Università degli Studi di Torino

Dipartimento di Informatica

Corso di Laurea in Informatica

Anno Accademico 2020/2021

Progettazione e realizzazione di una applicazione web per la gestione dei meccanismi di elezione dei nodi sealer all'interno di blockchain Ethereum Proof-of-Authority

Relatore

Claudio Schifanella

Tutor Aziendale

Katia Manfrin

Candidato

Andrea Mattone

Abstract

Durante il tirocinio presso la Consosft Sistemi S.p.A. ho preso parte al progetto PININ. Un progetto di tracciabilità agroalimentare che si appoggia alla tecnologia blockchain. Più specificatamente mi sono occupato della modalità di assegnazione di permessi all'interno di una rete. Come verrà spiegato approfonditamente nella tesi, all'interno di una blockchain di tipo Proof of Authority (ovvero la blockchain utilizzata all'interno del progetto PININ) è necessario elargire dei permessi ai vari nodi del network. Questi permessi all'interno di una blockchain Proof of Authority vengono elargiti tramite delle votazioni da parte di chi, all'interno del network, ha l'autorità di effettuarle. Il mio lavoro di tesi è basato proprio su questo, ovvero sviluppare un applicativo che ci fornisse un modo comodo di effettuare le votazioni per poter elargire o meno a un nuovo nodo della rete i permessi per poter essere riconosciuto come nodo Sealer all'interno della rete.

Ho lavorato prevalentemente con tecnologie front end, quali React.js e Javascript per lo sviluppo dell'applicativo denominato "andromeda". Inoltre ho utilizzato degli smart contract ethereum scritti con il linguaggio di programmazione Solidity per interagire direttamente con la blockchain. Il tutto è stato testato su una blockchain privata creata tramite Geth, ovvero una delle tre implementazioni originali del protocollo ethereum.

Indice

Codice

Figure

1 Introduzione

1.1 L'azienda

Consoft Sistemi S.p.A. è un'azienda italiana presente sul mercato dell'information communication Technology dal 1986 con sedi a Torino, Milano, Genova, Roma e Tunisi, con oltre 400 dipendenti e un fatturato superiore ai 29 milioni di Euro.

Consoft Sistemi ha focalizzato la propria offerta su diverse aree tematiche nell'ambito delle quali è in grado di realizzare soluzioni end-to-end per i propri clienti attraverso attività di consulenza tecnologica e metodologica, formazione, system integration ed erogazione di servizi in insourcing/outsourcing. Accanto alla capogruppo Consoft Sistemi sono attive altre quattro società: CS InIT, specializzata nello scouting di soluzioni software innovative e nella loro distribuzione sul mercato italiano, Consoft Consulting focalizzata su temi specifici della pubblica amministrazione, Consoft Sistemi MEA e CA Soft Consulting per espandere l'offerta della capogruppo, nel mercato nord-africano e medio-orientale.

Figura 1: Consoft Sistemi

1.2 Il progetto - PININ

PININ - PIemuNt chèINa e' un progetto nell'ambito agroalimentare, più nello specifico in ambito tracciabilità, rintracciabilità e autenticazione dei prodotti agroalimentari. Ha come obiettivo quello d'incrementare la qualità dei prodotti agroalimentari piemontesi introducendo tecnologie per la tracciabilità e l'autenticazione dei prodotti agroalimentari e per la protezione dei diritti di proprietà intellettuale dei marchi agroalimentari piemontesi di qualità tramite l'individuazione di falsi e truffe. Le tecnologie principalmente utilizzate all'interno del progetto sono la Blockchain, l'Intelligenza Artificiale e Big Data. Permettendo di creare un innovativo sistema di tracciatura dei prodotti alimentari lungo tutta la filiera. Il focus del progetto PININ mira a costruitre un'infrastruttura distribuita e decentralizzata basata sulla Blockchain che permetta una tracciabilità a livello di lotto e che sia scalabile lungo tutta la filiera. La blockchain permetterà inoltre la tracciabilità

dell'utilizzo dei fondi europei per l'allevamento per quel che riguarda il bestiame nei pascoli alpini, sempre per evitare truffe. In molti casi potrebbe essere sufficiente l'utilizzo di un database crittografato, in cui i dati sono firmati digitalmente dalla sorgente da cui arrivano ma uno dei vantaggi chiave dell'utilizzo delle tecnologie Blockchain è quello della disintermediazione, cioè la possibilità di non avere bisogno di un trusted third party che certifica la veridicità del dato.

2 La Blockchain

2.1 Introduzione

La blockchain è una struttura dati condivisa e immutabile, formata da una lista di record crescente. Questa tecnologia è basata sul concetto di *distributed ledger*, ovvero sistemi basati su un registro distribuito. Questi record sono chiamati "blocchi". I blocchi sono tra loro collegati in modo che ognuno di essi abbia il riferimento al blocco precedente, formando così una catena di blocchi, da cui deriva il termine *blockchain*.

2.2 Distributed Ledger

La blockchain è una forma di Distributed Ledger, ovvero un registro distribuito basato sul consenso di dati digitali replicati, condivisi e sincronizzati. E' fondato su una rete peer-to-peer di nodi interconnessi che comunicano tra loro, ognuno dei quali possiede una copia del registro e contribuisce al consenso. Essendo il Distributed Ledger distribuito permette l'eliminazione completa di un ente centrale che faccia da garante dei dati, spostando il focus sull'approvazione di ogni transazione da parte della maggior parte dei partecipanti del network. Questo inoltre permette di risolvere il problema del *single point of failure*. Ogni nodo della rete possiede una copia del Distributed Ledger, e quando si verifica una modifica, che viene approvata tramite il consenso collettivo, allora ogni nodo modifica la sua copia del Ledger aggiungendo la modifica e mantenendo così il registro sincronizzato. Il Database del sistema è quindi duplicato per tutti i vari partecipanti al network. La sicurezza del sistema è garantita attraverso l'ausilio della crittografia asimmetrica, che vedremo in seguito. [1]

2.3 Il Funzionamento della Blockchain

La Blockchain vuole fondamentalmente eliminare la fiducia verso enti terzi per il mantenimento di un registro. Questo significa che non si deve necessariamente avere un ente centrale che faccia da garante per la veridicità di una transazione. Il network intero permette di confermare una transazione accettabile o confutarne una invece inammissibile. Le transazioni contengono delle operazioni e viaggiano nel network tramite degli scambi di messaggi, che come vedremo in seguito potranno corrispondere a scambi di valuta e

addirittura al deploy di un vero e proprio programma, detto smart contract. L'obiettivo è di permettere a tutti i partecipanti del network di essere concordi sull'ordine cronologico delle transazioni. Per arrivare a una soluzione si utilizzano gli algoritmi di consenso, che analizzeremo nel dettaglio in seguito. In sostanza vengono sfruttati dei problemi difficili da risolvere, la cui correttezza è facilmente verificabile, possiamo ad esempio fare un'analogia con un sudoku, un problema che richiede molto più tempo per essere risolto rispetto alla verifica della sua correttezza. La risoluzione di un problema e la relativa conferma di veridicità da parte del network portano al consenso e all'accettazione di un blocco. E' importante scegliere un problema adeguato, in quanto se questo fosse troppo semplice la rete sarebbe vulnerabile, mentre se fosse troppo complesso la generazione di nuovi blocchi richiederebbe troppo tempo e bloccherebbe l'intero sistema. La soluzione che le prime versioni di blockchain adottano è quella basata sul concetto di Hash Chain, in sostanza i blocchi sono concatenati tramite degli hash (spiegati in seguito), ogni blocco ha un puntatore al blocco precedente utilizzando appunto gli hash come puntatori. La modifica di un'informazione all'interno di un blocco porterebbe a una modifica dell'hash e questo invaliderebbe la catena. Questo concetto si basa sulla nozione di consenso distribuito. I nodi della rete preposti alla risoluzione di questi complessi problemi matematici (nelle blockchain di prima generazione, in quanto in altre blockchain gli algoritmi di consenso sono differenti) necessari a garantire l'autenticità dei blocchi sono chiamati *nodi miner*. Se un nodo del network vuole effettuare una transazione, questa verrà inserita in un pool di transazioni. I miner pescano la transazione da questa coda e la verifica comporterà la ricezione di una commissione da parte del miner.

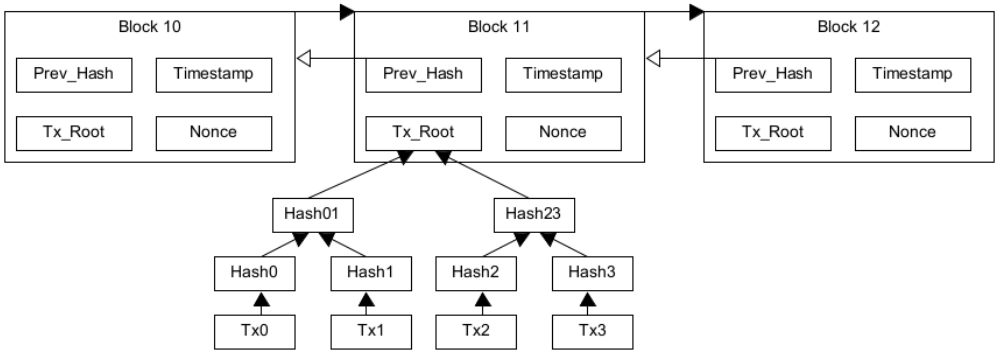

Figura 2: Blockchain

2.4 Il consenso distribuito

Il consenso distribuito sta a indicare un accordo comune da parte di tutti i nodi della rete peer-to-peer, così da rendere decentralizzato il network, in quanto nessun nodo è più importante degli altri. Le transazioni, quando vengono inviate, vengono inserite all'interno di un transaction pool, da cui vengono pescate per essere verificate e inserite nel successivo blocco della catena. Fino a quando la transazione non è inserita nella blockchain allora non è riconosciuta come valida della rete. Grazie a questa fiducia nel network si riesce a garantire che il network funzioni senza avere un ente terzo che funga da intermediario e faccia da garante. Per capire meglio a cosa serve il consenso distribuito possiamo fare un'analogia con un problema matematico teorizzato dai matematici Leslie Lamport, Marshall Paese e Robert Shostak nel 1982: "Il problema dei generali bizantini". La metafora e' basata sull'ipotesi di avere dei generali bizantini che devono assediare una citta'. I generali possono comunicare tra loro esclusivamente tramite scambio di messaggi portati da dei messaggeri. Tra i messaggeri ci possono pero' essere dei traditori che possono, in corso d'opera, modificare il messaggio. Tornando in contesto informatico il problema consiste nel trovare un accordo tra le componenti di un network, le cui parti comunicano tramite lo scambio di messaggi. Garantendo la fiducia tra le parti del network senza una figura centrale che faccia da garante. [2] Successivamente nella tesi verranno spiegati in dettaglio vari algoritmi di consenso.

