UNIVERSITÀ DEGLI STUDI ROMA TRE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Tesi Di Laurea

Studio dell'architettura payment channel per blockchain basate su smart contract con linguaggi turing completi

Laureando

Federico Ginosa

Matricola 457026

Relatore

Alberto Paoluzzi

Correlatore

Federico Spini

Anno Accademico 2017-2018

Questa è la dedica

Indice

Elenco delle figure

Ringraziamenti

Grazie a tutti.

Capitolo 1

Blockchain

La blockchain è una struttura dati autenticata e distribuita. Generalmente una struttura dati permette due tipologie di operazioni:

- interrogazioni
- aggiornamenti

In un'ADS le interrogazioni forniscono assieme alla risposta una prova verificabile dell'integrità della soluzione fornita.

Cosa significa questo? Ipotizziamo che Alice possegga una lista L:

```
L = { L1 -> L2 -> L3 -> L4 }
```

Partendo da questa lista, si costruisce un albero di questo tipo: per ciascun elemento della lista calcoliamo l'hash. Poi per ciascuna coppia di hash calcoliamo l'hash della concatenazione, fino ad arrivare alla radice di quest'albero.

Alice possiede la lista L e l'intero albero, Bob invece possiede esclusivamente il nodo radice.

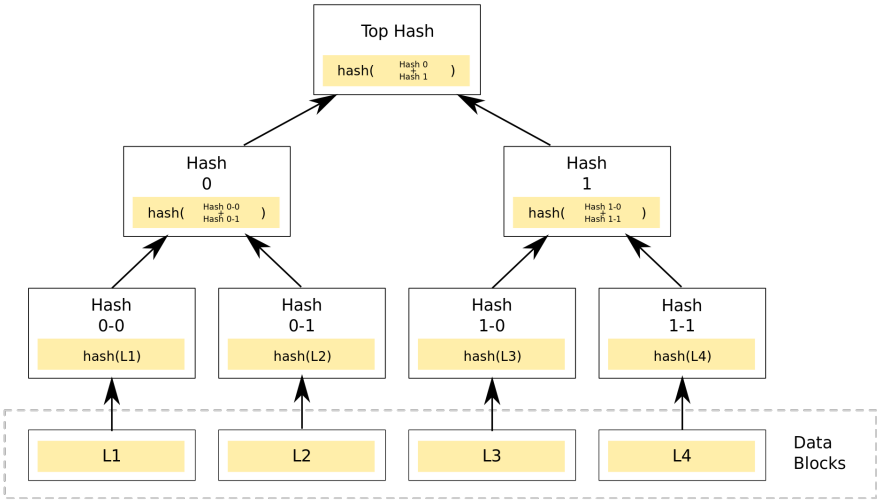

Figura 1.1: Merkle tree

Ipotizziamo che Bob voglia sapere se L3 sia contenuto in L; farà una query di questo tipo:

```
query: L contains L3
```

Alice riceve questa query e risponde così:

```
result: true
proof: Hash(0), Hash(1-1), Hash(1)
```

Bob ottiene una risposta e oltre a questo ottiene una prova di essa. Infatti concatenando ed effettuando l'hash della prova fornita, coerentemente alla costruzione dell'albero, può facilmente ricostruire la root dell'albero, verificando che coincida con quanto posseduto.

Questa struttura dati autenticata prende il nome di Merkle Tree ed è uno degli elementi fondanti della blockchain. In particolare la blockchain è per l'appunto una catena di blocchi con un ordine definito. Ciascun blocco contiene un

certo numero di transazioni, le quali a loro volta hanno un certo ordine. Per ciascun blocco viene costruito uno di questi alberi.

Che utilità ha tutto questo? Ipotizziamo che esista una blockchain B con 500000 blocchi e che ciascun blocco abbia un peso di 1MB; il peso dell'intera blockchain sarà di 500GB.

Bob vuole sapere se tx_n sia contenuta nel blocco B_m, ma non ha a disposizione un nodo di calcolo con 500GB di archivio, cosa può fare? Semplicemente memorizzerà solo l'hash root di ciascun blocco. Ipotizzando che un hash abbia un peso di 1MB e che il numero di blocchi sia 500000, lo spazio d'archiviazione necessario a Bob è minore di un 1MB.

