, Daten aus der realen Welt abzufragen, um die Blockchain für mehr als nur ein Transaktionsbuch zu nutzen. Daher gibt es jetzt mehr Szenarien, in denen Anwendungen oder Verträge auf der Blockchain gespeichert werden können. Da diese Schnittstelle nun vorhanden ist, ergeben sich jedoch zahlreiche Fragen und Zweifel. Unterschiedliche Antworten von der realen Schnittstelle machen den Vertrag angesichts der vorhersehbaren Ergebnisse von Smart Contracts beispielsweise nicht reproduzierbar.

Im Rahmen der Seminararbeit wird eine Web3-Applikation mit einem Smart-Contract im Ethereum Mainnet-Netzwerk entwickelt, getestet und veröffentlicht welcher Daten aus der echten Welt via einem Oracle abfragt.

Inhaltsverzeichnis

1 Einleitung

Die Finanzmärkte haben seit der Finanzkrise im Jahr 2008 mit der Entwicklung und Förderung der digitalen Kryptowährungen reagiert. Diese gelten als Alternative zum bisherigen Zentralbankgeld [1]. Allen voran geht der Bitcoin, welcher das verloren gegangene Vertrauen in das klassische Währungssystem kompensieren soll.

Diese Entwicklung wurde ermöglicht durch eine Technologie, die sich Blockchain nennt. Auf der Blockchain können Vorgänge dargestellt und erläutert werden. Inzwischen ist die Blockchain aber auch ausserhalb der Finanzwelt nützlich. Neue Trends zeigen, dass die Technologie zudem in Handelssystemen für Diamanten oder Rohstoffe eingesetzt werden können. Zum Abschluss des Jahres 2016 fand erstmals ein in grenzüberschreitendes Rohstoffgeschäft per Blockchain statt [2].

Um die Blockchain und ihre Eigenschaften besser zu verstehen, soll im Folgenden der Unterschied zu klassischen Finanzsystemen dargestellt werden. Zu bedenken bleibt dabei, dass diese alternative Währung eingeführt wurde, um unabhängig von Regierungen und Banken zu agieren. Das Konzept der Kryptowährung soll in der Verwendung der Blockchain erläutert werden.

Insbesondere soll im Rahmen dieser Arbeit die Ethereum Blockchain sowie deren Smart Contracts untersuchen. Als Arbeitsdefinition sollen die Smart-Contracts als eine Adresse, welche in der Blockchain eingebettet wird, gelten. Technisch betrachtet verfügt ein Smart Contracts über eine Eingabe, eine Ausgabe und gegebenenfalls eine Geschäftslogik. Meistens werden Smart-Contracts verwendet, um gewisse repetitive Transaktionen auf der Blockchain zu definieren. Smart-Contracts können, um eine Transaktion durchzuführen Daten aus der Blockchain abfragen, um zum Beispiel zu schauen, ob eine zu belastende Adresse noch über genügend Ressourcen verfügt. Diese Transaktionen werden in einem Block gespeichert, welcher danach von den Worker-Nodes (Miners) verifiziert wird. Smart-Contracts, wie auch alle anderen Transaktionen auf der Blockchain müssen deterministisch sein.

Verschiedene Worker-Nodes müssen zu verschiedenen Zeiten mit derselben Eingabe immer dieselbe Ausgabe erhalten. Um die Blockchain nicht nur für ein Buch von Transaktionen zu verwenden, kamen später die Anforderung Daten aus der echten Welt abzufragen. Somit werden nun mehr Szenarien möglich, um Applikationen oder Verträge auf der Blockchain zu speichern. Mit dieser Schnittstelle folgen aber nun auch diverse Fragen und Unsicherheiten. Gerade in Anbetracht auf die deterministischen Resultate eines Smart-Contracts könnten nun zum Beispiel verschiedene Antworten von der Schnittstelle der echten Welt kommen, was den Vertrag nicht reproduzierbar macht.

1.1 Fragestellung

Um die Zukunft der Smart Contracts zu betrachten, soll die vorliegende Arbeit die Kommunikationsschnittstelle von Smart Contracts in der Ethereum-Blockchain mit Ereignissen der echten Welt untersuchen. Anhand der folgenden Fragen soll diese Untersuchung vollzogen werden:

- **Wie können Smart Contracts Daten aus der echten Welt abfragen?**
- **Gibt es ein Sicherheitsrisiko beim Abruf von Non-Blockchain Daten?**
- **Wie werden Antworten eines Oracles von verschiedenen Nodes verifiziert?**
- **Was passiert, wenn sich die Daten zwischenzeitlich verändern?**

1.2 Vorgehen

Im Theorieteil dieser Arbeit soll die vorhandene Literatur einem Review unterzogen werden, so dass im Ergebnis eine Zusammenfassung des aktuellen Forschungsstandes zur Thematik Smart Contracts vorliegt. Der empirische Teil dieser Arbeit dient der Entwicklung eines eigenen Smart-Contracts, welcher die Daten aus der echten Welt abfragen kann. Welche Daten abgefragt werden oder was der Smart-Contract genau für eine Bindung hat, wird im Rahmen der Arbeit konkret ausgearbeitet. Es werden vorhandene Oracles untersucht und deren Methodik dokumentiert.
Das Vorgehen, um einen Smart Contract zu programmieren, soll mithilfe von Chainlink beschrieben werden. Bei Chainlink handelt es sich um eine auf der Ethereum-Blockchain basierende Anwendung. Ihre Funktion liegt in der Produktion von hochwertigen Daten für Smart Contracts. Damit können Smart Contracts mit Daten im Umfeld der Blockchain verknüpft werden [3].
Im Folgenden soll nun beschreiben werden, wie Smart Contracts auf der Ethereum Mainnet Blockchain sowie die Verwendung von Chainlink-Randomness-Generator-Feeds entwickelt werden.

Um mit der Programmierung zu starten, sind zunächst einige Vorbereitungen zu treffen:

- Installation von NodeJS v14 oder höher und NPM
- Installation von Git

Der in der Arbeit verwendete Code ist im internen GitLab der FFHS zu finden:

- https://git.ffhs.ch/luca.hostettler/sema

Die aktuell produktive Version der Applikation ist unter folgendem Link zu erreichen:

- https://sema.hostettler.io

2 Theoretische Grundlagen

2.1 Bitcoin und Ethereum

Nach der erfolgreichen Einführung des Bitcoins kam ein junger Informatiker namens Vitalik Buterin im Jahr 2013 auf die Idee auf dem dezentralen Netzwerk weitere Programme zu entwickeln. Zu beachten ist der Unterschied zwischen Bitcoin als Kryptowährung und der Ethereum Plattform. Bei Letzterer besteht die Möglichkeit durch Smart Contracts einen Vertrag zwischen zwei Parteien, ohne eine dritte Partei zu schliessen. Dies ist möglich, da Ethereum ein Netzwerk aus Computern und Smartphones darstellt. Buterin hatte zuvor bereits vier Jahre den Bitcoin genutzt und war von dessen Konzept überzeugt. Sein Ziel war es nun die Blockchain Technologie für sogenannte DApps (Decentralized Applications) zu entwickeln. Darunter waren Apps zu verstehen, die ebenfalls dezentral sind. Erste Versuche brachten ihn jedoch nicht direkt zu seinem Ziel. Heute als Omni bekannt, begann sein erster Versuch mit dem Mastercoin, der sich allerdings als unzureichend erwies. Dies ist der Ursprung des Stablecoins Tether. Buterin sah sich nach Optimierungsmöglichkeiten um, damit sein Ziel der Entwicklung von DApps doch noch wirklich werden konnte. Die Funktionsweise von DApps beruht auf einem Code, der auf einem dezentralen Netzwerk wie Bitcoin oder Ethereum arbeitet. Der Unterschied gegenüber üblichen Apps liegt genau darin, denn die Apps arbeiten auf zentralen Servern. Dadurch besteht dabei jedoch das Risiko der Manipulation oder Zensur. Damit kommt der Funktionsweise der DApps ein hoher Sicherheitsaspekt zu. Bei einer Anwendung muss immer eine Problemstellung vorhanden sein. Dies kann beispielsweise eine Vertragsverhandlung zwischen zwei oder mehreren Parteien sein [5, S. 45].

Möchte ein Automobilhersteller einen Rahmenvertrag mit seinem Lieferant treffen, sind in der Regel mehrere Dokumente notwendig. Dazu gehört einerseits eine globale Vereinbarung (GVA) wie auch eine Datenschutzvereinbarung (DPA) und ein Service Level Agreement (SLA). Dokumentiert werden Leistungsumfang oder die Reaktionszeit. Bei der Bearbeitung über die DApp sind die Dokumente nur auf der Blockchain abgelegt. Dort sind sie verschlüsselt abgelegt. Es besteht nicht die Notwendigkeit die Dokumente via Mail auszutauschen, da jeder Teilnehmer die Dokumente auf seinem Rechner hat [6].

Zu diesem Zweck wird zunächst ein vordefinierter Vertrag aufgesetzt, der allen Anforderungen der beteiligten Parteien entspricht. Diese Informationen werden im Smart Contract integriert und können nur geändert werden, wenn alle Parteien ihre Zustimmung geben. Im nächsten Schritt muss das behandelte Ereignis eintreffen, so dass der Versicherungsschutz zum Tragen kommt. Dadurch werden die Bedingungen, welche im Vorvertrag festgehalten wurden, automatisch ausgelöst. Daraufhin wird eine Auszahlung getätigt und der umgehend und effizient Vorgang abgeschlossen werden (siehe Abbildung 1).