2.5 Gli incentivi

Il motore che invoglia i partecipanti al network a confermare i blocchi sono gli incentivi. In sostanza per ogni blocco validato viene riconosciuta una ricompensa al miner (colui che ha generato il blocco). E analogamente questo accade con le fees delle transazioni.

2.6 Funzioni di Hash

La funzione di Hash è una funzione non invertibile che mappa un contenuto digitale di lunghezza arbitraria in un altro di lunghezza predefinita. Una funzione di Hash restituisce la posizione dell'elemento associato ad una chiave k. H(k) è il valore di Hash della chiave k e l'elemento di chiave k è in posizione h(k). L'Hash è spesso usato in crittografia e le sue caratteristiche principali sono:

- E' semplice calcolare l'Hash di un messaggio
- E' complesso trovare la sorgente che ha generato un dato hash (relativo alla non invertibilità delle funzioni di Hash)
- E' complesso modificare un messaggio senza modificare il relativo Hash (resistenza debole alle collisioni)
- E' difficile trovare due messaggi che abbiano lo stesso Hash (resistenza forte alle collisioni)

Gli Hash possono quindi essere usati come una sorta di "impronta digitale" con lunghezza prefissata di un messaggio.

2.7 Il double spending problem

Il double spending problem è un problema in ambito digitale che si basa sul concetto di unicità di una risorsa. Per capire meglio di cosa stiamo parlando prendiamo come esempio lo scambio di una valuta digitale. In un sistema economico ci deve essere un qualche tipo di Database o struttura dati in cui indicare la quantita' di valuta posseduta da ogni partecipante al sistema economico. Supponiamo di avere tre partecipanti a questo sistema economico chiamati A, B e C. Se A possiede 100 unita' di valuta e le invia a B, bisognera' impedire che possa nuovamente inviarle a C. Il double

spending problem e' quindi il risultato di spendere la stessa valuta con successo per piu' di una volta. Astraendo dall'esempio economico, se facciamo ad esempio riferimento al problema dei generali bizantini, il double spending sarebbe quello per cui un generale invia due messaggi differenti contemporaneamente. La blockchain protegge dal double spending problem verificando ogni transazione aggiunta alla catena di blocchi, per garantire che gli input per la transazione non siano gia' stati "spesi" in precedenza.

2.8 I wallet

Il wallet è dove vengono mantenute la coppia di chiavi pubblica e privata dell'utente. La chiave pubblica è utilizzata durante la ricezione di transazioni. I wallet ci permettono d'interagire con gli account legati a quella coppia di chiavi pubblica-privata. I wallet in sostanza ci forniscono gli strumenti necessari per interagire con la blockchain, ovvero generano le informazioni necessarie per inviare e ricevere le criptovalute tramite transazioni sulla blockchain. I wallet possono essere di due tipi

- **Hot** Gli hot wallet sono i wallet connessi in qualche modo alla rete internet. Sono molto semplici da creare generalmente e l'accesso alle criptovalute legate agli hot wallet sono velocemente reperibili.
- **Cold** I cold wallet sono quelli privi di connessione a internet, sono un qualunque mezzo fisico per archiviare le chiavi offline. Sono più sicuri rispetto agli hot wallet in quanto permettono di evitare attacchi informatici.

2.9 Dove e come viene mantenuta la blockchain

La blockchain deve essere mantenuta all'interno di una struttura informatica. Il blocco della blockchain è formato da un'intestazione e da varie transazioni e per rappresentarlo ci si appoggia sui Merkle Tree. Un Merkle Tree è una struttura dati basata sull'Hash, sostanzialmente è un albero in cui l'Hash di un nodo padre è calcolato tramite quello dei nodi figli. Si costruisce un albero prelevando l'Hash da ogni transazione e posizionandoli come nodi foglia di un albero binario, successivamente si costruisce l'albero a ritroso verso la radice calcolando i valori degli hash padri fino alla radice. Per effettuare una verifica di una transazione è semplicemente necessario calcolare l'hash della transazione nel punto in cui la transazione dovrebbe essere, e poi utilizzando

gli altri hash risaliamo l'albero calcolando gli hash dei vari nodi, se arrivati al nodo radice ho ottenuto l'hash corretto allora la transazione è valida.

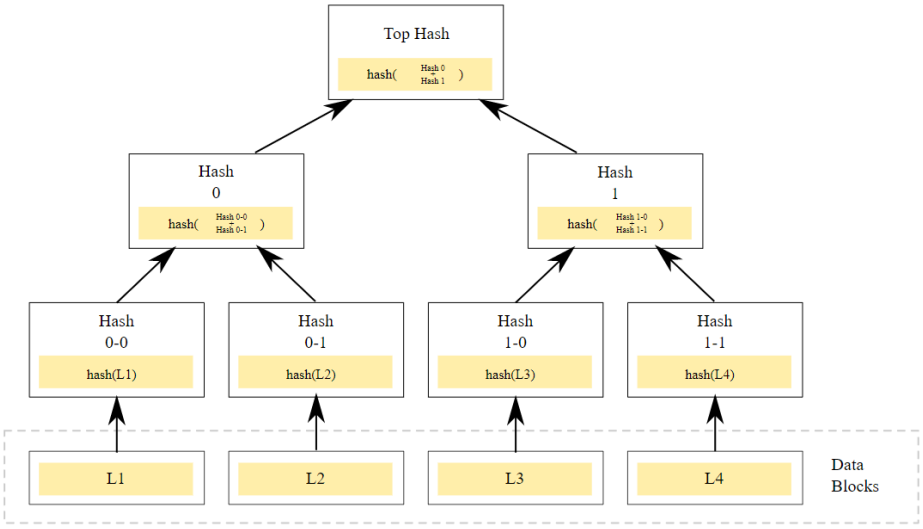

Figura 3: Merkle Tree

2.10 La crittografia asimmetrica

Per aumentare la sicurezza delle blockchain l'hash viene combinato con tecniche di crittografia asimmetrica, ovvero basate su chiave pubblica e privata, utilizzate per firmare una transazione. Queste chiavi sono legate tra di loro matematicamente. I dati possono essere firmati tramite la crittografia asimmetrica, ovvero si effettua una cifratura dell'hash e si invia sul canale già cifrato della blockchain. Per verificare che il dato sia firmato correttamente si può verificare la firma tramite la chiave pubblica del sender. Le chiavi pubbliche e private sono ottenute tramite la crittografia ellittica, un tipo di crittografia asimmetrica basata sulle curve ellittiche. La chiave pubblica è necessaria da parte della rete per poter inviare transazioni verso il possessore di quella chiave. Il mittente cifra il messaggio utilizzando la chiave pubblica. Il destinatario (possessore della coppia di chiavi) può decifrare il messaggio utilizzando la chiave privata. Questo fa si che soltanto il possessore della coppia di chiavi possa leggere i messaggi che gli sono stati inviati. Gli algoritmi a cifratura asimmetrica vengono utilizzati nella firma digitale e quindi appunto per firmare una transazione. Ogni transazione può essere quindi

verificata grazie alla firma.

Le firme digitali si basano su tre concetti fondamentali:

- **Autenticità** Assicurare e garantire che chi ha firmato il documento si è assunto la responsabilità del suo contenuto.
- **Integrità** E' la condizione che serve a dimostrare che dal momento in cui il documento è stato firmato non è stato più modificato.
- **Non ripudio** Chi ha firmato il documento non può più disconoscerne il contenuto.

L'algoritmo utilizzato nelle blockchain è basato sulle curve ellittiche ed è chiamato Elliptic Curve Digital Signature Algorithm. [10]

3 Classificazione delle blockchain e algoritmi di consenso

3.1 Classificazione delle blockchain

Le blockchain si possono classificare in base a tre parametri principali che vedremo qui di seguito.

3.1.1 Blockchain permissioned e permissionless

- **Blockchain permissionless** Una blockchain permissionless non necessita di autorizzazioni per far entrare un nodo a far parte del network. Inoltre un qualsiasi nodo che entra nella rete può fare tutte le operazioni relative alla blockchain. Questo garantisce una decentralizzazione totale del sistema.

- **Blockchain permissioned** Al contrario delle permissionless, le blockchain permissioned sono controllate da un'entità. Questo ente ha il diritto di scegliere chi può far parte o meno della blockchain. In questo modo c'è una forte centralizzazione del sistema e si va a perdere, secondo molti, il senso per cui è nata originariamente la blockchain, ovvero la completa decentralizzazione.

3.1.2 Tipi di accesso

Il tipo di accesso indica il grado di visibilità della blockchain, e può essere di tre tipi.

- **Blockchain pubblica** In una blockchain pubblica chiunque può accedervi ed entrare a far parte del network, minare blocchi ed effettuare transazioni.

