



Configurazione e amministrazione.

Estensione dello scambio di etereum (EEE).

Guida per l'utente

versione 1.0.0.0 Beta

Novembre 2020.

Blockoin.org è un marchio registrato di Bankoin Inc, sotto licenza d'uso gratuito e commerciale. Termini e condizioni d'uso su: www.CoinSolidation.org

Contenuto

1.	Introduzione.....	3
2.	Cos'è la programmazione Blockly?.....	5
3.	Cos'è un'estensione?.....	5
4.	Che cos'è BlockoinEthereum e BlockoinINFURA?.....	5
5.	Concetti di base applicati nella piattaforma Ethereum.	6
6.	Installazione e configurazione dell'estensione e dell'ambiente di prova dell'Ethereum.....	13
7.	Definizione e utilizzo dei blocchi (funzione generica).....	20
8.	Scambio di caratteristiche ed eventi di Ethereum Extension (EEE).	21
9.	Passi per creare un gettone CryptoToken o Cryptomoney Token.	32
10.	Come mettere in vendita un nuovo bene o il vostro gettone della cripta (Token ERC20)...	33
11.	Calcolo del costo della transazione standard e della transazione del contratto Smart.....	56
12.	Quote di Coinsolidation.org	60
13.	Crea la tua App Android (Exchange) in 15 minuti.	61
14.	Token CoinSolidation.	64
15.	Licenze e utilizzo del software.	65

1. Introduzione.

Ethereum è una piattaforma open source, decentralizzata a differenza di altre catene a blocchi, Ethereum può fare molto di più. È programmabile, il che significa che gli sviluppatori possono utilizzarlo per creare nuovi tipi di applicazioni. La sua moneta digitale si chiama "Etere".

Queste applicazioni decentralizzate (o "dapps") ottengono i vantaggi della tecnologia di crittomontaggio e della catena a blocchi. Sono affidabili e prevedibili, il che significa che una volta "caricati" nell'Ethereum, funzioneranno sempre secondo i tempi previsti. Possono controllare gli asset digitali per creare nuovi tipi di applicazioni finanziarie. Possono essere decentralizzati, il che significa che nessuna entità o persona li controlla.

In questo momento, migliaia di sviluppatori in tutto il mondo stanno creando applicazioni su Ethereum e inventando nuovi tipi di applicazioni, molte delle quali possono essere utilizzate oggi:

- Portafogli di cripto-valuta che consentono di effettuare pagamenti istantanei a basso costo con ETH o altri asset
- Applicazioni finanziarie che consentono di prendere in prestito, prestare o investire i vostri beni digitali
- Mercati decentralizzati, che consentono lo scambio di beni digitali, o anche lo scambio di "previsioni" sugli eventi del mondo reale.
- Giochi in cui si ha un patrimonio nel gioco e si possono anche vincere soldi veri.

Come funziona l'etere?

L'etere, come altre valute criptate, utilizza un libro digitale condiviso in cui vengono registrate tutte le transazioni. È accessibile al pubblico, completamente trasparente e molto difficile da modificare in seguito.

Questa si chiama **catena di blocco**, e si costruisce attraverso il processo di **estrazione**.

I minatori hanno la responsabilità di verificare gruppi di transazioni etere per formare "blocchi" e di codificarli risolvendo algoritmi complessi. Questi algoritmi possono essere più o meno difficili per mantenere una certa coerenza nel tempo di elaborazione dei blocchi (circa uno ogni 14 secondi).

I nuovi blocchi vengono poi collegati alla precedente catena di blocchi e il minatore in questione riceve una **ricompensa, ovvero** un numero fisso di *gettoni di etere*. Normalmente si tratta di 5 unità di etere, anche se questo numero può essere visto come variabile se la valuta della cripta continua a crescere.

Come funziona l'Ethereum?

La *catena di blocco* dell'Ethereum è molto simile a quella del bitcoin, ma il suo linguaggio di programmazione permette agli sviluppatori di creare software attraverso il quale gestire le transazioni e automatizzare alcuni risultati. Questo software è noto come un **contratto intelligente**.

Se un contratto tradizionale descrive i termini di una relazione, un contratto intelligente si assicura che tali termini siano rispettati scrivendo in codice. Si tratta di programmi che eseguono automaticamente il contratto una volta che le condizioni predefinite sono soddisfatte, eliminando il ritardo e il costo che esiste quando si esegue un contratto manualmente.

Per fare un semplice esempio, un utente dell'Ethereum potrebbe creare un contratto intelligente per inviare una determinata quantità di etere ad un amico in una certa data. Scriveranno questo codice nella stringa di blocco e quando il contratto sarà completato (cioè al raggiungimento della data concordata) l'etero verrà inviato automaticamente.

Questa idea di base può essere applicata a configurazioni più complesse, e il suo potenziale è probabilmente illimitato, con progetti che hanno già fatto notevoli progressi in settori come quello assicurativo, immobiliare, dei servizi finanziari, dei servizi legali e della microfinanza.

I contratti intelligenti hanno anche diversi vantaggi aggiuntivi:

1. Eliminano la figura dell'intermediario, offrendo all'utente un controllo totale e minimizzando i costi aggiuntivi
2. Sono registrati, criptati e duplicati nella catena di blocco pubblico, dove tutti gli utenti possono vedere l'attività di mercato
3. Eliminare il tempo e lo sforzo richiesti nei processi manuali

Naturalmente, i contratti intelligenti sono ancora un sistema molto nuovo con molti dettagli da rifinire. Il codice è tradotto alla lettera, quindi eventuali errori durante la creazione del contratto potrebbero portare a risultati indesiderati che non possono essere modificati.

DApps vs. contratti intelligenti

I contratti intelligenti condividono le analogie con le DApp (applicazioni decentralizzate), ma le separano anche da alcune importanti differenze.

Come i contratti intelligenti, una DApp è un'interfaccia che collega un utente a un servizio di un provider attraverso una rete peer network decentralizzata. Ma, mentre i contratti intelligenti richiedono la creazione di un numero fisso di partecipanti, le DApp non hanno alcun limite al numero di utenti. Inoltre, non si limitano solo ad applicazioni finanziarie come i contratti intelligenti: una DApp può servire a qualsiasi scopo si possa pensare.

2. Cos'è la programmazione Blockly?

Blockly è una metodologia di **programmazione visiva** basata sul linguaggio di programmazione JavaScript, che consiste in un semplice insieme di comandi che possiamo combinare come se fossero i pezzi di un puzzle. È uno strumento molto utile per chi vuole **imparare a programmare** in modo intuitivo e semplice o per chi sa già programmare e vuole vedere le potenzialità di questo tipo di programmazione.

Blockly è una forma di programmazione in cui non è necessario alcun background in nessun tipo di linguaggio informatico, questo perché si tratta solo di unire blocchi grafici come se stessimo giocando a lego o a un puzzle, basta avere un po' di logica ed è tutto!

Chiunque può creare programmi per cellulari (smartphone) senza pasticciare con quei linguaggi di programmazione difficili da capire, basta mettere insieme i blocchi in modo grafico in modo semplice, facile e veloce da creare.

3. Cos'è un'estensione?

Un'estensione è un modulo realizzato in un linguaggio di programmazione Java dato in un file con estensione .AIX

Nel progetto Blockoin.org ci siamo basati sulla creazione di estensioni user-friendly per l'area finanziaria con cui si possono realizzare applicazioni mobili in pochi minuti senza avere una vasta conoscenza della programmazione.

4. Che cos'è BlockoinEthereum e BlockoinINFURA?

BlockoinEthereum e BlockoinINFURA sono software di libero utilizzo (estensioni) che includono le seguenti soluzioni tecnologiche (algoritmi) per poter creare di essere utilizzati nella rete dell'Ethereum:

- CREAZIONE DI NUOVI CONTI SULLA PIATTAFORMA ETHEREUM.
- CONSULTAZIONE DEL BILANCIO, TRASFERIMENTI DI ATTIVI (ETERE CRITTOMONICO E GETTONI).
- CREAZIONE DI NUOVI GETTONI ERC20 E CRYPTOMONEDA-TOKEN.
- COMPILAZIONE, CREAZIONE, CONSULTAZIONE E PUBBLICAZIONE DI CONTRATTI INTELLIGENTI
- CREAZIONE DI TRANSAZIONI OFFLINE E ONLINE.
- CARICAMENTO DELLE CHIAVI PRIMARIE E DELLE CHIAVI PUBBLICHE.
- TASSI DI CAMBIO E CONSULTAZIONE DELLE STAZIONI DI SERVIZIO.
- SVILUPPO, TEST ED ETEREO AMBIENTI DI RETE (RETI) DI BASE
- INCLUDE LE 39 CARATTERISTICHE DELL'INFURA.
-

5. Concetti di base applicati nella piattaforma Ethereum.

Cos'è una catena di blocco?

La blockchain è generalmente associata a Bitcoin e ad altre valute crittografiche, ma queste sono solo la punta dell'iceberg, poiché non viene utilizzata solo per la moneta digitale, ma può essere utilizzata per qualsiasi informazione che possa avere un valore per gli utenti e/o le aziende. Questa tecnologia, che ha le sue origini nel 1991, quando Stuart Haber e W. Scott Stornetta hanno descritto il primo lavoro su una catena di blocchi criptografati, è stata notata solo nel 2008, quando è diventata popolare con l'arrivo del bitcoin. Ma attualmente il suo utilizzo è richiesto in altre applicazioni commerciali e si prevede che crescerà nel medio futuro in diversi mercati, come le istituzioni finanziarie o l'Internet degli oggetti tra gli altri settori.

La blockchain, meglio conosciuta con il termine blockchain, è un singolo record concordato distribuito su più nodi (dispositivi elettronici come PC, smartphones, tablet, ecc.) in una rete. Nel caso delle valute criptate, possiamo pensarlo come il libro contabile in cui viene registrata ciascuna delle operazioni.

Il suo funzionamento può essere complesso da capire se entriamo nei dettagli interni della sua realizzazione, ma l'idea di base è semplice da seguire.

Viene memorizzato in ogni blocco:

- 1.- una serie di registrazioni o transazioni valide,
- 2.- informazioni relative a quel blocco,
- 3.- il suo collegamento con il blocco precedente e il blocco successivo attraverso l'hash di ogni blocco –un codice unico che sarebbe come l'impronta digitale del blocco.

Pertanto, **ogni blocco** ha un **posto specifico e non mobile all'interno della catena**, in quanto ogni blocco contiene informazioni provenienti dall'hash del blocco precedente. L'intera catena è memorizzata su ogni nodo di rete che compone la blockchain, quindi **una copia esatta della catena è memorizzata su tutti i partecipanti alla rete**.

Che cos'è un indirizzo o un account all'interno della piattaforma Blockchain Ethereum?

Si tratta di una stringa di 42 caratteri nella piattaforma dell'Ethereum che rappresenta un numero in base esadecimale, dove i beni definiti nell'Ethereum saranno depositati o inviati. In altre piattaforme a catena di blocco, ad esempio, il numero di caratteri del conto o dell'indirizzo può essere diverso:

0x5d2Acdb34c279Aa6d1e94a77F7b18aB938BFb2bB

Cos'è un kryptomoney?

Si tratta di una moneta digitale o virtuale progettata per funzionare come mezzo di scambio. Utilizza la crittografia (sicurezza digitale) per mettere in sicurezza e verificare le transazioni, oltre che per controllare la creazione di nuove unità di una particolare crittografia.

Che cos'è l'Etere?

L'etere è la valuta digitale nativa della piattaforma a catena di blocchi Ethereum, una piattaforma decentralizzata sviluppata nel 2013. Un Etere è un'unità che può essere suddivisa in unità più piccole chiamate WEI ed ha la seguente equivalenza.

1 Etere = 1.000.000.000.000.000.000 Wei. (Nell'espressione matematica è 10^{18}).

Quali unità vengono gestite quando si utilizza un etere?

1	etere
10^3	finney
10^6	szabo
10^9	Shannon o GWEI questo viene utilizzato per il GasPrice.
10^{12}	babbage
10^{15}	lacci d'amore
10^{18}	wei

Cos'è un gettone?

I gettoni sono risorse digitali che possono essere utilizzate all'interno di un determinato ecosistema di progetto.

La principale distinzione tra gettoni e monete criptate è che le prime richiedono un'altra piattaforma a catena di blocco (non la propria) per funzionare. L'Ethereum è la piattaforma più comune per la creazione di gettoni, soprattutto grazie alla sua funzione di contratto intelligente. I gettoni creati sulla catena di blocco dell'Ethereum sono generalmente noti come gettoni ERC-20, anche se esistono altri tipi di gettoni più specializzati, come il gettone

ERC-721 utilizzato principalmente per beni da collezione (carte, uso nei videogiochi, opere d'arte, ecc.).

Cos'è il gas?

Il gas è il costo dell'esecuzione di un'operazione o di un insieme di operazioni sulla rete dell'Ethereum. Queste operazioni possono essere diverse: dalla realizzazione di una transazione all'esecuzione di un contratto intelligente o alla creazione di un'applicazione decentralizzata.

In altre parole, più semplicemente, il Gas è l'unità di misura del lavoro svolto nell'Ethereum.

Come nel mondo fisico, anche nell'Ethereum ci sono lavori che costano più di altri: se l'operazione che vogliamo eseguire richiede un maggiore utilizzo di risorse da parte dei nodi che formano la piattaforma, questo farà aumentare anche il Gas e viceversa.

Il Gas serve principalmente a svolgere tre funzioni sulla piattaforma Ethereum:

1.- Assegna un costo all'esecuzione dei compiti; nell'Ethereum, anche l'esecuzione dei compiti ha un costo. **A seconda della difficoltà del compito o della velocità con cui vogliamo che il compito venga elaborato, il costo computazionale di tale operazione sarà maggiore o minore di conseguenza e il numero di gas aumenterà o diminuirà in proporzione.**

2.- Mette in sicurezza il sistema; Il sistema Ethereum è un sistema sicuro e questo è in gran parte possibile grazie al Gas.

Richiedendo il pagamento di una commissione per ogni transazione effettuata, la piattaforma garantisce che non vengano elaborate in rete transazioni inutilizzabili. Questo aiuta a rendere la blockchain più leggera, in quanto non aggiungerà molti Megabyte inutili di informazioni alla blockchain.

Inoltre, con il Gas, il sistema è anche protetto contro lo "spam" e l'uso infinito dei loop: istruzioni per eseguire compiti ripetitivi in codice.

Ad esempio, se il gas non esistesse, nulla impedirebbe di ripetere all'infinito un'operazione, facendo crollare il sistema e rendendolo inutilizzabile.

3.- Premiare i minatori; I minatori sono sistemi esterni indipendenti distribuiti in tutto il mondo che si occupano di eseguire le transazioni all'interno della piattaforma Ethereum. Quando effettuiamo una transazione o eseguiamo un contratto intelligente, "paghiamo" una certa quantità di Gas.

Questo gas serve a "pagare" i minatori per le risorse che hanno utilizzato (hardware, elettricità e tempo) e **aggiunge anche una ricompensa** per il loro lavoro. Pertanto, potremmo dire che il Gas aiuta anche a mantenere l'equilibrio della piattaforma.

Si parla di "pagare" il gas - tra virgolette - perché questo è il costo delle operazioni. In altre parole, si "paga" la spesa che costa l'Ethereum per l'elaborazione delle transazioni.

Cos'è il prezzo del gas?

Un'unità Gas corrisponde all'esecuzione di un'istruzione, come ad esempio un passo del computer.

Come fanno i minatori a "incassare" il gas che acquistano come ricompensa?

Il gas in sé non vale nulla e quindi non può essere addebitato. Per "caricare" questo gas usato, è necessario dare valore a queste risorse consumate, cioè dare un valore monetario al lavoro svolto dai minatori. La quantità di Gas utilizzata in una transazione o in un contratto intelligente ha un prezzo equivalente in Etere. Questo prezzo è chiamato "prezzo del gas", viene normalmente assegnato dai minatori ed è variabile a seconda di quanto è occupata la catena di blocco dell'Ethereum. Normalmente viene gestito in unità GWEI.

Cos'è il Limite di Gas?

Questi dati indicano il valore massimo di Gas che una transazione può consumare per essere valida.

Normalmente, il software utilizzato per effettuare le transazioni sulla rete dell'Ethereum calcola automaticamente una stima della quantità di Gas necessaria per effettuare la transazione e ce la mostra immediatamente.

Cos'è la stazione di servizio?

È il luogo in cui si fa riferimento ai valori trattati dai minatori per decidere all'unanimità quale sia il GasPrice necessario per effettuare i diversi tipi di transazioni nella catena di blocco dell'Ethereum.

A seconda di quanto GasPrice è selezionato, il tempo di priorità dato all'esecuzione della transazione sarà ponderato, normalmente vengono gestiti tre tipi di GasPrice: TRADER: ASAP, FAST < 2 minuti, STANDARD < 5 minuti. Si tratta di valori medi e possono variare nel tempo a seconda del carico di lavoro (trazioni) della rete principale dell'Ethereum.

<https://ethgasstation.info/>

Cos'è uno scambio?

Uno scambio di kryptomaniaci è il punto d'incontro dove si svolgono scambi di kryptomaniaci in cambio di denaro fiat o di altri kryptomaniaci. In queste borse online si genera il prezzo di mercato che segna il valore delle crittomie in base alla domanda e all'offerta.

Che cosa sono i tassi di cambio?

Questi sono i tassi per il valore di un Etere nelle valute di ogni paese. Ad esempio, al momento della creazione di questo manuale, un Etere ha un valore in dollari USA di 430,94 dollari.

Cos'è un contratto intelligente?

In ethereum un contratto Smart è un programma in linguaggio di programmazione chiamato Solidity che si trova all'interno della blockchain Ethereum, che ha istruzioni da eseguire automaticamente sulla base di regole prestabilite, questa proprietà fa un'evoluzione in tutte le blockchain esistenti in quanto è possibile creare molti contratti Smart adattati a diversi settori privati e pubblici rendendo più efficienti le operazioni delle aziende o se del caso adattarsi a sistemi o programmi di ogni tipo.

Che cos'è il "nonce"?

Si tratta di un contatore incrementale "numero esadecimale" che tiene traccia delle transazioni in ogni direzione o conto creato nell'Ethereum.

Cos'è una transazione?

Si tratta dell'esecuzione o del trasferimento di un qualche tipo di bene non tangibile che può essere dato un valore prestabilito all'interno del sistema Ethereum e che può essere successivamente modificato in un valore tangibile per una società o una persona.

Che cos'è txHash?

Si tratta di un numero esadecimale che aiuta a tracciare il risultato nel dettaglio di ogni transazione.

Quali tipi di transazioni esistono?

Si hanno due tipi, uno è la transazione "offline" che si crea senza la necessità di avere la connessione alla rete principale di Ethereum può essere memorizzata fino a quando non si sceglie di connettersi alla rete di Ethereum e rilasciare la transazione, hanno il vantaggio della sicurezza perché l'intera transazione viene elaborata offline che evita qualsiasi anomalia che potrebbe essere nella connessione di rete. L'altra transazione è quella "online", che deve sempre essere collegata a Internet con i vantaggi e gli svantaggi in termini di sicurezza che la connessione comporta.

Che cos'è INFURA.io?

Infura.io è una piattaforma che fornisce un insieme di strumenti e infrastrutture che permettono agli sviluppatori di portare facilmente la propria blockchain applicativa dal testing, al ridimensionamento, con un accesso semplice e affidabile all'Ethereum e all'IPFS.

