HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

Bachelorarbeit

Herr
Roy Michaelis

Evaluierung und Implementierung von Orakeln zur Bereitstellung von externen Daten für Smart Contracts in Ethereum

Mittweida, 2017

Fakultät Angewandte Computer und
Biowissenschaften

Bachelorarbeit

Evaluierung und Implementierung von Orakeln zur Bereitstellung von externen Daten für Smart Contracts in Ethereum

Autor:

Herr Roy Michaelis

Studiengang:
Informatik

Seminargruppe:
IF12w1-B

Erstprüfer:
Herr Prof. Dr. Andreas Ittner

Zweitprüfer:
Herr Dipl. Ing.(FH) Michael Meisel

Einreichung:
Mittweida 27.01.2017

Verteidigung/Bewertung:
Mittweida, 2017

Faculty Applied Computer Sciences &
Biosciences

Bachelorthesis

Evaluation and implemenation of oracles to provide external data for smart contracts in ethereum

author:
Mr. Roy Michaelis

course of studies:
Computer Sciences

seminar group:
IF12w1-B

first examiner:
Mr. Prof. Dr. Andreas Ittner

second examiner:
Mr. Dipl. Ing.(FH) Michael Meisel

submission:
Mittweida, 27.01.2017

defence/ evaluation:
Mittweida, 2017

Bibliografische Beschreibung:

Michaelis, Roy:
Evaluierung und Implementierung von Orakeln zur Bereitstellung von externen Daten für Smart Contracts in Ethereum. - 2017 – V, 52, VIII S.
Mittweida, Hochschule Mittweida, Fakultät Angewandte Computer und Biowissenschaften, Bachelorarbeit, 2017

Referat:

In dieser Arbeit geht es um sogenannte Orakel. Dies sind Anbieter die es ermöglichen, externe Daten in Smart Contracts der Ethereum Plattform, einzubinden. Im Zuge dieser Arbeit soll zunächst evaluiert werden, welche Orakel es derzeit schon gibt und wie diese Funktionieren. Es soll eine vergleichende Analyse durchgeführt werden, welche die Eigenschaften der verschiedenen Ansätze untersucht und bewertet. Wenn möglich sind alternative Ansätze zu erarbeiten. Zum Abschluss ist ein Smart Contract zu programmieren, welcher entweder von einem selbst entwickelten oder einem existierenden Orakel zum Abfragen von externen Daten gebraucht macht.

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ABI	Application Binary Interface
API	Application Programming Interface
ASIC	Application-Specific Integrated Curcuit
DAO	Decentralised Autonomous Organisation
DOS	Denial of Service
EOA	Externally Owned Accounts
IPFS	InterPlanetary File System
UTXO	Unspent Transaction Outputs

1 Übersicht

Im einleitenden Kapitel wird die Motivation und Zielsetzung dieser Bachelorarbeit genau erläutert. Weiterhin wird es eine kurze Übersicht zu allen Kapiteln dieser Arbeit geben.

1.1 Einleitung

Ethereum ist eine Kryptowährung, mit der sich Smart Contracts programmieren lassen. Dies sind von Computern automatisch geprüfte und ausgeführte Verträge. (Vgl. 1) Dabei gibt es Anwendungsfälle, in denen solche Verträge der Ethereum Plattform auf externe Daten zugreifen müssen.

Diese Daten werden zum Beispiel bei Versicherung benötigt, die in Smart Contracts abgebildet werden und automatisiert eine Zahlung auslösen, sobald ein vordefinierter Fall eingetreten ist. So genannte Orakel sind eine Möglichkeit, diese Daten für Smart Contracts bereit zu stellen.

Ein Problem dabei ist, dass diese Thematik noch recht neu ist, und es deswegen noch nicht viele Anbieter gibt, die diesen Service bereitstellen.

Daher ist das Ziel der vorliegenden Arbeit, Anbieter zu finden und diese zu untersuchen. Es wird evaluiert, welche Orakel es derzeit schon gibt, und wie sie den Service implementiert haben. Dabei wird eine vergleichende Analyse durchgeführt, welche die Eigenschaften der verschiedenen Ansätze untersucht und bewertet. Anhand eines Beispiels, wird gezeigt, wie der Service zu nutzen ist.

Als abschließende Aufgabe, wird ein solches Orakel lokal abgebildet und dessen Aufbau und Ablauf erläutert. Weiterhin werden 2 Contracts geschrieben, welche von dem Dienst Gebrauch machen, um eine beispielhafte Nutzung zu aufzuzeigen.

1.2 Übersicht Kapitel

Diese Bachelorarbeit besteht aus fünf Kapitel.

In **Kapitel 2** werden die theoretischen Grundlagen für diese Arbeit aufgezeigt. Dabei handelt es ich um die Ethereum Technologie und dessen Bezug zu Bitcoin, Smart Contracts und welche Bedenken es dabei gibt. Auch wird die Funktion eines Orakels genauer beschrieben und welche Ansätze es gibt, um die Sicherheit der Daten aus den Quellen zu gewährleisten.

Anschließend werden in **Kapitel 3** die Anbieter vorgestellt, untersucht und verglichen. Es wird eine Einschätzung zu jeder Kategorie geben, sowie eine abschließende Zusammenfassung.

Nachfolgend wird in **Kapitel 4** der Dienst eines Orakels lokal abgebildet und erläutert. Dabei werden die dafür notwendigen Schritte dokumentiert.

Im letzten **Kapitel** gibt es ein abschließendes Fazit zu der Arbeit und einen kurzen Ausblick auf neue Ansätze.

2 Grundlagen

In diesem Kapitel geht es um die Grundlagen, die für diese Arbeit wichtig sind. Dabei wird zuerst Ethereum genau erklärt und kurz auf Bitcoin, als Referenz eingegangen. Die Funktionsweise von Ethereum ist dabei ein wichtiger Bestandteil des Abschnittes. Anschließend wird der Begriff Smart Contract kurz im Allgemeinen, sowie ausführlich in Bezug auf Ethereum erklärt. Der letzte Abschnitt beschäftigt sich mit Orakeln, worum es sich dabei handelt und welche Aufgabe sie haben. Ein wichtiger Punkt hierbei ist die Sicherheit der Datenquellen und welche Ansätze es dafür gibt.

2.1 Ethereum

Einleitung

Ethereum ist eine dezentrale, digitale Währung, welche von den Nutzern betrieben wird und ohne zentrale Autorität oder Vermittler auskommt. Die Rechner der Nutzer sind mittels eines Peer-to-Peer Netzwerkes verbunden, erstellen Transaktionen, führen diese aus und speichern diese in Blöcken ab. Sie werden dann in einer Kette aneinandergehangen, der sogenannten Blockchain. Es wird jeweils nur der aktuellste Block an das Ende der Kette angehangen und jeder enthält den mathematischen Beweis, der ihn als korrekten Nachfolger des vorhergehenden Blocks verifiziert. (Vgl. 2)

Es ist eine offene Plattform, welche auf dem Prinzip und der Funktionsweise von Bitcoin aufbaut und von niemanden kontrolliert werden kann, da ihr Quellcode für jeden zugänglich ist, und viele Personen bei der Entwicklung mitgewirkt haben. Dadurch werden auftretende Fehler zeitnah behoben und stetig neue Funktionen hinzugefügt. Das Ethereum Protokoll ist als sehr flexibel und anpassungsfähig gedacht und erlaubt somit das einfache Erstellen von Programmen auf dieser Plattform. (Vgl. 2) (Vgl. 3)

Ethereum basiert auf Bitcoin, was 2008 von Satoshi Nakamoto in seinem Paper: „Bitcoins: A Peer-to-Peer Electronic Cash System“ (2) vorgestellt und beschrieben wurde, basierend auf einem Konzept namens „Krypto-Währung“ von Wei Dai aus dem Jahr 1998. (4) (3) 2009 wurde die erste Version von Bitcoin von Nakamoto in einer Kryptographie-Mailingliste veröffentlicht und gilt damit auch als Machbarkeitsbeweis. Damit ist er der Erfinder der neuen Technologie mit einer exponentiell gewachsenen Gemeinschaft. (Vgl. 5)

2014 begannen die Gründer von Ethereum Vitalik Buterin, Garvin Wood und Jeffrey Wilcke die Arbeit an einer Blockchain der nächsten Generation. Sie sollte die Basis für die Smart Contract Plattform sein. Dabei folgten sie fünf einfachen Prinzipien: (Vgl. 6)

1. Um die Benutzung für Anwender und Entwickler so einfach wie möglich zu halten, wurden alle komplexen Funktionen in eine Schicht programmiert, zu welcher man von außen keinen direkten Kontakt hat.

2. Es gibt keinerlei Beschränkungen was die Nutzung angeht, was Anwendern wiederum große Freiheiten gewährt. Gleichfalls wird durch Transaktionsgebühren sichergestellt, dass diese nicht missbraucht werden.

3. Alle Funktionen und Operationen sind so einfach wie möglich gehalten, um diese in vielen möglichen Kombinationen nutzen zu können. Somit kann eine Reihe von einfachen Funktionen effizienter gestaltet werden, indem einige Funktionalitäten herausgenommen werden, wenn sie nicht nötig sind.

4. Als Konsequenz daraus, wurden ein paar Funktionen absichtlich nicht implementiert, da sie von den Anwendern selbst umgesetzt werden können.

5. Die Entwickler sind bereit ein größeres Risiko bei einigen Funktionsweisen einzugehen, falls sich daraus wesentliche Vorteile ergeben.

Natürlich bringt die Technologie auch Probleme mit sich. Eines davon ist die sogenannte 51% Attacke. Dabei verschafft sich ein Angreifer über 51% der Rechenleistung im Netzwerk und hat somit ein hohes Maß an Kontrolle über dieses. Wobei die Chancen für eine solche Attacke, auch schon mit etwas weniger als 50% der Leistung, sehr hoch sind. (Vgl. 7) Der Angreifer hat dann die Möglichkeit, das Senden von Transaktionen für bestimmte Adressen oder Contracts zu verhindern, Geld doppelt auszugeben und Miner, für eine kurze Zeit, am Finden von neuen Blöcken zu hindern. (Vgl. 8) Wenn eine solche Attacke über einen langen Zeitraum ausgeführt wird, kann dies den Konsens und damit auch die Währung sowie das System zerstören. Das Ganze ist allerdings mit auch enormen Kosten für den Angreifer verbunden, da er sich entsprechend der Gesamtleistung im Netz, über 51% aneignen und betreiben muss. Somit lohnt sich der Aufwand und die Kosten nur, wenn es dem Angreifer darum geht, den Konsens zu zerstören und die Währung zu Fall zu bringen.

Funktionsweise

Ethereum hat viele Parallelen bei der Funktionsweise zu Bitcoin. Es besitzt auch eine Blockchain und setzt ebenso auf ein dezentrales Rechnernetz. Als Währung kommen hier Ether zum Einsatz. Zudem bietet Ethereum eine eingebaute, Turing vollständige Programmiersprache, mit denen Smart Contracts geschrieben werden können. Weiterhin kann man dezentrale Anwendungen damit entwickeln, bei deren der Programmcode verteilt ausgeführt wird, es aber trotzdem eine Nutzeroberfläche gibt.

Bei Ethereum müssen Nutzer eine kleine Gebühr für das Nutzen des Netzwerkes zahlen. Diese verhindert Spam Attacken wie DOS und unendliche Schleifen.

Für jeden Schritt im Programm, also auch für Rechenarbeit und das Nutzen von Speicher, muss der Sender der Transaktion eine Gebühr in Ether zahlen.

Ethereum setzt auf einen Proof-of-Work (PoW) Algorithmus, der „Memory-hard“ (Vgl. 9 S. 23-28) ist. Dies soll das Problem der immer schnelleren Berechnung lösen, indem auch Speicher zur Berechnung genutzt wird. Somit ist es nicht mehr möglich, effizientere und schnellere Recheneinheiten zu bauen, weil die Größe von Speichereinheiten eine physikalische Grenze erreich hat und kaum noch kleiner zu produzieren ist. Dies sorgt dafür, dass es noch keine ASICs gibt, da sich die Entwicklung solcher für Ethereum noch nicht rentiert. GPUs sind dank des integrierten Speichers und der hohen Geschwindigkeit bei parallelen Berechnungen bisher am besten zum Minen von Blöcken geeignet. Somit können auch normale Rechner mit den passenden Grafikkarten zum Minen genutzt werden. Dadurch wird die Sicherheit erhöht, weil die Rechenleistung dezentraler verteilt und nicht in großen Rechenfarmen konzentriert ist, wie es mit dem PoW von Bitcoin der Fall ist. (Vgl. 10)

Da der Proof-of-Work Algorithmus, den Bitcoin verwendet, beispielsweise einen so hohen Energieverbrauch wie Island aufweist (Vgl. 11), wird er als sehr umweltschädlich angesehen. Daher wird an einem Ansatz gearbeitet, der einen besseren Konsens-Algorithmus hervorbringen soll, den sogenannten Proof-of-Stake. Dieser nutzt einen Mechanismus der als „Virtual Mining“ bezeichnet wird. Im Gegensatz zu PoW, der auf Hardware setzt, soll PoS von den Münzen in der Blockchain abhängig gemacht werden. Anstatt Hardware im Wert von 1000€ zum Minen zu kaufen, werden beim PoS Münzen im selben Wert und damit ein „Recht“ zum Minen von Blöcken erkauft. Die Münzen werden im Algorithmus deponiert und dieser wählt dann zufällig einen Miner aus, der an einem neuen Block arbeiten darf. Natürlich bekommt man mit dem doppelten Einsatz auch die doppelte Chance darauf, die Rechte zum Arbeiten an neuen Blöcken zu erhalten. Falls ein Miner einen Block nicht innerhalb einer gewissen Zeit validiert hat, wird ein zweiter ausgewählt, der daran arbeiten darf. (Vgl. 12)

Events sind ein Teil von Contracts, und speichern die übergebenen Parameter im Transaktionsprotokoll, einer speziellen Datenstruktur in der Blockchain, wenn sie aufgerufen werden. Die Protokolle werden mit den Transaktionen fest verknüpft und mit in die Blockchain eingebunden. Aus dem inneren der Contracts kann nicht darauf zugegriffen werden. (Vgl. 13)

State Transition Function

Ein Zustand (State) wird durch das Ausführen von Transaktionen innerhalb der State Transition Function in den Folgezustand überführt. In Bitcoin besteht der State aus dem Unspent Transactions Output (UTXOs) Cache und in Ethereum aus Accounts. Ein solcher Account besteht aus einem Kontostand, der Adresse, dem Contract Code und einen internen Speicher.

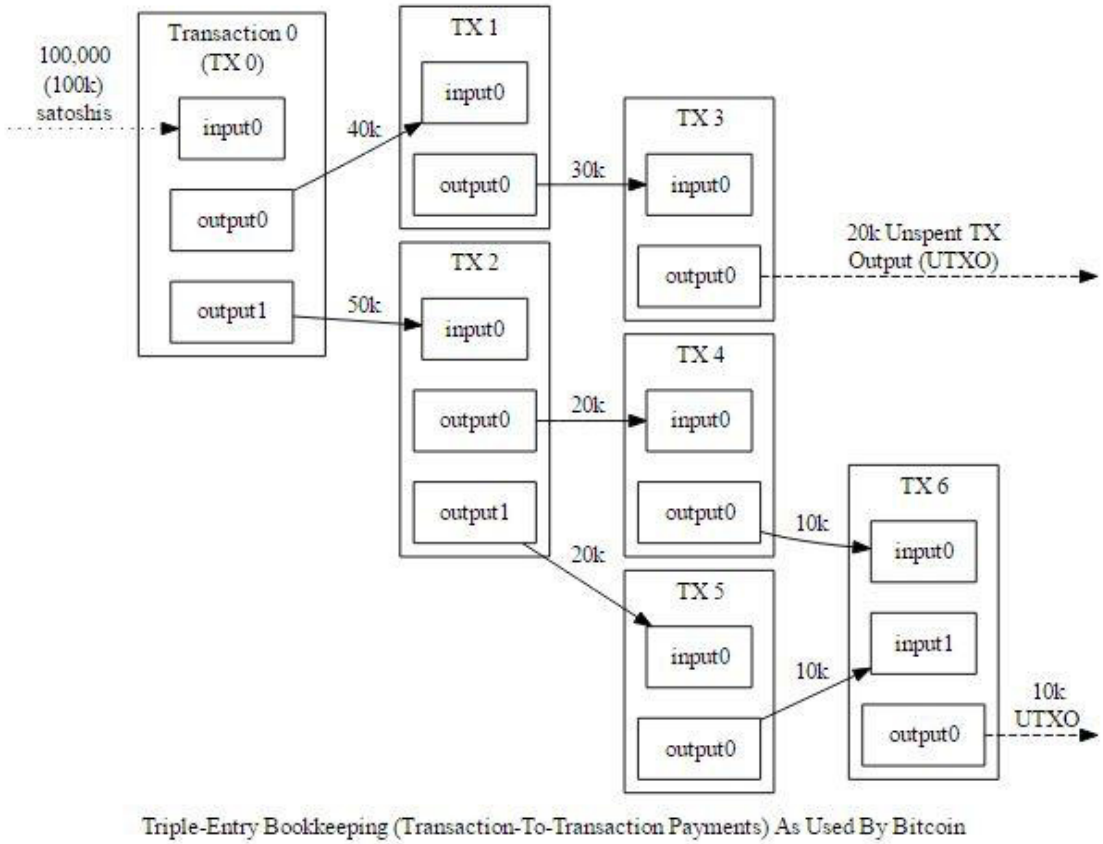

Abbildung 1: UTXO in Bitcoin (Vgl. 14)

Während mit dem UTXOs der Weg jeder Münze genau nachverfolgt werden kann, und es somit unmöglich ist, Münzen auszugeben die man nicht besitzt, ist dies bei Accounts mit Transaktionen umgesetzt, welche gültig sind, wenn der Sender dafür bezahlen kann. Dann wird dem Empfänger der Wert gutgeschrieben und vom Konto des Absenders abgezogen. Falls die Transaktion zu einem Contract Account geschickt wurde, wird der dort vorhandene Code anschließend ausgeführt. (Vgl. 6)

Dabei gibt es zwei Arten von Accounts:

- Die Externally Owned Accounts (EOAs) also Accounts in Fremdbesitz die von Privaten Schlüsseln kontrolliert werden.
- Contract Accounts, also Vertrags-Accounts die vom eigenen Code kontrolliert werden und nur von einem EOA aktiviert werden können.

Der Unterschied zwischen EOAs und Contract Accounts ist der, dass EOAs von der Person kontrolliert werden, die auch Zugriff auf die Privaten Schlüssel hat. Contract Accounts können nur von dem Programmcode im Contract geleitet werden. Wenn sie von einer Person kontrolliert werden sollen, müssen sie auch so programmiert werden. Dann wird eine spezifische Adresse von einem EOA angegeben, welche dann Zugriff auf die Funktionen hat. (2)

Die Vorteile von UTXOs und Accounts sind in der folgenden Tabelle gegenübergestellt:

UTXO	Accounts
Mehr Sicherheit, weil für jede Transaktion eine neue Adresse verwendet wird.	Einsparungen beim Speicherplatz, weil Accounts alle UTXOs vereinigt und Transaktionen in Ethereum um den Faktor 2-2,5 kleiner sind als in Bitcoin.
Mehr Möglichkeiten für Skalierung, weil nur der Besitzer der Münzen den Merkle-Baum für dessen Besitznachweis verwaltet.	Bessere Übertragbarkeit von Münzen, weil es kein Blockchain Konzept für die Herkunft dieser gibt.
	Accounts sind einfacher zu programmieren und zu verstehen, besonders sobald man es mit komplexeren Skripten zu tun bekommt.
	Der Bezug von Münzen bleibt immer erhalten und einfache Clients können jederzeit alle Daten eines Accounts beziehen.

