UNIVERSITÀ DEGLI STUDI DEL MOLISE

BASE DI DATI II

MoliseCoin

ALDO MANCO	**a.manco@studenti.unimol.it**	**159165**
FRANCESCO FERRINI	f.ferrini@studenti.unimol.it	158369

Cryptocurrency

Le cryptovalute sono un **asset digitale paritario** e **decentralizzato**.

Esse utilizzano un **sistema distribuito** P2P i cui nodi risultano costituiti da computer di utenti, situati potenzialmente in tutto il globo, dove tutti possiedono una copia del registro digitale (**Blockchain**) che contiene tutte le transazioni.

Su questi computer vengono eseguiti appositi programmi che svolgono funzioni nel ruolo di **portamonete**. Non c'è attualmente alcuna autorità centrale che le controlla.

Le transazione sono quasi impossibili da manomettere tramite appositi meccanismi che ne prevengono l'aggiornamento (**Proof-of-Work**).

Le transazioni vengono controllate e poi elaborate collettivamente dagli utenti nella rete (**Mining**), pertanto non c'è una gestione di tipo "centralizzato".

I controlli e le transazioni possono essere fatti e provati solo **crittograficamente**.

Queste proprietà uniche nel loro genere, non possono essere esplicate dai sistemi di pagamento tradizionale.

Blockchain + Proof-of-Work + Encryption + Distributed System

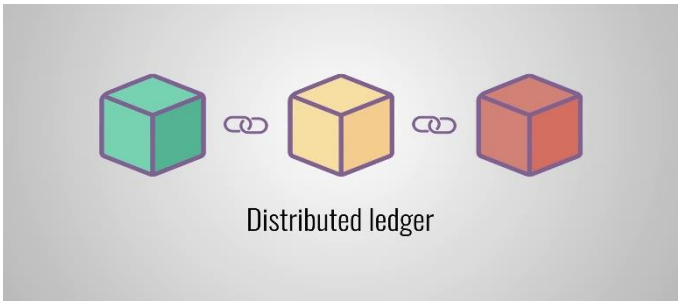
Blockchain

La blockchain è una **struttura dati condivisa** e **immutabile**.

È definita come un **registro digitale** le cui voci sono raggruppate in **blocchi** concatenati in ordine **cronologico**, la cui integrità è garantita dall'uso della **crittografia**.

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.

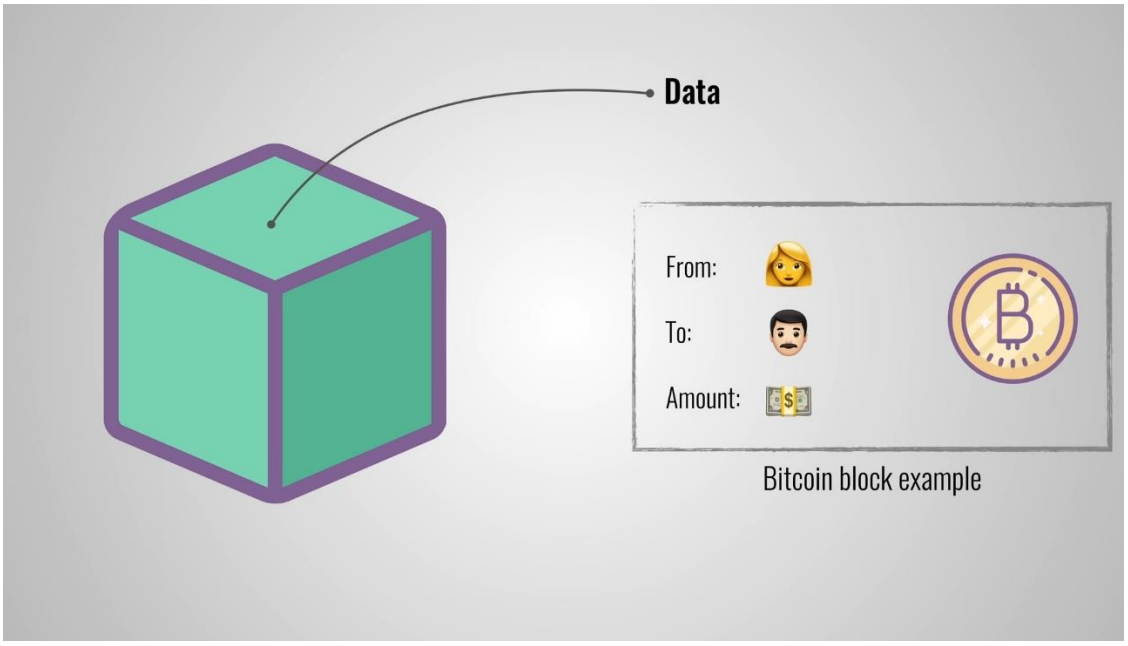
"Una Serie di Blocchi Collegati
tra Loro che contengono Informazioni."

