Università degli Studi di Verona

FACOLTÀ DI SCIENZE E INGEGNERIA
Corso di Laurea Triennale in Informatica

Tesi di laurea triennale

LiteChain2DB

Un framework per la memorizzazione e l'analisi integrata della blockchain di Litecoin tramite un database relazionale

Candidato:
Armando De Berti
Matricola VR421559

Relatore:
DR. Sara Migliorini

Anno Accademico 2019–2020

Indice

Introduzione

L'obiettivo di questa ricerca è definire una struttura dati relazionale che consenta di memorizzare le informazioni contenute nella blockchain della criptovaluta Litecoin e di integrarle con altre fonti di dati, al fine di studiare ed analizzare il suo contenuto dal punto di vista statistico, come l'andamento del suo valore nel tempo.

Questo risultato è stato ottenuto sfruttando una libreria scritta in python, che permette di leggere i dati della catena memorizzati in files .dat, scaricabili da Internet e disponibili per chiunque, e di caricarli in un database PostgreSQL.

Lo studio è iniziato dall'analisi delle librerie esistenti per caricare il contenuto della blockchain in un database relazionale.

Dopo aver individuato la giusta libreria, l'ho configurata per comunicare col database ed ho caricato tutti i blocchi. Successivamente, ho analizzato la struttura del database generato, per verificarne la correttezza e completezza, ed ho apportato alcune modifiche, come l'aggiunta di alcune tabelle, per meglio rappresentare l'informazione e svolgere le interrogazioni di analisi.

Le indagini sono state svolte attraverso query SQL ed i risultati delle interrogazioni sono stati rappresentati con grafici. L'analisi dei grafici ha permesso di fare interessanti ragionamenti sul mondo delle criptovalute, sulla dinamicità del loro mercato e di quanto esso risenta degli eventi che accadono nel mondo.

Il risultato finale dello studio è stata la realizzazione di una libreria di funzioni, in grado di fare interrogazioni complesse sul database e di integrare altre sorgenti dati e sopperire ad alcune mancanze di informazioni.

Capitolo 1

La tecnologia Blockchain

1.1 Funzionamento

La tecnologia blockchain (letteralmente "catena di blocchi") è un sistema di registrazione dati distribuito, ovvero nel quale le informazioni sono salvate su dei "blocchi", in modo totalmente decentralizzato. La sicurezza del loro contenuto è garantita da tecniche crittografiche ed un innovativo protocollo di consenso. Le informazioni vengono raggruppate in blocchi, a loro volta concatenati a formare una catena, la quale viene costantemente aggiornata con nuovi elementi dagli utenti.

Il primo esempio di implementazione di questa tecnologia è stata l'introduzione della blockchain di Bitcoin nel 2008.[1] Essa è stata la prima criptovaluta ideata e tuttora quella più conosciuta, concepita da Satoshi Nakamoto, pseudonimo di un'identità non ancora svelata.

La blockchain può essere utilizzata per memorizzare diversi tipi di informazione. Ogni blocco è infatti un registro, nel quale sono salvati diversi record in modo sicuro, verificabile e trasparente. Nell'ambito delle criptovalute si memorizzano le transazioni.

Uno dei principi di funzionamento più importanti è l'algoritmo di consenso basato sul Proof of Work (PoW). Si tratta di un problema matematico che, se risolto, permette di allacciare un nuovo blocco alla catena. Per risolvere il PoW sono necessarie risorse hardware di particolare potenza, rendendo quindi il procedimento complesso. La sua difficoltà permette di garantire l'immutabilità della rete, anche in assenza di una figura centralizzata, in quanto, l'accettazione di un nuovo blocco, viene convalidata solo se la maggioranza dei nodi concorda sulla sua legittimità. Una volta che i dati sono stati scritti, non possono essere retroattivamente alterati, senza che vengano modificati tutti i blocchi successivi ad esso. Grazie alla particolare natura del protocollo e dello schema di validazione di nuovi blocchi, anche la modifica necessita quindi del consenso della maggioranza della rete.[1] Il PoW è quindi il cardine su cui è basato il consenso distribuito della blockchain ed il suo principale garante.

Un altro metodo di sicurezza della blockchain è la crittografia a chiave pubblica. Questo sistema, che permette di scambiare messaggi in modo sicuro, viene gestito da due componenti: la chiave pubblica e quella privata. Supponiamo che, per esempio, Alice voglia mandare un messaggio a Bob. Per prima cosa, lo cifra con la chiave pubblica di quest'ultimo. Poi Bob può in seguito decifrarlo usando la propria chiave privata ed ottenere il testo in chiaro. Un eventuale attaccante potrebbe intercettare il messaggio e

[1] Wikipedia - https://it.wikipedia.org/wiki/Blockchain

provare a decifrarlo, ma non riuscirebbe nell'impresa, in quanto solo Bob possiede la sua chiave privata. Per questi motivi la chiave privata non va assolutamente diffusa o perduta, esattamente come una qualsiasi altra password.

Il meccanismo si basa sul fatto che, se con una delle due chiavi si cifra (o codifica) un messaggio, allora quest'ultimo potrà essere decifrato solo con l'altra. Questo metodo di crittografia è anche chiamato "crittografia asimmetrica", perché evita qualunque problema connesso alla necessità di uno scambio in modo sicuro dell'unica chiave utile alla cifratura/decifratura, presente invece nella crittografia simmetrica.[2]

La crittografia a chiave pubblica/privata può essere usata anche come meccanismo di firma. In questo caso la chiave privata è usata per firmare e la chiave pubblica per verificare l'autenticità della firma. Questa modalità di funzionamento è quella usata nell'ambito della blockchain.

La chiave pubblica è usata per generare nuovi indirizzi (o account) attraverso i quali è possibile ricevere tokens. I tokens di valore inviati nella rete vengono registrati come appartenenti a questo indirizzo. Invece la chiave privata è come una password, che permette al suo proprietario di accedere alle sue risorse digitali, oppure di interagire con le varie funzionalità della blockchain.[1]

Partendo da questi principi, sicurezza, trasparenza e consenso, la blockchain è diventata la declinazione in digitale di un nuovo concetto di fiducia, al punto che alcuni ritengono che possa assumere anche un valore per certi aspetti di tipo "sociale e politico". In questo caso la blockchain è da vedere come una piattaforma che consente lo sviluppo e la concretizzazione di una nuova forma di rapporto sociale, che, grazie alla partecipazione di tutti, è in grado di garantire a tutti la possibilità di verificare, di "controllare", di disporre di una totale trasparenza sugli atti e sulle decisioni, i quali vengono registrati in archivi, che hanno caratteristica di essere inalterabili, immodificabili e dunque immuni da corruzione.[3]

1.2 Struttura e scoperta dei blocchi

Un blocco è strutturato in due parti: header (intestazione) e body (corpo). I record (le transazioni) sono salvati nel secondo, mentre il primo è invece diviso in sette campi.

Ogni transazione a sua volta specifica una serie di inputs (chiamati transaction inputs), cioè un insieme di tokens attualmente associati ad un account e non ancora spesi, e outputs (chiamati transaction outputs). L'insieme dei tokens correntemente assegnati ad uno specifico account è detto Unspent Transaction Output (UTXO). Una transazione consuma UTXOs e produce nuovi UTXOs. La somma di tutti i tokens all'interno di ogni transaction input deve essere uguale a quello di tutti gli outputs. Quando il risultato di un transaction output viene usato come input per un'altra transazione, deve comunque essere speso nella sua interezza. Però il mittente può anche non usare tutta la somma in input. Egli deve, quindi, reindirizzare se stesso nei destinatari, in modo tale da rientrare in possesso dei tokens che non voleva effettivamente usare. Questa manovra viene chiamata "change" (resto). Se la differenza fra i tokens dei transaction inputs e quella degli outputs non è equivalente a zero, la differenza viene chiamata "fee" (commissione).

Nell'ambito di Bitcoin e di Litecoin, i sette campi dell'header sono rispettivamente:

[2] Wikipedia - https://it.wikipedia.org/wiki/Crittografia_asimmetrica

[3] Mauro Bellini, "Blockchain: cos'è, come funziona e gli ambiti applicativi in Italia" - https://www.blockchain4innovation.it/esperti/blockchain-perche-e-cosi-importante/

Versione	02000000
Hash del blocco precedente (PrevHash)	E87C17C45768w7e1643fsd5481sd3f4131df681
Merkle root	697we168t4v1a4rv3v1e3r43c4er14ca8c4168a
Timestamp	358b0553
Bits	535f0119
Nonce	48750933
Numero di transazione	64

Il campo Versione dipende dal software utilizzato. Il campo PrevHash è l'hash di 256 bit, calcolato sul precedente blocco della catena. Il Merkle root è l'hash degli hash di tutte le transazioni nel blocco. Il campo Timestamp rappresenta il time stamp dell'ultima transazione con un algoritmo conosciuto come Unix hex timestamp. Questo non va confuso con timestamp UNIX, valore utilizzato nell'ambito dei sistemi operativi UNIX e UNIX-like, che esprime invece bit numerici in secondi dal 1970-01-01T 00:00 UTC. Il campo Bits ed il campo Nonce invece rivestono un'importanza fondamentale per la scoperta di nuovi blocchi. Infine il Numero di transazione identifica il numero di transazioni presenti nel blocco.[1]

La scoperta di un nuovo blocco è tutt'altro che semplice. Per fare ciò bisogna mettere a disposizione il proprio hardware e risolvere il Proof of Work. Per far sì che un nuovo blocco venga accettato, bisogna fare in modo che l'hash del nuovo sia minore o al massimo uguale al valore nel campo Bits del blocco precedente, cioè deve iniziare con un numero di zeri definiti tramite questo campo. Per questo motivo il campo Bits viene infatti definito come indicatore della difficoltà target. Diminuirne il valore, significa aumentare la difficoltà della scoperta del blocco successivo. Per far sì che l'output della funzione di hash vari, si agisce modificando in modo casuale il campo Nonce. Il campo Nonce è un valore a 8 byte attraverso il quale si cerca di ottenere un blocco il cui hash soddisfa i requisiti richiesti.[1]

Una volta costruito un blocco il cui hash rispetta il criterio descritto, il blocco viene aggiunto alla catena insieme alle transazioni contenute in esso. Inoltre i suoi scopritori vengono premiati in criptovalute.

Gli utenti che lavorano per costruire e mantenere la catena sono definiti "miners", in quanto, proprio come i minatori, scoprono nuovi blocchi e ne ricevono in premio monete quando risolvono il PoW.

Il meccanismo con cui le transazioni vegono scelte per essere inserite in un blocco non è particolarmente complesso. Ogni volta che viene creata una transazione, essa viene messa in un elenco chiamato "mempool". Ogni miner ne possiede uno, contenente alcune transazioni ancora da convalidare. Ogni miner decide in modo autonomo quali transazioni prelevare dal mempool e mettere in un blocco. Ogni volta quindi che un blocco viene scoperto, le sue transazioni vengono tolte dal mempool ed inserite nel suo body. Questa decisione dipende da una serie di fattori, tra cui principalmente la commissione (fee) che l'utente intende pagare per la sua transazione. Essa è specificata come differenza tra l'input e l'output della transazione e finisce nel premio per lo scopritore del blocco. Più la fee è alta, maggiore è la precedenza che una transazione ha sulle altre. Il sistema infatti tende a favorire una transazione sulla quale i miners possono guadagnare il più possbile.

Un altro fattore che contribuisce alla scelta è la dimensione della transazione in termini di spazio. Lo spazio nel blocco non è certamente infinto, quindi bisogna anche valutare quanta memoria è disponibile per la memorizzazione delle transazioni.