- **Blockchain privata** In una blockchain privata si ha invece un controllo sugli accessi, e si può decidere chi può farne parte e chi no. è generalmente proprietà di un ente singolo.

- **Blockchain consortile** Le blockchain consortili sono un'accezione delle blockchain private, in quanto è presente un controllo degli accessi ma non c'è disparità tra i partecipanti al network, ovvero non c'è un ente centrale.

3.2 Algoritmi di consenso

L'obiettivo di questi algoritmi è di creare consenso tra tutte le parti che compongono la rete. Ci sono molti tipi di algoritmi di consenso.

- **Proof of Work (PoW)** Il Proof of Work è l'algoritmo di consenso alla base di molte blockchain, come Bitcoin o Ethereum (anche se Ethereum si sta muovendo verso il Proof of Stake). Si basa sulla potenza di calcolo degli elaboratori che vengono utilizzati per risolvere un problema matematico complesso. Chi riesce a risolvere questo problema viene ricompensato dal network tramite degli incentivi, come per esempio della criptovaluta. Il concetto si basa su complessi problemi matematici e sulla necessità di dimostrare in modo semplice le soluzioni a questi. Esistono diverse tipologie di problemi matematici utilizzati per il PoW, quello più conosciuto è relativo alla funzione di Hash, ovvero dover trovare un input partendo da un output. La difficoltà del problema dipende dal numero di utenti, dalla potenza totale disponibile e dal carico della rete. La hash di ogni blocco contiene la hash del blocco a lui precedente, garantendo così l'immutabilità della catena. Il PoW è sicuro in quanto se si volesse modificare un'informazione all'interno della catena sarebbe necessario ricalcolare tutti gli hash della catena da quel blocco in poi, e questo richiederebbe una potenza di calcolo non raggiungibile, rendendo così assolutamente sicura la blockchain. Gli svantaggi del consenso PoW sono i costi elevati di potenza di calcolo necessaria per risolvere questi calcoli. Una soluzione è stata implementata da Ethereum che utilizza un algoritmo di consenso memory-intensive chiamato Ethash [13]. E' comunque presente il rischio degli attacchi 51%, ovvero i casi in cui qualcuno riesca a prendere il controllo del 50%+1 della rete, arrivando quindi a controllare l'esito degli eventi al suo interno, in quanto potrebbe approvare ogni transazione. [3]

- **Proof of Stake (PoS)** Il PoS è l'algoritmo di consenso nato dopo il PoW. I nodi che fanno da miner in PoS vengono chiamati "Validatori". La decisione su quale nodo deve validare un blocco è parzialmente random, segue anche infatti una politica di fiducia, secondo la quale secondo alcuni criteri alcuni nodi hanno più probabilità di altri di essere estratti per validare un blocco. Il nodo scelto dalla rete fa da garante per il blocco coniato offrendo della valuta come garanzia. Se il blocco è accettabile allora il Validatore ottiene la commissione, altrimenti perde

la valuta offerta quando aveva deciso di garantire un blocco. [4]

- **Proof of Authority (PoA)** Nella PoA le transazioni e i blocchi vengono validati da degli account detti "validatori". I validatori firmano un blocco e un blocco firmato entra a far parte della blockchain e viene considerato come accettato. I validatori sono predeterminati all'interno del genesis file. Possono essere proposti alla rete nuovi validatori, ma devono essere votati da almeno il 50%+1 degli attuali validatori. Una politica simile si adotta nel caso in cui si voglia rimuovere un validatore. In questa tesi vengono analizzati proprio i meccanismi relativi a una rete Proof of Authority (PoA).[5]

- **Proof of Burn (PoB)** L'algoritmo di consenso basato sul Proof of Burn è molto simile al PoW ma permette un consumo energetico molto più basso. Gli utenti possono dimostrare il loro impegno nei confronti del network, ottenendo il diritto di "minare" e convalidare transazioni effettuando un *coin burn*. Questo processo di eliminazione delle monete consiste nell'inviarle a indirizzi verificabili pubblicamente e che sono inutilizzabili. [6]

- **Proof of Space** Il Proof of Space è un meccanismo di consenso basato sull'allocazione di memoria. Un partecipante al network alloca una grossa quantità di memoria per la risoluzione di un problema della blockchain. Quest'allocazione prova l'interesse del nodo a effettuare una transazione. [7]

- **Proof of Elapsed Time (PoET)** Nell'algoritmo PoET ogni nodo del network deve bloccarsi in attesa di un tempo random, il nodo che finisce per primo "vince" il nuovo blocco. Sostanzialmente ogni nodo della blockchain genera un tempo random e passa in sleep e il primo che si sveglia ottiene il diritto di minatore e vince il blocco nella rete blockchain. [8]

3.3 Un focus sul meccanismo di consenso Proof-of-Authority

3.3.1 Che cos'è il meccanismo di consenso Proof-of-Authority

Nelle piattaforme blockchain permissioned tutti i nodi sono pre-autenticati, questo permette di usare i meccanismi di consenso che permettono un elevato

tasso di transazioni e molti altri benefici. Nel Proof-of-Authority i diritti per generare nuovi blocchi sono soltanto dei nodi che hanno provato la loro autorità al sistema. Per ottenere questa autorità e il diritto di generare nuovi blocchi, il nodo deve passare un'autenticazione preliminare.

3.3.2 I nodi validatori

I nodi validatori sono quei nodi che nelle blockchain PoA hanno i diritti per generare nuovi blocchi. Inoltre sono quei blocchi su cui si basa tutto il sistema di autenticazione. I nodi validatori (Sealer) hanno il diritto di eleggere nuovi nodi a validatori, e dare quindi l'autorità a questi ultimi sia di generare nuovi blocchi, sia di eleggere a loro volta nuovi nodi Sealer.

3.3.3 Algoritmi per la gestione dei nodi validatori

Esistono diversi agloritmi per la gestione dei nodi validatori, quello utilizzato all'interno di questo lavoro di tesi è il protocollo di consenso Clique EIP-225 [16] utilizzato sulla blockchain Ethereum, di cui parleremo più avanti. Clique è basato sul design della mainnet di ethereum, questo lo rende facilmente utilizzabile su ogni client. La parte tecnica di Clique verrà spiegata succesivamente nel modulo relativo ad Andromeda.

4 Ethereum

4.1 Introduzione

Ethereum è una piattaforma open source globale per applicazioni decentralizzate. Su ethereum è possibile scrivere codice per controllare il valore digitale, che funzioni esattamente come programmato e sia accessibile da ogni parte del mondo.

La blockchain di ethereum è più versatile rispetto a quella di bitcoin in quanto è programmabile dagli utenti stessi. La principale differenza sta soprattutto nel fatto che ethereum oltre a fornire transazioni di criptovalute permette di scrivere e implementare gli smart contracts, ovvero dei veri e propri account controllati da del codice. Nasce principalmente in quanto i precedenti protocolli di blockchain esistenti erano degli "Strumenti monouso" progettati esclusivamente per una specifica applicazione, ad esempio la blockchain bitcoin era progettato per la criptovaluta, mentre quella di Namecoins era progettato per il domain name registration. La seconda generazione di protocolli blockchain, tra cui Ethereum, comprende invece moltissimi tipi di transazioni possibili basati sulla struttura della blockchain. Inoltre se un'applicazione diventa particolarmente rilevante verrà aggiunta a questa lista di transazioni possibili mantenendo quindi il protocollo blockchain aggiornato. Per capire meglio il concetto possiamo immaginare il sistema di cui stiamo parlando come uno smartphone, abbiamo un sistema operativo, che possiamo vedere come la blockchain, e su di esso possiamo avere delle app, ovvero gli applicativi che sviluppiamo con la blockchain. Gli sviluppatori possono poi creare nuovi applicativi e aggiungerli allo "store". Fondata nel 2013 in forma completa per mano di Vitalik Buterin, sviluppatore di origini Russe. Buterin dopo la stesura del Bitcoin Magazine si mette all'opera per la creazione di una blockchain più innovativa rispetto a quella di bitcoin, ovvero Ethereum. Nel 2014 ethereum diventa ufficialmente pubblica e in poco tempo la criptovaluta basata sulla blockchain ethereum, ovvero L'Ether (ETH), entra a far parte delle criptovalute più importanti e utilizzate al mondo.

Figura 4: Ethereum

4.2 Le differenze rispetto alle precedenti blockchain

La differenza sostanziale del protocollo Ethereum rispetto alle precedenti versioni di blockchain è la presenza di un linguaggio Turing completo che permette lo sviluppo di smart contract sulla blockchain tramite l'inserimento degli stessi all'interno di transazioni. Se una transazione contenente uno smart contract viene approvata viene creato un account il cui indirizzo è legato allo smart contract, e tramite questo indirizzo si può invocare direttamente. Chiunque può creare applicazioni definendo un contratto. Ethereum inoltre utilizza un algoritmo PoW denominato ethash specifico per il protocollo ethereum. La PoW di Ethereum utilizza i Grafi Aciclici Orientati associati a una memorizzazione a livelli dipendenti.

4.3 EVM - Ethereum Virtual Machine

La Ethereum Virtual Machine (EVM) è l'ambiente di esecuzione che permette l'esecuzione degli smart contract all'interno del protocollo blockchain ethereum, è una vera e propria macchina virtuale, considerata anche come ambiente protettivo in quanto ciò che viene eseguito all'interno della EVM è isolato rispetto al sistema esterno della blockchain. Ogni nodo del network possiede la propria EVM, e ogni nodo che vuole scrivere uno smart contract dovrà compilare il codice di alto livello per produrre il bytecode EVM. La Ethereum Virtual Machine è basata sull'architettura *stack-based* come le classiche virtual machine (ad esempio la JVM). All'interno delle EVM troviamo infatti:

- **Stack** Lo stack adotta la politica LIFO ed è dove vengono mantenute le operazioni. I dati inseriti all'interno dello stack sono volatili, e le operazioni possibili sono la PUSH e la POP, eseguite tramite politica LIFO.
- **Memory** La Memory è un array di byte anch'esso volatile, ma senza limitazioni in dimensione.
- **Storage** Lo Storage è una memoria persistente, parte del sistema, utilizzata per gli smart contracts.
- **Envroiment Variables** Contiene le variabili d'ambiente a cui la EVM potrebbe dover far accesso, come ad esempio il numero del blocco, la difficoltà di mining o l'hash del blocco precedente.