Tabelle 1: Vergleich UTXO & Accounts (Vgl. 6)

Ethereum Blockchain

Ein großes Problem bei dem Versuch, Blöcke einer Blockchain in sehr kurzen Abständen zu erzeugen ist, dass die Sicherheit enorm darunter leidet und es zu einer hohen Verwurfsrate kommt. Weil es immer eine gewisse Zeit dauert um einen, durch Mining erstellen Block im Netzwerk zu verteilen, kann es vorkommen, das ein Block von Miner A zu lange braucht um zu Miner B zu gelangen. Er wird also seinen eigenen Block als richtig ansehen und daran weiterarbeiten, obwohl er am Block von Miner A weiterarbeiten müsste und sein Block verworfen wurde. Es kommt folglich zur Zentralisation: Angenommen Miner A ist ein Miningpool mit 35% Hashrate und B einer mit 10%. Dann hat A eine Verwurfsrate von 65% (zu 35% werden die Blöcke selbst erzeugt und können somit nicht verworfen werden) und B von 90%. Wenn das Intervall der Blockerstellung also so kurz ist, dass es zu einer hohen Verwurfsrate kommt, wird Miner A wesentlich effektiver arbeiten, weil weniger Blöcke verworfen, und somit auch die Kontrolle über das Netzwerk haben. (Vgl. 6)

Einen Ansatz zur Lösung dieses Problems haben Yonatan Sompolinsky und Aviv Zohar mit dem im Dezember 2013 veröffentlichten „Greedy Heaviest Observed Subtree“ GHOST Protokoll vorgestellt. (Vgl. 15) Dabei werden die abgelaufenen Blöcke in die Berechnung der längsten Kette mit aufgenommen. Ethereum adaptiert dieses Modell mit der Besonderheit, dass Miner für verworfene Blöcke und die, die sie integrieren, eine anteilige Vergütung (7/8 des Basiswertes) bekommen. Dies reduziert den Effekt der Zentralisierung von Mining Pools im Netzwerk mit großer Hashrate und reduziert die Rate verworfener Blöcke. Es werden Onkelblöcke (verworfene) nur bis zu einer Blocktiefe von sieben inkludiert. Folgende fünf Eigenschaften hat der Blockzeitalgorithmus: eine Blockzeit von 12 Sekunden, ein Limit von sieben Blöcken für Vorfahren, ein Limit von einem Block für Abkömmlinge, Gültigkeitsbedingungen für Onkelblöcke und das Aufteilen von Belohnungen.

Ethereum Virtual Machine

In der Ethereum Virtual Machine (EVM) wird der Transaktions- und Contract Code ausgeführt. Sie ist gleichzeitig der Hauptunterschied zwischen Ethereum und anderen Modellen. Entwickler können Programme in bereits bestehenden Programmiersprachen wie JavaScript oder Phyton schreiben, und sie in der EVM ausführen. Auf jedem Netzknoten in Ethereum läuft eine EVM und auf allen wird der gleiche Code ausgeführt. Dies ist so konzipiert, um den Konsens im Netzwerk zu bewahren. Dieser dezentrale Konsens sorgt für eine sehr hohe Fehlertoleranz, komplette Sicherheit gegen Ausfälle und macht die Daten in der Blockchain unveränderbar. (Vgl. 16)

Anwendungsbeispiele

Durch die Implementierung einer Turing vollständigen Programmiersprache, ergeben sich sehr viele mögliche Anwendungsfälle. Smart Contracts sind nur einer davon.

Das Unternehmen Slock.it bietet zum Beispiel einen Service an, bei dem materielle Dinge wie Speicherplatz oder die Rechenleistung eines Servers per Ethereum zur Vermietung angeboten werden. (Vgl. 17)

Um ihre Idee zu finanzieren, wollten die Brüder Jentzsch nicht auf traditionelle Investoren setzten und kamen auf den Gedanken, das Ganze per Crowdfunding zu realisieren. Dafür gründeten sie die DAO, welches eine Investmentfirma war, die nur aus Code bestand. Es ist die erste weltweit, welche mithilfe von Crowdfunding entstanden ist und wo jeder, der Anteile an der Firma hält, bei Wahlen stimmberechtigt ist. (Vgl. 18 S. 1)

Decentralized Apps erlauben die Umsetzung von vielen neuen Ideen. KYC-Chain erlaubt etwa einen Konsens über die Identität von Personen zu erstellen und ein sehr hohes Level an Vertrauen zu bewahren. Dabei haben nur die Nutzer selbst die Erlaubnis, mittels ihres privaten Schlüssels, anderen Zugriff auf ihre Identität, und nur unter bestimmten Bedingungen, zu gestatten. Dabei werden die Daten von den Kryptographischen Protokollen der Blockchain abgesichert. (Vgl. 19)

2.2 Smart Contracts

Schon 1994 wurde der Begriff „Smart Contract“ von dem Juristen und Informatiker Nick Szabo geprägt. Er definierte einen Smart Contract als ein computerisiertes Transaktionsprotokoll, welches die Bedingungen eines Vertrages ausführt. Die hauptsächlichen Aufgaben eines Smart Contracts sind demnach: das Ausführen von häufig vorkommenden Dingen wie Zahlungen oder Pfandrechten, verhindern von Fehlern, dem Einsparen von Vertrauenspersonen, das Verhindern von Betrug und Einsparen von anderen Transaktionskosten. (20) Einfache Smart Contracts sind zum Beispiel beim Digitalen Rechtemanagement und beim Nutzen eines EC-Automaten zu finden. Dort werden jeweils automatisch Vertragsbedingungen geprüft und die entsprechende Aktion ausgeführt.

In Ethereum bestehen Smart Contracts üblicherweise aus einer „Wenn – Dann – Sonst“ Logik. Die Programmierung kann auf verschiedenste Weise geschehen. Es gibt Online bzw. Offline Compiler, mit denen die Contracts geschrieben und kompiliert werden können. Und es gibt Anbieter, die dies automatisiert, mithilfe einiger übergebener Parameter tun. (Vgl. 21) Nach der Kompilierung werden die Contracts Deployed, das heißt, per Transaktion in das Netzwerk verteilt, indem der Ethereum Virtual Machine (EVM) Code an eine leere Adresse gesendet wird. (Vgl. 22 S. 220)

Dort werden sie dann in der (EVM) ausgeführt, welches eine quasi-Turing-vollständige Maschine ist, und auf einer einfachen stapelartigen Architektur basiert. Diese führt die Anweisungen in Bytecode aus, und wird nur durch das „Gas“ begrenzt. (Vgl. 23 S. 10) Jede Funktion hat einen zugewiesenen Gas-Wert basierend auf den Kosten, welche bei der Ausführung verursacht werden. Das Gas wird dynamisch in Ether umgerechnet und muss mit der Transaktion gezahlt werden. Weiterhin gibt es ein Gas-Limit, um Nutzer vor Fehlern in der Programmierung zu schützen, welche sehr viel davon verbrauchen können. (Vgl. 24)

Bedenken und Nachteile bei Smart Contracts

Die Auswahl der Programmiersprache ist besonders wichtig, da diese eine potentielle Angriffsfläche ist. Falls die gewählte Sprache Lücken oder Fehler aufweist, können Angreifer diese ausnutzen und großen Schaden anrichten. (Vgl. 25) (Vgl. 26) Ebenfalls muss das Programm fehlerfrei programmiert worden sein, da ein Fehler im Code eine Lücke für Angriffe bietet oder die Funktion des Contracts beeinträchtigen kann. (Vgl. 25) Als Beispiel ist hier die Attacke auf die DAO im Juni 2016 zu erwähnen, als ein Unbekannter durch eine Sicherheitslücke, Tokens im Wert von mehreren Millionen Dollar erbeutet hat.

Eine andere Bedrohung sind sogenannte DOS Attacken, bei denen das Netzwerk mit Transaktionen überflutet wird, bei denen die Nodes z.B. Status Informationen von Festplatten einlesen müssen und dadurch die Erstellung von Blöcken verlangsamen. Dies geschah Ende September 2016 und sorgte dafür, dass die Erstellung von Blöcken zwischen 30 und 60 Sekunden dauerte, was eine Verlangsamung um den Faktor 2-4 ist. (Vgl. 27)

Dies betrifft nicht nur die Blöcke, sondern auch die Contracts, welche durch die längeren Blockzeiten, in ihrer Funktion beeinträchtigt werden.

Smart Contracts können unter Umständen auch im Finanzmarkt eingesetzt werden. Da es jedoch zu unvorhersehbaren Ereignissen kommen kann und bei der Programmierung nicht jeder mögliche Fall vorhersehbar ist, kann es von Nöten sein, dass ein Mensch in die Ausführung eingreifen muss. Dies ist jedoch derzeit nur begrenzt möglich, da diese Art der Nutzung nicht vorgesehen ist und daher die passenden Funktionen fehlen.

Der wohl größte Nachteil von Smart Contracts ist, dass sie nicht die gleiche Rechtsgrundlage haben wie ein normaler Vertrag. Da ein Contract lediglich ein Computerprogramm ist, kann er Willenserklärungen inhaltlich nicht ausdrücken. Somit ist er rechtlich kein Vertrag, sondern nur eine computerbasierte Umsetzung dessen und ermöglich eine Vereinfachung der Vertragsausführung. Falls es zu Streitigkeiten in der Auslegung eines Contracts kommt, kann ein Richter dies nicht direkt entscheiden, da der in den meisten Fällen nicht direkt les- und verstehbar ist. Es muss also ein Sachverständiger hinzugezogen werden. Auch ist nicht eindeutig geklärt, welches Recht bei Transaktionen mit internationalen Partnern angewendet werden muss. Bei herkömmlichen Verträgen wird dies zum Beispiel im vorherein festgelegt, um bei Vertragsverstößen bestimmen zu können, welcher Gesetze zur Anwendung kommen. (Vgl. 28) In Deutschland wird ein Vertag lediglich als software-gestützte Umsetzung eines traditionellen Vertrages angesehen und hat keine weitere Rechtswirkung. (29)

2.3 Orakel

In diesem Kapitel wird erläutert, was ein Orakel ist und welche Aufgabe es erfüllt. Dabei wird auf die Techniken eingegangen, die sicherstellen sollen, dass die Daten auch aus der angegebenen Datenquelle stammen und nicht verfälscht wurden.

Orakel – Definition und Aufgabe

Ein Orakel ist ein Dienst für Smart Contracts, dessen Hauptaufgabe es ist, Daten einer externen Datenquelle abzufragen und in einen Contract zu schreiben. Dies ist notwendig, da Smart Contracts von sich aus nicht auf Daten außerhalb der Blockchain zugreifen können. Für viele Anwendungsfälle, etwa Versicherungen, Wechselkurse oder einem verteilten Speichersystem, ist es allerdings wichtig, Zugriff auf solche Daten zu erhalten. Es kann fast jeder beliebige Wert als Datenquelle genutzt werden, solange er eine einfache Form wie z.B. eine Zahl, einen Ausdruck oder ein ja/nein besitzt. (Vgl. 30)

In Ethereum bieten Orakel den Service, Daten einer externen Datenquelle in einen Smart Contract einzubringen. Dabei kann der Kunde aus mehreren angebotenen Datenquellen wählen sowie die Laufzeit und weitere Features angeben. Die gängigsten sind Daten einer URL (zumeist Wechselkurse von Währungen), Blockchain und IPFS. Auch Dinge wie Sportwetten, persönliche Ziele (31), oder Fragen per WolframAlpha (32) lassen sich als Quelle für die externen Daten verwenden. Darauf wird später beim Vergleich der derzeitigen Anbieter noch genauer eingegangen.

Sicherheit der Datenquellen

Ein großes Problem bei Orakeln ist die Sicherheit der Datenquelle. Dies fordert ein hohes Maß an Vertrauen gegenüber dem Anbieter des Orakels. Einige der Probleme die auftreten können, sind:

- Der Prozess hinter dem Orakel ist nicht komplett einsehbar, daher kann man dem Ergebnis nie 100 Prozentig vertrauen.
- Das Orakel kann zwischen der Erstellung eines Contracts und dessen Auflösung verschwinden, und somit keinerlei Ergebnisse mehr liefern.
- Man kann das Orakel bestechen um absichtlich falsche Ergebnisse zu liefern. Dies kann allerdings mit TLSNotary verhindert werden.
  (Vgl. 33)

Wenn man einen Anbieter nutzt um Daten von externen Seiten in den Contract zu bringen, muss man sicher sein, dass die vom ihm verwendete Quelle auch korrekte Daten übermittelt. Theoretisch könnte der Anbieter auch selbst ausgedachte Werte übermitteln und den Ausgang des Vertrages verfälschen. Um dem Kunden gegenüber einen Beweis zu erbringen, dass die übertragenen Daten auch wirklich von der erwarteten Datenquelle stammen, gibt es zwei Ansätze. Zum einen den TLSNotary Proof, der ein Teil des Geheimnisses der HTTPS Sitzung nutzt um die Sicherheit zu gewährleisten.

Zum anderen gibt es den dezentralen Ansatz, welcher von Orisi.org verfolgt wird und mehrere Orakel als Datenquelle einsetzt. Das erhöht die Sicherheit enorm, da nun das Ergebnis von mehreren Anbietern übereinstimmen muss. (34)

2.3.1 TLSNotary

TLSNotary erlaubt es dem Anbieter einen Beweis zu liefern, dass eine bestimmte Website zu einem bestimmten Zeitpunkt aufgerufen wurde. Dies funktioniert indem der Aufrufer der Seite einen kleinen Teil der sicherheitsrelevanten Daten, die genutzt wurden um die HTTPS Sitzung aufzubauen, temporär speichert. Der Aufrufer legt dabei zu keinem Zeitpunkt die Sitzungsschlüssel offen oder dekodiert Daten ohne Authentifizierung. So wird die Sicherheit der TLS Sitzung bewahrt, obwohl ein paar der Geheimnisse preisgegeben werden. TLSNotary setzt derzeit die TLS Version 1.0 oder 1.1 voraus. In der folgenden Abbildung, ist die Funktionsweiße von TLSNotary noch einmal genauer erklärt: (Vgl. 35 S. 1)

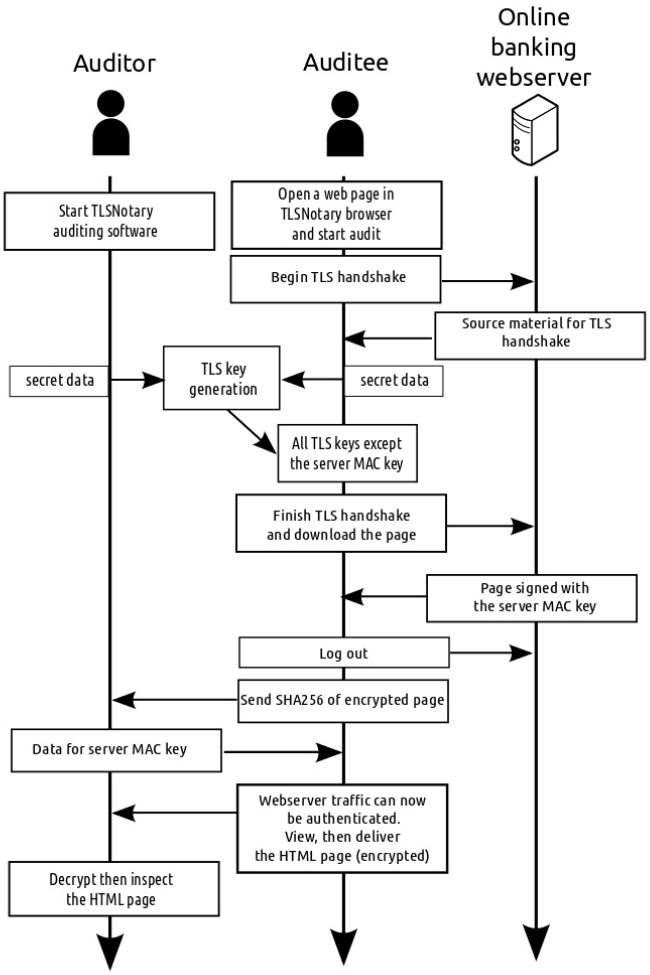

Abbildung 2: Funktionsweise TLSNotary (Vgl. 36)

Leider ist bisher Oraclize.it der einzige Anbieter der TLS nutzt, um die Echtheit der Daten zu beglaubigen. (Vgl. 37) Dafür muss man den 4-fachen Preis zusätzlich zu den Kosten für die normale Abfrage zahlen. Auch wenn das nur 1 bzw. 4 Dollarcent sind. Weiterhin wird TLSNotary nur für URLs, welche HTTPS unterstützen und Blockchain als Datenquelle angeboten. Für alle anderen Datenquellen wird dies derzeit noch nicht unterstützt. Oraclize.it nutzt eine Virtuelle Maschine von Amazon Web Service (AWS) um das TLSNotary Geheimnis zu speichern. Dieser Ansatz hat folgende Vor- und Nachteile: (Vgl. 38)

Vorteile

- Weil Oraclize ein AWS Orakel (Vgl. 39) nutzt, ist es für ihren Datenabrufdienst schwerer, die Inhalte einer Website zu fälschen.
- Jemand der in Oraclize's Orakel-Contract eindringt, hat nicht automatisch die Möglichkeit Ergebnisse zu fälschen.
- Die Verifizierung einer Signatur des AWS Orakel ist eine vergleichsweise „billige" Operation für einen Computer, somit könnte Oraclize's „honesty proof" auf der Blockchain verifiziert werden, falls der TLS notarisierte Inhalt kurz genug ist.

Nachteile

- Amazon selbst oder jeder der die AWS Plattform hacken kann, hat die Möglichkeit die „proofs of honesty" zu fälschen indem der Private Key des AWS gestohlen wird.
- Es gibt keine Möglichkeit jemanden anderem zu beweisen, dass man jemanden beim Fälschen von Oraclize's Beweisen ertappt hat.
- Auch wenn die Daten einer Website durch ein Orakel in einen Contract geschrieben werden, kann der Server der Seite immer noch falsche Daten ausgeben.

2.3.2 Dezentraler Ansatz

Eine andere Herangehensweise an das Thema ist, nicht nur eine einzige Datenquelle zu verwenden, sondern die Datenabfrage von mehreren verschiedenen Klienten erledigen zu lassen. Somit kann sichergestellt werden, dass eine Verfälschung der Daten, wie es bei nur einer Quelle möglich wäre, nicht stattfinden kann bzw. deutlich schwieriger ist. Dabei kann die Sicherheit mit einer 2/3 Mehrheit, ähnlich wie es bei einer Multisig Transaktion der Fall ist, gewährleistet werden. Natürlich wird die Chance auf die richtige Antwort erhöht, umso mehr Klienten die Quellenabfrage ausführen.