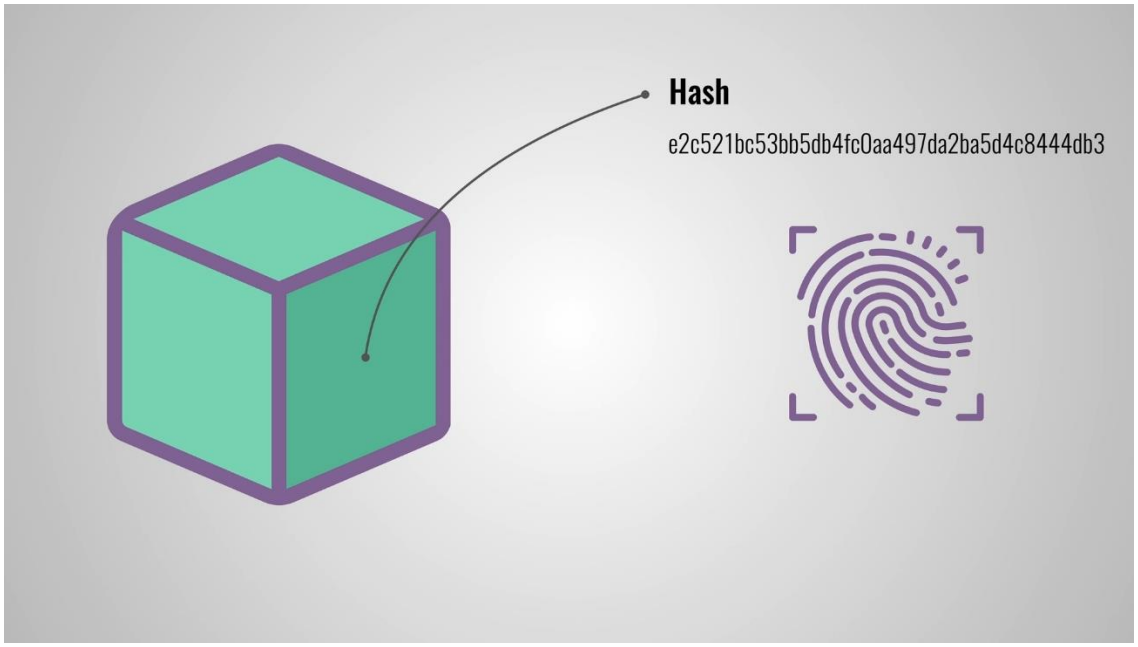


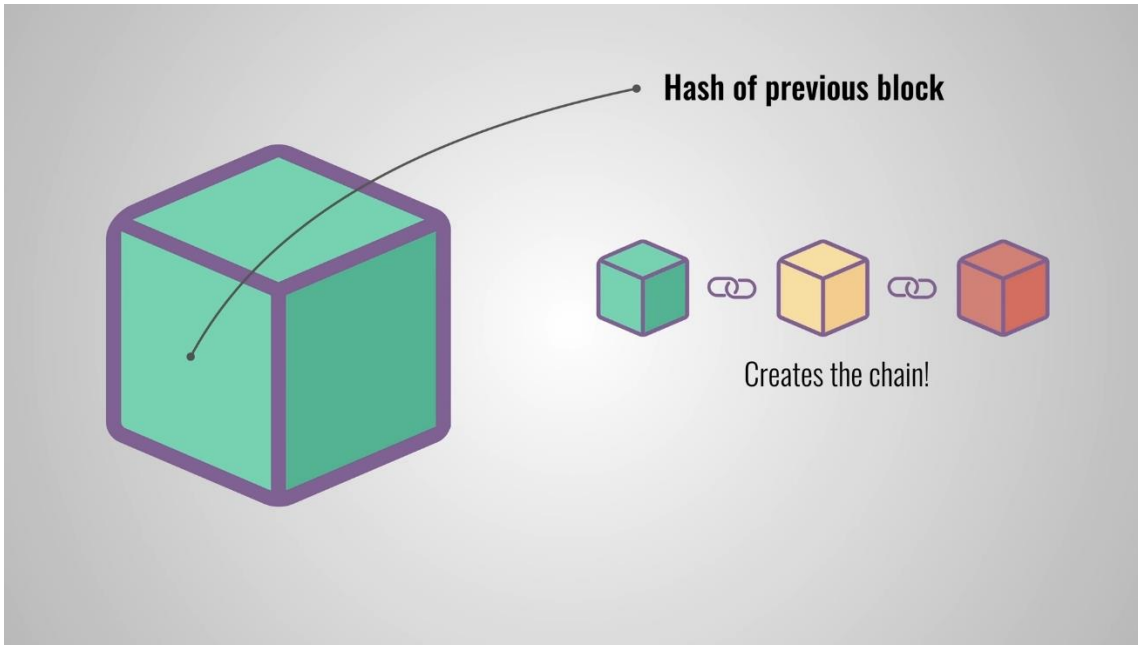
La caratteristica principale della Blockchain viene eprfettamente descritta dalla quasi impossibilità di poter manomettere la struttura. Ma questo come è possibile?

Ogni Blocco contiene:

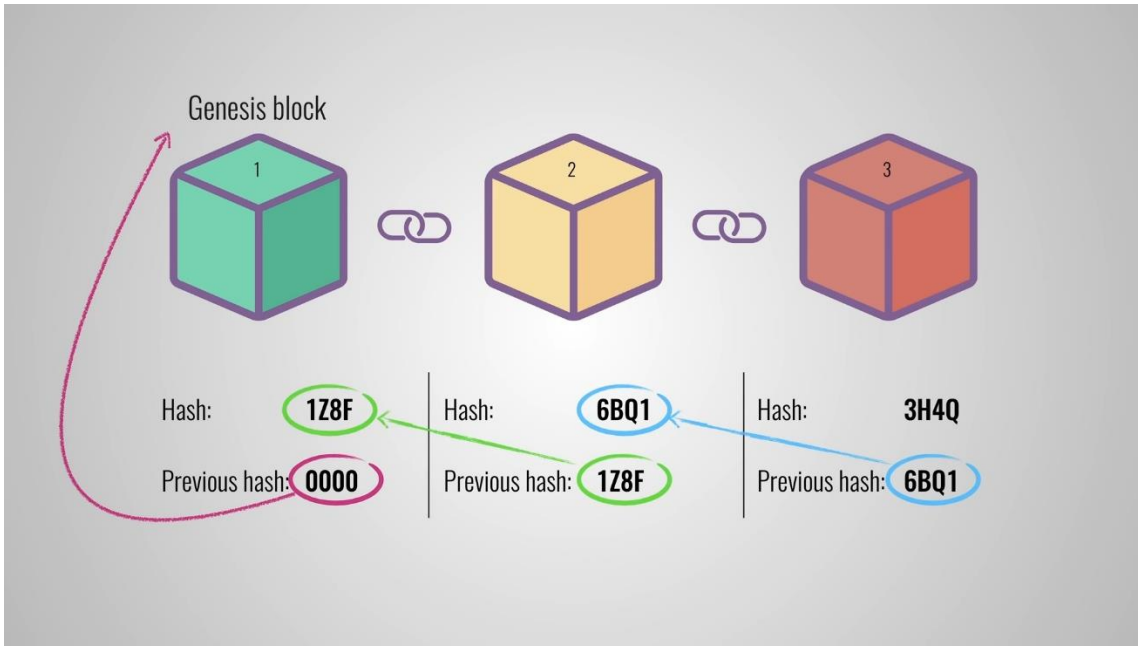
- **Dati =>** insieme di transazioni
- **Hash** => fingerprint, codice univoco generato con l'algoritmo di crittografia SHA256 il cui valore dipende da:
    - Dato contenuto al suo interno
    - Data & Ora (Timestamp) del dato
    - Hash del Blocco Precedente
- **Hash Precedente** => codice univoco del blocco creato in precedenza, come in una lista concatenata





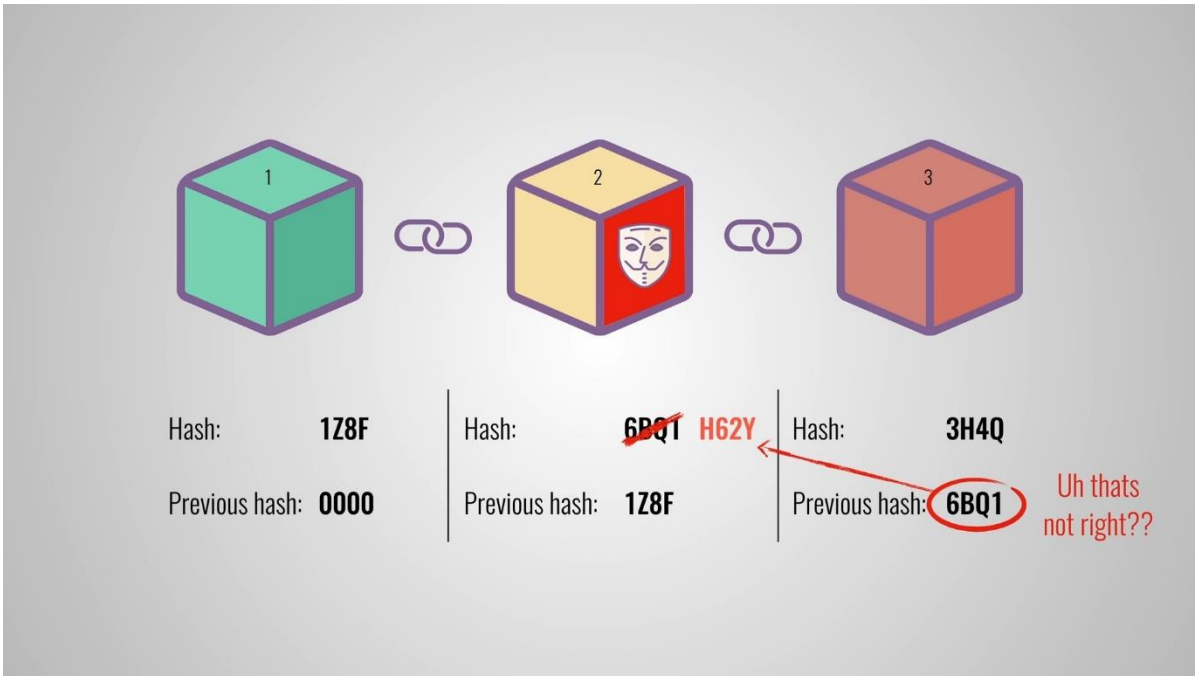


Il Primo Blocco di una Blockchain è chiamato “**Genesis Block**”



Siccome il codice **Hash** di un blocco **dipende** dal **dato** contenuto al suo interno, se noi cambiamo il dato, di conseguenza cambierà anche il codice Hash, quindi potremo pensare di cambiare il codice “Hash Precedente” del blocco successivo e la blockchain continuerebbe ad essere valida, quindi verrebbe facilmente manomessa.

Ma siccome il codice **Hash** di un blocco **dipende** anche dal codice **Hash del blocco precedente**, siamo costretti a cambiare i dati contenuti all'interno di tutti i blocchi successivi a quello che vogliamo manomettere. Visto e considerato che i computer odierni sono molto veloci e sono in grado di calcolare 100.000+ Hash/Secondo, a questo status di cose potremmo essere ancora in grado di manomettere l'intera blockchain e renderla valida.



