



Konfiguration und Verwaltung.

Erweiterung des Ethereum-Austausches.

Benutzerhandbuch

Version 1.0.0 Beta

November 2020.

Blockoin.org ist ein eingetragenes Warenzeichen von Bankoin Inc, unter einer freien und kommerziellen Lizenz. Nutzungsbedingungen unter: www.CoinSolidation.org

Inhalt

1.	Einleitung.....	3
2.	Was ist blockweise Programmierung?	5
3.	Was ist eine Erweiterung?.....	5
4.	Was ist BlockoinEthereum und BlockoinINFURA?	6
5.	Grundlegende Konzepte, die in der Ethereum-Plattform angewendet werden.	6
6.	Installation und Konfiguration der Erweiterungs- und Ethereum-Testumgebung.	13
7.	Definition und Verwendung von Blöcken (generische Funktion)	21
8.	Funktionen und Veranstaltungen von Exchange Ethereum Extension (EEE).....	22
9.	Schritte zum Erstellen eines CryptoToken oder Cryptomoney-Tokens.	33
10.	Wie Sie einen neuen Vermögenswert oder Ihr Crypt-Token zum Verkauf anbieten können (Token ERC20).	34
11.	Berechnung von Standard-Transaktionskosten und Smart contract transaction.....	58
12.	Gebühren bei Coinsolidation.org	62
13.	Erstellen Sie Ihre Android-App (Exchange) in 15 Minuten.....	63
14.	Token COINSolidation.....	66
15.	Lizenzierung und Nutzung von Software.	67

1. Einleitung.

Ethereum ist eine Open-Source-Plattform, dezentralisiert im Gegensatz zu anderen Blockketten, Ethereum kann viel mehr tun. Es ist programmierbar, was bedeutet, dass Entwickler damit neue Arten von Anwendungen erstellen können. Seine digitale Währung heißt "Äther".

Diese dezentralisierten Anwendungen (oder "dapps") profitieren von den Vorteilen der Kryptomontage- und Blockkettentechnologie. Sie sind zuverlässig und vorhersehbar, was bedeutet, dass sie, sobald sie einmal in das Ethereum "geladen" sind, immer termingerecht laufen. Sie können digitale Vermögenswerte kontrollieren, um neue Arten von Finanzanwendungen zu erstellen. Sie können dezentralisiert sein, was bedeutet, dass sie nicht von einer einzigen Entität oder Person kontrolliert werden.

Gegenwärtig erstellen Tausende von Entwicklern auf der ganzen Welt Anwendungen auf Ethereum und erfinden neue Arten von Anwendungen, von denen Sie heute viele nutzen können:

- Krypto-Währungsportfolios, die es Ihnen ermöglichen, günstige, sofortige Zahlungen mit ETH- oder anderen Vermögenswerten zu tätigen
- Finanzanwendungen, die es Ihnen ermöglichen, Ihre digitalen Vermögenswerte zu leihen, auszuleihen oder zu investieren
- Dezentralisierte Märkte, die es Ihnen ermöglichen, digitale Bestände auszutauschen oder sogar "Vorhersagen" über Ereignisse in der realen Welt auszutauschen.
- Spiele, bei denen Sie Vermögen im Spiel haben und sogar echtes Geld gewinnen können.

Wie funktioniert der Äther?

Ether verwendet, wie andere Krypto-Währungen auch, ein gemeinsames digitales Buch, in dem alle Transaktionen aufgezeichnet werden. Sie ist öffentlich zugänglich, völlig transparent und lässt sich im Nachhinein nur sehr schwer ändern.

Dies wird als **Blockkette** bezeichnet, und sie wird durch den **Abbauprozess** aufgebaut.

Die Bergleute sind dafür verantwortlich, Gruppen von Äthertransaktionen zu "Blöcken" zu verifizieren und diese durch das Lösen komplexer Algorithmen zu kodieren. Diese Algorithmen können mehr oder weniger schwierig sein, um eine gewisse Konsistenz in der Blockverarbeitungszeit (etwa eine alle 14 Sekunden) zu gewährleisten.

Die neuen Blöcke werden dann mit der vorherigen Blockkette verknüpft und der betreffende Bergmann erhält eine **Belohnung, d.h.** eine feste Anzahl von Ätherplättchen. Normalerweise

sind dies 5 Äther-Einheiten, obwohl diese Zahl als variabel angesehen werden kann, wenn die Kryptenwährung weiter steigt.

Wie funktioniert Ethereum?

Die Ethereum-Blockkette ist der Bitcoin-Blockkette sehr ähnlich, aber ihre Programmiersprache ermöglicht es Entwicklern, Software zu erstellen, mit der Transaktionen verwaltet und bestimmte Ergebnisse automatisiert werden können. Diese Software ist als **intelligenter Vertrag** bekannt.

Wenn ein traditioneller Vertrag die Bedingungen einer Beziehung beschreibt, stellt ein intelligenter Vertrag sicher, dass diese Bedingungen erfüllt werden, indem er sie in einem Code schreibt. Dabei handelt es sich um Programme, die den Vertrag automatisch ausführen, sobald die vordefinierten Bedingungen erfüllt sind, wodurch die Verzögerung und die Kosten, die bei der manuellen Ausführung einer Vereinbarung entstehen, eliminiert werden.

Um ein einfaches Beispiel zu nennen: Ein Ethereum-Benutzer könnte einen intelligenten Vertrag erstellen, um eine bestimmte Menge Äther an einem bestimmten Datum an einen Freund zu schicken. Sie würden diesen Code in den Blockstring schreiben, und wenn der Vertrag abgeschlossen ist (d.h. wenn das vereinbarte Datum erreicht ist), wird der Äther automatisch verschickt.

Diese Grundidee lässt sich auch auf komplexere Konfigurationen anwenden, und ihr Potenzial ist wahrscheinlich unbegrenzt, mit Projekten, die in Sektoren wie Versicherungen, Immobilien, Finanzdienstleistungen, Rechtsdienstleistungen und Mikrofinanzierung bereits bemerkenswerte Fortschritte gemacht haben.

Intelligente Verträge haben auch einige zusätzliche Vorteile:

1. Sie eliminieren die Figur des Vermittlers, bieten dem Benutzer totale Kontrolle und minimieren zusätzliche Kosten.
2. Sie werden registriert, verschlüsselt und in der öffentlichen Blockkette vervielfältigt, wo alle Benutzer das Marktgeschehen sehen können.
3. Eliminieren Sie den Zeit- und Arbeitsaufwand für manuelle Prozesse

Natürlich sind intelligente Verträge noch ein sehr neues System mit vielen Details, an denen noch gefeilt werden muss. Der Code wird wörtlich übersetzt, so dass Fehler bei der Vertragserstellung zu unerwünschten Ergebnissen führen können, die nicht geändert werden können.

DApps vs. intelligente Verträge

Intelligente Verträge weisen Ähnlichkeiten mit **DApps** (dezentralisierte Anträge) auf, trennen sie aber auch von einigen wichtigen Unterschieden.

Wie intelligente Verträge ist ein DApp eine Schnittstelle, die einen Benutzer über ein dezentralisiertes Peer-Netzwerk mit einem Dienst eines Anbieters verbindet. Aber während bei intelligenten Verträgen eine feste Teilnehmerzahl erforderlich ist, gibt es bei DApps keine Begrenzung der Nutzerzahl. Darüber hinaus sind sie nicht nur auf finanzielle Anwendungen wie intelligente Verträge beschränkt: ein DApp kann jeden erdenklichen Zweck erfüllen.

2. Was ist blockweise Programmierung?

Blockly ist eine **visuelle Programmiermethodik**, die auf der Programmiersprache JavaScript basiert. Sie besteht aus einem einfachen Satz von Befehlen, die wir kombinieren können, als wären sie die Teile eines Puzzles. Es ist ein sehr nützliches Werkzeug für diejenigen, die **lernen** wollen, **auf** intuitive und einfache Weise **zu programmieren**, oder für diejenigen, die bereits programmieren können und das Potenzial dieser Art von Programmierung erkennen wollen.

Blockly ist eine Form des Programmierens, bei der man keinen Hintergrund in irgendeiner Computersprache benötigt, weil es nur das Zusammenfügen von Grafikblöcken ist, als ob wir Lego oder ein Puzzle spielen würden, man braucht nur etwas Logik und das war's!

Jeder kann Programme für Mobiltelefone (Smartphones) erstellen, ohne sich mit diesen schwer verständlichen Programmiersprachen herumschlagen zu müssen, indem er einfach Blöcke in grafischer Form zusammensetzt.

3. Was ist eine Erweiterung?

Eine Erweiterung ist ein in einer Java-Programmiersprache erstelltes Modul, das in einer Datei mit der Erweiterung .AIX

Im Projekt Blockoin.org basierten wir auf der Schaffung benutzerfreundlicher Erweiterungen für den Finanzbereich, mit denen sie in wenigen Minuten mobile Anwendungen erstellen können, ohne über umfangreiche Programmierkenntnisse zu verfügen.

4. Was ist BlockoinEthereum und BlockoinINFURA?

BlockoinEthereum und BlockoinINFURA sind frei nutzbare Software (Erweiterungen), die die folgenden technologischen Lösungen (Algorithmen) zur Erstellung und Verwendung im Netzwerk des Ethereums beinhaltet:

- EINRICHTUNG NEUER KONTEN AUF DER ETHEREUM-PLATTFORM.
- BILANZBERATUNG, VERMÖGENSÜBERTRAGUNGEN (CRYPTOMONEDA-ÄTHER UND TOKEN)
- SCHAFFUNG NEUER ERC20-TOKEN UND CRYPTOMONEDA-TOKEN.
- ZUSAMMENSTELLUNG, ERSTELLUNG, BERATUNG UND VERÖFFENTLICHUNG VON INTELLIGENTEN VERTRÄGEN
- ERSTELLUNG VON OFFLINE- UND ONLINE-TRANSAKTIONEN.
- LADEN VON PRIMÄR- UND ÖFFENTLICHEN SCHLÜSSELN.
- WECHSELKURSE & TANKSTELLENKONSULTATION.
- ENTWICKLUNGS-, TEST- UND ETHEREUM-KERNNETZUMGEBUNGEN (NETZWERKE)
- UMFASST DIE 39 INFURA-MERKMALE.

5. Grundlegende Konzepte, die in der Ethereum-Plattform angewendet werden.

Was ist eine Blockkette?

Die Blockkette wird im Allgemeinen mit Bitcoin und anderen Krypto-Währungen in Verbindung gebracht, aber diese sind nur die Spitze des Eisbergs, da sie nicht nur für digitales Geld verwendet wird, sondern für alle Informationen, die für Benutzer und/oder Unternehmen einen Wert haben können. Diese Technologie, deren Ursprünge auf 1991 zurückgehen, als Stuart Haber und W. Scott Stornetta die erste Arbeit an einer Kette von kryptografisch gesicherten Blöcken beschrieben, wurde erst 2008 bemerkt, als sie mit der Einführung der Bitcoin populär wurde. Gegenwärtig wird seine Verwendung jedoch in anderen kommerziellen Anwendungen nachgefragt, und es wird prognostiziert, dass sie in mittlerer Zukunft auf mehreren Märkten, wie z.B. bei Finanzinstituten oder im Internet der Dinge (Internet of Things IoT), neben anderen Sektoren, wachsen wird.

Die Blockkette, besser bekannt unter dem Begriff Blockkette, ist ein einzelner, vereinbarter Datensatz, der über mehrere Knoten (elektronische Geräte wie PCs, Smartphones, Tablets usw.) in einem Netzwerk verteilt ist. Im Falle der Krypto-Währungen können wir uns das als das Buchhaltungsbuch vorstellen, in dem jede der Transaktionen aufgezeichnet wird.

Seine Funktionsweise kann kompliziert zu verstehen sein, wenn wir auf die internen Details seiner Umsetzung eingehen, aber die Grundidee ist einfach zu verfolgen.

Sie wird in jedem Block gespeichert:

1.- eine Anzahl gültiger Datensätze oder Transaktionen,

2.- Informationen über diesen Block,

3.- seine Verknüpfung mit dem vorherigen Block und dem nächsten Block durch den Hash jedes Blocks – ein eindeutiger Code, der wie der Fingerabdruck des Blocks aussehen würde.

Daher hat **jeder Block** einen **bestimmten und unverrückbaren Platz innerhalb der Kette**, da jeder Block Informationen aus dem Hash des vorherigen Blocks enthält. Die gesamte Kette wird auf jedem Netzwerknoten gespeichert, der die Blockkette bildet, so dass **eine exakte Kopie der Kette auf allen Netzwerkteilnehmern gespeichert wird**.

Was ist eine Adresse oder ein Konto innerhalb der Blockkette der Ethereum-Plattform?

Es handelt sich um eine Zeichenfolge von 42 Zeichen in der Ethereum-Plattform, die eine Zahl in hexadezimaler Basis darstellt, in der die im Ethereum definierten Vermögenswerte hinterlegt oder versandt werden. Bei anderen Blockketten-Plattformen kann z.B. die Anzahl der Zeichen des Kontos oder der Adresse unterschiedlich sein:

0x5d2Acdb34c279Aa6d1e94a77F7b18aB938BFb2bB

Was ist ein Kryptomoney?

Es handelt sich um eine digitale oder virtuelle Währung, die als Tauschmittel fungieren soll. Es verwendet Kryptographie (digitale Sicherheit), um Transaktionen zu sichern und zu verifizieren, sowie um die Schaffung neuer Einheiten eines bestimmten Kryptomoney zu kontrollieren.

Was ist Äther?

Ether ist die native digitale Währung der Ethereum-Blockchain-Plattform, einer 2013 entwickelten dezentralen Plattform. Ein Äther ist eine Einheit, die in kleinere Einheiten namens WEI unterteilt werden kann und folgende Äquivalenz hat

1 Äther = 1.000.000.000.000.000.000 Wei. (Im mathematischen Ausdruck ist es 10^{18}).

Welche Einheiten werden bei der Verwendung eines Äthers gehandhabt?

1	ether
10^3	finney
10^6	szabo
10^9	Shannon or GWEI this is use for the GasPrice.
10^{12}	babbage

10^{15}	lovelace
10^{18}	wei

Was ist ein Zeichen?

Tokens sind digitale Assets, die innerhalb eines bestimmten Projekt-Ökosystems verwendet werden können.

Der Hauptunterschied zwischen Tokens und Krypto-Währungen besteht darin, dass erstere eine andere Blockkettenplattform (nicht ihre eigene) benötigen, um zu funktionieren. Ethereum ist die gebräuchlichste Plattform zur Erstellung von Tokens, vor allem wegen seiner intelligenten Vertragsfunktion. Die auf der Ethereum-Blockkette erzeugten Wertmarken sind allgemein als ERC-20-Marken bekannt, obwohl es auch andere, speziellere Arten von Marken gibt, wie z.B. die ERC-721-Marke, die hauptsächlich für Sammelobjekte (Karten, Verwendung in Videospielen, Kunstwerke usw.) verwendet wird.

Was ist Gas?

Gas sind die Kosten für die Durchführung einer Operation oder einer Reihe von Operationen im Ethereum-Netz. Diese Operationen können mehrere sein: von der Durchführung einer Transaktion bis zur Ausführung eines intelligenten Vertrags oder der Erstellung einer dezentralisierten Anwendung.

Mit anderen Worten, einfacher ausgedrückt, das Gas ist die Einheit zur Messung der im Ätherum geleisteten Arbeit.

Wie in der physischen Welt gibt es auch im Ethereum Arbeitsplätze, die mehr kosten als andere: Wenn die Operation, die wir durchführen wollen, einen höheren Ressourceneinsatz der Knotenpunkte, die die Plattform bilden, erfordert, dann wird auch das Gas zunehmen und umgekehrt.

Das Gas dient in erster Linie dazu, auf der Ethereum-Plattform drei Funktionen zu erfüllen:

1.- Weist der Ausführung von Aufgaben Kosten zu; in Ethereum hat die Ausführung von Aufgaben auch Kosten. **Je nach Schwierigkeit der Aufgabe oder der Geschwindigkeit, mit der diese Aufgabe bearbeitet werden soll, werden die Berechnungskosten entsprechend höher oder niedriger sein**, und die Anzahl der Gase wird proportional zu- oder abnehmen.

2.- Sichert das System; Das Ethereum-System ist ein sicheres System, und dies ist weitgehend dank des Gases möglich.

Indem für jede durchgeführte Transaktion eine Provision verlangt wird, stellt die Plattform sicher, dass keine unbrauchbaren Transaktionen im Netzwerk verarbeitet werden. Dies trägt dazu bei, die Blockkette leichter zu machen, da dadurch nicht viele nutzlose Megabyte an Informationen in die Blockkette eingefügt werden.

Darüber hinaus ist das System mit dem Gas auch gegen "Spam" und die unendliche Verwendung von Schleifen geschützt: Anweisungen zur Ausführung sich wiederholender Aufgaben per Code.

Wenn es das Gas nicht gäbe, würde zum Beispiel nichts verhindern, dass eine Aufgabe unendlich oft wiederholt wird, das System kollabiert und unbrauchbar macht.

3.- Belohnung der Bergleute; Die Bergleute sind unabhängige externe Systeme, die über die ganze Welt verteilt sind und für die Ausführung der Transaktionen innerhalb der Ethereum-Plattform verantwortlich sind. Wenn wir eine Transaktion vornehmen oder einen intelligenten Vertrag ausführen, "zahlen" wir eine bestimmte Menge Gas.

Dieses Gas dient dazu, die Bergleute für die von ihnen genutzten Ressourcen (Hardware, Elektrizität und Zeit) zu "bezahlen" und fügt außerdem eine Belohnung für ihre Arbeit hinzu. Daher könnte man sagen, dass das Gas auch dazu beiträgt, das Gleichgewicht der Plattform zu erhalten.

Wir sprechen davon, Gas - in Anführungszeichen - zu "bezahlen", denn das ist es, was der Betrieb kostet. Mit anderen Worten, Sie "zahlen" für die Kosten, die Ethereum für die Bearbeitung der Transaktionen entstehen.

Was ist der Gaspreis?

Eine Gaseinheit entspricht der Ausführung einer Anweisung, wie z.B. einem Computerschritt.

Wie "kassieren" die Bergleute also das Gas, das sie als Belohnung erwerben?

Das Gas selbst ist nichts wert und kann daher nicht aufgeladen werden. Um dieses verbrauchte Gas "in Rechnung zu stellen", ist es notwendig, diesen verbrauchten Ressourcen einen Wert beizumessen, d.h. der von den Bergleuten geleisteten Arbeit einen Geldwert zuzuordnen. Die in einer Transaktion oder einem intelligenten Vertrag verwendete Gasmenge hat einen entsprechenden Preis in Äther. Dieser Preis wird als "Gaspreis" bezeichnet, er wird normalerweise von den Bergleuten festgelegt und ist je nach Auslastung der Ethereum-Blockkette variabel. Sie wird normalerweise in den GWEI-Einheiten bearbeitet.

Was ist ein Gaslimit?

Diese Daten geben den maximalen Wert von Gas an, den eine Transaktion verbrauchen kann, um gültig zu sein.

Normalerweise berechnet die Software, die für Transaktionen im Ethereum-Netz verwendet wird, automatisch eine Schätzung der Gasmenge, die für die Durchführung der Transaktion benötigt wird, und zeigt sie uns sofort an.

Was ist eine Tankstelle?

Es ist der Ort, an dem die von den Bergleuten gehandhabten Werte herangezogen werden, um im Konsens zu entscheiden, welcher Gaspreis für die verschiedenen Arten von Transaktionen in der Ethereum-Blockkette erforderlich ist.

Je nachdem, wie viel Gaspreis gewählt wird, wird die Prioritätszeit für die Ausführung der Transaktion gewichtet, normalerweise werden drei Arten von Gaspreisen behandelt: HANDEL: SOFORT, SCHNELL < 2 Minuten, STANDARD < 5 Minuten. Dies sind Durchschnittswerte und können je nach der Arbeitsbelastung (Traktionen) des Hauptnetzes des Ethereum variieren.

<https://ethgasstation.info/>

Was ist ein Austausch?

Ein Kryptomaniac Exchange ist der Treffpunkt, an dem der Austausch von Kryptomaniacs im Austausch gegen Fiat-Geld oder andere Kryptomaniacs stattfindet. In diesen Online-Tauschbörsen wird der Marktpreis generiert, der den Wert der Kryptomünzen auf der Grundlage von Angebot und Nachfrage markiert.

Was sind Wechselkurse?

Dies sind die Kurse für den Wert eines Äthers in den Währungen der einzelnen Länder. Zum Beispiel hat ein Äther am Tag der Erstellung dieses Handbuchs einen Wert in US-Dollar von \$430,94

Was ist ein intelligenter Vertrag?

In Ethereum ist ein Smart-Vertrag ein Programm in der Programmiersprache Solidity, das sich innerhalb der Blockkette Ethereum befindet, die über Anweisungen verfügt, die automatisch auf der Grundlage von vorher festgelegten Regeln ausgeführt werden sollen. Diese Eigenschaft stellt eine Weiterentwicklung in allen bestehenden Blockketten dar, da Sie viele Smart-Verträge erstellen können, die an verschiedene private und öffentliche Sektoren angepasst sind, um den Betrieb von Unternehmen effizienter zu gestalten oder gegebenenfalls an Systeme oder Programme aller Art anzupassen.

Was bedeutet "nonce"?

Es handelt sich um einen inkrementellen "Hexadezimalzahl"-Zähler, der die Transaktionen in jeder Richtung oder auf jedem in Ethereum eingerichteten Konto verfolgt.

Was ist eine Transaktion?

