

bloqs.at

Stefan Huber

Version 0.0.1

Inhalt

Einführung	1
Bitcoin	1
Szenario 1	2
Einführung	2
Smart Contracts	3
Implementierung mit Solidity	4
Nutzung des Smart Contracts	5
Anhang A: Ethereum Entwicklungsumgebung	7
testrpc	7
truffle	7

Einführung

Bitcoin

"Bitcoin: A Peer-to-Peer Electronic Cash System" [1: Bitcoin: A Peer-to-Peer Electronic Cash System, Satoshi Nakamoto (<http://bitcoin.com/bitcoin.pdf>)] ist eine Publikation von unbekannten Autoren mit dem Pseudonym "Satoshi Nakamoto". Diese wurde am 1. November 2008 innerhalb einer Mailinglist [2: Link zur archivierten Mailingliste (<http://www.mail-archive.com/cryptography@metzdowd.com/msg09959.html>)] veröffentlicht. Die Publikation beschreibt auf 9 Seiten konzeptionell ein Transfersystem von virtuellen Tokens (Bitcoins), welches auf einem Peer-to-Peer Ansatz basiert. Das Interessante dabei ist, dass keine zentrale Stelle (Mittelsmann bzw. Third Party) benötigt wird um:

1. Transaktionen zwischen den Teilnehmern durchzuführen und zu prüfen
2. Neue Tokens im System zu schöpfen (sog. Minting)

Szenario 1

Im folgenden soll anhand eines einfachen Beispiels das Konzept der sog. Smart Contracts eingeführt werden. Anzumerken ist, dass dabei nur die technische Machbarkeit betrachtet wird. Rechtliche Aspekte werden dabei vollkommen außer acht gelassen, da dies als reine Demonstration zum Verständnis gedacht ist.

Einführung

Am Beispiel einer Sportwette sollen die Funktionsweise und die Eigenschaften eines Smart Contracts aufgezeigt werden. Dazu wird die Sportwette auf das Ergebnis eines Fußballspiels betrachtet. Ein Fußballspiel zwischen zwei Mannschaften A und B kann generell drei Ergebnisse bzw. Endzustände aufweisen:

- Mannschaft A gewinnt und Mannschaft B verliert
- Mannschaft B gewinnt und Mannschaft A verliert
- das Spiel geht unentschieden aus

Um nun eine Wette auf eines der möglichen Ergebnisse abzugeben, würde normalerweise ein sog. Buchmacher konsultiert. Dieser würde Wetten von unterschiedlichen Teilnehmern, mit unterschiedlichen Wetteinsätzen und natürlich unterschiedlichen Tipps entgegennehmen. Dies sollte klarerweise alles vor dem Start eines Fußballspiels geschehen. Aus den gesamten abgegebenen Wetten kann dann die sog. Gewinnquote für die jeweiligen Ergebnisse berechnet werden. Diese Quote gibt an mit welchem Faktor der Wetteinsatz im Falle eines richtigen Tipps multipliziert wird. Aus der Quote errechnet sich somit der mögliche Gewinn für die Wettteilnehmer. In Tabelle 1 wird ein Beispiel eines möglichen Wettverlaufs dargestellt. In diesem vereinfachten Beispiel ist davon auszugehen, dass keine Gebühren für den Buchmacher bzw. keine Steuern erhoben werden.

Ergebnis	kummulierte Einsätze	Prozent	Gewinnquote
Mannschaft A gewinnt	€ 20.000	25%	4
Mannschaft B gewinnt	€ 50.000	62,5%	1,6
unentschieden	€ 10.000	12,5%	8
gesamt	€ 80.000	100%	

Die Wettquote berechnet sich somit konkret über den gesamten Wetteinsatz dividiert durch die kumulierten Wetteinsätze aller Teilnehmer für den jeweiligen Tipp. Die Gewinnquote bedeutet somit für einen Wettteilnehmer, dass für jeden Euro der investiert wird das 4-, 1,6- bzw. 8-fache als Gewinn verbucht werden kann. An den Beispielzahlen ist zu erkennen, dass eine Gewinnquote immer größer als 1 sein muss, andernfalls würde die Teilnahme an einer Wette keinen Sinn machen. Zusätzlich ist zu sehen, dass der Tipp mit dem kleinsten kumulierten Wetteinsatz die höchste Gewinnchance bietet. Eine Wette auf das Ergebnis mit der höchsten Gewinnchance ist oft auch mit dem größten Verlustrisiko verbunden.



Wetten könnten auch als Vorhersagen betrachtet werden und die kummulierten Wetteinsätze als Eintrittswahrscheinlichkeit gedeutet werden. Im Beispiel der Fußballwette könnte dies so interpretiert werden, dass mit einer Wahrscheinlichkeit von 12,5% ein Unentschieden gespielt wird. Blockchainstartups wie **Augur** [6: Augur: <https://augur.net>], **Gnosis** [7: Gnosis: <https://gnosis.pm/>] oder **Stox** [8: Stox: <https://www.stox.com>] machen sich genau diesen Umstand zu nutze und bietet Vorhersagen der 'kollektiven Intelligenz' als Dienstleistung an. Dies ist auch unter den Bezeichnungen 'crowd sourcing' oder 'prediction markets' bekannt.