4.4 Gli account

Un account ethereum è un'entità che possiede della valuta della blockchain di riferimento e può inviare e ricevere transazioni. Gli account possono essere di due tipi: [9]

- **Externally-Owned Account (EOA)** Un EOA è un account di possesso di un user. Non costa nulla creare un EOA e da esso si possono effettuare transazioni verso altri EOA o verso CA.
- **Contract Account (CA)** Fanno parte della seconda generazione di blockchain, come ethereum, un CA permette d'identificare uno smart contract di cui è stato fatto il deploy sulla blockchain. La creazione di un CA ha un costo in quanto si usa lo storage del network. Si possono solo effettuare transazioni di risposta a transazioni in entrata.

4.5 Gli smart contract

Gli smart contract sono dei programmi scritti in un linguaggio di programmazione Turing completo, ovvero un linguaggio di programmazione che ci permette d'implementare una qualsiasi macchina di Turing. Un linguaggio turing-complete è quindi basato sulla macchina di turing e può essere utilizzato per risolvere ogni problema che ammette soluzione. Sostanzialmente uno smart contract è un programma che viene pubblicato sulla blockchain, e quindi ne viene garantita l'immutabilità. Gli smart contract sono la rivoluzione principale del protocollo Ethereum rispetto a Bitcoin. Uno smart contract è quindi del codice e dati dispiegati usando transazioni firmate crittograficamente sulla blockchain. Quando si lancia una transazione su ethereum può essere diretta ad un address CA. In questo caso si sta attivando lo smart contract relativo a questo address.

4.6 Struttura di un blocco

Un blocco in ethereum è composto dai seguenti campi [11]

Block Height E' conosciuto anche come il block number. Indica la lunghezza della blockchain fino a questo blocco. è incrementato di 1 all'aggiunta di un altro blocco.

Timestamp La data e l'ora nelle quali il blocco è stato minato.

Transactions Indica il numero di transazioni nel blocco.

Mined by Minatore che ha minato il blocco con successo.

Block Reward Ricompensa del minatore in Ether.

Difficulty La difficoltà richiesta per minare un nuovo blocco.

Total Difficulty Difficoltà totale della blockchain fino a questo blocco.

Size La dimensione del blocco è data dal GAS Limit del blocco stesso.

Gas Used GAS utilizzato nel blocco e percentuale relativa al GAS Limit.

Gas Limit GAS Limit totale delle transazioni nel blocco.

Extra Data Tutti i dati includibili dal miner nel blocco.

Hash Hash del blocco.

Parent Hash Hash del blocco precedente a questo nella blockchain.

Nonce Il nonce del blocco è un valore utilizzato durante il mining per dimostrare il PoW del blocco.

4.7 ETH

L'Ether, o ETH, è la valuta nativa di Ethereum. E' una "moneta digitale" che può essere inviata via Internet istantaneamente e a basso costo, ed è usata anche in molte applicazioni basate su Ethereum. Il modo più semplice per ottenere ETH è acquistarli. Ci sono molte piattaforme di scambio per criptovalute su cui è possibile acquistare ETH, ma quelle che devi utilizzare dipendono da dove vivi e come vuoi pagare. Gli Exchange sono gli enti generalmente preposi a fare compra vendita di moneta digitale. Sono enti

che acquistano, in grandi quantità, della moneta e la rivendono poi ai loro clienti. La sotto unità di ETH è il Wei (1 ETH = 1e+18 wei). Le criptovalute sono divisibili praticamente all'infinito e non c'è un limite alla dimensione minima in cui dividere un'unità di valuta. [12]

4.8 Solidity

Solidity è un linguaggio Turing completo di alto livello orientato agli oggetti utilizzato per l'implementazione di smart contract. Solidity è basato su linguagi come Python, C++ e javascript ed è progettato per la Ethereum Virtual Machine. E' un linguaggio tipizzato staticamente, supporta l'ereditarietà, le librerie e i tipi complessi definiti dall'utente. I contratti solidity sono simili alle classi dei linguaggi object-oriented. Ogni contratto può contentere dichiarazioni di variabili di stato, modificatori di funzione, tipi di stuttura e di enumerazione, funzioni ed eventi. I contratti possono inoltre ereditare da altri contratti.

- **Variabili** Le variabili hanno valori memorizzati in modo permanente nella memoria del contratto.

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity >0.7.0 <0.8.0;
3  contract Example{
4      uint myVariable; //variabile di stato
5      // ...
6  }
```

Codice 1: Solidity - Variabili

- **Funzioni** Le funzioni sono delle unità eseguibili di codice. Possono essere definite all'interno o all'esterno del contratto.

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity >0.7.0 <0.8.0;
3  contract Example{
4      function myFunc() public{
5          // ...
6      }
7  }
8  function myFunc2(uint x) pure returns(uint){
9      return 1
10 }
```

Codice 2: Solidity - Funzioni

- **Modificatori di funzione** I modificatori di funzione sono usati per modificare la semantica delle funzioni in modo dichiarativo. Non è consentito l'overloading.

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity >=0.4.22 <0.9.0;
3
4  contract Purchase {
5      address public seller;
6
7      modifier onlySeller() { // Modifier
8          require(
9              msg.sender == seller,
10             "Only seller can call this."
11         );
12         _;
13     }
14
15     function abort() public view onlySeller { //
           Modifier usage
16         // ...
17     }
18 }
```

Codice 3: Solidity - Modificatori di funzione

- **Eventi** In solidity è possibile gestire gli eventi.

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity >=0.4.21 <0.9.0;
3
4  contract SimpleAuction {
5      event HighestBidIncreased(address bidder, uint
           amount); // Event
6
7      function bid() public payable {
8          // ...
9          emit HighestBidIncreased(msg.sender, msg.
               value); // Triggering event
10     }
11 }
```

Codice 4: Solidity - Eventi

- **Strutture** Le strutture sono tipi definiti dall'utente che possono raggruppare variabili di diversi tipi.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.0 <0.9.0;

contract Ballot {
    struct Voter { // Struct
        uint weight;
        bool voted;
        address delegate;
        uint vote;
    }
}
```

Codice 5: Solidity - Strutture

- **Tipi di dati** Solidity è un linguaggio statically typed, ovvero ogni variabile ha bisogno di avere il tipo dichiarato. Supporta tutti i tipi di base, i boolean, gli shift, il tipo "address", il tipo "contract" e molti altri.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.0 <0.9.0;

contract MappingExample {
    mapping(address => uint) public balances;

    function update(uint newBalance) public {
        balances[msg.sender] = newBalance;
    }
}

contract MappingUser {
    function f() public returns (uint) {
        MappingExample m = new MappingExample();
        m.update(100);
        return m.balances(address(this));
    }
}
```

Codice 6: Solidity - Tipi di dati

Solidity mantiene comunque una precisa e aggiornata documentazione online (docs.soliditylang.org)

4.9 Gas

Come si può essere sicuri che uno smart contract una volta lanciato termini? Questo è un problema detto *di terminazione*. Non esiste una soluzione in quanto non esiste un algoritmo che ci permetta di sapere a priori se un programma terminerà o meno. Ethereum utilizza un metodo quindi per interrompere un programma per evitare il problema di terminazione. La soluzione è data dal concetto di *GAS*.

Al momento della creazione, una transazione viene caricata con una certa quantità di gas, detta *GAS Limit*. Grazie alla quale possiamo limitare l'esecuzione dello smart contract risolvendo il problema di terminazione.

Durante l'esecuzione dello smart contract da parte della EVM il GAS viene esaurito.

Il creatore della transazione imposta un *Gas Price*.

I miner come ricompensa per aver minato la transazione otterranno il quantitativo in ETH pari a GAS Price * gas.

Se al termine della transazione è rimasta una certa quantità di gas questa verrà rimborsata al creatore della transazione, se invece viene esaurito il gas, ovvero si supera il GAS Limit, viene lanciata una eccezione che fa abortire lo smart contract eseguendo una reverse di ogni operazione effettuata. Il miner verrà comunque pagato per la transazione.

