Sonderdruck aus JavaSPEKTRUM 03/2017

Woher weiß die Blockchain, dass Dortmund gewonnen hat?

Orakel-Dienste speisen externe Informationen in die Blockchain

Christoph Niemann

Mit der Ethereum-Blockchain lassen sich digitale Verträge, sogenannte Smart Contracts, über die Blockchain-Technologie abwickeln. Neben Bitcoin ist Ethereum die zweitgrößte Kryptowährung, setzt aber mit dem Fokus auf intelligente Verträge einen anderen Schwerpunkt. Dabei gibt es eine Schwierigkeit: Die Blockchain selbst kann nicht auf externe Dienste zugreifen, also auch keine Informationen abfragen. Was tut man, wenn ein Smart Contract an externe Ereignisse als Bedingung geknüpft ist?

© Wikimedia

Ethereum hat sich als zweite große Blockchain neben Bitcoin etabliert. Die Kryptowährung stellt – neben Bezahlfunktionen, wie sie auch Bitcoin bietet – intelligente Verträge, sogenannte *Smart Contracts*, in den Vordergrund. Die Smart Contracts codieren analog zu Papierverträgen Vertragsbedingungen zwischen Vertragspartnern.

Smart Contracts können andere Verträge in der Blockchain aufrufen oder selbst komplexe Berechnungen anstellen. Da eine Blockchain ein „Append-only"-System ist, kann ein einmal in die Blockchain gestellter Vertrag unter normalen Umständen nicht mehr verändert werden. Der Vertrag wird die dort codierten Vertragsbedingungen durchsetzen. Jeder kann diesen Vertrag einsehen und prüfen, ob er sich verhält wie versprochen.

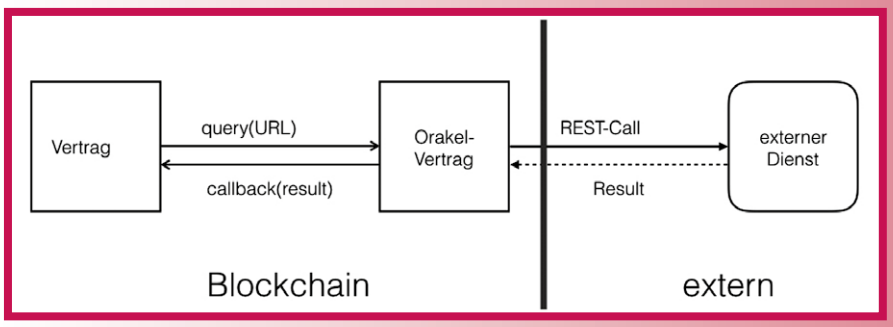

Abb. 1: Basisfunktionalität eines Orakels

Eine Wette als Vertrag

Ein Beispiel für einen einfachen Smart Contract ist eine Wette: Der Vertrag ist die Plattform für die Wette, Nutzer nehmen teil, indem sie einen Wetteinsatz zahlen. Sobald das Wetterereignis eingetreten ist, stellt der Vertrag den oder die Gewinner fest und veranlasst die Auszahlung der Gewinne. Der Code des Vertrags ist in der Blockchain einsehbar, daher muss ein Nutzer ihm nicht *vertrauen*, sondern kann *verifizieren*, dass der Gewinn ausgezahlt wird.

Eine solche Verifizierung funktioniert, wenn alle nötigen Daten in der Blockchain vorhanden sind: Der Vertrag hat darauf Zugriff und kann die Daten verarbeiten. Schwierig wird es, wenn das Wetterereignis sich *nicht* auf ein Ereignis in der Blockchain selbst bezieht, sondern auf ein Ereignis aus der realen Welt, wie die Wette auf ein Bundesliga-Spiel. Zwar sind alle Vertragsklauseln in der Blockchain überprüfbar, das wesentliche Ereignis, das Ereignis des Bundesligaspiels, jedoch nicht. Der Vertrag kann das Wettereignis ohne externe Daten nicht auswerten.

Der Datenfluss in die Blockchain ist eine Einbahnstraße: Daten können von anderen Quellen in die Blockchain als Transaktion geschrieben werden *(push)*, ein Abruf von Daten aus der Blockchain heraus *(pull)* ist nicht möglich.