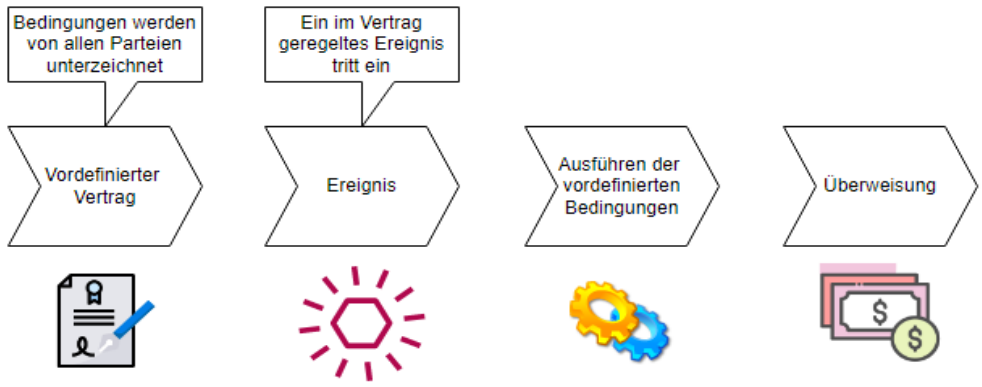


Abbildung 1 - Prozess Vertragswesen

Noch im selben Jahr entwickelte Buterin dann das Whitepaper für Ethereum. Darin wird sowohl das Design des Protokolls als auch die dazugehörige Architektur sorgfältig beschrieben. Ende des Jahres 2013 fand er sich mit weiteren Entwicklern zusammen und startete Ethereum [7].

Zu den Entwicklern gehörten auch Gavin Wood und Joseph Lubin. Dies beeinflusste auch die Smart Contracts, denn diese konnten dadurch weiterentwickelt werden. Smart Contracts sind «Computerprotokolle», welche Verträge repräsentieren und den Vorgang der Vertragsbildung abkürzen können. Bei diesen «intelligenten Verträgen» bestehen die Bedingungen der Vereinbarung zwischen «Käufer» und «Verkäufer» aus Codezeilen, wie zum Beispiel bei Code-Snippet, ein kleiner Teil PHP-Code. Dies beruht auf der von Gavin Wood eigenständig entwickelte Programmiersprache Solidity. Sie besitzt die Eigenschaft der «Turing-Complete» Programmierbarkeit [5]. Damit eröffnete sich die Chance für DApps und Smart-Contracts auf der Blockchain. Im Zusammenhang mit Ethereum bedeutet ein Smart Contract ein Computerprotokoll, welches eine Kontrolle des digitalen Vermögenswertes darstellt. Sobald ein Signal für eine Überweisung an das Protokoll gesendet wird, wird diese auch vollzogen. Sie kommen zur Anwendung, wenn es um Mietzahlungen, Gehaltszahlungen oder Versicherungsbeiträge geht [8, S. 45].

Das Jahr 2014 brachte einen erfolgreichen ICO. Ein ICO (Initial Coin Offering) entspricht bei einer Kryptowährung einem Börsengang eines Wertpapiers. Es dient der erstmaligen Aufnahme von Kapital. Als Gegenleistung wird eine bestimmte Anzahl an Token an die Investoren herausgegeben. Bereits im August desselben Jahres brachte dies dem Unternehmen 16.000.000 USD [9]. Damit konnte die Entwicklung sehr stark beschleunigt werden, sodass am 30. Juli 2015 der erste Block der Ethereum-Blockchain gemined wurde. Ethereum wurde als Supercomputer bekannt.

Die nächsten fünf Jahre bedingten eine weiterhin erfolgreiche Entwicklung von Ethereum als ein internationales Netzwerk, das aus mehreren Tausend Computer bestand, die alle miteinander verbunden sind. Viele Entwickler beteiligten sich daran. Die Möglichkeiten, welche Bitcoin für Zahlungen anbot, übernahm Ethereum im Bereich des Programmierens. Ähnlich aufgebaut ist auch das Netzwerk, denn das Ethereum-Netzwerk baut sich aus Minern und Nodes auf der ganzen Welt auf. Das bedeutet, dass Nutzer des Netzwerks Rechenleistung zur Verfügung stellen und gleichzeitig einen sogenannten Node repräsentieren. Jeder Node, welcher als erster die geforderte Rechenleistung erzeugt, produziert einen neuen Block in der Blockchain. Dies ist dann ein Nachweis dafür, dass die erforderliche Arbeit geleistet wurde (Proof of Work). Sie bilden die Grundlage für die Rechenleistung der Smart Contracts und DApps, welche auf Ethereum beruhen.

Die Miner bekommen pro geminten Block einen Anreiz in Ether ausgezahlt. Ether stellt dabei die Kryptowährung von Ethereum dar, analog zu Bitcoin. Momentan, Stand 2020, dauert es 12 Sekunden um ein Ether zu minen [10]. Im Jahr 2022 wird eine Dauer von fünf Sekunden angenommen [11]. Ether wird benötigt, um den Computer betreiben zu können. Es ist die Grundlage einer Transaktion, die zur Ausführung mehrere Ether benötigt. Dies bezieht sich auch auf die Ausführung eines Smart Contracts. Damit hat es Ether heute auf den zweiten Platz der Kryptowährungen geschafft, wenn die Marktkapitalisierung betrachtet wird. Der Blockchain-Explorer Etherscan besagt, dass es inzwischen 220.000 verschiedene Token auf Ethereum-Basis gibt. Ausserdem existieren 3.150 DApps, deren Anzahl stetig wächst [9].

2.2 Die Ethereum Philosophie

Das Ethereum-Protokoll sollte so einfach wie möglich sein, selbst auf Kosten einer gewissen Datenspeicherung oder Zeitineffizienz. Ein durchschnittlicher Programmierer sollte idealerweise in der Lage sein, die gesamte Spezifikation zu befolgen und umzusetzen, um das beispiellose Demokratisierungspotential der Kryptowährung voll auszuschöpfen und die Vision von Ethereum als ein Protokoll zu fördern, das allen offensteht. Eine Optimierung, die die Komplexität erhöht, sollte nicht berücksichtigt werden, es sei denn, diese Optimierung bietet einen erheblichen Nutzen.

Universalität: Ein wesentlicher Bestandteil der Designphilosophie von Ethereum ist, dass Ethereum keine "Merkmale" hat. Stattdessen bietet Ethereum eine interne Turing-vollständige Skriptsprache, mit der ein Programmierer jeden intelligenten Vertrag oder Transaktionstyp erstellen kann, der mathematisch definiert werden kann [12].

Die Teile des Ethereum-Protokolls sind so modular und trennbar wie möglich gestaltet. Im Laufe der Entwicklung ist es das Ziel, ein Programm zu erstellen, bei dem der Anwendungsstapel ohne weitere Änderungen weiter funktioniert, wenn eine kleine Protokolländerung an einem Ort vorgenommen wird.

Daten-Strukturen wie Ethash, modifizierte Patricia-Merkel-Trie, und RLP (Recursive-Length Prefix Serialization) werden als separate Bibliotheken mit vollständigen Funktionen implementiert. Dies ist so, dass, obwohl sie in Ethereum verwendet werden, auch wenn Ethereum bestimmte Funktionen nicht benötigt, solche Funktionen auch in anderen Protokollen verwendet werden können. Die Entwicklung von Ethereum erfolgt maximal, um das gesamte Kryptowährung-Ökosystem und nicht nur sich selbst zu fördern.

Details des Ethereum-Protokolls sind nicht endgültig festgelegt. Durch die Verwendung der Protokollarchitektur oder der Ethereum Virtual Machine (EVM) wird die Skalierbarkeit und Sicherheit erheblich verbessert.

Das Protokoll versucht nicht, bestimmte Nutzungskategorien aktiv einzuschränken oder zu verhindern. Alle Regulierungsmechanismen im Protokoll sind so konzipiert, dass sie den Schaden direkt regulieren und nicht versuchen, bestimmten unerwünschten Anwendungen entgegenzutreten. Ein Programmierer kann sogar ein Endlosloop über Ethereum ausführen, solange er bereit ist, die Transaktionsgebühr pro Rechenschritt weiter zu zahlen [7].

2.3 Dezentralisierte Autonome Organisationen

Eine allgemeine Übersicht zum Codieren eines DAO lautet wie folgt. Das einfachste Design ist einfach ein selbstmodifizierender Code, der sich ändert, wenn zwei Drittel der Mitglieder einer Änderung zustimmen. Obwohl der Code theoretisch unveränderlich ist, kann man dies leicht umgehen und de facto veränderlich sein, indem Teile des Codes in separaten Verträgen enthalten sind und die Adresse der aufzurufenden Verträge im veränderbaren Speicher gespeichert ist. Bei einer einfachen Implementierung eines solchen DAO-Vertrags würde es drei Transaktionstypen geben, die sich durch die in der Transaktion bereitgestellten Daten unterscheiden [7].