Orisi ist ein Projekt, welches sich genau auf diese Art von Service spezialisiert hat. Es verwendet mehrere Orakel um wirklich sicherzustellen, dass die angeforderten Daten, auch korrekt sind.

Der ganze Ablauf ist wie folgt beschrieben:

Zwei Partner einigen sich auf einen Contract, bei dem eine Zahlung getätigt wird, sobald eine Kondition wahr wird. Beispielsweiße sendet Alice 1 Bitcoin an Bob, sobald eine festgelegte Website „Regen“ für die Stadt Las Vegas meldet. Anschließend wählen Alice und Bob von der Liste der verfügbaren vorausgewählten Orakel, sieben Stück aus. Es können zwar alle möglichen Orakel gewählt werden, aber die vorausgewählten sind die zuverlässigsten und weiterhin spart es Zeit bei der Auswahl. Jetzt sendet Alice ihr Geld an eine kurzzeitig erstellte Multisigadresse. Das Geld wird darin aufbewahrt, bis 4 der 7 Orakel das tatsächliche Ergebnis ausgeben. Dann wird das Geld an Bob ausgezahlt. Die Zahlung geht an Alice zurück, falls das Ereignis nicht eintrifft.

Eine Adresse, die mit 4 aus 7 Signaturen funktioniert, ist aber nicht wünschenswert, da die Orakel ein willkürliches Ergebnis ausgeben und damit den Ausgang beeinflussen können. Um dies zu verhindern, soll auch die Signatur des Empfängers mit in die Multisigadresse einbezogen werden. Da eine 1+m aus n Signatur allerdings nicht standardisiert ist, wird eine 8 aus 11 Signatur erstellt (n + 1 aus 2n – m +1). Dies lässt sich einfach logisch Nachvollziehen: Bobs Signatur soll für die Entscheidung unbedingt notwendig sein. Wenn man Bob nun die Möglichkeit bietet, drei Signaturen einzubringen, entsteht eine 7 aus 10 Multisigadresse. Allerdings können die sieben Signaturen auch alle von den Orakeln stammen und Bobs Entscheidung ist nicht mehr von Relevanz. Es braucht also mindestens acht Signaturen, sieben von den Orakeln und eine weitere von Bob, damit die Multisigadresse zu einer Entscheidung kommt. Somit ergibt sich eine 8 aus 11 Multisigadresse, bei der sieben Orakel und viert Signaturen von Bob inbegriffen sind.

Nun wird von Alice eine „entsperr“ Transaktion erstellt, welche die Gelder der Multisigadresse ausgibt und gleichzeitig auch die Orakel und das Orisi Projekt zahlt. Alice sendet die Nachricht jetzt z.B. mittels einer Bitmessage an die einzelnen Orakel, mitsamt den Regeln für eine Rückerstattung. Das Bitmessage Protokoll wird verwendet, um die IP Adresse der Orakel zu schützen und damit mehr Sicherheit zu bieten. Die Orakel prüfen die Transaktion auf ihre Gültigkeit, fügen sie zu ihrer Aufgabenliste hinzu und erklären ihre Kenntnisnahme. Beide Partner prüfen ob die Orakel die Transaktion in ihrer Liste haben, und senden das Geld an die Multisigadresse um den Vertrag zu aktivieren. Sobald die geforderte Bedingung erfüllt ist, signieren die Orakel nach und nach die Transaktion und verbreiten sie. Wenn genügend Signaturen vorhanden sind, sendet Bob die Transaktion in das Netzwerk und das Geld wird freigegeben. Falls die Bedingung nicht erfüllt sein sollte, greifen die von Alice erstellten Regeln für die Rückerstattung und das Geld wird auf das von ihr angegebene Konto, transferiert. (Vgl. 33)

3 Vergleich Anbieter

In diesem Kapitel werden die einzelnen Anbieter, für das Einbringen von externen Daten in Smart Contracts, untersucht und miteinander verglichen. Der Vergleich wird in die Kategorien: Art der Implementierung, Datenquellen, Kosten und weitere Features aufgeteilt. Zum Abschluss wird zu jedem Anbieter ein Smart Contract geschrieben und dieser mit den gleichen Datenquellen aus den Orakeln, ausgeführt.

3.1 Anbieter

Folgende drei Anbieter wurden für den Vergleich ausgewählt: Oraclize.it (Vgl. 19), SmartContracts.com (Vgl. 40) und Realitykeys.com (Vgl. 41).

Die Auswahl erfolge anhand der Umsetzung, Funktionsweiße und verfügbaren Datenquellen. Um beim Vergleich und der beispielhaften Nutzung gemeinsame Anhaltspunkte zu erhalten, ist es wichtig, dass alle Anbieter mindestens eine Datenquelle zur Auswahl haben, die von allen angeboten wird. In diesem Fall handelt es sich um das Abrufen von Wechselkursen.

Oraclize ist ein Unternehmen das 2015 von Thomas Bertani gegründet wurde. (Vgl. 42) Sie binden einen kryptografischen Beweis in die Blockchain ein, womit überprüft werden kann, dass die übermittelten Daten nicht verändert wurden. (Vgl. 42) Dies ist durch die Implementierung von TLSNotary möglich. Oraclize hat seine komplette API sehr umfangreich dokumentiert, außerdem wird diese auch in regelmäßigen Abständen auf den neuesten Stand gebracht. (Vgl. 43)

Ein weiterer Anbieter ist SmartContracts, der 2014 von Sergey Nazarov und Steve Ellis gegründet wurde. (Vgl. 44) (Vgl. 45) Die kürzlich veröffentlichte Plattform unterstützt derzeit die Ethereum und Bitcoin Blockchain. (Vgl. 46)

Der letzte Anbieter in diesem Vergleich ist RealityKeys, welcher 2013 als Startup von Edmund Daniel Edgar gegründet worden ist. Im Sinne der Definition eines Orakels ist RealityKeys keiner, da dieser nur die API anbietet, welche als Datenquelle in Smart Contracts verwendet werden kann. Die Daten sind mit Hilfe von kryptographischen Funktionen verschlüsselt worden und bieten damit mehr Sicherheit, als solche aus ungeschützten Datenquellen. Wechselkurse und das Überwachen von Konten einer Kryptowährung sind ein Teil der möglichen Quellen. (Vgl. 47)

3.2 Art der Implementierung

Damit Daten von einem Orakel in einem Smart Contract genutzt werden können, müssen die Anbieter einen sogenannten Orakel-Contract erstellen, und die von der Quelle geladenen Daten dort hineinschreiben. Ein solcher Aufbau ist in Abbildung 2 zu sehen. Der Smart Contract greift dann wiederum auf diesen Orakel-Contract zu, und kann die Daten auslesen und selbst nutzen. Dieser Service wird von Oraclize und SmartContract angeboten. Wobei der Orakel-Contract bei Oraclize selbst geschrieben werden muss, wohingegen dieser bei SmartContract automatisch erstellt wird. Realitykeys stellt lediglich eine API zur Verfügung, mit deren Hilfe auf die eigentlichen Daten zugegriffen werden kann. Somit muss der Orakel-Contract selbst geschrieben werden.

Oraclize

Um einen Orakel-Contract mithilfe von Oraclize zu erstellen, muss dieser in Solidity oder Serpent selbst programmiert werden. Dazu finden sich in der Dokumentation Beispiele zu jeder Datenquelle und deren Verwendung im Orakel-Contract. Ein solcher Contract hat immer den gleichen Aufbau: Zuerst wird Oraclize's API von Github.com importiert um die vorgefertigten Funktionen zum Abruf nutzen zu können. Anschließend wird der Contract erstellt und ein Name vergeben. Weiterhin wird eine Variable angelegt, in der später das Ergebnis der Abfrage gespeichert wird und 2 Events definiert, eines für die Anfrage an Oraclize und das andere für das Aktualisieren der Variable. In der Main Funktion wird die Update Funktion aufgerufen, welche wiederum eine neue Abfrage bei Oraclize initialisiert und diese anschließend mit den angegeben Werten ausführt. Hier wird auch die Art der Abfrage und die Datenquelle angegeben. Für eine Abfrage von einer Website sieht das Ganze dann so aus:

```
function update() payable {
    newOraclizeQuery("Oraclize query was sent, standing by for the answer..");
    oraclize_query("URL", "xml(https://www.fueleconomy.gov/ws/rest/fuelprices).fuelPrices.diesel");
}
```

Als erstes Argument bekommt „oraclize_query“ die Art der Quelle und als zweites den Link zur Quelle. In diesem Fall eine Adresse, wo aktuelle Spritpreise zu finden sind. Dazu kann JSON oder XML als Datenformat genutzt werden.

Wenn die Abfrage erfolgreich war, wird von Oraclize die __callback Funktion aufgerufen und das Ergebnis übergeben. Darin wird die lokale Variable im Contract dann auf den neuen Wert aktualisiert.

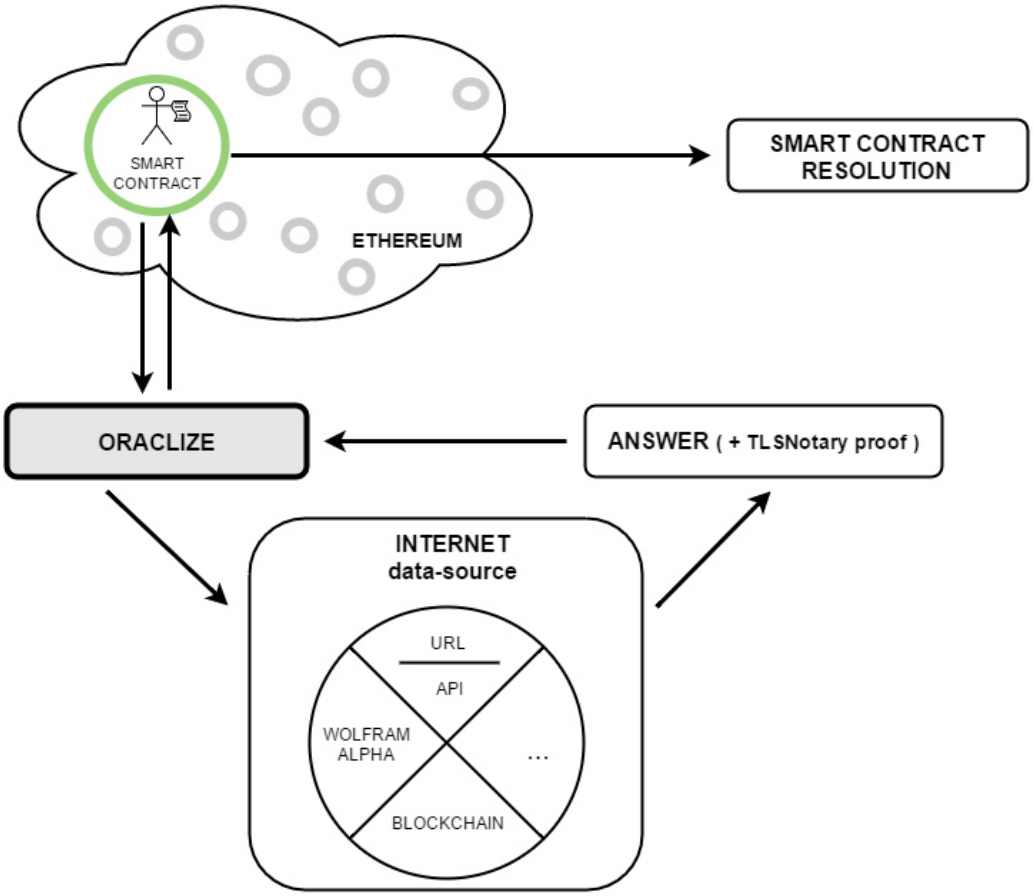

Abbildung 3: Service Oraclize (Vgl. 48)

SmartContracts

SmartContracts nutzt für das Erstellen eines Orakel-Contracts eine grafische Oberfläche. Dabei können die Contracts sehr einfach und schnell über eine Eingabemaske erstellt werden. Um die Datenquelle anzugeben, kann ein Link auf eine Website verwendet werden, welch die Daten im JSON Format ausgibt. Per Suchmaske kann die Auswahl noch verfeinert werden. Nach Angabe eines Titels, einer Beschreibung, dem Anhängen von Dokumenten und dem Signieren, wird der Contract erstellt und in die Blockchain geschrieben. (Vgl. 49)

Bei einem Smart Bond kann eine Bitcoin Adresse auf eine Zahlung in einer bestimmten Höhe überwacht werden und anschließend ein Betrag an eine andere Adresse ausgesendet werden. Falls die Zahlung nicht rechtzeitig ankommt, kann der Betrag an einen alternativen Empfänger gesendet werden. (Vgl. 50)

Der Service Level Agreement Contract bietet die Möglichkeit, eine Webseite nach ihrer Suchposition auf einer beliebigen Google Seite und definierten Suchbegriff zu überwachen. Falls die Position zu dem festgelegten Zeitpunkt in einem gewissen Bereich ist, kann wiederum eine Zahlung an einen Empfänger, und falls nicht, an einen anderen Empfänger gesendet werden. (Vgl. 50)

Mit folgender Grafik wird deutlich, welchen Service von SmartContract und Oraclize offeriert wird.

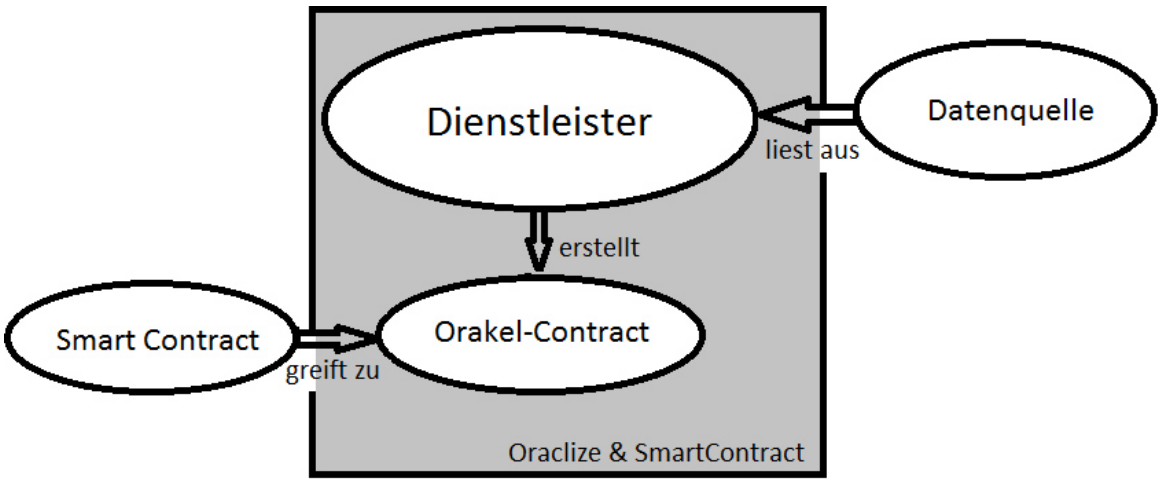

Abbildung 4: Service Smart Contracts

RealityKeys

Da Realitykeys keine Orakel-Contracts erstellt, werden hier andere Methoden benutzt, um die Abfragen auszuführen. Dabei wird mit POST und GET Anforderungen gearbeitet. Um eine Abfrage zu starten, müssen alle gewünschten Werte den dafür vorgesehenen Parametern zugeordnet werden. Das ganze Konstrukt wird anschließend hinter die, der Datenquelle zugehörigen Anfrage-URL, geschrieben und mithilfe eines passenden Programmes abgeschickt. So eine Anfrage sieht für eine Abfrage des Kontostandes einer Bitcoin Adresse folgendermaßen aus:

wget -qO- https://www.realitykeys.com/api/v1/blockchain/new --post-data="chain=XBT&address=1F1tAaz5x1HUXrCNLbtMDqcw6o5GNn4xqX&which_total=total_received&comparison=ge&value=1000&settlement_date=2015-09-23&objection_period_secs=604800&accept_terms_of_service=current&use_existing=1"

Wget ist in diesem Fall das Programm, das die POST Anforderung ausführt.

Wenn ein schon vorhandener Fakt abgefragt werden soll, kann dies mittels eines GET Requests ausgeführt werden. Dazu wird dieselbe Basis-URL verwendet wie schon beim POST Request, jedoch wird statt „/new“ und die weiteren Parameter, lediglich die ID des abzufragenden Faktes angegeben und welche Version der Geschäftsbedingungen akzeptiert werden. Die ID wird nach der Erstellung eines Faktes ausgegeben.

Beispielsweise so:

https://www.realitykeys.com/api/v1/blockchain/2?accept_terms_of_service=current

Alternativ können Fakten mit Wechselkursen, Blockchain Daten, Sportergebnissen, Runkeeper und Wikidata auch über eine einfache Weboberfläche erstellt werden. Dabei gilt es nur, die korrekten Werte und ein Datum für die Laufzeit auszuwählen. (Vgl. 41)

Wie man sieht, erstellt Realitykeys keinen Orakel-Contract, sondern fungiert nur als Vermittler zwischen Datenquelle und Contract. Dabei entstehen natürlich nicht dieselben Kosten wie bei Oraclize oder SmartContract.

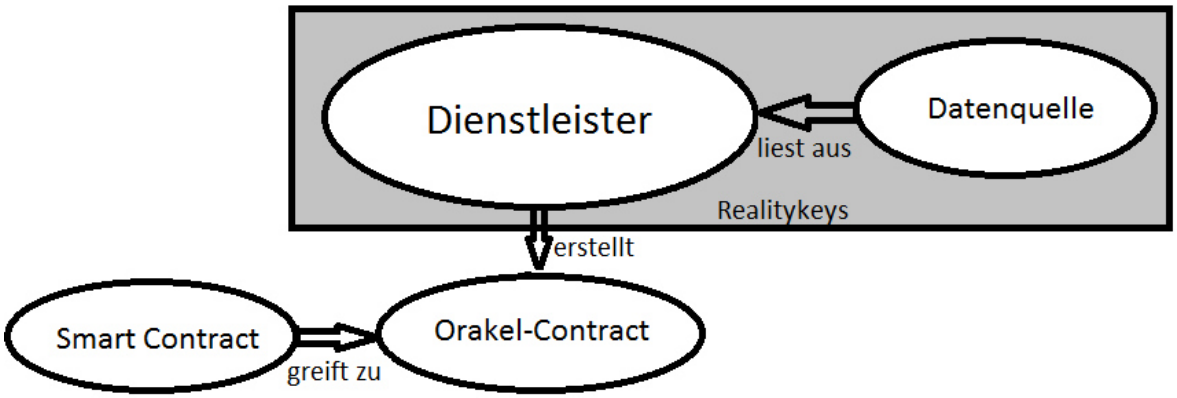

Abbildung 5: Service Realitykeys

Einschätzung: Oraclize und SmartContract bieten zwar denselben Service an, jedoch ist die Implementierung bei Oraclize am umfangreicher. Damit bieten sich hier auch die meisten Möglichkeiten zur Individualisierung, da der Orakel-Contract selbst geschrieben werden muss. Dies ermöglicht, einige Funktionen vom Smart Contract in den Orakel-Contract auszulagern, womit sich bei der Erstellung von Contracts Geld sparen lässt, da diese kleiner und einfacher gehalten werden können. Weiterhin ist es einfacher, einen Smart Contract für den Service zu schreiben, da der ganze Orakel-Contract einsehbar ist, und er leicht auf diesen angepasst werden kann. Um unerfahrene Nutzer nicht zu überfordern, gibt es eine gute Dokumentation und einige ausführliche Programmierbeispiele auf der Webseite.

