Praxissession 6 - Layer 2 Skalierung PaymentChannels

Einfacher unidirektionaler Paymentchannel

> In der heutigen Session werden Sie einen einfachen, unidirektionalen Paymentchannel implementieren und testen.

Smart Contract des Paymentchannels

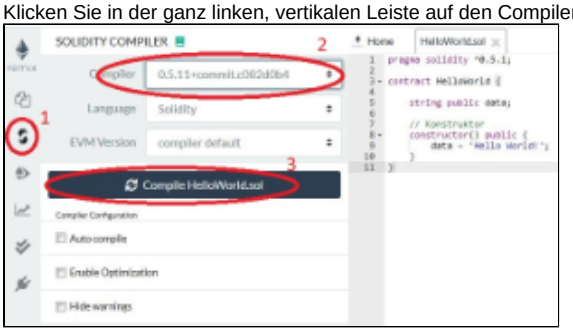
PaymentChannel.sol

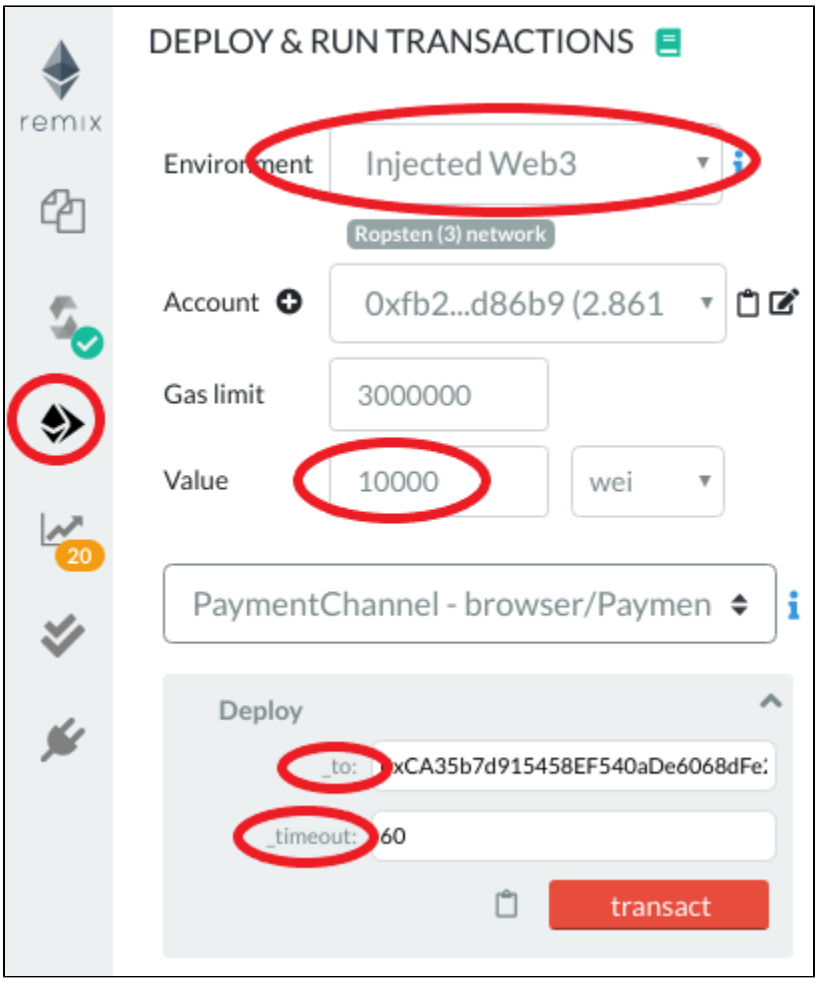
```
pragma solidity ^0.5.0;

contract PaymentChannel {

    address payable public channelSender;
    address payable public channelRecipient;
    uint public startDate;
    uint public channelTimeout;

    constructor(address payable _to, uint _timeout) public payable {
        channelRecipient = _to;
        channelSender = msg.sender;
        startDate = now;
        channelTimeout = _timeout;
    }

    function checkTicket(
        bytes32 _h,
        uint8 _v,
        bytes32 _r,
        bytes32 _s,
        uint _value
    )
    public view
    returns (bool)
    {
        // check message
        require(_h == keccak256(abi.encodePacked(_value)));

        // check funds
        require(_value <= address(this).balance);

        // check signatures
        _h = keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", _h));
        require(ecrecover(_h, _v, _r, _s) == channelSender);

        // checks completed successfull
        return true;
    }

    function payRecipient(
        bytes32 _h,
        uint8 _v,
        bytes32 _r,
        bytes32 _s,
        uint _value
    )
    public
    {
        // only recipient
        require(msg.sender == channelRecipient);

        // check ticket
        require(checkTicket(_h,_v,_r,_s,_value));

        // transfer funds
```

```
        channelRecipient.transfer(_value);

        // close channel and transfer remaining funds to sender
        selfdestruct(channelSender);
    }

    function closeChannel() public {
        // only Sender
        require(msg.sender == channelSender);

        // check if timeout is reached
        require(startDate + channelTimeout < now);

        // close channel and transfer remaining funds to sender
        selfdestruct(channelSender);
    }
}
```