Es handelt sich um die Ausführung oder Übertragung einer Art von nicht materiellem Vermögenswert, dem innerhalb des Ethereum-Systems ein vorab festgelegter Wert zugewiesen werden kann und der später in einen materiellen Wert für ein Unternehmen oder eine Person umgewandelt werden kann.

Was ist txHash?

Es handelt sich um eine hexadezimale Zahl, die es ermöglicht, das Ergebnis jeder Transaktion im Detail nachzuvollziehen.

Welche Arten von Transaktionen gibt es?

Sie haben zwei Arten, eine ist die Transaktion "offline", die ohne die Notwendigkeit einer Verbindung zum Hauptnetzwerk von Ethereum erstellt wird, kann gespeichert werden, bis Sie sich für eine Verbindung zum Netzwerk von Ethereum entscheiden und die Transaktion freigeben, haben den Vorteil der Sicherheit, weil die gesamte Transaktion offline verarbeitet wird, was jede Anomalie verhindert, die in der Netzwerkverbindung sein könnte. Die andere Transaktion ist die "Online"-Transaktion, die immer mit dem Internet verbunden sein muss, mit den sicherheitstechnischen Vor- und Nachteilen, die diese Verbindung mit sich bringt.

Was ist INFURA.io?

Infura.io ist eine Plattform, die eine Reihe von Werkzeugen und Infrastrukturen bereitstellt, die es Entwicklern ermöglichen, ihre Anwendungsblockkette vom Testen bis hin zur Skalierung einfach und zuverlässig auf Ethereum und IPFS zuzugreifen.

Was ist eine Adresse im Ethereum-Netzwerk?

Eine Adresse oder ein Konto besteht aus drei Teilen, der Adresse, dem öffentlichen Schlüssel und dem privaten Schlüssel. Diese beiden Schlüssel sind eine Folge von Zahlen und Zeichen im Hexadezimalformat, die zum Senden und Empfangen (aktiv) oder Äther (digitale Währung) verwendet werden.

Der Primärschlüssel sollte niemals mit irgendjemandem geteilt werden, da er die Freigabe des Saldos (unterzeichnet die Transaktionen) auf dem Konto autorisiert.

Der öffentliche Schlüssel ist der gesamten Öffentlichkeit bekannt und wird mit jedermann geteilt, da er als Referenz dient, um zu bestätigen, dass die Transaktion wertmäßig korrekt ist und an wen sie gesendet wird.

Beispiele:

```
"private": "429a043ea6393b358d3542ff2aab9338b9c0ed928e35ec0aed630b93adb14a1c",
"public":
"049b4b7e72701a09d3ee09165bba460f2549494a9d9fd7a95aaac57c2827eac162fd9e105b
2461cd6594ca8ca6a8daf10fe982f918be1b0060c87db9cfbcd289a8",
"address": "88ab6dcecc3603c7042f4334fc06db8e8d7062d5"
}
```

Was ist Coinsolidation.org?

Es handelt sich um eine Plattform zur Konsolidierung von Krypto-Währungen. Wir haben die ersten hybriden Adressen in der zentralisierten und dezentralisierten Finanzwelt geschaffen und dabei die Blockly-Technologie verwendet, die eine Form der visuellen Programmierung darstellt, ohne dass fortgeschrittene Programmierkenntnisse auf der Grundlage funktionaler Erweiterungen erforderlich sind. Siehe unser WhitePaper unter www.coinsolidation.org

Welche Arten von Netzwerken zur Erprobung und Hauptnetzwerk in der Blockkette Ethereum?

<https://mainnet.infura.io>

Hauptnetz, Produktionsnetz, in dem alle Transaktionen in Echtzeit durchgeführt werden.

<https://ropsten.infura.io>

Das Ropsten-Testnetzwerk erlaubt es Blockkettenentwicklungen, ihre Arbeit in einer Live-Umgebung zu testen, aber ohne die Notwendigkeit von echten ETH-Token, mit einem Algorithmus namens (*Proof-of-Work*).

<https://kovan.infura.io>

Kovan-Testnetz, das im Gegensatz zu seinem Vorgänger ropsten einen Algorithmus namens "Proof of Authority" verwendet.

<https://rinkeby.infura.io>

Test Network, verwendet einen Algorithmus namens Proof of Authority. Einer der am häufigsten für Tests verwendeten.

<https://goerli.infura.io>

Goerli ist ein öffentliches Testnetzwerk für Ethereum, das die POA (Proof of Authority) Konsensusmaschine mit verschiedenen Kunden unterstützt. Spam-sicher.

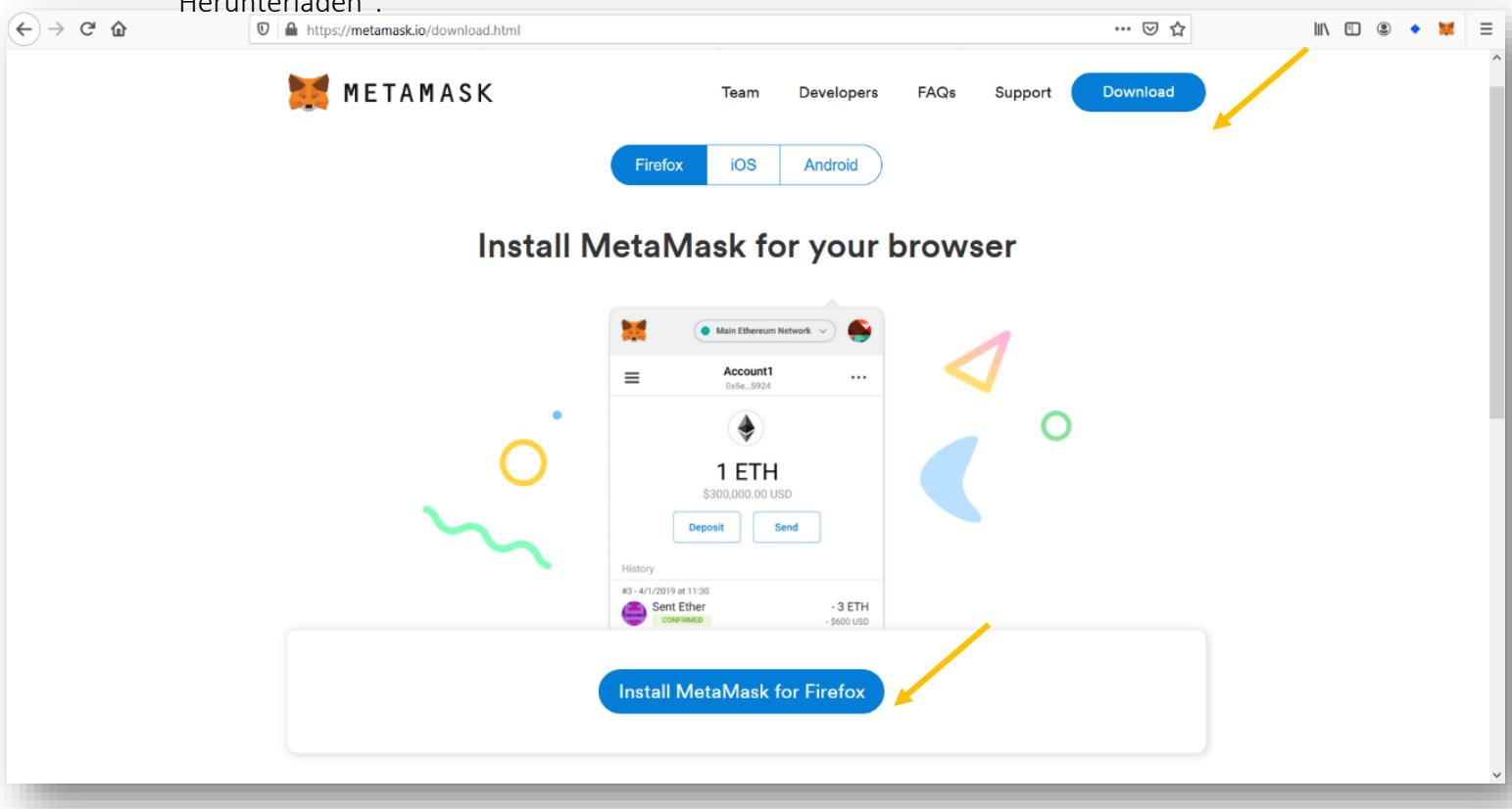
Insgesamt gibt es 5 Netzwerke, die auf der Blockkette Ethereum basieren, eines für die Produktion oder Haupt- und vier für Testzwecke, dann werden wir das Netzwerk von Rinkeby nutzen, um unsere Testumgebung zu installieren.

6. Installation und Konfiguration der Erweiterungs- und Ethereum-Testumgebung.

Wir brauchen zunächst eine Testumgebung. Erfahren Sie, wie Sie die verschiedenen Funktionalitäten der beiden Erweiterungen nutzen können, die zum Erstellen, Versenden, Veröffentlichen, Überprüfen und Erhalten von Daten aus allen Transaktionen, die wir auf der Ethereum-Plattform durchführen, verwendet werden.

Als erstes müssen wir unsere Testumgebung einrichten. Wir werden die Anwendung **METAMASK** installieren müssen. Dies ist eine Brieftasche zum Erstellen, Importieren von Ethereum-Konten und Verwalten von Transaktionen über den Browser in unserem Internet-Browser, den wir verwenden werden, ist Mozilla und kann auch in Chrome verwendet werden.

Gehen Sie auf die offizielle Website [www.metamask.io](https://metamask.io) und klicken Sie auf die Schaltfläche "Herunterladen".

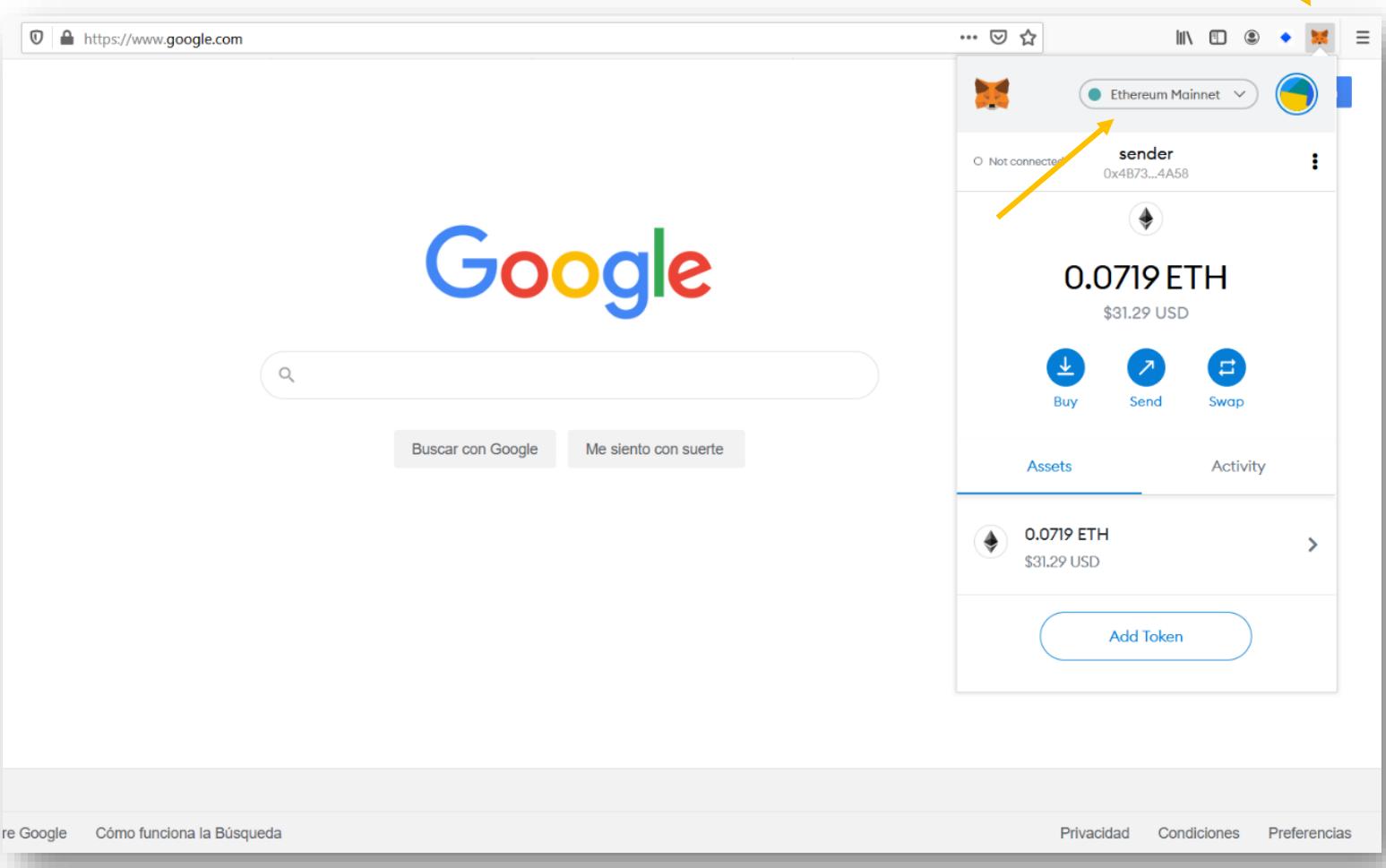


Klicken Sie dann auf die Schaltfläche "MetaMask für Firefox installieren" und akzeptieren Sie die Standardoptionen. Nach der Installation sehen wir oben rechts ein Icon, auf dem wir sehen, dass die Metamask-Software bereits installiert ist.

Klicken Sie auf das Metamask-Symbol und es erscheint die Option, ein bestehendes Konto zu importieren oder ein neues Konto zu erstellen. In unserem Fall werden wir ein Konto importieren, das wir mit der Methode der "Wiederherstellung durch Saatgut" bereits laufen

haben. Wenn Sie noch keines haben, erstellen Sie einfach ein neues Konto. In diesem Fall werden wir in jedem Fall nach einem Passwort gefragt, das uns zugeteilt wird.

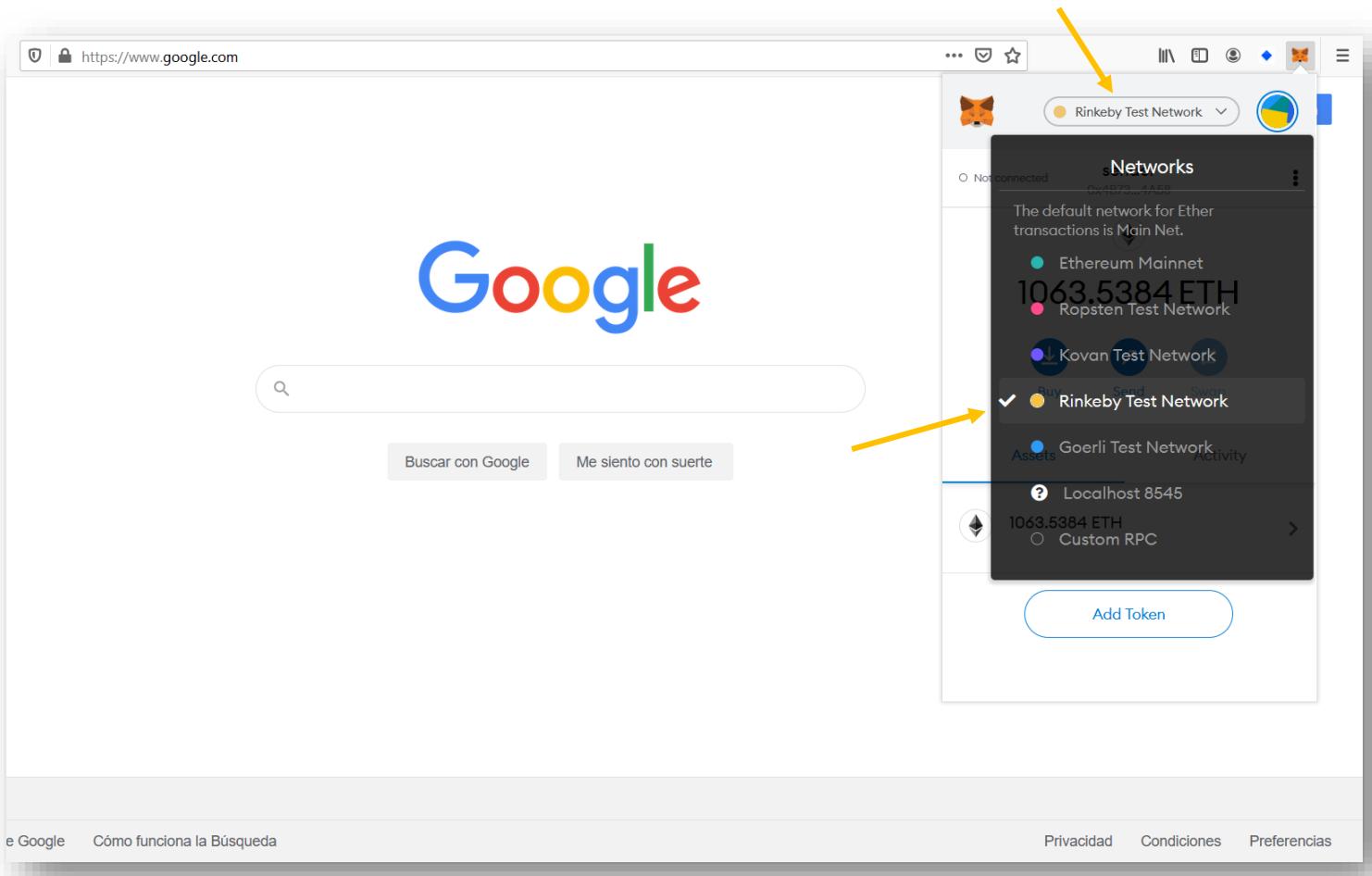
Später geben wir mit dem "Passwort" ein und können überprüfen, welchen Saldo wir haben, logischerweise wird der Saldo null sein, wenn er neu ist.



The screenshot shows a web browser window with the URL <https://www.google.com>. The main content is the Google search page. Overlaid on the bottom right is a wallet interface for Ethereum. The interface shows a balance of **0.0719 ETH** (worth \$31.29 USD). It includes buttons for **Buy**, **Send**, and **Swap**. At the top, it says "Ethereum Mainnet" and "sender 0x4B73...4A58". There are also tabs for **Assets** and **Activity**. A yellow arrow points from the text above to the "Ethereum Mainnet" button in the wallet interface.

Wir prüfen nun, wie wir einige Äther (digitale Münzen) auf unser Rinkeby-Testkonto einzahlen werden.

Oben haben wir die Möglichkeit, die Art des Netzwerks zu wählen, das wir benutzen werden, standardmäßig, wenn Sie es eingeben, platziert es Sie im "Ethereum Mainnet", aber wenn Sie klicken, können wir alle optionalen Netzwerke überprüfen, die wir wählen können, in unserem Fall wählen wir das Rinkeby-Netzwerk.



Der nächste Schritt ist die Einzahlung von Äthern auf unser auf das Rinkery-Netzwerk bezogenes Testkonto.

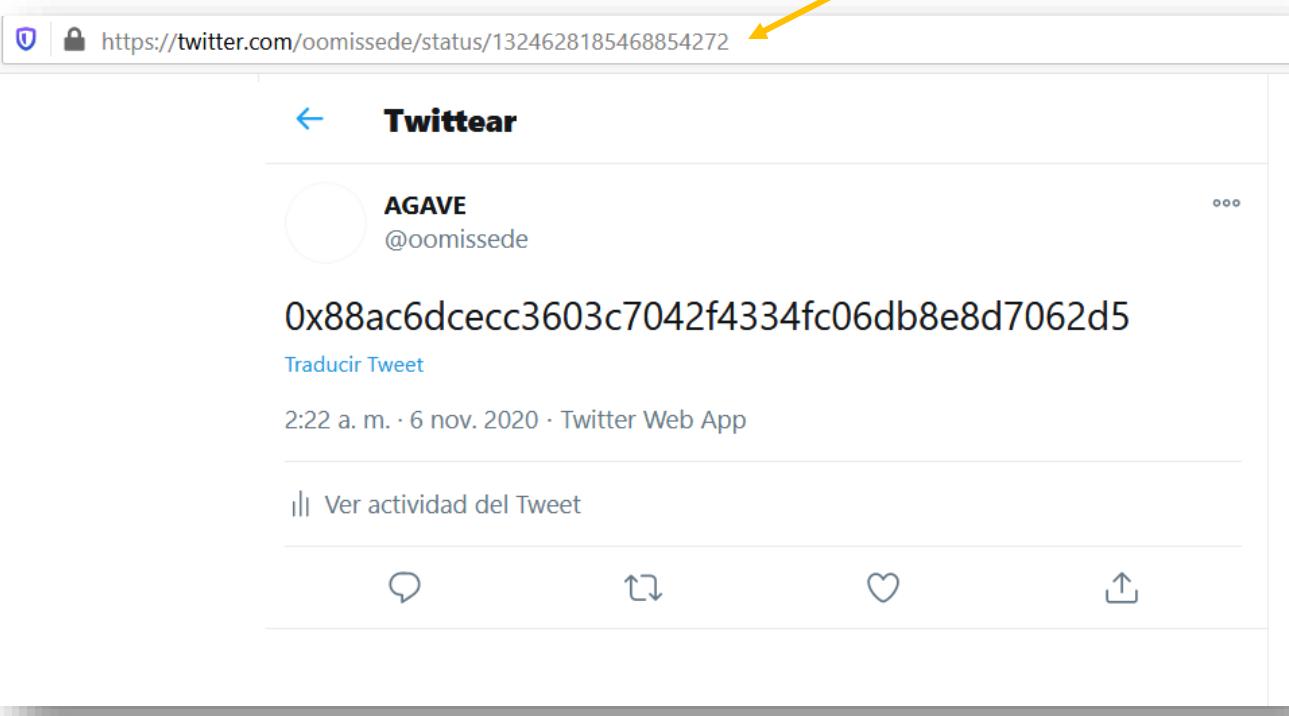
WICHTIGER HINWEIS: Die Äther (digitale Währung), die wir auf unser Konto beim Rinkery-Testnetzwerk hinterlegen, haben auf dem Kryptomarkt keinen Wert, sie werden von uns nur für Software-Tests verwendet. Prüfen Sie immer, an welchem Netzwerk wir arbeiten, um Fehler bei Transaktionen zu vermeiden.