Un elemento che entra inoltre in gioco è la parentela fra le transazioni. Quando un output di una transazione viene utilizzato come input per un'altra, si crea una sorta di parentela fra le due. Perché una transazione "figlia" possa essere inserita in un blocco, prima devono essere già stati inseriti tutti i suoi "genitori" all'interno della catena.[4]

Un ultimo criterio importante che viene considerato è l'anzianità della transazione. Le transazioni più vecchie hanno maggiore possbilità di essere scelte. Questo fattore serve per evitare il fenomeno della "starvation", ovvero che una transazione non venga inserita e che quindi rimanga pendente a tempo indeterminato.

Le transazioni vengono considerate valide, quindi permanentemente aggiunte alla blockchain e rimosse dal mempool, solo quando il blocco che le contiene ha ottenuto almeno 6 validazioni, cioè solo quando sono stati aggiunti 6 blocchi dopo di lui alla catena lungo la diramazione.

Può capitare che vengano selezionati più blocchi come nuovi elementi da inserire. In questo caso vengono attaccati entrambi alla catena creando una biforcazione (fork). Gli utenti lavorano su entrambi i rami appena scoperti, fintantoché non viene scoperto un nuovo blocco in una delle biforcazioni. Gli utenti abbandonano quindi la strada senza successore e si concentrano sull'altra. L'obiettivo è infatti quello di estendere la catena in lunghezza. Il blocco abbandonato diventa quindi "orfano" e le transazioni al suo interno rimesse nel mempool, in quanto non raggiungeranno mai il minimo di validazioni. Gli scopritori del blocco orfano non vengono, generalmente, premiati.[5]

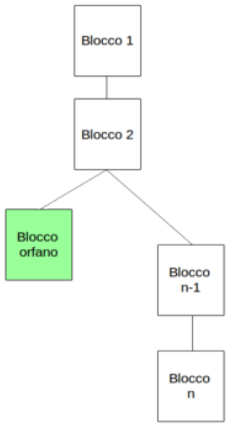

Figura 1.1: Fork di una chain. Il blocco verde non ha successori, quindi diventa orfano.

Il processo di scoperta di nuovi blocchi viene definito "mining". Negli anni, svolgere questa attività è diventato sempre più costoso, quindi i minatori digitali si sono riuniti in gruppi, chiamati "pool" che lavorano insieme e si dividono i guadagni.

Per far sì che la difficoltà rimanga costante rispetto alla potenza di calcolo e quindi per garantire che i nuovi blocchi siano aggiunti ad intervalli di tempo costanti, il valore del

[4] Brian Mancini, "CS and Bitcoin Mempool Transaction Selection" - https://www.derpturkey.com/cs-bitcoin-mempool-transaction-selection/

[5] Investopedia, Orphan block - https://www.investopedia.com/terms/o/orphan-block-cryptocurrency.asp

campo Bits viene modulato algoritmicamente. In Bitcoin il tempo medio di scoperta di un blocco è 10 minuti e per far sì che questo valore non cambi, la difficoltà viene modulata ogni 2016 blocchi.[6] Litecoin, essendo un derivato di Bitcoin, segue le stesse regole, ma in questo caso il suo tasso di scoperta desiderato è 2,5 minuti.[7]

Utilizzare un metodo così complicato per la scoperta di nuovi blocchi ha vantaggi e svantaggi.

Tra i lati positivi abbiamo in primis il consenso distribuito. Infatti, un nuovo blocco viene aggiunto e le transazioni al suo interno considerate valide, solo dopo un attento lavoro di mining.

Guidare la "storia" della rete richiederebbe quindi una grande capacità di calcolo, costosa da ottenere per pochi individui. Questo fa sì che nessuno possa impossessarsi della rete e quindi di centralizzarla. Se dovesse succedere una cosa del genere, la notizia si diffonderebbe, gli utenti abbandonerebbero la rete, e questa perderebbe tutto il suo valore, rendendo inutile ogni sforzo. Per questo stesso motivo anche gli attacchi DoS risulterebbero costosi e con risultati deludenti. Un altro vantaggio è che l'unica caratteristica considerata all'interno della rete, è quindi la potenza di calcolo. Non conta infatti possedere più monete degli altri. Chi possiede grosse quantità di denaro, quindi, non ha maggiore controllo sulla rete, al contrario dei sistemi economici tradizionali.[8]

Tra gli svantaggi, uno su tutti è il costo. Il processo di mining richiede macchine altamente specializzate, capaci di risolvere in tempi brevi algoritmi estremamente complessi. Questi dispositivi non sono solo estremamente costosi, ma consumano enormi quantità di energia elettrica, che diviene una spesa consistente. Si tratta di una pericolosa minaccia alla decentralizzazione del sistema, in quanto solo una piccola fetta dell'utenza può permettersi questo genere di investimenti.[8]

È esattamente l'altra faccia della medaglia.

[6] Bitcoin dalla teoria alla pratica, "Che cosa è la Proof of Work?" - https://medium.com/@satoshiwantsyou/come-funziona-il-proof-of-work-bitcoin-blockchain-eac6ba859253

[7] Forum - https://bitcoin.stackexchange.com/questions/79820/how-does-the-litecoin-difficulty-get-calculated#:~:text=Bitcoin%20creates%202016%20blocks%2C%20and,10%20minutes%2C%20the%20difficulty%20decreases.

[8] Andrew Tar, "Proof-of-Work, spiegata semplicemente" - https://it.cointelegraph.com/explained/proof-of-work-explained

Capitolo 2

Litecoin

Figura 2.1: Logo di Litecoin

2.1 Cenni di storia

Litecoin è una criptovaluta peer-to-peer che sfrutta la tecnologia blockchain, nata nel 2011 grazie all'idea di un dipendente di Google laureatosi al MIT, Charlie Lee. Egli non ha mai nascosto la sua identità, al contrario dell'ideatore del fratello più famoso Bitcoin, Satoshi Nakamoto, sul quale si hanno invece scarsissime informazioni.

Lee è molto attivo nello sviluppo e progresso della sua "creatura". Ha sempre affermato di essersi ispirato a Bitcoin, definito da lui "un'invenzione geniale", e dopo essersi dimesso da Google, ha fondato la Litecoin Foundation, che gestisce tuttora.

Litecoin, la cui sigla del token è LTC, è stata distribuita inizialmente mediante un client open source su GitHub, il 7 ottobre 2011. Il suo codice sorgente era scritto nel linguaggio C++.

La rete Litecoin è entrata ufficialmente in funzione il 13 ottobre 2011. In pochi anni ha avuto un discreto successo grazie alle sue qualità uniche che la rendevano preferibile a Bitcoin.

Nel 2013 il valore complessivo di Litecoin è cresciuto in modo massiccio, con un trend del 100% ogni giorno. Numeri che hanno fatto parlare importanti agenzie mediatiche, come il Wall Street Journal, CNBC, il New York Times e il The Economist, di Litecoin come alternativa (o addirittura come possibile successore) a Bitcoin. Peccato che poi, tutto ciò, sia stata una promessa mancata, dato che, col tempo, si è fatto superare, in termini di popolarità e di valore, da altre criptovalute, come Ethereum e Ripple, e se la giochi con altre a ridosso del podio.[9]

[9] "Litecoin (LTC): cos'è e come fare trading" - https://www.comefaretradingonline.com/criptovalute/litecoin.php

I Paesi nei quali Litecoin ha più successo sono: Paesi Bassi, Repubblica Ceca, Finlandia e Russia. Quindi, soprattutto quelli del Nord Europa e dell'Est Europa. Nel gennaio 2014, la Banca nazionale della Finlandia ha addirittura dichiarato che Litecoin poteva essere considerato una merce di scambio vera e propria.[9]

Questo va considerato un grande riconoscimento da parte di uno stato, verso una forma di moneta decentralizzata.

Infatti, solitamente, un sistema di pagamento di questo tipo è inviso ai governi, proprio perché non controllabile e quindi ritenuto da essi pericoloso.

2.2 Differenze con Bitcoin

Le differenze tra Litecoin e Bitcoin, che le hanno permesso di diffondersi e di acquisire notorietà, sono varie. Principalmente, la sua notevole velocità nel mining di nuovi blocchi, che rende le transazioni di LTC molto più veloci. Nel maggio del 2017, viene segnato un record mondiale, in quanto una transazione da Zurigo a San Francisco, sulla piattaforma decentralizzata, impiegò meno di 1 secondo per ottenere la conferma.[10]

Questa caratteristica, secondo Lee, renderà in futuro Litecoin molto più famoso.

> *Il mio sogno è che le persone usino Litecoin ogni giorno per acquistare oggetti comuni. Diventerà solo un metodo di pagamento come un altro.*
>
> *- Charlie Lee, creatore di Litecoin*

Se ad esempio, utilizzassimo Litecoin per pagare un caffè, non vorremmo certo dover aspettare 10 minuti prima che la transazione andasse a buon fine.

La grande rapidità è dovuta all'algoritmo implementato. Bitcoin utilizza il tradizionale algoritmo SHA-256 mentre invece Litecoin sfrutta una tecnologia differente: l'algoritmo "Scrypt".[11]

Questi due algoritmi influenzano in maniera decisiva la velocità con la quale è possibile minare i blocchi delle due criptovalute. Con Litecoin è possibile minare un blocco in circa 2,5 minuti, mentre con Bitcoin ne sono necessari ben 10. L'algoritmo SHA-256 utilizzato da Bitcoin è celebre per essere estremamente complesso. Per questo i miners, negli anni sono passati all'impiego di macchine con hardware specifico (ASIC), per minare i blocchi della criptovaluta più importante al mondo.

Scrypt, l'algoritmo utilizzato da Litecoin, risulta essere estremamente più efficiente, grazie alla sua progettazione, che non consente di utilizzare hardware specifico, come accade invece con Bitcoin.[10]

Infatti Scrypt viene definito "memory hard problem", in quanto tende a premiare l'utilizzo della memoria ram, invece che la sola potenza di calcolo.

Questo permette quindi di attenuare la differenza che invece si è venuta a creare in Bitcoin fra i vari miners, in base all'hardware utilizzato per la scoperta di nuovi blocchi.

Litecoin è quindi in un certo senso più "democratico", fa meno distinzioni fra chi usa un hardware specifico e chi invece la CPU.

Un'altra sostanziale differenza fra le due è il tetto massimo di monete in circolazione. Per Litecoin è infatti più alto. Bitcoin ha un massimale di 21 milioni, contro gli 84 di Litecoin. Inoltre il valore di LTC in altre valute comuni, rispetto a BTC, è decisamente inferiore per ora, e ciò lo rende molto più economico.[12] I Litecoin sono spesso definiti come "argento" delle criptovalute, mente i Bitcoin sono l'"oro".

[10] "Litecoin: cos'è, come funziona e come guadagnare [2020]" - https://www.criptovalute24.com/litecoin/#Litecoin_storia
[11] Wikipedia - https://it.wikipedia.org/wiki/Litecoin
[12] StormGain, "Cos'è la criptovaluta Litecoin?" - https://stormgain.com/it/blog/what-litecoin-cryptocurrency

2.3 Tipi di indirizzo

Come detto nell'introduzione, i tokens in circolazione sono associati ad indirizzi (account) a loro volta generati a partire da chiavi pubbliche. In Litecoin esistono diversi tipi di indirizzo. Essi si dividono in primis in due tipologie, in base al numero di firme digitali da cui sono generati: a singola firma o multi-firma.

