Einstieg in die Public-Ledger-Entwicklung mit NEM

In signo veritas

Grégory Saive

Die Blockchain-Implementierung NEM eignet sich wegen ihrer JavaScript-API gut für eigene Experimente. Ein kleines Beispiel soll Appetit auf mehr machen.

iX-TRACT

- NEM ist eine Distributed-Ledger-Implementierung, die das einfache Aufsetzen einer eigenen Blockchain erlaubt.
- Durch die JSON-Node.js-API ist die Programmierung von NEM-Anwendungen einem großen Kreis von Entwicklern möglich.
- Erforderlich sind Basiskenntnisse in Kryptografie.

NEM, das 2015 in Asien gestartete New Economy Movement, hat mit seiner Währung XEM mittlerweile eine Marktkapitalisierung von knapp 10 Milliarden US-Dollar erreicht und damit Platz 8 der Kryptowährungen.

NEMs Blockchain-Verfahren unterscheidet sich in einigen wichtigen Punkten von der Bitcoin-Technik. Es beruht zum einen nicht auf einem Proof-of-Work-, sondern auf einem Proof-of-Importance-Verfahren und ist damit nicht so ressourcenvergeudend wie Bitcoin (siehe Kasten „Proof of …“). Und es gibt eine JavaScript-API, die einen leichten Einstieg in eigene Blockchain-Experimente ermöglicht.

So landete auch der Autor dieser Zeilen auf der Suche nach einer leicht zugänglichen Einstiegsbasis für Entwickler nach einer langen Reise durch White Papers, Quellcode und andere Onlinedokumentationen bei NEM. Das Nutzen der NEM-Bibliothek ist unkompliziert durch eine RESTful HTTP/JSON-API möglich. Man kann sich zudem eine eigene, „customized“ Blockchain erstellen. Es braucht dazu wenig mehr als etwas Grundwissen über Peer-to-Peer-Netzwerke und ein bisschen Spaß an der Kryptografie (siehe Kasten „PacNEM“).

Doch genug der Vorrede. Um die Technik etwas konkreter werden zu lassen, zunächst ein Blick auf die NEM-Blockchain.

Der letzte Block der öffentlichen NEM-Blockchain lässt sich über die REST-API im Browser auslesen:

```
http://hugealice.nem.ninja:7890/chain/last-block
```

hugealice.nem.ninja ist NEMs Blockchain-Explorer. Der Aufruf gibt ein JSON-Objekt zurück. Ganz unten erscheint die **Height** der Blockchain, die aktuelle Anzahl der Blöcke. Außerdem sieht man den Hash des nächstälteren Blocks, Zeitstempel für die Generierung des Blocks, die Signatur des Miners sowie verschiedene Transaktionen.

Transaktionen mit NEM

Diese Transaktionen sind von jedem Netzwerkknoten validiert und werden über P2P-Protokolle verteilt. Für die aktive Teilnahme an einer Blockchain, also das Generieren neuer Blöcke, muss ein Knoten die vollständige Kopie der Blockchain-Datenbank herunterladen. Dieser Prozess ermöglicht eine sichere Kommunikation, denn es gibt keine einzelne En-

tität, die das Validieren einer Transaktion zensieren oder blockieren könnte.

Bei diesen Transaktionen muss es nicht unbedingt um Zahlungen gehen, sondern zum Beispiel auch um das Verteilen von Software, Eigentumsbeweise durch Zeitstempel, Beglaubigungen von Dokumenten et cetera. Oder gar ein Multiplayer-Browser-Pacman-Spiel, siehe Kasten „PacNEM“.

Als Illustration soll hier die Implementierung eines einfachen Eigentumsbeweises für ein beliebiges Dokument auf der NEM-Blockchain dienen. Das lässt sich mit dem NEM-SDK für Node.js einfach erledigen. Konkret soll die Software einen öffentlich einsehbaren Beweis dafür liefern, dass ein Dokument mit einem ausgewählten NEM-Account verknüpft wurde. Dadurch ist es möglich, die Integrität des Dokuments nachzuvollziehen. Für solche Programmierübungen steht eine andere NEM-Blockchain zur Verfügung, nämlich bigalice2.nem.ninja:7890.

NEM benutzt asymmetrische Kryptografie mit den bekannten Public-Private-Key-Paaren. Der private Schlüssel dient dem Signieren von Daten, der öffentliche Schlüssel der Überprüfung der Signatur. Beim Verschlüsseln funktioniert es anders herum: Man verschlüsselt mit dem öffentlichen Schlüssel des Empfängers, der den Code mit seinem privaten Key wieder entschlüsseln kann.

