Introduzione alla programmazione di Smart Contract per Ethereum

Università di Verona
Imbriani Paolo - VR500437
Professoressa Migliorini Sara

January 14, 2025

Contents

1 Blockchain

Definition 1.1. Una *blockchain* è una catena di blocchi. I blocchi contengono informazioni sulle transazioni eseguite.

Definition 1.2. Una *transazione* è una struttura dati che racchiude informazioni circa il trasferimento di un certo quantitativo di token.

Esempio

Supponiamo di voler creare un consorzio tra noi studenti, dove ci vogliamo scambiare appunti. Di cosa abbiamo bisogno?

- Un **registro** condiviso e immutabile in cui memorizziamo gli scambi che vengono effettuati
- Per ogni **transazioni** vengono memorizzate informazioni come: quando, cosa, quanto, ecc.

Chi lo tiene il registro? Un entità terza scelta dal gruppo quindi bisogna mettersi d'accordo a chi affidare questo incarico. In una rete piccola come il nostro consorzio, potrebbe anche andare bene fare un'accordo tra di noi. Ma quando il consorzio diventa molto grande diventa difficile scegliere un entità centrale che si occupi di tenere il registro. L'idea per le blockchain è quella di eliminare l'entità centrale e affidare il compito di tenere il registro a **tutti** i partecipanti della rete. Quindi ciascun membro mantiene una *copia* personale del registro.

1.1 Caratteristiche delle blockchain

- Il **protocollo di consenso decentralizzata** permette di scrivere sul registro ma solo ad un costo. Se qualcuno "mente" riceve una penalità che rende tale comportamento non conveniente.
- **Immutabilità** significa che una volta che un blocco è stato scritto non può essere modificato.

1.2 Bitcoin

L'innovazione principale della blockchain è la definizione di un protocollo per la costruzione di una rete peer-to-peer in grado di raggiiungere un consenso circa uno stato globale senza l'intervento di **un'entità centrale**.

1.3 Ciclo di vita di una transazione bitcoin

Prendiamo come esempio Alice e Bob che vogliono scambiarsi dei bitcoin.

1. **Alice crea e firma la transazione**. Dichiara che vuole spedire un certo quantitativo di bitcoin a Bob. Utilizza uno strumento chiamato *wallet* per fare ciò, ovvero un software che consente di memorizzare le proprie chiavi private, di mantenere l'elenco dei token in nostro possesso, ci permette di

riceve, inviare e spendere le proprie criptovalute. La funzionalità principale di un **wallet** è quella di firmare le transazioni e la creazione degli indirizzi. Un indirizzo può essere visto come un IBAN. La probabilità che vengano creati due indirizzi uguali è molto bassa ma non è zero. Quando il wallet di Alice viene avviato per la prima volta:

- Genera una sequenza finita di bit generati casualmente ma SICURI (una chiave privata segreta).
- Calcola un indirizzo attraverso un'estrazione della chiave privata (hashing).
- Mostra l'indirizzo come una stringa alfanumerica o QR Code.
- L'indirizzo non è un informazione sensibile: Alice può pubblicarlo sulla sua pagina web.
- La chiave privata è un segreto: Alice deve mantenerla al sicuro.

  Definition 1.3. Un *wallet gerarchico deterministico* è un tipo di wallet che genera una serie di chiavi private e pubbliche che vengono derivate secondo una struttura ad albero.

  Definition 1.4. In bitcoin un *Unspent Transaction Output (UTXO)* è un insieme di token che sono attualmente asseganti ad un indirizzo e che non sono stati ancora spesi. L'insieme degli UTXO è una parte fondamentale dello stato globale della blockchian di Bitcoin.
  Il wallet di Alice seleziona un insieme di UTXO tra quelli a disposizione di Alice come input delle transazione in modo da avere un quantitativo sufficiente di token.

2. **Broadcast e validazione della transazione**: Alice invia la transazione al network.

3. **Mining o creazione dei blocchi**: Decidono che transazioni includere nel blocco.