Ethereum ha dei gas price predefiniti per ogni transazione, ma il mittente può decidere il prezzo in Wei (sottovaluta di ETH) di ogni unità di GAS. Questo porta ad una competizione che sta facendo alzare sempre di più il costo in ETH del GAS, in quanto ai miner conviene eseguire smart contract con prezzo ETH relativo al GAS più alto, in quanto la loro commissione sarà più alta. [14]

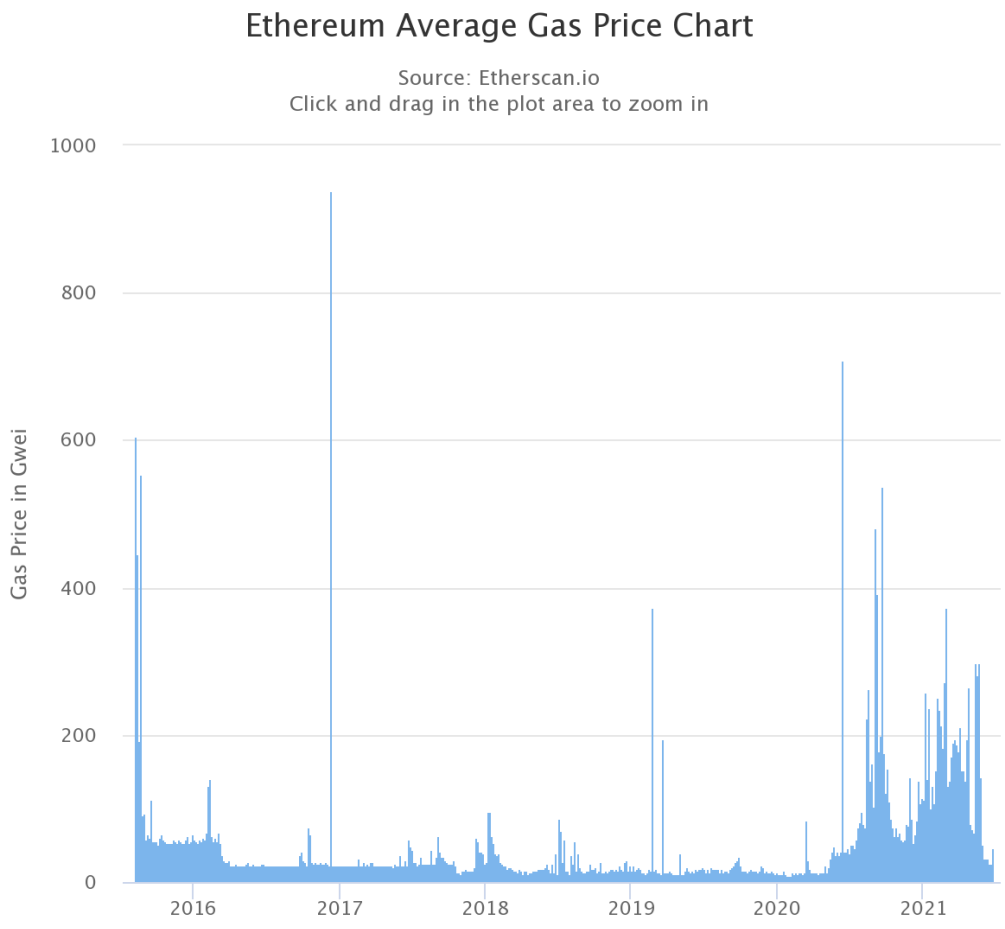

Figura 5: Ethereum average gas price

4.10 DApp

Le DApp sono delle applicazioni decentralizzate e open-source. Sono sostanzialmente delle interfacce per gli smart contract Solidity. Vengono eseguite sopra un network peer-to-peer, in questo caso la blockchain, in modo da rendere ogni informazione sempre condivisibile a tutti i partecipanti.

Una volta distribuito sulla blockchain il codice della dapp non può essere ne rimosso ne modificato, chiunque può utilizzare le funzionalità della dapp.

Nessuno può impedire d'inviare transazioni usando una dapp.

I pagamenti sono integrati in quanto essendo nativi di ethereum (ETH) non c'è bisogno d'integrarli tramite intermediari terzi.

L'anonimato è garantito in quanto per la maggior parte delle dapp è sufficente possedere un wallet per poterle utilizzare.

La crittografia inoltre le rende estremamente sicure.

Un'applicazione per essere considerata una dApp deve essere completamente open-sourced, deve operare in maniera autonoma e senza un'entità che controlli la maggior parte dei token. L'applicazione può adattare il suo protocollo ma tutte le modifiche devono essere decise tramite un protocollo di consenso dai suoi utenti.

5 Andromeda

5.1 Il progetto

Il progetto è sviluppato nel contesto del progetto di ricerca PININ. Il lavoro mira a costruire un'infrastruttura distibuita e decentralizzata basata su una Blockchain di tipo Proof of Authority EIP-225 (clique). Qualsiasi nodo Signer della rete può aggiungere un nodo alla rete facendo una richiesta di elezione. Il nodo verrà aggiunto in blockchain soltanto dopo che la maggioranza dei Signer avranno votato acconsentendo all'entrata nella rete del nodo proposto. Analogamente si possono effettuare votazioni anche per rimuovere un nodo dalla posizione di Signer oppure semplicemente votare contro l'ammissione di un nodo proposto. Verrà inoltre mantenuto uno storico di tutti i voti effettuati. Questa votazione verrà effettuata tramite l'app "Andromeda", sviluppata durante questo lavoro di tesi.

5.2 Andromeda

Andromeda è un'applicazione del progetto PININ, questa tesi si articola nello sviluppo di un front-end in React.js per visualizzare i servizi back-end preesistenti di Andromeda tra cui la costruzione di un blockchain explorer per la visualizzazione dei blocchi in blockchain, le relative informazioni di un blocco, e le transazioni collegate al blocco stesso.
Inoltre viene sviluppata la parte di elezione in blockchain per la parte front-end React.js, back-end Java spring e scrittura d'informazioni di supporto sulla blockchain tramite smart contract Solidity.
Il front end è stato sviluppato modularmente così da mantenere il codice pulito e organizzato, favorendo low coupling e high coesion.
Il modulo di blockchain explorer è relativamente semplice. Vengono infatti visualizzati tabellarmente i blocchi della blockchain che arrivano tramite un servizio di back-end in formato JSON.
Per il modulo elezioni vedremo in seguito le tecnologie utilizzate.

6 Elezioni in una blockchain PoA su Andromeda

6.1 Creazione blockchain PoA

Per lo sviluppo del modulo di Elezioni di Andromeda ho esegito i test su una rete blockchain proof of authority in locale creata tramite geth (GoEhtereum), ovvero una delle tre implementazioni originali del protocollo Ethereum. Di seguito vediamo come creare la blockchain locale PoA.

Per prima cosa è necessario inizializzare il workspace e creare la struttura che ci permetterà di contenere la nostra rete. Creiamo una directory per la rete e le sotto-directory per i vari nodi della rete.

```
1    $> mkdir myEthPoa
2    $> cd myEthPoa
3    $> mkdir node1 node2 node3
```

Codice 7: geth - Inizializzazione directory

Successivamente procediamo alla creazione degli account. I wallet hanno una coppia di chiavi privata-pubblica per interagire con la blockchain. Un nodo Sealer ha bisogno della chiave privata per mandare transazioni sulla rete, e ha bisogno di una chiave pubblica per identificarsi all'interno del network. Un nodo sealer puo votare altri nodi per eleggere un nodo come Sealer all'interno della blockchain PoA. Per inizializzare gli account procediamo nel seguente modo per ogni nodo, impostando poi una password per il nodo stesso.

```
1    myEthPoa$> geth --datadir node1/ account new
2    myEthPoa$>
3    INFO [05-11|13:39:34.555] Maximum peer count ETH=50 LES=0
         total=50
4    Your new account is locked with a password. Please give a
         password. Do not forget this password.
5    Password: pswnode1
6    RepeatPassword: pswnode1
7
8    Your new key was generated
9
10   Public address of the key:   0
        x146dCA62D5353bB20C116E3F8A14A4f975C3A173
11   Path of the secret key file: node1/keystore/UTC
        --2021-05-11T11-52-58.571600400Z--146
        dca62d5353bb20c116e3f8a14a4f975c3a173
```

```
12
13      - You can share your public address with anyone. Others
           need it to interact with you.
14      - You must NEVER share the secret key with anyone! The
           key controls access to your funds!
15      - You must BACKUP your key file! Without the key, it s
           impossible to access account funds!
16      - You must REMEMBER your password! Without the password,
           it s impossible to decrypt the key!
```

Codice 8: geth - Creazione account

Otterremo così anche il public address dell'account. Possiamo salvarci ogni password all'interno di un file "node1/pswnode1.txt" per facilitare poi un procedimento che vedremo successivamente.

Procediamo ora alla creazione del genesis file. Il genesis block è il primo blocco della blockchain, "il blocco zero" che indica l'inizio della blockchain e contiene tutti i dati di inizializzazione della blockchain. Procederemo alla creazione di questo file tramite *puppeth*.

```
1  $>puppeth
2
3  Please specify a network name to administer (no spaces,
      hyphens or capital letters please)
4  > myethpoa
5
6  What would you like to d o? (default = stats)
7   1. Show network stats
8   2. Configure new genesis
9   3. Track new remote server
10  4. Deploy network components
11 > 2
12
13 What would you like to d o? (default = create)
14  1. Create new genesis from scratch
15  2. Import already existing genesis
16 > 1
17
18 Which consensus engine to use? (default = clique)
19  1. Ethash - proof-of-work
20  2. Clique - proof-of-authority
21 > 2
22
```

```
How many seconds should blocks take? (default = 15)
> 5
```

Codice 9: geth - Puppeth 1

Ora possiamo procedere a indicare quali account saranno dei Sealer indicando il loro address.

```
Which accounts are allowed to seal? (mandatory at least one)
> 0x146dCA62D5353bB20C116E3F8A14A4f975C3A173
> 0xd087cCDB19b917bbb5880a623675CBde4308B124
> 0x

Which accounts should be pre-funded? (advisable at least one)
> 0x146dCA62D5353bB20C116E3F8A14A4f975C3A173
> 0xd087cCDB19b917bbb5880a623675CBde4308B124
> 0x

Should the precompile-addresses (0x1 .. 0xff) be pre-funded
   with 1 wei? (advisable yes)
> yes
```

Codice 10: geth - Puppeth 2

Successivamente impostiamo il chain id della nostra rete e configuriamo le ultime impostazioni del genesis.

```
Specify your chain/network ID i f you want an explicit one (
   default = random)
> 1555
INFO [05-11|14:07:17.360] Configured new genesis block

What would you like to d o? (default = stats)
 1. Show network stats
 2. Manage existing genesis
 3. Track new remote server
 4. Deploy network components
> 2

 1. Modify existing configurations
 2. Export genesis configurations
 3. Remove genesis configuration
> 2

Which folder to save the genesis specs into? (default =
   current)
  Will create myethpoa.json, myethpoa-aleth.json, myethpoa-
     harmony.json, myethpoa-parity.json
```

```
19 >
```