Zum Nachvollziehen dieses Artikels muss das JavaScript-Framework Node.js auf dem Rechner vorhanden sein, dann lässt sich durch den folgenden Befehl das NEM-SDK installieren:

```
npm install nem-sdk
```

Nun in Node.js den JavaScript-Anwendungen das NEM-SDK bekannt machen:

```
node
> let sdk = require("nem-sdk").default;
```

Jetzt könnte man seine erste NEM-Adresse erstellen. Eine NEM-Adresse ist eine vereinfachte, menschenlesbare Ableitung des öffentlichen Schlüssels. Ope-

Proof of …

Die Grundidee des Blockchain-Verfahrens ist, dass jeder neu angefügte Block der Kette den Hashwert seines Vorgängers enthält, der wiederum schon den Hashwert seines eigenen Vorgängers enthält. So kann man den Inhalt eines Blocks nur manipulieren, indem man die ganze folgende Kette neu schreibt.

Da das mit aktueller Rechnertechnik machbar ist, braucht es eine irgendwie geartete Hürde gegen diese Manipulationsmöglichkeit.

Bitcoin, die bekannteste Blockchain-Implementierung, nutzt das sogenannte **Proof-of-Work**-Verfahren. Es wird nicht jeder Hashwert akzeptiert, sondern er darf eine bestimmte Größe nicht überschreiten. Dazu verändert man ein bestimmtes Feld im Ausgangsblock, die sogenannte Nonce, so lange, bis der Hashwert stimmt. Abstrakter ausgedrückt: Man fordert einen sinnfreien Fleißbeweis, der natürlich Zeit und damit Geld respektive Energie kostet.

Eine Alternative ist ein **Proof-of-Stake**-Verfahren. Wer einen neuen Block anfügen will, muss dazu, etwa durch Besitz der entsprechenden Kryptowährung oder durch Aufnahme in einen Kreis von Stakeholdern, autorisiert sein.

NEM hingegen nutzt ein Verfahren namens **Proof of Importance**. Man muss, um überhaupt einen neuen Block anfügen zu können, 10 000 XEM mindestens 10 Tage lang gehalten haben. Außerdem wird ein Score vergeben, der sich mit jeder Transaktion ab einer bestimmten Größenordnung erhöht.

PacNEM, ein Blockchain-Pacman

PacNEM war das erste Distributed-Ledger-Projekt des Autors und eines langjährigen Arbeitskollegen. Es belohnt den Spieler mit einer hausgemachten Kryptowährung, den pacnem: cheese-Coins, die er für seinen Highscore auf seine NEM-Wallet überwiesen bekommt.

Unter anderem demonstriert PacNEM exemplarisch die Manipulationssicherheit einer Blockchain. So war der Autor bei PacNEM seit den ersten Wochen nicht mehr führend im Top-10-High-Score. Doch obwohl er als Administrator Zugang zur PacNEM-Blockchain hatte, konnte er die Scores in der Liste nicht mehr ändern, denn das PacNEM-Spiel validiert sie semiautonom.

Semiautonom, weil im Hintergrund eine MongoDB läuft. Die dient aber nur als Zwischenspeicher, da das PacNEM-Spiel selbst die ausgehenden Transaktionen bestätigt und alle Daten von NEM-Blockchain-Knoten liest.

Das Spiel ist recht einfach gehalten, es diente dem Autor als Fingerübung in Sachen Distributed-Ledger-Programmierung und als Beispiel für verschiedenste Features von NEM (URL siehe ix.de/ix1807056).

rative Adressen der NEM-Blockchain beginnen immer mit dem Buchstaben N, die letzten vier Zeichen sind eine Prüfsumme für den Rest der Adresse. Zum Ausprobieren sollte man auf Adressen der NEM-Testnet-Blockchain zurückgreifen, die mit einem T beginnen.

Gestartet wird mit dem Generieren eines privaten Schlüssels. Dazu dient die CryptoJS-Library, die Teil des NEM-SDK ist. Beim Erzeugen wird auch der NEM-Netzwerkknoten und die benutzte Netzwerk-ID angegeben. Letztere ist -104 für das Testnet oder 104 für das Mainnet (die Live-Blockchain). Die weiteren Angaben im Artikel sind Befehlszeilen in Node.js.

Aber: Dazu bräuchte es ein initiales virtuelles XEM-Guthaben auf dieser Blockchain. Für einen leichteren Einstieg haben wir diesen Schritt schon einmal erledigt und ein paar Test-XEM auf einen privaten Account einbezahlt, der im Folgenden genutzt wird.

```
let endpoint = {
'host': 'http://bigalice2.nem.ninja',
'port': 7890
};
let networkId = -104;
let privateKey = "dd19f3f3178c0867771eed 7
    180310a484e1b76527f7a271e3c8b5264e4a5aa414";
// let privateKey = sdk.crypto.js.lib. 7
                WordArray.random(32).toString();
```

Ohne vorgegebenen Key käme die auskommentierte Zeile zum Einsatz.