Der Vertrag hätte dann Klauseln für jede dieser Klauseln. Es würde eine Aufzeichnung aller Änderungen des offenen Speichers zusammen mit einer Liste derjenigen führen, die für sie gestimmt haben. Es hätte auch eine Liste aller Mitglieder. Wenn eine Speicheränderung zwei Drittel der Mitglieder erreicht, die dafür stimmen, kann eine abschliessende Transaktion die Änderung ausführen. Ein ausgefeiltes Skelett verfügt auch über eine integrierte Abstimmungsfunktion für Funktionen wie das Senden einer Transaktion, das Hinzufügen von Mitgliedern und das Entfernen von Mitgliedern und kann sogar eine Abstimmungsdelegation im Stil einer Liquid Democracy vorsehen (d.h. jeder kann jemanden zuweisen, der für sie abstimmt, und eine Zuweisung vornehmen ist transitiv. Wenn also A B und B C zuweist, bestimmt C die Stimme von A. Dieses Design würde es dem DAO ermöglichen, als dezentrale Gemeinschaft organisch zu wachsen, und es den Menschen ermöglichen, die Aufgabe, herauszufinden, wer Mitglied ist, schliesslich an Spezialisten zu delegieren, obwohl Spezialisten im Gegensatz zum "aktuellen System" im Laufe der Zeit leicht ein- und aussteigen können, wenn einzelne Community-Mitglieder ihre Ausrichtung ändern [7].

2.4 Smart-Contracts

Smart Contracts sind digitale Vereinbarungen zwischen Parteien, die automatisch in Kraft treten, wenn bestimmte Bedingungen erfüllt sind, z. B. wenn ein bestimmter Schwellenwert überschritten wird. Bereits vor der Einführung von Bitcoin im Jahr 1997 gab es das Konzept der Smart Contracts. Die Idee wurde erstmals von Nick Szabo vorgeschlagen. Smart Contracts im Ethereum-Netzwerk müssen deterministisch und Turing-komplett sein. Das gleiche Eingabeergebnis muss von allen Node verwendet werden, um die gleiche Aktion auszuführen. Es wäre für zahlreiche Node schwierig, einen neuen Block ohne Konsens zu bestätigen. Da es sich bei der Blockchain um ein geschlossenes System handelt, ist dies für Verträge, die die Daten der Blockchain nutzen, kein Problem.

2.4.1 Vorteile von Smart Contracts

- Es besteht die Hoffnung, dass die Verwendung von Smart Contracts die Geschäftsabläufe von Unternehmen jeder Grösse effizienter macht. Dadurch können Betriebskosten gesenkt und Ressourcen eingespart werden, z. B. die Anzahl der Mitarbeiter, die zur Überwachung des Status eines komplexen Prozesses benötigt werden, der als Reaktion auf unternehmensübergreifende Umstände ausgeführt wird.
- Smart Contracts können die Abwicklung von unternehmensweiten Geschäftsprozessen beschleunigen.
- Smart Contracts machen die Überwachung von Geschäftsvorgängen durch eine dritte Partei überflüssig, da sie automatisch vom Netzwerk durchgeführt werden.
- Darüber hinaus ist es möglich, dass Smart Contracts Blockchains und andere Distributed-Ledger-Technologien nutzen, um alle Aktivitäten im Zusammenhang mit den durchgeführten komplexen Verfahren aufzuzeichnen. Durch automatisierte Transaktionen wird das Risiko menschlicher Fehler minimiert und eine präzise Vertragsausführung gewährleistet.

2.4.2 Herausforderungen mit Smart Contracts

- Smart Contracts bieten ein gewisses Mass an Sicherheit für Geschäftstransaktionen, an denen viele Parteien beteiligt sind. Die Technologie steckt jedoch noch in den Kinderschuhen, und Hacker finden immer neue Wege, um die Ziele der Unternehmen, die die Regeln festlegen, zu umgehen. In den Anfängen von Ethereum nutzten Hacker Smart Contracts, um 50 Millionen Dollar in Kryptowährung zu stehlen. Viele verschiedene Sicherheitsprobleme bei Smart Contracts können mit unterschiedlichen Erkennungstechnologien erkannt werden.

- Hacker müssen daran gehindert werden, Ereignisse nachzuahmen, die dazu führen, dass Smart Contracts von einem Orakel, einer der Streaming-Datenquellen, die Ereignisaktualisierungen bereitstellen, fehlerhaft ausgeführt werden. Komplexe Einstellungen können es dem System erschweren, die richtigen Ereignisse zu erzeugen.
- Smart Contracts können die Ausführung von Prozessen, an denen mehrere Parteien beteiligt sind, beschleunigen, unabhängig davon, ob sie mit der Absicht und dem Verständnis aller Parteien übereinstimmen. Smart Contracts können jedoch die Schwere des Schadens erhöhen, wenn die Ereignisse ausser Kontrolle geraten und es keinen Mechanismus gibt, um schädliche Aktionen zu stoppen oder rückgängig zu machen. Laut dem Marktforschungsunternehmen Gartner sind die Skalierbarkeit und Verwaltbarkeit von Smart Contracts noch ungelöste Herausforderungen.
- Die Implementierung und Verwaltung von Smart Contracts ist kompliziert. Änderungen an ihnen sind häufig schwierig oder unmöglich, weil sie bereits so eingerichtet sind. Dies hat zwar einen Sicherheitsvorteil, doch können die Parteien keine Änderungen vornehmen oder neue Teile hinzufügen, ohne eine neue Vereinbarung zu treffen. Dies ist eine wichtige Einschränkung von Smart Contracts.

2.4.3 Besonderheit von Smart Contracts

- Kryptografie wird eingesetzt, um Verträge zu sichern und die Manipulation von Aufzeichnungen zu verhindern. Es hat eine Reihe von Fällen gegeben, in denen Smart Contracts gehackt wurden und eingezahlte Gelder entwendet wurden, obwohl das System oft recht sicher ist. Wenn ein Smart Contract in einem öffentlichen Netzwerk läuft, kann jeder sehen, was er ist und wofür er verwendet wird.
- Obwohl sie von der Bandbreite und der Überlastung der zugrunde liegenden Netze abhängen, werden Smart Contracts oft fast sofort für alle Parteien auf allen teilnehmenden Computern ausgeführt, vorausgesetzt, die erforderlichen Voraussetzungen sind erfüllt.
- Smart Contracts funktionieren automatisch; es ist also nicht nötig, eine Taste zu drücken. Darüber hinaus können sie, wenn sie einmal in Kraft getreten sind, nicht mehr von einer zentralen Behörde wie Banken oder Maklern oder sogar vom Netzwerk geändert oder verwaltet werden.
- Smart Contracts stützen sich nicht auf die Unsicherheit der Sprache und die Bedeutung von Wörtern, da sie in Code geschrieben sind. Dies kann unpraktisch sein, insbesondere wenn es Grauzonen gibt, die flexible Antworten erfordern.

2.5 Proof of Work / Proof of Stake

Eine Transformation von PoW zu PoS bringt nicht nur eine grössere Skalierbarkeit mit sich, sondern zudem eine Reduktion der Energieressourcen und somit eine Kosteneffizienz [14]. Das Proof-of-Work-Prinzip schafft eine Übersicht über den Besitz, indem eine dezentrale Validierung der Transaktionen vorgenommen wird. Auf diese Weise entsteht ein Protokoll durch den Konsensmechanismus des Proof-of-Work. Dadurch werden sogenannte «double spendings» verhindert und die Nutzersicherheit erhöht [6]. Dazu unterstützen die Nutzer des Netzwerkes die Rechenleistung, indem sie eine problemorientierte Aufgabe aktivieren. Der Nutzer, der das Problem löst, befreit einen «Block» und ermöglicht damit seine Integration in die Blockchain. Auf diese Weise regelt sich zudem die Herausforderung der Entscheidungsfindung zwischen den verschiedenen Blockchain, denn die Anzahl der Nutzer einer Blockchain entscheidet über die Länge derselben, weil sie durch die Leistung der Nutzer unterstützt wird. Dann kommt es zu einer self-fulfilling-prophecy, denn die längste Chain wird als «Beste» angesehen. In einer Situation, in der mehrere Blocks bearbeitet werden, erfolgt die Integration der Kette als erstes, bei der der Proof-Of-Work positiv ausfällt. Daraus entwickelt sich der Schwierigkeitsgrad der Rechenleistung [15, S. 2].

Die Dauer einer Transaktion beim Bitcoin beträgt 10 Minuten. Derzeit liegt der Verdienst beim Bitcoin-Mining zwischen 0,4 und 2 Bitcoin. Dieser entsteht durch den Anteil an Transaktionsgebühren. Zusätzlich erhalten Miner seit Mai 2020 eine Belohnung von 6,25 Bitcoin für das Erstellen eines Blocks. Bei jedem Halving, welches alle vier Jahre stattfindet, wird dieser Wert halbiert. Im Jahr 2020 hat das letzte Halving stattgefunden [16].