Der Anbieter SmartContract nutzt eine Weboberfläche, um die Erstellung von Orakel-Contracts einfacher zu gestalten. Einerseits erhöht das den Komfort und sorgt für eine bessere Verwaltung von Contracts, andererseits werden aber auch der Funktionsumfang und die Personalisierung dieser etwas eingeschränkt. Um auf den Wert im Orakel-Contract Zugriff zu erhalten, wird die ABI in den Details auf der Webseite angezeigt.

Um neue Fakten zu erstellen, ist Realitykeys gut geeignet. Jedoch muss anschließend der Orakel-Contract manuell geschrieben und deployed werden. Der Service ist gut dokumentiert und dank der optional bereitstehenden Weboberfläche, auch einfach zu nutzen.

3.3 Verfügbare Datenquellen

Jeder Anbieter hat eine andere Priorität bei der Auswahl von Datenquellen für die Abfrage. Während eine URL und Blockchain als Quelle bei allen Anbietern möglich ist, gibt es RunKeeper und WikiData nur bei Realitykeys (Vgl. 31) und WolframAlpha beziehungsweiße IPFS nur bei Oraclize. (Vgl. 51) Im Folgenden werden nun die möglichen Datenquellen von jedem Anbieter gezeigt und kurz beschrieben:

3.3.1 Oraclize

Oraclize bietet derzeit sechs verschiedene Datenquellen an:

URL: Mit der URL Datenquelle kann auf jede öffentliche API oder Website zugegriffen werden. Dafür muss sichergestellt sein, dass die ausgewählte Seite die HTTP Get und Post Methoden unterstützt. Um diese Datenquelle zu nutzen, muss die URL an Oraclize übergeben werden. Optional können noch andere Parameter als Text übergeben werden. Die Antwort der Website wird dann von Oraclize weitergeleitet und wenn gefordert, der TLSNotary Beweis angehangen. (Vgl. 52)

Blockchain: Die Blockchain API bietet eine einfache Möglichkeit, Zugriff auf alle Blockchain basierten Daten zu erhalten. Die Datenquellen bieten dabei eine Art Abkürzung zu den meisten Block Explorer APIs. Mögliche Argumente für die Abfrage sind: (Vgl. 53)

- Bitcoin Blockchain Größe
- Litecoin Hashrate
- Bitcoin Schwierigkeit
- „1NPFRDJuEdyqEn2nmLNaWMfojNksFjbL4“ Kontostand

WolframAlpha: WolframAlpha als Datenquelle ermöglicht das Weiterleiten von Anfragen an die WA Engine und gibt die Antwort, falls vorhanden, als Text aus. Dabei kann es vorkommen, dass es gar keine Ausgabe von WA gibt, weil die Eingabe in die Suchmaschine keinen Sinn macht. Auch wird bei dieser Datenquelle kein TLSNotary Beweis geliefert, weil es gegen die Geschäftsbedingungen von Wolfram verstößt, die komplette API Antwort zurückzugeben. (Vgl. 32)

IPFS: Mit der Datenquelle IPFS kann auf Inhalte zugegriffen werden, welche das IPFS Protokoll verwenden. InterPlanetary File System ist ein dezentrales Speichernetz, welches Rechner mittels peer-to-peer verbindet. Jede Datei bekommt einen einzigartigen Hashwert und wird auf allen Rechnen gespeichert, die Interesse daran haben. Zusätzlich bekommt jede Datei noch Informationen darüber, wo sie zu finden ist.

Es gibt eine Versionshistorie, womit Änderungen nachverfolgt werden können, und keine doppelten Dateien. Wenn eine Datei aufgerufen werden soll, wird im Netzwerk ermittelt, welche Rechner diese gespeichert haben. (Vgl. 54)

Bei der Anfrage muss dabei der IPFS Multihash der aufzurufenden Datei angegeben werden. Eine solche Abfrage könnte beispielsweiße so aussehen:

oraclize_query("IPFS", " QmT78zSuBmuS4z925WZfrqQ1qHaJ56DQaTfyMUF7F8ff5o");

und gibt den Inhalt der Datei aus: „hello world\n“.

Falls Oraclize den Inhalt einer Datei nicht innerhalb von 20 Sekunden aufrufen kann, schlägt die Anfrage fehl. (Vgl. 55)

decrypt: Decrypt kann wie andere Quelle verwendet werden, wurde jedoch speziell für den Einsatz mit dem Feature „nested“ entwickelt, um eine teilweise Verschlüsselung der Abfrage zu ermöglichen. Dabei wird die komplette Abfrage mithilfe des öffentlichen Schlüssels von Oraclize verschlüsselt und so in der Blockchain gespeichert. Entschlüsselt werden kann sie nur von Oraclize selbst, mithilfe des privaten Schlüssels, wenn das „entrypted_queries_tools.py“ Python Script aufgerufen wird. Dieser Aufruf sieht in der CLI so aus:

python encrypted_queries_tools.py -e -p
044992e9473b7d90ca54d2886c7addd14a61109af202f1c95e218b0c99eb060c7134c4ae463
45d0383ac996185762f04997d6fd6c393c86e4325c469741e64eca9 "YOUR QUERY"

(Vgl. 56)

Computation: Computation erlaubt das Ausführen eines Programmes in der Blockchain. Dabei muss das auszuführende Programm vorher von einem Dockerfile beschrieben werden, welches nach dem Kompilieren und Starten das Programm aufruft. Das aufgerufene Programm muss ein Ergebnis in die Standartausgabe schreiben. Und dies so schnell wie möglich, denn es sind maximal 5 Minuten für den Aufruf des Dockerfiles und der Ausführung des Programms erlaubt. Anschließend wird das Programm in ein Archiv gepackt und in IFPS hochgeladen. Mit dem Link zum Archiv in IFPS kann danach das Programm aufgerufen werden, und das Ergebnis wird in den Contract geschrieben. (Vgl. 57)

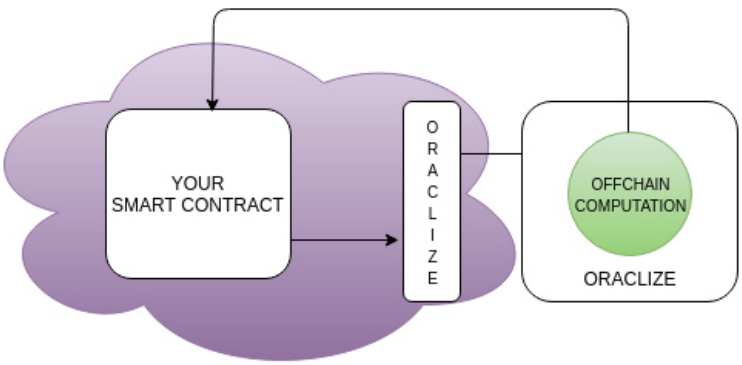

Abbildung 6: Computation Oraclize (Vgl. 57)

3.3.2 SmartContract

Es sind fünf verschiedene Kategorien verfügbar, in denen jede beliebige Quelle verwendet werden kann. Zur Auswahl steht Ethereum Orakel, Bitcoin Orakel, Service-Level-Agreement, GPS Überwachung und ein Smart Bond. (Vgl. 50)

Das Ethereum und Bitcoin Orakel unterscheiden sich nur in der Blockchain, in der die Verträge erstellt werden. Als Quelle für die externen Daten kann dabei jede URL angegeben werden, welche Daten im JSON Format ausgibt. Hierbei können eventuell weitere Kosten auftreten. Ein Orakel-Contract in Ethereum kann lediglich die Daten der angegebenen Quelle über den vorher bestimmten Zeitraum abrufen und speichern. Ein Orakel-Contract in der Bitcoin Blockchain kann zusätzlich eine Zahlung ausführen in Abhängigkeit davon, ob der abgerufene Wert gleich oder größer/kleiner als ein Vergleichswert ist, oder ein bestimmter Wert enthalten ist. (Vgl. 50)

Mit einem Smart Bond ist es möglich, Adressen zu überwachen und in Abhängigkeit von eingehenden Zahlungen, ausgehende zu tätigen.

Ein Service Level Agreement (SLA) kann die Suchposition einer Webseite überwachen und je nach Position an einem bestimmten Datum, Zahlungen ausführen.

Für die GPS Überwachung muss man eine Anfrage per Mail an SmartContract senden und den weiteren Verlauf persönlich klären. (Vgl. 50)

3.3.3 Realitykeys

Auch Realitykeys hat derzeit, genau wie Oraclize, 6 Quellen zur Auswahl, wovon eine veraltet ist. Die Ergebnisse können auf verschiedene Arten ausgeben werden: Wahrheitswerte, Datenwerte oder nur Fakten. Das System funktioniert per GET und POST anfragen als auch über eine Weboberfläche.

Wechselkurse: Wechselkurse können von normalen Währungen, Kryptowährungen und aus einer Kombination von beiden überwacht werden.

Dazu muss eine POST Anfrage an „https://www.realitykeys.com/api/v1/exchange/new“ gesendet werden, mit folgenden möglichen Parametern:

fromcur:	die Währung, die als Basis dient	XBT
comparison:	größer oder kleiner als mit ge oder lt	ge
tocur:	die Währung, in die gewandelt wird	USD
value:	der Vergleichswert	1000
settlement_date	Datum, an dem der Wert geprüft wird	2017-05-07
objection_prediod_secs	Sekunden, auf die bis zur Prüfung gewartet wird	604800
accept_terms_of_service	Version der ToS, welche akzeptiert wird	current
use_existing	vorhanden Key nutzen, falls er gleiche Parameter nutzt	1

Bei allen Quellen wird, falls die Anfrage gelingt, ein JSON Objekt zurückgegeben indem alle Daten enthalten sind, die zur Erstellung verwendet wurden. Falls ein vorhandener Fakt abgefragt und gefunden wurde, wird die ID ausgegeben.

Blockchain: Mit der Blockchain Quelle kann der aktuelle Wert, der in einer Bitcoin Adresse gespeicherten Bitcoins, abgefragt werden. Dazu wird der aktuelle Wechselkurs von der Seite „blockr.io“ genutzt. Um eine Bitcoin Adresse abzufragen muss eine POST Anfrage an https://www.realitykeys.com/api/v1/blockchain/new gesendet und folgende Parameter genutzt werden:

chain	die Abkürzung der zu nutzenden Blockchain	XBT
address	die Adresse des Accounts	BTC Adresse
which_total	Parameter, welcher Wert abgefragt werden soll	total_received
comparison	größer oder kleiner als (ge oder lt)	ge
value	Vergleichswert	1000
settlement_date	Datum, an dem der Wert geprüft wird	2017-05-07
objection_prediod_secs	Sekunden, auf die bis zur Prüfung gewartet wird	604800
accept_terms_of_service	Version der ToS, welche akzeptiert wird	current
use_existing	vorhanden Key nutzen, falls er gleiche Parameter nutzt	1

Fußball Ergebnisse: Fußballergebnisse können mithilfe der API von football-api.com geprüft werden. Obwohl Realitykeys kein Geld für eine Abfrage verlangt, muss der Service von football-api.com bezahlt werden.

Hier lautet die Adresse für den POST https://www.realitykeys.com/api/v1/soccer/new und folgende Parameter sind zulässig:

football_api_match_id	Die Match ID von football-api.com	1940587
football_api_team1_id	Die ID von Team1 (Heimteam)	9002
football_api_team2_id	Die ID von Team2 (Auswärtsteam)	9406
comparison	Der Stand von Team1 zu Team2, mit gt, lt, ge, le, eq für Sieg, Niederlage, Sieg oder Unentschieden, Niederlage oder Unentschieden oder Unentschieden	gt
settlement_date	Datum, an dem der Wert geprüft wird	2017-05-07
objection_prediod_secs	Sekunden, auf die bis zur Prüfung gewartet wird	604800
accept_terms_of_service	Version der ToS, welche akzeptiert wird	current
use_existing	vorhanden Key nutzen, falls er gleiche Parameter nutzt	1

RunKeeper: Mit dieser Abfrage können die eigenen Fortschritte von RunKeeper überwacht werden. Dies ist eine Plattform, mit der man seinen Fortschritt bei vielen sportlichen Aktivitäten verflogen, eigene Ziele setzen und Erfolge teilen kann. Mithilfe von Realitykeys lassen sich Ziele für gewisse Aktivitäten setzen und diese zu einem festgelegten Zeitpunkt kontrollieren und auswerten. Die URL lautet https://www.realitykeys.com/api/v1/runkeeper/new und die Parameter für die Abfrage sind die folgenden:

user_id	die Runkeeper ID, von welcher Aktivitäten verfolgten werden sollen	29908850
activity	die zu verfolgende Aktivität	walking
measurement	zu messender Wert, total_distance oder distance_cumulative	total_distance
comparison	ge oder lt für größer oder kleiner gleich	ge
goal	der Zielwert	2000
settlement_date	Datum, an dem der Wert geprüft wird	2017-05-07
objection_prediod_secs	Sekunden, auf die bis zur Prüfung gewartet wird	604800
accept_terms_of_service	Version der ToS, welche akzeptiert wird	current
use_existing	vorhanden Key nutzen, falls er gleiche Parameter nutzt	1

WikiData: Hiermit können Daten aus der Datenbank von WikiData abgefragt werden. Dies ist eine Plattform ähnlich der von Wikipedia, wovon einzelne Fakten von Suchbegriffen abgefragt und mit anderen verglichen werden können.

Die Abfrage muss an die URL https://www.realitykeys.com/api/v1/wikidata/new gesendet werden:

note1_id	ID des ersten Objekts	Q76
not1_attribute_id	ID des 1. Objekts mit Relation zum 2. Objekt	P21
note2_id	ID des zweiten Objekts	Q6581097
settlement_date	Datum, an dem der Wert geprüft wird	2017-05-07
objection_prediod_secs	Sekunden, auf die bis zur Prüfung gewartet wird	604800
accept_terms_of_service	Version der ToS, welche akzeptiert wird	current
use_existing	vorhanden Key nutzen, falls er gleiche Parameter nutzt	1

Human-verified Data: Eine Abfrage von Fakten, die im Vorfeld persönlich mit Realitykeys abgesprochen wurden. Dabei wird der HashWert in der Abfrage angegeben. Die URL dafür lautet: https://www.realitykeys.com/api/v1/human/new und folgende Parameter sind möglich:

proposition_hash	Der HashWert des Fakts 432fd3605924bc527396c920f0664b939b7cdd3a96167eee1d72c04eea0606b6	
settlement_date	Datum, an dem der Wert geprüft wird	2017-05-07
objection_prediod_secs	Sekunden, auf die bis zur Prüfung gewartet wird	604800
accept_terms_of_service	Version der ToS, welche akzeptiert wird	current
use_existing	vorhanden Key nutzen, falls er gleiche Parameter nutzt	1

Einschätzung: Es wird eine ausreichende Anzahl von verfügbaren Datenquellen bei allen Anbietern geboten. Während Realitykeys und Oraclize für die gängigsten eine extra API mit festgelegten Parametern anbieten, geht SmartContract einen komplett anderen Weg und separiert die Services nach ihrem Zweck und ohne feste Parameter. Dabei kann jede beliebige Webseite als Quelle für die Daten verwendet werden, solange sie eine Ausgabe im JSON oder XML Format bietet. Das Verwenden der Datenquellen ist bei allen Anbietern sehr einfach, da entweder Vorgaben über die zu nutzenden Werte vorhanden sind, oder diese, wie bei SmartContract, komplett frei gewählt werden können.

3.4 Kosten für Nutzung

Oraclize

Bei Oraclize ist die erste, von einer Adresse gesendete Abfrage, sowie die erste Aktualisierung des Wertes gratis. Dies bedeutet, dass das Erstellen eines Contracts und damit auch das erstmalige Abrufen des Wertes von der angegebenen Quelle komplett gratis ist. Dabei übernimmt Oraclize auch die Kosten für den Callback, also das Schreiben des Wertes in den Contract. Diese werden bis zum standartmäßigen GasLimit von 200k Gas übernommen. Weiterhin werden alle Preise anhand der Datenquelle und eventuellen Notarisierung mit TLSNotary berechnet. Der Standartpreis beträgt 0.01$ und 0.04$ für TLSNotary, falls verfügbar. Für das Testnet werden keinerlei Gebühren verlangt. (Vgl. 58)

SmartContract

Bei SmartContract werden die Preise je nach Anfrage berechnet, und können für geplante Abfragen im Voraus bezahlt werden oder nach Aufforderung, für On-Demand Abfragen. Auch sind die Preise abhängig von der Ausführungsdauer der Contracts. Ein sehr einfacher Contract der den Wechselkurs von Euro zu US Dollar speichert und eine Laufzeit von einem Monat hat, kostet beispielsweiße 0.19$.

Realitykeys

Bei Realitykeys lässt sich ein Fakt registrieren und automatisiert kontrollieren, ohne dass dabei Kosten entstehen. Dies ist möglich, weil der Anbieter keine Orakel-Contracts, zum Speichern der Daten erstellt. Falls einem die automatische Überprüfung nicht sicher genug ist, lässt sich gegen eine Zahlung von 10$, ein Check durch einen Mitarbeiter durchführen. Diese ist per Bitcoin bezahlbar.

Im Folgenden eine Übersicht über die genauen Preise der Anbieter und eine Beispielrechnung für 1 und 1000 Wechselkurs Abfragen mit einer Laufzeit von einem Tag und ohne weitere Features:

Oraclize.it			
		Beweis	
Datenquelle	Basis Preis	ohne	TLS
URL	0.01$	0$	0.04$
Blockchain	0.01$	0$	0.04$
WolframAlpha	0.03$	0$	nicht verfügbar
IFPS	0.01$	0$	nicht verfügbar
Realitykeys.com			
Datenquelle	Basis Preis	menschliche Kontrolle	
alle	0$	10$	
SmartContract.com			
Datenquelle	Basis Preis		
alle	abhängig von Quelle und Laufzeit		

Tabelle 2: Preisübersicht Anbieter (Vgl. 58) (Vgl. 59)

1 Abfrage

Oraclize.it	0.010$
SmartContract.com	0.025$

Tabelle 3: 1 Abfrage Oraclize & SmartContract

1000 Abfragen

Oraclize.it	10$
SmartContract.com	25$

Tabelle 4: 1000 Abfragen Oraclize & SmartContract

Einschätzung: Einen Orakel-Contract zu erstellen, kostet bei Oraclize und SmartContract ähnlich viel. Für einige wenige Contracts sollten daher die Kosten bei der Wahl eines Anbieters, keine Rolle spielen. Erst bei dem Wunsch mehrere, eventuell hunderte oder tausende Contracts zu erstellen, macht ein Blick auf die zu erwartenden Kosten Sinn, denn dann wird ein Unterschied deutlich. Da Realitykeys keinen Orakel-Contract erstellt, fallen hier auch keine weiteren Kosten an. Dieser muss vom Nutzer geschrieben werden und kann je nach Komplexität auch ähnliche Kosten verursachen. Letztendlich muss allerdings abwägt werden, ob man mehr zahlt um eventuelle Features wie TLSNotary oder einer besseren Implementierung zu nutzen oder darauf verzichten kann.