Cos'è un indirizzo sulla rete dell'Ethereum?

Un indirizzo o un conto è composto da tre parti, l'indirizzo, la chiave pubblica e la chiave privata, queste due chiavi sono una stringa di numeri e caratteri in formato esadecimale che vengono utilizzati per inviare e ricevere (attivo) o etere (valuta digitale).

La chiave primaria non deve mai essere condivisa con nessuno in quanto è quella che autorizza il rilascio del saldo (firma le transazioni) detenuto sul conto.

La chiave pubblica è nota a tutto il pubblico ed è condivisa con chiunque in quanto è il riferimento per confermare che la transazione è corretta sia in termini di valore che a chi la invia.

Esempi:

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

Che cos'è Coinsolidation.org?

Si tratta di una piattaforma di consolidamento delle valute crittografiche, abbiamo creato i primi indirizzi ibridi nel mondo finanziario centralizzato e decentralizzato, utilizzando la tecnologia Blockly che è una forma di programmazione visiva senza la necessità di una conoscenza avanzata della programmazione basata su estensioni funzionali. Vedi il nostro WhitePaper su www.coinsolidation.org

Quali tipi di reti per i test e la rete principale nella catena di blocco Ethereum?

<https://mainnet.infura.io>

Rete principale, rete di produzione dove tutte le transazioni sono effettuate in tempo reale.

<https://ropsten.infura.io>

La rete di test Ropsten permette agli sviluppatori di blockchain di testare il loro lavoro in un ambiente live, ma senza la necessità di veri e propri token ETH, con un algoritmo chiamato (*Proof-of-Work*).

<https://kovan.infura.io>

Kovan test network, che a differenza del suo predecessore ropsten utilizza un algoritmo chiamato Proof of Authority.

<https://rinkeby.infura.io>

Test Network, utilizza un algoritmo chiamato Proof of Authority. Uno dei più comunemente usati per i test.

<https://goerli.infura.io>

Goerli è una rete pubblica di test per l'Ethereum che il motore di consenso POA (Proof of Authority) supporta con vari clienti. A prova di spam.

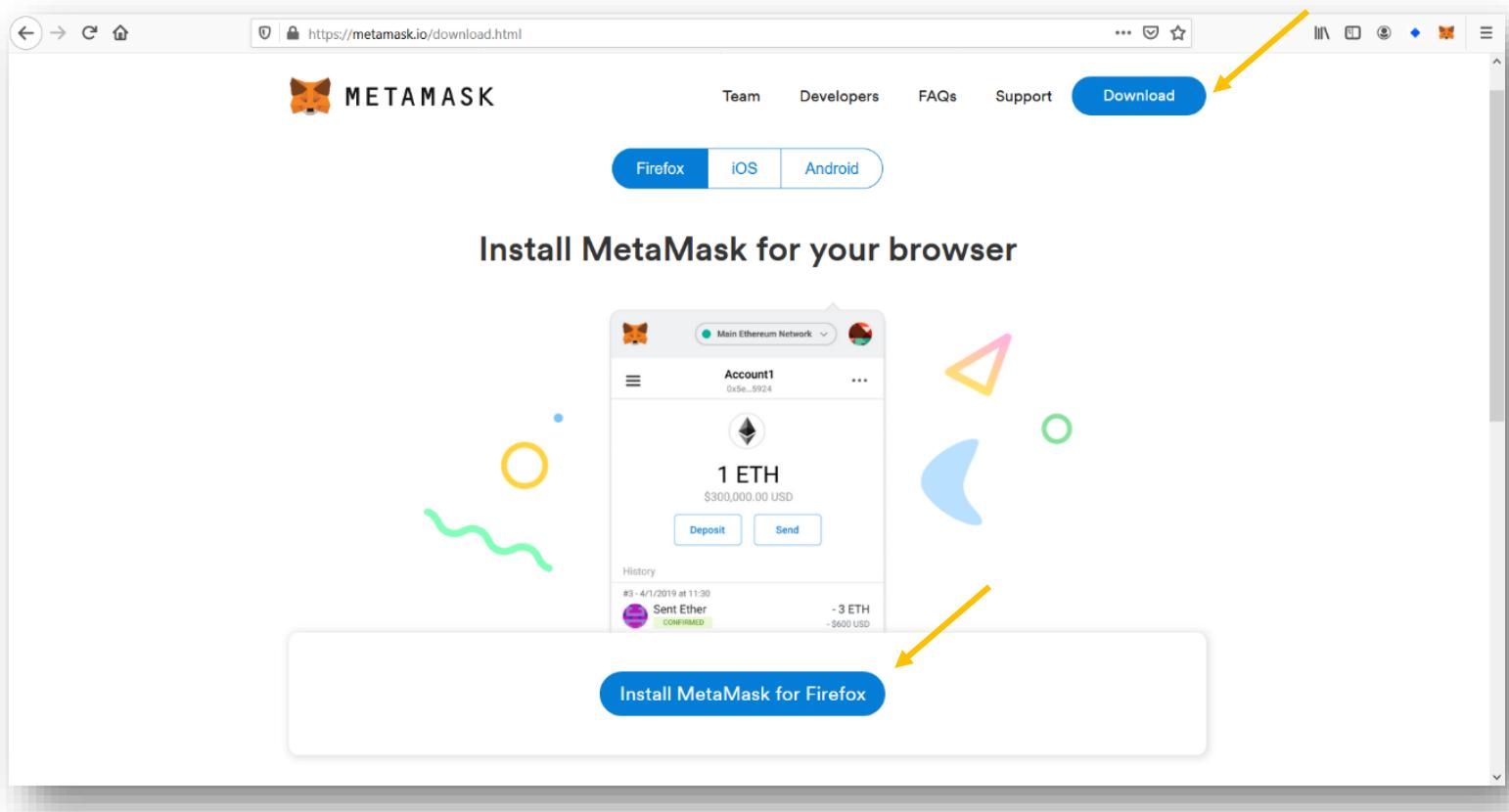
In totale ci sono 5 reti basate sulla catena a blocchi Ethereum, una per la produzione o principale e quattro per i test, poi useremo la rete di Rinkeby per installare il nostro ambiente di test.

6. Installazione e configurazione dell'estensione e dell'ambiente di prova dell'Ethereum.

Abbiamo bisogno prima di un ambiente di prova. Per imparare ad utilizzare le diverse funzionalità delle due estensioni che verranno utilizzate per creare, inviare, pubblicare, rivedere e ottenere dati da tutte le transazioni che effettuiamo sulla piattaforma Ethereum.

La prima cosa che dobbiamo fare è allestire il nostro ambiente di prova. Dovremo installare l'applicazione **METAMASK**, un portafoglio per creare, importare conti Ethereum e gestire le transazioni dal browser del nostro browser internet che utilizzeremo è Mozilla e può essere utilizzato anche in Chrome.

Andate sul sito ufficiale [www.metamask.io](https://metamask.io) e cliccate sul pulsante "Down load".

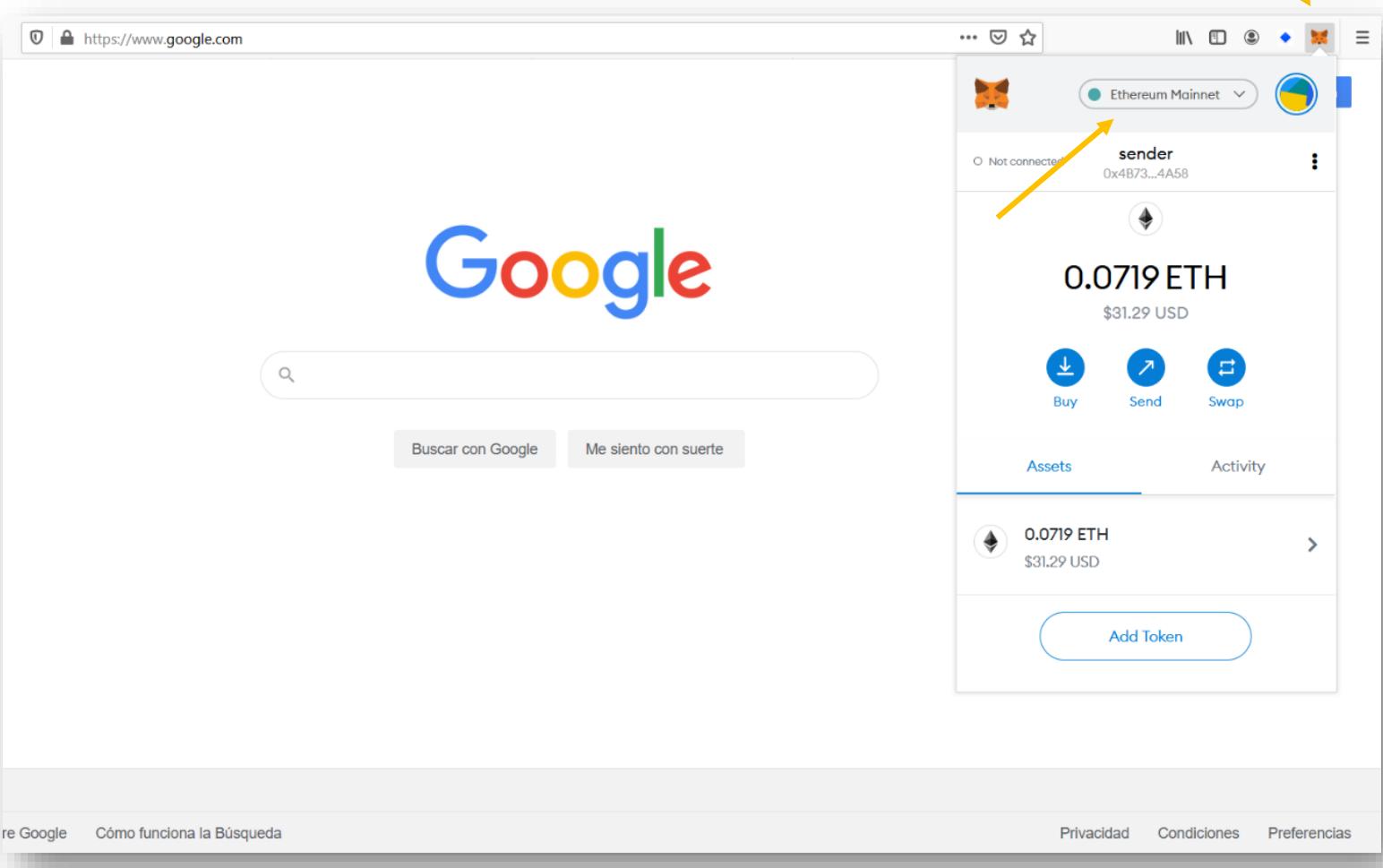


Quindi cliccare sul pulsante "Installare MetaMask per Firefox" e accettare le opzioni predefinite. Dopo l'installazione vedremo un'icona in alto a destra dove vedremo già installato il software Metamask.

Cliccare sull'icona Metamask e apparirà l'opzione per importare un account esistente o crearne uno nuovo. Nel nostro caso importeremo un account che abbiamo già in esecuzione con il metodo del "restore through seed". Se non ne avete uno, create semplicemente un

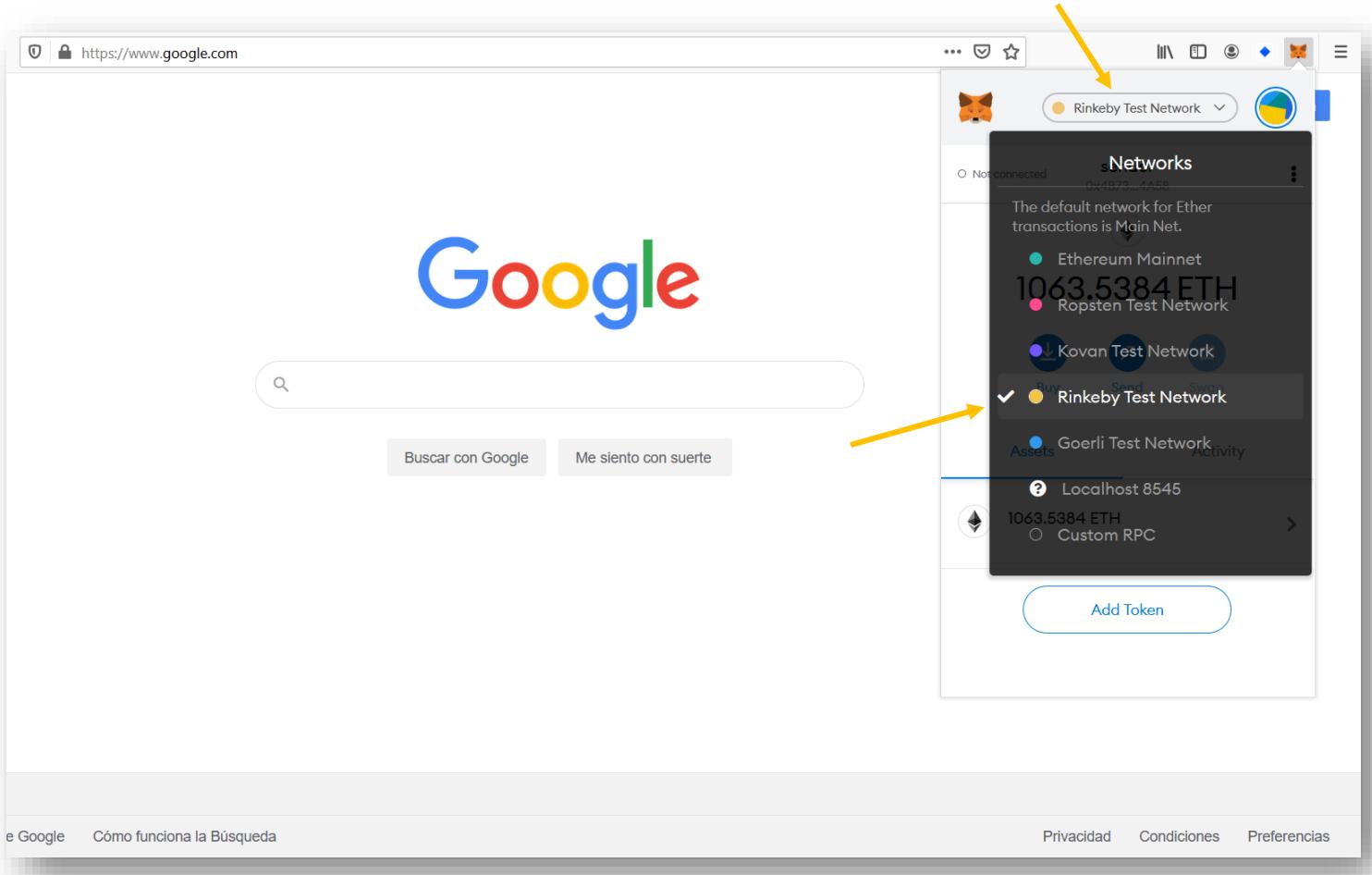
nuovo account. In questo caso ci verrà richiesta una password da assegnare in entrambi i casi.

Più tardi entriamo con la "password" e possiamo controllare quale saldo abbiamo, logicamente se è nuovo il saldo sarà nullo.



Ora procediamo a rivedere come depoſiteremo alcuni eteri (monete digitali) sul nostro conto di prova Rinkeby.

In alto abbiamo l'opzione di scegliere il tipo di rete che useremo, per default quando la inserisci ti colloca nella "Ethereum Mainnet", tuttavia, quando clicchi possiamo rivedere tutte le reti opzionali che possiamo scegliere, nel nostro caso abbiamo selezionato la rete Rinkeby.



Il passo successivo è quello di depositare gli eteri sul nostro conto di prova di riferimento della rete Rinkery.

NOTA IMPORTANTE: Gli eteri (valuta digitale) che depositiamo sul nostro conto riferiti alla rete Rinkery test non hanno alcun valore nel mercato dei criptati, saranno utilizzati da noi solo per il test del software. Controllate sempre su quale rete stiamo lavorando per evitare errori nelle transazioni.

Per ottenere l'etere per i test dobbiamo eseguire la seguente procedura.

In qualsiasi account twitter che hai dovremo inserire twitter e creare un commento che includa solo l'account e/o l'indirizzo che abbiamo in Metamask poi dovremo copiare il link del commento dato che abbiamo questo andremo al link successivo per ancorare il nostro account.

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

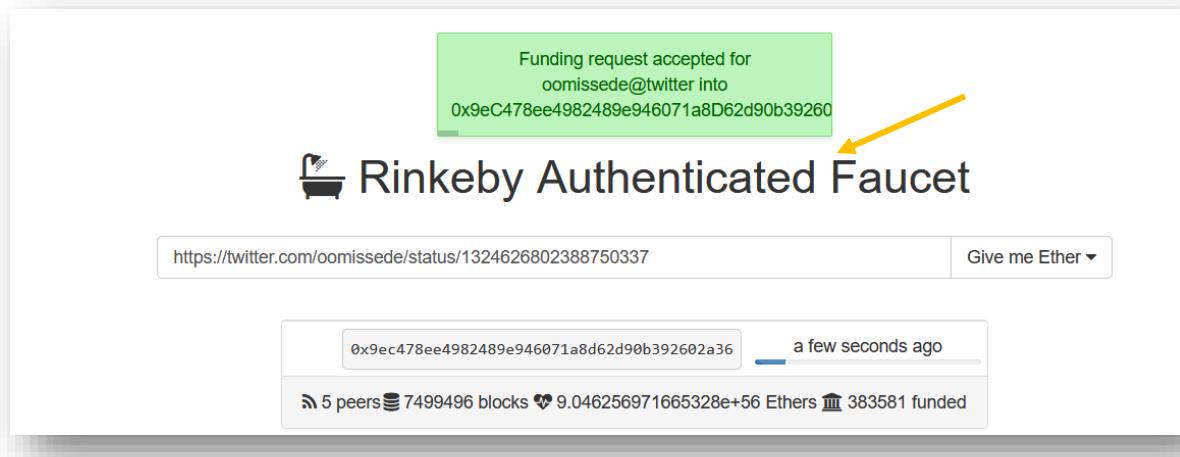
Abbiamo creato un commento su Twitter e abbiamo copiato il link dal commento.

The screenshot shows a Twitter post from the user AGAVE (@oomissede). The tweet contains a single line of text: "0x88ac6dcecc3603c7042f4334fc06db8e8d7062d5". Below the tweet, there is a "Traducir Tweet" button and the timestamp "2:22 a. m. · 6 nov. 2020 · Twitter Web App". At the bottom of the tweet card, there are four interaction icons: a speech bubble, a retweet icon, a heart, and a share icon. A yellow arrow points from the text above to the URL in the browser's address bar, which reads "https://twitter.com/oomissede/status/1324628185468854272".

Sul sito <https://faucet.rinkeby.io/> copiamo il link di twitter e sul pulsante "Give me Ether" scegliamo la quantità desiderata.

The screenshot shows the Rinkeby Authenticated Faucet website. At the top, it displays the URL "https://faucet.rinkeby.io". Below the URL, there is a search bar containing the Twitter URL "https://twitter.com/oomissede/status/132462802388750337". To the right of the search bar is a button labeled "Give me Ether ▾". A yellow arrow points to the URL in the browser's address bar. Another yellow arrow points to the dropdown menu next to the "Give me Ether" button, which lists three options: "3 Ethers / 8 hours", "7.5 Ethers / 1 day", and "18.75 Ethers / 3 days". The main content area of the page includes instructions on how to request funds via Twitter or Facebook, information about reCaptcha protection, and a pending requests tracking section. At the bottom right, there is a small "Powered by CoinGecko" logo.