Codice 11: geth - Puppeth 3

Possiamo ora procedere alla creazione del servizio bootnode. I nodi all'interno di una rete, per sapere a quali indirizzi si trovano gli altri nodi, utilizzano il bootnode, che permette di fare da dispatcher delle richieste. Siccome i nodi possono avere IP dinamici, tramite il bootnode che possiede un IP statico, possono sapere sempre a quali indirizzi sono associati gli altri nodi della rete (in quanto il bootnode ne tiene traccia).

```
1 myEthPoa$> bootnode -genkey boot.key
2 myEthPoa$> bootnode -nodekey boot.key -verbosity 9
```

Codice 12: geth - Bootnode

Il campo "enode" serve a identificare il bootnode della rete, e verrà usato dai nodi al momento della run. Per lanciare un nodo usiamo il seguente comando.

```
1 geth --identity 'node' --allow-insecure-unlock --mine -unlock
    0x146dCA62D5353bB20C116E3F8A14A4f975C3A173 --password
   node1/password.txt --datadir node1/ --syncmode 'full' --
   port 30311 --rpc --rpcaddr 127.0.0.1 --rpcport 8501 --
   rpccorsdomain "*" --rpcapi 'personal,db,eth,net,web3,
   txpool,miner,clique' --bootnodes 'enode://...@127.0.0.1:0?
   discport=30301' --networkid 1555 --ipcpath "./node1/geth.
   ipc"
```

Codice 13: geth - Run di un nodo

- **identity** Nome del nodo.
- **datadir** Directory del nodo.
- **syncmode** Valore che indica l'opzione di sync del block (fast, full, light).
- **port** Porta di ascolto del network (diversa per ogni nodo).
- **rpc** Avvia l'interfaccia http JSON-RPC.
- **rpcaddr** Interfaccia di ascolto del server http-RPC (nel nostro caso sempre 127.0.0.1).
- **rpcport** Porta di ascolto del sever http-RPC (diversa per ogni nodo).

- **rpccorsdomain** Lista dei domini dai quali accettare richieste.
- **rpcapi** Lista delle API offerte dall'interfaccia http-RPC.
- **bootnode** Enode del bootnode.
- **networkid** Identificatore della rete.
- **ipcpath** Path nel qule salvare il file geth.ipc per poter eseguire una .

Per i nodi non sealer omettiamo il "- -mine".

A questo punto il nostro network è online e la blockchain PoA è correttamente inizializzata e verrà usata come base di testing per le elezioni in andromeda. Nell'immagine di seguito possiamo vedere quattro terminali, i primi tre con i nodi attivi della blockchain, i primi due Sealer e il terzo semplice nodo del network, e il quarto contiene invece l'istanza del bootnode.

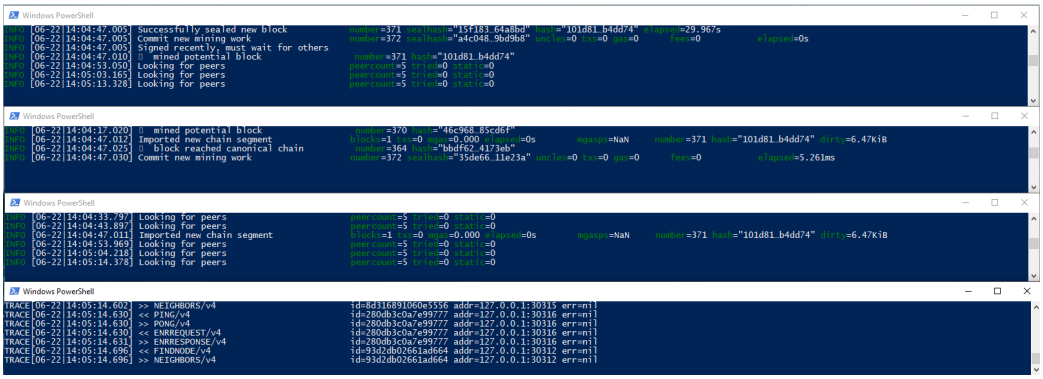

Figura 6: Blockchain Geth

6.2 Votazioni tramite Clique in una blockchain PoA

Nel caso della blockchain utilizzata, le votazioni per l'elezione di un nodo vengono effettuate tramite Clique EIP-225 [16].

Clique è un protocollo di consenso proof-of-authority che permette di effettuare operazioni sulla blockchain.

Il modulo elezioni (che esporremo successivamente) utilizza delle richieste tramite clique per effettuare le proposte di votazione. [17]

- **clique_getSigners** Ritorna una lista dei nodi autorizzati Signer.

Client	**Method invocation**
Console	clique.getSigners(blockNumber)
RPC	{"method": "clique_getSigners", "params": [blockNumber]}

- **clique_proposals** Ritorna i voti su cui l'account sta attualmente votando.

Client	**Method invocation**
Console	clique.proposals()
RPC	{"method": "clique_proposals", "params": []}

- **clique_propose** Aggiunge una nuova votazione. Se il parametro auth è settato a "true", il signer sta votando per ammettere l'address alla lista dei Signers autorizzati. Se auth è invece settato a "False" il voto è contro l'address.

Client	**Method invocation**
Console	clique.propose(address, auth)
RPC	{"method": "clique_propose", "params": [address, auth]}

- **clique_discard** Annulla una votazione corrente.

Client	**Method invocation**
Console	clique.discard(address)
RPC	{"method": "clique_discard", "params": [address]}

Da Andromeda si utilizzeranno l'IP e la porta del nodo chiamante per effettuare chiamate. Di seguito un esempio di votazione da javascript tramite clique.

```
1     const transport = new HTTPTransport('http
          ://127.0.0.1:8501');
2     const client = new Client(new RequestManager([transport])
          );
3     await client.request({
4                 mode: 'no-cors',
5                 method: 'clique_propose',
6                 params: [accountToVote,true],
7             });
```

Codice 14: andromeda - Votazione clique

6.3 Persistenza delle votazioni

Il limite di clique è che non è possibile mantenere uno storico dei voti. Infatti dopo aver votato un Sealer, se questo viene accettato o rifiutato dal 50%+1 dei partecipanti al network, il voto da noi effettuato non è più recuperabile successivamente. Quindi è necessario un altro modo per poter tener traccia di tutti i voti effettuati in passato, che siano o no andati a buon fine. Per fare questo ho deciso di appoggiarmi a uno smart contract scritto in Solidity, così

da mantenere la lista dei voti effettuati direttamente sulla blockchain, per garantirne così l'immutabilità e la non contraffacibilità dei dati. Il contratto viene chiamato ogni qualvolta si effettua una votazione, in questo modo salvo all'interno della blockchain che account ha votato, a quale account è diretto il voto, il voto effettuato, e la data di quando è stato effettuato il voto. Di seguito vediamo il contratto Solidity. In sostanza ogni qualvolta viene effettuata una votazione viene lanciata una emit di un evento con all'interno le informazioni della votazione. In questo modo è possibile sia mantenere le votazioni di un account sia filtrare i voti per address e data.

```
1  pragma solidity >=0.4.22;
2  pragma experimental ABIEncoderV2;
3
4  contract AndromedaElections{
5
6      event VoteEvent(address indexed votingAccountIndexer,
           uint32 indexed myDateIndexer, address votedAccount,
           string myVote);
7
8      function voteCommitted(address votedAccount, string
           memory myVote, uint32 myDate) public{
9          emit VoteEvent(msg.sender, myDate, votedAccount,
               myVote);
10     }
11
12
13 }
```

Codice 15: andromeda-contracts - AndromedaElections.sol

Vediamo ora come fare il deploy del contratto solidity sulla blockchain di test. Abbiamo visto che la nostra blockchain è attiva tramite geth sul localhost. Per fare il deploy di un contratto sulla blockchain utilizzeremo truffle [18]. Procediamo all'installazione di truffle all'interno della directory scelta per mantenere i nostri contratti.

```
1      $> npm install -g truffle
```

Codice 16: andromeda-contracts - installazione di truffle

Inizializziamo ora l'ambente di sviluppo. Posizionati nella directory scelta eseguiamo il seguente comando.

```
1      $andromeda-contracts> truffle init
```

Codice 17: andromeda-contracts - inizializzazione di truffle

Procediamo ora con il modificare il file "truffle-config.js" inserendo i dati della nostra blockchain, come l'indirizzo dell'host, la porta desiderata sulla quale chiamare le transazioni e l'id della rete.

```
1     module.exports = {
2     networks: {
3         development: {
4           host: "127.0.0.1",
5           port: 8501,
6           network_id: "*" // Match any network id
7         }
8       }
9     };
```

Codice 18: andromeda-contracts - truffle-config.js

Possiamo ora inserire il nostro AndromedaElections.sol all'interno di ./contracts Successivamente modifichiamo ./migrations/2_deploy_contrats.js aggiungendo una dependency per ciascun contratto di cui vogliamo effettuare un deploy.

```
1     const AndromedaElections = artifacts.require("./
         AndromedaElections.sol");
2
3     module.exports = function(deployer) {
4       deployer.deploy(AndromedaElections);
5     };
```

Codice 19: andromeda-contracts - 2_deploy_contracts.js

Effettuate queste operazioni possiamo effettuare la compilazione e il deploying del contratto sulla blockchain.

```
1     $>truffle compile
2     $>truffle migrate --network development
3     $>truffle migrate
```

Codice 20: andromeda-contracts - compilazione e migrazione truffle

Truffle durante la migrate ci fornirà l'address del contratto, che utilizzeremo tramite andromeda per effettuare chiamate e lanciare transazioni.

```
1  ======================
2
3     Replacing 'AndromedaElections'
4     ------------------------------
5     > transaction hash:    0xc075d79928609203af1de9ea...08b82
6     > Blocks: 0            Seconds: 28
```

```
   > contract address:    0
      x69bD8041D72515B6a47D9e0868405CfCA1723BB1
   > block number:        5
   > block timestamp:     1624960099
   > account:             0
      x611462386c55C7954cC4fF86f43CACfAE758132C
   > balance:
      904625697166532776746...6906558.246370341821325312
   > gas used:            608293 (0x94825)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.01216586 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.01216586 ETH


Summary
=======
> Total deployments:   2
> Final cost:          0.01600472 ETH
```