La firma digitale è un meccanismo di verifica dell'autenticità di un messaggio. Se, ad esempio, Alice vuole mandare un messaggio a Bob usando la firma digitale, dovrebbe prima di tutto generare il messaggio e poi farne l'hash. Il risultato viene chiamato "impronta" del documento o "digest". In seguito, utilizza la propria chiave privata per codificare il digest. Questa è la firma vera e propria. Non è possibile da essa risalire alla chiave privata o al testo originario. Infine Alice invia la firma insieme al testo e ad una copia della sua chiave pubblica. Bob riceve il messaggio, decodifica la firma ed ottiene l'impronta. La confronta poi con l'hash che esegue sul testo e verifica se sono uguali. In caso affermativo, il testo può essere attribuito in modo incontrovertibile ad Alice, essendo lei l'unica in possesso della chiave privata usata per la firma.[13] [14]

Questo metodo, nell'ambito delle criptovalute, viene usato nelle transazioni. Perché una transazione possa essere convalidata, bisogna prima di tutto sbloccare i fondi. Questo viene attuato "dimostrando" che li si possiede, ovvero provando che si detiene la chiave privata che li gestisce. Tale dimostrazione avviene attraverso la condivisione della chiave pubblica, secondo il meccanismo indicato sopra.[15]

Gli indirizzi standard a firma singola, necessitano solo di una chiave pubblica e di una sola privata.

Quelli multi-firma invece funzionano in modo diverso. Necessitano di un numero superiore di firme, generate attraverso chiavi private, per accedere ai fondi. La quantità dipende dalla natura dell'indirizzo. È come una cassetta di sicurezza con più lucchetti, i quali si possono aprire soltanto con più chiavi diverse. Ad esempio, in un indirizzo multi-firma "2 a 3", si hanno a disposizione tre chiavi ma, per poter muovere il denaro del conto, ne sono necessarie due. Questo aumenta la sicurezza dell'account. Se una chiave venisse rubata da un attaccante tramite phishing, ad esempio, questo avrebbe comunque bisogno di almeno un'altra per poter accedere al conto corrente.

Nella blockchain di Litecoin gli indirizzi cominciano con la lettera maiuscola "L", al contrario in Bitcoin con il numero "1". Gli indirizzi multi-firma invece non hanno delle regole. Ogni indirizzo, tranne quelli standard, può essere multi-firma.

Gli indirizzi si possono poi dividere in due ulteriori categorie: indirizzi mainnet e indirizzi testnet. I primi sono quelli che appartengono alla rete principale, ovvero quelli impiegati per le transazioni delle criptovalute. Gli altri appartengo invece alla rete di test. Essa è una rete a parte, che viene sfruttata per eseguire test dagli sviluppatori, permette loro di fare esperimenti, prendere rischi e cercare i modelli più stabili da poter poi implementare nella rete principale. Le monete utilizzate nella testnet non hanno alcun valore e non possono essere spese al di fuori di essa, perché tecnicamente non esistono.[16]

Gli indirizzi Litecoin sono divisi poi in 4 sotto-categorie:

Indirizzi legacy: sono gli indirizzi standard e sono quelli più utilizzati. Ogni indirizzo ha una propria chiave pubblica e privata associate, infatti non vengono usati come

[13] Wikpedia - https://it.wikipedia.org/wiki/Firma_digitale
[14] 10 Bitcoin.it, "Firma digitale" - http://www.10bitcoin.it/firma-digitale/
[15] learn me a bitcoin, "Digital Signatures" - https://learnmeabitcoin.com/beginners/digital_signatures#:~:text=-A%20digital%20signature%20is%20something,digital%20signature%20to%20prove%20it.
[16] Ani, "Crypto Mainnet vs Testnet: What is the Difference?" - https://www.altcoinbuzz.io/bitcoin-and-crypto-guide/crypto-mainnet-vs-testnet/

multi-firma. Vengono usati quindi dai singoli utenti. Cominciano con “L” nella mainnet e con “m” nella testnet.[17]

Indirizzi p2sh: il loro funzionamento è diverso dagli indirizzi standard. Generalmente, il destinatario di una transazione, riceve anche uno script detto "ScriptPubKey", o anche "locking script", che indica come i fondi ricevuti debbano essere spesi. Con lo standard p2sh invece, il ricevente riceve l’hash di uno script, detto "redeem script hash". Questo ha l’effetto di sollevare il mittente dalla descrizione di dettagli a lui inutili e muovere questa responsabilità verso il destinatario. Essi sfruttano anche la tecnologia SegWit e sono quindi detti SegWit annidati. Vengono inoltre spesso usati come indirizzi multi-firma. Sono quindi adoperati da gruppi di utenti che decidono di sfruttare un account comune. Questo tipo di indirizzi ha subito un passaggio di versione. Inizialmente cominciavano con “3”. Questo perché Litecoin ha mantenuto lo standard di Bitcoin, generando non pochi problemi. Infatti, mandando fondi ad un indirizzo p2sh, poteva capitare che le monete venissero inviate ad un indirizzo appartenente all’altra chain, andando irrimediabilmente perdute. Per questo, in tempi recenti, Litecoin ha deciso di adottare un nuovo standard di indirizzi p2sh, i quali cominciano con la lettera “M” e sono perfettamente convertibili con quelli dello standard precedente.[18] Nella testnet, invece, cominciano con “2”, versione vecchia, e con “Q”, versione nuova.[17]

Indirizzi SegWit nativi: lo standard Segregated Witness, abbreviato SegWit, permette di avere transazioni più corte in termini di spazio, perché separa dalla transazione alcuni dati sulla firma. Ciò permette di avere più transazioni in unico blocco e quindi rende la conferma delle stesse molto più veloce. Inoltre permette di ridurre la commissione per ogni transazione. Gli indirizzi nativi funzionano molto meglio di quelli annidati ed hanno un controllo maggiore sugli errori. Vengono quindi utilizzati per la loro "leggerezza" e velocità, ma essendo un’implementazione recente, soffrono ancora però di qualche problema di compatibilità.[19] Cominciano con “ltc1” nella mainnet e con “tlct1” nella testnet. Sfruttano un altro tipo di codifica rispetto alla base 58, ovvero l’algoritmo bech 32. Per questo motivo, l’algoritmo che li ricava è diverso da quello delle altre tipologie.[20]

Indirizzi non comuni: essi non seguono nessuno standard in particolare e sono il caso più comune di indirizzi multi-firma. L’algoritmo per ricavarli dall’hash della chiave pubblica dipende dall’explorer preso come riferimento. Qui si fa riferimento all’explorer **BLOCKCYPHER**[21] [22], nel quale cominciano con “4”. Ho scelto come modello questo explorer in quanto esso utilizza uno standard abbastanza riconoscibile per questo tipo di indirizzi.

Non è facilmente verificabile, causa la natura particolare di questi tipo di indirizzi, con quale carattere comincino nella testnet.

[17] Forum - https://bitcoin.stackexchange.com/questions/62781/litecoin-constants-and-prefixes
[18] crypto facilities, "Litecoin Address Format" - https://cryptofacilities.zendesk.com/hc/en-us/articles/360006040974-Litecoin-Address-Format
[19] Ledger Academy, "Bitcoin: What’s the difference between SegWit and Native SegWit (Bech32)?" - https://www.ledger.com/academy/difference-between-segwit-and-native-segwit
[20] Forum - https://www.reddit.com/r/litecoin/comments/97gp7p/litecoin_segwit_bech32_address_prefix/
[21] Link - https://live.blockcypher.com/ltc/
[22] Versione libreria da terminale python - https://github.com/blockcypher/blockcypher-python

2.4 Algoritmo per gli indirizzi

Gli indirizzi non vengono direttamente salvati nei blocchi e questo costituiva un ostacolo per il mio lavoro.

Dopo varie ricerche, ho scoperto che avrei dovuto applicare un algoritmo particolare alla chiave pubblica. In seguito, sono riuscito ad individuare il metodo adatto per ricavare l'indirizzo da una chiave pubblica da una fonte in rete.[23] I passaggi dell'algoritmo trovato erano particolarmente compressi, quindi li ho espansi, per rendere tutto più comprensibile. In base al tipo di indirizzo che si vuole ricavare, esiste un particolare prefisso da aggiungere, composto da due caratteri, spiegato nella tabella 2.1, anch'essa ricavata online.[17]

Nell'algoritmo 1 il prefisso è chiamato "pr".

Algorithm 1 Algoritmo per ricavare l'indirizzo dalla chiave pubblica

1: **procedure** ADDR($pubkey$)
2: $p = pubkey$
3: $p = p.upper()$ ▷ rendo maiuscole tutte le lettere
4: $p = sha256(p)$ ▷ applico l'algoritmo di hash sha256
5: $p = p.upper()$
6: $p = rip160(p)$ ▷ applico l'algoritmo di hash ripemd160
7: $p = p.upper()$
8: $p = pr + p$ ▷ aggiungo il numero 30 all'inizio della stringa
9: $p2 = p.upper()$
10: $p2 = sha256(p2)$
11: $p2 = p2.upper()$
12: $p2 = sha256(p2)$
13: $p2 = p2.upper()$
14: $p3 = p + p2[0:8]$ ▷ concateno le due parti, troncando la seconda che rappresenta il checksum
15: $p3 = bytes.fromhex(p3)$ ▷ trasformo la stringa di esadecimali in un bytearray
16: $p3 = base58.b58encode(p3)$ ▷ codifico in base 58
17: **return** $p3$

Si presentava però un altro problema. Nella catena non sempre viene salvata la chiave pubblica di un indirizzo. Spesso viene conservato solo l'hash, rendendo il mio algoritmo parzialmente inutile. Fortunatamente, uno dei passaggi intermedi della lavorazione della chiave pubblica, consiste anche nel farne l'hash. Quindi mi è bastato rimuovere alcuni passaggi per ottenere l'algoritmo 2, il quale ricava l'indirizzo direttamente dall'hash.

Questo algoritmo mi è anche servito per ricavare il prefisso corretto per gli indirizzi non comuni della tabella 2.1, attraverso la sua esecuzione, su un hash con indirizzo già noto, di tutte le possibili combinazioni di pr.

[23] Algoritmo individuato in questo forum - https://bitcoin.stackexchange.com/questions/65282/how-is-a-litecoin-address-generated

Algorithm 2 Algoritmo per ricavare l'indirizzo dall'hash della chiave pubblica

1: **procedure** ADDR_HASH($hashpubkey$)
2: $p = hashpubkey$
3: $p = p.upper()$
4: $p = pr + p$
5: $p2 = p.upper()$
6: $p2 = sha256(p2)$
7: $p2 = p2.upper()$
8: $p2 = sha256(p2)$
9: $p2 = p2.upper()$
10: $p3 = p + p2[0:8]$
11: $p3 = bytes.fromhex(p3)$
12: $p3 = base58.b58encode(p3)$
13: **return** $p3$

Tipologia	Mainnet	Testnet
legacy	30	6f
p2sh old	05	c4
p2sh new	32	3a
non comune	08	X

Tabella 2.1: Tabella degli indirizzi con i rispettivi prefissi.

L'algoritmo 3, trovato in un forum in Internet[24], serve per ricavare invece gli indirizzi SegWit nativi ed è diverso dagli altri. Esso infatti sfrutta lo ScriptPubKey e non l'hash della chiave pubblica.

