DECENTRALIZATION

In ambito blockchain, con decentralizzazione si riferisce al trasferimento del controllo e del processo decisionale da un'entità centralizzata (individuo, organizzazione o gruppo) a una rete distribuita. Le reti decentralizzate si sforzano di ridurre il livello di fiducia che i partecipanti devono riporre l'uno nell'altro, di avere un sistema di regole chiare e definite e un sistema di incentivi che spinga i vari attori del network a comportarsi correttamente.
La maggior parte dei sistemi finanziari e governativi nel mondo tradizionale sono centralizzati, il che significa che sono controllati e gestiti da un'unica autorità.
Ci sono diversi svantaggi in questo approccio, derivanti dal fatto che qualsiasi autorità centrale è un *single point of failure* : qualsiasi malfunzionamento ai vertici della gerarchia, anche non intenzionale o deliberato, ha un importante effetto negativo sull'intero sistema. Inoltre, la singola autorità centrale può introdurre censura a sua discrezione.
La centralizzazione può essere un problema anche a livello tecnico, infatti l'hacking di uno dei principali elementi di un'entità centralizzata, come un database, potrebbe compromettere l'intero network.
La decentralizzazione invece distribuisce il potere e le informazioni su più punti, rendendo la rete più resiliente ad attacchi informatici e consentendo una governance decentralizzata. Inoltre, una rete ben decentralizzata, come Bitcoin, è anche permissionless, chiunque nel mondo può infatti unirsi al network, come holder, miner o con un nodo, senza ricevere il permesso da nessuno e senza poterne essere a priori escluso.

CONSENSUS

Con consenso ci riferiamo ad un meccanismo che porta ogni attore di un network decentralizzato (come una blockchain) a concordare sull’attuale stato del network stesso. Ad esempio su quali transazioni siano avvenute in passato e i saldi dei vari account. E’ necessario che questo consenso venga raggiunto in maniera efficiente, onesta, affidabile e nel minor tempo possibile. Nel corso degli anni si sono sviluppati diversi meccanismi di consenso, i cui due più famosi sono la Proof of Work e la Proof of Stake. Altri esempi sono la Proof of Capacity o la Proof of Authority

HASH, MERKLE TREE

Hash: Ci riferiamo a una funzione di hash. Una famiglia di funzioni del tipo H(a) = h e caratterizzata da diverse proprietà

1. Hiding property: H è una *one-way function,* dato h non è possibile ricostruire a
2. Collision resistance: cambiando solo leggermente *a*, *h* cambia completamente. Per questo motivo è sostanzialmente impossibile trovare a≠b tali che *H(a) = H(b)*. Gli hash sono quindi "unici"
3. Puzzle friendliness: dato *a* e un numero qualsiasi *k*, non è possibile trovare analiticamente *x* tale che *H(k x) = h.* Questo viene usato nella proof of work di Bitcoin, dove k contiene le informazioni sul nuovo blocco e il blocco precedente, mentre h è un numero con una determinata quantità di 0 davanti. Un miner riesce ad aggiungere il proprio blocco alla blockchain e ricevere la block reward quando trova, a tentativi, una delle x corrette

The hash function used in Bitcoin is sha-256.

Merkle Tree: è una rappresentazione compatta di diverse transazioni, i quali hash sono combinati a due a due fino ad avere un solo hash, il merkle root. L'utilizzo di questa tecnica comporta due principali vantaggi:

1. Sfruttando il Merkle Tree e la sua struttura ad albero sono in grado di verificare se una transazione appartiene a un certo blocco con minori risorse computazionali rispetto all'ispezione di tutte le transazioni
2. Durante le operazioni di mining. Infatti il miner può applicare la funzione di hash non su tutta la lista di transazioni bensì sul suo solo Merkle root, velocizzando le operazioni

SMART CONTRACTS, ETHEREUM, DAPP, ERC20

Smart contract: Uno Smart Contract è un programma che viene eseguito direttamente sulla blockchain. Si tratta di una raccolta di codice e dati che è identificata da un indirizzo specifico sulla blockchain.

Gli smart contract quindi sono un tipo di account. Sono dotati di un saldo e possono inviare e ricevere transazioni. Non sono però controllati da un entità fisica, ma agiscono secondo istruzioni precise decise dagli sviluppatori del contract stesso. Gli account degli utenti possono interagire con uno Smart Contract eseguendo le funzioni predefinite. Gli smart contract, una volta creati, non sono eliminabili e le interazioni con

essi sono irreversibili. Sia la creazione sia l'interazione con uno smart contract richiedono il pagamento di una fee, che su Ethereum viene pagata in ETH.

Esempi di applicazioni sono le ICOs oppure i Decentralized Exchanges come Uniswap.