Vertrag in die Blockkette laden

> Deployen Sie den PaymentChannel nun im Ethereum Testnetzwerk **Ropsten** unter Verwendung der Remix IDE und Meta Mask so wie sie es bereits in Praxissession 2a - Ethereum Grundlagen gelernt haben.

- Öffnen Sie die Seite https://remix.ethereum.org
- Wählen Sie die Sprache **Solidity** aus und klicken Sie auf "**New File**" und wählen Sie einen Dateinamen z. B.: **PaymentChannel.sol**
- 
- Kopieren Sie den Quelltext des Paymentchannels von ob in das Editorfenster.
- Klicken Sie in der ganz linken, vertikalen Leiste auf den Compiler und kompilieren Sie den Paymentchannel Contract.
- 
- Wenn der Vertrag fehlerfrei kompiliert werden konnte, klicken Sie in der vertikalen Leiste auf den Deployer:
  - Wählen Sie unter **Environment: Injected Web3** (da wir MetaMask verwenden)
  - Geben Sie unter **Value** den Betrag ein, mit dem der Channel "aufgeladen werden soll" z. B.: 1000 Wei
  - Klappen Sie die Felder neben dem **Deploy Button** aus.
  - Tragen Sie unter **_to** den Empfänger ein. Dazu können Sie den gleichen Account verwenden, mit dem Sie den Contract erstellen.
  - Tragen Sie unter **_timeout** ein kleine Zeitspanne zum Testen ein z. B.: 60 (1Minute)
  - Klicken Sie schließlich auf **transact** und bestätigen Sie die Transaktion in MetaMask.



Test Dapp

Nun werden wir eine kleine Dapp zum Testen des PaymentChannels implementieren. Die Vorgehensweise ist dabei analog zur Praxissession 4 - Grundlagen Dapp Entwicklung. Die Anleitung geht davon aus, dass Sie diese Session erfolgreich abgeschlossen haben.

Um auf die Blockchain zugreifen zu können, wollen wir wieder https://infura.io verwenden. Wenn Sie noch keinen Infra-API Key haben, legen Sie sich bitte einen neuen analog zur Anleitung in Praxissession 4 - Grundlagen Dapp Entwicklung an.

1. Öffnen Sie ein neues Konsolenfenster und navigieren Sie in den Ordner ihrer Dapps.

Terminalfenster / Konsole

```
$ cd ~/mydapps
```

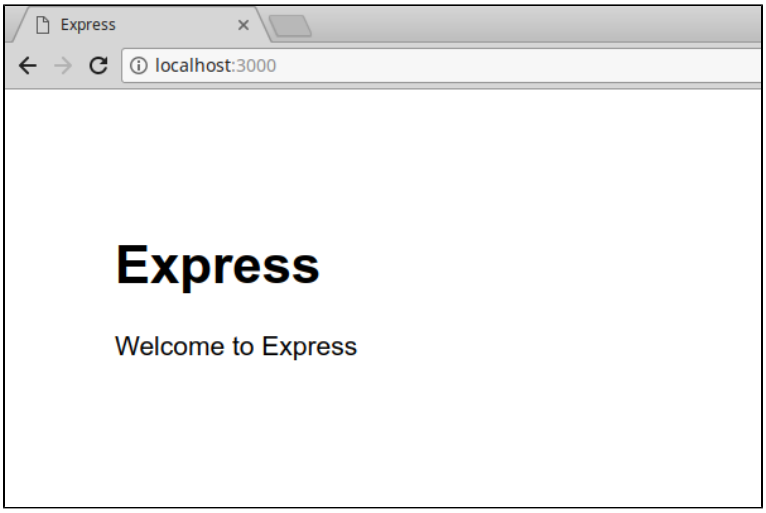
2. Legen Sie das Grundgerüst für eine neue Dapp an und installieren Sie die Dependencies.

```
$ ./node_modules/.bin/express --view=pug pcdapp
$ cd pcdapp
$ npm install
$ npm install web3
```