Infine, se tutto è corretto, daremo un annuncio che il deposito è stato accettato e, a seconda del carico di lavoro del sistema di rete Rinkeby, il deposito avverrà in pochi minuti o potrebbe richiedere più tempo.



Ora che abbiamo Ethers sul nostro conto possiamo iniziare i test sulla piattaforma Ethereum.

Abbiamo due estensioni per interagire con la catena di blocco dell'Ethereum.

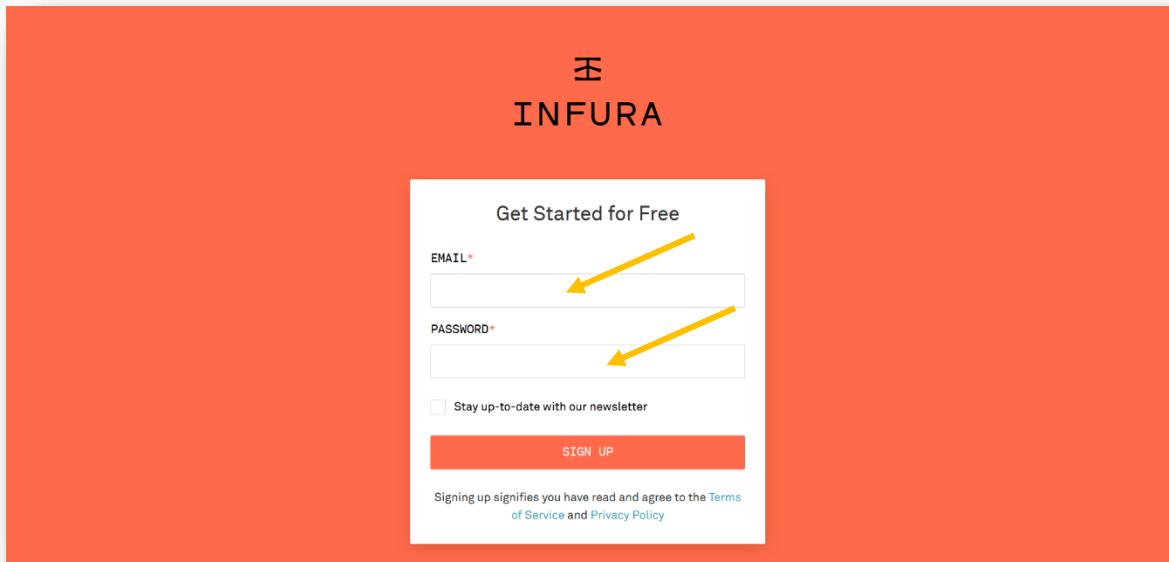
L'estensione di **co-solidamento**. **BlockoinETHEREUM.aix** contiene le funzioni per eseguire le seguenti funzioni:

- Creazione di nuovi conti (indirizzi) in catena di blocco Ethereum (privateKey, publicKey, Address)
- Memorizzazione dei dati del conto (indirizzi) in file binari.
- Importazione di conti di file binari (indirizzi)
- Ottenere un conto (indirizzo) tramite privateKey.
- Creazione e invio di transazioni tra conti (indirizzi) "online".
- Creazione, firma e invio di transazioni PushRaw offline.
- Consultazione dei dettagli della transazione Tx.
- Controllo del saldo degli indirizzi e dei contratti intelligenti.
- Compilatore per contratti Smart.
- Creazione, pubblicazione ed esecuzione di contratti intelligenti.
- Creazione, pubblicazione ed esecuzione del Token ERC20 (gettone di criptomonia).
- Ottenere il codice ABI da contratto intelligente.
- Verifica della connessione di rete.
- Interrogazione del valore dell'etere nel criptomercato di qualsiasi paese del mondo (valuta del paese) - Tassi di cambio.
- Consultazione GasPrice.
- Consultazione di "nonce" di un conto specifico.

L'estensione della coinsoidazione. BlockoinFURA.aix ci fornisce le 40 funzionalità della piattaforma Infura.io. Per i dettagli, consultare la documentazione json-rpc direttamente all'indirizzo <https://infura.io/docs>.

Per poter utilizzare l'estensione INFURA è necessario creare un account sul sito infura.io in quanto abbiamo bisogno di una KEY API per poter inviare le query alla rete Ethereum, così come per poter utilizzare le reti di test Ethereum.

Il modo in cui apriamo un conto è semplice, come mostrato di seguito.



Una volta creato l'account, entriamo e possiamo avere le KEY API delle diverse reti. Andiamo in alto a sinistra e clicchiamo su Ethereum, poi vediamo i progetti, **creiamo** un nuovo progetto e ci presentiamo a quel progetto.

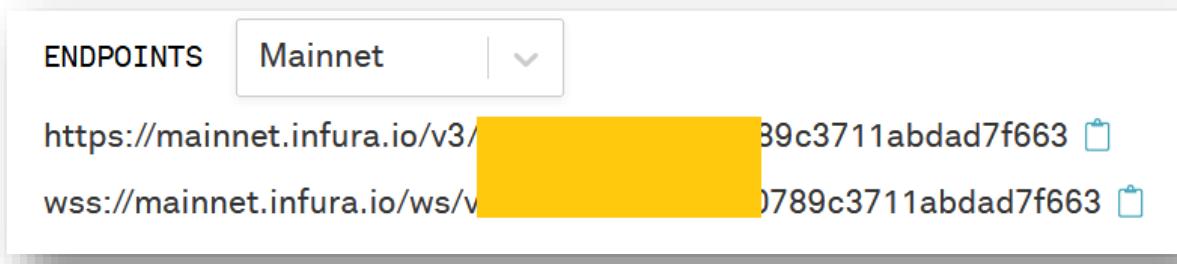
A screenshot of the Infura dashboard for the Ethereum project. The dashboard has a dark header with the Infura logo and a navigation bar on the left. The main area shows a project named "COINSOLIDATION" created on September 12, 2020. It includes sections for "REQUESTS PREV 7 DAYS" (with a line graph showing activity), "PLAN" (Core plan at 100,000 Requests/Day), and a "See how other customers are using our Ethereum API" section. Arrows point from the text in the first two paragraphs to various elements on the dashboard: one arrow points to the "CREATE NEW PROJECT" button, another to the "COINSOLIDATION" project card, and a third to the "REQUESTS PREV 7 DAYS" chart.

All'interno del progetto avremo i dati delle KEY API e delle diverse reti che possiamo utilizzare.

All'interno del progetto possiamo rivedere l'API KEY (PROJECT ID) e selezionare su quale rete lavoreremo o i link completi degli ENDPOINTS da scegliere.

The screenshot shows the Infura.io dashboard for Ethereum. The left sidebar has tabs for COINSOLIDATION, ETHEREUM (selected), FILECOIN, DOCS, COMMUNITY, and SUPPORT. The main area is titled 'COINSOLIDATION' and 'SETTINGS'. It shows the 'KEYS' section with 'PROJECT ID' (39c3711abdad7f663) and 'PROJECT SECRET' (1ac27e75434133134). The 'ENDPOINTS' dropdown is set to 'Mainnet', with other options like Ropsten, Kovan, Rinkeby, and Görli listed below. There are checkboxes for 'PROJECT SEC' and security requirements.

Per esempio, per utilizzare la rete principale dell'Ethereum prendiamo il seguente campionato:

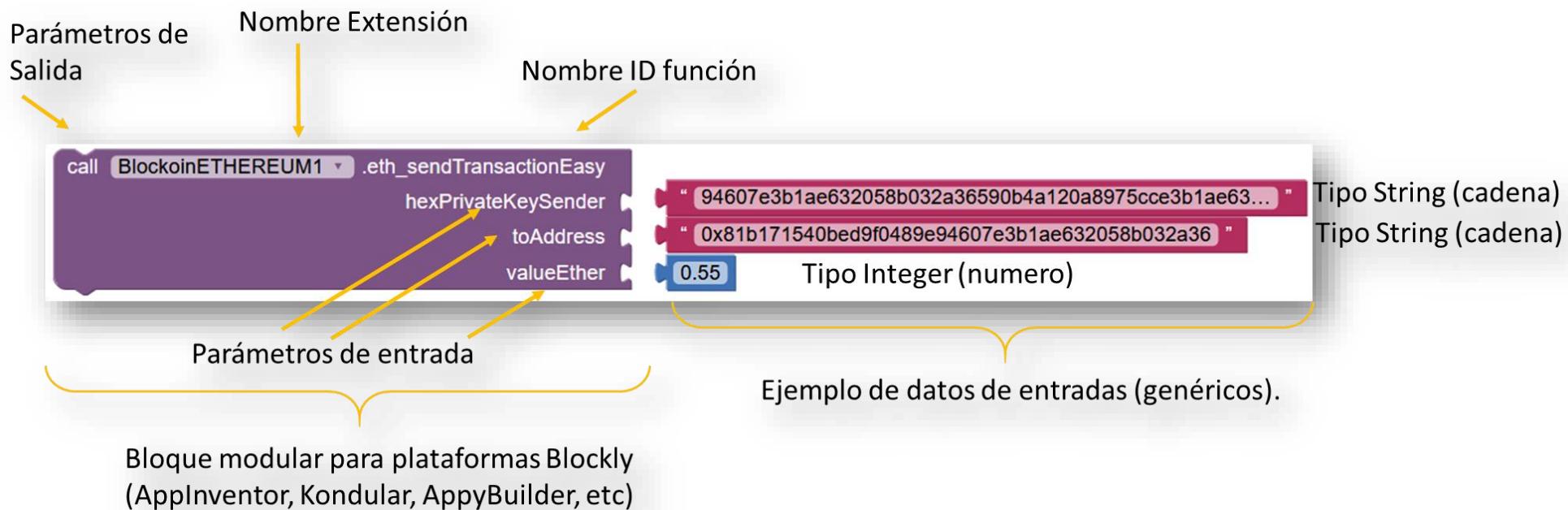


NOTA: Le estensioni sono state testate sui sistemi AppInventor, Kondular, Thunkable e AppyBuilder.

7. Definizione e utilizzo dei blocchi (funzione generica)

Inizieremo spiegando la distribuzione dei dati che tutti i blocchi avranno, la loro sintassi d'uso e la loro configurazione.

Nell'esempio seguente possiamo vedere un blocco modulare e i suoi parametri di ingresso e di uscita, così come i tipi di dati di ingresso, questi dati possono essere di tipo Stringa (stringa di caratteri) o Integer (intero o decimale). Mostriamo come viene utilizzato e lo configuriamo per il suo corretto funzionamento.



Ogni blocco di modulo avrà la sua descrizione e sarà nominato nel caso in cui abbia qualche dipendenza obbligatoria o opzionale di altri blocchi usati come parametri di input, il processo di integrazione sarà annunciato.

8. Scambio di caratteristiche ed eventi di Ethereum Extension (EEE).

***Prova di rete che useremo **Rinkeby**, come urlNetwork, quando si desidera effettuare transazioni reali è sufficiente cambiare la rete urlNetwork per **mainnet**.

Blocco per controllare la connessione a Internet - (**CheckInternetConnection**).

```
call BlockoinETHEREUM1 .CheckInternetConnection
```

Parametri di ingresso: Non applicabile.

Parametri di uscita: Restituisce "True" se si ha il collegamento o restituisce "False" se non c'è collegamento.

Descrizione: Bloccare per controllare la connessione a Internet e inviare dati (transazioni).

Bloccare per generare un **nuovo** indirizzo "Offline" - (**GenerateNewAddressEthereum**)

```
call BlockoinETHEREUM1 .GenerateNewAddressEthereum  
phraseHex "exchange ethereum extension for systems blockly "
```

Parametri di ingresso: **phrasaHex <String>**.

Parametri di uscita: Evento (**OutputGenerateNewAddressEthereum**)

Uscite: **PrivateKey<Stringa>**, **PublicKey<Stringa>** , **indirizzoEtehreum<Stringa>**.

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

Descrizione: creare un nuovo indirizzo etereo (conto) sulla base di una frase o di una sequenza di numeri. Il nuovo indirizzo può essere creato senza connessione di rete o internet - "Offline".

Blocca per generare un nuovo indirizzo "Online" - (**GeneraNuovoIndirizzoEthereum**)

```
call BlockoinETHEREUM1 .GenerateNewAddressEthereumAPI
```

Parametri di ingresso: Non applicabile.

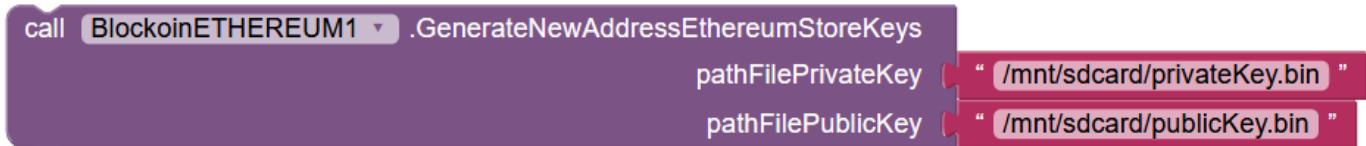
Parametri di uscita: ritorno in formato dati JSON; privateKey, publicKey, indirizzo.

Esempio di uscita:

```
{
  "private": "9ab4b2fba728a55643414f26adc04eea080740860fbe39b13fe4acb43dbb9f83",
  "public": "0421d559aa98d6fc77688ae9f19b3dc502c05f0b7f70bbe924cb712d6243a489695b1c1ee03993b3b6d97277
  97e780677a5469800b4d98374bdb910ed99fa2b5c8",
  "address": "92a2f157d5aec3fa79f92995fea148616d82c5ef"
}
```

Descrizione: Creare un nuovo indirizzo di ethereum (conto). E' necessario avere l'accesso o la connessione a internet in quanto la generazione avviene attraverso il servizio REST API - "Online".

Bloccare per generare un nuovo indirizzo "Offline" e salvare le chiavi pubbliche e private in file binari - ([GenerateNewAddressEthereumStoreKeys](#))



Parametri di ingresso: `pathFilePrivateKey<String>` , `pathFilePublicKey<String>`.

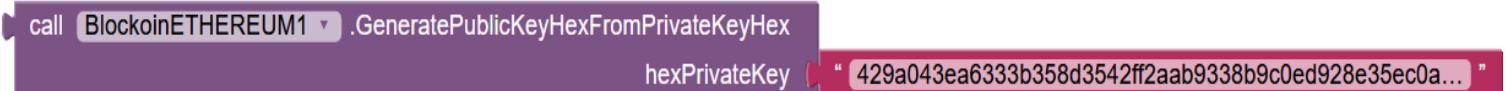
Parametri di uscita: Evento ([OutputGenerateNewAddressEthereumStoreKeys](#))

Uscite: `addressEthereum<String>` , `privateKeyECC<String>` , `publicKeyECC<String>` , `privateKeyHex<String>` , `publicKeyHex<String>`.



Descrizione: crea un nuovo indirizzo etero casuale (conto) e memorizza le chiavi pubbliche e private in file binari che verranno utilizzati per l'importazione e l'esportazione dei dati del conto. Il nuovo indirizzo può essere creato senza connessione di rete o internet - "Offline".

Blocco per generare la chiave pubblica - (**GeneratePublicKeyHexFromPrivateKeyHex**).



Parametri di ingresso: **hexPrivateKey <String>**.

Parametri di uscita: Evento (**OutputGeneratePublicKeyHexFromPrivateKeyHex**)

Uscite: **indirizzo<Stringa>** , **publicKeyHex<Stringa>**.



Descrizione: Crea la chiave pubblica in base all'inserimento di una chiave privata.

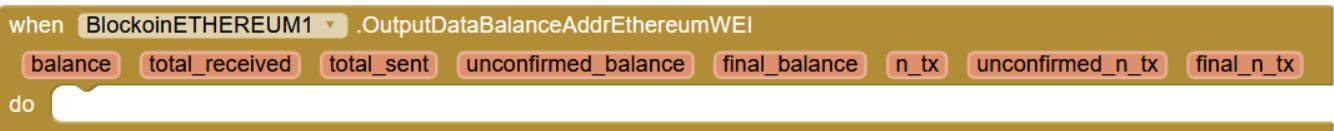
Bloccare per **ottenere il saldo** di un indirizzo - (**GetBalanceAddrEthereum**).



Parametri di ingresso: **hexPrivateKey <String>**.

Parametri di uscita: Evento (**OutputGeneratePublicKeyHexFromPrivateKeyHex**)

Uscite: **saldo<Stringa>** , **totale_ricevuto<Stringa>** , **totale_invito<Stringa>** , **saldo_non_confermato <Stringa>** , **saldo_finale<Stringa>** , **n_tx<Stringa>** , **non confermato_n_tx<Stringa>** , **final_n_tx<Stringa>** .



Descrizione: visualizza i dati di bilancio e i dati dettagliati del conto (indirizzo).

Bloccare per verificare se il vostro cellulare ha l'interfaccia di rete attivata - (GetDataNetworkConnection).

```
call BlockoinETHEREUM1 .GetDataNetworkConnection
```

Parametri di ingresso: Non applicabile.

Parametri di uscita: Evento (OutputGeneratePublicKeyHexFromPrivateKeyHex)

Uscite: nome di interfaccia<Stringa>, è collegato<Stringa>.

```
when BlockoinETHEREUM1 .OutputGetDataNetworkConnection
  interfacename isconnected
  do [ ]
```

Descrizione: visualizza il nome dell'interfaccia mobile e fornisce se l'interfaccia è attivata o meno.