Bob avendo a disposizione l'hash root di ciascun blocco e la possibilità di contattare un fullnode[1], può verificare che una certa transazione sia contenuta in un certo blocco, interrogando quest'ultimo. Il full node fornirà oltre alla risposta, la prova della risposta, il che garantisce a Bob che il risultato sia integro e che non sia stato manomesso da una terza parte.

Per quanto riguarda gli aggiornamenti, generalmente in una struttura dati essi devono rispettare determinate convenzioni. Come detto precedentemente la blockchain è una struttura dati distribuita; questo significa che la struttura dati in questione non è memorizzata in un unico nodo, ma in più nodi che prendono il nome di peer. La distribuzione può essere effettuata con due diversi approcci:

- replicazione: ciascun peer possiede una copia della stessa struttura dati
- sharding: ciascun peer possiede una porzione della struttura dati
- replicazione/sharding: il mix delle due tecniche precedenti

[1] nodo di calcolo che possiede l'intera blockchain

Attualmente la tecnologia blockchain si basa sul primo approccio, ovvero la replicazione. Questo significa che esistono N nodi e che ciascun nodo possiede una copia della stessa struttura dati.

Come detto gli aggiornamenti di una struttura dati devono seguire determinate convenzioni, una di queste convenzione è la seguente: una volta inserito un blocco, questo blocco non può essere modificato.

Come è possibile garantire una condizione simile in un sistema distribuito? Bada bene, questo problema non è risolto dal merkle tree, perché nel merkle tree si da per scontato che Bob abbia quanto meno i root hash di ciascun blocco. In questo caso Bob deve ancora ottenere i root hash, quindi l'uso esclusivo del merkle tree non ci garantisce molto in tal senso.

In altre parole ciò di cui abbiamo bisogno è un consenso della rete, ovvero, abbiamo bisogno che la rete possa confermare a Bob che la struttura dati da lui posseduta non sia stata manomessa.

Un possibile approccio è il seguente: Bob sceglie un nodo casuale, scarica da esso l'intera blockchain e ne effettua l'hash. Poi chiede a ciascun nodo del sistema di votare per la validità della struttura dati. Ciascun nodo vota fornendo a Bob l'hash dello della struttura dati nello stato corrente. Se il risultato coincide con quanto posseduto da Bob per un valore maggiore al 50% Bob prende per buona la struttura dati posseduta, altrimenti no.

Qui sorge il primo problema di questa soluzione. Come facciamo ad essere certi in un sistema distribuito che un nodo fornisca un unico voto? Ovvero come facciamo ad evitare che un nodo possa votare più volte?

Estremizzando questo ragionamento, sia N il numero di nodi, un nodo x potrebbe votare $N/2 + 1$ volte, inducendo Bob a credere che una struttura dati manomessa in realtà non lo sia.

Questo attacco prende il nome di sybil attack o pseudospoofing e l'approccio di voto democratico basato sull'entità del nodo non è immune ad esso.

Un approccio alternativo è basato sulla proof of work. L'idea è questa: abbiamo detto di dover garantire che la blockchain fornita a Bob non sia stata manomessa, ovvero che non sia possibile fornire a Bob una blockchain con blocchi precedente modificati. Ciò che possiamo fare è rendere difficile l'aggiunta di nuovi blocchi alla catena.

In particolare per aggiungere un nuovo blocco, occorrerà fornire una "prova di lavoro", ovvero la prova che per inserire questo blocco sia stato effettuato del lavoro; questa prova è rappresentata da un nonce. Infatti per poter aggiungere un blocco alla catena, occorrerà fornire un nonce, che se concatenato alle transazioni e successivamente hashato, deve restituire un hash che abbia un numero D di zero.

Questa operazione prende il nome di mining. D rappresenta la difficoltà corrente di mining e può variare, in particolare il valore di D viene calibrato sulla base del tempo impiegato dalla rete per minare un blocco.