3. Starten Sie den Webserver:

```
$ npm start
```

4. Test Sie die neue Dapp, indem Sie im Browser http://localhost:3000 aufrufen.



Dapp mit dem Payment Channel verbinden

1. Öffnen Sie den **Komodo Editor** und navigieren Sie in den Ordner **/mydapps/pcdapp**.
2. Passen Sie nun die Datei **routes/index.js** an**.** Achten Sie dabei darauf:

   in **Zeile 5** **<infura-api-key>** durch ihren **Infura API Key** zu ersetzen und
   in **Zeile 9** die "**address**"' mit der Adresse des von Ihnen erstellten Payment Channels anzupassen. Diese können Sie aus der Remix IDE kopieren:

routes/index.js

```
var express = require('express');
var router = express.Router();

var Web3JS = require("web3");
var web3 = new Web3JS(new Web3JS.providers.HttpProvider("https://ropsten.infura.io/<infura-api-key>"));

var abi = [{"constant": false,"inputs": [],"name": "closeChannel","outputs": [],"payable": false,"
stateMutability": "nonpayable","type": "function"},{"constant": false,"inputs": [{"name": "_h","type":
"bytes32"},{"name": "_v","type": "uint8"},{"name": "_r","type": "bytes32"},{"name": "_s","type":
"bytes32"},{"name": "_value","type": "uint256"}],"name": "payRecipient","outputs": [],"payable": false,"
stateMutability": "nonpayable","type": "function"},{"inputs": [{"name": "_to","type": "address"},
{"name": "_timeout","type": "uint256"}],"payable": true,"stateMutability": "payable","type":
"constructor"},{"constant": true,"inputs": [],"name": "channelRecipient","outputs": [{"name": "","type":
"address"}],"payable": false,"stateMutability": "view","type": "function"},{"constant": true,"inputs":
[],"name": "channelSender","outputs": [{"name": "","type": "address"}],"payable": false,"
stateMutability": "view","type": "function"},{"constant": true,"inputs": [],"name": "channelTimeout","
outputs": [{"name": "","type": "uint256"}],"payable": false,"stateMutability": "view","type":
"function"},{"constant": true,"inputs": [{"name": "_h","type": "bytes32"},{"name": "_v","type": "uint8"},
{"name": "_r","type": "bytes32"},{"name": "_s","type": "bytes32"},{"name": "_value","type": "uint256"}],"
name": "checkTicket","outputs": [{"name": "","type": "bool"}],"payable": false,"stateMutability":
"view","type": "function"},{"constant": true,"inputs": [],"name": "startDate","outputs": [{"name": "","
type": "uint256"}],"payable": false,"stateMutability": "view","type": "function"}]

var address = '0x...';
var pc = new web3.eth.Contract(abi, address);

/* GET home page. */
router.get('/', function(req, res, next) {

  var balance;
  var recipient;
  var sender;
  var start;
  var timeout;
  var amount;

  web3.eth.getBalance(address).then(function(_balance){
    balance = web3.utils.fromWei(_balance,'Finney');
    return pc.methods.startDate().call();
  }).then(function(_start){
    start = _start;
    return pc.methods.channelSender().call();
  }).then(function(_sender){
    sender = _sender;
    return pc.methods.channelRecipient().call();
  }).then(function(_recipient){
    recipient = _recipient;
    return pc.methods.channelTimeout().call();
  }).then(function(_timeout){
    timeout = _timeout;
    res.render('index', { title: 'Paymentchannel Demo', e: null, recipient:recipient, sender:sender,
start:start, timeout:timeout, balance:balance });
  }).catch(function(error){
    console.log("E");
    res.render('index', { title: 'Paymentchannel Demo', e:error});
  });

});

module.exports = router;
```