Als Motivation für die Nutzer gilt in diesem System insbesondere die erste Transaktion. Durch die Lösung dieser Rechenleistung wird eine neue Münze generiert, die dem Produzenten des Blockes zugeordnet wird. Dadurch erspart sich das Netzwerk die Notwendigkeit einer kontrollierenden Instanz. Die Datenbank unterliegt stets einer chronologischen Erweiterung, sobald eine Transaktion durchgeführt wird. Daher kommt auch der Vergleich mit einer Kette im engl. Chain, denn die Transaktionen werden nacheinander zu einer Kette verbunden. Sobald ein Block vervollständigt ist, entsteht der nächste. Dadurch bekommt jeder Block eine Prüfsumme (Hashwert) des vorherigen Blocks. Dieser gesamte Vorgang ist unter dem Begriff «Mining» bekannt [15].

Trotz aller Veröffentlichungen kann die Anonymität der Nutzer gewahrt werden. Pro Transaktion ist lediglich der Betrag, die Empfangs- und die Senderadresse bekannt, als auch die Transaktions-ID. Deshalb sind die öffentlichen Schlüssel nicht zuzuordnen.

Dagegen steht das Proof-of-Stake Konzept, welches von jedem Nutzer einen Nachweis der Anteile erfordert. Dieser Nachweis wird in regelmässigen Abständen immer wieder verlangt.

Damit besteht für jede Transaktion ein Nachweis, dass die angewiesene Transaktion auch ausgeführt wurde. Innerhalb des Blockchain-Netzwerks entsteht so eine Einigung darüber, welcher Nutzer den nächsten Block mittels gewichteter Zufallsauswahl erzeugen darf [17, S. 34].
Die Funktionsweise über eine PoW-Blockchain garantiert eine Sicherheit vor Angriffen. Bei einem Angriff werden 25 bis 50 Prozent der gesamten Rechenleistung für das Mining genutzt. Beim Proof-of-Stake-System kann es zu einem sogenannten 51%-Angriff kommen. Wenn 51 Prozent der Token im Besitz eines Nutzers sind, kann dieser seine Position ausnutzen, um Manipulationen der Blockchain zu erzeugen. Hinzu kommt das Risiko des sogenannten „stake grinding“, wobei Schritte erfolgen, welche den Validator betreffen und ihn so verändern, dass er einen Vorteil bringt. Hybride Smart Contracts fördern die ökologische Nachhaltigkeit, indem sie durch fortschrittliche Verifizierungstechniken rund um die wahren Auswirkungen grüner Initiativen bessere Anreize schaffen, an grünen Praktiken teilzunehmen.

2.6 Transformation zu Ethereum 2.0

Im Fokus der Aufmerksamkeit in diesem Kontext steht derzeit vor allem die DeFi Bewegung (Decentralized Finance). Diese Anwendung ist ebenfalls eine App der Blockchain-Umgebung, welche die Finanzwelt grundlegend verändern könnte [13].
Trotzdem gibt es parallel dazu noch weitere Entwicklungen. Am 01. Dezember 2020 ist das lang ersehnte Projekt Ethereum 2.0 auf den Markt veröffentlicht worden. Dies sollte ein entscheidender Meilenstein für das Netzwerk Ethereum sein. Einer der entscheidenden Gründe dafür war der Hype um CryptoKitties. Das Online-Spiel CryptoKitties erfreut sich einer so grossen Beliebtheit, dass andere Transaktionen mit der Währung blockiert wurden. Dies hängt damit zusammen, dass alle Käufe im Blockchain-Netzwerk Ethereum ablaufen. Aktuell sind auf Ethereum 15 Transaktionen pro Sekunde möglich. Dies ist noch nicht ausreichend, um einer hohen Zahl an Nutzern im Millionenbereich eine optimale Nutzung zu gewährleisten. Um die Apps jedoch umsetzbar zu machen, muss die Blockgrösse erhöht werden, damit Transaktionen vermehrt werden können. So kommt es beim Upgrade Ethereum 2.0 zu einer Transformation der Konsensebene von Proof-of-Work zu Proof-of-Stake.
Die Phase 2 der Integration von Ethereum 2.0 beinhaltet die gesamte Funktionalität des Systems. Es entsteht aus den Shard-Chains eine strukturierte Kette, die mittels Datencontainer erzeugt wird. Gleichzeitig werden Smart Contracts eingebunden. Damit verwaltet jeder Shard ein virtuelles Gerät, das durch eWASM funktioniert. Diese Phase dient der Unterstützung von Verträgen, Zuständen und Konten. Daneben kommt das Konzept der

Ausführungsumgebungen (Execution Environments, EEs) zum Einsatz. Dies bedeutet, dass jedem Shard ein Zugang zu allen Ausführungsumgebungen gewährt wird, worin Transaktionen getätigt werden können. Neben der Ausübung von Smart-Contracts vollzieht sich eine Interaktion. Weitere Entwicklungen sind in Zukunft noch zu erwarten. Insgesamt sind sechs Phasen vorgesehen [18, S. 522]. Ethereum 2.0 bedeutet für seine Entwickler nicht nur eine Chance, sondern auch ein Risiko. Dieses wird eingegangen, um der Community einen Vorteil zu verschaffen. Sobald ETH und ETH2 nebeneinander vorhanden sind, kommt es zu einer Ausweitung von Kapital. Es sind nun zwei Chains vorhanden, auf denen Ether entstehen. Die Verbundenheit mit der Beacon Chain macht den Tausch mit ihr irreversible. Dadurch wirken sich die Entwicklungen auch auf den Kurs von Ethereum aus [19].

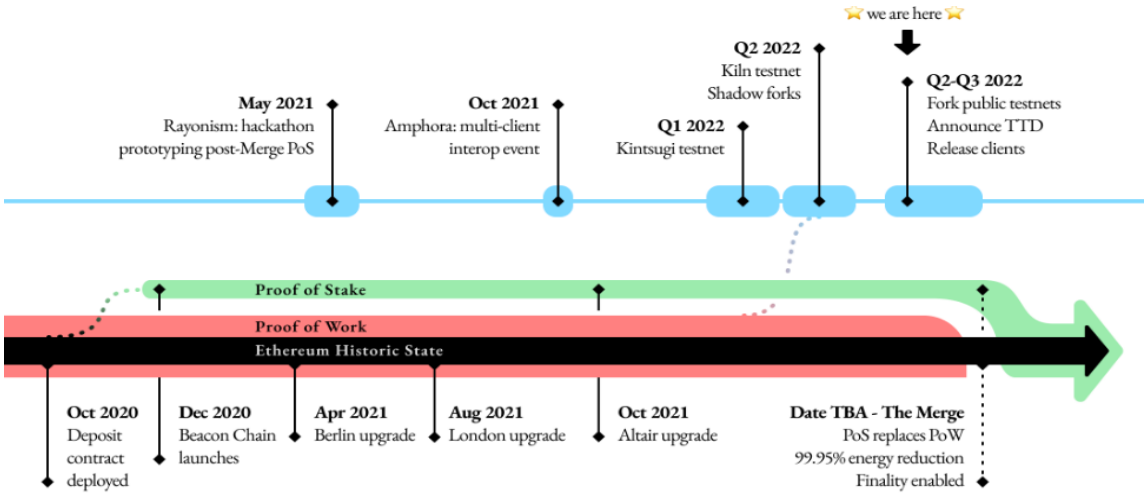


Abbildung 2 - Ethereum 2.0 Entwicklung (Quelle: blog.ethereum.org/2022/01/24/the-great-eth2-renaming)

2.7 Oracles

Oracles sind ein entscheidendes Werkzeug, um Smart Contracts mit Umweltdaten aus Sensormesswerten, Satellitenbildern und fortschrittlichen ML-Berechnungen zu versorgen, die es Smart Contracts dann ermöglichen, Belohnungen an Menschen auszugeben, die Wiederaufforstung betreiben oder sich für bewussten Konsum engagieren. Orakel unterstützen auch viele neue Formen von Kohlenstoffzertifikaten, um die Auswirkungen des Klimawandels auszugleichen [20].

Mit der Oracle-Netzwerk von Chainlink kann das Oracle-Problem gelöst werden, denn es verfügt über ein dezentralisiertes Oracle-Netzwerk. Ein Orakel erhält für seine Leistung Token von Chainlink. Aus technischer Sicht liegt das Oracle zur Hälfte in der Blockchain und zur anderen Hälfte ausserhalb der Blockchain. Im Inneren der Blockchain wird die Vereinbarung zwischen dem Orakel und dem Smart Contract geschaffen. Der äussere Teil des Oracles wird auch Knoten genannt. Diese Knoten stellen Smart Contracts Off-Chain-Daten bereit. Dabei generieren sie Link-Token. Durch die hohe Anzahl an Knoten