Smart Contracts

Üblicherweise würde eine Sportwette wie oben beschrieben von einem Buchmacher abgewickelt. Dieser Buchmacher würde sich um die ordnungsmäßige Abwicklung der Wette kümmern. Je nach Vertrauen gegenüber dem Buchmacher, würden Wettteilnehmer dort Wetten abschließen oder auch nicht. Je nach Einsicht der Wettteilnehmer könnte ein Buchmacher Gewinnquoten zu seinen Gunsten manipulieren oder hohe Gebühren für Gewinne erheben.

Die ordnungsmäßige bzw. korrekte Abwicklung einer Sportwette kann jedoch formal wie in einem Computerprogramm beschrieben werden. Diesen Umstand haben sich bereits etliche online Sportwettanbieter zu nutze gemacht. Anstelle einer vertrauenswürdigen Person, welche als Buchmacher Wetten entgegennimmt, kann dies auch über eine Software Anwendung geschehen. Hierbei wird jedoch nur die Abwicklung an sich digitalisiert, eine Manipulation bzw. das Bezahlen hoher Gebühren ist dabei noch nicht ausgeschlossen. Gemeinhin ist es so, dass Software Anwendungen zentral verwaltet sind und es ist nicht möglich den Programmcode einzusehen. Es wäre einem Wettteilnehmer also nicht möglich zu prüfen, ob die Software des Sportwettanbieters so agiert wie angepriesen.

Ein Smart Contract hingegen ist ähnlich einem Computerprogramm, das nachvollziehbar genau das macht, was beschrieben ist und dies unmanipulierbar durchführt. Im Englischen wird dies auch mit der Aussage "Code is Law" bekräftigt. Dies deutet darauf hin, dass so wie es im Quelltext des Smart Contracts beschrieben ist, so gilt es auch (ähnlich wie ein Gesetz). Konzeptionell ist ein Smart Contract einem Computerprogramm sehr ähnlich, jedoch gibt es einige unterscheidende Merkmale:

- Smart Contracts sind autonom in ihrer Ausführung. Falls ein Smart Contract einmal im Netzwerk bereitgestellt ist, steht dieser dort für immer bereit. Die programmierten Regeln sind somit gültig und können nicht mehr abgeschaltet oder verändert werden.
- Die Kommunikation mit Smart Contracts wird über Transaktionen realisiert. Dabei kann eine Transaktion sowohl Daten als auch Werte, abgebildet als virtueller Token, enthalten. Mit der Transaktion wird dem Smart Contract die Hoheit über die übermittelten Werte übertragen.
- Für die Ausführung eines Smart Contracts muss bezahlt werden. Die Mining-Knoten, welche das Netzwerk betreiben, bekommen eine Gebühr (Mining Fee) als Entschädigung für ihre Arbeitsleistung. Jede Transaktion, welche mit dem Smart Contract interagiert, muss auch genügend Mittel bereitstellen, damit die programmierten Regeln ausgeführt werden.

Konkret ist die Ausführung des Smart Contracts in Abschnitte gegliedert. Jeder Abschnitt wird

durch eine Transaktion bzw. durch ein Ereignis gestartet. Am Beispiel der Sportwette können 3 Abschnitte identifiziert werden:

- Abschnitt 1: Innerhalb dieses Abschnitts können Wettteilnehmer ihre Wetten platzieren. Diese Phase würde solange andauern, bis das Fußballspiel gestartet wird. Nach diesem Ereignis können keine Wetten mehr platziert werden.
- Abschnitt 2: Dieser Abschnitt würde während das Fußballspiels ablaufen. Der Smart Contract wäre im sog. Leerlauf und würde nur auf das Ereignis "Ende des Fußballspiels" warten.
- Abschnitt 3: Der letzte Abschnitt, nachdem das Spielergebnis feststeht, wäre die Auszahlung. Die Wettteilnehmer, welche die Wette zu ihren Gunsten platziert haben, können sich nun ihre Gewinne auszahlen.

Implementierung mit Solidity

Der Solidity Programmcode ist innerhalb einem Verzeichnisses auf github [9: Szenario 1 auf github: <https://github.com/getblogs/scenario01>] hinterlegt und einsehbar. In diesem Abschnitt werden einige Aspekte des Programmcodes erläutert. Dies soll auch dazu dienen, wesentliche Konzepte der Programmiersprache Solidity einzuführen.

Listing 1

```
contract SimpleBet {  
}
```

Jeder Smart Contract wird wie oben veranschaulicht (Listing 1) über das Schlüsselwort `contract` deklariert. Dies hat eine starke Ähnlichkeit zur Deklaration von Klassen in Objektorientierten Programmiersprachen (OOP). Im allgemeinen gibt es eine Vielzahl Ähnlichkeiten mit OOP, sodass für Softwareentwickler das Erlernen von Solidity eine geringe Lernkurve aufweist.