Algorithm 3 Algoritmo per ricavare l'indirizzo SegWit dall'hash

1: **procedure** ADDR_SW_SPK($scriptpubkey$)
2: $spk = binascii.unhexlify(scriptpubkey)$ ▷ Trasformo lo spk da esadecimale a binario. Deve essere di lunghezza pari
3: **if** $spk[0]$ **then** ▷ Ricavo la versione in base al primo valore
4: $version = spk[0] - 0x30$
5: **else**
6: $version = 0$
7: $program = spk[2:]$
8: **return** $bech32.encode('ltc', version, program)$ ▷ Codifico in bech 32 l'hash attraverso la versione ed il prefisso

Dove "ltc" può essere sostituito con "tltc" per gli indirizzi testnet

[24] Algoritmo individuato in questo forum - https://bitcoin.stackexchange.com/questions/91748/how-to-use-python-reference-for-encoding-a-bech32-address

Capitolo 3

Studi sul database

3.1 Librerie esplorate

Inizialmente il mio lavoro si è concentrato su come entrare in possesso di una copia della chain di Litecoin. Non è stato particolarmente difficile. Il primo passo era quello di creare un wallet, o portafoglio, dove tenere il mio saldo. L'obiettivo non era quello di investire o di fare trading, ma era il passaggio successivo quello interessante, ovvero scaricare tutta la blockchain dalla rete. È stato un procedimento lungo e che ha richiesto parecchio lavoro. Alla fine però sono riuscito nel mio intento.

La mia ricerca vera e propria è cominciata da qui. Lo scopo iniziale, era quello di scaricare una libreria, che mi permettesse di leggere i dati della catena. Questa era divisa in circa 186 files con estensione .dat. Questi erano il vero cuore della blockchain. In essi vengono salvate tutte le transazioni che gli utenti operano fra loro. Ogni file .dat contiene infatti più blocchi della catena, ed in ogni blocco risiedono più transazioni.

Una delle prime librerie che ho scoperto è stata **BlockAPI**.[25] È stata sviluppata in Italia dai ricercatori dell'università di Cagliari e sembrava fornire esattamente quello che cercavo. Può infatti costruire una view della blockchain e salvarla in un database. Il problema era che purtroppo la libreria è compatibile solo con Bitcoin e Ethereum, quindi ho dovuto scartarla. La compatibilità con Ethereum non era un problema. Avrei potuto infatti sfruttare quella con Bitcoin. Litecoin e Bitcoin sono infatti simili, ma per quanto lo siano, purtroppo, non sono totalmente uguali. L'algoritmo implementato per gli hash dei blocchi è diverso, e quindi non mi fidavo ad applicare un algoritmo differente alla catena di Litecoin per caricare e salvare i blocchi sul database.

Un'altra libreria interessante che ho trovato si chiama **Blockchain Postgres Import**.[26] A detta degli sviluppatori, funziona con Bitcoin, ma anche con tutti i suoi similari. È stata scritta in Go, un linguaggio di programmazione ideato da Google. Non era proprio quello che cercavo, in quanto avrei preferito una libreria scritta in Python. Però decisi di non scartarla del tutto, ma di tenerla da parte se non avessi trovato candidati migliori.

La terza libreria che ho individuato è nota come **bitcoin-blk-file-reader**.[27] Essa legge i dati contenuti nei files .dat della catena e poi mostra i risultati. Non trasferisce i files su qualche database, quindi richiedeva ulteriore lavoro per completare il passaggio al DB. Il suo punto forte è l'elevata semplicità con cui si poteva eseguirla. Funzionava solo

[25] Reperibile a questo link - https://github.com/blockchain-unica/blockapi
[26] Link - https://github.com/blkchain/blkchain
[27] Link - https://github.com/mrqc/bitcoin-blk-file-reader

con Bitcoin, però in questo caso ciò poteva non essere un problema. Data la sua praticità con cui mostrava le informazioni basilari della catena, quali ad esempio gli hash delle transazioni, ho deciso di tenerla in considerazione, sfruttandola per i test.

La libreria successiva si chiama **fast-dat-parser**[28] ed è particolare. È scritta in C++, ha caratteristiche simili alla precedente che avevo scoperto, ha però molte più funzioni. Non può però trasferire su database. Decisi comunque di tenerla da parte.

Nelle ricerche mi sono anche imbattuto nella libreria del core di Litecoin, **Litecore Library**.[29] Può essere interessante da analizzare, ma è pensata per gli sviluppatori della catena. Permette infatti di creare nuovi indirizzi e transazioni. Purtroppo però, il mio scopo era quello di analizzare, quindi ho dovuto scartarla, nonostante in futuro potrebbe rivelarsi utile per altri scopi.

La penultima libreria incontrata, **bitcoin-blockchain-parser**[30], funziona anch'essa da explorer per la catena, ma non per salvare i dati su database. Decisi di tenere anch'essa da parte.

La libreria che mi ha dato infine l'aiuto che cercavo si chiamava **Abe**.[31] Essa è multi-configurabile, ovvero può funzionare con diverse chain di diverse criptovalute, e può caricare i dati estratti su database gestiti da dbms diversi. Le catene compatibili, oltre a Litecoin, sono le seguenti: Bitcoin, Bitleu, BlackCoin, Californium, CryptoCash, Dash, Hirocoin, KeccakChain, LegacyNoBit8, Maxcoin, Namecoin, NmcAuxPowChain, NovaCoin, NvcChain, PpcPosChain, ScryptJaneChain, Sha256Chain, Sha256NmcAuxPowChain, Testnet, Unbreakablecoin, X11Chain e X11PosChain. Essa, inoltre, crea automaticamente una struttura dati nel database. Quindi la mia indagine si focalizzava anche nello studio di quest'ultimo.

Inizialmente ho lavorato sul mio pc. Il mio scopo era quello di caricare i dati su un server gestito da PostgreSQL, quindi ho dovuto configurare la libreria perché fosse utile al mio fine. Prima di tutto ho dovuto settare la configurazione per la connessione col database in un file. Questo sarebbe poi servito alla libreria **Psycopg2** per la connessione attraverso python su un DB PostgreSQL. Ho poi dovuto indicare il tipo di database che avevo intenzione di usare ed il tipo di caricamento dei blocchi. Ho infatti indicato che i blocchi venissero letti direttamente dalla catena che avevo precedentemente scaricato, scrivendo inoltre il percorso della loro directory. Ho inoltre indicato che tipo di catena volevo caricare. Infine ho dovuto scaricare un file .so a parte. Esso si chiama **ltc_scrypt.so** ed è una libreria che viene caricata dinamicamente, funzionante su sistemi Linux e contiene l'algoritmo scrypt di Litecoin. È facilmente trovabile in rete.

Il processo di caricamento nel database si avviava digitando il comando **python -m Abe.abe –config abe-pg.conf –commit-bytes 100000 –no-serve** da terminale nella cartella principale della libreria. Durante l'upload dei blocchi nel DB ho deciso poi di sfuttare la libreria *bitcoin-blk-file-reader*, in quanto era quella più semplice da utilizzare per test veloci. A causa della mancanza di spazio non ero riuscito a caricare tutti i blocchi nel database, ma in quel momento non era un problema. Ho cominciato allora a fare gli studi e gli esperimenti.

Come riferimento per i test ho scelto gli explorer **Blockchair**[32] e **BLOCKCYPHER**.[21] [22] In particolare **BLOCKCYPHER** mi è tornato molto utile. Esso possiede infatti una versione in python da terminale, che mi ha permesso di fare test in maniera particolarmente semplice e di ricavare alcune informazioni altrimenti non ottenibili.

[28] Link - https://github.com/bitcoinjs/fast-dat-parser
[29] Link - https://github.com/litecoin-project/litecore-lib
[30] Link - https://github.com/losh11/python-litecoin-blockchain-parser
[31] Link - https://github.com/bitcoin-abe/bitcoin-abe
[32] Link - https://blockchair.com/it/litecoin

Il secondo obiettivo del mio studio della blockchain era anche l'analisi del mutamento del valore del LTC, in monete tradizionali, nel tempo. Procurarsi uno storico del prezzo non è stato un problema. Essendo informazioni di dominio pubblico, moltissimi siti tengono ora traccia del valore della moneta, anche a scopo di mercato.[33] Sono riuscito a recuperare i valori del cambio dollari/LTC dal 2014 fino al 2020 in un file .csv. Mi sembrava buono come intervallo di tempo da analizzare.

Dopo aver preso confidenza con il database che avevo costruito con la libreria ed aver fatto vari test, decisi di proseguire con il successivo step. Mi sono quindi connesso al server dell'università attraverso una connessione ssh ed ho trasferito la libreria sul server. Successivamente l'ho configurata per la lettura dei blocchi della blockchain, che mi erano stati prontamente resi disponibili, e per il trasferimento degli stessi nel database PostgreSQL lì presente. Il server universitario disponeva infatti di risorse maggiori, e quindi avrebbe potuto benissimo contenere tutti i blocchi nel database che stavo andando a riempire.

Il processo ha richiesto quasi un mese di lavoro, ma alla fine ho realizzato il mio scopo.

Successivamente, ho scritto un programma che trasferisse i dati dal file dei prezzi .csv in una tabella del database.

3.2 Schema del database

Il database costruito a partire dall'utilizzo della libreria Abe[31] è composto da una serie di tabelle, dette entità, e da qualche vista, le quali aiutano a rendere più veloci le query. Gli elementi si dividono in due categorie:

Elementi originali: questi sono quelli creati dalla libreria automaticamente. Tra di essi vi sono entità, viste ed indici.

Elementi creati: sono quelli che sono stati creati in seguito per rendere il lavoro, di studio e di stesura di una libreria, più semplice. Vi si trovano entità ed indici.

3.2.1 Schema ER

Lo schema ER è descritto dalla figura 3.1. Gli elementi in rosso sono quelli aggiunti rispetto alla libreria originale, per effetto dell'integrazione di altre fonti di informazione oppure per memorizzare risultati di computazioni utili per le interrogazioni successive. Vengono riportate solo le entità più significative per il resto della trattazione. Va ricordato infatti che Abe, la libreria utilizzata, può funzionare anche con altre blockchain legate ad altre criptovalute, quindi alcune entità non sono funzionali al nostro caso, oppure sono entità di servizio poco interessanti. La vera potenza della libreria, infatti, consiste nel poter salvare più catene all'interno dello stesso sistema. Potrebbe rivelarsi uno studio particolarmente interessante, quindi, salvare altre chain di altre criptovalute, come ad esempio Bitcoin, e metterle a confronto con quella di Litecoin già presente.

L'entità **Chain** indica sempre il primo blocco della catena e contiene i dettagli di ogni catena possibile, compreso Bitcoin. Nel nostro caso, nessuna delle altre chain ha segnato qualche valore come primo blocco, tranne che quella di Litecoin, in quanto unica catena raccolta e analizzata.