Der Wettvertrag kann das Ergebnis der Wette nur weiterverarbeiten und die Gewinne auszahlen, wenn das Ergebnis des Fußballspiels als Transaktion in die Blockchain gelangt.

Im einfachsten Fall gäbe es einen Service, der *blockchain aware* ist und die Fußballergebnisse in die Blockchain schreibt. In der Regel interagieren externe Dienste jedoch nicht direkt mit einer Blockchain. Viele Dienste stellen allerdings APIs zur Verfügung, mit denen Daten per REST-Call abgerufen werden können. Diese Programmierschnittstellen bieten die Daten in der Regel als JSON-Objekt an und damit bereits in einem Format, das von Programmen gut weiterverarbeitet werden kann. Lediglich die Transaktion in die Blockchain fehlt noch.

Diese Funktion übernehmen sogenannte Orakel-Dienste. Sie stellen *in* der Blockchain das Orakel bereit: ein Vertrag, an den andere Verträge Anfragen schicken. Das Orakel liefert dem anfragenden Vertrag ein Ergebnis zurück. Für den anfragenden Vertrag zeigt sich das Orakel als Black Box: Auf eine Anfrage liefert es – wie das antike Orakel von Delphi – eine Antwort, die offen für Interpretationen ist. Der Vertrag weiß nicht, wie die Antwort zustande gekommen ist. Er kann dem Orakel zunächst nur glauben. Der Code des Orakels liegt in der Blockchain und kann verifiziert werden. Allerdings benötigt auch das Orakel Daten aus der realen Welt. Es arbeitet dafür mit einem Dienst *außerhalb* der Blockchain zusammen, der dafür

sorgt, dass die angefragten Daten per Transaktion in die Blockchain kommen.

Ein Nutzer, der an der Wette teilnimmt, kann den Wettvertrag überprüfen, dem Orakel-Dienst muss er jedoch vertrauen. Mechanismen, die dieses Vertrauen rechtfertigen, beschreibe ich unten.

Orakel in der Praxis

Der Orakel-Dienst *oraclize.it* [oraclize.it] stellt für Ethereum (und auch für Bitcoin) ein Orakel bereit, an das Anfragen gestellt werden können. Er sorgt dafür, dass die Anfrage eines Vertrags an den Server gestellt wird, und ruft dann eine Callback-Funktion im anfragenden Vertrag auf.

Für unser Beispiel einer Fußballwette greift der Vertrag auf die Programmierschnittstelle von football-data.org zurück. Sie bietet die Ergebnisse der Bundesligaspiele über ein REST-API an. Der vollständige Code des Beispiels ist auf Github verfügbar [FootballBet]. Im Code in Listing 1 sind die Aufrufe relativ einfach (hier leicht verkürzt). Für die Listings wurde Solidity verwendet, eine für die Ethereum Virtual Machine entwickelte Sprache, deren Syntax an JavaScript angelehnt ist.

```
contract FootballBet is usingOraclize {
  struct Game {
    string gameId; string date;
    string status; string homeTeam;
    string awayTeam;
    uint homeTeamGoals; uint awayTeamGoals;
    Result result;
  }
  struct Request {
    bool initialized; bool processed;
    string key;
  }
  Game game;
  mapping (bytes32 => Request) requests;

  function queryFootballData(string gameId,
                             string key, uint gas) public {
    if (oraclize_getPrice('URL') > this.balance) {
      Info('Oraclize query was NOT sent, ' +
           'please add some ETH to cover for the query fee');
    } else {
      string memory url = generateUrl(
        'https://api.football-data.org/v1/fixtures/',
        gameId, '?head2head=0', key);
      bytes32 requestId = oraclize_query('URL', url, gas);
      requests[requestId] = Request(true, false, key);
      Info('Oraclize query was sent, standing by for the answer..');
    }
  }
  function __callback(bytes32 myid, string result) public {
    Request memory r = requests[myid];

    if (r.initialized && !r.processed) {
    // new response
    if (r.key.toSlice().equals(JSON_FIXTURE.toSlice())) {
      var (success, tokens, numberTokens) = JsmnSol.parse(result, 45);
      if (success) {
        for (uint k=0; k<=numberTokens; k++) {
          // get Token contents
          if (tokens[k].jsmnType == JsmnSol.JsmnType.STRING) {
            string memory key = JsmnSol.getBytes(result, tokens[k]);
            if (key.toSlice().equals(JSON_STATUS.toSlice())) {
              game.status = JsmnSol.getBytes(result, tokens[++k]);
            } else if (key.toSlice().equals(JSON_HOME_TEAM.toSlice())) {
              game.homeTeam = JsmnSol.getBytes(result, tokens[++k]);
            } else if (key.toSlice().equals(JSON_AWAY_TEAM.toSlice())) {
              game.awayTeam = JsmnSol.getBytes(result, tokens[++k]);
            }
```

```
          }
        }
      }
    }
    Info(gameToString());
    requests[myid].processed = true;
    }
  }
}
```