Aus diesem tunlichst vertraulich zu haltenden privaten Schlüssel lässt sich ein öffentlicher Schlüssel nebst Adresse erstellen:

```
let kp = sdk.crypto.keyPair.create(privateKey);
let publicKey = kp.publicKey.toString();
let address = sdk.model.address.toAddress 7
                        (publicKey, networkId);
```

Die Zuordnung ist eindeutig, man kann die letzten Befehlszeilen beliebig oft wiederholen. Ansehen kann man sich das Ergebnis mit:

```
console.log(address);
> TA3SH7QUTG6OS4EGHSES426552FAJYZR2PHOBLNA
```

An diese NEM-Wallet-Adresse könnte man jetzt beispielsweise XEM schicken, zuvor müsste man den privaten Schlüssel in NEMs NanoWallet importieren. Aber darum sollte es ja gar nicht gehen. Sinn der Generierung der ersten Adresse war deren Nutzung für ein Programmierprojekt.

Daten auf die Blockchain

Auf der Blockchain soll eine einfache Textdatei mit dem Inhalt „Diese Datei gehört dem *iX*-Leser Grégory Saive“ fälschungssicher gespeichert und ihrem Eigentümer zugeordnet werden.

Dazu ist mit der generierten Adresse eine Transaktion zu erstellen, die eine NEM-Adresse mit einem bestimmten Dokument verlinkt. Beide werden mit dem privaten Schlüssel signiert, der Inhalt mit einem SHA512-Hash festgehalten:

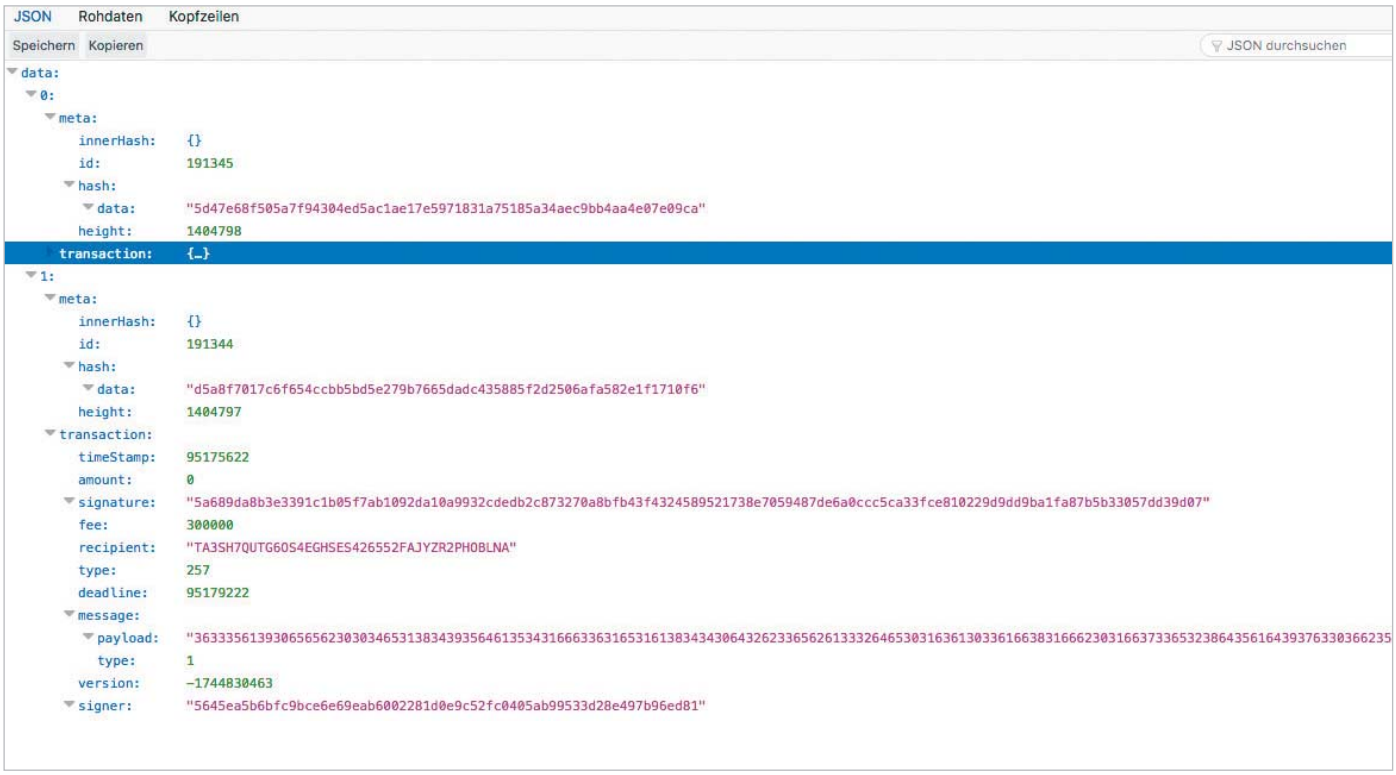
```
let data = `Diese Datei gehört 7
                    dem iX-Leser Grégory Saive.`
let hash = sdk.crypto.js.SHA512(data);
```