Listing 2

```
contract SimpleBet is Owned {  
}
```

Solidity unterstützt auch das Konzept der Vererbung (Listing 2). Dies ist hilfreich um die Wiederverwendbarkeit von Programmcode zu erhöhen. Über das Schlüsselwort `is` erbt `SimpleBet` von `Owned`. Alle Funktionen, welche in `Owned` definiert sind, sind somit auch Teil von `SimpleBet`. Neben der einfachen Vererbung werden auch `Interfaces` bzw. `abstrakte Methoden` unterstützt, dies wird in einem späteren Abschnitt noch thematisiert.

Listing 3

```
contract SimpleBet is Owned {  
  
    // unique identifier of sports game  
    string public game;  
  
    function SportsBet(string _game) {  
        game = _game;  
    }  
}
```

Eine wesentliche und wichtige Eigenschaft von Smart Contracts ist es Zustände zu speichern. Dazu können Attribute bzw. Variablen deklariert werden. In Listing 3 wird dazu eine Variable mit Namen `game` deklariert. Diese Variable hat den Datentyp `string` und kann Zeichenketten enthalten. Solidity unterstützen unterschiedlichste Datentypen, eine umfassende Liste ist der Dokumentation [10: Liste aller Solidity Datentypen: <http://solidity.readthedocs.io/en/latest/types.html>] zu entnehmen. Mit der Angabe `public` wird die Sichtbarkeit der Variable für andere Smart Contracts angegeben. Somit kann die Variable `game` von anderen Smart Contracts abgefragt werden. Neben `public` gibt es auch die Sichtbarkeit `private`, diese Angabe würde die Variable vor anderen Smart Contracts verstecken (ebenfalls innerhalb der Vererbungshierarchie).



Sichtbarkeit von Zuständen

Alle Zustände, welche innerhalb eines Smart Contracts hinterlegt werden, sind grundsätzlich über die Blockchain öffentlich einsehbar. Die Sichtbarkeit einer Variable innerhalb eines Smart Contracts (`public` oder `private`) bezieht sich dabei nur auf die programmatischen Zugriffsmöglichkeiten anderer Smart Contracts. Es gibt keine Möglichkeit Daten "nicht-öffentlich" zu speichern innerhalb einer Blockchain. Somit sind alle Daten die von einem Smart Contract zur Bearbeitung benötigt werden öffentlich.

Nutzung des Smart Contracts

Der Smart Contract wurde mit der Programmiersprache Solidity entwickelt, welche in einen Ethereum Smart Contract kompiliert werden kann. Das Ergebnis dieser Kompilierung ist letztlich nichts anderes als Bytecode, welcher auf der Ethereum Virtual Machine (EVM) ausgeführt werden kann.

Der kompilierte Smart Contract kann über eine Ethereum Wallet im Netzwerk bereitgestellt werden. Dies erfordert eine Transaktion an eine leere Adresse, welche als Nachricht den kompilierten Smart Contract enthält. Für diese Bereitstellung muss der Transaktion natürlich genügend Gas als Gebühr mitgegeben werden. Diese Bereitstellung führt ebenfalls den Konstruktor des Smart Contracts aus, das bereitgestellte Gas muss auch dafür reichen.

Nachdem der Smart Contract bereitgestellt wurde, kann über Transaktionen mit seiner öffentlichen Schnittstelle (Funktionen) kommuniziert werden. Der Smart Contract besitzt eine eindeutige Adresse und kann darüber identifiziert und angesprochen werden. Nutzer können Transaktionen an diese Adresse senden. Innerhalb der Nachricht können die gewünschten Funktionen bzw.

Funktionsparameter angegeben werden. Am Beispiel der Sportwette würde innerhalb der Nachricht ein Tipp für das Ergebnis angegeben.

Anhang A: Ethereum Entwicklungsumgebung

testrpc

Smart Contracts, welche für die Ethereum Blockchain entwickelt werden, sollten auch entsprechend getestet werden. Erst dannach sollten die Smart Contracts im Livenet installiert werden. Neben dem Livenet gibt es für Ethereum auch reale Testnetzwerke, welche von der Ethereum Community betrieben werden. Um Smart Contracts am eigenen Rechner zu entwickeln und testen, kann auf ein simuliertes Testnetzwerk zurückgegriffen werden. Die node.js Anwendung `testrpc` stellt ein solches simuliertes Netzwerk bereit.

Der Quellcode und etwas Dokumentation zu `testrpc` findet sich auf github [11: Link zu testrpc: <https://github.com/ethereumjs/testrpc>]. Die Anwendung kann mithilfe des `node package managers` installiert werden. Dazu muss folgender Befehl ausgeführt werden.

```
$ npm install -g ethereumjs-testrpc
```

Nach Installation kann das simulierte Netzwerk über die Kommandozeile gestartet werden. Defaultmäßig wird das Netzwerk auf Port `8545` ausgeführt. Dies kann jedoch über Optionen an den Startbefehl geändert werden. Details zu den unterschiedlichen Optionen ist der Dokumentation zu entnehmen.

truffle