3.5 Weitere Features

Zusätzlich zu den verschiedenen Datenquellen, bieten die Orakel noch weitere Features an. So zum Beispiel eine höhere Sicherheit durch das Nutzen von TLSNotary (siehe 2.3.1) und IFPS als Onlinespeicher für diesen, der Verschlüsselung von Abfragen und mehr Komfort durch die Möglichkeit, Dokumente bei der Erstellung des Vertrages anzuhängen oder diesen von fremden Personen signieren zu lassen.

TLSNotary wird aktuell nur von Oraclize unterstützt. Zusätzlich kann IPFS als Speicher für den TLS Beweis verwendet werden. Etwa, wenn der Preis für das Liefern der Daten sehr hoch ist, z.B. bei Ethereum Transaktionen. Dazu wird im Orakel-Contract einfach die Zeile

oraclize_setProof(proofType_TLSNotary | proofStorage_IPFS)

hinzugefügt und damit der Beweis mithilfe von TLS geführt und per IPFS gespeichert. (Vgl. 60)

Weiterhin bietet Oraclize den Android Beweis, bei dem die HTTPS Verbindung von einem Android Gerät hergestellt wird und mithilfe eines SafetyNet Software Nachweises der komplette Ablauf abgesichert wird. (Vgl. 61)

Nested ist eine sekundäre Datenquelle die von Oraclize angeboten wird. Sie ermöglicht das Verknüpfen von Abfragen von verschiedenen Seiten und Daten. So können zwei oder mehrere Abfragen verschachtelt werden und als ein Ergebnis ausgegeben werden. Die API dafür sieht vor, dass eine verknüpfte Abfrage mit einer speziellen Syntax verwendet wird. Dazu wird die Datenquelle in „[ ]“ geschrieben während danach die Abfrage folgt. Die zweite Quelle wird dann in die Angabe „${„ und „}“ geschrieben. Ein Beispiel sieht folgendermaßen aus: (Vgl. 62)

[WolframAlpha] temperature in ${[IPFS]
QmP2ZkdsJG7LTw7jBbizTTgY1ZBeen64PqMgCAWz2koJBL}

SmartContract bietet eine ganz andere Art von Features. Zum Beispiel können Zahlungen außerhalb eines Contracts von dessen Bedingungen abhängig gemacht werden oder eine Verbindung zu traditioneller Infrastruktur hergestellt werden. (Vgl. 63) Auch ist es möglich, juristische Dokumente in einen Contract zu integrieren und diese signieren zu lassen, so dass sie eine rechtskräftige elektronische Unterschrift haben. (Vgl. 63)

Encryption, also die Verschlüsselung der Anfrage, ist bei Realitykeys und ebenso bei Oraclize möglich. Bei letzterem wird dessen öffentlicher Schlüssel verwendet, um die Anfrage zu verschlüsseln und der private um sie wieder lesbar zu machen. Dazu wird die gewünschte Datenquelle mithilfe eines bereitgestellten Phyton Scripts verschlüsselt und anschließend der Parameter in der eigentlichen Anfrage angegeben. (Vgl. 64) Bei Realitykeys gibt es dafür ein von User geschriebenes Node.js Programm (Vgl. 65), welches das „Elliptic Curve Integrated Encryption Scheme“, kurz ECIES verwendet. (Vgl. 66)

Zu beachten ist noch, dass die Contracts SmartContract und Fakten bei Realitykeys jeweils eine feste Laufzeit haben, die im Vorfeld angegeben werden muss. Contracts die mit Oraclize erstellt wurden, können so lange genutzt werden, bis sie manuell mittels einer „kill()“ Funktion gelöscht werden.

Einschätzung: Jeder Anbieter hat verschiedene Features in seinen Service integriert. Dabei bietet Oraclize die größte Auswahl und abgesehen von SmartContract, auch die einfachste Implementierung. Da der Contract sowieso komplett selbst programmiert werden muss, lassen somit schnell zusätzliche Features nutzen. Realitykeys bietet nur einen sehr begrenzten Umfang, jedoch das wichtige Verschlüsselungsfeature. Bei SmartContracts können die weiteren Funktionen sehr einfach über die Oberfläche genutzt werden. Hier können Dateien per Drag und Drop einfach hinzugefügt und weitere Personen zum Signieren eines Vertrages angegeben werden.

3.6 Beispielhafte Nutzung

Abschließend wird hier nun gezeigt, wie ein Orakel-Contract bei den Anbietern zu erstellen ist und wie diese zu nutzen sind. Als Beispiel wird dabei der Wechselkurs von 1 Ether in Euro verwendet, der bei SmartContract und Oraclize von Cryptocompare.com bezogen wird und bei Realitykeys von Poloniex.com stammt.

Oraclize

Der Contract wurde in Solidity geschrieben wird im Folgenden genauer beschrieben.

In den ersten beiden Zeilen wird die Compilerversion auf 0.4.0 begrenzt und die Oraclize API aus der angegeben Quelle importiert. Anschließend beginnt der Contract mit dem Namen ExchangeETHEUR: es werden 2 Events deklariert und 1 öffentliche Variable, welche später den Wechselkurs beinhaltet. Die Funktion ExchangeETHEUR() wird beim Erstellen des Contracts angewählt und ruft die Updatefunktion auf, welche dann eine neue Abfrage bei Oraclize erstellt. Hier werden die Art der Abfrage und die Datenquelle angegeben, von der Oraclize den Wert abruft. Die __callback Funktion wird von Oraclize verwendet, um den neuen Wert in den Contract zu schreiben. Dabei wird erst geprüft, ob der Aufrufer der Updatefunktion auch der Ersteller und damit Eigentümer des Contracts ist, und falls nicht, bricht die Aktion ab. Anschließend wird das Event newETHEUR ausgelöst und der neue Wert in das Protokoll des Contracts geschrieben sowie der Wert in Eurocent umgewandelt und abgespeichert.

```
pragma solidity ^0.4.0;

import "github.com/oraclize/ethereum-api/oraclizeAPI.sol";


contract ExchangeETHEUR is usingOraclize {

        uint public ETHEUR;      // Variable, in welcher der Tauschkurs gespeichert wird

        event newOraclizeQuery(string description);

        event newETHEUR(string rate);


        function ExchangeETHEUR() {

                update(); // Beim Erstellen des Vertrages aufrufen

        }

        function __callback(bytes32 myid, string result) {

                if (msg.sender != oraclize_cbAddress()) throw;  // Prüfen, ob Aufrufer auch Ersteller
                ist

                newETHEUR(result); // Aufruf des Events und Kurs übergeben, um ihn ins Protokoll zu
                schreiben



                ETHEUR = parseInt(result, 2); // Umwandeln von € in Eurocent

        }

        function update() payable {

                newOraclizeQuery("Anfrage wurde gesendet, warte auf Antwort..");

                oraclize_query("URL", "json(https://min-
                api.cryptocompare.com/data/price?fsym=ETH&tsyms=EUR).EUR");

          }

}
```

Nach dem deployen in die Ethereum Blockchain, wird einmalig die Updatefunktion aufgerufen und der Wert zum ersten Mal in den Contract geschrieben. Anschließend kann die Funktion wiederholt manuell aufgerufen werden. Der Orakel-Contract hat keine festgelegte Laufzeit und ist somit unbegrenzt gültig. Im Anhang Teil 1 befindet sich ein Smart Contract, der auf den hier gezeigten Orakel-Contract zugreift und den Wert selbst speichert.

SmartContract

Dieser Anbieter hat die Besonderheit, dass man sich erst einen Account erstellen muss, um den Service nutzen zu können. Anschließend kann ein Contract erstellt werden und man hat die folgenden 5 Optionen zur Auswahl:

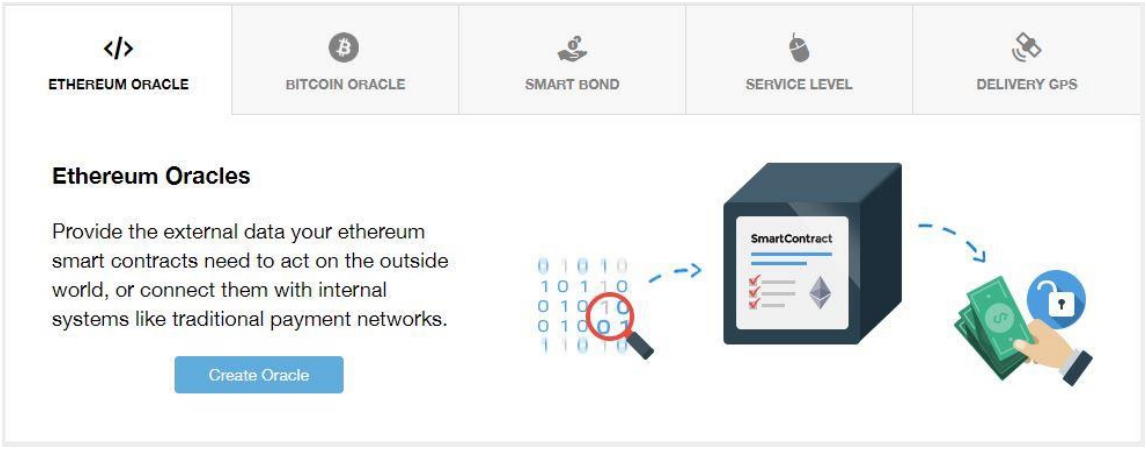

Abbildung 7: Übersicht Quellen SmartContract (Vgl. 50)

Mit „Create Oracle“ wird ein Ethereum Contract erstellt und es können Datenquelle und Laufzeit angegeben werden. Nachdem ein Titel und eine kurze Beschreibung vergeben sowie Dokumente angehangen wurden, kann der Contract erstellt werden. Nun muss er von den angegebenen Personen signiert werden, was im Normalfall nur die eigene Person ist.

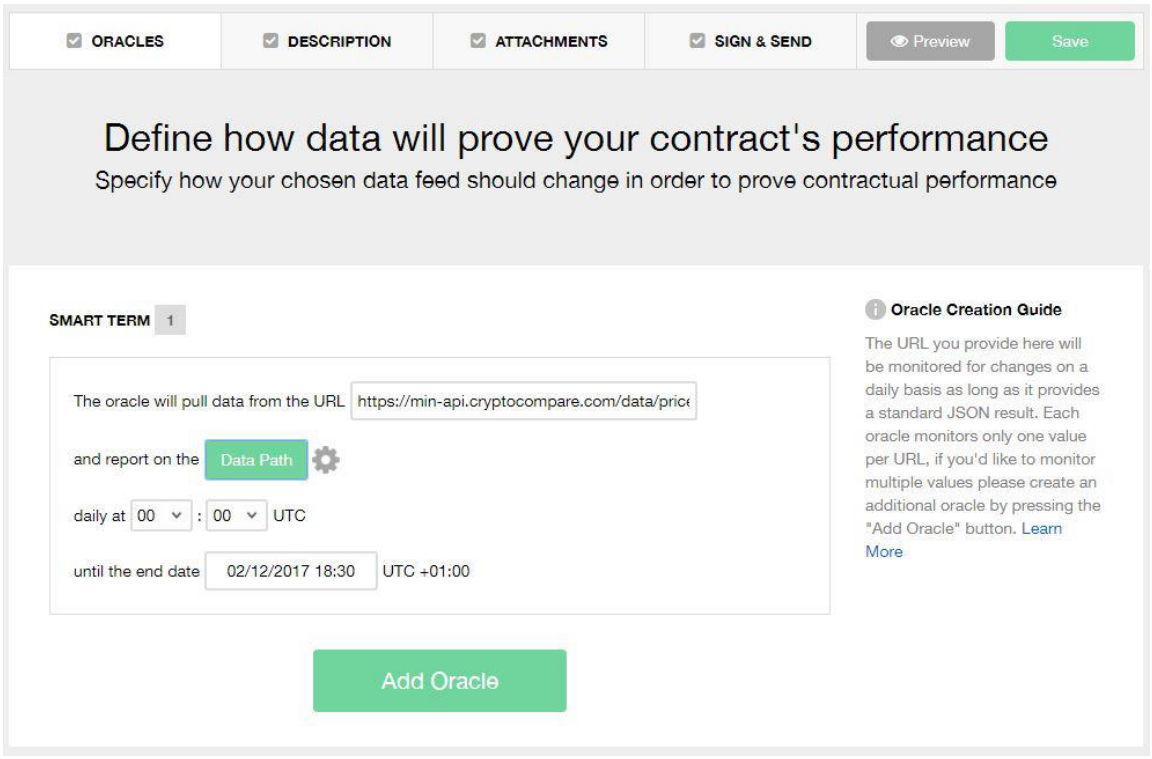

Abbildung 8: Aufbau Erstellungsprozess bei SmartContract (Vgl. 67)

Anschließend kann auf der rechten Seite die Laufzeit abgelesen und unter „Oracle Activity“ alle Aktivitäten des Vertrages eingesehen werde. In der Abbildung 11 gezeigten Übersicht lassen sich noch einmal alle Details ansehen. So zum Beispiel die verwendete Datenquelle und der Wert, der überwacht wird. Ebenso kann man sich hier die ABI des Contracts anzeigen lassen, welche für den Zugriff aus einem Smart Contract nötig ist. Ein solcher Contract ist im Anhang Teil 2 zu finden.

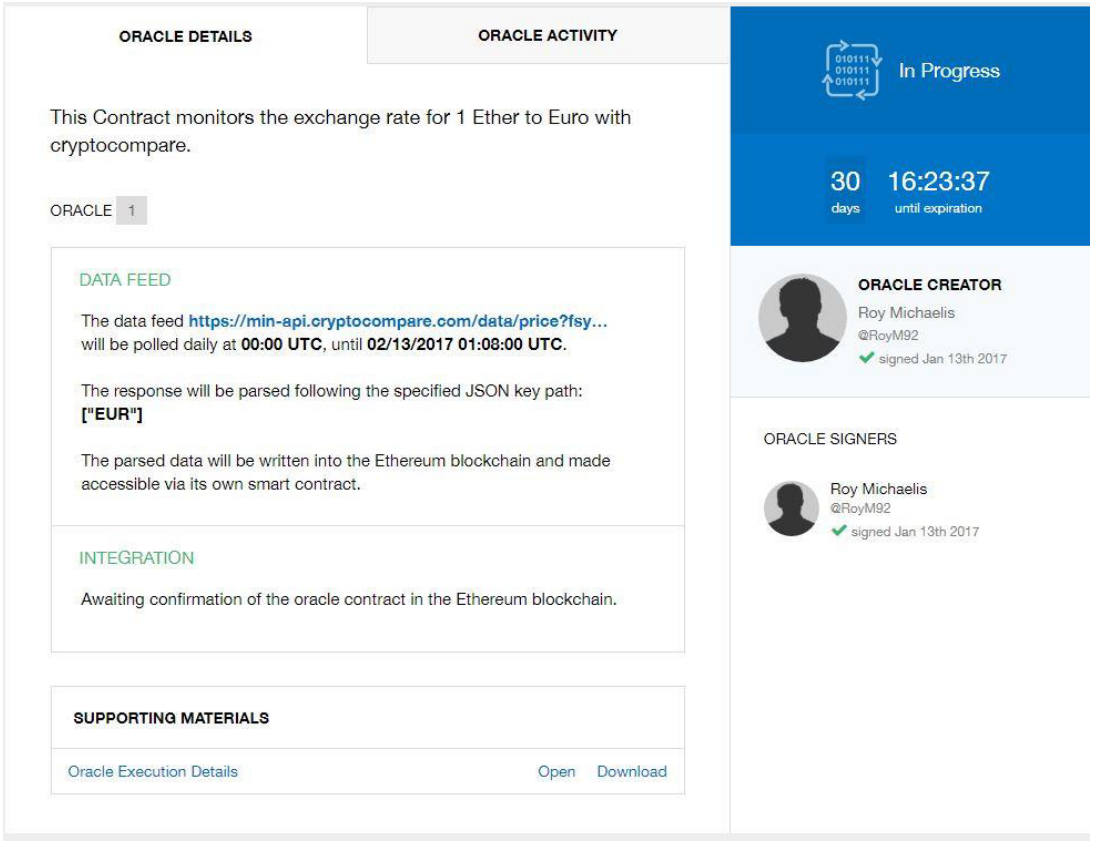

Abbildung 9: Übersicht Exchange Contract Smart Contract (Vgl. 68)

Realitykeys

Wie oben schon im Teil Implementierung geschrieben, können bei Realitykeys die Fakten per POST Befehl oder auch über die Weboberfläche erstellt werden. Im Folgenden wird zeigt, wie die Erstellung über die Webseite abläuft und der dazu passende POST Befehl aussieht.

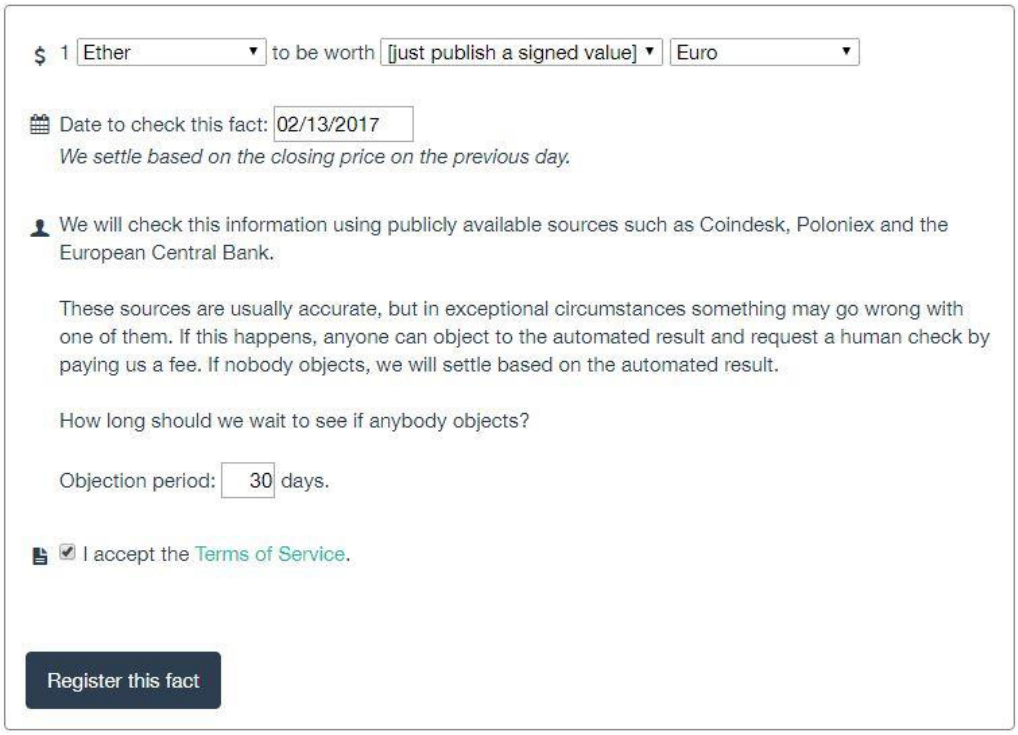

Abbildung 10: Erstellung Orakel Realitykeys (Vgl. 69)

Über die Startseite und den Knopf „Follow an exchange rate“ gelangt man zu der oben zu sehenden Eingabemaske. Dort wird die Basis- und Tauschwährung ausgewählt und wie diese zueinanderstehen. Im Beispiel soll nur der signierte Wert ausgegeben werden, ohne weitere Vergleiche anzustellen. Weiterhin werden hier das Datum, zu dem der Fakt auf seine Bedingung geprüft werden soll, und die Laufzeit angegeben. Hier wird eine Zeitspanne von 30 Tagen zu Grunde gelegt und der Fakt kann registriert werden.

