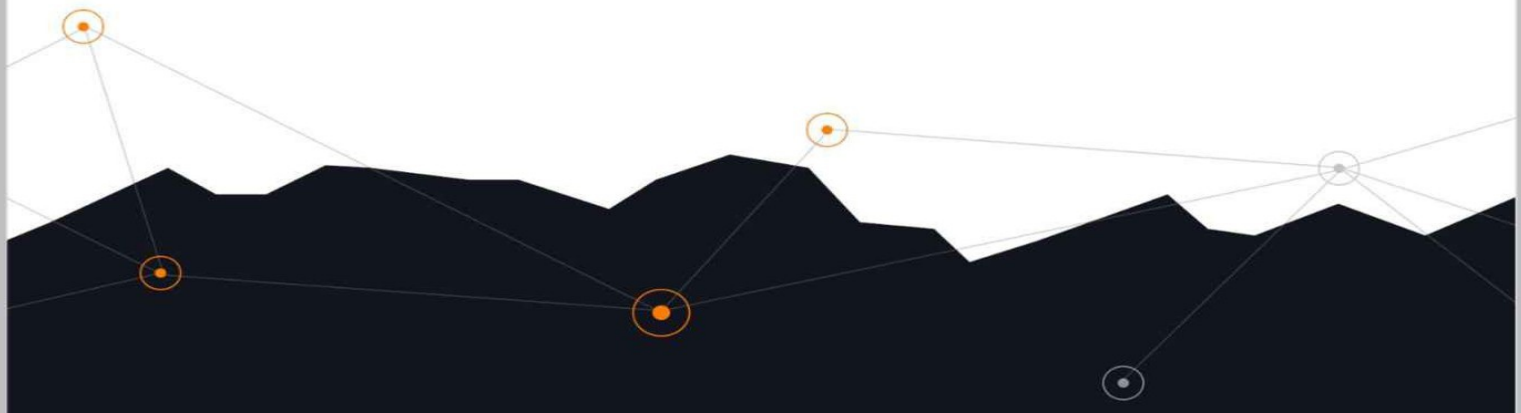
Andreas M. Antonopoulos

Mastering Bitcoin



Traduzione italiana della guida completa al mondo di bitcoin e della blockchain

2ª edizione aggiornata a SegWit e Lightning Network

Mastering Bitcoin Traduzione italiana - 2ª edizione

Andreas M. Antonopoulos e Riccardo Masutti

DEDICA

Dedicato ad Hal Finney (1956-2014), sviluppatore di PGP Corporation e destinatario della prima transazione bitcoin di Satoshi Nakamoto.

Grazie per tutto ciò che hai fatto per Bitcoin, Hal!

Basato e tradotto dalla 2nd edition, 2nd print di Mastering Bitcoin pubblicata da Andreas M. Antonopoulos.

Ultima revisione: Novembre 2019

Indice

Prefazione

Scrivere il Bitcoin Book

Mi sono imbattuto per la prima volta in bitcoin a metà del 2011. La mia reazione immediata è stata più o meno "Pfft! Una moneta per nerd!" e l'ho ignorata completamente per altri sei mesi, fallendo nel cogliere la sua importanza. Questa è una reazione che ho visto ripetuta anche tra molte delle persone intelligenti che conosco, e ciò mi consola un po'. La seconda volta che ho ri-incontrato bitcoin, in una discussione di una mailing list, ho deciso di leggere il white-paper scritto

da Satoshi Nakamoto, per studiare la sorgente autoritativa e vedere di che si trattasse. Mi ricordo ancora il momento nel quale finii di leggere quelle nove pagine, quando realizzai che bitcoin non era semplicemente una valuta digitale, ma un network basato sulla fiducia che poteva anche fornire le basi per costruire molto di più di semplici valute. La realizzazione che "questa non è una moneta, è un network decentralizzato basato su fiducia," mi ha portato a un viaggio di quattro mesi per divorare ogni frammento di informazione su bitcoin che potevo trovare. Sono rimasto incollato e diventato ossessionato, spendendo 12 o più ore al giorno attaccato allo

schermo, leggendo, scrivendo testo e codice e imparando più che potevo. Sono emerso da questo stato di fuga, dimagrito di 9 chili dalla mancanza di pasti costanti, determinato a dedicare tutto me stesso per lavorare su bitcoin.

Due anni più tardi, dopo aver creato un certo numero di piccole startup per esplorare vari servizi e prodotti relativi al Bitcoin, ho deciso che era il momento di scrivere il mio primo libro. Bitcoin mi ha rapito, gettandomi in un frenetico vortice di creatività; è stata la più eccitante tecnologia che ho incontrato da quando c'è internet.

Pubblico di

Riferimento

Questo libro è prevalentemente inteso per i programmatori. Se tu sai usare un linguaggio di programmazione, questo libro ti insegnerà come funzionano le monete basate su algoritmi crittografici, come usarle e come sviluppare software che lavori con esse. I primi capitoli sono adatti anche come introduzione approfondita per Bitcoin per i non programmatori e per coloro che cercano di capire il funzionamento interno di Bitcoin e delle crittovalute.

Convenzioni usate in questo libro

Le seguenti convenzioni tipografiche sono usate in questo libro:

Italic

Indica nuovi termini, URLs, indirizzi email, filename, e estensioni.

Larghezza costante

Usata per listati di programmi, come per paragrafi che si riferiscono a elementi di programmi come nomi di funzioni o variabili, database, tipi di dati, variabili d'ambiente, istruzioni e parole chiave.

Larghezza costante in grassetto

Mostra comandi o altro testo che dovrebbe essere digitato

letteralmente dall'utente.

Larghezza costante in corsivo

Mostra testo che dovrebbe essere sostituito con valori definiti dall'utente o con valori determinati dal contesto.

TIP	Questa voce sta a significare un tip/suggerimento.
	Questa voce sta

NOTA	a significare una nota.
ATTENZIONE	Questa icona indi un avvertimen o cautela.

Esempi di Codice

Gli esempi sono illustrati in Python, C++, usando la riga di comando di un sistema operativo Unix-like come Linux o MacOS. Tutti I frammenti di codice sono disponibili nel repository

GitHub (https://github.com/bitcoinbook/bitcoinb nella directory code del repo principale. Fai un fork del codice del libro, prova gli esempi di codice o sottoponi delle correzioni via GitHub.

Tutti i frammenti di codice possono essere replicati in gran parte dei sistemi operativi con un'installazione minimale di compilatori e interpreti per i rispettivi linguaggi. Dove necessario, forniamo istruzioni d'installazione basilari ed esempi passo-passo dell'output di quelle istruzioni.

Alcuni dei frammenti di codice e degli output sono stati riformattati per la stampa. In tutti questi casi le linee

sono state divise da una barra rovesciata (\), seguita da un carattere newline. Quando si trascrivono gli esempi si rimuovano questi due caratteri e si uniscano di nuovo le linee; si dovrebbero vedere risultati identici a quelli nell'esempio.

Tutti i frammenti di codice usano dove possibile valori reali e calcoli, in modo che si possa costruire di esempio in esempio e vedere gli stessi risultati in ogni codice si scriva per calcolare gli stessi valori. Per esempio, le chiavi private e le corrispondenti chiavi pubbliche e indirizzi sono tutti reali. Le transazioni di prova, i blocchi, e i riferimenti in blockchain sono stati tutti introdotti

nella reale catena dei blocchi bitcoin e sono parte del registro pubblico; dunque si possono consultare in qualsiasi sistema bitcoin.

Utilizzare gli Esempi di Codice

Questo libro è qui per aiutarti a fare il tuo lavoro. In generale, se viene offerto del codice di esempio con questo libro, è possibile utilizzarlo nei tuoi programmi e documentazione. Non è necessario contattarci per ottenere il permesso, a meno che tu non stia riproducendo una parte significativa del codice. Ad esempio, la scrittura di un programma che utilizza diversi

blocchi di codice da questo libro non richiede il permesso. La vendita o la distribuzione di un CD-ROM di esempi tratti dal libro richiede il permesso. Rispondere a una domanda citando questo libro e citando il codice di esempio non richiede il permesso. Includere una quantità significativa di codice di esempio da questo libro nella documentazione del prodotto richiede un'autorizzazione.

Apprezziamo, ma non richiediamo, un'attribuzione. Un'attribuzione include solitamente Il titolo, l'autore e l'ISBN. Per esempio: "Mastering Bitcoin – Traduzione Italiana (2a edizione) di Riccardo Masutti, ISBN 9781081849115."

Indirizzi e Transazioni Bitcoin in Questo Libro

Gli indirizzi bitcoin, le transazioni, le chiavi, i codici QR e i dati blockchain utilizzati in questo libro sono per la

maggior parte reali. Ciò significa che puoi sfogliare la blockchain, esaminare le transazioni offerte come esempi, recuperarle con i tuoi script o programmi, ecc.

Tuttavia, nota che le chiavi private utilizzate per costruire gli indirizzi sono stampate in questo libro o sono state "bruciate". Ciò significa che se invii denaro a uno di questi indirizzi, i soldi saranno persi per sempre, o in alcuni casi tutti quelli che leggeranno questo libro potranno impossessarsene utilizzando le chiavi private stampate qui.

	NON INVIARE

ATTENZIONE	SOLDI A NESSUNO DEGLI INDIRIZZI IN QUESTO LIBRO. I tuoi soldi saranno presi da un altro lettore e persi per sempre.

Contattare l'Autore della versione italiana

Puoi contattarmi al mio sito web personale:
https://www.riccardomasutti.com

PGP A3D9 93A5 537F 982B - https://keybase.io/riccardomasutti

Seguimi su Twitter: https://twitter.com/riccardomasutti/

Seguimi su Linkedin: https://www.linkedin.com/in/riccardom

Questo libro è stato frutto di diversi mesi di lavoro. La traduzione italiana della prima edizione è stata portata avanti da Riccardo Masutti e da molti altri traduttori sulla piattaforma Transifex.com; la seconda edizione è stata revisionata e tradotta nella sua completezza da Riccardo Masutti, aggiornata con le ultime innovazioni di Bitcoin. Puoi effettuare una donazione e supportare l'autore della versione

italiana inviando la quantità di BTC che desideri a questo indirizzo:

bc1qgcvzk6wk8w95h545s3tttna03nge04

Riconoscimenti

Questo libro rappresenta gli sforzi e i contributi di molte persone. Sono grato per tutto l'aiuto che ho ricevuto da amici, colleghi e persino da estranei, che si sono uniti a me in questo sforzo per scrivere il libro tecnico definitivo su criptovalute e bitcoin.

È impossibile fare una distinzione tra la tecnologia Bitcoin e la comunità bitcoin, e questo libro è tanto un prodotto di quella comunità quanto è un libro sulla tecnologia Bitcoin. Il mio lavoro su questo libro è stato incoraggiato, acclamato, sostenuto, e premiato da tutta la comunità bitcoin dall'inizio fino alla fine. Più di ogni altra cosa, questo libro mi ha permesso di essere parte di una comunità meravigliosa per due anni e non posso ringraziarvi abbastanza per avermi accettato in questa comunità. Ci sono troppe persone da menzionare per nome, persone che ho incontrato in occasione di conferenze, eventi, seminari, meetup, pizza e piccoli

incontri privati, così come molti che comunicavano con me via Twitter, su reddit, su bitcointalk.org, e su GitHub che hanno avuto un impatto su questo libro. Ogni idea, analogia, domanda, risposta, e spiegazione che si trova in questo libro è stata, ad un certo punto, ispirata , provata, o migliorata attraverso le mie interazioni con la comunità. Grazie a tutti per il vostro sostegno; senza di voi questo libro non sarebbe nato. Sarò per sempre grato di tutto questo.

Il percorso per diventare un autore inizia molto prima del primo libro, naturalmente. La mia prima lingua (e la scuola) era greca, così ho dovuto fare un corso di scrittura inglese correttive

nel mio primo anno di università. Devo ringraziare Diana Kordas, il mio insegnante di scrittura inglese, che mi ha aiutato a costruire la fiducia e le competenza che ho. Più tardi, come un professionista, ho sviluppato le mie capacità tecniche di scrittura sul tema del data center, scrivendo per la rivista Network World. Devo ringraziare John Dix e John Gallant, che mi hanno dato il mio primo lavoro di scrittura come editorialista al Network World e al mio editore Michael Cooney e il mio collega Johna Fino Johnson che ha curato le mie colonne e le ha rese adatte alla pubblicazione. Scrivere 500 parole a settimana per quattro anni mi ha dato

abbastanza esperienza per prendere in considerazione l'idea di diventare un autore.

Grazie anche a coloro che mi hanno supportato quando ho sottoposto la mia proposta del libro a O'Reilly, per aver provvisto referenze e aver rivisto la proposta. Un ringraziamento, specialmente John Gallant, Gregory Ness, Richard Stiennon, Joel Snyder, Adam B. Ringraziamenti speciali a Richard Kagan e Tymon Mattoszko, che hanno revisionato versioni iniziali della proposta e Matthew Owain Taylor, che ha redatto la proposta.

Grazie a Cricket Liu, autore del titolo O'Reilly DNS e BIND, che mi ha introdotto a O'Reilly. Grazie anche a

Michael Loukides e Allyson MacDonald di O'Reilly, che ha lavorato per mesi per contribuire a rendere questo libro reale. Allyson é stata particolarmente paziente quando le scadenze non sono state mantenute e consegnati in ritardo e di come gli eventi vita siano intervenuti nel nostro programma di lavoro. Per la seconda edizione, ringrazio Timothy McGovern per averne guidato il processo, Kim Cofer per la revisione e Rebecca Panzer per la creazione di nuove illustrazioni e diagrammi.

Le prime bozze dei primi capitoli sono stati le più difficili, perché Bitcoin è un argomento difficile da dipanare. Ogni volta che ho tirato in ballo la

tecnologia bitcoin, ho dovuto tirare ballo tutta la faccenda. Sono più volte rimasto bloccato e un po' scoraggiato mentre lottavo per rendere l'argomento facile da capire e creare una narrazione intorno ad un argomento tecnico così denso. Alla fine, ho deciso di raccontare la storia di bitcoin attraverso le storie delle persone che utilizzano Bitcoin e tutto il libro è diventato molto più facile da scrivere. Devo ringraziare il mio amico e mentore, Richard Kagan, che mi ha aiutato a svelare la storia e superare i momenti di blocco dello scrittore, e Pamela Morgan, che ha esaminato le prime bozze di ogni capitolo della prima e seconda

edizione del libro e ha fatto le domande più difficili per renderlo migliore. Inoltre, grazie agli sviluppatori del gruppo di San Francisco Bitcoin Meetup Dev, Taariq Lewis e Denise Terry per aver contribuito a testare il materiale in anticipo. Grazie inoltre ad Andrew Naugler per la progettazione infografica.

Durante lo sviluppo del libro, ho fatto le prime bozze disponibili su GitHub e invitato il pubblico a commentare. Più di un centinaio di commenti, suggerimenti, correzioni e contributi sono stati presentati in risposta. Tali contributi sono esplicitamente riconosciuti, con i miei ringraziamenti,

in Bozza della Early Release (Contributi da GitHub). Un grazie particolare ai volontari Ming T. Nguyen (1a edizione) e Will Binns (2a edizione), che i sono offerti per la gestione dei contributi GitHub e hanno aggiunto molti contributi significativi.

Quando il libro era nella fase di bozza, è passato attraverso molti round di revisione tecnica. Grazie a Cricket Liu e a Lorne Lantz per la loro revisione approfondita, commenti e supporto.

Molti sviluppatori bitcoin hanno contribuito ad esempi di codice, recensioni, commenti e incoraggiamenti. Grazie a Amir Taaki e Eric Voskuil per gli snippet di codice esempio e per i molti commenti

di pregio; Chris Kleeschulte per aver contribuito all'appendice, Vitalik Buterin e Richard Kiss per aver aiutato con la matematica della curva ellittica ed altri contributi al codice; a Gavin Andresen per correzioni, commenti e per l'incoraggiamento; a Michalis Kargakis per i commenti, contributi e per lo scritto su btcd; e a Robin Inge per le correzioni degli errori che hanno migliorato la seconda edizione. Nella seconda edizione, ho ricevuto un sacco di aiuto da diversi developer di Bitcoin Core, tra cui Eric Lombrozo che ha reso più comprensibile Segregated Witness, Luke Dashjr che mi ha aiutato nel capitolo circa le transazioni, Johnson

Lau che ha revisionato Segregated Witness ed altri capitoli, e tanti altri. Un grazie particolare a Joseph Poon, Tadge Dryja e Olaoluwa Osuntokun che hanno spiegato Lightning Network, revisionato I miei scritti, e risposto a diverse domande nei momenti in cui mi bloccavo per mancanza di conoscenza.

Devo tutto l'amore che ho per le parole e per i libri a mia madre, Theresa, che mi ha cresciuto in una casa con libri che potevi trovare a ogni parete. Mia madre mi ha inoltre comprato il mio primo computer nel 1982, anche se si considera una che ha un po paura della tecnologia. Mio padre, Meneleaos, un ingegnere civile che ha appena

pubblicato il suo libro a 80 anni, era colui che mi ha insegnato il pensiero logico e analitico e mi ha trasmesso un'amore per la scienza e l'ingegneria.

Grazie a tutti per avermi aiutato durante questo viaggio.

Bozza della Early Release (Contributi da GitHub)

Molti contributori hanno aiutato con commenti, correzioni e aggiunte alla bozza di rilascio anticipato su GitHub.

Grazie a tutti per i vostri contributi a questo libro. Di seguito è riportato un elenco di contributori GitHub notevoli, incluso il loro ID GitHub tra parentesi:

- Akira Chiku (achiku)
- Alex Waters (alexwaters)
- Andrew Donald Kennedy (grkvlt)
- bitcoinctf
- Bryan Gmyrek (physicsdude)
- Casey Flynn (cflynn07)
- cclauss
- Chapman Shoop (belovachap)
- Christie D'Anna (avocadobreath)
- Cody Scott (Siecje)
- coinradar
- Cragin Godley (cgodley)
- Craig Dodd (cdodd)
- dallyshalla
- Darius Kramer (dkrmr)
- David Huie (DavidHuie)

- Diego Viola (diegoviola)
- Dirk Jäckel (biafra23)
- Dimitris Tsapakidis (dimitris-t)
- Dmitry Marakasov (AMDmi3)
- drstrangeM
- Ed Eykholt (edeykholt)
- Ed Leafe (EdLeafe)
- Edward Posnak (edposnak)
- Elias Rodrigues (elias19r)
- Eric Voskuil (evoskuil)
- Eric Winchell (winchell)
- Erik Wahlström (erikwam)
- effectsToCause (vericoin)
- Esteban Ordano (eordano)
- ethers
- fabienhinault

- Frank Höger (francyi)
- Gaurav Rana (bitcoinsSG)
- genjix
- halseth
- Holger Schinzel (schinzelh)
- Ioannis Cherouvim (cherouvim)
- Ish Ot Jr. (ishotjr)
- ivangreene
- James Addison (jayaddison)
- Jameson Lopp (jlopp)
- Jason Bisterfeldt (jbisterfeldt)
- Javier Rojas (fjrojasgarcia)
- Jeremy Bokobza (bokobza)
- JerJohn15
- Joe Bauers (joebauers)
- joflynn

- Johnson Lau (jl2012)
- Jonathan Cross (jonathancross)
- Jorgeminator
- Kai Bakker (kaibakker)
- Lucas Betschart (lclc)
- Magomed Aliev (30mb1)
- Mai-Hsuan Chia (mhchia)
- Marco Falke (MarcoFalke)
- Marzig (marzig76)
- Matt McGivney (mattmcgiv)
- Maximilian Reichel (phramz)
- Michalis Kargakis (kargakis)
- Michael C. Ippolito (michaelcippolito)
- Mihail Russu (MihailRussu)
- Minh T. Nguyen (enderminh)

- Nagaraj Hubli (nagarajhubli)
- Nekomata (nekomata-3)
- Philipp Gille (philippgille)
- Robert Furse (Rfurse)
- Richard Kiss (richardkiss)
- Ruben Alexander (hizzvizz)
- Sam Ritchie (sritchie)
- Sebastian Falbesoner (theStack)
- Sergej Kotliar (ziggamon)
- Seiichi Uchida (topecongiro)
- Simon de la Rouviere (simondlr)
- Stephan Oeste (Emzy)
- takaya-imai
- Thiago Arrais (thiagoarrais)
- Thomas Kerin (afk11)
- venzen

- Will Binns (wbnns)
- wintercooled
- wjx
- Wojciech Langiewicz (wlk)
- Yancy Ribbens (yancyribbens)
- yurigeorgiev4

Glossario

Questo glossario comprende molti dei termini usati con riferimento al bitcoin. Si tratta di parole usate di frequente nel testo quindi si consiglia di memorizzarle per una rapida consultazione.

indirizzo

Un indirizzo bitcoin si presenta come segue:
1DSrfJdB2AnWaFNgSbv3MZC2m749
Consiste in una stringa di lettere e numeri ed è essenzialmente una versione criptata base58check dell'hash a 160 bit di una chiave pubblica. Proprio come chiedi ad

altri di inviare una e-mail al tuo indirizzo e-mail, chiedi ad altri di inviarti bitcoin a uno dei tuoi indirizzi bitcoin.

bip

Proposte di miglioramento per il Bitcoin. Un'insieme di proposte che i membri della comunità bitcoin hanno presentato per migliorare i bitcoin. Ad esempio, BIP-21 è una proposta per migliorare lo schema dell'uniform resource identifier (URI).

bitcoin

Il nome dell'unità di valuta (la moneta), il network e il software.

blocco

Un gruppo di transazioni, marcate da un timestamp e dall'impronta univoca del blocco precedente. Per poter fornire una proof of work, è necessario calcolare l'header del blocco, e successivamente validare le transazioni. I blocchi validi sono inseriti all'interno della blockchain principale mediante il consenso della rete.

blockchain

Una lista di blocchi validati, ognuno che collega il precedente fino al blocco di origine (genesis block).

Problema dei Generali

Bizantini

Un sistema informatico affidabile deve essere in grado di far fronte al fallimento di uno o più dei suoi componenti. Un componente guasto può presentare un tipo di comportamento che viene spesso trascurato, ovvero inviare informazioni in conflitto a parti diverse del sistema. Il problema di affrontare questo tipo di fallimento è espresso in modo astratto come il Problema dei Generali Bizantini.

coinbase

Un campo speciale utilizzato come unico input per le transazioni coinbase. Il coinbase consente di

richiedere il premio del blocco e fornisce fino a 100 byte per dati arbitrari. Da non confondere con la transazione Coinbase.

coinbase transaction

La prima transazione di un blocco. Viene sempre generata da un miner, include un singolo coinbase. Da non confondere con Coinbase.

cold storage

Si riferisce a mantenere una riserva di bitcoin offline. Il cold storage si ottiene quando le chiavi private Bitcoin vengono create ed archiviate in un ambiente offline sicuro. Il cold storage è importante per chiunque abbia aziende che operano in bitcoin.

I computer online sono vulnerabili ad attacchi hacker e non dovrebbero essere utilizzati per memorizzare una quantità significativa di bitcoin.

colored coins

Un protocollo Bitcoin 2.0 open source che consente agli sviluppatori di creare risorse digitali sulla blockchain di bitcoin utilizzando le sue funzionalità oltre alla valuta.

conferme

Quando una transazione viene inclusa in un blocco, ha una conferma. Non appena un altro blocco verrà validato nella stessa blockchain, la transazione avrà due conferme, e così via. Sei o più conferme sono

considerate una prova sufficiente che la transazione non possa essere annullata.

consensus

Quando più nodi, di solito la maggior parte dei nodi sulla rete, hanno tutti gli stessi blocchi nella loro migliore blockchain validata localmente.
Da non confondere con le regole del consensus.

Regole del consensus

Le regole di convalida dei blocchi che seguono i full node per restare in consenso con gli altri nodi. Da non confondere con il consenso.

difficoltà

Un'impostazione valida per tutta la rete, che regolamenta quanta potenza computazionale è necessaria per produrre una soluzione proof of work.

retargeting del livello di difficoltà

Un ricalcolo della difficoltà per tutta la rete, che si verifica ogni 2016 blocchi, e tiene in considerazione la potenza di calcolo dei 2016 blocchi precedenti.

livello di difficoltà

La difficoltà a cui tutta la potenza computazionale della rete troverà nuovi blocchi approssimativamente ogni 10 minuti.

double-spending

Il double spending è il risultato di spendere dei bitcoin con successo più di una volta. Bitcoin protegge dalla doppia spesa verificando ogni transazione aggiunta alla catena di blocchi per garantire che gli input per la transazione non siano stati già spesi in precedenza.

ECDSA

Elliptic Curve Digital Signature Algorithm o ECDSA è un algoritmo crittografico utilizzato da Bitcoin per garantire che i fondi possano essere spesi solo dai legittimi proprietari.

extra nonce

Mano a mano che aumentava la difficoltà, i minatori spesso passavano attraverso tutti i 4 miliardi di valori del nonce senza trovare un blocco. Poiché lo script coinbase può archiviare tra 2 e 100 byte di dati, i minatori hanno iniziato a utilizzare quello spazio come spazio extra nonce, consentendo loro di esplorare una gamma molto più ampia di valori di intestazione di blocco per trovare blocchi validi.

commissioni

Il mittente di una transazione, spesso include il pagamento di una tariffa per la rete, che processerà la transazione richiesta. La maggior parte delle transazioni richiede il

pagamento di una tariffa minima di 0.5 mBTC.

fork

La fork, nota anche come accidental fork, si verifica quando due o più blocchi condividono la stessa altezza del blocco, dividendo la blockchain in due parti. Tipicamente si verifica quando due o più minatori trovano blocchi quasi nello stesso istante. Può anche essere considerata come parte di un attacco.

genesis block

Il primo blocco della blockchain, utilizzato per inizializzare la criptovaluta.

hard fork

L'hard fork, noto anche come Hard-Forking Change, è una divergenza permanente nella blockchain che si verifica quando i nodi non aggiornati non possono convalidare i blocchi creati dai nodi aggiornati che seguono le nuove regole di consenso. Da non confondere con la fork, la soft fork, la software fork o la fork Git.

hardware wallet

Un portafoglio hardware è una tipologia speciale di portafoglio bitcoin che memorizza le chiavi private dell'utente in un dispositivo hardware sicuro.

hash

L'impronta digitale di un certo input binario.

hashlocks

Un hashlock limita la spesa di un output finché non viene rivelato pubblicamente un certo dato. Gli hashlock hanno la proprietà utile che, una volta che un hashlock viene aperto pubblicamente, è possibile aprire qualsiasi altro hashlock protetto usando la stessa chiave. Ciò rende possibile creare più output che sono tutti legati dallo stesso hashlock e che diventano tutti spendibili allo stesso tempo.

HD protocol

La creazione di chiavi tramite

Hierarchical Deterministic (HD) e il trasferimento (BIP32), che consente di creare chiavi ‘figlie’ da chiavi principali in una gerarchia ordinata.

HD wallet

Wallet che utilizzano il protocollo di trasferimento (BIP32) e creazione di chiavi Hierarchical Deterministic (protocollo HD).

HD wallet seed

Il seed di un wallet o il root seed è un valore corto (solitamente una serie di parole) utilizzato come seme per generare la chiave privata principale e il master chain code di un wallet HD.

HTLC

Un Hashed TimeLock Contract o HTLC è una classe di pagamenti che utilizza hashlock e timelock per richiedere che il destinatario di un pagamento riconosca di aver ricevuto il pagamento prima di una scadenza generando una prova di pagamento crittografica o revocando la possibilità di richiedere il pagamento, restituendolo al mittente.

KYC

Know Your Customer (KYC) è il processo di un'azienda che identifica e verifica l'identità dei propri clienti. Il termine è anche usato per riferirsi al regolamento bancario che regola

queste attività.

LevelDB

LevelDB è un archivio di valori-chiave su disco open source. LevelDB è una libreria leggera, per uso singolo e per la persistenza, con collegamenti a molte piattaforme.

Lightning Network

Lightning Network è un'implementazione degli Hashed Timelock Contracts (HTLCs) con canali di pagamento bidirezionali che consente di instradare in modo sicuro i pagamenti su più canali di pagamento peer-to-peer. Ciò consente la formazione di una rete in cui qualsiasi peer sulla rete stessa

può pagare qualsiasi altro peer anche se non hanno un canale aperto tra loro.

Locktime

Locktime, o più tecnicamente nLockTime, è la parte di una transazione che indica il primo momento o il primo blocco in cui tale transazione può essere aggiunta alla blockchain.

mempool

Il bitcoin Mempool (pool di memoria) è una raccolta di tutti i dati delle transazioni in un blocco che sono stati verificati dai nodi bitcoin, ma che non sono ancora stati confermati.

merkle root

Il nodo radice di un merkle tree, un discendente di tutte le coppie hash nel tree. Le intestazioni dei blocchi devono includere un merkle root valido derivante da tutte le transazioni in quel blocco.

merkle tree

Un tree costruito da hashing di dati accoppiati (le 'leaves', foglie), dove vengono associati ed viene effettuato l'hashing dei risultati fino a quando rimane un singolo hash, ovvero il merkle root. In Bitcoin, le foglie (leaves) sono quasi sempre transazioni da un singolo blocco.

miner

Un nodo della rete che trova soluzioni valide alla proof of work per i nuovi blocchi mediante numerosi calcoli.

multisignature

Multisignature (multisig) si riferisce a richiedere più di una chiave per autorizzare una transazione bitcoin.

rete

Una rete peer-to-peer che propaga le transazioni e i blocchi a tutti i nodi bitcoin della stessa.

nonce

Il "nonce" in un blocco bitcoin è un campo a 32 bit (4 byte) il cui valore è impostato in modo che l'hash del

blocco contenga una serie di zeri iniziali. Il resto dei campi non può essere modificato, in quanto hanno un significato definito.

transazioni off-chain

Una transazione off-chain è il movimento di valore al di fuori della blockchain. Mentre una transazione sulla blockchain, solitamente indicata semplicemente come una transazione, modifica la blockchain e dipende dalla blockchain per determinarne la validità, una transazione off-chain si basa su altri metodi per registrare e convalidare la transazione stessa.

opcode

Codici operativi dal linguaggio

Bitcoin Script che trasmettono dati o eseguono funzioni all'interno di un pubkey script o signature script.

Open Assets protocol

L'Open Assets Protocol è un protocollo semplice e potente basato sulla blockchain di Bitcoin. Permette l'emissione ed il trasferimento di risorse create dall'utente. Il protocollo Open Assets è un'evoluzione del concetto dei colored coin.

OP_RETURN

Un codice operativo utilizzato in uno degli output in una transazione OP_RETURN. Da non confondere con la transazione OP_RETURN.

transazione OP_RETURN

Un tipo di transazione che aggiunge dati arbitrari ad uno script pubkey che i nodi completi non devono memorizzare nel loro database UTXO. Da non confondere con l'opcode OP_RETURN.

blocco orfano (orphan block)

Blocchi il cui il blocco padre (parent block) non è stato elaborato dal nodo locale, quindi non possono ancora essere completamente convalidati. Da non confondere con il stale block.

transazioni orfane (orphan transactions)

Transazioni che non possono entrare nella pool a causa di una o più transazioni di input mancanti.

output

Output, output di transazione (transaction output) o TxOut è un output in una transazione che contiene due campi: un campo valore per il trasferimento di zero o più satoshi e uno script pubkey per indicare quali condizioni devono essere soddisfatte per quei spendere nuovamente quei satoshi.

P2PKH

Transazioni che pagano un indirizzo bitcoin P2PKH o script Pay To Pubkey Hash. Un output bloccato da

uno script P2PKH può essere sbloccato (speso) presentando una chiave pubblica ed una firma digitale creata dalla chiave privata corrispondente.

P2SH

P2SH o Pay-to-Script-Hash è una nuova e potente tipologia di transazione che semplifica in maniera massiccia l'uso di transaction script complessi. Con P2SH lo script complesso che indica la condizione per spendere l'output (redeem script) non è presente nel locking script. Invece, solo un hash di quest'ultimo è presente nel locking script.

indirizzo P2SH

Gli indirizzi P2SH sono codifiche Base58Check dell'hash da 20 byte di uno script, gli indirizzi P2SH utilizzano il prefisso di versione "5", che risulta in indirizzi codificati Base58Check che iniziano con un "3". Gli indirizzi P2SH nascondono tutta la complessità, in modo che la persona che effettua un pagamento non veda lo script.

P2WPKH

La firma di un P2WPKH (Pay-to-Witness-Public-Key-Hash) contiene le stesse informazioni di una spesa P2PKH, ma si trova nel campo testimone (witness) invece del campo scriptSig. Anche lo scriptPubKey viene modificato.

P2WSH

La differenza tra P2SH e P2WSH (Pay-to-Witness-Script-Hash) riguarda la modifica della posizione di prova crittografica dal campo scriptSig al campo witness e lo scriptPubKey che viene modificato.

paper wallet

In un senso più specifico, un paper wallet (portafoglio di carta) è un documento contenente tutti I dati necessari per generare un qualsiasi numero di chiavi private di Bitcoin, formando un wallet di chiavi multiple. Tuttavia, molto spesso le persone utilizzano questo termine per indicare qualsiasi modalità di

gestione di bitcoin offline sotto un documento fisico. Questa seconda definizione include inoltre I paper key e I codici sotto forma di codici riscattabili.

payment channels

Un micropayment channel (canale di micropagamento) o payment channel (canale di pagamento) è una classe di tecniche pensate per dare la possibilità agli utenti di effettuare transazioni multiple senza doverle registrare tutte nella blockchain di Bitcoin. In un payment channel tipico, solamente due transazioni vengono aggiunte nella blockchain ma un numero potenzialmente infinito può essere effettuato dai partecipanti.

pooled mining

Il pooled mining è un tipologia di mining dove più client contribuiscono alla generazione di un blocco, e successivamente si dividono il block reward in base alla potenza computazionale data in contributo al gruppo.

Proof-of-Stake

Proof-of-Stake (PoS) è un metodo grazie a cui un network alla base di una criptovaluta punta a raggiungere un consenso distribuito. Proof-of-Stake chiede agli utenti di provare il possesso di un certo ammontare di valuta (il loro ‘stake’ della valuta stessa).

Proof-of-Work

Una porzione di dati che richiede un significativo sforzo computazionale, per essere trovata. In bitcoin, i minatori devono trovare una soluzione numerica all'algoritmo SHA256 che soddisfa un obiettivo stabilito dall'intera rete, la difficoltà calcolata.

ricompensa

Un ammontare, incluso in ogni nuovo blocco, quale ricompensa dalla rete per il minatore che troverà la soluzione Proof-Of-Work. Al momento è pari a 25BTC per blocco.

RIPEMD-160

RIPEMD-160 è una funzione di hash crittografica a 160 bit. RIPEMD-160 è una versione rafforzata di RIPEMD con un risultato hash di 160 bit e dovrebbe essere sicuro per i prossimi dieci anni o più.

satoshi

Un satoshi è la più piccola denominazione di un bitcoin che può essere registrata sulla blockchain. È l'equivalente di 0.00000001 bitcoin e prende il nome del creatore di Bitcoin, Satoshi Nakamoto.

Satoshi Nakamoto

Satoshi Nakamoto è il nome utilizzato da una persona o gruppo di persone che ha pensato e creato il design di

Bitcoin e la propria originale implementazione di referenza, Bitcoin Core. Come parte dell'implementazione, lui,lei o loro (Satoshi Nakamoto) hanno creato la prima definizione di database blockchain. Nel processo sono stati I primi a risolvere il problema del double-spending nelle valute digitali. La loro identità rimane ad oggi misteriosa.

Script

Bitcoin utilizza un sistema di scripting per le transazioni. Script è semplice, stack-based e processato da sinistra e destra. È per scelta non-Turing complete, con nessun ciclo.

ScriptPubKey (aka pubkey script)

ScriptPubKey o pubkey script, è uno script incluso negli output che detta le condizioni che devono essere seguite da un certo numero di satoshi per essere spesi. Il dato per seguire tali condizioni può essere inserito in un signature script.

ScriptSig (aka signature script)

ScriptSig o signature script, è il dato generato dal mittente, ed è quasi sempre utilizzato come una variabile per soddisfare un pubkey script.

chiave segreta (o chiave

privata)

Un numero segreto che sblocca i bitcoin inviati all'indirizzo corrispondente. Una chiave segreta è rappresentata da una stringa alfanumerica simile a:

```
5J76sF8L5jTtzE96r66Sf8cka9y44wdpJjMv
```

Segregated Witness

Segregated Witness è una proposta di aggiornamento del protocollo Bitcoin la cui innovazione tecnologica separa la signature data dalle transazioni bitcoin. Segregated Witness è una proposta di soft fork; un cambio che, tecnicamente, rende le regole del protocollo Bitcoin più restrittive.

SHA

Il Secure Hash Algorithm o SHA è una famiglia di funzioni hash crittografiche pubblicate dal National Institute of Standards and Technology (NIST).

Simplified Payment Verification (SPV)

SPV o Simplified Payment Verification è un metodo per verificare che delle particolari transazioni siano state inserite in un blocco senza la necessità di scaricare l'intero blocco. Questo metodo è utilizzato da una serie di client Bitcoin 'lightweight' (leggeri).

soft fork

soft fork o Soft-Forking Change è una fork temporanea nella blockchain che si verifica comunemente quando i minatori che utilizzano nodi non aggiornati non seguono una nuova regola di consenso di cui i nodi non sono a conoscenza. Da non confondere con fork, hard fork, software fork o Git fork.

stale block

Un blocco che è stato minato con successo ma che non è ancora stato incluso nella migliore e più attuale blockchain, probabilmente perché un qualche altro blocco alla stessa altezza ha avuto la propria catena

estesa per prima. Da non confondere con orphan block.

timelocks

Un timelock limita la spesa di alcuni bitcoin fino a una specifica ora futura o all'altezza del blocco. I timelocks sono di rilievo in molti contratti di Bitcoin, compresi i canali di pagamento e i gli hashed timelock contracts.

transazione

In parole povere, un trasferimento di bitcoin da un indirizzo ad un altro. Più precisamente, una transazione è una struttura di dati che esprime un trasferimento di valore. Le transazioni sono trasmesse attraverso

la rete bitcoin, raggruppate dai minatori ed incluse all'interno di blocchi, registrati permanentemente nella blockchain.

transaction pool

Una serie disordinata di transazioni che non sono ancora incluse in un blocco nella catena principale, ma per cui abbiamo gli input di transazione.

Turing completeness

Un linguaggio di programmazione è chiamato "Turing complete" quando può avviare un qualsiasi programma che una macchina di Turing può avviare, garantendone abbastanza tempo e memoria.

unspent transaction output (UTXO)

UTXO è un output di transazione non speso che può essere speso come un input in una nuova transazione.

wallet

Software che detiene tutti I tuoi indirizzi bitcoin e le relative chiavi private. Viene utilizzato per inviare, ricevere e gestire I tuoi bitcoin.

Wallet Import Format (WIF)

WIF o Wallet Import Format è un formato di interscambio di dati pensato per permettere l'esportazione e l'importazione di una private key

con un valore indicante se utilizza o meno una public key compressa.

Alcune definizioni fornite sono state ottenute con una licenza CC-BY dal Wiki bitcoin o da altre fonti di documentazione open source.

Introduzione

Cos'è Bitcoin?

Bitcoin è una raccolta di concetti e tecnologie che formano le basi per un ecosistema di denaro digitale. Le unità di valuta chiamate bitcoin vengono utilizzate per immagazzinare e trasferire valore tra i partecipanti del network Bitcoin. Gli utenti Bitcoin comunicano tra loro utilizzando il protocollo Bitcoin principalmente attraverso internet, sebbene possano esser utilizzate altre reti di trasporto. L'intero protocollo Bitcoin, disponibile come codice sorgente aperto (open source software), può

essere utilizzato su un'ampia gamma di dispositivi digitali, inclusi portatili e smartphone, rendendo questa tecnologia facilmente accessibile.

Gli utenti possono trasferire bitcoin sulla rete per fare praticamente tutto ciò che può essere fatto con le valute tradizionali, tra cui comprare e vendere beni, inviare denaro a persone o organizzazioni o estendere il credito. I bitcoin possono essere acquistati, venduti e scambiati con altre valute presso cambiavalute specializzati. Bitcoin in un certo senso è la forma perfetta di denaro per internet perché è veloce, sicuro e senza confini.

A differenza delle valute tradizionali, i bitcoin sono interamente virtuali. Non

ci sono monete fisiche o persino monete digitali di per sé. Le monete sono implicite nelle transazioni che trasferiscono il valore dal mittente al destinatario. Gli utenti di bitcoin possiedono chiavi che consentono loro di dimostrare la proprietà di bitcoin nella rete bitcoin. Con queste chiavi possono firmare le transazioni per sbloccare il valore e spenderlo trasferendolo a un nuovo proprietario. Le chiavi vengono spesso memorizzate in un portafoglio digitale sul computer o sullo smartphone di ciascun utente. Il possesso della chiave che può firmare una transazione è l'unico prerequisito per spendere bitcoin, mettendo il controllo interamente nelle mani di

ciascun utente.

Bitcoin è un sistema distribuito, peer-to-peer. Come tale non esiste un server "centrale" o un punto di controllo. I Bitcoin vengono creati attraverso un processo chiamato "mining", che comporta la competizione per trovare soluzioni a un problema matematico durante l'elaborazione delle transazioni bitcoin. Qualsiasi partecipante alla rete bitcoin (vale a dire, chiunque utilizzi un dispositivo che esegue un bitcoin "full node") può operare come minatore, utilizzando la potenza di elaborazione del proprio computer per verificare e registrare le transazioni. Ogni 10 minuti, in media, un bitcoin miner è in grado di

convalidare le transazioni degli ultimi 10 minuti e viene ricompensato con dei nuovi bitcoin. Essenzialmente, il bitcoin mining decentralizza le funzioni di emissione di valuta e di compensazione di una banca centrale e sostituisce la necessità di qualsiasi banca centrale.

Il protocollo bitcoin include algoritmi incorporati che regolano la funzione di mining attraverso la rete. La difficoltà del processo che i minatori devono eseguire è regolata dinamicamente in modo tale che, in media, qualcuno ci riesca ogni 10 minuti, indipendentemente da quanti minatori (e quanta elaborazione) siano in competizione in qualsiasi momento. Il

protocollo dimezza inoltre la velocità con cui vengono creati nuovi bitcoin ogni 4 anni e limita il numero totale di bitcoin che verranno creati per un totale fissato a 21 milioni di monete. Il risultato è che il numero di bitcoin in circolazione segue da vicino una curva facilmente prevedibile che si avvicina a 21 milioni entro il 2140. A causa della diminuzione del tasso di emissione del bitcoin, nel lungo periodo, la valuta bitcoin è deflazionistica. Inoltre, bitcoin non può essere inflazionato "stampando" denaro nuovo oltre al tasso di emissione previsto.

Dietro le quinte, il bitcoin è anche il nome del protocollo, una rete peer-to-

peer e un'innovazione delle reti di calcolo distribuito. La valuta bitcoin è in realtà solo la prima applicazione di questa invenzione. Bitcoin rappresenta il culmine di decenni di ricerca in crittografia e sistemi distribuiti e include quattro innovazioni chiave riunite in una combinazione unica e potente. Bitcoin consiste di:

- Un network peer-to-peer decentralizzato (il protocollo bitcoin)
- Un libro-mastro di transazioni pubblico (la blockchain)
- Un insieme di regole per la convalida indipendente delle transazioni e l'emissione di valuta (regole di consenso)

- Un meccanismo per raggiungere un consenso globale decentrato sulla blockchain valida (Algoritmo Proof-of-Work)

Come sviluppatore, vedo bitcoin come una connessione a Internet, una rete per la propagazione del valore e la garanzia della proprietà delle risorse digitali tramite il calcolo distribuito. C'è molto di più nel bitcoin di quanto non sembri.

In questo capitolo inizieremo spiegando alcuni concetti e termini principali, ottenendo il software necessario e utilizzando bitcoin per transazioni semplici. Nei capitoli seguenti inizieremo a scartare gli strati di tecnologia che rendono possibile il

bitcoin ed esamineremo i meccanismi interni della rete bitcoin e del protocollo.

Valute Digitali Pre-Bitcoin

La fattibilità del denaro digitale è strettamente legato agli sviluppi della crittografia. Ciò non sorprende se si considerano le sfide fondamentali legate all'uso dei bit per rappresentare il valore che può essere scambiato per beni e servizi. Tre domande sono fondamentali per chiunque accetti il denaro digitale:

1. Posso fidarmi che il denaro sia autentico e non contraffatto?
2. Posso fidarmi che il denaro digitale possa essere speso una sola volta (noto come il problema della "doppia spesa")?
3. Posso essere sicuro che nessun altro possa rivendicare che questo denaro appartiene a lui e non a me?

Gli emittenti di cartamoneta combattono costantemente il problema della contraffazione usando carte sempre più sofisticate e tecnologie di stampa. I soldi fisici risolvono facilmente il problema della doppia spesa perché la stessa banconota non può essere in due

posti contemporaneamente. Ovviamente, i soldi convenzionali vengono spesso archiviati e trasmessi digitalmente. In questi casi, le questioni di contraffazione e di doppia spesa vengono gestite eliminando tutte le transazioni elettroniche attraverso le autorità centrali che hanno una visione globale della valuta in circolazione. Per il denaro digitale, che non può sfruttare gli inchiostri esoterici o le strisce olografiche, la crittografia fornisce la base per fidarsi della legittimità del valore rivendicato da un utente. Nello specifico, le firme digitali crittografiche consentono a un utente di firmare una risorsa o

transazione digitale comprovante la proprietà di tale risorsa. Con l'architettura appropriata, anche le firme digitali possono essere utilizzate per affrontare il problema della doppia spesa.

Quando la crittografia è iniziata a diventare sempre più disponibile e compresa alla fine degli anni '80, molti ricercatori hanno iniziato a provare a usare la crittografia per realizzare monete digitali. Questi primi progetti di monete digitali emettevano valuta digitale, generalmente sostenuta da una moneta nazionale o da metalli preziosi come l'oro.

Sebbene queste prime valute

digitali funzionassero, erano centralizzate e, di conseguenza, erano facili da attaccare da parte di governi e hacker. Le prime valute digitali utilizzavano una camera di compensazione centrale per regolare tutte le transazioni a intervalli regolari, proprio come un sistema bancario tradizionale. Sfortunatamente, nella maggior parte dei casi queste nascenti valute digitali sono state prese di mira da governi preoccupati e alla fine contestate per essere inesistenti. Alcune sono fallite in crash spettacolari quando la casa madre improvvisamente è stata messa in liquidazione. Per essere robusti

contro l'intervento degli antagonisti, siano essi governi legittimi o elementi criminali, era necessaria una valuta digitale decentralizzata per evitare un singolo punto di attacco. Bitcoin è un sistema di questo tipo, decentralizzato a partire dalla progettazione e privo di qualsiasi autorità centrale o punto di controllo che possa essere attaccato o corrotto.

Storia di Bitcoin

Bitcoin fu inventato nel 2008 con la pubblicazione di un documento intitolato "Bitcoin: Un sistema di contanti elettronici Peer-to-Peer," (https://bitcoin.org/bitcoin.pdf) scritto

sotto l'alias di Satoshi Nakamoto. Nakamoto ha combinato diverse invenzioni precedenti come b-money e HashCash per creare un sistema di contanti elettronici completamente decentralizzato che non si basi su un'autorità centrale per l'emissione o la regolamentazione di una valuta e la convalida delle transazioni. L'innovazione chiave consisteva nell'utilizzare un sistema di calcolo distribuito (chiamato algoritmo "Proof-of-Work") per condurre una "elezione" globale ogni 10 minuti, consentendo alla rete decentralizzata di arrivare ad un "consenso" sullo stato delle transazioni. Questo risolve elegantemente il problema della

doppia spesa in cui è possibile spendere una singola unità di valuta due volte. In precedenza, il problema della doppia spesa era una debolezza della valuta digitale e veniva risolto eliminando tutte le transazioni attraverso una stanza di compensazione centrale.

La rete bitcoin è stata avviata nel 2009, sulla base di un documento pubblicato da Nakamoto e riveduto da molti altri programmatori. L'implementazione dell'algoritmo Proof of Work (mining) che fornisce sicurezza e resilienza per bitcoin è aumentata in modo esponenziale in termini di potenza e ora supera la potenza di elaborazione combinata dei

migliori supercomputer del mondo. Il valore di mercato totale di Bitcoin ha a volte superato i $ 135 miliardi di dollari USA, a seconda del tasso di cambio bitcoin-dollaro. La più grande transazione elaborata finora dalla rete era di $ 400 milioni di dollari USA, trasmessa istantaneamente ed elaborata per una commissione di $ 1.

Satoshi Nakamoto si è ritirato dal pubblico nell'aprile 2011, lasciando la responsabilità di sviluppare il codice e la rete a un fiorente gruppo di volontari. L'identità della persona o delle persone dietro bitcoin è ancora sconosciuta. Tuttavia, né Satoshi Nakamoto né nessun altro esercita il controllo individuale sul sistema

bitcoin, che opera sulla base di principi matematici completamente trasparenti, codice open source e consenso tra i partecipanti. L'invenzione stessa è rivoluzionaria e ha già generato nuove conoscenze nei campi dell'informatica distribuita, dell'economia e dell'econometria.

Una Soluzione a un Problema Computazionale Distribuito

L'invenzione di Satoshi Nakamoto è

anche una soluzione pratica e originale per un problema nel calcolo distribuito, noto come il "Problema dei Generali Bizantini". In breve, il problema consiste nel cercare di concordare una linea di condotta o lo stato di un sistema scambiando informazioni su una rete inaffidabile e potenzialmente compromessa. La soluzione di Satoshi Nakamoto, che utilizza il concetto di Proof of Work per raggiungere il consenso senza un'autorità centrale fidata, rappresenta una svolta nell'informatica distribuita ed ha un'ampia applicabilità oltre alla valuta. Può essere utilizzata per

ottenere un consenso sulle reti decentralizzate per dimostrare l'equità delle elezioni, delle lotterie, dei registri patrimoniali, della notarizzazione digitale e altro.

I vari Utilizzi di Bitcoin, gli Utenti e le loro Storie

Bitcoin è un'innovazione nell'antica tecnologia del denaro. Al suo interno, il denaro facilita semplicemente lo scambio di valore tra le persone. Pertanto, per comprendere appieno il bitcoin ed i suoi usi, lo esamineremo dal punto di vista delle persone che lo utilizzano. Ognuna delle persone e le

loro storie, come elencato qui, illustra uno o più casi d'uso specifici. Li vedremo in tutto il libro:

Negozio Nord Americano di prodotti comuni

Alice vive nella Bay Area della California settentrionale. Ha sentito parlare di bitcoin dai suoi amici tecnici e vuole iniziare a usarlo. Seguiremo la sua storia mentre impara su bitcoin, ne acquisisce alcuni, e poi spende alcuni dei suoi bitcoin per comprare una tazza di caffè al Bob's Cafe di Palo Alto. Questa storia ci introdurrà al software, agli scambi ed alle transazioni di base dal punto di vista di un consumatore.

Negozio di cose di valore Nord Americano

Carol è proprietaria di una galleria d'arte in San Francisco. Vende quadri costosi in cambio di bitcoin. Questa storia introdurrà il rischio di un "attacco 51%" per venditori di oggetti di valore.

Servizi di liberi professionisti esteri

Bob, il proprietario del bar a Palo Alto, sta costruendo un nuovo sito web. Ha stipulato un contratto con uno sviluppatore web indiano, Gopesh, che vive a Bangalore, in India. Gopesh ha accettato di essere pagato in bitcoin. Questa storia esaminerà l'uso di bitcoin per l'esternalizzazione, i lavori a

contratto ed i bonifici internazionali.

Web store

Gabriel è un giovane adolescente intraprendente a Rio de Janeiro, che gestisce un piccolo negozio online che vende t-shirt, tazze da caffè e adesivi con il marchio bitcoin. Gabriel è troppo giovane per avere un conto in banca, ma i suoi genitori stanno incoraggiando il suo spirito imprenditoriale.

Donazioni

Eugenia è la direttrice di un'associazione benefica per bambini nelle Filippine. Recentemente ha scoperto bitcoin e vuole usarlo per raggiungere un nuovo gruppo di

donatori stranieri e nazionali per raccogliere fondi per la sua beneficenza. Sta anche indagando sui modi per usare bitcoin per distribuire rapidamente i fondi verso le aree di bisogno. Questa storia mostrerà l'uso di bitcoin per la raccolta fondi globale e l'uso di un libro mastro aperto per la trasparenza nelle organizzazioni di beneficenza.

Import/export

Mohammed è un importatore di elettronica a Dubai. Sta cercando di usare bitcoin per comprare componenti elettronici dagli Stati Uniti e dalla Cina ed importarli negli Emirati Arabi Uniti per accelerare il processo di pagamento delle

importazioni. Questa storia mostrerà come il bitcoin può essere utilizzato per i grandi pagamenti internazionali business-to-business legati ai beni reali.

Bitcoin Mining

Jing è uno studente di ingegneria informatica a Shanghai. Ha costruito un impianto di "mining" per minare bitcoin usando le sue capacità ingegneristiche per integrare i suoi redditi. Questa storia esaminerà la base "industriale" del bitcoin: l'attrezzatura specializzata utilizzata per proteggere la rete bitcoin ed emettere nuova valuta.

Ognuna di queste storie si basa sulle persone reali e le industrie reali che

attualmente utilizzano bitcoin per creare nuovi mercati, nuove industrie e soluzioni innovative per le problematiche economiche globali.

Per Iniziare

Bitcoin è un protocollo a cui è possibile accedere utilizzando un'applicazione client che parla il suo protocollo. Un "wallet bitcoin" è l'interfaccia utente più comune per il sistema bitcoin, proprio come un browser Web è l'interfaccia utente più comune per il protocollo HTTP. Esistono molte implementazioni e marche di portafogli bitcoin, proprio come ci sono molti marchi di browser Web (ad es. Chrome, Safari, Firefox e

Internet Explorer). E proprio come tutti noi abbiamo i nostri browser preferiti (Mozilla Firefox, Yay!) E i nostri cattivi (Internet Explorer, Yuck!), i portafogli bitcoin variano in termini di qualità, prestazioni, sicurezza, privacy e affidabilità. Esiste anche un'implementazione di riferimento del protocollo bitcoin che include un portafoglio, noto come "Satoshi Client" o "Bitcoin Core", che deriva dall'implementazione originale scritta da Satoshi Nakamoto.

Scegliere un Bitcoin Wallet

I wallet Bitcoin sono una delle applicazioni più sviluppate nell'ecosistema bitcoin (wallet =

portafoglio). C'è una forte concorrenza, e mentre probabilmente in questo momento è in fase di sviluppo un nuovo wallet, diversi wallet sviluppati lo scorso anno non vengono più aggiornati. Molti wallet si concentrano su piattaforme specifiche o usi specifici e alcuni sono più adatti ai principianti mentre altri sono pieni di funzionalità per utenti esperti. La scelta di un wallet è altamente soggettiva e dipende dall'uso e dalla competenza dell'utente. È quindi impossibile raccomandare una marca o un wallet specifico. Tuttavia, possiamo classificare i wallet bitcoin in base alla loro piattaforma e funzione e fornire una certa chiarezza su tutti i

diversi tipi di wallet esistenti. Meglio ancora, spostare chiavi o semi tra i wallet bitcoin è relativamente facile, quindi vale la pena provare diversi wallet fino a trovare quello che si adatta alle tue esigenze.

I wallet bitcoin possono essere suddivisi in categorie come in seguito elencato, a seconda della piattaforma:

Portafogli desktop

Il wallet desktop è stato il primo tipo di portafoglio bitcoin creato come programma di riferimento e molti utenti utilizzano wallet desktop per le funzionalità, l'autonomia ed il controllo che offrono. L'esecuzione su sistemi operativi di uso generale come Windows e Mac OS presenta

tuttavia alcuni svantaggi nella sicurezza, in quanto queste piattaforme sono spesso poco sicure e configurate in modo inadeguato.

Mobile wallet

Un mobile wallet è il tipo più comune di portafoglio bitcoin. Funzionando su sistemi operativi smart-phone come Apple iOS e Android, questi portafogli sono spesso un'ottima scelta per i nuovi utenti. Molti sono progettati per massimizzare semplicità e facilità d'uso, ma ci sono anche portafogli mobili dotati di funzionalità complete per utenti esperti.

Web wallet

I web wallet sono accessibili tramite un browser Web e memorizzano il portafoglio dell'utente su un server di proprietà di terzi. Questo sistema è simile alla webmail in quanto si basa interamente su un server di terze parti. Alcuni di questi servizi operano utilizzando il codice lato client in esecuzione nel browser dell'utente, che mantiene il controllo delle chiavi bitcoin nelle mani dell'utente. La maggior parte, tuttavia, presenta un compromesso prendendo il controllo delle chiavi bitcoin dagli utenti in cambio della facilità d'uso. Non è consigliabile memorizzare grandi quantità di bitcoin su sistemi di terze parti.

Hardware wallet

I portafogli hardware sono dispositivi che gestiscono un portafoglio bitcoin autonomo e sicuro su hardware specifici. Funzionano tramite porta USB con un browser Web desktop o tramite NFC (Near Field Communication) su un dispositivo mobile. Gestendo tutte le operazioni relative ai bitcoin su hardware specifici, questi portafogli sono considerati molto sicuri e adatti per la memorizzazione di grandi quantità di bitcoin.

Paper wallet

Le chiavi per controllare i bitcoin possono anche essere stampati per la memorizzazione a lungo termine.

Questi sono noti come portafogli di carta (paper wallet), anche possono essere utilizzati altri materiali (legno, metallo, ecc.). I portafogli di carta offrono un modo low-tech ma altamente sicuro di conservare bitcoin a lungo termine. Lo storage offline è spesso chiamato *cold storage*.

Un altro modo per classificare i portafogli bitcoin è il loro grado di autonomia ed il modo in cui interagiscono con la rete bitcoin:

Client full-node

Un client completo, o "nodo completo", è un client che memorizza l'intera cronologia delle transazioni bitcoin (ogni transazione di ogni

utente, sempre), gestisce portafogli degli utenti e può avviare transazioni direttamente sulla rete bitcoin. Un nodo completo gestisce tutti gli aspetti del protocollo e può convalidare indipendentemente l'intera blockchain e qualsiasi transazione. Un client full-node consuma notevoli risorse del computer (ad es. più di 125 GB di disco, 2 GB di RAM) ma offre completa autonomia e verifica indipendente delle transazioni.

Client leggero

Un client leggero, noto anche come client simple-payment-verification (SPV), si connette ai full node di bitcoin (citati in precedenza) per

l'accesso alle informazioni sulle transazioni bitcoin, ma memorizza il portafoglio utente in locale e crea, convalida e trasmette in modo indipendente le transazioni. I clienti leggeri interagiscono direttamente con la rete bitcoin, senza un intermediario.

Client API di terze parti

Un client API di terze parti è un client che interagisce con bitcoin tramite un sistema di API di terze parti (API, Application Programming Interface) anziché collegarsi alla rete bitcoin direttamente. Il portafoglio può essere memorizzato dall'utente o da server di terze parti, ma tutte le transazioni passano attraverso una

terza parte.

Combinando queste categorizzazioni, molti portafogli bitcoin rientrano in pochi gruppi, i tre più comuni sono il desktop full client, il portafoglio mobile light e il wallet web di terze parti. Le linee tra le diverse categorie sono spesso sfocate, poiché molti portafogli vengono eseguiti su più piattaforme e possono interagire con la rete in modi diversi.

Per gli scopi di questo libro, dimostreremo l'uso di una varietà di client bitcoin scaricabili, dall'implementazione di riferimento (Bitcoin Core) ai portafogli mobili e web. Alcuni esempi richiedono l'uso di Bitcoin Core, che, oltre a essere un

client completo, espone anche le API al portafoglio, alla rete e ai servizi di transazione. Se stai pianificando di esplorare le interfacce programmatiche nel sistema bitcoin, dovrai eseguire Bitcoin Core o uno dei client alternativi.

Avvio Rapido

Alice, che abbiamo introdotto precedentemente, non è un utente tecnico e solo di recente ha sentito parlare di bitcoin dal suo amico Joe. Durante una festa, Joe è ancora una volta entusiasta di spiegare bitcoin a tutti intorno a lui e offre una dimostrazione. Incuriosita, Alice chiede come può iniziare con bitcoin.

Joe dice che un portafoglio mobile è la cosa migliore per i nuovi utenti e raccomanda alcuni dei suoi portafogli preferiti. Alice scarica "Mycelium" per Android e la installa sul suo telefono.

Quando Alice esegue Mycelium per la prima volta, come con molti portafogli bitcoin, l'applicazione crea automaticamente un nuovo portafoglio per lei. Alice vede il portafoglio sul suo schermo, come mostrato in Figura 1 (nota: non inviare bitcoin a questo indirizzo di esempio, altrimenti sarà perso per sempre).

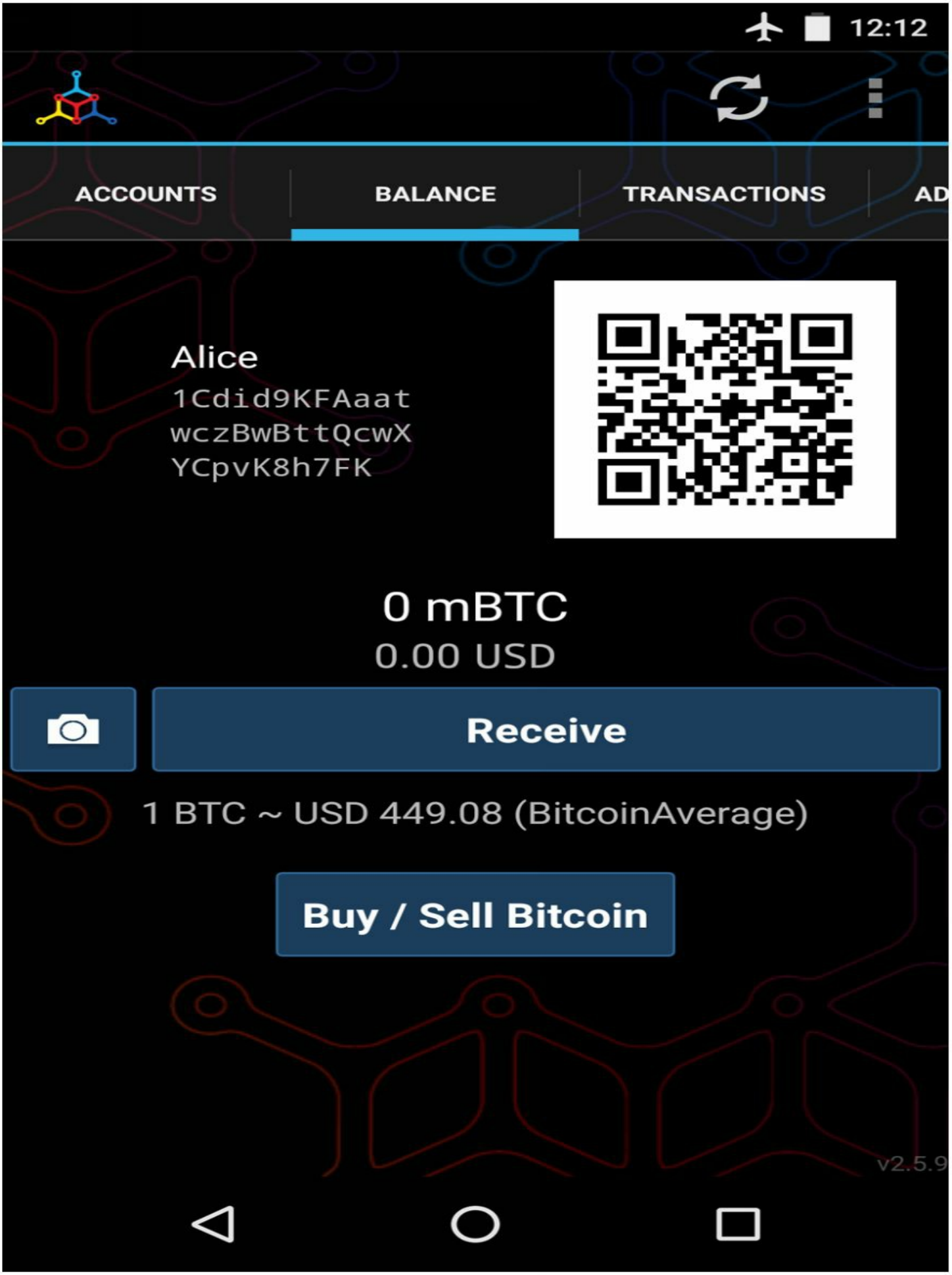

12:12
ACCOUNTS
BALANCE
TRANSACTIONS
AD
Alice
1Cdid9KFAaat
wczBwBttQcwX
YCpvK8h7FK
0 mBTC
0.00 USD
Receive
1 BTC ~ USD 449.08 (BitcoinAverage)
Buy / Sell Bitcoin
v2.5.9

Figura 1. Mycelium Mobile Wallet

La parte più importante di questa schermata è l'indirizzo *bitcoin di Alice*. Sullo schermo appare una lunga serie di lettere e numeri: 1Cdid9KFAaatwczBwBttQcwXYCpvK Accanto all'indirizzo bitcoin del portafoglio c'è un codice QR, una forma di codice a barre che contiene le stesse informazioni in un formato che può essere scansionato da una fotocamera dello smartphone. Il codice QR è il quadrato con un motivo di punti bianchi e neri. Alice può copiare l'indirizzo bitcoin o il codice QR nei suoi appunti toccando sul codice QR o

il pulsante Ricevi. Nella maggior parte dei portafogli, anche il codice QR viene ingrandito, in modo che possa essere scansionato più facilmente da una fotocamera dello smartphone.

Gli indirizzi Bitcoin iniziano con 1 o 3. Come gli indirizzi email possono essere condivisi con altri utenti bitcoin che possono usarli per inviare bitcoin direttamente al tuo portafoglio. Non c'è nulla di sensibile, dal punto di vista della sicurezza, sull'indirizzo bitcoin. Può essere

TIP

pubblicato ovunque
senza mettere a rischio la
sicurezza dell'account. A
differenza degli indirizzi
e-mail, puoi creare nuovi
indirizzi tutte le volte che
vuoi, che indirizzano tutti
i fondi al tuo portafoglio
In effetti, molti portafogli
moderni creano
automaticamente un
nuovo indirizzo per ogni
transazione per
massimizzare la privacy.
Un portafoglio è
semplicemente una
raccolta di indirizzi con
all'interno le chiavi che

	sbloccano i fondi.

Alice ora è pronta a ricevere fondi. L'applicazione del suo portafoglio ha generato in modo casuale una chiave privata insieme al corrispondente indirizzo bitcoin. A questo punto, il suo indirizzo bitcoin non è noto alla rete bitcoin o "registrato" con alcuna parte del sistema bitcoin. Il suo indirizzo bitcoin è semplicemente un numero che corrisponde a una chiave che può utilizzare per controllare l'accesso ai fondi. È stato generato indipendentemente dal suo portafoglio senza riferimento o registrazione con alcun servizio. Infatti, nella maggior parte dei portafogli, non esiste alcuna

associazione tra l'indirizzo bitcoin e qualsiasi informazione identificabile esternamente, compresa l'identità dell'utente. Fino al momento in cui questo indirizzo viene fatto riferimento come destinatario di valore in una transazione pubblicata sul registro bitcoin, l'indirizzo bitcoin è semplicemente parte del vasto numero di indirizzi validi in bitcoin. Solo una volta associato a una transazione, diventa parte degli indirizzi noti nella rete.

Alice è ora pronta per iniziare a usare il suo nuovo portafoglio bitcoin.

Ottenere il tuo primo Bitcoin

Il primo e spesso più difficile compito

per i nuovi utenti è acquisire alcuni bitcoin. A differenza di altre valute straniere, non è ancora possibile acquistare bitcoin presso una banca o un chiosco cambiavalute.

Le transazioni con Bitcoin sono irreversibili. La maggior parte delle reti di pagamento elettroniche come carte di credito, carte di debito, PayPal e trasferimenti bancari sono reversibili. Per chi vende bitcoin, questa differenza introduce un rischio molto elevato che l'acquirente con corrisponda un pagamento dopo aver ricevuto bitcoin, frodando il venditore. Per mitigare questo rischio, le aziende che accettano pagamenti elettronici tradizionali in cambio di bitcoin di

solito richiedono agli acquirenti di sottoporsi a verifiche di identità e controlli di affidabilità creditizia, che possono richiedere diversi giorni o settimane. Come nuovo utente, questo significa che non puoi comprare bitcoin all'istante con una carta di credito. Con un po 'di pazienza e pensiero creativo, tuttavia, non sarà necessario.

Ecco alcuni metodi per ottenere bitcoin come nuovo utente:

- Trova un amico che ha bitcoin e compra alcuni da lui o lei direttamente. Molti utenti bitcoin iniziano in questo modo. Questo

metodo è il meno complicato. Un modo per incontrare persone con bitcoin è partecipare a un meetup locale di bitcoin elencato su Meetup.com (https://bitcoin.meetup.com).

- Utilizza un servizio riservato come localbitcoins.com per trovare un venditore nella tua zona per comprare bitcoin per contanti in una transazione di persona.
- Guadagna bitcoin vendendo un prodotto o un servizio per bitcoin. Se

sei un programmatore, vendi le tue capacità di programmazione. Se sei un parrucchiere, taglia i capelli per bitcoin.

- Uso di ATM bitcoin nella propria città. Un ATM bitcoin è una macchina che accetta contanti e invia bitcoin al tuo portafoglio smartphone bitcoin. Trova un ATM bitcoin vicino a te utilizzando una mappa online da Coin ATM Radar (http://coinatmradar.com) .
- Utilizzare un Exchange di

Criptovalute collegato al proprio conto bancario. Molti paesi ora hanno exchange che offrono un mercato per acquirenti e venditori per scambiare bitcoin con valuta locale. I servizi di quotazione dei tassi di cambio, come BitcoinAverage (https://bitcoinaverage.com), mostrano spesso un elenco di exchange per ciascuna valuta.

	Uno dei vantaggi di bitcoin rispetto ad altri sistemi di pagamento è

TIP

che, se usato correttamente, offre agli utenti molta più privacy. Acquisire, detenere e spendere bitcoin non richiede che tu divulghi informazioni sensibili e personalmente identificabili a terzi. Tuttavia, dove il bitcoin tocca i sistemi tradizionali, come gli scambi di valute, spesso si applicano regolamenti nazionali e internazionali. Per scambiare bitcoin con la tua valuta nazionale, ti

verrà spesso richiesto di fornire la prova di identità e informazioni bancarie. Gli utenti devono essere consapevoli che una volta che un indirizzo bitcoin è collegato a un'identità, tutte le transazioni bitcoin associate sono anche facili da identificare e tracciare. Questo è uno dei motivi per cui molti utenti scelgono di mantenere account di scambio dedicati scollegati al loro

portafoglio.

Alice è stata presentata a bitcoin da un amico, quindi ha un modo semplice per acquisire il suo primo bitcoin. Successivamente, vedremo come compera bitcoin dal suo amico Joe e come Joe manda il bitcoin nel suo portafoglio.

Trovare il prezzo corrente di Bitcoin

Prima che Alice possa acquistare bitcoin da Joe, devono concordare sul *tasso di cambio* tra bitcoin e dollari USA . Questo fa sorgere una domanda comune per chi è nuovo in bitcoin: "Chi stabilisce il prezzo del bitcoin?"

La risposta breve è che il prezzo è fissato dai mercati.

Il bitcoin, come la maggior parte delle altre valute, ha un *tasso di cambio variabile* della moneta. Ciò significa che il valore del bitcoin nei confronti di qualsiasi altra valuta varia a seconda dell'offerta e della domanda nei vari mercati in cui viene scambiato. Ad esempio, il "prezzo" del bitcoin in dollari USA viene calcolato in ciascun mercato in base allo scambio più recente di bitcoin e dollari USA. Come tale, il prezzo tende a fluttuare minutamente più volte al secondo. Un servizio di determinazione dei prezzi aggregherà i prezzi di diversi mercati e calcolerà

una media ponderata in base al volume che rappresenta l'ampio tasso di cambio di mercato di una coppia di valute (ad esempio, BTC / USD).

Esistono centinaia di applicazioni e siti Web in grado di fornire il tasso corrente di mercato. Ecco alcuni dei più popolari:

Bitcoin Average *(http://bitcoinaverage.com)*

Un sito che fornisce una semplice visualizzazione della media ponderata per il volume per ciascuna valuta.

CoinCap *(http://coincap.io)*

Un servizio che elenca la capitalizzazione di mercato e i tassi

di cambio di centinaia di cripto-valute, incluso bitcoin.