Um Äther für Tests zu erhalten, müssen wir das folgende Verfahren durchführen.

In jedem Twitter-Account, den Sie haben, müssen wir twittern und einen Kommentar erstellen, der nur den Account und / oder die Adresse enthält, die wir in Metamask haben, dann müssen wir den Link des Kommentars kopieren, da wir diesen haben, gehen wir zum nächsten Link, um unseren Account zu verankern.

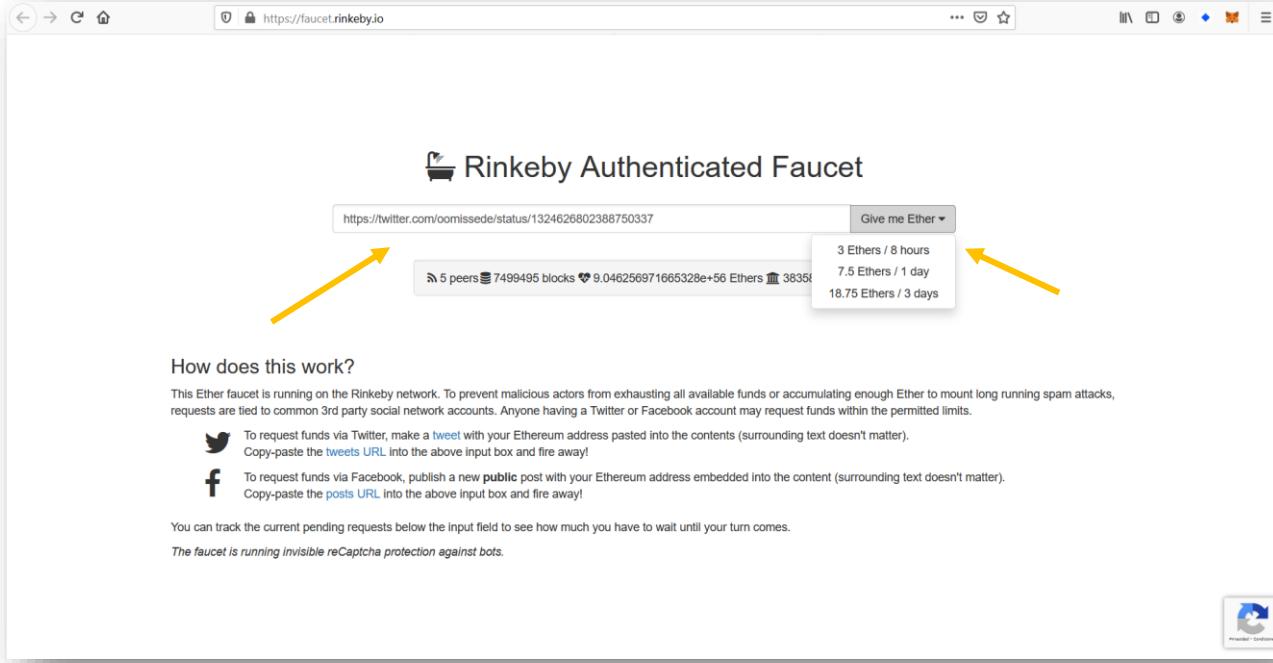
<https://faucet.rinkeby.io/>

Wir haben einen Twitter-Kommentar erstellt und den Link aus dem Kommentar kopiert.

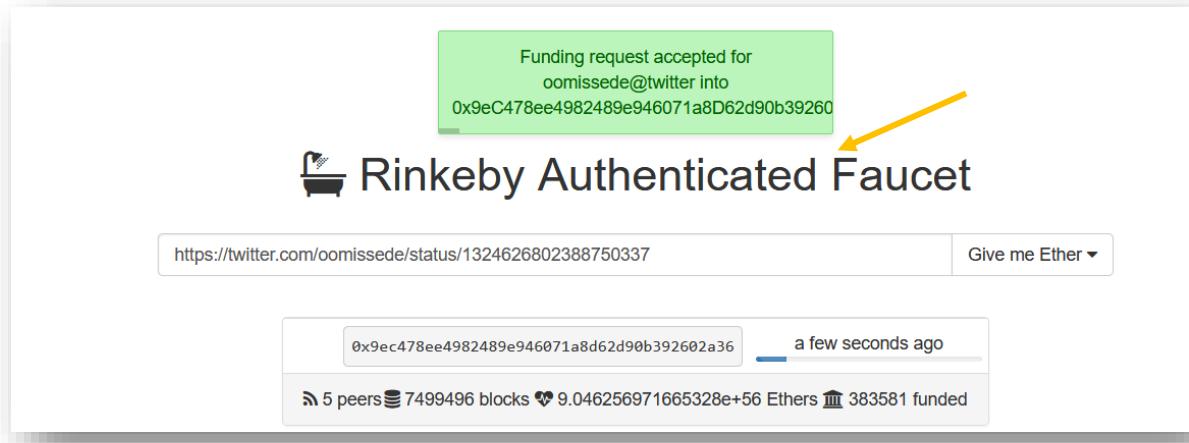


The screenshot shows a Twitter post by user @oomissede. The post contains a single line of text: "0x88ac6dcecc3603c7042f4334fc06db8e8d7062d5". Below the tweet, the timestamp "2:22 a. m. · 6 nov. 2020 · Twitter Web App" is visible. At the bottom of the tweet card, there are four interaction icons: a speech bubble, a retweet symbol, a heart, and a share icon. A yellow arrow points to the URL in the browser's address bar.

Auf der Website <https://faucet.rinkeby.io/> kopieren wir den Twitter-Link und wählen auf der Schaltfläche "Give me Ether" die gewünschte Menge aus.



The screenshot shows the Rinkeby Authenticated Faucet website. At the top, it displays the URL "https://twitter.com/oomissede/status/132462802388750337". Below this, there is a dropdown menu titled "Give me Ether" with three options: "3 Ethers / 8 hours", "7.5 Ethers / 1 day", and "18.75 Ethers / 3 days". Two yellow arrows point to the URL input field and the dropdown menu. The main content area contains instructions on how to request funds via Twitter or Facebook and a note about reCaptcha protection.



Jetzt, da wir Ethers auf unserem Konto haben, können wir mit den Tests auf der Ethereum-Plattform beginnen.

Wir haben zwei Erweiterungen zur Interaktion mit der Ethereum-Blockkette.

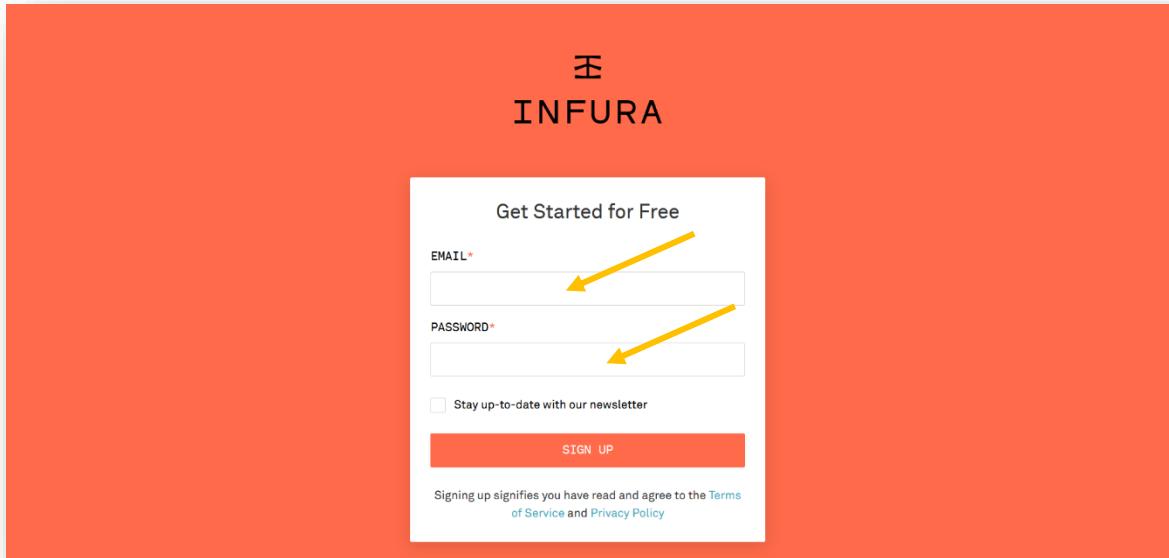
Die Erweiterung der **Ko-Konsolidierung. BlockoinETHEREUM.aix** enthält die Funktionen zur Ausführung der folgenden Funktionen:

- Erstellung neuer Konten (Adressen) in der Blockkette Ethereum (privateKey, publicKey, Adresse)
- Speicherung von Kontodaten (Adressen) in binären Dateien.
- Importieren binärer Dateikonten (Adressen)
- Erhalten eines Kontos (Adresse) durch privateKey.
- Erstellung und Versand von Transaktionen zwischen Konten (Adressen) "online".
- Erstellung, Unterzeichnung und Versand von Offline-PushRaw-Transaktionen.
- Konsultation von Transaktionsdetails Tx.
- Bilanzprüfung von Adressen und intelligenten Verträgen.
- Compiler für Smart-Verträge.
- Erstellung, Veröffentlichung und Ausführung von intelligenten Verträgen.
- Erstellung, Veröffentlichung und Ausführung von Token ERC20 (cryptomoney token).
- ABI-Code aus Smart-Vertrag beziehen.
- Verifizierung der Netzwerkverbindung.
- Abfrage des Wertes von Äther auf dem Kryptomarkt in einem beliebigen Land der Welt (Landeswährung des Landes) - Wechselkurse.
- GasPrice-Konsultation.
- Konsultation des "nonce" eines bestimmten Kontos.

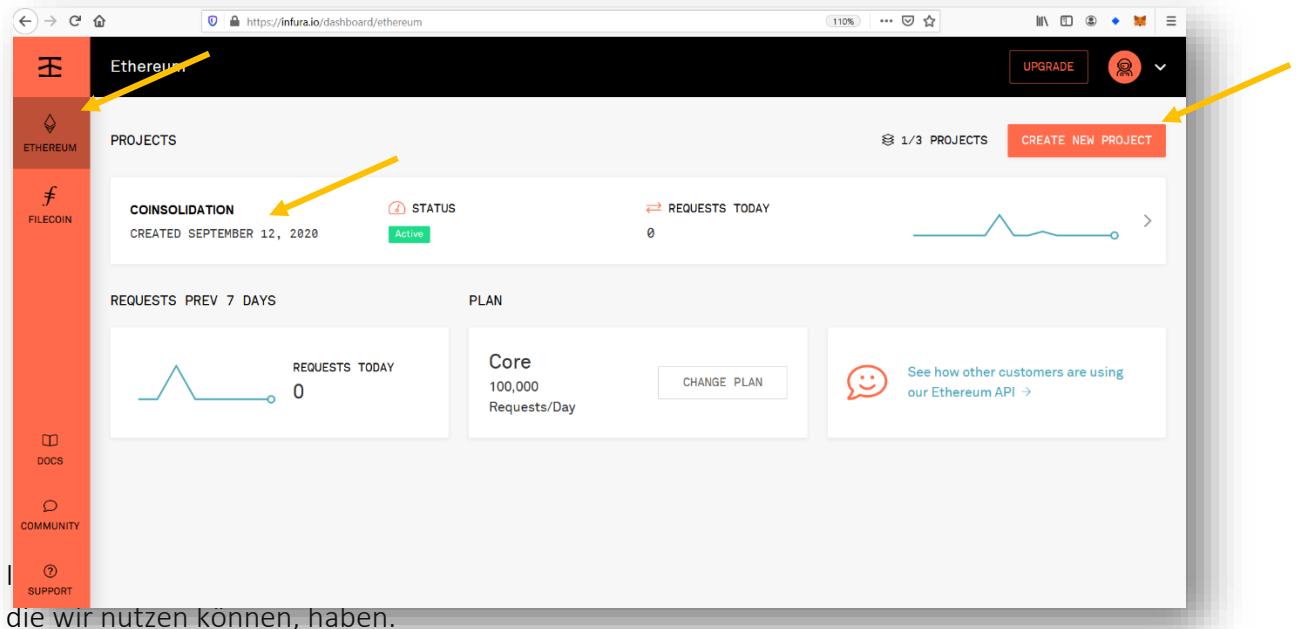
Die Erweiterung der **Co-Solidierung. BlockoinINFURA.aix** stellt uns die 40 Funktionalitäten der Infura.io Plattform zur Verfügung. Für Details konsultieren Sie bitte direkt die json-rpc Dokumentation unter <https://infura.io/docs>

Um die INFURA-Erweiterung nutzen zu können, müssen Sie ein Konto auf der Website infura.io einrichten, da wir eine KEY-API benötigen, um Anfragen an das Ethereum-Netzwerk senden zu können, sowie um die Ethereum-Testnetzwerke nutzen zu können.

Wie wir ein Konto eröffnen, ist einfach, wie unten dargestellt.



Sobald das Konto erstellt ist, treten wir ein und können die KEY API der verschiedenen Netzwerke haben. Wir gehen nach links oben und klicken auf Ethereum, dann sehen wir die Projekte, wir **erstellen** ein neues Projekt und stellen uns diesem Projekt vor.



die wir nutzen können, haben.

Innerhalb des Projekts können wir den API-Schlüssel (PROJEKT-ID) überprüfen und auswählen, an welchem Netzwerk wir arbeiten werden, oder die vollständigen Links der ENDPUNKTE auswählen.

The screenshot shows the Infura dashboard interface for the Ethereum project. The left sidebar has tabs for COINSOLIDATION, ETHEREUM (selected), and FILECOIN. The main area has tabs for REQUESTS and SETTINGS (selected). A 'SAVE CHANGES' button is at the top right. Under 'KEYS', there are fields for 'PROJECT ID' (containing a redacted value) and 'PROJECT SECRET' (containing a redacted value). A dropdown menu for 'ENDPOINTS' is open, showing 'Mainnet' as the current selection. Other options in the dropdown are 'Ropsten', 'Kovan', 'Rinkeby', and 'Görli'. At the bottom, there are two checkboxes: 'PROJECT SEC' with the sub-option 'Require project secret for all requests' and 'JWT REQUIRED' with the sub-option 'Require JWT for all requests'.

Um zum Beispiel das Hauptnetzwerk des Ethereum zu nutzen, würden wir folgende Lige nehmen:

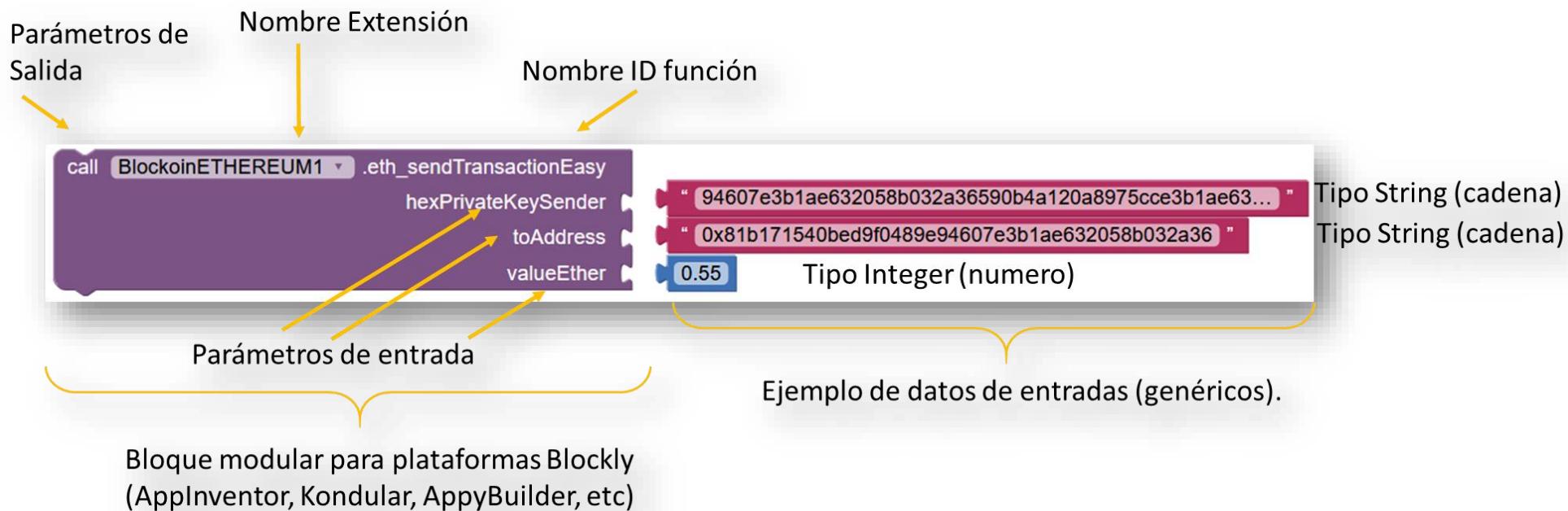


HINWEIS: Die Erweiterungen wurden auf AppInventor-, Kondular-, Thunkable- und AppyBuilder-Systemen getestet.

7. Definition und Verwendung von Blöcken (generische Funktion)

Wir beginnen damit, die Verteilung der Daten, die alle Blöcke haben werden, ihre Syntax der Verwendung und Konfiguration zu erklären.

Im folgenden Beispiel sehen wir einen modularen Block und seine Ein- und Ausgabeparameter sowie die Arten von Eingabedaten, diese Daten können vom Typ String (Zeichenfolge) oder Integer (ganzzahlig oder dezimal) sein. Wir zeigen, wie sie benutzt wird, und konfigurieren sie für ihr einwandfreies Funktionieren.



Jeder Modulblock hat seine eigene Beschreibung und wird benannt, falls er eine obligatorische oder optionale Abhängigkeit(en) von anderen Blöcken als Eingabeparameter hat, wird der Integrationsprozess angekündigt.

8. Funktionen und Veranstaltungen von Exchange Ethereum Extension (EEE).

***Testnetzwerk, das wir **Rinkeby** als urlNetwork verwenden werden, wenn Sie echte Transaktionen durchführen möchten, müssen Sie nur das urlNetwork-Netzwerk für **mainnet** ändern.

Block zur Überprüfung der Internetverbindung - (**CheckInternetConnection**).

call **BlockoinETHEREUM1** .**CheckInternetConnection**

Eingabeparameter: Nicht zutreffend.

Ausgabeparameter: Gibt "Wahr" zurück, wenn Sie eine Verbindung haben, oder "Falsch", wenn keine Verbindung besteht.

Beschreibung: Sperre zur Überprüfung der Internetverbindung und zum Senden von Daten (Transaktionen).

Block zum Generieren einer neuen "Offline"-Adresse - (**GenerateNewAddressEthereum**)

call **BlockoinETHEREUM1** .**GenerateNewAddressEthereum**
phraseHex “exchange ethereum extension for systems blockly”

Eingabeparameter: **phrasaHex <Zeichenkette>**.

Ausgabeparameter: Ereignis (**OutputGenerateNewAddressEthereum**)

Ausgänge: **PrivateKey<Zeichenkette>**, **PublicKey<Zeichenkette>** ,
AdresseEthereum<Zeichenkette>.

when **BlockoinETHEREUM1** .**OutputGenerateNewAddressEthereum**
privKeyEther **pubKeyEther** **addressEthereum**
do

Beschreibung: Erstellen Sie eine neue Ethereum-Adresse (Konto) auf der Grundlage eines Satzes oder einer Zahlenfolge. Die neue Adresse kann ohne Netzwerk- oder Internetverbindung erstellt werden - "Offline".

Block zum Generieren einer neuen "Online"-Adresse - (**GenerateNewAddressEthereum**)

call **BlockoinETHEREUM1** .GenerateNewAddressEthereumAPI

Eingabeparameter: Nicht zutreffend.

Ausgabeparameter: Rückgabe im JSON-Datenformat; privateKey, publicKey, Adresse.

Ausgabe-Beispiel:

```
{  
  "private": "9ab4b2fba728a55643414f26adc04eea080740860cbe39b13fe4acb43dbb9f83",  
  "public":  
    "0421d559aa98d6fc77688ae9f19b3dc502c05f0b7f70bbe924cb712d6243a489695b1c1ee03993b3b6d97277  
    97e780677a5469800b4d98374bdb910ed99fa2b5c8",  
  "address": "92a2f157d5aec3fa79f92995fea148616d82c5ef"  
}
```

Beschreibung: Erstellen Sie eine neue ethereum-Adresse (Konto). Es ist notwendig, Zugang oder eine Verbindung zum Internet zu haben, da die Generierung über den REST API-Dienst - "Online" - erfolgt.

Block zum Generieren einer neuen "Offline"-Adresse und zum Speichern der öffentlichen und privaten Schlüssel in Binärdateien - (**GenerateNewAddressEthereumStoreKeys**)

call **BlockoinETHEREUM1** .GenerateNewAddressEthereumStoreKeys
pathFilePrivateKey **" /mnt/sdcard/privateKey.bin "**
pathFilePublicKey **" /mnt/sdcard/publicKey.bin "**

Eingabeparameter: pathFilePrivateKey<String> , pathFilePublicKey<String>.