Pertanto sarà necessario aggiungere alla nostra **blockchain** un **meccanismo** in grado di rendere impossibile, anche ai supercomputer, di manomettere la blockchain.

Proof-Of-Work

“Meccanismo che rallenta la creazione di un nuovo blocco”

Nel caso del **Bitcoin**, sono necessari **circa 10 minuti** per risolvere il **gioco matematico** imposto dalla **Proof-of-Work** necessario per **controllare l’integrità** e poter aggiungere un **nuovo blocco** alla **blockchain**.

Codice Hash:

```
04c9030c10160458880ddc0d291026473cfc56c15a3c2775d1567c395ef06dfb2a5c96b7a857ac530cbfa859f028d43bfb704d01cfcec6e44a694c7a1159e73859
```

Proof-of-Work => Nel calcolo del codice Hash, utilizzare anche una variabile “**Nonce**” il cui valore cambia finchè non viene trovato un codice Hash che ha 8 Zeri al suo inizio.

Codice Hash con Proof-of-Work:

```
000000004c9030c10160458880ddc0d291026473cfc56c15a3c2775d1567c395ef06dfb2a5c96b7a857ac530cbfa859f028d43bfb704d01cfcec6e44a694c7a115
```

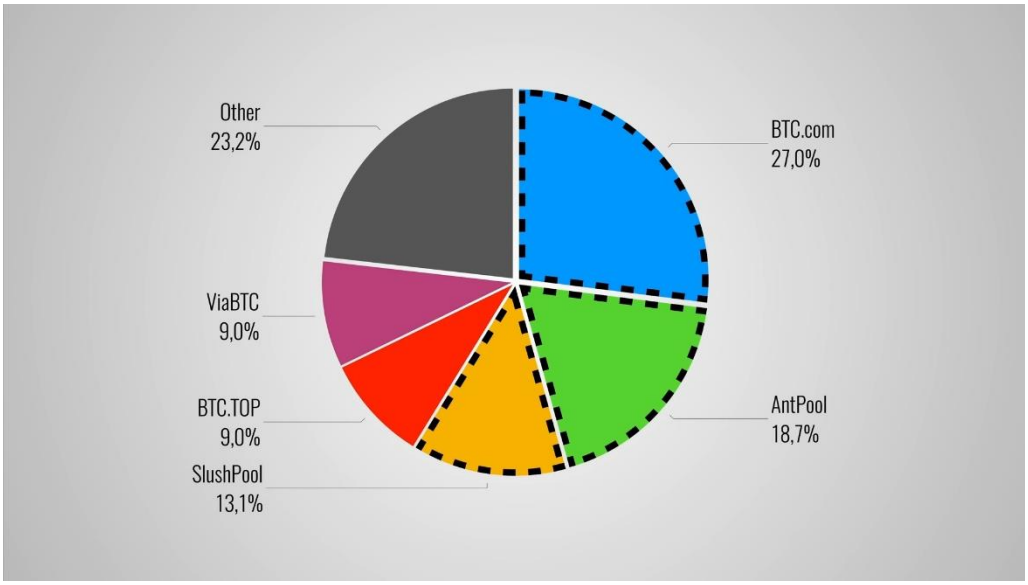
Questi giochi matematici vengono risolti da utenti chiamati “**Minatori**”, che competono tra loro, ed il 1° che trova una soluzione al gioco viene premiato con delle monete (**Miner Reward**), più gli utenti sono equipaggiati, più il Miner Reward è alto.

Ma a causa di questa competizione, i minatori stanno costruendo server farm sempre più grandi e potenti e si stima che il loro consumo di elettricità sia di 54TWh all’anno, lo stesso consumo di 5'000'000 di famiglie americane.

Inoltre più è alta la densità di Proof-of-Work calcolati, più aumentano le possibilità di essere scelti per calcolare il blocco successivo.

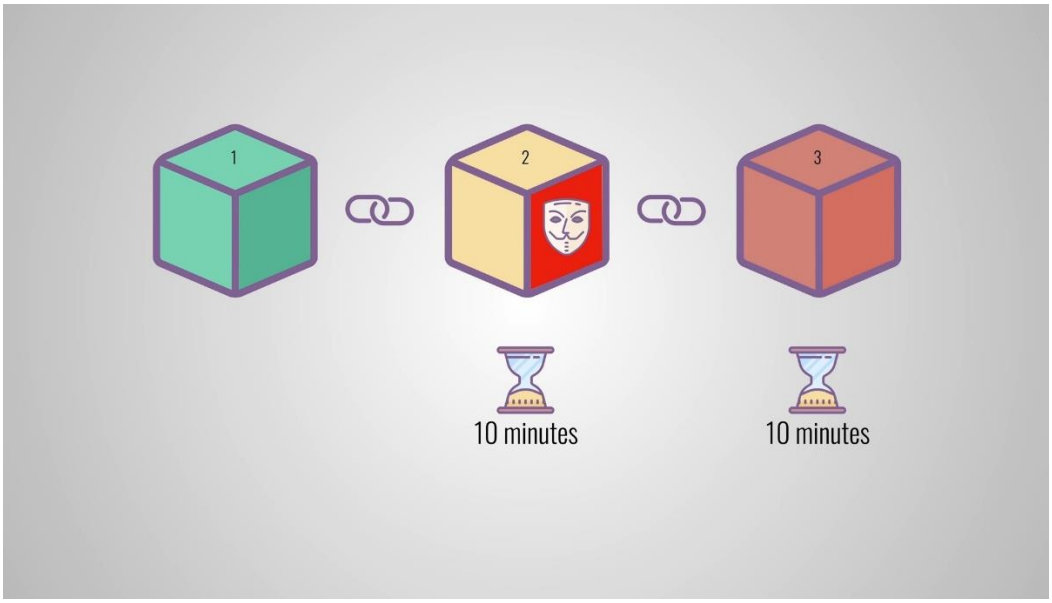
Per questo motivo i minatori più potenti si stanno alleando per creare delle “**Mining Pools**”, cioè uniscono il loro equipaggiamento per fare questi calcoli, **in modo da far aumentare la possibilità di essere scelti per il calcolo del blocco successivo**, e distribuire equamente le monete tra i membri.

Ma in questo modo il potere viene **centralizzato** in poche **Mining Pools**, invece che decentralizzato come esso è per definizione:



Se BTC.com, AntPool e SlushPool si unissero tra loro, avrebbero la maggioranza dell'equipaggiamento e potrebbero iniziare ad approvare transazione fraudolente.

Manomettere la **blockchain** significa ricalcolare la Proof-of-Work per tutti i blocchi successivi, cioè 10 minuti per ogni blocco.

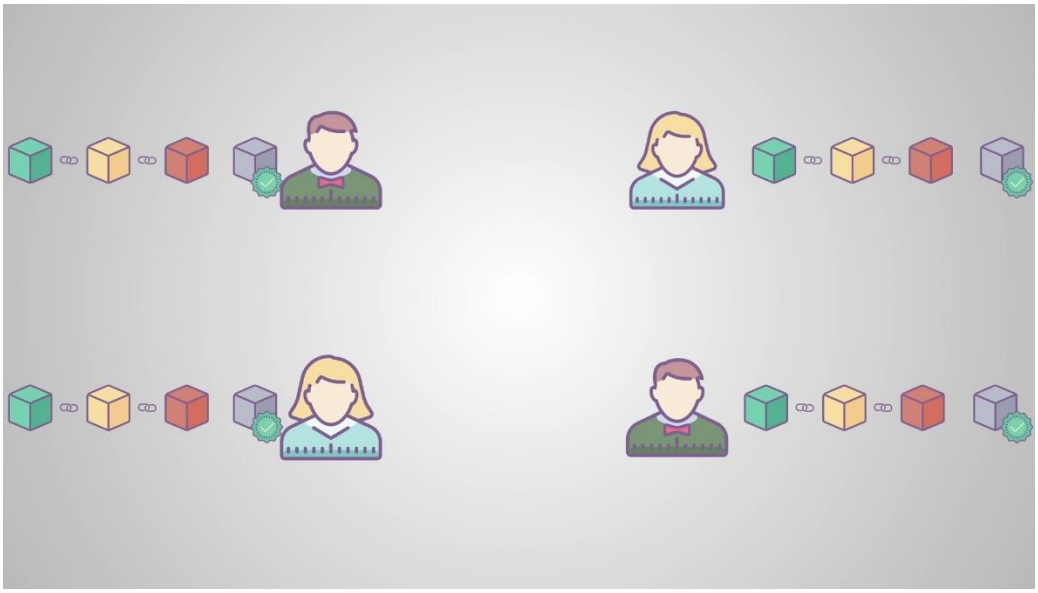


Pertanto la **sicurezza della Blockchain** dipende dall'utilizzo di **algoritmi di hashing** e da **giochi matematici che la Proof-of-Work impone al codice Hash**, che la rendono quasi impossibile da manomettere.