Codice 21: andromeda-contracts - truffle migration

In questo modo noi avremo modo di effettuare transazioni direttamente da andromeda usando l'address del contratto.

```
    const contractAddr = '0
       x69bD8041D72515B6a47D9e0868405CfCA1723BB1';
    const ElectionsContract = new web3.eth.Contract(contract,
        contractAddr);

    const result = await ElectionsContract.methods.
       voteCommitted(textFieldAddNodeToPropose, 'true', today
       ).send({
                        from: MY_ACCOUNT,
                    });
```

Codice 22: andromeda - chiamata a un contratto

6.4 Modulo Elezioni

Per il modulo elezioni vengono unite le chiamate Clique per effettuare le votazioni, con le quali effettivamente avviene la votazione dei nodi sulla Blockchain, riconosciute dall'intera rete, e le chiamate al contratto Solidity per mantenere una cronologia di tutte le votazioni effettuate fin'ora. Tramite javascript viene effettuata una chiamata clique_propose con parametri settati in modo da inviare la votazione al nodo indicato tramite interfaccia grafica. Successivamente, dopo che la votazione è andata a buon fine, viene salvato il voto in blockchain, inviando una transazione al contratto AndromedaElections.sol tramite una chiamata asincrona con mittente il nodo che sta effettuando la votazione.

```
1
2   async function handleClickAddNodeToProposedNodes(event) {
3     event.preventDefault();
4     handleToggle();
5     setBackDropMsg('Voting in progress...');
6     if (textFieldAddNodeToPropose.localeCompare('')) {
7       /** Add the node in blockchain */
8       console.log(textFieldAddNodeToPropose);
9       var nodeAddr = getNodeAddressFromName(
            textFieldAddNodeToPropose);
10      console.log(nodeAddr);
11      if (nodeAddr.localeCompare(null)) {
12        console.log('voted to propose started');
13        await client.request({
14          mode: 'no-cors',
15          method: 'clique_propose',
16          params: [nodeAddr, true],
17        });
18        console.log('voted to propose ended');
19
20        /** Add the vote to the history */
21
22        var today = new Date();
23        var todayBuono = new Date(
24          today.getFullYear(),
25          today.getMonth(),
26          today.getDate()
27        ).getTime();
28        console.log(todayBuono);
29
```

```
30          console.log('voted to propose save in blockchain
                cronology started');
31          setBackDropMsg('Saving vote history...');
32          const result = await ElectionsContract.methods
33            .voteCommitted(nodeAddr, 'true', todayBuono / 1000)
34            .send({
35              from: MY_ACCOUNT,
36            });
37          console.log('voted to propose save in blockchain
                cronology ended');
38          console.log('result: ', result);
39
40          handleClose();
41          giveFeedback('Account voted to become a Sealer!', '
                success');
42        } else {
43          console.log('errore');
44          handleClose();
45          giveFeedback('Error, the given account does not match
                !', 'error');
46        }
47      } else {
48        handleClose();
49        giveFeedback('Error!', 'error');
50      }
51    }
```

Codice 23: andromeda - votazione nodo e salvataggio nella cronologia votazioni

Dopo aver inizializzato l'ambiente di test otterremo il seguende ambiente di testing. le prime 6 finestre contengono ognuna un nodo della rete blockchain, Abbiamo da sinistra verso destra e dall'alto verso il basso dal nodo 1 al nodo 6. In basso possiamo invece vedere a sinistra il bootnode, che attivamente effettua il dispatch delle richieste dei nodi. mentre in basso a destra possiamo vedere la finiestra in cui tramite truffle abbiamo effettuato il deploy del contratto AndromedaElections.sol sulla blockchain appena citata, possiamo infatti vedere l'address del contratto tra le prime righe.

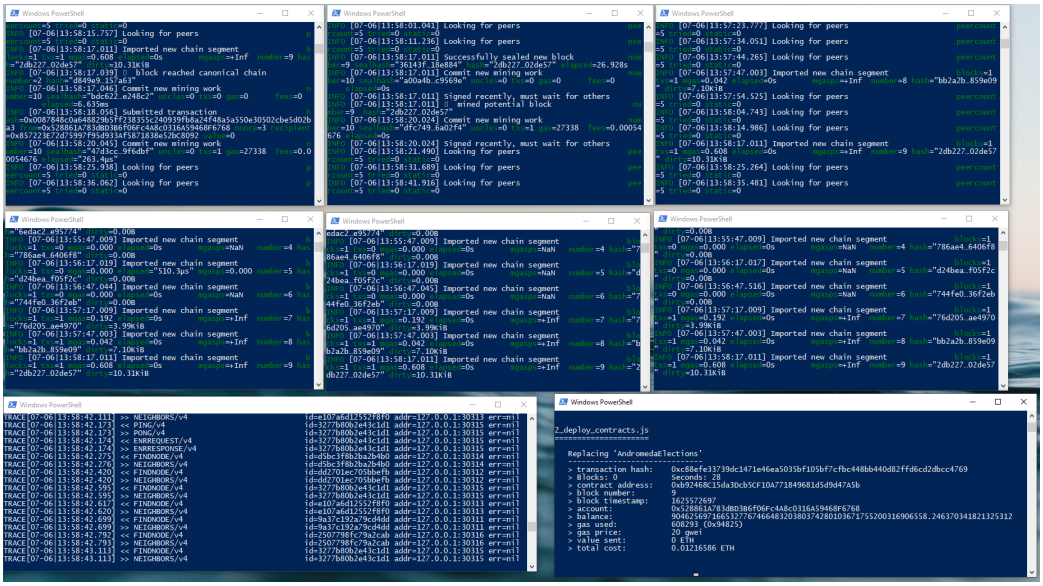

Figura 7: Andromeda ambiente di test

Il modulo elezioni è suddiviso in 4 finestre. la prima finestra comprende le informazioni relative alla rete, contiene infatti l'elenco di tutti i nodi presenti all'interno della blockchain, questa informazione viene ottenuta tramite il backend di andromeda con una semplice chiamata GET che restituisce un file JSON con le informazioni relative al nome del nodo e all'account. Questo permette di avere una rubricazione degli account. In questo modo infatti possiamo sapere il nome del nodo e il relativo account. Per quanto riguarda la lista dei nodi Signers questa viene recuperata tramite una chiamata clique direttamente alla blockchain, così da essere sempre aggiornata, mentre i nodi non Signers vengono calcolati per differenza tra i due precedenti insiemi.

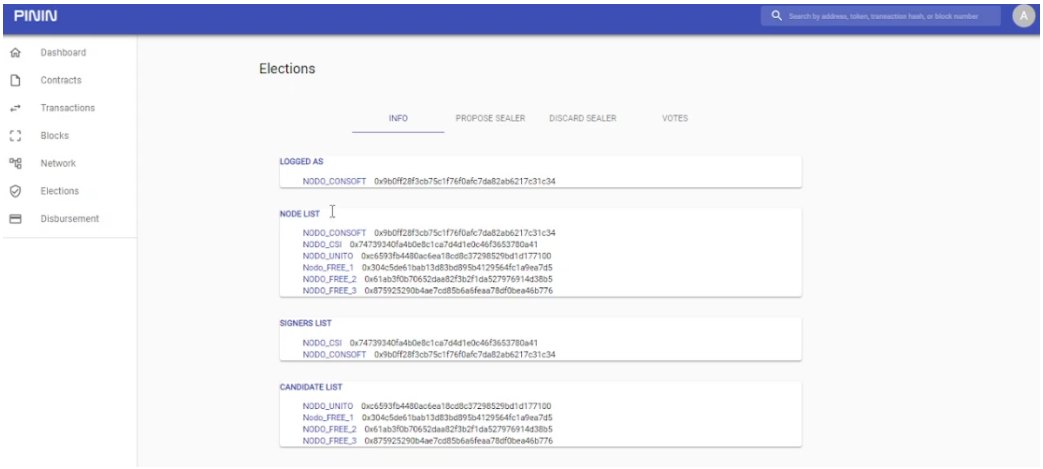

Figura 8: Andromeda info tab

La seconda e la terza finestra servono invece a votare positivamente o negativamente un determinato account. Tramite rubricazione è sufficiente inserire direttamente il nome del nodo e non l'intero account per poter votare. Se si propone un sealer viene effettuata una votazione tramite clique per aggiungere il nodo alla lista dei Signers, se poi questo verrà votato positivamente dal 50%+1 dei partecipanti al network allora entrerà a far parte dei Signer. Se invece si effettua un voto di discard si può rimuovere un account dalla lista degli account Sealers, e il procedimento è analogo a quello della propose. L'unica differenza sta nella chiamata clique effettuata, che in un caso prende come parametro di voto "true", mentre nell'altro "false", come spiegato nella sezione "Votazioni tramite Clique in una blockchain PoA".