Ausgabeparameter: Ereignis (**OutputGenerateNewAddressEthereumStoreKeys**)

Ausgaben: **addressEthereum<String>** , **privateKeyECC<String>** , **publicKeyECC<String>** , **privateKeyHex<String>** , **publicKeyHex<String>**.

when **BlockoinETHEREUM1** .OutputGenerateNewAddressEthereumStoreKeys
 addressEthereum privateKeyECC publicKeyECC privatekeyHex publicKeyHex
do

Beschreibung: Erstellt eine neue zufällige Ethereum-Adresse (Konto) und speichert die öffentlichen und privaten Schlüssel in Binärdateien, die zum Importieren und Exportieren von Kontodaten verwendet werden. Die neue Adresse kann ohne Netzwerk- oder Internetverbindung erstellt werden - "Offline".

Block zur Generierung des öffentlichen Schlüssels -
(GeneratePublicKeyHexFromPrivateKeyHex).

```
call BlockoinETHEREUM1 .GeneratePublicKeyHexFromPrivateKeyHex
    hexPrivateKey "429a043ea6333b358d3542ff2aab9338b9c0ed928e35ec0a..."
```

Eingabeparameter: **hexPrivateKey <Zeichenkette>**.

Ausgabeparameter: Ereignis (**OutputGeneratePublicKeyHexFromPrivateKeyHex**)

Ausgaben: **Adresse<Zeichenkette>** , **publicKeyHex<Zeichenkette>**.

```
when BlockoinETHEREUM1 .OutputGetAddressEthereumFromPrivateKey
    address publicKeyHex
do
```

Beschreibung: Erstellt den öffentlichen Schlüssel basierend auf der Eingabe eines privaten Schlüssels.

Block zum Abrufen des Saldos einer Adresse - (**GetBalanceAddrEthereum**).

```
call BlockoinETHEREUM1 .GetBalanceAddrEthereum
    addressEthereum "0x92a2f157d5aec3fa79f92995fea148616d82c5ef"
```

Eingabeparameter: **hexPrivateKey <Zeichenkette>**.

Ausgabeparameter: Ereignis (**OutputGeneratePublicKeyHexFromPrivateKeyHex**)

Ausgänge: **balance<String>** , **total_received<String>** , **total_sent<String>** ,
unbestätigt_balance <String> , **final_balance<String>** , **n_tx<String>** ,
unbestätigt_n_tx<String> , **final_n_tx<String>**.

```
when BlockoinETHEREUM1 .OutputDataBalanceAddrEthereumWEI
    balance total_received total_sent unconfirmed_balance final_balance n_tx unconfirmed_n_tx final_n_tx
do
```

Beschreibung: Zeigt Bilanz- und detaillierte Kontodaten (Adresse) an.

Blockieren, um zu prüfen, ob Ihr Mobiltelefon die Netzschmittstelle aktiviert hat - (GetDataNetworkConnection).

call BlockinETHEREUM1 .GetDataNetworkConnection

Eingabeparameter: Nicht zutreffend.

Ausgabeparameter: Ereignis (OutputGeneratePublicKeyHexFromPrivateKeyHex)

Ausgänge: **interfacename<Zeichenkette>**, **istverbunden<Zeichenkette>**.

when BlockinETHEREUM1 .OutputGetDataNetworkConnection
interfacename isconnected
do [empty]

Beschreibung: Zeigt den Namen der mobilen Schnittstelle an und gibt an, ob die Schnittstelle aktiviert ist oder nicht.

Block zur Unterzeichnung der Transaktion "Offline" -

call BlockinETHEREUM1 .SignerGenericPushRawTransactionOffline
urlNetwork " [https://rinkeby.infura.io/v3/...] 440789c3..."
hexPrivateKeySender " 9eC478ee49824890b4a120a8975cce3b1ae632058b0301f8..."
nonceNumber get global nonce
gasPrice " 25000000000 "
gasLimit 21000
toAddress " 0x92a2f157d5aec3fa79f92995fea148616d82c5ef "
valueWei " 10000000000000000000 " x " 0.01 "

Erforderliche Abhängigkeit(en): Block (eth_getTransactionCount), Block (eth_SendRawTransactionInfura)

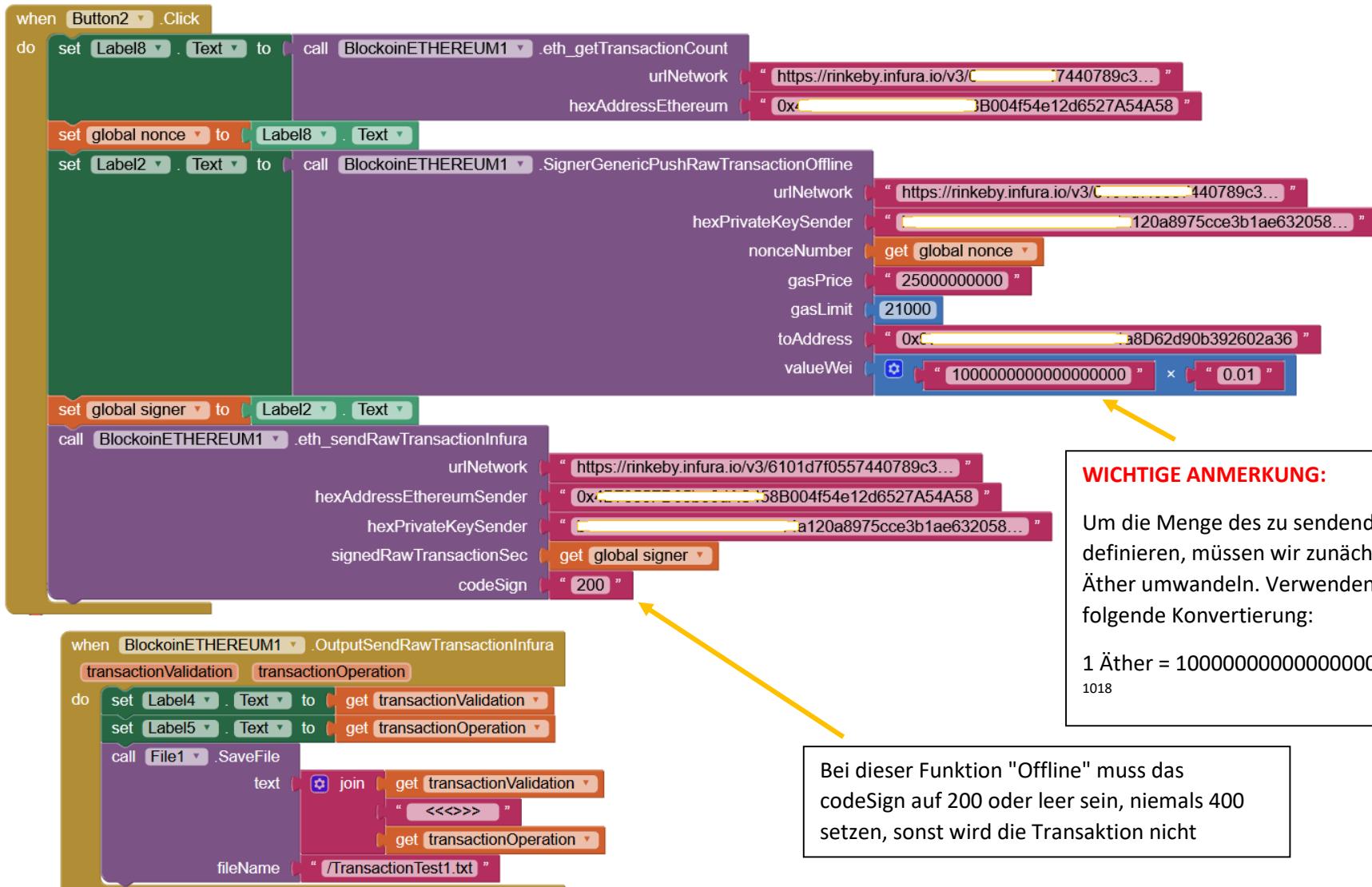
Parámetros de entrada: **urlNetwork <String>**, **hexPrivateKeySender <String>**, **nonceNumber <String>**, **gasPrice <String>**, **gasLimit <Integer>**, **toAddress <String>**, **valueWEI <Integer>**.

Ausgabeparameter:

Outputs: Unterschriebene Transaktion zum Versenden. <Zeichenkette>.

Beschreibung: Bereitet eine neue Transaktion für den Versand vor (verschlüsselt und signiert). Diese kann ohne Netzwerk- oder Internetverbindung bearbeitet werden - "Offline".

Beispiel für die vollständige Verwendung mit Blockabhängigkeiten (**SignerGenericPushRawTransactionOffline**).



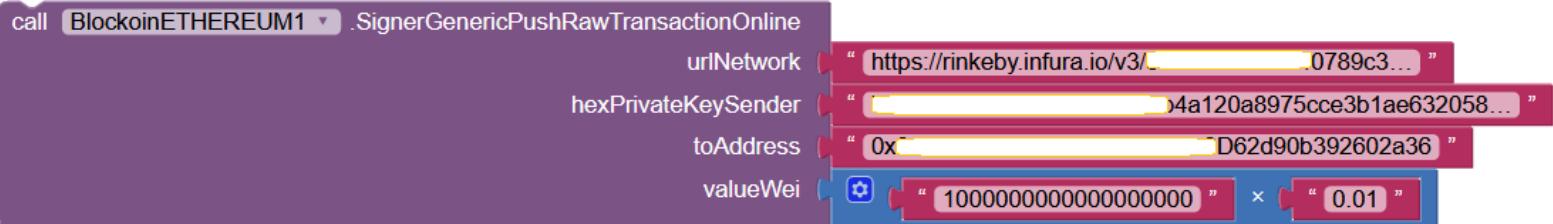
WICHTIGE ANMERKUNG:

Um die Menge des zu sendenden Äthers zu definieren, müssen wir zunächst das WEI in Äther umwandeln. Verwenden Sie die folgende Konvertierung:

1 Äther = 100000000000000000000000000000000 WEI =
10¹⁸

Bei dieser Funktion "Offline" muss das codeSign auf 200 oder leer sein, niemals 400 setzen, sonst wird die Transaktion nicht

Block zur Unterzeichnung einer "Online"-Transaktion -
 (SignerGenericPushRawTransactionOnline).

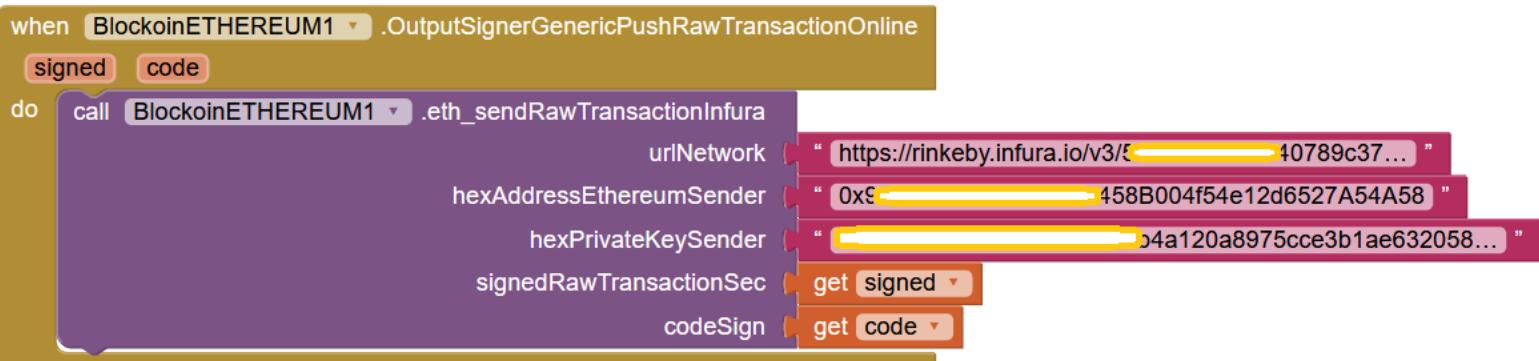


Obligatorische Einheit(en): Block (eth_SendRawTransactionInfura).

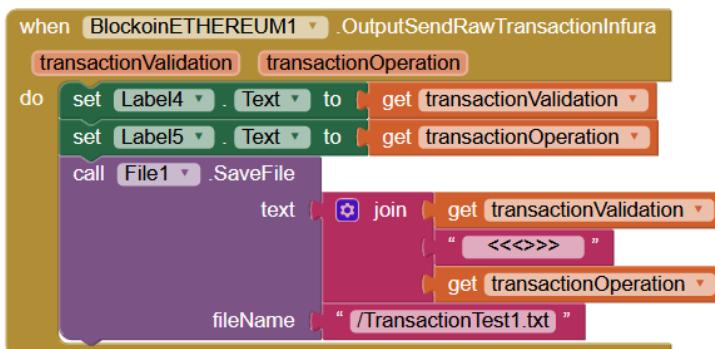
Eingabeparameter: urlNetzwerk <Zeichenkette>, hexPrivateKeySender <Zeichenkette>, toAddress <Zeichenkette>, WertWEI <Ganzzahl>.

Ausgabeparameter: Ereignisse, die in der folgenden Reihenfolge verwendet werden (OutputSignerGenericPushRawTransactionOnline) und (OutputSendRawTransactionInfura).

Ausgaben: signed<String>, code<String>.

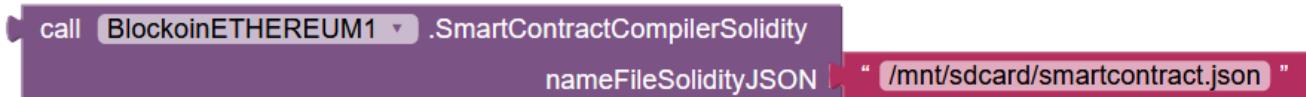


Ausgabeblock eth_sendRawTransactionInfura: transactionValidation<String>, transactionOperation<String>.



Beschreibung: Bereitet eine neue Transaktion für den Versand vor (verschlüsselt und signiert). Eine Netzwerk- oder Internetverbindung ist erforderlich - "Online".

Block für die **Kompilierung von** Smart contract "Online" - (**SmartContractCompilerSolidity**).



Eingabeparameter: **nameDateiSolidityJSON <Zeichenkette>**.

Ausgabeparameter: Zeigt den kompilierten intelligenten Vertrag an, diese Funktion hilft uns zu überprüfen, ob er gut geschrieben ist, bevor wir einen intelligenten Vertrag im Ethereum-Netzwerk veröffentlichen.

Ausgaben: **Kompilierter Code**.

Der Smart-Vertrag muss in einer Datei im JSON-Format vorliegen.

Beispiel für einen Basisvertrag von Smart in der Sprache Solidity.

```
pragma festigkeit ^0.5.0;

tödlicher Vertrag {
    Adresseigentümer;
    function mortal() { Eigentümer = msg.sender; }
    Funktion kill() { if (msg.sender == Besitzer) suicide(owner); } { if
        (msg.sender == Besitzer) suicide(owner); }
    Vertragsbegrüßer ist tödlich {
        String-Begrüßung;
        Funktion greeter(string _greeting) public { greeting = _greeting; }
        Funktion greet() Konstante gibt (Zeichenkette) zurück {Gruß zurückgeben;}
```

Beispiel eines früheren Smart-Vertrags im JSON-Format mit Kommentaren.

```
# Soliditätszusammenstellung mittels nicht veröffentlichtem Test
überprüfen
# Am Beispiel der Solidität des "Begrüßer"-Vertrags, der "Hallo Welt" von
Ethereum.
```

Datei: smartcontract.json

```
{
    "Solidität": "contract mortal { \n        /* Variable owner vom Typ address */\n        address owner definieren;\n        /* diese Funktion wird bei der\n        Initialisierung ausgeführt und setzt den Eigentümer des Vertrages */\n        function mortal() { owner = msg.sender; }\n        /* Funktion zur\n        Rückgewinnung der Gelder auf dem Vertrag */\n        function kill() { if\n            (msg.sender == owner) suicide(owner); }\n    }\n    contract greeter is mortal {\n        /* definieren variable Begrüßung vom Typ string */\n        string greeting;\n        /* dies läuft bei der Ausführung des Vertrages ab */\n        Funktion greeter(string _greeting) public {\n            greeting =
```

```
_greeting;\n } \n      /* Hauptfunktion */\n      Funktion greet()\nkonstante Rückgaben (string) {\n      return greeting;\n } \n} \n      ",\n"params." ("Hallo BlockCypher-Test")\n}
```

WICHTIGER HINWEIS: Das JSON-Format muss immer einen Zeilenumbruch am Ende jeder Zeile haben.

Beispiel einer kompilierten Ausgabe von Smartcontract.

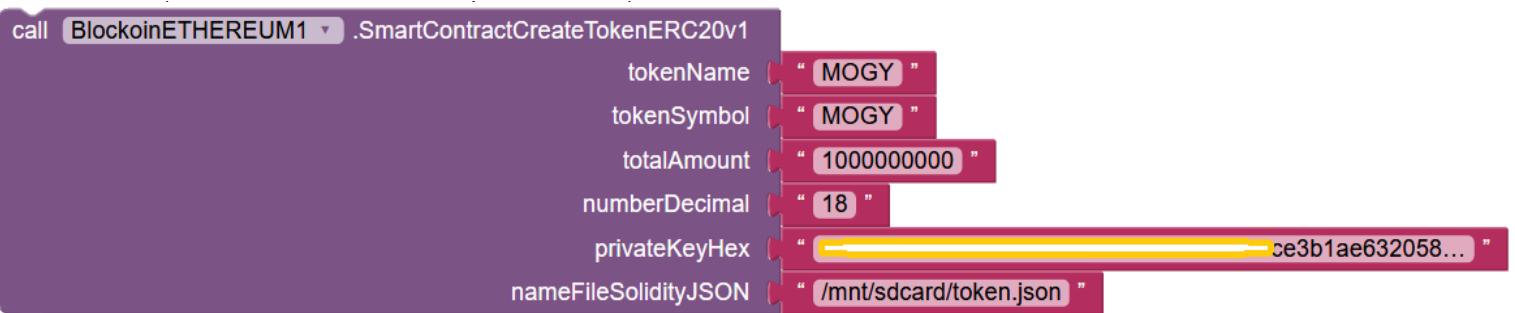
```
[\n{\n    "Name": " u003cstdin\u003e:Begr\u00fc\u00dfer",\n    "Solidit\u00e4t": "contract mortal {\n        /* Variable owner vom Typ\n        address*/\n        address owner definieren;\n        /* diese Funktion wird\n        bei der Initialisierung ausgef\u00fchrt und setzt den Eigent\u00fcler des Vertrages\n        */\n        function mortal() { owner = msg.sender; }\n        /* Funktion zur\n        R\u00fcckgewinnung der Gelder auf dem Vertrag */\n        function kill() { if\n            (msg.sender == owner) suicide(owner); }\n    }\n    contract greeter is mortal\n    {\n        /* definieren variable Begr\u00fc\u00dfer vom Typ string */\n        string greeting;\n        /* dies l\u00e4uft bei der Ausf\u00fchrung des Vertrages ab */\n        Funktion greeter(string _greeting) public {\n            greeting =\n            _greeting;\n        }\n        /* Hauptfunktion */\n        Funktion greet()\n        konstante R\u00fcckgaben (string) {\n            return greeting;\n        }\n    }\n    "M\u00fclleimer":\n    "606060405260405161023e38038061023e8339810160405280510160008054600160a060\n    020a031916331790558060016000509080519060200190828054600181600116156101000\n    203166002900490600052602060002090601f016020900481019282601f10609f57805160\n    ff19168380011785555b50608e9291505b808211560cc57600081558301607d565b50505\n    061016e806100d06000396000f35b828001600101855582156076579182015b8281111560\n    7657825182600050559160200191906001019060b0565b509056606060405260e060020a6\n    00035046341c0e1b58114610026578063cfae321714610068575b005b6100246000543373\n    ffffffffffffffffffffffff908116911614156101375760005473fff\n    fffffffffffff908116911614156101375760005473fff\n    a06020601f600260001961010086881615020190941693909304928301819004028101604\n    0526080828152929190828280156101645780601f10610139576101008083540402835291\n    60200191610164565b6040518080602001828103825283818151815260200191508051906\n    0200190808383829060006004602084601f0104600f02600301f150905090810190601f16\n    80156101295780820380516001836020036101000a031916815260200191505b509250505\n    06\n    "abi": [\n        {\n            "Konstante": falsch,\n            "Eingaben": [],\n            "Name": "t\u00f6ten",\n            "Ausg\u00e4nge": [],\n            "Typ": "Funktion".\n        },\n        {\n            "Konstante": wahr,\n            "Eingaben": [],\n            "Name": "Gru\u00dff",\n            "Ausg\u00e4nge": [\n                {\n
```

```

        "Name": "",
        "Typ": "Zeichenfolge".
    },
],
"Typ": "Funktion".
},
{
"Eingaben": [
{
    "Name": "_Gruß",
    "Typ": "Zeichenfolge".
}
],
"Typ": "Erbauer".
}
],
"params": [
    "Hallo BlockCypher-Test"
]
},
{
    "Name": "sterblich",
    "Solidität": "contract mortal {\n/* Variable owner vom Typ\naddress*/\n    address owner definieren;\n/* diese Funktion wird\nbei der Initialisierung ausgeführt und setzt den Eigentümer des Vertrages\n*/\n    function mortal() { owner = msg.sender; }\n/* Funktion zur\nRückgewinnung der Gelder auf dem Vertrag */\n    function kill() { if\n(msg.sender == owner) suicide(owner); }\n}\n\ncontract greeter is mortal\n/* definieren variable Begrüßung vom Typ string */\n    string greeting;\n/* dies läuft bei der Ausführung des Vertrages ab */\n    Funktion greeter(string _greeting) public {\n        greeting =\n_greeting;\n        /* Hauptfunktion */\n        Funktion greet()\nkonstante Rückgaben (string) {\n            return greeting;\n        }\n    }",
    "Mülleimer": "606060405260008054600160a060020a03191633179055605c8060226000396000f36060\n60405260e060020a600035046341c0e1b58114601a575b005b60186000543373ffffffffffff\nffffffffffffffffffffffffffff90811691161415605a5760005473ffffffffffff\nffffffffffffffffffffffff16ff5b56",
    "abi": [
{
    "Konstante": falsch,
    "Eingaben": [],
    "Name": "töten",
    "Ausgänge": [],
    "Typ": "Funktion".
},
{
    "Eingaben": [],
    "Typ": "Erbauer".
}
],
"params": [
    "Hallo BlockCypher-Test"
]
}
]
}

```

Block zum Kompilieren, Erstellen und Veröffentlichen von ERC20-Token -

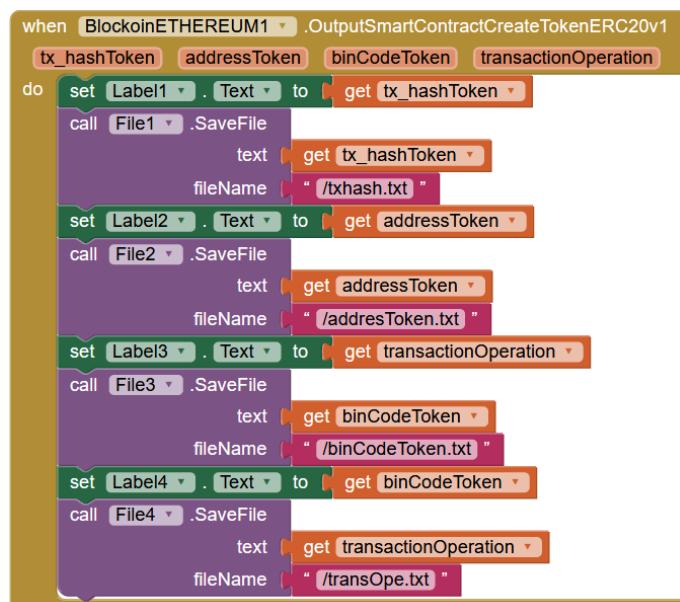


Obligatorische Einheit(en): Block (**CreateTestingFie**). **WICHTIG**: Zuerst müssen Sie diesen Block verwenden, um sicherzustellen, dass der Pfad korrekt ist, denn wenn Sie keinen gültigen Pfad im Block angeben (**SmartContractCreateTokenERC20v1**), wird die Token-Erstellung nicht ausgeführt, da der Eingabeparameter "nameFileSolidityJSON" zum Erstellen einer temporären Datei verwendet wird.