Chicago Mercantile Exchange Bitcoin Reference Rate

(http://bit.ly/cmebrr)

Un tasso di riferimento che può essere utilizzato per riferimento istituzionale e contrattuale, fornito come parte dei feed di dati sugli investimenti da parte della CME.

Oltre a questi vari siti e applicazioni, la maggior parte dei portafogli bitcoin convertirà automaticamente importi tra bitcoin e altre valute. Joe utilizzerà il suo portafoglio per convertire automaticamente il prezzo prima di inviare bitcoin ad Alice.

Inviare e ricevere Bitcoin

Alice ha deciso di scambiare $ 10 dollari americani per bitcoin, in modo da non rischiare troppo per questa nuova tecnologia. Dà a Joe $ 10 in contanti, apre la sua applicazione wallet Mycelium e seleziona Receive. Questo mostra un codice QR con il primo indirizzo bitcoin di Alice.

Joe quindi seleziona Invia sul portafoglio nel suo smartphone e gli viene presentato uno schermo contenente due input:

- Un indirizzo bitcoin di destinazione
- L'importo da inviare, in bitcoin (BTC) o nella sua valuta locale (USD)

Nel campo di input per l'indirizzo bitcoin, c'è una piccola icona che assomiglia a un codice QR. Ciò consente a Joe di scansionare il codice a barre con la sua fotocamera dello smartphone in modo che non debba digitare l'indirizzo bitcoin di Alice, che è piuttosto lungo e difficile da scrivere. Joe tocca l'icona del codice QR e attiva la fotocamera dello smartphone, scansionando il codice QR visualizzato sullo smartphone di Alice.

Joe ora ha l'indirizzo bitcoin di Alice impostato come destinatario. Joe inserisce l'importo di $ 10 dollari USA e il suo portafoglio lo converte accedendo al tasso di cambio più

recente da un servizio online. Il tasso di cambio al momento è $ 100 dollari USA per bitcoin, quindi $ 10 dollari statunitensi vale 0.10 bitcoin (BTC), o 100 millibitcoin (mBTC) come mostrato nello screenshot dal portafoglio di Joe (vedi Figura 2).

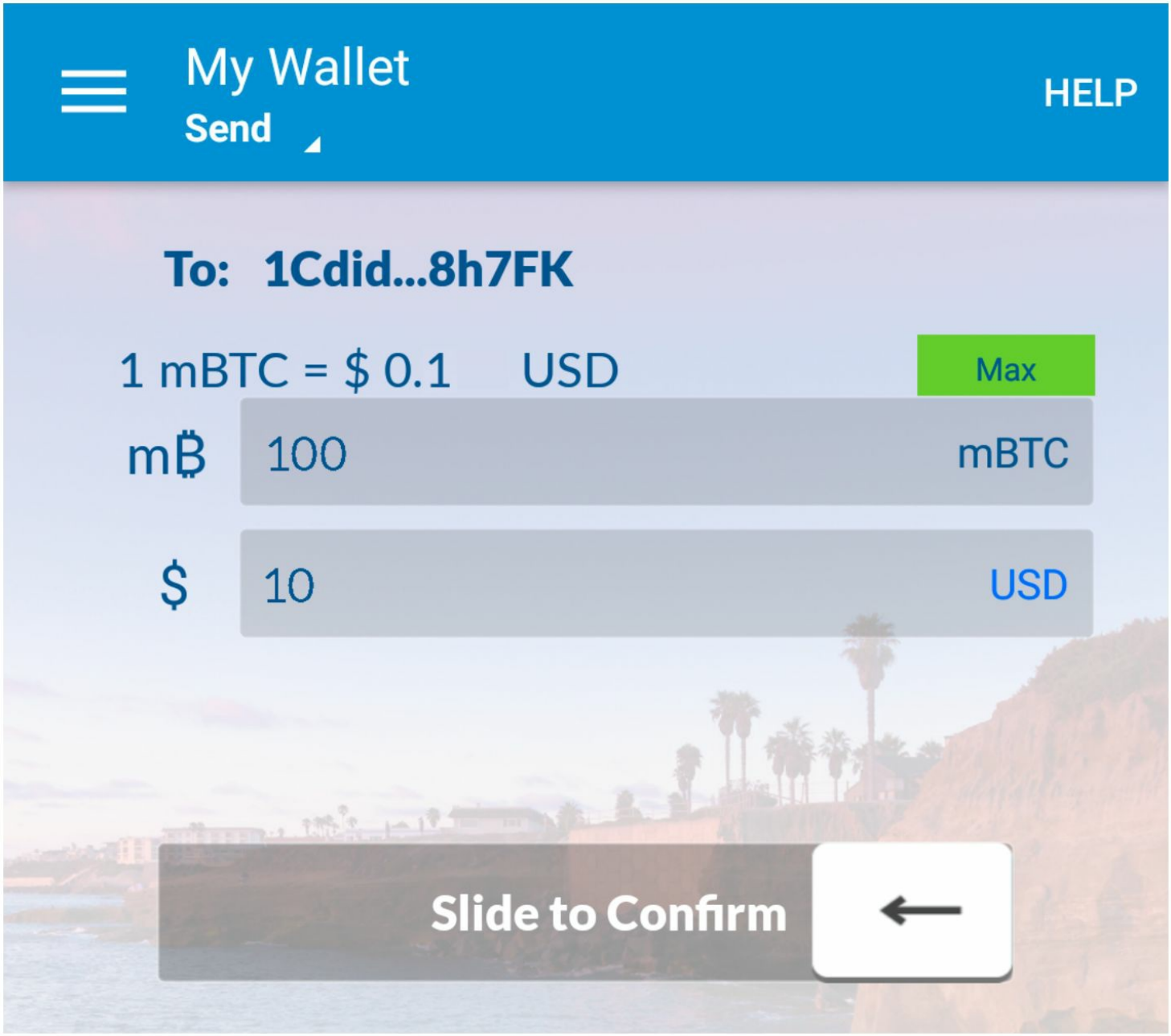


Figura 2. Schermata di invio del portafoglio bitcoin mobile Airbitz

Joe quindi controlla attentamente per

assicurarsi che abbia inserito l'importo corretto, perché sta per trasmettere denaro e gli errori sono irreversibili. Dopo aver ricontrollato l'indirizzo e l'importo, preme Invia per trasmettere la transazione. Il portafoglio di bitcoin mobile di Joe costruisce una transazione che assegna 0,10 BTC all'indirizzo fornito da Alice, reperendo i fondi dal portafoglio di Joe e firmando la transazione con le chiavi private di Joe. Questo dice alla rete bitcoin che Joe ha autorizzato un trasferimento di valore al nuovo indirizzo di Alice. Poiché la transazione viene trasmessa tramite il protocollo peer-to-peer, si propaga rapidamente attraverso la rete

bitcoin. In meno di un secondo, la maggior parte dei nodi ben collegati nella rete riceve la transazione e vede l'indirizzo di Alice per la prima volta.

Nel frattempo, il portafoglio di Alice è costantemente "in ascolto" delle transazioni pubblicate sulla rete bitcoin, cercando quelle che corrispondono agli indirizzi nei suoi portafogli. Pochi secondi dopo che il portafoglio di Joe trasmette la transazione, il portafoglio di Alice indicherà che sta ricevendo 0,10 BTC.

Conferme

All'inizio, l'indirizzo di Alice

mostrerà la transazione da Joe come "Unconfirmed". Ciò significa che la transazione è stata propagata alla rete ma non è ancora stata registrata nel registro delle transazioni bitcoin, noto come blockchain. Per essere confermata, una transazione deve essere inclusa in un blocco e aggiunta alla blockchain, che avviene in media ogni 10 minuti. In termini finanziari tradizionali questo è noto come *clearing*. Per ulteriori dettagli sulla propagazione, la convalida e la cancellazione (conferma) delle transazioni bitcoin, vedere il Capitolo 10.

Alice è ora l'orgogliosa proprietaria

di 0.10 BTC che può spendere. Nel prossimo capitolo vedremo il suo primo acquisto con bitcoin ed esamineremo le tecnologie di transazione e propagazione sottostanti in modo più dettagliato.

Come Funziona Bitcoin

Transazioni, Blocchi, Mining, e la Blockchain

Il sistema bitcoin, a differenza dei sistemi bancari e di pagamento tradizionali, si basa sulla fiducia decentralizzata. Invece di un'autorità fiduciaria centrale, in bitcoin la fiducia viene raggiunta come una proprietà emergente dalle interazioni dei diversi partecipanti al sistema. In questo capitolo, esamineremo bitcoin

da un livello elevato rintracciando una singola transazione attraverso il sistema bitcoin e osserveremo come diventa "attendibile" e accettata dal meccanismo del consenso distribuito e viene infine registrata sulla blockchain, il registro distribuito di Tutte le transazioni. I capitoli successivi approfondiranno la tecnologia alla base delle transazioni, della rete e del mining.

Panoramica su Bitcoin

Nel diagramma di sintesi mostrato in Figura 3, vediamo che il sistema bitcoin è composto da utenti con portafogli contenenti chiavi, da transazioni che vengono propagate

attraverso la rete e da minatori che producono (tramite calcolo competitivo) la blockchain di consenso, che è il registro affidabile di tutte le transazioni.

Ogni esempio in questo capitolo si basa su una transazione reale effettuata sulla rete bitcoin, simulando le interazioni tra gli utenti (Joe, Alice, Bob e Gopesh) inviando fondi da un portafoglio ad un altro. Durante il monitoraggio di una transazione attraverso la rete bitcoin nella blockchain, utilizzeremo un sito *blockchain explorer* per visualizzare ogni passaggio. Un blockchain explorer è un'applicazione web che funziona come un motore di ricerca

bitcoin, in quanto consente di cercare indirizzi, transazioni e blocchi e visualizzare le relazioni e i flussi tra di essi.

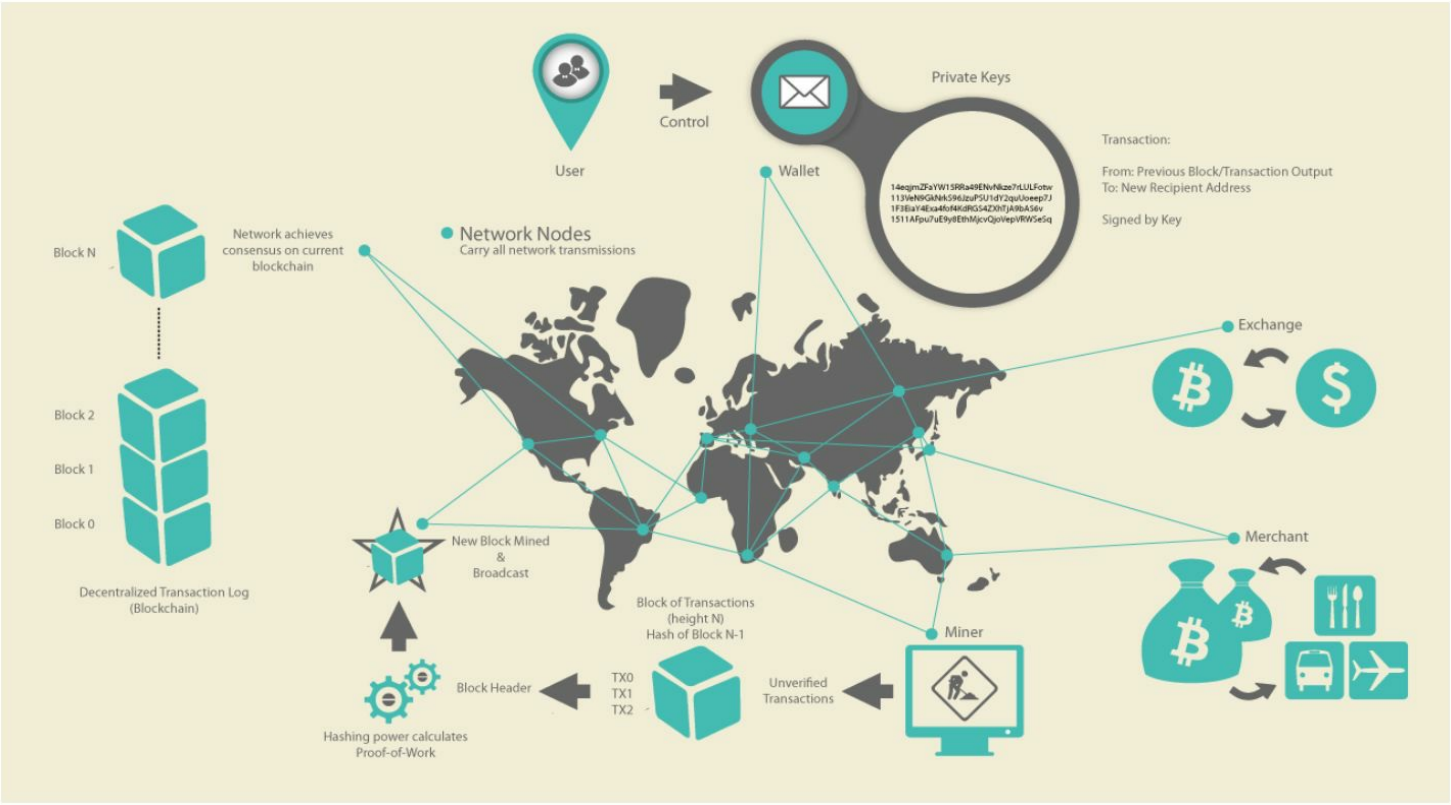


Figura 3. Panoramica su Bitcoin

I più popolari Blockchain Explorer comprendono:

- BlockCypher Explorer

  (https://live.blockcypher.com)

- blockchain.info

  (https://blockchain.info)

- BitPay Insight

  (https://insight.bitpay.com)

Ognuno di questi ha una funzione di ricerca che può essere usato con un indirizzo bitcoin, con hash della transazione, con un numero di blocco o hash del blocco e recuperare le informazioni corrispondenti dalla rete bitcoin. Con ogni esempio di transazione o blocco, forniremo un URL in modo che tu possa cercarlo da solo e studiarlo in dettaglio.

Pagare un Caffè

Alice, introdotta nel capitolo precedente, è un nuovo utente che ha appena acquisito il suo primo bitcoin. Nel capitolo "Ottieni il tuo primo Bitcoin", Alice ha incontrato il suo amico Joe per scambiarsi qualche soldo con bitcoin. La transazione creata da Joe ha finanziato il portafoglio Alice con 0.10 BTC. Ora Alice farà la sua prima transazione al dettaglio, comprando una tazza di caffè nella caffetteria di Bob a Palo Alto, in California.

Bob's Cafe ha recentemente iniziato ad accettare pagamenti in bitcoin aggiungendo un'opzione bitcoin al

proprio registratore di cassa. I prezzi di Bob's Cafe sono elencati nella valuta locale (dollari USA), ma alla cassa i clienti hanno la possibilità di pagare in dollari o in bitcoin. Alice fa il suo ordine per una tazza di caffè e Bob lo inserisce nel registratore di cassa, come fa per tutte le transazioni. Il POS converte automaticamente il prezzo totale da dollari USA a bitcoin al tasso di mercato prevalente e visualizza il prezzo in entrambe le valute:

```
Totale:
$1.50 USD
0.015 BTC
```

Bob dice "Questo è un dollaro, cinquanta o quindici millibitcoin".

Il registratore di cassa di Bob creerà automaticamente anche un codice QR speciale contenente una *richiesta di pagamento* (vedi Figura 4).

A differenza di un codice QR che contiene semplicemente un indirizzo bitcoin di destinazione, una richiesta di pagamento è un URL codificato in QR che contiene un indirizzo di destinazione, un importo di pagamento e una descrizione generica come "Bob’s Cafe". Ciò consente a un bitcoin wallet di precompilare le informazioni utilizzate per inviare il pagamento e mostrare all’utente una descrizione a misura d’uomo. È possibile eseguire la scansione del codice QR con un’applicazione

portafoglio bitcoin per vedere cosa vedrebbe Alice.

Figura 4. QR code per la richiesta di pagamento

TIP	Prova a scansionare questo con il tuo portafoglio per vedere

	l’indirizzo e l’importo ma NON INVIARE DENARO.

Il codice QR della richiesta di pagamento codifica il seguente URL, definito in BIP-21:

```
bitcoin:1GdK9UzpHBzqzX2A9JFP3Di4weB
amount=0.015&
label=Bob%27s%20Cafe&
message=Purchase%20at%20Bob%27s%20
```

Parti dell'URL

Un indirizzo bitcoin: "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmo
L'importo del pagamento: "0.015"
Un'etichetta per l'indirizzo del destinatario: "Bob's Cafe"
Una descrizione del pagamento: "Acquisto

al Bar di Bob"

Alice usa il suo smartphone per scansionare il codice a barre sul display. Il suo smartphone mostra un pagamento di 0.0150 BTC a Bob's Cafe e seleziona "Invia" per autorizzare il pagamento. Entro pochi secondi (all'incirca lo stesso periodo di una operazione con una carta di credito), Bob vede la transazione sul registro, completando la transazione.

Nelle sezioni seguenti, esamineremo questa transazione in modo più dettagliato. Vedremo come è stato costruito il portafoglio di Alice, come è stato propagato attraverso la rete, come è stato verificato e, infine, come Bob può spendere tale importo nelle

transazioni successive.

NOTA	La rete bitcoin può eseguire transazioni in valori frazionari, ad es. da millibitcoin (1/1000 di bitcoin) fino a 1 / 100.000.000 di bitcoin, che è noto come satoshi. In questo libro useremo il termine "bitcoin" per riferirsi a qualsiasi quantità di valuta bitcoin dall'unità più piccola (1 satoshi) al numero totale (21.000.000) di tutti i bitcoin che

	verranno mai estratti.

Puoi esaminare la transazione di Alice verso Bob's Cafe sulla blockchain utilizzando un sito block explorer (Esempio 1):

```
https://blockexplorer.com/tx/0627052b6f
```

Transazioni Bitcoin

In termini semplici, una transazione comunica alla rete che il proprietario di alcuni bitcoin ha autorizzato il trasferimento di tale valore ad un altro proprietario. Il nuovo proprietario può

ora spendere il bitcoin creando un'altra transazione che autorizza il trasferimento a un altro proprietario e così via, in una catena di proprietà.

Input e output delle transazioni

Le transazioni sono come le righe di un registro contabile a partita doppia. Ogni transazione contiene uno o più "input", che sono come debiti contro un account bitcoin. Dall'altra parte della transazione, ci sono uno o più "output", che sono come crediti aggiunti a un account bitcoin. Gli input e gli output (debiti e crediti) non si sommano necessariamente allo stesso importo. Invece, gli output si sommano

a un po 'meno degli input e la differenza rappresenta una implicita *transaction fee*, che è un piccolo pagamento raccolto dal minatore che inserisce la transazione nel libro mastro. Una transazione bitcoin viene mostrata come una voce di libro mastro contabile in Figura 5.

La transazione contiene anche una prova di proprietà per ogni quantità di bitcoin (input) il cui valore viene speso, sotto forma di una firma digitale dal proprietario, che può essere validata indipendentemente da chiunque. In termini di bitcoin, "spending" sta firmando una transazione che trasferisce il valore da una transazione precedente a un nuovo

proprietario identificato da un indirizzo bitcoin.

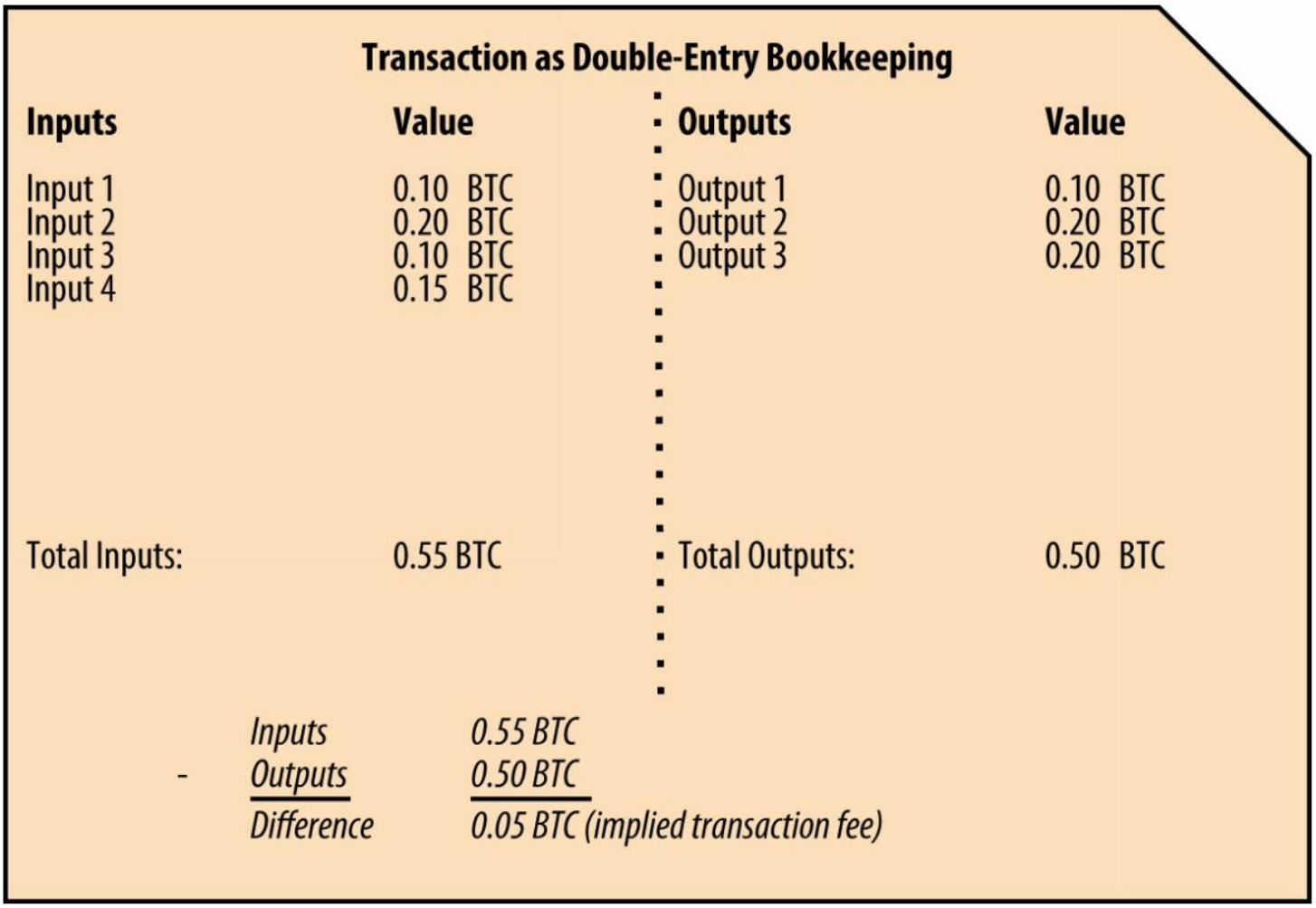


Figura 5. Transazioni come libro di contabilità in partita doppia

Catene di transazioni

Il pagamento di Alice a Bob's Cafe utilizza l'output di una transazione precedente come input. Nel capitolo precedente, Alice ha ricevuto bitcoin dal suo amico Joe in cambio di denaro. Quella transazione ha creato un valore bitcoin bloccato dalla chiave di Alice. La sua nuova transazione a Bob's Cafe fa riferimento alla transazione precedente come input e crea nuovi output per pagare la tazza di caffè e ricevere il resto. Le transazioni formano una catena, in cui gli input dell'ultima transazione corrispondono agli output delle transazioni precedenti. La chiave di Alice fornisce la firma che sblocca quelle uscite di transazione precedenti,

dimostrando così alla rete bitcoin di essere proprietaria dei fondi. Lei allega il pagamento per il caffè all'indirizzo di Bob, quindi "carica" quell'output con il requisito che Bob produca una firma per poter spendere tale importo. Questo rappresenta un trasferimento di valore tra Alice e Bob. Questa catena di transazioni, da Joe a Alice a Bob, è illustrata in Figura 6.

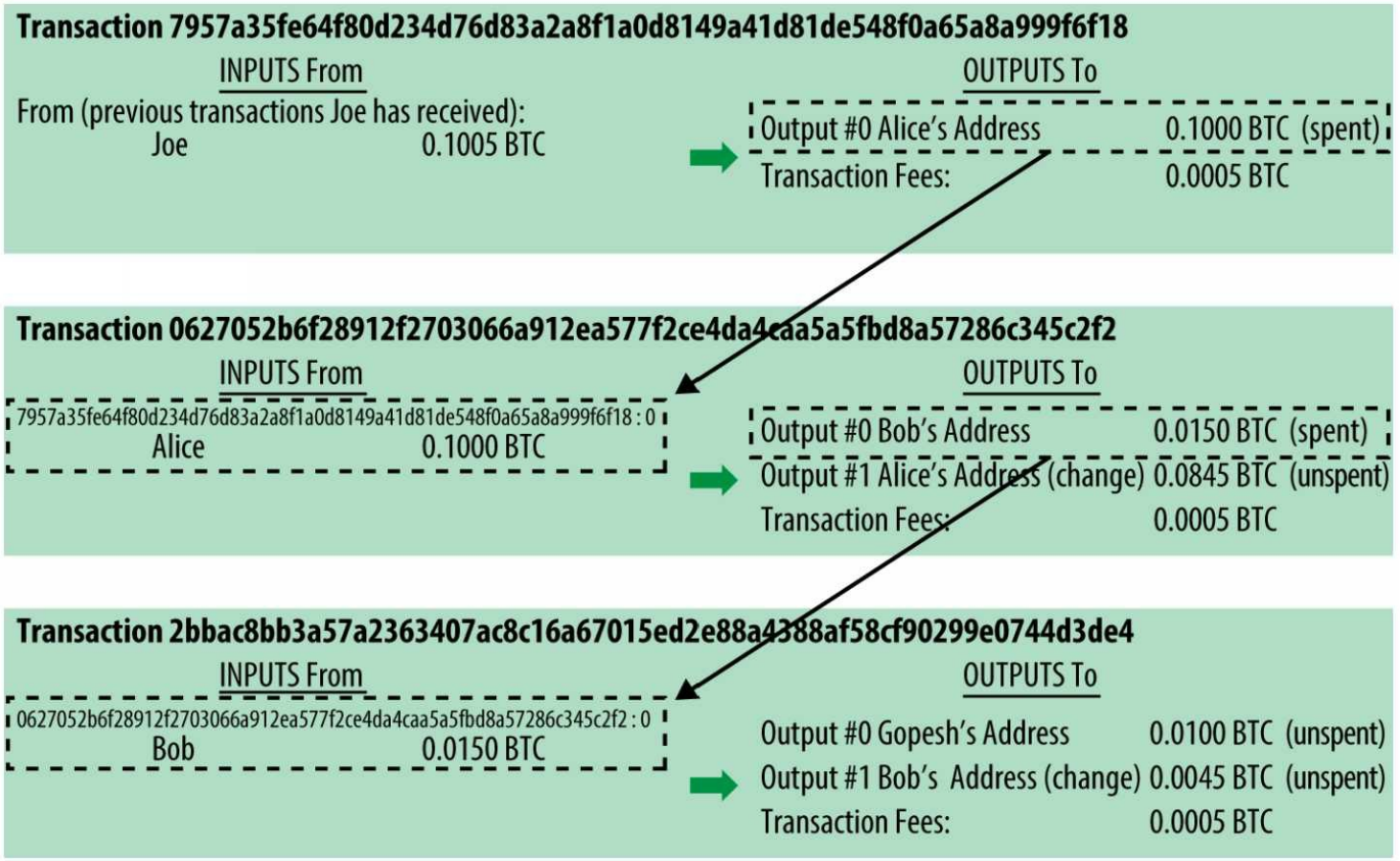


Figura 6. Una catena di transazioni, dove l'output di una transazione è l'input della transazione successiva

Molte transazioni bitcoin includeranno uscite che fanno riferimento sia a un indirizzo del nuovo proprietario e un indirizzo del proprietario corrente,

chiamato indirizzo *change* (di cambio). Questo perché gli input di transazione, come le banconote, non possono essere divisi. Se acquisti un oggetto da $ 5 USD in un negozio ma usi una banconota da $ 20 per pagare l'articolo, ti aspetti di ricevere $ 15 di dollari in cambio. Lo stesso concetto si applica agli input delle transazioni bitcoin. Se hai acquistato un articolo che costa 5 bitcoin ma ha solo un input da 20 bitcoin da utilizzare, invierai un output di 5 bitcoin al proprietario del negozio e un output di 15 bitcoin a te stesso come modifica (meno eventuali commissioni di transazione applicabili). È importante sottolineare che l'indirizzo del resto non deve

essere lo stesso indirizzo di quello dell'input e per motivi di privacy è spesso un nuovo indirizzo del portafoglio del proprietario.

Portafogli diversi possono utilizzare strategie diverse quando si aggregano gli input per effettuare un pagamento richiesto dall'utente. Possono aggregare molti piccoli input o utilizzarne uno uguale o superiore al pagamento desiderato. A meno che il portafoglio non possa aggregare gli input in modo tale da corrispondere esattamente al pagamento desiderato più le spese di transazione, il portafoglio dovrà generare alcuni resti. Questo è molto simile a come le persone gestiscono i contanti. Se in

tasca usi sempre banconote di taglia grande, finirai con una tasca piena di spiccioli. Se usi solo gli spiccioli, avrai sempre solo banconote grandi. La gente trova inconsciamente un equilibrio tra questi due estremi e gli sviluppatori dei wallet bitcoin si sforzano di programmare questo equilibrio.

In sintesi, le *transazioni* spostano valore dagli *input di transazione* agli *output di transazione*. Un input è un riferimento all'output di una transazione precedente, che mostra da dove proviene il valore. Un output di transazione invia un valore specifico ad un indirizzo bitcoin di un nuovo proprietario e può includere un output

di resto al proprietario originale. Gli output di una transazione possono essere utilizzati come input in una nuova transazione, creando così una catena di proprietà quando il valore viene spostato da proprietario a proprietario (vedi Figura 2-4).

Forme di Transazioni Comuni

La forma più comune di transazione è un semplice pagamento da un indirizzo a un altro, che spesso include alcuni "resti" restituiti al proprietario originale. Questo tipo di transazione ha un input e due output ed è mostrato in Figura 6.

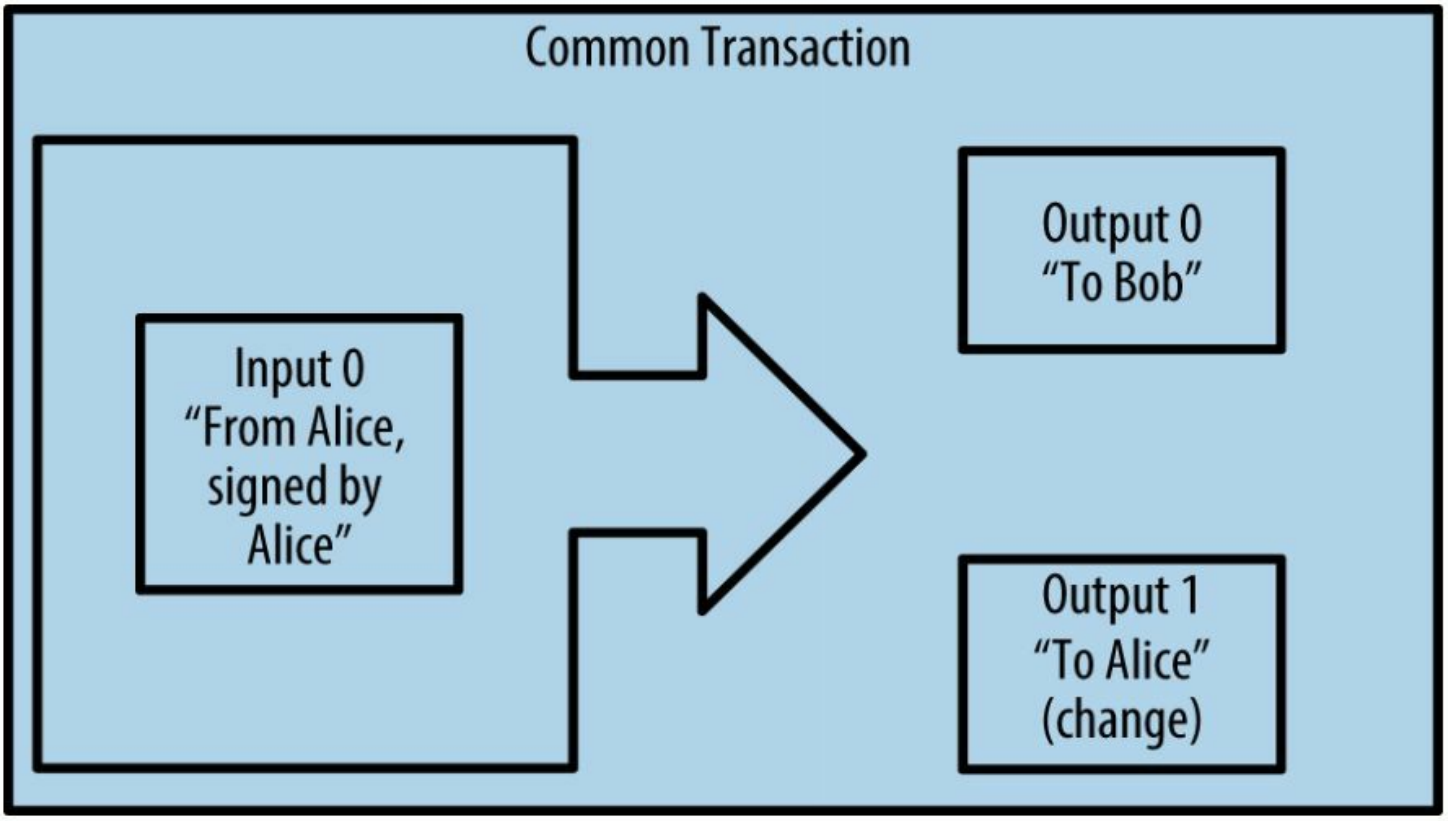


Figura 7. Transazione più comune

Un'altra forma comune di transazione è quella che aggrega multipli input in un singolo output (vedi Figura 7). Questo rappresenta l'equivalente nel mondo reale dello scambiare una pila di monete e banconote per una banconota singola di valore maggiore.

Transazioni come queste sono talvolta generate da applicazioni wallet per "far pulizia" di transazioni di valore piccolo che sono state ricevute come resto dei precedenti pagamenti.

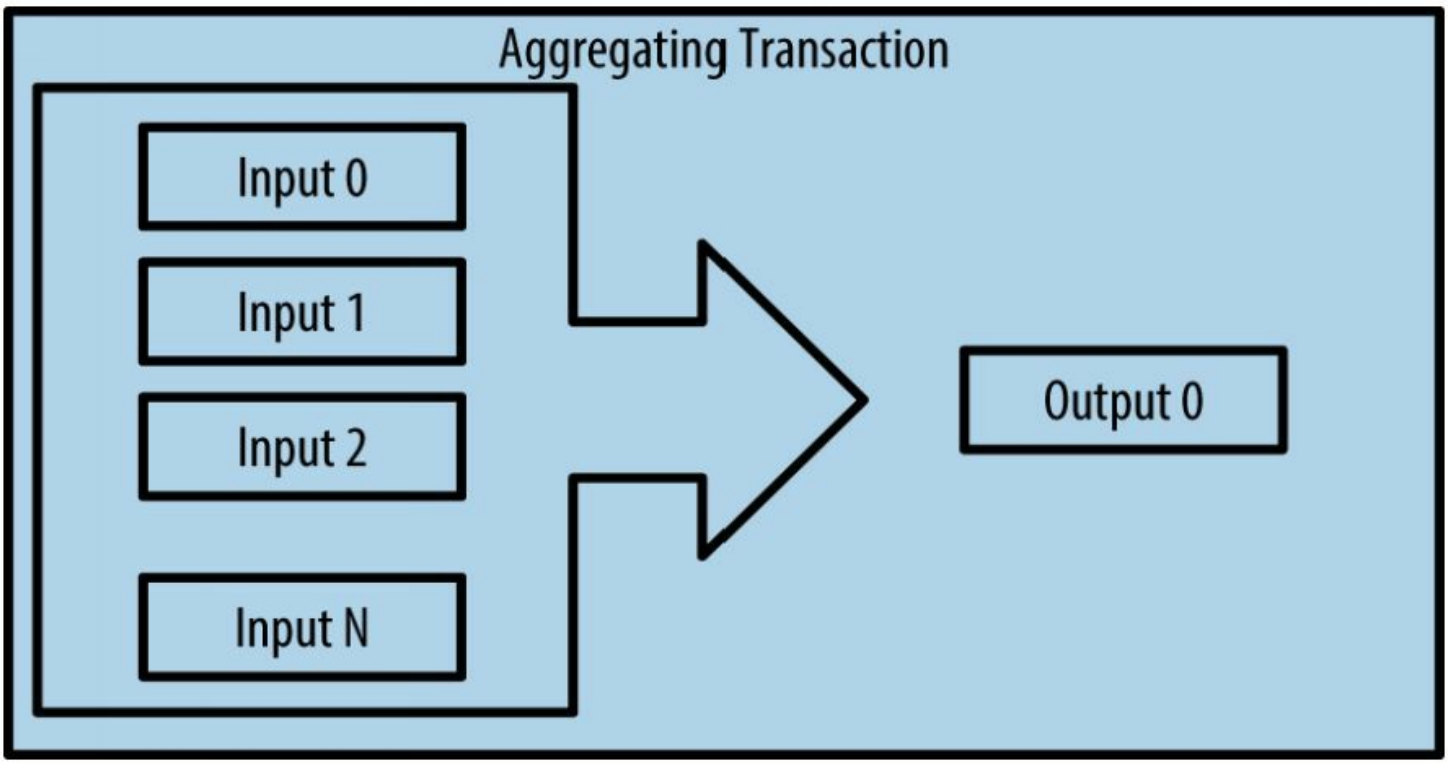


Figura 8. Transazioni aggregratrici di fondi

Infine, un altro tipo di transazione che viene spesso visualizzata sul registro di bitcoin è una transazione che

distribuisce un input a più output che rappresentano più destinatari (vedi Figura 9). Questo tipo di transazione viene talvolta utilizzato da entità commerciali per distribuire fondi, ad esempio quando si elaborano pagamenti di salari a più dipendenti.

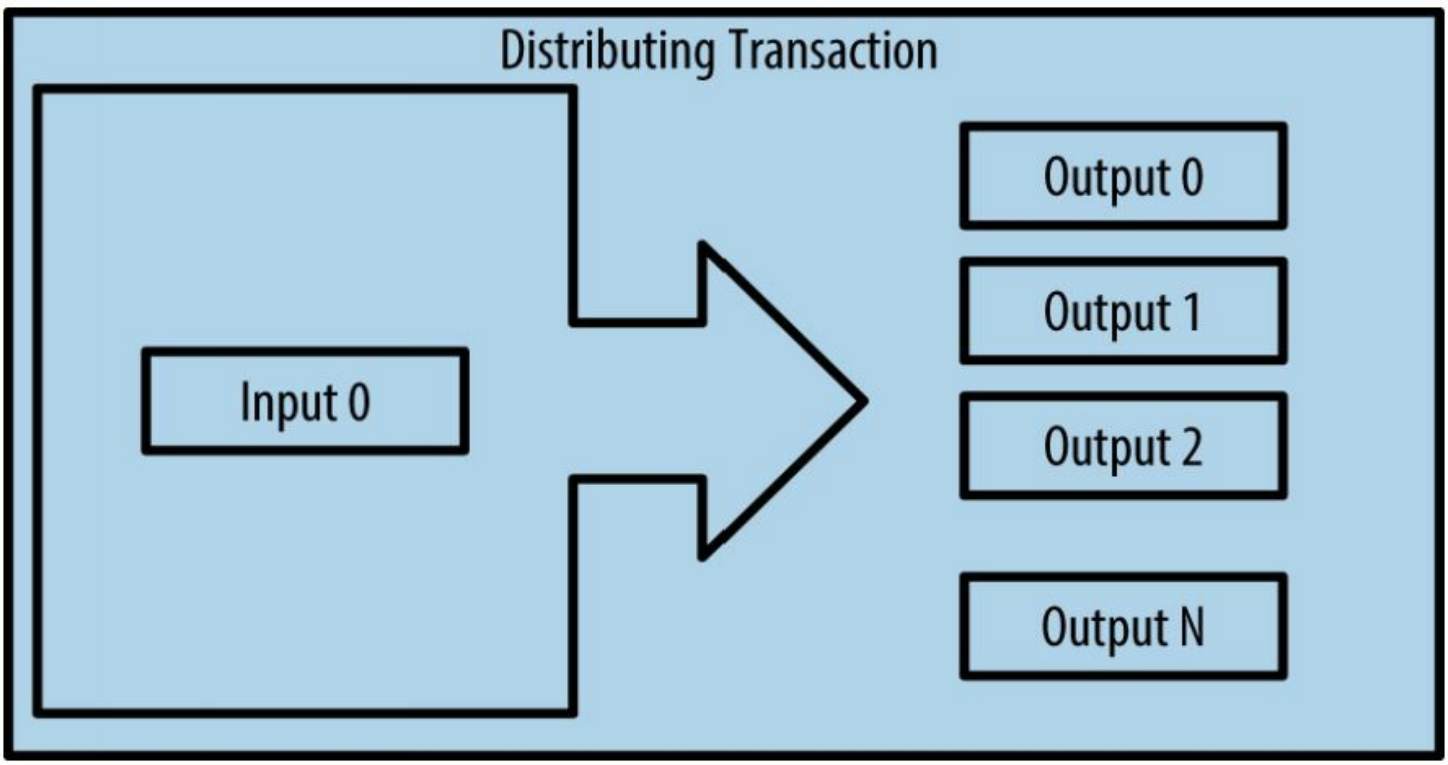


Figura 9. Transazioni che distribuiscono fondi

Costruire una Transazione

Il wallet di Alice contiene tutta la logica per selezionare gli input e gli output appropriati per creare una transazione sulle richieste di Alice. Alice ha solo bisogno di specificare una destinazione e una quantità, e il resto avviene nell'applicazione wallet senza che lei veda i dettagli. È importante sottolineare che un wallet può costruire transazioni anche se è completamente offline. E' come compilare un assegno a casa e successivamente inviarlo alla banca in una busta, allo stesso modo la transazione non ha bisogno di essere

costruita e firmata mentre il wallet è connesso alla rete bitcoin.

Ottenere gli Input Giusti

Il portafoglio di Alice deve prima trovare input che possano pagare l'importo che lei desidera inviare a Bob. La maggior parte dei wallet tiene traccia di tutte le uscite disponibili che appartengono agli indirizzi nel wallet. Pertanto, il portafoglio di Alice conterrebbe una copia dell'uscita della transazione dalla transazione di Joe, che è stata creata in cambio di denaro. Un'applicazione portafoglio bitcoin che viene eseguita come client full-node contiene effettivamente una copia di ogni output non speso da ogni

transazione nella blockchain. Ciò consente ad un wallet di costruire input di transazione e di verificare rapidamente le transazioni in entrata come aventi input corretti. Tuttavia, poiché un client full-node occupa molto spazio su disco, la maggior parte dei wallet utente esegue client "leggeri" che tracciano solo gli output non spenti dell'utente.

Se l'applicazione wallet non mantiene una copia degli unspent output, può interrogare il network bitcoin per recuperare quest'informazione, utilizzando una varietà di API disponibili da servizi differenti o può interrogare un nodo con un'application programming interface (API) call.

L'Esempio 2 mostra una richiesta API, costruita come un comando HTTP GET verso un URL specifico. Questo URL restituirà tutti gli unspent output delle transazioni riguardanti un indirizzo, fornendo a qualunque applicazione le informazioni necessarie per costruire gli input della transazione per spenderli. Utilizziamo un semplice client HTTP da riga di comando cURL per recuperare la risposta.

Esempio 2. Ricerca tutti gli unspent output (output non spesi) per l'indirizzo bitcoin di Alice

```
$ curl
https://blockchain.info/unspent?
active=1Cdid9KFAaatwczBwBttQcwXYC
```


```
{

"unspent_outputs":[

    {
    "tx_hash":"186f9f998a5...2836dd734d28
    "tx_index":104810202,
    "tx_output_n": 0,
    "script":"76a9147f9b1a7fb68d60c536c2
    "value": 10000000,
    "value_hex": "00989680",
    "confirmations":0
    }
```

```
]
}
```

La risposta nell'Esempio 2 mostra un unspent output (un output che non è stato ancora speso) di proprietà dell'indirizzo di Alice 1Cdid9KFAaatwczBwBttQcwXYCpvK La risposta include una referenza alla transazione nella quale questo unspent output è contenuto (il pagamento fatto da Joe) e il suo valore in satoshi, 10 milioni, equivalente a 0.10 bitcoin. Con questa informazione, l'applicazione wallet di Alice può costruire la transazione per trasferire quel valore all'indirizzo della nuova proprietaria.

TIP	Visualizza la transazione da Joe a Alice (http://bit.ly/1tAeeGr)

Come puoi notare, il wallet di Alice contiene abbastanza bitcoin in un singolo unspent output per il pagamento del caffè. Se non fosse stato così, l'applicazione wallet di Alice avrebbe dovuto "frugare" nella pila di unspent output più piccoli, come scegliere delle monete da un borsellino fino a che non se ne saranno trovate abbastanza da poter pagare il caffè. In entrambi i casi, ci sarà probabilmente bisogno di ottenere indietro un po di resto, il quale vedremo nella prossima sezione,

quando l’applicazione wallet creerà gli output della transazione (pagamenti).

Creare gli Output

Un nuovo output di transazione è creato nella forma di uno script che crea un blocco sul valore e può essere riscattato solo da l’introduzione di una soluzione allo script. In termini più semplici, l’output di transazione di Alice conterrà uno script che dice qualcosa come, "Questo output è pagabile a chiunque possa presentare una firma dalla chiave corrispondente all’indirizzo pubblico di Bob." Visto che solo Bob ha il wallet con le chiavi corrispondenti a quell’indirizzo, solo

il wallet di Bob potrà presentare una simile firma per riscattare questo output. Alice quindi "bloccherà" il valore dell'output con una richiesta di una firma da Bob.

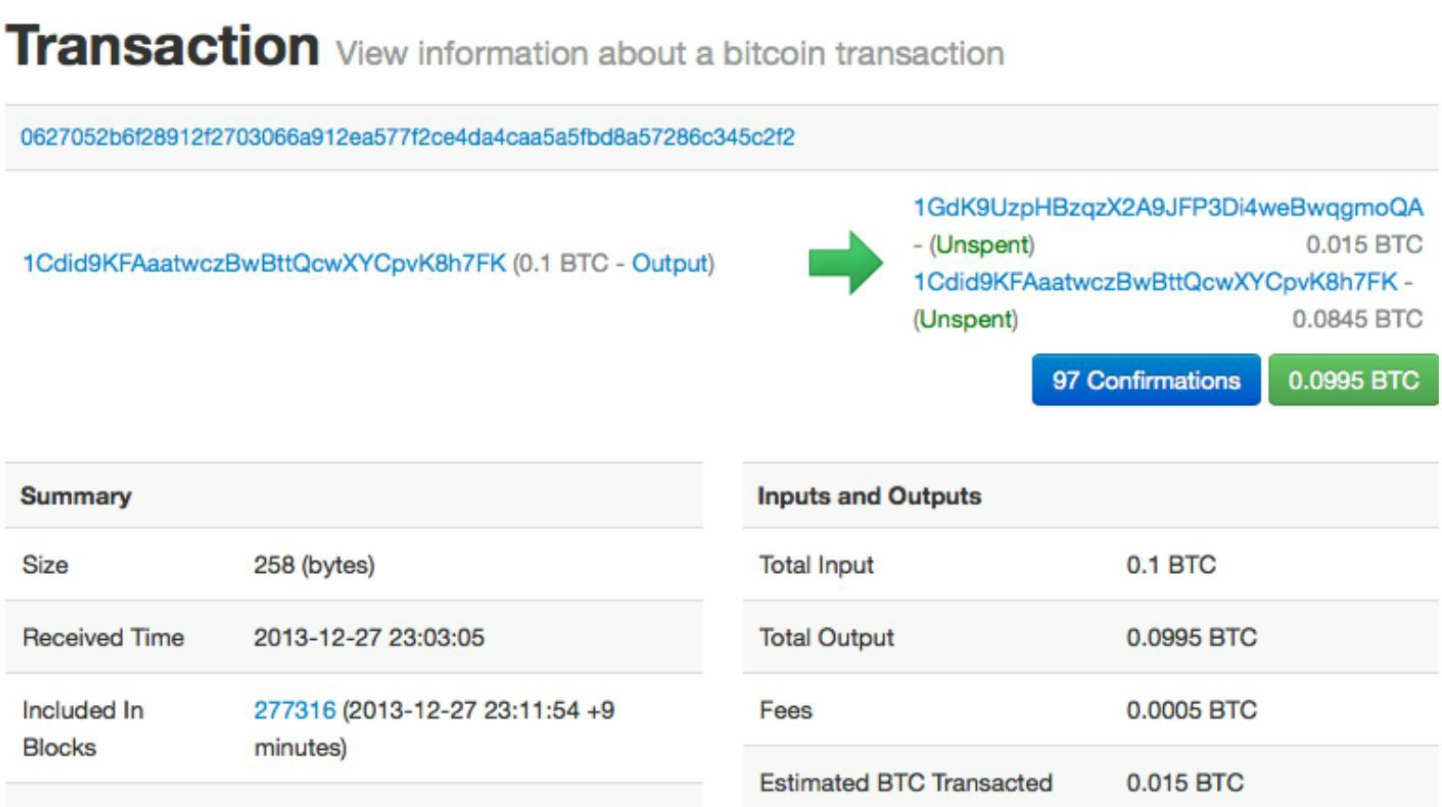
Questa transazione inoltre includerà un secondo output, visto che i fondi di Alice sono nella forma di un output di 0.10 BTC, troppi soldi per la tazza di caffè da 0.015 BTC. Ad Alice spetteranno 0.085 9 BTC di resto. Il pagamento per il resto di Alice è creato dal wallet di Alice proprio nella stessa transazione usata per il pagamento a Bob. Essenzialmente, il wallet di Alice divide i fondi in due pagamenti: uno a Bob, e uno indietro a se stessa. Alice può infine usare

l'output del resto in una transazione successiva, spendendolo in un secondo momento.

Infine, perchè il network processi la transazione in un tempo ragionevole, l'applicazione wallet di Alice aggiungerà una piccola commissione (fee) di transazione. Questa non è esplicita nella transazione; è data dalla differenza tra gli input e gli output. Se invece di prendere 0.085 come resto, Alice crea solo 0.0845 come secondo output, si avrà un 0.0005 BTC (mezzo millibitcoin) rimanente. L'input di 0.10 BTC non è speso completamente con i due output, perchè la loro somma sarà inferiore a 0.10. La differenza risultante è la transaction fee

(commissione della transazione) che sarà recuperata dal miner come commissione per aver incluso la transazione in un blocco e averla scritta sul registro blockchain.

La transazione risultante può essere vista utilizzando un'applicazione web blockchain explorer, come mostrato in Figura 10.



Transaction View information about a bitcoin transaction

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK (0.1 BTC - Output)

1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA - (Unspent) 0.015 BTC

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK - (Unspent) 0.0845 BTC

97 Confirmations | 0.0995 BTC

Summary	
Size	258 (bytes)
Received Time	2013-12-27 23:03:05
Included In Blocks	277316 (2013-12-27 23:11:54 +9 minutes)

Inputs and Outputs	
Total Input	0.1 BTC
Total Output	0.0995 BTC
Fees	0.0005 BTC
Estimated BTC Transacted	0.015 BTC

Figura 10. La transazione di Alice al Bar di Bob

TIP	Visualizza la transazione da Alice al Bar di Bob (https://bit.ly/1u0FIGs).

Aggiungere la Transazione al Ledger (libro mastro)

La transazione creata dall'applicazione wallet di Alice ha una grandezza di 258 byte e contiene tutto il necessario per confermare la proprietà dei fondi e assegnare loro

nuovi proprietari. A questo punto, la transazione deve essere trasmessa al network bitcoin dove diventerà parte del registro distribuito (la blockchain). Nella prossima sezione vedremo come una transazione diviene parte di un nuovo blocco e come questo blocco è "minato". Infine, noteremo come questo nuovo blocco, una volta aggiunto alla blockchain, acquista sempre più "fiducia" dal network man mano che vengono aggiunti ulteriori blocchi.

Trasmettere la transaction

Visto che la transazione contiene tutte le informazioni necessarie per essere processata, non importa come o da

dove sia trasmessa al network bitcoin. La rete bitcoin è un network peer-to-peer, dove ogni client bitcoin partecipa connettendosi a altri numerosi client bitcoin. Lo scopo del network bitcoin è di propagare transazioni e blocchi a tutti i partecipanti.

Come si propaga

Ogni sistema, come un server, applicazione desktop, o wallet, che partecipa al network bitcoin "parlando" il protocollo bitcoin è chiamato *nodo bitcoin*. L'applicazione wallet di Alice può inviare la nuova transazione a qualsiasi altro client bitcoin sia raggiungibile tramite una

connessione Internet: via cavo, WiFi, o via cellulare. Il wallet bitcoin di Alice non deve essere connesso direttamente al wallet bitcoin di Bob e non dovrà usare la connessione Internet offerta dal bar, anche se entrambe siano valide opzioni. Ogni nodo (altro client) del network bitcoin che riceva una transazione valida che non è stata vista prima la inoltrerà ad altri nodi ai quali è connesso, una tecnica di propagazione chiamata *flooding*. Così facendo, la transazione si propagherà rapidamente attraverso il network peer-to-peer, raggiungendo una gran percentuale di nodi in pochi secondi.

Il Punto di Vista di Bob

Se l'applicazione wallet bitcoin di Bob è direttamente connessa all'applicazione wallet di Alice, l'applicazione wallet di Bob potrebbe essere il primo nodo a ricevere la transazione nel giro di pochi secondi. Il wallet di Bob identificherà immediatamente la transazione di Alice come un pagamento in ricezione perchè contiene output redimibile dalle chiavi di Bob. L'applicazione wallet di Bob può anche indipendentemente verificare che la transazione sia formata correttamente, che utilizzi come input, i precedenti output non spesi, e che contenga sufficienti fee di transazione per essere inclusa nel blocco successivo. A

questo punto Bob può assumere, con poco rischio, che la transazione sarà nel giro di poco tempo inclusa in un blocco e confermata.

	Un idea sbagliata sulle transazioni bitcoin è che debbano essere "confermate" aspettando 10 minuti per un nuovo blocco, o fino a 60 minuti per sei conferme. Anche se le conferme assicurano che la transazione sia stata accettata dall'intera rete un ritardo del genere non è necessario per un valore di transazione

TIP	piccolo come quello per pagare un caffè. Un venditore può accettare come valida una transazione di piccolo valore senza conferme senza incorrere in un rischio maggiore di un pagamento tramite carta di credito senza che si mostri un documento di identità o una firma cosa che i venditori ad oggi accettano senza problemi.

Il Mining di Bitcoin

La transazione è in questo modo propagata sulla rete bitcoin. Non entra a far parte del libro mastro condiviso (la blockchain) fino a che non è verificata e inclusa in un blocco da un processo chiamato mining.

Il sistema della fiducia usato da bitcoin è basato sulla computazione. Le transazioni sono raggruppate in *blocchi*, che richiedono un'enorme quantità di potenza computazionale per essere verificate come valide. Il processo di mining in bitcoin serve a due scopi:

- I nodi miner validano tutte le transazioni seguendo le regole del

consenso di bitcoin. Pertanto, mining offre sicurezza per le transazioni bitcoin rifiutando transazioni non valide o non corrette.

- Il Mining crea nuovi bitcoin per ogni blocco, quasi come una banca centrale emette nuova moneta. La quantità di bitcoin creata per blocco è fissa e diminuisce col tempo, seguendo un programma di emissione fisso.

Il mining raggiunge un buon equilibrio tra costi e premi. L'industria di mining usa l'elettricità per risolvere un problema matematico. Un minatore di successo raccoglierà un *reward* sotto forma di nuovi bitcoin e commissioni

delle transazioni. Tuttavia, il premio verrà raccolto solo se il minatore ha correttamente convalidato tutte le transazioni, seguendo le regole del *consenso*. Questo delicato equilibrio offre sicurezza per bitcoin senza un'autorità centrale.

Un buon modo per descrivere l'attività di mining è quello di immaginarsi una partita di Sudoku avente un tabellone di gioco veramente enorme che si resetta ogni volta che una soluzione viene trovata e la quale difficoltà di gioco si aggiusta di modo che si impieghi approssimativamente 10 minuti per trovare una soluzione. Immaginati una partita di sudoku gigante, con una dimensione di

numerose righe e colonne. Se io ti mostro un tabellone completato, tu potrai facilmente e velocemente verificarne la correttezza. Al contrario, se il tabellone ha qualche casella piena e molte vuote, ci vorrà un notevole sforzo (lavoro, "work") per risolverlo! La difficoltà del sudoku può essere aggiustata cambiando la dimensione del tabellone (aumentando o riducendo le righe e le colonne), ma potrà comunque essere verificato piuttosto facilmente anche se esso è molto grande in dimensioni. Il "problema da risolvere" usato in bitcoin è basato su di un hash crittografico e esibisce caratteristiche simili al problema di risoluzione di un

puzzle come il sudoku: è asimmetricamente difficile da risolvere ma molto facile da verificare, e la sua difficoltà può essere aggiustata.

Precedentemente abbiamo introdotto Jing, un imprenditore di Shangai. Jing gestisce una *mining farm*, che è un'azienda che gestisce migliaia di mining computer, competendo con tutti gli altri miner per il premio. Ogni circa 10 minuti, Jing entra a far parte insieme agli altri miner in una gara globale per trovare una soluzione per un blocco di transazioni. Trovare una soluzione simile, la cosi-chiamata "Proof of Work" (prova di lavoro), richiede decine di miliardi in

operazioni di hashing al secondo in tutta la rete bitcoin. L'algoritmo di Proof-of-Work comporta l'effettuare ripetutamente un'operazione di hashing dell'header del blocco e un numero casuale con l'algoritmo crittografico SHA256 fino a che non emerga una soluzione corrispondente a un determinato pattern. Il primo miner a trovare tale soluzione vince il round della competizione e pubblica il blocco nella blockchain.

Jing ha iniziato a partecipare al mining nel 2010 utilizzando un computer fisso molto veloce per trovare una proof of work per nuovi blocchi adeguata. Man mano che altri miner hanno iniziato a far parte del network bitcoin, la

difficoltà del problema è aumentata rapidamente. Presto, Jing e altri miner sono passati a aggiornamenti hardware più specializzati, come a schede grafiche dedicate (GPUs) di fascia alta come quelle usate su computer fissi per gaming o su varie console di gioco. Al momento della scrittura di questo libro, la difficoltà è così alta che è redditizio effettuare mining solo con circuiti integrati specifici per l'applicazione selezionata (application-specific integrated circuits, ASIC), essenzialmente centinaia di algoritmi di mining stampati su chip hardware, eseguiti in parallelo su un singolo chip al silicio. Jing inoltre è entrato a far parte di una

mining pool la quale pressoché nello stesso modo in cui una lottery pool permette a molti partecipanti di entrare a far parte di essa e dividere gli sforzi e i proventi. Jing al momento gestisce migliaia di macchine ASIC per effettuare mining di bitcoin 24 ore al giorno. Jing paga I costi per l'elettricità consumata rivendendo i bitcoin che è capace di generare dal mining, creando un po' di guadagno dai profitti.

Effettuare Mining delle Transazioni presenti nei Blocchi

Nuove transazioni fluiscono

costantemente nel network da wallet utente e altre applicazioni. Non appena queste transazioni vengono notate dai nodi del network bitcoin, esse vengono aggiunte a una pool temporanea di transazioni non-verificate, pool mantenuta da ogni singolo nodo. Man mano che I miner cercano di generare un nuovo blocco, aggiungono transazioni non verificate prese da questa pool, in un nuovo blocco e cercano di risolvere un problema molto difficile (conosciuto come proof-of-work) per provare la validità di questo nuovo blocco. Il processo del mining è spiegato in dettaglio nel Capitolo 10.

Le transazioni vengono aggiunte al

nuovo blocco, ordinate dando priorità alle fee più alte e poi per altri criteri. Ogni miner inizia il processo di mining di un nuovo blocco di transazioni nel momento in cui riceve il blocco precedente dal network, sapendo che egli ha perso l'ultimo round della competizione. Il miner crea immediatamente un nuovo blocco, lo riempie con transazioni e con le informazioni (l'hash) del blocco precedente, e inizia a calcolare la proof of work per il nuovo blocco. Ogni miner include una transazione speciale nel suo blocco, una che paga al proprio indirizzo bitcoin una ricompensa per i nuovi bitcoin creati (attualmente 12.5 BTC per blocco). Se

il miner trova una soluzione che rende il blocco valido, egli "vince" questa ricompensa perchè il suo blocco "vincente" è aggiunto alla blockchain globale e la transazione di ricompensa che lui ha incluso in esso diventa spendibile. Jing, il quale partecipa in una mining pool, ha impostato il suo software per creare nuovi blocchi che assegneranno la ricompensa all'indirizzo di una mining pool. A questo punto, una frazione della ricompensa sarà distribuita a Jing e agli altri miner in proporzione alla quantità di lavoro che hanno contribuito a fornire nell'ultimo round.

La transazione di Alice è stata raccolta dalla rete e inclusa nel gruppo di

transazioni non ancora verificate. Una volta validata dal software che si occupa del mini,g è stata inclusa in un nuovo blocco, chiamato *candidate block*, generato dalla mining pool di Jing. Tutti I miner della mining pool cominciano immediatamente a trovare una soluzione proof-of-work per il candidate block. Approssimativamente cinque minuti dopo che la transazione è stata trasmessa dal wallet di Alice, l'ASIC miner di Jing ha trovato una soluzione per il candidate block e l'ha annunciato al resto della rete. Non appena il resto dei miner validano il blocco vincente, cominciano nuovamente la gara per generare il blocco successivo.

Il blocco vincente di Jing entra quindi a far parte della blockchain come il blocco #277316, contenente 420 transazioni, inclusa quella di Alice. Il blocco contenente la transazione di Alice viene segnalato come "conferma" della stessa transazione.

TIP	È possibile visionare blocco che includ transazione di A (https://blockchain.info/bl height/277316).

Circa 19 minuti dopo, un nuovo blocco, il #277317, è confermato tramite il lavoro di mining di un altro miner. Visto che questo nuovo blocco

è basato sul blocco precedente (#277316) che conteneva la transazione di Alice, ha aggiunto una grande quantità di potenza computazionale su quello stesso blocco, rafforzando in questo modo la fiducia in quelle transazioni. Ogni blocco su cui è effettuato mining sopra di quello contenente la transazione è una conferma addizionale. Man mano che i blocchi si impilano uno sopra l'altro, diventa esponenzialmente più difficile invertire la transazione, in questo modo rendendo il network sempre più sicuro.

Nel diagramma della Figura 11 possiamo notare il blocco #277316, contenente la transazione di Alice. Al

di sotto di esso ci sono 277,316 blocchi (includenti il blocco #0), collegati l'uno all'altro in una catena di blocchi (blockchain) fino al blocco #0, conosciuto come il genesis block. Col passare del tempo, mano a mano che la "height" (altezza) dei blocchi aumenta, allo stesso modo aumenta anche la difficoltà computazionale per ogni blocco e per la catena nel complesso. I blocchi su cui è stato effettuato mining dopo quello che contiene la transazione di Alice, agiscono come garanzia ulteriore, man mano che si accumulano su maggior computazione in una catena sempre più lunga. Per convenzione, ogni blocco con più di sei conferme è considerato

irrevocabile, perchè si necessiterebbe di un'ammontare immenso di computazione per invalidarlo e ricalcolare sei blocchi. Esamineremo il processo di mining e le modalità in cui la fiducia viene costruita in maggiore dettaglio nel Capitolo 10.

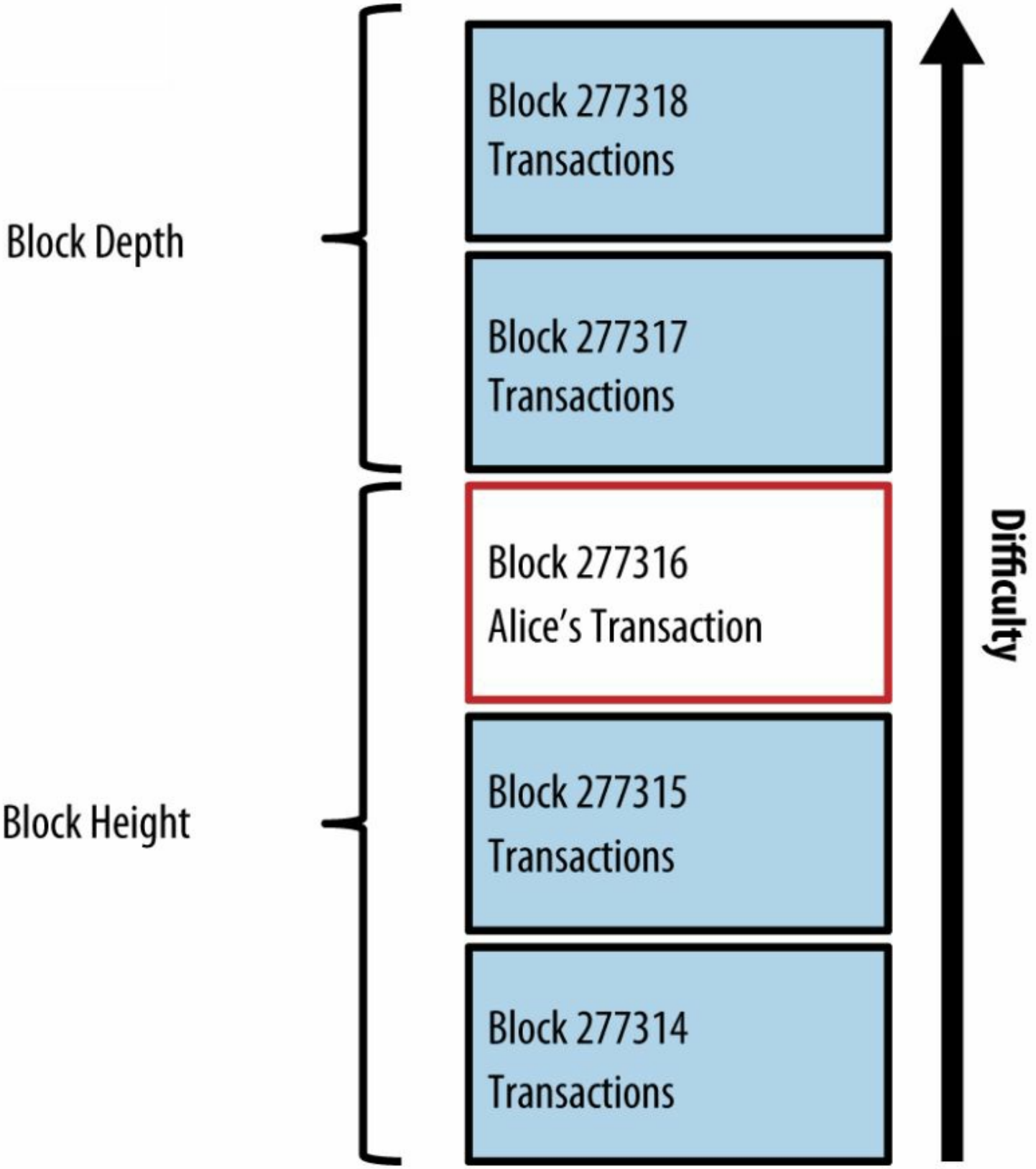


Figura 11. La transazione di Alice inclusa nel blocco

Spendere la Transazione

Adesso che la transazione di Alice è stata inclusa nella blockchain e contenuta in un blocco, è diventata parte del registro distribuito bitcoin e visibile a tutte le applicazioni bitcoin. Ogni client bitcoin può verificare indipendentemente la transazione come valida e spendibile. I client con indice completo (full-index o full-node) possono tracciare la provenienza dei fondi dal momento che i bitcoin sono stati generati un un blocco, incrementalmente da transazione a

transazione, fino a che non arrivano all'indirizzo di Bob. I light-client possono effettuare quella che viene chiamata una verifica di pagamento semplificata (vedi "Nodi di Simplified Payment Verification (SPV)") confermando che la transazione si trovi nella blockchain e abbia numerosi blocchi confermati dopo di essa, quindi provvedendo alla garanzia che il network la accetti come valida.

Bob a questo punto può spendere l'output da questa e da altre transazioni. Per esempio, Bob può pagare un cliente o fornitore trasferendo valore dal pagamento del caffè a questi nuovi proprietari. Più probabilmente, il software di bitcoin

di Bob aggregherà molti pagamenti piccoli in un pagamento più grande, probabilmente concentrando tutto il guadagno dei bitcoin del giorno in un'unica transazione. Questo muoverà i vari pagamenti verso un singolo output (ed un singolo indirizzo). Per vedere un diagramma di aggregazione di transazione, vedi la Figura 8.

Non appena Bob spende i pagamenti ricevuti da Alice e da altri clienti, egli estende la catena delle transazioni. Assumiamo che Bob paghi il suo web designer Gopesh a Bangalore per una nuova pagina web del suo sito. Adesso la catena di transazioni assomiglierà alla Figura 12.

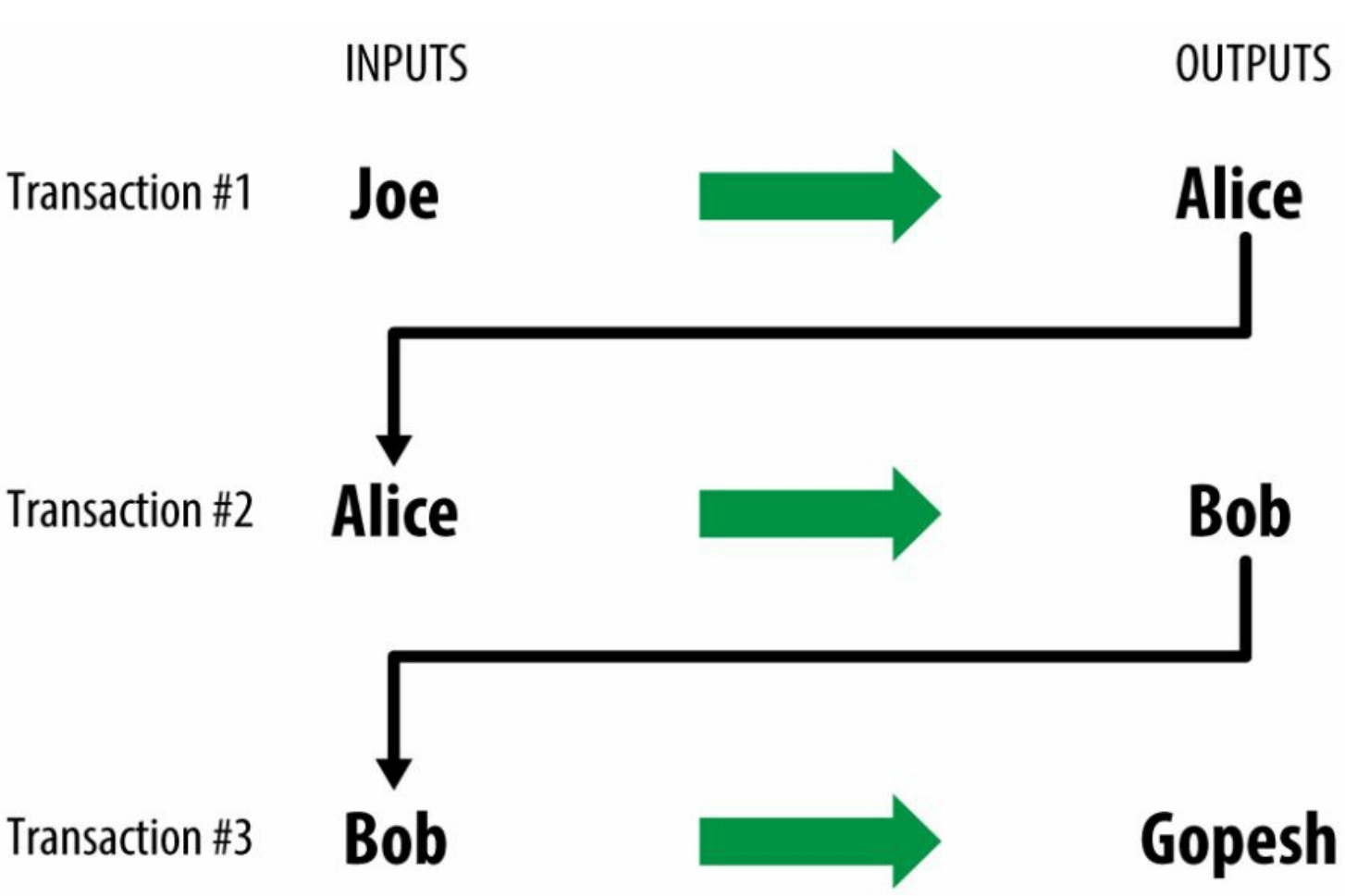


Figura 12. La transazione di Alice come parte di una catena di transazioni da Joe a Gopesh

In questo capitolo abbiamo visto come le transazioni creano una catena che sposta il valore da proprietario a

proprietario. Abbiamo tracciato anche la transazione di Alice, dal momento in cui è stata creata nel suo portafoglio, attraverso la rete bitcoin e ai minatori che l'hanno registrata sulla blockchain. Nel resto di questo libro esamineremo le tecnologie specifiche dietro i portafogli, gli indirizzi, le firme, le transazioni, la rete e infine l'estrazione (mining).