Abbildung 11: Übersicht Orakel Nr.12168 Realitykeys (Vgl. 70)

Es können nun alle für den Contract relevanten Daten eingesehen werden wie zum Beispiel der Hashwert des Faktes, der Schlüssel mit dem Realitykeys den Fakt nach dem Eintreten der Bedingung signiert, und die dazugehörige Ethereum Adresse. Auch kann hier der aktuelle Wechselkurs auf der rechten Seite gesehen werden und bis wann der Fakt gültig ist. Falls eine Kontrolle des Contracts durch einen Mitarbeiter gewünscht wird, sind dort Informationen zu finden, bis wann und wohin die zuständige Zahlung erfolgen muss.

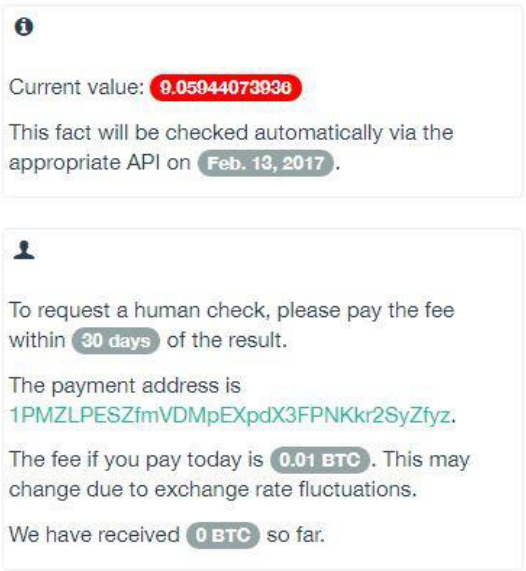

Abbildung 12: Übersicht2 Orakel Nr.12168 Realitykeys (Vgl. 70)

Alternativ lässt sich der Contract aber natürlich auch per POST Befehl erstellen und muss im Beispiel so aussehen:

https://www.realitykeys.com/api/v1/exchange/new --post-data="fromcur=ETH&tocur=EUR&settlement_date=2017-02-13&objection_period_secs=2592000&accept_terms_of_service=current&use_existing=1"

3.7 Zusammenfassung und Einschätzung

Der größte Unterschied zwischen den Anbietern ist bei den Datenquellen und der Implementierung zu finden.

Oraclize geht hier den Weg mit komplett selbst programmierbaren Contracts mittels Solidity, Serpent oder LLL. Da Oraclize auch einige zusätzliche Features wie TLSNotary Beweise und die Speicherung dieser mit Hilfe von IPFS, sowie die Verschlüsselung von Abfragen und einen Android Proof bietet, hat man hier auch die meisten Möglichkeiten zur Individualisierung und Anpassung an die persönlichen Wünsche. Die Kosten sind eindeutig für jede Quelle und Funktion gegliedert in der umfangreichen Dokumentation zu finden und für den Angebotenen Service auch sehr günstig. Somit ist ein gutes Preis-Leistungs-Verhältnis gegeben. Der Anbieter eignet sich daher für Kunden, die sehr spezielle Anforderungen an einen Contract haben und diesen komplett selbst gestalten wollen. Da die Preise genau aufgeschlüsselt sind, lassen sich die Kosten gut im Voraus kalkulieren.

SmartContract hat keine feste Preisliste. Die Kosten für einen Contract richten sich nach Datenquelle und der Laufzeit. Diese muss auch wie bei Realitykeys, bei der Erstellung des Vertrages angegeben werden. Die Kosten sind allerdings etwas höher als bei den anderen beiden Anbietern, wie im Kapitel 3.4 zu sehen ist. Erstellt werden die Contracts über eine sehr einfach und übersichtlich gehaltene Weboberfläche und können dort zusammengestellt werden. Als Features werden die Möglichkeiten, geschäftliche Dokumente anzughängen und den Vertrag von mehreren Personen unterschreiben zu lassen, angeboten. Die gebotenen Datenquellen und Services sollten besonders für geschäftliche Kunden interessant sein.

Bei Realitykeys werden keine Orakel-Contracts erstellt, sondern nur eine API als Datenquelle bereitgestellt. Dadurch erhält man Zugang zu Daten die andernfalls, aufgrund des Fehlens des JSON Formates, nicht nutzbar wären. Erwähnenswert sind hier besonders football-api.com und RunKeeper, worauf man ohne Realitykeys keinen Zugriff bekommen kann. Diese sogenannten Realitykeys lassen sich entweder per Weboberfläche erstellen, oder mittels eines POST Befehls an die Website, indem alle relevanten Informationen enthalten sind. Auch hier werden die gängigsten Datenquellen wie Bitcoin und Wechselkurse angeboten. Weiterhin gibt es noch einige spezielle, wie die oben erwähnten, und Wikidata.

Ähnlich wie SmartContract bietet Realitykeys auch die Möglichkeit, Quellen egal welcher Art zu verwenden, wenn dies vorher persönlich abgesprochen wurde. Das Erstellen von Fakten ist durch die Oberfläche und der Eingabemaske wesentlich einfacher, als es bei Oraclize der Fall ist. Abgesehen von der Verschlüsselung der Abfragen, werden hier allerdings keine weiteren Features angeboten. Da das Bereitstellen von Daten durch die eigene API keine Kosten, wie Transaktionsgebühren beim Erstellen von Contracts verursacht, ist der Preis nicht direkt vergleichbar mit den anderen Anbietern. Falls dem Kunden die automatische Abfrage von Datensätzen nicht genügt, da es sich eventuell um sensible Daten handelt oder er dem Computer misstraut, lässt sich für 10$ eine Prüfung durch einen Mitarbeiter erwerben. Somit ist der Anbieter für Nutzer interessant, welche etwa auf spezielle Daten zugreifen möchten, oder mehr Sicherheit bei der Abfrage von Datenquellen wünschen.

4 Programmierbeispiel – lokales Orakel

4.1 Aufgabe

Um einen Wert in einem Smart Contract täglich aktuell zu halten, muss man eine vertrauenswürdige Website mit den aktuellen Wechselkursen aufrufen, den passenden Kurs finden, die Funktion manuell aufrufen und den Wert eintragen. Der Aufwand mag für einen Vertrag, der einmal täglich aktualisiert werden soll, noch überschaubar sein, wird jedoch für mehrere Verträge mit unterschiedlichen Werten und Datenquellen sehr schnell zur Herausforderung. Diesen Service bieten Orakel an und in diesem Beispiel soll gezeigt werden, wie ein solcher Dienst durch ein lokales Programm abgebildet werden kann. Dafür wird ein Orakel-Contract angelegt, der vom Programm den aktuellen Wechselkurs von 1 Ether in Euro übergeben bekommt. Ein Smart Contract soll als Beispiel dienen, wie z.B. ein Kunde auf den Orakel-Contract zugreifen und die Variable auslesen kann.

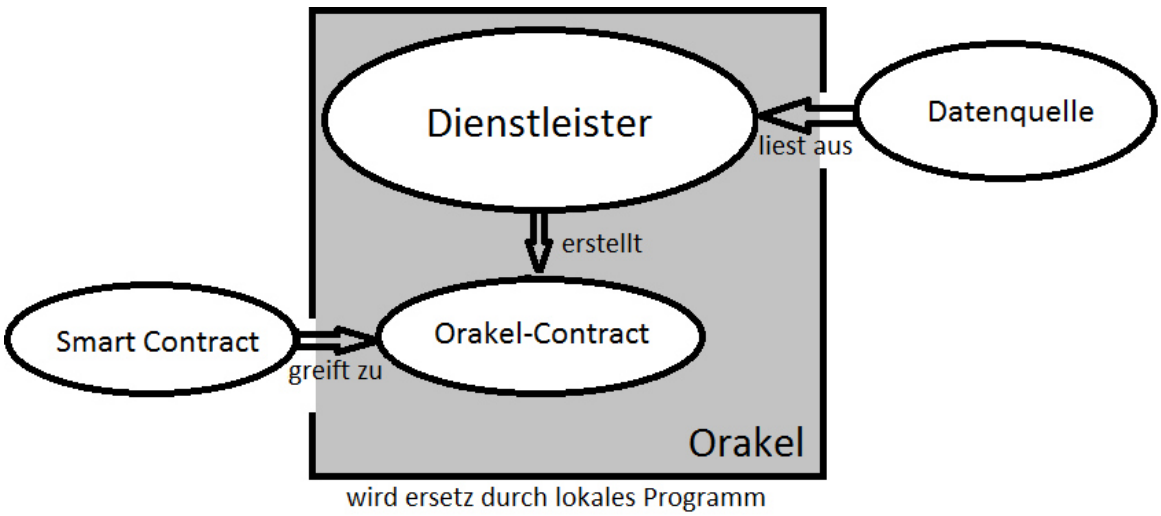

Abbildung 13: Aufgabe lokales Programm

4.2 Anforderung

Um dem lokalen Dienst Zugriff auf das Ethereum Netzwerk zu gewähren, wird ein Ethereum-Client gebraucht. Der Dienst muss die Webseite aufrufen, die Daten herunterladen und entsprechend dem Datenformat im Contract, parsen. Um den Wert in den Contract zu schreiben, muss das Programm die Update Funktion aufrufen und den Wert übergeben. Da für das Aufrufen der Funktion der Account freigegeben werden muss, braucht es eine Möglichkeit, das Passwort verschlüsselt einzugeben.

Um den Aufruf der Daten/Funktion und das Aktualisieren des Wertes überwachen zu können, soll eine Ausgabe relevanter Daten in der Konsole erfolgen. Weiterhin müssen zwei Contracts geschrieben werden. Ein Orakel-Contract, der vom Dienst aufgerufen wird und die Variable speichert, und ein Smart Contract, der den Wert aus dem Orakel-Contract lädt und ihn verwendet.

4.3 Folgen

Als Ethereum-Client wurde Parity ausgewählt. Dieser stellt die Testnet Blockchain von Ethereum lokal bereit und bietet gleichzeitig eine Wallet per Weboberfläche an. (Vgl. 71)

Parity kann von der Website „https://ethcore.io/parity.html“ geladen werden und steht für Ubuntu, MacOSX und Windows zur Verfügung. Die aktuellste Version führt einige Verbesserungen ein. Darunter Warp-Sync, eine verbesserte Synchronisation der Blockchain, ein schnelleres Importieren der Blöcke und ein verbesserter Speicherabdruck. Auch das Webinterface wurde erheblich verbessert und die Hard-forks EIP-155 (replay-attack prevention), 160 (EXP cost fixes) und 161 (empty account cleanup) werden unterstützt. (Vgl. 72)

Parity kann per Konsole oder über das integrierte Webinterface gesteuert werden. Dieses findet man standartmäßig über die Adresse „http://127.0.0.1:8180“.

Dort gelangt man zuerst auf die Account Ansicht, in der man die Möglichkeit hat, seine vorhandenen Accounts hinzuzufügen oder neue anzulegen. Im Menüpunkt „Settings“ kann man einstellen welche Punkte im Menü angezeigt werden sollen.

Unter „Contracts“ werden alle Smart Contracts angezeigt die von diesem Account aus deployed wurden bzw. überwacht werden. Es ist möglich, mit diesen zu interagieren und Funktionen auszuführen, den Namen zu ändern oder den Contract zu löschen.

Bei „Status“ lassen sich alle relevanten Daten über den Parity Client, der Verbindung und dem Mining anzeigen. So zum Beispiel der aktuellste Block, die Anzahl an Verbindungen zu anderen Knoten und der Netzwerk Port bzw. die Adresse für den Zugriff auf RPC, welche später gebraucht wird um eine Verbindung mit JavaScript herstellen zu können. Mit einem Klick auf den Play Knopf unten rechts, kann man sich alle von Parity protokollierten Ereignisse anzeigen lassen.

Für das Beispiel wurde die Programmiersprache Solidity und zum Schreiben der Solidity Realtime Compiler genutzt, der im Browser als Web-Anwendung unter: „https://ethereum.github.io/browser-solidity“ zur Verfügung steht.

Das Programm, welches als lokales Orakel eingesetzt werden soll, wurde in JavaScript mit NodeJS geschrieben.

Dies ermöglicht das Nutzen von Modulen wie „Request“ für das Abrufen von Daten aus Webseiten, „read“ für die Eingabe von Passwörtern und „Web3“ zur Interaktion mit dem Contract.

4.4 Ausführung

Parity

Nachdem Parity geladen und installiert ist, kann es per Eingabeaufforderung gestartet werden. Beim Start werden zusätzlich zum Aufruf von Parity selbst, noch weitere Parameter übergeben. So zum Beispiel der Befehl „--warp“, der die Warp-Synchronisation startet und „--chain=ropsten“, wodurch das Ropsten Testnet als zu benutzendes Netzwerk festgelegt wird. Da der Funktionsaufruf vom Orakel per Transaktionen zum Contract geschieht, muss der Account in der Konsole dafür freigeschaltet werden. Dazu müssen noch die folgenden API’s per http-RPC Interface verfügbar gemacht werden: „--rpcapi "eth,net,web3,personal" “. Am wichtigsten ist hier die „personal“ API, da nur mit dieser der Account freigeschalten werden kann. (Vgl. 73)

Der komplette Aufruf sieht dann so aus: „parity --warp --chain=ropsten --rpcapi „eth,net,web3,personal““.

Nun fängt Parity mit dem Synchronisieren der Blockchain an und lädt Block für Block auf den PC herunter. Das kann je nach Anzahl der Peers und der eigenen Internetverbindung einige Zeit dauern.

Orakel-Contract

Der Contract im Anhang Teil 2 ist der Orakel-Contract und dient als Speicher für die, vom Dienst erhaltene, Variable. Dieser hat Funktionen für das Aktualisieren und Ausgeben des Wechselkurses sowie des löschen des Contracts. Er wird nun über die Weboberfläche von Parity deployed. Dazu wählt man im Menü den Punkt „Contracts“ und klickt auf „Deploy Contract“. Dort muss nun ein Name vergeben werden und die ABI und der Bytecode angegeben werden. Diese findet man im Solidity Browser nachdem der Contract kompiliert wurde. Nachdem alle Werte eingetragen sind, wählt man „create“ und wird nach dem Passwort für den Account gefragt. Nach erfolgreicher Eingabe dessen, wird einem der Transaktionshash angezeigt und wie viele Miner der Transaktion schon zustimmen.

Wenn der Contract bestätigt wurde, findet man ihn in der Übersicht aller Contracts wieder und kann diesen öffnen. Dort kann man nun die Anzahl der Ether in dem Contract sehen, den Contract löschen bzw. dessen Name bearbeiten und die programmierten Funktionen ausführen. So kann man nun zum Beispiel die Funktion mit dem Knopf „Execute“ UpdateETHEUR wählen und dort einen beliebigen Wert eingeben. Das Passwort für den Account wird wieder abgefragt und nach erfolgreicher Bestätigung der Transaktion, wird der Wert im Contract aktualisiert.

Smart Contract

Um das Beispiel zu vollenden, wird ein Smart Contract benötigt, der auf den, vom lokalen Dienst aktualisierten, Orakel-Contract zugreift und die Variable ausliest.

Dazu muss zuerst ein Prototyp vom aufzurufenden Contract angegeben werden:

```
pragma solidity ^0.4.6;


// Der Contract, in dem der Tauschkurs geschrieben wurde
contract PriceFeed {
        uint256 ETHEUR;                 // Variable, in der der Tauschkurs zwischengespeichert wird


        // Die Funktion, die den Tauschkurs beim Aufruf ausgibt
        function GetETHEUR() returns(uint256){
                return ETHEUR;          // gibt die Variable ETHEUR an den Aufrufer zurück
        }
}
```

Nun folgt der eigentliche Contract. Dieser hat einen ähnlichen Aufbau wie der erste, mit dem Unterschied, dass es eine Variable gibt, in der die Adresse des Orakel-Contract angegeben wird. Der Smart Contract ist im Anhang Teil 3 zu finden. Nach dem deployen des Smart Contract, kann per Aufruf von Funktion setPF() der Wert aus dem Orakel-Contract gelesen und in den Smart Contract abgespeichert werden.

Erläuterung Programmierung

Um mit Ethereum zu kommunizieren wird web3 verwendet. Dafür gibt es eine ganze Reihe von Funktionen mit denen man auf Accounts zugreifen, Transaktionen senden und Funktionen ausführen kann. Geschrieben wurde dieses Programm in JavaScript und dem Modul „web3“ welches die oben genannten Funktionen zur Verfügung stellt.