Distributed System

Un **sistema distribuito** consiste di un **insieme di calcolatori autonomi**, **connessi** fra loro tramite una **rete** e un **middleware di distribuzione**, che permette ai computer di **coordinare le loro attività** e di **condividere le risorse del sistema**, in modo che gli utenti percepiscano il sistema come **un unico servizio integrato di calcolo**.

Nelle **cryptovalute**, la blockchain non viene **gestita** da una sola entità centrale, ma **da tutti i computer nella rete P2P**, dove a chiunque è concesso di entrare, e quando qualcuno entra, gli viene data una copia dell'intera blockchain.



Quando qualcuno aggiunge un nuovo **blocco** alla **blockchain**, ogni **utente** riceve la nuova blockchain e **verifica** se essa è **valida** oppure no, se viene dato il **consenso** da più del **50% degli utenti della rete P2P**, allora il blocco viene **aggiunto** alla **blockchain** di tutti.

Proof-Of-Stake

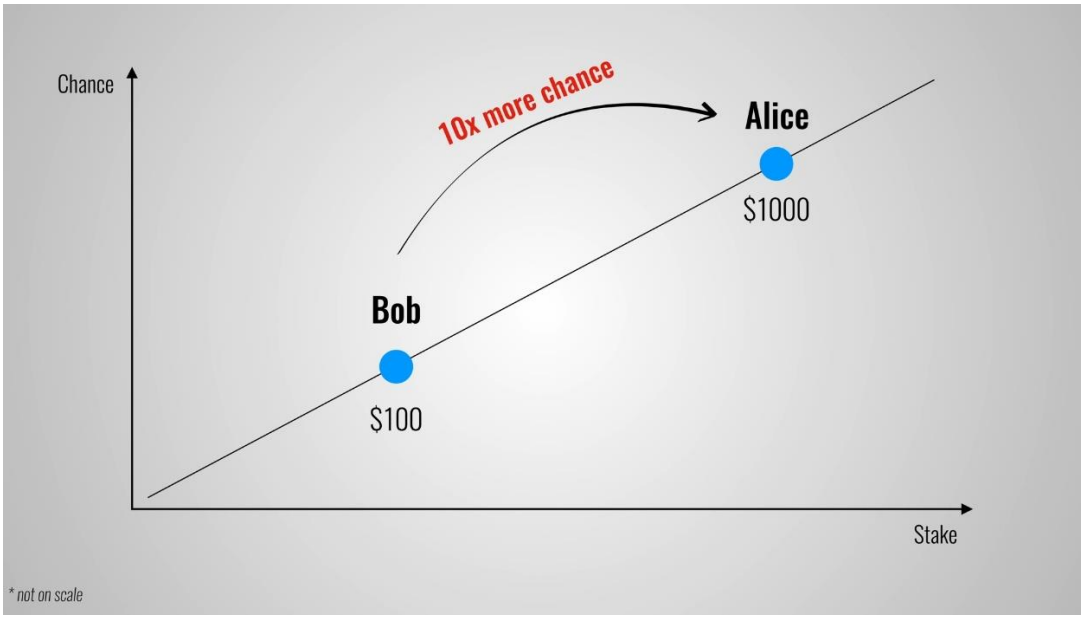
“Meccanismo che rallenta la creazione di un nuovo blocco,

più equo rispetto a Proof-of-Work”

Nel Meccanismo **Proof-of-Stake**, non abbiamo più “minatori” ma “**validatori**”, dove la possibilità di essere scelti per fare aggiungere il prossimo blocco è **casuale** ma tendente verso **chi ha più monete**, che viene riconosciuto come chi ha più capacità d’elaborazione e quindi di risolvere il quesito più velocemente.

In questo modo si favorisce chi è più **ricco**, ma è comunque meglio del sistema **Proof-of-Work** dove chi è più ricco può sfruttare **l’economia scalare**:

Mentre nel caso della **Proof-of-Stake** la curva è lineare:



E’ possibile **fidarsi** dei **validatori** perché essi **perderanno** parte delle loro **monete** se approvano **transazioni fraudolente**, inoltre la quantità di monete che essi ricevono come premio dipende da quante monete hanno, quindi perdere delle monete significa anche avere un minore guadagno.

L’utente può prelevare le sue **monete** più le **tasse guadagnate** nel ruolo di **validatore** solo dopo un determinato periodo di tempo, perchè la blockchain deve avere il tempo necessario per verificare se sono stati approvati dei blocchi fraudolenti.

Anche nella **Proof-of-Stake** c’è il problema della **centralizzazione** del potere, ma in questo caso è molto più complicato:

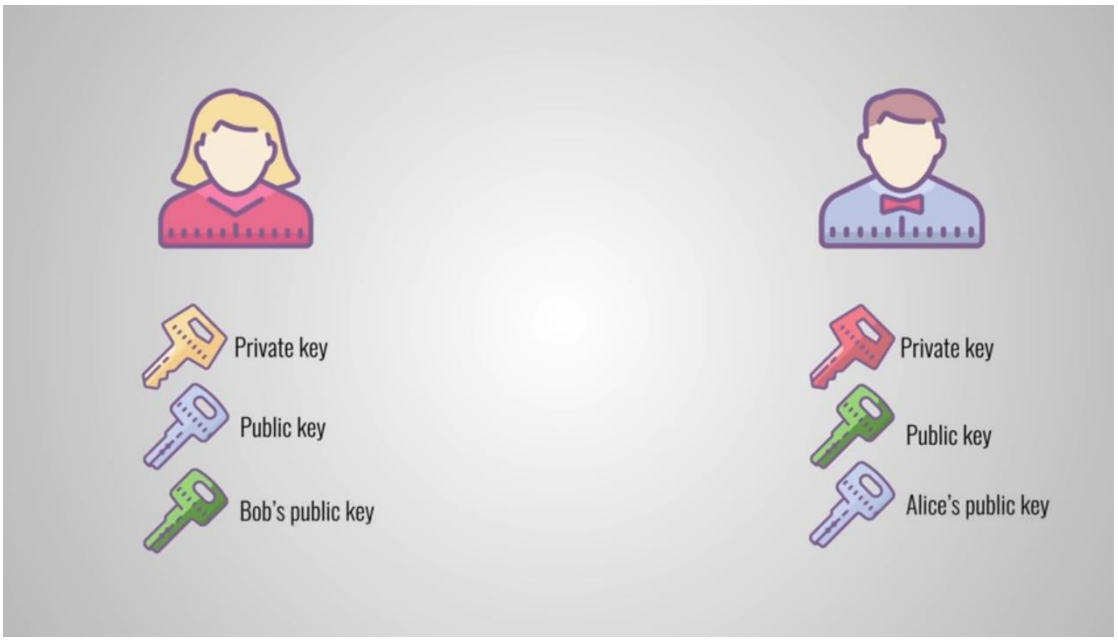
I **validatori**, per **controllare** la **blockchain**, se il **Bitcoin** usasse questo sistema, dovrebbero possedere almeno **$79 miliardi di dollari**, pertanto si evita l’agevolazione dell’economia scalare.