Bitcoin Core: L'Implementazione di Riferimento

Bitcoin è un progetto *open source* e il codice sorgente è disponibile con una licenza open (MIT), scaricabile ed utilizzabile liberamente per qualsiasi scopo. Open source significa più che semplicemente libero da usare. Significa anche che bitcoin è sviluppato da una comunità aperta di volontari. All'inizio, quella comunità consisteva solo in Satoshi Nakamoto. Entro il 2016, il codice sorgente di bitcoin aveva più di 400 contributori

con circa una dozzina di sviluppatori che lavoravano al codice quasi a tempo pieno e diverse decine di altri su base part-time. Chiunque può contribuire al codice, incluso te!

Quando bitcoin fu creato da Satoshi Nakamoto, il software fu effettivamente completato prima che il whitepaper riprodotto nell'Appendice A fosse scritto. Satoshi voleva assicurarsi che funzionasse prima di scriverne a riguardo. Quella prima implementazione, semplicemente nota come "Bitcoin" o "Satoshi client", è stata pesantemente modificata e migliorata. Si è evoluto in ciò che è noto come Bitcoin Core, per differenziarlo da altre implementazioni

compatibili. Bitcoin Core è l'*implementazione di riferimento* del sistema bitcoin, il che significa che è il riferimento autorevole su come ogni parte della tecnologia dovrebbe essere implementata. Bitcoin Core implementa tutti gli aspetti del bitcoin, compresi i wallet, un motore di validazione di blocchi e transazioni e un nodo di rete completo nella rete bitcoin peer-to-peer.

	Anche se Bi Core in un'implementa di riferimento portafoglio, quest'ultimo r destinato

ATTENZIONE

all’utilizzo
portafoglio pe
utenti o pe
applicazioni.
sviluppatori
applicazioni
invitati a cos
portafogli
utilizzando
standard mo
come BIP-3
BIP-32
“Mnemonic
Words BIP-3
“HD Wallets (
32/BIP-44).
sta per *Bi*
Improvement

	Proposal.

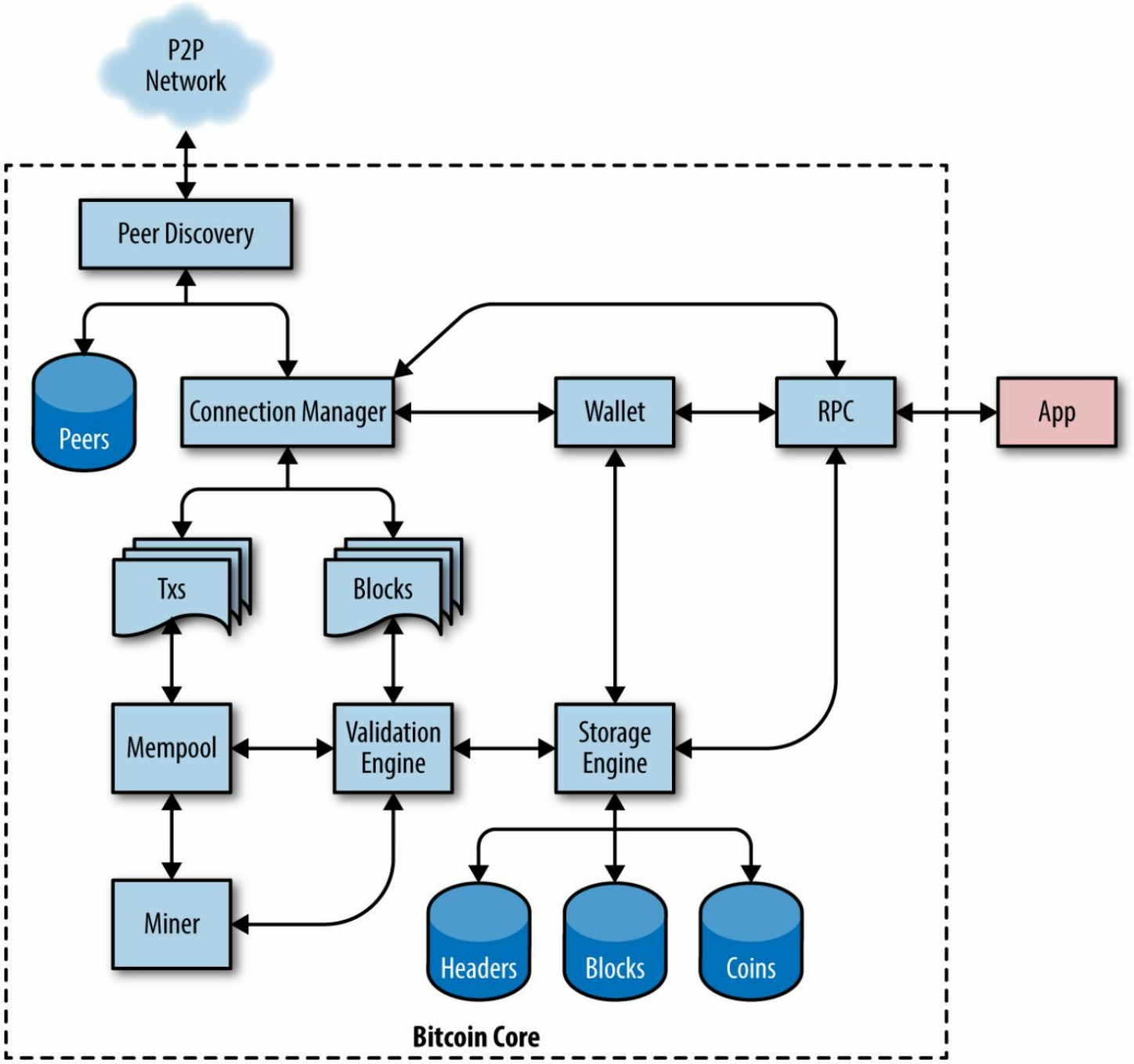


Figura 13. L'architettura di Bitcoin Core

Ambiente di sviluppo

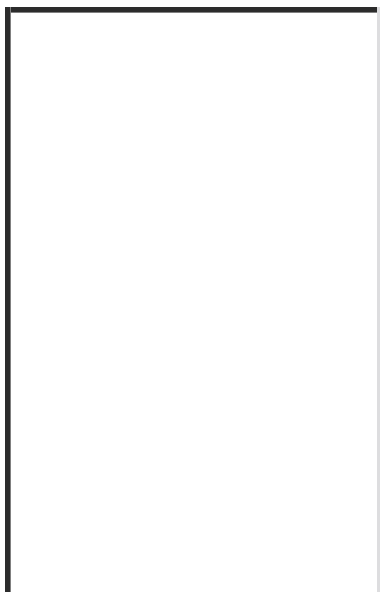
Bitcoin

Se sei uno sviluppatore, ti consiglio di configurare un ambiente di sviluppo con tutti gli strumenti, le librerie e il software di supporto per la scrittura di applicazioni bitcoin. In questo capitolo altamente tecnico, passeremo attraverso questo processo passo dopo passo. Se il materiale diventa troppo denso (e non stai effettivamente configurando un ambiente di sviluppo) sentiti libero di saltare al capitolo successivo, che è meno tecnico.

Compilare Bitcoin Core dal Codice

Sorgente

Il codice sorgente di Bitcoin Core può essere scaricato come un file ZIP o clonandone il sorgente principale dal repository presente su GitHub. Sulla pagina bitcoin di GitHub (https://github.com/bitcoin/bitcoin), seleziona Download ZIP dalla barra a destra. In alternativa, utilizza il comando git da terminale per creare una copia locale del codice sorgente sulla tua macchina



In molti degli esempi di questo capitolo useremo l'interfaccia della riga di comando del sistema operativo (conosciuta

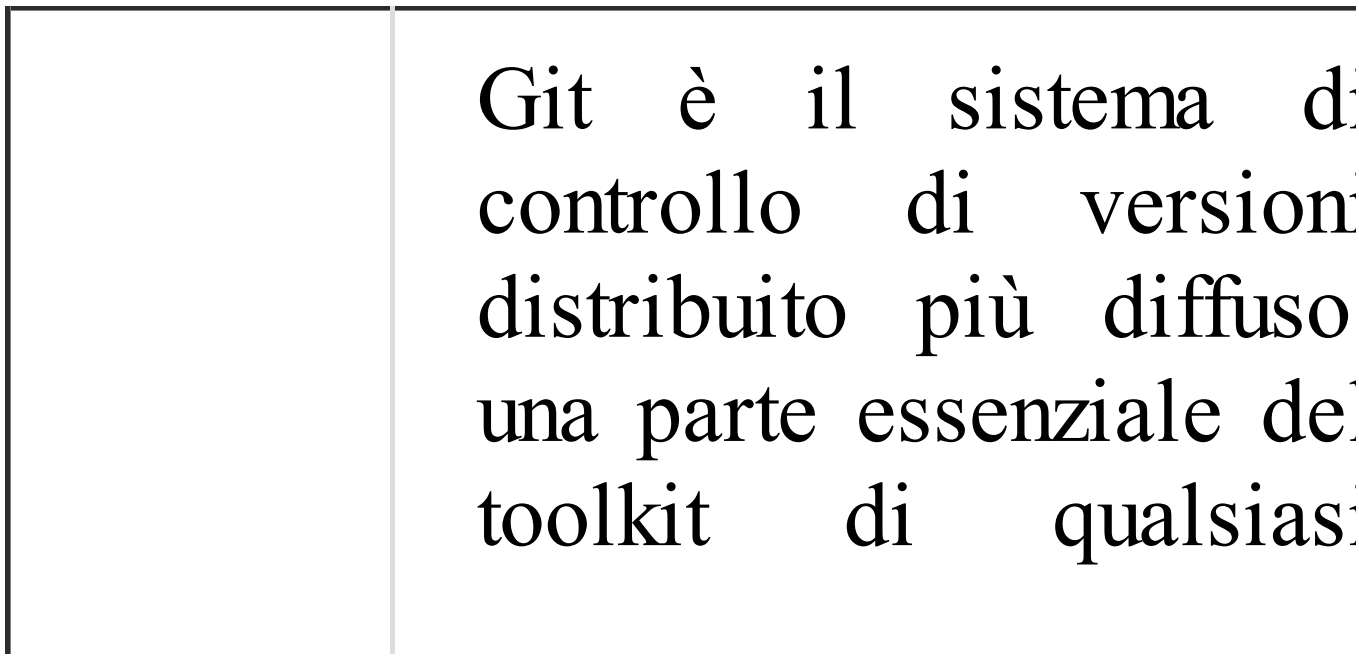
TIP	anche come "shell") accessibile tramite un'applicazione "terminale". La shell visualizzerà un prompt digiterai un comando; e la shell risponde con un po' di testo e una nuova richiesta per il tuo prossimo comando. Il prompt può apparire diverso sul tuo sistema ma nei seguenti esempi è denotato da un simbolo $. Negli esempi, quando vedi un testo dopo un $ non digitare il simbolo $ ma digita il comando

	immediatamente successivo, quindi prem Invio per eseguire i comando. Negli esempi le linee sotto ciascur comando sono le risposte del sistema operativo a quel comando. Quando vedi il prefisso $ successivo, saprai che è un nuovo comando e dovresti ripetere i processo.

In questo esempio, stiamo usando il comando git per creare una copia locale ("clone") del codice sorgente:

```
$ git clone
```

```
https://github.com/bitcoin/bitcoin.git
Cloning into 'bitcoin'...
remote: Counting objects: 102071, done.
remote: Compressing objects: 100%
(10/10), done.
Receiving objects: 100%
(102071/102071), 86.38 MiB | 730.00
KiB/s, done.
remote: Total 102071 (delta 4), reused 5
(delta 1), pack-reused 102060
Resolving deltas: 100% (76168/76168),
done.
Checking connectivity... done.
$
```



TIP	sviluppatore. Potrebbe essere necessario installare il comando git o un'interfaccia utente grafica per git sul tuo sistema operativo, se non lo hai già.

Non appena l'operazione del comando git clone sarà completata, avrai una copia locale completa del repository del codice nella directory bitcoin. Entra dentro questa directory digitando cd bitcoin nel prompt/terminale:

```
$ cd bitcoin
```

Selezionare una versione di Bitcoin Core

Di default, la copia locale verrà sincronizzata con il codice più recente, che potrebbe essere una versione beta o unstable di bitcoin. Prima di compilare il codice, seleziona una versione specifica facendo il check di un release tag. Questo farà sincronizzare la copia locale con la versione specifica del codice del repository identificato con un'etichetta detta tag. I tag sono usati dai developer per segnare release specifiche del codice per numero di versione. Per prima cosa, per trovare la lista di tutti i tag disponibili, utilizzeremo il comando git tag:

```
$ git tag
v0.1.5
```

```
v0.1.6test1
v0.10.0
...
v0.11.2
v0.11.2rc1
v0.12.0rc1
v0.12.0rc2
...
```

La lista dei tag mostra tutte le versioni di bitcoin rilasciate. Per convenzione, l e *release candidate*, che sono fatte per testing, hanno il suffisso "rc". Le release stabili (stable) che possono essere eseguite su sistemi in produzione non hanno suffisso. Dalla lista precedente, seleziona la release maggiore che al momento della scrittura di questo libro è la v0.11.2. Per sincronizzare il codice locale con

questa versione, utilizza il comando git checkout:

```
$ git checkout v0.15.0
HEAD is now at 3751912... Merge
#11295: doc: Old fee_estimates.dat are
discarded by 0.15.0
```

Puoi confermare di avere la versione desiderata digitando il comando git status:

```
$ git status
HEAD detached at v0.15.0
nothing to commit, working directory
clean
```

Configurazione della Build di Bitcoin Core

Il codice sorgente include la documentazione, che può essere

trovata in molti file. Controlla la documentazione principale che si trova n e l *README.md* nella directory *bitcoin* digitando **more README.md** nel prompt e usando la barra spaziatrice per procedere alla pagina successiva. In questo capitolo, compileremo il bitcoin client da riga di comando, anche conosciuto come bitcoind su Linux. Ricontrolla le istruzioni per compilare il client da riga di comando bitcoind per la tua distribuzione digitando **more doc/build-unix.md**. Istruzioni alternative per Mac OS X e Windows sono disponibili nella directory *doc*, rispettivamente in *build-osx.md* o *build-windows.md*.

Revisiona attentamente i requisiti di build, che sono nella prima parte della documentazione della build. Queste sono le librerie che devono essere presenti sul tuo sistema prima che tu possa iniziare a compilare bitcoin. Se questi prerequisiti non sono presenti, il processo di build fallirà con un errore. Se questo avviene perchè ti manca un prerequisito, puoi installarlo e poi continuare il processo di build da dove eri rimasto. Assumendo che i prerequisiti siano installati, inizierai il processo di build generando una serie di script di build usando lo script *autogen.sh*.

```
$ ./autogen.sh
...
```

```
glibtoolize: copying file 'build-
aux/m4/libtool.m4'
glibtoolize: copying file 'build-
aux/m4/ltoptions.m4'
glibtoolize: copying file 'build-
aux/m4/ltsugar.m4'
glibtoolize: copying file 'build-
aux/m4/ltversion.m4'
...
configure.ac:10: installing 'build-
aux/compile'
configure.ac:5: installing 'build-
aux/config.guess'
configure.ac:5: installing 'build-
aux/config.sub'
configure.ac:9: installing 'build-
aux/install-sh'
configure.ac:9: installing 'build-
aux/missing'
Makefile.am: installing 'build-
aux/depcomp'
```

```
...
```

Lo script *autogen.sh* crea una serie di script di configurazione automatici che interrogheranno il tuo sistema per scoprire i settaggi corretti e che assicureranno che tu abbia tutte le librerie necessarie per compilare il codice. La cosa più importante di queste è che lo script configure offre una serie di opzioni differenti per personalizzare il processo di build. Digita **./configure --help** per vedere tutte le opzioni disponibili:

```
$ ./configure --help
`configure' configures Bitcoin Core
0.15.0 to adapt to many kinds of systems.

Usage (Utilizzo): ./configure [OPTION]...
[VAR=VALUE]...
```

```
...
Funzionalità Opzionali:
  --disable-option-checking  ignore
unrecognized --enable/--with options
  --disable-FEATURE       do not include
FEATURE (same as --enable-
FEATURE=no) (non includere la
funzionalità FEATURE)
  --enable-FEATURE[=ARG]  include
FEATURE [ARG=yes]

  --enable-wallet         enable wallet
(default is yes)

  --with-gui[=no|qt4|qt5|auto]
...
```

Lo script configure permette di abilitare o disabilitare alcune funzioni di bitcoind attraverso l'uso dei flag --

enable-FEATURE e --disable-FEATURE, nei quali FEATURE è sostituita dal nome della funzionalità da abilitare/disabilitare, come listato nell'output del comando help. In questo capitolo, compileremo il client bitcoind con tutte le funzionalità di base. Non useremo i flag di configurazione, ma dovresti ricontrollarli per capire quali funzionalità opzionali fanno parte del client. Se ti trovi in un contesto accademico, le restrizioni del tuo laboratorio potrebbero richiedere l'installazione di applicazioni nella tua home directory (ad esempio, usando --prefix=$HOME).

Ecco alcune opzioni utili che

sovrascrivono il comportamento predefinito dello script configure:

--prefix=$HOME

Questo sovrascrive il percorso di installazione predefinito (che è /usr/local/) per il file eseguibile risultante. Usa $HOME per mettere tutto nella tua home directory o in un percorso diverso.

--disable-wallet

Questo è usato per disabilitare l'implementazione del wallet di riferimento.

--with-incompatible-bdb

Se stai creando un portafoglio, consenti l'uso di una versione

incompatibile della libreria Berkeley DB.

--with-gui=no

Non viene creata l'interfaccia grafica utente, che richiede la libreria Qt. Questo crea solo il server e la riga di comando bitcoin.

Successivamente, esegui il configure script per scoprire automaticamente tutte le librerie necessarie e creare uno script di build personalizzato per il tuo sistema:

```
$ ./configure
checking build system type... x86_64-
unknown-linux-gnu
```

```
checking host system type... x86_64-
unknown-linux-gnu
checking for a BSD-compatible install...
/usr/bin/install -c
checking whether build environment is
sane... yes
checking for a thread-safe mkdir -p...
/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)...
yes
...
[many pages of configuration tests follow]
...
$
```

Se tutto va bene, il comando configure terminerà creando gli script di build personalizzati che ci permetteranno di compilare bitcoind. Se qualche libreria è mancante o se ci sono errori,

il comando configure terminerà con un errore invece di creare gli script di build. Se un errore viene riportato, sarà probabilmente perchè ti manca una libreria o c'è una libreria non compatibile. Ricontrolla nuovamente la documentazione di build e sii sicuro di aver installato tutti i prerequisiti mancanti. A questo punto lancia configure nuovamente e ricontrolla se l'errore sia stato risolto.

Gli Executables di Bitcoin Core

Successivamente verrà compilato il codice sorgente, un processo che può richiedere fino ad un'ora per essere completato, a seconda della velocità

della CPU e della memoria disponibile. Durante il processo di compilazione dovresti vedere l'output ogni pochi secondi o ogni pochi minuti, o un errore se qualcosa va storto. Se si verifica un errore o il processo di compilazione viene interrotto, può essere ripreso in qualsiasi momento digitando nuovamente **make**. Digita make per avviare la compilazione dell'applicazione eseguibile:

```
$ make
Making all in src
  CXX      crypto/libbitcoinconsensus_la-
hmac_sha512.lo
  CXX      crypto/libbitcoinconsensus_la-
ripemd160.lo
  CXX      crypto/libbitcoinconsensus_la-
```

```
sha1.lo
  CXX      crypto/libbitcoinconsensus_la-
sha256.lo
  CXX      crypto/libbitcoinconsensus_la-
sha512.lo
  CXX      libbitcoinconsensus_la-hash.lo
  CXX
primitives/libbitcoinconsensus_la-
transaction.lo
  CXX      libbitcoinconsensus_la-
pubkey.lo
  CXX      script/libbitcoinconsensus_la-
bitcoinconsensus.lo
  CXX      script/libbitcoinconsensus_la-
interpreter.lo

[... seguono molti altri messaggi di
compilazione ...]

$
```

Su un sistema veloce con più di una

CPU, è possibile impostare il numero di processi di compilazione paralleli. Ad esempio, make -j 2 userà due core se disponibili. Se tutto va bene, Bitcoin Core è ora compilato. Dovresti eseguire l'unit test suite con make check per garantire che le librerie collegate funzionino. Il passo finale è installare i vari eseguibili sul tuo sistema usando il comando make install. Potrebbe essere richiesta la password dell'utente, poiché questo passaggio richiede privilegi amministrativi:

```
$ make check && sudo make install
Password:
Making install in src
 ../build-aux/install-sh -c -d '/usr/local/lib'
libtool: install: /usr/bin/install -c bitcoind
```

```
/usr/local/bin/bitcoind
libtool: install: /usr/bin/install -c bitcoin-
cli /usr/local/bin/bitcoin-cli
libtool: install: /usr/bin/install -c bitcoin-
tx /usr/local/bin/bitcoin-tx
...
$
```

L'installazione predefinita di bitcoind è */usr/local/bin*. Puoi confermare che Bitcoin Core è installato correttamente chiedendo al sistema il percorso degli eseguibili, come segue:

```
$ which bitcoind
/usr/local/bin/bitcoind

$ which bitcoin-cli
/usr/local/bin/bitcoin-cli
```

Esecuzione di un nodo

Bitcoin Core

La rete peer-to-peer di Bitcoin è composta da "nodi" della rete, gestiti principalmente da volontari e da alcune delle aziende che creano applicazioni bitcoin. Quelli che gestiscono i nodi bitcoin hanno una visione diretta e autorevole della blockchain di bitcoin, con una copia locale di tutte le transazioni, validata in modo indipendente dal proprio sistema. Tramite un proprio nodo, non è necessario affidarsi a terzi per convalidare una transazione. Inoltre, attraverso un nodo bitcoin si contribuisce alla rete di bitcoin rendendola più robusta.

L'esecuzione di un nodo, tuttavia, richiede un sistema connesso in modo permanente con risorse sufficienti per elaborare tutte le transazioni bitcoin. A seconda che si scelga di indicizzare tutte le transazioni e di mantenere una copia completa della blockchain, potrebbe anche essere necessario molto spazio su disco e RAM. A partire dall'inizio del 2018, un nodo dell'indice completo richiede 2 GB di RAM e almeno 160 GB di spazio su disco (consultare https://blockchain.info/charts/blocks-size). I nodi bitcoin trasmettono e ricevono anche transazioni e blocchi bitcoin, consumando larghezza di banda internet. Se la tua connessione

Internet è limitata, ha un limite di dati basso o viene limitata e addebitata, probabilmente non dovresti eseguire un nodo bitcoin o eseguirlo in un modo che ne limita la larghezza di banda (vedi Esempio di configurazione di un sistema a risorse limitate).

	Bitcoin Core conserva una copia completa della blockchain di default con ogni transazione che sia mai avvenuta sulla rete bitcoin sin dal suo inizio nel 2009. Questo set di dati ha una dimensione di dozzine di

TIP	gigabyte e viene scaricato in modo incrementale nell'arco di diversi giorni o settimane, a seconda della velocità della CPU e della connessione Internet. Bitcoin Core non sarà in grado di elaborare transazioni o aggiornare i saldi dei conti fino a quando non viene scaricato il set completo di dati della blockchain. Assicurati di avere abbastanza spazio su disco, larghezza di banda e tempo per

	completetare la sincronizzazione iniziale. Puoi configurare Bitcoin Core per ridurre la dimensione della blockchain scartando i vecchi blocchi (vedi Esempio di configurazione di un sistema con risorse limitate), ma dovrai comunque scaricare l'intero set di dati prima di scartare i dati.

Nonostante questi requisiti di risorse, migliaia di volontari gestiscono nodi bitcoin. Alcuni sono in esecuzione su

sistemi semplici come un Raspberry Pi (un computer da $ 35 dollari americani dalle dimensioni di un mazzo di carte). Molti volontari gestiscono anche i nodi bitcoin sui server noleggiati, di solito alcune varianti di Linux. È possibile utilizzare un'istanza *Virtual Private Server* (VPS) o *Cloud Computing Server* per eseguire un nodo bitcoin. Tali server possono essere affittati per $ 25 a $ 50 USD al mese da una varietà infinita di provider.

Perché dovresti eseguire un nodo? Ecco alcuni dei motivi più comuni:

- Se stai sviluppando software bitcoin e hai bisogno di affidarti a un nodo

bitcoin per l'accesso programmabile (API) alla rete e alla blockchain.

- Se stai creando applicazioni che devono convalidare le transazioni in base alle regole di consenso di bitcoin. In genere, le società di software bitcoin eseguono diversi nodi.
- Se vuoi supportare bitcoin. L'esecuzione di un nodo rende la rete più solida e in grado di servire più portafogli, più utenti e più transazioni.

- Se non vuoi affidarti a terzi per elaborare o convalidare le tue transazioni.

Se stai leggendo questo libro e sei interessato allo sviluppo di software bitcoin, dovresti eseguire il tuo nodo.

Configurazione del nodo Bitcoin Core

Bitcoin Core cercherà un file di configurazione nella sua directory dei dati ad ogni avvio. In questa sezione esamineremo le varie opzioni di configurazione e configureremo un file di configurazione. Per individuare il file di configurazione, esegui *bitcoind -printtoconsole* nel terminale e fai

attenzione alle prime righe.

```
$ bitcoind -printtoconsole
Bitcoin version v0.15.0
Using the 'standard' SHA256
implementation
Using data directory
/home/ubuntu/.bitcoin/
Using config file
/home/ubuntu/.bitcoin/bitcoin.conf
...
[a lot more debug output]
...
```

Puoi premere Ctrl-C per arrestare il nodo dopo aver determinato la posizione del file di configurazione. Di solito il file di configurazione si trova nella directory dei dati *.bitcoin* sotto la home directory dell'utente. Apri il file di configurazione

attraverso il tuo editor preferito.

Bitcoin Core offre oltre 100 opzioni di configurazione che modificano il comportamento del nodo, l'archiviazione della blockchain e molti altri aspetti del suo funzionamento. Per vedere un elenco di queste opzioni, esegui bitcoind –help:

```
$ bitcoind --help
Bitcoin Core Daemon version v0.15.0

Usage:
  bitcoind [options]                     Start
Bitcoin Core Daemon

Options:

  -?
```

```
      Print this help message and exit

  -version
      Print version and exit

  -alertnotify=<cmd>
      Execute command when a relevant
alert is received or we see a really
      long fork (%s in cmd is replaced by
message)
...
[many more options]
...

  -rpcthreads=<n>
      Set the number of threads to service
RPC calls (default: 4)
```

Ecco alcune delle opzioni più importanti che è possibile impostare nel file di configurazione o come

parametri della riga di comando per bitcoind:

alertnotify

Esegui un comando o uno script specificato per inviare avvisi di emergenza al proprietario di questo nodo, solitamente via email.

conf

Un percorso alternativo per il file di configurazione. Questo ha senso solo come parametro della riga di comando per bitcoind, in quanto non può trovarsi all'interno del file di configurazione a cui fa riferimento.

datadir

Seleziona la directory e il filesystem

in cui inserire tutti i dati blockchain. Di default questa è la sottodirectory *.bitcoin* della tua home directory. Assicurati che questo filesystem abbia diversi gigabyte di spazio libero.

prune

Riduci i requisiti di spazio su disco di molti gigabyte, eliminando i vecchi blocchi. Utilizzalo su un nodo con risorse limitate che può affidarsi alla blockchain completa.

txindex

Mantiene un indice di tutte le transazioni. Ciò significa una copia completa della blockchain che consente di recuperare a livello di

codice qualsiasi transazione per ID.

dbcache

La dimensione della cache UTXO. L'impostazione predefinita è 300 MiB. Aumentala sugli hardware di fascia alta e riducila sugli hardware di fascia bassa per risparmiare memoria a spese di una lettura del disco più lenta.

maxconnections

Imposta il numero massimo di nodi da cui accettare le connessioni. La riduzione di questa impostazione predefinita ridurrà il consumo della larghezza di banda. Utilizza questa opzione se hai un limite di dati o la tua connesione viene addebitata a

gigabyte.

maxmempool

Limita il pool di memoria delle transazioni di diversi megabyte. Usalo per ridurre l'uso della memoria su nodi con limiti di memoria.

maxreceivebuffer/maxsendb

Limita il buffer di memoria per connessione. Utilizzalo su nodi con limiti di memoria.

minrelaytxfee

Imposta la tariffa minima per le transazione che inoltrerai. Al di sotto di questo valore, la transazione

viene considerata non standard, rifiutata dal pool di transazioni e non inoltrata.

Indice del Database delle Transazioni e Opzione txindex

Per impostazione predefinita, Bitcoin Core crea un database contenente *solo* le transazioni correlate al portafoglio dell'utente. Se vuoi essere in grado di accedere a *qualsiasi* transazione con comandi come getrawtransaction (vedi

Esplorando e Decodificando le Transazioni), è necessario configurare Bitcoin Core per creare un indice di transazione completo, che può essere ottenuto con l'opzione txindex. Imposta txindex = 1 nel file di configurazione di Bitcoin Core. Se non si imposta questa opzione inizialmente e successivamente si imposta l'indicizzazione completa, sarà necessario riavviare bitcoind con l'opzione -reindex e attendere che venga ricostruito l'indice.

La configurazione di esempio di un nodo full-index mostra come è possibile combinare le opzioni precedenti, con un nodo

completamente indicizzato, in esecuzione come backend API per un'applicazione bitcoin.

Esempio 2. Configurazione esempio di un nodo full-index

```
alertnotify=myemailscript.sh "Alert: %s"

datadir=/lotsofspace/bitcoin
txindex=1
```

La configurazione di esempio di un sistema con risorse limitate mostra un nodo con risorse limitate in esecuzione su un server più piccolo.

Esempio 3. Configurazione

esempio di un sistema con risorse limitate

```
alertnotify=myemailscript.sh "Alert: %s"
```

```
maxconnections=15
prune=5000
dbcache=150
maxmempool=150
maxreceivebuffer=2500
maxsendbuffer=500
```

Una volta modificato il file di configurazione e impostato le opzioni che meglio rappresentano le tue esigenze, puoi testare bitcoind con questa configurazione. Esegui Bitcoin Core con l'opzione printtoconsole da

eseguire in primo piano con l'output sulla console:

```
$ bitcoind -printtoconsole

Bitcoin version v0.15.0
InitParameterInteraction: parameter
interaction: -whitelistforcerelay=1 ->
setting -whitelistrelay=1
Assuming ancestors of block
000000000000000003b9ce759c2a087d52
have valid signatures.
Using the 'standard' SHA256
implementation
Default data directory
/home/ubuntu/.bitcoin
Using data directory /lotsofspace/.bitcoin
Using config file
/home/ubuntu/.bitcoin/bitcoin.conf
Using at most 125 automatic connections
(1048576 file descriptors available)
```

Using 16 MiB out of 32/2 requested for signature cache, able to store 524288 elements
Using 16 MiB out of 32/2 requested for script execution cache, able to store 524288 elements
Using 2 threads for script verification
HTTP: creating work queue of depth 16
No rpcpassword set - using random cookie authentication
Generated RPC authentication cookie /lotsofspace/.bitcoin/.cookie
HTTP: starting 4 worker threads
init message: Verifying wallet(s)...
Using BerkeleyDB version Berkeley DB 4.8.30: (April 9, 2010)
Using wallet wallet.dat
CDBEnv::Open: LogDir=/lotsofspace/.bitcoin/database ErrorFile=/lotsofspace/.bitcoin/db.log
scheduler thread start

```
Cache configuration:
* Using 250.0MiB for block index
database
* Using 8.0MiB for chain state database
* Using 1742.0MiB for in-memory UTXO
set (plus up to 286.1MiB of unused
mempool space)
init message: Loading block index...
Opening LevelDB in
/lotsofspace/.bitcoin/blocks/index
Opened LevelDB successfully

[... altri messaggi d'avvio ...]
```

Puoi premere Ctrl-C per interrompere il processo una volta che sei sicuro che stia caricando le impostazioni corrette e stia funzionando come previsto.

Per eseguire Bitcoin Core in

background come processo, avvialo con l'opzione daemon, attraverso bitcoind -daemon.

Per monitorare lo stato di avanzamento e di runtime del tuo nodo bitcoin, usa il comando bitcoin-cli getblockchaininfo:

```
$ bitcoin-cli getblockchaininfo
```

```
{
  "chain": "main",
  "blocks": 0,
  "headers": 83999,
  "bestblockhash":
"000000000019d6689c085ae165831e934ff
  "difficulty": 1,
  "mediantime": 1231006505,
  "verificationprogress":
3.783041623201835e-09,
  "chainwork":
```

```
"0000000000000000000000000000000(
  "pruned": false,
  [...]
}
```

Questo mostra un nodo con un'altezza blockchain di 0 blocchi e 83999 intestazioni. Il nodo attualmente recupera le intestazioni di blocco della catena migliore e in seguito continua a scaricare i blocchi completi.

Una volta che sei soddisfatto delle opzioni di configurazione che hai selezionato, dovresti aggiungere bitcoin agli script di avvio nel tuo sistema operativo, in modo che funzioni continuamente e si riavvii al riavvio del sistema operativo. Troverai una serie di script di avvio di

esempio per vari sistemi operativi nella directory dei sorgenti di bitcoin sotto *contrib/init* e un file *README.md* che mostra quale sistema usa quale script.

Bitcoin Core Application Programming Interface (API)

Il Core Client Bitcoin implementa un'interfaccia JSON-RPC a cui si può accedere anche utilizzando il programma da riga di comando bitcoin-cli. La riga di comando (detta anche terminale o shell) ci permette di

sperimentare interattivamente con tutte le potenzialità che sono anche disponibili attraverso l'API. Per cominciare, invoca il comando help per visualizzare una lista dei comandi bitcoin RPC disponibili:

```
$ bitcoin-cli help
addmultisigaddress nrequired ["key",...] (
"account" )
addnode "node" "add|remove|onetry"
backupwallet "destination"
createmultisig nrequired ["key",...]
createrawtransaction
[{"txid":"id","vout":n},...]
{"address":amount,...}
decoderawtransaction "hexstring"
...
...
verifymessage "bitcoinaddress" "signature"
"message"
```

```
walletlock
walletpassphrase "passphrase" timeout
walletpassphrasechange "oldpassphrase"
"newpassphrase"
```

Ognuno di questi comandi può richiedere un numero di parametri. Per ottenere ulteriore aiuto, una descrizione dettagliata e informazioni sui parametri, aggiungi il nome del comando dopo help. Ad esempio, per visualizzare la guida sul comando RPC getblockhash:

```
$ bitcoin-cli help getblockhash
getblockhash height

Returns hash of block in best-block-chain
at height provided.

Arguments:
```

```
1. height         (numeric, required) The
height index

Result:
"hash"         (string) The block hash

Examples:
> bitcoin-cli getblockhash 1000
> curl --user myusername --data-binary
'{"jsonrpc": "1.0", "id":"curltest", "method":
"getblockhash", "params": [1000] }' -H
'content-type: text/plain;'
http://127.0.0.1:8332/
```

Alla fine delle informazioni di aiuto vedrai due esempi del comando RPC, usando l'helper bitcoin-cli o il client HTTP curl. Questi esempi mostrano come chiamare il comando. Copia il primo esempio e vedi il risultato:

```
$ bitcoin-cli getblockhash 1000
00000000c937983704a73af28acdec37b049
```

Il risultato è l'hash di un blocco, descritto in maggior dettaglio nei seguenti capitoli. Ma per ora, questo comando dovrebbe restituire lo stesso risultato sul tuo sistema, dimostrando che il tuo nodo Bitcoin Core è in esecuzione, che sta accettando comandi e che ha informazioni sul blocco 1000 da darti.

Nelle prossime sezioni mostreremo alcuni comandi RPC molto utili e il loro output.

Ottenere Informazioni sullo Stato del Client Bitcoin Core

Comando: getinfo

Il comando RPC di Bitcoin getinfo mostra informazioni base riguardo lo stato del nodo bitcoin del network , del wallet, e del database blockchain. Utilizza bitcoin-cli per eseguirlo:

```
$ bitcoin-cli getinfo
{
 "version": 110200,
"protocolversion": 70002,
 "blocks": 396367,
 "timeoffset": 0,
 "connections": 15,
 "proxy": "",
 "difficulty": 120033340651.23696899,
 "testnet": false,
```

```
  "relayfee" : 0.00010000,
  "errors" : ""
}
```


I dati sono ritornati in JavaScript Object Notation (JSON), un formato che può facilmente essere compreso da tutti i linguaggi di programmazione, ma che è anche abbastanza leggibile dall'utente. Tra questi dati vediamo il numero di versione del client software bitcoin (110200) e il protocollo bitcoin (70002). Vediamo il numero corrente dell'altezza del blocco, che ci mostra quanti blocchi sono conosciuti da questo client (396367). Vediamo anche varie informazioni sulla rete bitcoin e le impostazioni relative a questo client.

TIP	Ci vorrà un po' di tempo probabilmente più di un giorno, prima che bitcoind "raggiunga" l'altezza della blockchain (block height) attuale man manc che scarica i blocchi da altri client bitcoin. Puoi controllare il progresso utilizzando getinfo per vedere il numero di blocchi conosciuti.

Esplorare e Decodificare le Transazioni

Comandi: getrawtransaction, decoderawtransaction

Alice ha comprato una tazza di caffè da Bob's Cafe. La sua transazione è stata registrata sulla blockchain con ID transazione (txid) 0627052b6f28912f2703066a912ea577f
Usiamo l'API per recuperare ed esaminare quella transazione passando l'ID della transazione come parametro:

```
$ bitcoin-cli getrawtransaction
0627052b6f28912f2703066a912ea577f2ce4d
5fbd8a57286c345c2f2

0100000001186f9f998a5aa6f048e51dd8419a
000008b4830450221008 84d142d86652a3f47
ae24cb02204b9f039ff08df09cbe9f6addac960
```

```
10484ecc0d46f1918b30928fa0e4ed99f16a0fb
89d172787ec3457eee41c04f4938de5cc17b4a
0001976a914ab68025513c3dbd2f7b92a94e05
147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc02
```

TIP	Gli ID di transazione non sono autoritativi fino a che una transazione non è confermata. L'assenza di un hash di transazione nella blockchain non significa che la transazione non sia stata processata. Questo è conosciuto come "transaction malleability,"

	malleabilità di transazione, perchè gli hash di transazione possono essere modificate prima di essere confermati in un blocco. Dopo essere stati confermati, i txid sono immutabili e autoritativi.

Il comando getrawtransaction restituisce una transazione serializzata in notazione esadecimale. Per decodificarlo, usiamo il comando decoderawtransaction, passando i dati esadecimali come parametro. Puoi copiare l'hex restituito da

getrawtransaction e incollarla come parametro su decoderawtransaction:

```
$ bitcoin-cli decoderawtransaction
0100000001186f9f998a5aa6f048e51dd8419a
a0f1a8a2836dd734d2804fe65fa35779000000
6ec719bbfbd040a570b1deccbb6498c75c4ae24
cad530a863ea8f53982c09db8f6e3813014104
e0735e7ade8416ab9fe423cc5412336376789d
336a8d752adfffffffff0260e3160000000019
d50f654e788acd0ef8000000000001976a9147
88ac00000000

{
  "txid":
"0627052b6f28912f2703066a912ea577f2ce4d
  "size": 258,
  "version": 1,
```

```
 "locktime": 0,
 "vin": [
   {
    "txid":
"7957a35fe64f80d234d76d83a2...8149a41d81
    "vout": 0,
    "scriptSig": {

"asm":"3045022100884d142d86652a3f47ba47

"hex":"483045022100884d142d86652a3f47ba
    },
    "sequence": 4294967295
   }
 ],
 "vout": [
   {
    "value": 0.01500000,
```

```
    "n": 0,
    "scriptPubKey": {
      "asm": "OP_DUP OP_HASH160
ab68...5f654e7 OP_EQUALVERIFY
OP_CHECKSIG",
      "hex":
"76a914ab68025513c3dbd2f7b92a94e0581f5c
      "reqSigs": 1,
      "type": "pubkeyhash",
      "addresses": [

"1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA
      ]
    }
  },
  {
    "value": 0.08450000,
    "n": 1,
```

```
    "scriptPubKey": {
      "asm": "OP_DUP OP_HASH160
7f9b1a...025a8 OP_EQUALVERIFY
OP_CHECKSIG",
      "hex":
"76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3
      "reqSigs": 1,
      "type": "pubkeyhash",
      "addresses": [

"1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK"
      ]
    }
  }
 ]
}
```

La decodifica della transazione mostra tutte le parti di questa transazione, inclusi gli input e gli output. In questo caso possiamo notare che la transazione che ha accreditato il nostro indirizzo con 15 millibit ha usato un input e ha generato due output. L'input di questa transazione era l'output di una transazione precedentemente confermata (mostrato come "vin txid" che inizia con 7957a35fe). I due output corrispondono al valore di 15 millibit e ad un output contenente il resto reinviato al mittente.

Possiamo esplorare ulteriormente la blockchain esaminando la transazione precedente referenziata dal suo txid in questa transazione usando lo stesso

comando (i.e., getrawtransaction). Saltando da transazione a transazione possiamo seguire una catena di transazioni precedenti fino a che i bitcoin non siano stati trasmessi da un indirizzo di un proprietario a un altro.

Esplorare i Blocchi

Comandi: getblock, getblockhash

Esplorare i blocchi è simile ad esplorare le transazioni. Tuttavia, i blocchi possono essere referenziati dalla block height o dal block hash. Per prima cosa, troviamo un blocco in base all'altezza. Abbiamo visto che la transazione di Alice era inclusa nel blocco 277316.

Usiamo il comando getblockhash, che

prende l'altezza del blocco come parametro e restituisce l'hash del blocco per quel blocco:

```
$ bitcoin-cli getblockhash 277316
000000000000001b6b9a13b095e96db41c4
```

Ora che sappiamo in quale blocco è stata inclusa la transazione di Alice, possiamo interrogare quel blocco. Usiamo il comando getblock con l'hash del blocco come parametro:

```
$ bitcoin-cli getblock
000000000000001b6b9a13b095e96db4
1b2cc7bdc4
{
  "hash":
"000000000000001b6b9a13b095e96db
```

```
  "confirmations": 37371,
  "size": 218629,
  "height": 277316,
  "version": 2,
  "merkleroot":
"c91c008c26e50763e9f548bb8b2fc3237
  "tx": [

"d5ada064c6417ca25c4308bd158c34b7

"b268b45c59b39d759614757718b9918c

"04905ff987ddd4cfe603b03cfb7ca50ee8

"32467aab5d04f51940075055c2f20bbd1
```

"561c5216944e21fa29dd12aaa1a45e339

[... hundreds of transactions ...]

"78b300b2a1d2d9449b58db7bc71c3884

"6c87130ec283ab4c2c493b190c20de4b2

"6f423dbc3636ef193fd8898dfdf7621dca

"802ba8b2adabc5796a9471f25b02ae6ae

"eaaf6a048588d9ad4d1c092539bd571dc

"e67abc6bd5e2cac169821afc51b207127

"d38985a6a1bfd35037cb7776b2dc8679

```
"45ea0a3f6016d2bb90ab92c34a7aac976

"c098445d748ced5f178ef2ff96f2758cbe
  ],
  "time": 1388185914,
  "mediantime": 1388183675,
  "nonce": 924591752,
  "bits": "1903a30c",
  "difficulty": 1180923195.258026,
  "chainwork":
"0000000000000000000000000000000
  "previousblockhash":
"00000000000002a7bbd25a417c0374
  "nextblockhash":
"00000000000000010236c269dd6ed71
```

```
}
```

Il blocco contiene 419 transazioni e la 64esima transazione listata (0627052b..) è il pagamento di Alice. La voce height ci dice che questo è il 277316esimo blocco della blockchain.

Utilizzo dell'interfaccia programmatica di Bitcoin Core

L'helper bitcoin-cli è molto utile per esplorare l'API di Bitcoin Core e le funzioni di test. Ma il punto centrale di un'interfaccia di programmazione dell'applicazione è l'accesso programmatico alle funzioni. In questa

sezione dimostreremo come accedere a Bitcoin Core da un altro programma.

L'API di Bitcoin Core è un'interfaccia JSON-RPC. JSON sta per JavaScript Object Notation ed è un modo molto conveniente per presentare i dati che sia gli umani che i programmi possono facilmente leggere. RPC sta per Remote Procedure Call, il che significa che stiamo chiamando procedure (funzioni) remote (sul nodo Bitcoin Core) tramite un protocollo di rete. In questo caso, il protocollo di rete è HTTP o HTTPS (per connessioni crittografate).

Quando abbiamo usato il comando bitcoin-cli per ottenere aiuto su un comando, ci ha mostrato un esempio

dell'utilizzo di curl, il versatile client HTTP della riga di comando per costruire una di queste chiamate JSON-RPC:

```
$ curl --user myusername --data-binary
'{"jsonrpc": "1.0", "id":"curltest", "method":
"getblockchaininfo", "params": [] }' -H
'content-type: text/plain;'
http://127.0.0.1:8332/
```

Questo comando mostra che curl invia una richiesta HTTP all'host locale (127.0.0.1), connettendosi alla porta bitcoin predefinita (8332) e inviando una richiesta jsonrpc per il metodo getinfo usando codifica text/plain.

Se stai implementando una chiamata JSON-RPC nel tuo programma, puoi utilizzare una libreria HTTP generica

per costruire la chiamata, in modo simile a quanto mostrato nell'esempio curl precedente.

Tuttavia, nella maggior parte dei linguaggi di programmazione ci sono librerie che "avvolgono" l'API di Bitcoin Core in un modo che rende molto più semplice tutto ciò. Utilizzeremo la libreria python-bitcoinlib per semplificare l'accesso all'API. Ricorda che questo richiede di avere un'istanza di Bitcoin Core in esecuzione, che verrà utilizzata per effettuare chiamate JSON-RPC.

Lo script Python nell'Esempio 4 effettua una semplice chiamata getinfo e stampa il parametro block dai dati restituiti da Bitcoin Core.

Esempio 4. Esecuzione di getinfo tramite l'API JSON-RPC di Bitcoin Core

```
from bitcoin.rpc import RawProxy
```

```
# Create a connection to local Bitcoin
Core node
p = RawProxy()

# Run the getblockchaininfo command,
store the resulting data in info
info = p.getblockchaininfo()

# Retrieve the 'blocks' element from
the info
print(info['blocks'])
```

L'esecuzione ci dà il seguente

risultato:

```
$ python rpc_example.py
394075
```

Ci dice che il nostro nodo locale Bitcoin Core ha 394075 blocchi nella sua blockchain. Non è un risultato spettacolare, ma dimostra l'uso di base della libreria come interfaccia semplificata per l'API JSON-RPC di Bitcoin Core.

Successivamente, utilizziamo le chiamate getrawtransaction e decodetransaction per recuperare i dettagli del pagamento del caffè di Alice. Nell'Esempio 5, recuperiamo la transazione di Alice ed elenchiamo gli output della transazione. Per ogni output, mostriamo l'indirizzo e il

valore del destinatario. Come promemoria, la transazione di Alice aveva un output per pagare Bob's Cafe e un output per il resto ad Alice.

Esempio 5. Recupero di una transazione e iterazione dei suoi output

```
from bitcoin.rpc import RawProxy

p = RawProxy()

# Alice's transaction ID
txid =
"0627052b6f28912f2703066a912ea577f2

# First, retrieve the raw transaction in
```

```
hex
raw_tx = p.getrawtransaction(txid)

# Decode the transaction hex into a
JSON object
decoded_tx =
p.decoderawtransaction(raw_tx)

# Retrieve each of the outputs from the
transaction
for output in decoded_tx['vout']:
    print(output['scriptPubKey']
['addresses'], output['value'])
```

Eseguendo questo codice, otteniamo:

```
$ python rpc_transaction.py
([u'1GdK9UzpHBzqzX2A9JFP3Di4weBwqgr
Decimal('0.01500000'))
([u'1Cdid9KFAaatwczBwBttQcwXYCpvK8h7
Decimal('0.08450000'))
```

Entrambi gli esempi precedenti sono

piuttosto semplici. Non hai davvero bisogno di un programma per eseguirli; potresti altrettanto facilmente usare l'helper bitcoin-cli. Il prossimo esempio, tuttavia, richiede diverse centinaia di chiamate RPC e dimostra in modo più chiaro l'utilizzo di un'interfaccia programmatica.

Nell'Esempio 6, prima recuperiamo il blocco 277316, quindi recuperiamo ciascuna delle 419 transazioni all'interno per riferimento a ciascun ID transazione. Successivamente, iteriamo attraverso ciascuno degli output della transazione e sommiamo il valore.

Esempio 6. Recupero di un blocco e aggiunta di tutti gli

output di transazione

```
from bitcoin.rpc import RawProxy

p = RawProxy()

# The block height where Alice's
transaction was recorded
blockheight = 277316

# Get the block hash of block with
height 277316
blockhash =
p.getblockhash(blockheight)

# Retrieve the block by its hash
block = p.getblock(blockhash)

# Element tx contains the list of all
```

```
transaction IDs in the block
transactions = block['tx']

block_value = 0

# Iterate through each transaction ID in
the block
for txid in transactions:
    tx_value = 0
    # Retrieve the raw transaction by ID
    raw_tx = p.getrawtransaction(txid)
    # Decode the transaction
    decoded_tx =
p.decoderawtransaction(raw_tx)
    # Iterate through each output in the
transaction
    for output in decoded_tx['vout']:
        # Add up the value of each output
        tx_value = tx_value +
output['value']
```

```
    # Add the value of this transaction to
the total
    block_value = block_value +
tx_value

print("Total value in block: ",
block_value)
```

Eseguendo questo codice, otteniamo:

```
$ python rpc_block.py

('Total value in block: ',
Decimal('10322.07722534'))
```

Il nostro codice di esempio calcola che il valore totale transato in questo blocco è 10.322.07722534 BTC (inclusi il reward di 25 BTC e 0.0909 BTC in tasse). Confrontalo con l'importo riportato da un sito di Block Explorer cercando l'hash o l'altezza

del blocco. Alcuni explorer di blocchi riportano il valore totale escludendo la ricompensa ed escludendo le tasse. Vedi se riesci a individuare la differenza.

Client Alternativi, Librerie, e Toolkits

Esistono molti client alternativi, librerie, toolkit e persino implementazioni di nodi completi nell'ecosistema bitcoin. Queste sono implementate in una varietà di linguaggi di programmazione, offrendo ai programmatori interfacce native per il loro linguaggio.

Le sezioni seguenti elencano alcune

delle migliori librerie, client e toolkit, organizzati dai linguaggi di programmazione.

C/C++

Bitcoin Core
(https://github.com/bitcoin/bitcoin)
L'implementazione di riferimento di bitcoin

libbitcoin
(https://github.com/libbitcoin/libbitcoin)
Toolkit di sviluppo C ++ multipiattaforma, nodo e libreria di consenso

bitcoin explorer
(https://github.com/libbitcoin/libbitcoin-explorer)
Lo strumento da riga di comando di

Libbitcoin

picocoin
(https://github.com/jgarzik/picocoin)
Un client leggero in C per Bitcoin, scritto da Jeff Garzik

JavaScript

bcoin (http://bcoin.io/)
Un'implementazione di nodo completo modulare e scalabile con API

Bitcore (https://bitcore.io/)
Nodo completo, API e libreria di Bitpay

BitcoinJS
(https://github.com/bitcoinjs/bitcoinjs-lib)

Una pura libreria Bitcoin JavaScript per node.js e browser

Java

bitcoinj (https://bitcoinj.github.io)

Una libreria client Java full-node

Bits of Proof (BOP) (https://bitsofproof.com)

Un'implementazione Java di classe enterprise di bitcoin

PHP

bitwasp/bitcoin (https://github.com/bit-wasp/bitcoin-php)

Una libreria di bitcoin PHP e progetti correlati

Python

python-bitcoinlib
(https://github.com/petertodd/python-bitcoinlib)

Una libreria bitcoin Python, una libreria di consenso e un nodo di Peter Todd

pycoin
(https://github.com/richardkiss/pycoin)

Una libreria bitcoin Python di Richard Kiss

pybitcointools
(https://github.com/vbuterin/pybitcointo

Una libreria bitcoin Python di Vitalik Buterin

Ruby

bitcoin-client
(https://github.com/sinisterchipmunk/bitc
client)

Un wrapper di libreria Ruby per l'API JSON-RPC

Go

btcd (https://github.com/btcsuite/btcd)

Un client bitcoin full node in Go

Rust

rust-bitcoin
(https://github.com/apoelstra/rust-bitcoin)

Libreria di Rust per bitcoin per serializzazione, analisi e chiamate API

C#

NBitcoin
(https://github.com/MetacoSA/NBitcoin
Libreria di bitcoin completa per il framework .NET

Objective-C

CoreBitcoin
(https://github.com/oleganza/CoreBitcoi
Bitcoin toolkit per ObjC e Swift

Esistono molte altre librerie e client in una varietà di altri linguaggi di programmazione.

Chiavi, Indirizzi

Potresti aver sentito che il bitcoin è basato su *crittografia*, che è una branca della matematica ampiamente usata in sicurezza informatica. Crittografia significa "scrittura segreta" in greco, ma la scienza della crittografia comprende più della semplice scrittura segreta, che viene definita codifica. La crittografia può anche essere utilizzata per dimostrare la conoscenza di un segreto senza doverlo rivelare (firma digitale), o provare l'autenticità dei dati (impronta digitale). Questi tipi di prove crittografiche sono gli strumenti matematici fondamentali per bitcoin e

ampiamente utilizzati nelle applicazioni bitcoin. Ironicamente, la crittografia non è una parte importante di bitcoin, poiché i suoi dati di comunicazione e transazione non sono codificati e non devono essere codificati per proteggere i fondi. In questo capitolo introdurremo alcune delle crittografie utilizzate in bitcoin per controllare la proprietà dei fondi, sotto forma di chiavi, indirizzi e portafogli.

Introduzione

La proprietà dei bitcoin si stabilisce attraverso l'uso delle *chiavi private*, *gli indirizzi bitcoin*, e *le firme digitali*. Le chiavi digitali non sono

realmente conservate nella rete, ma sono invece create e conservate in un file dall'utente, o semplicemente un database, chiamato *wallet*. Le chiavi digitali contenute all'interno del portafoglio dell'utente sono completamente indipendenti dal protocollo bitcoin e possono essere generate e gestite dal software del portafoglio senza fare riferimento alla blockchain o dover accedere a internet. Le chiavi permettono molte interessanti proprietà di bitcoin, includendo anche una fiducia e un controllo decentralizzati, attestati di proprietà e un modello di *prova di sicurezza crittografica*.

La maggior parte delle transazioni

bitcoin richiede che una firma digitale valida sia inclusa nella blockchain, che può essere generata solo con una chiave segreta; quindi, chiunque abbia una copia di quella chiave ha il controllo dei bitcoin. La firma digitale utilizzata per spendere fondi è anche indicata come *witness*, un termine usato in crittografia. I dati dei testimoni in una transazione bitcoin testimoniano la vera proprietà dei fondi spesi.

Le chiavi vengono create a coppie; chiave privata (segreta) e chiave pubblica. Pensa alla chiave pubblica come il numero di conto corrente e alla chiave privata come un PIN segreto, o firma su un assegno, che

fornisce il controllo sul conto. Queste chiavi digitali sono viste raramente da un utente bitcoin. Per la maggior parte del tempo, sono conservate all'interno di un file nel wallet e gestite dal software del wallet bitcoin

Nella fase di pagamento di una transazione di bitcoin, la chiave pubblica del beneficiario è rappresentata dalla sua impronta digitale, chiamata *indirizzo bitcoin*, la quale è usata come il nome del beneficiario di un assegno (i.e., "Pagato a favore di") Nella maggior parte dei casi, un indirizzo bitcoin è generato da e corrisponde a una chiave pubblica. Tuttavia, non tutti gli indirizzi bitcoin rappresentano una

chiave pubblica; essi possono rappresentare anche altri beneficiari, per esempio gli script, come vedremo dopo in questo capitolo. In questo modo, gli indirizzi bitcoin astraggono il destinatario dei fondi rendendo flessibili le destinazioni delle transazioni, in modo simile agli assegni cartacei: un unico strumento di pagamento che può essere usato per pagare direttamente su conto corrente di persone e aziende, pagare le bollette o pagare in contanti. L'indirizzo Bitcoin è l'unica rappresentazione delle chiavi che gli utenti vedranno normalmente, dato che rappresenta l'informazione che dev'essere condivisa con il mondo.

Prima di tutto, introdurremmo la crittografia e spiegheremo la matematica usata in bitcoin. Dopodiché, andremo a vedere come le chiavi sono formate, conservate e gestite. Andremo ad analizzare i vari format di codifica usati per rappresentare le chiavi pubbliche e private, gli indirizzi e gli indirizzi script. Per concludere, andremo ad osservare metodi avanzati di utilizzo degli indirizzi e delle chiavi: vanity, firmamultipla, indirizzi script e wallet di carta

La Crittografia a Chiave Pubblica e le Criptovalute

La crittografia a chiave pubblica venne

inventata nel 1970 e divenne subito una delle basi matematiche dei computer e della sicurezza informatica.

Fin dall'invenzione della crittografia a chiave pubblica, sono state scoperte molte funzioni matematiche, come l'elevazione a potenza dei numeri primi e la moltiplicazione della curva ellittica. Queste funzioni matematiche sono praticamente irreversibili, sono facili da calcolare in una direzione ma impossibili da calcolare nella direzione opposta. Sulla base di queste funzioni matematiche, la crittografia consente la creazione di segreti digitali e firme digitali non falsificabili. Bitcoin usa la

moltiplicazione a curva ellittica come fondamenta per la sua crittografia a chiave pubblica.

In bitcoin, usiamo la crittografia a chiave pubblica per creare una coppia di chiavi che controlla l'accesso ai bitcoin. La coppia di chiavi è formata da una chiave privata e —derivata da essa— un'unica chiave pubblica. La chiave pubblica è usata per ricevere i fondi, mentre la chiave privata è usata per autorizzare le transazioni in uscita.

C'è una relazione matematica tra la chiave pubblica e quella privata che permette alla chiave privata di essere usata per generare firme sui messaggi. Questa firma può essere validata attraverso la chiave pubblica senza

rivelare la chiave privata.

Quando si spendono bitcoin, il proprietario attuale dei bitcoin presenta la sua chiave pubblica e una firma (differente ogni volta, ma creata dalla stessa chiave privata) in una transazione per autorizzare il rilascio di quei bitcoin. Attraverso la presentazione della chiave pubblica e della firma, tutti i membri della rete bitcoin possono verificare e accettare la transazione come valida, confermando che la persona che trasferisce i bitcoin li ha posseduti al momento del trasferimento.

	Nella maggior parte dei wallet, le chiavi private

TIP	e pubbliche, per convenienza, sono conservate insieme come *coppia di chiavi*. Tuttavia, la chiave pubblica può essere calcolata dalla chiave privata, quindi è possibile conservare anche solo la chiave privata.

Chiavi Private e Pubbliche

Un portafoglio bitcoin contiene una raccolta di coppie di chiavi, ognuna composta da una chiave privata e una

pubblica. La chiave privata (k) è un numero, di solito scelto a caso. Dalla chiave privata, viene utilizzata la moltiplicazione a curva ellittica, una funzione di crittografia unidirezionale, per generare una chiave pubblica (K). Dalla chiave pubblica (K), viene utilizzata una funzione di hash crittografico unidirezionale per generare un indirizzo bitcoin (A). In questa sezione, si inizierà con la generazione della chiave privata, analizzando la matematica relativa alla curva ellittica che viene utilizzata per generare una chiave pubblica e, infine, generare un indirizzo bitcoin dalla chiave pubblica. Il rapporto tra la chiave privata, la chiave pubblica e

l’indirizzo bitcoin è mostrato in Figura 14.

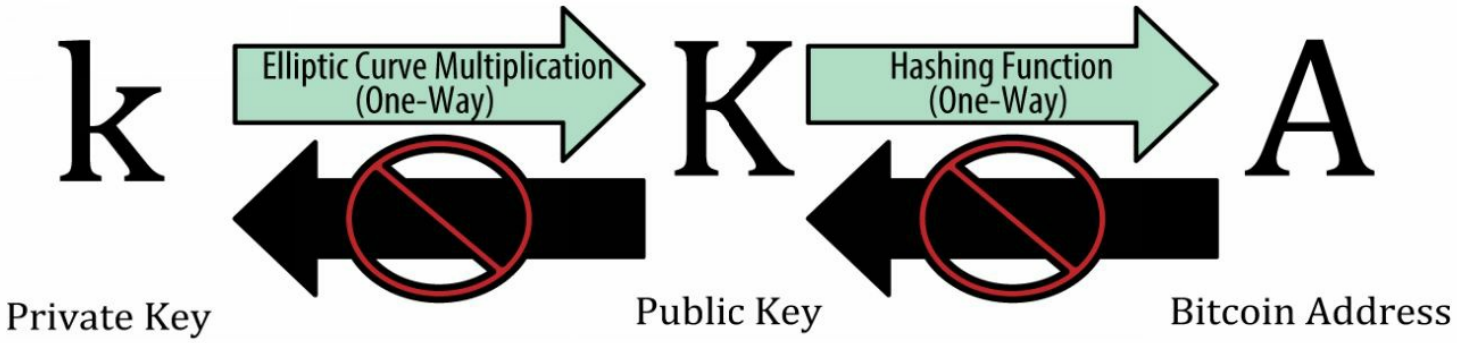


Figura 14. La transazione di Alice inclusa nel blocco #277316

Perché la crittografia asimmetrica è usata in bitcoin? Non è usata per "codificare" (rendere segreta) la transazione. Piuttosto, la proprietà più utile del crittografia asimmetrica è l’abilità di generare *firme*

digitali. La chiave privata può essere applicata come impronta digitale di una transazione per produrre una firma numerica. Questa firma può essere effettuata solamente da chi è a conoscenza della chiave privata. In ogni caso, chiunque abbia accesso alla chiave pubblica e all'impronta della transazione le può usare per *verificare* la firma. Questa utile proprietà della crittografia asimmetrica rende possibile a chiunque di verificare le firme in ogni transazione, assicurando allo stesso tempo ai proprietari delle chiavi private di produrre firme valide.

Chiavi Private

Una chiave privata è semplicemente un numero scelto a caso. La proprietà e il controllo sulla chiave privata è alla base del controllo dell'utente su tutti i fondi associati all'indirizzo bitcoin corrispondente. La chiave privata viene utilizzata per creare firme, dimostrando la proprietà dei fondi utilizzati in una transazione, che sono necessarie per trasferire i bitcoin. La chiave privata deve rimanere segreta in ogni momento, perché rivelarla a terzi equivale a dare loro il controllo sui bitcoin associati a quella chiave. La chiave privata deve essere conservata e protetta da perdite

accidentali, perché se persa non può essere recuperata e i fondi associati ad essa saranno anch'essi persi per sempre.

TIP	La chiave privata di bitcoin è solo un numero. Puoi ottenere la tua chiave privata in modo casuale con solamente una moneta una penna e un foglio: lancia la moneta 256 volte e otterrai la cifra binaria di una chiave privata casuale che potrai usare in un portafoglio bitcoin. La chiave pubblica può

	quindi essere generata dalla chiave privata.

Generare una chiave privata da un numero casuale

Il primo e più importante passo per generare le chiavi è di trovare una fonte sicura di entropia, o casualità. Creare una chiave bitcoin é essenzialmente la stessa cosa di "Scegli un numero tra 1 e 2^{256}." Il metodo esatto usato per scegliere quel numero non é importante fintanto che non è prevedibile o ripetibile. Il software bitcoin usa il generatore di numeri casuali del sistema operativo per produrre 256 bit di

entropia (casualità). Di solito, il generatore di numeri casuali dell'OS è inizializzato da una fonte umana di casualità, che è il motivo per il quale qualche programma potrebbe chiederti di muovere il mouse a caso per qualche secondo.

Più accuratamente, la chiave privata potrebbe essere un qualsiasi numero da 1 e n - 1, dove n è una costante (n = 1.158 * 10^{77}, un po meno di 2^{256}) definita come l'ordine della curva ellittica usata in bitcoin (vedi Crittografia a Curve Ellittiche). Per creare una chiave del genere, sceglieremo casualmente un numero da 256-bit e controlleremo che sia inferiore di n - 1. Nei termini da programmatori,

questo viene realizzato tramite il dare in pasto una stringa più lunga di bit casuali, all'algoritmo di hash SHA256 che produrrà convenientemente un numero da 256-bit. Se il risultato è meno di n - 1, avremo una chiave privata adeguata. Altrimenti, riproveremo nuovamente con un altro numero casuale.

	Non riscriver codice per cr un generatore numeri casua non utilizzare generatore numeri cas "semplice" offerto dal linguaggio

ATTENZIONE

programmazio
Utilizza
generatore
numeri pseu
randomici
crittograficam
sicuro
(CSPRNG)
ottenga un s
da una fonte
sufficiente
entropia. St
la
documentazion
della libreria
numeri cas
che hai scelto
essere certo

	sia crittograficam sicura. corretta implementazio del CSPRN critica per sicurezza per chiavi.

La seguente è una chiave privata generata in modo casuale (k) mostrata nella sua rappresentazione esadecimale (256 bits mostrati come 64 cifre esadecimali, ognuna da 4 bit):

```
1E99423A4ED27608A15A2616A2B0E9E5
```

	La dimensione di una

TIP	chiave privata bitcoin 2256 è un numero inimmaginabilmente grande. E approssimativamente 1077 cifre decimali. In paragone, è stato stimato che l'universo visibile contenga 1080 atomi.

Per generare una nuova chiave con il client Bitcoin Core, utilizza il comando getnewaddress. Per ragioni di sicurezza, esso mostrerà solamente la chiave pubblica, non la chiave privata. Per chiedere a bitcoind di esporre la chiave privata, utilizza il comando dumpprivkey. Il comando

dumpprivkey mostra la chiave privata in un formato Base58 checksum-encoded chiamato *Wallet Import Format* (WIF), il quale esamineremo con maggiore dettaglio in Formati delle Chiavi Private. Ecco un esempio per generare e mostrare una chiave privata utilizzando questi due comandi:

```
$ bitcoin-cli getnewaddress
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoX
$ bitcoin-cli dumpprivkey
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoX
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3
```

Il comando dumpprivkey apre il wallet e estrae la chiave privata che è stata generata dal comando getnewaddress. Non è invece possibile per bitcoind

conoscere la chiave privata avendo solo la chiave pubblica, a meno che non siano entrambe salvate nel wallet.

TIP	Il comando dumpprivkey non genera una chiave privata da una chiave pubblica, visto che è impossibile. Il comando semplicemente rivela la chiave privata che è già conosciuta al wallet e che è stata generata dal comando getnewaddress

È inoltre possibile utilizzare lo strumento Bitcoin Explorer da riga di comando per generare e visualizzare le

chiavi private con i comandi seed, ec-new and ec-to-wif:

```
$ bx seed | bx ec-new | bx ec-to-wif
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfc
```

Chiavi Pubbliche

La chiave pubblica è calcolata dalla chiave privata utilizzando la proprietà di moltiplicazione delle curve ellittiche, la quale è un'operazione irreversibile: $K = k * G$, dove k è la chiave privata, G è un punto della curva ellittica costante chiamato punto di generazione e K è la chiave pubblica risultante. L'operazione inversa, conosciuta come il "trovare il logaritmo discreto"--calcolando k affinché K sia difficile da trovare come eseguendo la seguente operazione:

provare tutti i possibili valori di k, ovvero con una ricerca brute-force. Prima di dimostrare come generare una chiave pubblica da una chiave privata, osserviamo la crittografia delle curve ellittiche un poco più in dettaglio.

La moltiplicazione a curva ellittica è un tipo di funzione chiamata dai crittografici *funzione "trap door"*: è molto semplice da calcolare in una direzione (moltiplicazione) e impossibile da calcolare nella direzione opposta (divisione). I proprietario della chiave

TIP	privata può tranquillamente creare una chiave pubblica e condividerla con chiunque sapendo che nessuno può invertire la funzione e calcolare la chiave privata dalla pubblica. Questo trucchetto matematico è diventato la base per una firma digitale sicura, non falsificabile, e che prova la proprietà dei bitcoin.

Crittografia a Curve Ellittiche

La crittografia a curva ellittica è di tipo asimmetrico o crittografia a

chiave pubblica basata su un logaritmo discreto espresso dall'addizione e la moltiplicazione di punti su una curva ellittica.

La Figura 15 è un esempio di curva ellittica, simile a quella usata da bitcoin.

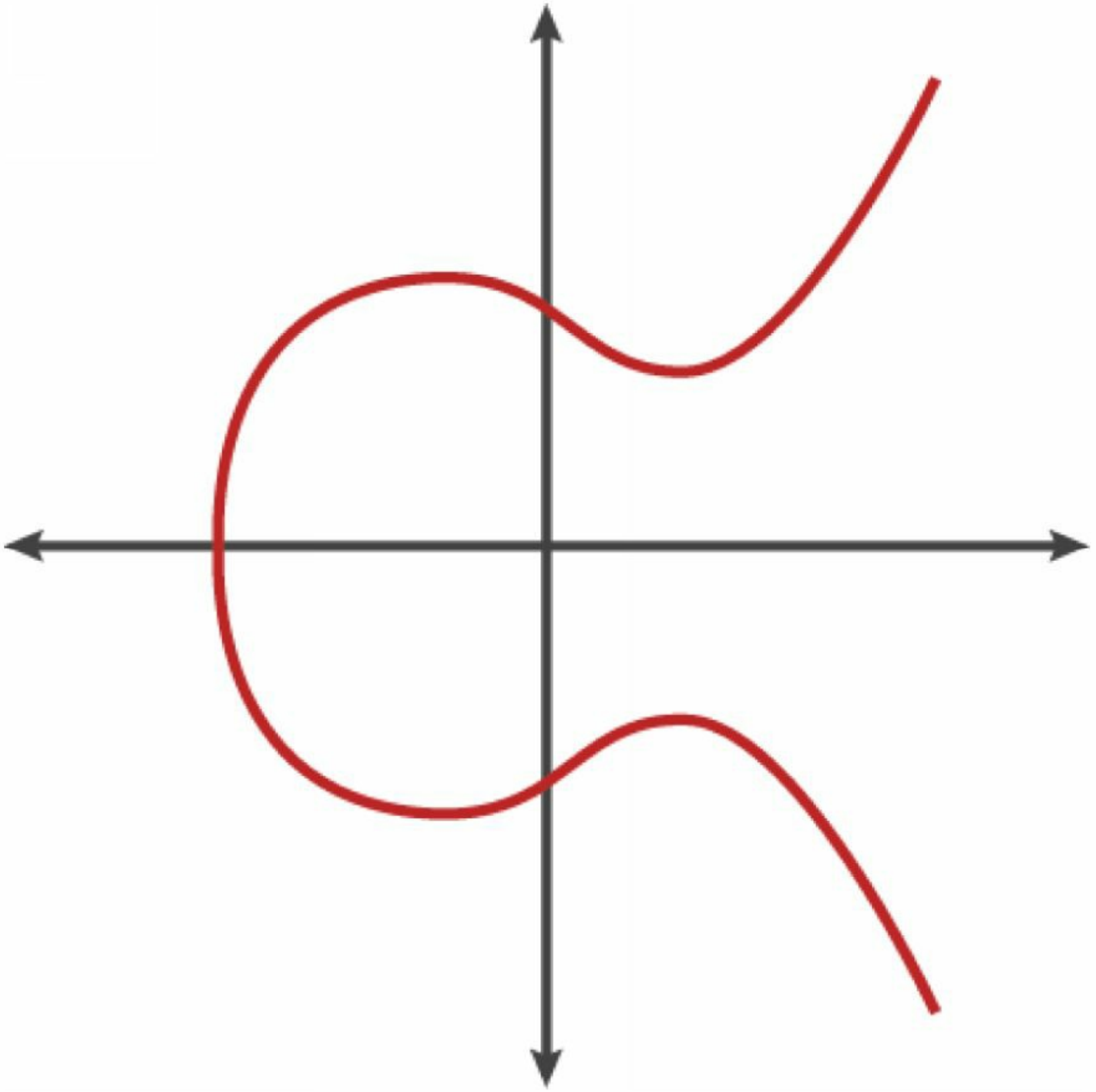


Figura 15. Una curva ellittica

Bitcoin utilizza una curva ellittica

specifica e una serie di costanti matematiche, definite in uno standard detto secp256k1, stabilita dal National Institute of Standards and Technology (NIST) - Istituto Nazionale degli Standard e della Tecnologia. La curva secp256k è definita dalla seguente funzione, la quale produce una curva ellittica:

$y^2 = (x^3 + 7)$ over (F_p)

o

$y^2 \bmod p = (x^3 + 7) \bmod p$

Il *mod p* (modulo del numero primo p) indica che questa curva è su un campo finito di numeri primi di ordine *p*, che possiamo scrivere come F_p, dove $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$, un

numero primo molto grande.