wird ein vollkommen dezentraler Prozess geschaffen. Die Sicherheit wird dadurch gewährleistet. Verstärkt wird diese Sicherheit durch die Überprüfung der von Oracles unzuverlässig produzierten Daten an den Knoten [3].
Unter Oracles ist eine innovative Technologie zu verstehen, welche Daten aus einer Quelle integrieren kann und gleichzeitig an einen oder mehrere Smart Contracts weiterzuleiten. Somit werden in den Smart Contracts die Bedingungen für den Vertrag automatisch durchgeführt. Dabei ist es nicht entscheidend, welche Art von Daten gespeichert und weitergeleitet werden. Es ist lediglich entscheidend, dass die Funktion der Smart Contracts erfüllt wird. Innerhalb der Welt der Blockchain bieten diese Oracles eine zusätzliche Innovation, denn sie schaffen die Möglichkeit für die Blockchain mit der realen Welt zu kommunizieren. Dabei handelt es sich um eine wechselseitige Kommunikation. Oracles übernehmen Daten aus einer Blockchain wie Ethereum und transferieren sie an externe Anwendungen. Damit erlaubt es einen sicheren Zugriff für Anwender der Blockchain [21].
Dabei nehmen die Oracles die Funktion eines Vermittlers ein, welcher die Ereignisse aus der realen Welt verifiziert und den Smart Contracts zur Verfügung stellt. Damit wird ein idealer Anknüpfungspunkt für die Smart Contracts zur Aussenwelt geschaffen, denn die Funktionsweise dieser beruht in der Regel auf Folgendem: «Wenn Bedingung A eingetreten ist, wird Operation B ausgeführt". Eine Überprüfung der Bedingungen kann jedoch nur von aussen erfolgen. Soll beispielsweise ein Dokument beglaubigt werden, muss zunächst die Echtheit des Dokumentes verifiziert werden [22, S. 1].
Durch die ausschliessliche Handlung im virtuellen Raum entstehen vermehrt Risiken durch Cyberkriminalität. Sämtliche virtuellen Vermögenswerte sind deshalb von der Gefahr des Online-Betruges unterlegen. Die Smart Contracts waren in der Vergangenheit bereits beliebtes Ziel von Hackern.
In Form von NFTs wurden vom DeFi protocol Poly Network $600 Millionen erbeutet. [23, S. 10].
Hervorzuheben ist der Zusammenhang zwischen Oracles und den Stablecoins. Innerhalb der DeFi-Anwendungen bildet der Stablecoin einen hohen Wertbetrag. Um Stablecoins zu ermöglichen, ist es notwendig, dass Orakel dezentralisiert sind und über mehrere Schichten kryptografisch sicher. Zudem können Oracles können Preisbindungen mit höchster Sicherheit einhalten, da Aktualisierungsraten in Sekundenschnelle erfolgen. Eine weitere Unterstützung für den Stablecoin basiert auf der Überprüfung der Reserven einer Stablecoin, welche als Proof of Reserves (PoR)-Audit bezeichnet wird. Dabei ruft Oracle Daten ab, welche Stablecoin-Emittenten brauchen, um eine möglichst hohe Präzision hinsichtlich der tatsächlichen Besicherung der von ihnen in Umlauf gebrachten Stablecoins aufrechtzuerhalten. Diese PoR-Referenz-Feeds agieren autonom von dezentralen Netzwerken.

Insbesondere die Transparenz der Überprüfung der Sicherheiten in Echtzeit schützen vor Cyberkriminalität [24].

2.7.1 Oracle Typen

Für die Verbindung zwischen der Blockchain und der Aussenwelt stehen unterschiedliche Oracle-Typen zur Verfügung [25]:

2.7.1.1 Software Oracle

Weitergabe von Informationsdaten, die aus Online-Quellen stammen, wie Temperatur, Preise von Waren und Gütern, Flug- oder Zugverspätungen usw.

2.7.1.2 Hardware Oracle

Daten aus der physischen Welt, wie z. B. ein Auto, das ein Hindernis überwindet, indem seine Bewegungssensoren das Hindernis erkennen werden an einen Smart Contract gesendet.

2.7.1.3 Eingehende Oracle

Bereitstellung von Daten aus der Aussenwelt.

2.7.1.4 Ausgehende Oracle

Bereitstellung von Smart Contracts, welche Daten an die Aussenwelt senden können. Dazu gehört beispielsweise ein intelligentes Schloss aus der physischen Welt, welches Zahlungen an seine Blockchain-Adresse erhält und automatisch entsperrt werden muss.

2.7.1.5 Konsensbasierte Oracle

Daten aus menschlichen Konsens- und Vorhersagemärkten wie Augur und Gnosis, um das Risiko für Marktmanipulationen zu vermeiden. Dazu wird ein Bewertungssystem für Orakel implementiert.

Zur weiteren Sicherheit kann eine Kombination verschiedener Orakel verwendet werden, wobei beispielsweise drei von fünf Orakeln den Ausgang eines Ereignisses bestimmen könnten [25].

3 Entwicklung einer Web3-Applikation

Es existieren zwei Optionen, um Daten an die Smart Contracts zu geben. Die erste Möglichkeit besteht über Referenzdaten. On-Chain Referenzpunkte greifen auf externe Daten zu. Diese Option kann sehr schnell durchgeführt werden, ist jedoch in ihrem Angebot nur begrenz verfügbar. Die zweite Option wird als Request & Receive und ermöglicht das Empfangen von Daten durch den Smart Contracts von einem Off-Chain-API ausserhalb der Blockchain. Im Gegensatz zur ersten Option gestaltet sich diese Option deutlich umfangreicher [26].

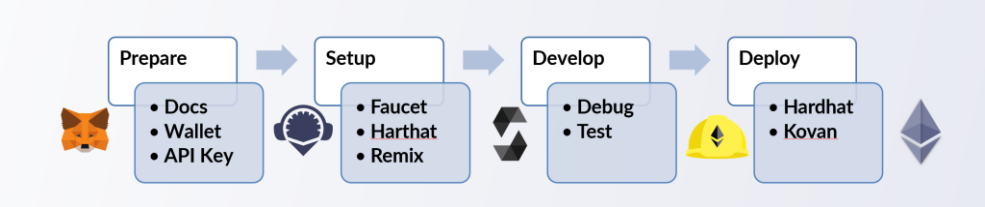


Abbildung 3 - Technischer Prozess zur Entwicklung eines DApps

3.1 Prepare

Vor dem Aufsetzen und Programmieren müssen wir uns in die Dokumentation einlesen.
Wir verwenden Hardhat [27] als Boilerplate Projekt.
Hardhat ist eine gesamte Entwicklungsumgebung, welche uns auch mit lokalen Wallets mit genügend Guthaben ausrüstet, um diverse Anwendungsfälle zu testen.
Das Hardhat-CLI startet eine lokale Entwicklungs-Chain mit den Wallets.

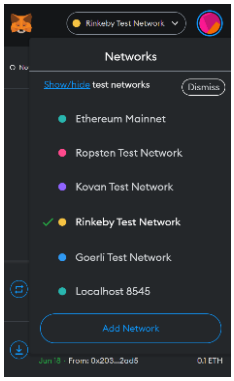


Abbildung 4 - Hardhat Network unter Localhost 8545

Um später das DApp in das öffentliche Test-Netzwerk «Rinkeby» hochzuladen, brauchen wir dort ein Wallet sowie ein API-Key, dasselbe gilt für das Produktive Ethereum-Mainnet Netzwerk.
Die API-Keys können über Etherscan gelöst werden [28].
Die Wallets erstellen wir mit MetaMask [29]. Es braucht jeweils für jedes Netzwerk ein eigens Wallet. Guthaben kann nicht von einem Test-Netzwerk in ein Produktive Netzwerk transferiert werden.

3.2 Setup

Die Idee ist eine Applikation mit Frontend, bei welcher sich ein Benutzer mit seinem Wallet (Account) via MetaMask anmelden kann. Er soll einen Smart-Contract ausführen können, welcher ein Oracle aufruft. Das Oracle soll zu Demo-Zwecken eine Zufallszahl berechnen und diese dem Benutzer zurückgeben.
Die Benutzerschnittstelle wir mit einer React-Applikation [30] realisiert.
Der Smart-Contract wird in Solidity [31] programmiert.
Wir erstellen ein neues leeres Git-Repository für den gesamten Code:

```
git init C:\_dev\ffhs\sema
```

Code Snippet 1 – Initialisiere ein neues Git-Repository

3.2.1 Contracts (Hardhat)

In dem Ordner führen wir den ersten hardhat Befehl via npx aus.

```
npx hardhat
```

Code Snippet 2 - Run npx hardhat

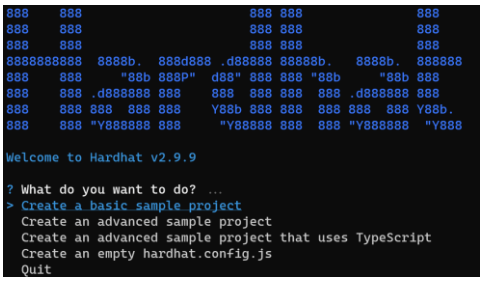


Abbildung 5 - Interaktives Projektsetup dank Hardhart

Wir wählen ”Create an advanced sample project that uses TypeScript”. Zusätzlich geben wir das Projektverzeichnis an und bestätigen das Erstellen einer .gitIgnore Datei.
Hardhat installiert diverse Abhängigkeiten für die Entwicklung von Smart-Contracts.
Auflisten der Wallets im Hardhat-Netzwerk:

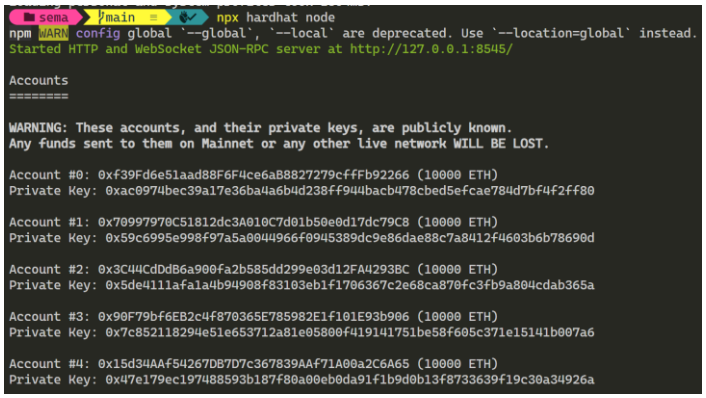


Abbildung 6 - Hardhat Wallets (Private und Public Keys) mit jeweils 10'000 Ethereum Guthaben

Wir sehen, dass Hardhat das Netzwerk gestartet hat und die Wallets abfragte.