   Definition 1.5. Il *mining* è il processo attraverso il quale i nodi creano nuovi blocchi a partire da un insieme di transazioni "pendenti". La costruzione di un blocco richiede lavoro (risoluzione di un problema matematico complesso).

   Ogni nodo (miner) riceve dalla rete le nuove transazioni e le memorizza in un'area temporeanea chiamata *mempool*. Circa ogni 10 minuti, ogni nodo individualmente sceglie un gruppo di queste per tentare la creazione di un nuovo blocco. Durante la creazione di un nuobo blocco (mining) ciascun nodo include:

   (a) Le transazioni da lui selezionate
   (b) Coinbase (transazione speciale che non ha input e ha come l'output l'indirizzo del miner e una quantità di token pari all'attuale mining reward, inizialmente 50 BTC ora 6.25)
   (c) hashing del blocco precedente
   (d) Bisogna aggiungere un Nonce (numero casuale tale per cui l'hash del blocco inizia con un certo numero di zeri all'inizio prefissato basato dalla difficoltà)

4. **Transazione confermata**: Il blocco viene aggiunto alla blockchain e la transazione è confermata solo dopo 6 conferme (dopo aver aggiunto 6 blocchi dopo il mio).

1.3.1 Fork Blockchain di Bitcoin

1. Stato iniziale: tutti i nodi hanno una visione comune della rete.
2. Fork: due nodi espandono la blockchain in modo diverso. Si verifica uno "split" della rete.
3. Uno split si espande ulteriormente.
4. La rete riconverge e vince la rete più lunga.

Ecco perché servono più conferme. Per garantire che ad un certo punto tutti i nodi della rete abbiano ricevuto tutti i blocchi della rete.

2 Ethereum e Smart Contracts

Definition 2.1. Uno *smart contract* è un accordo tra due o più parti, il cui rispetto può essere forzato automaticamente senza il bisogno di intermediario.

Ethereum è stato il primo progetto che ha implementato il concetto di Smart Contract in una blockchain.

Definition 2.2. In Ethereum uno smart contract è *programma* memorizzato all'interno della blockchain che viene eseguito automaticamente quando si verifica un evento (scheduling di una transazione) specifico.

Theorem 2.3. *Gli smart contract di Ethereum sono scritti in un linguaggio* ***Turing completo*** *e compilati in bytecode eseguibile sulla* ***Ethereum Virtual Machine (EVM)****.*

2.1 Smart Contract

Una volta che ho scritto e ho compilato il mio programma, lo devo pubblicare. Devo PAGARE per fare in modo che esso venga inserito nella blockchain. Quindi anche l'esecuzione deve essere pagata.

Osservazione. Ethereum non è solamente una criptovaluta (ETH).

- I nodi di una rete Ethereum costituiscono quello che viene definito un computer globale
- I nodi di Ethereum mettono a disposizione una EVM in grado di eseguire i programmi.
- Le risorse computazionali all'interno della EVM sono **limitate**. (non si tratta di un computer distribuito, ma di un computer replicato).

2.2 Ethereum

Theorem 2.4. *Ethereum può essere inteso come una macchina a stati ("infiniti") deterministica: ogni transazione produce un cambiamento da uno stato s ad uno stato successivo t.*

Una blockchain è un registro distribuito che memorizza transazioni, aggregate in blocchi. In Ethereum una transazione rappresenta un cambiamento all'interno di una memoria general purpose (una mappa che contiene qualsiasi tipo di coppia [chiave, valore]).

Attenzione! I cambiamenti apportati alla memoria devono essere deterministici, altrimenti non è possibile raggiungere un consenso.

Ethereum si comporta come un **computer globale decentralizzato**. Gli smart contract sono programmi che vengono eseguiti all'interno della EVM.

Definition 2.5. La *Ethereum Virtual Machine (EVM)* è una singleton globale: ciascun nodo mantiene una propria copia della EVM il cui stato deve essere consistente con la versione detenuta dagli altri nodi.