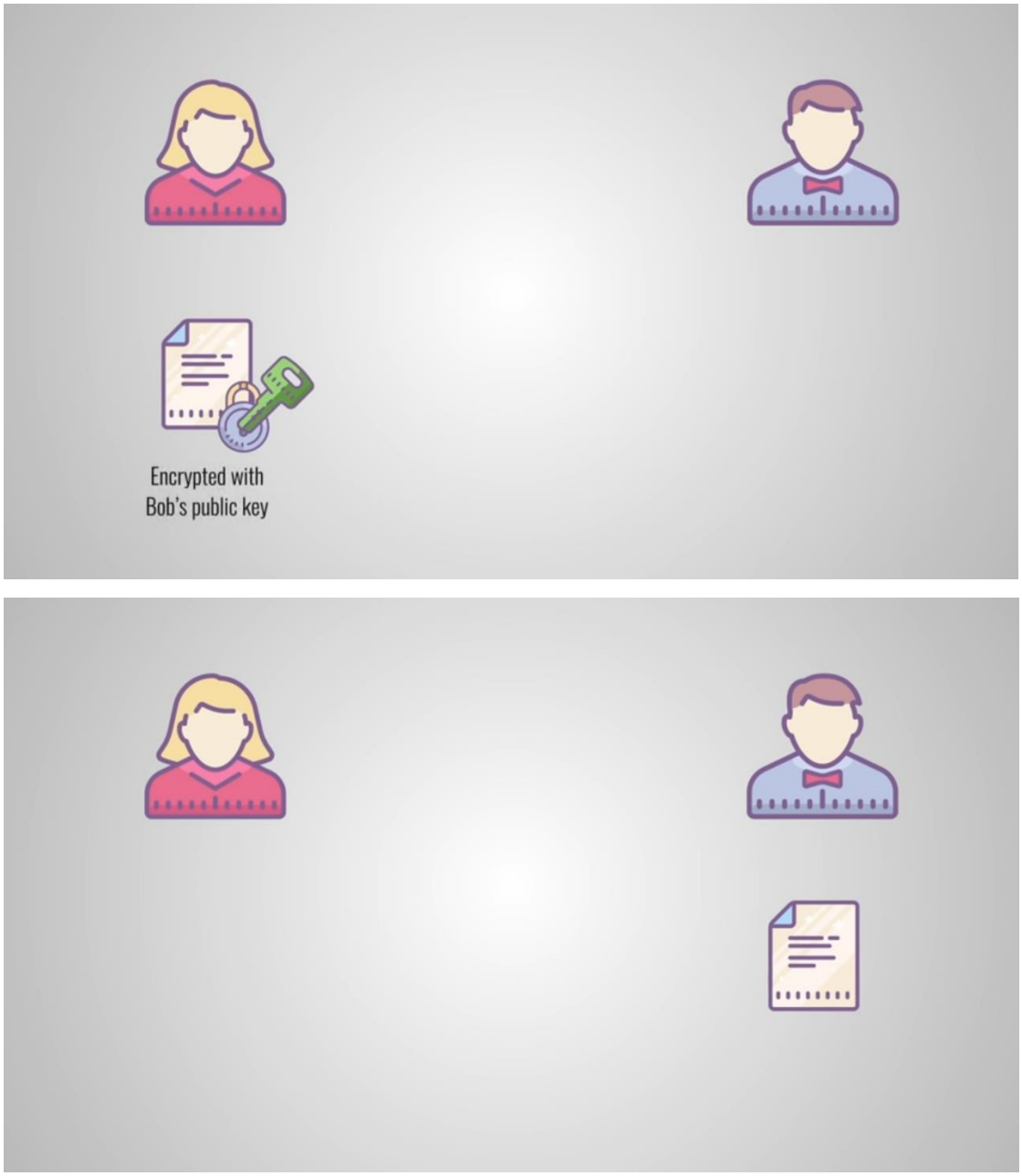
Encryption

La **crittografia asimmetrica** è un tipo di crittografia dove, ad ogni attore coinvolto nella comunicazione è associata una **coppia di chiavi generate**:

- La **chiave pubblica**, che deve essere **distribuita**;
- La **chiave privata**, appunto **personale** e **segreta**;

Il meccanismo si basa sul fatto che, le 2 chiavi sono matematematicamente correlate tra loro, e pertanto se con la chiave pubblica si codifica un messaggio, allora quest'ultimo sarà decifrato solo con la chiave privata.





Anche se sono correlate matematicamente, non è possibile derivare la chiave privata di qualcuno partendo dalla sua chiave pubblica.

Nelle **cryptovalute** viene usata la **crittografia asimmetrica** in modo che solo il **proprietario** di un **portafoglio** può **ritirare** le sue **monete** o **trasferirle**.

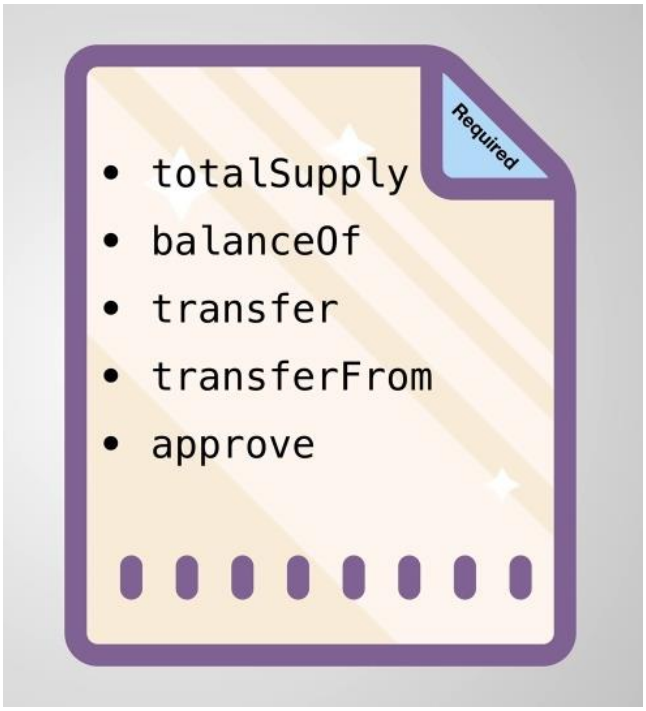
La **chiave pubblica** di un **utente** viene calcolata con **l'algoritmo di crittografia SECP256K1** che utilizza come parametro una **curva ellittica**.

La **chiave privata** di un utente, che corrisponde ad un codice **hash**, viene calcolata con **l'algoritmo di crittografia SHA256** (Secure Hash Algorithm) creato dall'NSA (National Security Agency).

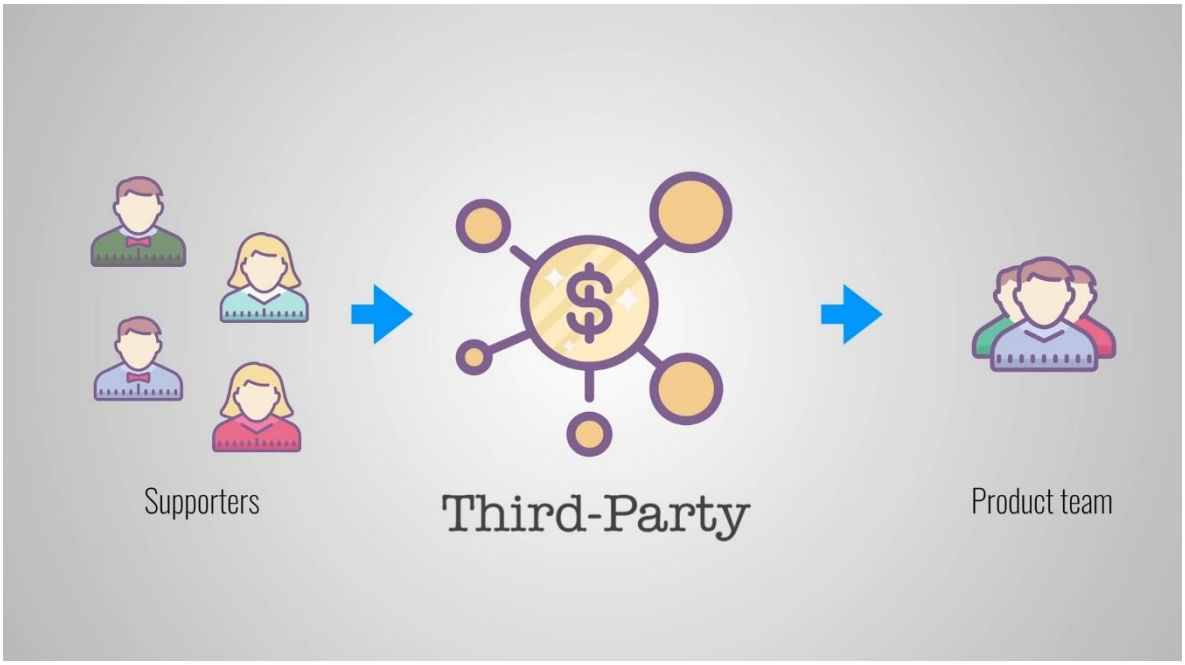
Smart Contracts

Gli Smart Contracts sono dei mini programmi completamente digitali che simulano il funzionamento dei normali contratti, essi vengono programmati dal mittente e dal destinatario, che gli impartiscono delle regole su come gestire gli asset.