Zuerst müssen die notwendigen Module importiert und mit einem Namen versehen werden:

```
// require für web3, request und read Module
var Web3 = require('web3');          // web3 für Interaktion mit parity
var request = require('request');    // request für Abfrage von Websiten
var read = require('read');          // read zum einlesen des Passwortes auf
der Konsole
```

Anschließend muss die Adresse des Contracts in einer Variable gespeichert werden. Diese wird angezeigt, nachdem man den Contract in der Parity Weboberfläche deployed hat.

```
// Adresse des Contracts
var contractAddress = "0x0189ECDAaDc7b1154A5DF6529a6c55e793dBfd6E";
```

Die Datenquelle für den Wechselkurs wird ebenfalls angegeben:

```
// URL der JSON-Datenquelle
var urlcrypto = "https://min-api.cryptocom-
pare.com/data/price?fsym=ETH&tsyms=EUR";
```

Auch die ABI wird gebraucht um auf den Contract zugreifen zu können. Diese findet man nach der Kompilierung, unter „Interface“ in Solidity:

```
// ABI von Contract
var contractABI = [{
    "constant":false,
    "inputs":[],
    "name":"kill",
    "outputs":[],
    "payable":false,
    "type":"function"
},{
    "constant":false,
    "inputs":[{"name":"result","type":"uint256"}],
    "name":"UpdateETHEUR",
    "outputs":[],
    "payable":false,
    "type":"function"
},{
    "constant":false,
    "inputs":[],
    "name":"GetETHEUR",
    "outputs":[{"name":"","type":"uint256"}],
    "payable":false,
    "type":"function"
},{
    "inputs":[],
    "payable":false,
    "type":"constructor"
}];
```

Nun muss eine Verbindung zum Parity Client erstellt werden. Dies wird per RPC bewerkstelligt und dazu braucht man die Adresse und den Port der auf der Weboberfläche unter „Status“ bei „rpc interface“ und „rpc port“ zu finden ist. Aus „local“ und „8545“ ergibt sich folgende Adresse „127.0.0.1:8545“ oder auch „localhost:8545“.

```
// Überprüfen ob schon ein Web3 Provider besteht, andernfalls neuen erstellen
if (typeof web3 !== 'undefined') {
    web3 = new Web3(web3.currentProvider);
} else {
    // Web3 Provider mit lokalen Parity Client verbinden
    var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
}
```

Da die Verbindung nun hergestellt ist, wird der erste Account der in Parity vorhanden ist, für spätere Zugriffe in eine Variable geschrieben:

```
// Adresse vom ersten Account in MyAddress schreiben
var MyAddress = web3.eth.accounts[0];
```

Nun kann per web3.eth auf die weiteren Funktionen von eth zugriffen werden. Um den Überblick zu bewahren, werden ein paar relevante Daten welche die Parity Version, Provider, letzten Block und den Account mit dem Kontostand anzeigen, ausgegeben.

```
// Ein paar Ausgaben für die Console zur Verbindung und Account
console.log("Eth Node Version    : ", web3.version.node);
console.log("Connected           : ", web3.isConnected());
console.log("Provider            : ", web3.currentProvider);
console.log("syncing             : ", web3.eth.syncing);
console.log("Latest Block        : ", web3.eth.blockNumber + "\n");
console.log("Accounts[0]         : ", MyAddress);
console.log("Balance[0]          : ", web3.eth.getBalance(MyAddress).toNumber()+" Wei\n");
```

Anschließend wird die Verbindung zum Contract herstellt. Dazu wird die Funktion contract() von eth aufgerufen und die ABI des Orakel-Contract übergeben. Das Ganze sieht dann so aus:

```
// ABI von Contract in MyContract speichern
var MyContract = web3.eth.contract(contractABI);

// Adresse hinzufügen
var PriceFeed = MyContract.at(contractAddress);
```

Nachdem die Verbindung zum Contract hergestellt wurde, kann auf dessen Funktionen zugegriffen werden. Nun muss der Wechselkurs aus der Datenquelle geladen und geparst werden.

```
// Request Funktion die JSON Daten von der Website holt und die Update
Funktion im Contract auslöst
request(urlcrypto, function (error, response, body) {
    // Errorcheck
    if(error){
        return console.log('Error:', error);
    }

    // Alles außer Statuscode 200 erzeugt Fehler
    if(response.statusCode !== 200){
            return console.log('Invalid Status Code Returned:',
response.statusCode);
    }
    // Nichts gefunden? Dann kanns ja weitergehen!
    console.log(body); // Kurze Ausgabe des Wertes auf der Konsole

    // Die eigentliche Zahl vom String ausschneiden und in Eurocent
umwandeln
    var formattedString = body.toString().slice(7,-1).replace(".","");

    // Ausgabe zum Kontrollieren
    console.log("aktueller Tauschkurs in Eurocent: " + formattedString);
```

Die letzte Aufgabe des Programmes ist dann, das Passwort von der Konsole einzulesen, den Account zu entsperren und die Funktion UpdateETHEUR() per Transaktion aufzurufen um den Wechselkurs zu übergeben. Das Passwort wird bei der Eingabe verschlüsselt angezeigt.

```
// Einlesen des Passwortes auf der Konsole und Aufruf der Updatefunktion.
Es muss eine Transaktion mit dem neuen Wert gesendet werden
    read({ prompt: 'Passwort: ', silent: true }, function(er, password) {
        console.log("Account freischalten :",
web3.personal.unlockAccount(MyAddress, password, 300));

        console.log("Versuche die Zahl in Contract zu schreiben......\n");
        console.log("Transaktionsadresse : ",
PriceFeed.UpdateETHEUR.sendTransaction(formattedString,{from: MyAddress}));

        process.exit(0);

        });

});
```

(Vgl. 74) (Vgl. 75)

5 Fazit und Ausblick

Obwohl Ethereum als Plattform für Smart Contracts erst 2014 (Vgl. 76) hervorgekommen ist, gibt es schon einige Anwendungsgebiete die diese digitalen Verträge nutzen. Es ließen sich auch nach einer umfangreichen Recherche nur wenige Anbieter finden, die externe Daten in Smart Contracts einbringen. Es sind zwei unterschiedliche Ansätzen zu erkennen: Oraclize und SmartContract beziehen die Daten von der angegebenen Quelle und schreiben sie in einen dafür erstellten Orakel Contract. Darauf kann dann von einem Smart Contract zugegriffen werden. Ein anderer Ansatz ist bei Realitykeys zu sehen, welcher keinen Orakel-Contract erstellt, sondern lediglich einen erweiterten Zugang zu den Datenquellen anbietet. Dies soll die Sicherheit erhöhen und Zugriff auf Webseiten ermöglichen, die keine Ausgabe im JSON oder XML Format zur Verfügung stellen. Die Unterschiede spiegeln sich auch in den Preismodellen wieder: Oraclize und SmartContract lassen sich abhängig von der Quelle und der Anzahl an Abfragen bezahlen. Realitykeys hingegen muss für die Leistung nur bezahlt werden, wenn eine menschliche Überprüfung der Daten gefordert wird.

Ausblick

Es ist zu erwarten, dass es in Zukunft mehr und mehr Orakel gibt, die sich auf den Finanzmarkt und dessen Vorhersage spezialisieren. Hier ist das Potential für einen neuen Markt an Diensten am größten, da sich damit hohe Gewinne erzielen lassen. (Vgl. 77) Dabei wird es allerdings immer wichtiger, dass auf die bereitgestellten Daten auch wirklich verlass ist. Die derzeitigen verfügbaren Beweise wie TLSNotary und Android Proof bringen noch einige Schwierigkeiten mit sich. So ist TLSNotary z.B. Versionsabhängig (setzt TLS1.0 oder 1.1 voraus) und braucht viele verteilte Server, damit der Beweis auch sicher ist. (Vgl. 78)

Neue Ansätze auf dem Gebiet bieten etwa Intel's Secure Guard Extension (SGX) und AndroidProof, welche beide Hardware einsetzen, um einen sicheren Beweis zu liefern.

SGX nutzt Hardware, um sichere Bereiche, sogenannte „Enklaven", für Software bereitzustellen. Sensible Daten können dort abgelegt und verarbeitet werden. Die Enklave ist nur im Prozessor entschlüsselt zu lesen und ist damit auch vor dem auslesen des RAM sicher. Zugriff darauf ist, auch von Programmen mit höheren Privilegien, nicht möglich. Der Befehlssatz, der SGX möglich macht, wurde 2015 mit der 6. Generation „Skylake" von Intels Prozessoren eingeführt. (Vgl. 79) Mit WolfSSL bietet die Firma WolfSSL derzeit schon eine Technologie an, welche SGX nutzt um TLS zu erweitern. (Vgl. 80)

Bei Android Proof wird ein Android Gerät genutzt, um eine HTTPS Verbindung herzustellen und die erhaltene Antwort mit SHA256 zu hashen. Anschließend wird mit einem, auf dem Gerät erstellen Schlüsselpaar, der Hashwert signiert und an die API von SaftyNet geschickt. Der Hardwarenachweis, und Schlüsselspeicher sind in einer Trusted Execution Environment (TEE) umgesetzt und der SafetyNet Softwarenachweis bei Google. Die TEE erlaubt den Nachweis, dass die Schlüssel wirklich in einem TEE erstellt und aufbewahrt werden, der Bootloader des Gerätes gesichert und verifiziert ist und die neuesten Updates auf dem Gerät installiert sind. Durch das SafetyNet wird weiterhin sichergestellt, dass das Gerät nicht gerootet wurde und eventuelle Fremdsoftware darauf läuft. Somit kann der komplette Weg, vom Aufruf der Website bis zum erhalten der Daten, nachgewiesen und gesichert werden. (Vgl. 81)

6 Literatur

1. **Foundation, Ethereum.** Ethereum.org. [Online] 2016. [Zitat vom: 08. 12 2016.] https://www.ethereum.org/.

2. **Community, Ethereum.** Ethdocs.org Introduction. *What is Ethereum.* [Online] 2016. [Zitat vom: 14. 11 2016.] http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html.

3. **Bitcoin.org.** Bitcoin.org. *Was ist Bitcoin.* [Online] 2016. [Zitat vom: 10. 10 2016.] https://bitcoin.org/de/faq#was-ist-bitcoin.

4. **Dai, Wei.** Weidai.com. *BMoney.* [Online] 1998. [Zitat vom: 2016. 11 28.] http://www.weidai.com/bmoney.txt.

5. **Bitcoin.org.** Bitcoin.org. *Wer hat Bitcoin erfunden.* [Online] 2017. [Zitat vom: 15. 01 2017.] https://bitcoin.org/de/faq#wer-hat-bitcoin-erfunden.

6. **Community, Ethereum.** Ethereum Wiki. *Design Rationale.* [Online] 04. 06 2016. [Zitat vom: 16. 01 2017.] https://github.com/ethereum/wiki/wiki/Design-Rationale.

7. **BitcoinProject.** BitcoinProject. *Developer Guide Proof of Work.* [Online] 2017. [Zitat vom: 23. 01 2017.] https://bitcoin.org/en/developer-guide#proof-of-work.

8. **LearnCryptography.com.** Learn Cryptography. *51% Attack.* [Online] 2016. [Zitat vom: 23. 01 2017.] https://learncryptography.com/cryptocurrency/51-attack.

9. **Gibson, Steve.** Gibson Research Corporation. *Security Now Transcript 388.* [Online] 2012. [Zitat vom: 16. 01 2017.] https://www.grc.com/sn/sn-388.pdf.

10. **Community, Ethereum.** Ethdocs.org Introduction. *How does Ethereum work?* [Online] 06. 01 2017. [Zitat vom: 16. 01 2017.] http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html#how-does-ethereum-work.

11. **O'Dwyer, Karl J. und Malone, David.** Bitcoin Mining and its Energy Footprint. [Online] 27. 06 2014. [Zitat vom: 21. 01 2017.] https://karlodwyer.github.io/publications/pdf/bitcoin_KJOD_2014.pdf.

12. **Buterin, Vitalik.** Ethereum Wiki. *Proof-of-Stake FAQ.* [Online] 29. 12 2016. [Zitat vom: 21. 01 2017.] https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ.

13. **Foundation, Ethereum.** Solidity.readthedocs.io. *Contracts Events.* [Online] 21. 01 2017. [Zitat vom: 23. 01 2017.] http://solidity.readthedocs.io/en/latest/contracts.html#events.

14. **Project, Bitcoin.** Bitcoing.org Developer Guide. *Blockchain Overview.* [Online] 2017. [Zitat vom: 23. 01 2017.] https://bitcoin.org/en/developer-guide#block-chain-overview.

15. **Sompolinsky, Yonatan und Zohar, Aviv.** Accelerating Bitcoin's Transaction Proccessing. [Online] 12 2013. [Zitat vom: 23. 01 2017.] http://eprint.iacr.org/2013/881.pdf.

16. **Community, Ethereum.** Ethdocs.org Indroduction. *What is Ethereum Ethereum Virtual Machine.* [Online] 2017. [Zitat vom: 23. 01 2017.] http://ethdocs.org/en/latest/introduction/what-is-ethereum.html#ethereum-virtual-machine.

17. **Jentzsch, Simon, Jentzsch, Christoph und Tual, Stephan.** Slock.it. *Solutions.* [Online] 2015. [Zitat vom: 16. 01 2017.] https://slock.it/solutions.html.

18. **Grassegger, Hannes.** Zeit.de. *Die erste Firma ohne Menschen.* [Online] 26. 05 2016. [Zitat vom: 16. 01 2017.] http://www.zeit.de/digital/internet/2016-05/blockchain-dao-crowdfunding-rekord-ethereum.

19. **Oraclize.it.** Oraclize.it. *About.* [Online] 2016. [Zitat vom: 17. 12 2016.] http://www.oraclize.it/#about.

20. **Schmidt, Oliver.** Datenschutzbeauftragter-Info. *Smart Contracts: Vertragsabwicklung durch Computer.* [Online] 22. 9 2015. [Zitat vom: 15. 11 2016.] https://www.datenschutzbeauftragter-info.de/smart-contracts-vertragsabwicklung-durch-computer/.

21. **Blockchaintechnologies.com.** Blockchaintechnologies. *Blockchain Smart Contracts.* [Online] 2016. [Zitat vom: 15. 01 2017.] http://www.blockchaintechnologies.com/blockchain-smart-contracts.

22. **Foundation, Ethereum.** Ethereum Frontier Guide. *Creating Contract.* [Online] [Zitat vom: 23. 01 2017.] https://ethereum.gitbooks.io/frontier-guide/content/creating_contract.html.

23. **Wood, Gavin.** Ethereum: A secure decentralised generalised transaction ledger. *EIP-150 Revision.* [Online] [Zitat vom: 23. 01 2017.] http://gavwood.com/Paper.pdf.

24. **Araoz, Manuel.** Zeppelin Blog. *The Hitchhiker's Guide to Smart Contracts in Ethereum.* [Online] 29. 06 2016. [Zitat vom: 23. 01 2017.] https://medium.com/zeppelin-blog/the-hitchhikers-guide-to-smart-contracts-in-ethereum-848f08001f05#.3liyepulw.

25. **Farrell , Scott, et al.** Kwm.com. *10 Things you need to know about Smart Contracts.* [Online] 06 2016. [Zitat vom: 14. 01 2017.] http://www.kwm.com/en/au/knowledge/insights/10-things-you-need-to-know-smart-contracts-20160630.

26. **Biederbeck, Max.** Wired.de. *Wie aus dem Hack des Blockchain Fonds DAO ein Wirtschaftskrimi wurde.* [Online] 21. 11 2016. [Zitat vom: 14. 01 2017.] https://www.wired.de/collection/business/wie-aus-dem-hack-des-blockchain-fonds-dao-ein-wirtschaftskrimi-wurde.

27. **Buterin, Vitalik.** Blog.Ethereum.com. *Transaction Spam Attack next Steps.* [Online] 22. 09 2016. [Zitat vom: 15. 01 2017.] https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/.

28. **Djazayeri, Dr. Alexander.** juris GmbH. *Juristisches Informationssystem für die BRD.* [Online] 20. 12 2016. [Zitat vom: 15. 01 2017.] https://www.juris.de/jportal/portal/page/homerl.psml?nid=jpr-NLBAADG000916&cmsuri=%2Fjuris%2Fde%2Fnachrichten%2Fzeigenachricht.jsp.

29. **Hellinger, Axel.** Hellinger.eu Smart Contract. *Wirksamkeit & Unwirksamkeit von Vertragsprogrammen.* [Online] 20. 06 2016. [Zitat vom: 15. 01 2017.] https://hellinger.eu/blog/smart-contract/.

30. **Community, Ethereum.** Ethereum Wiki White Paper. *Applications.* [Online] 25. 12 2016. [Zitat vom: 23. 01 2017.] https://github.com/ethereum/wiki/wiki/White-Paper#applications.

31. **Realitykeys.com.** realitykeys.com Developers. [Online] 2016. [Zitat vom: 30. 11 2016.] https://www.realitykeys.com/developers.

32. **Oraclize.it.** docs.oraclize.it Datasources. *WolframAlpha.* [Online] 04. 01 2017. [Zitat vom: 30. 11 2016.] http://docs.oraclize.it/#datasources-wolframalpha.

33. **Lewis, Antony.** Orisi White Paper. [Online] 29. 11 2014. [Zitat vom: 15. 01 2017.] https://github.com/orisi/wiki/wiki/Orisi-White-Paper.

34. **Kolinko, Tomasz, Pstrucha, Grzegorz und Kucharsk, Kuba .** Orisi.org. *Orisi - Distributed Oracle System.* [Online] 06. 06 2015. [Zitat vom: 30. 11 2016.] https://github.com/orisi/orisi.

35. **TLSNotary.org.** https://tlsnotary.org. *TLSnotary - a mechanism for independently.* [Online] 10. 09 2014. [Zitat vom: 17. 11 2016.] https://tlsnotary.org/TLSNotary.pdf.

36. **Group, TLSNotary.** TLSNotary Project. *Introduction.* [Online] 24. 02 2015. [Zitat vom: 19. 11 2016.] https://github.com/tlsnotary/tlsnotary.

37. **Oraclize.it.** docs.oraclize.it Security. *TLSNotary Proof.* [Online] 04. 01 2017. [Zitat vom: 17. 11 2016.] http://docs.oraclize.it/#security-tlsnotary-proof.

38. **Coleman, Jeff.** Ethereum.Stackexchange.com. *How does Oraclize handle the TLSnotary secret?* [Online] 27. 02 2016. [Zitat vom: 19. 11 2016.] http://ethereum.stackexchange.com/a/1636.

39. **dansmith.** Bitcointalk.org. *Setting up an oracle machine on Amazon's AWS.* [Online] 25. 09 2013. [Zitat vom: 27. 11 2016.] https://bitcointalk.org/index.php?topic=301538.0.

40. **SmartContract.com.** smartcontract.com. [Online] 2016. [Zitat vom: 22. 01 2017.] https://smartcontract.com/.

41. **Realitykeys.com.** realitykeys.com. [Online] 2016. [Zitat vom: 13. 01 2017.] https://www.realitykeys.com/.

42. **Bertani, Thomas.** Oraclize Blog. *The provably honest Oracle Service is finally here.* [Online] 4. 11 2015. [Zitat vom: 19. 12 2016.] https://blog.oraclize.it/oraclize-the-provably-honest-oracle-service-is-finally-here-3ac48358deb8#.a6gzfkzdh.

43. **Oraclize.it.** docs.oralize.it Overview. [Online] 18. 11 2016. [Zitat vom: 20. 12 2016.] https://docs.oraclize.it/?javascript#overview.

44. **Nazarov, Sergey.** LinkedIn.com. *Sergey Nazarov.* [Online] 2016. [Zitat vom: 20. 12 2016.] https://www.linkedin.com/in/sergeydnazarov.

45. **SmartContracts.com.** Angel.co SmartContract. [Online] 2016. [Zitat vom: 21. 12 2016.] https://angel.co/smartcontract.

46. **Kastelein, Richard.** The-Blockchain.com. *SmartContract.com Rolls Out Smart Oracles Platform, Connecting the Ethereum and Bitcoin Blockchains to the World's Traditional Data and Financial Infrastructure.* [Online] 30. 07 2016. [Zitat vom: 21. 12 2016.] http://www.the-blockchain.com/2016/07/30/smartcontract-com-rolls-smart-oracles-platform-connecting-ethereum-bitcoin-blockchains-worlds-traditional-data-financial-infrastructure/.

47. **Realitykeys.com.** realitykeys.com About. [Online] 2016. [Zitat vom: 20. 12 2016.] https://www.realitykeys.com/about.

48. **Oraclize.** Blog.oraclize.it. *Understanding Oracles.* [Online] 18. 02 2016. [Zitat vom: 22. 01 2017.] https://blog.oraclize.it/understanding-oracles-99055c9c9f7b#.diap28478.

49. **SmartContract.com.** smartcontract.com Create. [Online] 2016. [Zitat vom: 09. 01 2017.] https://create.smartcontract.com/#/drafts/bd791b07b291ba1efda06a8d327cf06b.

