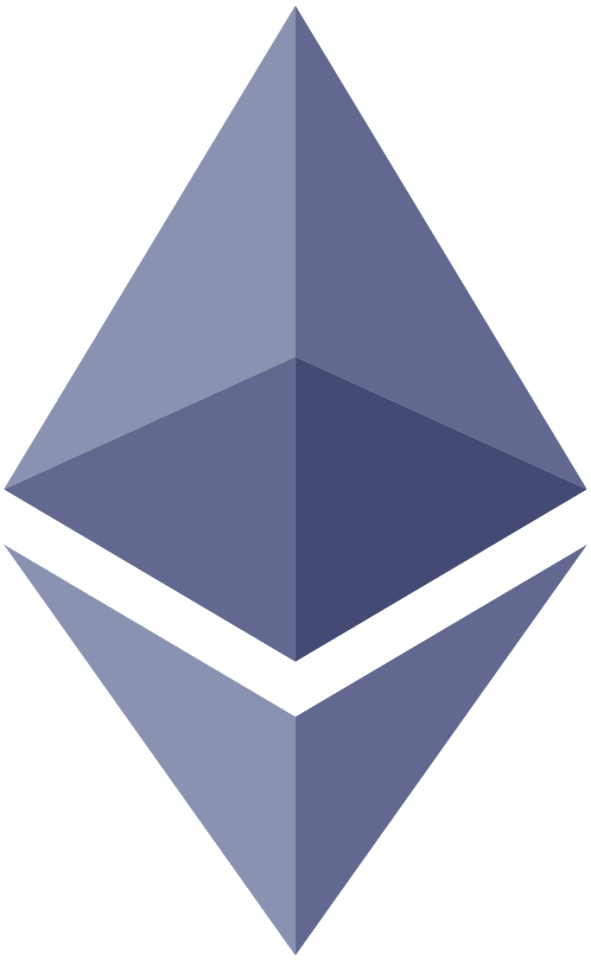
Smart Contracts

Moroni Luca - Matr: 311279

Giugno 2019



Contents

1 Smart Contracts

1.1 Accenno alle blockchain

La blockchain (letteralmente "catena di blocchi") è una struttura dati condivisa e immutabile. È definita come un registro digitale le cui voci sono raggruppate in blocchi, concatenati in ordine cronologico, e la cui integrità è garantita dall'uso di sistemi crittografici. Sebbene la sua dimensione sia destinata a crescere nel tempo, è immutabile in quanto, di norma, il suo contenuto una volta scritto non è più né modificabile né eliminabile, a meno di non invalidare l'intera struttura.

Non è richiesto che i nodi coinvolti conoscano l'identità reciproca o si fidino l'uno dell'altro. Difatti, per garantire la coerenza tra le varie copie, l'aggiunta di un nuovo blocco è globalmente regolata da un protocollo condiviso, la validita di un blocco deve essere approvato dal 50% + 1 dei nodi, ciò rende impossibile una manomissione della rete a meno di non prenderne il controllo di una fetta decisamente assurda. Una volta autorizzata l'aggiunta del nuovo blocco, ogni nodo aggiorna la propria copia privata: la natura stessa della struttura dati garantisce l'assenza di una sua manipolazione futura, questo è garantito da funzioni hashing e dal fatto che ogni nodo (della chain, solitamente il cui contenuto corrisponde ai dati che la blockchain memorizza, nel caso di bitcoin e affini sono le transazioni) per essere inserito nella blockchain deve contenere la stringa di hash (sha256, nel caso di BTC) del blocco precedente, andandolo a cristallizzare all'intero della catena. Le caratteristiche che accomunano i sistemi sviluppati con le tecnologie Blockchain sono decentralizzazione, disintermediazione, tracciabilità dei trasferimenti, trasparenza/verificabilità, immutabilità del registro e programmabilità dei trasferimenti. I nodi della rete inoltre vengono considerati tutti allo stesso livello, in un sistema peer-to-peer, ciò aggiunge la carattestiristca "distribuito".

Le blockchain garantiscono inoltre l'impossibilita di un double spending in quanto protocolli come BTC utilizzano controlli sulla validità e unicità della transazione prima di farne richiesta di inserimento nella chain stessa, Accenniamo che i controlli si basano direttamente su una fiducia nei confronti della rete e del ledger quindi è una definizione quasi induttiva. Il caso base puo essere rappresentato dalla veridicita della transazione 0, e il passo induttivo mantiene vera la condizione basandosi sui metodi crittografici e logici utilizzati dal protocollo stesso.