Eingabeparameter: **tokenName <String>**, **tokenSymbol <String>**, **totalAmount <String>**, **numberDecimal <String>**, **privateKeyHex <Integer>**, **nameFileSolidityJSON <String>** Diese Datei ist der gültige Pfad, um eine temporäre Datei zu erstellen, Sie müssen sicherstellen, dass der Pfad gültig ist, um zu testen, dass die Datei erstellt wird Sie können den Block (**CreateTestingFile**) verwenden, nachdem Sie ihn verwendet haben, überprüfen Sie, ob er erfolgreich erstellt wurde.

Ausgabeparameter: Ereignis (**OutputSmartContractTokenERC20v1**).

Ausgaben: **tx_hashToken<Zeichenkette>** , **addressToken<Zeichenkette>** , **binCodeToken<Zeichenkette>** , **trasactionOperation<Zeichenkette>**.



Beschreibung: Intelligenter Vertrag "Token ERC20" - aktiv im Ethereum-Netzwerk veröffentlicht. Bei Version 1 (v1) ist der Parameter für den Gasgrenzwert bereits auf 500.000 WEI eingestellt.

Beispiel für die Ausgabe der vorherigen Funktion **SmartContractCreateTokenERC20v1**.



9. Schritte zum Erstellen eines CryptoToken oder Cryptomoney-Tokens.

Schritt 1.

Überprüfen Sie, ob der temporäre Pfad auf dem mobilen Gerät, auf dem der Smart-Vertrag erstellt wurde, gültig ist und eine Datei erfolgreich erstellt werden kann. Dies geschieht mit Hilfe des Blocks (**CreateTestingFile**). Vergewissern Sie sich, dass die in der Eingabe "pathTestFile" angegebene Testdatei erstellt wurde, der Pfad lautet im Allgemeinen: [/mnt/sdcard/name_datei.txt](#)

Schritt 2 (optional).

In diesem Schritt prüfen wir, ob das Konto (Adresse), von dem die Operation abgebucht wird, über genügend Guthaben verfügt, um die Transaktion der Erstellung und Veröffentlichung eines Smart-Vertrags durchzuführen. Dies kann mit dem Block (**eth_VerifiBalanceForTransaccionSmartContract**) überprüft werden. Diese Verwendung dieses Blocks ist optional, da die Blöcke, die den **SmartContractCreateTokenERC20v1** oder **SmartContractCreateTokenERC20v2** erzeugen, diese Verifizierung bereits intern enthalten.

Schritt 3.

Wählen Sie aus, welchen Block Sie zum Erstellen des ERC20-Tokens verwenden möchten. Sie haben zwei Optionen:

a.- Block **SmartContractCreateTokenERC20v1** hat bereits den impliziten Wert des Gaslimits mit einem Wert von 500.000 WEI zugewiesen.

b.- Block **SmartContractCreateTokenERC20v2** hat die Möglichkeit, die Gasbegrenzung nach den Bedürfnissen des Endbenutzers oder Entwicklers zu konfigurieren. Es ist zu beachten, dass es bei einem sehr niedrigen Gasgrenzwert von weniger als 350.000 Wei sehr gut möglich ist, dass der Smart contr.

Schritt vier.

Verwenden Sie entweder die Blöcke **SmartContractCreateTokenERC20v1** oder **SmartContractCreateTokenERC20v2** und stellen Sie sicher, dass bei Verwendung der Eingangsvariablen "nameFileSolidityJSON" diese gleich der bereits in Schritt 1 geprüften Eingangsvariablen "pathTestFile" des Blocks (**CreateTestingFile**) ist.

Schritt fünf.

Bevor die Erstellung eines ERC20-Tokens mit einem der Blöcke **SmartContractCreateTokenERC20v1** oder **SmartContractCreateTokenERC20v2** ausgeführt wird, wird empfohlen, die Ereigniswerte (Ergebnisse) entsprechend zu speichern, um die Ergebnisse zu sichern (tx_hashToken, addressToken, binCodeToken, transactionOperation). Siehe Beispiel der Ausgabe der vorherigen Funktion **OutPutSmartContractCreateTokenERC20v1**.

Schritt sechs.

Führen Sie die Erstellung des ERC20-Tokens durch und veröffentlichen Sie es dann zum Verkauf. Siehe Abschnitt 10.

10. Wie Sie einen neuen Vermögenswert oder Ihr Crypt-Token zum Verkauf anbieten können (Token ERC20).

Da wir einen ERC20 - Cryptomoney Token (siehe **SmartContractCreateTokenERC20v1 Block** oder mit dem **SmartContractCreateTokenERC20v2 Block**) erstellt haben, müssen wir ihn in irgendeine Börse hochladen, damit ihn jeder in der Welt kaufen kann. Ein Exchange ist eine Website im Internet, auf der neue Token veröffentlicht werden.

Der Austausch wird in zwei Arten eingeteilt: zentralisiert und dezentralisiert. Der Hauptunterschied besteht darin, dass man von einer Art internationalen Körperschaft (zentralisiert) verwaltet und geprüft wird, und die dezentralisierten Körperschaften haben niemanden bei den Rechnungsprüfern. Dies mag zwar mehr Zuversicht geben, aber die Realität sieht so aus, dass die dezentralisierten in letzter Zeit stärker geworden sind und meist ohne größere Probleme eingesetzt werden.

Eine der besten Praktiken beim Umgang mit Vermögenswerten jeglicher Art ist es, nicht alle auf einem Konto zu haben, sondern sie auf mehrere Konten zu verteilen, um die Sicherheit sowohl der privaten als auch der öffentlichen Schlüssel zu gewährleisten.

In unserem Fall werden wir einen dezentralen Austausch verwenden, aber bereits mit einer nicht schlechten Vorgesichte, werden wir www.forkdelta.app verwenden.

Zu diesem Zeitpunkt müssen wir bereits die Anwendung für den Browser (Mozilla oder Chrome) **METAMASK** www.metamask.io installiert haben, da die Börse, um Ihre Seite www.forkdelta.app besuchen zu können, sich mit dem Konto verbinden muss, das wir bei Ethereum haben.

Ein wichtiger Punkt ist, dass unser Ethereum-Konto, das wir bereits in METAMASK haben, einen Saldo von mehr oder weniger 10 USD aufweisen sollte, da wir bei der Veröffentlichung unseres neuen ERC20-Tokens, das wir mit dem **SmartContractCreateTokenERC20v1-Block** oder mit dem **SmartContractCreateTokenERC20v2-Block** erstellt haben, die Transaktion bezahlen müssen, um es an der Börse zu veröffentlichen.

Um die Börse www.forkdelta.app nutzen zu können, benötigen wir die folgenden Token-Daten, die wir zum Verkauf an der Börse veröffentlichen wollen.

Adresse des neuen ERC20-Tokens, den wir zuvor erstellt haben.

0x54093F2C720b18Fd795645b5101A37EB49d1A94a

Anzahl der Dezimalstellen, die unsere Berührung verwendet.

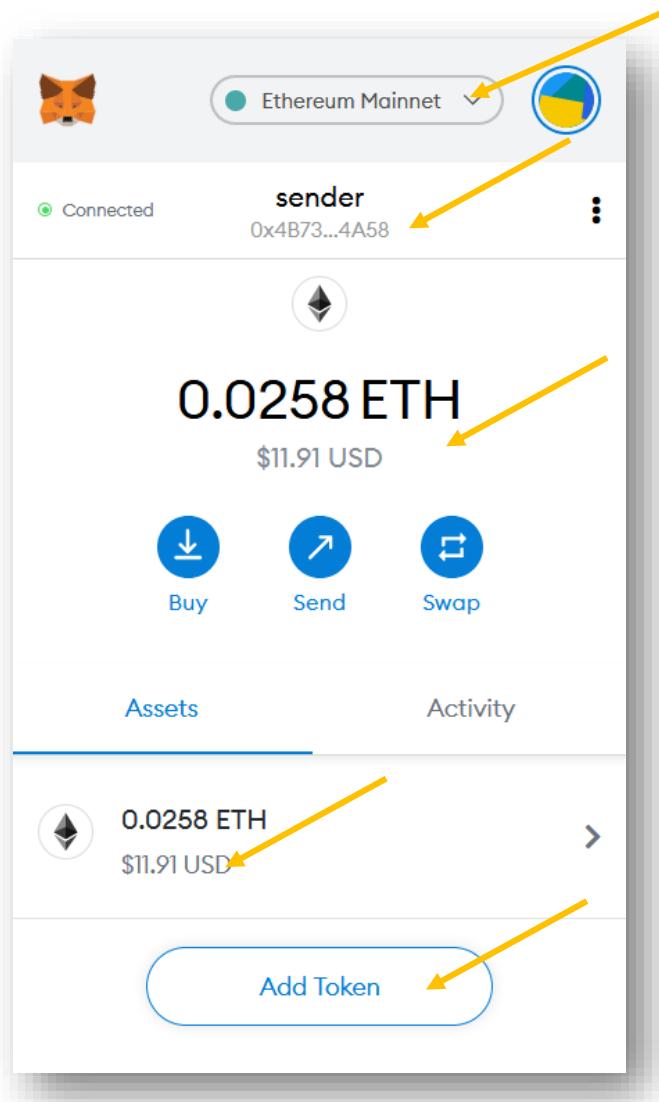
18

Der Name, der das Token identifiziert.

MOGIE

Beginnen wir damit, zu überprüfen, ob das von uns erstellte Token die oben genannten Parameter aufweist und ob es mit den Parametern erstellt wurde, mit denen es ursprünglich erstellt wurde.

Gehen wir zu **METAMASK** und vergewissern wir uns zuerst, dass wir auf dem Konto sind, mit dem wir den Token erstellt haben. Gehen Sie dann nach unten und klicken Sie auf die Schaltfläche "Add Token".



Wir werden nun unsere neue Marke hinzufügen, um Ihren aktuellen Kontostand zu sehen. Nachdem Sie auf die Schaltfläche "Benutzerdefiniertes Token" oben auf der Seite geklickt haben, geben Sie die gewünschten Informationen in die folgenden Felder ein und klicken dann auf die Schaltfläche "Weiter". Danach erscheint unsere neue Wertmarke mit dem Saldo, den Sie haben. Wenn Sie keinen Saldo haben, prüfen Sie, ob Sie sich in dem Konto befinden, mit dem Sie den Token erstellt haben. Wenn Sie sich nicht in einem Konto befinden, mit dem Sie den Token erstellt haben, haben Sie einen Nullsaldo, da Sie noch keinen Token gekauft haben.

The image consists of two side-by-side screenshots of a mobile application interface for managing tokens on the Ethereum Mainnet.

Screenshot 1: Add Tokens

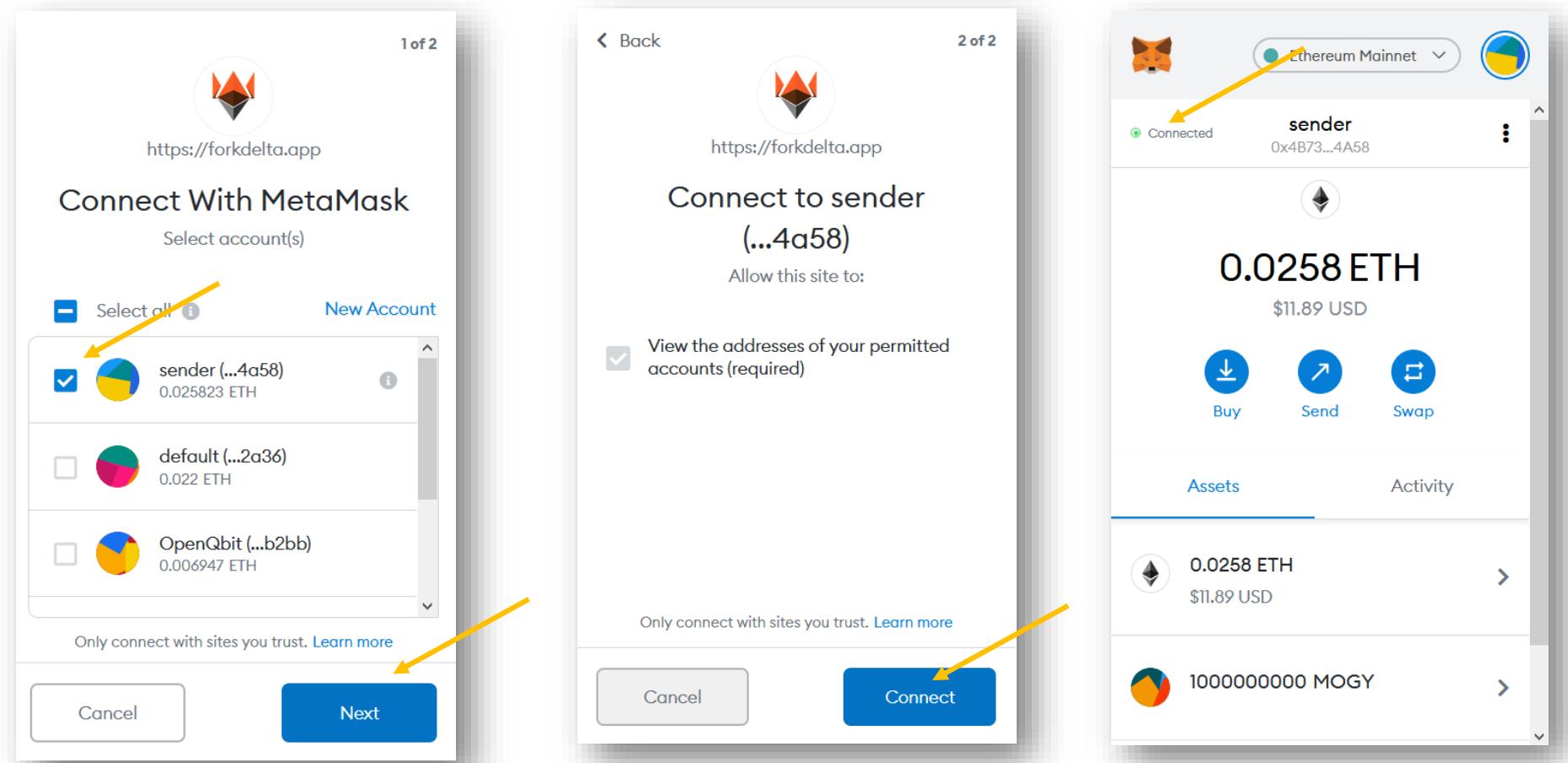
- Top navigation bar: Ethereum Mainnet dropdown, network switch icon.
- Section title: **Add Tokens**.
- Input fields:
 - Search (disabled)
 - Custom Token** (selected)
 - Token Contract Address: `0x54093F2C720b18Fd795645b5101A3...`
 - Token Symbol: `MOGY`
 - Decimals of Precision: `18`
- Buttons at the bottom: **Cancel** and **Next**.

Screenshot 2: Token Overview

- Top navigation bar: Ethereum Mainnet dropdown, network switch icon.
- Section title: **sender / MOGY**.
- Token details:
 - Icon: A circular logo divided into four quadrants (orange, yellow, green, blue).
 - Balance: **1000000000 MOGY**
- Actions:
 - Send** button with a blue arrow icon.
 - Swap** button with a blue double arrow icon.
- Text: **You have no transactions**.

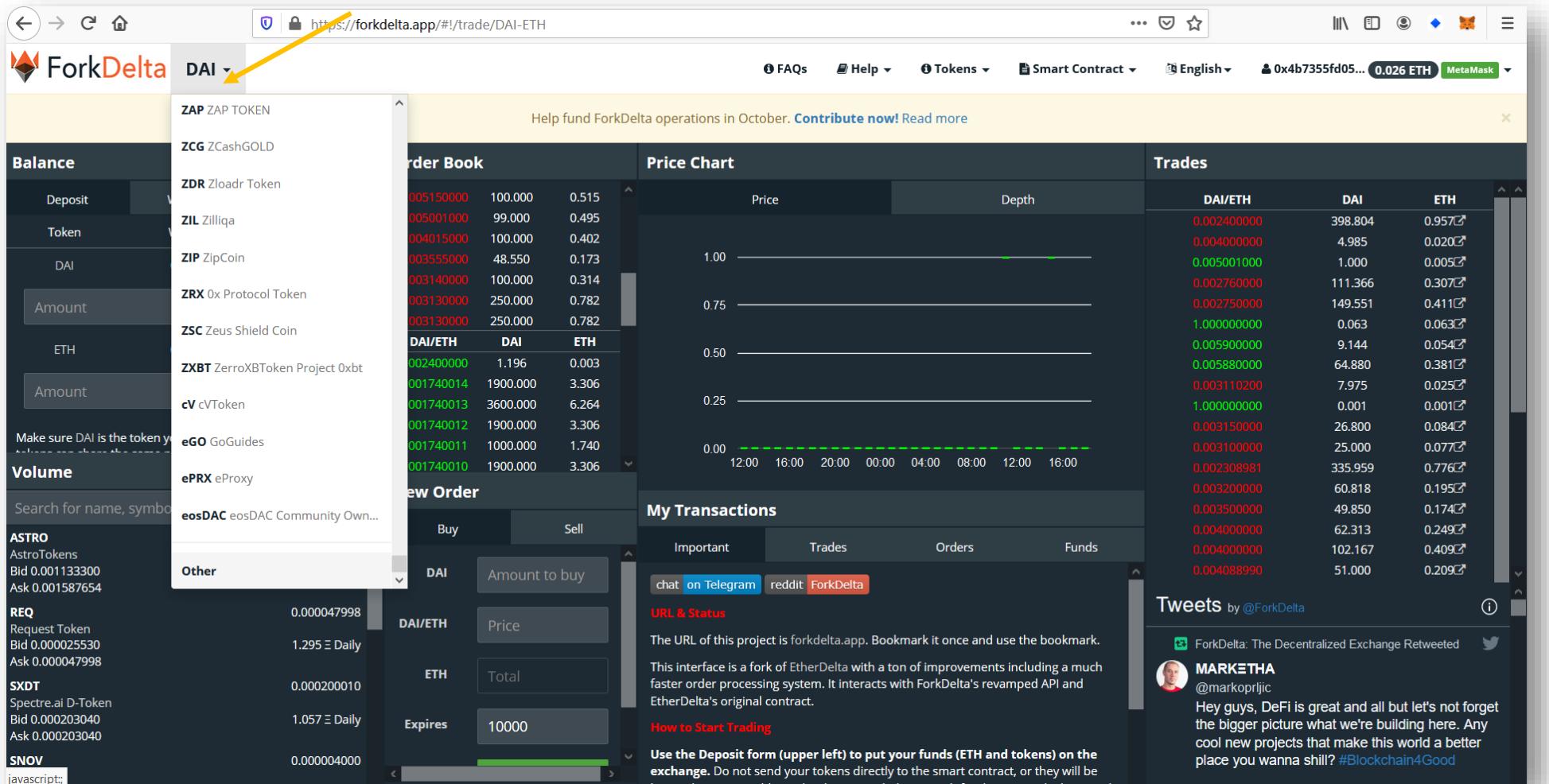
Sobald wir unseren neuen Token zum Verkauf an der Börse [www.forkdelta.app](https://forkdelta.app) hochladen.