Essi sono immagazzinati all'interno della blockchain.



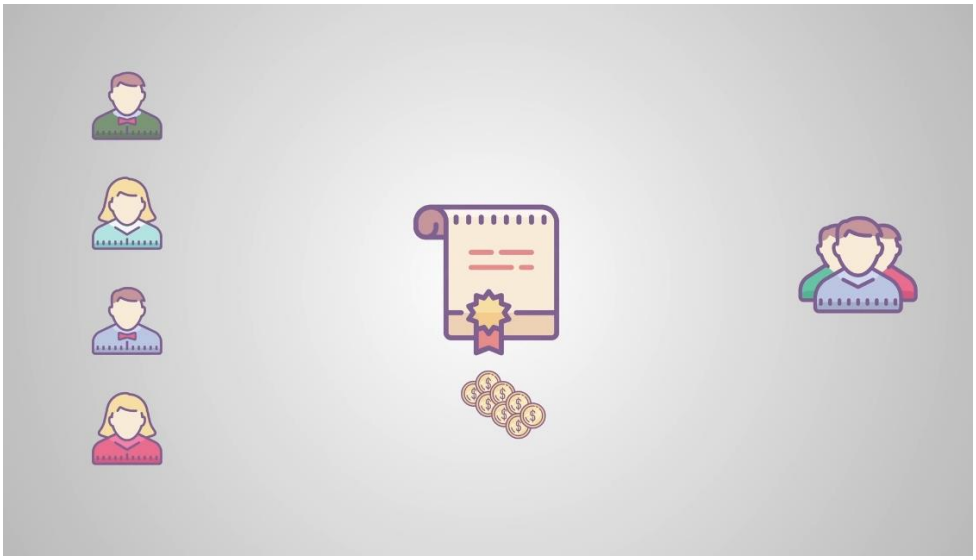
Rispetto ai contratti "analogici" essi hanno un grande vantaggio, perché fanno a meno di un ente di terze parti che gestisca uno scambio tra 2 persone.



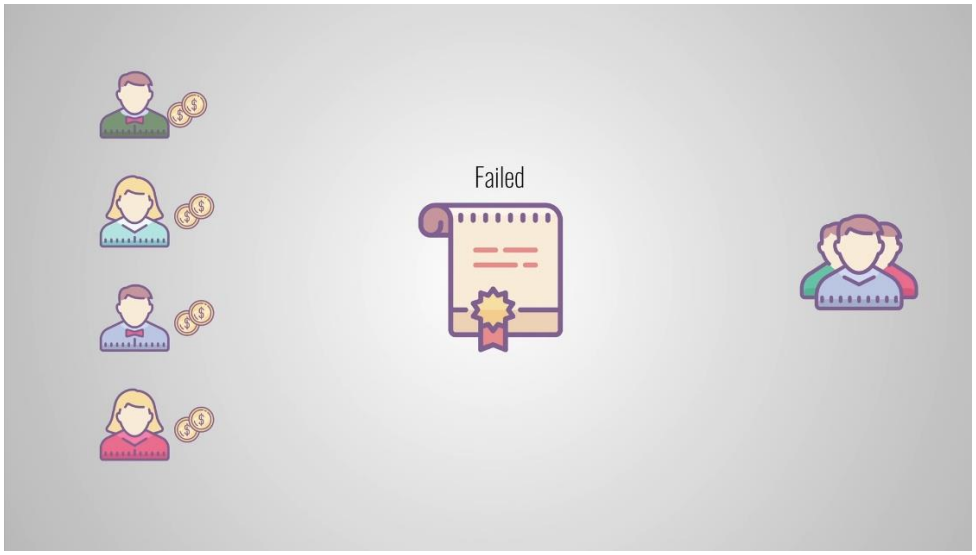
Lo scambio e le sue condizioni vengono gestite automaticamente dal mini programma, di cui possiamo fidarci perché, essendo immagazzinato all'interno della blockchain ha per definizione le seguenti caratteristiche:

- Distribuito => una volta programmato il contratto ed aggiunto alla blockchain, tutti gli utenti ne vengono in possesso, e ne validano l'integrità, e se qualcuno prova a manomettere il contratto, viene bocciata la modifica e lasciato allo stato iniziale.
  In questo modo nessuno è in controllo delle monete finché il contratto non decide, in base alle istruzioni che gli sono state date, di sbloccare il denaro a favore di qualcuno.
- Immutabile => una volta programmato il contratto, esso non può essere manomesso, ne per volontà, ne per imbroglio, da nessuno

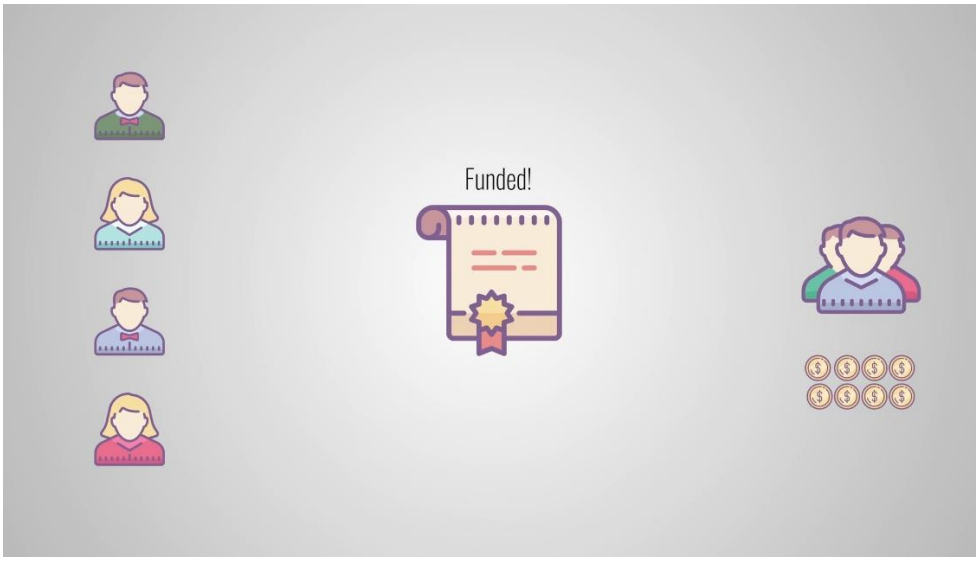
I committenti ed il professionista programmano uno "Smart Contract", che al verificarsi di determinate condizioni decide se dare le monete al professionista o restituirle ai committenti:



Il servizio richiesto al professionista non viene completato e le monete vengono restituite al committente:



Il servizio richiesto al professionista viene completato e le monete vengono assegnate al professionista:



MoliseCoin - Back-End – JavaScript

Transaction Class

```
/* Importiamo la libreria SHA256 contenente i metodi per l'utilizzo dell'algoritmo di crittografia
*/

const SHA256 = require('crypto-js/sha256');

/* Importiamo la libreria elliptic e l'algoritmo SECP256K1
*/

const EC = require('elliptic').ec;
const cryptoAlgorithm = new EC('secp256k1');

/* Creamo una classe "Transazione" per descriverne gli attributi e le funzionalitÃ  attuabili al fine dell'utilizzo della crittovaluta
*/

class Transaction{

    /* Attributi necessari per fare una transazione
    */

    constructor(fromAddress, toAddress, amount){
        this.fromAddress = fromAddress;
        this.toAddress = toAddress;
        this.amount = amount;
    }

    /* Generiamo un codice Hash in base agli attributi della transazione, esso sarÃ  la nostra chiave privata
    */

    calculateHash(){
        return SHA256(this.fromAddress + this.toAddress + this.amount).toString;
    }

    /* Certifica la transazione, prende come parametro in input l'indirizzo del mittente, se esso Ã¨ uguale all'indirizzo dell'utente che sta cercando di fare la transazione, allora calcolo il codice Hash in base ai dati della transazione, e genero la "signature" (chiave privata) univoca della transazione in base esadecimale.
    */

    signTransaction(signingKey){

        if(this.fromAddress !== signingKey.getPublic('hex')){
            throw new Error("Non puoi firmare la transazione di un altro portafoglio!");
        }

        const hashTransaction = this.calculateHash();
        const rawSignature = signingKey.sign(hashTransaction, 'base64');
        this.signature = rawSignature.toDER('hex');
    }
```

```
    /* Per verificare che la transazione sia valida, se essa non ha un mittente allora Ã¨ valida perchÃ© si tratta di un Mining Reward, altrimenti Ã¨ necessario verificare che ci sia una firma e che essa sia valida, cioÃ¨ che la chiave pubblica generata con i dati del mittente e la chiave privata generata con i dati della transazione generino la firma corretta.
    */


    isValid(){

        if(this.fromAddress==null){
            return true;
        }

        if(!this.signature || this.signature.length===0){
            throw new Error('Nessuna firma trovata in questa transazione');
        }

        const publicKey = cryptoAlgorithm.keyFromPublic(this.fromAddress, 'hex');
        return publicKey.verify(this.calculateHash, this.signature);
    }
}
```

Block Class

```
/* Creamo una classe "Blocco" per descriverne gli attributi e le funzionalitÃ  attuabili al fine dell'utilizzo della crittovaluta
*/

class Block{

  /* Ogni blocco ha una data di creazione, un insieme di transazioni, un codice hash univoco, il codice hash del blocco precedente (tale che vi sia un collegamento tra tutti i blocchi della blockchain) ed un numero "nonce", utile comr variabile al "gioco matematico" della Proof-of-Work
  */

    constructor(timestamp, transactions, previousHash=''){
        this.timestamp = timestamp;
        this.transactions = transactions;
        this.previousHash = previousHash;
        this.hash = this.calculateHash();
        this.nonce = 0;
    }

    /* Genera un codice hash univoco con gli attributi del blocco e serializzando l'array di transazioni
    */

    calculateHash(){
        return SHA256(this.timestamp + this.previousHash + JSON.stringify(this.transactions) + this.nonce).toString();
    }
```

```
    /* Gioco matematico nel quale devo continuare a generare un codice hash, facendo variare la variabile "nonce", finchÃ© non trovo un codice che abbiamo almeno (hardness) zeri all'inizio del codice
    */

    mineBlock(hardness){

        while(this.hash.substring(0, hardness) !== Array(hardness+1).join("0")){
            this.nonce++;
            this.hash = this.calculateHash();
        }
    }

    /* Controlla che tutte le transazioni all'interno del blocco siano state firmate e che la loro firma sia valida
    */

    hasValidTransactions(){

        for (const transaction of this.transactions) {

            if(!transaction.isValid){
                return false;
            }
        }

        return true;
    }
}
```

Blockchain Class

```
/* Creamo una classe "Blockchain" per descriverne gli attributi e le funzionalitÃ  attuabili al fine dell'utilizzo della crittovaluta
*/

class Blockchain{

  /* La blockchain ha un insieme di blocchi, un insieme di transazioni pendenti che saranno inserite nel blocco successivo quando ci sarÃ  un minatore disponibile a certificare le transazioni, una variabile hardness che definisce il numero di zeri che il codice hash della transazione deve avere al suo inizio, e il "premio", cioÃ¨ il numero di monete che devono essere date all'utente che ha impiegato le sue risorse hardware al fine di certificare le transazioni del nuovo blocco
  */

    constructor(){
        this.chain = [this.createGenesisBlock()];
        this.hardness = 3;
        this.pendingTransactions = [];
        this.miningReward = 100;
    }

    /* Creazione del primo blocco (genesis block) con degli attributi hard-coded
    */

    createGenesisBlock(){
        return new Block("26/03/2020", "Genesis Block", "0");
    }
```

```
    /* Restituisce l'ultimo blocco certificato all'interno della blockchain
    */

    getLatestBlock(){
        return this.chain[this.chain.length-1];
    }

    /* Viene eseguita quando c'Ã¨ un minatore pronto per certificare delle transazioni,
crea un array di transazioni dove inserisce il Mining Reward (premio in monete per il
minatore da nessun mittente, cioÃ¨ dal sistema) e tutte le transazioni pendenti della
blockchain, cioÃ¨ quelle fatte dopo la creazione dell'ultimo blocco.
      Viene creato un nuovo blocco che contiene l'orario attuale, l'array delle
transazioni e l'hash dell'ultimo blocco della blockchain.
      Poi il blocco viene certificato risolvendo un gioco matematico dove Ã¨ necessario
rigenerare il codice hash cambiando il valore della variabile "nonce" finchÃ© non
abbiamo (hardness) zeri all'inizio di esso.
      Infine svuoto l'array delle transazioni pendenti della blockchain.
    */

    minePendingTransactions(miningRewardAddress){

        const rewardTransaction = new Transaction(null, miningRewardAddress,
this.miningReward);
        this.pendingTransactions.push(rewardTransaction);

        let block = new Block(Date.now(), this.pendingTransactions,
this.getLatestBlock().hash);
        block.mineBlock(this.hardness);

        console.log('block successfully mined!');
        this.chain.push(block);

        this.pendingTransactions = [];
    }

    /* Se la transazione che si vuole aggiungere a quelle pendenti non ha un indirizzo
di partenza e uno di destinazione, oppure non ha una firma valida allora restituisce un
errore.
      Altrimenti aggiunge la transazione all'array delle transazioni pendenti della
blockchain.
    */

    addTransaction(transaction){

        if(!transaction.fromAddress || !transaction.toAddress){
            throw new Error("La transazione deve includere l’indirizzo del mittente e
del destinatario");
        }

        if(!transaction.isValid){
            throw new Error("Impossibile aggiungere una transazione non valida");
        }

        this.pendingTransactions.push(transaction);
    }
```