[33] Ho trovato lo storico a questo link - https://finance.yahoo.com/quote/LTC-USD/history?period1=1410912000&period2=1598572800&interval=1d&filter=history&frequency=1d

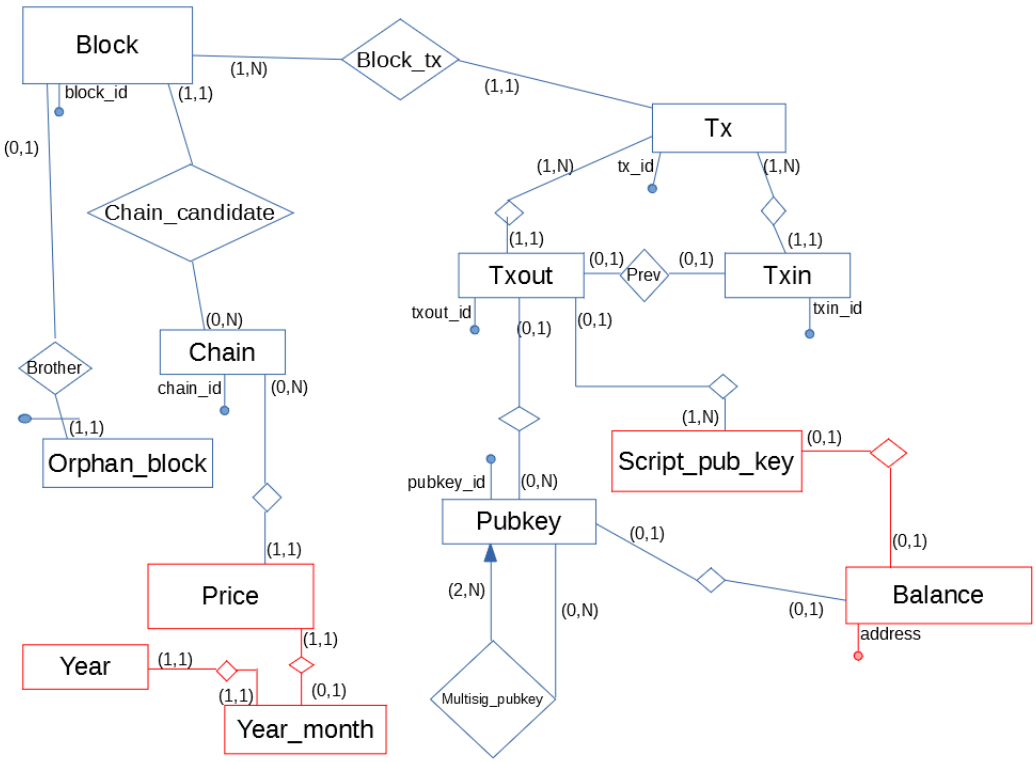

Figura 3.1: Schema ER. Gli elementi in rosso sono quelli creati

La relazione **Chain_candidate** invece mette in relazione ogni blocco con la catena di cui fa parte. Nello specifico caso considerato, dato che tutti i blocchi fanno parte della catena di Litecoin, sono tutti relazionati alla stessa istanza dell'entità **Chain**.

La relazione **Block_tx** mette in relazione blocchi e transazioni, salvando per ogni blocco, le corrispondenti transazioni. Ogni transazione può appartenere ad un solo blocco. Non vengono infatti considerate le transazioni dei blocchi orfani, in quanto non rintracciabili all'interno del database.

La relazione fra **Block** e **Tx** ha come cardinalità minima 1 dal lato del blocco, perché un blocco non può esistere senza almeno una transazione al suo interno.

La relazione fra **Block** e **Orphan_block** è particolare. Le chiavi primarie dei blocchi sono generate in modo sequenziale da PostgreSQL durante il caricamento di dati al suo interno, mentre, quelle degli orfani, corrispondono a quelle che avrebbero avuto se fossero stati integrati nella catena. Ogni blocco orfano ha un fratello legittimo, corrispettivo nell'entità dei blocchi, pertanto l'entità **Orphan_block** è da considerarsi debole. Questo legame viene enfatizzato nella base di dati assegnando lo stesso id del blocco legittimo, all'orfano nell'entità **Orphan_block**. La chiave primaria dell'entità degli orfani viene generata quindi in contemporanea con quella dei blocchi di **Block**, per questo quindi i loro identificativi coincidono. Significa che si forma una fork per ogni blocco minato. Questo è abbastanza plausibile, considerando la velocità e la relativa facilità con cui vengono minati i blocchi Litecoin. L'unico blocco il cui id non è presente anche all'interno della tabella degli orfani, ovvero non ha un fratello orfano, è ovviamente il blocco origine. Infatti non avrebbe alcun senso se pure il primo blocco avesse un alternativo orfano, perché la catena comincia proprio da lui. Non avendo un predecessore, quell'orfano non potrebbe essere stato generato, come alternativa, da un probabile predecessore.

La cardinalità minima di questa relazione è quindi 0 mentre la massima è 1 perché viene comunque salvato un solo fratello orfano per ogni blocco legittimo. Fork con più alternative, che quindi generano più di un blocco orfano, sono infatti particolarmente rare, ed in questa chain non ne sono presenti.

L'entità **Tx** possiede sempre almeno un transaction input ed un transaction output, quindi la cardinalità minima con entrambe è sempre 1.

Le entità **Txin** e **Txout** sono legate all'entità **Tx**. Ogni transaction input e output ha infatti una relazione con una ed una sola istanza dell'entità delle transazioni.

La relazione **Prev** rappresenta il legame della parentela, descritto prima. La cardinalità dal lato dell'entità **Txin** è $(0, 1)$, poichè un transaction input può corrispondere ad un solo output precedente, ad eccezione dei tokens ricevuti come compensa del mining, che non provengono da transazioni precedenti. Dal lato dell'entità **Txout**, può essere generato al massimo un solo transaction input. Infatti ogni transazione padre ha solo una figlia.

La tabella dei transaction outputs inoltre è collegata con quella delle chiavi pubbliche. Ogni txout può avere al massimo una pubkey associata, al contrario dei txins, che invece non possiedono questo campo nella loro tabella. La chiave è comunque ricavabile seguendo il percorso di parentela, interrogando il genitore. Questo è dovuto al fatto che nei files .dat non viene salvato l'indirizzo per i transaction inputs, ma viene ricavato rintracciando il genitore. La vista **Txin_detail** esegue questa computazione, e grazie ad essa non è necessario ricalcolare ogni volta questo percorso manualmente. I transaction outputs che non hanno un id di chiave pubblica, sono quelli che riguardano gli indirizzi SegWit.

L'entità **Pubkey** in pratica rappresenta gli indirizzi, escludendo quelli di tipo SegWit, degli account. Essa è a sua volta legata a se stessa attraverso il legame con la relazione **Multisig_pubkey**, che indica l'id delle chiavi multi-firma e quelle delle chiavi associa-

te. Ogni chiave multi-firma ha associate almeno due chiavi pubbliche, mentre una chiave pubblica può non essere legata ad una multifirma. L'entità delle chiavi pubbliche è inoltre in relazione con **Txout**, ma con cardinalità minima 0, in quanto non tutte le chiavi pubbliche partecipano a transazioni.

L'entità creata **Balance** è quella più interessante. Essa è stata creata da me come tabella per conservare tutti gli account analizzati dalle funzioni nella libreria **litecoin.py** descritta nella capitolo 4. Il campo dell'indirizzo dell'entità **Balance** funziona come una chiave primaria e viene ricavato dalle funzioni della libreria **litecoin.py**. Ogni istanza della tabella dei bilanci è legata ad almeno una chiave pubblica, o ScriptPubKey, appartenenti rispettivamente alle entità **Pubkey** e **Script_pub_key**. La relazione con quest'ultima ha cardinalità minima 0 e massima 1 dal lato della tabella dei saldi perché un record al suo interno può non essere legato ad uno ScriptPubKey. Questo permette di distinguere gli indirizzi SegWit dagli altri all'interno dell'entità. Dall'altro lato della relazione invece il minimo ed il massimo sono comunque 0 e 1, perché uno ScriptPubKey può non apparire o apparire al massimo una volta sola nell'entità **Balance**, legato ad uno specifico account. Questo può succedere nel caso uno ScriptPubKey appartenga ad un account il cui saldo non è ancora stato valutato dalla libreria. Lo stesso identico discorso vale per la relazione fra **Balance** e **Pubkey**.

L'entità **Price** è collegata con **Chain** attraverso l'id della catena. Ogni prezzo possiede infatti l'identificativo della chain di cui fa parte, anche se per ora, è presente solo lo storico dei prezzi di Litecoin. I prezzi sono storicizzati (tramite un apposito attributo nell'entità **Price**) ed inoltre sono legati anche all'entità **Year_month** per semplificare le analisi statistiche: esiste un'istanza in **Year_month** per ogni anno e mese per cui è memorizzato un prezzo. Da notare che la relazione ha cardinalità minima 0 dal lato **Year_month**, perché può capitare che un anno e mese presente in **Year_month** non sia presente nelle date della tabella dei prezzi.

La relazione tra **Year_month** e **Year** invece ha cardinalità minima e massima 1 da entrambi i lati, perché ogni anno che appare in una, appare anche nell'altra.

La tabella **Script_pub_key** è legata a **Txout** perché deriva da essa. Possiede infatti tutti gli ScriptPubKey prelevati dall'entità dei transaction outputs, che però non hanno un id di chiave pubblica. Quindi la cardinalità ha come minimo 1 e massimo N dal lato dell'entità **Script_pub_key** in quanto ogni suo elemento deve appartenere anche all'entità **Txout** e può comparire in più transaction outputs. Al contrario, ogni record di quest'ultima può avere un solo corrispettivo, o non averne nessuno, nell'altra entità.

3.2.2 Schema Relazionale

Per evitare confusione, si evita di riportare ogni campo di ogni entità, ma solo quelli più utilizzati durante lo studio. Gli elementi sono elencati divisi fra originali e creati.

Le entità originali sono le seguenti:

block Campi: block_id, block_hash, block_ntime, block_nbits, block_nnonce, block_value_in, block_value_out, ...

Entità che riguarda i blocchi e tutti i loro campi. Può capitare che i blocchi risultino in ordine diverso, se confrontati con quelli di un explorer comune. Questo può essere dovuto al fatto che sui files .dat i blocchi non sono sempre salvati in ordine sequenziale, data la dinamicità con cui la rete viene sviluppata ed allungata. Oppure perché durante il caricamento degli stessi nel database, possa essere avvenuto un qualche tipo di errore. Data la lunghezza e la delicatezza del processo, non è da escludere anche questa opzione. Non risulta comunque un grave problema, perché

all'interno dell'ambiente del database rimane comunque tutto coerente. L'"ordine degli explorer", ovvero l'ordine reale dei blocchi indicato dagli explorer in rete e non quello secondo i loro id, può essere facilmente ricavato sfruttando il campo block_ntime, il quale rappresenta la data in formato timestamp UNIX di creazione del blocco e che permette qundi di ordinare in maniera corretta i blocchi.

Nell'entità non esiste un campo che esplicita l'id del blocco precedente o di quello successivo. Questo perché ciò avrebbe reso la tabella ancora più pesante, aggiungendo un campo le cui informazioni sono in realtà ricavabili in una maniera più leggera. Per verificare l'id del blocco precedente e quello successivo, si possono utilizzare infatti due query particolari.

Quella per il blocco precedente, cerca il block_ntime più grande fra quelli più piccoli rispetto al blocco che si sta analizzando.

SELECT $b2.block\_id$
FROM $block\ b1, block\ b2$
WHERE $b1.block\_ntime > b2.block\_ntime$ **AND** $b2.block\_ntime$ **in** (
SELECT **MAX**($b3.block\_ntime$)
FROM $block\ b3$
WHERE $b3.block\_ntime < b1.block\_ntime$
)
AND $b1.block\_id =$**??**

Mentre la query per ottenere l'id del blocco successivo, consiste nel trovare l'id del blocco che possiede il block_ntime più piccolo fra quelli più grandi, rispetto a quello del blocco in questione.

SELECT $b2.block\_id$
FROM $block\ b1, block\ b2$
WHERE $b1.block\_ntime < b2.block\_ntime$ **AND** $b2.block\_ntime$ **in** (
SELECT **MIN**($b3.block\_ntime$)
FROM $block\ b3$
WHERE $b3.block\_ntime > b1.block\_ntime$
)
AND $b1.block\_id =$**??**

In entrambe, al posto di "**??**" va sostituito l'id del blocco di cui si intendono scoprire il predecessore ed il successore.