Dato che questa curva è definita su un campo finito di ordine primo invece che su un campo di numeri reali, viene osservata come un pattern di punti sparpagliati in due dimensioni, ed è questa proprietà che li rende difficili da visualizzare. In ogni caso, la matematica utilizzata è identica a quella di una curva ellittica su (un campo finito) di numeri reali. Come esempio, la Figura 16 mostra la stessa curva ellittica su di un campo finito molto più piccolo di ordine primo 17, mostrando un pattern di punti su di una griglia. La curva ellittica bitcoin secp256k1 può essere visualizzata come ad un pattern molto più complesso di punti su di una griglia

impensabilmente ampia.

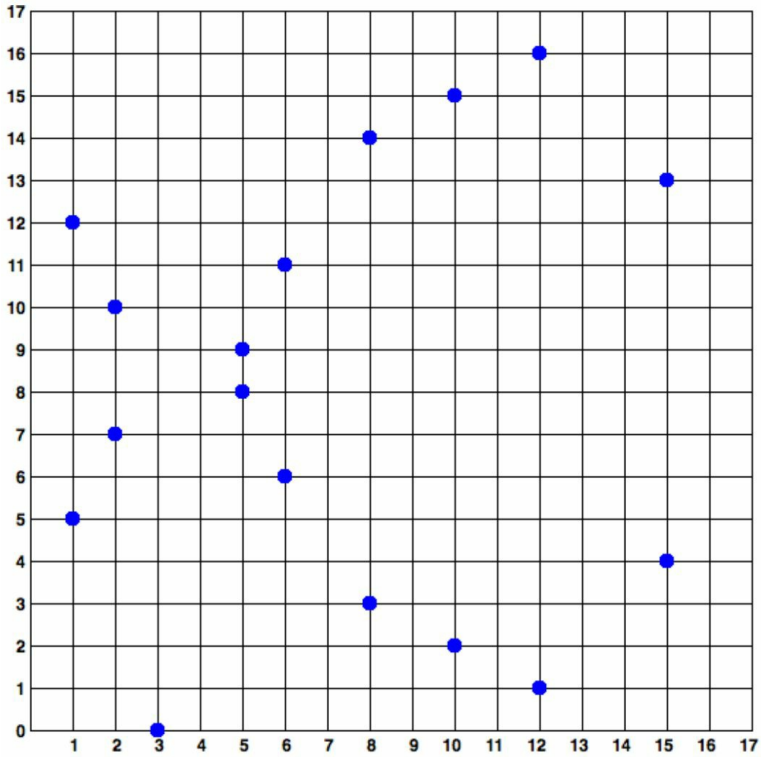


Figura 16. Crittografia a curve ellittiche: visualizzando una curva ellittica su F(p), con p=17.

Quindi, per esempio, il seguente è un punto P con coordinate (x,y) che è un punto sulla curva secp256k1:

P =

```
(55066263022773436695787188951685
32670510020758816978083085130507041
```

L'Esempio 7 mostra come verificarlo usando Python:

Esempio 7. L'utilizzo di Python per confermare che il punto si trova sulla curva ellittica

```
Python 3.4.0 (default, Mar 30 2014,
19:23:13)
[GCC 4.2.1 Compatible Apple LLVM
5.1 (clang-503.0.38)] on darwin
Type "help", "copyright", "credits" or
"license" for more information.
>>> p =
11579208923731619542357098500868
```

```
>>> x =
55066263022773436695787188951685
>>> y =
32670510020758816978083085130507
>>> (x ** 3 + 7 - y**2) % p
0
```

Nella matematica delle curve ellittiche, c'è un punto chiamato il "punto all'infinito," il quale corrisponde quasi al ruolo del numero 0 in un'addizione. Nei computer, è rappresentato qualche volta come x = y = 0 (che non soddisfa l'equazione delle curve ellittiche, ma è un caso semplice e separato che può essere verificato).

Esiste anche un operatore +, chiamato "addizione," il quale ha alcune

proprietà simili a quello tradizionale sui numeri reali che i bambini imparano a scuola. Dati due punti P1 e P2 sulla curva ellittica, esiste un terzo punto $P_3 = P_1 + P_2$, anch'esso sulla curva ellittica.

Geometricamente, questo terzo punto P3 viene calcolato disegnando una linea tra P1 e P2. Questa linea andrà a intersecare la curva ellittica in esattamente un altro punto addizionale. Chiamiamo questo punto P3 = (x, y). Lo riflettiamo sull'asse x per ottenere P3 = (x, –y).

Ci sono un paio di casi particolari che chiariscono il bisogno di "puntare all'infinito".

Se P1 e P2 sono lo stesso punto, la linea "tra" P1 e P2 dovrebbe estendersi per essere la tangente sulla curva al punto P1. Questa tangente intersecherà la curva in esattamente un nuovo punto. Puoi utilizzare tecniche dal calcolo per determinare l'inclinazione (angolo) della linea tangente. Queste tecniche funzionano curiosamente, anche se noi stiamo restringendo il nostro interesse ai punti sulla curva con due coordinate intere!

In alcuni casi (cioè se P_1 e P_2 hanno gli stessi valori x ma diversi valori y), la linea tangente sarà esattamente verticale, nel qual caso P_3 = "punto all'infinito."

Se P1 è il "punto all'infinito," allora anche la sua somma $P_1 + P_2 = P_2$. Allo stesso modo, se P_2 è il punto all'infinito, allora è valido anche $P_1 + P_2 = P_1$. Questo mostra come il punto all'infinito abbia il ruolo di 0.

È il caso che + sia associativo, il quale significa anche che è valido $(A + B) + C = A + (B + C)$. Quindi possiamo andare a scrivere $A + B + C$ senza ambiguità e senza bisogno di utilizzare parentesi.

Ora che abbiamo definito l'addizione, possiamo definire la moltiplicazione nel modo standard che estende l'addizione. Per un punto P sulla curva ellittica, se k è un numero intero,

allora kP = P + P + P + ... + P (k volte) è anche valido. Nota che k è talvolta confuso e chiamato "esponente" in questo caso.

Generare una Chiave Pubblica

Partendo da una chiave privata sotto forma di un numero generato in modo casuale *k*, lo moltiplichiamo con un punto predeterminato sulla curva denominato il *generator point G* per produrre un altro punto in qualche altra parte della curva, che corrisponde alla chiave pubblica *K*. Il punto generato è specificato come parte dello standard secp256k1 ed è sempre lo stesso per tutte le chiavi in bitcoin:

$$K = k * G$$

dove k è la chiave privata, G è il punto generatore e K è la chiave pubblica risultante, un punto sulla curva. Poiché il punto di generazione è sempre lo stesso per tutti gli utenti di bitcoin, una chiave privata k moltiplicata per G avrà sempre la stessa chiave pubblica K. La relazione tra k e K è fissa, ma può essere calcolata solo in una direzione, da k a K. Ecco perché un indirizzo bitcoin (derivato da K) può essere condiviso con chiunque e non rivela la chiave privata dell'utente (k).

	Una chiave privata può essere convertita in una chiave pubblica, ma una

TIP	chiave pubblica non può essere ri-convertita in una chiave privata perchè la funzione matematica utilizzata è unidirezionale.

Implementando la moltiplicazione complessa (teoria delle curve ellittiche), prendiamo la chiave privata *k* generata precedentemente e la moltiplichiamo con il punto generatore *G* per trovare la chiave pubblica *K*:

```
K =
1E99423A4ED27608A15A2616A2B0E9E5
* G
```

La Chiave Pubblica *K* è definita come

punto K = (x,y):

```
K = (x, y)

dove,

x =
F028892BAD7ED57D2FB57BF33081D5CF
y =
07CF33DA18BD734C600B96A72BBC4749
```

Per visualizzare la moltiplicazione di un punto con un numero intero, utilizzeremo la curva ellittica più semplice rispetto ai numeri reali - ricorda, la matematica è la stessa. Il nostro obiettivo è trovare il multiplo *kG* del punto *G* del generatore. Equivale ad aggiungere *G* a se stesso, k volte di seguito. Nelle curve ellittiche, aggiungere un punto a se

stesso equivale a tracciare una linea tangente al punto e a trovare dove questa interseca nuovamente la curva, quindi a riflettere quel punto sull'asse x.

La Figura 17 mostra il processo di derivazione di *G*, *2G*, *4G*, come un'operazione geometrica sulla curva.

TIP	La maggior parte delle implementazioni di bitcoin utilizzano OpenSSL cryptographic library (http://bit.ly/1ql7bn8) per gestire la matematica della curva ellittica. Ad esempio, per derivare la

chiave pubblica, usano la funzione EC_POINT_mul().

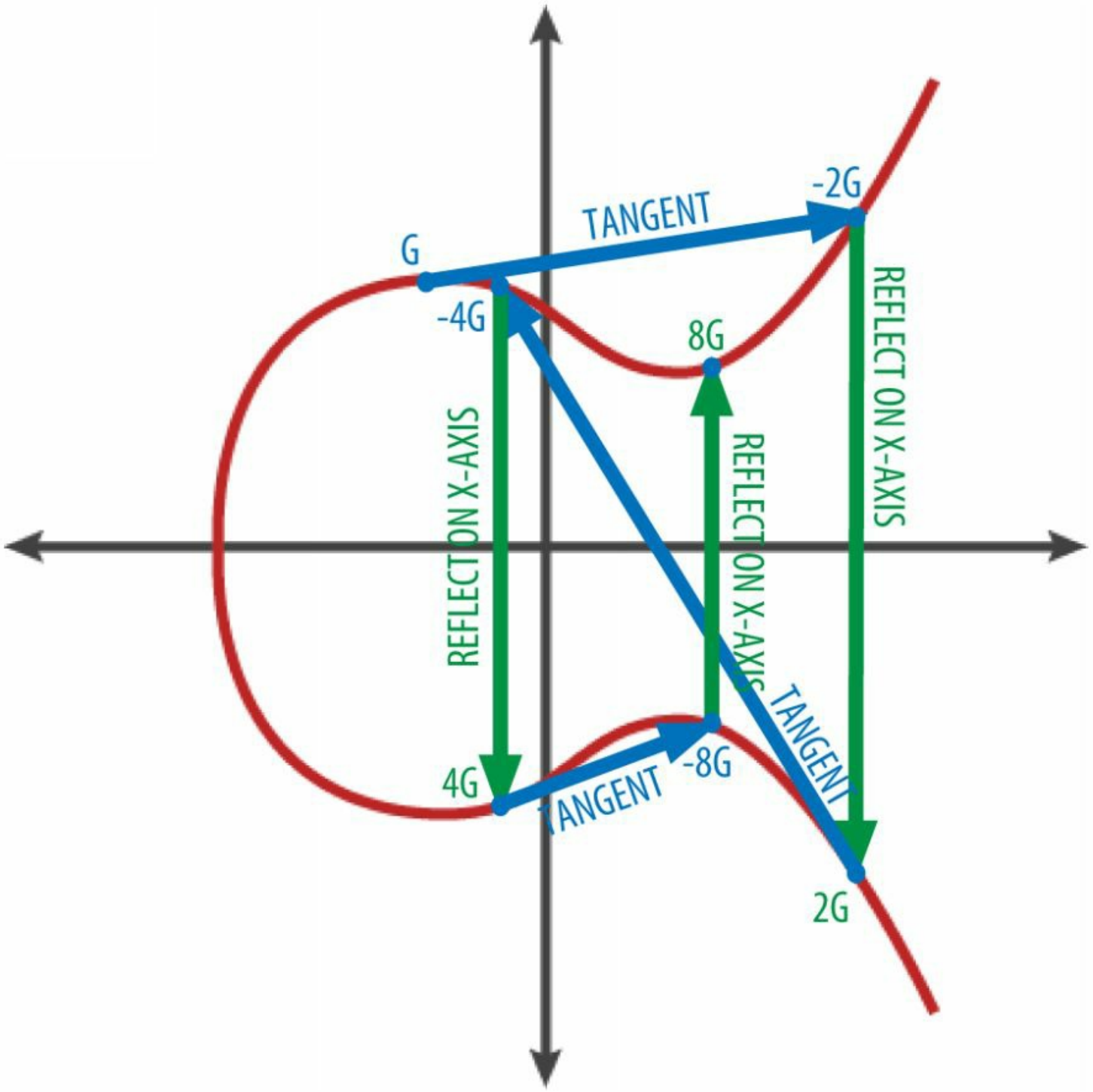


Figura 17. Crittografia Ellittica: la moltiplicazione di un punto G per un intero k su una curva ellittica

Indirizzi Bitcoin

Un indirizzo bitcoin è una stringa di cifre e caratteri che possono essere condivisi con chiunque desideri inviarti denaro. Gli indirizzi prodotti dalle chiavi pubbliche sono costituiti da una stringa di numeri e lettere, che inizia con la cifra "1". Ecco un esempio di indirizzo bitcoin:

1J7mdg5rbQyUHENYdx39WVWK7fsLpEoX

L'indirizzo bitcoin è quello che appare

comunemente in una transazione come il "destinatario" dei fondi. Se volessimo paragonare una transazione bitcoin ad un assegno, l'indirizzo bitcoin e' il beneficiario, che è cio che si scrive sulla riga seguita da "Pagato all'ordine di". Su un assegno cartaceo, tale beneficiario può a volte essere il nome di un titolare di un conto bancario, ma può anche includere società, istituzioni o anche denaro contante. Poiché gli assegni cartacei non devono specificare un account (n.d.t. si fa riferimento ad assegni al portatore che in Italia non è più possibile utilizzare), ma piuttosto utilizzare un nome astratto come destinatario di fondi, ciò rende gli

assegni cartacei molto flessibili come strumenti di pagamento. Le transazioni bitcoin usano un'astrazione simile, l'indirizzo bitcoin, per renderle molto flessibili. Un indirizzo bitcoin può rappresentare il proprietario di una coppia di chiavi privata/pubblica o può rappresentare qualcos'altro, come uno script di pagamento, come vedremo nei prossimi capitoli. Per ora, esaminiamo il caso semplice, un indirizzo bitcoin che rappresenta, ed è derivato da, una chiave pubblica.

L'indirizzo bitcoin è derivato dalla chiave pubblica attraverso l'uso di hashing crittografico a senso unico. Un "algoritmo di hashing" o semplicemente "algoritmo di hash" e'

una funzione a senso unico che produce un'impronta digitale o "hash" di un input di dimensione arbitraria. Le funzioni di hash crittografiche sono usate ampiamente in bitcoin: negli indirizzi bitcoin, negli indirizzi di script, e negli algoritmi di mining proof-of-work. Gli algoritmi usati per creare un indirizzo bitcoin da una chiave pubblica sono il Secure Hash Algorithm (SHA) ed il RACE Integrity Primitives Evaluation Message Digest (RIPEMD), specificatamente SHA256 e RIPEMD160.

Iniziando con la chiave pubblica K, calcoliamo l'hash SHA256 e poi calcoliamo l'hash RIPEMD160 del risultato, producendo un numero a

160-bit (20-byte): dove K è la chiave pubblica e A è l'indirizzo bitcoin risultante.

A = RIPEMD160(SHA256(K))

dove *K* è la chiave pubblica e *A* è l'indirizzo bitcoin risultante.

TIP	Un indirizzo bitcoin *non* è la stessa cosa di una chiave pubblica. Gli indirizzi bitcoin sono derivati da una chiave pubblica usando una funzione a senso unico.

Gli indirizzi bitcoin sono quasi sempre presentati agli utenti in una codifica

chiamata "Base58Check", che usa 58 caratteri (sistema numerico Base58) ed un checksum per facilitare la leggibilità umana, evitando ambiguità, e proteggere contro errori nella trascrizione ed ingresso di indirizzi. Base58Check è utilizzato anche in molti altri modi in bitcoin, se c'è il bisogno per l'utente di leggere e trascrivere correttamente un numero, come un indirizzo bitcoin, una chiave privata, una chiave criptata, o un hash di script. Nella prossima sezione esamineremo il meccanismo di codifica e decodifica di Base58Check, e le rappresentazioni risultanti. La Figura 18 illustra la conversione di una chiave pubblica in un indirizzo

bitcoin.

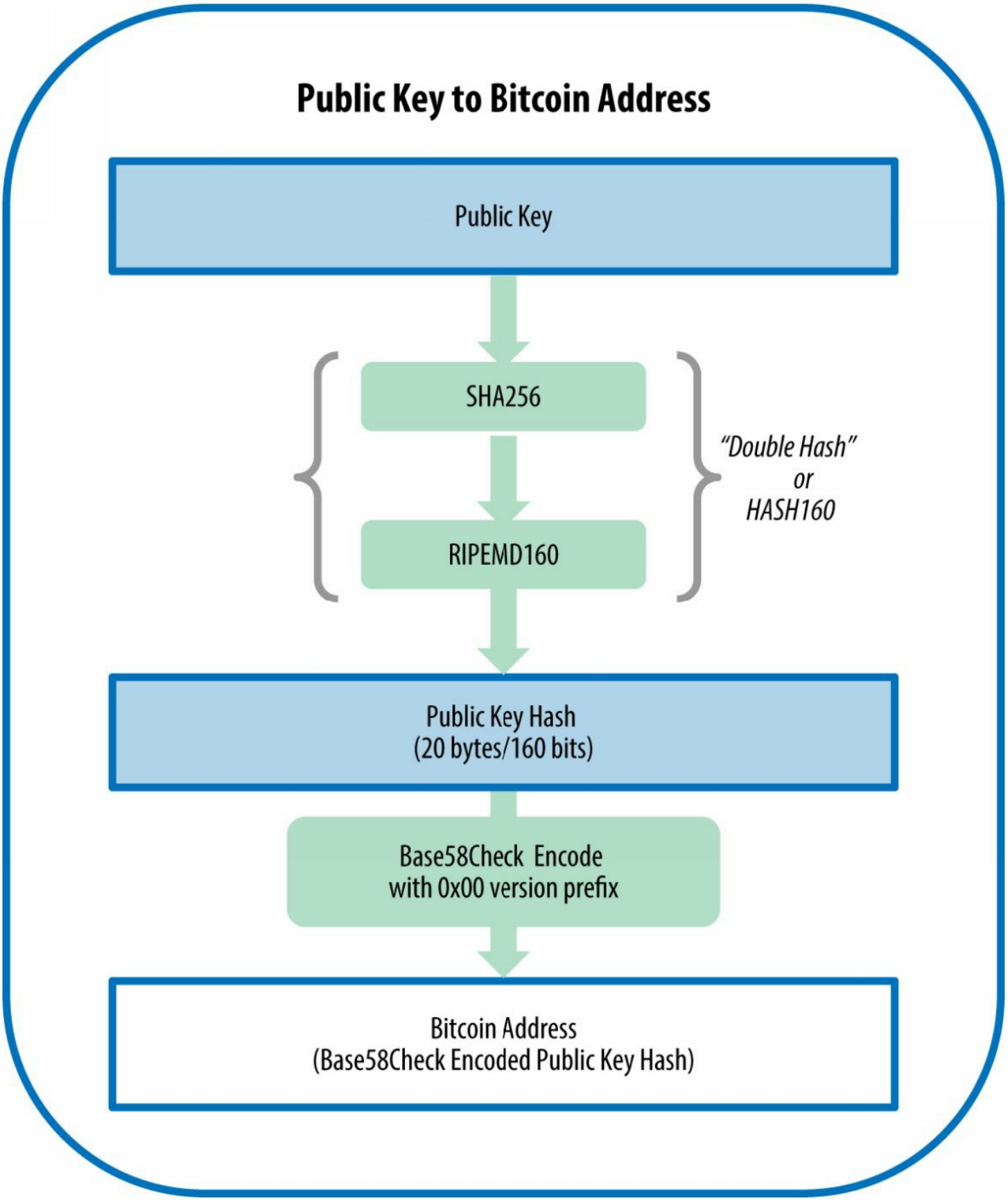


Figura 18. Da chiave

pubblica a indirizzo bitcoin: conversione di una chiave pubblica in un indirizzo bitcoin

Encoding Base58 e Base58Check

Al fine di rappresentare lunghi numeri in maniera compatta, usando alcuni simboli, molti sistemi di computer usano rappresentazioni alfanumeriche miste con una base (o radice) superiore a 10. Ad esempio, se il sistema decimale tradizionale usa i 10 numerali 0 fino a 9, il sistema esadecimale usa 16, con le lettere A fino ad F come 6 simboli aggiuntivi.

Un numero rappresentato nel formato esadecimale è più corto rispetto all'equivalente rappresentazione. Anche più compatto, la rappresentazione Base-64 usa 26 lettere minuscole, 26 lettere maiuscole, 10 numerali, e due ulteriori caratteri come "+" e "/" per trasmettere dati binari attraverso medium testuali come email. Base-64 e' maggiormente utilizzato per aggiungere allegati binari alle email. Base58 e' un formato di codifica binario testuale sviluppato per l'impiego in bitcoin and usato in molte altre criptovalute. Offre un equilibrio tra compatta rappresentazione, leggibilità, e riconoscimento e prevenzione degli

errori. Base58 e' un sottoinsieme di Base64, impiegando lettere maiuscole e minuscole e numeri, ma omettendo alcuni caratteri che sono frequentemente confusi per altri e che possono sembrare identici quando visualizzati in alcuni tipi di font. In particolare, Base58 e' Base64 senza lo 0 (numero zero), O (o maiuscola), l (L minuscola), I (i maiuscola), ed il simbolo "\+" e "/". O, più semplicemente, è un insieme di lettere minuscole e maiuscole e numeri senza il quartetto (0, O, l, I) appena menzionato. L'Esempio 8 mostra l'alfabeto Base58 completo.

Esempio 8. L'alfabeto

bitcoin in Base58

123456789ABCDEFGHJKLMNPQRSTU

Per aggiungere ulteriore sicurezza rispetto ad errori di battitura o di trascrizione, Base58Check e' un formato di codifica Base58, usato di frequente in bitcoin, che ha un codice per la verifica degli errori integrato. Il checksum sono degli ulteriori quattro byte aggiunti alla fine dei dati che vengono codificati. Il checksum deriva dall'hash dei dati codificati e può quindi essere utilizzato per rilevare e prevenire errori di trascrizione e di battitura. Quando viene presentato con un codice Base58Check, il software di

decodifica calcola il checksum dei dati e lo confronta con il checksum incluso nel codice. Se i due non corrispondono, ciò indica che è stato introdotto un errore e che i dati di Base58Check non sono validi. Ad esempio, ciò impedisce che un indirizzo bitcoin errato venga accettato dal software wallet come destinazione valida, un errore che altrimenti comporterebbe una perdita di fondi.

Per convertire i dati (un numero) in un formato Base58Check, prima aggiungiamo un prefisso ai dati, chiamato "byte di versione", che serve per identificare facilmente il tipo di dati che è codificato. Ad esempio, nel caso di un indirizzo bitcoin il prefisso

è zero (0x00 in esadecimale), mentre il prefisso usato quando si codifica una chiave privata è 128 (0x80 in esadecimale). Un elenco di prefissi di versione comuni è mostrato in Tabella 4-1.

Successivamente, calcoliamo il checksum a "doppio-SHA", cioè applichiamo l'algoritmo di hashing SHA256 due volte sul risultato precedente (prefisso e dati).

```
checksum = SHA256(SHA256(prefix+data))
```

Dal risultante hash a 32 byte (hash del hash), prendiamo solo i primi quattro byte. Questi quattro byte servono come codice di controllo degli errori o checksum. Il checksum è concatenato

(aggiunto) alla fine.

Il risultato è composto da tre elementi: un prefisso, i dati e un checksum. Questo risultato è codificato usando l'alfabeto Base58 descritto in precedenza. La Figura 19 illustra il processo di codifica Base58Check.

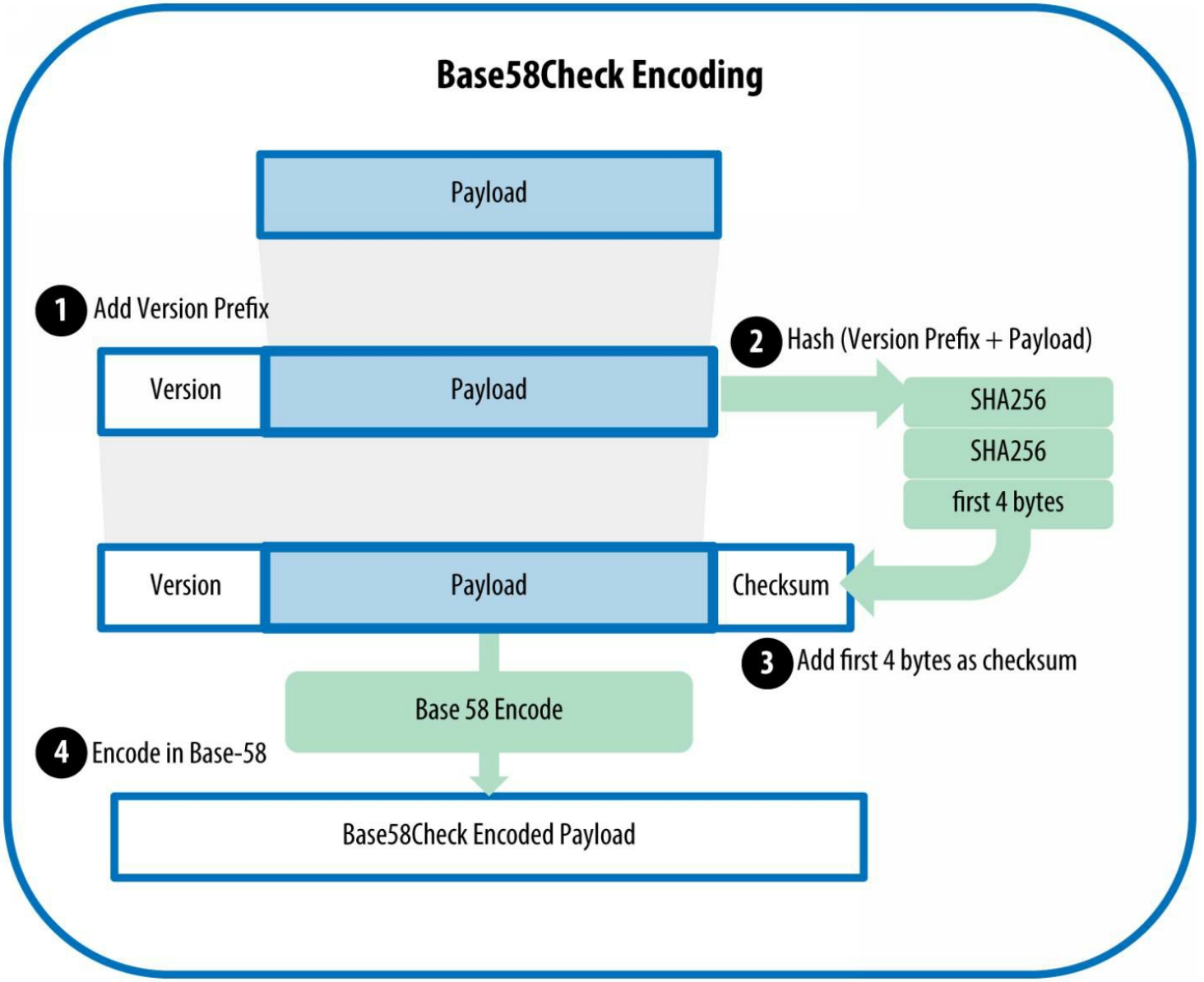


Figura 19. encoding Base58Check: un formato Base58, versionato, con checksum per codificare dati bitcoin senza generare

ambiguità

In bitcoin, la maggior parte dei dati presentati all'utente è codificata Base58Check per renderli compatti, facili da leggere e facilitare la rilevazione degli errori. Il prefisso di versione nella codifica Base58Check viene utilizzato per creare formati facilmente distinguibili, che quando codificati in Base58 contengono caratteri specifici all'inizio della stringa codificata in Base58Check. Questi caratteri rendono facile per gli umani identificare il tipo di dati codificati e come usarli. Questo è ciò che differenzia, per esempio, un indirizzo bitcoin codificato Base58Check che inizia con un 1 da un

formato WIF con chiave privata codificata Base58Check che inizia con un 5. Alcuni esempi di prefissi di versione e i relativi caratteri Base58 sono mostrati Tabella 4-1.

Tabella 4-1. Prefisso di versio Base58Check e esempi del risultato

Tipo	**Prefisso di versione (esadecimale)**	**Prefiss del risulta Base5**
Indirizzo Bitcoin	0x00	1

Indirizzo Pay-to-Script-Hash	0x05	3
Indirizzo Bitcoin Testnet	0x6F	m oppure
Private Key WIF	0x80	5, K, L
BIP-38 Encrypted Private Key	0x0142	6P
BIP-32 Extended	0x0488B21E	xpub

Public Key		

Formati di Chiavi

Sia le chiavi private che quelle pubbliche possono essere rappresentate in numerosi formati diversi. Queste rappresentazioni codificano tutte lo stesso numero, anche se hanno un aspetto diverso. Questi formati vengono principalmente utilizzati per semplificare la lettura e la trascrizione delle chiavi da parte delle persone senza introdurre errori.

Formati delle Chiavi Private

La chiave privata può essere rappresentata in numerosi formati diversi, ognuno dei quali corrisponde

allo stesso numero di 256-bit. La Tabella 4-2 mostra tre formati comuni utilizzati per rappresentare le chiavi private. Diversi formati sono utilizzati in diverse circostanze. I formati binari esadecimali e grezzi sono utilizzati internamente nel software e raramente mostrati agli utenti. Il WIF viene utilizzato per importare / esportare chiavi tra portafogli e spesso utilizzate nelle rappresentazioni di codici QR (codice a barre) di chiavi private.

Tabella 4-2. Rappresentazioni chiavi private (formati di codifica)

Tipo	Prefisso	Descrizion
Raw	Nessuno	32 bytes

Hex	Nessuno	64 ci esadecimal
WIF	5	Base58Che encoding: Base58 c prefisso versione checksum 128 e 32 bi
WIF-compressed	K o L	Come sop con l’aggiunta un suffis 0x01 precedente all’encodin

La Tabella 1 mostra la chiave provata generata nei seguenti tre formati.

Tabella 1. Esempio: Stessa chiave, f

Format	Private key
Hex	1e99423a4ed27608a1
WIF	5J3mBbAH58CpQ3Y
WIF-compressed	KxFC1jmwwCoACiC

Tutte queste rappresentazioni sono modi diversi di mostrare lo stesso numero, la stessa chiave privata.

Sembrano diverse, ma qualunque di questi formati possono essere convertiti in qualsiasi altro formato.

Usiamo il comando wif-to-ec dal Bitcoin Explorer per mostrare entrambe le chiavi WIF rappresentanti la stessa chiave privata:

```
$ bx wif-to-ec
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfc
1e99423a4ed27608a15a2616a2b0e9e52ced

$ bx wif-to-ec
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3
1e99423a4ed27608a15a2616a2b0e9e52ced
```

Decodifica da Base58Check

I comandi di Bitcoin Explorer semplificano la scrittura di script di shell e "pipes" da riga di comando che

manipolano chiavi bitcoin, indirizzi e transazioni. È possibile utilizzare Bitcoin Explorer per decodificare il modulo in formato Base58Check da riga di comando.

Usiamo il comando base58check-decode per decodificare e decomprimere la chiave:

```
$ bx base58check-decode
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfc
wrapper
{
   checksum 4286807748
   payload
1e99423a4ed27608a15a2616a2b0e9e52ced
   version 128
}
```

Il risultato contiene una chiave come payload, il prefisso Wallet Import

Format (WIF) 128, e un checksum.

Nota che il "payload" della chiave compressa è accodato con il suffisso 01, stando a segnalare che la chiave pubblica derivata è da generare compressa.

```
$ bx base58check-decode
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3
wrapper
{
    checksum 2339607926
    payload
1e99423a4ed27608a15a2616a2b0e9e52ced
    version 128
}
```

Codifica da esadecimale a Base58Check

Per codificare in Base58Check

(l'opposto del comando precedente), usiamo il comando base58check-encode da Bitcoin Explorer fornendo la chiave privata esadecimale, seguita dal Prefisso della versione WIF (Wallet Import Format) 128:

```
bx base58check-encode
1e99423a4ed27608a15a2616a2b0e9e52ced
--version 128
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfc
```

Codifica da hex (chiave compressa) a Base58Check

Per effettuare la codifica in Base58Check come chiave privata "compressa", accodiamo il suffisso 01 alla chiave esadecimale e poi la codifichiamo come abbiamo fatto

prima:

```
$ bx base58check-encode
1e99423a4ed27608a15a2616a2b0e9e52ced
--version 128
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3
```

Il formato WIF-compressed risultante inizia con una "K". Questo denota che la chiave privata contenuta ha il suffisso di "01" e sarà usata per produrre solo chiavi pubbliche compresse.

Formati di Chiavi Pubbliche

Le chiavi pubbliche vengono presentate anche in diversi modi, soprattutto come chiavi pubbliche *compresse* o *non compresse*.

Come abbiamo visto in precedenza, la

chiave pubblica è un punto sulla curva ellittica costituito da una coppia di coordinate + (x, y) . Di solito viene presentato con il prefisso 04 seguito da due numeri a 256 bit, uno per la coordinata *x* del punto, l'altro per la coordinata *y*. Il prefisso 04 viene utilizzato per distinguere le chiavi pubbliche non compresse dalle chiavi pubbliche compresse che iniziano con 02 o 03.

Ecco la chiave pubblica generata dalla chiave privata che abbiamo creato in precedenza, mostrata con le coordinate x e y:

```
x =
F028892BAD7ED57D2FB57BF33081D5CF
y =
```

```
07CF33DA18BD734C600B96A72BBC4749
```

Qui si può notare la stessa chiave pubblica mostrata come un numero da 520-bit (130 cifre esadecimali) con il prefisso 04 seguito dalle coordinate x e y, come 04 x y:

```
K =
04F028892BAD7ED57D2FB57BF330
```

```
07CF33DA18BD734C600B96A72BB
```

Chiavi pubbliche compresse

Le chiavi pubbliche compresse sono state introdotte in bitcoin per ridurre la dimensione delle transazioni e risparmiare spazio su disco su nodi che memorizzano la blockchain. La maggior parte delle transazioni include

la chiave pubblica, necessaria per convalidare le credenziali del proprietario e per trasferire il bitcoin. Ogni chiave pubblica richiede 520 bit (prefisso \+ x \+ y), che quando moltiplicato per diverse centinaia di transazioni per blocco o decine di migliaia di transazioni al giorno, aggiunge una quantità significativa di dati alla blockchain.

Come abbiamo visto nella sezione Chiavi Pubbliche, una chiave pubblica è un punto (x, y) su una curva ellittica. Poiché la curva esprime una funzione matematica, un punto sulla curva rappresenta una soluzione all'equazione e, quindi, se conosciamo la coordinata x, possiamo calcolare la

coordinata y risolvendo l'equazione y2 mod p = (x ^ 3 ^ + 7) mod p. Ciò ci consente di memorizzare solo la coordinata x del punto chiave pubblico, omettendo la coordinata y e riducendo la dimensione della chiave e lo spazio richiesto per memorizzarla di 256 bit. Una riduzione di quasi il 50% delle dimensioni in ogni transazione consente di risparmiare molto spazi dati nel tempo!

Mentre le chiavi pubbliche non compresse hanno un prefisso 04, le chiavi pubbliche compresse iniziano con un prefisso 02 o 03. Vediamo perché ci sono due possibili prefissi: perché il lato sinistro dell'equazione è y^2, ciò significa che la soluzione per y

è una radice quadrata, che può avere un valore positivo o negativo. Visivamente, ciò significa che la coordinata y risultante può essere sopra l'asse x o sotto l'asse x. Come si può vedere dal grafico della curva ellittica in Figura 4-2, la curva è simmetrica, il che significa che viene riflessa come uno specchio dall'asse x. Quindi, mentre possiamo omettere la coordinata y, dobbiamo memorizzare il segno di y (positivo o negativo), o in altre parole, dobbiamo ricordare se era sopra o sotto l'asse x poiché ciascuna di queste opzioni rappresenta un diverso punto e una chiave pubblica diversa. Quando si calcola la curva ellittica nell'aritmetica binaria sul

campo finito dell'ordine primo p, la coordinata y è pari o dispari, che corrisponde al segno positivo / negativo come spiegato in precedenza. Pertanto, per distinguere tra i due possibili valori di y, memorizziamo una chiave pubblica compressa con il prefisso 02 se y è pari, e 03 se è dispari, consentendo al software di dedurre correttamente la coordinata y la coordinata x e decomprimere la chiave pubblica per le coordinate complete del punto. La compressione della chiave pubblica è illustrata in Figura 20.

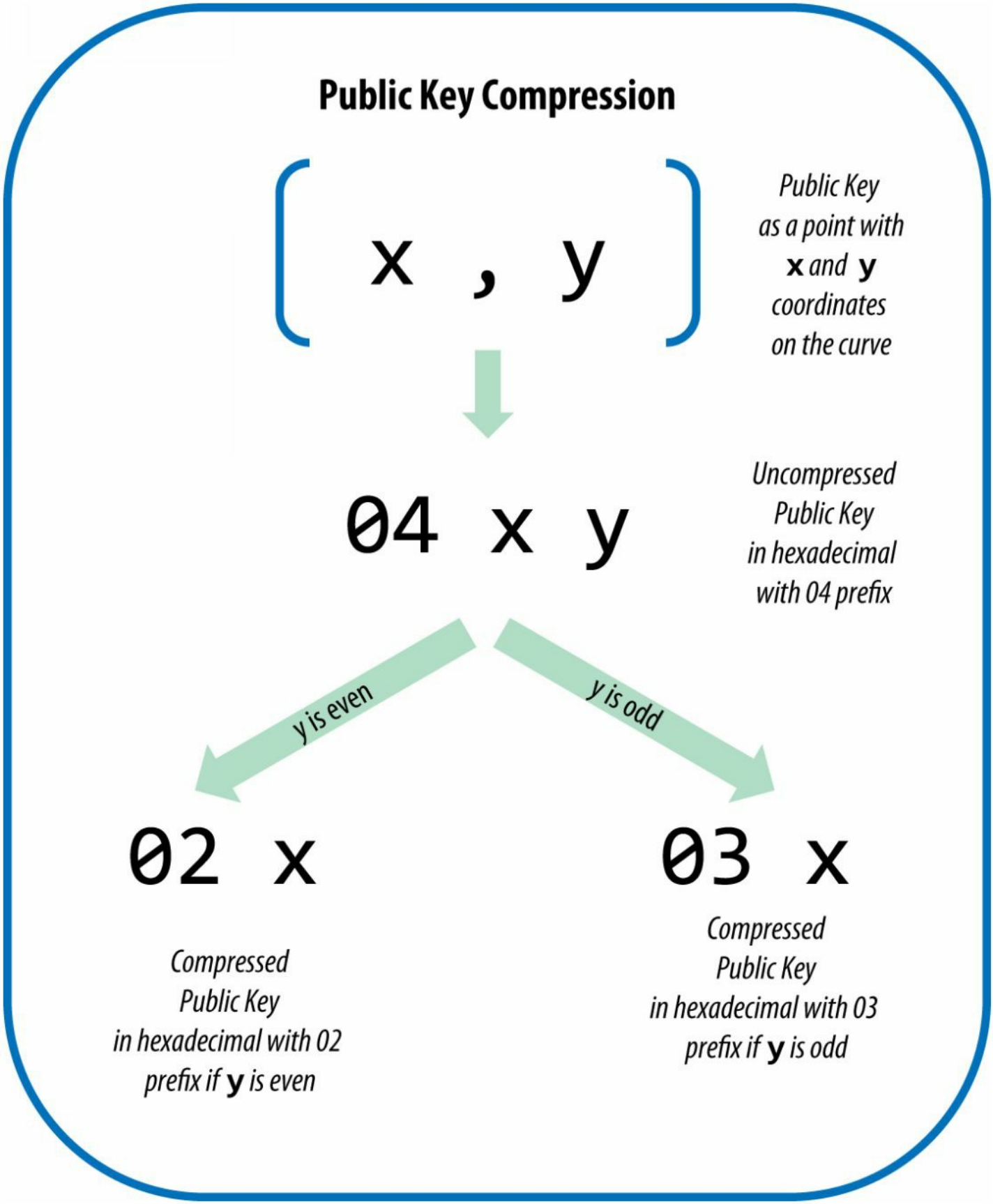


Figura 20. Compressione di

una chiave pubblica

Ecco la stessa chiave pubblica generata in precedenza, mostrata come una chiave pubblica compressa memorizzata in 264 bit (66 cifre esadecimali) con il prefisso 03 che indica che la coordinata y è dispari:

```
K =
03F028892BAD7ED57D2FB57BF33081D5
```

Questa chiave pubblica compressa corrisponde alla stessa chiave privata, il che significa che è generata dalla stessa chiave privata. Tuttavia, sembra diversa dalla chiave pubblica non compressa. Ancora più importante, se convertiamo questa chiave pubblica compressa in un indirizzo bitcoin

usando la funzione di double-hash (RIPEMD160(SHA256(K))) produrremo un indirizzo bitcoin *differente*. Ciò può essere fonte di confusione, perché significa che una singola chiave privata può produrre una chiave pubblica espressa in due formati diversi (compressi e non compressi) che producono due indirizzi bitcoin diversi. Tuttavia, la chiave privata è identica per entrambi gli indirizzi bitcoin.

Le chiavi pubbliche compresse stanno diventando gradualmente l'impostazione predefinita tra i client bitcoin, il che sta avendo un impatto significativo sulla riduzione delle dimensioni delle transazioni e quindi

della blockchain. Tuttavia, non tutti i client supportano ancora chiavi pubbliche compresse. I client più recenti che supportano chiavi pubbliche compresse devono tenere conto delle transazioni da client precedenti che non supportano chiavi pubbliche compresse. Questo è particolarmente importante quando un'applicazione wallet sta importando chiavi private da un'altra applicazione wallet di bitcoin, perché il nuovo portafoglio ha bisogno di scansionare la blockchain per trovare le transazioni corrispondenti a queste chiavi importate. Per quali indirizzi bitcoin deve essere cercato il portafoglio bitcoin? Gli indirizzi

bitcoin saranno prodotti da chiavi pubbliche non compresse o da chiavi pubbliche compresse? Entrambi sono indirizzi bitcoin validi e possono essere firmati con la chiave privata, ma sono indirizzi diversi!

Per risolvere questo problema, quando le chiavi private vengono esportate da un portafoglio, il formato di importazione del wallet utilizzato per rappresentarli viene implementato in modo diverso nel nuovo wallet bitcoin, per indicare che queste chiavi private sono state utilizzate per produrre chiavi pubbliche *compresse* e quindi indirizzi bitcoin compressi. Ciò consente al wallet di importazione di distinguere tra chiavi private

provenienti da portafogli più vecchi o più recenti e cercare sulla blockchain le transazioni con gli indirizzi bitcoin corrispondenti rispettivamente alle chiavi pubbliche non compresse o compresse. Nella prossima sezione diamo un'occhiata a come funziona nel dettaglio.

Chiavi private compresse

Ironia della sorte, il termine "chiave privata compressa" è fuorviante, perché quando una chiave privata viene esportata come compressa WIF, in realtà è un byte *più lungo* di una chiave privata "non compressa". Questo perché ha aggiunto il suffisso 01 (che puoi vedere nella Tabella 4-

4), che significa che proviene da un nuovo portafoglio e dovrebbe essere usato solo per produrre chiavi pubbliche compresse. Le chiavi private non sono compresse e non possono essere compresse. Il termine "chiave privata compressa" significa realmente "chiave privata da cui devono essere derivate le chiavi pubbliche compresse", mentre "chiave privata non compressa" significa in realtà "chiave privata da cui derivare le chiavi pubbliche non compresse". Si dovrebbe solo fare riferimento al formato di esportazione come "WIF-compressed" o "WIF" e non fare riferimento alla chiave privata come "compressa" per evitare ulteriore

confusione.

La Tabella 2 mostra la stessa chiave, codificata nei formati WIF e WIF-compressed.

Tabella 2. Esempio: Stessa chiave, f

Formato	**Chiave privata**
Esadecimale	1E99423A4ED27608
WIF	5J3mBbAH58CpQ3Y
Hex-compressed	1E99423A4ED27608
WIF-compressed	KxFC1jmwwCoACiC

Nota che il formato della chiave privata con compressione esadecimale ha un byte in più alla fine (01 in hex). Mentre il prefisso della versione di codifica Base58 è lo stesso (0x80) per entrambi i formati WIF e WIF-compressed, l'aggiunta di un byte alla fine del numero fa sì che il primo carattere della codifica Base58 passi da 5 a K o L. Pensa a questo come l'equivalente Base58 della differenza di codifica decimale tra il numero 100 e il numero 99. Mentre 100 è una cifra più lunga di 99, ha anche un prefisso di 1 invece di un prefisso di 9. Come la lunghezza cambia, influisce sul prefisso. In Base58, il prefisso 5 cambia in K o L mentre la lunghezza

del numero aumenta di un byte.

Ricorda, questi formati non sono usati in modo intercambiabile. In un nuovo portafoglio che implementa chiavi pubbliche compresse, le chiavi private verranno sempre esportate solo come WIF-compressed (con un prefisso K o L). Se il wallet è un'implementazione meno recente e non utilizza chiavi pubbliche compresse, le chiavi private verranno sempre esportate come WIF (con un prefisso 5). L'obiettivo qui è quello di segnalare al portafoglio l'importazione di queste chiavi private se deve cercare la blockchain per chiavi e indirizzi pubblici compressi o non compressi.

Se un portafoglio bitcoin è in grado di

implementare chiavi pubbliche compresse, utilizzerà quelle in tutte le transazioni. Le chiavi private nel portafoglio verranno utilizzate per ricavare i punti della chiave pubblica sulla curva, che verranno compressi. Le chiavi pubbliche compresse verranno utilizzate per produrre indirizzi bitcoin e quelle saranno utilizzate nelle transazioni. Quando si esportano chiavi private da un nuovo portafoglio che implementa chiavi pubbliche compresse, il file WIF viene modificato, con l'aggiunta di un suffisso di un byte 01 alla chiave privata. La chiave privata codificata Base58Check risultante è chiamata "WIF compressed" e inizia con la

lettera K o L, invece di iniziare con "5" come nel caso delle chiavi codificate WIF (non compresse) dai portafogli più vecchi.

TIP	"Chiavi private compresse" è un termine improprio! Non sono compressi; piuttosto, il formato WIF-compressed significa che dovrebbero essere usati solo per generare chiavi pubbliche compresse e i loro corrispondenti indirizzi bitcoin. Ironia della sorte, una chiave privata codificata "WIF-compressed" è un byte

	più lungo perché ha i suffisso aggiunto 01 per distinguerlo da un "nor compresso".

Implementando le Chiavi e gli Indirizzi in C++

Diamo un'occhiata al processo completo di creazione di un indirizzo bitcoin, da una chiave privata, a una chiave pubblica (un punto sulla curva ellittica), a un indirizzo a doppio hash e infine alla codifica Base58Check. Il codice C++ nell'Esempio 9 mostra il processo passo-passo completo, dalla

chiave privata all'indirizzo bitcoin codificato Base58Check. L'esempio di codice utilizza la libreria libbitcoin introdotta in Client Alternativi, Librerie, e Toolkits per alcune funzioni di supporto.

Esempio 9. Creazione di un indirizzo bitcoin Base58Check-encoded da una chiave privata

```
#include <bitcoin/bitcoin.hpp>


int main()
{
    // Private secret key string as base16
```

```
    bc::ec_secret decoded;
    bc::decode_base16(decoded,

"03810900731 3a5807b2eccc082c8c3fbb9

    bc::wallet::ec_private secret(
        decoded,
bc::wallet::ec_private::mainnet_p2kh);

    // Get public key.
    bc::wallet::ec_public
public_key(secret);
    std::cout << "Public key: " <<
public_key.encoded() << std::endl;

    // Create Bitcoin address.
    // Normally you can use:
    //    bc::wallet::payment_address
payaddr =
    //
public_key.to_payment_address(
```

```
    //
bc::wallet::ec_public::mainnet_p2kh);
    //  const std::string address =
payaddr.encoded();

    // Compute hash of public key for
P2PKH address.
    bc::data_chunk public_key_data;

public_key.to_data(public_key_data);
    const auto hash =
bc::bitcoin_short_hash(public_key_data);

    bc::data_chunk unencoded_address;
    // Reserve 25 bytes
    //   [ version:1  ]
    //   [ hash:20    ]
    //   [ checksum:4 ]
    unencoded_address.reserve(25);
    // Version byte, 0 is normal BTC
address (P2PKH).
```

```
    unencoded_address.push_back(0);
    // Hash data
    bc::extend_data(unencoded_address,
hash);
    // Checksum is computed by hashing
data, and adding 4 bytes from hash.

bc::append_checksum(unencoded_address);
    // Finally we must encode the result
in Bitcoin's base58 encoding.
    assert(unencoded_address.size() ==
25);
    const std::string address =
bc::encode_base58(unencoded_address);

    std::cout << "Address: " << address
<< std::endl;
    return 0;
}
```

Il codice utilizza una chiave privata

predefinita per produrre lo stesso indirizzo bitcoin ogni volta che viene eseguito, come mostrato in Esempio 10.

Esempio 10. Compilando ed eseguendo il codice addr

```
# Compila il codice addr.cpp
$ g++ -o addr addr.cpp -std=c++11
$(pkg-config --cflags --libs libbitcoin)
# Esegui l'eseguibile addr Run the addr
executable
$ ./addr
Public key:
0202a40662421 1f2abbdc68da3df929f938c
Address:
1PRTTaJesdNovgne6Ehcdu1fpEdX7913CK
```

TIP	Il codice in Compilando ed eseguendo il codice addr produce un indirizzo bitcoin (1PRTT...) da una chiave pubblica compressa (vedi Chiavi pubbliche compresse) Se hai usato la chiave pubblica non compressa produrrebbe un indirizzo bitcoin diverso (14K1y ...).

Implementando le Chiavi e gli Indirizzi in Python

La libreria in Python più completa è pybitcointools by Vitalik Buterin. In Esempio 11, abbiamo usato la libreria pybitcointools (importata come "bitcoin") per generare e mostrare chiavi e indirizzi nei vari formati.

Esempio 11. Generazione di chiavi e indirizzi e formattazione tramite la libreria pybitcointools

```
from __future__ import
print_function

import bitcoin

# Generate a random private key
```

```
valid_private_key = False
while not valid_private_key:
    private_key = bitcoin.random_key()
    decoded_private_key =
bitcoin.decode_privkey(private_key,
'hex')
    valid_private_key = 0 <
decoded_private_key < bitcoin.N

print("Private Key (hex) is: ",
private_key)
print("Private Key (decimal) is: ",
decoded_private_key)

# Convert private key to WIF format
wif_encoded_private_key =
bitcoin.encode_privkey(decoded_private_k
'wif')
print("Private Key (WIF) is: ",
wif_encoded_private_key)
```

```
# Add suffix "01" to indicate a
compressed private key
compressed_private_key = private_key
+ '01'
print("Private Key Compressed (hex)
is: ", compressed_private_key)

# Generate a WIF format from the
compressed private key (WIF-
compressed)
wif_compressed_private_key =
bitcoin.encode_privkey(

bitcoin.decode_privkey(compressed_private
'hex'), 'wif_compressed')
print("Private Key (WIF-Compressed)
is: ", wif_compressed_private_key)

# Multiply the EC generator point G
with the private key to get a public key
point
```

```
public_key =
bitcoin.fast_multiply(bitcoin.G,
decoded_private_key)
print("Public Key (x,y) coordinates is:",
public_key)

# Encode as hex, prefix 04
hex_encoded_public_key =
bitcoin.encode_pubkey(public_key,
'hex')
print("Public Key (hex) is:",
hex_encoded_public_key)

# Compress public key, adjust prefix
depending on whether y is even or odd
(public_key_x, public_key_y) =
public_key
compressed_prefix = '02' if
(public_key_y % 2) == 0 else '03'
hex_compressed_public_key =
compressed_prefix +
```

```
(bitcoin.encode(public_key_x,
16).zfill(64))
print("Compressed Public Key (hex)
is:", hex_compressed_public_key)

# Generate bitcoin address from public
key
print("Bitcoin Address (b58check) is:",
bitcoin.pubkey_to_address(public_key))

# Generate compressed bitcoin address
from compressed public key
print("Compressed Bitcoin Address
(b58check) is:",

bitcoin.pubkey_to_address(hex_compresse
```

Eseguendo key-to-address-ecc-example.py mostra l'output ottenuto eseguendo questo codice.

Esempio 12. Eseguendo key-to-address-ecc-example.py

```
$ python key-to-address-ecc-example.py
Private Key (hex) is:
 3aba4162c7251c891207b747840551a71939b
Private Key (decimal) is:
 26563230048437957592232553826663696
Private Key (WIF) is:
 5JG9hT3beGTJuUAmCQEmNaxAuMacCTfXu
Private Key Compressed (hex) is:
 3aba4162c7251c891207b747840551a71939b
Private Key (WIF-Compressed) is:
 KyBsPXxTuVD82av65KZkrGrWi5qLMah5Sd
Public Key (x,y) coordinates is:
 (41637322786646325214887832269588396
 16388935128781238405526710466724741
Public Key (hex) is:
 045c0de3b9c8ab18dd04e3511243ec2952002
243bcefdd4347074d44bd7356d6a53c495737
Compressed Public Key (hex) is:
```

```
025c0de3b9c8ab18dd04e3511243ec2952002
Bitcoin Address (b58check) is:
1thMirt546nngXqyPEz532S8fLwbozud8
Compressed Bitcoin Address (b58check) is:
14cxpo3MBCYYWCgF74SWTdcmxipnGUsPv
```

Esempio 13 è un altro esempio, utilizzando la libreria Python ECDSA per la matematica della curva ellittica senza utilizzare alcuna libreria specializzata di bitcoin .

Esempio 13. Uno script che dimostra la matematica a curve ellittiche usata per le chiavi bitcoin

```
import ecdsa
```

```
import os

# secp256k1, http://www.oid-
info.com/get/1.3.132.0.10
_p =
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
_r =
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE
_b =
0x0000000000000000000000000000000
_a =
0x0000000000000000000000000000000
_Gx =
0x79BE667EF9DCBBAC55A06295CE870
_Gy =
0x483ada7726a3c4655da4fbfc0e1108a8fd
curve_secp256k1 =
ecdsa.ellipticcurve.CurveFp(_p, _a, _b)
generator_secp256k1 =
ecdsa.ellipticcurve.Point(curve_secp256k1
_Gx, _Gy, _r)
```

```
oid_secp256k1 = (1, 3, 132, 0, 10)
SECP256k1 =
ecdsa.curves.Curve("SECP256k1",
curve_secp256k1,

generator_secp256k1, oid_secp256k1)
ec_order = _r

curve = curve_secp256k1
generator = generator_secp256k1



def random_secret():
    convert_to_int = lambda array:
int("".join(array).encode("hex"), 16)

    # Collect 256 bits of random data
from the OS's cryptographically secure
    # random number generator
    byte_array = os.urandom(32)
```

```
    return convert_to_int(byte_array)


def get_point_pubkey(point):
    if (point.y() % 2) == 1:
        key = '03' + '%064x' % point.x()
    else:
        key = '02' + '%064x' % point.x()
    return key.decode('hex')


def
get_point_pubkey_uncompressed(point):
    key = ('04' +
           '%064x' % point.x() +
           '%064x' % point.y())
    return key.decode('hex')


# Generate a new private key.
secret = random_secret()
```

```
print("Secret: ", secret)

# Get the public key point.
point = secret * generator
print("EC point:", point)

print("BTC public key:",
get_point_pubkey(point).encode("hex"))

# Given the point (x, y) we can create
the object using:
point1 =
ecdsa.ellipticcurve.Point(curve,
point.x(), point.y(), ec_order)
assert(point1 == point)
```

L’esempio 4-9 mostra l’output prodotto dall’esecuzione di questo script.

	L’esempio s

utilizza os.uran
che riflette
generatore di nu
casuali
crittograficamente
sicuro (CSP
fornito dal sis
operativo sottost
Nel caso di un sis
operativo simile
UNIX come Linu
basa su /dev/uran
e nel caso

NOTA	Windows, es CryptGenRandom Se non viene tr una fonte generazione casualità, restituito NotImplementedE Il generatore di nu casuali utilizzato a scopo dimostr non è appropriat la generazione

	chiavi bitcoin qualità di produ poiché non implementato sufficiente sicurez

Esempio 14. Installando la libreria Python ECDSA ed eseguendo lo script ec_math.py

```
$ # Installa PIP, il package manager
di Python
```

```
$ sudo apt-get install python-pip
```

```
$ # Installa la libreria Python ECDSA
$ sudo pip install ecdsa
$ # Esegui lo script
$ python ec-math.py
Secret:
38090835015954358862481132628887 4
EC point:
(700488535318671794898577504976069
105262206478686743191060800263479
BTC public key:
029ade3effb0a67d5c8609850d797366af42
```

Chiavi avanzate e Indirizzi

Nelle sezioni seguenti osserveremo le forme avanzate di chiavi e indirizzi quali, chiavi private criptate, indirizzi script e multifirma, vanity address e paper wallet.

Encrypted Private Keys (BIP-38)

Le chiavi private devono rimanere segrete. Il bisogno di riservatezza delle chiavi private è una verità ovvia che è abbastanza difficile da ottenere nella pratica, perché è in conflitto con l'altrettanto importante obiettivo di disponibilità. Mantenere privata la chiave privata è molto più difficile quando è necessario archiviare i backup della chiave privata per evitare di perderla. Una chiave privata memorizzata in un portafoglio crittografato da una password potrebbe essere protetta, ma è necessario eseguire il backup di tale

portafoglio. A volte, gli utenti devono spostare le chiavi da un portafoglio all'altro per aggiornare o sostituire il software del portafoglio, ad esempio.

Anche i backup delle chiavi private possono essere archiviati su carta (vedi I Paper Wallet) o su supporti di memorizzazione esterni, ad esempio un'unità flash USB. Ma cosa succede se il backup stesso viene rubato o perso? Questi obiettivi di sicurezza in conflitto hanno portato all'introduzione di uno standard portatile e conveniente per crittografare le chiavi private in un modo che può essere compreso da diversi portafogli e client bitcoin, standardizzati da Bitcoin Improvement Proposal 38 o BIP0038 (vedi Bitcoin

Improvement Proposals).

BIP0038 propone uno standard comune per crittografare le chiavi private con una passphrase e codificarle con Base58Check in modo che possano essere archiviate in modo sicuro su supporti di backup, trasportate in modo sicuro tra i portafogli o conservate in qualsiasi altra condizione in cui la chiave potrebbe essere esposta. Lo standard per la crittografia utilizza il Advanced Encryption Standard (AES), è uno standard stabilito dal National Institute of Standards and Technology (NIST) e ampiamente utilizzato nelle implementazioni di crittografia dei dati per scopi commerciali e applicazioni

militari.

Uno schema di crittografia BIP0038 accetta come input una chiave privata bitcoin, solitamente codificata nel Wallet Import Format (WIF), come una stringa Base58Check con un prefisso di "5". Inoltre, lo schema di crittografia BIP0038 accetta una passphrase, una lunga password, in genere composta da più parole o da una stringa complessa di caratteri alfanumerici. Il risultato dello schema di crittografia BIP0038 è una chiave privata crittografata con codifica Base58Check che inizia con il prefisso 6P. Se vedi una chiave che inizia con 6P, significa che è crittografata e richiede una passphrase per

convertirla (decodificarla) in una chiave privata formattata WIF (prefisso 5) che può essere utilizzata in qualsiasi portafoglio. Molte applicazioni di wallet ora riconoscono le chiavi private crittografate BIP0038 e richiederanno all'utente una passphrase per decrittografare e importare la chiave. Le applicazioni di terze parti, come l'utilissimo Bit Address (scheda Dettagli portafogli) basato su browser, possono essere utilizzate per decrittografare le chiavi BIP0038.

Il caso d'uso più comune per le chiavi crittografate BIP0038 è il paper wallet che può essere utilizzato per eseguire il backup di chiavi private su un pezzo

di carta. Finché l'utente seleziona una passphrase forte, un portafoglio cartaceo con chiavi private crittografate BIP0038 è incredibilmente sicuro e un ottimo modo per creare un archivio bitcoin offline (noto anche come "cold storage").

Prova le chiavi criptate nella Esempio di chiave privata codificata BIP0038 usando bitaddress.org per vedere come ottenere la chiave decriptata inserendo la passphrase.

Tabella 5. Esempio di chiave privat

Private Key (WIF)	5J3mBbAH58CpQ3Y

Passphrase	MyTestPassphrase
Encrypted Key (BIP-38)	6PRTHL6mWa48xSop

Pay-to-Script Hash (P2SH) e Indirizzi Multi-Sig

Come sappiamo, gli indirizzi bitcoin tradizionali iniziano con il numero "1" e derivano dalla chiave pubblica, che deriva dalla chiave privata. Sebbene chiunque possa inviare bitcoin a un indirizzo "1", quel bitcoin può essere speso solo presentando la corrispondente firma della chiave privata e l'hash della chiave pubblica.

Gli indirizzi di Bitcoin che iniziano

con il numero "3" sono indirizzi di hash pay-to-script (P2SH), a volte chiamati erroneamente indirizzi multi-firma o multi-sig. Designano il beneficiario di una transazione bitcoin come hash di uno script, invece del proprietario di una chiave pubblica. La funzionalità è stata introdotta a gennaio 2012 con Bitcoin Improvement Proposal 16, o BIP0016 (vedi Bitcoin Improvement Proposals), ed è ampiamente adottata perché offre l'opportunità di aggiungere funzionalità all'indirizzo stesso. A differenza delle transazioni che "inviano" fondi a tradizionali indirizzi "1" bitcoin, anche noti come pay-to-public-key-hash (P2PKH), i fondi

inviati a indirizzi che iniziano per "3", richiedono qualcosa di più della presentazione di un hash della chiave pubblica e una firma della chiave privata come prova della proprietà. I requisiti sono designati al momento della creazione dell'indirizzo, all'interno dello script, e tutti gli input a questo indirizzo saranno gravati dagli stessi requisiti.

Un indirizzo di hash pay-to-script viene creato da uno script di transazione, che definisce chi può spendere un output di transazione (per maggiori dettagli, vedi Pay-to-Script-Hash (P2SH)). La codifica di un indirizzo hash pay-to-script implica l'uso della stessa funzione double-hash

utilizzata durante la creazione di un indirizzo bitcoin, applicata solo sullo script anziché sulla chiave pubblica:

```
script hash = RIPEMD160(SHA256(script))
```

Lo "script hash" risultante è codificato con Base58Check con un prefisso di versione 5, che risulta in un indirizzo codificato che inizia con 3. Un esempio di indirizzo P2SH è +3F6i6kwkevjR7AsAd4te2YB2zZyASE , che può essere derivato utilizzando i comandi di Bitcoin Explorer come segue:

```
$ echo \
'DUP HASH160
[89abcdefabbaabbaabbaabbaabbaabbaabbaabba
EQUALVERIFY CHECKSIG' > script
```

```
$ bx script-encode < script | bx sha256 | bx
ripemd160 \
| bx base58check-encode --version 5
3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM
```

TIP	P2SH non è necessariamente lo stesso di una transazione standard multi-firma. Un indirizzo P2SH *molto spesso* rappresenta uno script multi-firma, ma potrebbe anche

	rappresentare unc script che codifica altri tipi di transazioni.

Indirizzi Multi-firma e P2SH

Attualmente, l'implementazione più comune della funzione P2SH è lo script di indirizzo multi-firma. Come suggerisce il nome, lo script sottostante richiede più di una firma per dimostrare la proprietà e quindi spendere fondi. La funzionalità multi-signature bitcoin è progettata per richiedere le firme M (anche note come "soglia") da un totale di N chiavi, noto come M-of-N multi-sig,

dove M è uguale o minore di N. Ad esempio, Bob, il proprietario del coffee shop che abbiamo presentato nell'Introduzione potrebbe utilizzare un indirizzo multi-firma che richiede 1-di-2 firme da una chiave a lui appartenente e una chiave appartenente al coniuge, garantendo che entrambi possano firmare per passare una transazione uscita bloccata a questo indirizzo. Questo sarebbe simile a un "conto comune" come implementato nel sistema bancario tradizionale in cui entrambi i coniugi possono spendere con una singola firma. O Gopesh, il web designer pagato da Bob per creare un sito web, potrebbe avere un indirizzo multi-firma 2-of-3

per la sua attività che garantisce che nessun denaro possa essere speso a meno che almeno due dei partner commerciali firmino una transazione.

Esploreremo come creare transazioni che spendono fondi da indirizzi P2SH (e multi-signature) in Transazioni.

Vanity Address

Gli indirizzi di vanity sono indirizzi bitcoin validi che contengono messaggi leggibili dall'uomo. Ad esempio, 1LoveBPzzD72PUXLzCkYAtGFYmK5v è un indirizzo valido che contiene le lettere che formano la parola "Love" come le prime quattro lettere Base-58. Gli indirizzi "di vanità" richiedono la

generazione e il test di miliardi di chiavi private candidate, finché non si ottiene un indirizzo bitcoin con lo schema desiderato. Sebbene vi siano alcune ottimizzazioni nell'algoritmo di generazione vanità, il processo consiste essenzialmente nel selezionare una chiave privata a caso, derivando la chiave pubblica, derivando l'indirizzo bitcoin e verificando se corrisponde al modello di vanità desiderato, ripetendo miliardi di volte fino a che non verrà trovata una corrispondenza.

Una volta trovato un vanity address che corrisponde al modello desiderato, la chiave privata da cui è stata derivata può essere utilizzata dal

proprietario per spendere bitcoin esattamente nello stesso modo di qualsiasi altro indirizzo. I vanity address non sono meno o più sicuri di qualsiasi altro indirizzo. Essi dipendono dalla stessa Elliptic Curve Cryptography (ECC) e Secure Hash Algorithm (SHA) come qualsiasi altro indirizzo. Non è più semplice trovare la chiave privata di un vanity address quanto non lo sia di qualsiasi altro indirizzo.

I n Introduzione, abbiamo introdotto Eugenia, direttrice di un'associazione no-profit che opera nelle Filippine. Mettiamo che Eugenia sta organizzando una raccolta fondi in bitcoin e voglia usare un bitcoin vanity

address per pubblicizzare la raccolta fondi. Eugenia creerà un vanity address che inizia con "1Kids" per promuovere la raccolta fondii della no-profit che si occupa di bambini bisognosi. Vediamo come questo vanity address sarà generato e come la no-profit di Eugenia si dovrà comportare riguardo alla sicurezza.

Generare vanity address

È importante rendersi conto che un indirizzo bitcoin è semplicemente un numero rappresentato da simboli nell'alfabeto Base58. La ricerca di un modello come "1Kids" può essere vista come ricerca di un indirizzo nell'intervallo compreso tra

1Kids111111111111111111111111
a
1Kidszzzzzzzzzzzzzzzzzzzzzzzzzz.
Ci sono circa 58_{29} (circa $1{,}4 * 10_{51}$) indirizzi in quell'intervallo, tutti a partire da "1Kids". La gamma di vanity address che iniziano con "1Kids" mostra l'intervallo di indirizzi che hanno il prefisso 1Kids.

Tabella 6. La gamma di vanity a iniziano con "1Kids"

Da	1Kids111111111111111111
	1Kids111111111111111111
	1Kids111111111111111111

	...
A	1Kidszzzzzzzzzzzzzzzzzzzzzzzz

Diamo un'occhiata al pattern "1Kids" come numero e vediamo con quale frequenza possiamo trovare questo pattern in un indirizzo bitcoin (vedi La frequenza di un vanity pattern(1KidsCharity) e il tempo-di-ricerca su un PC fisso)). Un PC desktop medio, senza hardware specializzato, può cercare circa 100.000 chiavi al secondo.

Tabella 7. La frequenza di u (1KidsCharity) e il tempo-di-ricerca

Lunghezza	Pattern	Frequ

1	1K	1 su chiavi
2	1Ki	1 su 3,
3	1Kid	1 195,0(
4	1Kids	1 su millio
5	1KidsC	1 su milion
6	1KidsCh	1 su miliar

7	1KidsCha	1 su mila miliar
8	1KidsChar	1 su mila miliar
9	1KidsChari	1 su milion miliar
10	1KidsCharit	1 su milion miliar
11	1KidsCharity	1 su mila

		milion miliar

Come puoi vedere, Eugenia non creerà presto il vanity address "1KidsCharity", anche se ha accesso a diverse migliaia di computer. Ogni carattere aggiuntivo aumenta la difficoltà di un fattore di 58. I pattern con più di sette caratteri sono generalmente trovati da hardware specializzato, come desktop personalizzati con unità di elaborazione grafica multiple (GPU). Si tratta spesso di "rig" di mining bitcoin che non sono più redditizi per il bitcoin mining, ma possono essere utilizzati per trovare gli indirizzi di

vanità. Le ricerche di vanità su sistemi GPU sono più veloci di molti ordini di grandezza rispetto a una CPU generica.

Un altro modo per trovare un vanity address è esternalizzare il lavoro a un pool di vanity miner, come ad esempio il pool su Vanity Pool. Un pool è un servizio che consente a chi possiede hardware GPU di guadagnare bitcoin alla ricerca di vanity address per gli altri. Per un piccolo pagamento (0,01 bitcoin o circa $ 5 al momento della stesura di questo libro), Eugenia può esternalizzare la ricerca di un vanity address con sette caratteri e ottenere risultati in poche ore invece di dover eseguire una ricerca della CPU per mesi.

Generare un vanity address è un esercizio di forza bruta: prova una chiave casuale, controlla l'indirizzo risultante per vedere se corrisponde al modello desiderato, e continua a ripetere fino a quando non verrà trovato. Miner di vanity address mostra un esempio di "vanity miner", un programma progettato per trovare gli indirizzi di vanità, scritti in C ++. L'esempio usa la libreria libbitcoin, che abbiamo introdotto in Client Alternativi, Librerie, e Toolkits.