3.2.2 Frontend

Die React-Applikation erstellen wir im gleichen Repository, jedoch ein einem separaten Ordner. Dazu führen wir im Hauptordner des Repository folgender Befehl aus.

```
npx create-react-app app-frontend --template typescript
```

Code Snippet 3 - Run npx create-react-app

Wir sehen einen neuen Unterordner «app-frontend». In diesem befindet sich die erstellte React-Applikation, in welcher im nächsten Schritt das Login via MetaMask implementieren, sowie die Funktionalität einen Betrag von einem Wallet zu einem anderen Wallet zu senden.

3.2.3 Remix

An diesem Punkt führen wir noch eine Plattform ein, um die Smart-Contracts einfach zu veröffentlichen. Remix ist ein Editor mit Superkräften, wir können, wie bereits Smart-Contracts in diverse Netzwerke veröffentlichen und diese testen [32].
Remix ist eine Browser-App welche unser lokales Repository öffnen und die Dateien darin direkt bearbeiten kann.
Wir installieren die Remix CLI via npm und öffnen unser lokales Repository im Browser mit Remix.

```
npm install --global @remix-project/remixd

remixd ./ --remix-ide https://remix.ethereum.org
```

Code Snippet 4 - Installation von Remix via npm und Starten des lokalen Remix-Servers

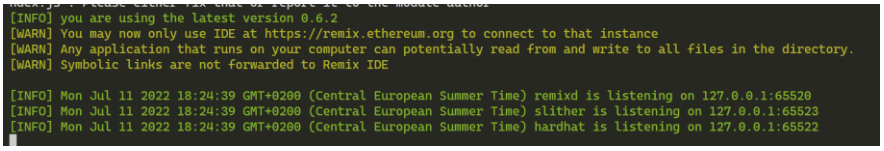


Abbildung 7 - Ausgabe von Remix nach dem Starten des Servers

3.3 Develop

Wir haben nun eine lokale Entwicklungsumgebung sowie mehrere Wallets im Hardhat-Netzwerk, um erste Gehversuche mit den Smart-Contracts zu machen.

3.3.1 Smart-Contract

Wir entwickeln zuerst einen einfachen Smart-Contract welcher «nur» eine Variabel besitzt, welche von aussen über eine Funktion aktualisiert werden kann.
Wir nennen den Smart-Contract «HelloWorld».

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract HelloWorld {
  string public message;

  constructor(string memory initialMessage) {
    message = initialMessage;
  }

  function updateMessage(string memory newMessage) public {
    message = newMessage;
  }
}
```

Code Snippet 5 - Code des HelloWorld.sol Smart-Contracts in der Programmiersprache Solidity.

Wir sehen im Code die Ähnlichkeit zu JavaScript. Der Contract «HelloWorld» ist die Definition eines Objektes, welches eine Variabel «message», einen Konstruktor sowie eine Methode zum Aktualisieren der Variabel besitzt.

Mit Remix können wir den Contract in Bytecode respektive das ABI (Application Binary Interface) kompilieren.

Wir Verbinden Remix zu unserem lokalen Hardhat-Network und kompilieren den Smart-Contract.

```
npx hardhat node
```

Code Snippet 6 - Starte lokales Hardhat Network

Die Ausgabe ist dieselbe wie in Abbildung 6. Nun navigieren wir im Remix-Browser zu unserem Contract und kompilieren diesen.

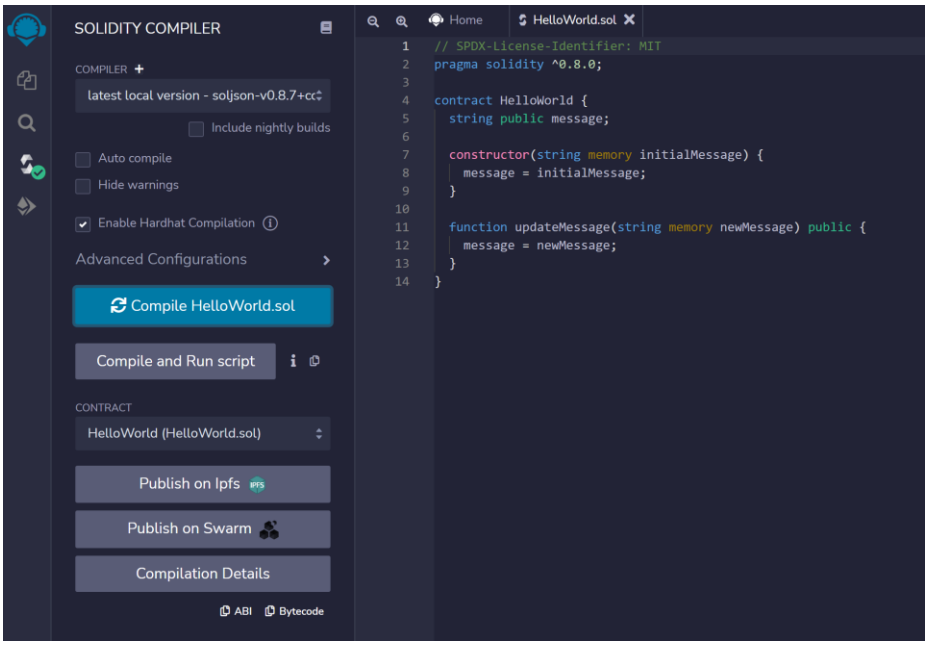


Abbildung 8 - Die Remix Oberfläche verfügt über einen Button (Blau) zum Kompilieren des aktuel geöffneten Smart-Contracts

3.3.2 MetaMask Login

In der Benutzeroberfläche unserer Applikation realisieren wir nun das Anmelden eines Benutzers mit seinem Wallet in MetaMask.
MetaMask stellt uns dafür eine Dokumentation für ihre JavaScript API (Onboarding Library) zur Verfügung [33].
Als erstes Installieren wir die Abhängigkeiten für das Login mit npm:

```
"dependencies": {
      "@metamask/onboarding": "^1.0.1",
      "web3": "^1.7.3",
      "ethers": "^5.6.8",
      ...
}
```

Code Snippet 7 - Erweitern der Dependencies in der package.json Datei

Wir prüfen als nächstes gleich beim Starten der App ob die Ethereum-Library von MetaMask vorhanden ist, wenn nicht zeigen wir eine Fehlermeldung:

```
// useEffect check if window.ethereum is available, if not, show error page
useEffect(() => {
  if (window.ethereum) {
    setWeb3(new Web3(window.ethereum));
  } else {
    setMetaMaskNotInstalled(true);
  }
}, []);
```

Code Snippet 8 - Prüfen, ob MetaMask vorhanden ist.

Der Gesamte Code kann später dem internen Repository entnommen werden. An dieser Stelle soll noch gezeigt werden, wie eine Transaktion ausgelöst werden kann. Die Eigenschaften der Transaktion können aus der Dokumentation von MetaMask entnommen werden.

```
const transactionParameters = {
      nonce: "0x00", // ignored by MetaMask
      gasPrice: weiAmountGasPrice, // customizable by user during MetaMask confirmation.
      gas: web3.utils.toHex(76248), // customizable by user during MetaMask confirmation.
      to: "0x4760D1e8798c35AEFD80BE42Bd3C4B94E30E6EE8", // Required except during contract publications.
      from: accounts[0].id, // must match user's active address.
      value: weiAmount, // Only required to send ether to the recipient from the initiating external account.
      data: "0xe0c86289", // Optional, but used for defining smart contract creation and interaction.
      chainId: "0x4", // Used to prevent transaction reuse across blockchains. Auto-filled by MetaMask.
};