Si possono anche utilizzare le funzioni **find_prevblock** e **find_follblock** della sezione 4.2 della libreria **litecoin.py**, che restituiscono rispettivamente l'id del blocco precedente e quello successivo a quello dato, secondo l'ordine degli explorer. Esse sfruttano le query appena spiegate per assolvere il loro compito.

block_tx Campi: block_id, tx_id, tx_pos

Mette in collegamento i blocchi con le loro rispettive transazioni attraverso i loro id. La libreria ha evitato di salvare l'id del blocco direttamente dentro la transazione, probabilmente per rendere la struttura il più generale possibile, per permettere la memorizzazione anche di altri tipi di blockchain.

chain Campi: chain_id, chain_name, chain_address_version, chain_last_block_id, ...

Entità che rappresenta le caratteristiche generali della catena. Serve per distinguere i blocchi appartenenti a blockchain diverse, dato che Abe può funzionare anche con altri tipi di catene. Per ogni catena viene salvato anche l'id del primo blocco nel campo chain_last_block_id, nonostante il nome.

chain_candidate Campi: chain_id, block_id, block_height, ...

Indica la relazione fra block e chain.

multisig_pubkey Campi: multisig_id, pubkey_id

Indica la relazione fra le chiavi pubbliche multi-firma e le loro rispettive componenti, attraverso i loro id.

orphan_block Campi: block_id, block_hashprev

Entità che rappresenta un blocco orfano.

Un discorso interessante va fatto sugli hash di quest'entità e di **Block**. I blocchi orfani utilizzano un formato diverso rispetto a quello dei blocchi tradizionali. Inoltre l'hash dei blocchi orfani indica il blocco precedente a loro, secondo l'"ordine degli explorer".

Il formato dell'hash dei blocchi orfani è ottenuto applicando l'algoritmo **sha-256** mentre quello dei blocchi della catena è **scrypt**. Ogni volta che un blocco viene salvato nel database, l'hash del blocco precedente, in formato **sha-256**, viene prelevato e salvato nell'entità **Orphan_block**, dove è presente l'orfano con il suo identico id. L'hash del blocco corrente, calcolato in formato **scrypt**, invece viene salvato nell'entità **Block**.

Solitamente il formato **scrypt** viene utilizzato nel calcolo del PoW mentre **sha-256** per tutti gli altri scopi.[34] Gli explorer tendono ad usare il formato **sha-256**, in quanto è lo stesso di Bitcoin, e quindi la libreria ha deciso di tenerlo nell'entità degli orfani per avere una maggiore tracciabilità degli stessi, mentre **scrypt** è più utile nell'entità dei blocchi perché si occupa del PoW, che negli orfani risulterebbe invece totalmente inutile. I due standard sono convertibili attraverso la funzione **convert_blockhash** spiegata nella sezione 4.2. Essa sfrutta il campo block_ntime dell'entità dei blocchi e block_hashprev di quella degli orfani per effettuare la conversione.

pubkey Campi: pubkey_id, pubkey_hash, pubkey

Rappresenta la chiave pubblica, contiene anche l'hash.

tx Campi: tx_id, tx_hash, tx_version, tx_size, ...

Entità delle transazioni, contiene l'hash e l'id, utile per tracciare i transaction inputs ed outputs che la compongono.

txin Campi: txin_id, tx_id, txin_pos, txout_id, txin_scriptsig, txin_sequence

Transaction inputs, contiene anche l'id della transazione di cui fa parte.

txout Campi: txout_id, tx_id, txout_pos, txout_value, txout_scriptpubkey, pubkey_id

Transaction outputs, contiene anche l'id della transazione di cui fa parte.

[34] Fonte - https://litecoin.info/index.php/Block_hashing_algorithm

Le due views permettono di ricavare e riunire insieme moltissime informazioni riguardo le transazioni. La loro importanza è quindi strategica nell'interrogazione del database perché permette di mettere in relazione l'entità **Balance** con le entità **Txin** e **Txout** evitando un lungo percorso di join di tabelle, che potrebbe risultare oneroso e poco intuitivo.

Le due views sono:

txin_detail Campi: chain_id, block_id, block_hash, tx_pos, tx_id, tx_hash, tx_version, tx_size, txin_id, txin_pos, prevout_id, txin_scriptsig, txin_sequence, txin_value, txin_scriptpubkey, pubkey_id, pubkey_hash, pubkey, ...

Vista che contiene tutti i dettagli sui transaction inputs. Molto utile per le interrogazioni.

txout_detail Campi: chain_id, block_id, block_hash, tx_pos, tx_id, tx_hash, tx_version, tx_size, txout_id, txout_pos, txout_value, txout_scriptpubkey, pubkey_id, pubkey_hash, pubkey, ...

Vista che contiene tutti i dettagli sui transaction outputs. Molto utile per le interrogazioni.

Le entità create sono invece le seguenti:

balance Campi: address, count_ltc, pubkey_hash, scriptpubkey

Descrive il bilancio di un indirizzo. Funziona con le funzioni della libreria **litecoin.py** spiegate nella sezione 4.2. Le sue funzioni permettono l'interazione accurata con questa entità. Il campo del conteggio dei tokens, viene effettuato attraverso una funzione della libreria del capitolo 4, che effettua una query apposita sulle viste e salva il risultato. I campi pubkey_hash e scriptpubkey vengono prelevati, attraverso le funzioni, direttamente dalle entità **Pubkey** e **Script_pub_key**. Una volta inserito un indirizzo tramite la funzione **countLTC** della libreria, viene salvato qui col suo bilancio e, a seconda del tipo di indirizzo, viene riempito un campo fra pubkey_hash e scriptpubkey e l'altro viene lasciato null.

Codice SQL di creazione:

CREATE TABLE *balance* (
address **VARCHAR**(100) **PRIMARY KEY**,
count_ltc **BIGINT**,
pubkey_hash **CHAR**(40),
scriptpubkey **VARCHAR**(2000000)
)

price Campi: price_date, price_pr, price_day, price_epoch_min, price_epoch_max, price_chain_id

Rappresenta tutti i prezzi, giorno per giorno, con la loro data. È il risultato dello storico salvato in una tabella.

Codice SQL di creazione:

CREATE TABLE *price* (
price_date **DATE**,

price_pr **NUMERIC**(6,3),
price_day **VARCHAR**(10),
price_epoch_min **INTEGER**,
price_epoch_max **INTEGER**,
price_chain_id **NUMERIC**(10,0)
)

Per riempire questa tabella bisogna usare il programma **prices.py** trovabile su **GitHub**, all'indirizzo **https://github.com/smigliorini/blockchain/tree/master/litecoin**. Esso accetta come argomento da linea di comando il nome del file contenente lo storico dei prezzi ed esegue una media del valore massimo e minimo giornalieri presenti nello storico, popolando poi la tabella. L'id della catena va inserito manualmente a parte nel DB. È consigliabile usare uno storico con la stessa struttura di quello usato da me.[33]

script_pub_key Campi: scriptpubkey

Contiene tutti gli ScriptPubKey che non hanno un corrispettivo pubkey_id, prelevati dall'entità **Txout**. Sono tutti potenziali ScriptPubKey da cui si possono ricavare indirizzi SegWit, dato che essi non possiedono una chiave pubblica. L'utilità di questa entità deriva dal fatto di rappresentare in una sola tabella tutti gli ScriptPubKey utili, senza dover ogni volta eseguire una query per escludere i valori di nessuna utilità.

Codice SQL di creazione:

CREATE TABLE *script_pub_key* (
scriptpubkey **VARCHAR**(2000000)
)

La tabella va riempita con la query:

INSERT INTO *script_pub_key*
SELECT DISTINCT *txout_scriptpubkey*
FROM *txout*
WHERE *pubkey_id* **is null**

year Campi: year_yr, epoch_min, epoch_max

Contiene gli anni dal 2010 al 2020, con il loro epoch all'inizio dell'anno ed alla fine. Servono per fare dei raggruppamenti, sfruttando il campo block_ntime dell'entità block, svolgendo query di tipo statistico.

Codice SQL di creazione:

CREATE TABLE *year* (
year_yr **INTEGER**,
epoch_min **INTEGER**,
epoch_max **INTEGER**
)

La tabella va riempita avviando il programma **year.py** trovabile su **GitHub**, all'indirizzo **https://github.com/smigliorini/blockchain/tree/master/litecoin**.

year_month Campi: year_month_yr, year_month_mn, epoch_min, epoch_max

Contiene gli anni divisi in mesi dal 2010 al 2020, ed il loro epoch all'inizio ed alla fine del mese. Ha la stessa funzione dell'entità year.

Codice SQL di creazione:

CREATE TABLE *year_month* (
year_month_yr **INTEGER**,
year_month_mn **INTEGER**,
epoch_min **INTEGER**,
epoch_max **INTEGER**
)

La tabella va riempita poi avviando il programma **month.py** trovabile su **GitHub**, all'indirizzo **https://github.com/smigliorini/blockchain/tree/master/litecoin**.

3.3 Note su come aggiornare il database

Conoscendo la natura della blockchain, si potrebbe voler aggiornare il database. In questo caso, il primo passo sarebbe quello di installare i nuovi blocchi, scaricandoli nella cartella della chain. Dopodiché il secondo passo sarebbe quello di avviare una versione configurata appositamente di Abe, con il comando **python -m Abe.abe –config abe-pg.conf –commit-bytes 100000 –no-serve**.

Gli ultimi passaggi sono facoltativi, ovvero aggiornare le entità create. La prima su cui agire è quella dei prezzi. Bisognerebbe scaricare uno storico dei prezzi più recente, ed inserirlo nell'entità **price**, sostituendo quello vecchio. La seconda sarebbe **year**, inserendo gli anni che mancano ed il loro corrispettivo epoch di inizio e fine. Ho scritto un programmino in Python, che accetta l'intervallo di anni che si vogliono inserire, passando da terminale l'anno d'inizio intervallo e l'anno di fine. Se si prova ad inserire un anno che è già presente, allora viene stampato un errore. Stesso discorso vale per **year_month**. I due files si chiamano rispettivamente **update_year.py** e **update_year_month.py**. Vanno poi aggiornati i bilanci dell'entità **balance**, attraverso la funzione **remake_count** della libreria scritta da me, **litecoin.py**. Tutti i sorgenti elencati sono trovabili su **GitHub**, all'indirizzo **https://github.com/smigliorini/blockchain/tree/master/litecoin**. L'ultima entità da aggiornare sarebbe **script_pub_key**. Il processo può essere lungo e si attua attraverso la query:

INSERT INTO *script_pub_key*

SELECT DISTINCT *txout_scriptpubkey*

FROM *txout*

WHERE *pubkey_id* **is null AND** *txout_scriptpubkey* **not in** (

SELECT *scriptpubkey*

FROM *script_pub_key*

)

Un ulteriore interessante aggiornamento riguarda l'inserimento di una nuova catena nel database ed il confronto con quella di Litecoin già presente.

Capitolo 4

Libreria

Dopo aver sfruttato le informazioni acquisite studiando la struttura del database nelle sue parti più interessanti, ho scritto in Python una libreria di funzioni, **litecoin.py**. Si può scaricare da **GitHub**, all'indirizzo **https://github.com/smigliorini/blockchain/tree/master/litecoin**. Essa interagisce con la base di dati e permette di eseguire indagini più complesse, sfruttando le opprtunità messe a disposizione dal linguaggio di programmazione.