Teniamo conto anche che la tecnologia blockchain è molto studiada in

ambito accademico, al di fuori delle transazioni di monete, anche per il suo utilizzo come semplice database distribuito immutabile nel quale potenzialmente si potrebbero salvare una quantita indefinita di dati in modo permanente

1.2 proof of work

Proof of Work, o PoW, è un algoritmo che viene utilizzato da diverse criptovalute - come Bitcoin, Ethereum, Litecoin - per raggiungere un accordo decentralizzato tra diversi nodi nel processo di aggiunta di un blocco specifico alla blockchain.

Hashcash (SHA-256) è la funzione Proof of Work utilizzata dal Bitcoin. La criptovaluta obbliga i miners a risolvere dei problemi matematici estremamente complessi e computazionalmente difficili per poter aggiungere blocchi alla blockchain. Tale funzione produce un tipo di dati molto specifici che vengono utilizzati per verificare che sia stata eseguita una notevole quantità di lavoro - da qui il termine Proof of Work, in italiano "prova di lavoro", ricordiamo che la difficolta di mining è gestita da alcuni parametri definiti dalla rete, come ad esempio il nonce, delimitato dallo spazio di manovra sotto al quale l'hash generato è reso valido, piu il valore limite è basso piu la difficolta aumenta.

In Pratica gli algoritmi di proof of work nel validare i blocchi fanno si che tra tutto il lavoro concorrente che viene eseguito in parallelo solo quello di un nodo (utente della rete) non è svolto invano, cio è sia un contro per lo spreco indiscriminato di corrente sia un pro per la sicurezza della rete e degli attacchi alla stessa, poichè la potenza di calcolo per la falsificazione sarabbe superiore a quella necessaria per il mining di un lecito.

1.3 proof of stake

Il Proof of Stake costituisce un metodo alternativo, un modo attraverso cui i nodi raggiungono un consenso. È stato proposto per la prima volta da un utente del forum Bitcointalk nel 2012 perché il PoW richiedeva troppa elettricità ed energia.

Il sistema di consenso Proof of Work come precedentemente detto è estremamente oneroso sotto un punto di vista computazionale e conseguentemente energetico. Il Proof of Stake, invece, utilizza molta meno energia ed è quindi molto più ecologico del Proof of Work.

Nel modello di consenso Proof of Stake, il numero di token di valuta digitale detenuti da ciascun utente, è una questione importante all'interno del sistema. Più grande è la partecipazione ("stake"), ovvero la quantità di token posseduti da un utente, maggiori sono le probabilità che non si stia violando il sistema. Più un nodo trae beneficio da una criptovaluta, più è probabile che questi si comporti in modo ottimale.

I blocchi della Proof of Stake, a differenza dei blocchi della Proof of Work, non vengono minati, ma coniati. I partecipanti che possiedono una partecipazione significativa nei sistemi Proof of Stake vengono selezionati su base pseudocasuale per coniare i blocchi e aggiungerli alla blockchain. Il processo di selezione pseudocasuale entra in funzione dopo che il sistema ha analizzato diversi fattori al fine di garantire che siano selezionati solo gli individui con una quota maggiore, ma anche altri con una stake inferiore.

IL Proof of Stake viene applicata generalmente alle criptovalute preminate, così da consentire all'utente di accedervi attraverso la partecipazione. Ciò significa che l'offerta complessiva delle criptovalute Proof of Stake viene fissata sin dall'inizio e che non vi è alcun premio per la creazione dei blocchi, come avviene invece nella Proof of Work. L'unico incentivo per i coiner in questo sistema è rappresentato dalle commissioni di transazione associate allo specifico blocco coniato.

Attualmente si stanno sviluppando diverse versioni di un algoritmo chiamato **Casper** per la migrazione della blockchain ethereum da un modello di proof of work ad un modello di proof of stake, tale passaggio dalla ethereum 1.0 alla ethereum 2.0 è stato nominato "Serenity", è attualmente non certa la data di release l'ultima data fissata era per genneaio 2020 che è stata rinviata a data da destinarsi.

1.4 La base: I contratti

Prima di parlare di smart contract sarebbe giusto fare un accenno a cosa sono i contratti da un punto di vista giuridico. Un contratto è un accordo tra due o più parti per estinguere un rapporto giuridico partimoniale. Un contratto inoltre per essere non nullo, ovvero valido, deve avere bene definiti tre piu uno requisiti:

- L'**Accordo** delle parti, le parti devono aver, in modo lecito e senza sotterfugi o atti illeciti, dato il proprio consenso ai termini del contratto stesso, pena annullabilita e/o nullità.

- La **Causa** del contratto (la funzione economico-sociale dello stesso)
- Deve essere ben definito e economicamente tangibile l'**Oggetto** del contratto
- La **Forma**, solo se richiesta dalla legge sotto pena di nullità

Ricordiamo che un contratto per ritenersi concluso deve vedere le parti consapovoli delle tre caratteristiche sopra elencate e le quali devono essere a conoscenze del consenso delle altre.