Und anschließend als Eigentumsbeweis signiert:

```
let keypair = sdk.crypto.keyPair. 7
                            create(privateKey);
let signed = keypair.sign(hash.toString());
```

Ergebnis ist ein 64 Zeichen langer Hex-Code, also ein 32-Byte-Hash. Dieser

Die NEM-Blockchain ist öffentlich einsehbar – hier der letzte Block, erreichbar über die URL http://hugealice.nem.ninja:7890/chain/last-block (Abb. 1).



Details einer Transaktionsabfrage in NEM (Abb. 2)

Hash verbindet Dokumenteninhalt und Eigentümer, denn nur Letzterer ist in der Lage, ihn zu erstellen.

Nun gilt es, die erstellten Daten in eine Transaktion einzubinden. Die im Folgenden genutzte Adressvariable ist die oben generierte eigene Adresse. Schritt 1:

```
let trx =
sdk.model.objects.create ↵
            ('transferTransaction')(address,
0, signed);
```

Auch die Transaktion muss signiert werden, damit die NEM-Netzwerkknoten sie validieren.

Das vorbereitete Objekt wird im Anschluss mit dem privaten Schlüssel verknüpft:

```
let creds = {password: ,', privateKey:
privateKey};
let prepared = sdk.model.transactions. ↵
               prepare('transferTransaction') ↵
                       (creds, trx, networkId);
```

Schlussendlich fehlt nur noch der P2P-Broadcast zum Verteilen im Netz (das vollständige Listing finden Sie unter ix.de/ix1807056):

```
let result =
sdk.model.transactions.send(creds, prepared,
endpoint);
```

Da nun die Transaktion verteilt ist, wird sie nach der Bestätigung im nächsten Block der NEM-Blockchain angezeigt. Doch zunächst gilt sie als „unconfirmed“ (nicht bestätigt). Auch solche unbestätigte Transaktionen sind bei NEM abfragbar:

```
http://bigalice2.nem.ninja:7890/account ↵
            /unconfirmedTransactions?address=↵
       TA3SH7QUTG6OS4EGHSES426552FAJYZR2PHOBLNA
```

Transaktion in der Blockchain

Ist bis hierhin alles fehlerfrei verlaufen, muss die Transaktion kurzzeitig in dieser Liste zu finden sein. Nach etwa einer Minute sollte sie auch im nächsten Block der NEM-Blockchain integriert, also bestätigt sein. Überprüfbar ist das mit:

Jede Transaktion in einem Block ist überprüfbar (Abb. 3).

```
http://bigalice2.nem.ninja:7890/account/ ↵
                transfers/incoming?address= ↵
      TA3SH7QUTG6OS4EGHSES426552FAJYZR2PHOBLNA
```

Dies ist die Liste aller eingehenden Transaktionen von TA3SH7QUTG6OS4EGHSES426552FAJYZR2PHOBLNA. Ersetzt man das „incoming" durch „outgoing", bekommt man die ausgehenden Transaktionen angezeigt. Da die Transaktion von der eigenen an die eigene Adresse ging, kommt sie in beiden Listen vor.

Zum besseren Verständnis seien die ausgegebenen Daten erläutert. Metadaten einer Transaktion beinhalten unter anderem die Block-"Höhe" des Blocks, in der die Transaktion bestätigt wurde, und den Transaktions-Hash zusammen mit der Transaktions-ID. Die beiden Felder dienen der Nachvollziehbarkeit der Historie von Transaktionen. Sie sehen im JSON-Format ungefähr wie folgt aus:

```
""meta": {
  "innerHash": {},
  "id": 191345,
  "hash": {
   "data": "5d47e68f505a7f94304ed5ac ↵
      1ae17e5971831a75185a34aec9bb4aa4e07e09ca"
  },
  "height": 1404798
}
```

Das leer gebliebene *innerHash*-Feld wird nur bei Multi-Signatur-Transaktionen verwendet und soll erst einmal nicht weiter interessieren. Die Felder *id, hash* und *height* identifizieren eine Transaktion auf der NEM-Blockchain.