4.1 Funzioni che non richiedono cursori

Queste sono quelle che funzionano anche senza una connessione, basta passare i parametri corretti.

addr_funct (hashpubkey, pr)

Funzione basata sugli algoritmi 2 e 3. Se pr è un carattere fra **L, m, M, 3, 2, Q, 4** oppure se è un carattere fra **30, 6f, 32, 05, c4, 3a, 08** allora restituisce l'indirizzo costruito a partire dall'hashpubkey. Se pr è **l** oppure True invece, viene restituito un indirizzo cominciante con **ltc1**, se **t** oppure False, con **tltc1**. In questi due casi hashpubkey deve essere uno scriptpubkey della lunghezza corretta. In tutti gli altri casi viene restituito **None**.

getAddress (pubkey, pr)

Funzione basata sull'algoritmo 1. Svolge i passaggi per trasformare una pubkey in un hashpubkey, poi chiama **addr_funct** passandole gli argomenti necessari.

4.2 Funzioni che richiedono cursori

Per utilizzare queste funzioni, è necessario stabilire una connessione col database attraverso la libreria **psycopg2** di python e passare il cursore come parametro.

find_hpub (cur, address, isMS)

Funzione che permette di risalire all'hash della chiave pubblica, fornendo un indirizzo ed il cursore da utilizzare nella ricerca. La funzione riconosce il tipo dell'indirizzo cercato, confrontando il primo carattere con i pattern conosciuti. Il parametro isMS è un booleano che indica se l'indirizzo cercato è multi-firma, oppure no. Il valore di default è False. La funzione richiama tutte le chiavi pubbliche e

confronta l'indirizzo indicato con il risultato ottenuto dall'applicazione della funzione **addr_funct** su ogni chiave pubblica. In caso di indirizzi SegWit, vengono chiamati gli ScriptPubKey dell'entità **script_pub_key**. La ricerca può essere lenta. Se isMS è True, vengono cercate solo le chiavi pubbliche, il cui id compare nel campo **multisig_id** dell'entità **multisig_pubkey**, rendendo la ricerca molto più veloce. Il risultato che viene restituito è la chiave pubblica, o lo scriptpubkey, e un booleano che indica se l'indirizzo è SegWit oppure no. In caso qualche parametro non soddisfi le ricerche, viene restituito **None,None**.

La particolarità di questa funzione è lo svolgere una mansione controcorrente. Infatti non è generalmente possibile risalire ad una chiave pubblica da un indirizzo.

countLTC (cur, pubkey_hash, address, isSW, update)

Riceve come argomenti il cursore, l'hash della chiave pubblica o lo scriptpubkey, l'indirizzo, un booleano che indica se l'indirizzo è SegWit ed un altro che indica se fare l'update oppure no. Update è False di default. Attraverso query di addizione, questa funzione calcola il bilancio di un indirizzo, sommando tutti valori dei transaction inputs in cui appare l'hash della chiave pubblica, o lo scriptpubkey, cercata. Svolge poi lo stesso procedimento con i transaction outputs ed a queste sottrae il valore degli inputs. Il risultato viene salvato nell'entità **balance**, insieme all'indirizzo ed all'hash della chiave pubblica o lo scriptpubkey. Se update è True, allora viene cercato l'indirizzo in **balance** e ne viene aggiornato il bilancio relativo, invece di inserire una nuova riga nella tabella.

Dato che i valori LTC scambiati fra le transazioni sono molto piccoli, essi sono maggiorati rispetto al loro reale valore. I numeri che si vedono nel database, vanno moltiplicati per 10^{-8} per ottenere il reale valore. Il conteggio che viene salvato in tabella è quello che si trova nella chain, mentre quello restituito dalla funzione è quello reale.

getLTCcount (cur, address)

Partendo dall'indirizzo fornito, ricerca il bilancio nell'entità **balance**. In caso affermativo, lo trasforma nel reale valore e lo restituisce. In caso negativo, richiama **find_hpub** passandole il cursore e l'indirizzo. Se la ricerca non dà risultati, allora restituisce **None**. In caso contrario, prova a cercare se la chiave pubblica o lo script trovati sono comunque presenti in **balance**. In caso affermativo, significa che il bilancio relativo è già presente, ma con un indirizzo diverso, quindi converte il conteggio in valore reale e lo restituisce, senza quindi salvare un nuovo record nell'entità dei bilanci. In caso negativo, richiama la funzione **countLTC**, ottenendo il conteggio che poi restituisce.

getLTCcount_pubkeyhash (cur, pubkey_hash, pr)

Funzione che lavora in modo simile a **getLTCcount**. Prima cerca in **balance**, altrimenti su tutto il database. Il valore di pr deve essere **L, m, M, 3, 2, Q, 4, l, t**, ed in base a questo decide dove cercare. Se pr è **l** o **t**, allora cerca fra gli scriptpubkey, altrimenti fra i pubkey_hash. Poi viene chiamata la funzione **countLTC**.

Se tutto va per il meglio, la funzione restituisce indirizzo e valore del bilancio reale. Se pr non rispetta i pattern, allora viene sollevata un eccezione. Se il valore del parametro pubkey_hash non viene trovato, allora la funzione restituisce **None,None**.

remake_count (cur, address)

Esegue solo se address è già presente in **balance**, altrimenti restituisce **None**. In base al primo carattere di address ricava il pubkey_hash o lo scriptpubkey da **balance**, dà valore ad un booleano isSW e poi chiama la funzione **countLTC**, passandole tutti i parametri necessari. In questo caso la funzione del conteggio viene eseguita con update uguale a **True**, in modo tale da aggiornare il bilancio dell'indirizzo scelto, invece che inserire una nuova riga.

find_follblock (cur, block_id)

Accetta come argomento un id di un blocco del database, e ritorna l'id del blocco successivo nel database, nell'ordine degli explorer, svolgendo una query. Sfrutta il campo block_ntime per ricavare l'ordine dei blocchi. La query utilizzata consiste nel richiamare tutti i blocchi con timestamp maggiore del blocco scelto e poi fra questi scegliere quello col tempo minore. La query in questione è descritta nella sottosezione 3.2.2. Ritorna **None** se l'id del blocco passato come argomento non esiste nel database o se il blocco non ha un successore.

find_prevblock (cur, block_id)

Accetta come argomento un id di un blocco del database, e ritorna l'id del blocco precedente nel database, nell'ordine degli explorer, svolgendo una query. Sfrutta il campo block_ntime per ricavare l'ordine dei blocchi. La query utilizzata consiste nel richiamare tutti i blocchi con timestamp minore del blocco scelto e poi fra questi scegliere quello col tempo maggiore. La query in questione è descritta nella sottosezione 3.2.2. Ritorna **None** se l'id del blocco passato come argomento non esiste nel database, o se il blocco non ha un predecessore.

convert_blockhash (cur, block_hash)

Accetta in input un hash di blocco **scrypt** e lo converte in un hash **sha-256**, attraverso una query. Prima di tutto, cerca l'id il successore del blocco nell'ordine degli explorer, con la funzione **find_follblock**, dopodiché richiama l'hash dal campo block_hashprev dall'entità **Orphan_block**. L'unico blocco di cui non può convertire l'hash è l'ultimo, in quanto privo di successore. Ciò viene risolto con una condizione particolare nel codice della funzione, che permette di sfruttare la libreria python di **BLOCKCYPHER**[21] [22], che permette di comunicare con l'explorer e restituire l'hash corretto. In questo caso viene prima cercato l'hash di una transazione appartenente al blocco, poi viene cercata la stessa transazione nell'explorer e ne viene restituito l'hash del blocco. Questa condizione sfrutta il fatto che ogni transazione è identificata univocamente dal suo hash e che può appartenere ad un blocco soltanto. Se l'hash del blocco passato come argomento non esiste nel database, ritorna **None**.

query (cur, statem)

Funzione molto basilare, che permette di eseguire una query, mandandola al database e stampando a video il risultato.

Capitolo 5

Analisi dei dati

5.1 Studio del valore nel tempo

Dall'analisi del database ho potuto, attraverso query, fare degli studi di tipo statistico. Una delle prime indagini che ho compiuto è stata l'analisi del cambio dollari/LTC nel tempo, attraverso un grafico.

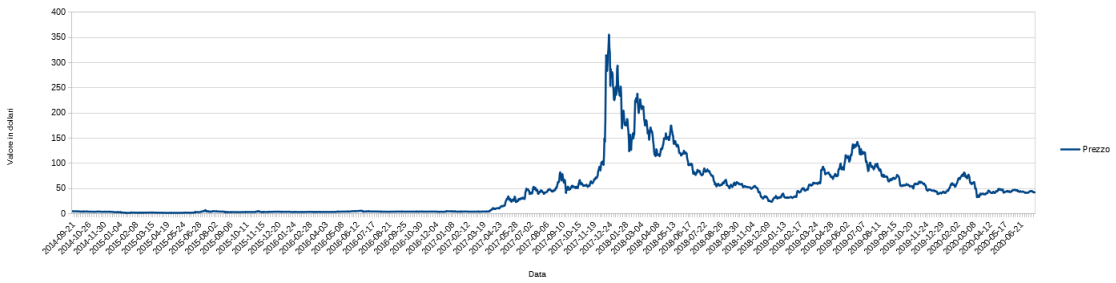

Figura 5.1: Grafico dei prezzi

Dall'osservazione della figura 5.1 si possono fare alcune considerazioni. Il valore di Litecoin è partito basso e lo è rimasto per qualche anno. Agli inizi del 2017 ha cominciato a salire moderatamente, poi ha raggiunto un picco a fine estate, quindi è sceso brevemente. Poco dopo, ha subito una spinta notevole fino a raggiungere il suo massimo valore a fine anno. In seguito, è cominciata una lenta discesa che ha portato il valore a stabilizzarsi intorno ai numeri che osserviamo ancora oggi. Per ora la situazione è abbastanza stabile.

Il boom di Litecoin del 2017 può essere dovuto ad un evento simile, avvenuto nello stesso periodo a Bitcoin. Anch'esso, infatti, ha subito un'impennata nel suo valore, trainando la quasi totalità del mercato delle criptovalute. Il repentino decollo di Bitcoin ha diverse cause e diversi fattori hanno contribuito a questa salita. Per capirle, bisogna contestualizzare il periodo in cui il boom è avvenuto.

Nel 2017 sono avvenuti vari eventi di estrema importanza, che potrebbero aver manifestato la loro influenza anche sull'economia. Ad esempio, l'elezione dell'europeista Emmanuel Macron alla presidenza di Francia, ha portato i mercati europei al rialzo ed al rafforzamento nella fiducia della moneta unica. Un altro fattore potrebbe essere il taglio delle tasse, varato dall'amministrazione Trump. Nonostante questi fattori possano sembrare

scollegati dal valore delle criptomonete, in realtà non è così. Essendo Bitcoin una moneta diffusa online, in tutto il mondo, e con un valore molto volatile, risente maggiormente degli eventi che succedono nel pianeta.

Uno studio interessante svolto dall'University of Texas, ha analizzato questa ascesa repentina. Gli studiosi hanno stabilito che, in realtà, potrebbe essere frutto di un'attenta manipolazione del mercato. Per controllare ed alzare il valore infatti, sarebbe stata impiegata un'altra criptovaluta, Tether. Questa non è una moneta digitale tradizionale, ma invece una stablecoin, il cui valore è legato alle monete tradizionali. Essa sarebbe stata sistematicamente impiegata per acquistare coins di Bitcoin, nei momenti in cui il loro valore cominciava a scendere. Questo processo avrebbe guidato il mercato, causandone l'ascesa di prezzo.[35]

5.2 Studio transaction inputs e outputs

Un'analisi svolta sul database sulle transazioni mi ha permesso di indagare meglio le conseguenze del picco dei prezzi del 2017.