Da un contratto poi ne derivano le obbligazioni dello stesso che legano le parti in base ad un obbligo (prestazione generalmente) appunto che una parte deve all'altra, le parti assumeranno i ruoli di debitore (chi effetua la prestazione di denaro o professionale) e il creditore colui che la subisce.

1.5 Cosa è uno smart contract

Gli smart contract sono effettivamente dei contratti digitali che comprendono codice eseguibile che rappresenta le clausole del contratto stesso.

La blockchain che attualmente mette a disposizione un servizio di smart-contract e la loro creazione tramite un linguaggio di scripting turing complete è proprio **Ethereum**. Uno smart contract da un punto di vista tecnico, si compone di tre parti:

- il codice, che diventa l'espressione della logica contrattuale;
- gli eventi, che il programma acquisisce e che vanno a interagire con il contratto;
- gli effetti determinati dagli eventi.

Dal punto di vista giuridico uno smart contract è la "trasposizione" in codice di programmazione di un contratto, il quale gestisce in modo totalmente autonomo e sicuro (basandosi sulla sicurezza della blockchain) il flusso protocollare del contratto le relative clausole e gli obblighi "monetari" di entrambe le parte, facendo rimanere il tutto nero su bianco all'interno della blockchain in modo permanente, e emanando degli eventi che possono essere processati da applicativi esterni (tramite l'utilizzo di framework come web3.js, per magari allacciare applicativi ai nostri contratti) e dalla

blockchain stessa. Uno smart contract inoltre riesce a codificare ed implementare anche eventuali **termini** e **Condizioni** di un contratto oltre a tutte proprieta strutturali sopracitate. Uno smart contract viene associato alla tipologia dei **Contratti Cibernetici**, nel quale la macchina è astratta come l'alter ego dei contraenti. Il concetto dietro agli SmartContract è affine a quello che è presente dietro qualsiasi criptovaluta libera attualmente nel "mercato", ovvero la decentralizzazione (in tal caso, assenza di societa contrattuali), la sicurezza(sistemi di crittografia) e la trasparenza(contenuto dei blocchi in chiaro).

2 Ethereum

Di seguito un accenno a quello che è la piattaforma Ethereum. Tratteremo degli account, delle transazioni e Del carburante dei nostri contratti, il Gas. Tutto cio che viene eseguito all'interno della blockchain Ethereum viene eseguito tramite il middleware **EVM** (Ethereum Virtual Machine) cio che la EVM legge è del bytecode, solitamete il codice per programmare sopra questa macchina è **Solidity** lo stesso utilizzato per il progetto presentato alla fine. La EVM è completamente isolata, cio significa che il codice eseguito al suo interno non ha accesso alla rete al filesystem o altri processi.

2.1 Account

In Ethereum con il termine account contrariamente a come potremmo immaginare non ci riferiamo solamente agli account degli "utenti" della blockchain i quali sono definiti come come **External Accounts**; ma ci si riferisce anche ai **contracts account** che sono i contratti stessi. Questi due tipi di account sono trattati nello stesso modo dalla Ethereum Virtual Machine (EVM). Ogni account ha una mappa chiave valore (vedere mapping nell'esempio finale) che mappa parole da 256-bit a parole da 256-bit, questa struttura dati viene definita **Storage** dell'account. Infine ogni account ha un **Balance** (in wei, una sottovaluta degli ether) che puo essere modificato spedendo delle transazioni che contengono Ether

2.2 Transazioni

Una transazione è un messaggio che viene spedito da un account ad un altro e puo includere un payload (bits contenenti il messaggio) o Ether. Se l'account contiene codice (ovvero gli external contract) il payload viene passato come parametro alla procedura chiamata.

Se il target account non è settato la transazione crea un nuovo contratto, il payload di tale transazione è letto come bytecode EVM e eseguito. L'output di tale procedura è permanentemente mantenuto come codice del contratto. cio significa che per creare un contratto non è necessario mandare il codice tramite transazione, ma invece viene rimandato dalla EVM al chiamante e finche il costruttore non ha finito di configurare il codice del contratto quest'ultimo non puo essere eseguito.

2.3 Gas

Ogni transazione è caricata con un determinato ammontare di **Gas**, il cui obiettivo è quello di limitare il carico di lavoro necessario all'esecuzione della transazione e pagare per l'esecuzione. Finche la EVM esegue la transazione il gas viene gradualmente prelevato a seconda di specifiche regole. il valore del gas è configurato dal creatore della transazione che ha da pagare **gas price* gas** all'account ricevente. se del gas viene lasciato in eccesso alla fine della transazione, è rimborsato al creatore della transazione. Se il gas ad un certo punto dell'esecuzione della transazione assume un valore negativo viene sollevata un eccezione di tipo **OutOfGas-Exception** la quale reversa tutte le modifiche effetuate dall'esecuzione riportando il tutto allo stato prima dell'inizio della transazione stessa.