Esempio 14. Miner di vanity address

```
#include <random>
```

```
#include <bitcoin/bitcoin.hpp>

// The string we are searching for
const std::string search = "1kid";

// Generate a random secret key. A
random 32 bytes.
bc::ec_secret
random_secret(std::default_random_engine
engine);
// Extract the Bitcoin address from an
EC secret.
std::string bitcoin_address(const
bc::ec_secret& secret);
// Case insensitive comparison with the
search string.
bool match_found(const std::string&
address);

int main()
{
```

```
    // random_device on Linux uses
"/dev/urandom"
    // CAUTION: Depending on
implementation this RNG may not be
secure enough!
    // Do not use vanity keys generated
by this example in production
    std::random_device random;
    std::default_random_engine
engine(random());

    // Loop continuously...
    while (true)
    {
        // Generate a random secret.
        bc::ec_secret secret =
random_secret(engine);
        // Get the address.
        std::string address =
bitcoin_address(secret);
        // Does it match our search string?
```

```
(1kid)
        if (match_found(address))
        {
            // Success!
            std::cout << "Found vanity
address! " << address << std::endl;
            std::cout << "Secret: " <<
bc::encode_base16(secret) <<
std::endl;
            return 0;
        }
    }
    // Should never reach here!
    return 0;
}

bc::ec_secret
random_secret(std::default_random_engine
engine)
{
    // Create new secret...
```

```
    bc::ec_secret secret;
    // Iterate through every byte setting a
random value...
    for (uint8_t& byte: secret)
        byte = engine() %
std::numeric_limits<uint8_t>::max();
    // Return result.
    return secret;
}

std::string bitcoin_address(const
bc::ec_secret& secret)
{
    // Convert secret to payment address
    bc::wallet::ec_private
private_key(secret);
    bc::wallet::payment_address
payaddr(private_key);
    // Return encoded form.
    return payaddr.encoded();
}
```

```
bool match_found(const std::string&
address)
{
    auto addr_it = address.begin();
    // Loop through the search string
comparing it to the lower case
    // character of the supplied address.
    for (auto it = search.begin(); it !=
search.end(); ++it, ++addr_it)
        if (*it != std::tolower(*addr_it))
            return false;
    // Reached end of search string, so
address matches.
    return true;
}
```

	Compilare e eseguire l'esempio di vanity-miner usa std::random_device. A

NOTA	seconda dell'implementazione, esso può riflettere un generatore di numeri casuali cittograficamente sicuro (CSRNG) fornito dal sottostante sistema operativo. Nel caso di un sistema operativo UNIX-like come LInux, attinge da /dev/urandom. Mentre il generatore di numeri casuali usato qui si limita a scopi dimostrativi, non è adeguato a generare

	chiavi bitcoin di production-quality ir quanto nor implementate cor sicurezza sufficiente.

Il codice d'esempio deve essere compilato usando un compilatore C e linkato alla libreria libbitcoin (che deve essere installata sul sistema). Per eseguire il codice d'esempio, lancia l'eseguibile vanity-miner++ senza parametri (vedi Compilare e eseguire l'esempio di vanity-miner) ed il programma tenterà di trovare un vanity address che inizia con "1kid".

Esempio 15. Compilare e

eseguire l'esempio di vanity-miner

```
$ # Compila il codice con g++
```

```
$ g++ -o vanity-miner vanity-miner.cpp
$(pkg-config --cflags --libs libbitcoin)
$ # Esegui l'esempio
$ ./vanity-miner
Found vanity address!
1KiDzkG4MxmovZryZRj8tK81oQRhbZ46
Secret:
57cc268a05f83a23ac9d930bc8565bac4e2
$ # Eseguilo nuovamente per ottenere
un risultato differente
$ ./vanity-miner
Found vanity address!
1Kidxr3wsmMzzouwXibKfwTYs5Pau8TUF
Secret:
7f65bbbbe6d8caae74a0c6a0d2d7b5c6663d
```

```
# Utilizza "time" per conoscere il
tempo necessario per trovare un
risultato
$ time ./vanity-miner
Found vanity address!
1KidPWhKgGRQWD5PP5TAnGfDyfWp5y
Secret:
2a802e7a53d8aa237cd059377b616d2bfcfa

real 0m8.868s
user 0m8.828s
sys  0m0.035s
```

Il codice d'esempio impiegherà pochi secondi per trovare una corrispondenza per il pattern "kid", come possiamo vedere quando usiamo il comando Unix time per misurare il tempo d'esecuzione. Puoi cambiare il pattern di ricerca nei codice sorgente e

vedere quanto più tempo impiegano dei pattern di quattro o cinque caratteri!

Sicurezza dei vanity address

I vanity address possono essere usati per migliorare e per sconfiggere le misure di sicurezza; sono veramente un'arma a doppio taglio. Utilizzato per migliorare la sicurezza, un indirizzo distintivo rende più difficile per gli avversari sostituire il proprio indirizzo e ingannare i tuoi clienti nel pagarli al posto tuo. Sfortunatamente, i vanity address permettono anche a chiunque di creare un indirizzo che assomigli a qualsiasi indirizzo casuale, o anche a un altro indirizzo di

vanità, *ingannando* così i tuoi clienti.

Eugenia potrebbe pubblicare un indirizzo generato casualmente (e.g., 1J7mdg5rbQyUHENYdx39WVWK7fsLp a cui altri utenti possono inviare le loro donazioni. Oppure, potrebbe generare un vanity address che inizi con 1Kids, per renderlo più riconoscibile.

In entrambi i casi, uno dei rischi dell'utilizzo di un singolo indirizzo fisso (piuttosto che un indirizzo dinamico separato per donatore) è che un ladro potrebbe essere in grado di infiltrarsi nel tuo sito Web e sostituirlo con il proprio indirizzo, deviando così le donazioni a se stesso. Se hai pubblicizzato il tuo indirizzo di donazione in un certo numero di luoghi

diversi, i tuoi utenti possono ispezionare visivamente l'indirizzo prima di effettuare un pagamento per assicurarsi che sia lo stesso che hanno visto sul tuo sito web, sulla tua email e sul tuo volantino. Nel caso di un indirizzo casuale come 1J7mdg5rbQyUHENYdx39WVWK7fsL l'utente medio controllerà forse i primi caratteri "1J7mdg" e si accontenterà che l'indirizzo corrisponda. Usando un generatore di indirizzi di vanità, qualcuno con l'intento di rubare sostituendo un indirizzo dall'aspetto simile può generare rapidamente indirizzi che corrispondono ai primi caratteri, come mostrato in Generando un vanity address che sia simile a

un’indirizzo casuale.

Tabella 8. Generando un vanity a un’indirizzo casuale

Indirizzo Casuale Originario	1J7mdg5rbQyUHENYd
Vanity (4-character match)	1J7md1QqU4LpctBetH
Vanity (5-character match)	1J7mdgYqyNd4ya3UE
Vanity (6-character	1J7mdg5WxGENmwyJ

match)

Quindi un vanity address aumenta la sicurezza? Se Eugenia genera l'indirizzo vanity 1Kids33q44erFfpeXrmDSz7zEqG2FesZ è probabile che gli utenti guardino la parola del pattern di vanità e *alcuni caratteri oltre*, ad esempio notando la parte "1Kids33" dell'indirizzo. Ciò costringerebbe un attaccante a generare un indirizzo di vanità corrispondente ad almeno sei caratteri (due in più), spendendo uno sforzo che è 3,364 volte (58 x 58) più alto dello sforzo che Eugenia ha speso per la sua vanità di quattro caratteri. Essenzialmente, lo sforzo che Eugenia

spende (o paga una "vanity pool") "spinge" l'attaccante a dover produrre un vanity pattern più lungo. Se Eugenia paga una "vanity pool" per generare un indirizzo di vanità di 8 caratteri, l'attaccante verrebbe spinto fino a 10 caratteri, che è impossibile da usare su un personal computer e costoso anche con un impianto di vanity mining o una vanity pool. Ciò che è conveniente per Eugenia diventa inaccessibile per l'aggressore, specialmente se il bottino ottenuto dalla frode non è abbastanza alto da coprire il costo della generazione dell'indirizzo di vanità.

I Paper Wallet

I portafogli di carta sono chiavi private bitcoin stampate su carta. Spesso il portafoglio di carta include anche il corrispondente indirizzo bitcoin per comodità, ma questo non è necessario perché può essere derivato dalla chiave privata. I portafogli di carta sono un modo molto efficace per creare backup o archiviare bitcoin offline, nota anche come "cold storage". Come meccanismo di backup, un portafoglio di carta può fornire sicurezza contro la perdita della chiave a causa di un incidente informatico come un guasto del disco rigido, il furto o la cancellazione accidentale. Come meccanismo di "conservazione a freddo", se le chiavi

del portafoglio cartaceo vengono generate offline e mai archiviate su un sistema informatico, sono molto più sicure contro hacker, keylogger e altre minacce informatiche online.

I paper wallet sono disponibili in molte forme, dimensioni e design, ma essenzialmente sono solo una chiave e un indirizzo stampati su carta. La forma più semplice di paper wallet—un fogliostampato con l'indirizzo bitcoin e la chiave privata. mostra la forma più semplice di un paper wallet.

Tabella 9. La forma più semplice di privata.

Public address

1424C2F4bC9JidNjjTUZCbUxv6Sa

I portafogli di carta possono essere generati facilmente utilizzando uno strumento come il generatore JavaScript lato client su *bitaddress.org*. Questa pagina contiene tutto il codice necessario per generare chiavi e portafogli di carta, anche se completamente disconnessi da Internet. Per usarlo, salva la pagina HTML sull'unità locale o su un'unità flash USB esterna. Disconnettiti da Internet e apri il file in un browser. Ancora meglio, avvia il tuo computer usando un sistema operativo incontaminato, come un sistema

operativo Linux avviabile da CD-ROM. Qualsiasi tasto generato con questo strumento mentre offline può essere stampato su una stampante locale tramite un cavo USB (non in modalità wireless), creando così portafogli di carta le cui chiavi esistono solo sulla carta e non sono mai state memorizzate su alcun sistema online. Metti questi paper wallet in una cassaforte a prova di fuoco e "invia" bitcoin al loro indirizzo bitcoin, per implementare una soluzione di "cold storage" semplice ma altamente efficace.

Un esempio di un paper wallet semplice da bitaddress.org mostra un portafoglio di carta generato dal sito

bitaddress.org.

Figura 20. Un esempio di un paper wallet semplice da bitaddress.org

Lo svantaggio del semplice sistema a paper wallet e' che le chiavi stampate sono vulnerabili al furto. Un ladro che e' in grado di avere accesso al foglio può sia rubarlo che fotografare le

chiavi e prendere il controllo dei bitcoin bloccati con queste chiavi. Un sistema di memorizzazione paper wallet ancora più sofisticato usa chiavi private criptate BIP-38. Le chiavi stampate sul paper wallet sono protette da una passphrase che il proprietario ha memorizzato. Senza la passphrase, le chiavi criptate sono inutilizzabili. Tuttavia, sono ancora superiori a un portafoglio protetto da passphrase perché le chiavi non sono mai state online e devono essere recuperate fisicamente da una cassaforte o da un'altra memoria fisicamente protetta. Un esempio di un paper wallet protetto da password dal sito bitaddress.org . La passphrase è

"test." mostra un portafoglio di carta con una chiave privata crittografata (BIP-38) creata sul sito bitaddress.org.

Figura 21. Un esempio di un paper wallet protetto da password dal sito bitaddress.org . La passphrase è "test."

Sebbene
possa
depositare for
in
portafoglio
carta più vol
dovresti
prelevare tutt
fondi una vo
sola, spenden
tutto. Ques
perché
processo
sblocco
spendere for
alcuni
portafogli
potrebbero

ATTENZIONE

generare
change addre
(n.d.r. indiriz
di
cambiamento)
se si spen
meno
dell'intero
importo. Inolt
se il comput
che usi p
firmare
transazione
compromesso,
rischi
esporre
chiave priva
Spendendo

	l’intero saldo un portafo solo una vol si riduce rischio compromissio della chiave. hai bisogno sc di una picc quantità, invia fondi rimane a un nuo portafoglio carta ne stessa transazione.

Esistono paper wallet con dimensioni

e forme differenti e con diverse funzionalità. Alcuni sono fatti per essere dati come regalo e hanno temi particolari come per esempio temi di Natale e Capodanno. Altri sono stati progettati per essere conservati in una cassetta di sicurezza o cassaforte con la chiave privata nascosta in qualche maniera, sia con un etichetta opaca grattabile, o piegata e sigillata con un foglio di alluminio adesivo a prova di apertura inattesa. Le Figure 21 e 22 mostrano vari esempi di portafogli di carta con funzioni di sicurezza e di backup.

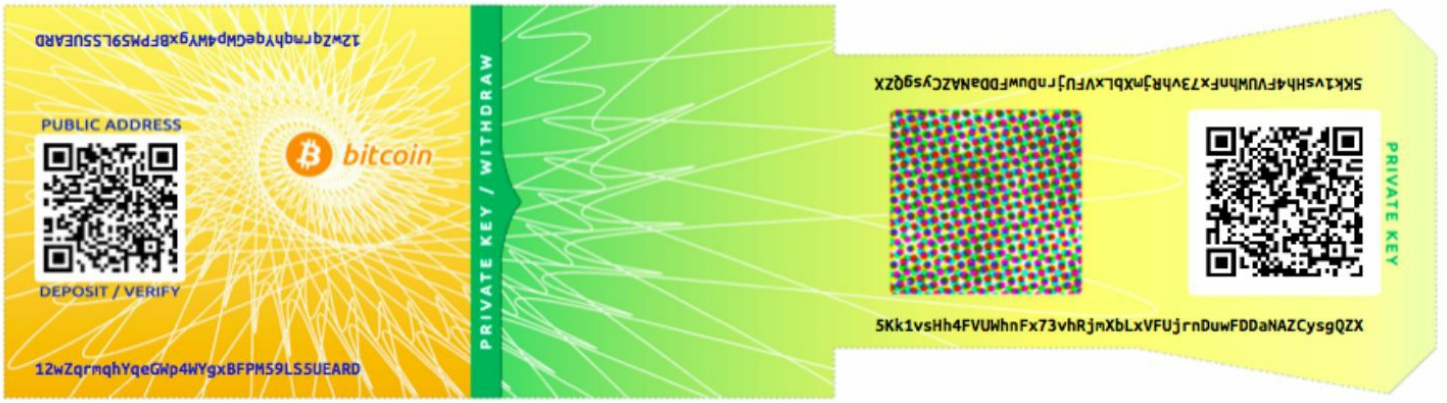


Figura 21. Un esempio di un paper wallet da bitcoinpaperwallet.com con la private key stampata su una parte pieghevole.

Figura 22. Il paper wallet di bitcoinpaperwallet.com con la chiave privata nascosta.

Altri modelli presentano copie aggiuntive della chiave e dell'indirizzo, sotto forma di matrici staccabili simili a matrici di biglietti, che consentono di archiviare più copie per proteggersi da incendi, inondazioni o altri disastri naturali.

Figura 23. Un esempio di un

paper wallet con copie aggiuntive di un backup "campione".

Portafogli

Il termine "wallet" viene usato per descrivere diverse cose in bitcoin.

Ad un livello avanzato, il portafoglio è un applicazione che funge da interfaccia primaria per l'utente. Il portafoglio controlla l'accesso ai fondi dell'utente, gestendo le chiavi e gli indirizzi, tenendo traccia del bilancio, e creando e firmando le transazioni.

Più precisamente, dal punto di vista del programmatore, la parola "wallet" significa struttura di dati usata per conservare e gestire le chiavi dell'utente.

In questo capitolo vedremo un secondo significato, dove i portafogli sono contenitori di chiavi private, implementati solitamente come file strutturati o semplici database.

Panoramica sulla tecnologia Wallet

In questa sezione sintetizzeremo le varie tecnologie usate per costruire portafogli bitcoin sicuri, flessibili e user-friendly.

Un errore comune è quello di credere che un wallet bitcoin contiene bitcoin. Di fatto, il wallet contiene solo le chiavi. I "coins" sono registrati nella blockchain di bitcoin. Gli utenti

controllano le monete all'interno della rete firmando le transazioni con le chiavi presenti nei loro wallet. Nel senso che, un portafoglio bitcoin è una *keychain*.

TIP	I portafogli bitcoin contengono le chiavi non i coins. Ogni utente ha un portafoglio contenente le chiavi. I portafogli sono vere e proprie *keychains* contenenti un paio di chiavi, pubblica e privata (vedi Chiavi Private e Pubbliche) Gli utenti firmano le transazioni con le loro

	chiavi, potendo provare la proprietà dell'outpu della transazione (le loro monete). I *coins* sono archiviati nella blockchain sotto forma di output di transazione (spesso con il nome di vout o txout).

Esistono due tipologie principali di wallets, distinti dal fatto che le chiavi che contengono abbiano una relazione tra di loro o meno.

La prima tipologia è detta *nondeterministic wallet*, nella quale ogni chiave è generata, in maniera

indipendente dall'altra, da un numero casuale. Le chiavi non hanno nessuna relazione fra di loro. Questo wallet è conosciuto anche come portafoglio JBOK dalla frase "Just a Bunch Of Keys.", "Soltanto un mazzo di chiavi."

Il secondo tipo di wallet è detto *deterministic wallet*. Tutte le chiavi di questo portafoglio hanno una relazione fra di loro e possono essere generate nuovamente avendo a disposizione il seed originale. Esistono numerosi metodi di derivazione delle chiavi *key derivation* adoperati nei portafogli deterministici. Il metodo di derivazione più comune utilizza una struttura ad albero *tree* conosciuta come *hierarchical deterministic* o *HD*

wallet.

I *deterministic wallets* vengono inizializzati da un seed. Per renderlo facile da utilizzare, i seeds vendono codificati in parole Inglesi, conosciute come *mnemonic code words*.

Nelle prossime sezioni andremo ad introdurre ad un livello complesso ognuna di queste tecnologie.

Wallet (Casuali) Nondeterministici

Nel primo portafoglio bitcoin (ora chiamato Bitcoin Core), i portafogli erano raccolte di chiavi private generate casualmente. Ad esempio, il client Bitcoin Core originale genera 100 chiavi private casuali quando

viene avviato per la prima volta e genera più chiavi in base alle esigenze, utilizzando ciascuna chiave una sola volta. Tali portafogli vengono sostituiti con portafogli deterministici perché sono ingombranti da gestire, per eseguire il backup ed importare. Lo svantaggio delle chiavi casuali è che se si generano molte di esse è necessario conservare le copie di tutti, il che significa che di quel portafoglio deve essere eseguito il backup di frequente. Ogni chiave deve essere salvata, oppure i fondi che controlla vengono persi irrevocabilmente se il portafoglio diventa inaccessibile. Ciò contrasta direttamente con il principio di evitare il riutilizzo degli indirizzi,

utilizzando ciascun indirizzo bitcoin per una sola transazione. Il riutilizzo degli indirizzi riduce la privacy associando più transazioni e indirizzi tra loro. Un portafoglio non deterministico di tipo 0 è una scelta scadente di un portafoglio, soprattutto se si desidera evitare il riutilizzo degli indirizzi perché significa gestire molte chiavi, il che crea la necessità di backup frequenti. Sebbene il client Bitcoin Core includa un portafoglio Type-0, l'uso di questo portafoglio è sconsigliato dagli sviluppatori di Bitcoin Core. La Figura 24 mostra un portafoglio non deterministico, contenente una raccolta libera di chiavi casuali.

TIP	L'utilizzo di wallet non deterministici è sconsigliato per qualsiasi cosa al di fuori di un semplice test. Sono semplicemente troppo difficile da esportare ed utilizzare. Al posto di questi ultimi, utilizza un industry-standard-based *HD wallet* con un *mnemonic seed* adatto al backup.

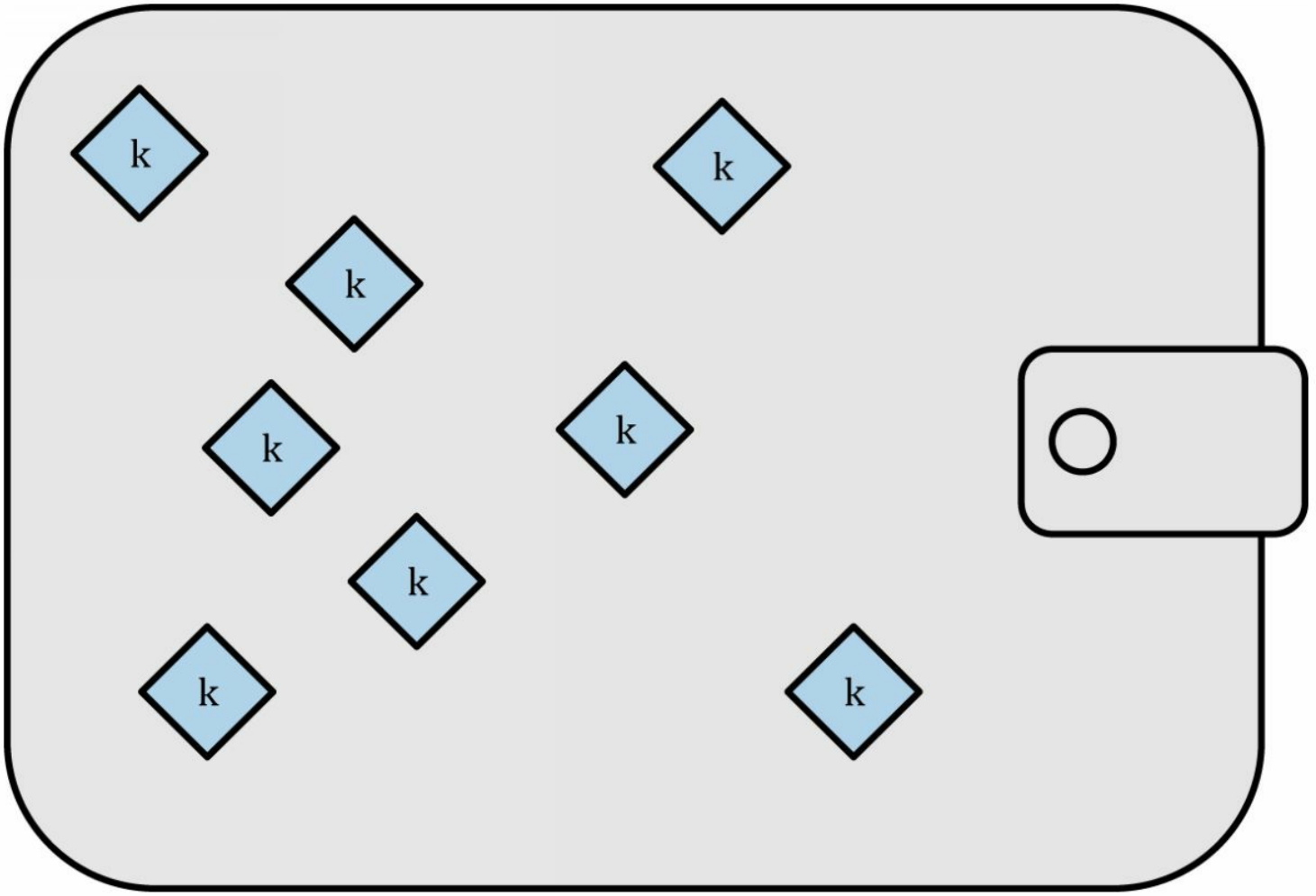


Figura 24. Wallet Type-0 nondeterministico (casuale): una serie di chiavi generate randomicamente

Wallet (Seeded) Deterministici

Deterministici, o “seeded”, wallet sono dei wallet che contengono chiavi private che a loro volta sono tutte derivate da un unico seed, attraverso l’uso di una funzione di hashing one-way. Il seed è un numero generato casualmente che viene combinato con altri dati, come un index number o “chain code” (vedi HD Wallets (BIP-32/BIP-44)) per derivare le chiavi private. In un wallet deterministico, il seed è insufficiente per recuperare tutte le chiavi derivate, quindi un singolo backup in fase di creazione è sufficiente. Il seed è inoltre sufficiente per l’importazione o l’esportazione di un intero wallet, permettendo una semplice migrazione di tutte le chiavi

dell'utente finale attraverso diverse implementazioni wallet. La Figura 25 mostra un diagramma logico di un wallet deterministico.

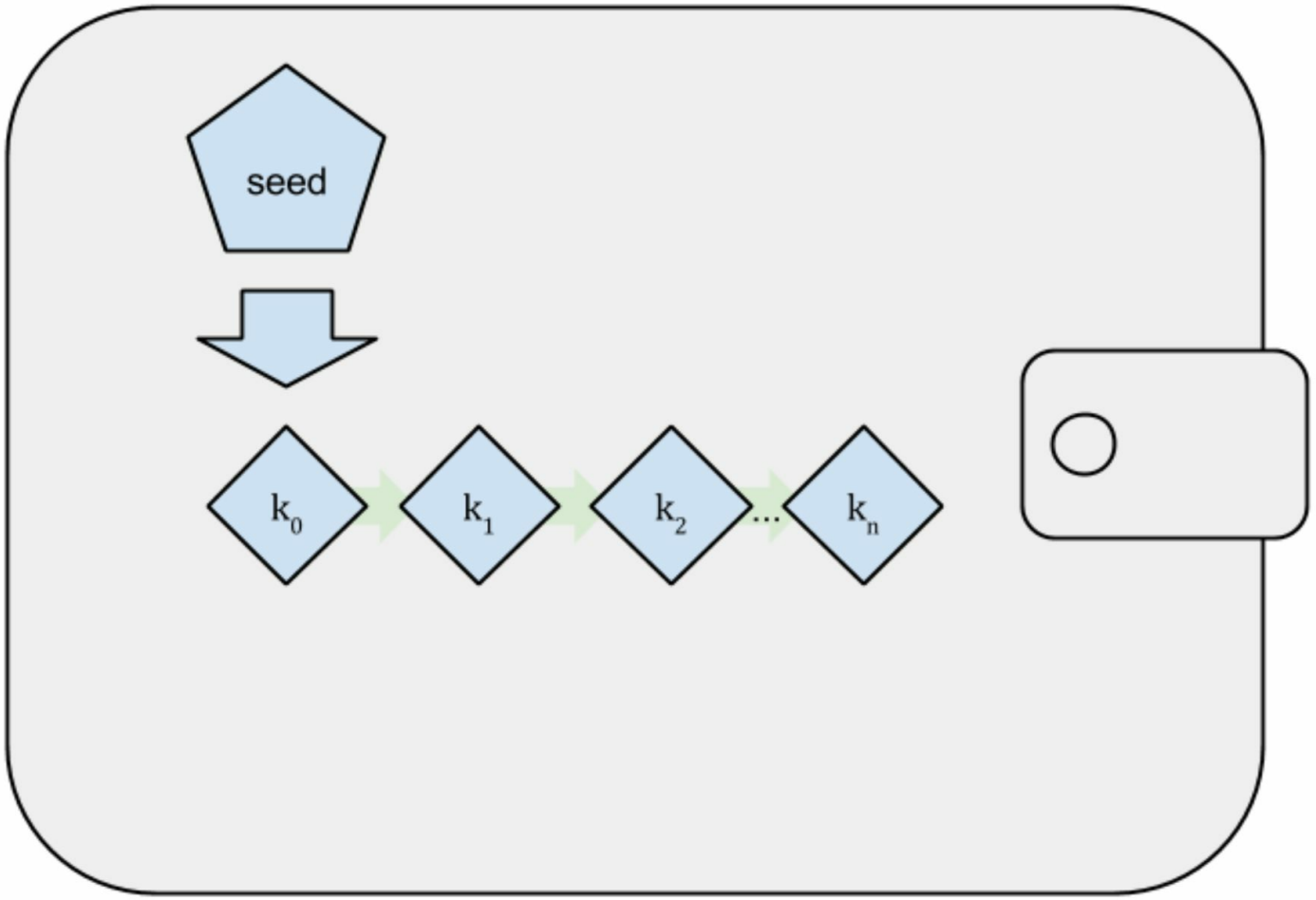


Figura 25. Type-1 deterministic (seeded) wallet: una sequenza determinata di chiavi derivate da un seed

HD Wallets (BIP-32/BIP-44)

I wallet Deterministici sono stati

sviluppati per consentire di derivare chiavi multiple da un singolo "seme" (seed). La forma più avanzata di wallet deterministico è il *hierarchical deterministic wallet* o *HD wallet* definito dallo standard BIP0032. I wallet gerarchici deterministici contengono chiavi derivate in una struttura ad albero, tale che da una chiave madre si può far derivare una sequenza di chiavi figlie, e da ognuna di queste si può far derivare una sequenza di chiavi nipoti, e così via, ad una profondità infinita. Questa struttura ad albero è illustrata in Figura 26.

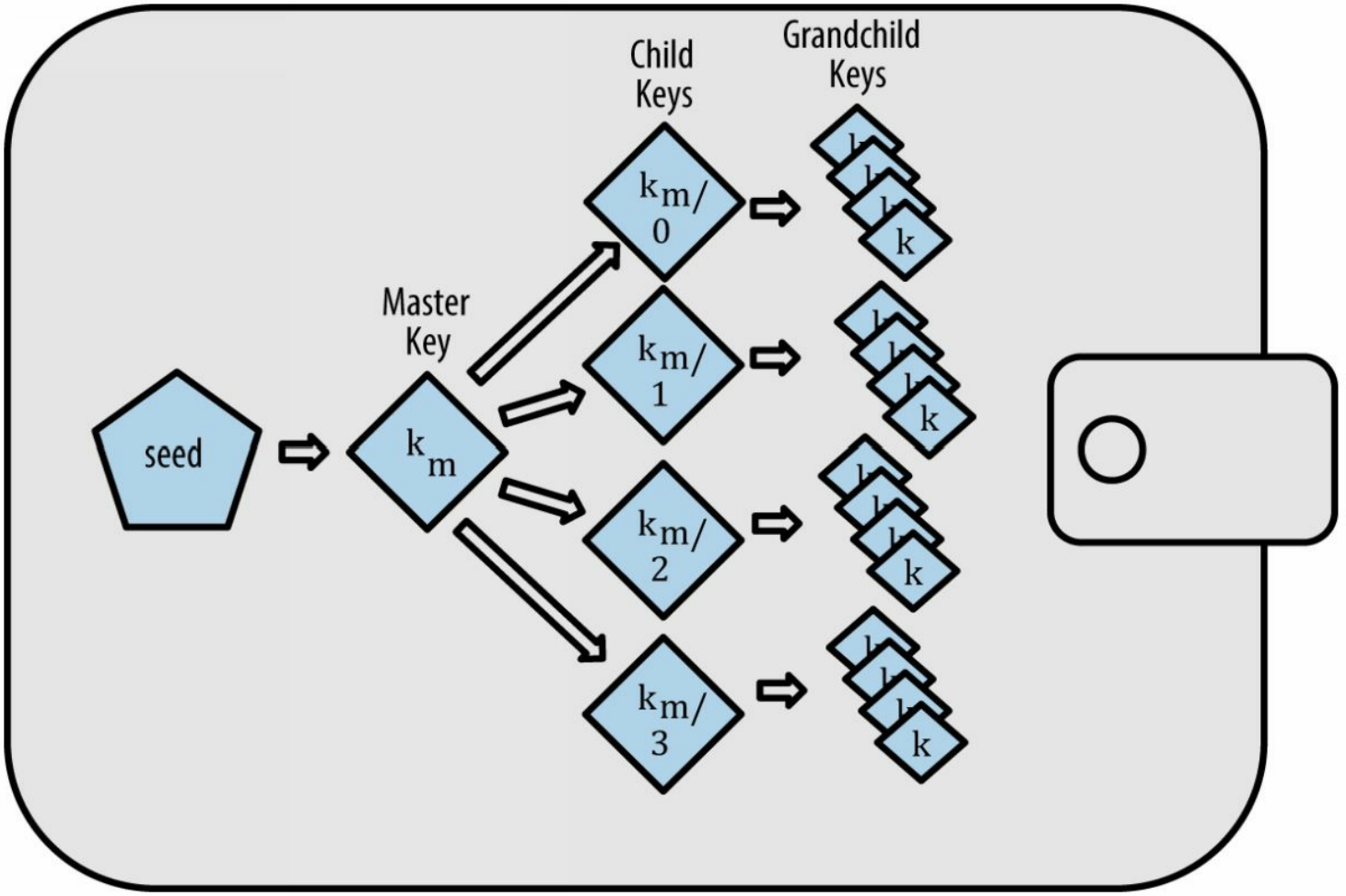


Figura 26. Type-2 wallet gerarchico deterministico: una albero di chiavi generate da un singolo seme

Gli HD wallet offrono due importanti vantaggi rispetto alle chiavi casuali (non deterministiche). In primo luogo,

la struttura ad albero può essere usata per rappresentare una struttura organizzativa aggiuntiva, ad esempio un ramo specifico di sottochiavi potrebbe essere utilizzato per ricevere i pagamenti in arrivo e un altro ramo per ricevere il resto da pagamenti in uscita. Alcuni rami di chiavi potrebbero essere utilizzate in diversi rami aziendali, assegnando diversi rami a dipartimenti, società affiliate, funzioni specifiche, o categorie contabili.

Il secondo vantaggio dei wallet HD è che gli utenti possono creare una sequenza di chiavi pubbliche senza avere accesso alle corrispondenti chiavi private. Questo permette ai

wallet HD di essere utilizzati su un server non sicuro o con sola capacità di ricezione, con l'emissione di una diversa chiave pubblica per ogni transazione. Le chiavi pubbliche non devono essere precaricate o derivati in anticipo, inoltre il server non deve avere le chiavi private necessarie per spendere i fondi.

Semi e codici mnemonici (BIP-39)

I portafogli HD sono un meccanismo molto potente per la gestione di molte chiavi e indirizzi. Sono ancora più utili se combinati in un modo standardizzato di creare semi da una sequenza di parole inglesi facili da

trascrivere, esportare ed importare attraverso i portafogli. Questo è noto come *mnemonico* e lo standard è definito da BIP-39. Oggi, la maggior parte dei portafogli bitcoin (così come i portafogli per altre criptovalute) usa questo standard e può importare ed esportare semi per il backup e il ripristino usando mnemonici interoperabili.

Diamo un'occhiata a questo da una prospettiva pratica. Quale dei seguenti semi è più facile da trascrivere, registrare su carta, leggere senza errori, esportare e importare in un altro portafoglio?

1. Un seme per un portafoglio deterministico, in esadecimale

0C1E24E5917779D297E14D45F14E1

1. Un seme per un portafoglio deterministico, formato da un mnemonico di 12 parole

army van defense carry jealous true
garbage claim echo media make crunch

Pratiche consigliate per i Wallet

Poiché la tecnologia dei portafogli bitcoin si è consolidata, alcuni standard di settore comuni hanno fatto emergere quello che rende i portafogli bitcoin ampiamente interoperabili, facili da usare, sicuri e flessibili. Questi standard comuni sono:

- Parole in codice mnemonico, basate

su BIP-39

- HD wallets, basati su BIP-32
- Multipurpose HD wallet structure, basati su BIP-43
- Portafogli multivaluta e multiaccount, basati su BIP-44

Questi standard potrebbero cambiare o diventare obsoleti causa sviluppi futuri, ma per ora formano un set di tecnologie che sono diventate de facto gli standard dei wallet bitcoin.

Gli standard sono stati accettati da una serie di software wallet e hardware wallet per bitcoin, rendendo tutti questi wallet capaci di “parlare” tra di loro. Un utente può esportare un seed mnemonico generato in uno di questi

wallet ed importarlo in un altro wallet, recuperando tutte le transazioni, le chiavi e gli indirizzi.

Alcuni esempi di software wallet che supportano questi standard includono (alfabeticamente) Breadwallet, Copay, Multibit HD e Mycelium. Esempi di hardware wallet che supportano questi standard includono (alfabeticamente) KeepKey, Ledger e Trezor.

I capitoli seguenti esamineranno ognuna di queste tecnologie in dettaglio.

	Se stai implementando un wallet bitcoin dovrebbe essere costruito con un HD

TIP	wallet, con un seed codificato come codice mnemonico per il backup, seguendo gli standard BIP-32, BIP-39, BIP-43 e BIP-44 come descritto nei seguenti capitoli.

Utilizzando un Bitcoin Wallet

In I vari Utilizzi di Bitcoin, gli Utenti e le loro Storie abbiamo introdotto Gabriel, un teenager intraprendente in Rio de Janeiro, che gestisce un semplice ecommerce e vende t-shirts su Bitcoin, tazze da caffè e adesivi.

Gabriel utilizza un Trezor, un

hardware wallet per bitcoin, per gestire in sicurezza i propri bitcoin. Il Trezor è un semplice device USB che memorizza le chiavi (in forma di HD wallet) e firma le transazioni. Il wallet Trezor implementa tutti gli standard di settore discussi in questo capitolo, quindi Gabriel non si affida a nessuna tecnologia proprietaria o soluzione di un singolo fornitore.

Figura 27. Un device Trezor: un bitcoin wallet HD sottoforma di dispositivo hardware

Quando Gabriel ha usato Trezor per la prima volta, il dispositivo ha generato un codice mnemonico e un seed da un generatore hardware di numeri casuali

incorporato. Durante questa fase di inizializzazione, il wallet mostra una sequenza numerata di parole, una per una, sullo schermo

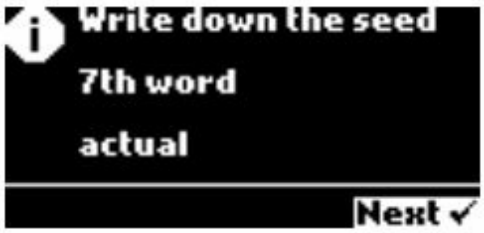


Figura 28. Trezor che mostra una delle tante mnemonic wordsAnnotando queste mnemonic words

Gabriel ha creato un backup (vedi la Tabella 10) che può essere utilizzato per il recupero in caso di perdita o danneggiamento del dispositivo Trezor. Questo codice mnemonico può essere utilizzato per il recupero in un nuovo Trezor o in uno dei tanti

pacchetti software o hardware compatibili. Si noti che la sequenza di parole è importante, per questo motivo i backup delle parole mnemoniche hanno spazi numerati per ogni parola. Gabriel ha dovuto registrare attentamente ogni parola nello spazio numerato per preservare la sequenza corretta.

Tabella 10. Il backup cartaceo delle mnemonic words del wallet di Gabriel

1.	*army*	**7.**	*garbage*
2.	*van*	**8.**	*claim*
3.	*defense*	**9.**	*echo*

4.	*carry*	**10.**	*media*
5.	*jealous*	**11.**	*make*
6.	*true*	**12.**	*crunch*

NOTA	Nella Tabella qui sopra viene mostrato un mnemonico di 12 parole per semplicità. In effetti, la maggior parte dei portafogli hardware genera un codice mnemonico di 24 parole più sicuro. Il mnemonico è usato esattamente nello stesso modo indipendentemente

	dalla lunghezza.

Per la prima implementazione del suo negozio web, Gabriel usa un singolo indirizzo bitcoin, generato sul suo dispositivo Trezor. Questo singolo indirizzo è utilizzato da tutti i clienti per tutti gli ordini. Come vedremo, questo approccio presenta alcuni inconvenienti e può essere migliorato con un portafoglio HD.

Dettagli sulla Tecnologia dei Wallet

Esaminiamo ora ciascuno degli importanti standard di settore che sono utilizzati da molti portafogli bitcoin in

dettaglio.

Mnemonic Code Words (BIP-39)

I codici mnemonici sono sequenze di parole inglesi che rappresentano (codificano) un numero casuale usato come seme per generare un portafoglio deterministico. La sequenza di parole è sufficiente per ricreare il seme e da lì ricreare il portafoglio e tutte le chiavi da esso derivate. Un'applicazione wallet che implementa i wallet deterministici con il codice mnemonico mostrerà all'utente, quando si crea per la prima volta un wallet, una sequenza di 12-24 parole. Questa sequenza di parole è il

backup del wallet e può essere utilizzata per recuperare e ricreare tutte le chiavi nella stessa o in qualsiasi applicazione wallet compatibile. Le parole di codice mnemonico facilitano agli utenti il backup dei portafogli perché sono facili da leggere e trascrivere correttamente, rispetto a una sequenza casuale di numeri.

	Le parole mnemoniche vengono spesso confuse con "brainwallets". Non sono la stessa cosa. La differenza principale è che un brainwallet è costituito da parole scelte dall'utente, mentre

TIP	le parole mnemoniche sono create casualmente dal portafoglio e presentate all'utente. Questa importante differenza rende le parole mnemoniche molto più sicure, perché gli umani sono fonti molto povere di casualità.

I codici mnemonici sono definiti in BIP-39 (vedi Bitcoin Improvement Proposals). Nota che BIP-39 è una bozza di proposta e non uno standard. Nello specifico, c'è uno standard

diverso, con un diverso set di parole, usato da wallet Electrum e che ha anticipato il BIP-39 . BIP0039 è usato dal wallet Trezor e alcuni altri portafogli, ma è incompatibile con l'implementazione di Electrum. BIP-39 è stato proposto dalla società dietro il portafoglio hardware Trezor ed è incompatibile con l'implementazione di Electrum. Tuttavia, BIP-39 ha ora ottenuto un ampio supporto del settore attraverso dozzine di implementazioni interoperabili e dovrebbe essere considerato lo standard di settore de facto.

BIP-39 definisce la creazione di un codice mnemonico e di un seed, che descriveremo di seguito in 9 step. Per

semplicità, il processo è diviso in due parti: gli step da 1 a 6 sono descritti in Generazione di parole mnemoniche e gli step da 7 a 9 sono descritti in Dal mnemonic al seed.

Generazione di parole mnemoniche

Le parole mnemoniche sono generate automaticamente dal portafoglio usando il processo standardizzato definito in BIP-39. Il portafoglio parte da una fonte di entropia, aggiunge un checksum e quindi mappa l'entropia in un elenco di parole:

. Crea una sequenza casuale (entropia) da 128 a 256 bit.

. Crea un checksum della sequenza casuale prendendo qualcuno dei primi bit del suo hash SHA256.

. Aggiungi il checksum alla fine della sequenza casuale.

. Dividi la sequenza in due sezioni di 11 bit

. Mappa ogni valore di 11 bit ad una parola delle 2048 parole predefinite.

. Il codice mnemonico è la sequenza di parole.

La Figura 29 mostra come l'entropia viene utilizzata per generare le parole mnemoniche.

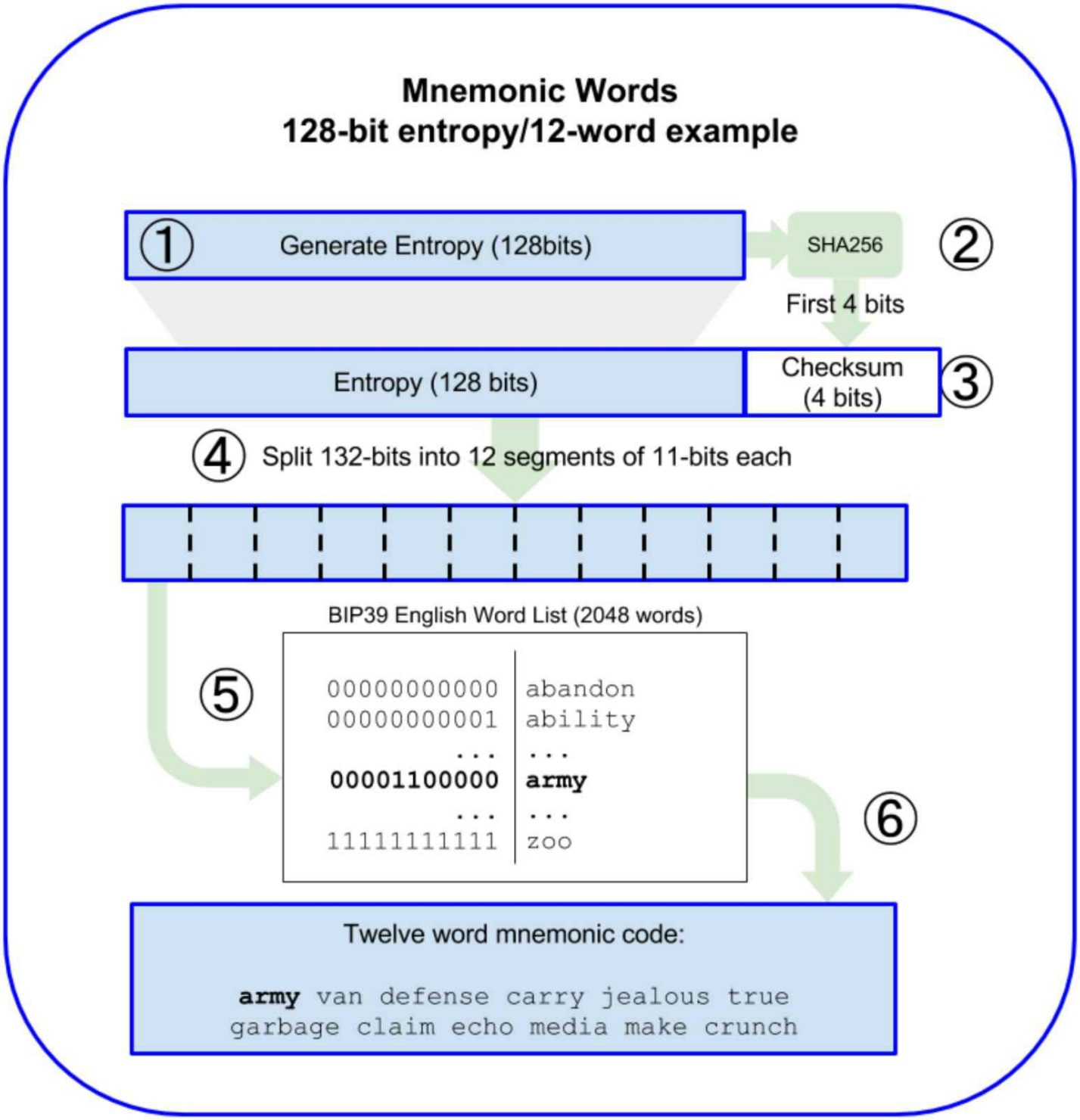


Figura 29. Generazione di entropia e codifica come parole mnemoniche

Codici mnemonici: entropia e numero di parole mostra la relazione tra la grandezza dei dati dell'entropia e la lunghezza dei codici mnemonici in parole.

Tabella 11. Codici mnemonici: e numero di parole

Entropia (bit)	Checksum (bit)	Entropy + checksum (bits)
128	4	132

160	5	165
192	6	198
224	7	231
256	8	264

Dal mnemonic al seed

Le parole mnemoniche rappresentano entropia con una lunghezza da 128 a 256 bit. L'entropia viene quindi utilizzata per derivare un seme più lungo (512 bit) attraverso l'uso della funzione di allungamento della chiave PBKDF2. Il seme prodotto viene quindi utilizzato per costruire un portafoglio deterministico e derivarne le chiavi.

La funzione di allungamento della chiave richiede due parametri: il mnemonico e il sale (salt). Lo scopo di un salt in una funzione di allungamento della chiave è rendere difficile la creazione di una tabella di ricerca che consenta un attacco di forza bruta. Nello standard BIP-39, il salt ha un altro scopo: consente l'introduzione di una passphrase che funge da ulteriore fattore di sicurezza a protezione del seme, come descriveremo più dettagliatamente in Passphrase opzionale in BIP-39.

Il processo descritto nei passaggi da 7 a 9 continua dal processo descritto in precedenza:

7. Il primo parametro della funzione di

allungamento dei tasti PBKDF2 è il mnemonico prodotto dal punto 6

8. Il secondo parametro della funzione di allungamento della chiave PBKDF2 è un valore salt. Il sale è composto dalla costante di stringa "mnemonica" concatenata con una stringa passphrase facoltativa fornita dall'utente.
9. PBKDF2 estende i parametri mnemonici e il salt usando 2048 round di hashing con l'algoritmo HMAC-SHA512, producendo un valore a 512 bit come output finale. Quel valore a 512 bit è il seed.

La Figura 30 mostra come un mnemonic viene utilizzato per generare un seme.

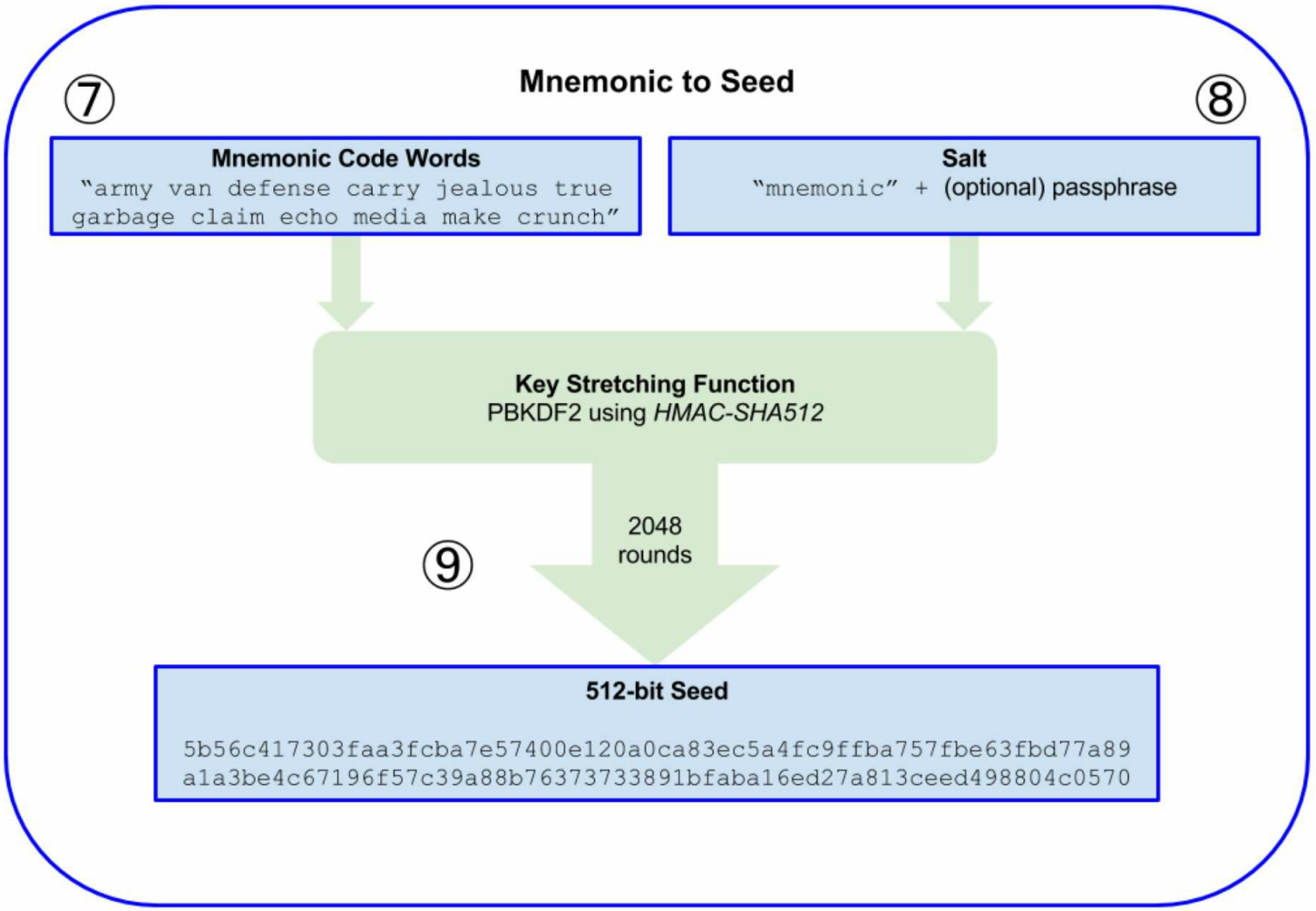


Figura 30. Dal mnemonico al seed

La funzione di allungamento della chiave, con i suoi 2048 round di hashing, è una protezione molto efficace

TIP	contro gli attacchi di forza bruta contro il mnemonico o la frase segreta. Rende estremamente costoso (in termini computazionali) provare più di qualche migliaio di passphrase e combinazioni mnemoniche, mentre il numero di possibili semi derivati è vasto (2512).

Le tabelle seguenti mostrano alcuni esempi di codice mnemonico e i seed che producono (senza alcuna passphrase).

Tabella 12. Codice di entropia mnem

Entropia di input (128 bit)	0c1e24e5917779d297
Mnemonic (12 parole)	army van defense carr
Passphrase	(nessuna)
Seed (512 bit)	5b56c417303faa3fcba a88b76373733891bfal

Tabella 13. Codice di entropia mnem

Entropia di input (128 bit)	0c1e24e5917779d297

Mnemonic (12 parole)	army van defense carr
Passphrase	SuperDuperSecret
Seed (512 bit)	3b5df16df2157104cfd 715861dc8a18358f80

Tabella 14. Codice di entropia mnem

Entropia di input (128 bit)	2041546864449caff93
Mnemonic (12 parole)	cake apple borrow sil measure invite love tra
Passphrase	(nessuna)

Seed (512 bit)	3269bce2674acbd188 5f1e0deaa082df8d487

Passphrase opzionale in BIP-39

Lo standard BIP-39 consente l'uso di una passphrase opzionale nella derivazione del seme. Se non viene usata la passphrase, il mnemonico viene allungato con un sale costituito dalla stringa costante "mnemonic", producendo un seme specifico a 512 bit da qualsiasi mnemonico dato. Se si utilizza una passphrase, la funzione di stretching produce un seme diverso da quello stesso mnemonico. Infatti, dato un singolo mnemonico, ogni

passphrase possibile porta a un seme diverso. In sostanza, non esiste una passphrase "errata". Tutte le passphrase sono valide e tutte portano a semi diversi, formando una vasta serie di possibili portafogli non inizializzati. L'insieme dei possibili portafogli è così grande (2_{512}) che non vi è alcuna possibilità pratica di forzare brute o indovinare casualmente uno che è in uso.

TIP	Non ci sono passphrase "errate" in BIP-39. Ogni passphrase porta a un portafoglio, che se non utilizzato in precedenza sarà vuoto.

La passphrase opzionale crea due importanti caratteristiche:

- Un secondo fattore (qualcosa di memorizzato) che rende un mnemonico inutile da solo, proteggendo i backup mnemonici da un ladro.
- Una forma di negazione plausibile o "portafoglio di coercizione", in cui una passphrase scelta porta a un portafoglio con una piccola quantità di fondi utilizzati per distrarre un attaccante dal portafoglio "reale" che contiene la maggior parte dei fondi.

Tuttavia, è importante notare che l'uso di una passphrase introduce anche il

rischio di perdita:

- Se il proprietario del portafoglio è inabilitato o morto e nessun altro conosce la passphrase, il seme è inutile e tutti i fondi memorizzati nel portafoglio sono persi per sempre.
- Viceversa, se il proprietario esegue il backup della passphrase nello stesso posto del seme, sconfigge lo scopo di un secondo fattore.

Sebbene le passphrase siano molto utili, dovrebbero essere utilizzate solo in combinazione con un processo attentamente pianificato per il backup e il recupero, considerando la possibilità di sopravvivere al proprietario e consentire alla sua famiglia di recuperare la proprietà dei

bitcoin.

Lavorare con codici mnemonici

BIP-39 è implementato come una libreria in molti diversi linguaggi di programmazione:

python-mnemonic

L'implementazione di riferimento dello standard da parte del team di SatoshiLabs che ha proposto BIP-39, in Python

bitcoinjs/bip39

Un'implementazione di BIP-39, come parte del popolare framework bitcoinJS, in JavaScript

libbitcoin/mnemonic

Un'implementazione di BIP-39, come parte del popolare framework Libbitcoin, in C ++

Esiste anche un generatore BIP-39 implementato in una pagina web standalone, che è estremamente utile per i test e la sperimentazione. La Figura 31 mostra una pagina Web autonoma che genera mnemonics, seed e chiavi private estese.

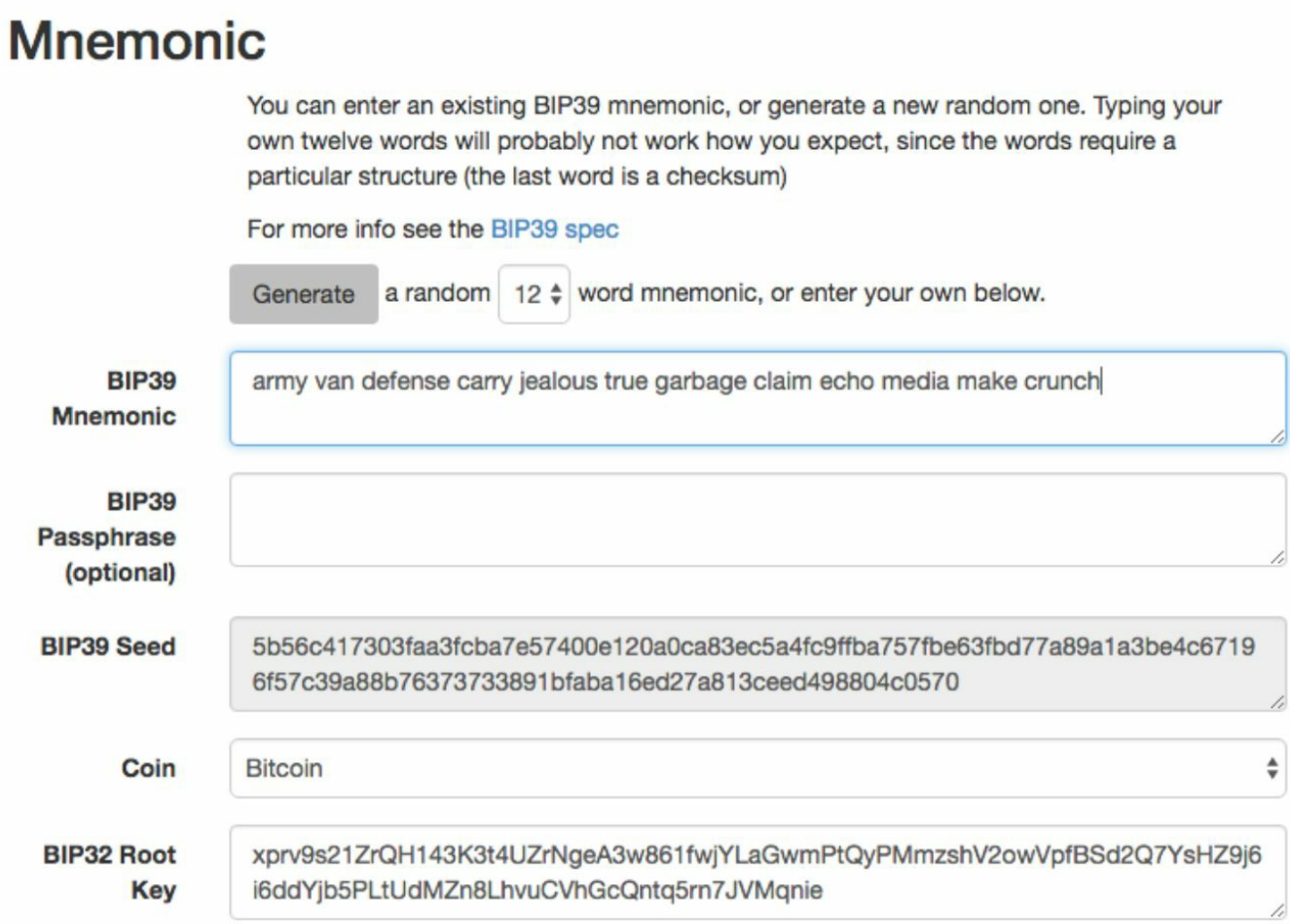


Figura 31. Un generatore BIP-39 come pagina Web autonoma

La pagina (https://iancoleman.github.io/bip39/) può essere utilizzata offline in un browser, oppure online.

Creare un HD Wallet dal Seed

I wallet HD sono creati da un singolo *root seed*, che è un numero casuale a 128-, 256-, o 512-bit. Tutto il resto nei wallet HD è deterministicamente derivato da questo seme radice, che rende possibile ricreare l'intero wallet HD da quel seme in ogni wallet HD compatibile. Questo rende semplice effettuare copie di sicurezza, ripristinare, esportare, e importare wallet HD contenenti migliaia o anche milioni di chiavi semplicemente trasferendo solo il seme radice. Il seme radice viene spesso rappresentato da una sequenza

mnemonica di parole, come descritto nella precedente sezione Mnemonic Code Words, per rendere più semplice alle persone trascriverle e conservarle.

Il processo di creare le chiavi master e il codice master chain per un wallet HD è mostrato in Figura 32.

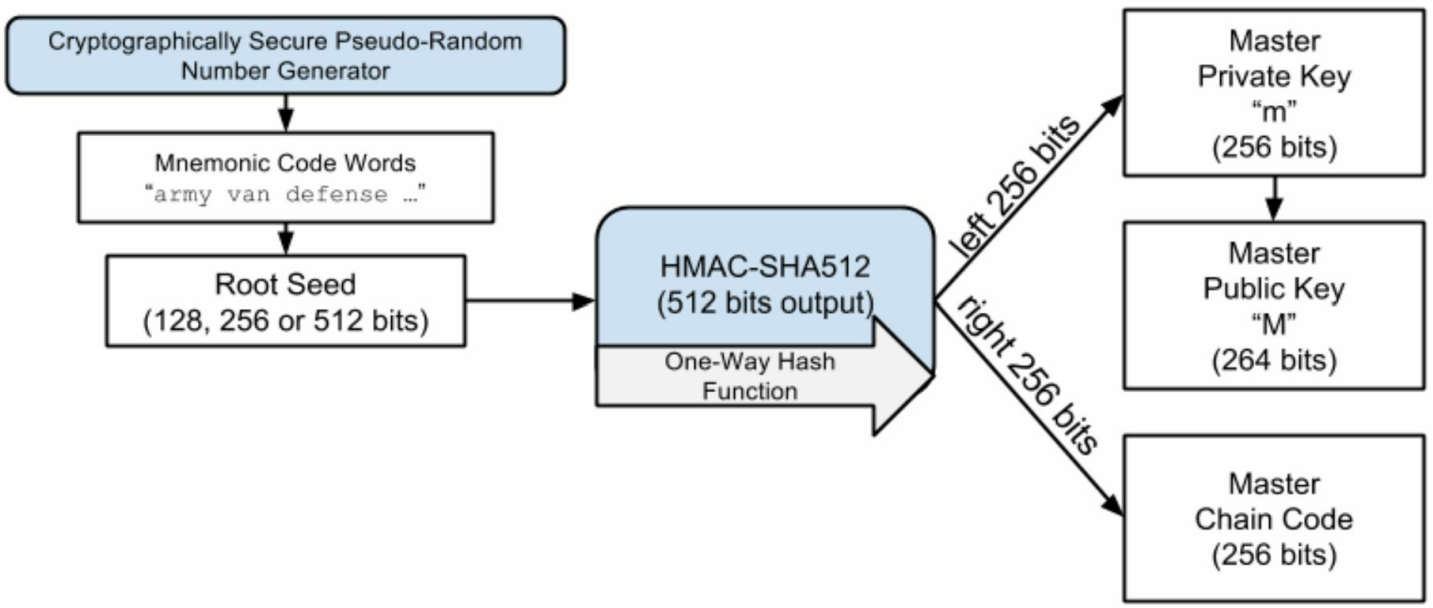


Figura 32. Creare le chiavi master e il chain code da un

seme radice (seed)

Il seme radice è l'input nell'algoritmo HMAC-SHA512 e l'hash risultante viene usato per creare una *chiave privata master* (m) e un *codice master chain*.

La chiave privata master (m) poi genera una chiave pubblica master corrispondente (M), usando il normale processo di moltiplicazione m * G della curva ellittica che abbiamo visto precedentemente in questo capitolo. Il codice chain viene usato per introdurre entropia nella funzione che crea le chiavi figlie dalle chiavi madre, come vedremo nella prossima sezione.

Il codice catena (c) viene utilizzato per

introdurre l'entropia nella funzione che crea chiavi figlie dalle chiavi padre, come vedremo nella sezione successiva.

Derivazione delle chiavi private figlie

I portafogli deterministici gerarchici usano una funzione *child key derivation* (CKD) per derivare le chiavi figlie dalle chiavi genitori.

Le funzioni di derivazione delle chiavi figlie sono basate su una funzione hash mono-direzionale che combina:

- Una chiave madre privata o pubblica (chiave ECDSA non compressa)
- Un seme chiamato chain code (256

bit)

- Un numero indice (32 bit)

Il chain code viene utilizzato per introdurre dati apparentemente casuali nel processo, in modo che non sia sufficiente l'indice per derivare altre chiavi figlie. In questo modo, avere una chiave figlia non rende possibile trovare le chiavi sorelle, a meno che non si possegga anche il chain code. Il seme iniziale del chain code (alla radice dell'albero) è composto da dati casuali, mentre i successivi chain code sono derivate da ogni chain code padre.

Questi tre elementi vengono combinati e codificati come hash per generare chiavi figlie, come di seguito.

La chiave pubblica principale, il chain code, e il numero indice vengono combinati e codificati con l'algoritmo hash HMAC-SHA512 per produrre un hash di 512 bit. L'hash risultante viene diviso in due metà. I 256 bit della metà di destra dell' hash risultante diventa il chain code per il figlia. I 256 bit della metà di sinistra dell'hash e il numero indice vengono aggiunti alla chiave privata madre per produrre la chiave privata figlia. In [CKDpriv], lo vediamo illustrato con il set di indice 0 per produrre lo 0'esimo (primo per indice) figlio del genitore.

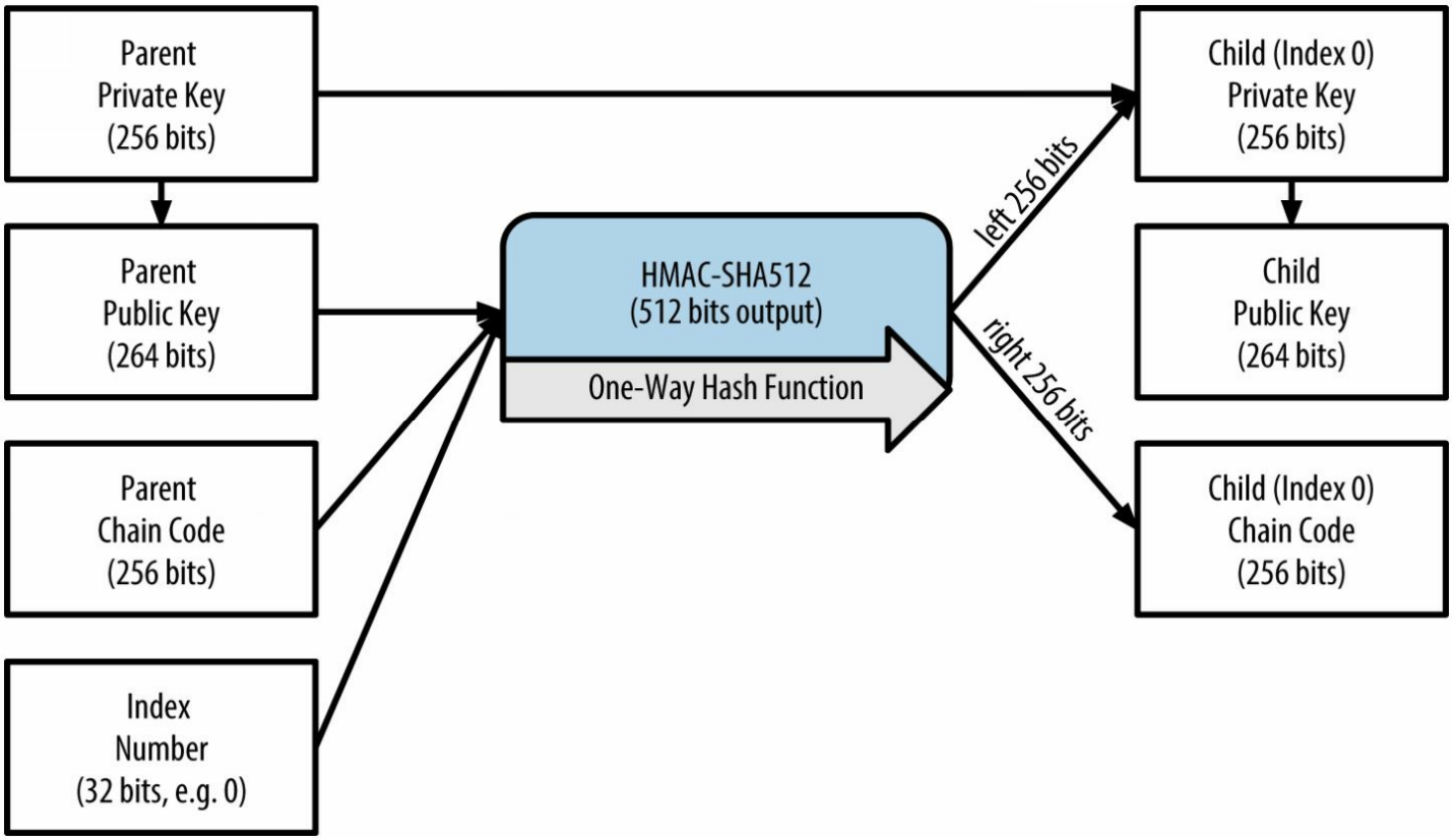


Figura 33. Estendere una chiave privata madre per creare una chiave privata figlia

Cambiare l'indice ci consente di estendere la chiave madre e creare gli altri figli in sequenza, ad esempio Figlio 0, Figlio 1, Figlio 2, ecc. Ogni

chiave madre può avere specificatamente 2,147,483,647 (231) figlie (231 è la metà dell'intero intervallo disponibile, 232, perché l'altra metà è riservata a un tipo speciale di derivazione di cui parleremo più avanti in questo capitolo).

Ripetendo il processo di un livello verso il basso nell'albero, ogni figlia può a sua volta diventare madre e creare le proprie figlie, in un numero infinito di generazioni.

Usare le chiavi figlie derivate

Le chiavi private figlie sono indistinguibili dalle chiavi non deterministiche (casuali). Poiché la

funzione di derivazione è una funzione unidirezionale, la chiave figlia non può essere utilizzata per trovare la chiave madre. La chiave figlia inoltre non può essere usata per trovare eventuali sorelle. Se avete l'n°esima figlia, non è possibile trovare le sue sorelle, come la figlia n-1 o la figlia n + 1, o qualsiasi altra figlia che fa parte della sequenza. Solo la chiave madre e il codice della catena possono produrre tutte le figlie. Senza il codice della catena della figlia, la chiave figlia non può essere utilizzata neanche per ricavare eventuali nipoti. È necessaria sia la chiave privata figlia che il codice della catena figlia per iniziare un nuovo ramo e derivare nipoti.

Quindi la chiave figlia privata di per se stessa per cosa può essere utilizzata? Può essere utilizzata per fare una chiave pubblica e un indirizzo bitcoin. Quindi, può essere utilizzata per firmare transazioni per spendere tutto ciò che è stato pagato su tale indirizzo.

TIP	Una chiave figlia privata, la corrispondente chiave pubblica, e l'indirizzo bitcoin sono tutti indistinguibili da chiavi e indirizzi creati in modo casuale. Il fatto che siano parte di una sequenza non è visibile

	all'esterno della funzione HD wallet che li ha generati. Una volta create, operano esattamente come chiavi "normali".

Chiavi estese

Come abbiamo visto in precedenza, la funzione di derivazione delle chiavi può essere utilizzata per creare figlie a qualsiasi livello della struttura, sulla base dei tre ingressi: una chiave, un codice a catena, e l'indice della figlia desiderata. I due ingredienti essenziali sono il codice della chiave e della catena, e quando combinati vengono

chiamati una *chiave estesa*. Il termine "chiave estesa" potrebbe anche essere pensato come "chiave estensibile" perché tale chiave può essere usata per derivare delle chiavi figlie.

Le chiavi estese sono memorizzate e rappresentate semplicemente come la concatenazione della chiave a 256 bit e il codice a catena a 256 bit in una sequenza di 512 bit. Esistono due tipi di chiavi estese. Una chiave privata estesa è la combinazione di una chiave privata e codice catena e può essere utilizzato per derivare chiavi figlie private (e da loro, chiavi figlie pubbliche). Una chiave pubblica estesa è una chiave pubblica e catena codice, che può essere utilizzato per

creare *chiavi pubbliche figlie*, come descritto in Generare una Chiave Pubblica.

Pensate a una chiave estesa come alla radice di un ramo nella struttura del portafoglio HD. Con la radice del ramo, è possibile derivare il resto del ramo. La chiave privata estesa può creare un ramo completo, mentre la chiave pubblica estesa può creare solo un ramo di chiavi pubbliche.

TIP	Una chiave estesa è composta da una chiave privata o pubblica e chain code (codice catena). Una chiave estesa può creare figli

	generando un propric ramo nella struttura ac albero. Condividere una chiave estesa dà accesso a tutto il ramo.

Chiavi estese sono codificate utilizzando Base58Check, per poter essere esportate e importate facilmente tra i diversi portafogli compatibili BIP0032. La codifica Base58Check per le chiavi estese utilizza un numero di versione speciale che produce il prefisso "xprv" e "xpub" quando codificato in caratteri a Base58, per renderli facilmente riconoscibili. Poiché la chiave estesa è 512 o 513 bit, è anche molto più lunga di altre

stringhe Base58Check codificate che abbiamo visto in precedenza.

Ecco un esempio di una chiave *privata* estesa, codificata in Base58Check:

```
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQ
```

Ecco la corrispondente chiave *pubblica* estesa, anche questa codificata in Base58Check:

```
xpub67xpozcx8pe95XVuZLHXZeG6XWXH
```

Derivazione di chiave pubblica figlia

Come menzionato precedentemente, una caratteristica molto utile dei wallet gerarchici deterministici è l'abilità di derivare chiavi pubbliche figlie da chiavi pubbliche madri, *senza*

avere le chiavi private. Questo ci fornisce due modi di derivare una chiave pubblica: sia dalla chiave privata figlia, o direttamente dalla chiave pubblica madre.