50. —. smartcontract.com Choose. [Online] 2016. [Zitat vom: 12. 01 2017.] https://testnet.smartcontract.com/#/choose.

51. **Oraclize.it.** docs.oraclize.it Datasources. [Online] 04. 01 2017. [Zitat vom: 11. 01 2017.] http://docs.oraclize.it/#datasources.

52. —. docs.oraclize.it Datasources. *URL.* [Online] 04. 01 2017. [Zitat vom: 11. 01 2017.] http://docs.oraclize.it/#datasources-url.

53. —. docs.oraclize.it Datasources. *Blockchain.* [Online] 04. 01 2017. [Zitat vom: 11. 01 2017.] http://docs.oraclize.it/#datasources-blockchain.

54. **Labs, Protocol.** Github.com. *IPFS - The Permanent Web.* [Online] 06. 01 2017. [Zitat vom: 22. 01 2017.] https://github.com/ipfs/ipfs.

55. **Oraclize.it.** docs.oraclize.it Datasources. *IPFS.* [Online] 04. 01 2017. [Zitat vom: 11. 01 2017.] http://docs.oraclize.it/#datasources-ipfs.

56. —. docs.oraclize.it Datasources. *Decrypt.* [Online] 04. 01 2017. [Zitat vom: 11. 01 2017.] http://docs.oraclize.it/#datasources-decrypt.

57. —. docs.oraclize.it Datasources. *Computation.* [Online] 04. 01 2017. [Zitat vom: 11. 01 2017.] http://docs.oraclize.it/#datasources-computation.

58. —. docs.oraclize.it Pricing. [Online] 04. 01 2017. [Zitat vom: 06. 12 2016.] http://docs.oraclize.it/#pricing.

59. **Realitykeys.com.** realitykeys.com Pricing. [Online] 2016. [Zitat vom: 06. 12 2016.] https://www.realitykeys.com/pricing.

60. **Oraclize.it.** docs.oraclize.it Ethereum Integration. *TLSNotary Proof.* [Online] 04. 01 2017. [Zitat vom: 11. 01 2017.] http://docs.oraclize.it/#ethereum-integration-tlsnotary-proof.

61. —. docs.oraclize.it Security. *TLSNotary Proof.* [Online] 04. 01 2017. [Zitat vom: 11. 01 2017.] http://docs.oraclize.it/#security-tlsnotary-proof.

62. —. docs.oraclize.it Datasources. *Nested.* [Online] 04. 01 2017. [Zitat vom: 11. 01 2017.] http://docs.oraclize.it/#datasources-nested.

63. **SmartContract.com.** smartcontract.com Features. [Online] 2016. [Zitat vom: 11. 01 2017.] https://smartcontract.com/features.

64. **Oraclize.it.** docs.oraclize.it Additional Features. *Encrypted Queries.* [Online] 04. 01 2017. [Zitat vom: 11. 01 2017.] http://docs.oraclize.it/#additional-features-encrypted-queries.

65. **bardi.harborow.** Bitcointalk Node.js ECIES. *Encrypt messages to Bitcoin Addresses.* [Online] 28. 05 2014. [Zitat vom: 11. 01 2017.] https://bitcointalk.org/index.php?topic=627927.0.

66. **Realitykeys.com.** realitykeys.com Developers. *Encryption.* [Online] 2016. [Zitat vom: 11. 01 2017.] https://www.realitykeys.com/developers/resources#encryption.

67. **SmartContract.com.** smartcontract.com Testnet Draft. [Online] 2017. [Zitat vom: 03. 01 2017.] https://testnet.smartcontract.com/#/drafts/083961451e4d086bbcd4921e1750da00.

68. —. smartcontract.com Test Contract. *ExchangeETHEURCrypto.* [Online] 2017. [Zitat vom: 03. 01 2017.] https://testnet.smartcontract.com/#/contracts/5e483ce991deefba762f99dce67a477d.

69. **Realitykeys.com.** realitykeys.com Exchange New. [Online] 2016. [Zitat vom: 04. 01 2017.] https://www.realitykeys.com/exchange/new.

70. —. realitykeys.com Exchange 12168. [Online] 2016. [Zitat vom: 04. 01 2017.] https://www.realitykeys.com/exchange/12168.

71. **Ethcore.io.** Ethcore.io. *Parity.* [Online] 2016. [Zitat vom: 08. 11 2016.] https://ethcore.io/parity.html.

72. —. Blog.ethcore.io. *Announcing Parity 1.4.* [Online] 07. 11 2016. [Zitat vom: 08. 11 2016.] https://blog.ethcore.io/announcing-parity-1-4/.

73. **Trón, Viktor.** Go-Ethereum Command Line Options. [Online] 17. 11 2016. [Zitat vom: 10. 11 2016.] https://github.com/ethereum/go-ethereum/wiki/Command-Line-Options.

74. **Manidos.** Web3 Javascript Dapp API. *Contract Accounts.* [Online] 09. 08 2016. [Zitat vom: 1. 11 2016.] https://github.com/ethereum/wiki/wiki/JavaScript-API#contract-methods.

75. **Häßler, Ulrike.** Mediaevent.de. *CSS, HTML und Javascript mit {still}.* [Online] 09 2016. [Zitat vom: 10. 11 2016.] https://www.mediaevent.de/javascript/Javascript-String-Methoden.html.

76. **Community, Ethereum.** Ethdocs.org Introduction. *A Next Generation Blockchain.* [Online] 2016. [Zitat vom: 12. 21 2016.] http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html#a-next-generation-blockchain.

77. **Dhaliwal, Shivdeep.** The Cointelegraph. *Ethereum-Based Oracle Hold Key to Future.* [Online] 20. 07 2016. [Zitat vom: 23. 01 2017.] https://cointelegraph.com/news/ethereum-based-oracle-holds-key-to-future.

78. **Merzdovnik, Georg, et al.** Whom You Gonna Trust? *A Longitudinal Study on TLS Notary Services.* [Online] [Zitat vom: 23. 01 2017.] https://www.sba-research.org/wp-content/uploads/publications/TLSnotaries_preprint.pdf.

79. **Anati, Ittai, et al.** Intel Developer Zone. *Innovative Technology for CPU Based Attestation and Sealing.* [Online] 14. 08 2016. [Zitat vom: 23. 01 2017.] https://software.intel.com/en-us/articles/innovative-technology-for-cpu-based-attestation-and-sealing.

80. **WolfSSL, Inc.** WolfSSL Products. *WolfSSL.* [Online] 2017. [Zitat vom: 23. 01 2017.] https://www.wolfssl.com/wolfSSL/Products-wolfssl.html.

81. **Oraclize.it.** Ipfs.io Android Proof. *Authenticated Data Gathering using Android Hardware Attestation and SafetyNet.* [Online] 22. 11 2016. [Zitat vom: 23. 01 2017.] https://ipfs.io/ipfs/QmWt9AZ3imf5eJsLUCPYmz1tNUD5TuLwMfcQsx8ai7z7Ud.

82. **Bitcoin.de.** Bitcoin.de. *Chart.* [Online] 2017. [Zitat vom: 16. 01 2017.] https://www.bitcoin.de/de/chart.

83. **Bitcoinist, Edmund Edgar.** Bitcoinist.net. *Realitykeys a certificate authority for facts.* [Online] 14. 08 2014. [Zitat vom: 20. 12 2016.] http://bitcoinist.com/reality-keys-a-certificate-authority-for-facts/.

84. **BTC-Echo.com.** Btc-Echo.com. *Was sind Bitcoins.* [Online] 2016. [Zitat vom: 13. 12 2016.] https://www.btc-echo.de/was-sind-bitcoins/.

85. **Coinmarketcap.com.** Crypto-Currency Market Capitalizations. [Online] 20. 11 2016. [Zitat vom: 20. 11 2016.] https://coinmarketcap.com/.

86. **Coleman, Jeff.** Ethereum.Stackexchange.com. *During a 51% attack, What Can the Attacker Actually Accomplish?* [Online] 24. 01 2016. [Zitat vom: 15. 01 2017.] http://ethereum.stackexchange.com/questions/542/during-a-51-attack-what-can-the-attacker-actually-accomplish.

87. **Limited, KYC-Chain.** Kyc-chain.com. *Why KYC-Chain?* [Online] 2016. [Zitat vom: 16. 01 2017.] http://kyc-chain.com/#why.

88. **Nakamoto, Satoshi.** Bitcoin.org. *Bitcoin: A Peer-to-Peer Electronic Cash System.* [Online] 2009. [Zitat vom: 16. 01 2017.] https://bitcoin.org/bitcoin.pdf.

89. **Oraclize.it.** LinkedIn.com. *Oraclize.* [Online] 05 2016. [Zitat vom: 20. 12 2016.] https://www.linkedin.com/company/oraclize-srl.

90. **Keuper, Ralf.** IT-Finanzmagazin.de. *Blockchain & Bitcoins: Wie Smart Contracts Banken überflüssig machen.* [Online] 13. 2 2015. [Zitat vom: 11. 10 2016.] http://www.it-

finanzmagazin.de/blockchain-bitcoins-wie-smart-contracts-banken-ueberfluessig-machen-werden-9078/.

91. **Slock.it.** Slock.it. *The Ethereum Computer.* [Online] 20. 11 2016. [Zitat vom: 20. 11 2016.] https://slock.it/ethereum_computer.html.

92. **Spiegel.de.** Spiegel Online. *Europäische Bankenaufsicht warnt vor Bitcoin.* [Online] 13. 12 2013. [Zitat vom: 16. 01 2017.] http://www.spiegel.de/wirtschaft/unternehmen/bitcoin-eu-bankenaufsicht-warnt-vor-virtuellen-waehrungen-a-938797.html.

93. **Systems, Social Minds Information.** Social Minds About. *Information Systems.* [Online] [Zitat vom: 20. 12 2016.] http://www.socialminds.jp/about.htm.

94. **Bitcoin.org.** Bitcoin.org FAQ. *Was sind die Vorteile von Bitcoin.* [Online] 2017. [Zitat vom: 16. 01 2017.] https://bitcoin.org/de/faq#was-sind-die-vorteile-von-bitcoin.

Anlagen

Anlagen, Teil 1

Der Smart Contract Oraclize:

```
pragma solidity ^0.4.4;
// Orakel-Contract
contract ExchangeETHEUR {
        uint256 ETHEUR;                    // Variable in der Wechselkurs zwischengespeichert wird
        // Die Funktion die den Wechselkurs beim Aufruf ausgibt
        function GetETHEUR() returns(uint256){
                return ETHEUR;
        }
}
// Smart Contract, der Orakel-Contract aufruft
contract Contract2_Oraclize {

        // Die Adresse wo der ursprüngliche Contract zu finden ist. Dieser wird in Variable strg
        gespeichert.
        ExchangeETHEUR strg =
ExchangeETHEUR(0xE5181a4a24E15b2C169CA6fdBD547A877C1b1D4A);
        uint256 public ETHEUR_O;      // Eine Variable für den Wechselkurs
        address owner;              // Eigentümer des Smart Contract

        // Contract2_Oraclize Funktion, startet mit Contract, legt Besitzer fest und ruft Variable ab
        function Contract2_Oraclize(){
        owner = msg.sender;       // Eigentümer ist derjenige, der Smart Contract zuerst aufruft
        setPF();
        }
```

```
        // Ruft die Variable von ExchangeETHEUR ab und schreibt die bekommene Variable in
        ETHEUR

        function setPF(){

                ETHEUR_O = strg.GetETHEUR();

        }

        // Funktion, die den Smart Contract löscht und alle Geldmittel an den Ersteller sendet

        function kill(){

                if (msg.sender == owner)

                        suicide(msg.sender);

        }

}
```

Anlagen, Teil 2

ABI des Orakel-Contract von SmartContract.com

```
contract Oracle{
        function Oracle();
        function update(bytes32 newCurrent);
        function current()constant returns(bytes32 current);
}
```

Und der entsprechende Smart Contract:

```
pragma solidity ^0.4.4;
// Orakel-Contract
contract Oracle {
        function current()constant returns(bytes32 current){
                return current;
        }
}
// Smart Contract
contract Contract2_SmartContract {

        // Die Adresse wo der Orakel-Contract zu finden ist. Dieser wird in Variable strg gespeichert.
        Oracle strg = Oracle(0x2E4b11da42dEc711C1394024f463929bD317460f);
                bytes32 public ETHEUR_S;      // Eine Variable für den Wechselkurs
                address owner;                // Eigentümer des Smart Contract
```

```
        // Contract2_SmartContract Funktion, startet mit Contract, legt Besitzer fest und ruft
        Variable ab

        function Contract2_SmartContract(){

                owner = msg.sender;       // Eigentümer ist derjenige, der Smart Contract zuerst
                aufruft

                setPF();

        }

        // Ruft die Variable von Oracle ab und schreibt die bekommene Variable in ETHEUR_S

        function setPF(){

                ETHEUR_S = strg.current();

        }

        // Funktion, die den Smart Contract löscht und alle Geldmittel an den Ersteller sendet

        function kill(){

                if (msg.sender == owner)

                        suicide(msg.sender);

        }

}
```

Anlagen, Teil 3

Orakel-Contract des Programmierbeispiels

```
// Solidity & Oracle
pragma solidity ^0.4.6;                    // legt Solidity Compiler auf Version 0.4.6 fest

// Dies ist der Smart Contract - PriceFeed
contract PriceFeed {

        uint256 ETHEUR;            // Variable in der Tauschkurs zwischengespeichert wird
        address owner;           // Eigentümer des Smart Contract

        // PriceFeed Funtion, startet mit Smart Contract, legt Eigentümer fast
        function PriceFeed(){
                owner = msg.sender;    // Eigentümer ist derjenige der Smart Contract zuerst aufruft
        }

        // Get Funktion, die Tauschkurs zurückgibt
        function GetETHEUR() returns(uint256){
                return ETHEUR;        // gibt die Variable ETHEUR an den Aufrufer zurück
        }

        // Update Funktion, bekommt aktuellen Kurs übergeben und aktualisiert Variable
        function UpdateETHEUR(uint256 result){
                ETHEUR = result;
        }

        // Funktion die den Smart Contract löscht und alle Geldmittel an den Ersteller sendet
        function kill(){
                if (msg.sender == owner)
                        suicide(msg.sender);
        }
}
```

Der Smart Contract aus dem Programmierbeispiel

```
// Der neue Contract als Beispiel zum Aufruf der Get Funktion
contract Contract2 {

        // Die Adresse wo der ursprüngliche Contract zu finden ist.
        //Dieser wird in Variable strg gespeichert.
        PriceFeed strg = PriceFeed(0x0189ECDAaDc7b1154A5DF6529a6c55e793dBfd6E);
        uint256 public ETHEUR;       // Eine Variable für den Tauschkurs
        address owner;               // Eigentümer des Smart Contract

        // Contract2 Funktion, startet mit Contract, legt Besitzer fest und ruft setPF() auf
        function Contract2(){
                // Eigentümer ist derjenige, der den Smart Contract zuerst aufruft
                owner = msg.sender;
                setPF();
        }
        // Ruft Get Funktion von PriceFeed auf und schreibt die bekommene Variable in ETHEUR
        function setPF(){
                ETHEUR = strg.GetETHEUR();
        }
        // Funktion die den Smart Contract löscht und alle Geldmittel an den Ersteller sendet
        function kill(){
                if (msg.sender == owner)
                        suicide(msg.sender);
        }
}
```

Anlagen, Teil 4

Quellcode für den lokalen Orakel Dienst:

```
// require für web3, request und read Module
var Web3 = require('web3');          // web3 für Interaktion mit parity
var request = require('request');    // request für Abfrage von Websiten
var read = require('read');          // read zum einlesen des Passwortes auf
der Konsole
// Adresse des Contracts
var contractAddress = "0x0189ECDAaDc7b1154A5DF6529a6c55e793dBfd6E";
// URL der JSON-Datenquelle
var urlcrypto = "https://min-
api.cryptocompare.com/data/price?fsym=ETH&tsyms=EUR";


// ABI von Contract
var contractABI = [{
    "constant":false,
    "inputs":[],
    "name":"kill",
    "outputs":[],
    "payable":false,
    "type":"function"
},{
    "constant":false,
    "inputs":[{"name":"result","type":"uint256"}],
    "name":"UpdateETHEUR",
    "outputs":[],
    "payable":false,
    "type":"function"
},{
    "constant":false,
    "inputs":[],
    "name":"GetETHEUR",
    "outputs":[{"name":"","type":"uint256"}],
    "payable":false,
    "type":"function"
},{
    "inputs":[],
    "payable":false,
    "type":"constructor"
}];

// Überprüfen ob schon ein Web3 Provider besteht, andernfalls neuen
erstellen
if (typeof web3 !== 'undefined') {
    web3 = new Web3(web3.currentProvider);
} else {
    // Web3 Provider mit lokalen Parity Client verbinden
    var web3 = new Web3(new
Web3.providers.HttpProvider("http://localhost:8545"));
}
// Adresse vom ersten Account in MyAddress schreiben
var MyAddress = web3.eth.accounts[0];
```

```
// Ein paar Ausgaben für die Console zur Verbindung und Account
console.log("Eth Node Version    : ", web3.version.node);
console.log("Connected          : ", web3.isConnected());
console.log("Provider           : ", web3.currentProvider);
console.log("syncing            : ", web3.eth.syncing);
console.log("Latest Block       : ", web3.eth.blockNumber + "\n");
console.log("Accounts[0]        : ", MyAddress);
console.log("Balance[0]         : ",
web3.eth.getBalance(MyAddress).toNumber()+" Wei\n");

// ABI von Contract in MyContract speichern
var MyContract = web3.eth.contract(contractABI);

// Adresse hinzufügen
var PriceFeed = MyContract.at(contractAddress);

// Request Funktion die JSON Daten von der Website holt und die Update
Funktion im Contract auslöst
request(urlcrypto, function (error, response, body) {
    // Errorcheck
    if(error){
        return console.log('Error:', error);
    }

    // Alles außer Statuscode 200 erzeugt Fehler
    if(response.statusCode !== 200){
            return console.log('Invalid Status Code Returned:',
response.statusCode);
    }
    // Nichts gefunden? Dann kanns ja weitergehen!
    console.log(body); // Kurze Ausgabe des Wertes auf der Konsole

    // Die eigentliche Zahl vom String ausschneiden und in Eurocent
umwandeln
    var formattedString = body.toString().slice(7,-1).replace(".","");

    // Ausgabe zum Kontrollieren
    console.log("aktueller Tauschkurs in Eurocent: " + formattedString);
   // Einlesen des Passwortes auf der Konsole und Aufruf der
Updatefunktion. Es muss eine Transaktion mit dem neuen Wert gesendet werden
    read({ prompt: 'Passwort: ', silent: true }, function(er, password) {
        console.log("Account freischalten :",
web3.personal.unlockAccount(MyAddress, password, 300));

        console.log("Versuche die Zahl in Contract zu schreiben......\n");
        console.log("Transaktionsadresse : ",
PriceFeed.UpdateETHEUR.sendTransaction(formattedString,{from: MyAddress}));

        process.exit(0);

        });

});
```

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, den 27.01.2017

Michaelis Roy