Di seguito lo pseudocodice di quello che potrebbe essere un algoritmo di proof of work.

```
def proofOfWork (block, D):
    nonce = 0
    do
        nonce++
        result = hash(nonce | block)
    while (result.substring(0, D) == ('0' * D))
```

Andiamo a vedere nel dettaglio il contenuto di block:

- nonce: come già detto è il valore che permette di variare il risultato dell'hash per far si che rispetti una determinata proprietà

- roothash: il nodo root del merkle tree costruito sulla base delle transazioni che si vuole aggiungere al blocco corrente
- prev_hash: l'hash del precedente blocco

Perché tutto questo dovrebbe risolvere il problema della manomissione di blocchi precedenti? Di seguito in figura è descritto lo stato corrente della blockchain B:

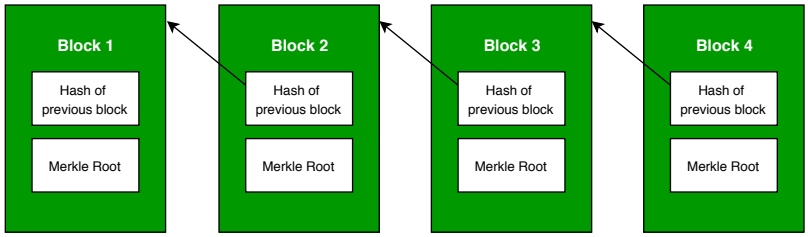

Figura 1.2: Blockchain con blocchi non manomessi

Immaginiamo ora che un nodo malevolo voglia manomettere il blocco numero due della catena:

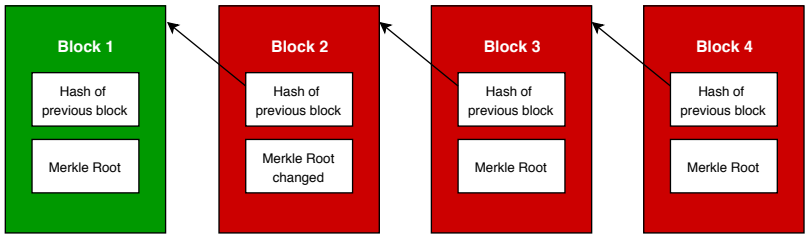

Figura 1.3: Blockchain con un blocco manomesso minato

Come è possibile vedere dalla figura, manomettere il blocco numero due, implica invalidare i blocchi 2, 3 e 4; infatti variando il valore del merkel root, varia anche il valore dell'hash, che con tutta probabilità non avrà più i primi D caratteri uguali a zero. Questo significa che il nodo malevolo dovrà calcolare un nuovo nonce per validare il blocco numero due.

Immaginiamo che il nodo malevolo si imbarchi in questa impresa, il risultato finale sarà questo:

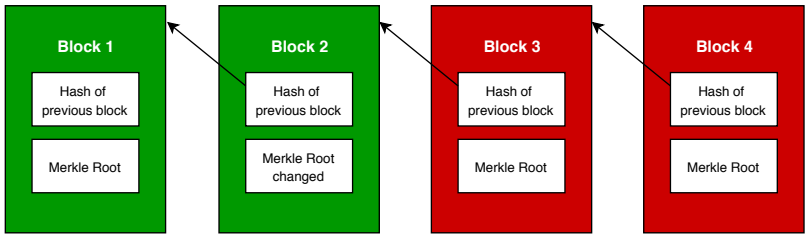

Figura 1.4: Blockchain con blocchi manomessi

Ovvero avrà validato il blocco numero 2, ma i blocchi numero 3 e 4 saranno ancora invalidi, questo perché essi codificano al loro interno l'hash del blocco precedente, che a questo punto sarà chiaramente diverso.

Ciò che dovrà fare dunque il nodo malevolo è ricalcolare un nonce corretto per ciascun blocco successivo a quello manomesso. Questa operazione porterà via un gran quantitativo di tempo, durante il quale la rete avrà calcolato nuovi blocchi, di cui il nodo malevolo dovrà calcolare un nuovo nonce; è chiaro che in questa corsa contro il tempo, si ha qualche possibilità di vincere solo se si è in possesso di un gran quantitativo di potenza computazionale.