Ciascun nodo deve eseguire individualmente sulla propria copia locale le chiamate agli smart contract che sono attivate attraverso le transazioni.

2.2.1 Externally Owned Account (EOA) e contratti

In Ethereum esistono due tipi di indirizzi:

- Externally Owned Account (EOA): indirizzo associato ad una chiave privata. Fondi associati e generati dai wallet. Hanno il controllo sull'accesso ai fondi.
- Account di contratti: sono associati agli smart contract. Non hanno una chiave privata associata e possono inviare e ricevere token.

Una transazione può essere inizializzata solo da un EOA (mittente). Se il destinatiario della transazione è l'indirizzo di un contratto, lo scheduling della transazione causa l'esecuzione del contratto della EVM. Le transazioni possono contenere dei dati aggiuntivi che indicano quale funziona speciifica del contratto deve essere eseguita e quali parametri vanno passati. Un contratto può - durante la sua esecuzione - richiamare le funzionalità di altri contratti.

Definition 2.6. Deployare un contratto vuol dire registrare un contratto sulla blockchain e creare una transazione speicale che ha come indirizzo di destinazione 0x0 (chiamato indirizzo zero).

2.2.2 Transazioni di Ethereum

Ogni volta che viene effettuata una transazione verso un indirizzo del contratto, questa causa l'esecuzione del metodo indicato sulla EVM. Una transazione verso l'indirizzo di un contratto può contenere parametri **ether, data** o entrambi.

- **Ether**: quantità di token da inviare al contratto

- **Data**: dati aggiuntivi che vengono passati al contratto

Anche qua abbiamo il numero **Noance** associato a ciascun account che conta quante transazioni sono state inviate da un indirizzo. Garantendo così che tutte le transazioni mandate da un singolo indirizzo siano ordinate. Questo perché dobbiamo garantire lo stato progressivo che è presente nella memoria.

2.2.3 GAS

Una valuta parallela di scambio, chiamata **Gas**, è il "carburante" per controllare la quantità massima di risorse che una transazione può consumare durante la sua esecuzione. Ciascuna transazione che invoca una funzione di uno smart contract deve indicare:

- **gasPrice**: prezzo che il mittente è disposto a pagare per unità di gas (maggiore è il prezzo, potenzialmente maggiore è la velocità dello scheduling della transazione)
- **gasLimit**: massimo numero di unità di Gas che si è disposti a pagare per completare la transazione.

Attenzione! Se termina il Gas prima che il contratto abbia completato la sua esecuzione, quest'ultima viene terminata con un errore ed il Gas consumato fino a questo momento viene perso!

2.3 Consenso

Definition 2.7. Il *consenso* è un accordo generalizzato che deve essere raggiunto.

Esistono più definizioni:

Definition 2.8. In [informatica] il *consenso* è legato al problema più generale di sincronizzare dello stato di un sistema distribuito, in modo tale che i partecipanti ad un sistema distribuito possano raggiungere un accordo circa lo stato globale del sistema

2.4 Proof of Stake

Chi possiede più token più ha il diritto di scrivere sulla blockchain. La blockchain tiene traccia di una serie di *validatori* chiunque possiede degli Ethereum può dientare un validatore attraverso una transazione attraverso una transazione speciale con cui blocca i propri token in un deposito. I validatori possono perdere i propri depositi se mentono.

2.4.1 Validatori di Ethereum

Per partecipare alla rete Ethereum come validatore, un utente deve 32 ETH in un contratto di deposito a mettere in esecuzione 3 software sul proprio sistema.

- Execution client (impacchetta le transazioni e le esegue localmente)
- Consensus client (Creazione dei blocchi che poi verranno trasmessi sulla rete)

- Validator client (verifica che i blocchi siano validi)

Verrai inserito in una lista di attesa e verrai scelto per scrivere sulla blockchain in base al numero di token che possiedi. Quando un blocco è stato validato da un validatore, questo manda un voto (attestation) a favore del blocco ricevuto agli altri nodi della rete.