Blocco per firmare la transazione "Offline" - (SignerGenericPushRawTransactionOffline)

```
call BlockoinETHEREUM1 .SignerGenericPushRawTransactionOffline
  urlNetwork " [https://rinkeby.infura.io/v3/...440789c3... ] "
  hexPrivateKeySender " [9eC478ee49824890b4a120a8975cce3b1ae632058b0301f8... ] "
  nonceNumber get global nonce
  gasPrice " [25000000000 ] "
  gasLimit 21000
  toAddress " [0x92a2f157d5aec3fa79f92995fea148616d82c5ef ] "
  valueWei " [1000000000000000000 ] " x " [0.01 ] "
```

Dipendenze richieste: Block (eth_getTransactionCount), Block (eth_SendRawTransactionInfura)

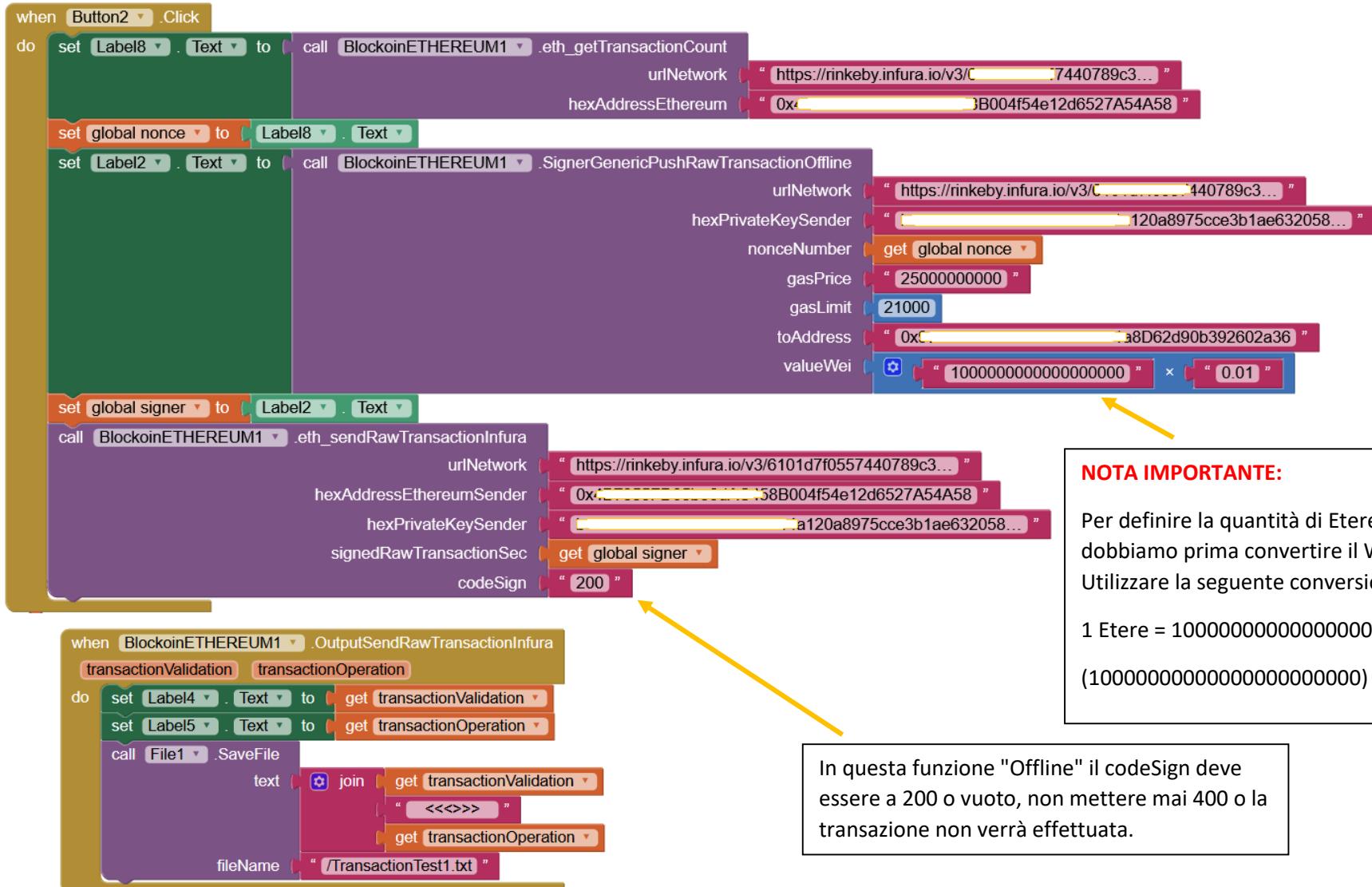
Parámetros de entrada: urlNetwork <String>, hexPrivateKeySender <String>, nonceNumber <String>, gasPrice <String>, gasLimit <Integer>, toAddress <String>, valueWEI <Integer>.

Parametri di uscita:

Uscite: transazione firmata da inviare. <Stringa>.

Descrizione: Prepara una nuova transazione da inviare (cifrata e firmata). Questo può essere elaborato senza connessione di rete o internet - "Offline".

Esempio di utilizzo completo con dipendenze di blocco (`SignerGenericPushRawTransactionOffline`).



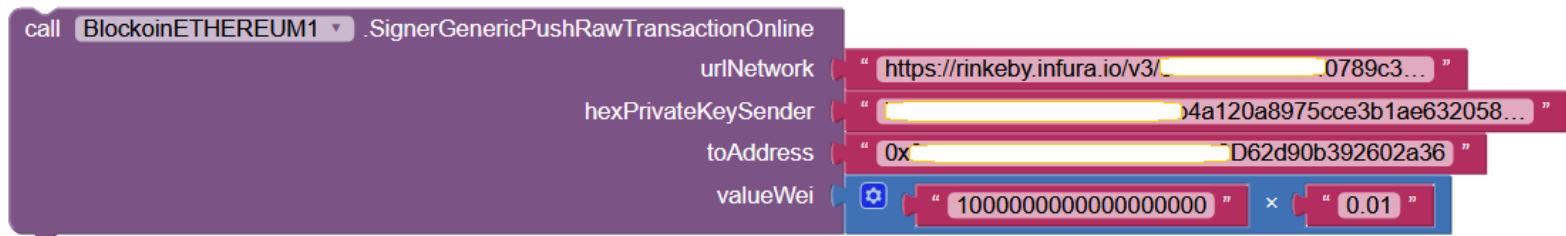
NOTA IMPORTANTE:

Per definire la quantità di Etere da inviare, dobbiamo prima convertire il WEI in etero. Utilizzare la seguente conversione:

(100000000000000000000000) x No. etere

In questa funzione "Offline" il codeSign deve essere a 200 o vuoto, non mettere mai 400 o la transazione non verrà effettuata.

Blocco per firmare la transazione "Online" - (SignerGenericPushRawTransactionOnline).

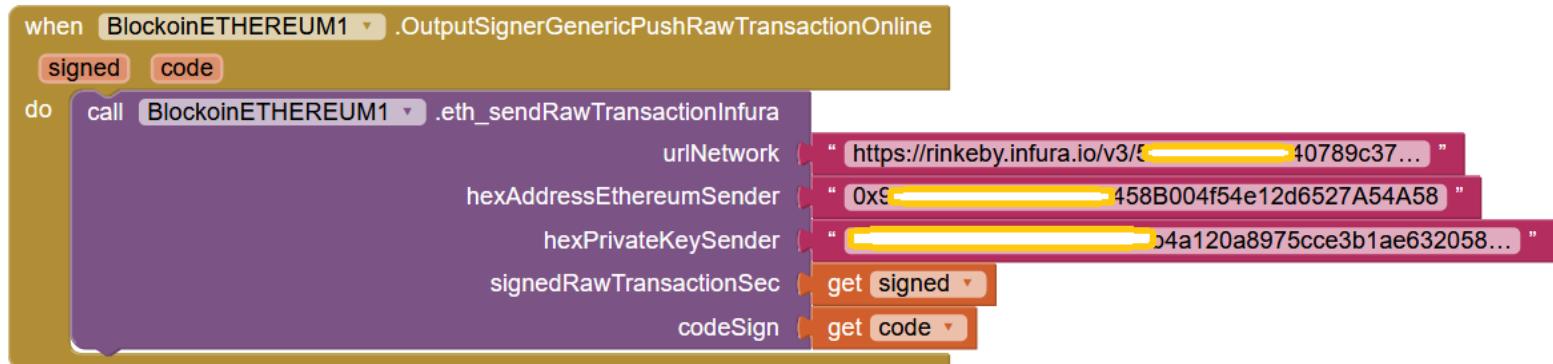


Unità(e) obbligatoria(e): Block (`eth_SendRawTransactionInfura`).

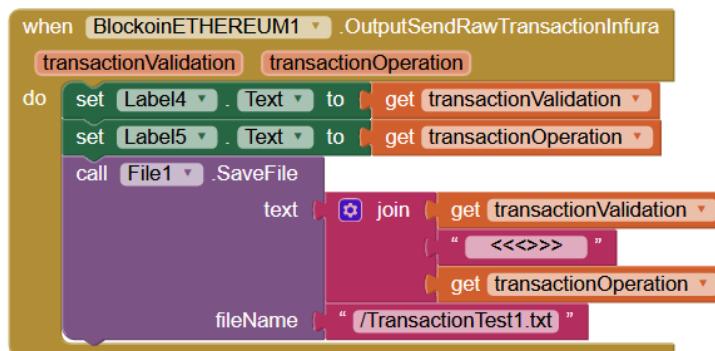
Parametri di ingresso: urlNetwork <String>, hexPrivateKeySender <String>, toAddress <String>, valueWEI <Integer>.

Parametri di uscita: Eventi utilizzati nel seguente ordine
(OutputSignerGenericPushRawTransactionOnline) e **(OutputSendRawTransactionInfura)**.

Uscite: **firmato<Stringa>**, codice<Stringa>.

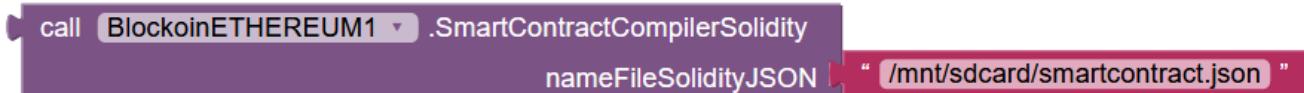


Gli output bloccano `eth_sendRawTransactionInfura`: `transactionValidation<String>` , `transactionOperation<String>`.



Descrizione: Prepara una nuova transazione da inviare (cifrata e firmata). È necessaria una connessione di rete o internet - "Online".

Blocco per la **compilazione del** contratto Smart "Online" - (**SmartContractCompilerSolidity**).



Parametri di ingresso: **nomeFileSolidityJSON <Stringa>**.

Parametri di output: Mostra il contratto smart compilato, questa funzione ci aiuta a verificare se è ben scritto prima di pubblicare un contratto smart nella rete dell'Ethereum.

Uscite: **Codice compilato**.

Il contratto Smart deve essere in un file in formato JSON.

Esempio di un contratto Smart di base in linguaggio Solidity.

```
solidità pragmatica ^0.5.0;

contratto mortale {
    indirizzo proprietario;
    funzione mortale() { proprietario = msg.sender; }
    funzione kill() { se (msg.mittente == proprietario) suicidio(proprietario);
    } }
    contratto saluto è mortale {
        saluto con la corda;
        funzione saluto(stringa _greeting) pubblica { saluto = _greeting; }
        funzione saluto() ritorni costanti (stringa) {ritorno saluto;}
```

Esempio di precedente contratto Smart in formato JSON con commenti.

```
# Controlla la compilazione della solidità tramite test non pubblicati
# Usando l'esempio della solidità del contratto "greeter", il "mondo dei
saluti" dell'Ethereum.
```

File: smartcontract.json

```
{
  "solidità": "contract mortal {\n    /* Definire il proprietario della\n    variabile del tipo indirizzo*/\n    proprietario dell'indirizzo;\n    /* questa funzione viene eseguita all'inizializzazione e imposta il\n    proprietario del contratto */\n    funzione mortal() { proprietario =\n    msg.sender; }\n    /* Funzione per recuperare i fondi sul contratto */\n    funzione kill() { se (msg.mittente == proprietario)\n    suicidio(proprietario); }\n}\n\ncontract greeter è mortale {\n    /*\n    definire saluto variabile del tipo stringa */\n    stringa di saluto;\n    /* questo viene eseguito quando il contratto viene eseguito */\n}
```

```

funzione greeter(string _greeting) public {\n            greeting =\n_greeting;\n}\n        /* funzione principale greet() constant\nreturns (string) {\n            return greeting;\n}\n        }\n            return\n            greeting",\n"Params." ["Ciao Coinsolidation Test"]\n}

```

NOTA IMPORTANTE: Il formato JSON deve sempre avere un'interruzione di riga alla fine di ogni riga.

Esempio di uscita Smartcontract compilata.

```

[
{
    "nome": " u003cstdin\u003e:greeter",
    "solidità": "contract mortal {\n/* Definire il proprietario della\nvariabile del tipo indirizzo*/\n    proprietario dell'indirizzo;\n/* questa funzione viene eseguita all'inizializzazione e imposta il\nproprietario del contratto */\n    funzione mortal() { proprietario =\nmsg.sender; }\n    /* Funzione per recuperare i fondi sul contratto */\n    funzione kill() { se (msg.mittente == proprietario)\n        suicidio(proprietario); }\n}\n\ncontract greeter è mortale {\n/*\ndefinire saluto variabile del tipo stringa */\n    stringa di saluto;\n/* questo viene eseguito quando il contratto viene eseguito */\n    funzione greeter(string _greeting) public {\n            greeting =\n_greeting;\n        }\n        /* funzione principale greet() constant\nreturns (string) {\n            return greeting;\n}\n        }\n            return\n            greeting",
    "cestino":\n"606060405260405161023e38038061023e8339810160405280510160008054600160a060\n020a031916331790558060016000509080519060200190828054600181600116156101000\n203166002900490600052602060002090601f016020900481019282601f10609f57805160\nff19168380011785555b50608e9291505b8082111560cc57600081558301607d565b50505\n061016e806100d06000396000f35b828001600101855582156076579182015b8281111560\n7657825182600050559160200191906001019060b0565b509056606060405260e060020a6\n00035046341c0e1b58114610026578063cfae321714610068575b005b6100246000543373\nffffffffffffffffff908116911614156101375760005473fff\nffffffffffff908116911614156101375760005473fff\na06020601f600260001961010086881615020190941693909304928301819004028101604\n0526080828152929190828280156101645780601f10610139576101008083540402835291\n60200191610164565b6040518080602001828103825283818151815260200191508051906\n0200190808383829060006004602084601f0104600f02600301f150905090810190601f16\n80156101295780820380516001836020036101000a031916815260200191505b509250505\n06
    "abi": [
        {
            "costante": falso,
            "ingressi": [],
            "nome": "uccidere",
            "uscite": [],
            "tipo": "funzione".
        },
        {
            "costante": vero,

```

```

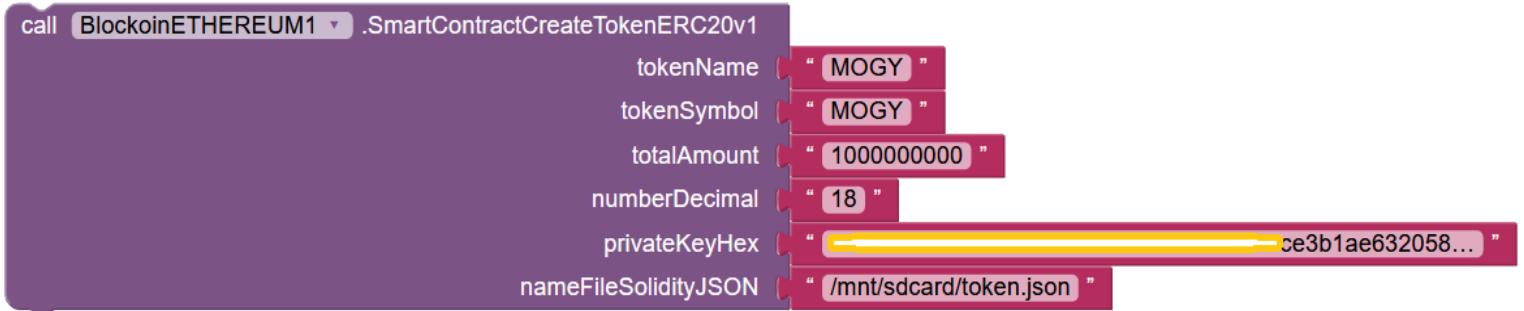
    "ingressi": [],
    "nome": "saluto",
    "uscite": [
        {
            "nome": "",
            "tipo": "stringa"
        }
    ],
    "tipo": "funzione".
},
{
    "ingressi": [
        {
            "nome": "_greeting",
            "tipo": "stringa"
        }
    ],
    "tipo": "costruttore"
}
],
"params": [
    "Ciao Coinsolidation Test"
]
},
{
    "nome": "mortale",
    "solidità": "contract mortal {\n /* Definire il proprietario della\n variabile del tipo indirizzo*/\n     proprietario dell'indirizzo;\n/* questa funzione viene eseguita all'inizializzazione e imposta il\n proprietario del contratto */\n     funzione mortal() { proprietario =\nmsg.sender; }\n/* Funzione per recuperare i fondi sul contratto */\n     funzione kill() { se (msg.mittente == proprietario)\nsuicidio(proprietario); }\n}\n\ncontract greeter è mortale {\n     /*\n      definire saluto variabile del tipo stringa */\n     stringa di saluto;\n/* questo viene eseguito quando il contratto viene eseguito */\n     funzione greeter(string _greeting) public {\n         greeting =\n_greeting;\n     }\n     /*\n      funzione principale greet() constant\nreturns (string) {\n         return greeting;\n     }\n     return\n     greeting",\n    "cestino":\n"606060405260008054600160a060020a03191633179055605c8060226000396000f36060\n60405260e060020a600035046341c0e1b58114601a575b005b60186000543373ffffffffffff\nffffffffffffffffffff90811691161415605a5760005473ffffffffffff\nffffffffffffffffffff16ff5b56",
    "abi": [
        {
            "costante": falso,
            "ingressi": [],
            "nome": "uccidere",
            "uscite": [],
            "tipo": "funzione".
},
{
            "ingressi": [],
            "tipo": "costruttore"
}
],

```

```

    "params": [
        "Ciao Coinsolidation Test"
    ]
}
]
Bloccare per compilare, creare e pubblicare il gettone ERC20 -

```

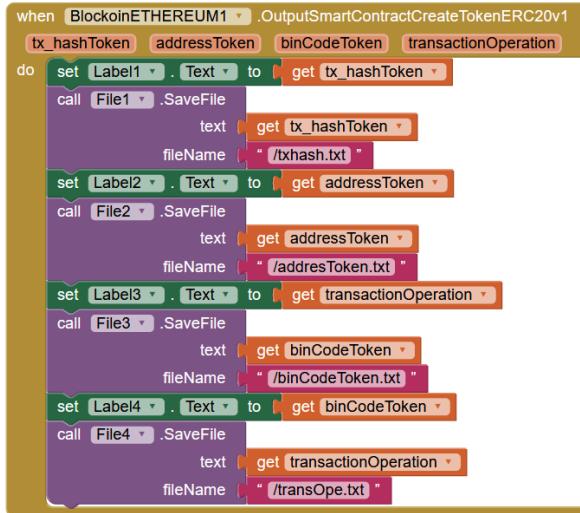


Unità obbligatoria: Block (`CreateTestingFile`). **IMPORTANTE**: Per prima cosa bisogna usare questo blocco per assicurarsi che il percorso sia corretto, questo perché se non si dà un percorso valido nel blocco (`SmartContractCreateTokenERC20v1`) la creazione del token non verrà eseguita perché il parametro di input "nameFileSolidityJSON" viene usato per creare un file temporaneo.

Parametri di ingresso: `tokenName` <String>, `tokenSymbol` <String>, `totalAmount` <String>, `numberDecimal` <String>, `privateKeyHex` <Integer>, `nameFileSolidityJSON` <String>. Questo file è il percorso valido per **creare un file temporaneo**, è necessario assicurarsi che il percorso sia valido per verificare che il file sia stato creato è possibile utilizzare il blocco (`CreateTestingFile`) dopo averlo utilizzato verificare che sia stato creato con successo.

Parametri di uscita: Evento (`OutputSmartContractTokenERC20v1`).

Uscite: `tx_hashToken<String>` , `addressToken<String>` , `binCodeToken<String>` , `transactionOperation<String>`.



Descrizione: Contratto intelligente "Token ERC20" - asset pubblicato sulla rete Ethereum. La versione 1 (v1) ha già impostato il parametro Limite gas a 500.000 WEI.

Esempio di uscita della funzione precedente `SmartContractCreateTokenERC20v1`.



9. Passi per creare un gettone CryptoToken o Cryptomoney Token.

Passo 1.

Verificare che il percorso temporaneo sul dispositivo mobile in cui è stato creato il contratto Smart sia valido e che un file possa essere creato con successo. Questo viene fatto utilizzando il Block (**CreateTestingFile**). Verificare che il file di test indicato nell'input "pathTestFile" sia stato creato, genericamente il percorso è dato da: **/mnt/sdcard/nome_file.txt**

Fase 2 (opzionale).