Ethereum: Ethereum è una blockchain pubblica e open-source che consente l'implementazione di smart contracts e applicazioni decentralizzate (dApp). L'idea è creare un 'World Computer' che rimpiazzi l'attuale architettura basata su server utilizzando migliaia di nodi mantenuti e localizzati in varie parti del mondo in modo decentralizzato

La sua valuta di riferimento è ETH. Viene utilizzata come reward per i miners (come BTC) e per il pagamento delle commissioni sulla rete.

Dapp: Le Dapp (decentralized applications) hanno un funzionamento simile alle tradizionali applicazioni per computer, ma con la sostanziale differenza di essere decentralizzate: non si trovano su un unico server ma distribuite su una blockchain. Il loro funzionamento si basa sugli smart contract. Questo porta alcuni vantaggi, come la mancanza di un singolo point of failure, l'immutabilità e la sicurezza dei propri dati. In compenso, sono più complesse e difficili da sviluppare e anche il debug è più impegnativo.

ERC20: Ci riferiamo con ERC20 a un protocollo per la creazione di token sulla blockchain di Ethereum. I token vengono creati tramite uno smart contract e vengono definiti ERC20 quando implementano una serie di funzioni di base, come l'invio di tokens, la richiesta della supply totale esistente o la ricerca del saldo di qualsiasi wallet. I token ERC20 sono fungibili, cioè le singole unità di cui sono composti sono intercambiabili e indistinguibili tra loro. L'esistenza di uno standard come ERC20 porta alcuni vantaggi: rendono più semplice e uniforme il lavoro degli sviluppatori, sono semplici, sicuri veloci ed economici da creare.

Sono stati e sono largamente utilizzati per ottenere fondi tramite cui sviluppare il proprio progetto, ad esempio tramite ICOs.

Alcuni ERC20 tokens molto famosi sono USDT, LINK e MKR

DIFFERENCE ETHEREUM-BITCOIN

Una prima differenza è nel fatto che Bitcoin vuole essere principalmente un mezzo per scambiare valore tra utenti (ed eventualmente per conservare valore), mentre Ethereum è una piattaforma che consente l'implementazione di smart contracts e di applicazioni decentralizzate (dApp)
Una delle differenze principali è che le transazioni sulla rete Ethereum possono contenere codice eseguibile con cui gli utenti possono interagire, mentre su Bitcoin le transazioni rappresentano solo lo storico dei movimenti e non ci si può interagire.
Al momento, entrambi utilizzano la Proof of Work per la validazione dei blocchi, ma Ethereum sta implementando un meccanismo alternativo basato su una Proof of Stake, compiendo una scelta diversa da Bitcoin

HARD FORK

Poiché Bitcoin è una rete decentralizzata, i partecipanti alla rete devono concordare delle regole per convalidare le transazioni, al fine di raggiungere un consenso sulle quantità di BTC possedute da ogni address. Questo si traduce in una singola catena di dati (blocchi) verificati e che tutti concordano essere corretti.

Talvolta, invece, la blockchain subisce una biforcazione (fork). In particolare ci si riferisce con Hard Fork ad un biforcamento dovuto all' aggiornamento del software e che richiede che tutti i nodi della rete effettuino l'upgrade per continuare ad interagire con la blockchain.

Un hard fork può essere pianificato, come l' hard fork Byzantium su Ethereum, volto a migliorare la privacy e la scalabilità ed accettato all'unanimità dalla rete; può essere dovuto ad un bug critico che ha creato molti danni, come nel caso di The Dao dove più del 10% di tutti gli ETH esistenti furono rubati; oppure da una differenza di vedute e una conseguente divisione della community che decide di perseguire progetti diversi, come nel caso di Bitcoin Cash.

ELLIPTIC CURVES AND USAGE

Elliptic curves:

Le curve ellittiche sono utili per le loro diverse proprietà. In primo luogo, le curve ellittiche sono gruppi. I gruppi sono entità matematiche con le seguenti proprietà:

1. Chiusura: se a e b sono in un gruppo G, allora a + b è nel gruppo G
2. Associatività: (a + b) + c = a + (b + c)
3. Elemento Identità: a + 0 = 0 + a = a
4. Per ogni a esiste b tale che a + b = 0

Inoltre, le curve ellittiche hanno alcune loro proprietà specifiche:

1. L'elemento identità è il punto all'infinito, 0
2. L'inverso del punto P è quello simmetrico rispetto all'asse x
3. L'addizione è definita come: dati tre punti allineati su una retta secante la curva, diversi da zero, P, Q e R hai P + Q + R = 0. L'ordine non ha importanza per questi tre punti. Nel caso in cui la retta sia tangente in P e passi per Q, ho 2P + Q = 0.

Con 3. , ottengo che P+Q = -R e posso sommare punti. Si noti inoltre che R è la riflessione sull'asse x di R.