Una chiave pubblica estesa può quindi essere utilizzata per derivare tutte le chiavi pubbliche (e solo le chiavi pubbliche) in quel ramo della struttura HD del wallet.

Questo collegamento può essere utilizzato per creare distribuzioni di sicurezza a chiave pubblica molto sicure in cui un server o un'applicazione ha una copia di una chiave pubblica estesa e nessuna chiave privata. Questo tipo di implementazione può produrre un

numero infinito di chiavi pubbliche e indirizzi bitcoin, ma non può spendere parte del denaro inviato a tali indirizzi. Nel frattempo, su un altro server più sicuro, la chiave privata estesa può derivare tutte le chiavi private corrispondenti per firmare le transazioni e spendere i soldi.

Un'applicazione comune di questa soluzione è l'installazione di una chiave pubblica estesa su un server Web che serve un'applicazione di e-commerce. Il server Web può utilizzare la funzione di derivazione della chiave pubblica per creare un nuovo indirizzo bitcoin per ogni transazione (ad esempio, per un carrello degli acquisti del cliente). Il

server web non avrà alcuna chiave privata che potrebbe essere vulnerabile ai furti. Senza wallet HD, l'unico modo per farlo è generare migliaia di indirizzi bitcoin su un server sicuro separato e quindi precaricarli sul server e-commerce. Tale approccio è macchinoso e richiede una manutenzione costante per garantire che il server e-commerce non "esaurisca" le chiavi.

Un'altra applicazione comune di questa soluzione è per la cold-storage (n.d.r. conservazione a freddo) o portafogli hardware. In tale scenario, la chiave privata estesa può essere memorizzata su un portafoglio cartaceo o dispositivo hardware (come un

wallet hardware Trezor), mentre la chiave pubblica estesa può essere essere tenuta online L'utente può creare a propria discrezione indirizzi per la ricezione, mentre le chiavi private vengono archiviate in modo sicuro offline. Per spendere i fondi, l'utente può utilizzare la chiave privata estesa su un client bitcoin con firma offline o firmare transazioni sul dispositivo del portafoglio hardware (ad es. Trezor). Estendere una chiave pubblica madre per creare una chiave pubblica figlia illustra il meccanismo per estendere una chiave pubblica padre per ricavare chiavi pubbliche figlie.

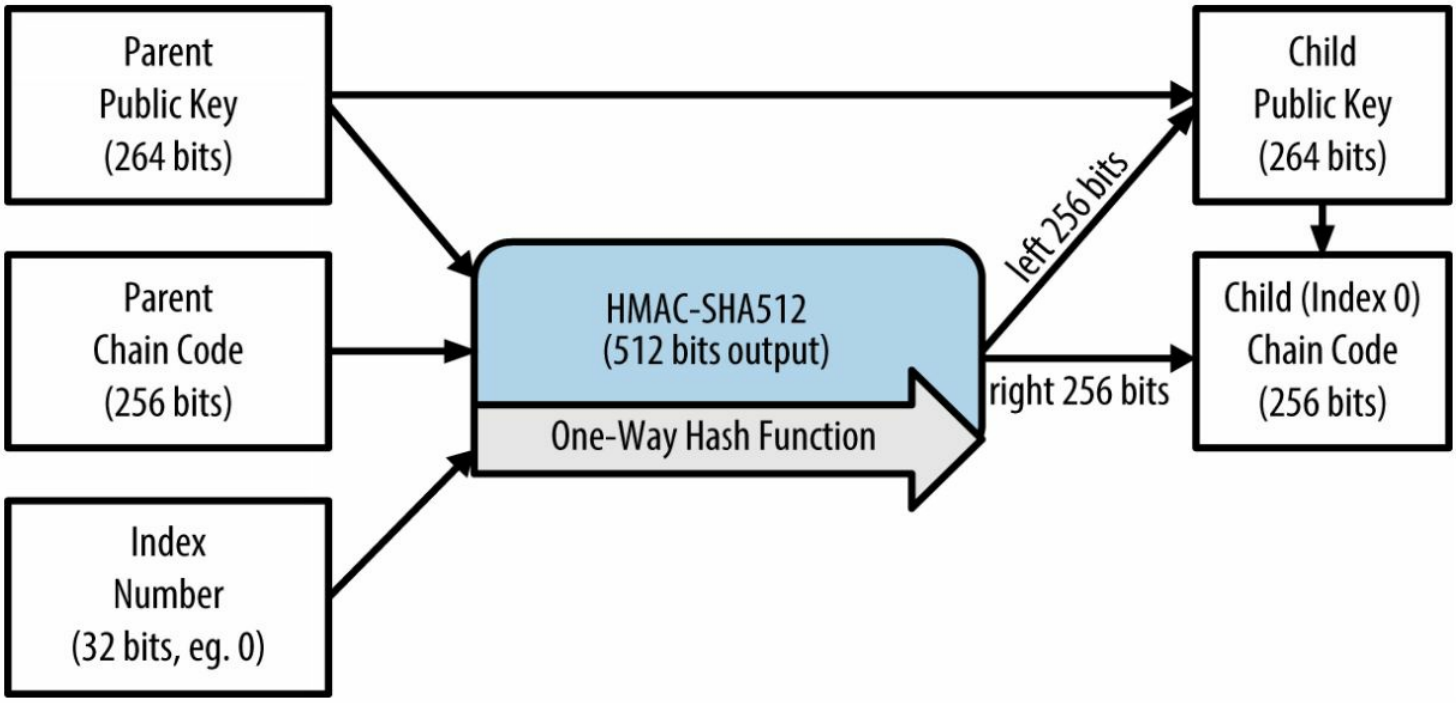


Figura 34. Estendere una chiave pubblica madre per creare una chiave pubblica figlia

Utilizzo di una Chiave Pubblica Estesa in un negozio Web

Vediamo come vengono utilizzati i portafogli HD continuando la nostra

storia con il negozio web di Gabriel.

Gabriel ha creato il suo negozio web per hobby, basandosi su una semplice sito web ospitato da Wordpress. Il suo negozio era piuttosto semplice con solo poche pagine e un modulo d'ordine con un unico indirizzo bitcoin.

Gabriel ha usato il primo indirizzo bitcoin generato dal suo dispositivo Trezor come indirizzo bitcoin principale per il suo negozio. In questo modo, tutti i pagamenti in entrata verrebbero corrisposti a un indirizzo controllato dal suo portafoglio hardware Trezor.

I clienti invieranno un ordine utilizzando il modulo e invieranno il

pagamento all'indirizzo bitcoin pubblicato da Gabriel, attivando un'e-mail con i dettagli dell'ordine che Gabriel ha elaborato. Con pochi ordini ogni settimana, questo sistema ha funzionato abbastanza bene.

Tuttavia, il piccolo web store ha avuto un discreto successo e ha attirato molti ordini dalla comunità locale. Ben presto, Gabriel fu sopraffatto. Con tutti gli ordini che pagavano lo stesso indirizzo, è diventato difficile abbinare correttamente gli ordini e le transazioni, soprattutto quando sopraggiungevano più ordini con lo stesso importo.

Il portafoglio HD di Gabriel offre una soluzione migliore grazie alla

possibilità di ricavare chiavi figlie pubbliche senza conoscere le chiavi private. Gabriel può caricare una chiave pubblica estesa (xpub) sul suo sito Web, che può essere utilizzata per ricavare un indirizzo univoco per ogni ordine cliente. Gabriel può spendere i fondi dal suo Trezor, ma il xpub caricato sul sito web può solo generare indirizzi e ricevere fondi. Questa caratteristica dei portafogli HD è una grande funzionalità di sicurezza. Il sito web di Gabriel non contiene alcuna chiave privata e quindi non ha bisogno di alti livelli di sicurezza.

Per esportare xpub, Gabriel usa il software basato sul web insieme al portafoglio hardware di Trezor. Il

dispositivo Trezor deve essere collegato per poter esportare le chiavi pubbliche. Nota che i portafogli hardware non esporteranno mai le chiavi private, quelle rimangono sempre sul dispositivo. La Figura 34 mostra l'interfaccia web che Gabriel usa per esportare xpub.

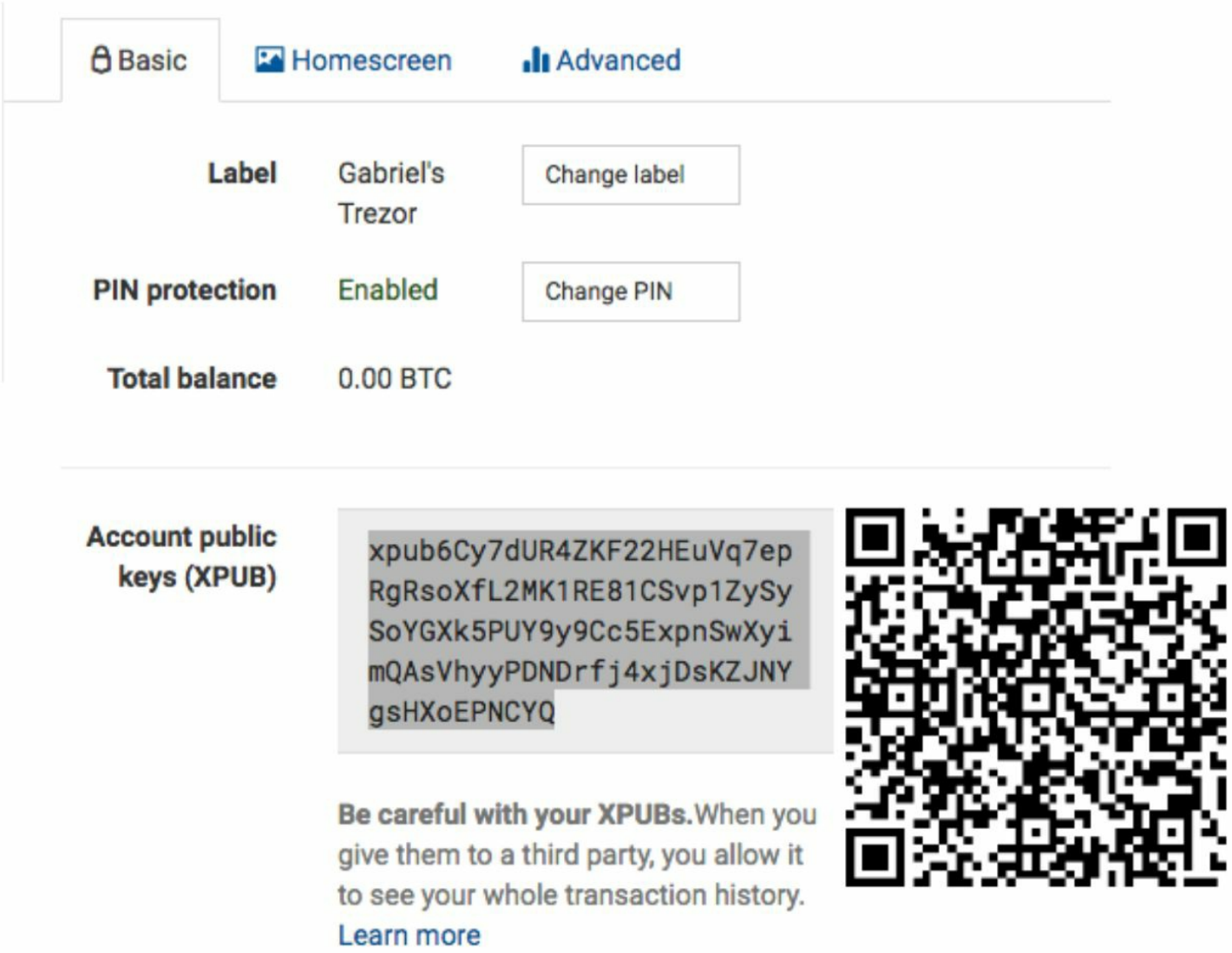


Figura 34. Esportare un xpub da un portafoglio hardware Trezor

Gabriel copia l'xpub nel software del suo negozio web dove accetta bitcoin. Usa *Mycelium Gear*, che è un plug-in

pensato per i web-store, open source e pensato per una varietà di piattaforme di hosting e contenuti web. Mycelium Gear utilizza xpub per generare un indirizzo univoco per ogni acquisto.

Derivazione di chiave figlia Hardened

La possibilità di derivare un ramo di chiavi pubbliche da una chiave pubblica estesa è molto utile, ma ha un rischio potenziale. L'accesso a una chiave pubblica estesa non consente l'accesso alle chiavi private secondarie. Tuttavia, poiché la chiave pubblica estesa contiene il codice catena, se una chiave privata figlio è nota o in qualche modo trapelata, può

essere utilizzata con il codice catena per derivare tutte le altre chiavi private secondarie. Una singola chiave privata del figlio trapelata, insieme a un codice di catena padre, rivela tutte le chiavi private di tutti i figli. Peggio ancora, la chiave privata figlio insieme a un codice catena padre può essere utilizzata per dedurre la chiave privata genitore.

Per contrastare questo rischio, i wallet HD utilizzano una funzione di derivazione alternativa chiamata *hardened derivation*, che "interrompe" la relazione tra la chiave pubblica padre e il codice della catena figlio. La funzione di derivazione consolidata utilizza la chiave privata padre per

derivare il codice catena figlio, anziché la chiave pubblica padre. Questo crea un "firewall" nella sequenza genitore / figlio, con un codice di catena che non può essere usato per compromettere una chiave privata genitore o fratello. La funzione di hardened derivation sembra quasi identica alla normale derivazione della chiave privata figlio, tranne per il fatto che la chiave privata genitore viene utilizzata come input per la funzione hash, invece della chiave pubblica padre, come mostrato nel diagramma in Derivazione hardened di una chiave figlio; omette la chiave pubblica padre.

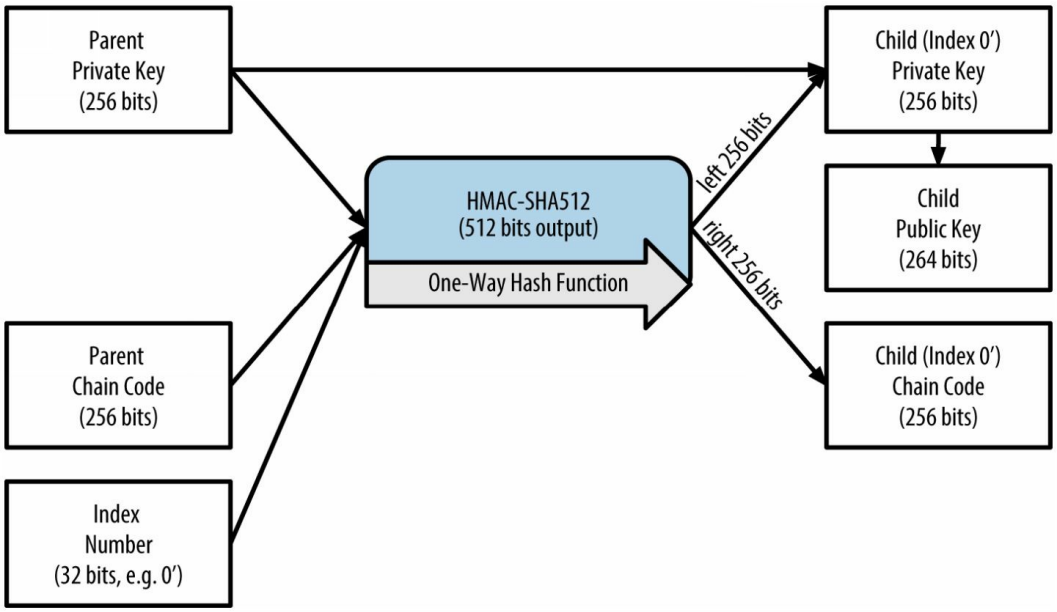


Figura 35. Derivazione hardened di una chiave figlio; omette la chiave pubblica padre

Quando la funzione di derivazione delle chiavi private hardened è utilizzata, la chiave privata figlia e il chain code sono completamente diversi da quello che risulterebbe da una normale funzione di derivazione. Il

"ramo" risultante di chiavi può essere usato per produrre chiavi pubbliche estese che non siano vulnerabili, perché il chain code contenuto in esse non può essere compromesso per rivelare alcuna chiave privata. La derivazione delle chiavi hardened è quindi utilizzata per creare uno "spazio" nell'albero precedente dove le chiavi pubbliche estese sono usate.

In termini semplici, se si desidera utilizzare la convenienza di una chiave pubblica estesa per derivare rami di chiavi pubbliche, senza esporti al rischio di un codice di catena trapelato, si dovrebbe derivare da un genitore hardened, piuttosto che da un genitore normale. Come best practice,

i figli di livello 1 delle chiavi master derivano sempre dalla hardened derivation, per impedire la compromissione delle chiavi master.

Numeri di indice per derivazioni normali e hardened

Il numero indice utilizzato nella funzione di derivazione è un numero intero a 32 bit. Per distinguere facilmente tra le chiavi derivate dalla normale funzione di derivazione rispetto alle chiavi derivate dalla hardened derivation, questo numero indice è diviso in due intervalli. I numeri indice compresi tra 0 e 2 ^ 31 ^ -1 (da 0x0 a 0x7FFFFFFF) sono

utilizzati *solo* per la derivazione normale. I numeri di indice compresi tra 231 e 232-1 (da 0x80000000 a 0xFFFFFFFF) sono utilizzati *solo* per la hardened derivation. Pertanto, se il numero indice è inferiore a 231, ciò significa che il figlio è normale, mentre se il numero indice è uguale o superiore a 231, il figlio è hardened.

Per rendere il numero di indice più facile da leggere e mostrare, viene visualizzato il numero di indice per i figli hardened a partire da zero, ma con il simbolo di primo. La prima chiave figlio è quindi visualizzata come 0, mentre il primo figlio hardened (con indice 0x80000000) viene visualizzato come 0'. In

sequenza poi, il secondo figlio hardened avrebbe l'indice valorizzato a 0x80000001 e verrebbe visualizzato come 1', e così via. Quando si vede un indice portafogli HD i, significa 231i.

Identificatore di una chiave di un wallet HD (path, percorso)

Le chiavi in un wallet HD vengono identificate utilizzando una convenzione di denominazione "path", con ogni livello dell'albero separato da un carattere barra (/) (vedere Esempi di path in un wallet HD). Le chiavi private derivate dalla chiave privata principale iniziano con "m". Le chiavi pubbliche derivate dalla chiave

pubblica principale iniziano con "M". Pertanto, la prima chiave privata secondaria della chiave privata principale è m/0. La prima chiave pubblica figlio è M/0. Il secondo nipote del primo figlio è m/0/1, e così via.

La "discendenza" di una chiave si può leggere da destra verso sinistra, fino a che non si raggiunge la master key dalla quale è stata derivata. Per esempio, l'identificatore m/x/y/z descrive la chiave che è la z-esima figlia della chiave m/x/y, la quale è la y-esima figlia della chiave m/x, che è la x-esima figlia di m.

Tabella 15. Esempi di path in u
wallet HD

path HD	Chiave descritta
m/0	La chiav privata de primo figlio (0 dalla chiav privata maste (m)
m/0/0	La prima chiav privata nipote figlia del prim figlio (m/0)
m/0'/0	La prima nipote figlia del prim figlio *hardene*

	(m/0)
m/1/0	La prima chiav privata "nipote derivata da secondo "figlio (m/1)
M/23/17/0/0	La prima chiav pubblica pro pro-pronipote del prim pronipote de 18° nipote de 24° figlio

Navigando la struttura ad

albero degli HD wallet

La struttura ad albero del wallet HD offre un'enorme flessibilità. Ogni chiave estesa genitore può avere 4 miliardi di figli: 2 miliardi di figli normali e 2 miliardi di hardened children (n.d.r. induriti, temprati). Ognuno di questi figli può avere altri 4 miliardi di figli, e così via. L'albero può essere profondo quanto vuoi, con un numero infinito di generazioni. Con tutta quella flessibilità, tuttavia, diventa abbastanza difficile navigare su questo albero infinito. È particolarmente difficile trasferire i wallet HD tra le implementazioni, poiché le possibilità di organizzazione interna in filiali e sottobranchi sono

infinite.

Due proposte di miglioramento dei bitcoin (BIP) offrono una soluzione a questa complessità, creando alcuni standard proposti per la struttura degli alberi del wallet HD. BIP-43 propone l'uso del primo indice figlio hardened come identificatore speciale che indica lo "scopo" della struttura ad albero. Sulla base di BIP-43, un wallet HD deve utilizzare solo un ramo di livello 1 dell'albero, con un indice che identifica la struttura e lo spazio dei nomi del resto dell'albero definendone lo scopo. Ad esempio, un portafoglio HD che utilizza solo il ramo m/i'/ è destinato a significare uno scopo specifico e tale scopo è identificato

dal numero di indice "i".

Estendendo tale specifica, BIP-44 propone una struttura multi account come numero "scopo" 44 sotto BIP-43. Tutti i wallet HD che seguono la struttura BIP-44 sono identificati dal fatto che hanno usato solo un ramo dell'albero: m/44/.

BIP-44 specifica la struttura consistente di cinque livelli ad albero predefiniti:

```
m / purpose' / coin_type' / account' / change / address_index
```

Lo "scopo" di primo livello è sempre impostato su 44. Il "coin_type" di secondo livello specifica il tipo di criptovaluta, consentendo wallet HD

multivaluta in cui ogni valuta ha il proprio sottoalbero sotto il secondo livello. Ci sono tre valute definite per ora: Bitcoin è m/44/0, Bitcoin Testnet è: m/44/1; e Litecoin è: m/44/2.

Il terzo livello dell'albero è l'account che consente agli utenti di suddividere i loro portafogli in sottoconti con logiche separate, per scopi contabili o organizzativi. Ad esempio, un wallet HD potrebbe contenere due "account" bitcoin pass: m/44/0/0 e pass: m/44/0/1. Ogni account è il root della propria sottostruttura.

Al quarto livello, "change", un portafoglio HD ha due sottoalberi, uno per la creazione di indirizzi di ricezione e uno per la creazione di

indirizzi di cambiamento. Si noti che mentre i livelli precedenti utilizzavano la hardened derivation, questo livello utilizza la derivazione normale. Questo per consentire a questo livello dell'albero di esportare chiavi pubbliche estese da utilizzare in un ambiente non protetto. Gli indirizzi utilizzabili sono derivati dal portafoglio HD come figli del quarto livello, rendendo il quinto livello dell'albero "address_index". Ad esempio, il terzo indirizzo di ricezione per i pagamenti bitcoin nell'account principale sarebbe M/44/0/0/0/2. BIP-44 HD wallet structure examples mostra alcuni altri esempi.

Tabella 16. BIP-44 Esempi della s

HD wallet

path HD
M/44'/0'/0'/0/2
M/44'/0'/3'/1/14

m/44'/2'/0'/0/1

Transazioni

Introduzione

Le transazioni sono la parte più importante del sistema bitcoin. Tutto il resto in bitcoin è progettato per assicurare che le transazioni possano essere create, propagate nel network, validate e infine aggiunte al registro globale delle transazioni (la blockchain). Le transazioni sono strutture dati che codificano il valore del trasferimento tra partecipanti nel sistema bitcoin. Ogni transazione è un dato pubblico nella blockchain di bitcoin, il registro contabile globale a partita doppia (double-entry

bookkeeping).

In questo capitolo esamineremo le varie forme di transazione, cosa contengono, come crearle, come sono verificate e come diventano parte del registro permanente di tutte le transazioni. Quando usiamo il termine "portafoglio" o “wallet” in questo capitolo, ci riferiamo al software che costruisce le transazioni, non solo il database delle chiavi.

Transazioni in Dettaglio

I n Come Funziona Bitcoin, abbiamo esaminato la transazione con cui Alice pagava il caffè al coffee shop di Bob

usando un block explorer (La transazione di Alice al Bar di Bob).

L'applicazione block explorer mostra una transazione dall'"indirizzo" di Alice all'"indirizzo" di Bob. Questa è una visione molto semplificata di ciò che è contenuto in una transazione. In effetti, come vedremo in questo capitolo, gran parte delle informazioni mostrate sono costruite da Block Explorer e non sono effettivamente nella transazione.

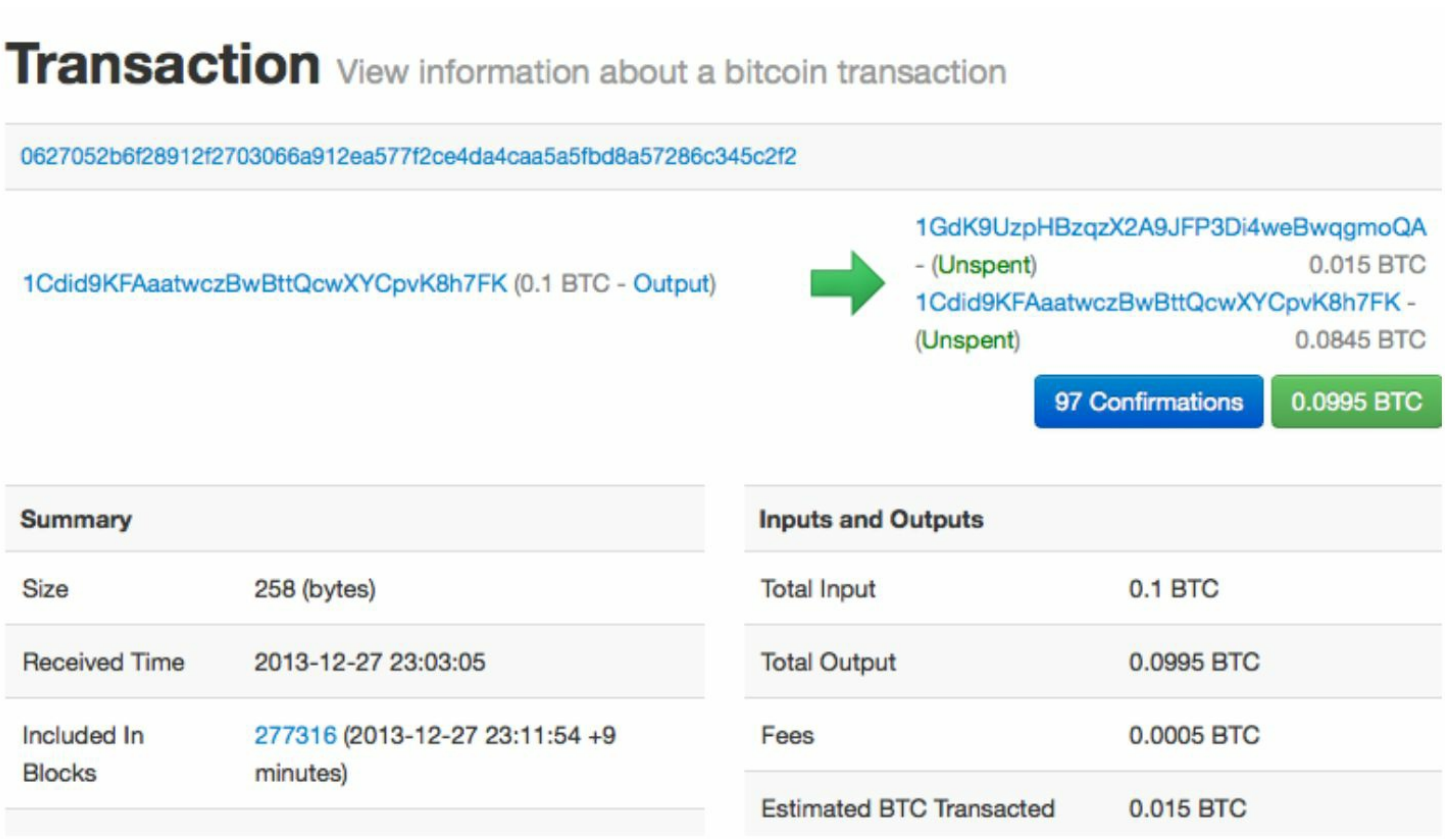


Figura 36. La transazione di Alice al Bar di Bob

Transazioni — Dietro le Scene

Dietro le quinte, una transazione reale sembra molto diversa da una transazione fornita da un tipico block explorer. In effetti, la maggior parte

dei costrutti di alto livello che vediamo nelle varie interfacce utente di applicazioni bitcoin non esistono realmente nel sistema bitcoin.

Possiamo utilizzare l'interfaccia della riga di comando di Bitcoin Core (getrawtransaction e decoderawtransaction) per recuperare la transazione "raw" di Alice, decodificarla e vedere cosa contiene. Il risultato è simile a questo:

La transazione di Alice decodificata

```
{
 "version": 1,
 "locktime": 0,
 "vin": [
   {
```

```
      "txid":
"7957a35fe64f80d234d76d83a2a8f1a0d814
      "vout": 0,
      "scriptSig" :
"3045022100884d142d86652a3f47ba4746e
0484ecc0d46f1918b30928fa0e4ed99f16a0f
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.01500000,
      "scriptPubKey": "OP_DUP
OP_HASH160
ab68025513c3dbd2f7b92a94e0581f5d50f65
OP_EQUALVERIFY OP_CHECKSIG"
    },
    {
      "value": 0.08450000,
      "scriptPubKey": "OP_DUP
OP_HASH160
```

```
7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025
OP_EQUALVERIFY OP_CHECKSIG",
    }
  ]
}
```

Potresti notare alcune cose su questa transazione, soprattutto le cose che mancano! Dov'è l'indirizzo di Alice? Dov'è l'indirizzo di Bob? Dov'è l'input 0.1 "inviato" da Alice? In bitcoin non ci sono monete, mittenti, destinatari, nessun saldo, nessun account e nessun indirizzo. Tutte queste cose sono costruite a un livello più alto a vantaggio dell'utente, per rendere le cose più facili da capire.

Potresti anche notare molti campi strani e indecifrabili e stringhe

esadecimali. Non preoccuparti, spiegheremo ogni campo mostrato qui in dettaglio in questo capitolo.

Output e Input di una Transazione

Il componente fondamentale di una transazione bitcoin è un *output di transazione*. Gli output di transazione sono blocchi indivisibili di valuta bitcoin, registrati sulla blockchain e riconosciuti come validi dall'intera rete. I nodi completi di Bitcoin tracciano tutti gli output disponibili e spendibili, noti come *output di transazione non spesi* o *UTXO*. La raccolta di tutti gli UTXO è nota come

set UTXO e attualmente è di svariati milioni di UTXO. Il set UTXO cresce man mano che viene creato UTXO e si riduce quando UTXO viene utilizzato. Ogni transazione rappresenta un cambiamento (transizione di stato) nel set UTXO.

Quando diciamo che il portafoglio di un utente ha "ricevuto" bitcoin, ciò che intendiamo è che il portafoglio ha rilevato un UTXO che può essere speso con uno dei tasti controllati da quel portafoglio. Quindi, il "saldo" bitcoin di un utente è la somma di tutti gli UTXO che il portafoglio dell'utente può spendere e che può essere sparpagliato tra centinaia di transazioni e centinaia di blocchi. Il

concetto di equilibrio viene creato dall'applicazione wallet. Il wallet calcola il saldo dell'utente analizzando la blockchain e aggregando il valore di qualsiasi UTXO che il wallet può spendere con le chiavi che controlla. La maggior parte dei wallet gestisce un database o utilizza un servizio di database per memorizzare un set di riferimento rapido di tutti gli UTXO che possono utilizzare con le chiavi che controllano.

Una UTXO può avere un valore arbitrario denominato in multipli di satoshi. Proprio come i dollari che possono essere divisi alla seconda cifra decimale come i centesimi, i bitcoin possono essere divisi fino

all'ottava cifra decimale come i satoshi. Sebbene un output possa avere qualsiasi valore arbitrario, una volta creato è indivisibile. Questa è una caratteristica importante degli output che devono essere enfatizzati: gli output sono unità di valore *discrete* e *indivisibili*, denominate in satosì interi. Un output non speso può essere interamente consumato da una transazione.

Se un UTXO è più grande del valore desiderato della transazione, deve tuttavia essere consumato nella sua interezza e il resto deve essere generato nella transazione. In altre parole, se hai una UTXO da 20 bitcoin e vuoi pagare 1 bitcoin, la tua

transazione deve consumare tutto l'UTXO da 20 bitcoin e produrre due output: uno che paga 1 bitcoin al tuo destinatario desiderato e un altro che paga 19 bitcoin come resto indietro al tuo wallet. Come risultato, la maggior parte delle transazioni bitcoin genereranno resto.

Immagina un cliente che compra una bevanda da 1.50$, prendendo il suo wallet e provando a trovare una combinazione di monete e banconote per coprire il costo da 1.50$. Il cliente sceglierà l'esatto resto se disponibile (una banconota da un dollaro e due da 25 cent), o una combinazione di tagli più piccoli (sei monete da 25cent), o se necessario, una unità più grande

quale una banconota da cinque dollari. Se si da troppa moneta, come per esempio 5$, al proprietario del negozio, ci si aspetterà di ricevere 3.50$ di resto, i quali ritorneranno al suo wallet e in questo modo lei li avrà disponibili per transazioni future.

Nello stesso modo, una transazione bitcoin deve essere creata da una UTXO di un'utente in qualsiasi taglio questo utente abbia a disposizione. Gli utenti non possono tagliare una UTXO a metà così come non non possono tagliare una banconota da un dollaro a metà e utilizzarla come moneta. L'applicazione wallet dell'utente selezionerà solitamente dagli UTXO disponibili da parte dell'utente varie

unità per comporre una somma più grande o uguale del valore di transazione desiderato.

Come nella vita reale, l'applicazione bitcoin può usare diverse strategie per soddisfare la somma da trasferire (in questo caso l'acquisto): combinando diverse unità più piccole, trovando il resto giusto, o usando una singola unità più grande del valore della transazione e creare il resto. Tutto questo complesso assemblaggio di UTXO spendibili è fatto dal wallet dell'utente automaticamente ed è invisibile agli utenti. E' solo rilevante se stai costruendo programmaticamente transazioni raw da UTXO.

Una transazione consuma UTXO

precedentemente registrate e crea nuovi output di transazione che possono essere consumati da una transazione futura. In questo modo, blocchi di valore bitcoin vanno avanti da proprietario a proprietario in una catena di transazioni che consumano e creano UTXO.

L'eccezione alla catena di output e input è un tipo speciale di transazione chiamata la transazione *coinbase*, che è la prima transazione in ogni blocco. Questa transazione è immessa li (nella blockchain) dal miner "vincitore" e crea bitcoin "nuovi-di-zecca" pagabili a quel miner come ricompensa per aver effettuato il lavoro di mining. Questo è il modo (e l'unico modo,

ndt.) in cui la massa monetaria (money supply) di bitcoin viene creata durante il processo di mining, come vedremo nel Il Mining e Il Consenso.

TIP	Cosa viene per primo? Gli input o gli output l'uovo o la gallina? Tecnicamente, gli output vengono prima a causa delle transazioni coinbase, le quali generano nuovi bitcoin esse non hanno input e creano output dal niente.

Output della Transazione

Ogni transazione bitcoin crea output,

che sono registrati sul registro di bitcoin (ledger). Quasi tutti questi output, con una eccezione (vedi Data Recording Output (RETURN)) creano pezzi spendibili di bitcoin chiamati unspent transaction outputs o UTXO (output di transazione non spesi), i quali sono poi riconosciuti da tutto il network e disponibili per il proprietario per spenderli in una futura transazioni. Inviare a qualcuno bitcoin equivale a creare un output di una transazione unspent (non spesa), cioè un UTXO, registrato al loro indirizzo e disponibile per la loro spesa.

Gli UTXO sono tracciati da ogni client bitcoin full-node come serie di dati chiamati la UTXO set o UTXO pool,

salvati in un database. Le nuove transazioni consumano (spendono) uno o più di questi output dal set di UTXO.

Gli output di una Transazione consistono di due parti:

- Un importo di bitcoin, denominati in *satoshi*, l'unità più piccola di bitcoin
- Un puzzle crittografico che determina le condizioni richieste per spendere l'output

Il puzzle crittografico è anche conosciuto come un *locking script*, un *witness script* o uno scriptPubKey.

Il linguaggio di scripting delle transazioni, usato nel locking script

menzionato precedentemente, è discusso in dettaglio in Script di Transazione e Linguaggio Bitcoin Script.

Ora, diamo un'occhiata alla transazione di Alice (mostrata precedentemente in Transazioni — Dietro le Scene) e vediamo se siamo in grado di identificare gli output. Nella codifica JSON, gli output sono in un array (lista) di nome vout:

```
"vout": [
  {
    "value": 0.01500000,
    "scriptPubKey": "OP_DUP
OP_HASH160
ab68025513c3dbd2f7b92a94e0581f5d50f65
OP_EQUALVERIFY
    OP_CHECKSIG"
```

```
  },
  {
    "value": 0.08450000,
    "scriptPubKey": "OP_DUP
OP_HASH160
7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025
OP_EQUALVERIFY OP_CHECKSIG",
  }
]
```

Come puoi vedere, la transazione contiene due output. Ogni output è definito da un valore e un puzzle crittografico. Nella codifica mostrata da Bitcoin Core, il valore è mostrato in bitcoin, ma nella transazione stessa è registrato come un intero denominato in satoshi. La seconda parte di ogni output è il puzzle crittografico che stabilisce le condizioni per la spesa.

Bitcoin Core lo mostra come scriptPubKey e ci mostra una rappresentazione leggibile dall'uomo della sceneggiatura.

Il tema del blocco e dello sblocco di UTXO sarà discusso più avanti, in Costruzione dello Script (Lock + Unlock). Il linguaggio di scripting che viene utilizzato per lo script in scriptPubKey è discusso in Script di Transazione e Linguaggio Bitcoin Script. Ma prima di approfondire questi argomenti, dobbiamo comprendere la struttura generale degli input e degli output delle transazioni.

Transaction serialization—outputs

Quando le transazioni vengono trasmesse sulla rete o scambiate tra applicazioni, vengono *serializzate*. La serializzazione è il processo di conversione della rappresentazione interna di una struttura di dati in un formato che può essere trasmesso un byte alla volta, noto anche come flusso di byte. La serializzazione è più comunemente utilizzata per codificare strutture di dati per la trasmissione su una rete o per la memorizzazione in un file. Il formato di serializzazione di un output di transazione è mostrato in Tabella 17.

Tabella 17. Transaction outp serialization

Dimensione	Campo	Descrizion
8 byte (little-endian)	Quantità	Valore Bitcoin satoshi (1 8 bitcoin)
1–9 byte (VarInt)	Locking-Script Size	Lunghezza Locking-Script byte, seguire
Variabile	Script di Locking	Uno scri che definisce condizioni necessarie per spendere

		l'output

La maggior parte delle librerie e dei framework di bitcoin non memorizzano le transazioni internamente come flussi di byte, poiché ciò richiederebbe un parsing complesso ogni volta che fosse necessario accedere a un singolo campo. Per comodità e leggibilità, le librerie di bitcoin memorizzano le transazioni internamente in strutture di dati (di solito strutture orientate agli oggetti).

Il processo di conversione dalla rappresentazione del flusso di byte di una transazione alla struttura di dati di rappresentazione interna di una

libreria è chiamato *deserializzazione* o *analisi delle transazioni*. Il processo di conversione in un flusso di byte per la trasmissione in rete, per l'hashing o per l'archiviazione su disco è chiamato *serializzazione*. La maggior parte delle librerie di bitcoin ha funzioni integrate per la serializzazione delle transazioni e la deserializzazione.

Verifica se puoi decodificare manualmente la transazione di Alice dalla forma esadecimale serializzata, trovando alcuni degli elementi che abbiamo visto in precedenza. La sezione contenente i due output è evidenziata in Esempio 16 per aiutarti:

Esempio 16. La transazione di Alice, serializzata e presentata in formato esadecimale

```
0100000001186f9f998a5aa6f048e51
4d2804fe65fa35779000000008b483
ba4746ec719bbfbd040a570b1deccb
ff08df09cbe9f6addac960298cad530a
01410484ecc0d46f1918b30928fa0e4
16ab9fe423cc5412336376789d17278
7b4a10fa336a8d752adfffffffff02 60e3
8025513c3dbd2f7b92a94e0581f5d5
1976a9147f9b1a7fb68d60c536c2fd8
00000000
```

Ecco alcuni suggerimenti:

- Ci sono due output nella sezione evidenziata, ciascuno serializzato come mostrato nella serializzazione dell'output della transazione.
- Il valore di 0.015 bitcoin è 1,500,000 satoshi. Che equivale a 16 e3 60 in esadecimale.
- Nella transazione serializzata, il valore 16 e3 60 è codificato nell'ordine byte little-endian (least-significant-first-first), quindi assomiglia a 60 e3 16.
- La lunghezza di scriptPubKey è di 25

byte, che è 19 in esadecimale.

Input della Transazione

Gli input delle transazioni identificano (per riferimento) quale UTXO verrà utilizzato e forniranno la prova di proprietà attraverso uno script di sblocco.

Per costruire una transazione, un portafoglio seleziona dall'UTXO che controlla, UTXO con un valore sufficiente per effettuare il pagamento richiesto. A volte un UTXO è sufficiente, altre volte ne è necessario più di uno. Per ogni UTXO che verrà utilizzato per effettuare questo pagamento, il wallet crea un input che punta all'UTXO e lo sblocca con uno

script di sblocco.

Diamo un'occhiata ai componenti di un input in modo più dettagliato. La prima parte di un input è un puntatore a un UTXO in riferimento all'hash della transazione e a un indice di output, che identifica l'UTXO specifico in quella transazione. La seconda parte è uno script di sblocco, che il wallet costruisce per soddisfare le condizioni di spesa impostate nell'UTXO. Molto spesso, lo script di sblocco è una firma digitale e una chiave pubblica che dimostra la proprietà del bitcoin. Tuttavia, non tutti gli script di sblocco contengono firme. La terza parte è un numero di sequenza, che sarà discusso più avanti.

Considera il nostro esempio in Transazioni — Dietro le Scene. Gli input della transazione sono una matrice (lista) chiamata vin:

Gli input della transazione nella transazione di Alice


```
"vin": [
  {
    "txid":
"7957a35fe64f80d234d76d83a2a8f1a0d814
    "vout": 0,
    "scriptSig" :
"3045022100884d142d86652a3f47ba4746e
0484ecc0d46f1918b30928fa0e4ed99f16a0f
    "sequence": 4294967295
  }
]
```


Come puoi vedere, c'è un solo input

nell'elenco (perché un UTXO conteneva un valore sufficiente per effettuare questo pagamento). L'input contiene quattro elementi:

- Un ID transazione, che fa riferimento alla transazione che contiene l'UTXO che viene speso
- Un indice di output (vout), che identifica quale UTXO da quella transazione è referenziato (il primo è zero)
- Un scriptSig, che soddisfa le condizioni poste sull'UTXO, sbloccandolo per la spesa

- Un numero di sequenza (che discuteremo in seguito)

Nella transazione di Alice, l'input punta all'ID della transazione:

```
7957a35fe64f80d234d76d83a2a8f1a0d8149
```

e l'indice di output 0 (cioè il primo UTXO creato da quella transazione). Lo script di sblocco è costruito dal portafoglio di Alice recuperando prima l'UTXO di riferimento, esaminando lo script di blocco e quindi utilizzandolo per creare lo script di sblocco necessario per soddisfarlo.

Guardando solo l'input potresti aver notato che non sappiamo nulla di

questo UTXO, a parte un riferimento alla transazione che lo contiene. Non conosciamo il suo valore (importo in satoshi), e non conosciamo lo script di blocco che stabilisce le condizioni per spenderlo. Per trovare queste informazioni, dobbiamo recuperare l'UTXO di riferimento recuperando la transazione sottostante. Si noti che poiché il valore dell'input non è esplicitamente dichiarato, dobbiamo anche utilizzare l'UTXO di riferimento per calcolare le commissioni che verranno pagate in questa transazione (vedi Commissioni sulla Transazione (transaction fee)).

Non è solo il portafoglio di Alice che deve recuperare UTXO di riferimento

negli input. Una volta che questa transazione viene trasmessa alla rete, ogni nodo di convalida dovrà anche recuperare l'UTXO a cui si fa riferimento negli input della transazione per convalidare la transazione.

Le transazioni da sole sembrano incomplete perché mancano di contesto. Fanno riferimento all'UTXO nei loro input ma senza recuperare quello UTXO non possiamo conoscere il valore degli input o le loro condizioni di blocco. Quando si scrive software bitcoin, ogni volta che si decodifica una transazione con l'intento di convalidarla o di contare le tasse o di controllare lo script di

sblocco, il codice dovrà prima recuperare l'UTXO di riferimento dalla blockchain per creare il contesto implicito ma non presente nei riferimenti UTXO degli input. Ad esempio, per calcolare l'importo pagato in tasse, è necessario conoscere la somma dei valori di input e output. Ma senza recuperare l'UTXO referenziato negli input, non si conosce il loro valore. Quindi un'operazione apparentemente semplice come il conteggio delle tasse in una singola transazione, infatti, comporta più passaggi e dati da più transazioni.

Possiamo usare la stessa sequenza di comandi con Bitcoin Core che abbiamo usato per recuperare la

transazione di Alice (getrawtransaction e decoderawtransaction). Con ciò possiamo ottenere l'UTXO referenziato nell'input precedente e dare un'occhiata:

L'UTXO di Alice dalla transazione precedente, referenziato nell'input

```
"vout": [
   {
    "value": 0.10000000,
    "scriptPubKey": "OP_DUP
OP_HASH160
7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025
OP_EQUALVERIFY OP_CHECKSIG"
   }
 ]
```

Vediamo che questo UTXO ha un valore di 0.1 BTC e che ha uno script di blocco (scriptPubKey) che contiene "OP_DUP OP_HASH160...".

TIP	Per comprendere appieno la transazione di Alice, abbiamo dovuto recuperare le transazioni precedenti a cui si fa riferimento come input. Una funzione che recupera le transazioni precedenti e gli output delle transazioni non spesi è molto comune ed esiste in quasi tutte le librerie e API di bitcoin.

Serializzazione delle transazioni — input

Quando le transazioni sono serializzate per la trasmissione sulla rete, i loro ingressi sono codificati in un flusso di byte come mostrato in Tabella 18.

Tabella 18. Transaction input serial

Dimensione	Campo	Descri
32 byte	Hash della Transazione	Puntato transaz contene l'UTX andrà s

4 byte	Indice dell'Output	Il indice dell'UT speso; è 0
1–9 byte (VarInt)	Unlocking-Script Size	Lunghe byte script sblocc seguire
Variabile	Unlocking-Script	Uno sc soddisf condizi dello s blocco
4 byte	Numero	Utilizza

	sequenziale	locktim disabil (0xFFF

Come per gli output, vediamo se possiamo trovare gli input dalla transazione di Alice nel formato serializzato. Innanzitutto, gli input decodificati:


```
"vin": [
  {
    "txid":
"7957a35fe64f80d234d76d83a2a8f1a0d814
    "vout": 0,
    "scriptSig" :
"3045022100884d142d86652a3f47ba4746e
0484ecc0d46f1918b30928fa0e4ed99f16a0f
    "sequence": 4294967295
  }
```

```
],
```

Ora, vediamo se siamo in grado di identificare questi campi nella codifica esadecimale serializzata in Esempio 17:

Esempio 17. La transazione di Alice, serializzata e presentata in notazione esadecimale

```
0100000001186f9f998a5aa6f048e51
4d2804fe65fa35779000000008b483
ba4746ec719bbfbd040a570b1deccb
ff08df09cbe9f6addac960298cad530
01410484ecc0d46f1918b30928fa0e4
16ab9fe423cc5412336376789d1727
```

7b4a10fa336a8d752adfffffffff0260e
8025513c3dbd2f7b92a94e0581f5d50
1976a9147f9b1a7fb68d60c536c2fd8
000

Suggerimenti:

- L'ID della transazione viene serializzato in ordine di byte invertito, quindi inizia con (hex) 18 e termina con 79
- L'indice di output è un gruppo di zeri a 4 byte, facile da identificare
- La lunghezza dello scriptSig è 139 byte, o 8b in esadecimale

- Il numero di sequenza è impostato su FFFFFFFF, di nuovo facile da identificare

Commissioni sulla Transazione (transaction fee)

La maggior parte delle transazioni include commissioni di transazione, che compensano i miner bitcoin, per garantire la sicurezza della rete. Le commissioni fungono anche da meccanismo di sicurezza per sé, rendendo economicamente impossibile per gli aggressori inondare la rete di transazioni. Il mining, le fee e i reward raccolti dai minatori sono discussi in modo più dettagliato in Il Mining e Il

Consenso.

Questa sezione esamina come le commissioni di transazione sono incluse in una transazione tipica. La maggior parte dei wallet calcola e include automaticamente le commissioni di transazione. Tuttavia, se si stanno costruendo transazioni a livello di programmazione o si utilizza un'interfaccia a riga di comando, è necessario eseguire manualmente il conteggio e includerle.

Le spese di transazione servono da incentivo per includere (minare) una transazione nel blocco successivo e anche per disincentivare le transazioni "spam" o qualsiasi tipo di abuso del sistema, imponendo un piccolo costo

su ogni transazione. Le spese di transazione sono accreditate al minatore che mina il blocco e registra la transazione sulla blockchain.

Le commissioni di transazione sono calcolate in base alla dimensione della transazione in kilobyte, non al valore della transazione in bitcoin. Nel complesso, le commissioni di transazione sono stabilite in base alle forze di mercato all'interno della rete bitcoin. I miner danno la priorità alle transazioni in base a molti criteri diversi, comprese le commissioni, e possono persino elaborare transazioni gratuitamente in determinate circostanze. Le commissioni di transazione influiscono sulla priorità

di elaborazione, il che significa che una transazione con commissioni sufficienti sarà probabilmente inclusa nel blocco più vicino, mentre una transazione con commissioni insufficienti o nulle potrebbe essere ritardata, elaborata su una base di sforzo massimo dopo alcuni blocchi, o non elaborata affatto. Le commissioni di transazione non sono obbligatorie e le transazioni senza commissioni potrebbero essere elaborate alla fine; tuttavia, includere le commissioni di transazione, incoraggia l'elaborazione prioritaria.

Nel tempo, il modo in cui le commissioni di transazione vengono calcolate e l'effetto che hanno sulla

priorità delle transazioni si è evoluto. Inizialmente, le commissioni di transazione erano fisse e costanti in tutta la rete. Gradualmente, la struttura delle commissioni è stata allentata in modo che potesse essere influenzata dalle forze di mercato, in base alla capacità della rete e al volume delle transazioni. Almeno dall'inizio del 2016, i limiti di capacità in bitcoin hanno creato una competizione tra le transazioni, con conseguenti commissioni più elevate e rendendo effettivamente le transazioni gratuite un ricordo del passato. Le transazioni a tariffa zero o a tariffa molto bassa raramente vengono estratte e talvolta non vengono nemmeno propagate

attraverso la rete.

In Bitcoin Core, le politiche di pagamento delle commissioni sono impostate dall'opzione minrelaytxfee. L'attuale valore predefinito di minrelaytxfee è 0.00001 bitcoin o un centesimo di millibitcoin per kilobyte. Pertanto, per impostazione predefinita, le transazioni con una tariffa inferiore a 0.00001 bitcoin sono considerate gratuite e vengono inoltrate solo se c'è spazio nel programma; altrimenti, vengono abbandonate. I nodi Bitcoin possono ignorare il criterio di addebito tariffario predefinito regolando il valore di minrelaytxfee.

Qualsiasi servizio bitcoin che crea transazioni, inclusi portafogli, scambi,

applicazioni di vendita al dettaglio, ecc., d*eve* implementare tariffe dinamiche. Le tariffe dinamiche possono essere implementate tramite un servizio di stima delle commissioni di terze parti o con un algoritmo di stima delle commissioni integrato. Se non sei sicuro, inizia con un servizio di terze parti e man mano che acquisisci esperienza, progetta e implementa il tuo algoritmo se desideri rimuovere la dipendenza di terze parti.

Gli algoritmi di stima delle tariffe calcolano la tariffa appropriata, in base alla capacità e alle commissioni offerte dalle transazioni "concorrenti". Questi algoritmi vanno dal

semplicistico (costo medio o mediano nell'ultimo blocco) a sofisticato (analisi statistica). Stimano la tariffa necessaria (in satoshi per byte) che darà ad una transazione un'alta probabilità di essere selezionata e inclusa in un certo numero di blocchi. La maggior parte dei servizi offre agli utenti la possibilità di scegliere tariffe alte, medie o basse. Alta priorità significa che gli utenti pagano commissioni più elevate ma è probabile che la transazione venga inclusa nel blocco successivo. La priorità media e bassa significa che gli utenti pagano commissioni di transazione inferiori, ma le transazioni potrebbero richiedere molto più tempo

per confermare.

Molte applicazioni di wallet utilizzano servizi di terze parti per i calcoli delle commissioni. Un servizio popolare è *http://bitcoinfees.21.co*, che fornisce un'API e un grafico visivo che mostra la tariffa in satoshi/byte per priorità diverse.

	Le tariffe statiche non sono più praticabili sulla rete bitcoin. I portafogli che fissano le tariffe statiche produrranno un'esperienza utente scadente in quanto le transazioni saranno

TIP	spesso "bloccate" e non confermate. Gli utenti che non capiscono le transazioni e le commissioni di bitcoin sono costernati dalle transazioni "bloccate" perché pensano di aver perso i loro soldi.

Il grafico in Figura 37 mostra la stima in tempo reale delle tariffe in 10 incrementi di satoshi/byte e il tempo di conferma previsto (in minuti e numero di blocchi) per le transazioni con commissioni in ciascun intervallo. Per ogni intervallo di commissioni (ad es. 61-70 satoshi/byte), due barre

orizzontali mostrano il numero di transazioni non confermate (1405) e il numero totale di transazioni nelle ultime 24 ore (102.975), con commissioni in tale intervallo. Sulla base del grafico, la commissione ad alta priorità raccomandata in questo momento è di 80 satoshi/byte, una tassa che probabilmente determinerebbe l'estrazione della transazione nel blocco immediatamente successivo (zero block delay). Per prospettiva, la dimensione mediana della transazione è 226 byte, quindi la tariffa raccomandata per una dimensione della transazione sarebbe di 18.080 satoshi (0.00018080 BTC).

I dati di stima della tariffa possono essere recuperati tramite una semplice API REST HTTP, all'indirizzo https://bitcoinfees.21.co/api/v1/fees/rec
Ad esempio, sulla riga di comando utilizzando il comando curl:

Utilizzo dell'API di stima della tariffa

```
$ curl https://bitcoinfees.21.co/api/v1/fees/recomm

{"fastestFee":80,"halfHourFee":80,"hourFee":
```

L'API restituisce un oggetto JSON con la stima della commissione corrente per la conferma più rapida (quickFee), la conferma entro tre blocchi (halfHourFee) e sei blocchi (hourFee),

in satoshi per byte.

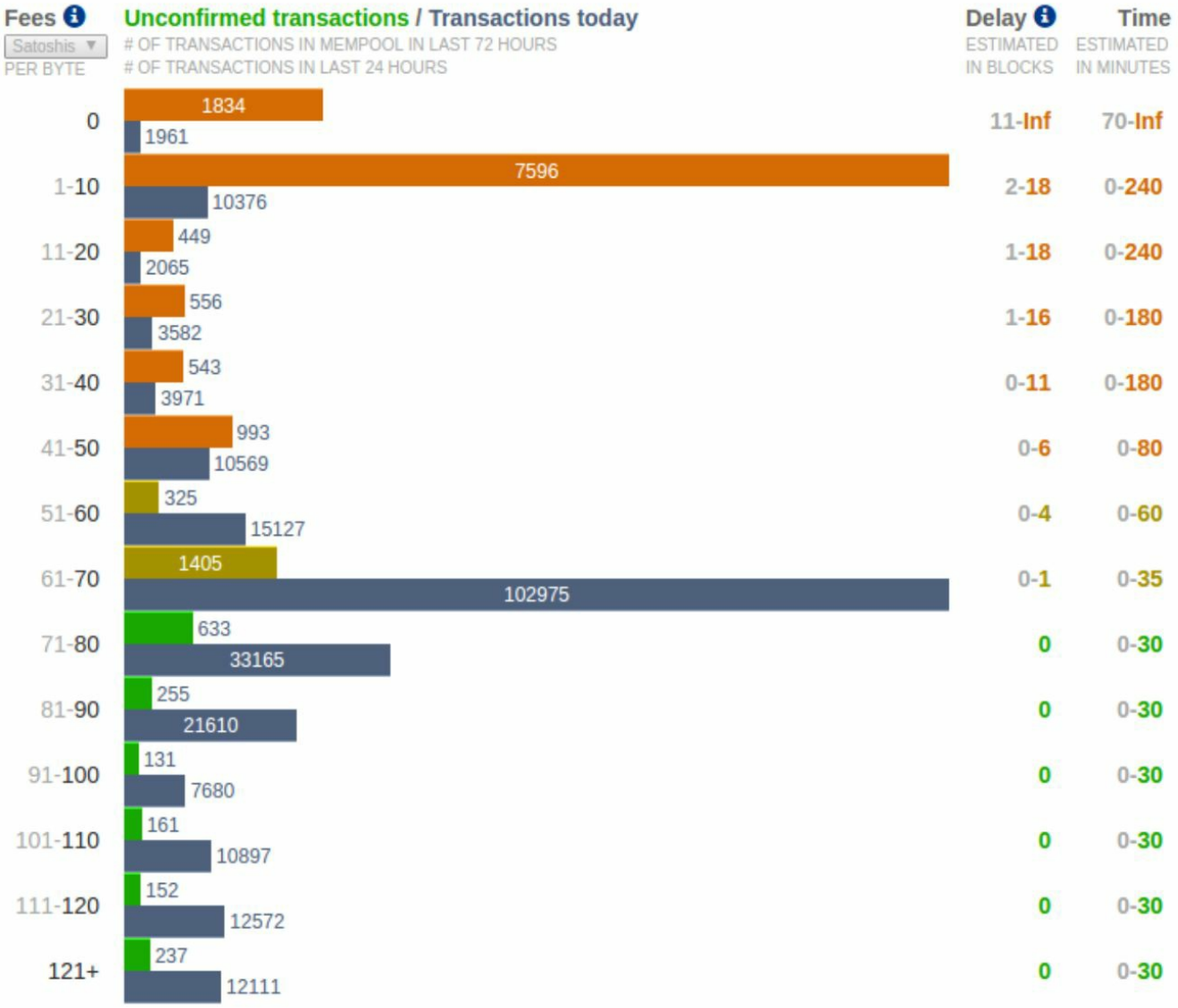


Figura 37. Servizio di stima delle tariffe bitcoinfees.21.co

Impostare le Commissioni sulle Transazioni

La struttura dati delle transazioni non ha un campo per le commissioni. Invece, le commissioni sono implicite come la differenza tra la somma degli input e la somma degli output. Qualsiasi importo in eccesso che rimane dopo che tutti gli output sono stati detratti da tutti gli input equivale alla commissione che viene destinata ai miner.

Le transaction fee sono l'eccesso degli input meno gli output:

```
Fees = Sum(Inputs) – Sum(Outputs)
```

Questo è un elemento un po' confuso delle transazioni e un punto importante da comprendere, perché se stai

costruendo le tue transazioni devi assicurarti di non includere inavvertitamente un compenso molto elevato esagerando con gli input. Ciò significa che devi conteggiare tutti gli input, se necessario modificandoli, o finirai per dare un mancia molto grande ai miner!

Ad esempio, se si utilizza un UTXO da 20 bitcoin per eseguire un pagamento da 1 bitcoin, è necessario includere 19 bitcoin di resto in output verso il tuo wallet . Altrimenti, il resto di 19 bitcoin verrà conteggiato come una commissione di transazione e verrà raccolto dal miner che minerà la tua transazione in un blocco. Sebbene tu possa ricevere un'elaborazione

prioritaria e rendere molto felice un miner, questo probabilmente non è ciò che intendevi fare.

ATTENZIONE	Se dimentichi di aggiungere un output a cui inviare il resto in una transazione costruita manualmente, il resto verrà attribuito come transaction fee. "Tieni il resto!" non è sempre quello

	che intendevi.

Vediamo come funziona in pratica, osservando nuovamente l'acquisto del caffè effettuato da Alice. Alice vuole spendere 0.015 bitcoin per pagare il caffè. Per assicurare che questa transazione sia processata rapidamente, Alice vorrà includere una fee di transazione, diciamo 0.001. Questo vorrà dire che il costo totale della transazione sarà di 0.016 bitcoin. Il suo wallet dovrà quindi recuperare una serie di UTXO che arrivi a 0.016 o più e, se necessario, emettere il resto. Diciamo che il suo wallet abbia 0.2 bitcoin disponibili nelle UTXO.

Dovrà quindi consumare queste UTXO, creare un output verso il Caffè di Bob da 0.015, e un secondo output con 0.184 bitcoin come resto re-inviati indietro al proprio wallet, lasciando 0.001 bitcoin non allocati, come fee implicita per la transazione.

Ora diamo un'occhiata ad uno scenario differente. Eugenia, la nostra direttrice dell'organizzazione no profit nelle Filippine, ha completato una campagna di raccolta fondi per comprare libri di scuola per i bambini. Ha ricevuto molte piccole donazioni provenienti da persone in tutto il mondo, totalizzando 50 bitcoin, quindi il suo wallet è pieno di molti pagamenti molto piccoli (UTXO). A questo punto Eugenia

vuole acquistare centinaia di libri di scuola da un editore locale.

L'applicazione wallet di Eugenia per costruire una transazione singola per il pagamento, deve recuperare un serie di UTXO disponibili, relativi a varie transazioni più piccole. Questo vuol dire che la transazione risultante recupererà da più di un centinaio di UTXO di poco valore per quanto riguarda l'input e emetterà un solo output, pagando l'editore del libro. Una transazione con così tanti input sarà più grande di un kilobyte, forse dai 2 ai 3 kilobyte di dimensione. Come risultato, richiederà una fee più alta della network fee minima di 0.0001 bitcoin.

L'applicazione wallet di Eugenia calcolerà le fee appropriate misurando la dimensione della transazione e moltiplicandola per la fee per kilobyte. Molti wallet pagano un valore di fee più alto per transazioni di dimensione maggiore per assicurarsi che la transazione sia eseguita rapidamente. Si ottiene una fee maggiorata non perchè Eugenia sta spendendo più soldi, invece è perchè la sua transazione è più complessa e di più grande dimensione—la fee è indipendente rispetto al valore in bitcoin della transazione.

Script di Transazione e Linguaggio Bitcoin

Script

Il linguaggio di script delle transazioni bitcoin, chiamato *Script*, è un linguaggio di esecuzione basato su notazione inversa di tipo Forth. Se questo suona come qualcosa di incomprensibile, probabilmente non hai studiato i linguaggi di programmazione degli anni '60, ma va bene - lo spiegheremo in questo capitolo. Sia lo script di blocco posto su un UTXO che lo script di sblocco sono scritti in questo linguaggio di scripting. Quando una transazione viene convalidata, lo script di sblocco in ciascun input viene eseguito insieme allo script di blocco corrispondente

per verificare se soddisfa la condizione di spesa.

Script è un linguaggio molto semplice che è stato progettato per essere limitato in ambito ed eseguibile su una gamma di hardware, forse semplice come un dispositivo incorporato. Richiede un'elaborazione minima e non può fare molte delle cose fantasiose che i moderni linguaggi di programmazione possono fare. Per il suo uso nella convalida dei soldi programmabili, questa è una funzione di sicurezza deliberata.

Oggi, la maggior parte delle transazioni processate attraverso il network bitcoin hanno la forma "Alice paga Bob" e sono basate sullo stesso

script chiamato script Pay-to-Public-Key-Hash. Tuttavia, le transazioni bitcoin non sono limitate allo script "Alice paga Bob". In effetti, gli script di blocco possono essere scritti per esprimere una grande varietà di condizioni complesse. Per comprendere questi script più complessi, dobbiamo prima capire le basi degli script di transazione e del linguaggio di script.

In questa sezione, mostreremo i componenti di base del linguaggio di scripting della transazione bitcoin e mostreremo come può essere usato per esprimere semplici condizioni di spesa e come queste condizioni possono essere soddisfatte sbloccando

gli script.

TIP	La convalida della transazione con Bitcoin non è basata su un modello statico, ma è ottenuta attraverso l'esecuzione di un linguaggio di scripting. Questo linguaggio consente di esprimere una varietà quasi infinita di condizioni. È così che bitcoin ottiene il potere del "denaro programmabile".

Incompletezza di Turing

Il linguaggio script della transazione bitcoin contiene molti operatori, ma è deliberatamente limitato in un modo importante: non ci sono loop o funzionalità di controllo del flusso complesse oltre al controllo del flusso condizionale. Ciò garantisce che il linguaggio non sia *Turing Complete*, il che significa che gli script hanno una complessità limitata e tempi di esecuzione prevedibili. Lo script non è un linguaggio generico. Queste limitazioni assicurano che il linguaggio non possa essere utilizzato per creare un loop infinito o altra forma di "bomba logica" che possa essere incorporata in una transazione in un modo che causi un attacco

denial-of-service contro la rete bitcoin. Ricorda, ogni transazione è convalidata da ogni nodo completo sulla rete bitcoin. Un linguaggio limitato impedisce che il meccanismo di convalida della transazione venga utilizzato come vulnerabilità.

Stateless Verification

Il linguaggio di script usato nella transazione bitcoin è senza stato, in quanto non esiste uno stato prima dell'esecuzione dello script o uno stato salvato dopo l'esecuzione dello script. Pertanto, tutte le informazioni necessarie per eseguire uno script sono contenute nello script. Uno script verrà eseguito prevedibilmente allo

stesso modo su qualsiasi sistema. Se il tuo sistema verifica uno script, puoi essere sicuro che ogni altro sistema nella rete bitcoin verificherà anche lo script, il che significa che una transazione valida è valida per tutti e tutti lo sanno. Questa prevedibilità dei risultati è un vantaggio essenziale del sistema bitcoin.

Costruzione dello Script (Lock + Unlock)

Il motore di validazione di transazioni bitcoin fa affidamento su due tipi di script per validare le transazioni: un locking script e uno script di unlock.

Uno script di blocco è posto in un encumbrance su un output e specifica

le condizioni che devono essere soddisfatte per poter spendere l'output in futuro. Storicamente, lo script di blocco era chiamato scriptPubKey, perché di solito conteneva una chiave pubblica o un indirizzo bitcoin. In questo libro ci riferiamo ad esso come uno "script di blocco" per riconoscere la gamma più ampia di possibilità di questa tecnologia di scripting. Nella maggior parte delle applicazioni bitcoin, quello che chiamiamo script di blocco apparirà nel codice sorgente come scriptPubKey. Vedrai anche lo script di blocco denominato a *witness script* (vedi Segregated Witness) o più generalmente *puzzle crittografico*. Questi termini significano la stessa

cosa, a diversi livelli di astrazione.

Uno script di sblocco è uno script che "risolve" o soddisfa le condizioni poste su un output da uno script di blocco e consente di utilizzare l'output. Gli script di sblocco fanno parte di ogni input di transazione e la maggior parte delle volte contengono una firma digitale prodotta dal portafoglio dell'utente dalla sua chiave privata. Storicamente, lo script di sblocco si chiama *scriptSig*, perché di solito conteneva una firma digitale. Nella maggior parte delle applicazioni bitcoin, il codice sorgente si riferisce allo script di sblocco come scriptSig. Vedrai anche lo script di sblocco denominato *witness* (vedi Segregated

Witness). In questo libro, ci riferiamo ad esso come uno "script di sblocco" per riconoscere la gamma molto più ampia di requisiti di script di blocco, perché non tutti gli script di sblocco devono contenere firme.

Ogni client bitcoin convaliderà le transazioni eseguendo insieme gli script di blocco e sblocco. Per ogni input nella transazione, il software di convalida recupererà prima l'UTXO a cui fa riferimento l'input. L'UTXO contiene uno script di blocco che definisce le condizioni richieste per spenderlo. Il software di convalida prenderà quindi lo script di sblocco contenuto nell'input che sta tentando di spendere questo UTXO ed esegue i

due script. Lo script di sblocco e blocco vengono quindi eseguiti in sequenza. L'input è valido se lo script di sblocco soddisfa le condizioni dello script di blocco (vedi Esecuzione separata di script di sblocco e blocco). Tutti gli input sono convalidati in modo indipendente, come parte della convalida complessiva della transazione.

Nota che l'UTXO è permanentemente registrato nella blockchain, e quindi è invariabile e non è influenzato dai tentativi falliti di spenderlo per riferimento in una nuova transazione. Solo una transazione valida che soddisfa correttamente le condizioni dell'output determina che l'output viene

considerato come "esaurito" e rimosso dal set di output di transazione non spesi (set UTXO).

La Figura 38 è un esempio degli script di sblocco e blocco per il tipo più comune di transazione bitcoin (un pagamento a un hash di chiave pubblica), che mostra lo script combinato risultante dalla concatenazione degli script di sblocco e blocco prima della convalida di script.

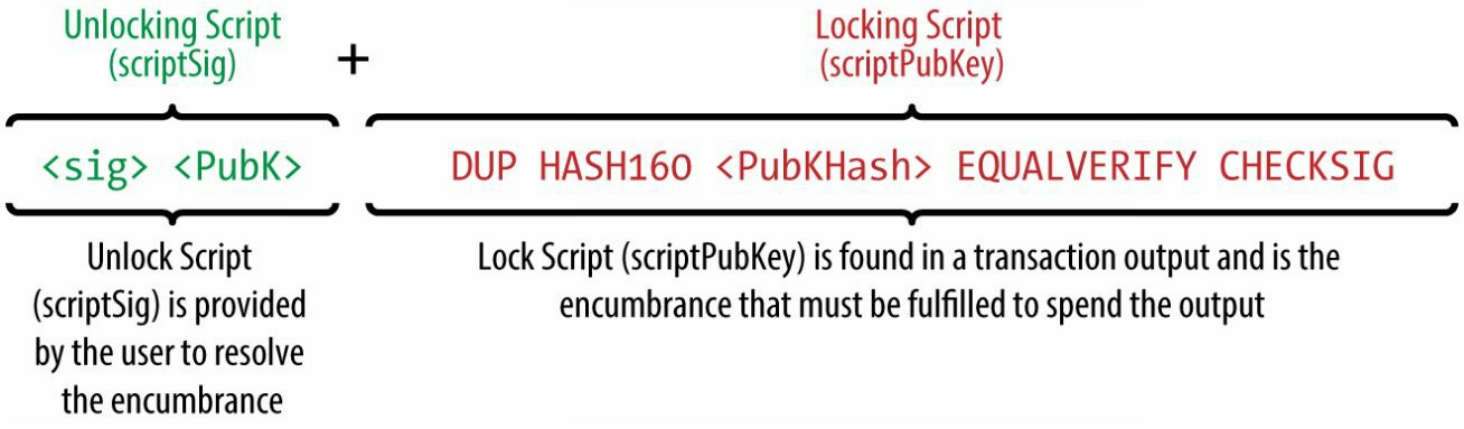


Figura 38. Combinando

scriptSig e scriptPubKey per valutare un transaction script

Lo stack di esecuzione dello script

Il linguaggio bitcoin scripting è detto un linguaggio stack-based perché utilizza una struttura dati chiamata *stack*. Lo stack (pila) è una struttura dati molto semplice, che può essere visualizzata come un mazzo di carte. Uno stack permette due operazioni: push e pop. Push aggiunge gli elementi in cima alla pila. Pop rimuove l'elemento in cima alla pila. Le operazioni su una pila possono agire solo sull'elemento più in alto nella

pila. Una struttura dati di stack viene anche chiamata coda Last-In-First-Out o "LIFO".

Il linguaggio di scripting esegue lo script elaborando ogni elemento da sinistra a destra. I numeri (costanti di dati) vengono inseriti nello stack. Gli operatori spingono o fanno saltare uno o più parametri dallo stack, agiscono su di loro e potrebbero spingere un risultato in pila. Ad esempio, OP_ADD farà sommare due elementi dallo stack, aggiungerli e inserire la somma risultante nello stack.

Gli operatori condizionali valutano una condizione, producendo un risultato booleano di VERO o FALSO. Ad esempio, OP_EQUAL fa compara

due elementi dallo stack e restituisce TRUE (TRUE è rappresentato dal numero 1) se sono uguali o FALSE (rappresentati da zero) se sono diversi. Gli script di transazioni Bitcoin di solito contengono un operatore condizionale, in modo che possano produrre il risultato TRUE che indica una transazione valida.

A simple script

Ora applichiamo ciò che abbiamo imparato su script e stack con alcuni semplici esempi.