3. Passen Sie nun die Datei **views/index.pug** zur Anzeige der Payment Channel Informationen an.

views/index.pug (die Einrückungen sind wichtig)

```
extends layout

block content
  h1= title
  div Payment Channel Daten
  hr
  div ChannelBalance: #{balance} Finney
  div ChannelRecipient: #{recipient}
  div ChannelSender: #{sender}
  div ChannelStart: #{start}
  div ChannelTimeout: #{timeout}
  div Fehler: #{e}
  hr
  div(id='info')
```

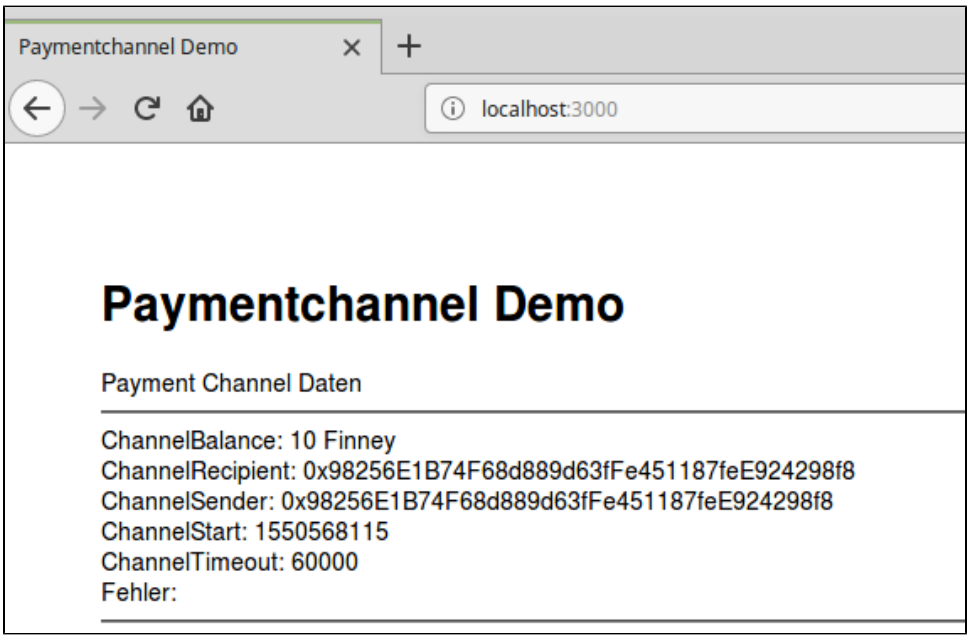
Da serverseitiger Code geändert wurde, muss der Webserver wieder neu gestartet werden. Beenden Sie den Webserver, in dem Sie **Strg+C** im Terminalfenster der laufenden Dapp drücken.

Starten Sie den Webserver neu:

Terminalfenster / Konsole

```
$ npm start
```

4. Aktualisieren Sie im Webbrowser anschließend die URL: http://localhost:3000:



Verbinden der Dapp mit einem Ethereum Account aus Meta Mask

Nun wollen wir die Dapp dahingehend anpassen, dass man sie mit Meta Mask benutzen kann.

1. Zunächst benötigen Sie die zwei JavaScript-Bibliotheken **web3.min.js** und **jquery-3.3.1.min.js** die Sie sich bitte in den Ordner **public /javascripts** kopieren.

   web3.min.js
   jquery-3.3.1.min.js

2. Erstellen Sie nun im **Komodo-Editor** im Ordner **mydapps/pcdapp/public/javascripts** eine neue Datei **client.js**.
3. Die Datei **client.js** übernimmt die Kommunikation mit **Meta Mask** und wird wie folgt angepasst:
4. Nun verlinken wir noch die zuvor kopierten Bibliotheken sowie die soeben erstelle Datei in die Ausgabe der Dapp. Dazu passen Sie bitte die Datei **views/layout.pug** wie folgt an.