Listing 1: FootballBet

Der Vertrag leitet sich von **usingOraclize** ab. Diesen Elternvertrag stellt der Orakel-Dienst bereit, um das Orakel zu nutzen. Insbesondere sind damit die beiden Funktionen **oraclize_query('URL', url, gas)** und **__callback(bytes32 myid, string result)** definiert, über die mit dem Orakel interagiert wird.

Der Vertrag kann Anfragen an das Orakel stellen, indem er die Art der Datenquelle (URL) definiert und außerdem den Pfad für den Abruf der Daten mitgibt:

```
json(https://api.football-data.org/v1/fixtures/{gameId}?head2head=0)
  {jsonPath}
```

Der Pfad enthält neben der eigentlichen URL zusätzlich einen JSONPath-Ausdruck. Das Ergebnis des REST-Calls ist ein JSON-Objekt, das mit dem JSONPath spezifisch selektiert werden kann. Das Resultat für den Vertrag enthält nur das Ergebnis der Selektion. Da die String-Verarbeitung innerhalb der Blockchain relativ teuer ist, ist die Verkleinerung des Response-Objektes oft sinnvoll. Im Wettvertrag liefert die Programmierschnittstelle Football-Data ein JSON (s. Listing 2).

```
{
  "fixture": {
    "date": "2016-08-27T13:30:00Z",
    "status": "FINISHED",
    "matchday": 1,
    "homeTeamName": "FC Augsburg",
    "awayTeamName": "VfL Wolfsburg",
    "result": {
      "goalsHomeTeam": 0,
      "goalsAwayTeam": 2
    }
  }
}
```

Listing 2: Wettvertrag

Der JSONPath-Ausdruck **$.fixture.result** verkürzt das JSON-Objekt auf **{ "goalsHomeTeam": 0, "goalsAwayTeam": 2 }**. Im Callback lässt sich das einfacher weiterverarbeiten.

Der Aufruf von **oraclize_query** liefert eine eindeutige ID zurück, mit der der Aufruf der Callback-Funktion später einer spezifischen Abfrage zugeordnet werden kann. Der Vertrag speichert den übergebenen Schlüssel in einem Request-Objekt. Dieses Objekt wird in einem Mapping abgelegt, sodass anhand der ID auf das zugehörige Request-Objekt zugegriffen werden kann. Über die Eigenschaften **Request.processed** stellt der Vertrag sicher, dass jede Antwort einer Anfrage nur einmal verarbeitet wird. Diese Maßnahme schützt gegen Replay-Attacken.

Außerdem werden nur Responses im Callback ausgewertet, die initialisiert wurden. Das sichert die Eigenschaft **Request.initialized**. Per default ist sie mit false initialisiert, nur echte Anfragen initialisieren ein Request-Objekt mit **Request.initialized =**

`true`. Wenn das Request-Objekt diesen Wert gesetzt hat, hat es eine entsprechende Anfrage tatsächlich gegeben.

Orakel wollen bezahlt werden

Orakel-Dienste müssen neben dem Vertrag in der Blockchain auch Server außerhalb der Blockchain betreiben, um die Transaktionen mit den Abfrageergebnissen auszulösen. Daher verlangen sie für jeden Aufruf eine Gebühr. Diese Gebühr bezahlt der Vertrag, der das Orakel anspricht. Deshalb muss er über genügend Ether verfügen. `queryFootballData` stellt sicher, dass tatsächlich genug Budget im Vertrag liegt.