Their equation is $y^2 = x^3 + a * x + b$

Utilizzo: ECDSA

ECSDA ("Elliptical Curve Digital Signature Algorithm") è la crittografia dietro le chiavi private e pubbliche utilizzate in Bitcoin. Consiste nel combinare la matematica dietro i campi finiti e le curve ellittiche per creare equazioni unidirezionali, il che significa che puoi scegliere la tua chiave privata e calcolare facilmente la tua chiave pubblica. Tuttavia, non posso prendere la mia chiave pubblica (o quella di chiunque altro) e calcolare facilmente la loro chiave privata. Nel caso specifico, per Bitcoin ci vorrebbero trilioni di anni di continui tentativi di indovinare diverse chiavi private per capire quale crea una determinata chiave pubblica.

Per il protocollo Bitcoin si è scelto di usare lo standard secp256k1, i cui parametri sono a = 0 e b = 7

La chiave pubblica può essere facilmente derivata attraverso la moltiplicazione scalare con la formula:

chiave pubblica = chiave privata * punto base P

Dove P è un punto che viene scelto a caso. Questa operazione è una sequenza di addizioni e computazionalmente semplice. Una volta che scegli la tua chiave privata e la moltiplichi per il punto base P, ottieni un nuovo punto (x,y) nel campo finito/curva ellittica. Questa è la tua chiave pubblica. È invece computazionalmente molto difficile iniziare con la chiave pubblica e lavorare all'indietro per calcolare la chiave privata. L'equazione è una strada a senso unico.

PROOF OF WORK

Su Bitcoin il processo di creazione di nuovi blocchi (e quindi la certificazione delle transazioni avvenute) è affidato ai miners. Poiché i miners, in linea teorica, potrebbero manomettere delle transazioni per il proprio guadagno, è richiesto loro di spendere del potere computazionale, disincentivando così la truffa. Inoltre, per i miner che si comportano correttamente è allocata una reward in BTC per ogni blocco minato. Più in generale, ci riferiamo con il termine Proof of Work al meccanismo che permette il raggiungimento del consenso tra i vari attori del network, ad esempio sul numero di Bitcoin che ogni wallet detiene.

Il problema matematico che i miners devono risolvere è il seguente: trovare un nonce tale che

H(H(blocco precedente), M(transazioni + transazione entrate), nonce)

è un hash che inizia con C zeri. Dove H è una funzione di hash, mentre M è un merkle root. C invece controlla la complessità del problema e si adatta regolarmente ogni 2 settimane in base alla potenza complessiva del computer nella rete. Maggiore è la potenza di calcolo, maggiore è la difficoltà, maggiore è il numero di 0 con cui dovrebbe iniziare l'hash. L'obiettivo è produrre un blocco ogni 10 minuti.

Si noti che l’unico modo di trovare la soluzione è per tentativi, provando con differenti valori del nonce.

Quando un miner trova un nonce, invia a tutti i nodi della rete il blocco completo e la soluzione del problema che ha appena risolto. I nodi

valutano la correttezza del blocco prodotto e lo aggiungono alla blockchain, mentre il miner riceve la sua ricompensa in BTC

OLD SC CODE

Gabriele Corbo, [22/05/2022 16:28]

```
pragma solidity ^0.5.0;


contract Coin {
  // The keyword "public" makes those variables
  // easily readable from outside.
  address public minter;
 mapping (address => uint) public balances;


  // Events allow light clients to react to
  // changes efficiently.
  event Sent(address from, address to, uint amount);


  // This is the constructor whose code is
```

```
  // run only when the contract is created.
  constructor() public {
    minter = msg.sender;
  }

  function mint(address receiver, uint amount) public {
    require(msg.sender == minter);
    require(amount < 1e60);
    balances[receiver] += amount;
  }

  function send(address receiver1,address receiver2, uint amount) public {
    require(amount <= balances[msg.sender], "Insufficient balance.");
    balances[msg.sender] -= amount;
    balances[receiver1] += amount/2;
    balances[receiver2] += amount/2;
    emit Sent(msg.sender, receiver1, amount);
  }
```

```
}



pragma solidity ^0.4.18;


contract HelloCoin {


string public name = 'HelloCoin';

//currency name. Please feel free to change it


string public symbol = 'poli';

//choose a currency symbol. Please feel free to change it


mapping (address => uint) balances;

//a key-value pair to store addresses and their account balances


event Transfer(address _from, address _to, uint256 _value);

// declaration of an event. Event will not do anything but add a record to the log
```

```
constructor() public {

  //when the contract is created, the constructor will be called automatically

  balances[msg.sender] = 10000;

  //set the balances of creator account to be 10000. Please feel free to change it to any number you want.

}



function sendCoin(address _receiver, uint _amount) public returns(bool sufficient) {

  if (balances[msg.sender] < _amount) return false;

  // validate transfer

  balances[msg.sender] -= _amount;

  balances[_receiver] += _amount;

  emit Transfer(msg.sender, _receiver, _amount);

  // complete coin transfer and call event to record the log

  return true;

}
```

```
function getBalance(address _addr) public view returns(uint) {
  //balance check
  return balances[_addr];
}
}
```