public/javascripts/client.js

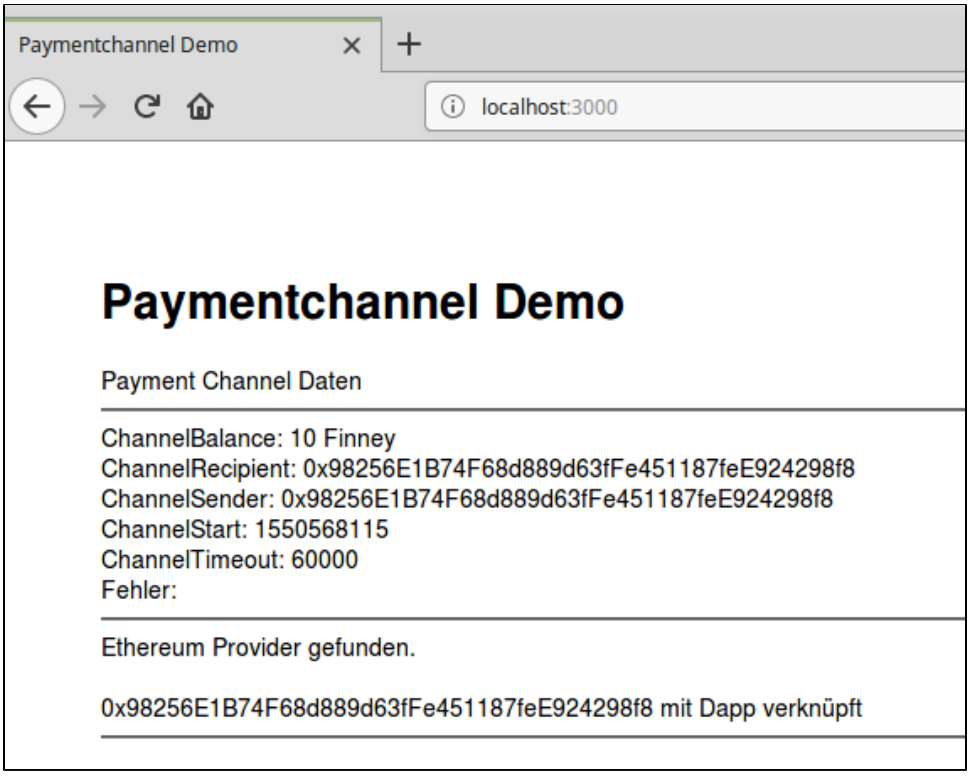
```
$(document).ready(function() {

    var account;

    if (typeof window.ethereum !== 'undefined') {

        // load web3 client library from metamask
        web3 = new Web3(ethereum);
        console.log("Web3 Api Version: " + web3.version);

        $('#info').html('Ethereum Provider gefunden.<br><br>');

        // enable metamask and load account
        ethereum.enable().then((a) => {
            account = a[0];
            $('#info').append(' ' + account + ' mit Dapp verknüpft');
        });

    } else {
        $('#info').html('Installieren Sie bitte <a href="https://metamask.io/">Meta Mask</a>.');
    }

});
```