Neben den Gebühren für den Orakel-Abruf fallen weitere Gebühren an, die der initiierende Transaktionspartner der Ethereum-Blockchain in Form von sogenanntem „Gas“ begleicht. Ruft das Orakel die Callback-Funktion auf, trägt es dafür die Transaktionskosten. Die Höhe der Kosten richtet sich nach der Komplexität der Callback-Funktion. Im Beispiel der Fußballwette liefert die Programmierschnittstelle ein JSON zurück. Im ersten Aufruf ist der Vertrag an mehreren Werten aus dem JSON interessiert: Er will die Namen der Heim- und der Gastmannschaft und außerdem den Status des Spiels ablegen. Daher muss der Vertrag das JSON-Objekt parsen und in seine Elemente zerlegen. String-Verarbeitung ist in Solidity relativ kostenintensiv. Daher ist die Callback-Funktion unserer Fußballwette eine teure Transaktion, die nicht mit den standardmäßig angenommenen 200.000 Gas auskommt. Der dritte Parameter gibt an, wie viel Gas man der Callback-Funktion zur Verfügung stellen will.

Der Vertrag kommt für die Gebühren des Orakels und die Transaktionskosten der Callback-Funktion auf. Beides wird beim Aufruf des Orakels an das Orakel gezahlt. Der Vertrag für den Aufruf des Orakels muss in jedem Fall über ausreichend Budget verfügen.

© Wikimedia

Orakel beweisen ihre Aufrichtigkeit

Das Vertrauen in die Blockchain leitet sich aus der Verifizierbarkeit aller Transaktionen in der Kette von Blöcken ab. Durch den Orakel-Dienst haben wir eine Möglichkeit, externe Daten in die Blockchain einzuspeisen. Jede Transaktion, die solche Daten benutzt, ist per Definition so vertrauenswürdig wie die Datenquelle. Ein Vertrag für Wetten auf ein Fußballspiel findet nur Benutzer, wenn die Fußballergebnisse unverfälscht in der Blockchain auftauchen.

Der erste Vertrauensanker ist die Datenquelle, in unserem Fall die Programmierschnittstelle von football-data.org. Für die Zwecke des Artikels nehmen wir an, dass die Quelle vertrauenswürdig ist. Wenn die Daten des API unverfälscht an den Vertrag übermittelt werden, kann ein Nutzer dem Vertrag genauso vertrauen wie der Datenquelle.

Mit einem Orakel-Dienst ist ein dritter Service zwischen Datenquelle und Vertrag in der Blockchain geschaltet, dem der Benutzer im einfachsten Fall zusätzlich vertrauen muss. Das ist ungünstig. Deshalb bietet oraclize.it an, die Daten bei der Übergabe an den Vertrag per TLSNotary [TLSNotary] zu notarisieren. Damit weist es nach, dass die übermittelten Daten in dieser Form von der Datenquelle zurückgeliefert wurden, der Nutzer muss dem Orakel nicht mehr vertrauen. Das Orakel kann die Daten bei der Übermittlung nicht eigenhändig verändern. Der TLSNotary-Proof verlängert die Vertrauenskette: Ein Benutzer muss weiterhin der Datenquelle vertrauen. Sowohl die Übermittlung durch das Orakel als auch die Auswertung des Ergebnisses im Vertrag sind transparent und nicht veränderbar.

```
function queryFootballData(string gameId,
                           string key, uint gas) public {
  if (oraclize_getPrice('URL') > this.balance) {
    Info('Oraclize query was NOT sent, ' +
         'please add some ETH to cover for the query fee');
  } else {
    string memory url = generateUrl(
      'https://api.football-data.org/v1/fixtures/',
      gameId, '?head2head=0', key);
    oraclize_setProof(proofType_TLSNotary | proofStorage_IPFS);
    bytes32 requestId = oraclize_query('URL', url, gas);
    requests[requestId] = Request(true, false, key);
    Info('Oraclize query was sent, standing by for the answer..');
  }
}
```