Wenn Sie sich auf der Exchange-Website befinden, werden Sie aufgefordert, sich mit der Website <https://forkdelta.app> zu verbinden. Klicken Sie unten auf die Schaltfläche "Weiter", dann auf "Verbinden" und schließlich können Sie überprüfen, ob Sie mit der Website forkdelta.app verbunden sind.



Es ist an der Zeit, das neue Token auf der Börse <https://forkdelta.app> freizugeben.

Klicken Sie auf das obere Menü "DAI" und gehen Sie zum Ende der Schrifttrolle und wählen Sie die Option "Andere".



The screenshot shows the ForkDelta interface. A yellow arrow points to the 'DAI' dropdown menu in the top left corner, which is currently open. The 'Other' option is highlighted in the dropdown menu. The main trading area displays the Order Book, Price Chart, and Trades sections. The Order Book table shows various DAI/ETH orders with prices ranging from 0.002400000 to 0.005001000. The Price Chart shows price levels at 1.00, 0.75, 0.50, 0.25, and 0.00. The Trades section lists recent trades for DAI/ETH, DAI, and ETH. Below the main trading area, there's a 'New Order' form and a 'My Transactions' section with tabs for Important, Trades, Orders, and Funds. A sidebar on the left shows the user's balance and a list of tokens like ZAP, ZCG, ZDR, ZIL, ZIP, ZRX, ZSC, ZXBT, cV, eGO, ePRX, and eosDAC. A bottom note advises users to deposit funds via the Deposit form.

DAI/ETH	DAI	ETH
0.002400000	398.804	0.957
0.004000000	4.985	0.020
0.005001000	1.000	0.005
0.002760000	111.366	0.307
0.002750000	149.551	0.411
1.000000000	0.063	0.063
0.005900000	9.144	0.054
0.005880000	64.880	0.381
0.003110200	7.975	0.025
1.000000000	0.001	0.001
0.003150000	26.800	0.084
0.003100000	25.000	0.077
0.002308981	335.959	0.776
0.003200000	60.818	0.195
0.003500000	49.850	0.174
0.004000000	62.313	0.249
0.004000000	102.167	0.409
0.004088990	51.000	0.209

Dann registrieren wir das neue Token mit den uns bereits bekannten Daten.

The screenshot shows the ForkDelta application interface. A modal window titled "Other token" is open, prompting for token registration details. The "Address" field contains the value `0x54093F2C720b18Fd795645b5101A37EB49d1A94a`. The "Name" field is filled with `MOGY`. The "Decimals" field is set to `18`. Three yellow arrows point from the left towards these three input fields. The background of the application shows a balance section with DAI and ETH amounts, and a "Trades" section listing recent transactions.

In diesem Moment wird das neue Token im oberen linken Teil der Börse erscheinen, wir haben es nur in die Börse hochgeladen, wir müssen es veröffentlichen, damit es jeder kaufen und sehen kann. Jetzt müssen wir etwas Äther (0,015) von unserem Konto an die Börse einzahlen.

The image consists of two side-by-side screenshots of the ForkDelta website, both titled "ForkDelta MOGY".

Left Screenshot:

- Balance Section:** Shows a table with three rows: "Token" (MOGY), "Wallet" (1000000000.000), and "ForkDelta" (0.000). Below this is a "Deposit" button.
- Deposit Form:** An "Amount" input field contains "MOGY" and a "Deposit" button.
- ETH Section:** Shows "ETH" with an "Amount" input field containing "0.026" and a "Deposit" button.
- Instructions:** A note at the bottom says "Make sure MOGY is the token you actually want to trade." and a "Volume" section below it has a search bar.

Right Screenshot:

- Balance Section:** Same table as the left, but the "ForkDelta" column now shows "0.000".
- Deposit Form:** The "Amount" input field now contains "0.015" and a "Deposit" button.
- ETH Section:** The "Amount" input field now contains "0.000" and a "Deposit" button.
- Instructions:** A note at the bottom says "Use this to deposit from your personal Ethereum wallet ("Wallet" column) to the current smart contract ("ForkDelta" column.)." and a "Volume" section below it has a search bar.

Da wir die Einzahlung vorgenommen haben, können wir so viele Wertmarken an der Börse hinterlegen, wie wir wollen www.forkdelta.app

Um eine bestimmte Anzahl von Marken einzuzahlen, müssen wir einen Kaufauftrag erstellen. Dies geschieht unten, wo "Neuer Auftrag" steht, und wir klicken auf die Option "Verkaufen". Dann geben wir die Anzahl der Marken ein, die wir jedem Käufer in der Welt zur Verfügung stellen wollen, den Preis in Äther, den wir für jede einzelne verkaufen wollen (Stückpreis), und der Parameter "Verfällt" ist die Zeitspanne, die wir diese Marken zum Verkauf haben wollen:

Verfällt Zeit = 14 Sekunden X eingegebener Betrag.

Beispiel: 14 Sekunden X 10000 = 140.000 Sekunden = 1,62 Tage

The screenshot shows the ForkDelta app interface for trading MOGY tokens. The top navigation bar includes back, forward, refresh, and home icons, along with a URL bar showing <https://forkdelta.app/#!/trade/0x54093f2c720b18fd795>. The main header says "ForkDelta MOGY".

Balance section:

Deposit	Withdraw	Transfer
Token	Wallet	ForkDelta
MOGY	1000000000.000	0.000
Amount		Deposit
ETH	1.057	0.00115
		Deposit

A note at the bottom of this section: "Make sure MOGY is the token you actually want to trade. Multiple tokens can share the same name."

Volume section:

	Search for name, symbol, or address
ASTRO	0.001156789
AstroTokens	
Bid 0.001133300	5.208 ⓢ Daily
Ask 0.001587654	
REQ	0.000047998
Request Token	
Bid 0.000025520	1.295 ⓢ Daily
Ask 0.000047998	
SXDT	0.000200010
Spectre.ai D-Token	
Bid 0.000203040	1.057 ⓢ Daily
Ask 0.000203040	
SNOV	0.000004000
Snovio	
Bid 0.000002002	0.801 ⓢ Daily
Ask 0.000008990	
POE	0.000001112
Po.et	
Bid 0.00000155	0.528 ⓢ Daily
Ask 0.000003330	

Order Book section:

MOGY/ETH	MOGY	ETH
There are no orders here.		

New Order form (highlighted with yellow arrows):

- Sell** (Buy/Sell switch)
- MOGY** (Token input)
- 0.05** (Price input)
- ETH** (Currency input)
- 500.000** (Amount input)
- Expires** (Time input)
- 10000** (Expires input)
- Sell** (Submit button)

Dort werden Ihre neuen Wertmarken zum Verkauf angeboten, jeder kann hereinkommen und sie kaufen. Manchmal erkennt es die neuen Token nicht, so dass sie über die Metamask-Anwendung verkauft werden müssen.

Beispiel einer Konsultation des neu geschaffenen Tokens auf der Website www.etherscan.io.

The screenshot shows the Etherscan interface for a specific transaction. The transaction hash is 0xd99ff4d5f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882. The status is marked as 'Success'. The transaction was included in block 11240201, which has 224 block confirmations. The timestamp is 47 mins ago (Nov-12-2020 02:49:56 AM +UTC), and it was confirmed within 30 seconds. The transaction originated from the address 0x4b7355fd05be6dac458b004f54e12d6527a54a58 and was sent to a contract address [Contract 0x54093f2c720b18fd795645b5101a37eb49d1a94a Created]. The value transferred was 0 Ether (\$0.00). The transaction fee was 0.011014691 Ether (\$5.06), and the gas price was 0.000000041 Ether (41 Gwei). The gas limit was 500,000, and the gas used by the transaction was 268,651 (53.73%).

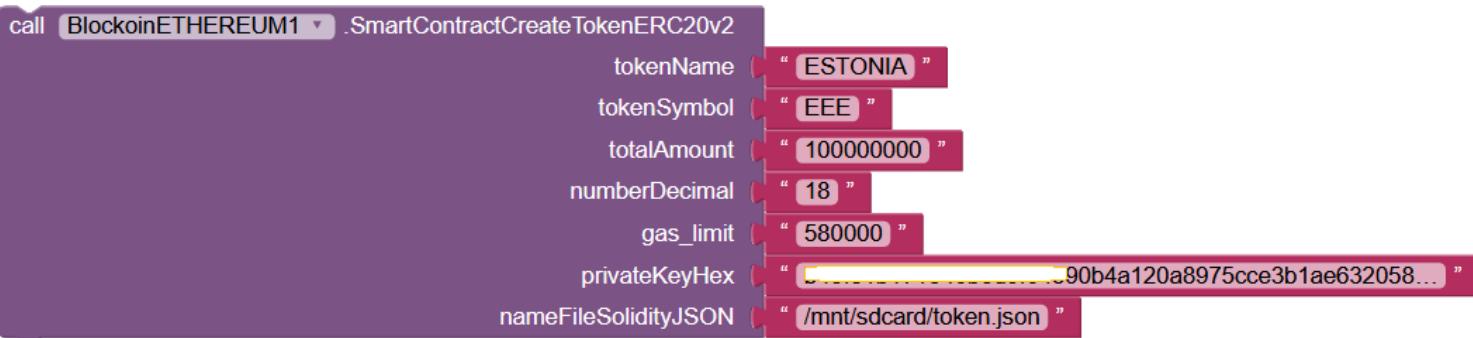
Transaktion (Tx_hash) der Gründung des Ethereum-Netzwerks.

Vertragsadresse neu erstelltes Token.

**Nur Kosten für das Ethereum-Netzwerk.
Beinhaltet nicht die Betriebskosten + 15**

Overview	Status	Comments
② Transaction Hash:	0xd99ff4d5f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882	🔗
② Status:	Success	
② Block:	11240201	224 Block Confirmations
② Timestamp:	47 mins ago (Nov-12-2020 02:49:56 AM +UTC)	Confirmed within 30 secs
② From:	0x4b7355fd05be6dac458b004f54e12d6527a54a58	🔗
② To:	[Contract 0x54093f2c720b18fd795645b5101a37eb49d1a94a Created]	✓
② Value:	0 Ether (\$0.00)	
② Transaction Fee:	0.011014691 Ether (\$5.06)	
② Gas Price:	0.000000041 Ether (41 Gwei)	
② Gas Limit:	500,000	
② Gas Used by Transaction:	268,651 (53.73%)	

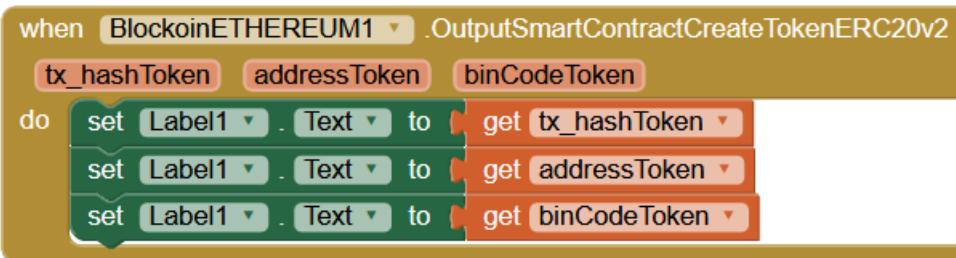
Block zum Kompilieren, Erstellen und Veröffentlichen von ERC20-Token - (SmartContractCreateTokenERC20v2)



Parámetros de entrada: **tokenName** <Zeichenkette>, **tokenSymbol** <Zeichenkette>, **totalAmount** <Zeichenkette>, **numberDecimal** <Zeichenkette>, **gas_limit** <Ganzzahl>, **privateKeyHex** <Ganzzahl>, **nameFileSolidityJSON** <Zeichenkette>.

Ausgabeparameter: Ereignis (OutputSmartContractTokenERC20v2).

Ausgaben: **tx_hashToken**<Zeichenkette> , **addressToken**<Zeichenkette> , **binCodeToken**<Zeichenkette>.



Beschreibung: Intelligenter Vertrag "Token ERC20" - im Ethereum-Netzwerk veröffentlichte Anlage. In der Version 2 (v2) kann der Wert des Gaslimits optimiert werden, denn je nach Smart Contract hängt es davon ab, wie schnell Sie die Veröffentlichung im Ethereum-Netz vornehmen wollen. Es wird empfohlen, dass mindestens ein Wert von 350.000 WEI für die Publikation ohne Ausfall oder Verzögerung erreicht werden sollte.

Der Unterschied zwischen den Funktionen der Erstellung von TokenERC20v1 und der Erstellung von TokenERC20v2 besteht darin, dass bei Version 1 der Parameter von GasLimit bereits vorkonfiguriert ist und bei Version 2 an die Bedürfnisse des Benutzers oder Entwicklers angepasst werden kann.

Block zum Aufrufen oder Ausführen von ERC20-Token - (**SmartContractExecution**)



Eingabeparameter: AdresseSmartContract<Zeichenkette>, NameDateiSolidityJSON<Zeichenkette>, FunktionAusführungSmartContract<Zeichenkette>.

Ausgabeparameter: Ausführung der im referenzierten Smart-Vertrag angegebenen Funktion.

Outputs: Ausführung eines intelligenten Vertrags.

HINWEIS: Um ausgeführt werden zu können, muss eine Datei im JSON-Format erstellt werden, die die Parameter des Primärschlüssels der Adresse enthält, an der Sie den Smart-Vertrag ausführen möchten; Sie müssen das Gaslimit (WEI) eingeben.

Beispiel einer JSON-Datei zur Ausführung der Funktionen des kompilierten Smart-Vertrags im Beispiel der oben erwähnten Compiler-Funktion. Der Dateiname kann frei gewählt werden.

Datei: call.json

```
{
  "privat": "3ca40...",
  "gas_limit": 20000
}
```

Beispiel für die Ausführungsangabe der "Gruß"-Funktion des Smart-Vertrags:

```
{
  "gas_limit": 20.000,
  "address": "0eb688e79698d645df015cf2e9db5a6fe16357f1",
  "Ergebnisse": [
    "Hallo BlockCypher-Test"
  ]
}
```

ERC20 Token-Eigenschaftsanzeige-Block - (**SmartContractGetCreationTx**)

call **BlockoinETHEREUM1** .SmartContractGetCreationTx

txSmartContract

" d99ff4d53f9b1f0687289674633d2acecf219011d163cdc0..." "

Eingabeparameter: **txSmartContract<String>**

Ausgabeparameter: Zeigt Eigenschaften des Smart-Vertrags an, auf den mit (**Tx_hash**) verwiesen wird.

Beschreibung: Zeigt die Hauptmerkmale und den ABI-Code des referenzierten Smart-Vertrags.

Beispiel-Eigenschaften des ERC20-Tokens, Beispiel-TokenName "MOGY", das zuvor mit der Funktion erstellt wurde (**martContractCreateTokenERC20v1**)

```
{
  "block_hash": "cadb8914c43ed28fb370c9e1b6f5d8818ba31a1e944d1f7049441b982902dea8",
  "Block_Höhe": 11240201,
  "block_index": 170,
  "hash": "d99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882",
  "Adressen": [
    "4b7355fd05be6dac458b004f54e12d6527a54a58"
  ],
  "insgesamt": 0,
  "Gebühren": 110146910000000,
  "Größe": 958,
  "gas_limit": 500.000,
  "Gas_verwendet": 268651,
  "Gas_Preis": 41000000000000,
  "contract_creation": wahr,
  "weitergeleitet_von": "200.77.24.87",
}
```



```

    }
]
```

Block zur Anzeige des ERC20-Token-Kompilierungscodes - (**SmartContractGetcodeABI**)

call BlockoinETHEREUM1 .SmartContractGetcodeABI
addressSmartContract " 0eb688e79698d645df015cf2e9db5a6fe16357f1 "

Eingabeparameter: **AdresseSmartContract<Zeichenkette>**

Ausgabeparameter: Zeigt den referenzierten Smart-Vertragscode an.

Beschreibung: Zeigt den ABI-Code des referenzierten Smart-Vertrags an.

Beispiel für den ABI-Code eines generischen Smart-Vertrags:

```
{
  "Solidität": "contract mortal {\n    /* Variable owner vom Typ address*/\n    address owner definieren;\n    /* diese Funktion wird bei der\n    Initialisierung ausgeführt und setzt den Eigentümer des Vertrages */\n    function mortal() { owner = msg.sender; }\n    /* Funktion zur\n    Rückgewinnung der Gelder auf dem Vertrag */\n    function kill() { if\n        (msg.sender == owner) suicide(owner); }\n    \n    contract greeter is mortal\n    /* definieren variable Begrüßung vom Typ string */\n    string greeting;\n    /* dies läuft bei der Ausführung des Vertrages ab */\n    Funktion greeter(string _greeting) public {\n        greeting =\n        _greeting;\n    }\n    /* Hauptfunktion */\n    Funktion greet()\n    konstante Rückgaben (string) {\n        return greeting;\n    }\n    \n    \"Mülleimer\":\n    \"606060405260405161023e38038061023e8339810160405280510160008054600160a060\n    020a031916331790558060016000509080519060200190828054600181600116156101000\n    203166002900490600052602060002090601f016020900481019282601f10609f57805160\n    ff19168380011785555b50608e9291505b808211560cc57600081558301607d565b50505\n    061016e806100d06000396000f35b828001600101855582156076579182015b828111560\n    7657825182600050559160200191906001019060b0565b509056606060405260e060020a6\n    00035046341c0e1b58114610026578063cfiae321714610068575b005b6100246000543373\n    fffffffffffffffffffffcccccccccccccccccccccccccccccccccccccccccccccccccccccccc\n    ffffffcccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc\n    a06020601f600260001961010086881615020190941693909304928301819004028101604\n    0526080828152929190828280156101645780601f10610139576101008083540402835291\n    60200191610164565b6040518080602001828103825283818151815260200191508051906\n    0200190808383829060006004602084601f0104600f02600301f150905090810190601f16\n    80156101295780820380516001836020036101000a031916815260200191505b509250505\n    06\n    \"abi\":\n    \"[{\\"constant\":false,\\"inputs\":[],\\"name\":\"kill\",\\\"outputs\":[],\\\"ty\n    pe\\\":\\\"function\\\"}, {\\"constant\":true,\\"inputs\":[],\\"name\":\"greet\\\",\\\"o\n    utputs\\\":[{\\\"name\\\":\\\"\\\",\\\"type\\\\":\\\"string\\\"}],\\\"type\\\\":\\\"function\\\"}, {\\"i\n    nputs\\\":[{\\\"name\\\\":\\\"_greeting\\\",\\\"type\\\\":\\\"string\\\"}],\\\"type\\\\":\\\"const\n    ructor\\\"}]\",\n    \"creation_tx_hash\":\n    \"61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9\",
```

```
"created": "2016-07-20T01:54:50Z",
"address": "0eb688e79698d645df015cf2e9db5a6fe16357f1"}
```

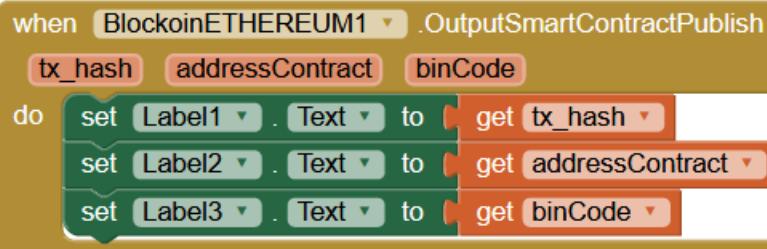
*Vor der Verwendung des folgenden Blocks (**SmartContractPublish**) müssen Sie den Block (**SmartContractCompilerSolidity**) verwenden, um zu prüfen, ob der Smart-Vertrag gut geschrieben ist.

Block zur Veröffentlichung des ERC20-Tokens im Ethereum-Netzwerk -
/**SmartContractPublish**

```
call [BlockoinETHEREUM1 ▾] .SmartContractPublish
      nameFileSolidityJSON " /mnt/sdcard/PublishSmartContract.json "
```

Eingabeparameter: **nameDateiSolidityJSON<Zeichenkette>**

Ausgabeparameter: Zeigt die Eigenschaften des referenzierten Smart-Vertrags an. In der Veranstaltung (**OutputSmartContractPublish**).



```
when [BlockoinETHEREUM1 .OutputSmartContractPublish]
  do
    set Label1 to get tx_hash
    set Label2 to get addressContract
    set Label3 to get binCode
```

Beschreibung: Veröffentlichen Sie im ethereum-Netzwerk den Smart-Vertrag, auf den in der JSON-Datei verwiesen wird.

Beispieldatei: **PublishSmartContract.json**

```
{
  "Solidität": "contract mortal {\n    /* Variable owner vom Typ address*/\n    address owner definieren;\n    /* diese Funktion wird bei der\n    Initialisierung ausgeführt und setzt den Eigentümer des Vertrages */\n    function mortal() { owner = msg.sender; }\n    /* Funktion zur\n    Rückgewinnung der Gelder auf dem Vertrag */\n    function kill() { if\n        (msg.sender == owner) suicide(owner); }\n    }\n    \n    contract greeter is mortal\n    {\n        /* definieren variable Begrüßung vom Typ string */\n        string greeting;\n        /* dies läuft bei der Ausführung des Vertrages ab */\n        Funktion greeter(string _greeting) public {\n            greeting = _greeting;\n        }\n        /* Hauptfunktion */\n        Funktion greet()\n        konstante Rückgaben (string) {\n            return greeting;\n        }\n    }\n    \n    \"params\": \"Hallo Test\", \n    \"veröffentlichen\": \"Begrüßer\", \n    \"privat\": \"3ca40...\" \n    \"gas_limit\": 500000\n}
```

Wie im obigen Code gezeigt, handelt es sich um denselben JSON-Code, der im Beispiel der Kompilierfunktion verwendet wird, und zwar um denselben Code, dem die Parameter am Ende der JSON-Datei hinzugefügt wurden:

Params: Implizite Parameter des Smart contr.