Anhand des Hash einer Transaktion ist es möglich, eine API-Anfrage bezüglich des Inhalts zu stellen:

```
http://bigalice2.nem.ninja:7890/transaction/ ↵
          get?hash=5d47e68f505a7f94304ed5ac1 ↵
        ae17e5971831a75185a34aec9bb4aa4e07e09ca
```

Das Ergebnis enthält im JSON-Format unter anderem:

```
""timeStamp": 95175808,
  "deadline": 89366634,
```

Der *timeStamp* wird in NEM-Zeit ausgedrückt, der Anzahl Sekunden, die seit der Genesis von NEM, dem Generieren des ersten Blocks, vergangen sind (und differiert bei eigenen Versuchen natürlich von den hier angegebenen Werten).

deadline ist die späteste Zeit für die Bestätigung der Transaktion, danach wird sie verworfen. Diese beiden Felder müssen nur in seltenen Fällen vom Benutzer gesetzt werden. Weiter geht es mit:

```
""amount": 0,
  "fee": 300000,
  "recipient": "TA3SH7QUTG6OS4EGHSES ↵
                        426552FAJYZR2PHOBLNA",
```

Nichts ist umsonst

Auch wenn das Feld *amount* auf 0 steht, ist es Teil jeder Transaktion, ausgedrückt ganzzahlig in Micro-XEM. Auch die Gebühr *(fee)* wird in Micro-XEM gezahlt, hier beträgt sie 0,3 XEM (1 XEM entsprach im März 2018 0,25 Euro).

Ein Empfänger ist unverzichtbar und steht im *recipient*-Feld. Beim Transaktionstyp *(type)* geht es um die Art der Transaktion. Enthält eine Transaktion Textdaten, werden sie nicht als Klartext gespeichert, sondern in Form einer hexadezimal dargestellten Payload, die man auch verschlüsselt speichern kann, sodass nur Absender und Empfänger die Nachricht lesen können.

In einer NEM-Transfertransaktion wird ein *message*-Feld mit einem Objekt mit zwei Feldern, *payload* und *type,* angegeben. Bei einem Inhalt von 2 anstatt von 1 im *type*-Feld ist die Nachricht verschlüsselt. Die Payload enthält die eigentliche Nachricht. Folgendes ist die hexadezimal dargestellte Payload für den oben angegebenen Text:

```
""message": {
"payload": "66...",
"type": 1
  },
```

Die Felder *signature* und *signer* sind für die kryptografische Beweissicherung auf der NEM-Blockchain verantwortlich. Die Signatur einer Transaktion ist ausschließlich mit dem privaten Schlüssel erstellbar, dessen öffentlicher Schlüssel im *signer*-Feld steht. Diese beiden Felder werden als 64-Byte-Hexadezimal-Hash für die Signatur und als 32-Bytes-Hexadezimal-Hash für den öffentlichen Schlüssel des Unterzeichners *(signer)* angegeben:

```
""signature": "d52f...d902",
  "signer": "fbad...f9d0c"
}
```

Dadurch ist eine Identitäts- und Integritätsprüfung möglich.

Im richtigen Leben ist vor allem das Feld *message* von Interesse, die eigentliche Nachricht – beliebige Objekte, die niemals mehr verschwinden, da sie auf der NEM-Blockchain nicht zu löschen sind.

Blockchain und Performance

Ein Peer-to-Peer-Netzwerk wie die NEM-Blockchain ist keine Allzweckwaffe, wie das in der öffentlichen Diskussion gerne vermittelt wird.

So ist auf der NEM-Blockchain pro Transaktion eine bestimmte Gebühr zu bezahlen, abhängig von Transaktionstyp und Inhalt. Diese Gebühr von 0,05 XEM pro angefangenem 31-Zeichen-Block wäre etwa für einen Chat reichlich teuer, aber nicht für einen Kaufvertrag einer Immobilie. Und vielleicht auch nicht für die Verifikation eines Lieferscheins bei B2B-Geschäften. Der Fantasie sind da zwar keine Grenzen gesetzt, aber Zweck, Mittel und Kosten müssen in einem vernünftigen Verhältnis zueinander stehen. (js@ix.de)

Grégory Saive

ist Software- und Blockchain-Entwickler.

Alle Links: ix.de/ix1807056