In particolare sarà possibile riuscire a disfare un blocco confermato e ricalcolare il nonce dei successivi, solo se si possiede più del 50% della potenza computazionale dell'intera rete, cosa decisamente più complessa e onerosa rispetto a generare una manciata di indirizzi ip.

Portiamo all'estremo questo ragionamento e ipotizziamo che esista un nodo con potenza computazionale maggiore del 50%, come possiamo evitare che il nodo disfi anche in questa situazione blocchi precedenti?

L'idea qui è di assegnare a ogni miner capace di calcolare il nonce correto dell'ultimo blocco una ricompensa. A questo punto, un nodo con tutta questa potenza computazionale, preferirà minare e ottenere i reward di ciascun blocco

invece che manomettere blocchi passati, minando in questo modo la bontà del network del quale è padrone indiscusso[1].

Capitolo 2

Payment channel

Un payment channel è un canale virtuale che permette a due entità di scambiare valore facendo un uso limitato della blockchain.

In particolare ad eccezione di alcune operazioni atte a mettere in piedi questo canale, tutte le altre avvengono off-chain, ovvero senza far uso della blockchain.

Un payment channel eredita tutte le proprietà di sicurezza della blockchain e inoltre garantisce:

- maggiore riservatezza: mentre le operazioni effettuate su blockchain sono pubbliche, quelle off-chain non lo sono.
- maggiore scalabilità: la blockchain per sua natura permette di effettuare un numero limitato di operazioni al secondo
- pagamenti istantanei: mentre nella blockchain le transazioni devono essere minate e successivamente confermate, in un payment channel le transazioni sono pressocché istantantee.

In questo lavoro è stato implementato un payment channel basato su smart contract. Di seguito vengono analizzati gli approcci presi in considerazione e

quelli infine adottati.

Deploy dello smart contract

La prima azione che occorre effettuare per instaurare un payment channel tra due entità consiste nel deploy del relativo smart contract su mainet. Questa operazione permetterà di ottenere il relativo indirizzo del contatto, che nelle successive fasi verrà adottato per richiamare le varie operazioni on-chain che si intende adottare. Il payment channel può trovarsi in uno di N stati; in fase di deploy lo stato è $INIT$.

Apertura del canale

La seconda operazione che è necessario effettuare consiste nell'apertura del canale. Alice apre il canale e blocca un quantitativo arbitrario di fondi all'interno dello smart contract; questo valore rappresenterà il suo bilancio iniziale nel canale corrente. Oltre a bloccare i fondi vengono fornite le informazioni necessarie all'apertura del canale, in particolare:

- **ethereumAddressB**: ovvero l'indirizzo ethereum della controparte, (E.G Bob).
- **host**: un host associato all'utente corrente, successivamente questo aspetto verrà chiarito.
- **gp**: ovvero il grace period, anche questo aspetto verrà chiarito successivamente.

L'apertura del canale può essere effettuata esclusivamente quando il payment channel si trova in stato di $INIT$, in qualunque altro caso verrà sollevata un'eccezione.

Terminata l'esecuzione della procedura di open, lo stato del payment channel passerà da $INIT$ a $OPENED$.

Join del canale

La terza e ultima operazione necessaria a instaurare un payment channel tra due entità consiste nel join del canale. Chiaramente questa operazione può essere effettuata dall'utente con public address uguale a **ethereumAddressB** espresso da Alice in fase di apertura. Inoltre questa operazione può essere effettuata esclusivamente quando lo stato del canale è $OPENED$.

Anche in questo caso l'utente richiamando la procedura bloccherà un numero arbitrario di fondi, che rappresenterà il bilancio iniziale dell'utente che ha effettuato il join, ovvero Bob. Inoltre Bob in fase di join fornirà il proprio **host** (anche qui, l'utilità di questa informazione verrà espressa successivamente). Una volta eseguita questa procedura, lo stato del canale passerà da $OPENED$ ad $ESTABLISHED$.

Modello propose-accept

Quelle descritte fino a questo punto sono delle azioni che devono essere svolte in catena, necessarie a mettere in piedi un canale. Da questo punto in poi Alice e Bob possono scambiarsi dei pagamenti istantanei senza toccare la catena.