Nella PoS di Ethereum, il tempo è diviso in slot (12 secondi) ed in epoche (32 slots).

2.5 Esecuzione di una transazione

1. Creazione e firma della transazinoe
2. Verifica della validità della transazione (che abbia firmato e pagato abbastanza Eth)
3. Aggiunta alla mempool locale
4. Broadcast

2.5.1 Validazione di una transazione

1. Scelta del proponente del nuovo blocco
2. Scelta ed esecuzione delle transazioni
3. Costruzione del blocco: transazioni + payload
4. Broadcast

2.5.2 Validazione di un blocco

1. Ricezione di un nuovo blocco
2. Ri-esecuzione delle transazioni
3. Verifica di un blocco
4. Broadcast

alla fine, dobbiamo finalizzarla. Ci sta bisogno di un certo numero di slot che però sono molto più brevi rispetto a Bitcoin (12 secondi) Ogni epoca andiamo a verificare cosa sia successo dei 32 slot precedenti e andiamo a vedere come hanno votato i validatori dei 32 slot che hanno composto l'epoca. I validatori votano le coppie di checkpoint che considerano valide:

- Se una coppia di checkpoint (primo blocco di ogni epoca) raccoglie il 66% , il checkpoint viene aggiornato
- Il checkpoint più recente diventa justified
- Il checkpoint precedente diventa finalized

Per eliminare un blocco finalizzato, un attacante deve essere disposto a perdere 1/3 del totale degli Ether messi in stack dalla rete. La finalizzazione richiede i 2/3 della maggioranza. Chi vota in maniera opposto rispetto alla maggioranza perde il deposito dopo un certo numero di voti "sbagliati".
I fork in una rete PoS sono molto più rari rispetto ai bitcoin.
Benifici della PoS:

- Migliore efficienza energetica
- Minori barriere per l'entrata nella rete (democratizzazione)
- Maggiore sicurezza: gli svantaggi economici per chi non si comporta correttamente nel PoS sono sensibilmente più alte rispetto al PoW. I validatori sono disincentivati a mentire essendo che hanno più token in stake.

2.5.3 Utilizzo di Solidity

La fase di testing è essenziale quando si programma in Solidity, questo perché, essendo basato su Javascript non è fortemente tipizzato e quindi è facile commettere errori e bug.
L'ambiente di sviluppo permette di testare e debuggare i contratti, dashboards per l'interazione con Metamask.

2.6 Primo contratto in Solidity: definizione di keywords

2.6.1 Modificatori di visibilità

Le funzioni private di uno smart contract possono visibili da tutti, quindi hanno effetto solo sulla possibilità o meno di eseguire quella chiamata.

2.6.2 Modificatori di Side Effect

- View: la funzione può accedere ma non può alterare lo stato delle variabili del contratto Quindi in una funzione view posso leggere le variabili definite nel contratto ma **non** modificarle
- Pure: la funzione non accede e non modifica lo stato del contratto.
- Payable: la funzione può ricevere pagamenti

Perché sono importanti questi modificatori? In realtà qualsiasi operazione in lettura non costa niente. Quindi non c'è necessita di pagare per leggere il contenuto di una blockchain. Quello che costa è la scrittura in una blockchain.

2.6.3 Tipi e valori di ritorno

Solidity permette di **far ritornare più valori** da una funzione.

```
function split(int n, int divisor) public returns (int , int) {
    return (n / divisor, n % divisor);
}
```

Nell'istruzione returns si può anche specificare il tipo ma anche il nome e la funzione ritornerà automaticamente i valori con i nomi specificati.

```
function split(int n, int divisor) public returns (int quotien ,
    int rest) {
    quotient = n / divisor;
    rest = n % divisor;
}
```

2.6.4 Tipi base

I seguenti sono i tipi base di Solidity:

- Booleani
  - costanti true e false e operatori logici
- Integer (int, unint)
  - signed e unsigned si possono dichiarare di dimensioni crescente con un incremento di 8bit
  - Se non viene specificata la dimensione si assume int256 e unint256 rispettivamente
- Numeri decimali a virgola fissa (fixed, ufixed)
  - signed o unsigned dove M è la dimensione totale in bit e N è la dimensione della parte decimale
- Address, oggetto che rappresente un indirizzo Ethereum a 20-bytes
  - Offre diverse funzioni utili, le principali sono balance and transfer
  - Non esiste un operatore instanceof quindi i cast non falliscono mai
- Array, Array di array, stringhe
- Mappe o Array associativi
  - Le chiavi possono essere interi, booleani, indirizzi, byte e stringhe
  - I valori possono essere di qualunque tipo
  - Non c'è un metodo contains(): è necessario memorizzare in un altro modo le informazioni oppure usare le librerie esterne
  - Non è possibilire calcolare l'insieme delle chiavi o l'insieme dei valori
- Enumerazioni
- Strutture

2.6.5 Tipi di memoria in Solidity

Ethereum prevede 4 tipi di memoria:

- Storage: area di memoria dove risiedono le variabili di stato di un contratto, rappresenta il contenuto di una blockchain, tuttavia è costosa da utilizzare
- Memoria Volatile: Area di memoria economica per memorizzare valori temporanei e viene cancellata tra due chiamate esterne

- Stack: Memoria utilizzata per variabili locali gratis
- Calldata: Per copiare parametri di una funzione

I parametri (non i valori di ritorno) di funzione dichiarate devono essere messe nella CallData Le variabili di stato invece vanno salvate nello storage. Le variabili di stato (per convenzione) hanno un underscore davanti al nome per distinguerle dai parametri, perché le variabili di stato non possono avere lo stesso nome dei parametri.

3 Fase di testing

Il passo successivo consiste nello scrivefe dei test per verificare che si comportino come ci aspettiamo.

Il testin dei contratti si avvale delle seguenti librerie:

- Ethers.js
- Mocha
- Chai

Il progetto di test creato tramite il comando `npm hardhat init` contiene al suo interno degli esempi di test. L'esecuizione dei test avviene tramite il comando `npm run test`. Possono essere creati vari script di test ed è possibile eseguirli tutti insieme o singolarmente.

3.1 Transazioni e Calls

Definition 3.1. Una transazione modifica lo stato della rete (contenuto della blockchain).

Definition 3.2. Una call può essere utilizzata per eseguire il codice sulla rete che non modificare permamentemente lo stato della rete. Accesso in sola lettura (pure, view).

4 Smart Contract Issues

4.1 Re-entrancy

In Ethereum uno smart contract può richiamare ed utilizzare funzioni definite in altri smart contract. Gli smart contract possono getstire Ether ed inviare Ether a degli EOA.

- Per eseguire il trasferimento di Ether, devono eseguire la funzioni external `transfer()` o `send()`
- Una external call può essere deviata da un attaccante, il quale può forzare uno smart contract ad eseguire del codice diverso attraverso la funzione di fallback, incluso delle chiamate ricorsive che ritornano all'attacante stesso.
- **Re-entrancy** è un attacco che sfrutta la possibilità di eseguire chiamate ricorsive in Ethereum.

- L'attacco diventa particolarmente pericoloso quando lo smart contract che viene attaccato gestisce Ether.
- Il codice malevolo esegue una funzione sul contratto vulnerabile, eseguendo delle operazioni che lo sviluppatore non aveva previsto.

Send() e Transfer() non sono sicure e quindi hanno introdotto un limite di gas (2300) attualmente non sufficienti per un attacco di reentrancy; in futuro potrebbero essere deprecate.

4.2 Wrap Around Data Types: Arithmetic Overflow and Underflow

L'EVM prevede per gli interi dei tipi di dato a lunghezza fissa. La dimensione delle variabili può essere sfruttata da un attaccante per causare un overflow o un underflow.