// txHash is a hex string
// As with any RPC call, it may throw an error
const txHash = await window.ethereum.request({
      method: "eth_sendTransaction",
      params: [transactionParameters],
});
```

Code Snippet 9 - Das Transaktionsobjekt verfügt über diverse Eigenschaften. Das Objekt wird der window.ethereum.request Funktion übergeben. Dies löst MetaMask's Plugin aus.

Wir können unseren Frontend Code lokal testen und die Transaktionen ebenfalls lokal mit dem Hardhat-Netzwerk ausführen.

Starten wir die React-Applikation, öffnet sich die Webseite und wir sehen, dass die Verbindung zu MetaMask bereits erfolgreich ist. Dies wird durch das Anzeigen der Adresse ganz unten im Bild sowie des vorhandenen Guthabens angezeigt.
Als nächstes Klicken wir auf den Knopf, welcher die Transaktion auslöst.
Diese Transaktion (rechts im Bild) öffnet das MetaMask Plugin, welches Informationen zur Transaktion enthält.

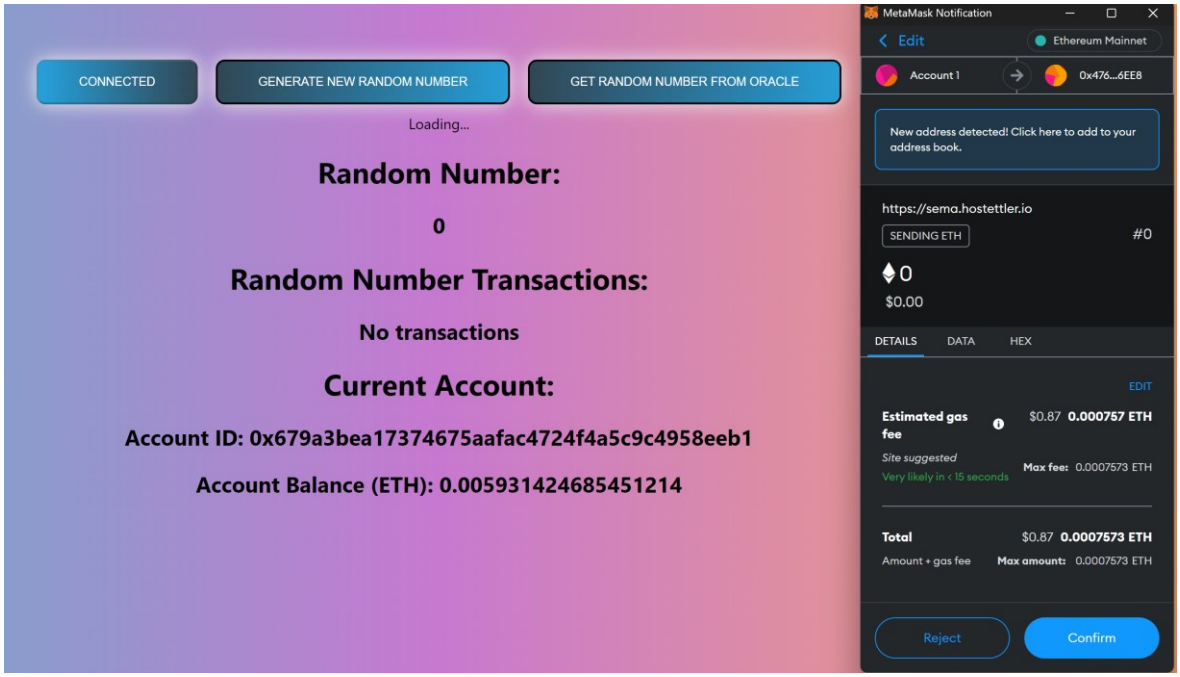


Abbildung 9 - Auslösen einer Transaktion durch das Betätigen eines Knopfes auf einer Webseite. Unser erstes DApp.

Noch ist die gesamte Applikation lokal und nicht veröffentlicht. Auch die Transaktion ist nur im lokalen Hardhat-Netzwerk verzeichnet und wurde nicht auf der öffentlichen Ethereum Blockchain registriert.

3.3.3 Oracle

Zum jetzigen Zeitpunkt haben wir noch keine externen Daten einbezogen. Trotz der bereits umfangreichen Entwicklungs- und Testumgebung sind die Daten in unserem Smart-Contract noch alle nur On-Chain Daten.
Wir schreiben nun einen neuen Smart-Contract, um Daten aus der echten Welt abzufragen. Um keine externe API zu benötigen, legen wir uns auf das Generieren einer Zufallszahl fest. Wir verwenden Chainlink, um eine Zufallszahl zu generieren. Dazu gibt uns Chainlink «Basis»-Verträge, welche in unserem konkreten Smart-Contract vererbt werden.

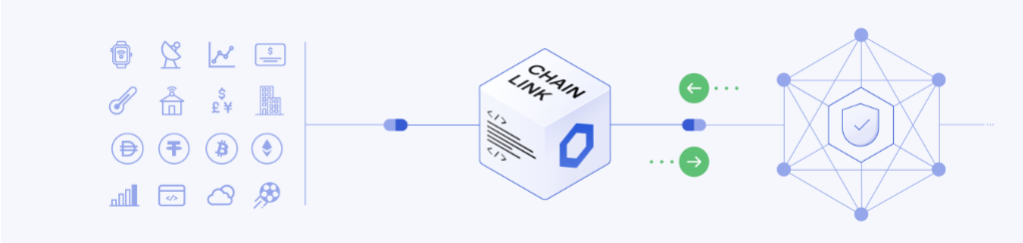


Abbildung 10 - Chainlink Architektur

Der Smart-Contract besitzt erneut eine Variabel welche die Zufallszahl speichert. Er hat des Weiteren eine Funktion, um eine neue Zufallszahl bei einem Oracle zu lösen. Das Aufrufen des Oracles geschieht über einen Coordinator von Chainlink.

3.3.3.1 VRF2 Subskription (Chainlink)

Der Smart-Contract nimmt Kontakt mit einer Chainlink VRF2 (Verifiable Random Function) Subscription auf. Diese Subscription muss zuerst im Chainlink-Portal registriert werden. Sie verfügt über ein eigenes Guthaben an LINK, der Währung von Chainlink.

Abbildung 11 - Chainlink VRF2 Subscription

Die Subscription verfügt zusätzlich über eine eigene Event-Historie aller Transaktionen. So können wir später prüfen, ob eine Transaktion erfolgreich war oder nicht.
Das Lösen einer neuen Zufallszahl via Oracles ist eine asynchrone Operation. Der Abfrage wir ein Callback mitgegeben, mit welchem der Wert der Zufallszahl später gesetzt werden kann. Auf der nächsten Seite sehen wir den Code für den Smart-Contract, welcher die Chainlink VRF2-Subscription für eine neue Nummer abfragt.

Um den Smart-Contract zu veröffentlichen, brauchen wir eine Subscription-ID, diese ID ist im Chainlink-Portal abrufbar.

```
constructor(uint64 subscriptionId) VRFConsumerBaseV2(vrfCoordinator) {
  COORDINATOR = VRFCoordinatorV2Interface(vrfCoordinator);
  s_owner = msg.sender;
  s_subscriptionId = subscriptionId;
}

// Assumes the subscription is funded sufficiently.
function requestRandomWords() external onlyOwner {
  // Will revert if subscription is not set and funded.
  s_requestId = COORDINATOR.requestRandomWords(
    keyHash,
    s_subscriptionId,
    requestConfirmations,
    callbackGasLimit,
    numWords
  );
}