In questa fase verificheremo se il conto (indirizzo) sul quale verrà addebitata l'operazione ha un saldo sufficiente per eseguire la transazione di creazione e pubblicazione di un contratto Smart. Questo può essere verificato utilizzando il blocco (**eth_VerifiBalanceForTransaccionSmartContract**). L'uso di questo blocco è opzionale in quanto i blocchi che generano lo **SmartContractCreateTokenERC20v1** o **SmartContractCreateTokenERC20v2** contengono già questa verifica internamente.

Fase 3.

Selezionate quale blocco utilizzare per creare il token ERC20, avete due opzioni:

a.- Block **SmartContractCreateTokenERC20v1** ha già il valore implicito di Gas Limit assegnato con un valore di 500.000 WEI.

b.- Block **SmartContractCreateTokenERC20v2** ha la possibilità di configurare il limite di gas in base alle esigenze dell'utente finale o dello sviluppatore. Va notato che se viene dato un limite di gas molto basso, inferiore a 350.000 Wei, è molto probabile che la Smart contr.

Quarto passo.

Utilizzare i blocchi **SmartContractCreateTokenERC20v1** o **SmartContractCreateTokenERC20v2** assicurandosi che quando si utilizza la variabile di input "nameFileSolidityJSON" sia uguale alla variabile di input "pathTestFile" del blocco (**CreateTestingFile**) già verificata al punto 1.

Quinto passo.

Prima di eseguire la creazione di un token ERC20 con uno qualsiasi dei blocchi **SmartContractCreateTokenERC20v1** o **SmartContractCreateTokenERC20v2**, si raccomanda di salvare i valori degli eventi (risultati) come appropriato per salvare i risultati (tx_hashToken, addressToken, binCodeToken, transactionOperation). Vedi esempio di uscita della funzione precedente **OutPutSmartContractCreateTokenERC20v1**.

Sesta fase.

Eseguire la creazione del token ERC20 e poi pubblicarlo per la vendita. Vedere la sezione 10.

10.Come mettere in vendita un nuovo bene o il vostro gettone della cripta (Token ERC20).

Da quando abbiamo creato un gettone ERC20 - Cryptomoney Token (vedi **SmartContractCreateTokenERC20v1 Block** o con il blocco **SmartContractCreateTokenERC20v2 Block**) dobbiamo caricarlo su qualche Exchange in modo che chiunque nel mondo possa acquistarlo. Uno scambio è un sito su Internet dove vengono pubblicati nuovi gettoni.

Le borse sono classificate in due tipi, centralizzate e decentralizzate. La differenza principale è che uno è governato e controllato da una sorta di organismo internazionale (centralizzato) e quelli decentralizzati non hanno nessuno con i revisori. Anche se questo potrebbe dare più fiducia, la realtà è che recentemente quelli decentralizzati hanno preso più forza e sono utilizzati per lo più senza grossi problemi.

Una delle migliori pratiche nella gestione di beni di qualsiasi tipo è quella di non averli tutti in un unico conto, ma di distribuirli in più conti per la sicurezza delle chiavi sia private che pubbliche.

Nel nostro caso useremo uno scambio decentralizzato, ma già con una storia non male, useremo www.forkdelta.app

In questo momento dobbiamo aver già installato l'applicazione per il browser (Mozilla o Chorme) **METAMASK** www.metamask.io questo perché lo scambio per visitare la vostra pagina www.forkdelta.app deve collegarsi al conto che abbiamo Ethereum.

Un punto importante è che il nostro conto Ethereum che abbiamo già in METAMASK dovrebbe avere un saldo di più o meno 10 USD, questo perché quando pubblichiamo il nostro nuovo gettone ERC20 Token che abbiamo creato con il blocco **SmartContractCreateTokenERC20v1** o con il blocco **SmartContractCreateTokenERC20v2** dovremo pagare la transazione per pubblicarlo nella Borsa.

Per poter utilizzare la Borsa www.forkdelta.app dobbiamo avere i seguenti dati simbolici che vogliamo pubblicare per la vendita sulla Borsa.

Indirizzo del nuovo token ERC20 che abbiamo creato in precedenza.

0x54093F2C720b18Fd795645b5101A37EB49d1A94a

Numero di decimali che usiamo al tatto.

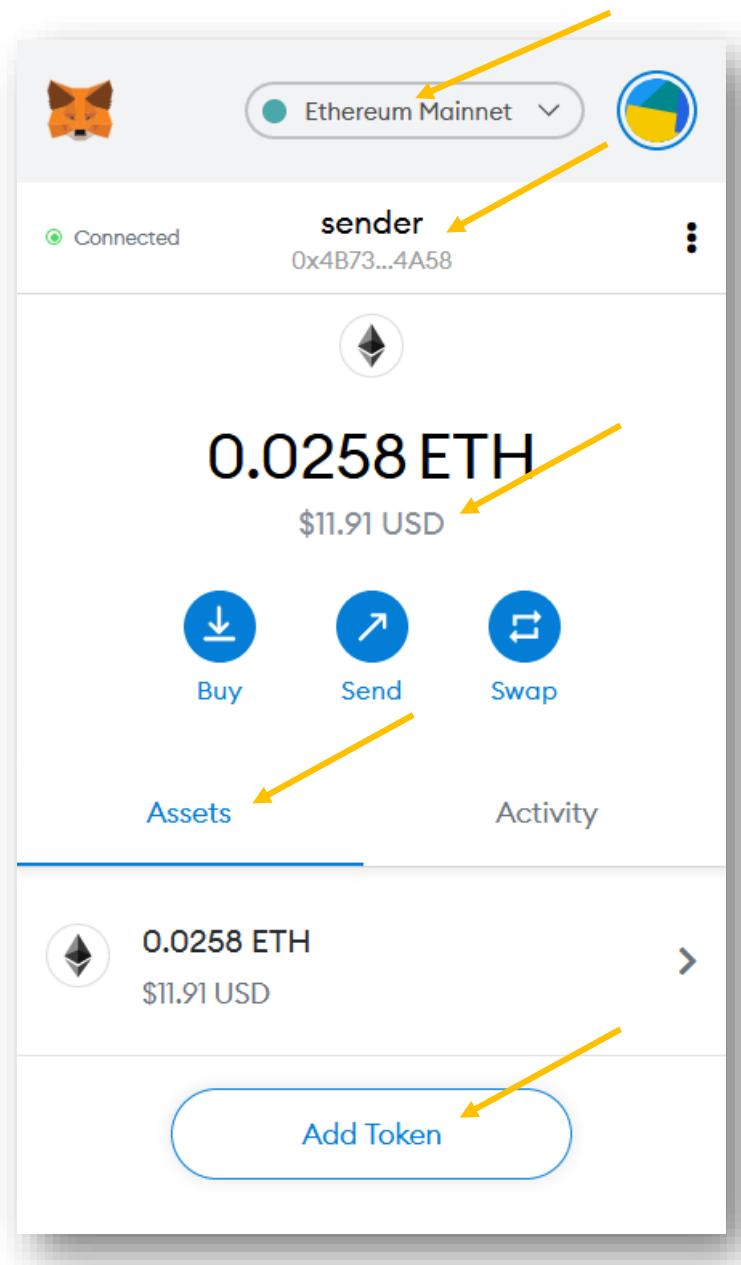
18

Il nome che identifica il gettone.

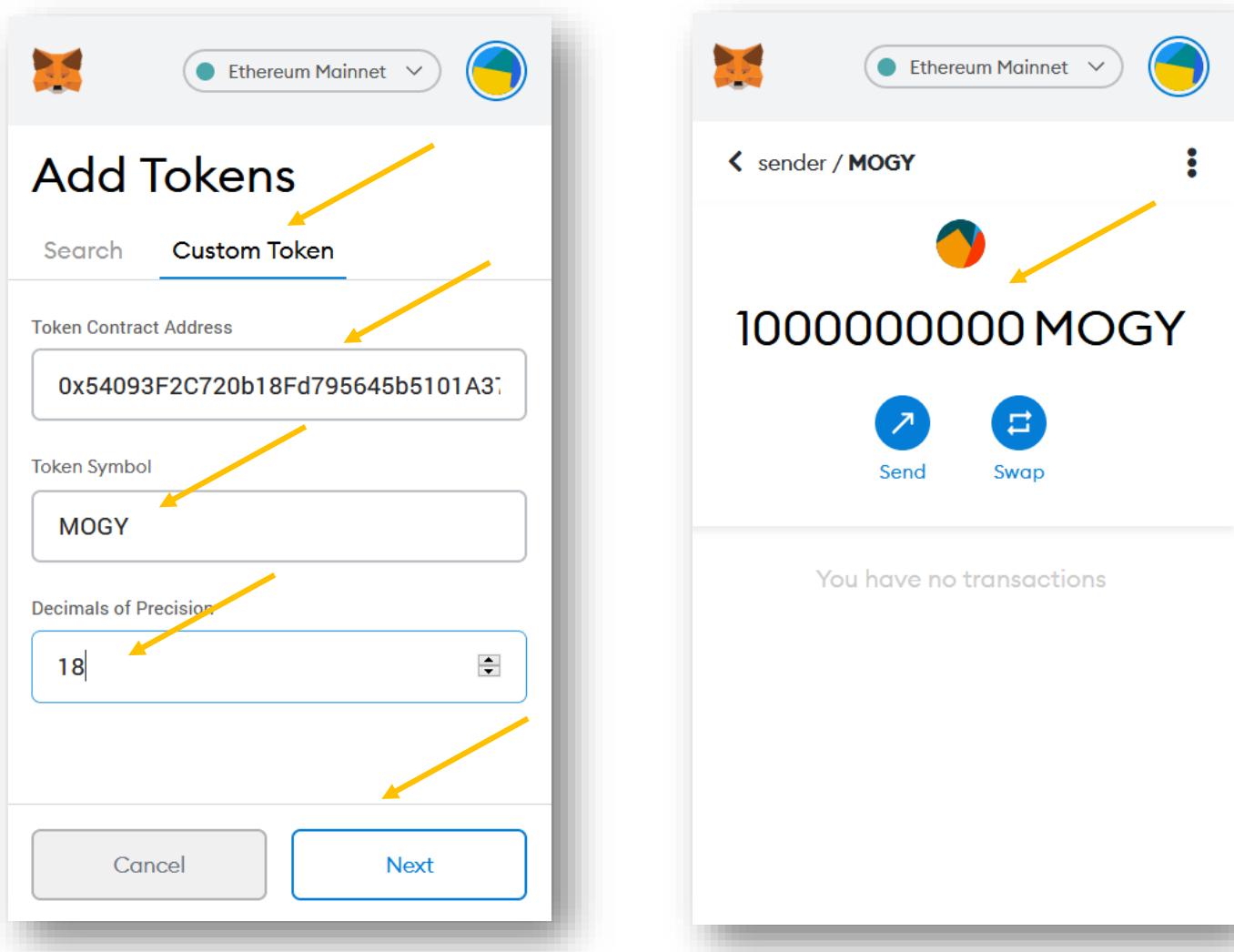
MOGY

Cominciamo col verificare che il token che abbiamo creato abbia i parametri sopra citati e che siano quelli con cui è stato creato inizialmente.

Andiamo da **METAMASK** e assicuriamoci prima di tutto di essere sul conto con cui abbiamo creato il gettone. Poi andate in fondo e cliccate sul pulsante "Aggiungi gettone".

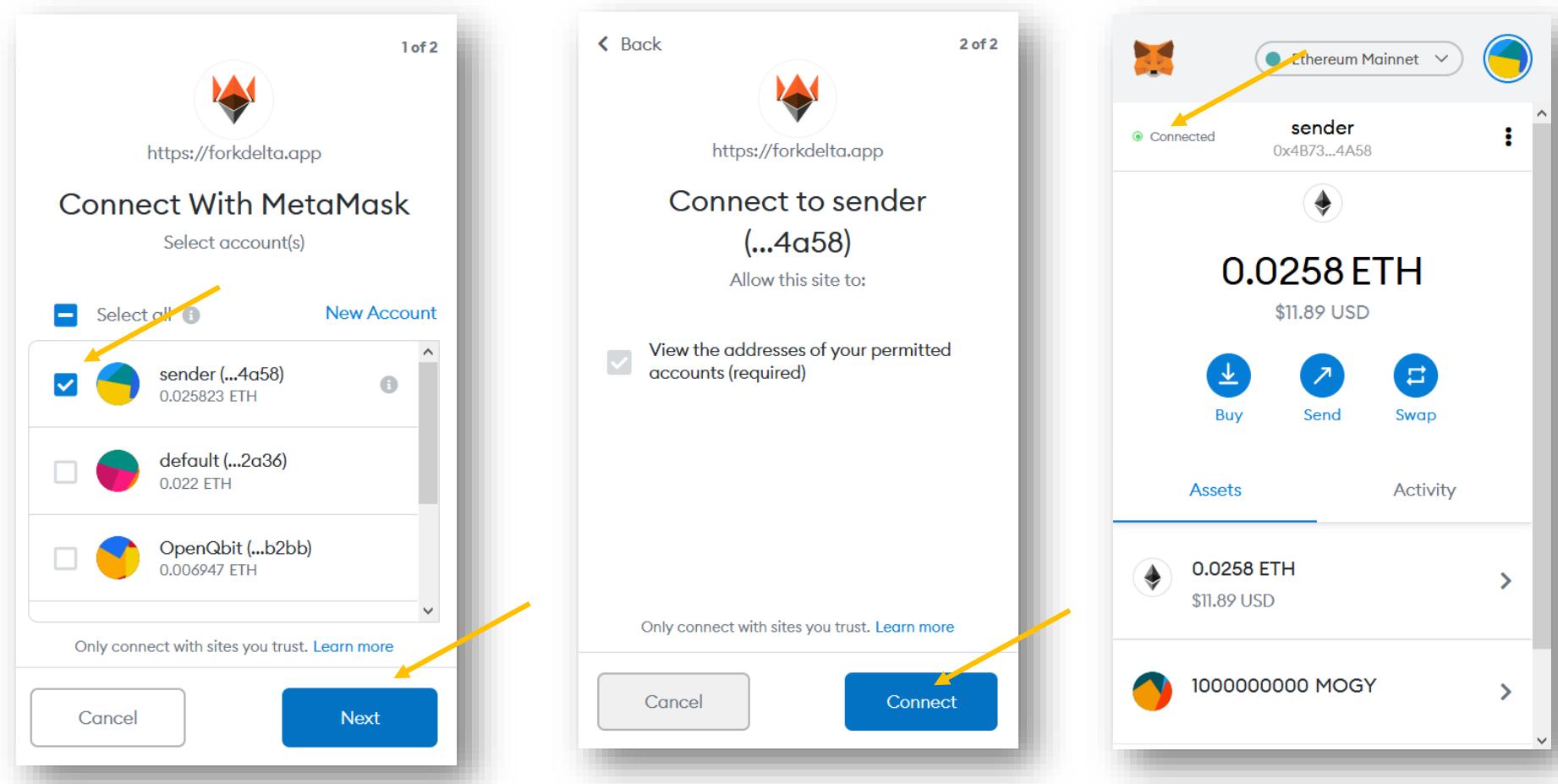


Aggiungeremo ora il nostro nuovo gettone per vedere il vostro attuale bilancio. Dopo aver cliccato sul pulsante "Gettone personalizzato" nella parte superiore della pagina, inserite le informazioni richieste nei seguenti campi e poi cliccate sul pulsante "Avanti". Dopo di che, il nostro nuovo gettone apparirà con il saldo che avete. Se non hai un saldo, controlla se sei nel conto con cui hai creato il gettone. Se non sei in un conto con cui hai creato il gettone, avrai un saldo zero perché non hai ancora acquistato alcun gettone.



Non appena caricheremo il nostro nuovo token in vendita sul sito [www.forkdelta.app](https://forkdelta.app)

Quando ti trovi nel sito di scambio, ti verrà chiesto di connetterti al sito <https://forkdelta.app>. Clicca sul pulsante "Avanti" qui sotto, poi "Connetti" e infine potrai verificare che sei connesso al sito di forkdelta.app



E' il momento di rilasciare il nuovo token sulla Borsa <https://forkdelta.app>

Cliccare sul menu superiore "DAI" e andare alla fine dello scorrimento e scegliere l'opzione "Altro".

The screenshot shows the ForkDelta interface for the DAI-ETH market. A yellow arrow points from the top-left towards the 'DAI' dropdown menu in the top navigation bar. Another yellow arrow points from the bottom-left towards the 'Other' option in the dropdown menu itself. The main trading area displays the Order Book, Price Chart, and Trades for the DAI-ETH pair. The 'Order Book' shows various buy and sell orders at different price levels. The 'Price Chart' shows the price movement over time. The 'Trades' section lists recent completed trades with their respective prices and volumes. Below the main trading area, there's a 'New Order' form and sections for 'My Transactions' and 'URL & Status'. On the right side, there's a 'Tweets' sidebar with a tweet from @ForkDelta.

Poi registriamo il nuovo gettone con i dati che già conosciamo.

The screenshot shows the ForkDelta app interface. A modal dialog titled "Other token" is open in the center. It contains three input fields: "Address" (0x54093F2C720b18Fd795645b5101A37EB49d1A94a), "Name" (MOGY), and "Decimals" (18). Yellow arrows point from the text to each of these fields. In the background, the main trading interface is visible, showing a "Balance" section with DAI and ETH amounts, a "Volume" section with a search bar, and a "Trades" section listing recent trades. The "Trades" section includes columns for DAI/ETH, DAI, and ETH, with various trade details listed below.

In questo momento il nuovo gettone apparirà nella parte superiore sinistra della Borsa, lo abbiamo solo caricato nella Borsa, dobbiamo pubblicarlo in modo che tutti possano acquistarlo e vederlo. Ora abbiamo bisogno di depositare un po' di etere (0,015) è sufficiente dal nostro conto allo scambio.

Token	Wallet	ForkDelta
MOGY	1000000000.000	0.000

Token	Wallet	ForkDelta
MOGY	1000000000.000	0.000
ETH	0.026	0.000
Amount	<input type="text" value="0.015"/>	<input type="button" value="Deposit"/>

Make sure MOGY is the token you actually want to trade.

Volume

Search for name, symbol, or address

ASTRO 0.001156789
AstroTokens
Bid 0.001133300 5.208Ξ Daily
Ask 0.001587654

REQ 0.000047998
Request Token
Bid 0.000025530 1.295Ξ Daily
Ask 0.000047998

SXDT 0.000200010
Spectre.ai D-Token
Bid 0.000203040 1.057Ξ Daily
Ask 0.000203040

SNOV 0.000004000
Snovio

Escribe aquí para buscar

Use this to deposit from your personal Ethereum wallet ("Wallet" column) to the current smart contract ("ForkDelta" column).

Dal momento che abbiamo effettuato il deposito possiamo depositare tutti i gettoni che vogliamo sulla Borsa www.forkdelta.app

Per depositare una quantità definita di gettoni dobbiamo creare un ordine di acquisto, questo viene fatto in basso dove c'è scritto "Nuovo Ordine" e clicchiamo sull'opzione "Vendi". Poi inseriamo la quantità di gettoni che vogliamo mettere a disposizione di qualsiasi acquirente nel mondo, il prezzo in Ether che vogliamo vendere ciascuno di essi (prezzo unitario) e il parametro "Scadenza" è la quantità di tempo che vogliamo avere questi gettoni per la vendita:

Tempo di scadenza = 14 secondi X importo inserito.