view/layout.pug

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content

    script(src="/javascripts/jquery-3.3.1.min.js")
    script(src="/javascripts/web3.min.js")
    script(src="/javascripts/client.js")
```

5. Nun testen wir, ob die Dapp den Account aus MetaMask korrekt auslesen kann. Aktualisieren Sie dazu im Webbrowser die URL: http://localhost:3000.

   ChannelRecipient, ChannelSender und der verknüpfte Account sollen die gleichen Werte anzeigen.



Funktionalität zum Erstellen von gültigen "Tickets" ("Abhebeerlaubnis")

Nun wollen wir gültige "Tickets" bzw. "Abhebeerlaubnisse" erstellen. Sie bilden die Grundlage der Payment Channels. Diese Abhebeerlaubnisse garantieren dem Empfänger, dass er sich Geld aus dem Payment Channel holen kann, ohne dass eine kostenpflichtige Transaktionen auf der Blockkette durchgeführt werden muss.

1. Dazu passen Sie zunächst die Datei **views/index.pug** an.

views/index.pug

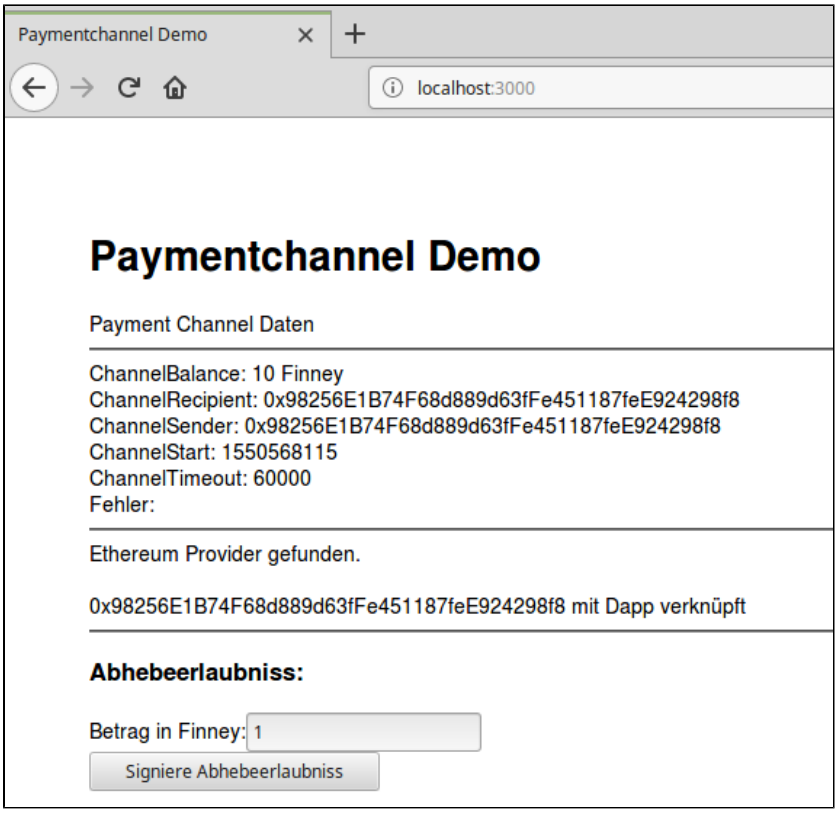
```
extends layout

block content
  h1= title
  div Payment Channel Daten
  hr
  div ChannelBalance: #{balance} Finney
  div ChannelRecipient: #{recipient}
  div ChannelSender: #{sender}
  div ChannelStart: #{start}
  div ChannelTimeout: #{timeout}
  div Fehler: #{e}
  hr
  div(id='info')
  hr
  h3 Abhebeerlaubniss:
  div Betrag in Finney:
    input(id='value' type='text')
  div
    input(id='createSig' type='button' value='Signiere Abhebeerlaubniss')
  div(id='ticketinfo')
```

2. Anpassen der Datei **public/javascript/client.js**

public/javascripts/client.js

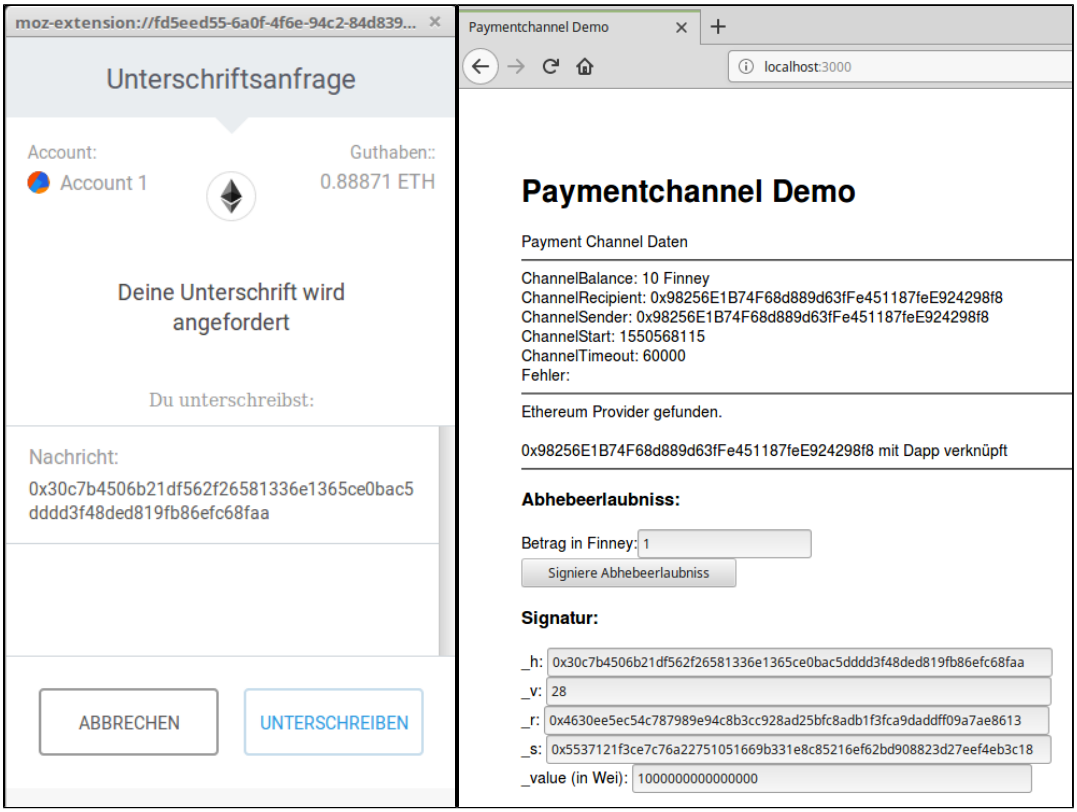
```
$(document).ready(function() {