Ho svolto infatti un'interrogazione, che mostrasse il numero di txins e txouts per ogni anno e mese a partire dal 2011, utilizzando due query e poi unendo i dati. La prima riguarda i transaction inputs:

SELECT $year\_month\_yr, year\_month\_mn,$ **COUNT**$(txin\_id)$

FROM $year\_month$ **JOIN** $block$ **ON** $block\_ntime >= epoch\_min$ **AND** $block\_ntime < epoch\_max$ **JOIN** $txin\_detail$ **ON** $txin\_detail.block\_id = block.block\_id$

GROUP BY $year\_month\_yr, year\_month\_mn$

ORDER BY $year\_month\_yr, year\_month\_mn$

Tempo di esecuzione: $718774.392ms$ $(11 : 58.774)$

Quella per i transaction outputs è molto simile:

SELECT $year\_month\_yr, year\_month\_mn,$ **COUNT**$(txout\_id)$

FROM $year\_month$ **JOIN** $block$ **ON** $block\_ntime >= epoch\_min$ **AND** $block\_ntime < epoch\_max$ **JOIN** $txout\_detail$ **ON** $txout\_detail.block\_id = block.block\_id$

GROUP BY $year\_month\_yr, year\_month\_mn$

ORDER BY $year\_month\_yr, year\_month\_mn$

Tempo di esecuzione: $413256.646ms$ $(06 : 53.257)$

Dall'unione dei due risultati ho prodotto un file .csv da cui poi ho ricavato un grafico, la figura 5.2.

Osservando il grafico si possono constatare un paio di cose. In principio, i transaction outputs sono molto maggiori degli inputs. Questo perché all'inizio della storia della catena, nessuno possedeva monete da poter dare agli altri. Tutti i transaction outputs, ricevono moneta direttamente dalla coinbase, quindi si ha un numero molto piccolo di

[35] Kate Rooney, "Much of bitcoin's 2017 boom was market manipulation, research says" - https://www.cnbc.com/2018/06/13/much-of-bitcoins-2017-boom-was-market-manipulation-researcher-says.html

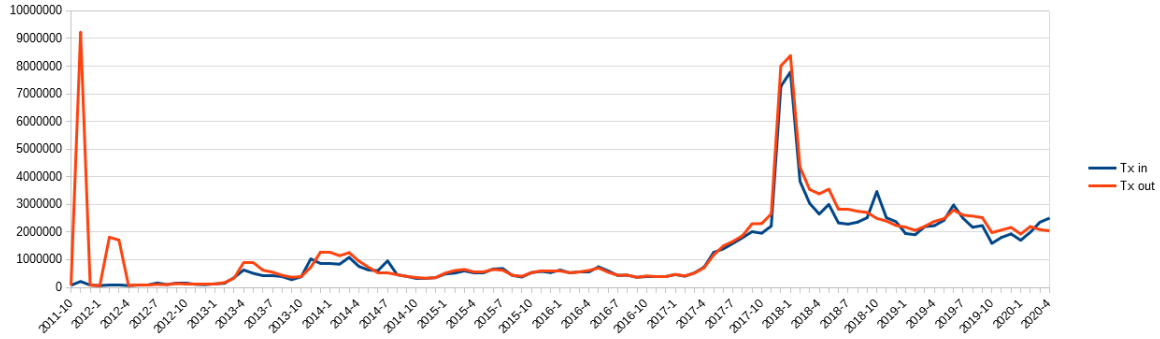

Figura 5.2: Grafico dei transaction inputs e outputs

transaction inputs che però muovono grandi quantità di tokens, per un numero molto più elevato di transaction outputs.

Dopo circa un anno, input e output hanno iniziato a muoversi assieme. Spesso il numero degli outputs supera gli inputs, ma comunque tendono a mantenere gli stessi livelli. Questo può anche essere dovuto al fatto che, in una transazione, tutta la moneta che entra deve anche uscire. Ciò può influire quindi in parte sul numero di transaction inputs e outputs.

Molto interessante è il picco del 2017. Quando il prezzo ha avuto quell'ascesa vertiginosa, anche il volume degli scambi è aumentato. Dopodiché, il valore ha cominciato a scendere ed a stabilizzarsi, esattamente come il numero di transazioni.

5.3 Analisi numero blocchi

Un'altra indagine interessante è lo studio del numero di blocchi minati nel tempo. Ho individuato i valori dividendoli in mese per mese, attraverso la query:

SELECT $year\_month\_yr, year\_month\_mn$, **COUNT**$(block\_id)$

FROM $year\_month$ **JOIN** $block$ **ON** $block\_ntime >= epoch\_min$ **AND** $block\_ntime < epoch\_max$

GROUP BY $year\_month\_yr, year\_month\_mn$

ORDER BY $year\_month\_yr, year\_month\_mn$

Tempo di esecuzione: $27949.533ms$ $(00 : 27.950)$

Come si può osservare nella figura 5.3, l'andamento dei blocchi si comporta in modo abbastanza stabile nel corso del tempo. Il numero di blocchi minati per mese oscilla sempre fra 15000 e 20000. Si può osservare come all'inizio il valore sia particolarmente alto, al contrario del resto del grafico. Questo può essere spiegato con il fatto che la catena era ancora agli inizi, quindi probabilmente la spinta inziale fu voluta e calcolata apposta per poter avviare la catena. Ricordiamo infatti che Litecoin ha origini note e pubbliche, quindi può essere plausibile un comportamento del genere da parte degli sviluppatori.

Un'altra spiegazione, è che un inizio così esplosivo sia dovuto alla maggiore facilià nella risoluzione del POW, quando la catena era ancora agli inzi, unito magari ad un probabile interesse dell'utenza verso quella che era una nuova criptovaluta. La stabilità del

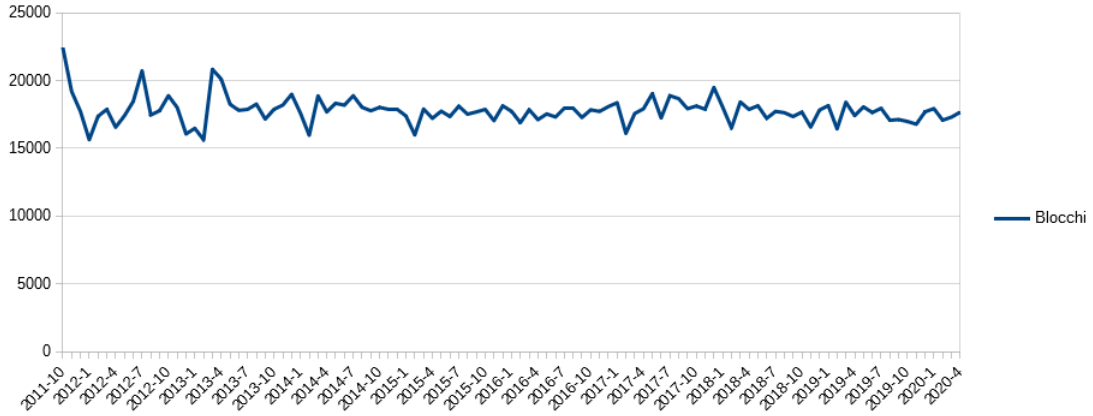

Figura 5.3: Grafico dei blocchi

grafico inoltre, indica un fattore molto importante: nonostante siano passati anni, i blocchi vengono ancora minati con la stessa frequenza, che si mantiene comunque elevata. Questo significa che il valore di un blocco minato ogni 2,5 minuti si mantiene costante.

5.4 Studi sugli indirizzi multi-firma

Altri piccoli studi, ma comunque interessanti, che ho svolto riguardano alcuni conteggi.

Ad esempio, con la query

SELECT COUNT (**DISTINCT** $multisig\_id$)

FROM $multisig\_pubkey$

ho contato il numero di chiavi a cui è associato un indirizzo multi-firma, ovvero 124.

Al contrario, contando il numero totale di chiavi pubbliche del database con l'interrogazione:

SELECT COUNT($pubkey\_id$)

FROM $pubkey$

il risultato è stato di 48902904.

Quindi significa che solo lo $0,000253\%$ delle chiavi nel database appartiene ad un indirizzo multi-firma.

Invece, con la query:

SELECT $multisig\_id$, **COUNT**($pubkey\_id$)

FROM $multisig\_pubkey$

GROUP BY $multisig\_id$

Ho potuto constatare che la maggior parte delle chiavi multi-firma è associata a due indirizzi, con eccezione di rari casi in cui il conteggio sale a tre. Questo potrebbe essere indice di una particolare tendenza dell'utenza, a preferire indirizzi "2 a 2" rispetto a tutte le altre possibilià del loro utilizzo. Questo tipo di indirizzi permette a due utenti di condividere un account, i cui fondi possono venire sbloccati solo con entrambe le chiavi possedute, una a testa, dagli intestatari.

Capitolo 6

Sitografia

- https://stormgain.com/it/blog/what-litecoin-cryptocurrency
- https://www.comefaretradingonline.com/criptovalute/litecoin.php
- https://www.criptovalute24.com/litecoin/#Litecoin_storia
- https://it.wikipedia.org/wiki/Litecoin
- https://it.wikipedia.org/wiki/Blockchain
- https://medium.com/@satoshiwantsyou/come-funziona-il-proof-of-work-bitcoin-blockchain-eac6ba859253
- https://www.blockchain4innovation.it/esperti/blockchain-perche-e-cosi-importante/
- https://it.cointelegraph.com/explained/proof-of-work-explained
- https://it.wikipedia.org/wiki/Crittografia_asimmetrica
- https://bitcoin.stackexchange.com/questions/65282/how-is-a-litecoin-address-generated
- https://github.com/bitcoin-abe/bitcoin-abe
- https://github.com/mrqc/bitcoin-blk-file-reader
- https://github.com/blockchain-unica/blockapi
- https://github.com/blkchain/blkchain
- https://github.com/bitcoinjs/fast-dat-parser
- https://github.com/litecoin-project/litecore-lib
- https://github.com/losh11/python-litecoin-blockchain-parser
- https://it.wikipedia.org/wiki/Firma_digitale
- http://www.10bitcoin.it/firma-digitale/
- https://learnmeabitcoin.com/beginners/digital_signatures#:~:text=A%20digital%20signature%20is%20something,digital-%20signature%20to%20prove%20it.

- https://www.ledger.com/academy/difference-between-segwit-and-native-segwit
- https://www.altcoinbuzz.io/bitcoin-and-crypto-guide/crypto-mainnet-vs-testnet/
- https://bitcoin.stackexchange.com/questions/91748/how-to-use-python-reference-for-encoding-a-bech32-address
- https://www.cnbc.com/2018/06/13/much-of-bitcoins-2017-boom-was-market-manipulation-researcher-says.html
- https://litecoin.info/index.php/Block_hashing_algorithm
- https://live.blockcypher.com/ltc/
- https://github.com/blockcypher/blockcypher-python
- https://blockchair.com/it/litecoin
- https://www.investopedia.com/terms/o/orphan-block-cryptocurrency.asp
- https://bitcoin.stackexchange.com/questions/79820/how-does-the-litecoin-difficulty-get-calculated#:∼:text=Bitcoin%20creates%202016%20blocks%2C%20and,10%20minutes%2C%20the-%20difficulty%20decreases.
- https://www.derpturkey.com/cs-bitcoin-mempool-transaction-selection/
- https://bitcoin.stackexchange.com/questions/73191/where-are-transactions-to-be-confirmed-stored
- https://bitcoin.stackexchange.com/questions/62781/litecoin-constants-and-prefixes
- https://finance.yahoo.com/quote/LTCUSD/ history?period1=1410912000&period2=1598572800-&interval=1d&filter=history&frequency=1d
- https://cryptofacilities.zendesk.com/hc/en-us/articles/360006040974-Litecoin-Address-Format
- https://www.reddit.com/r/litecoin/comments/97gp7p/litecoin_segwit_bech32_address_prefix/