Esempio: 14 secondi X 10000 = 140.000 secondi = 1,62 giorni

The screenshot shows the ForkDelta app interface for trading the MOGY token. On the left, there's a 'Balance' section with tabs for Deposit, Withdraw, and Transfer. Under Token, it shows MOGY with a balance of 1000000000.000 and 0.000 in ForkDelta. Below this are input fields for Amount (MOGY and ETH) and Deposit buttons. On the right, the 'Order Book' shows no orders for MOGY/ETH, MOGY, or ETH. A 'New Order' form is open, with yellow arrows pointing to the following fields:

- Buy/Sell: 'Sell'
- MOGY amount: '10000'
- MOGY/ETH price: '0.05'
- ETH amount: '500.000'
- Expires: '10000'

The 'Sell' button at the bottom of the form is highlighted with a red arrow.

Lì, i vostri nuovi gettoni sono in vendita, chiunque può entrare e comprarli. A volte non riconosce il nuovo gettone, quindi dovranno essere venduti dall'applicazione Metamask.

Esempio di consultazione sul sito www.etherscan.io del nuovo gettone creato.

The screenshot shows the Etherscan interface for a specific transaction. The transaction hash is 0xd99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882. The status is Success. The block number is 11240201 with 224 block confirmations. The timestamp is 47 mins ago (Nov-12-2020 02:49:56 AM +UTC) | Confirmed within 30 secs. The transaction originated from address 0x4b7355fd05be6dac458b004f54e12d6527a54a58 and went to a contract at address [Contract 0x54093f2c720b18fd795645b5101a37eb49d1a94a Created]. The value was 0 Ether (\$0.00). The transaction fee was 0.011014691 Ether (\$5.06). The gas price was 0.000000041 Ether (41 Gwei) and the gas limit was 500,000. The gas used by the transaction was 268,651 (53.73%). A yellow arrow points from the 'Tx_hash' label to the transaction hash field. Another yellow arrow points from the 'To' label to the contract address. A third yellow arrow points from the 'Gas Price' label to the gas price field. Three callout boxes provide additional context: one for the transaction hash, one for the contract address, and one for the gas price.

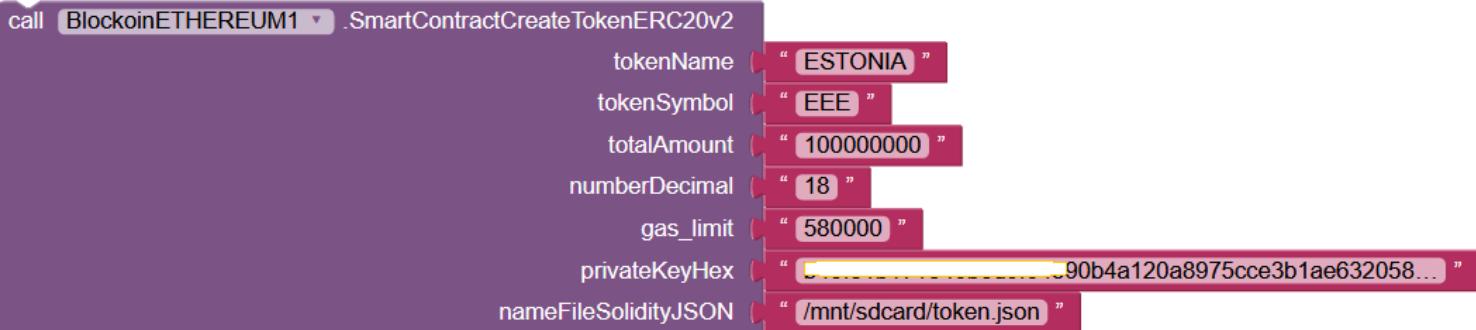
Transazione (Tx_hash) della creazione della rete Ethereum.

Indirizzo del contratto appena creato token.

**Solo il costo della rete dell'etereum.
Non include il costo dell'operazione + 15**

Overview	Status	Comments
② Transaction Hash:	0xd99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882	🔗
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%)	

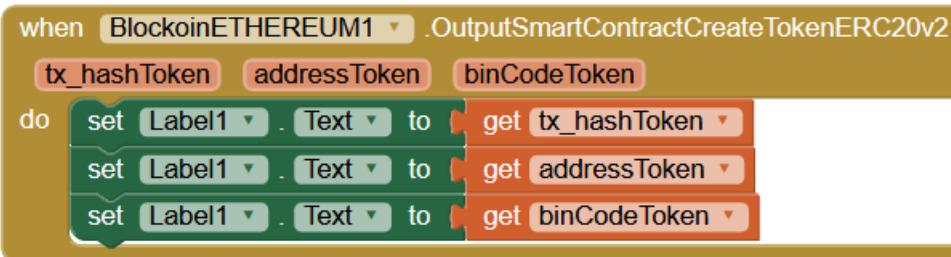
Bloccare per compilare, creare e pubblicare il gettone ERC20 - (SmartContractCreateTokenERC20v2)



Parámetros de entrada: `tokenName` <String>, `tokenSymbol` <String>, `totalAmount` <String>, `numberDecimal` <String>, `gas_limit` <Integer>, `privateKeyHex` <Integer>, `nameFileSolidityJSON` <String>.

Parametri di uscita: Evento (OutputSmartContractTokenERC20v2).

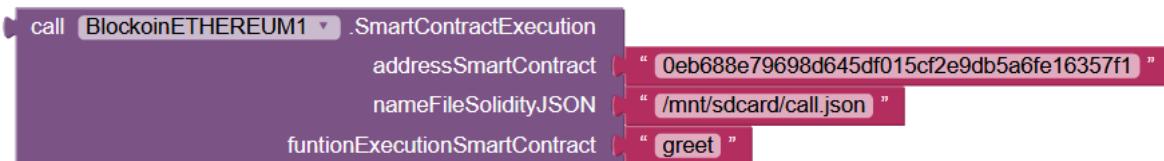
Uscite: `tx_hashToken`<Stringa>, `indirizzoToken`<Stringa>, `binCodeToken`<Stringa>.



Descrizione: Contratto intelligente "Token ERC20" - attivo pubblicato sulla rete Ethereum. La versione 2 (v2) può essere ottimizzata il valore del Gas Limit perché a seconda del contratto intelligente quanto velocemente si vuole effettuare la pubblicazione sulla rete dell'etereum. Si raccomanda che il valore minimo sia di 350.000 WEI per la pubblicazione senza errori o ritardi.

La differenza tra le funzioni di creazione di TokenERC20v1 e creazione di TokenERC20v2 è che nel caso della versione 1 il parametro di GasLimit è già preconfigurato e nella versione 2 può essere configurato secondo le esigenze dell'utente o dello sviluppatore.

Bloccare per chiamare o eseguire il token ERC20 - (**SmartContractExecution**)



Parametri di input: addressSmartContract<String>, nameFileSolidityJSON<String>, funtionExecutionSmartContract<String>.

Parametri di uscita: esecuzione della funzione specificata nel contratto Smart di riferimento.

Uscite: **esecuzione di un contratto intelligente**.

NOTA: Per essere eseguito, è necessario creare un file in formato JSON che contenga i parametri della chiave primaria dell'indirizzo che si vuole eseguire il contratto Smart, è necessario inserire il limite di gas (WEI).

Esempio di file JSON per eseguire le funzioni del contratto Smart compilato nell'esempio della funzione di compilazione di cui sopra. Il nome del file può essere arbitrario.

File: call.json

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

Esempio dell'output di esecuzione della funzione "saluto" del contratto Smart:

```
{
  "gas_limit": 20.000,
  "address": "0eb688e79698d645df015cf2e9db5a6fe16357f1",
  "risultati": [
    "Ciao Coinsolidation Test"
  ]
}
```

ERC20 Token Property Display Block - (**SmartContractGetCreationTx**)

call **BlockoinETHEREUM1** .SmartContractGetCreationTx
 txSmartContract " d99ff4d53f9b1f0687289674633d2acecf219011d163cdc0..."

Parametri di ingresso: **txSmartContract<Stringa>**

Parametri di uscita: Mostra le proprietà del contratto Smart con cui si fa riferimento (**Tx_hash**).

Descrizione: Mostra le caratteristiche principali e il codice ABI del contratto Smart di riferimento.

Proprietà di esempio del token ERC20, tokenNome del campione "MOGY" precedentemente creato con la funzione (**martContractCreateTokenERC20v1**)

```
{
  "block_hash": "cadb8914c43ed28fb370c9e1b6f5d8818ba31a1e944d1f7049441b982902dea8",
  "block_height": 11240201,
  "block_index": 170,
  "hash": "d99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882",
  "indirizzi": [
    "4b7355fd05be6dac458b004f54e12d6527a54a58"
  ],
  "totale": 0,
  "tasse": 11014691000000000,
  "taglia": 958,
  "gas_limit": 500.000,
  "gas_usato": 268651,
  "gas_price": 4100000000000,
  "contratto_creazione": vero,
  "trasmesso_da": "200.77.24.87",
  "confirmed": "2020-11-12T02:49:56Z",
  "received": "2020-11-12T02:50:13.185Z",
  "vedere": 0,
}
```


Blocco per visualizzare il codice di compilazione del token ERC20 - (SmartContractGetcodeABI)

call BlockoinETHEREUM1 .SmartContractGetcodeABI
addressSmartContract " 0eb688e79698d645df015cf2e9db5a6fe16357f1 "

Parametri di input: **addressSmartContract<String>**

Parametri di uscita: Mostra il codice di riferimento del contratto Smart.

Descrizione: Mostra il codice ABI del contratto Smart di riferimento.

Esempio di codice ABI di un contratto generico Smart:

```
{
  "solidità": "contract mortal {\\n /* Definire il proprietario della
variabile del tipo indirizzo*/\\n     proprietario dell'indirizzo;\\n
/* questa funzione viene eseguita all'inizializzazione e imposta il
proprietario del contratto */\\n     funzione mortal() { proprietario =
msg.sender; }\\n     /* Funzione per recuperare i fondi sul contratto */\\n
funzione kill() { se (msg.mittente == proprietario)
suicidio(proprietario); }\\n}\\n \\n \\n \\\ncontract greeter è mortale {\\n     /* definire saluto variabile del tipo stringa */\\n         stringa di saluto;\\n
/* questo viene eseguito quando il contratto viene eseguito */\\n
funzione greeter(string _greeting) public {\\n             greeting =
_greeting;\\n         }\\n         /* funzione principale greet() constant
returns (string) {\\n return greeting;\\n }\\n         }\\n             return
greeting",
  "cestino":
"606060405260405161023e38038061023e8339810160405280510160008054600160a060
020a031916331790558060016000509080519060200190828054600181600116156101000
203166002900490600052602060002090601f016020900481019282601f10609f57805160
ff19168380011785555b50608e9291505b808211560cc57600081558301607d565b50505
061016e806100d06000396000f35b828001600101855582156076579182015b828111560
7657825182600050559160200191906001019060b0565b509056606060405260e060020a6
00035046341c0e1b58114610026578063cf3ae321714610068575b005b6100246000543373
ffffffffffffffffffffffffffffffffffffffffffffff908116911614156101375760005473fff
ffffffffffffffffffffffffffffffffffffffffff16ff5b6100c9600060609081526001805460
a06020601f600260001961010086881615020190941693909304928301819004028101604
052608028152929190828280156101645780601f10610139576101008083540402835291
60200191610164565b6040518080602001828103825283818151815260200191508051906
0200190808383829060006004602084601f0104600f02600301f150905090810190601f16
80156101295780820380516001836020036101000a031916815260200191505b509250505
06
  "abi":
"[{\\"constant\":false,\\"inputs\":[],\\"name\":\"kill\",\\"outputs\":[],\\"ty
pe\\\":\"function\"}, {\\"constant\":true,\\"inputs\":[],\\"name\":\"greet\",\\"
outputs\":[{\\\"name\\\":\"\",\\\"type\\\\":\"string\"}],\\"type\\\":\"function\"}, {\\"
inputs\":[{\\\"name\\\\":\"_greeting\",\\\"type\\\\":\"string\"}],\\"type\\\":\"const
ructor\"}]",
  "creation_tx_hash":
"61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9",
  "created": "2016-07-20T01:54:50Z",
  "address": "0eb688e79698d645df015cf2e9db5a6fe16357f1"}
```

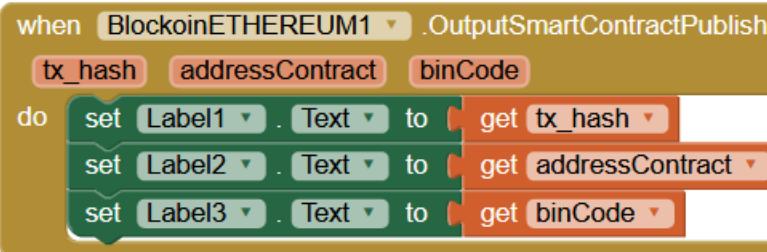
*Prima di utilizzare il seguente blocco (**SmartContractPublish**) è necessario utilizzare il blocco (**SmartContractCompilerSolidity**) per verificare se il contratto Smart è ben scritto.

Blocco per la pubblicazione del token ERC20 sulla rete Ethereum - (**SmartContractPublish**)



Parametri di ingresso: **nomeFileSolidityJSON<Stringa>**

Parametri di uscita: Mostra le proprietà del contratto Smart di riferimento. Nell'evento (**OutputSmartContractPublish**).



Descrizione: Pubblicare in ethereum network il contratto Smart a cui fa riferimento il file JSON.

File di esempio: **PublishSmartContract.json**

```
{
  "solidità": "contract mortal { \n /* Definire il proprietario della
variabile del tipo indirizzo*/\n     proprietario dell'indirizzo;\n\n/* questa funzione viene eseguita all'inizializzazione e imposta il
proprietario del contratto */\n     funzione mortal() { proprietario =
msg.sender; }\n\n     /* Funzione per recuperare i fondi sul contratto */
funzione kill() { se (msg.mittente == proprietario)
suicidio(proprietario); } \n\n contract greeter è mortale { \n     /* definire saluto variabile del tipo stringa */\n         stringa di saluto;\n\n/* questo viene eseguito quando il contratto viene eseguito */
funzione greeter(string _greeting) public { \n             greeting =
_greeting;\n         } \n\n     /* funzione principale greet() constant
returns (string) { \n         return greeting;\n     } \n         } \n         return
greeting",
  "Params": ["Ciao Test"],
  "pubblicare": ["saluto"],
  "privato": "3ca40...",
  "gas_limit": 500000
}
```

Come mostrato nel codice sopra è lo stesso codice JSON utilizzato nell'esempio della funzione di compilazione allo stesso codice i parametri sono stati aggiunti alla fine del file JSON:

Param: Parametri impliciti della Smart contr.

Pubblicare: Nome di come verrà pubblicato il contratto Smart.

Privato: la chiave primaria del conto che eseguirà il contratto Smart deve avere un saldo.

Gas_limit: è il saldo in WEI che si vuole spendere per la pubblicazione del contratto Smart.

Esempio di output al momento dell'esecuzione (pubblicazione del contratto Smart) il contratto Smart viene inserito nella rete dell'ethereum. Mostra la transazione eseguita "[creation_tx_hash](#)" e l'indirizzo assegnato del contratto Smart creato "[indirizzo](#)".

```
[  
 {  
     "nome": "saluto",  
     "solidità": "contract mortal {\n    /* Definire il proprietario della  
    variabile del tipo indirizzo*/\n    proprietario dell'indirizzo;\n    /* questa funzione viene eseguita all'inizializzazione e imposta il  
    proprietario del contratto */\n    funzione mortal() { proprietario =  
        msg.sender; }\n    /* Funzione per recuperare i fondi sul contratto */\n    funzione kill() { se (msg.mittente == proprietario)  
        suicidio(proprietario); }\n}\n\n\ncontract greeter è mortale {\n    /*  
    definire saluto variabile del tipo stringa */\n    stringa di saluto;\n    /* questo viene eseguito quando il contratto viene eseguito */\n    funzione greeter(string _greeting) public {\n        greeting =  
            _greeting;\n    }\n    /* funzione principale greet() constant  
    returns (string) {\n        return greeting;\n    }\n    }\n    return  
    greeting",  
    "cestino":  
"606060405260405161023e38038061023e8339810160405280510160008054600160a060  
020a031916331790558060016000509080519060200190828054600181600116156101000  
203166002900490600052602060002090601f016020900481019282601f10609f57805160  
ff19168380011785555b50608e9291505b8082111560cc57600081558301607d565b50505  
061016e806100d06000396000f35b828001600101855582156076579182015b828111560  
7657825182600050559160200191906001019060b0565b509056606060405260e060020a6  
00035046341c0e1b58114610026578063cf3ae321714610068575b005b6100246000543373  
ffffffffffffffffff908116911614156101375760005473fff  
ffffffffffff908116911614156101375760005473fff  
a06020601f600260001961010086881615020190941693909304928301819004028101604  
0526080828152929190828280156101645780601f10610139576101008083540402835291  
60200191610164565b6040518080602001828103825283818151815260200191508051906  
0200190808383829060006004602084601f0104600f02600301f150905090810190601f16  
80156101295780820380516001836020036101000a031916815260200191505b509250505  
06  
    "abi": [  
        {  
            "costante": falso,  
            "ingressi": [],  
            "nome": "uccidere",  
            "uscite": [],  
            "tipo": "funzione".  
        },  
        {  
            "costante": vero,  
            "ingressi": [],  
            "nome": "saluto",  
            "uscite": [  
                {  
                    "nome": "",  
                    "tipi": []  
                }  
            ]  
        }  
    ]  
}
```

```

        "tipo": "stringa"
    },
],
"tipo": "funzione".
},
{
"ingressi": [
{
"nome": "_greeting",
"tipo": "stringa"
}
],
"tipo": "costruttore
}
],
"gas_limit": 500.000,
"creation_tx_hash":
"61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9",
"address": "0eb688e79698d645df015cf2e9db5a6fe16357f1",
"params": [
"Ciao Test"
]
},
{
"nome": "mortale",
"solidità": "contract mortal {\n/* Definire il proprietario della\nvariabile del tipo indirizzo*/\n    proprietario dell'indirizzo;\n/* questa funzione viene eseguita all'inizializzazione e imposta il\nproprietario del contratto */\n    funzione mortal() { proprietario =\nmsg.sender; }\n    /* Funzione per recuperare i fondi sul contratto */\n    funzione kill() { se (msg.mittente == proprietario)\n        suicidio(proprietario); }\n}\n\ncontract greeter è mortale {\n    /*\ndefinire saluto variabile del tipo stringa */\n    stringa di saluto;\n/* questo viene eseguito quando il contratto viene eseguito */\n    funzione greeter(string _greeting) public {\n        greeting =\n_greeting;\n    }\n    /*\n        funzione principale greet()\n        constant\n        returns (string) {\n            return greeting;\n        }\n    }\n    return greeting,\n    "cestino":\n"606060405260008054600160a060020a03191633179055605c8060226000396000f36060\n60405260e060020a600035046341c0e1b58114601a575b005b60186000543373ffffffffffff\nffffffffffffffffffff90811691161415605a5760005473ffffffffffff\nffffffffffffffffffff16ff5b56",
"abi": [
{
"costante": falso,
"ingressi": [],
"nome": "uccidere",
"uscite": [],
"tipo": "funzione".
},
{
"ingressi": [],
"tipo": "costruttore
"}
],
"gas_limit": 500.000,

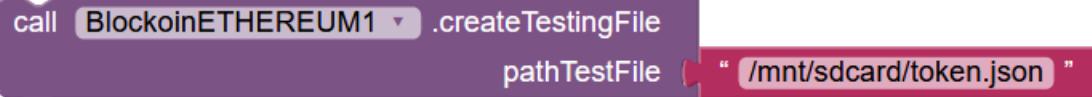
```

```

    "Params." ["Ciao Test"]
}
]

```

Blocco di test per creare il file - (**createTestingFile**)

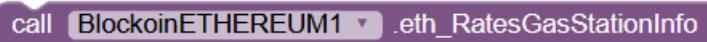


Parametri di ingresso: **pathTestFile<Stringa>**

Parametri di uscita: nel percorso di riferimento viene creato un file di prova.