    var account;

    if (typeof window.ethereum !== 'undefined') {

        // load web3 client library from metamask
        web3 = new Web3(ethereum);
        console.log("Web3 Api Version: " + web3.version);

        $('#info').html('Ethereum Provider gefunden.<br><br>');

        // enable metamask and load account
        ethereum.enable().then((a) => {
            account = a[0];
            $('#info').append(' ' + account + ' mit Dapp verknüpft');
        });

    } else {
        $('#info').html('Installieren Sie bitte <a href="https://metamask.io/">Meta Mask</a>.');
    }

    $('#createSig').on('click', function() {

        $('#ticketinfo').html('');

        var value = new web3.utils.BN(web3.utils.toWei($('#value').val(), 'finney'));
        var msg = web3.utils.soliditySha3(value);

        web3.eth.personal.sign(msg, account).then(function(sig) {

            var r = "0x" + sig.slice(2, 66);
            var s = "0x" + sig.slice(66, 130);
            var v = "0x" + sig.slice(130, 132);
            v = web3.utils.toDecimal(v);
            if (v == 0 || v == 1) v += 27;

            $('#ticketinfo').html('<h3>Signatur:</h3>_h: <input type="text" size="64" value="' + msg +
'"><br>_v: <input type="text" size="64" value="' + v + '"><br>_r: <input type="text" size="64" value="'
+ r + '"><br>_s: <input type="text" size="64" value="' + s + '"><br>_value (in Wei): <input type="text"
size="50" value="' + value + '">');

        });
    });