Listing 3: Proof-Typ

Im Smart Contract muss vor dem Aufruf des Orakels der „Proof-Typ“ gesetzt werden (s. Listing 3). Der Aufruf

```
oraclize_setProof(proofType_TLSNotary | proofStorage_IPFS);
```

weist den Orakel-Dienst an, den Serverabruf zu notarisieren und das Ergebnis (den „Proof“) im *InterPlanetary Filesystems* (IPFS) abzulegen. TLSNotary erlaubt es, eine mit TLS 1.0 oder TLS 1.1 gesicherte Verbindung gegenüber einer dritten Partei nachzuweisen. In diesem Fall soll der Abruf der Daten vom Webserver per TLS gesichert und *zusätzlich* überprüfbar sein. Daher weist oraclize.it beim Abruf des Servers gegenüber einem Auditor nach, dass Abruf und Antwort so stattgefunden haben, wie sie später in der Callback-Funktion an den Vertrag weitergeleitet werden.

Wenn ein TLSNotary-Proof beim Orakel-Aufruf angefordert wird, ruft das Orakel eine Callback-Funktion mit der Signatur `function __callback(bytes32 myid, string result, bytes proof)` auf. Zu-

sätzlich zu den schon bekannten Parametern kommt der `proof`. Dieser Parameter enthält den IPFS-Multihash, unter dem der eigentliche Proof angerufen und gegen den Auditor verifiziert werden kann.

Im Fall von oraclize.it ist das Orakel der sogenannte „Auditee“, also die Partei, die belegen möchte, dass der Abruf so stattgefunden und das Ergebnis geliefert hat. Der „Auditor“, also die Partei, gegenüber der das Orakel das belegen möchte, ist der Vertrag. Allerdings beruht der Mechanismus von TLSNotary darauf, dass während der Kommunikation ein Teil der normalen TLS-Kommunikation zurückgehalten wird. Da ein Vertrag in der Blockchain keine Geheimnisse haben kann, kann der Smart Contract kein Auditor sein. Oraclize.it verwendet dafür eine bei Amazon gehostete, speziell gesicherte AWS-VM. In ihr wird das sogenannte Secret gehalten, sie fungiert als Auditor.

Das Orakel weist mit TLSNotary nach, dass ein Auditor die Kommunikation zwischen Server und Orakel notarisiert. Dem Auditor selbst muss der Nutzer allerdings vertrauen. TLSNotary-Proof zu verwenden, ist besser, als dem Orakel ohne Nachweis zu vertrauen. Wer allerdings Zugang zur Amazon-VM bekäme, könnte mit dem Secret der AWS-VM die TLSNotary-Proofs fälschen. Um Abruf und Antwort des Orakel-Dienstes zu fälschen, bräuchte man zusätzlich Zugang zu oraclize.it. Der TLSNotary-Proof ist damit theoretisch fälschbar – aber immerhin doppelt gesichert.

Andere Schlüssel zur Wahrheit

Oraclize.it ist nicht der einzige Orakel-Dienst in der Ethereum-Blockchain. Einen anderen Ansatz benutzen *Reality Keys* [Reality Keys]: Dieser Dienst registriert das zu überprüfende Ereignis zuerst selbst. Bei der Wette auf ein Fußballspiel ruft ein Nutzer zuerst die Programmierschnittstelle von Reality Keys auf, die ein Ereignis registriert, das das Spiel und Ergebnis abbildet. So könnte man registrieren, dass Borussia Dortmund im nächsten Bundesligaspiel gewinnt (genauer: mehr Tore schießt als die gegnerische Mannschaft). Der Dienst erstellt zwei „Reality Keys“ (RK), die für die beiden möglichen Ergebnisse stehen: einen „Ja“-RK für einen Dortmund-Gewinn, und einen „Nein“-RK für den gegnerischen Gewinn oder ein Unentschieden. Der Dienst veröffentlicht die öffentlichen Schlüssel beider RK und einen Hashwert eines Fakts, über den es aus Ethereum-Verträgen geprüft werden kann.