L'approccio adottato è detto di propose-accept e si basa su due endpoint http pubblici esposti.

Questi endpoint sono rispettivamente disponibili sotto l'host dichiarato in fase di apertura del canale.

- **/propose**: questo endpoint permette a uno delle due entità di richiedere alla controparte di concordare una nuova propose Una propose è un'entità che propose di spostare una certa quantità di valore dall'entità richiedente alla controparte
- **/accept**: questo endpoint permette a una delle due entità di accettare una propose precedentemente ricevuta sull'endpoint /propose

Struttura di una propose

Una propose, ovvero una proposta di pagamento contiene essenzialmente quattro informazioni:

- **seq**: è il numero di sequenza, anche detto nonce; esso è un numero progressivo che parte da zero
- **contract_address**: l'indirizzo di riferimento del contratto che si sta utilizzando
- **balance_a**: il bilancio che si sta concordando per chi ha aperto il canale (E.G. Alice)
- **balance_b**: il bilancio che si sta concordando per chi ha fatto join del canale (E.G. Bob)

Poniamoci in una situazione d'esempio. Alice deploya e apre il canale, bloccando 1 eth. Bob effettua il join del canale e anche lui blocca 1 eth. Attualmente il bilancio di Alice e Bob è per entrambi di 1 eth e i fondi complessivi bloccati nel payment channel sono pari a 2 eth.

Ora Alice vuole inviare 0.2 eth a Bob; per farlo:

1. Costruisce un'opportuna propose

2. Effettua l'hash di quest'ultima
3. Firma l'hash della propose
4. Invia hash firmato e i valori in chiaro della propose

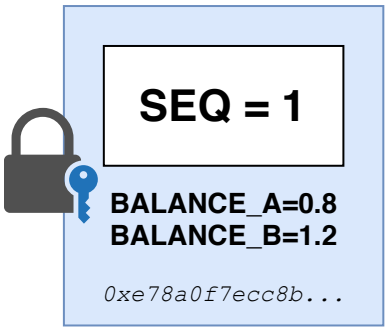

Figura 2.1: Propose firmata da Alice

Bob riceve la propose e:

1. Verifica di essere d'accordo con il contenuto
2. Verifica che la firma di Alice sia valida
3. Se decide di accettare la propose, in maniera speculare ad Alice effettua l'hash del contenuto della propose e lo firma con la sua di chiave privata
4. Infine invia la propose controfirmata ad Alice, tramite il suo endpoint pubblico /accept

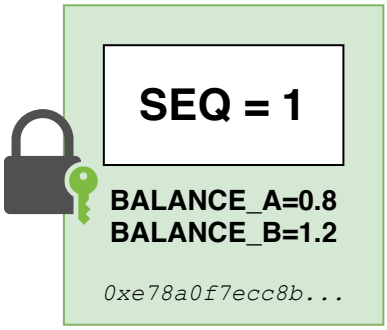

Figura 2.2: Propose firmata da Bob

A questo punto il bilancio off-chain di Alice e Bob è aggiornato, in particolare è rispettivamente di 0.8 ether per Alice e 1.2 ethere per Bob.

Seguendo lo stesso principio Alice e Bob possono concordare un numero arbitrario di propose (quindi di pagamenti), avendo sempre l'accortezza di aumentare il numero di sequenza; per esempio una successiva propose valida sarà:

```
seq=2; contract_address=0xe78...; a=0.9; b=1.1
```

In questa propose Bob ha inviato 0.1 eth ad Alice.

Chiudere un canale

Come detto con questo modello di propose e accept Alice e Bob possono concordare un numero arbitrario di pagamenti. Questi pagamenti sono pressochè istantanei e aggiornano il bilancio off-chain delle controparti. Ma cosa significa aggiornare il bilancio off-chain? Quando Alice invia una propose firmata e Bob l'accetta, non avviene un vero e proprio pagamento sulla rete ethereum, ma le due parti stipolano un accordo che firmano e che non possono rescindere. In fase di chiusura del contratto Alice o Bob potranno presentare l'ultima propose concordata e ritirare quanto gli spetta.

Entrambe le parti possono chiudere il canale in qualsiasi momento (basta che il canale si trovi in stato $ESTABLISHED$).