Veröffentlichen: Name, wie der Smart-Vertrag veröffentlicht wird.

Privat: Der Primärschlüssel des Kontos, das den Smart-Vertrag ausführt, muss einen Saldo aufweisen.

Gas_limit: Ist der Saldo in WEI, den Sie für die Veröffentlichung des Smart-Vertrags ausgeben möchten.

Beispiel für die Ausgabe bei der Ausführung (Veröffentlichung des Smart-Vertrags) der Smart-Vertrag wird in das ethereum-Netzwerk eingegeben. Sie zeigt die durchgeführte Transaktion "**creation_tx_hash**" und die zugewiesene Adresse des angelegten Smart-Vertrags "**address**".

```
[
{
  "Name": "Begrüßer",
  "Solidität": "contract mortal {\n    /* Variable owner vom Typ\naddress*/\n    address owner definieren;\n    /* diese Funktion wird\nbei der Initialisierung ausgeführt und setzt den Eigentümer des Vertrages\n*/\n    function mortal() { owner = msg.sender; }\n    /* Funktion zur\nRückgewinnung der Gelder auf dem Vertrag */\n    function kill() { if\n        (msg.sender == owner) suicide(owner); }\n}\n\ncontract greeter is mortal\n{\n    /* definieren variable Begrüßung vom Typ string */\n    string greeting;\n    /* dies läuft bei der Ausführung des Vertrages ab */\n    function greeter(string _greeting) public {\n        greeting =\n        _greeting;\n    }\n    /* Hauptfunktion */\n    function greet()\n        konstante Rückgaben (string) {\n            return greeting;\n        }\n    \"\n    \"Mülleimer\":\n"
"606060405260405161023e38038061023e8339810160405280510160008054600160a060
020a031916331790558060016000509080519060200190828054600181600116156101000
203166002900490600052602060002090601f016020900481019282601f10609f57805160
ff19168380011785555b50608e9291505b8082111560cc57600081558301607d565b50505
061016e806100d06000396000f35b828001600101855582156076579182015b8281111560
7657825182600050559160200191906001019060b0565b509056606060405260e060020a6
00035046341c0e1b58114610026578063cfae321714610068575b005b6100246000543373
fffffffffffffffffffff908116911614156101375760005473fff
fffffffffffffffffffff16ff5b6100c9600060609081526001805460
a06020601f600260001961010086881615020190941693909304928301819004028101604
0526080828152929190828280156101645780601f10610139576101008083540402835291
60200191610164565b6040518080602001828103825283818151815260200191508051906
0200190808383829060006004602084601f0104600f02600301f150905090810190601f16
80156101295780820380516001836020036101000a031916815260200191505b509250505
06
  "abi": [
    {
      "Konstante": falsch,
      "Eingaben": [],
      "Name": "töten",
      "Ausgänge": [],
      "Typ": "Funktion".
    },
    {
      "Konstante": wahr,
      "Eingaben": []
    }
  ]
}
```

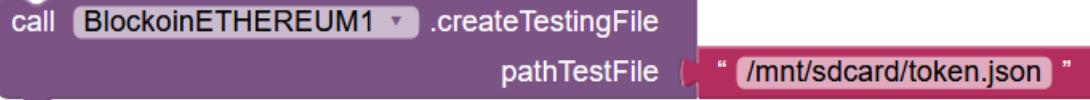
```

    "Name": "Gruß",
    "Ausgänge": [
        {
            "Name": "",
            "Typ": "Zeichenfolge".
        }
    ],
    "Typ": "Funktion".
},
{
    "Eingaben": [
        {
            "Name": "_Gruß",
            "Typ": "Zeichenfolge".
        }
    ],
    "Typ": "Erbauer".
}
],
"gas_limit": 500.000,
"creation_tx_hash":
"61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9",
"address": "0eb688e79698d645df015cf2e9db5a6fe16357f1",
"params": [
    "Hallo Test"
]
},
{
    "Name": "sterblich",
    "Solidität": "contract mortal {\n/* Variable owner vom Typ\naddress*/\n    address owner definieren;\n    /* diese Funktion wird\nbei der Initialisierung ausgeführt und setzt den Eigentümer des Vertrages\n*/\n    function mortal() { owner = msg.sender; }\n    /* Funktion zur\nRückgewinnung der Gelder auf dem Vertrag */\n    function kill() { if\n(msg.sender == owner) suicide(owner); }\n}\n\ncontract greeter is mortal\n    /* definieren variable Begrüßung vom Typ string */\n    string greeting;\n    /* dies läuft bei der Ausführung des Vertrages ab */\n    Funktion greeter(string _greeting) public {\n        greeting =\n        _greeting;\n    }\n    /* Hauptfunktion */\n    Funktion greet()\nkonstante Rückgaben (string) {\n        return greeting;\n    }\n",
    "Mülleimer": "
"606060405260008054600160a060020a03191633179055605c8060226000396000f36060
60405260e060020a600035046341c0e1b58114601a575b005b60186000543373ffffffffffff
ffffffffffffffffffff90811691161415605a5760005473fffffffffffff
ffffffffffffffffffff16ff5b56",
    "abi": [
        {
            "Konstante": falsch,
            "Eingaben": [],
            "Name": "töten",
            "Ausgänge": [],
            "Typ": "Funktion".
        },
        {
            "Eingaben": [],
            "Typ": "Erbauer".
        }

```

```
],
  "gas_limit": 500.000,
  "params": "Hallo Test"
}
]
```

Testblock zum **Erstellen einer Datei** - (**createTestingFile**)

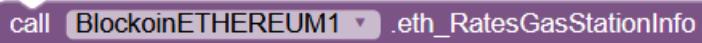


Eingabeparameter: **pathTestFile<String>**

Ausgabeparameter: Im referenzierten Pfad wird eine Testdatei erstellt.

Beschreibung: Dies dient dazu, den gültigen temporären Dateierstellungspfad zu prüfen, um sicherzustellen, dass er bei Verwendung des Blocks (**SmartContractCreateTokenERC20v1**) oder des Blocks (**SmartContractCreateTokenERC20v2**) korrekt ist.

Blockieren, um die Tarife des Gaspreises zu erhalten - (**eth_RatesGasStationInfo**).



Eingabeparameter: **nameDateiSolidityJSON<Zeichenkette>**

Ausgabeparameter: Zeigt die Eigenschaften des referenzierten Smart-Vertrags an. Im Ereignis (**OutputEth_GasStationInfo**) werden die gelieferten Werte in GWEI angegeben.

Der Gaspreis ist der Wert, der an die Systeme gezahlt wird, die die Transaktionen im Netz des Ethereum ausführen. Diese Systeme sind allgemein als "Bergleute" bekannt, und die Werte des Gaspreises hängen davon ab, wie schnell (Zeit und Priorität) die Transaktion im Netz des Ethereum durchgeführt wird.

Die gelieferten Werte basieren auf den folgenden Laufzeiten. Diese Zeiten sind ungefähre Angaben und können variieren, je nachdem, wie Sie mit den Anforderungen (Transaktionen) im Ethereum-Netzwerk zurechtkommen.

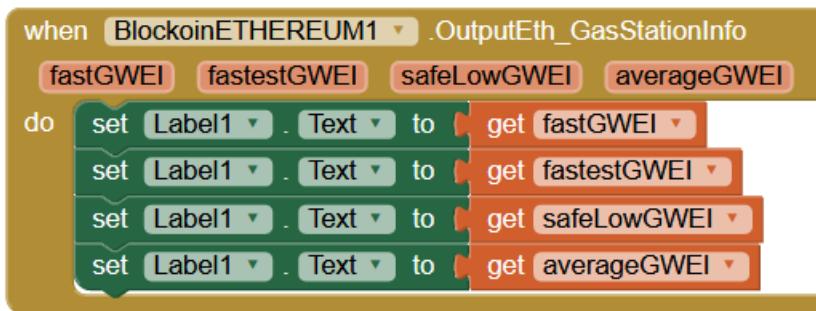
Schnell < 2 Minuten.

Schnellste < 30 Sekunden.

SafeLow < 30 Minuten.

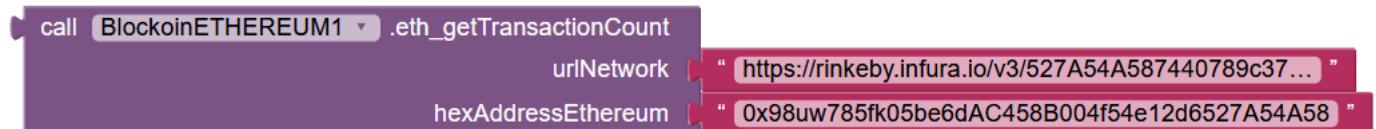
Durchschnittlich < 15 Minuten.

Transaktionen, die mit der Exchange Ethereum Extension (EEE) getätigten werden, verwenden immer den Gaspreis = Durchschnitt.



Beschreibung: Ruft den aktualisierten Gaspreis zum Zeitpunkt der Abfrage ab, um eine neue Transaktion zu erstellen

Block, um die Zahl "nonce" zu erhalten - (`eth_getTransactionCount`).



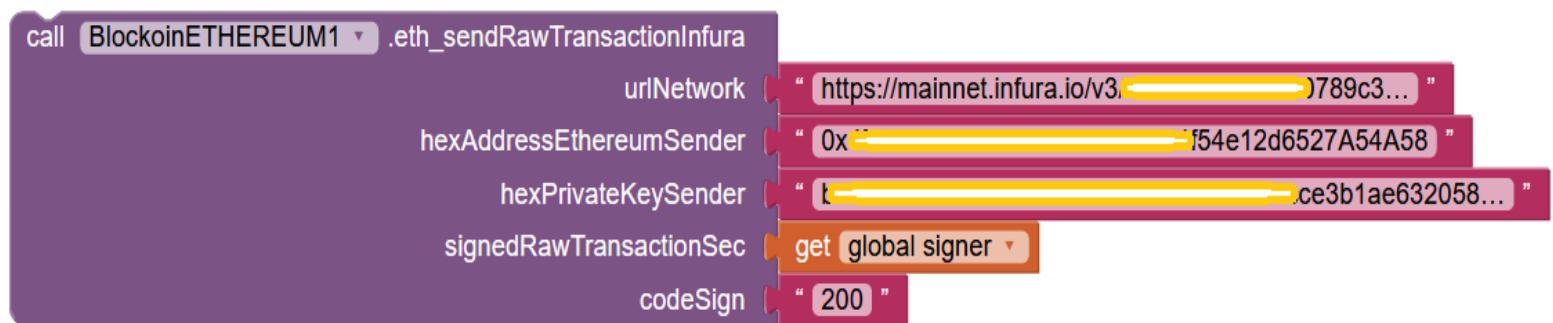
Eingabeparameter: `urlNetzwerk<Zeichenkette>`, `hexAdresseHierarchie<Zeichenkette>`.

Ausgabeparameter: Sie zeigt im hexadezimalen Format die laufende Nummer "nonce" der referenzierten Adresse an.

Die "Nonce"-Nummer ist eine inkrementelle Zahl, die die Anzahl der Transaktionen verfolgt, die von einer bestimmten Adresse aus getätigten wurden.

Beschreibung: Ermittelt die "Nonce"-Nummer der referenzierten Adresse.

Block zum Senden einer signierten Transaktion - (`eth_SendRawTransactionInfura`).

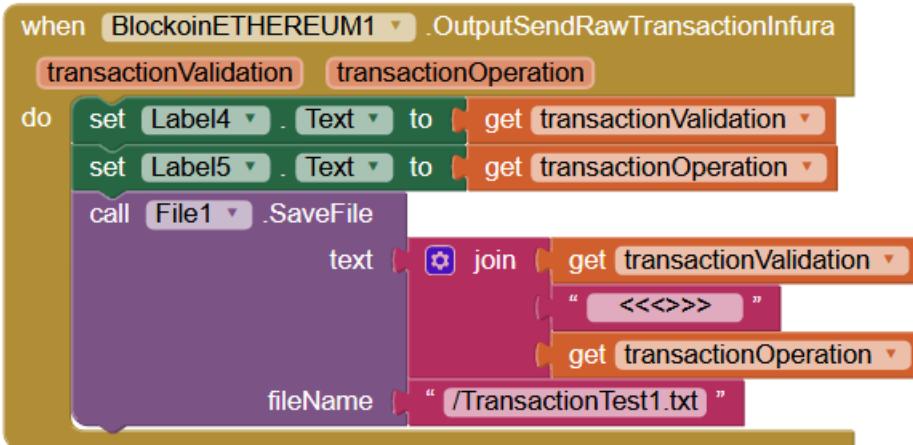


Erforderliche Abhängigkeit(en): Block (eth_getTransactionCount), Block (SignerGenericPushRawTransactionOffline). Überprüfen Sie das Beispiel des Blocks (SignerGenericPushRawTransactionOffline).

Eingabeparameter: urlNetzwerk <Zeichenkette>, hexAddressEthereumSender <Zeichenkette>, hexPrivateKeySender <Zeichenkette>, SignedRawTrasactionSec <Zeichenkette>, codeSign <Zeichenkette>.

Ausgabeparameter: Ereignis (OutputSendRawTransactionInfura)

Ausgaben: transactionValidation<String> , transactionOperation<String>.



Beschreibung: Es liefert als Ergebnis der Transaktionen zwei hexadezimale Werte. Der Wert transactionValidation ist die Transaktion, die über das Ethereum-Netzwerk durchgeführt wird und die impliziten Kosten für das Ethereum enthält. Der Wert der TransactionOperation ist die im Coinsolidation-Netzwerk durchgeführte Transaktion mit Kosten von 0,5 Cent USD für jede Transaktion, basierend auf dem Wert des Äthers zum Zeitpunkt der Transaktion. Diese Kosten werden für die Bezahlung von Dienstleistungen des Coinsolidation.org-Netzwerks verwendet und in die Wartung, Unterstützung und Erstellung von Erweiterungen für den Krypto- und Anlagensektor weltweit investiert.

Block zur Berechnung der Kosten einer Standard-Transaktion -
(eth VerifyBalanceForTransaction)



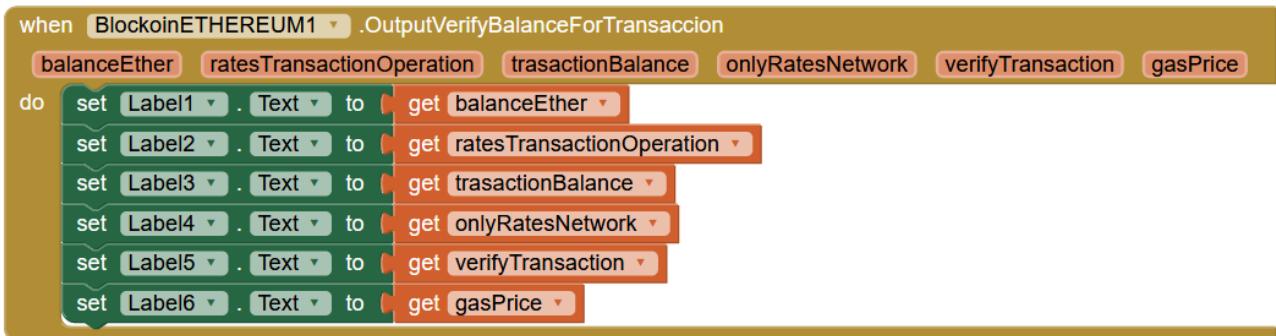
Eingabeparameter:

AdresseEthereumSender<Zeichenkette>,

WertEthertoSend<Zeichenkette>.

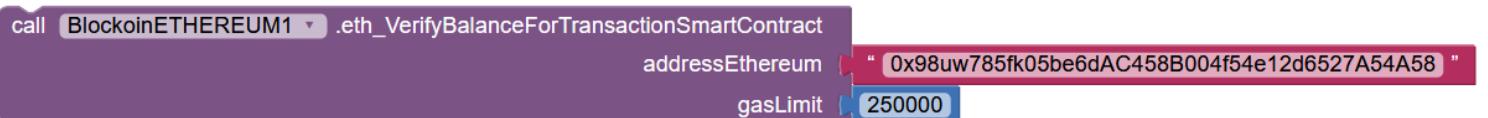
Ausgabeparameter: Ereignis (**OutputVerifyBalanceForTransaction**).

Ausgaben: **balanceEther<String>** , **ratesTransactionOperation<String>** , **trasactionBalance<String>** , **onlyRatesNetwork<String>** , **verifyTransaction<String>** , **gasPrice<String>**.



Beschreibung: Enthält Einzelheiten darüber, wie hoch die Kosten einer Standardtransaktion in Bezug auf die Einreiseadresse sein werden. Der Ausgabeparameter "verifyTransaction" sagt uns, ob die Transaktion "True" gemacht werden kann, oder ob die referenzierte Adresse nicht genügend Guthaben hat, um uns ein "False" zu geben.

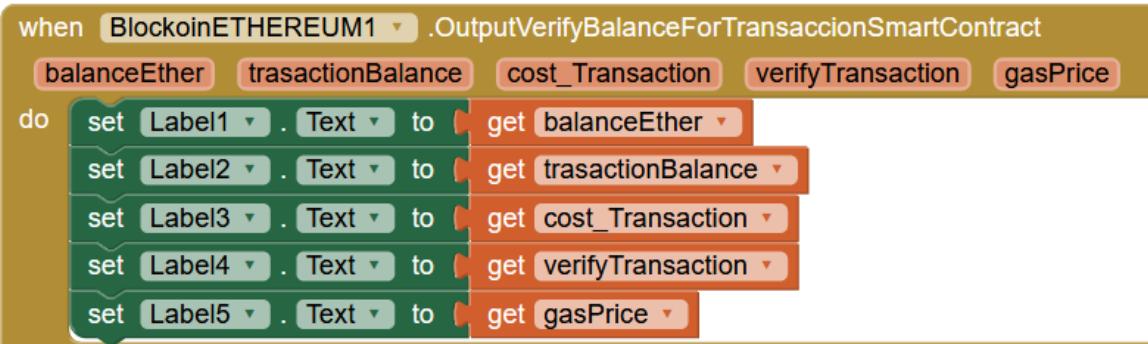
Block zur Berechnung der Kosten einer Standard-Transaktion -
(eth_VerifiBalanceForTransaccionSmartContract)



Eingabeparameter: **AdresseEthereum<Zeichenkette>**, **gasGrenze<Zeichenkette>**.

Ausgabeparameter: Ereignis (**OutputVerifyBalanceForTransaccionSmartContract**)

Ausgaben: **balanceEther<String>** , **trasactionBalance<String>** , **cost_Transaction<String>** , **verifyTransaction<String>** , **gasPrice<String>**.



Beschreibung: Enthält Einzelheiten darüber, wie hoch die ungefähren Kosten einer Standardtransaktion für den Versand eines **Smart-Vertrags** wären, mit Verweis auf die Einreiseadresse. Der Ausgabeparameter "verifyTransaction" sagt uns, ob die Transaktion "True" gemacht werden kann, oder ob die referenzierte Adresse nicht genügend Guthaben hat, um uns ein "False" zu geben.

balanceEther: Der Saldo der referenzierten Adresse wird in Äthern geliefert.

trasactionBalance: Saldo, nachdem die Transaktion durchgeführt wurde.

cost_Transaction: Sind die Kosten der Transaktion für die Veröffentlichung des Smart-Contrats.

verifyTransaction: (balanceEther minus cost_Transaction).

gasPrice: Aktueller Wert des von den "Miners" verwendeten GasPrice, dieser kann jede Minute variieren.

Blockieren, um Ether-Preis in der Währung des angegebenen Landes zu erhalten - (**eth_getExchangeRates**).



Eingabeparameter: **countryRates<String>**. Überprüfen Sie den Ausgabeparameter "country", wo er alle Landeswährungstypen enthält, um den gewünschten zu wählen.

Ausgabeparameter: Ereignis (**OutputEth_getExchangeRates**).

Ausgaben: **rates<String>, countries<String>** Ausgabe im JSON-Format alle Raten der Länder der Welt.



Beschreibung: Sie liefert den aktuellen Preis eines Äthers zum Wechselkurs der Währung des referenzierten Landes.

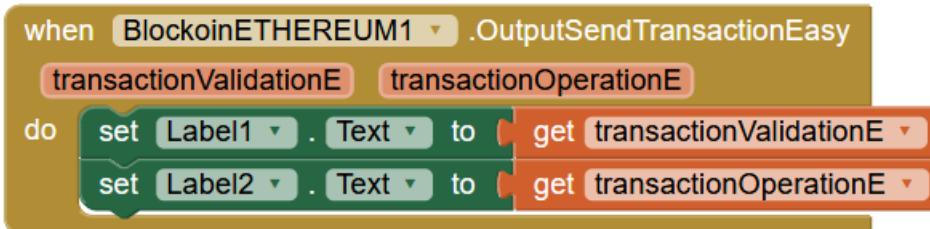
Block zur Durchführung einer Standardtransaktion mit den minimalen Eingabeparametern - (eth_sendTransactionEasy).



Eingabeparameter: hexPrivateKeySender<Zeichenkette>, bisAdresse<Zeichenkette>, WertÄther<Zeichenkette>.