function fulfillRandomWords(
  uint256, /* requestId */
  uint256[] memory randomWords
) internal override {
  s_randomWords = randomWords;
}
```

Code Snippet 10 - Konstrutor, Lösen einer neuen Zufallszahl sowie Setzen der neuen Zufallszahl im Smart-Contract Code

3.4 Deploy

Nach dem Kompilieren des Smart-Contracts aus Schritt 3.3.1 können wir mit dem letzten Menu-Punkt auf der linken Seite den kompilierten Smart-Contract in ein Netzwerk veröffentlichen.

Wenn der Smart-Contract einen Konstruktor mit einem Argument enthält, so müssen wie diesem Argument einen Wert zuweisen beim Veröffentlichen.

Im untenstehenden Bild sehen wir den HelloWorld Smart-Contract. Der erste orange Button in der Hälfte des Bildes dient zum Veröffentlichen des Contracts in die zuoberst ausgewählte Umgebung (Hardhat).

Gleich rechts neben dem orangen Knopf «Deploy» müssen wir einen initialen Wert für «message» eingeben.

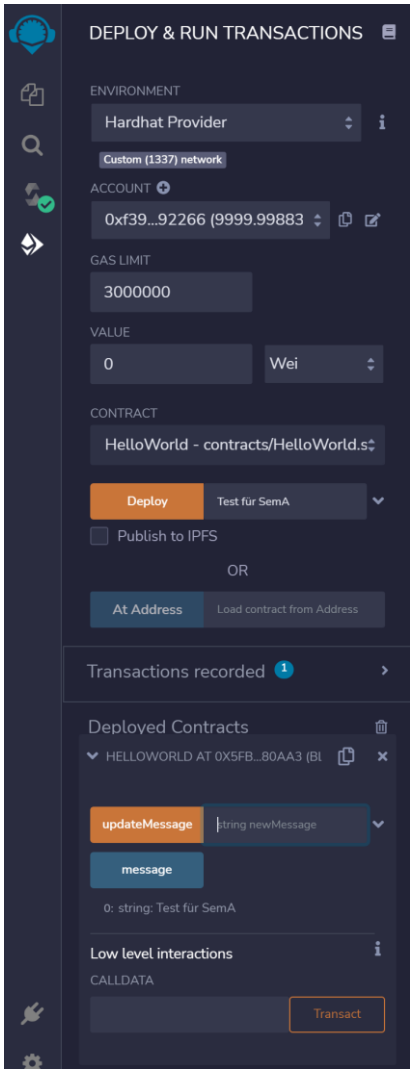


Abbildung 12 - Benutzeroberfläche für das Veröffentlichen eines Smart-Contracts in Remix

Sobald der Button Deploy auf der produktiven Ethereum Umgebung ausgeführt wird, werden alle Transaktionen in der Ethereum-Mainnet Blockchain registriert und verarbeitet.

4 Fazit

In der vorliegenden Arbeit wurde der aktuelle Forschungsstand zum Bereich Blockchain, Smart-Contracts und Oracles dargelegt und die Vorgehensweise zur Entwicklung eines Smart Contracts vorgelegt. Die entwickelte Smart Contract kann in die Ethereum Blockchain integriert werden.

Die erste Forschungsfrage «**Wie können Smart Contracts Daten aus der echten Welt abfragen?»** kann zusammenfasend Oracles als entscheidendes Werkzeug, für die Weitergabe von Daten aus der Aussenwelt auf die Blockchain weiterzuleiten. Dazu stehen unterschiedliche Typen von Oracles zur Verfügung.

Dabei kommt gleichzeitig die Frage nach der Sicherheit auf, die in der zweiten Forschungsfrage «**Gibt es ein Sicherheitsrisiko beim Abruf von Non-Blockchain Daten?»** formuliert wurde. Insgesamt muss mit einem Sicherheitsrisiko in Bezug auf die Blockchain und das Proof-of-Stake-System gerechnet werden. Manipulationen sind zudem im Sinne des „stake grinding“ in Bezug auf den Validator möglich.

Um die Sicherheit zu optimieren ist es sinnvoll zu wissen «**Wie werden Antworten eines Oracles von verschiedenen Nodes verifiziert?»**, was in der dritten Forschungsfrage festgehalten wurde. Der Proof-of-Work übernimmt eine dezentrale Validierung der Transaktionen, so dass ein Protokoll durch den Konsensmechanismus entsteht. Nutzer des Netzwerkes lösen eine problemorientierte Aufgabe und erzeugen so die Verifizierung.

Da der grundsätzliche Mechanismus dargelegt wurde, stellt sich zusätzlich abschliessend die Frage «**Was passiert, wenn sich die Daten zwischenzeitlich verändern?»** Wenn Smart-Contracts Daten aus einem einzelnen selbst-entwickelten Oracle beziehen, können Daten eine Sekunde später wechseln. Es auch sein, dass das Oracle zwischenzeitlich nicht verfügbar ist. Chainlink bietet eine-Standardlösung für diese Anwendungsfälle an. Ein Oracle von Chainlink wird erneut von verschiedenen dezentralen Nodes verifiziert und abgefragt. Es handelt sich nicht um eine proprietäre Plattform, sondern erneut um ein Netzwerk von Nodes welche nicht auf der Blockchain laufen.

5 Anhang

5.1 Abbildungsverzeichnis

5.2 Code-Snippet Verzeichnis

5.3 Literatur

[1] S. Nakamoto, «Bitcoin: A Peer-to-Peer Electronic Cash System», S. 9.
[2] B. Weiguny, «„Blockchain“: Bargeld, Banken und Betrüger», *FAZ.NET*. Zugegriffen: 16. Juni 2022. [Online]. Verfügbar unter: https://www.faz.net/aktuell/finanzen/devisen-rohstoffe/blockchain-soll-finanzwelt-revolutionieren-14120922.html
[3] «Blog: Chainlink und Smart Contracts». https://www.northcrypto.com/announcement/chainlink-und-smart-contract (zugegriffen 11. Juli 2022).
[4] «How to Build and Deploy a Solana Smart Contract», *Chainlink Blog*, 17. September 2021. https://blog.chain.link/how-to-build-and-deploy-a-solana-smart-contract/ (zugegriffen 11. Juli 2022).
[5] «Mastering Ethereum [Book]». https://www.oreilly.com/library/view/mastering-ethereum/9781491971932/ (zugegriffen 16. Juni 2022).
[6] «Consensus mechanisms», *ethereum.org*. https://ethereum.org (zugegriffen 16. Juni 2022).
[7] V. Buterin, «Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.», S. 36.
[8] G. Zheng, L. Gao, L. Huang, und J. Guan, *Ethereum Smart Contract Development in Solidity*, 1st ed. 2021 edition. Singapore: Springer, 2020.
[9] «Cryptocurrency Prices, Charts And Market Capitalizations», *CoinMarketCap*. https://coinmarketcap.com/ (zugegriffen 16. Juni 2022).
[10] «Was ist Ethash? - 0-100 Crypto Academy». https://0-100.io/glossar/ethash (zugegriffen 16. Juni 2022).
[11] «Ether Mining», *BTC-ACADEMY*. https://www.btc-echo.de/academy/bibliothek/wie-funktioniert-ether-mining/ (zugegriffen 16. Juni 2022).
[12] etherscan.io, «Ethereum (ETH) Blockchain Explorer», *Ethereum (ETH) Blockchain Explorer*. http://etherscan.io/ (zugegriffen 11. Juli 2022).
[13] «Benefits of Cryptocurrency», *Blockgeeks*, 20. November 2016. https://blockgeeks.com/5-benefits-cryptocurrency/ (zugegriffen 16. Juni 2022).
[14] «What is Ethereum? | Nuri Academy». https://nuri.com/academy/what-is-ethereum/ (zugegriffen 16. Juni 2022).
[15] A. Kaucher, «Blockchain: Funktionsweise und Applikationsmo¨glichkeiten», S. 10.
[16] «Was ist Ethereum (ETH)? Eine Einführung in das Blockchain-Projekt», *Blockchainwelt*. https://blockchainwelt.de/ethereum/ (zugegriffen 16. Juni 2022).
[17] R. Smith, «What is Delegated Proof of Stake?», *CoinCentral*, 3. Juli 2018. https://coincentral.com/what-is-delegated-proof-of-stake-exploring-the-consensus-algorithm/ (zugegriffen 16. Juni 2022).
[18] I. Bashir, *Mastering Blockchain: A deep dive into distributed ledgers, consensus protocols, smart contracts, DApps, cryptocurrencies, Ethereum, and more, 3rd Edition*. Birmingham Mumbai: Packt Publishing, 2020.
[19] «Blockchaincenter - Crypto Prices, Tools & Guides», *Blockchaincenter*. https://www.blockchaincenter.net/en/ (zugegriffen 16. Juni 2022).
[20] Chainlink, «Oracle Chainlink». https://chain.link/education/blockchain-oracles (zugegriffen 17. April 2022).

[21] «Was ist ein Oracle? Chainlink und Band erklärt», *Ledger*. https://www.ledger.com/de/academy/was-ist-ein-oracle-chainlink-und-band-erklaert (zugegriffen 16. Juni 2022).
[22] «Oracles», *BTC-ACADEMY*. https://www.btc-echo.de/academy/bibliothek/oracles/ (zugegriffen 16. Juni 2022).
[23] «Focus on cyber resilience», *BaFin*. https://www.bafin.de/SharedDocs/Veroeffentlichungen/EN/Fachartikel/2019/fa_bj_1904_Cyber-Resilienz_en.html (zugegriffen 16. Juni 2022).
[24] «Cryptographic Proof: Oracles & Stablecoins», *https://supraoracles.com/*. https://supraoracles.com/academy/cryptographic-proof-oracles-stablecoins/ (zugegriffen 16. Juni 2022).
[25] Valentin, «Blockchain Oracles», *BlockchainHub*. https://blockchainhub.net/blockchain-oracles/ (zugegriffen 11. Juli 2022).
[26] «Was ist Chainlink? - LINK Coin, Kurs und Prognose», *Bitcoin2Go*, 28. Mai 2020. https://bitcoin-2go.de/was-ist-chainlink/ (zugegriffen 11. Juli 2022).
[27] «Documentation | Ethereum development environment for professionals by Nomic Foundation». https://hardhat.org (zugegriffen 11. Juli 2022).
[28] «Etherscan Client Portal and Services». https://etherscan.io/myapikey (zugegriffen 11. Juli 2022).
[29] «Introduction | MetaMask Docs». https://docs.metamask.io/guide/ (zugegriffen 11. Juli 2022).
[30] *React* ·. Meta, 2022. Zugegriffen: 11. Juli 2022. [Online]. Verfügbar unter: https://github.com/facebook/react
[31] «Solidity — Solidity 0.8.15 documentation». https://docs.soliditylang.org/en/v0.8.15/ (zugegriffen 11. Juli 2022).
[32] «Remix - Ethereum IDE & community», *Remix - Ethereum IDE & community*. https://ethereum.github.io/ (zugegriffen 11. Juli 2022).
[33] «Onboarding Library | MetaMask Docs». https://docs.metamask.io/guide/onboarding-library.html (zugegriffen 11. Juli 2022).

5.4 Selbständigkeitserklärung

«Ich erkläre hiermit, dass ich diese Thesis selbständig verfasst und keine andern als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche kenntlich gemacht. Ich versichere zudem, dass ich bisher noch keine wissenschaftliche Arbeit mit gleichem oder ähnlichem Inhalt an der Fernfachhochschule Schweiz oder an einer anderen Hochschule eingereicht habe. Mir ist bekannt, dass andernfalls die Fernfachhochschule Schweiz zum Entzug des aufgrund dieser Thesis verliehenen Titels berechtigt ist.»

Zürich, 13.07.2022, 

Ort, Datum, Unterschrift