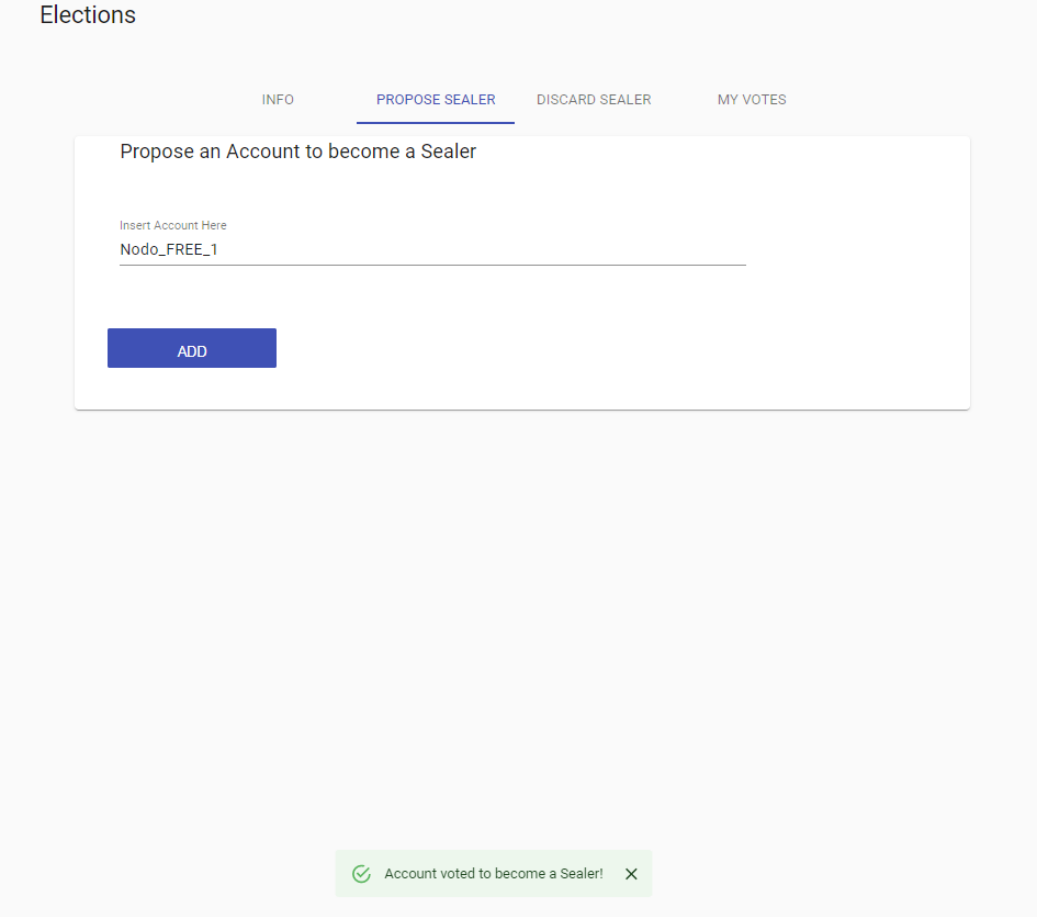

Figura 9: Andromeda propose tab

Infine abbiamo la finestra dello storico delle votazioni, Lo storico delle votazioni viene reperito direttamente dalla blockchain tramite una chiamata al contratto AndromedaElections.sol direttamente dal codice javascript di andromeda. In questo modo la lista delle votazioni non è contraffacibile ed è immutabile. I voti sono filtrabili così da poter ottenere oltre che a tutti i voti dell'intero sistema anche i voti di un singolo account o in una singola data.

```
1     async function getVotes(){
2
3         let result = await props.ElectionsContract.methods.
              getVotes().call({
4             from: props.MY_ACCOUNT,
```

```
5         });
6
7         let votedAccounts = [];
8         let votes = [];
9         let dates = [];
10        for(let counter=0;counter<result.vote.length;counter
              ++){
11            votedAccounts.push(result.nodeVoted[counter]);
12            votes.push(result.vote[counter]);
13            dates.push(result.date[counter]);
14        }
15        setMyVotedAccounts(votedAccounts);
16        setMyVotes(votes);
17        setMyDates(dates);
18    }
```

Codice 24: andromeda - Recupero storico voti

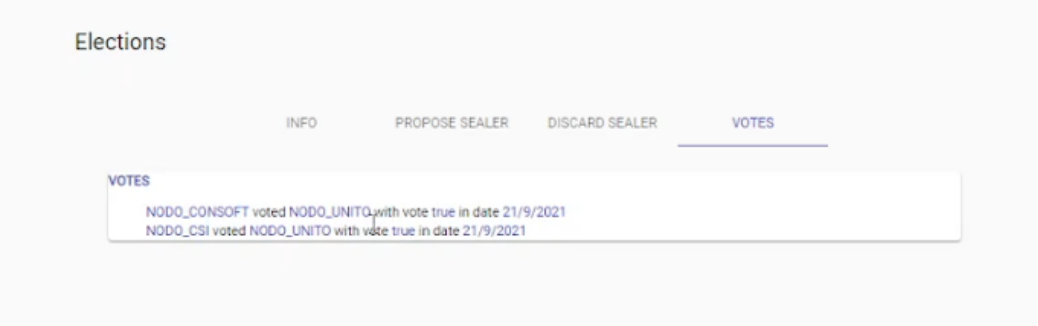

Figura 10: Andromeda cronologia voti

Per il filtro dei voti si utilizza una chiamata alla blockchain in javascript.

```
1
2   async function getVotes() {
3     const myRes = await props.ElectionsContract.getPastEvents
          ('VoteEvent', {
4       filter: {
5         votingAccountIndexer: filterAccount,
6         myDateIndexer: filterDate,
7       },
8       fromBlock: 1,
9       toBlock: 'latest',
```

```
10      });
11
12      setAllVotes(myRes);
13
14  }
```

Codice 25: andromeda - Recupero storico voti

Sviluppi futuri e conclusioni

In conclusione, la rete blockchain viene correttamente analizzata e le votazioni avvengono in modo corretto. Sono stati effettuati dei test di elezioni e di discarding di voti e tutto funziona correttamente. Attualmente è mancante la parte di login a un nodo della blockchain direttamente dall'applicazione di andromeda, quindi ora è necessario indicare a quale nodo della blockchain si ha intenzione di connettersi per effettuare la votazione direttamente dal codice. Questo tramite un sistema di login è chiaramente automatizzabile.

Bibliografia

[1] Distributed Ledger Technology: definzione e caratteristiche
blog.osservatori.net/it_it
/distributed-ledger-technology-significato

[2] Il problema dei generali Bizantini
cryptonomad.eu/il-problema-dei-generali-bizantini/

[3] Proof-of-work (PoW) — ethereum.org
ethereum.org/en/developers/docs/consensus-mechanisms/pow/

[4] Proof-of-stake (PoS) — ethereum.org
ethereum.org/en/developers/docs/consensus-mechanisms/pos/

[5] La Proof of Authority Spiegata — Binance Academy
academy.binance.com/it/articles/proof-of-authority-explained

[6] La Proof of Burn Spiegata — Binance Academy
academy.binance.com/it/articles/proof-of-burn-explained

[7] Proof of Capacity (Cryptocurrency) Definition
investopedia.com/terms/p/proof-capacity-cryptocurrency.asp

[8] Proof of Elapsed Time (PoET) (Cryptocurrency) Definition
investopedia.com/terms/p/proof-elapsed-time-
cryptocurrency.asp

[9] Ethereum Accounts — ethereum.org
ethereum.org/en/developers/docs/accounts/

[10] ECDSA
encryptionconsulting.com/education-center/what-is-ecdsa/

[11] Etherscan
etherscan.io/

[12] Cos'è ether(ETH)) — ethereum.org
ethereum.org/it/eth/

[13] ethash
https://eth.wiki/en/concepts/ethash/ethash

[14] Cos'è il gas in Ethereum? — Bit2Me Academy
academy.bit2me.com/it/cos%27%C3%A8-il-gas-in-Ethereum/

[15] Go Ethereum
geth.ethereum.org/

[16] Clique
eips.ethereum.org/EIPS/eip-225

[17] Clique API
geth.ethereum.org/docs/rpc/ns-clique

[18] Truffle
github.com/trufflesuite/truffle

Ringraziamenti

A conclusione di questo elaborato desidero menzionare tutte quelle persone che hanno contribuito alla conclusione del mio percorso universitario.

Un ringraziamento va al mio relatore Claudio Schifanella per la disponibilità, al mio tutor in azienda Alberto Ferrini per avermi guidato con entusiasmo in questi mesi attraverso questo percorso e ad Adrian e Alessandro per i consigli durante le giornate di lavoro in Consoft.

Ringrazio il mio compagno di corso Luchino con cui ho portato avanti i progetti universitari più importanti.

Grazie Teo, Giorgio e Magna per le cene del giovedì sera e le pentole costantemente oliate.
Grazie a tutto il gruppo dei Caimani di Borgo San Dalmazzo per essersi rivelati per me come una seconda famiglia. Grazie Mario, Simo, Sam, Rocco, Nico, Pette, Push, Samma, Karji, Jaco, Simi e Gabri.
Grazie Marzia per tutte le emozioni regalate.

Grazie a Nonna Marisa, Nonno Beppe, Nonna Silea e Nonno Marcello per tutto l'entusiasmo che mi avete trasmesso durante questo mio percorso, un grazie speciale a Nonno Beppe per avermi messo in mano il primo computer e avermi fatto nascere questa passione per l'informatica.
Grazie a mio fratello Fabrizio.

E per ultimi e più importanti voglio ringraziare i miei genitori, che hanno creduto in me nonostante tutte le difficoltà vissute, mi hanno appoggiato nella difficile scelta d'intraprendere questo percorso universitario e senza i quali non avrei potuto portare a termine il mio percorso di studi. Grazie Mamma e grazie Papà.

Grazie infinite a tutti voi.
Andrea.

Dichiarazione di originalità

Dichiaro di essere responsabile del contenuto dell'elaborato che presento al fine del conseguimento del titolo, di non avere plagiato in tutto o in parte il lavoro prodotto da altri e di aver citato le fonti originali in modo congruente alle normative vigenti in materia di plagio e di diritto d'autore. Sono inoltre consapevole che nel caso la mia dichiarazione risultasse mendace, potrei incorrere nelle sanzioni previste dalla legge e la mia ammissione alla prova finale potrebbe essere negata.