I n Validazione di uno script bitcoin usando la matematica, lo script 2 3 OP_ADD 5 OP_EQUAL mostra l'operatore di addizione aritmetica

OP_ADD, aggiungendo due numeri e mettendo il risultato nello stack, seguito dall'operatore condizionale OP_EQUAL, che controlla che la somma risultante sia uguale a 5. Per brevità, il prefisso OP_ è omesso nell'esempio passo-passo. Per ulteriori dettagli sugli operatori e le funzioni di script disponibili, vedi Linguaggio di Scripting delle Transazioni: Operatori, Costanti e Simboli.

La maggior parte degli script di blocco fa riferimento a un indirizzo bitcoin o a una chiave pubblica, richiedendo in tal modo una prova di proprietà per spendere i fondi, quindi lo script non deve essere così complesso. Qualsiasi

combinazione di script di blocco e sblocco che restituisce un valore VERO è valida. La semplice aritmetica che abbiamo usato come esempio del linguaggio di scripting è anche uno script di blocco valido che può essere usato per bloccare un output di transazione.

Usa una parte dello script aritmetico di esempio come locking script:

```
3 OP_ADD 5 OP_EQUAL
```

che può essere soddisfatta da una transazione che contiene un input con lo script di unlocking:

```
2
```

Il software di validazione combina gli script di locking e unlocking e lo script

risultante è:

```
2 3 OP_ADD 5 OP_EQUAL
```

Come abbiamo visto nell'esempio passo-passo in Validazione di uno script bitcoin usando la matematica, quando questo script è eseguito, il risultato è OP_TRUE, rendendo la transazione valida. Non solo si tratta di uno script di blocco dell'output della transazione valido, ma l'UTXO risultante potrebbe essere speso da chiunque abbia le abilità aritmetiche per sapere che il numero 2 soddisfa lo script.

	Le transazioni sono valide se il risultato principale nello stack è

TIP	VERO (indicato come {0x01}), qualsiasi altro valore diverso da zero o se lo stack è vuoto dopo l'esecuzione dello script. Le transazioni non sono valide se il valore superiore nello stack è FALSE (un valore vuoto di lunghezza zero indicato come {}) o se l'esecuzione dello script viene interrotta esplicitamente da un operatore, ad esempio OP_VERIFY, OP_RETURN o un terminatore condizionale

	come OP_ENDIF. Vedi Linguaggio di Scripting delle Transazioni: Operatori, Costanti e Simboli per i dettagli.

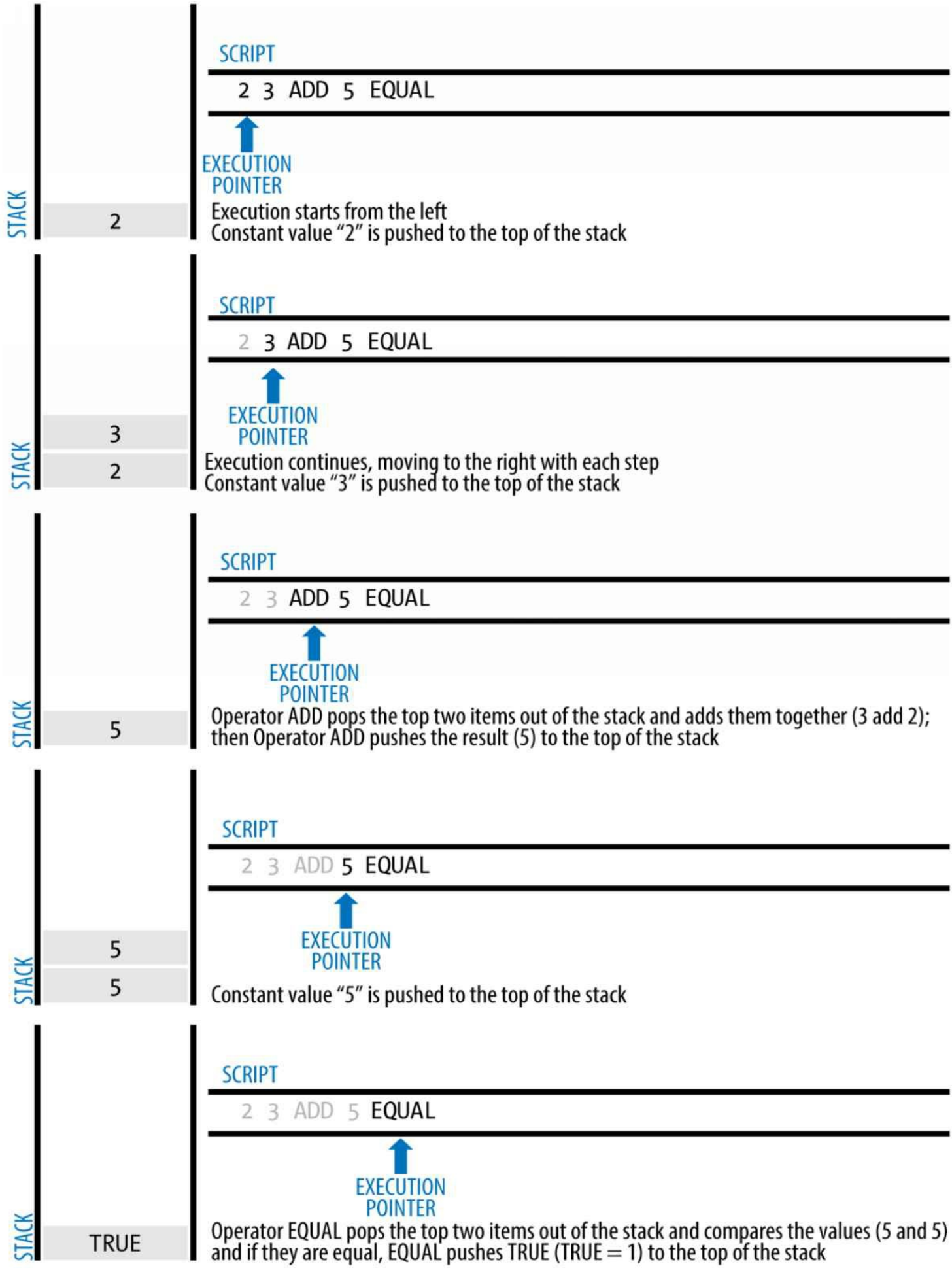

SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
2
Execution starts from the left
Constant value "2" is pushed to the top of the stack
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
3
2
Execution continues, moving to the right with each step
Constant value "3" is pushed to the top of the stack
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
5
Operator ADD pops the top two items out of the stack and adds them together (3 add 2);
then Operator ADD pushes the result (5) to the top of the stack
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
5
5
Constant value "5" is pushed to the top of the stack
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
TRUE
Operator EQUAL pops the top two items out of the stack and compares the values (5 and 5)
and if they are equal, EQUAL pushes TRUE (TRUE = 1) to the top of the stack

Figura 39. Validazione di uno script bitcoin usando la matematica

Quello che segue è uno script leggermente più complesso, che calcola 2 + 7 - 3 + 1. Si noti che quando lo script contiene diversi operatori di fila, lo stack consente ai risultati di un operatore di essere interpretati dall'operatore successivo:

```
2 7 OP_ADD 3 OP_SUB 1 OP_ADD 7 OP_EQUAL
```

Prova a convalidare lo script precedente usando carta e penna. Quando l'esecuzione dello script termina, dovresti rimanere con il valore VERO sullo stack.

Esecuzione separata di script di sblocco e blocco

Nel client bitcoin originale, gli script di sblocco e blocco venivano concatenati ed eseguiti in sequenza. Per motivi di sicurezza, questo è stato modificato nel 2010 a causa di una vulnerabilità che ha consentito a uno script di sblocco non valido di inviare dati nello stack e di danneggiare lo script di blocco. Nell'implementazione corrente, gli script vengono eseguiti separatamente con lo stack trasferito tra le due esecuzioni, come descritto di seguito.

Innanzitutto, lo script di sblocco viene eseguito utilizzando il motore di

esecuzione dello stack. Se lo script di sblocco viene eseguito senza errori (ad es. Non sono rimasti operatori "pendenti"), lo stack principale (non lo stack alternativo) viene copiato e viene eseguito lo script di blocco. Se il risultato dell'esecuzione dello script di blocco con i dati dello stack copiati dallo script di sblocco è "TRUE", lo script di sblocco è riuscito a risolvere le condizioni imposte dallo script di blocco e, pertanto, l'input è un'autorizzazione valida per utilizzare UTXO. Se un risultato diverso da "TRUE" rimane dopo l'esecuzione dello script combinato, l'input non è valido perché non è riuscito a soddisfare le condizioni di spesa

immesse nell'UTXO. Si noti che l'UTXO è permanentemente registrato nella blockchain, e quindi è invariabile e non è influenzato dai tentativi falliti di spenderlo inseriti in una nuova transazione. Solo una transazione valida che soddisfi correttamente le condizioni dell'UTXO risulterà nell'UTXO contrassegnata come "esaurita" e rimossa dal set di UTXO disponibile (non speso).

Pay-to-Public-Key-Hash (P2PKH)

La stragrande maggioranza delle transazioni elaborate sulla rete bitcoin sono transazioni P2PKH. Questi

contengono uno script di blocco che incorpora l'output con un hash di chiave pubblica, più comunemente noto come indirizzo bitcoin. Le transazioni che pagano un indirizzo bitcoin contengono script P2PKH. Un output bloccato da uno script P2PKH può essere sbloccato (speso) presentando una chiave pubblica e una firma digitale creata dalla corrispondente chiave privata (vedi Digital Signatures (ECDSA)).

Ad esempio, diamo un'occhiata al pagamento di Alice a Bob's Cafe. Alice ha effettuato un pagamento di 0,015 bitcoin all'indirizzo bitcoin del bar. L'output della transazione avrebbe uno script di blocco del

modulo:

```
OP_DUP OP_HASH160 <Cafe Public Key Hash> OP_EQUALVERIFY OP_CHECKSIG
```

Il Cafe Public Key Hash è equivalente all'indirizzo bitcoin del cafe, senza la codifica Base58Check. La maggior parte delle applicazioni mostrava l'hash della chiave pubblica nella codifica esadecimale e non il familiare indirizzo Base58Check del bitcoin formato che inizia con un "1".

Il locking script precedente può essere soddisfatto dall'unlocking script nella forma:

```
<Cafe Signature> <Cafe Public Key>
```

I due script insieme andrebbero a

formare il seguente script di validazione combinato:

```
<Cafe Signature> <Cafe Public Key>
OP_DUP OP_HASH160
<Cafe Public Key Hash>
OP_EQUALVERIFY OP_CHECKSIG
```

Quando viene eseguito, questo script combinato verrà valorizzato a TRUE se, e solo se, lo script di sblocco corrisponde alle condizioni impostate dallo script di blocco. In altre parole, il risultato sarà VERO se lo script di sblocco ha una firma valida dalla chiave privata del caffè che corrisponde all'hash della chiave pubblica impostato come encumbrance.

Le Figure 40 e 41 mostrano (in due

parti) un'esecuzione passo-passo dello script combinato, che dimostrerà che questa è una transazione valida.

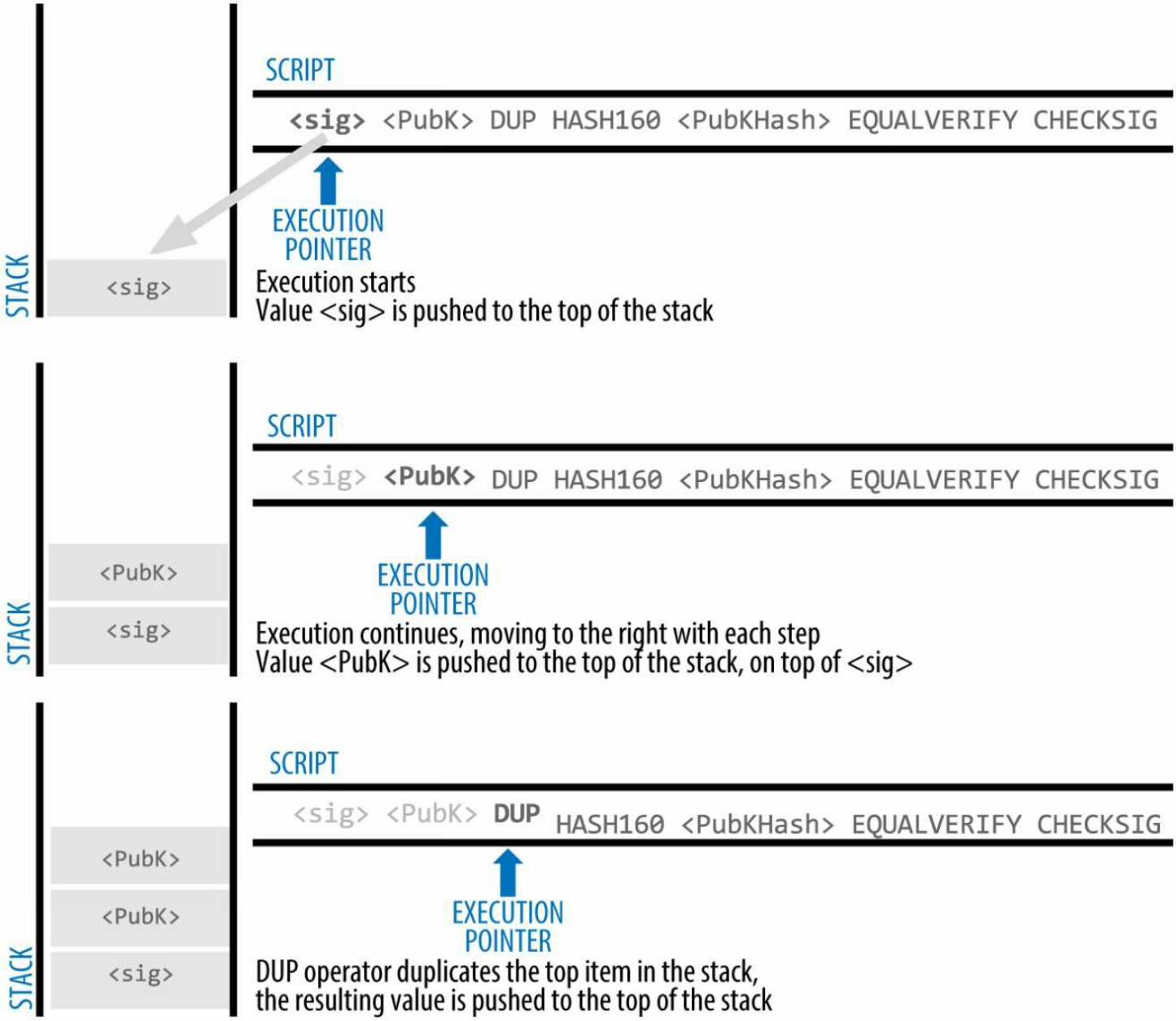


Figura 40. Valutando uno script per una transazione P2PKH (Parte 1 di 2)

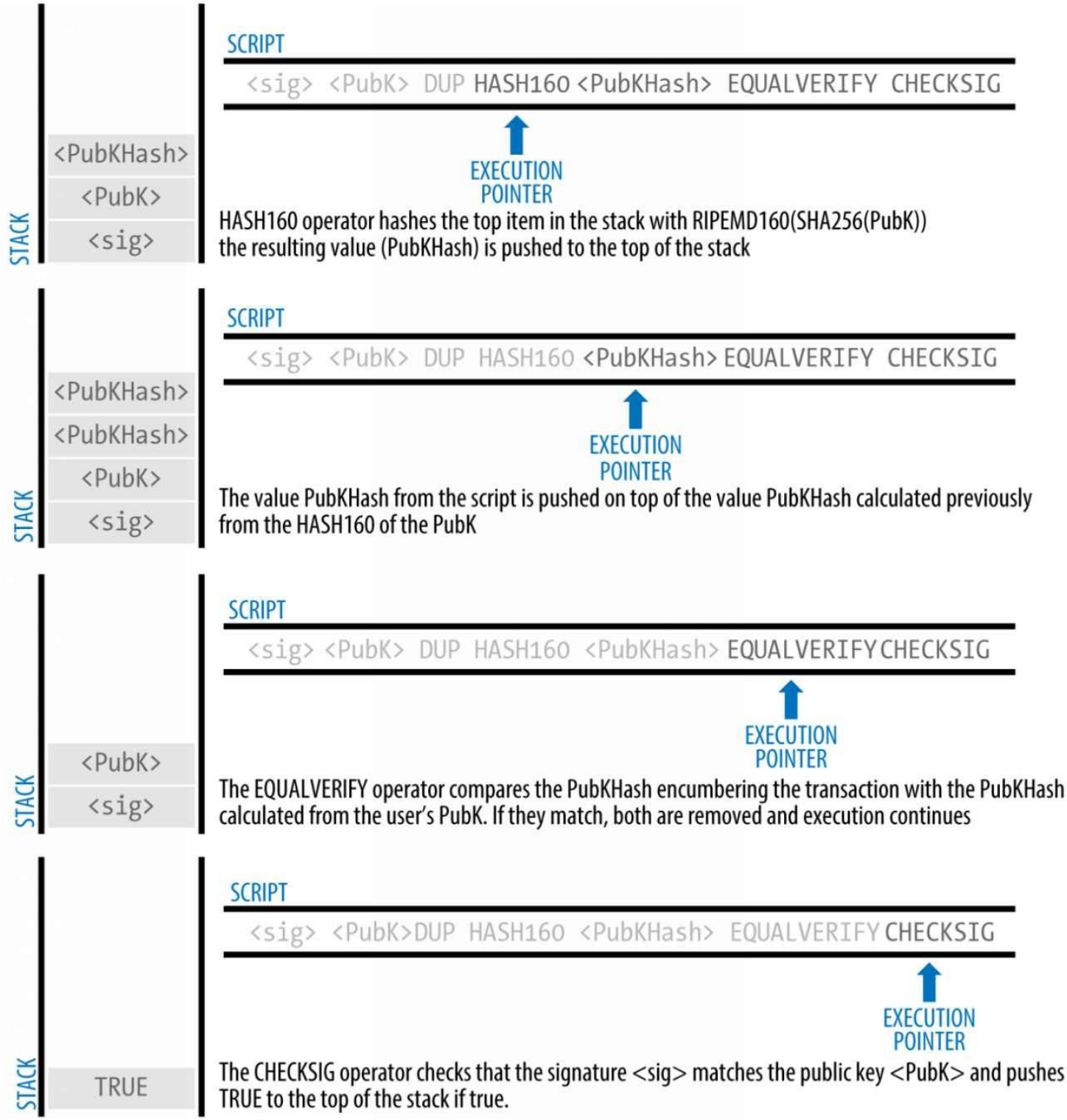


Figura 41. Eseguendo uno script per una transazione P2PKH (Parte 2 di 2)

Digital Signatures (ECDSA)

Finora, non abbiamo approfondito alcun dettaglio sulle "firme digitali". In questa sezione guardiamo a come funzionano le firme digitali e come possono presentare la prova di proprietà di una chiave privata senza rivelare quella chiave privata.

L'algoritmo di firma digitale utilizzato in bitcoin è l'*Algoritmo di firma digitale a curva ellittica*, o *ECDSA*. ECDSA è l'algoritmo utilizzato per le firme digitali basate sulla coppia di chiavi private/pubbliche della curva ellittica, come descritto in Crittografia

a Curve Ellittiche. ECDSA è utilizzato dalle funzioni di script OP_CHECKSIG, OP_CHECKSIGVERIFY, OP_CHECKMULTISIG e OP_CHECKMULTISIGVERIFY. Ogni volta che vedi quelli in uno script di blocco, lo script di sblocco deve contenere una firma ECDSA.

Una firma digitale ha tre scopi in bitcoin (vedere la barra laterale seguente). In primo luogo, la firma dimostra che il proprietario della chiave privata, che è implicitamente il proprietario dei fondi, ha *autorizzato* la spesa di tali fondi. In secondo luogo, la prova dell'autorizzazione è *innegabile* (non ripudio). In terzo

luogo, la firma dimostra che la transazione (o parti specifiche della transazione) *non sono state modificate* e *non possono essere modificate* da nessuno dopo che è stato firmato.

Si noti che ogni input di transazione è firmato indipendentemente. Questo è fondamentale, poiché né le firme né gli input devono appartenere o essere applicati dagli stessi "proprietari". Infatti, uno schema di transazione specifico chiamato "CoinJoin" utilizza questo fatto per creare transazioni multipartitiche per la privacy.

	Ogni input di transazione e qualsias firma che può

NOTA	contenere è *completamente* indipendente da qualsiasi altro input c firma. Più parti possono collaborare per costruire transazioni e firmare un solo input ciascuna

Definizione di "Firma Digitale" di Wikipedia

La firma digitale è un metodo matematico teso a dimostrare

l'autenticità di un messaggio o di un documento digitale inviato tra mittente e destinatario attraverso un canale di comunicazione non sicuro, garantendo al destinatario che:

- il mittente del messaggio sia chi dice di essere (autenticazione),
- il mittente non possa negare di averlo inviato (non ripudio)
- il messaggio non sia stato alterato lungo il percorso dal mittente al destinatario (integrità).

Le firme digitali si basano su schemi o protocolli crittografici comunemente usati nella distribuzioni di software, nelle transazioni finanziarie e in altri casi

in cui si debba rilevare la falsificazione o l'alterazione del messaggio.

Sorgente:
https://it.wikipedia.org/wiki/Firma_d

Come funzionano le firme digitali

Una firma digitale è uno *schema matematico* costituito da due parti. La prima parte è un algoritmo per la creazione di una firma, utilizzando una chiave privata (la chiave di firma), da un messaggio (la transazione). La seconda parte è un algoritmo che consente a chiunque di verificare la firma, dato anche il messaggio e una chiave pubblica.

Creazione di una firma digitale

In implementazione di bitcoin dell'algoritmo ECDSA, il "messaggio" firmato è la transazione, o più precisamente un hash di un sottoinsieme specifico dei dati nella transazione (vedi Signature Hash Types (SIGHASH)). La chiave di firma è la chiave privata dell'utente. Il risultato è la firma:

Sig = F_{sig}(F_{hash}(m), dA)\)\)

dove:

- *dA* è la chiave privata della firma
- *m* è la transazione (o parti di essa)

- *Fhash* è la funzione di hashing
- *Fsig* è l'algoritmo di firma
- *Sig* è la firma risultante

Maggiori dettagli sulla matematica dell'ECDSA possono essere trovati in Matematica dietro ECDSA.

La funzione *Fsig* produce una firma Sig che è composta da due valori, comunemente denominati R e S:

$$Sig = (R, S)$$

Ora che i due valori R e S sono stati calcolati, vengono serializzati in un flusso di byte utilizzando uno schema di codifica standard internazionale denominato *Distinguished Encoding Rules* o DER.

Serializzazione delle firme (DER)

Diamo nuovamente un'occhiata alla transazione che Alice ha creato. Nell'input della transazione c'è uno script di sblocco che contiene la seguente firma codificata DER dal portafoglio di Alice:

```
3045022100884d142d86652a3f47ba4746ec
```

Quella firma è un flusso di byte serializzato dei valori R e S prodotto dal portafoglio di Alice per dimostrare che possiede la chiave privata autorizzata a spendere quell'output. Il formato di serializzazione è composto da nove elementi come segue:

- 0x30—indicando l'inizio di una

sequenza DER

- 0x45—la lunghezza della sequenza (69 byte)
- 0x02—segue un valore intero
- 0x21—la lunghezza del numero intero (33 byte)
- R—
  00884d142d86652a3f47ba4746ec719
- 0x02—segue un altro intero
- 0x20—la lunghezza del numero intero (32 byte)
- S—
  4b9f039ff08df09cbe9f6addac960298c
- Un suffisso (0x01) che indica il tipo di hash utilizzato (SIGHASH_ALL)

Verifica se puoi decodificare la firma serializzata di Alice (codificata DER)

utilizzando questo elenco. I numeri importanti sono R e S; il resto dei dati fa parte dello schema di codifica DER.

Verifica della Firma

Per verificare la firma, è necessario avere la firma (R e S), la transazione serializzata e la chiave pubblica (che corrisponde alla chiave privata utilizzata per creare la firma). In sostanza, la verifica di una firma significa "Solo il proprietario della chiave privata che ha generato questa chiave pubblica potrebbe aver prodotto questa firma su questa transazione".

L'algoritmo di verifica della firma accetta il messaggio (un hash della

transazione o parti di esso), la chiave pubblica del firmatario e la firma (valori R e S) e restituisce VERO se la firma è valida per questo messaggio e la chiave pubblica.

Signature Hash Types (SIGHASH)

Le firme digitali vengono applicate ai messaggi, che nel caso del bitcoin sono le transazioni stesse. La firma implica un *impegno* da parte del firmatario a specifici dati di transazione. Nella forma più semplice, la firma si applica all'intera transazione, impegnando in tal modo tutti gli input, gli output e altri campi di transazione. Tuttavia, una firma può

impegnarsi solo su un sottoinsieme dei dati in una transazione, che è utile per un numero di scenari come vedremo in questa sezione.

Le firme bitcoin hanno un modo per indicare quale parte dei dati di una transazione è inclusa nell'hash firmato dalla chiave privata usando un flag SIGHASH. Il flag SIGHASH è un singolo byte che viene aggiunto alla firma. Ogni firma ha un flag SIGHASH e la flag può essere diversa da input a input. Una transazione con tre input firmati può avere tre firme con diversi flag SIGHASH, ciascuna firma firma (commit) diverse parti della transazione.

Ricorda, ogni input può contenere una

firma nel suo script di sblocco. Di conseguenza, una transazione che contiene diversi input può avere firme con diversi flag SIGHASH che impegnano parti diverse della transazione in ciascuno degli input. Si noti inoltre che le transazioni bitcoin possono contenere input da diversi "proprietari", che possono firmare solo un input in una transazione parzialmente costruita (e non valida), collaborando con altri utenti per raccogliere tutte le firme necessarie per effettuare una transazione valida. Molti dei tipi di flag SIGHASH hanno senso solo se si pensa a più partecipanti che collaborano al di fuori della rete bitcoin e aggiornano

una transazione parzialmente firmata.

Ci sono tre flag SIGHASH: ALL, NONE, e SINGLE, come mostrato in Tipi SIGHASH e loro significato.

Tabella 19. Tipi SIGHASH e lor significato

SIGHASH flag	Valore	Descrizione
ALL	0x01	La firma s applica tutti gli inpu e output
NONE	0x02	La firma s applica tutti g

		input, nessuno degli output
SINGLE	0x03	La firma s applica tutti g input, m solo l'outpu con lo stess numero d indice dell'input firmato

Inoltre, esiste un flag di modifica SIGHASH_ANYONECANPAY, che può essere combinato con ciascuno dei

flag precedenti. Quando ANYONECANPAY è impostato, viene firmato un solo input, lasciando il resto (ei relativi numeri di sequenza) aperti per la modifica. ANYONECANPAY ha il valore 0x80 ed è applicato da OR bit a bit, risultando nei flag combinati come mostrato in Tabella 20

Tabella 20. SIGHASH types with their meanings

SIGHASH flag	Valu
ALL|ANYONECANPAY	0x81

NONE|ANYONECANPAY	0x82
SINGLE|ANYONECANPAY	0x83

Il modo in cui i flag SIGHASH

vengono applicati durante la firma e la verifica è che viene eseguita una copia della transazione e determinati campi all'interno vengono troncati (impostati su lunghezza zero e svuotati). La transazione risultante è serializzata. Il flag SIGHASH viene aggiunto alla fine della transazione serializzata e il risultato viene sottoposto a hash. L'hash stesso è il "messaggio" che è firmato. A seconda del flag SIGHASH utilizzato, le diverse parti della transazione vengono troncate. L'hash risultante dipende da diversi sottoinsiemi dei dati nella transazione. Includendo il SIGHASH come ultimo passaggio prima dell'hashing, la firma commuta anche il tipo SIGHASH,

quindi non può essere modificato (ad es. Da un minatore).

NOTA	Tutti i tipi SIGHASH firmano il campo nLocktime della transazione (vedi Transaction Locktime (nLocktime)). Inoltre il tipo SIGHASH viene aggiunto alla transazione prima che sia firmato, in modo che non possa essere modificato una volta firmato.

Nell'esempio della transazione di

Alice (vedi la lista in Serializzazione di firme (DER)), abbiamo visto che l'ultima parte della firma codificata DER era 01, che è il flag SIGHASH_ALL. Ciò blocca i dati della transazione, quindi la firma di Alice sta commettendo lo stato di tutti gli input e output. Questo è il modulo di firma più comune.

Diamo un'occhiata ad alcuni degli altri tipi SIGHASH e come possono essere utilizzati nella pratica:

ALL|ANYONECANPAY

Questa costruzione può essere utilizzata per realizzare una transazione in stile "crowdfunding": qualcuno che cerca di raccogliere

fondi può costruire una transazione con un singolo output, la singola uscita paga l'importo "obiettivo" per la raccolta di fondi, che ovviamente non è valida, in quanto non ha input, tuttavia altri possono ora modificarlo aggiungendo un proprio input come donazione e firmano il proprio input con ALL | ANYONECANPAY. A meno che non vengano raccolti abbastanza input per raggiungere il valore dell'output, la transazione non è valida. Ogni donazione è un "impegno", che non può essere raccolto dalla raccolta fondi fino a quando non viene sollevato l'intero importo dell'obiettivo.

NONE

Questa costruzione può essere utilizzata per creare un "assegno al portatore" o un "assegno in bianco" di un importo specifico. Si impegna sull'input, ma consente di modificare lo script di blocco dell'output. Chiunque può scrivere il proprio indirizzo bitcoin nello script di blocco dell'output e riscattare la transazione. Tuttavia, il valore di output è bloccato dalla firma.

NONE|ANYONECANPAY

Questa costruzione può essere utilizzata per costruire un "raccoglitore di polvere". Gli utenti che hanno un piccolissimo UTXO nei loro portafogli non possono spenderli senza il costo di commissioni che

superano il valore della polvere stessa. Con questo tipo di firma, la polvere UTXO può essere donata a chiunque per essere aggregata e spesa quando si vuole.

Ci sono alcune proposte per modificare o espandere il sistema SIGHASH. Una di queste proposte è *Bitmask Sighash Modes* di Glenn Willen di Blockstream, come parte del progetto Elements. Ciò mira a creare un sostituto flessibile per i tipi SIGHASH che consenta "maschere di bit arbitrarie e miner riscrivibili di input e output" in grado di esprimere "schemi di precommitment contrattuali più complessi, come le offerte firmate con cambio di uno scambio di attività

distribuito".

NOTA	Non vedrai i flag SIGHASH presentati come un'opzione nell'applicazione wallet di un utente. Con poche eccezioni, i wallet costruiscono script P2PKH e firmano con flag SIGHASH_ALL. Per utilizzare un flag SIGHASH diverso, è necessario scrivere software per costruire e firmare transazioni. Ancora più importante, i flag SIGHASH

	possono essere utilizzati da applicazioni bitcoin speciali che consentono nuovi usi.

Matematica dietro ECDSA

Come accennato in precedenza, le firme vengono create da una funzione matematica Fsig che produce una firma composta da due valori R e S. In questa sezione guardiamo la funzione *Fsig* in dettaglio.

L'algoritmo della firma genera prima una coppia di chiavi pubblica privata *temporanea*. Questa coppia di chiavi

temporanee viene utilizzata nel calcolo dei valori R e S, dopo una trasformazione che coinvolge la chiave privata di firma e l'hash della transazione.

La coppia di chiavi temporanee si basa su un numero casuale k, che viene utilizzato come chiave privata *effimera* (temporanea). Da k, generiamo la corrispondente chiave pubblica temporanea P (calcolata come $P = k * G$, nello stesso modo in cui le chiavi pubbliche bitcoin sono derivate, vedi Chiavi Pubbliche). Il valore R della firma digitale è quindi la coordinata x della chiave pubblica effimera P.

Da lì, l'algoritmo calcola il valore S

della firma, in modo tale che:

$S = k\text{-}1\ (Hash(m) + dA * R)\ mod\ p$

dove:

- k è la chiave privata effimera
- R è la coordinata x della chiave pubblica effimera
- dA è la chiave privata della firma
- m sono i dati della transazione
- p è l'ordine primo della curva ellittica

La verifica è l'inverso della funzione di generazione della firma, utilizzando i valori R, S e la chiave pubblica per calcolare un valore P, che è un punto

sulla curva ellittica (la chiave pubblica temporanea utilizzata nella creazione della firma):

$P = S\text{-}1 * Hash(m) * G + S\text{-}1 * R * Qa$

dove:

- R e S sono i valori della firma
- Qa è la chiave pubblica di Alice
- m sono i dati della transazione che sono stati firmati
- G è il punto del generatore di curve ellittiche

Se la coordinata x del punto P calcolato è uguale a R, allora il verificatore può concludere che la firma è valida.

Si noti che nel verificare la firma, la chiave privata non è né conosciuta né rivelata.

TIP	ECDSA è necessariamente una matematica piuttosto complicata; una spiegazione completa va oltre lo scopo di questo libro. Una serie di ottime guide online ti accompagnano passo dopo passo: cerca "spiegazione ECDSA" o prova questo: http://bit.ly/2r0HhGB.

L'importanza della Casualità nelle Firme

Come abbiamo visto in Matematica dietro ECDSA, l'algoritmo di generazione della firma utilizza una chiave casuale k, come base per una coppia di chiavi privata/pubblica effimera. Il valore di k non è importante, *purché sia casuale*. Se lo stesso valore k viene utilizzato per produrre due firme su diversi messaggi (transazioni), la *chiave privata di firma* può essere calcolata da chiunque. Riutilizzare lo stesso valore per k in un algoritmo di firma porta all'esposizione della chiave privata!

ATTENZIONE	Se lo stesso valore k è utilizzato nell'algoritmo di firma su due diverse transazioni, la chiave privata può essere calcolata ed esposta al mondo!

Questa non è solo una possibilità teorica. Abbiamo visto questo problema portare all'esposizione di chiavi private in alcune diverse

implementazioni degli algoritmi di firma delle transazioni in bitcoin. Le persone hanno rubato dei fondi a causa del riutilizzo involontario di un valore *k*. Il motivo più comune per il riutilizzo di un valore *k* è un generatore di numeri casuali inizializzato in modo errato.

Per evitare questa vulnerabilità, la migliore pratica del settore è quella di non generare *k* con un generatore di numeri casuali con entropia, ma invece di utilizzare un processo deterministico-casuale seminato con i dati della transazione stessa. Ciò garantisce che ogni transazione produca un *k* differente. L'algoritmo standard del settore per

l'inizializzazione deterministica di k è definito in RFC 6979, pubblicato da Internet Engineering Task Force.

Se si sta implementando un algoritmo per la firma delle transazioni in bitcoin, *è necessario* utilizzare RFC 6979 o un algoritmo analogicamente deterministico-casuale per assicurarsi di generare un k differente per ogni transazione.

Indirizzi bitcoin, saldi ed altre astrazioni

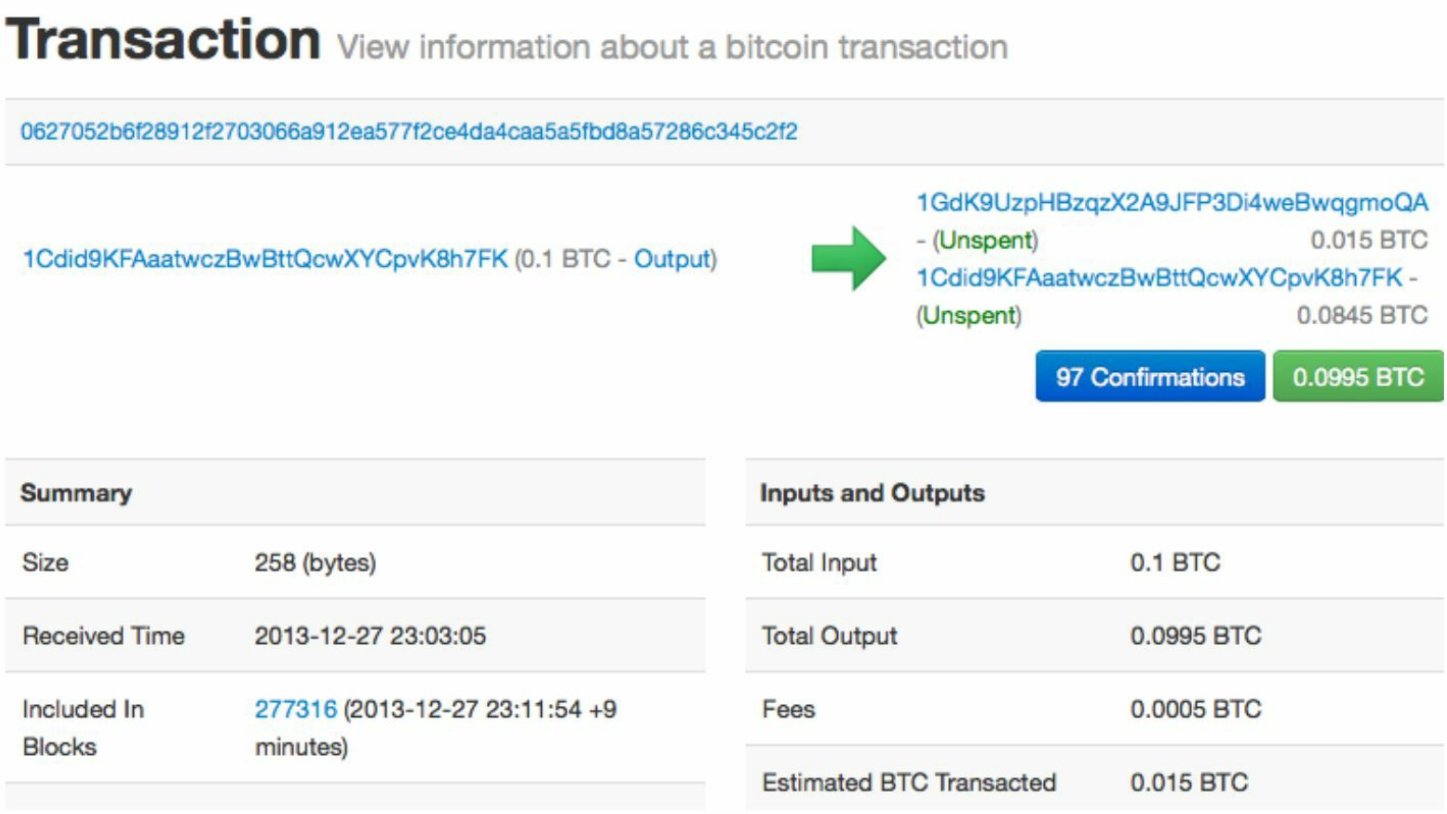
Abbiamo iniziato questo capitolo scoprendo che le transazioni sembrano molto diverse "dietro le quinte" rispetto a come vengono presentate nei

portafogli, negli explorer blockchain ed in altre applicazioni rivolte agli utenti. Molti dei concetti semplicistici e familiari dei capitoli precedenti, come gli indirizzi bitcoin ed i saldi, sembrano essere assenti dalla struttura delle transazioni. Abbiamo visto che le transazioni non contengono indirizzi bitcoin, di per sé, ma operano attraverso script che bloccano e sbloccano valori discreti di bitcoin. I saldi non sono presenti in questo sistema e tuttavia ogni applicazione di wallet mostra in modo visibile il saldo del portafoglio dell'utente.

Ora che abbiamo esplorato ciò che è effettivamente incluso in una transazione bitcoin, possiamo

esaminare come le astrazioni di livello superiore derivino dai componenti apparentemente primitivi della transazione.

Vediamo di nuovo in che modo la transazione di Alice è stata presentata su un famoso block explorer (La transazione di Alice al Bar di Bob).



Transaction View information about a bitcoin transaction

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK (0.1 BTC - Output)

1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA - (Unspent) 0.015 BTC

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK - (Unspent) 0.0845 BTC

97 Confirmations | 0.0995 BTC

Summary	
Size	258 (bytes)
Received Time	2013-12-27 23:03:05
Included In Blocks	277316 (2013-12-27 23:11:54 +9 minutes)

Inputs and Outputs	
Total Input	0.1 BTC
Total Output	0.0995 BTC
Fees	0.0005 BTC
Estimated BTC Transacted	0.015 BTC

Figura 42. La transazione di

Alice al Bar di Bob

Sul lato sinistro della transazione, l'explorer blockchain mostra l'indirizzo bitcoin di Alice come "mittente". In realtà, questa informazione non è nella transazione stessa. Quando l'explorer di blockchain ha recuperato la transazione, ha anche recuperato la transazione precedente a cui faceva riferimento nell'input ed ha estratto il primo output da quella stessa precedente transazione. All'interno di quell'output c'è uno script di blocco che blocca l'UTXO nell'hash della chiave pubblica di Alice (uno script P2PKH). Il blockchain explorer ha estratto l'hash della chiave pubblica e

lo ha codificato utilizzando la codifica Base58Check per produrre e visualizzare l'indirizzo bitcoin che rappresenta quella chiave pubblica.

Allo stesso modo, sul lato destro, l’explorer blockchain mostra i due output; il primo all'indirizzo bitcoin di Bob e il secondo all'indirizzo bitcoin di Alice (come resto). Ancora una volta, per creare questi indirizzi bitcoin, il blockchain explorer ha estratto lo script di blocco da ciascun output, riconoscendolo come uno script P2PKH ed estraendone l'hash della chiave pubblica dall'interno. Infine, l'explorer blockchain ha ricodificato l'hash della chiave pubblica con Base58Check per

produrre e visualizzare gli indirizzi bitcoin.

Se dovessi fare clic sull'indirizzo bitcoin di Bob, l'explorer blockchain ti mostrerebbe la vista come in Figura 43.

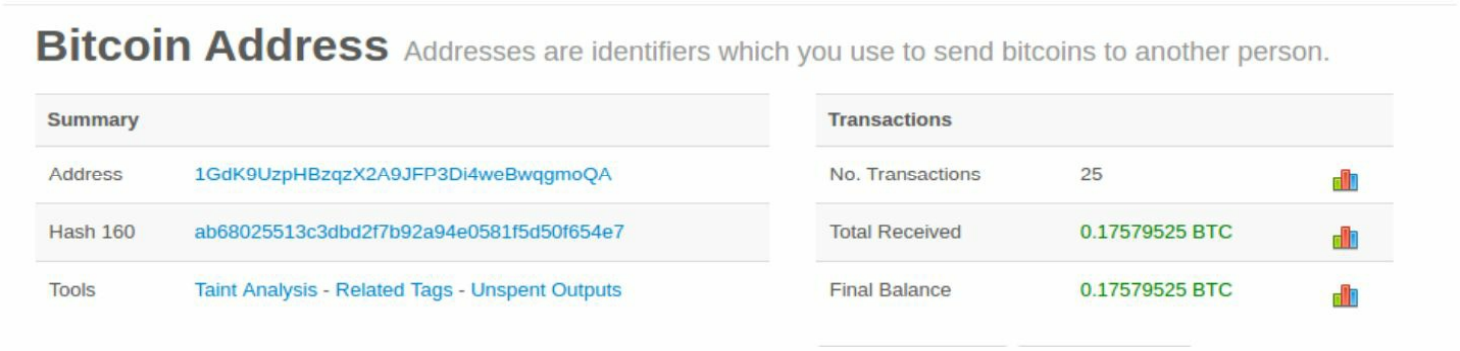


Figura 43. The balance of Bob's bitcoin address

Il blockchain explorer mostra il saldo dell'indirizzo bitcoin di Bob. Ma da nessuna parte nel sistema bitcoin esiste un concetto di "equilibrio". Piuttosto, i valori visualizzati qui sono costruiti

dal blockchain explorer come segue.

Per costruire l'importo "Total Received", il blockchain explorer decodificherà prima la codifica Base58Check dell'indirizzo bitcoin per recuperare l'hash a 160 bit della chiave pubblica di Bob che è codificata all'interno dell'indirizzo. Quindi, l'explorer blockchain cercherà nel database delle transazioni, cercando output con script di blocco P2PKH che contengono l'hash della chiave pubblica di Bob. Sommando il valore di tutti gli output, il blockchain explorer può produrre il valore totale ricevuto.

Costruire il saldo corrente (visualizzato come "Final Balance")

richiede più lavoro. Il blockchain explorer mantiene un database separato degli output attualmente non utilizzati, il set UTXO. Per mantenere questo database, il blockchain explorer deve monitorare la rete bitcoin, aggiungere UTXO appena creato e rimuovere UTXO utilizzato, in tempo reale, come appaiono nelle transazioni non confermate. Questo è un processo complicato che dipende dal tenere traccia delle transazioni mentre si propagano, oltre a mantenere il consenso con la rete bitcoin per garantire che venga seguita la catena corretta. A volte, l'explorer blockchain va fuori sincrono e la sua prospettiva del set UTXO è incompleta o errata.

Dal set UTXO, il blockchain explorer riassume il valore di tutti gli output non spesi che fanno riferimento all'hash della chiave pubblica di Bob e produce il numero "Final Balance" mostrato all'utente.

Per produrre questa immagine con questi due "saldi", l'explorer di blockchain deve indicizzare e cercare tra dozzine, centinaia o persino centinaia di migliaia di transazioni.

In sintesi, le informazioni presentate agli utenti attraverso applicazioni wallet, blockchain explorer e altre interfacce utente di bitcoin sono spesso composte da astrazioni di livello superiore derivate dalla ricerca di molte transazioni diverse,

dall'ispezione del loro contenuto e dalla manipolazione dei dati contenuti al loro interno. Presentando questa visione semplicistica delle transazioni bitcoin che assomigliano a assegni bancari da un mittente a un destinatario, queste applicazioni devono astrarre molti dettagli sottostanti. Si concentrano principalmente sui tipi comuni di transazioni: P2PKH con firme SIGHASH_ALL su ogni input. Pertanto, mentre le applicazioni bitcoin possono presentare più dell'80% di tutte le transazioni in un modo facile da leggere, a volte vengono bloccate da transazioni che si discostano dalla norma. Le transazioni

che contengono script di blocco più complessi, o diversi flag SIGHASH, o molti input e output, dimostrano la semplicità e la debolezza di queste astrazioni.

Ogni giorno, centinaia di transazioni che non contengono output P2PKH sono confermate sulla blockchain. Gli explorer di blockchain spesso presentano questi messaggi di avvertimento rossi dicendo che non possono decodificare un indirizzo. Il seguente link contiene le "strange transactions" più recenti che non sono state completamente decodificate: https://blockchain.info/strange-transactions.

Come vedremo nel prossimo capitolo,

queste non sono necessariamente transazioni “strane”. Sono transazioni che contengono script di blocco più complessi rispetto al P2PKH comune. Impareremo come decodificare e comprendere script più complessi e le applicazioni che supportano successivamente.

Metodi di transazione e scripting avanzati

Introduzione

Nel capitolo precedente abbiamo introdotto gli elementi base delle transazioni di bitcoin ed abbiamo visto il più comune script di transazione, lo script P2PKH. In questo capitolo vedremo metodi avanzati di scripting e come sfruttarli per creare transazioni in condizioni complesse.

Per prima cosa, vedremo cos'è lo script *firma multipla*. Dopodiché,

esamineremo il secondo più comune script di transazione, *Pay-to-Script-Hash*, il quale apre un interno mondo di script complessi. Poi; esamineremo nuovi script operativi che aggiungono una dimensione temporale al bitcoin, grazie al *timelocks*. Per finire, vedremo un cambio all'architettura della struttura delle transazioni, il *Segregated Witness*.

Multisignature

Gli script multi-signature (n.d.t. multi firma) impostano una condizione in cui N sono le chiavi pubbliche registrate nello script e almeno M di quelle deve fornire le firme per liberare encumbrance. Questo è anche noto

come schema M-of-N, dove N è il numero totale di chiavi e M è la soglia delle firme richieste per la convalida. Ad esempio, una firma multipla 2-di-3 è quella in cui tre chiavi pubbliche sono elencate come potenziali firmatari e almeno due di esse devono essere utilizzate per creare firme per una transazione valida per spendere i fondi.

In questo momento, gli script multi-firma *standard* sono limitati a un massimo di 15 chiavi pubbliche elencate, il che significa che puoi fare qualsiasi cosa, da 1-di-1 a 15-di-15 multi firma o qualsiasi combinazione all'interno di tale intervallo. La limitazione a 3 chiavi elencate

potrebbe essere revocata al momento della pubblicazione di questo libro, quindi controllare la isStandard() function per vedere cosa è attualmente accettato dalla rete. Si noti che il limite di 3 chiavi si applica solo agli script multisignature standard (noti anche come "bare"), non agli script multisignature avvolti in uno script Pay-to-Script-Hash (P2SH). Gli script multisignature P2SH sono limitati a 15 chiavi, consentendo la multisignature fino a 15-di-15. Impareremo a conoscere P2SH in Pay-to-Script-Hash (P2SH).

La forma generale di uno script di locking che imposta una condizione multi'signature M-di-N è:

```
M <Public Key 1> <Public Key 2> ... <Public Key N> N CHECKMULTISIG
```

dove N è il numero totale di chiavi pubbliche elencate e M è la soglia del numero di firme richiesto per spendere un output.

Uno script di locking che imposta una condizione multi-signnature-2-su-3 assomiglia a questo:

```
2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG
```

Il locking script precedente può essere soddisfatto con un unlocking script contenente coppie di firme e chiavi pubbliche:

```
<Signature B> <Signature C>
```

o una qualsiasi combinazione di due

firme dalle chiavi private corrispondenti alle tre chiavi pubbliche elencate.

I due script insieme formeranno uno script di validazione combinato:

```
<Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG
```

Quando eseguito, questo script combinato ritornerà TRUE se, e solo se, lo script di unlock pareggia le condizioni impostate dallo script di lock. In questo caso, la condizione è dove lo script di unlock ha una firma valida dalle due chiavi private che corrispondono a due delle tre chiavi pubbliche impostate come impaccio (encumberance).

Un bug nell'esecuzione di CHECKMULTISIG

C'è un bug nell'esecuzione di CHECKMULTISIG che richiede una piccolo workaround. Quando CHECKMULTISIG viene eseguito, dovrebbe consumare M + N + 2 elementi in pila come parametri. Tuttavia, a causa del bug, CHECKMULTISIG mostrerà un valore aggiuntivo o un valore maggiore del previsto.

Diamo un'occhiata in dettaglio usando il precedente esempio di validazione:

```
<Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG
```

Innanzitutto, CHECKMULTISIG apre l'elemento in alto, che è N (in questo esempio "3"). Quindi visualizza N elementi, che sono le chiavi pubbliche che possono firmare. In questo esempio, le chiavi pubbliche A, B e C. Quindi, viene visualizzato un elemento, che è M, il quorum (quante firme sono necessarie). In questo caso M = 2. A questo punto, CHECKMULTISIG dovrebbe visualizzare gli elementi M finali, ovvero le firme, e vedere se sono valide. Tuttavia, sfortunatamente, un bug nell'implementazione fa sì che CHECKMULTISIG mostri un altro oggetto (M + 1 totale) in più di quanto dovrebbe. L'elemento extra viene

ignorato quando si controllano le firme, quindi non ha alcun effetto diretto su CHECKMULTISIG stesso. Tuttavia, deve essere presente un valore aggiuntivo perché se non è presente, quando CHECKMULTISIG tenta di eseguire il pop su uno stack vuoto, causerà un errore di stack e un errore di script (contrassegnando la transazione come non valida). Poiché l'elemento extra viene ignorato, può essere qualsiasi cosa, ma viene usato abitualmente 0.

Poiché questo bug è diventato parte delle regole di consenso, deve essere replicato per sempre. Pertanto la corretta convalida dello script sarebbe simile a questa:

```
0 <Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG
```

Quindi lo script di sblocco effettivamente usato in multisig non è:

```
<Signature B> <Signature C>
```

ma invece è:

```
0 <Signature B> <Signature C>
```

D'ora in poi, se vedi uno script di sblocco multisig, dovresti aspettarti di vedere uno 0 in più all'inizio, il cui unico scopo è come soluzione a un bug che è diventato accidentalmente una regola di consenso.

Pay-to-Script-Hash (P2SH)

Pay-to-script-hash (P2SH) è stato introdotto nel 2012 come un nuovo potente tipo di transazione che semplifica enormemente l'utilizzo di script di transazioni complessi. Per spiegare la necessità di P2SH, diamo un'occhiata a un esempio pratico.

In Introduzione abbiamo introdotto Mohammed, un importatore di elettronica con base a Dubai. La società di Mohammed utilizza ampiamente la funzionalità multi firma di bitcoin per i suoi account aziendali. Gli script multi firma sono uno degli usi più comuni delle funzionalità di scripting avanzate di bitcoin e sono una funzionalità molto potente. La società di Mohammed utilizza uno

script multi firma per tutti i pagamenti dei clienti, noti in termini contabili come "crediti", o AR. Con lo schema multi firma, tutti i pagamenti effettuati dai clienti sono bloccati in modo tale da richiedere almeno due firme da rilasciare, da Mohammed e uno dei suoi partner o dal suo legale che ha una chiave di backup. Uno schema multi firma come quello offre controlli di governance aziendale e protegge da furti, appropriazione indebita o perdita.

Lo script risultante è abbastanza lungo ed è simile a questo:

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key> <Partner3 Public Key> <Attorney Public
```

Key> 5 CHECKMULTISIG

Anche se gli script multi-signature sono una funzionalità potente, sono un po difficili da usare. Dato lo script precedente, Mohammed dovrebbe comunicare lo script a ogni cliente prima di iniziare un pagamento. Ogni cliente avrebbe dovuto usare un wallet bitcoin specifico con l'abilità di creare script di transazione custom e ogni cliente avrebbe dovuto capire come creare una transazione con script custom (personalizzati). Inoltre, la transazione risultante sarà cinque volte più grande di una transazione con un semplice pagamento, perchè questo script contiene chiavi pubbliche molto lunghe. Il peso di queste transazioni

extra-large andrà a gravare sull'utente sotto forma di fee. Finalmente uno script così grande dovrà essere trasferito nel set delle UTXO in RAM in ogni full node, fino a che non sarà speso. Tutte queste problematiche fanno sì che l'uso di output script complessi sarà un po meno pratico.

Pay-to-script-hash (P2SH) è stato sviluppato per risolvere queste difficoltà pratiche e per rendere l'uso di script complessi facile come un pagamento a un indirizzo bitcoin. Con i pagamenti P2SH, lo script di blocco complesso viene sostituito con la sua digital fingerprint, un hash crittografico. Quando una transazione che tenta di passare l'UTXO viene

presentata in un secondo momento, deve contenere lo script che corrisponde all'hash, oltre allo script di sblocco. In termini semplici, P2SH significa "paga a uno script che corrisponde a questo hash, uno script che verrà presentato più tardi quando questo output sarà esaurito."

Nelle transazioni P2SH, lo script di blocco che viene sostituito da un hash viene chiamato *redeem script* perché viene presentato al sistema in fase di redemption anziché come script di blocco. Script complessi senza P2SH mostra lo script senza P2SH e Complex script come P2SH mostra lo stesso script codificato con P2SH.

Tabella 21. Script complessi senz

P2SH

Locking Script	2 PubKey1 PubKey PubKey3 PubKey PubKey5 CHECKMULTISIG
Unlocking Script	Sig1 Sig2

Tabella 22. Complex script com P2SH

Redeem Script	2 PubKey1 PubKey PubKey3 PubKey PubKey5 CHECKMULTISIG
Locking Script	HASH160 <20-byt hash of redeer

	script> EQUAL
Unlocking Script	Sig1 Sig2 <redeem script>

Come si può vedere dalle tabelle, con P2SH lo script complesso che descrive le condizioni per la spesa dell'output (redeem script) non viene presentato nello script di blocco. Invece, solo un hash di esso è nello script di blocco e lo script di riscatto stesso viene presentato in seguito, come parte dello script di sblocco quando l'output viene speso. Ciò sposta l'onere delle commissioni e della complessità dal mittente al destinatario (spender) della

transazione.

Osserviamo la ditta di Mohammed, lo script multi-signature complesso, e i risultanti script P2SH.

Per primo, lo script multi-signature che la ditta di Mohammed usa per tutti i pagamenti in entrata dai clienti:

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key> <Partner3 Public Key> <Attorney Public Key> 5 CHECKMULTISIG
```

Se i placeholder sono rimpiazzati con chiavi pubbliche reali (mostrati qui come numeri di 520-bit che iniziano con 04) noterai che lo script diverrà molto lungo:

```
2
04C16B8698A9ABF84250A7C3EA7EEDEI
```

```
5 CHECKMULTISIG
```

Lo script per intero può invece essere rappresentato da un hash crittografico di 20-byte, applicando in principio l'algoritmo di hashing SHA256 e infine l'algoritmo RIPEMD160 sul risultato. L'hash di 20-byte dello script precedente è:

Usiamo libbitcoin-explorer (bx) sulla riga di comando per produrre l'hash dello script, come segue:

```
echo \
2 \
[04C16B8698A9ABF84250A7C3EA7EEDE
\
[04A2192968D8655D6A935BEAF2CA23E
\
[047E63248B75DB7379BE9CDA8CE5751
```

```
\
[0421D65CBD7149B255382ED7F78E9465
\
[043752580AFA1ECED3C68D446BCAB69
\
5 CHECKMULTISIG \
| bx script-encode | bx sha256 | bx
ripemd160
54c557e07dde5bb6cb791c7a540e0a4796f5
```

La serie di comandi sopra prima codifica lo script di riscatto multisig di Mohammed come script bitcoin con codifica esadecimale serializzata. Il prossimo comando bx calcola l'hash SHA256 di quello. Il prossimo comando bx esegue nuovamente hashing con RIPEMD160, producendo l'hash finale dello script.

L'hash da 20 byte dello script di

riscatto di Mohammed è:

```
54c557e07dde5bb6cb791c7a540e0a4796f5
```

Una transazione P2SH blocca l'output di questo hash anziché dello script più utilizzato, utilizzando lo script di blocco:

```
HASH160
54c557e07dde5bb6cb791c7a540e0a4796f5
EQUAL
```

che, come puoi vedere, è molto più breve. Invece di "pagare per questo script multisignature a 5 chiavi", la transazione equivalente P2SH è "paga a uno script con questo hash". Un cliente che effettua un pagamento alla società di Mohammed deve solo includere questo script di blocco molto più breve nel suo pagamento.

Quando Mohammed e i suoi partner vogliono spendere questo UTXO, devono presentare lo script di riscatto originale (quello il cui hash ha bloccato l'UTXO) e le firme necessarie per sbloccarlo, in questo modo:

```
<Sig1> <Sig2> <2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG>
```

I due script sono combinati in due fasi. Nella prima, lo script di riscatto è comparato contro lo script di locking per essere sicuro che l'hash combaci:

```
<2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG> HASH160 <redeem scriptHash> EQUAL
```

Se l'hash dello script di riscatto combacia, lo script di unlock è

eseguito da solo, per sbloccare lo script di riscatto:

```
<Sig1> <Sig2> 2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG
```

Quasi tutti gli script descritti in questo capitolo possono essere implementati solo come script P2SH. Non possono essere utilizzati direttamente nello script di blocco di un UTXO.

P2SH Addresses

Un'altra parte importante della funzionalità P2SH è la possibilità di codificare un hash di script come un indirizzo, come definito in BIP0013. Gli indirizzi P2SH sono le codifiche Base58Check dell'hash da 20 byte di uno script, proprio come gli indirizzi

bitcoin sono le codifiche Base58Check dell'hash da 20 byte di una chiave pubblica. Gli indirizzi P2SH usano il prefisso di versione "5", che risulta in indirizzi codificati Base58Check che iniziano con un "3".

Ad esempio, lo script complesso di Mohammed, hash e Base58Check-codificati come indirizzo P2SH diventa 39RF6JqABiHdYHkfChV6USGMe6Ns
Possiamo confermarlo con il comando bx:

```
echo \
'54c557e07dde5bb6cb791c7a540e0a4796f5
 | bx address-encode -v 5
39RF6JqABiHdYHkfChV6USGMe6Nsr66G
```

Ora, Mohammed può dare questo

"indirizzo" ai suoi clienti e lo possono usare su quasi tutti i wallet bitcoin per fare un semplice pagamento, come se fosse un indirizzo bitcoin. Il prefisso 3 dà loro un suggerimento che si tratta di un tipo speciale di indirizzo, uno corrispondente a uno script anziché a una chiave pubblica, ma funziona esattamente allo stesso modo di un pagamento a un indirizzo bitcoin.

Gli indirizzi P2SH nascondono tutta la complessità, di modo che la persona che effettua il pagamento non veda lo script.

Benefici di P2SH

La funzione di hash pay-to-script offre i seguenti vantaggi rispetto all'utilizzo

diretto di script complessi negli output di blocco:

- Script complessi sono rimpiazzati da fingerprint più corti nell'output di transazione, rendendo la transazione più piccola.
- Gli script possono essere codificati come un indirizzo, in questo modo il mittente e al wallet del mittente non serve un codice complesso per implementare P2SH.
- P2SH sposta l'onere di costruzione dello script sul ricevente, non sul mittente.
- P2SH sposta l'onere nella memorizzazione dei dati per lo script lungo, dall'output (che si trova nel set UTXO) all'input (memorizzato

sulla blockchain).

- P2SH sposta il carico nella memorizzazione dei dati per lo script lungo, dal tempo presente (pagamento) a un tempo futuro (quando viene speso).
- P2SH sposta il costo della transazione di uno script lungo, dal mittente al destinatario, che deve includere lo script di riscatto lungo per spenderlo.

Script di riscatto e validazione

Prima della versione 0.9.2 del client Bitcoin Core, pay-to-script-hash era limitato ai tipi standard di script di transazione bitcoin, dalla funzione

isStandard(). Ciò significa che lo script di riscatto presentato nella transazione di spesa potrebbe essere solo uno dei tipi standard: P2PK, P2PKH o multi-firma, escludendo OP_RETURN e P2SH stesso.

A partire dalla versione 0.9.2 del client Bitcoin Core, le transazioni P2SH possono contenere qualsiasi script valido, rendendo lo standard P2SH molto più flessibile e consentendo la sperimentazione di molti nuovi e complessi tipi di transazioni.

Non sei in grado di inserire un P2SH in uno script di riscatto P2SH, perché la specifica P2SH non è ricorsiva. Inoltre, mentre è tecnicamente

possibile includere RETURN (vedi Data Recording Output (RETURN)) in uno script redeem, poiché nulla nelle regole impedisce di farlo, non è di pratico utilizzo poiché l'esecuzione di RETURN durante la convalida farà sì che la transazione venga contrassegnata come non valida.

Si noti che poiché lo script di riscatto non viene presentato alla rete finché non si tenta di spendere un output P2SH, se si blocca un output con l'hash di una transazione non valida, verrà elaborato a prescindere. Tuttavia, non potrai spenderlo perché la transazione di spesa, che include lo script di riscatto, non sarà accettata perché è uno script non valido. Questo

crea un rischio, perché puoi bloccare i bitcoin in un P2SH che non potranno più essere spesi successivamente. La rete accetterà l'ingombro P2SH anche se corrisponde a uno redeem script, non valido, poiché l'hash dello script non fornisce alcuna indicazione dello script che rappresenta.

	Gli script blocco P2S contengono l'hash di redeem scri che non fornis alcun indizio s contenuto del script stesso. transazione

ATTENZIONE	P2SH sa considerata valida accettata anc se lo script riscatto non valido. Potre bloccare accidentalmer bitcoin in mo che non possa essere successivame spesi.

Data Recording Output (RETURN)

Il registro di Bitcoin distribuito e con timestamp, la blockchain, ha usi potenziali ben oltre i pagamenti. Molti sviluppatori hanno provato a utilizzare il linguaggio di scripting delle transazioni per sfruttare la sicurezza e la resilienza del sistema per applicazioni come servizi notarili digitali, certificati azionari e contratti intelligenti. I primi tentativi di usare il linguaggio di script di bitcoin per questi scopi implicavano la creazione di output di transazioni che registravano i dati sulla blockchain; ad esempio, per registrare una digital fingerprint (n.d.t. impronta digitale informatica) di un file in modo tale che chiunque possa stabilire la prova

dell'esistenza di quel file in una data specifica facendo riferimento a tale transazione.

L'uso della blockchain di bitcoin per archiviare dati non correlati ai pagamenti con bitcoin è un argomento controverso. Molti sviluppatori considerano abusivo e ne scoraggiano l'utilizzo. Altri articoli sono una dimostrazione delle potenzialità della tecnologia blockchain e ne incoraggiano la sperimentazione. Chi obietta all'inclusione dei dati non relativi a pagamenti afferma che provocano "blockchain bloat", che causa un ulteriore aggravio su chi gestisce i full node, con un aumento del costo di archiviazione su disco per

i dati che non era previsto dovessero essere salvati sulla blockchain. Inoltre, tali transazioni di UTXO che non possono essere spesi, riguardano l'indirizzo bitcoin di destinazione come campo a 20 byte in formato libero. Per cui l'indirizzo che è stato usato per i dati, non corrisponde a una chiave privata e l'UTXO risultante non può essere speso; è un pagamento falso. Queste transazioni che non possono mai essere spese non vengono quindi mai rimosse dal set UTXO e fanno sì che la dimensione del database UTXO aumenti per sempre, "bloat".

Nella versione 0.9 del client Bitcoin Core, è stato raggiunto un

compromesso con l'introduzione dell'operatore RETURN. RETURN consente agli sviluppatori di aggiungere 80 byte di dati non relativi ad un pagamento a un output di transazione. Tuttavia, a differenza dell'uso di UTXO "falso", l'operatore RETURN crea un output specifico *provably unspendable*, che non ha bisogno di essere memorizzato nel set UTXO. RETURN gli output sono registrati sulla blockchain, quindi consumano spazio su disco e contribuiscono all'aumento delle dimensioni della blockchain, ma non sono memorizzati nel set UTXO e quindi non gonfiano il pool di memoria UTXO e non caricano la costosa RAM

dei full node.

Gli script RETURN sono fatti così:

```
RETURN <data>
```

La porzione di dati è limitata a 80 byte e più spesso rappresenta un hash, ad esempio l'output dall'algoritmo SHA256 (32 byte). Molte applicazioni mettono un prefisso davanti ai dati per aiutare a identificare l'applicazione. Ad esempio, il servizio di notarile digitale Proof of Existence utilizza il prefisso di 8 byte "DOCPROOF", che è codificato in ASCII in formato esadecimale 44 4f 43 50 52 4f 4f 46.

Tieni presente che non esiste uno "script di sblocco" che corrisponda a OP_RETURN che potrebbe essere

usato per "spendere" un output di OP_RETURN. L'intero punto di RETURN è che non è possibile spendere i soldi bloccati in quell'output, e quindi non è necessario che sia trattenuto nel set UTXO come potenzialmente spendibile- RETURN è *dimostrabilmente non spendibile*. RETURN è solitamente un output con una quantità di bitcoin pari a zero, poiché ogni bitcoin assegnato a tale output viene effettivamente perso per sempre. Se viene rilevato un RETURN dal software di convalida degli script, si ottiene immediatamente l'interruzione dell'esecuzione dello script di convalida e la marcatura della transazione come non valida.

Pertanto, se accidentalmente si fa riferimento a un output RETURN come input in una transazione, quella transazione non è valida.

Una transazione standard (quella che si conforma al controllo isStandard()) può avere solo un output RETURN. Tuttavia, un singolo output RETURN può essere combinato in una transazione con output di qualsiasi altro tipo.

Due nuove opzioni da riga di comando sono state aggiunte in Bitcoin Core dalla versione 0.10. L'opzione datacarrier controlla l'inoltro e l'estrazione di transazioni OP_RETURN, con l'impostazione predefinita su "1" per consentirle.

L'opzione datacarriersize richiede un argomento numerico che specifica la dimensione massima in byte dello script RETURN, 83 byte per impostazione predefinita, che consente un massimo di 80 byte di dati RETURN più un byte dell'opcode RETURN e due byte dell'opcode PUSHDATA.

	OP_RETURN è stato inizialmente proposto con un limite di 80 byte, ma il limite è stato ridotto a 40 byte quando la funzione è stata rilasciata. A febbraio 2015, nella versione 0.10 di

NOTA	Bitcoin Core, il limite è stato riportato a 80 byte. I nodi possono scegliere di non inoltrare o estrarre OP_RETURN, o solo inoltrare e minare OP_RETURN contenenti meno di 80 byte di dati.

Timelocks

I timelock sono restrizioni su transazioni o output che consentono la spesa solo dopo un certo periodo. Bitcoin ha avuto una funzione di timelock a livello di transazione dall'inizio. È implementato dal campo

nLocktime in una transazione. Sono state introdotte due nuove funzionalità di timelock tra la fine del 2015 e la metà del 2016 che offrono timelock a livello UTXO. Questi sono CHECKLOCKTIMEVERIFY e CHECKSEQUENCEVERIFY.

I timelock sono utili per postare le transazioni e bloccare i fondi per una data futura. Ancora più importante, i timelock estendono lo scripting bitcoin nella dimensione del tempo, aprendo la porta a complessi contratti multistep intelligenti.

Transaction Locktime (nLocktime)

Fin dall'inizio, bitcoin ha avuto una

funzionalità di timelock a livello di transazione. Transaction locktime è un'impostazione a livello di transazione (un campo nella struttura dei dati della transazione) che definisce il primo momento in cui una transazione è valida e può essere inoltrata sulla rete o aggiunta alla blockchain. Locktime è anche noto come nLocktime dal nome della variabile utilizzata alla base del codice di Bitcoin Core. È impostato su zero nella maggior parte delle transazioni per indicare propagazione ed esecuzione immediate. Se nLocktime è diverso da zero e inferiore a 500 milioni, viene interpretato come un'altezza di blocco,

il che significa che la transazione non è valida e non viene inoltrata o inclusa nella blockchain prima dell'altezza del blocco specificata. Se è superiore a 500 milioni, viene interpretato come un timestamp di epoca Unix (secondi da gennaio 1-1970) e la transazione non è valida prima del tempo specificato. Le transazioni con nLocktime che specificano un blocco o un orario futuro devono essere mantenute dal sistema di origine e trasmesse alla rete bitcoin solo dopo che diventano valide. Se una transazione viene trasmessa alla rete prima del nLocktime specificato, la transazione verrà rifiutata dal primo nodo come non valida e non verrà

inoltrata ad altri nodi. L'uso di nLocktime equivale a postdatare un controllo cartaceo.

Limitazioni di Transaction locktime

nLocktime ha il limite che, mentre consente di spendere alcuni output in futuro, non rende impossibile spenderli fino a quel momento. Spieghiamo il senso di tale affermazione con il seguente esempio.

Alice firma una transazione spendendo uno dei suoi output nell'indirizzo di Bob e imposta la transazione nLocktime a 3 mesi in futuro. Alice invia quella transazione a Bob. Con questa transazione Alice e Bob sanno

che:

- Bob non può trasmettere la transazione per riscattare i fondi fino alla scadenza di 3 mesi.
- Bob può trasmettere la transazione dopo 3 mesi.

Però:

- Alice può creare un'altra transazione, spendendo due volte gli stessi input senza un locktime. Quindi, Alice può spendere lo stesso UTXO prima che siano trascorsi i 3 mesi.
- Bob non ha alcuna garanzia che Alice non lo faccia.

È importante comprendere i limiti della transazione nLocktime. L'unica garanzia è che Bob non sarà in grado

di riscattarlo prima che siano trascorsi 3 mesi. Non c'è alcuna garanzia che Bob otterrà i fondi. Per ottenere tale garanzia, la restrizione del timelock deve essere posizionata sull'UTXO stesso ed essere parte dello script di blocco, piuttosto che sulla transazione. Questo è ottenuto dalla prossima forma di timelock, chiamata Check Lock Time Verify.

Check Lock Time Verify (CLTV)

A dicembre 2015 è stata introdotta una nuova forma di timelock in bitcoin come aggiornamento tramite soft fork. Basato su una specifica in BIP-65, un nuovo operatore di script chiamato

CHECKLOCKTIMEVERIFY (CLTV) è stato aggiunto al linguaggio di scripting. CLTV è un timelock per output, piuttosto che un timelock per transazione, come nel caso di nLocktime. Ciò consente una maggiore flessibilità nel modo in cui vengono applicati i timelock.

In termini semplici, aggiungendo l'opcode CLTV nello script redeem di un output, si limita l'output stesso, in modo che possa essere speso solo dopo che è trascorso il tempo specificato.

TIP	Mentre nLocktime è un timelock a livello di transazione, CLTV è un

	timelock basato su output.

CLTV non sostituisce nLocktime, ma piuttosto restringe l'UTXO specifico in modo che possano essere spesi solo in una transazione futura con nLocktime impostato su un valore maggiore o uguale.

L'opcode CLTV accetta un parametro come input, espresso come un numero nello stesso formato di nLocktime (altezza del blocco o tempo di epoca Unix). Come indicato dal suffisso VERIFY, CLTV è il tipo di opcode che interrompe l'esecuzione dello script se il risultato è FALSE. Se

risulta TRUE, l'esecuzione continua.