Vediamo in particolare come avviene la chiusura. Ipotizziamo che Alice voglia chiudere il canale e ritirare dunque 0.9 ether. Alice eseguirà la procedura close dello smart contract. I parametri formali della procedura close sono:

- seq
- balanceA

- balanceB
- sig: propose corrente firmata da Bob

Il contratto verificherà la propose firmata da Bob e presentata da Alice e in caso positivo aggiornerà lo stato del canale in $CLOSED$.

Quando Alice presenta una propose valida e porta lo stato del canale in $CLOSED$ non ritira ancora i suoi ether, ma inizializza un timer.

Questo timer ha la durata di gracePeriod, la variabile presentata in fase di apertura del canale.

Ritirare denaro da un canale

Come detto precedentemente la close di un payment channel porta il canale in uno stato di $CLOSED$ e inizializza un timer di durata pari a gracePeriod. Quando un canale si trova in $CLOSED$ e il timer è scaduto, entrambe le parti possono ritirare quanto riportato nella propose riportata in chiusura. Il ritiro del denaro avviene mediante una procedura denominata withdraw.

Argue di una propose

Prima di andare in chiusura lo stato corrente del payment channel è quello illustrato in figura.

Ovvero Alice e Bob hanno concordato rispettivamente 2 propose. Nella prima propose Alice ha inviato un pagamento di 0.2 ether a Bob e nella seconda Bob ha inviato un pagamento di 0.1 ether ad Alice.

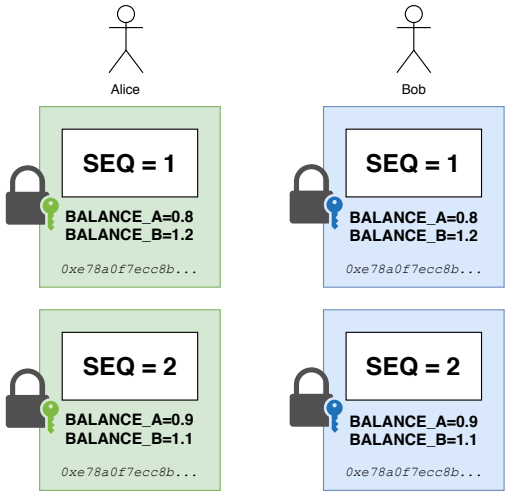

Figura 2.3: Stato corrente del payment channel

A questo punto se Bob volesse chiudere il canale e fosse onesto, dovrebbe presentare in chiusura la propose con seq=2.

Ipotizziamo però che Bob sia malevolo e presenti in chiusura la propose con seq=1; dal suo punto di vista questa propose è sicuramente più vantaggiosa, in quanto gli vedrebbe accreditati non 0.1 ether in più.

Per come è congegnata la close dello smart contract presentato nessuno impedisce a Bob di presentare questa propose e portare lo stato del canale in $CLOSED$.

Questo chiaramente non va bene, l'unica propose valida è l'ultima proposta. A questo punto si propose un meccanismo di arguing per contrastare questa tipologia di attacco.

In particolare, quando una controparte presenta una propose in chiusura, essa produce un evento pubblico, che indica il sequence number della propose. Quando Alice riceve un evento di close relativo a un suo canale di interesse, deve controllare il sequence number della propose presentata e verificare

che coincida con il sequence number dell'ultima propose concordata. In caso questo non fosse vero, Alice deve presentare tramite una procedura denominata **argue** dello smart contract l'ultima propose valida.

I parametri della procedura argue sono gli stessi della procedura close, il comportamento chiaramente è diverso. La procedura argue infatti verifica che la propose presentata sia valida e che il sequence number sia maggiore rispetto a quello dell'ultima propose presentata.

Nel caso in cui questo fosse vero il contratto punisce la controparte malevola inviando tutti i fondi del payment channel ad Alice.

La procedura di argue può essere effettuata solo quando il canale è in $CLOSED$ e il timer di durata pari al gracePeriod non è ancora scaduto.

Questo meccanismo permette di essere certi che le due parti siano oneste e presentino sempre e solo l'ultima propose valida.

[1] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).