Ausgabeparameter: Ereignis (OutputSendTransactionEasy)

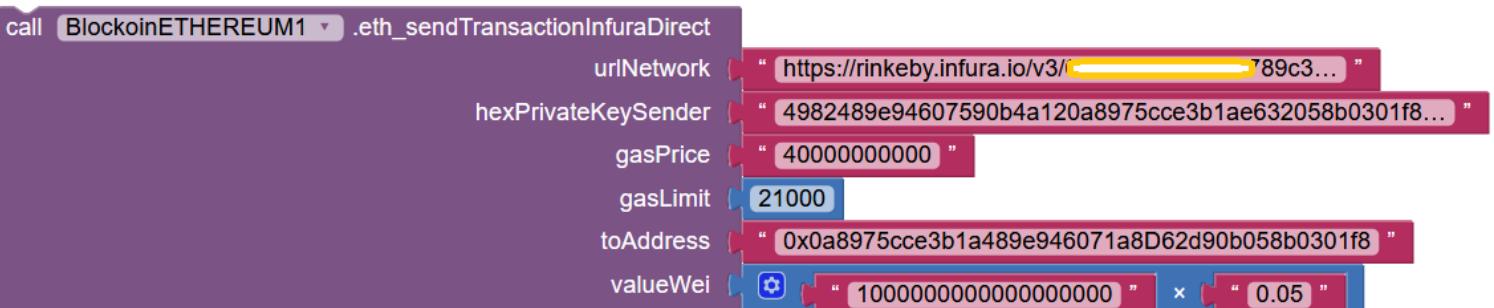
Ausgaben: transactionValidation<String>, transactionOperation<String>.



Beschreibung: Funktion zur Durchführung einer Standardtransaktion im Netzwerk von Ethereum, diese Funktion ist von unmittelbarem Nutzen, Sie brauchen kein Konto in INFURA zu haben und Sie brauchen nur 3 Eingabeparameter, Sie brauchen nur genug Guthaben zu haben, um die gewünschte Transaktion durchzuführen.

Transaktionen werden im Ethereum-Netzwerk direkt über die offizielle Ethereum Web3j-Bibliothek und unser Netzwerk Coinsolidation.org getätigkt.

Block zur Durchführung einer Standardtransaktion mit den minimalen Eingabeparametern - (eth_sendTransactionInfuraDirect).



Eingabeparameter: `urlNetzwerk<String>`, `hexPrivateKeySender<String>`, `gasPrice<String>`, `gasLimit<String>`, `toAddress<String>`, `valueWEI<String>`.

Ausgabeparameter: Ereignis (`OutputSendTransactionInfuraDirect`)

Ausgaben: `transactionValidation<String>` , `transactionOperation<String>`.



Beschreibung: Funktion zum Senden einer Standardtransaktion, die bereits die implizite digitale Signatur enthält. Dies ist nützlich für Personen, die bereits Vorkenntnisse über die Komponenten einer Transaktion haben und diese Parameter nach ihren Bedürfnissen optimieren möchten.

11. Berechnung von Standard-Transaktionskosten und Smart contract transaction

Für die Berechnung einer Standardtransaktion werden 3 Parameter im Ethereum-Netzwerk benötigt.

- 1.- **Gaspreis**.
- 2.- **Gas-Grenzwert**.
- 3.- **Aktueller Wert eines Äthers** (Ethereum digitale Währung).

Gaspreis: Dieser wird normalerweise in Einheiten von GWEI (GigaWEI) angegeben. Dies entspricht, wenn 1 Äther 1.000.000.000.000.000.000 wei hat, dann ist 1 GWEI gleich 1.000.000.000. Diese Einheit dient als Zahlungsmittel für die Systeme, die alle Transaktionen des Ethereum-Netzwerks ausführen und "Bergleute" genannt werden, sie ist über die ganze Welt verteilt. Der Gaspreis ist kein fester Wert, sondern variabel und kann sich von Minute zu Minute oder Sekunde ändern. Diejenigen, die den Wert des Gaspreises definieren, sind die "Bergleute" und es hängt davon ab, wie gesättigt das Ethereum-Netz ist.

GasLimit: Dieser Wert wird normalerweise in WEI-Einheiten angegeben, und bei einer Standard-Transaktion liegt der durchschnittliche Standardwert bei 21.000 WEI, obwohl er höher oder niedriger sein kann, je nachdem, welche Art von Transaktion Sie durchführen möchten. Bei einer Standard-Transaktion verwenden wir den Wert für 40.000 WEI, um sicherzustellen, dass wir keine Ablehnung haben, weil wir unter dem Gas-Limit liegen.

Der Wert des Äthers: Dieser Wert ist ebenfalls variabel und ist auf verschiedene Parameter eines globalen Finanzmarktes zurückzuführen. Dieser Wert kann von Einheiten bezogen werden, die den Wert des Äthers weltweit schon immer aktualisiert haben, die als zentralisierte und dezentralisierte Börsen bezeichnet werden.

WICHTIGE ANMERKUNG: In der Exchange Ethereum Extension (EEE) verwenden wir ein höheres Gaslimit (40.000), was darauf zurückzuführen ist, dass im Falle der Nichterreichung der von den "Bergleuten" festgelegten Mindestquote die TRANSAKTION NICHT GESCHÄFTIGT WIRD, jedoch das Ethereum-Netz, wenn es den Aufwand, den es bei der Berechnung der Transaktion gemacht hat, in Rechnung stellt, ohne sie zu machen, Aus diesem Grund erklären einige Nutzer nicht, warum ihre Transaktion nicht ausgeführt wird, jedoch wird ihnen ein Betrag oder das gesamte GasLimit in Rechnung gestellt, das zum Zeitpunkt der Transaktionsanfrage angeboten wurde, aus diesem Grund müssen wir uns immer darüber im Klaren sein, wie die Werte von GasLimit und Gaspreis aussehen werden.

Der GasPreis kann mit der Funktion `eth_GetRatesGasStation` abgefragt werden und das GasLimit für Standardtransaktionen muss höher als 21.000 WEI sein und Transaktionen zur Veröffentlichung und/oder Ausführung von Smart-Verträgen müssen mindestens 500.000 WEI betragen.

Standard-Transaktionskosten.

Sie wird durch die folgende Formel definiert:

$$\text{Kosten} = (\text{GasLimit} \times (\text{GasPreis} / (1.000.000.000.000.000))) \times \text{Ätherwert}.$$

Beispiel:

Nehmen Sie die folgenden Werte an:

GasLimit = 40.000, GasPreis = 45 GWEI, Ether-Wert = \$406.

$$\text{Kosten} = (40.000 \times (45.000.000.000.000 / (1.000.000.000.000.000.000))) \times 406$$

$$\text{Standard-Transaktionskosten} = 0,0018 \text{ Äther} \times 406 \text{ USD} = \$0,73 \text{ USD}$$

Im Falle einer Transaktion mit einem Smart-Vertrag ist es notwendig zu wissen, welche Art von Smart-Vertrag veröffentlicht wird, da die Kosten direkt proportional zum Arbeitsaufwand sind, den die "Bergleute" für die Bearbeitung des Smart-Vertrags aufbringen müssen, mit anderen Worten, der Umfang der Verarbeitung in den Computersystemen, die die "Bergleute" handhaben, ist einfacher.

Im Falle des Smart-Vertrags wäre ein Standardwert, das GasLimit mit einem Wert von 500.000 WEI zu beginnen.

Ein wichtiger Punkt, den es zu berücksichtigen gilt, ist, dass, wenn man ein GasLimit vorschlägt, die "Bergleute" nicht notwendigerweise den gesamten vorgeschlagenen Betrag nehmen werden, d.h. wenn man eine Transaktion sendet, berechnen die "Bergleute" den Berechnungsaufwand und nehmen das heraus, was aus dem GasLimit herausgenommen wird, das in einigen Fällen weniger oder gleich dem standardmäßig angebotenen GasLimit von 21.000 WEI sein kann.

Alle Transaktionen können auf der Website www.etherscan.io eingesehen werden, wo wir die Einzelheiten jeder Transaktion einsehen können.

Beispiel für eine Standard-Transaktion, bei der die Transaktion mit einem Gas-Limit von 40.000 WEIs gesendet wurde, die "Bergleute" bei der Berechnung der Verarbeitungskosten jedoch nur 52,5%, d.h. den Standardwert von 21.000 WEIs, nahmen.

The screenshot shows the Etherscan Transaction Details page for a specific Ethereum transaction. The transaction hash is 0x7dae251f21c616fb04a94112633c97e04d11c91f4d06dd9b85ed11112f02703a. The transaction was successful and has 2 block confirmations. It was timestamped 52 seconds ago on November 11, 2020, at 06:17:24 AM UTC. The transaction originated from address 0x4b7355fd05be6dac458b004f54e12d6527a54a58 and was sent to address 0x5d2acdb34c279aa6d1e94a77f7b18ab938fb2bb. The value transferred was 0.001084598698481562 Ether (\$0.50). The transaction fee was 0.000672 Ether (\$0.31), which is 21.000 WEI (0.000000032 Ether) multiplied by the gas price of 32 Gwei. The gas limit was set at 40,000, but only 21,000 (52.5%) was used. The gas price was 0.000000032 Ether (32 Gwei).

Parameter	Value	Notes
Transaction Hash	0x7dae251f21c616fb04a94112633c97e04d11c91f4d06dd9b85ed11112f02703a	Transaktionsoperation oder
Status	Success	
Block	11234630	2 Block Confirmations
Timestamp	52 secs ago (Nov-11-2020 06:17:24 AM +UTC)	Confirmed within 38 secs
From	0x4b7355fd05be6dac458b004f54e12d6527a54a58	
To	0x5d2acdb34c279aa6d1e94a77f7b18ab938fb2bb	Transaktionskosten in Äther: $21.000 \times 0,000000032 = 0,000672$ Äther (\$0,31 USD)
Value	0.001084598698481562 Ether (\$0.50)	
Transaction Fee	0.000672 Ether (\$0.31)	Vorgeschlagene Gasbegrenzung:
Gas Price	0.000000032 Ether (32 Gwei)	
Gas Limit	40,000	Tatsächlich bei der Transaktion verwendetes Gas-Limit:
Gas Used by Transaction	21,000 (52.5%)	
Nonce	36	
Position	79	

Wenn wir zur Transaktion des Smart-Vertrags zurückkehren und die 500.000 WEI GasLimit nehmen, erhalten wir die folgenden Kosten, wenn wir alles verwenden.

Intelligente Transaktionskosten für Verträge.

Sie wird durch die folgende Formel definiert:

Kosten = (GasLimit x (GasPreis / (1.000.000.000.000.000))) x Ätherwert.

Beispiel:

Nehmen Sie die folgenden Werte an:

GasLimit = 500.000, **GasPreis** = 45 GWEI, **Ether-Wert** = \$406.

Kosten = (500.000 x (45.000.000.000.000 / (1.000.000.000.000.000.000))) x 406

Transaktionskosten veröffentlichen Smart-Kontrakt = 0,0225 Äther x 406 USD = 9.135 USD

Alle Transaktionen können auf der Website www.etherscan.io eingesehen werden.

HINWEIS: Um die Gesamtkosten der Transaktion zu erhalten, müssen Sie diese addieren:

Gesamttransaktionskosten = *Ethereum-Netzwerkkosten + Coinsolidation.org-Gebühr*

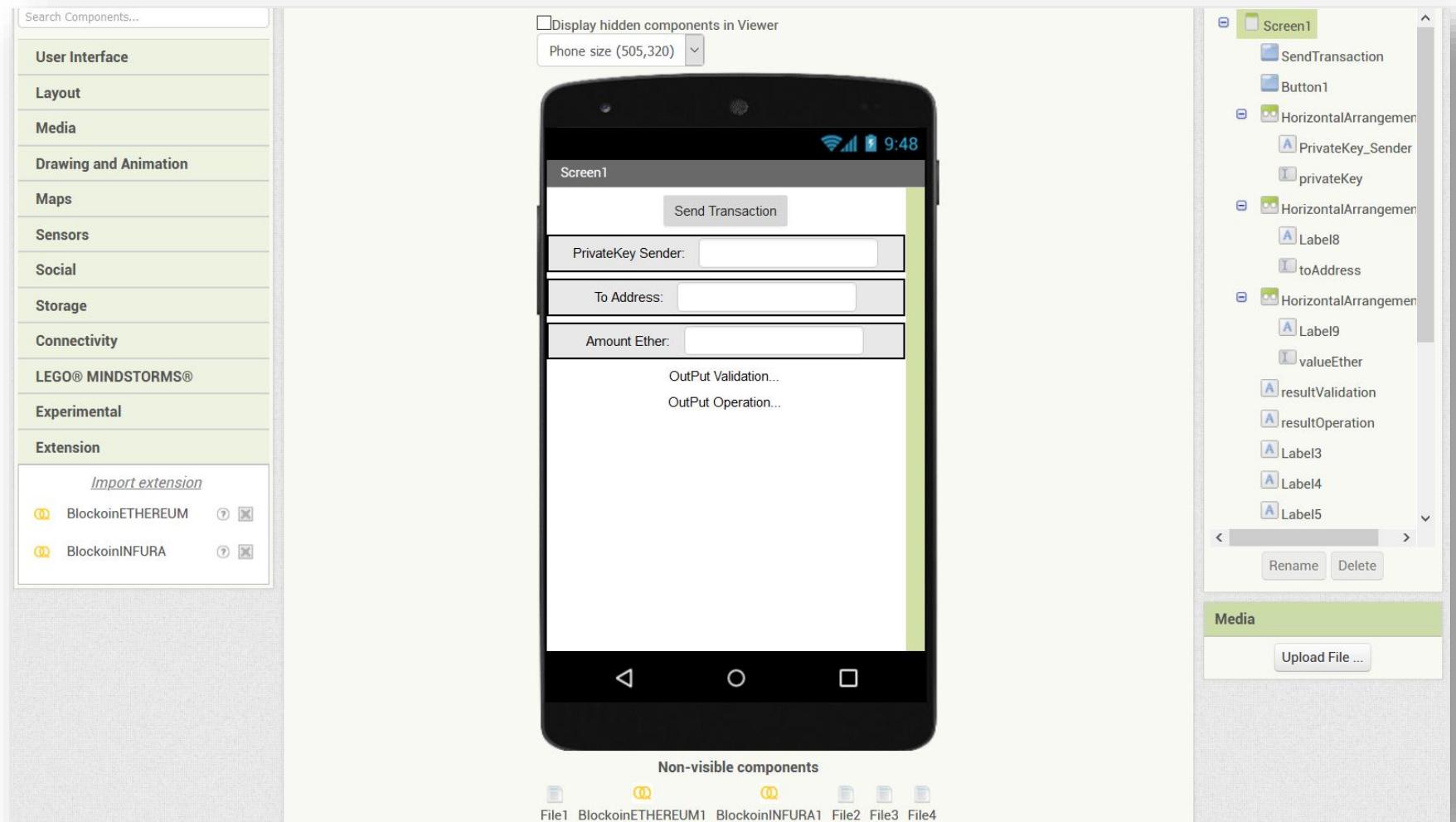
12. Gebühren bei Coinsolidation.org

Standardtransaktion: \$ 0,5 Cent USD + Kosten für das Ethereum-Netzwerk.

Transaktion veröffentlichen und/oder einen Smart-Vertrag ausführen: \$15 USD + Kosten des Ethereum-Netzwerks.

13. Erstellen Sie Ihre Android-App (Exchange) in 15 Minuten.

Entwurf im App Inventor (Bildschirm). - 5 Minuten.



Funktionsblöcke (`eth_SendTransactionEasy`) und Ereignis (`OutPutSendTransactionEasy`) - 5 Minuten

```

when [SendTransaction v].Click
do
  call BlockchainETHEREUM1 .eth_sendTransactionEasy
    hexPrivateKeySender [privateKey v].Text
    toAddress [toAddress v].Text
    valueEther [valueEther v].Text
end

when BlockchainETHEREUM1 .OutputSendTransactionEasy
do
  set resultValidation [Text v] to [get transactionValidationE v]
  call File1 .SaveFile
    text [get transactionValidationE v]
    fileName [/trasactionValidation.txt]
  set resultOperation [Text v] to [get transactionOperationE v]
  call File1 .SaveFile
    text [get transactionOperationE v]
    fileName [/trasactionOperation.txt]
end

```

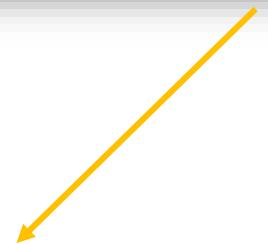
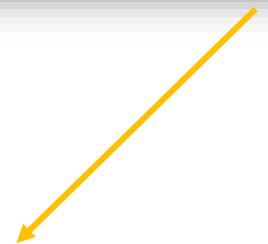
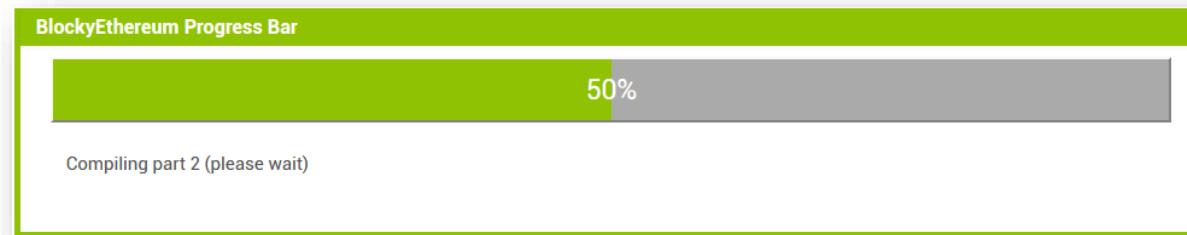
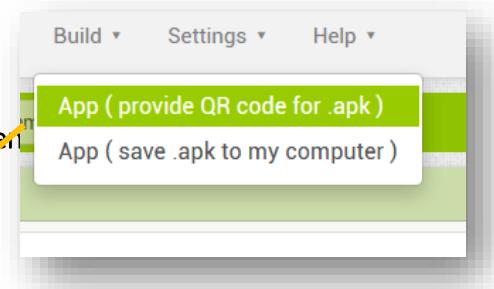
Daten eingeben:

- PrivateKey:** Primärschlüssel zur Adresse des Absenders.
- toAddress:** Hexadezimale Adresse des Empfängers.
- valueEther:** Geben Sie die Menge an Äther an, die versendet werden soll.

Speichern Sie die Ergebnisse in Textdateien:

- Funktion Datei1: Datei **trasactionValidation.txt**
- Funktion Datei2: Datei **trasactionOperation.txt**

Wir komplizieren, generieren die APK-Datei, um sie auf dem Android-Gerät zu installieren. - 5 Minuten



HINWEIS: Wenn die Transaktion ausgeführt wird, dauert es etwa 6 bis 8 Sekunden, bis die Schaltfläche "Transaktion senden" losgelassen wird. Aufgrund der Verbindungszeit mit dem Ethereum-Netzwerk.

14. Token COINsolidation.

Token Coinsolidation ist ein Projekt mit drei Hauptrichtlinien.

Stellen Sie sich vor, Sie haben Ihre eigene Token-Kryptowährung, um Ihr Geschäft auszubauen.

Die erste ist die Schaffung des ersten Netzes von Erweiterungen auf der Grundlage der visuellen Programmierung Methodik Blockly, in dem durch seine einfache und intuitive Nutzung kann von jedermann ohne Vorkenntnisse der Programmierung verwendet werden, die Erweiterungen, die in unserer Roadmap (White Paper) konsultiert werden können, sind auf zwei grundlegende Sektoren in der Weltwirtschaft, Sektor der Krypto-Währungen und / oder Tokens und dem Sektor der Währungen (Fiat) oder gemeinsame Nutzung weltweit wie der Dollar USD, Euro EU, Pfund oder jede andere Währung in Gebrauch gerichtet.

Die zweite Leitlinie im CoinSolidation-Projekt ist die Schaffung eines neuen Algorithmus zur Konsolidierung von Adressen aktueller und zukünftiger Kryptomontagen. Wir entwickeln einen neuen Modell-Algorithmus, um eine Adresse zu erstellen, die verschiedene Adressen aus verschiedenen Blockketten zu einer universellen Adresse konsolidiert, siehe unser (Weißbuch) unter www.Coinsolidation.org oder <https://github.com/coinsolidation/whitepaper>

Die dritte Leitlinie ist die Anwendung der Quantum Computing Technologie in der Sicherheit der CoinSolidation Umgebung, dies wird mit den bereits entwickelten Erweiterungen der QRNG (Quantum Random Number Generator) und PQC (Post-Quantum Computing) Sicherheitsalgorithmen angewendet. Diese finden Sie im offiziellen Extensions-Repository auf der Github-Website, <https://github.com/coinsolidation>

Allgemeine Merkmale der CryptoToken CoinSolidation

Name: COINsolidation

Symbol: CUAG

Typ: NFT

Startland: Estland

Offizielle Website: www.Coinsolidation.org

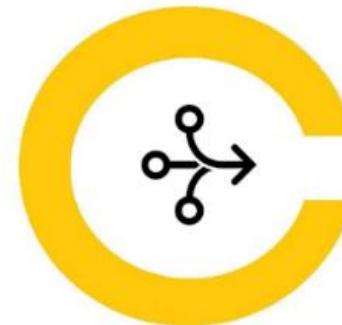
Unternehmen: Coinsolidation International.

Startdatum: 30 Dezember 2020

Erstellt von: Lugu Samaya.

Konsens-Algorithmus: Quantennachweis.

Adress-Algorithmus: Konsolidierte Universaladresse.



15. Lizenzierung und Nutzung von Software.

Lizenzierung, Nutzungsbedingungen und Konditionen siehe www.coinsolidation.org oder schreiben Sie an info@coinsolidation.org