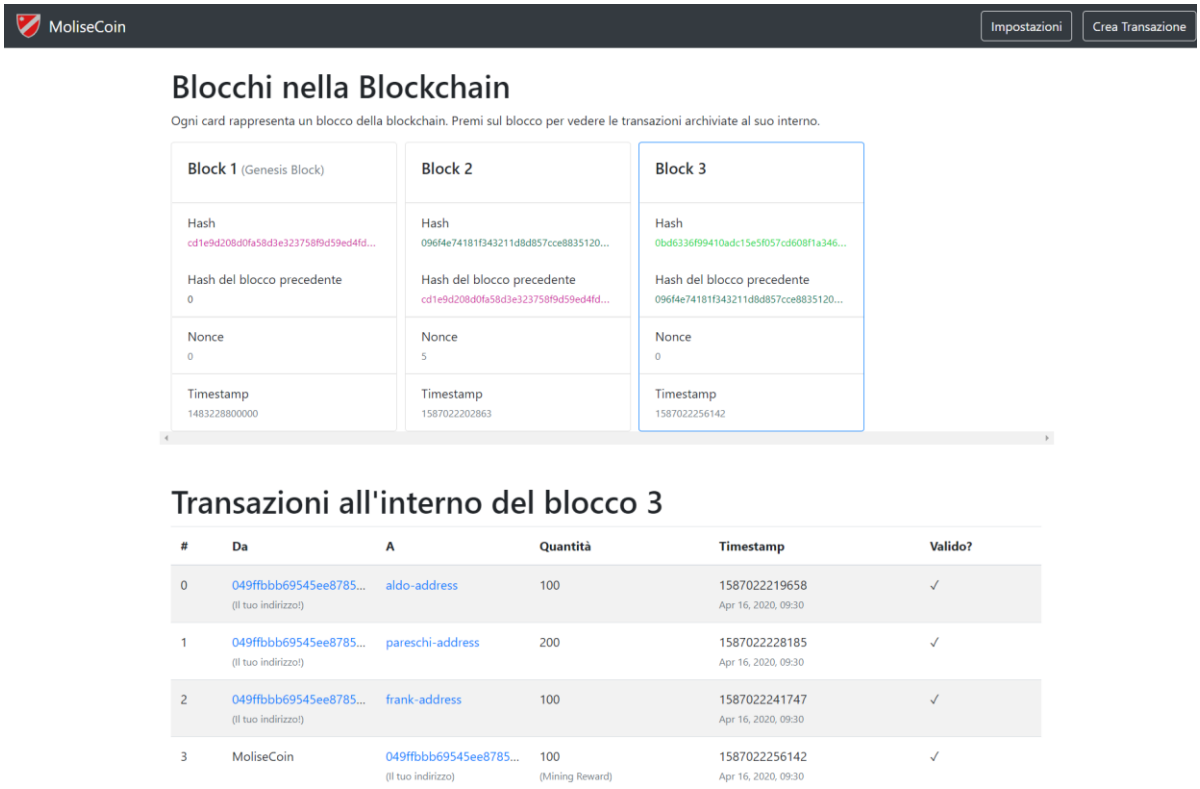
```
    /* Nelle criptovalute non esistono i profili degli utenti, ma ognuno ha un
indirizzo al quale posso essere accreditate delle monete, e per vedere quante monete
uno di essi ha Ã¨ necessario rileggere le transazioni di tutti blocchi e fare la somma
di tutte le entrate e le uscite, in questo modo Ã¨ impossibile manomettere la
blockchain.
    */

    getBalanceOfAddress(address){

        let balance = 0;

        for (const block of this.chain) {

            for (const transaction of block.transactions) {

                if(transaction.fromAddress === address){
                    balance-=transaction.amount;
                    console.log(3);
                }

                if(transaction.toAddress === address){
                    balance+=transaction.amount;
                    console.log(3);
                }
            }
        }
        return balance;
    }

    /* Per verificare che la blockchain non sia stata manomessa, Ã¨ necessario scorrere
tutti i blocchi e verificare che il blocco corrente abbia tutte transazioni valide,
verificare che esse non siano state cambiate, quindi che il codice hash sia lo stesso,
e che il previousHash sia uguale all'hash del blocco precedente.
    */

    isChainValid(){

        for (let i = 1; i < this.chain.length; i++) {

            const currentBlock = this.chain[i];
            const previousBlock = this.chain[i-1];

            if(!currentBlock.hasValidTransactions()){
                return false;
            }

            if(currentBlock.hash !== currentBlock.calculateHash()){
                return false;
            }

            if(currentBlock.previousHash !== previousBlock.hash){
                return false;
            }
        }
        return true;
    }
}

module.exports.Blockchain = Blockchain;
module.exports.Transaction = Transaction;
```

MoliseCoin - Front-End - Angular

1. Blockchain

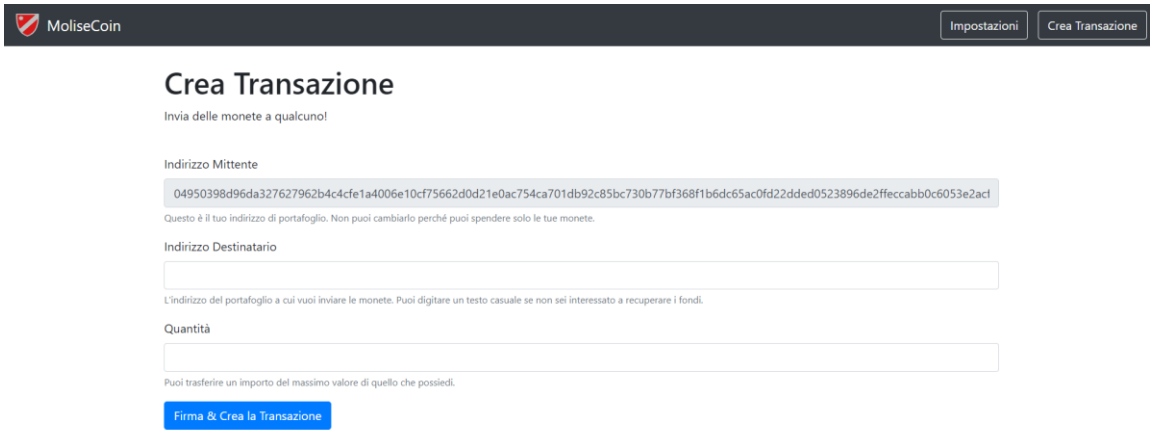
Essa mostra tutti i blocchi della blockchain e selezionando uno di essi vengono mostrate in basso tutte le transazioni che contiene.

(il colore del testo del codice hash dei blocchi, è definito dalle prime cifre del codice esadecimale hash del blocco, dopo gli zeri della Proof-of-Work)



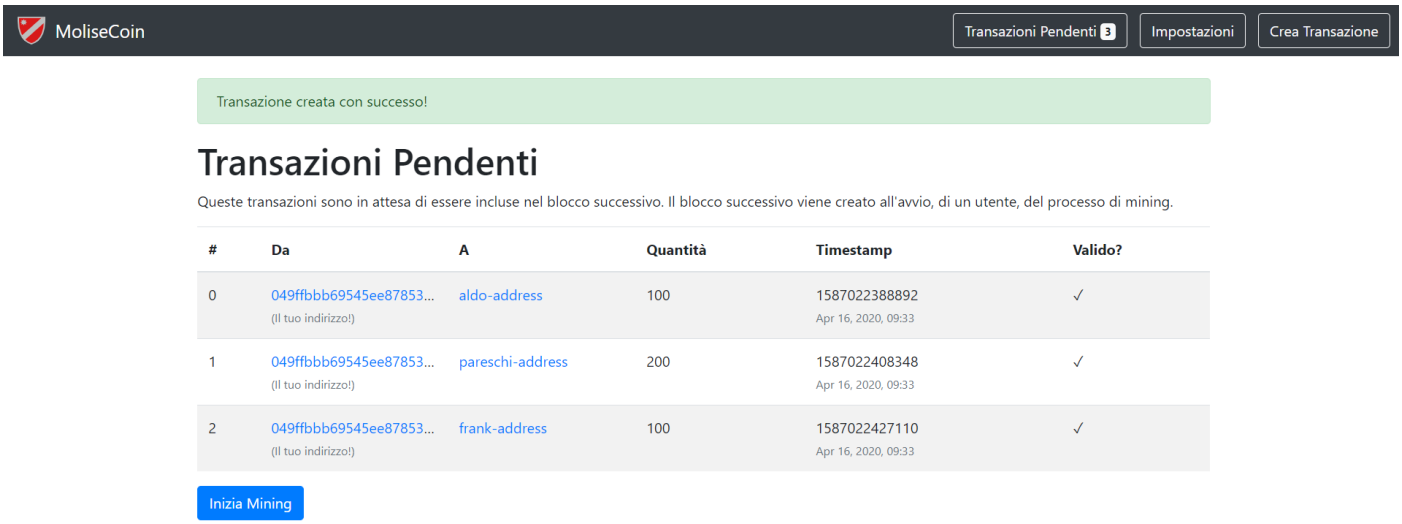
2. Crea Transazione

Prende in input i dati della transazione, se sono validi, aggiunge la transazione alle transazioni pendenti della blockchain.



3. Transazioni Pendenti

Mostra la lista delle transazioni pendenti e fornisce all'utente corrente la possibilità di minarle e quindi di creare un nuovo blocco.



4. Impostazioni

Creata solo a scopo di didattico, fornisce la possibilità di cambiare la difficoltà della Proof-Of-Work e il numero di monete del Mining Reward.