Bei Ablauf des Ereignisses prüft der Dienst das Ereignis und legt das Ergebnis fest. Für den RK, der das Ereignis abbildet, veröffentlicht er nun zusätzlich den privaten Schlüssel – bei einem Dortmund-Sieg ist das der private Schlüssel des „Ja“-RK. Der andere private Schlüssel wird nicht veröffentlicht. Zusätzlich trägt der Dienst für Ethereum das Ergebnis im zuvor spezifizierten Fakt ein und signiert es. Verträge in der Ethereum-Blockchain können anhand der Signatur prüfen, ob der Dienst „Reality Keys“ das Fakt nach Eintrag des Ergebnisses signiert hat. Wenn das der Fall ist, verfährt der Vertrag auf Basis des eingetragenen Ergebnisses weiter.

Die Vertrauenskette zwischen Vertrag, Orakel und Server ist ähnlich wie bei oraclize.it: Der Benutzer muss dem Server und dem Dienst Reality Keys vertrauen, dass er das richtige Ergebnis einträgt und veröffentlicht. Um dem Nutzer zumindest eine Einspruchsmöglichkeit zu geben, wird jedem Ereignis eine Berufungsinstanz eingeräumt. Wenn der Nutzer mit der automatisierten Festlegung des Ergebnisses nicht einverstanden ist, kann er innerhalb eines definierten Zeitraums gegen eine Gebühr einen manuellen Check anfordern, dessen Ergebnis dann bindend ist.

Fazit

Smart Contracts bieten die Möglichkeit, beliebige Berechnungen in der Blockchain durchzuführen. Oft sind sie auf Daten außerhalb der Blockchain angewiesen. Da ein Vertrag diese Daten nicht einfach selbst abrufen kann, haben sich Orakel-Dienste entwickelt. Oraclize.it ist der bekannteste Dienst für die Ethereum-Blockchain. Er lässt sich verhältnismäßig einfach in Verträge einbinden.

Die Vertrauenskette wird dabei jedoch unterbrochen: In der Blockchain sind alle Daten transparent und verifizierbar, für externe Daten gilt das nicht. oraclize.it weist per TLSNotary nach, dass es die Daten zwischen Datenquelle und Vertrag nicht verändert hat. Aus technischen Gründen belegt das Orakel das gegenüber einer bei AWS gehosteten VM. Der Benutzer muss damit nicht dem Anbieter des Orakel-Dienstes vertrauen, sehr wohl allerdings der Tatsache, dass niemand in Besitz des Secrets in der AWS-Instanz gekommen ist.

Andere Orakel für die Ethereum- oder die Bitcoin-Blockchain wählen einen anderen Ansatz. Reality Keys ist in der Einbindung nicht so weit fortgeschritten wie oraclize.it. Auch weist das Orakel nicht nach, dass es aufrichtig gehandelt hat. Das Orakel von Reality Keys eignet sich trotzdem für einfache Faktenchecks, da viele Parameter schon im zu registrierenden Fakt festgelegt werden. Das Ergebnis (*true* oder *false*) ist dann im Ethereum-Vertrag einfach auszuwerten.

Orakel-Dienste sind damit eine Möglichkeit, Daten von außerhalb der Blockchain in der Blockchain zu verarbeiten. Da die allermeisten Dienste heute nicht *blockchain aware* sind, eröffnen die Orakel den Smart Contracts viele Möglichkeiten. Aus Vertrauensgründen wären jedoch Ansätze vorzuziehen, in denen die Dienste die Daten direkt in die Blockchain übergeben. Bis dahin – und damit wohl auf absehbare Zeit – überbrücken die Orakel die Kluft zwischen Blockchain und externer Welt.

Links

[FootballBet] Beispielcode, `https://github.com/chrisdotn/footballBet`
[oraclize.it] `http://www.oraclize.it/`
[Reality Keys] `https://www.realitykeys.com/`
[Solidity] `http://solidity.readthedocs.io/en/latest/`
[TLSNotary] `https://tlsnotary.org/`

Christoph Niemann ist IT-Consultant bei MaibornWolff. Neben der Blockchain-Technologie beschäftigt er sich mit agilen Themen.
E-Mail: christoph.niemann@maibornwolff.de