3 Il Progetto

3.1 Rider Contract

A titolo di esempio e per un personale esercizio pratico ho provato a scrivere un semplice smart contract utilizzando il linguaggio solidity sopra citato. In tale esempio ho pensato ad un caso d'uso abbastanza attuale, quello di poter creare uno smart contract per il lavoro del rider, ovvero dei fattorini che attualmente sono molto comuni nelle società intermediarie che fanno spedire nelle abitazioni degli acquirenti oggetti acquistati in alcune piccole società

che vendono al dettaglio; tramite questo tipo di contratto qualsiasi gestore di tali esercizi commerciali può assoldare un rider in qualsiasi momento stipulando un contratto virtuale che lega il rider ad eseguire degli ordini (effettuati dai clienti a favore delle societa al dettaglio), i quali sono mantenuti nella memoria del contratto (storage) ed il rider rimane in "obbligo" finche ci sono ordini da fare, inoltre l'ammontare di ether di ogni transazione rilasciata dagli acquirenti viene trasferita al owner e una percentuale definita dallo stesso (vedere costruttore del contratto, si presume che sia l'owner a crearlo) viene lasciata al rider. Inoltre solamente gli acquirenti i quali indirizzi di account mappano gli ordini possono cancellare o confermare la consegna di quest'ultimo. Sono inoltre definiti degli eventi all'interno del codice del contratto che vengono mandati alla blockchain tramite l'esecuzione di alcune procedure e che possono essere gestite come annunciato in un paragrafo precedente tramite applicativi esterni e quindi si ha la possibilita di modulare dei veri e propri servizi per tali aziende fondati sui contratti digitali. Nella stesura di questo rudimentale contratto non sono state prese in considerazione parecchie di quelle che possono essere delle sottocasistiche reali, ma come annunciato ciò voleva solamente essere un esempio delle potenzialià degli smart contract e di quanto possono rendere piu snella e moderna la gestione di modalità contrattualistiche come quella presentata.

3.2 Il Codice

```
1  pragma solidity >=0.4.0 <0.7.0;
2
3  contract RiderContract {
4
5      address payable public owner;
6      address payable public rider;
7      uint private percentage_to_rider;
8
9      uint active_orders;
10     mapping(address => Order) orders;
11
12     struct Order {
13         uint id;
14         uint amount;
15         bool active;
16     }
17
```

```
18      event Ordered(address caller, address rider, uint id,
            uint amount);
19      event Delivered(address caller, address rider, uint id);
20      event Dismissed(address caller, address rider, uint id,
            uint amount);
21
22      constructor(address payable _rider, uint
            _percentage_to_rider) public {
23          rider = _rider;
24          owner = msg.sender;
25          percentage_to_rider = _percentage_to_rider;
26          active_orders = 0;
27      }
28
29      function prenotareOrdine(uint _id, uint _amount) public
            payable {
30          require(msg.value >= _amount, "pagamento
                insufficiente");
31          orders[msg.sender].id = _id;
32          orders[msg.sender].amount = _amount;
33          orders[msg.sender].active = true;
34          active_orders += 1;
35
36          emit Ordered(msg.sender, rider, _id, _amount);
37      }
38
39      function ordineArrivato() public payable {
40          Order storage order_delivered = orders[msg.sender];
41          require(order_delivered.active == true, "Non risulta
                Nessun ordine a suo favore");
42          uint amount_to_rider = order_delivered.amount *
                percentage_to_rider / 100;
43          uint amount_to_owner = order_delivered.amount -
                amount_to_rider;
44          rider.transfer(amount_to_rider);
45          owner.transfer(amount_to_owner);
46          active_orders -= 1;
47          orders[msg.sender].active = false;
48
49          emit Delivered(msg.sender, rider, order_delivered.id)
                ;
50      }
51
52      function cancellaOrdine() public payable {
53          Order storage order_dismissed = orders[msg.sender];
```

```
54          require(order_dismissed.active == true, "Non risulta
                Nessun ordine a suo favore");
55          msg.sender.transfer(order_dismissed.amount);
56          active_orders -= 1;
57
58          orders[msg.sender].active = false;
59
60          emit Dismissed(msg.sender, rider, order_dismissed.id,
                order_dismissed.amount);
61      }
62
63      function rilascioRider() public payable {
64          require(msg.sender == rider || msg.sender == owner, "
                per terminare il contratto devi essere il rider o
                l'owner");
65          require(active_orders == 0, "devi ancora portare a
                termine degli ordini");
66
67          selfdestruct(owner); // eventuali eth in eccesso
                vengono rilasciati all'owner
68      }
69  }
```

Listing 1: Runner Contract SOL