});
```

3. Aktualisieren Sie im Webbrowser anschließend die URL: http://localhost:3000:



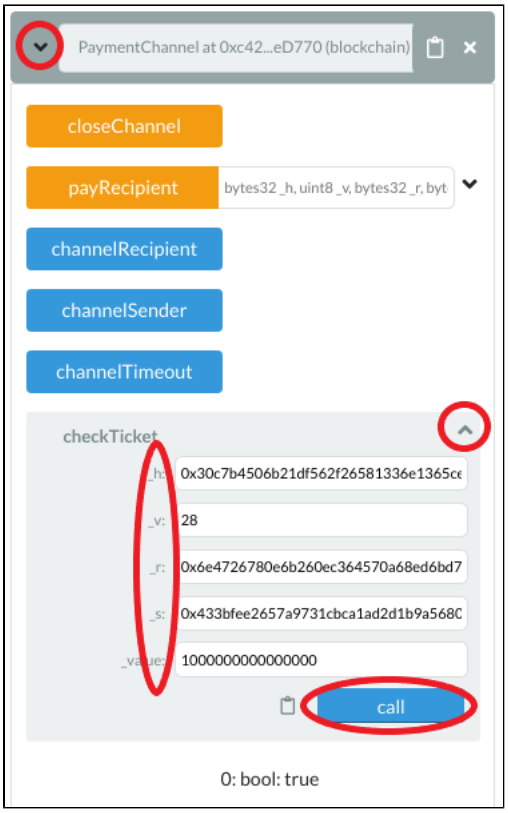
Erstellen und Testen von Tickets

1. Geben Sie in ihrer Dapp unter **Betrag in Finney** einen Betrag ein, der kleiner ist als die Balance des Payment Channel (z.B. **1 Finney**).
2. Klicken Sie auf **Signiere Abhebeerlaubniss** und bestätigen Sie die **Unterschriftsanfrage** des Meta Mask Plugins.



3. Öffnen Sie nun wieder die **Remix IDE** ihres Payment Channels.
4. Übertragen Sie die Werte ***_h, _v, _r, _s*** und ***value*** in die **checktTicket** Methode. Klicken Sie dazu auf den kleinen Pfeil, um die Eingabefelder anzuzeigen.

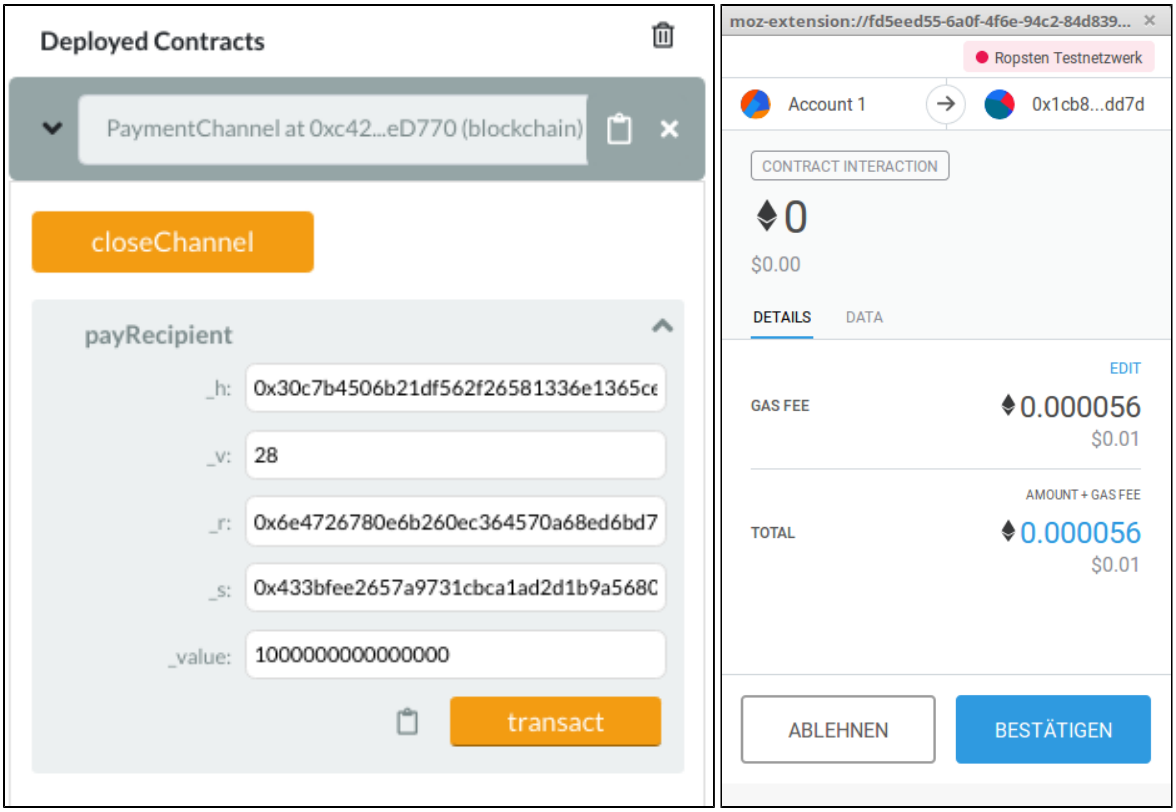
   Klicken Sie auf **call** und Sie sollten **true** als Rückgabewert erhalten.



Einlösen der Abhebeerlaubnis

1. Wenn die checkTicket Methode den Wert true zurück liefert, können wir nun den Channel schließen und den Empfänger entsprechend bezahlen.
2. Füllen Sie dazu in der **Remix IDE** die Methode **payRecipient** analog der checkTicket Methode und klicken Sie auf **transact.**

   Bestätigen Sie die Transaktion anschließend im **Meta Mask Pop Up**.



3. Nun kann es eine Weile dauern, bis die Transaktion gemined wurde. Sobald das der Fall ist, erhalten Sie eine Meldung in der **Konsole der Remix IDE**.

4. Schauen Sie sich die Transaktionen auf ihrem Payment Channel nun bei Etherscan an und prüfen Sie, ob der Channel korrekt geschlossen wurde.