Descrizione: Questo serve a controllare il percorso di creazione del file temporaneo valido per assicurarsi che sia corretto quando si utilizza il blocco (**SmartContractCreateTokenERC20v1**) o il blocco (**SmartContractCreateTokenERC20v2**).

Bloccare per ottenere le tariffe del prezzo del gas - (**eth_RatesGasStationInfo**).



Parametri di ingresso: **nomeFileSolidityJSON<Stringa>**

Parametri di uscita: Mostra le proprietà del contratto Smart di riferimento. Nel caso (**OutputEth_GasStationInfo**) i valori forniti sono riportati in GWEI.

Il Prezzo del Gas è il valore che sarà pagato ai sistemi che eseguono le transazioni nella rete dell'Ethereum. Questi sistemi sono comunemente noti come "minatori" e i valori del Prezzo del Gas sono una funzione della velocità (tempo e priorità) con cui la transazione viene eseguita nella rete dell'Ethereum.

I valori forniti si basano sui seguenti tempi di esecuzione. Questi tempi sono approssimativi e possono variare a seconda di come si procede con le richieste (transazioni) sulla rete dell'Ethereum.

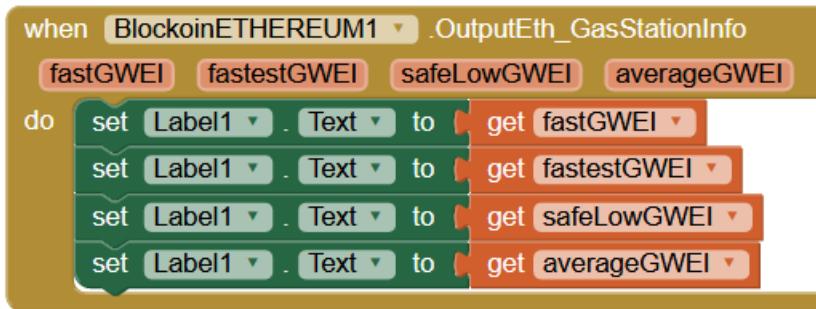
Veloce < 2 minuti.

Il più veloce < 30 secondi.

SafeLow < 30 minuti.

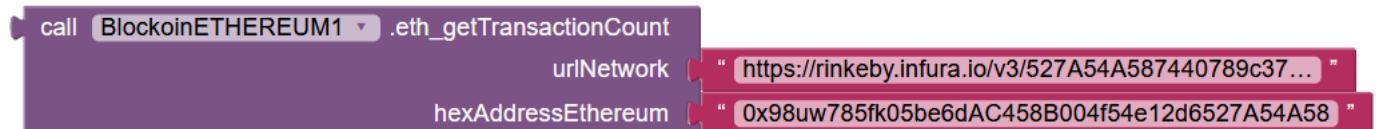
Media < 15 minuti.

Le transazioni effettuate con l'Exchange Ethereum Extension (EEE) utilizzano sempre il Prezzo del Gas = Media.



Descrizione: ottiene il prezzo del gas aggiornato al momento della richiesta per creare una nuova transazione

Bloccare per ottenere il numero "nonce" - (`eth_getTransactionCount`).



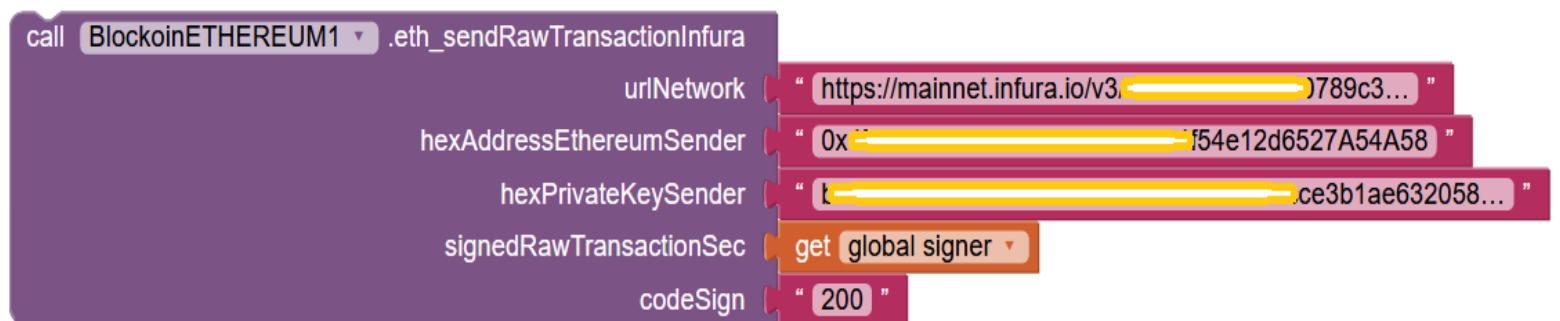
Parametri di ingresso: `urlNetwork<Stringa>`, `hexAddressEthereum<Stringa>`.

Parametri di uscita: mostra in formato esadecimale il numero consecutivo "nonce" dell'indirizzo di riferimento.

Il numero "nonce" è un numero incrementale che tiene traccia del numero di transazioni che sono state effettuate da un indirizzo specifico.

Descrizione: ottiene il numero "nonce" dell'indirizzo di riferimento.

Blocca l'invio di una transazione firmata - (`eth_SendRawTransactionInfura`).

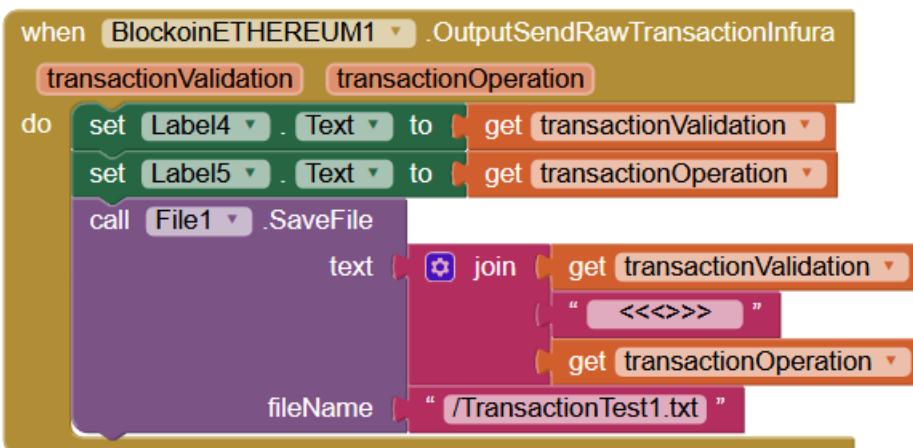


Dipendenze richieste: Block (eth_getTransactionCount), Block (SignerGenericPushRawTransactionOffline). Esaminare l'esempio del blocco (SignerGenericPushRawTransactionOffline).

Parametri di ingresso: urlNetwork <String>, hexAddressEthereumSender <String>, hexPrivateKeySender <String>, SignedRawTrasactionSec <String>, codeSign <String>.

Parametri di uscita: Evento (OutputSendRawTransactionInfura)

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



Descrizione: Fornisce due valori esadecimali come risultato delle transazioni. Il valore di transactionValidation è la transazione effettuata sulla rete dell'Ethereum che contiene il costo隐式的 dell'Ethereum. Il valore di transactionOperation è la transazione effettuata nella rete di Coinsolidation con il costo di 0,5 centesimi di dollari USA per ogni transazione in base al valore dell'etere al momento della transazione. Questo costo è per il pagamento dei servizi della rete di Coinsolidation.org ed è investito nella manutenzione, supporto e creazione di estensioni per il settore cripto e asset in tutto il mondo.

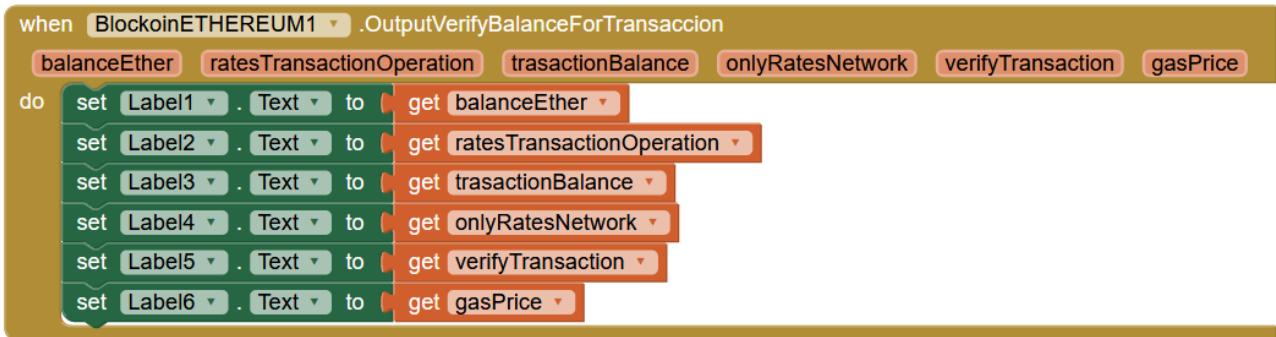
Blocco per calcolare il costo di una transazione standard - (eth_VerifiBalanceForTransaction)



Parametri di input: addressEthereumSender<String>, valueEthertoSend<String>.

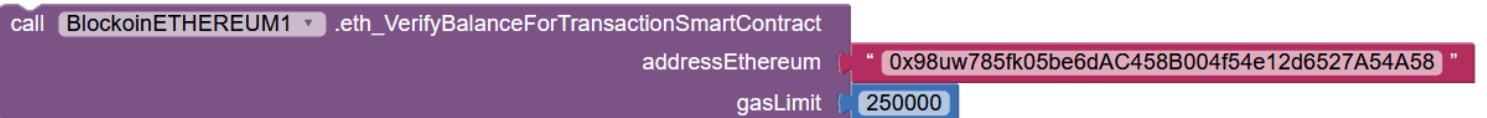
Parametri di uscita: Evento (OutputVerifyBalanceForTransaction).

Uscite: **balanceEther<String>** , **ratesTransactionOperation<String>** ,
trasactionBalance<String> , **OnlyRatesNetwork<String>** , **verifyTransaction<String>** ,
gasPrice<String>.



Descrizione: Fornisce i dettagli di quello che sarà il costo di una transazione standard con riferimento all'indirizzo di ingresso. Il parametro di uscita "verifyTransaction" ci dice se la transazione può essere fatta "True" o se l'indirizzo a cui si fa riferimento non ha un saldo sufficiente ci darà un "Falso".

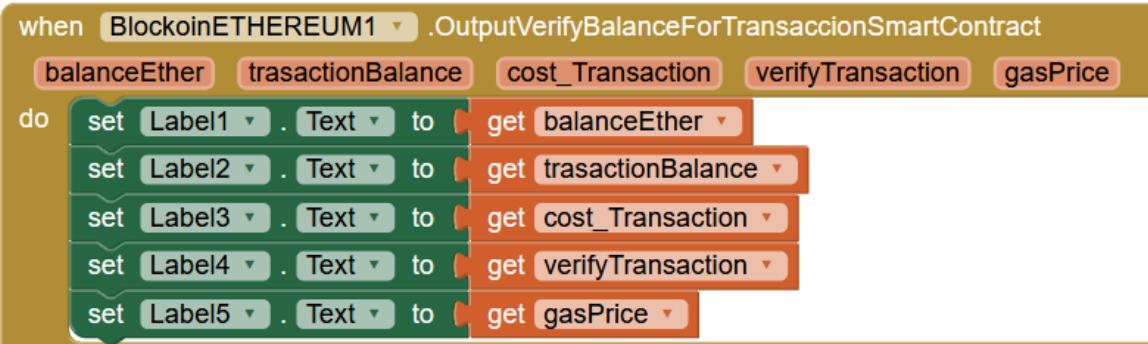
Blocco per calcolare il costo di una transazione standard -
(**eth_VerifiBalanceForTransaccionSmartContract**)



Parametri di ingresso: **addressEthereum<String>**, **gasLimit<String>**.

Parametri di uscita: Evento (**OutputVerifyBalanceForTransaccionSmartContract**)

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



Descrizione: Fornisce dettagli su quale sarebbe il costo approssimativo di una transazione standard per la pubblicazione di un **contratto Smart**, con riferimento all'indirizzo di entrata. Il parametro di uscita "verifyTransaction" ci dice se la transazione può essere fatta "True" o se l'indirizzo a cui si fa riferimento non ha un saldo sufficiente ci darà un "Falso".

BalanceEther: Il saldo dell'indirizzo di riferimento viene consegnato in eteri.

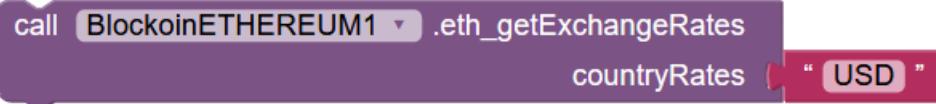
trasactionBalance: saldo dopo la transazione.

cost_Transaction: è il costo della transazione per pubblicare il contratto intelligente.

verifyTransaction: (balanceEther meno costo_Transaction).

gasPrice: Valore attuale del GasPrice utilizzato dai "Minatori", questo può variare ogni minuto.

Blocco per ottenere il prezzo Ether nella valuta del paese specificato - (eth_getExchangeRates).



```

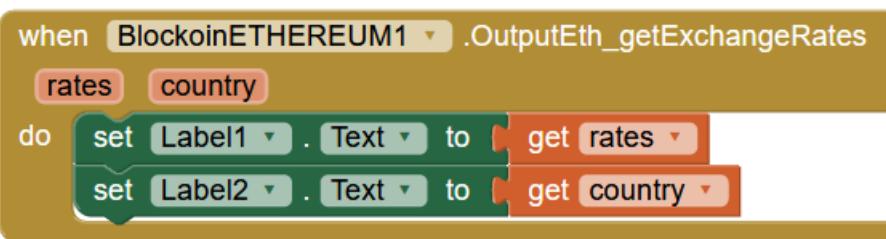
call [BlockoinETHEREUM1 v] .eth_getExchangeRates
    countryRates ["USD"]

```

Parametri di ingresso: **countryRates<String>**. Controllare il parametro di uscita "paese" dove contiene tutti i tipi di valuta del paese per scegliere quella desiderata.

Parametri di uscita: Evento (**OutputEth_getExchangeRates**).

Uscite: **tassi<Stringa>, paesi<Stringa>** uscita in formato JSON tutti i tassi dei paesi del mondo.



```

when [BlockoinETHEREUM1 v] .OutputEth_getExchangeRates
    rates
    country
do
    set [Label1 v].Text to (get [rates v])
    set [Label2 v].Text to (get [country v])
end

```

Descrizione: Fornisce il prezzo corrente di un Etere al tasso di cambio della valuta del paese di riferimento.

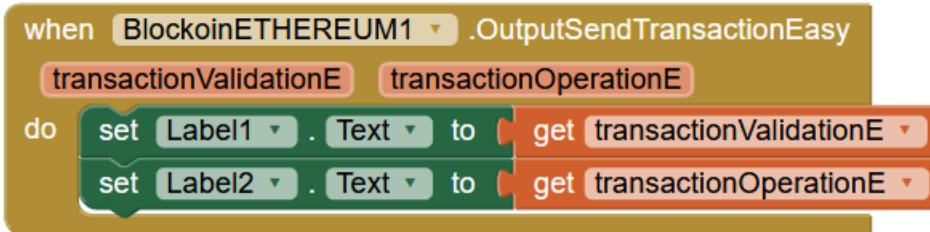
Bloccare per eseguire una transazione standard con i parametri minimi di input - (eth_sendTransactionEasy).



Parametri di ingresso: hexPrivateKeySender<String>, toAddress<String>, valueEther<String>.

Parametri di uscita: Evento (OutputSendTransactionEasy)

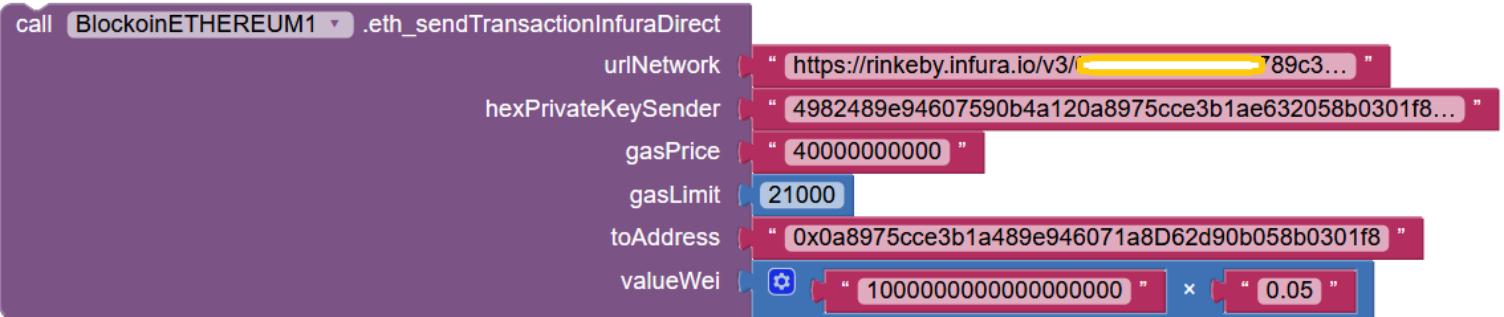
Outputs: transactionValidation<String> , transactionOperation<String>.



Descrizione: Funzione per effettuare una transazione standard nella rete dell'Ethereum, questa funzione è di immediata utilità non è necessario avere un conto in INFURA e sono necessari solo 3 parametri di ingresso, basta avere un saldo sufficiente per effettuare la transazione desiderata.

Le transazioni vengono immesse nella rete Ethereum direttamente utilizzando la biblioteca ufficiale Ethereum Web3j e la nostra rete Coinsolidation.org

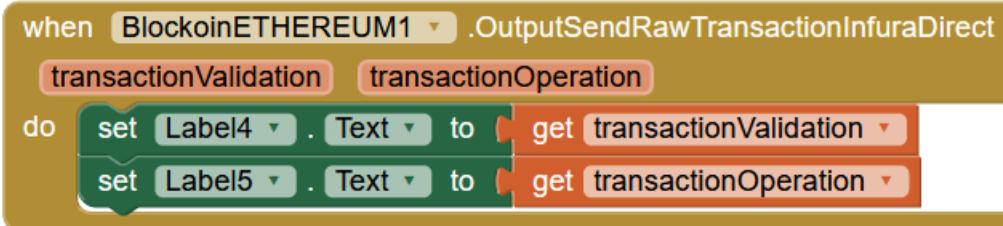
Bloccare per eseguire una transazione standard con i parametri minimi di input - (eth_sendTransactionInfuraDirect).



Parametri di input: `urlNetwork<String>`, `hexPrivateKeySender<String>`, `gasPrice<String>` , `gasLimit<String>`, `toAddress<String>`, `valueWEI<String>`.

Parametri di uscita: Evento (`OutputSendTransactionInfuraDirect`)

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



Descrizione: Funzione per l'invio di una transazione standard che contiene già la firma digitale implicita, utile per chi ha già una precedente conoscenza dei componenti di una transazione e vuole ottimizzare questi parametri in base alle proprie esigenze.

11. Calcolo del costo della transazione standard e della transazione del contratto Smart

Per il calcolo di una transazione standard sono necessari 3 parametri nella rete dell'Ethereum.

- 1.- Prezzo del gas.
- 2.- Limite di gas.
- 3.- Valore corrente di un Etere (moneta digitale dell'Ethereum).

GasPrezzo: Normalmente viene indicato in unità di GWEI (GigaWEI), che corrisponde, se 1 etere ha 1.000.000.000.000.000.000 wei, allora 1 GWEI equivale a 1.000.000.000.000. Questa unità serve come pagamento per i sistemi che eseguono tutte le transazioni della rete dell'Ethereum e sono chiamati "minatori", è distribuita in tutto il mondo. Il GasPrice non è un valore fisso ed è variabile e può cambiare da un minuto all'altro o secondo. Chi definisce il valore di GasPrice sono i "minatori" e dipende da quanto è satura la rete dell'Ethereum.

GasLimit: Questo valore è normalmente dato in unità di WEI e in una transazione standard un valore medio di default è di 21.000 WEI anche se può essere più alto o più basso a seconda del tipo di transazione che si vuole fare, in una transazione standard usiamo il valore per 40.000 WEI per garantire che non abbiamo il rifiuto per avere sotto il Gas Limit.

Valore di Ether: Anche questo valore è variabile ed è dovuto a diversi parametri di un mercato finanziario globale, questo valore può essere ottenuto da entità che hanno sempre aggiornato il valore di Ether in tutto il mondo chiamato scambi centralizzati e decentralizzati.

NOTA IMPORTANTE: Nella Exchange Ethereum Extension (EEE) utilizziamo un limite di gas più alto (40.000) questo è dovuto al fatto che in caso di non raggiungimento della quota minima stabilita dai "minatori" la TRANSAZIONE NON EFFETTIVA, tuttavia la rete dell'Ethereum, se addebiterà la spesa che ha effettuato nel calcolo della transazione senza effettuarla, Per questo motivo alcuni utenti non spiegano perché la loro transazione non viene eseguita, tuttavia verrà loro addebitato un importo o tutto il GasLimit che è stato offerto quando la richiesta di transazione è stata lanciata, per questo motivo dobbiamo sempre essere chiari su quali saranno i valori di GasLimit e Gas Price.

Il GasPrice può essere consultato con la funzione **eth_GetRatesGasStation** e il GasLimit per le transazioni standard deve essere superiore a 21.000 WEI e le transazioni per pubblicare e/o eseguire il contratto Smart devono essere almeno 500.000 WEI.

Costo di transazione standard.

È definito dalla seguente formula:

Costo = (GasLimit x (GasPrice / (1.000.000.000.000.000))) x Valore Etere.

Esempio:

Assumere i seguenti valori:

GasLimit = 40.000, **GasPrice** = 45 GWEI, **Valore in etere** = \$406.

Costo = (40.000 x (45.000.000.000.000 / (1.000.000.000.000.000))) x 406

Costo della transazione standard = 0,0018 etere x 406 USD = \$0,73 USD

Nel caso di una transazione di contratto Smart, è necessario sapere quale tipo di contratto Smart sarà pubblicato, poiché il costo è direttamente proporzionale al carico di lavoro che i "minatori" dovranno eseguire per elaborare il contratto Smart, in altre parole, la quantità di elaborazione nei sistemi informatici che i "minatori" gestiscono è più semplice.

Nel caso del contratto Smart un valore predefinito sarebbe quello di avviare il GasLimit con un valore di 500.000 WEI.

Un punto importante da tenere in considerazione è che quando si propone un GasLimit, i "minatori" non prenderanno necessariamente tutto l'importo proposto, cioè quando si invia una transazione, i "minatori" calcolano lo sforzo di calcolo ed estraggono ciò che sarà prelevato dal GasLimit, che può essere inferiore o uguale al GasLimit di default di 21.000 WEI offerto in alcuni casi.

Tutte le transazioni potranno essere consultate nel sito www.etherscan.io dove potremo consultare il dettaglio di ogni transazione.

Esempio di una transazione standard, in cui la transazione è stata inviata con un limite di gas di 40.000 WEI, tuttavia i "minatori" nel calcolo di quanto il costo di elaborazione sarebbe preso solo il 52,5%, cioè il valore predefinito che è di 21.000 WEI.

The screenshot shows the Etherscan Transaction Details page for a specific Ethereum transaction. The transaction hash is 0x7dae251f21c616fb04a94112633c97e04d11c91f4d06dd9b85ed11112f02703a. The transaction was successful and is currently in block 11234630 with 2 block confirmations. It was timestamped 52 seconds ago on Nov-11-2020 at 06:17:24 AM UTC, and it was confirmed within 38 seconds. 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 (52.5%) of the proposed gas limit of 40.000 WEI. The gas price was 0.000000032 Ether (32 Gwei).

Parameter	Value	Note
Transaction Hash	0x7dae251f21c616fb04a94112633c97e04d11c91f4d06dd9b85ed11112f02703a	
Status	Success	TransactionOperation o Validazione delle
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	
Value	0.001084598698481562 Ether (\$0.50)	Costo di transazione in etere: $21.000 \times 0,00000000032 = 0,000672$ Etere (\$0,31)
Transaction Fee	0.000672 Ether (\$0.31)	
Gas Price	0.000000032 Ether (32 Gwei)	Limite di gas proposto: 40.000 WEI
Gas Limit	40,000	
Gas Used by Transaction	21,000 (52.5%)	Limite effettivo di gas utilizzato nella transazione: 21.000
Nonce	36	

Ritornando alla transazione del contratto Smart e prendendo il 500.000 WEI GasLimit otteniamo il seguente costo se lo usiamo tutto.

Costo della transazione del contratto intelligente.

È definito dalla seguente formula:

Costo = (GasLimit x (GasPrice / (1.000.000.000.000.000)) x Valore Etere.

Esempio:

Assumere i seguenti valori:

GasLimit = 500.000, **GasPrice** = 45 GWEI, **Valore in etere** = \$406.

Costo = (500.000 x (45.000.000.000.000 / (1.000.000.000.000.000)) x 406

Costo della transazione pubblicare contratto Smart = 0,0225 etere x 406 USD = \$9.135 USD

Tutte le transazioni sono consultabili sul sito www.etherscan.io

NOTA: Per ottenere il costo totale della transazione è necessario sommarli:

Costo totale della transazione = costo della rete Ethereum + commissione di *Coinsolidation.org*

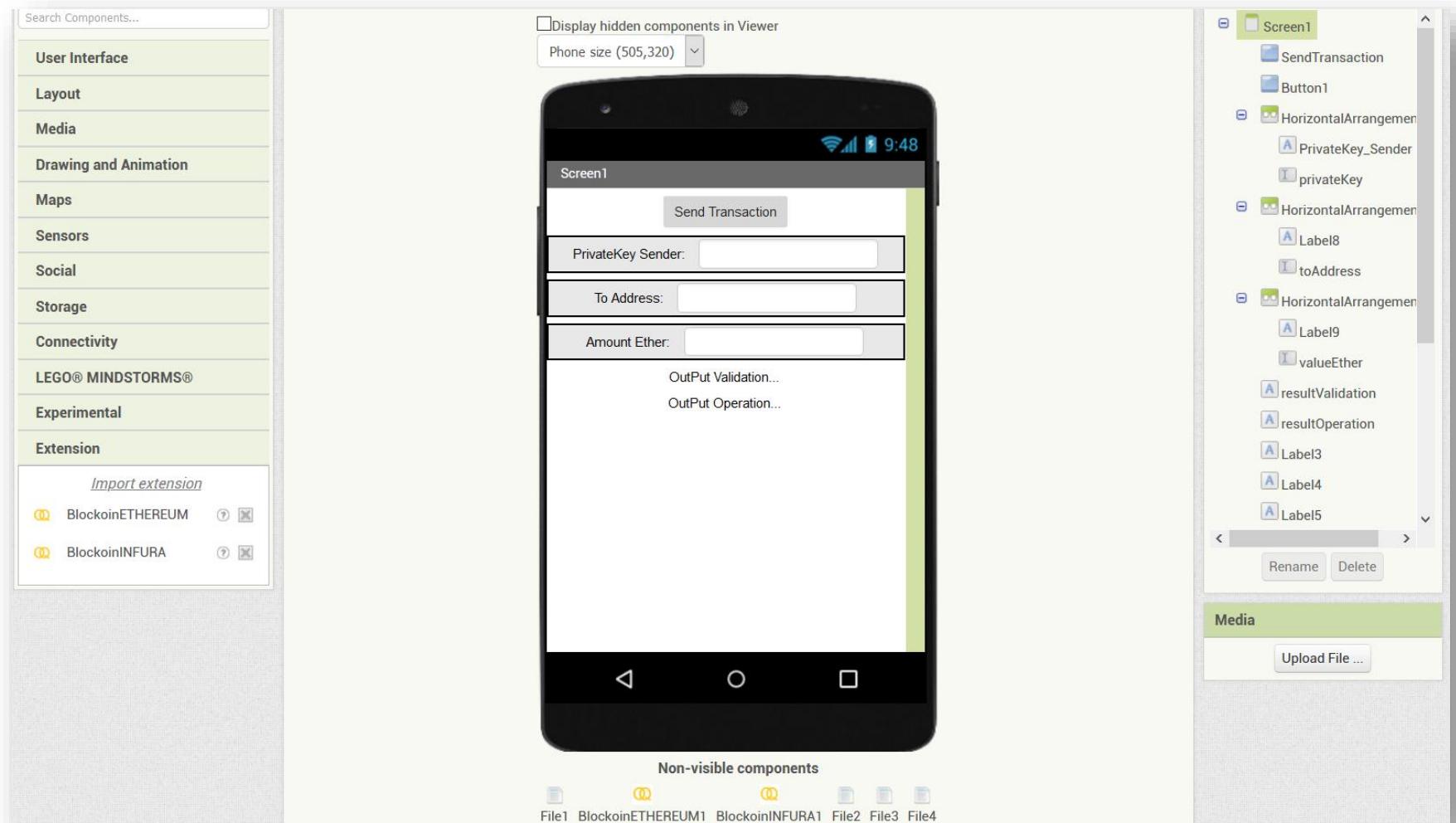
12. Quote di Coinsolidation.org

Transazione standard: \$ 0,5 centesimi USD + costo della rete Ethereum.

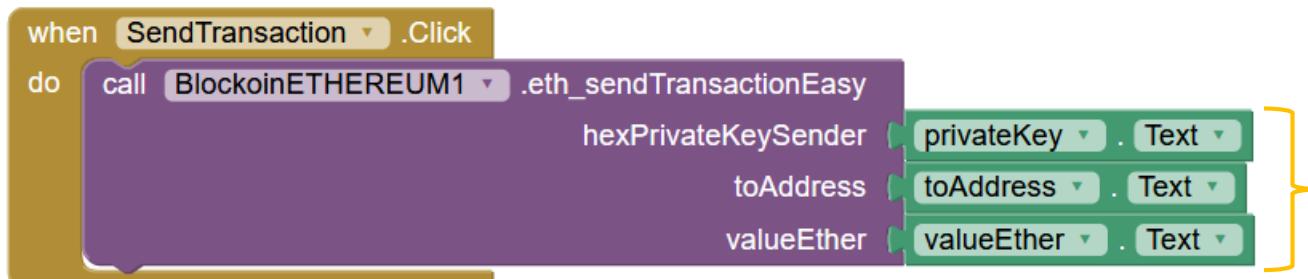
Transazione pubblicare e/o eseguire un contratto Smart: \$15 USD + costo della rete Ethereum.

13. Crea la tua App Android (Exchange) in 15 minuti.

Design in App Inventor (schermo). - 5 minuti.



Blocchi funzione (`eth_SendTransactionEasy`) ed evento (`OutPutSendTransactionEasy`) - 5 minuti

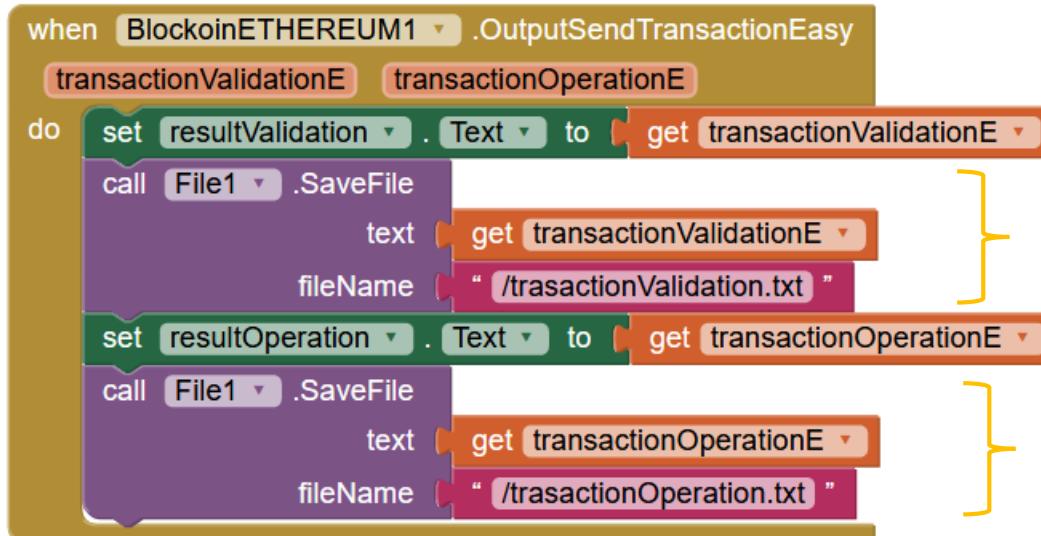


Dati di ingresso:

PrivateKey: chiave primaria per l'indirizzo del mittente.

toAddress: indirizzo esadecimale del destinatario.

valueEther: indicare la quantità di Etere che verrà inviata.



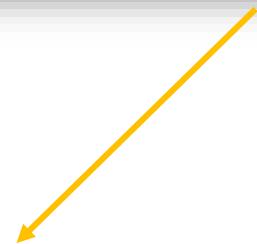
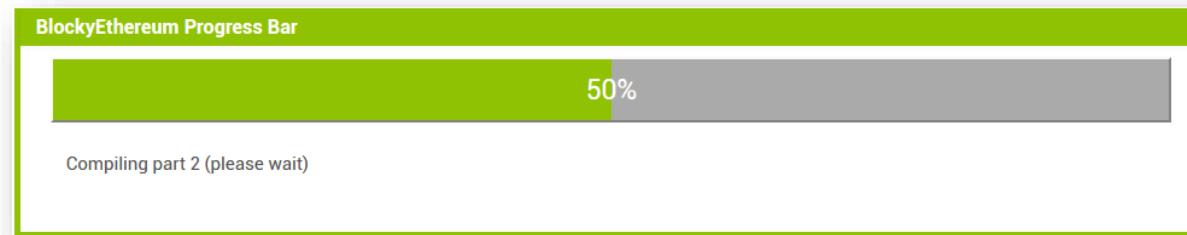
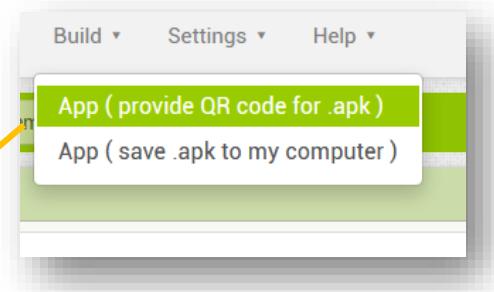
Salvare i risultati in file di testo:

Funzione File1: File **trasactionValidation.txt**

Salvare i risultati in file di testo:

Funzione File2: **Traslazione dei fileValidazione.txt**

Compiliamo, generiamo un file APK per installarlo sul dispositivo Android. - 5 minuti



NOTA: Quando la transazione viene eseguita, ci vorranno circa 6-8 secondi per rilasciare il pulsante "Invia transazione". A causa del tempo di connessione con la rete dell'Ethereum.

14. Token CoinSolidation.

Token Coinsolidation è un progetto con tre linee guida principali.

Immagina di avere la tua criptovaluta Token per far crescere la tua attività. Usando Coinsolidation puoi renderlo facile e semplice.

La prima è quella di creare la prima rete di estensioni basata sulla metodologia di programmazione visiva Blockly, in cui per il suo facile ed intuitivo utilizzo può essere utilizzata da chiunque non abbia precedenti conoscenze di programmazione, le estensioni consultabili nella nostra Roadmap (Libro Bianco) sono rivolte a due settori fondamentali dell'economia mondiale, il settore delle cripto-valute e/o gettoni ed il settore delle valute (fiat) o di uso comune a livello mondiale come il dollaro USD, l'Euro UE, la sterlina o qualsiasi altra valuta in uso.

La seconda linea guida del progetto CoinSolidation è la creazione di un nuovo algoritmo per consolidare gli indirizzi dei crittomontaggi attuali e futuri. Stiamo sviluppando un nuovo modello di algoritmo per creare un indirizzo che consolida diversi indirizzi di diverse catene di blocco in un indirizzo universale, vedi il nostro (Libro bianco) su www.Coinsolidation.org o <https://github.com/coinsolidation/whitepaper>.

La terza linea guida è quella di applicare la tecnologia del Quantum Computing nella sicurezza dell'ambiente CoinSolidation, questa sarà applicata con le estensioni già sviluppate degli algoritmi di sicurezza QRNG (Quantum Random Number Generator) e PQC (Post-Quantum Computing). Questi possono essere trovati nel repository ufficiale delle estensioni sul sito di Github, <https://github.com/coinsolidation>

Caratteristiche generali del CryptoToken CoinSolidation

Nome: CoinSolidation

Simbolo: CUAG

Tipo: NFT

Paese di lancio: Estonia

Sito web ufficiale: www.CoinSolidation.org

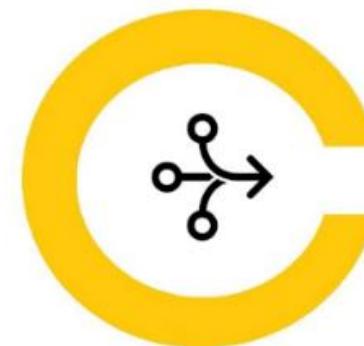
Azienda: CoinSolidation International.

Data di lancio: 30 dicembre 2020

Creata da: Lugu Samaya.

Algoritmo di consenso: Prova di Quantum.

Algoritmo di indirizzo: Indirizzo universale consolidato.



15. Licenze e utilizzo del software.

Licenze, termini e condizioni di utilizzo vedi www.coinsolidation.org o scrivi a info@coinsolidation.org