Per bloccare un output con CLTV, lo si inserisce nello script riscatto dell'output nella transazione che crea l'output. Ad esempio, se Alice sta pagando l'indirizzo di Bob, l'output normalmente conterrà uno script P2PKH come questo:

```
DUP HASH160 <Bob's Public Key Hash> EQUALVERIFY CHECKSIG
```

Per bloccarlo in un momento, ad esempio 3 mesi da ora, la transazione sarebbe una transazione P2SH con uno script riscattato come questo:

```
<now + 3 months> CHECKLOCKTIMEVERIFY DROP DUP HASH160 <Bob's Public Key Hash> EQUALVERIFY CHECKSIG
```

dove *ora + 3 mesi* è un blocco di altezza o valore temporale stimato 3 mesi dal momento in cui la transazione è stata estratta: altezza del blocco corrente + 12.960 (blocchi) o tempo di epoca Unix corrente + 7.760.000 (secondi). Per ora, non preoccuparti dell'opcode DROP che segue CHECKLOCKTIMEVERIFY; sarà spiegato a breve.

Quando Bob tenta di spendere questo UTXO, costruisce una transazione che fa riferimento all'UTXO come input. Usa la sua firma e la chiave pubblica nello script di sblocco di quell'input e imposta la transazione nLocktime in modo che sia uguale o superiore al timelock nel set Alice

CHECKLOCKTIMEVERIFY. Bob quindi trasmette la transazione sulla rete bitcoin.

La transazione di Bob viene valutata come segue. Se il parametro CHECKLOCKTIMEVERIFY impostato da Alice è inferiore o uguale al tempo di blocco n della transazione di spesa, l'esecuzione dello script continua (agisce come se fosse stata eseguita "nessuna operazione" o opcode NOP). Altrimenti, l'esecuzione dello script si arresta e la transazione non è più valida.

Più precisamente, CHECKLOCKTIMEVERIFY ha esito negativo e interrompe l'esecuzione, contrassegnando la transazione non

valida se (fonte: BIP-65):

1. la pila è vuota; o
2. l'elemento in cima alla pila è inferiore a 0; o
3. il tipo di lock-time (altezza rispetto al timestamp) del primo elemento dello stack e il campo nLocktime non sono gli stessi; o
4. l'elemento dello stack superiore è maggiore del campo nLocktime della transazione; o
5. il campo nSequence dell'input è 0xffffffff.

	CLTV e nLocktime usano lo stesso formato per descrivere i timelock, sia l'altezza

NOTA	di un blocco che i tempo trascorso ir secondi dall'epoca di Unix. In modo critico se utilizzati insieme, il formato di nLocktime deve corrispondere a quello di CLTV negli output: entrambi devono fare riferimento all'altezza del blocco o al tempo in secondi.

Dopo l'esecuzione, se CLTV è soddisfatto, il parametro temporale che lo ha preceduto rimane come l'elemento in cima allo stack e

potrebbe dover essere eliminato, con DROP, per la corretta esecuzione dei successivi codici di script. Vedrai spesso CHECKLOCKTIMEVERIFY seguito da DROP negli script per questo motivo.

Usando nLocktime in congiunzione con CLTV, lo scenario descritto in Limitazioni di Transaction locktime cambia. Alice non può più spendere i soldi (perché è bloccata con la chiave di Bob) e Bob non può spenderli prima che il periodo di blocco di 3 mesi sia scaduto.

Introducendo la funzionalità di timelock direttamente nel linguaggio di scripting, CLTV ci consente di sviluppare degli script complessi

molto interessanti.

Lo standard è definito in BIP-65 (CHECKLOCKTIMEVERIFY).

Relative Timelocks

nLocktime e CLTV sono entrambi *timelock assoluti* in quanto specificano un punto assoluto nel tempo. Le prossime due caratteristiche di timelock che esamineremo sono i *timelock relativi* in quanto specificano, come condizione di spendere un output, un tempo trascorso dalla conferma dell'output nella blockchain.

I timelock relativi sono utili perché consentono di escludere una catena di due o più transazioni interdipendenti,

imponendo un vincolo temporale su una transazione che dipende dal tempo trascorso dalla conferma di una transazione precedente. In altre parole, l'orologio non inizia a contare fino a quando l'UTXO non viene registrato sulla blockchain. Questa funzionalità è particolarmente utile nei canali bidirezionali di stato e nelle reti Lightning, come vedremo in Canali di Pagamento e Canali di Stato.

I timelock relativi, come i timelock assoluti, sono implementati sia con una funzionalità a livello di transazione che con un codice operativo a livello di script. Il timelock relativo a livello di transazione è implementato come regola di consenso sul valore di

nSequence, un campo di transazione impostato in ogni input di transazione. I timelock relativi a livello di script sono implementati con l'opcode CHECKSEQUENCEVERIFY (CSV).

I timelock relativi sono implementati secondo le specifiche in BIP-68, Relative lock-time using consensus-enforced sequence numbers e BIP-112, CHECKSEQUENCEVERIFY.

BIP-68 e BIP-112 sono stati attivati a maggio 2016 come aggiornamento soft fork alle regole di consenso.

Relative Timelocks con nSequence

I timelock relativi possono essere impostati su ogni input di una

transazione, impostando il campo nSequence in ogni input.

Original meaning of nSequence

Il campo nSequence era originariamente previsto (ma mai correttamente implementato) per consentire la modifica delle transazioni nel mempool. In tale uso, una transazione contenente input con valore nSequence inferiore a 232 - 1 (0xFFFFFFFF) indicava una transazione che non era ancora "finalizzata". Tale transazione sarebbe detenuta nel mempool fino a quando non sostituita da un'altra transazione che spende gli stessi input con un

valore nSequence più alto. Una volta ricevuta una transazione i cui input avevano un valore nSequence di 0xFFFFFFFF, sarebbero considerati "finalizzati" e minati.

Il significato originale di nSequence non è mai stato implementato correttamente e il valore di nSequence è impostato su 0xFFFFFFFF nelle transazioni che non utilizzano i timelock. Per le transazioni con nLocktime o CHECKLOCKTIMEVERIFY, il valore nSequence deve essere impostato su un valore inferiore a 231 affinché i timelock guards abbiano un effetto, come spiegato di seguito.

nSequence come timelock relativo imposto dal consenso

Dall'attivazione di BIP-68, si applicano nuove regole di consenso per qualsiasi transazione contenente un input il cui valore nSequence è inferiore a 231 (bit 1<<31 non impostato). A livello di codice, ciò significa che se il più significativo (bit 1 << 31) non è impostato, è una bandiera che significa "tempo di blocco relativo". In caso contrario (bit 1 << 31 impostato), il valore nSequence è riservato ad altri usi come l'abilitazione di CHECKLOCKTIMEVERIFY, nLocktime, Opt-In-Replace-By-Fee e

altri sviluppi futuri.

Gli input di transazione con valori nSequence inferiori a 2^{31} vengono interpretati come aventi un timelock relativo. Tale transazione è valida solo una volta che l'input è scaduto dall'importo relativo del tempo. Ad esempio, una transazione con un input con un timelock relativo nSequence di 30 blocchi è valida solo quando sono trascorsi almeno 30 blocchi dal momento in cui è stato estratto il riferimento UTXO nell'input. Poiché nSequence è un campo per input, una transazione può contenere un numero qualsiasi di input timelock, che devono essere sufficientemente invecchiati affinché la transazione sia valida. Una

transazione può includere entrambi gli input timelock (nSequence < 231) e gli input senza un timelock relativo (nSequence >= 231).

Il valore nSequence è specificato in blocchi o secondi, ma in un formato leggermente diverso da quello che abbiamo visto utilizzato in nLocktime. Viene usato un flag di tipo per distinguere tra valori che contano blocchi e valori che contano il tempo in secondi. Il flag di tipo è impostato nel 23° bit meno significativo (vale a dire, valore 1 << 22). Se il flag di tipo è impostato, il valore nSequence viene interpretato come un multiplo di 512 secondi. Se il flag di tipo non è impostato, il valore nSequence viene

interpretato come un numero di blocchi.

Quando si interpreta nSequence come un timelock relativo, vengono considerati solo i 16 bit meno significativi. Una volta valutati i flag (bit 32 e 23), il valore nSequence viene solitamente "mascherato" con una maschera a 16 bit (ad esempio, nSequence e 0x0000FFFF).

La Figura 44 mostra il layout binario del valore nSequence, come definito da BIP-68.

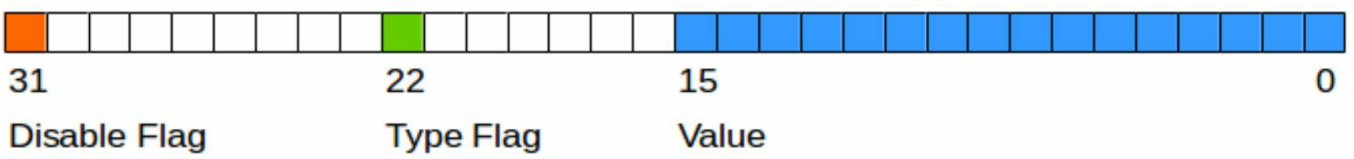


Figura 44. Definizione BIP-

68 della codifica nSequence (Fonte: BIP-68)

I timelock relativi basati sull'applicazione del consenso del valore nSequence sono definiti in BIP-68.

Lo standard è definito in BIP-68, Relative lock-time using consensus-enforced sequence numbers.

Relative Timelocks con CSV

Proprio come CLTV e nLocktime, esiste un opcode dello script per i timelock relativi che sfrutta il valore nSequence negli script. L'opcode è CHECKSEQUENCEVERIFY, comunemente definito CSV.

L'opcode CSV quando valutato nello script di riscatto di UTXO consente di spendere solo in una transazione il cui valore nSequence di input è maggiore o uguale al parametro CSV. In sostanza, ciò limita la spesa dell'UTXO fino a quando non è trascorso un certo numero di blocchi o secondi rispetto al momento in cui l'UTXO è stato estratto.

Come con CLTV, il valore in CSV deve corrispondere al formato nel valore nSequence corrispondente. Se CSV è specificato in termini di blocchi, deve esserlo anche nSequence. Se CSV è specificato in termini di secondi, deve esserlo anche nSequence.

I timelock relativi con CSV sono particolarmente utili quando diverse transazioni (concatenate) vengono create e firmate, ma non propagate, e mantenute "fuori catena". Una transazione figlia non può essere utilizzata fino a quando la transazione madre non è stata propagata, estratta e invecchiata entro il tempo specificato nel relativo timelock. Un'applicazione di questo caso d'uso può essere vista i n Canali di Pagamento e Canali di Stato and Canali di Pagamento Instradati (Lightning Network).

CSV è definito in dettaglio in BIP-112, CHECKSEQUENCEVERIFY.

Median-Time-Past

Come parte dell'attivazione dei relativi timelock, si è anche verificato un cambiamento nel modo in cui il "tempo" è calcolato per i timelock (sia assoluti che relativi). In bitcoin c'è una sottile, ma molto significativa, differenza tra tempo di wall e tempo di consenso. Bitcoin è una rete decentralizzata, il che significa che ogni partecipante ha una propria prospettiva temporale. Gli eventi sulla rete non avvengono istantaneamente ovunque. La latenza di rete deve essere fattorizzata nella prospettiva di ciascun nodo. Alla fine tutto è sincronizzato per creare un libro mastro comune. Bitcoin raggiunge il consenso ogni 10 minuti circa lo stato

del libro mastro come esisteva in *passato*.

I timestamp impostati nelle intestazioni dei blocchi sono impostati dai minatori. Esiste un certo grado di libertà consentito dalle regole di consenso per tenere conto delle differenze nella precisione dell'orologio tra i nodi decentralizzati. Tuttavia, questo crea uno spiacevole incentivo per i minatori a mentire circa il tempo in un blocco in modo da guadagnare commissioni extra includendo operazioni scadute e non ancora mature.

Per rimuovere l'incentivo a mentire e rafforzare la sicurezza dei timelock, è stato proposto e attivato un PIF

contemporaneamente ai PIF per i timelock relativi. Questo è BIP-113, che definisce una nuova misurazione del consenso del tempo chiamata *Median-Time-Past*.

Median-Time-Past è calcolato prendendo il timestamp degli ultimi 11 blocchi e individuando la mediana. Il tempo mediano diventa quindi il tempo di consenso e viene utilizzato per tutti i calcoli del timelock. Prendendo il punto medio da circa due ore nel passato, si riduce l'influenza del timestamp di qualsiasi blocco. Incorporando 11 blocchi, nessun singolo minatore può influenzare il timestamp al fine di ottenere commissioni da transazioni con un

timelock che non è ancora maturato.

Median-Time-Past cambia l'implementazione dei calcoli temporali per nLocktime, CLTV, nSequence e CSV. Il tempo di consenso calcolato da Median-Time-Past è sempre di circa un'ora dietro l'ora di wall. Se crei transazioni di timelock, dovresti tenerne conto quando valuti il valore desiderato da codificare in nLocktime, nSequence, CLTV e CSV.

Median-Time-Past è specificato in BIP-113.

Difesa dei Timelock Contro il Fee Sniping

Il Fee-Sniping è uno scenario di

attacco teorico, in cui i minatori che tentano di riscrivere i blocchi passati "colpiscono" (“snipe”) le transazioni a più alto costo dai blocchi futuri per massimizzare la redditività.

Ad esempio, supponiamo che il blocco più alto esistente sia il blocco n. 100.000. Se invece di tentare di estrarre il blocco #100,001 per estendere la catena, alcuni minatori tentano di ri-minare a #100.000. Questi minatori possono scegliere di includere qualsiasi transazione valida (che non sia ancora stata estratta) nel loro blocco candidato n. 100.000. Non devono rimodellare il blocco con le stesse transazioni. In effetti, hanno l'incentivo a selezionare le transazioni

più redditizie (commissione più alta per kB) da includere nel loro blocco. Possono includere tutte le transazioni che si trovavano nel "vecchio" blocco n. 100.000, nonché tutte le transazioni dal mempool corrente. Essenzialmente hanno la possibilità di trasferire le transazioni dal "presente" al "passato" riscritto quando ricreano il blocco n. 100.000.

Oggi, questo attacco non è molto redditizio, perché la ricompensa del blocco è molto più alta delle commissioni totali per blocco. Ma ad un certo punto nel futuro, le commissioni di transazione saranno la maggior parte del premio (o anche l'intero importo del premio). A quel

tempo, questo scenario diventa inevitabile.

Per prevenire il "fee sniping", quando Bitcoin Core crea transazioni, utilizza nLocktime per limitarli al "blocco successivo", per impostazione predefinita. Nel nostro scenario, Bitcoin Core imposterà nLocktime su 100.001 su ogni transazione creata. In circostanze normali, questo nLocktime non ha alcun effetto: le transazioni possono essere incluse solo nel blocco n. 100.001; è il prossimo blocco.

Ma con un attacco fork della blockchain, i miner non sarebbero in grado di minare transazioni con fee alte dal mempool, perché tutte quelle transazioni sarebbero bloccate a tempo

per bloccare il numero 100.001. Possono solo rimpiazzare 100.000 con qualsiasi transazione valida in quel momento, essenzialmente senza guadagnare nuove commissioni.

Per raggiungere questo obiettivo, Bitcoin Core imposta il nLocktime su tutte le nuove transazioni su <current block # + 1> e imposta nSequence su tutti gli input su 0xFFFFFFFE per abilitare nLocktime.

Script con Controllo di Flusso (Conditional Clauses)

Una delle più potenti funzionalità di

Bitcoin Script è il controllo del flusso, noto anche come clausole condizionali. Probabilmente hai familiarità con il controllo del flusso in vari linguaggi di programmazione che usano il costrutto IF...THEN...ELSE. Le clausole condizionali di Bitcoin sembrano un po 'diverse, ma sono essenzialmente lo stesso costrutto.

A livello base, gli opcode condizionali bitcoin ci permettono di costruire uno script di riscatto che ha due modi di essere sbloccato, a seconda di un risultato VERO/FALSO di valutazione di una condizione logica. Ad esempio, se x è TRUE, lo script di riscatto è A e lo script di

riscatto ELSE è B.

Inoltre, le espressioni condizionali di bitcoin possono essere "annidate" indefinitamente, il che significa che una clausola condizionale può contenerne un'altra all'interno di essa, che ne contiene un'altra, ecc. Il controllo del flusso di Bitcoin Script può essere usato per costruire script molto complessi con centinaia o persino migliaia di possibili percorsi di esecuzione. Non vi è alcun limite alla nidificazione, ma le regole di consenso impongono un limite alla dimensione massima, in byte, di uno script.

Bitcoin implementa il controllo del flusso utilizzando gli opcode IF,

ELSE, ENDIF e NOTIF. Inoltre, le espressioni condizionali possono contenere operatori booleani come BOOLAND, BOOLOR e NOT.

A prima vista, potresti trovare gli script di controllo del flusso di bitcoin confusionari. Questo perché Bitcoin Script è un linguaggio di stack. Allo stesso modo in cui 1 + 1 sembra "all'indietro" quando espresso come 1 1 ADD, le clausole di controllo del flusso in bitcoin sembrano anch'esse "all'indietro".

Nella maggior parte dei linguaggi di programmazione (procedurali) tradizionali, il controllo di flusso ha il seguente aspetto:

Pseudocodice del controllo del flusso nella maggior parte dei linguaggi di programmazione

```
if (condition):
  code to run when condition is true
else:
  code to run when condition is false
code to run in either case
```

In un linguaggio basato su stack come Bitcoin Script, la condizione logica viene prima dell'IF, il che fa sembrare "all'indietro", come questo:

Controllo del flusso di Bitcoin Script

```
condition
```

```
IF
  code to run when condition is true
ELSE
  code to run when condition is false
ENDIF
code to run in either case
```

Durante la lettura di Bitcoin Script, ricorda che la condizione da valutare viene *prima* dell'opcode IF.

Clausole Condizionali con Opcodes VERIFY

Un'altra forma di condizionale in Bitcoin Script è qualsiasi opcode che termina in VERIFY. Il suffisso VERIFY significa che se la condizione valutata non è TRUE, l'esecuzione dello script termina immediatamente e la transazione è considerata non

valida.

A differenza di una clausola IF, che offre percorsi di esecuzione alternativi, il suffisso VERIFY funge d a *clausola di salvaguardia*, continuando solo se viene soddisfatta una condizione preliminare.

Ad esempio, il seguente script richiede la firma di Bob e una pre-immagine (segreta) che produce un hash specifico. Entrambe le condizioni devono essere soddisfatte per sbloccare:

Un redeem script con una clausola di salvaguardia EQUALVERIFY.

```
HASH160 <expected hash>
EQUALVERIFY <Bob's Pubkey>
CHECKSIG
```

Per riscattarlo, Bob deve costruire uno script di sblocco che presenti una pre-immagine valida e una firma:

Uno script di sblocco per soddisfare il redeem script sopra citato

```
<Bob's Sig> <hash pre-image>
```

Senza presentare la pre-immagine, Bob non può accedere alla parte dello script che controlla la sua firma.

Questo script può anche essere scritto con un IF:

Un redeem script con una

clausola di salvaguardia IF

```
HASH160 <expected hash> EQUAL
IF
  <Bob's Pubkey> CHECKSIG
ENDIF
```

Lo script di sblocco di Bob è identico:

Uno script di sblocco per soddisfare il redeem script sopra citato

```
<Bob's Sig> <hash pre-image>
```

Lo script con IF fa la stessa cosa dell'uso di un opcode con un suffisso VERIFY; entrambi funzionano come clausole di guardia. Tuttavia, la costruzione VERIFY è più efficiente, utilizzando due codici opzionali minori.

Quindi, quando utilizzare VERIFY e quando IF? Se tutto ciò che stiamo cercando di fare è collegare una precondizione (clausola di salvaguardia), allora VERIFY è migliore. Se invece vogliamo avere più di un percorso di esecuzione (controllo del flusso), allora abbiamo bisogno di una clausola IF... ELSE per il controllo del flusso.

TIP	Un codice operativo come EQUAL spingerà il risultato (TRUE/FALSE) sullo stack, lasciandolo lì per la valutazione da opcodes successivi. Al contrario, l'opcode EQUALVERIFY non

	lascia nulla in stack. Gli opcode che terminano in VERIFY non lasciano il risultato in stack.

Utilizzo del Controllo di Flusso negli Script

Un uso molto comune per il controllo del flusso in Bitcoin Script è quello di costruire uno script di riscatto che offre più percorsi di esecuzione, ognuno un modo diverso di riscattare l'UTXO.

Diamo un'occhiata a un semplice esempio, in cui abbiamo due firmatari, Alice e Bob, e uno dei due è in grado di riscattare. Con multisig, questo

sarebbe espresso come uno script multisig 1-di-2. Per dimostrarlo, faremo la stessa cosa con una clausola IF:

```
IF
 <Alice's Pubkey> CHECKSIG
ELSE
 <Bob's Pubkey> CHECKSIG
ENDIF
```

Guardando questo script di riscatto, ci si potrebbe chiedere: "Dov'è la condizione? Non c'è nulla che preceda la clausola IF!"

La condizione non fa parte dello script di riscatto. Invece, la condizione verrà offerta nello script di sblocco, consentendo ad Alice e Bob di "scegliere" il percorso di esecuzione

desiderato.

Alice lo riscatta con lo script di sblocco:

```
<Alice's Sig> 1
```

L'1 alla fine funge da condizione (TRUE) che farà in modo che la clausola IF esegua il primo percorso di redenzione per cui Alice ha una firma.

Per riscattarlo, Bob dovrebbe scegliere il secondo percorso di esecuzione assegnando un valore FALSE alla clausola IF:

```
<Bob's Sig> 0
```

Lo script di sblocco di Bob mette uno 0 in stack, facendo sì che la clausola IF esegua il secondo script (ELSE),

che richiede la firma di Bob.

Poiché le clausole IF possono essere annidate, possiamo creare un "labirinto" di percorsi di esecuzione. Lo script di sblocco può fornire una "mappa" che seleziona il percorso di esecuzione effettivamente eseguito:

```
IF
  script A
ELSE
  IF
    script B
  ELSE
    script C
  ENDIF
ENDIF
```

In questo scenario, esistono tre percorsi di esecuzione (script A, script B e script C). Lo script di sblocco

fornisce un percorso sotto forma di una sequenza di valori VERO o FALSO. Per selezionare lo script di percorso B, ad esempio, lo script di sblocco deve terminare in 1 0 (TRUE, FALSE). Questi valori verranno inseriti nello stack, in modo che il secondo valore (FALSE) finisca in cima allo stack. La clausola IF esterna apre il valore FALSE ed esegue la prima clausola ELSE. Quindi il valore VERO si sposta in cima allo stack e viene valutato dall'IF interno (nidificato), selezionando il percorso di esecuzione B.

Usando questo costrutto, possiamo redimere script con decine o centinaia di percorsi di esecuzione, ognuno dei

quali offre un modo diverso di riscattare l'UTXO. Per spendere, costruiamo uno script di sblocco che naviga nel percorso di esecuzione inserendo i valori TRUE e FALSE appropriati nello stack in ogni punto di controllo del flusso.

Complex Script Example

In questa sezione combiniamo molti dei concetti di questo capitolo in un unico esempio.

Il nostro esempio utilizza la storia di Mohammed, il proprietario dell'azienda a Dubai che sta gestendo un'attività di import/export.

In questo esempio, Mohammed desidera costruire un conto aziendale con regole flessibili. Lo schema che crea richiede diversi livelli di autorizzazione in base ai timelock. I partecipanti allo schema multisig sono Mohammed, i suoi due partner Saeed e Zaira, e il loro avvocato della compagnia Abdul. I tre partner prendono decisioni basate su una regola di maggioranza, quindi due dei tre devono essere d'accordo. Tuttavia, in caso di problemi con le loro chiavi, vogliono che il loro avvocato sia in grado di recuperare i fondi con una delle tre firme dei partner. Infine, se tutti i partner sono indisponibili o inabili per un po', vogliono che

l'avvocato sia in grado di gestire direttamente l'account.

Ecco lo script di riscatto che Mohammed progetta per raggiungere tutto ciò (prefisso del numero di riga come XX):

Multi-firma variabile con Timelock

```
01  IF
02    IF
03      2
04    ELSE
05      <30 days>
CHECKSEQUENCEVERIFY DROP
06      <Abdul the Lawyer's Pubkey>
CHECKSIGVERIFY
07      1
08    ENDIF
```

```
09    <Mohammed's Pubkey> <Saeed's
Pubkey> <Zaira's Pubkey> 3
CHECKMULTISIG
10  ELSE
11    <90 days>
CHECKSEQUENCEVERIFY DROP
12    <Abdul the Lawyer's Pubkey>
CHECKSIG
13  ENDIF
```

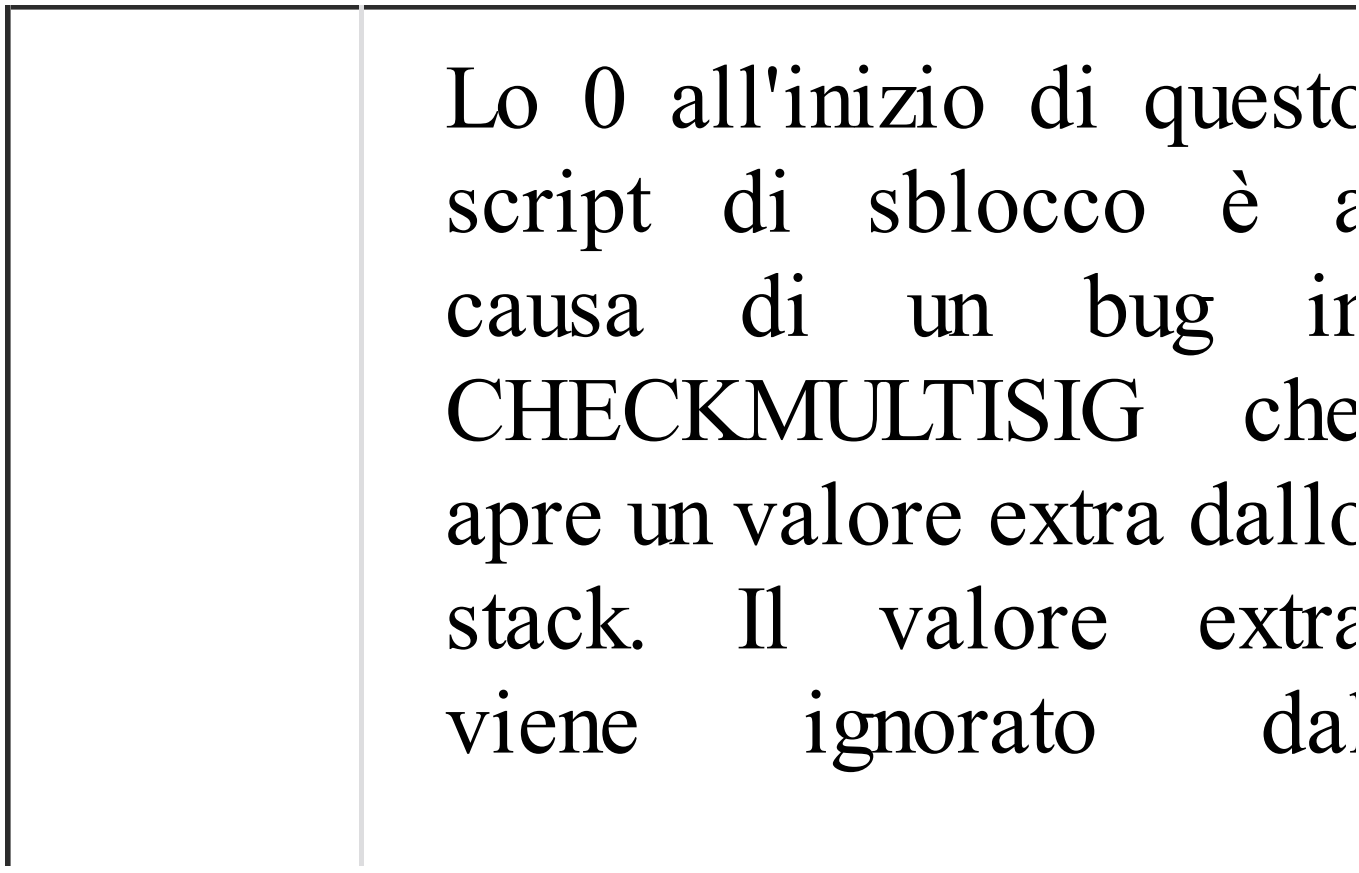
Lo script di Mohammed implementa tre percorsi di esecuzione usando le clausole di controllo del flusso IF...ELSE annidate.

Nel primo percorso di esecuzione, questo script funziona come un semplice multisig 2-di-3 con i tre partner. Questo percorso di esecuzione è costituito dalle righe 3 e 9. La riga 3 imposta il quorum del multisig su 2 (2-

di-3). Questo percorso di esecuzione può essere selezionato mettendo TRUE TRUE alla fine dello script di sblocco:

Script di sblocco per il primo percorso di esecuzione (multisig 2-di-3)

```
0 <Mohammed's Sig> <Zaira's Sig> TRUE TRUE
```



TIP	CHECKMULTISIG, ma deve essere presente o lo script non funzionerà come previsto. Scegliere 0 (di solito) è una soluzione al bug, come descritto in Un bug nell'esecuzione di CHECKMULTISIG.

Il secondo percorso di esecuzione può essere utilizzato solo dopo che sono trascorsi 30 giorni dalla creazione dell'UTXO. A quel tempo, richiede la firma di Abdul, l'avvocato e uno dei tre partner (un multisig 1-di-3). Ciò è ottenuto dalla riga 7, che imposta il quorum per il multisig su 1. Per

selezionare questo percorso di esecuzione, lo script di sblocco termina in FALSE TRUE:

Script di sblocco per il secondo percorso di esecuzione (Avvocato + 1-di-3)

```
0 <Saeed's Sig> <Abdul's Sig> FALSE TRUE
```

TIP	Perché FALSE TRUE? Non è ‘all’indietro’? Poiché i due valori vengono inseriti nello stack, con FALSE per primo, quindi TRUE per secondo. TRUE viene

	quindi inserito prima dal primo codice operativo IF.

Infine, il terzo percorso di esecuzione consente all'avvocato Abdul di spendere i fondi da solo, ma solo dopo 90 giorni. Per selezionare questo percorso di esecuzione, lo script di sblocco deve terminare con FALSE:

Script di sblocco per il terzo percorso di esecuzione (solo per l'avvocato)

```
<Abdul's Sig> FALSE
```

Prova a eseguire lo script su carta per vedere come si comporta nello stack.

Altre cose da considerare quando leggi questo esempio. Vedi se riesci a trovare le risposte:

- Perché l'avvocato non può riscattare il terzo percorso di esecuzione in qualsiasi momento selezionandolo con FALSE sullo script di sblocco?
- Quanti percorsi di esecuzione possono essere utilizzati per 5, 35 e 105 giorni, rispettivamente, dopo che l'UTXO è stato estratto?
- I fondi sono persi se l'avvocato perde la chiave? La tua risposta cambia se sono trascorsi 91 giorni?

- Come fanno i partner ad "azzerare" l'orologio ogni 29 o 89 giorni per impedire all'avvocato di accedere ai fondi?
- Perché alcuni opcode di CHECKSIG in questo script hanno il suffisso VERIFY mentre altri no?

Segregated Witness

Segregated Witness (segwit) è un aggiornamento delle regole di consenso del bitcoin e del protocollo di rete, proposto e implementato come un soft-fork BIP-9 attivato sulla mainnet di bitcoin il 1 ° agosto 2017.

In crittografia, il termine "testimone" (witness) è usato per descrivere una soluzione a un puzzle crittografico. In bitcoin, il testimone soddisfa una condizione crittografica posizionata su un output di transazione non speso (UTXO).

Nel contesto di bitcoin, una firma digitale è un tipo di testimone, ma un testimone è più ampiamente una soluzione in grado di soddisfare le condizioni imposte ad un UTXO e sbloccare quello stesso UTXO per essere speso. Il termine "testimone" è un termine più generico per uno "script di sblocco" o "scriptSig".

Prima dell'introduzione di segwit, ogni input in una transazione era seguito dai

dati del testimone che la sbloccavano. I dati del testimone sono stati incorporati nella transazione come parte di ciascun input. Il termine *segregated witness*, o *segwit* in breve, significa semplicemente separare la firma o lo script di sblocco di uno specifico output. Pensa "scriptSig separato" o "firma separata" nella forma più semplice.

Segregated Witness è quindi una modifica architettonica a bitcoin che mira a spostare i dati del testimone dal campo scriptSig (script di sblocco) di una transazione in una struttura di dati separata che accompagna una transazione. I clienti possono richiedere i dati delle transazioni con

o senza i relativi dati dei testimoni.

In questa sezione esamineremo alcuni dei vantaggi di Segregated Witness, descrivendone il meccanismo utilizzato per distribuire e implementare questa modifica dell'architettura e dimostrandone l'uso nelle transazioni e negli indirizzi.

Segregated Witness è definito dai seguenti BIP:

BIP-141

La definizione principale di Segregated Witness.

BIP-143

Verifica della firma della transazione per la versione 0 di SegWit

BIP-144

Servizi peer: nuovi messaggi di rete e formati di serializzazione

BIP-145

getblocktemplate Aggiornamenti per Segregated Witness (per mining)

BIP-173

Formato di indirizzo Base32 per output nativi witness v0-16

Perchè Segregated Witness?

Segregated Witness è un cambiamento architettonico che ha diversi effetti sulla scalabilità, la sicurezza, gli incentivi economici e le prestazioni di bitcoin:

Transaction Malleability

Spostando il testimone al di fuori

della transazione, l'hash della transazione utilizzato come identificatore non include più i dati del testimone stesso. Poiché i dati dei testimoni sono l'unica parte della transazione che può essere modificata da una terza parte (vedi Identificativi delle transazioni), la rimozione rimuove anche l'opportunità di attacchi di malleabilità delle transazioni. Con Segregated Witness, gli hash delle transazioni diventano immutabili da chiunque non sia il creatore della transazione stessa, il che migliora notevolmente l'implementazione di molti altri protocolli basati sulla costruzione di transazioni bitcoin avanzate, come i

canali di pagamento, le transazioni concatenate e Lightning Network.

Script Versioning

Con l'introduzione degli script Segregated Witness, ogni script di blocco è preceduto da un *numero di versione di script*, simile a come le transazioni e i blocchi hanno numeri di versione. L'aggiunta di un numero di versione di script consente di aggiornare il linguaggio di scripting in un modo compatibile con le versioni precedenti (ad esempio, utilizzando gli aggiornamenti di soft fork) per introdurre nuovi script, sintassi o semantica. La possibilità di aggiornare il linguaggio di scripting in modo non distruttivo accelera

notevolmente il tasso di innovazione in bitcoin.

Scalabilità di Rete e Archiviazione

I dati dei testimoni contribuiscono alla dimensione totale di una transazione. Gli script più complessi come quelli usati per i canali multisig o di pagamento sono molto grandi. In alcuni casi questi script rappresentano la maggioranza (oltre il 75%) dei dati in una transazione. Spostando i dati del testimone al di fuori della transazione, Segregated Witness migliora la scalabilità di bitcoin. I nodi possono sfoltire i dati del testimone dopo aver convalidato le firme, o ignorarlo del tutto quando si effettua una verifica di pagamento

semplificata. I dati dei testimoni non devono essere trasmessi a tutti i nodi e non devono essere memorizzati su disco da tutti i nodi.

Ottimizzazione della Verifica della Firma

Segregated Witness aggiorna le funzioni di firma (CHECKSIG, CHECKMULTISIG, ecc.) per ridurre la complessità computazionale dell'algoritmo. Prima di segwit, l'algoritmo utilizzato per produrre una firma richiedeva un numero di operazioni hash che era proporzionale alla dimensione della transazione. I calcoli per l'hashing dei dati sono aumentati in $O(n_2)$ rispetto al numero di operazioni di

firma, introducendo un notevole onere computazionale su tutti i nodi che verificano la firma. Con segwit, l'algoritmo viene modificato per ridurre la complessità a O (n).

Miglioramento della Firma Offline

Le firme Segregated Witness incorporano il valore (importo) a cui fa riferimento ciascun input nell'hash firmato. In precedenza, un dispositivo di firma offline, ad esempio un hardware wallet, doveva verificare la quantità di ciascun input prima di firmare una transazione. Di solito, ciò avveniva tramite lo streaming di una grande quantità di dati circa le transazioni precedenti a cui si faceva riferimento come input. Poiché

l'importo fa ora parte dell'hash di commit firmato, un dispositivo offline non ha bisogno delle transazioni precedenti. Se gli importi non corrispondono (sono rappresentati in modo errato da un sistema online compromesso), la firma non sarà valida.

Come funziona Segregated Witness

A prima vista, Segregated Witness sembra essere una modifica al modo in cui vengono costruite le transazioni e quindi una funzionalità a livello di transazione, ma non lo è. Piuttosto, Segregated Witness è una modifica del modo in cui vengono spesi i singoli

UTXO e quindi è una caratteristica circa gli output.

Una transazione può spendere output Segregated Witness o output tradizionali (inline-witness) o entrambi. Pertanto, non ha molto senso riferirsi a una transazione come "transazione Segregated Witness". Piuttosto, dovremmo fare riferimento a specifici output di transazione come "output Segregated Witness".

Quando una transazione spende un UTXO, deve fornire un testimone. In un UTXO tradizionale, lo script di blocco richiede che i dati dei testimoni siano forniti *in linea* nella parte di input della transazione che spende l'UTXO. Un UTXO Segregated

Witness, tuttavia, specifica uno script di blocco che può essere soddisfatto con i dati del testimone al di fuori dell'input (segregato).

Soft Fork (Backward Compatibility)

Segregated Witness rappresenta un cambiamento significativo nel modo in cui gli output e le transazioni sono architettate. Un tale cambiamento richiederebbe normalmente un cambiamento simultaneo in ogni nodo e wallet bitcoin per modificare le regole di consenso, ciò che è noto come un hard fork. Invece, Segregated Witness è stato introdotto con un cambiamento molto meno dirompente,

che è compatibile con le versioni precedenti, noto come una soft fork. Questo tipo di aggiornamento consente al software non aggiornato di ignorare le modifiche e continuare a funzionare senza interruzioni.

Gli output Segregated Witness sono costruiti in modo tale che i sistemi più vecchi che non sono ancora consapevoli di segwit possano ancora validarli. Per un vecchio wallet o nodo, un output Segregated Witness assomiglia a un output che *chiunque può spendere*. Tali output possono essere spesi con una firma vuota, quindi il fatto che non ci sia firma all'interno della transazione (è segregata) non invalida la transazione.

I nuovi wallet e i nodi, tuttavia, vedono l'output Segregated Witness e si aspettano di trovare un valido testimone nei dati dei testimoni della transazione.

Esempi di Output e Transazioni Segregated Witness

Esaminiamo alcune delle transazioni di esempio e vediamo come cambierebbero con Segregated Witness. Dapprima vedremo come un pagamento Pay-to-Public-Key-Hash (P2PKH) viene trasformato con il Segregated Witness. Quindi, esamineremo l'equivalente SegWit per gli script Pay-to-Script-Hash (P2SH).

Infine, vedremo come entrambi i precedenti programmi Segregated Witness possono essere incorporati all'interno di uno script P2SH.

Pay-to-Witness-Public-Key-Hash (P2WPKH)

In Pagare un Caffè, Alice ha creato una transazione per pagare a Bob una tazza di caffè. Quella transazione ha creato un output P2PKH con un valore di 0.015 BTC spendibile da Bob. Lo script dell'output è simile al seguente:

Esempio di script output P2PKH

```
DUP                                    HASH160
ab68025513c3dbd2f7b92a94e0581f5d50f65
```

```
EQUALVERIFY CHECKSIG
```

Con Segregated Witness, Alice crea uno script Pay-to-Witness-Public-Key-Hash (P2WPKH), che assomiglia a questo:

Esempio di script output P2WPKH

```
0
ab68025513c3dbd2f7b92a94e0581f5d50f65
```

Come puoi vedere, uno script di blocco di un output Segregated Witness è molto più semplice di un output tradizionale. Consiste di due valori che vengono inseriti nello stack di valutazione degli script. Per un client bitcoin obsoleto (non a conoscenza di segwit), i due push

apparirebbero come un output che chiunque può spendere e non richiede una firma (o meglio, può essere speso con una firma vuota). Per un client più recente e sensibile al settaggio, il primo numero (0) viene interpretato come un numero di versione (la *versione witness*) e la seconda parte (20 byte) è l'equivalente di uno script di blocco noto come *programma witness*. Il programma witness a 20 byte è semplicemente l'hash della chiave pubblica, come in uno script P2PKH

Ora, diamo un'occhiata alla transazione corrispondente che Bob usa per utilizzare questo output. Per lo script originale (non segwit), la

transazione di Bob dovrebbe includere una firma all'interno dell'input della transazione:

Transazione decodificata che mostra un output P2PKH speso con una firma

```
[...]
“Vin” : [
"txid":
"0627052b6f28912f2703066a912ea577f2ce
"vout": 0,
    "scriptSig": “<Bob’s scriptSig>”,
]
[...]
```

Tuttavia, per passare l'output Segregated Witness, la transazione non ha firma su quell'input. Invece, la transazione di Bob ha uno script vuoto

e include un Segregated Witness, al di fuori della transazione stessa:

Transazione decodificata che mostra un output P2WPKH speso con dati di testimone separati

```
[...]
“Vin” : [
"txid":
"0627052b6f28912f2703066a912ea577f2ce
"vout": 0,
    "scriptSig": “”,
]
[...]
“witness”: “<Bob’s witness data>”
[...]
```

Costruzione del wallet

P2WPKH

È estremamente importante notare che P2WPKH deve essere creato solo dal beneficiario (destinatario) e non convertito dal mittente da una chiave pubblica nota, uno script P2PKH o un indirizzo. Il mittente non ha modo di sapere se il portafoglio del destinatario ha la capacità di costruire transazioni segwit e di spendere output P2WPKH.

Inoltre, gli output P2WPKH devono essere costruiti dall'hash di una chiave pubblica *compressa*. Le chiavi pubbliche non compresse non sono standard in segwit e possono essere disabilitate esplicitamente da un futuro soft fork. Se l'hash utilizzato nel

P2WPKH proviene da una chiave pubblica non compressa, potrebbe non essere spendibile e potresti perdere fondi. Gli output P2WPKH devono essere creati dal portafoglio del beneficiario derivando una chiave pubblica compressa dalla propria chiave privata.

	P2WPKH deve essere costruito dal beneficiario (destinatario) convertendo una chiave pubblica compressa in

ATTENZIONE	un hash P2WPKH. Non dovresti mai trasformare uno script P2PKH, un indirizzo bitcoin o una chiave pubblica non compressa in uno script di test P2WPKH

Pay-to-Witness-Script-Hash (P2WSH)

Il secondo tipo di programma witness corrisponde ad uno script Pay-to-Script-Hash (P2SH). Abbiamo visto questo tipo di script in Pay-to-Script-Hash (P2SH). In questo esempio, P2SH è stato utilizzato dalla società di Mohammed per esprimere uno script multifirma. I pagamenti alla compagnia di Mohammed sono stati codificati con uno script di blocco come questo:

Esempio di script di output P2SH

```
HASH160
54c557e07dde5bb6cb791c7a540e0a4796f5
EQUAL
```

Questo script P2SH fa riferimento all'hash di uno *script redeem* che

definisce un requisito di multifirma 2-di-3 per spendere fondi. Per spendere questo output, la compagnia di Mohammed presenterebbe lo script di riscatto (il cui hash corrisponde all'hash dello script nell'output di P2SH) e le firme necessarie per soddisfare lo script di riscatto, tutto all'interno dell'input della transazione:

Transazione decodificata che mostra un output P2SH speso

```
[...]
“Vin” : [
"txid": "abcdef12345...",
"vout": 0,
    "scriptSig": “<SigA> <SigB> <2
PubA PubB PubC PubD PubE 5
```

```
CHECKMULTISIG>”,
]
```

Ora, diamo un'occhiata a come questo intero esempio dovrebbe essere aggiornato in segwit. Se i clienti di Mohammed utilizzano un portafoglio compatibile con la tecnologia segwit, effettuano un pagamento, creando un output Pay-to-Witness-Script-Hash (P2WSH) che sarebbe simile a questo:

Esempio di uno script di output P2WSH

```
0
a9b7b38d972cabc7961dbfbcb841ad4508d13
```

Di nuovo, come con l'esempio di P2WPKH, puoi vedere che lo script Segregated Witness equivalente è

molto più semplice e omette i vari script che vedi negli script P2SH. Invece, il programma Segregated Witness consiste di due valori spinti nello stack: una versione di testimone (0) e l'hash SHA256 a 32 byte dello script di riscatto.

	Mentre P2SH utilizza l'hash RIPEMD160 (SHA256 (script)) da 20 byte, il programma witness P2WSH utilizza un hash SHA256 (script) a 32 byte. Questa differenza nella selezione dell'algoritmo di hashing è intenzionale e viene utilizzato per

TIP	distinguere tra i due tipi di programmi witness (P2WPKH e P2WSH) per la lunghezza dell'hash e per fornire maggiore sicurezza a P2WSH (128 bit di sicurezza in P2WSH contro 80 bit di sicurezza in P2SH).

La società di Mohammed può spendere gli output di P2WSH presentando lo script di riscatto corretto e le firme sufficienti per soddisfarlo. Sia lo script di riscatto che le firme sarebbero segregati al di fuori della

transazione spendibile come parte dei dati dei testimoni. Nell'input della transazione, il portafoglio di Mohammed avrebbe messo uno script vuoto.

Transazione decodificata che mostra un output P2WSH spendibile con dati di witness separati

```
[...]
“Vin” : [
"txid": "abcdef12345...",
"vout": 0,
    "scriptSig": “”,
]
[...]
“witness”: “<SigA> <SigB> <2 PubA
PubB PubC PubD PubE 5
```

```
CHECKMULTISIG>”
[...]
```

Differenziazione tra P2WPKH e P2WSH

Nelle due sezioni precedenti, abbiamo dimostrato due tipi di programmi witness: Pay-to-Witness-Public-Key-Hash (P2WPKH) e Pay-to-Witness-Script-Hash (P2WSH). Entrambi i tipi di programmi witness sono costituiti da un numero di versione a byte singolo seguito da un hash più lungo. Sembrano molto simili, ma sono interpretati in modo molto diverso: uno è interpretato come un hash della chiave pubblica, che è soddisfatto da una firma e l'altro come uno script

hash, che è soddisfatto da uno script di riscatto. La differenza fondamentale tra essi è la lunghezza dell'hash:

- L'hash della chiave pubblica in P2WPKH è di 20 byte
- L'hash dello script in P2WSH è di 32 byte

Questa è l'unica differenza che consente ad un portafoglio di distinguere i due tipi di programmi witness. Osservando la lunghezza dell'hash, un portafoglio può determinare quale tipo di programma witness è, P2WPKH o P2WSH.

Aggiornamento a Segregated

Witness

Come possiamo vedere dagli esempi precedenti, l'aggiornamento a Segregated Witness è un processo diviso in due fasi. Innanzitutto, i portafogli devono creare speciali output di tipo segwit. Quindi, questi output possono essere spesi dai portafogli che sanno come costruire le transazioni Segregated Witness. Negli esempi, il portafoglio di Alice era a conoscenza di segwit e in grado di creare output speciali con script Segregated Witness. Il portafoglio di Bob è anch'esso consapevole di segwit e in grado di spendere tali output. Ciò che potrebbe non essere ovvio dall'esempio è che, in pratica, il

portafoglio di Alice necessita di sapere che Bob usa un portafoglio a conoscenza di segwit e può spendere tali output. In caso contrario, se il portafoglio di Bob non viene aggiornato e Alice tenta di effettuare pagamenti segwit a Bob, il portafoglio di Bob non sarà in grado di riconoscere questi pagamenti.

TIP	Per i pagamenti P2WPKH e P2WSH, sia il wallet del mittente che quello del destinatario devono essere aggiornati per poter utilizzare segwit. Inoltre, il portafoglio del mittente deve sapere che il

	portafoglio del destinatario è consapevole di segwit.

Segregated Witness non è stato implementato simultaneamente sull'intera rete. Piuttosto, Segregated Witness è implementato come un aggiornamento compatibile con le versioni precedenti, in cui *vecchi e nuovi client possono coesistere*. Gli sviluppatori di wallet aggiorneranno indipendentemente il software wallet per aggiungere funzionalità segwit. I tipi di pagamento P2WPKH e P2WSH vengono utilizzati quando sia il mittente che il destinatario sono

consapevoli di segwit. I tradizionali P2PKH e P2SH continueranno a funzionare per i portafogli non aggiornati. Ciò lascia due importanti scenari, che sono trattati nella prossima sezione:

- Abilità del wallet di un mittente che non è consapevole di segwit per effettuare un pagamento al wallet di un destinatario che può elaborare transazioni segwit
- Abilità del wallet di un mittente che è consapevole di segwit di riconoscere e distinguere tra i destinatari che sono consapevoli di

segwit e quelli che non lo sono, dai loro *indirizzi*.

Incorporare Segregated Witness all'interno di P2SH

Supponiamo, ad esempio, che il portafoglio di Alice non venga aggiornato a segwit, ma il portafoglio di Bob viene aggiornato e può gestire le transazioni segwit. Alice e Bob possono utilizzare transazioni "vecchie" non-segwit. Ma Bob vorrebbe probabilmente usare segwit per ridurre le commissioni di transazione, approfittando dello sconto che si applica ai dati dei testimoni.

In questo caso il portafoglio di Bob

può costruire un indirizzo P2SH che contiene uno script segwit al suo interno. Il portafoglio di Alice vede questo come un "normale" indirizzo P2SH e può effettuare pagamenti senza alcuna conoscenza di segwit. Il portafoglio di Bob può quindi spendere questo pagamento con una transazione segwit, sfruttando appieno il segwit e riducendo le spese di transazione.

Entrambe le forme di script witness, P2WPKH e P2WSH, possono essere incorporate in un indirizzo P2SH. Il primo è indicato come P2SH (P2WPKH) e il secondo è indicato come P2SH (P2WSH).

Pay-to-Witness-Public-Key-Hash dentro Pay-to-Script-Hash

La prima forma di script witness che esamineremo è P2SH (P2WPKH). Questo è un programma di test di Hash Key-to-Witness-Public-Key, incorporato in uno script Pay-to-Script-Hash, in modo che possa essere utilizzato da un portafoglio che non è a conoscenza di segwit.

Il wallet di Bob costruisce un programma witness P2WPKH con la chiave pubblica di Bob. Questo programma di witness viene quindi sottoposto a hashing e l'hash risultante viene codificato come uno script

P2SH. Lo script P2SH viene convertito in un indirizzo bitcoin che inizia con un "3", come abbiamo visto nella sezione Pay-to-Script-Hash (P2SH).

Il portafoglio di Bob inizia con il programma witness P2WPKH che abbiamo visto in precedenza:

Bob's P2WPKH witness program

```
0
ab68025513c3dbd2f7b92a94e0581f5d50f65
```

Il programma witness P2WPKH è costituito dalla versione di witness e dall'hash della chiave pubblica da 20 byte di Bob.

Il portafoglio di Bob quindi esegue

l'hashing del programma precedente, prima con SHA256, poi con RIPEMD160, producendo un altro hash da 20 byte.

Usiamo bx sulla riga di comando per replicare ciò:

HASH160 of the P2WPKH witness program

```
echo \
'0
[ab68025513c3dbd2f7b92a94e0581f5d50f6
  | bx script-encode | bx sha256 | bx
ripemd160
3e0547268b3b19288b3adef9719ec8659f4b
```

Successivamente, l'hash dello script di riscatto viene convertito in un indirizzo bitcoin. Usiamo di nuovo bx

sulla riga di comando:

P2SH address

```
echo \
'3e0547268b3b19288b3adef9719ec8659f4b
\
| bx address-encode -v 5
37Lx99uaGn5avKBxiW26HjedQE3LrDCZru
```

Ora Bob può mostrare questo indirizzo per dare la possibilità ai propri clienti di pagare per il loro caffè. Il portafoglio di Alice può effettuare un pagamento a 37Lx99uaGn5avKBxiW26HjedQE3LrD proprio come farebbe con qualsiasi altro indirizzo bitcoin.

Per pagare Bob, il portafoglio di Alice bloccherebbe l'output con uno script P2SH:

```
HASH160
3e0547268b3b19288b3adef9719ec8659f4b
EQUAL
```

Anche se il portafoglio di Alice non supporta segwit, il pagamento creato può essere speso da Bob con una transazione segwit.

Pay-to-Witness-Script-Hash dentro Pay-to-Script-Hash

Allo stesso modo, un programma witness P2WSH per uno script multisig o un altro script complicato può essere incorporato all'interno di uno script P2SH e di un indirizzo, dando la possibilità a qualsiasi portafoglio di realizzare pagamenti compatibili con segwit.

Come abbiamo visto in in Pay-to-Witness-Script-Hash (P2WSH), la compagnia di Mohammed utilizza pagamenti Segregated Witness per gli script multisignature. Per consentire a qualsiasi cliente di pagare la sua azienda, indipendentemente dal fatto che i suoi portafogli siano aggiornati a segwit, il portafoglio di Mohammed può incorporare il programma witness P2WSH all'interno di uno script P2SH.

Innanzitutto, il portafoglio di Mohammed blocca lo script di riscatto con SHA256 (solo una volta). Usiamo bx per far ciò sulla riga di comando:

Il portafoglio di Mohammed crea un programma witness

P2WSH

```
echo \
2 \
[04C16B8698A9ABF84250A7C3EA7EEDE
\
[04A2192968D8655D6A935BEAF2CA23E
\
[047E63248B75DB7379BE9CDA8CE5751
\
[0421D65CBD7149B255382ED7F78E9465
\
[043752580AFA1ECED3C68D446BCAB69
\
5 CHECKMULTISIG \
| bx script-encode | bx sha256
9592d601848d04b172905e0ddb0adde59f15
```

Successivamente, lo script di riscatto viene trasformato in un programma witness P2WSH:

```
0
9592d601848d04b172905e0ddb0adde59f15
```

Quindi, il programma witness stesso viene sottoposto ad hashing con SHA256 e RIPEMD160, producendo un nuovo hash da 20 byte, come usato nel P2SH tradizionale. Usiamo bx sulla riga di comando per far ciò:

L'HASH160 del programma witness P2WSH

```
echo \
'0
[9592d601848d04b172905e0ddb0adde59f1
| bx script-encode | bx sha256 | bx
ripemd160
86762607e8fe87c0c37740cddee880988b94
```

Successivamente, il wallet crea un indirizzo bitcoin P2SH da questo

stesso hash. Di nuovo, usiamo bx per calcolarlo sulla riga di comando:

Indirizzo bitcoin P2SH

```
echo \
'86762607e8fe87c0c37740cddee880988b9
 | bx address-encode -v 5
3Dwz1MXhM6EfFoJChHCxh1jWHb8GQqR
```

Ora, i clienti di Mohammed possono effettuare pagamenti a questo indirizzo senza bisogno di supporto segwit. Per inviare un pagamento a Mohammed, un portafoglio bloccherebbe l'output con il seguente script P2SH:

Script P2SH utilizzato per bloccare i pagamenti al multisig di Mohammed

```
HASH160
```

```
86762607e8fe87c0c37740cddee880988b94
EQUAL
```

La società di Mohammed può quindi costruire transazioni segwit per spendere questi pagamenti, sfruttando le funzionalità di segwit come le commissioni di transazione più basse.

Indirizzi Segregated Witness

Anche dopo l'attivazione segwit, ci vorrà del tempo prima che la maggior parte dei wallet vengano aggiornati. Inizialmente, segwit sarà incorporato in P2SH, come abbiamo visto nella sezione precedente, per facilitare la compatibilità tra i portafogli a conoscenza o meno di segwit.

Tuttavia, una volta che i wallet

supporteranno ampiamente segwit, ha senso codificare gli script witness direttamente in un formato di indirizzo nativo progettato per segwit, piuttosto che incorporarlo in P2SH.

Il formato dell'indirizzo segwit nativo è definito in BIP-173:

BIP-173

> Formato di indirizzo Base32 per output nativi witness v0-16

BIP-173 codifica solo witness (P2WPKH e P2WSH). Non è compatibile con script P2PKH o P2SH non-segwit. BIP-173 è una codifica Base32 con checksum, rispetto alla codifica Base58 di un indirizzo bitcoin "tradizionale". Gli addict BIP-173

sono anche chiamati indirizzi bech32, pronunciati "beh-ch trentadue", alludendo all'uso di un algoritmo di rilevamento degli errori "BCH" e di un set di codifica di 32 caratteri.

Gli indirizzi BIP-173 utilizzano un set di caratteri alfanumerici di 32 lettere minuscole, accuratamente selezionati per ridurre gli errori di lettura e/o scrittura. Scegliendo un set di caratteri minuscoli, bech32 è più facile da leggere e comunicare ed è il 45% più efficiente in fase di codifica nei codici QR.

L'algoritmo di rilevamento degli errori BCH è un enorme miglioramento rispetto all'algoritmo del checksum precedente (da Base58Check), che

consente non solo il rilevamento ma anche la *correzione* degli errori. Le interfacce di input degli indirizzi (come i campi di testo nei moduli) sono in grado di rilevare ed evidenziare quale carattere è stato più probabilmente digitato erroneamente quando viene rilevato un errore.

Dalle specifiche BIP-173, ecco alcuni esempi di indirizzi bech32:

Mainnet P2WPKH

bc1qw508d6qejxtdg4y5r3zarvary0c5x

Testnet P2WPKH

tb1qw508d6qejxtdg4y5r3zarvary0c5xv

Mainnet P2WSH

bc1qrp33g0q5c5txsp9arysrx4k6zdkfs4

Testnet P2WSH

tb1qrp33g0q5c5txsp9arysrx4k6zdkfs4r

Come puoi vedere in questi esempi, una stringa segwit bech32 ha una lunghezza massima di 90 caratteri e si compone di tre parti:

La parte leggibile dall'uomo

Questo prefisso "bc" o "tb" identifica mainnet o testnet.

Il separatore

La cifra "1", che non fa parte del set di codifica a 32 caratteri e può apparire solo in questa posizione come separatore

La parte dei dati

Un minimo di 6 caratteri alfanumerici, lo script testimone

codificato per il checksum

In questo momento, solo pochi portafogli accettano o producono indirizzi segwit bech32 nativi, ma con l'aumentare dell'adozione segwit, saranno sempre più presenti.

Identificativi di transazione

Uno dei maggiori vantaggi di Segregated Witness è che elimina la malleabilità delle transazioni da terze parti.

Prima di segwit, le transazioni potevano avere le firme relative leggermente modificate da terze parti, modificando l'ID di transazione (hash) senza modificare le proprietà fondamentali (input, output, importi).

Ciò ha creato opportunità per attacchi denial-of-service e attacchi contro wallet il cui codice era scritto male, che presupponevano che gli hash delle transazioni non confermate fossero immutabili.

Con l'introduzione di Segregated Witness, le transazioni hanno due identificatori, txid e wtxid. L'ID transazione tradizionale txid è doppio hash SHA256 della transazione serializzata, senza i dati del testimone. Una transazione wtxid è il doppio hash SHA256 del nuovo formato di serializzazione della transazione con i dati dei testimoni.

Il txid tradizionale viene calcolato esattamente nello stesso modo di una

transazione non-segwit. Tuttavia, poiché la transazione segwit ha scriptSig vuoti in ogni input, non c'è parte della transazione che può essere modificata da una terza parte. Pertanto, in una transazione segwit, il txid è immutabile da una terza parte, anche quando la transazione non è confermata.

Il wtxid è una sorta di ID "esteso", in quanto l'hash incorpora anche i dati dei testimoni. Se una transazione viene trasmessa senza dati di testimone, wtxid e txid sono identici. Si noti che poiché wtxid include i dati dei testimoni (firme) e poiché i dati dei testimoni possono essere malleabili, deve essere considerato malleabile

fino a quando la transazione non viene confermata. Solo il txid di una transazione segwit può essere considerato immutabile da terze parti e solo se *tutti* gli input della transazione sono input segwit.

TIP	Le transazioni Segregated Witness hanno due ID: txid e wtxid. Il txid è l'hash della transazione senza i dati del testimone e wtxid è l'hash comprensivo dei dati dei testimoni. Il txid di una transazione in cui tutti gli input sono input segwit non è suscettibile

	alla malleabilità delle transazioni da terze parti.

Nuovo algoritmo di firma di Segregated Witness

Segregated Witness modifica la semantica delle quattro funzioni di verifica della firma (CHECKSIG, CHECKSIGVERIFY, CHECKMULTISIG e CHECKMULTISIGVERIFY), cambiando il modo in cui viene calcolato il commitment hash della transazione.

Le firme nelle transazioni bitcoin vengono applicate a un *commitment hash*, che viene calcolato dai dati della transazione, bloccando parti specifiche dei dati che indicano l'impegno del firmatario per tali valori. Ad esempio, in una semplice firma di tipo SIGHASH_ALL, l'hash di impegno include tutti gli input e gli output.

Sfortunatamente, il modo in cui è stato calcolato il commitment hash ha introdotto la possibilità che un nodo che verifica le firme possa essere costretto a eseguire un numero significativo di calcoli computazionali. Nello specifico, le operazioni di hash aumentano in $O(n2)$

rispetto al numero di operazioni di firma nella transazione. Un utente malintenzionato potrebbe quindi creare una transazione con un numero molto elevato di operazioni di firma, costringendo all'intera rete bitcoin di eseguire centinaia o migliaia di operazioni hash per verificare tale transazione.

Segwit ha rappresentato un'opportunità per affrontare questo problema cambiando il modo in cui viene calcolato l'hash degli impegni. Per i programmi di segwit versione 0, la verifica della firma avviene utilizzando un algoritmo hash di impegno migliorato come specificato in BIP-143.

Il nuovo algoritmo raggiunge due obiettivi importanti. Innanzitutto, il numero di operazioni hash aumenta di un numero (O) nettamente più graduale rispetto al numero di operazioni di firma, riducendo l'opportunità di creare attacchi di tipo denial-of-service con transazioni eccessivamente complesse. In secondo luogo, l'hash dell'impegno ora include anche il valore (importi) di ciascun input come parte dell'impegno. Ciò significa che un firmatario può eseguire il commit su un valore di input specifico senza la necessità di "recuperare" e controllare la transazione precedente a cui fa riferimento l'input. Nel caso di

dispositivi offline, come i portafogli hardware, ciò semplifica enormemente la comunicazione tra l'host e il portafoglio hardware, eliminando la necessità di eseguire lo streaming di transazioni precedenti per la convalida. Un portafoglio hardware può accettare il valore di input "come dichiarato" da un host non attendibile. Poiché la firma non è valida se tale valore di input non è corretto, il portafoglio hardware non deve convalidare il valore prima di firmare l'input.

Incentivi Economici per Segregated Witness

I mining node e i full node di Bitcoin

comportano una serie costi viste le risorse utilizzate per supportare la rete bitcoin e mantenere la blockchain. Con l'aumento del volume delle transazioni bitcoin, aumenta anche il costo delle risorse (CPU, larghezza di banda della rete, spazio su disco, memoria). I miner sono compensati attraverso commissioni proporzionali alla dimensione (in byte) di ciascuna transazione. I nodi completi non-mining non vengono compensati, quindi incorrono in questi costi senza avere un ritorno economico; solitamente questi nodi vengono messi in funzione quando si ha la necessità di gestire un business senza l'ausilio di prodotti o servizi terzi.

Senza commissioni di transazione, la crescita dei dati generati da bitcoin aumenterebbe considerevolmente. Le tariffe hanno lo scopo di allineare le esigenze degli utenti bitcoin con l'onere che le loro transazioni impongono alla rete, attraverso un meccanismo di rilevazione dei prezzi basato sul mercato.

Il calcolo delle commissioni in base alla dimensione della transazione considera tutti i dati nella transazione come uguali in termini di costi. Ma dal punto di vista di nodi e miner, alcune parti di una transazione comportano costi molto più elevati rispetto ad altri. Ogni transazione aggiunta alla rete bitcoin influisce sul consumo di

quattro risorse a carico dei nodi:

Spazio sul disco

Ogni transazione è memorizzata nella blockchain, che col passare del tempo aumenta di dimensione. La blockchain è archiviata su disco e quindi necessita di sempre più spazio con l'aumentare delle transazioni eseguite; l'archiviazione può comunque essere ottimizzata effettuando il "pruning" delle transazioni precedenti.

CPU

Ogni transazione deve essere convalidata, il che richiede CPU.

Larghezza di banda

Ogni transazione viene trasmessa

(tramite propagazione flood) attraverso la rete almeno una volta. Senza alcuna ottimizzazione nel protocollo di propagazione dei blocchi, le transazioni vengono nuovamente trasmesse come parte di un blocco, raddoppiando l'impatto sulla capacità della rete.

Memoria

I nodi che convalidano le transazioni mantengono l'indice UTXO o l'intero set UTXO in memoria per accelerare la convalida. Poiché la memoria è almeno di un ordine di grandezza più costosa del disco, la crescita del set UTXO contribuisce in modo sproporzionato al costo di esecuzione di un nodo.

Come puoi vedere dall'elenco, non tutte le parti di una transazione hanno lo stesso impatto sul costo di esecuzione di un nodo o sulla capacità di bitcoin di scalare per supportare più transazioni. La parte più costosa di una transazione sono gli output appena creati, in quanto vengono aggiunti al set UTXO in memoria. In confronto, le firme (ovvero i dati dei testimoni) aggiungono il minimo onere alla rete e il costo di esecuzione di un nodo, poiché i dati dei testimoni vengono convalidati una sola volta e non vengono più utilizzati. Inoltre, subito dopo aver ricevuto una nuova transazione e convalidato i dati dei testimoni, i nodi possono scartare i

dati dei testimoni. Se le commissioni sono calcolate sulla dimensione della transazione, senza discriminare tra questi due tipi di dati, gli incentivi di mercato delle commissioni non sono allineati con i costi effettivi imposti da una transazione. In effetti, l'attuale struttura delle commissioni incoraggia effettivamente il comportamento contrario, perché i dati dei testimoni rappresentano la parte più grande di una transazione.

Gli incentivi creati dalle commissioni sono importanti perché influenzano il comportamento dei wallet. Tutti i portafogli devono implementare una strategia per la creazione di transazioni che tenga in considerazione

una serie di fattori, come la privacy (riduzione del riutilizzo degli indirizzi), la frammentazione (spiccioli per il “resto”) e le commissioni. Se le commissioni motivano in modo schiacciante l’utilizzo del minor numero possibile di input nelle transazioni da parte dei wallet, ciò può portare al prelievo dell'UTXO e alla modifica delle strategie di indirizzi che inavvertitamente gonfiano il set UTXO.

Le transazioni consumano UTXO nei loro input e creano nuovo UTXO con i loro output. Una transazione, quindi, che ha più input che output, si tradurrà in una diminuzione del set UTXO, mentre una transazione che ha più

output che input determinerà un aumento del set UTXO. Consideriamo l a *differenza* tra input e output e chiamiamola "Net-new-UTXO". Questa è una metrica importante, in quanto ci dice quale impatto avrà una transazione sulla più costosa risorsa di rete, il set UTXO in memoria. onere. Al contrario, una transazione con un Net-new-UTXO negativo riduce il carico. Vorremmo quindi incoraggiare le transazioni che sono negative Net-new-UTXO o neutrali con zero Net-new-UTXO.

Esaminiamo un esempio di quali incentivi vengono creati dal calcolo della commissione di transazione, con e senza Segregated Witness. Daremo

un'occhiata a due diverse transazioni. La transazione A è una transazione a 3 input, 2 output, che ha una metrica Net-new-UTXO di -1, il che significa che consuma un UTXO in più di quello che crea, riducendo il set UTXO di uno. La transazione B è una transazione a 2 input e 3 output, che ha una metrica Net-new-UTXO pari a 1, il che significa che aggiunge un UTXO al set UTXO, imponendo un costo aggiuntivo sull'intera rete bitcoin. Entrambe le transazioni utilizzano script multisignature (2-di-3) per dimostrare come script complessi aumentano l'impatto di Segregated Witness sulle commissioni. Supponiamo una commissione di 30 satoshi per byte e

uno sconto del 75% sulle commissioni dei dati dei testimoni:

Senza Segregated Witness

Commissione Transazione A: 25,710 satoshi

Commissione Transazione B: 18,990 satoshi

Con Segregated Witness

Commissione Transazione A: 8,130 satoshi

Commissione Transazione B: 12,045 satoshi

Entrambe le transazioni sono meno costose quando viene implementato Segregated Witness. Ma confrontando i costi tra le due transazioni, vediamo che prima di Segregated Witness, la

commissione è più alta per la transazione che ha un Net-new-UTXO negativo. Dopo Segregated Witness, le commissioni di transazione si allineano con l'incentivo a minimizzare la nuova creazione di UTXO non penalizzando inavvertitamente le transazioni con molti input.

Segregated Witness ha quindi due effetti principali sulle commissioni pagate dagli utenti bitcoin. In primo luogo, segwit riduce il costo complessivo delle transazioni scontando i dati dei testimoni e aumentando la capacità della blockchain di bitcoin. In secondo luogo, lo sconto di segwit sui dati dei testimoni corregge un disallineamento

degli incentivi che potrebbe gonfiare inavvertitamente il set UTXO.

La Rete Bitcoin

Architettura di Rete Peer-to-Peer

Bitcoin è strutturato come un'architettura peer-to-peer sopra internet. Il termine peer-to-peer, o P2P, sta a significare che i computer che partecipano alla rete sono (nodi n.d.t.) al pari degli altri, e sono tutti uguali, che non ci sono nodi "speciali", e che tutti i nodi condividono il compito di fornire servizi alla rete. I nodi della rete (network) sono connessi tra di loro in una rete mesh (a maglia n.d.t.) con una topologia

"piatta". Non c'è un server, nessun servizio centralizzato, e nessuna gerarchia di alcun tipo nella rete. I nodi in una rete peer-to-peer provvedono a fornire e utilizzano i servizi allo stesso tempo con reciprocità agendo come incentivo per la partecipazione. Le reti peer-to-peer sono resilienti per definizione, decentralizzate, e aperte. Il preminente esempio di un'architettura di una rete P2P si è verificato con lo stesso internet quando era ai suoi primordi, dove i nodi della rete IP erano uguali. L'architettura attuale di internet è più gerarchica, ma l'Internet Protocol mantiene ancora la sua topologia-piatta che ne è l'essenza. Dopo

bitcoin, l'applicazione più grande e con più successo delle tecnologie P2P è la condivisione dei file con Napster come suo pioniere e BitTorrent come l'evoluzione più recente dell'architettura.

L'architettura P2P di Bitcoin è molto più di una scelta tecnologica. Bitcoin è stato progettato per essere un sistema peer-to-peer di contante digitale, e l'architettura del network, è sia un riflesso che le fondamenta di questa caratteristica principale. La decentralizzazione del controllo è una delle scelte architetturali principali e può essere ottenuta solamente mantenendo una rete P2P di consenso piatta (tutti i nodi hanno gli stessi

privilegi) e decentralizzata.

Il termine "network bitcoin" si riferisce alla serie di nodi che eseguono il protocollo P2P bitcoin. Oltre al protocollo P2P di bitcoin, ci sono altri protocolli come quello di Stratum, che sono utilizzati per il mining e per wallet bitcoin mobile o leggeri. Questi protocolli addizionali sono forniti da server gateways di routing che hanno accesso alla rete bitcoin usando il protocollo P2P bitcoin, e estendono la rete a nodi che eseguono altri protocolli. Per esempio, i server Stratum si connettono ai nodi che effettuano mining attraverso il protocollo Stratum del network bitcoin principale e collegano il protocollo

Stratum al protocollo bitcoin P2P. Usiamo il termine "network di bitcoin esteso" (extended bitcoin network) per riferirsi a tutto il network generale che include il protocollo P2P bitcoin, i protocolli di pool-mining, il protocollo Stratum, e altri protocolli correlati che connettono i componenti al sistema bitcoin.

Tipi di Nodi e Ruoli

Anche se i nodi della rete P2P bitcoin sono uguali, possono avere differenti ruoli a seconda della funzionalità che stanno supportando. Un nodo bitcoin è una serie di funzioni: routing, database della blockchain, mining, e servizi wallet. Un nodo completo (full node)

con tutte queste quattro caratteristiche è mostrato in [Un nodo della rete bitcoin con tutte e quattro le funzioni: wallet, miner, database blockchain completo, e network routing](). .

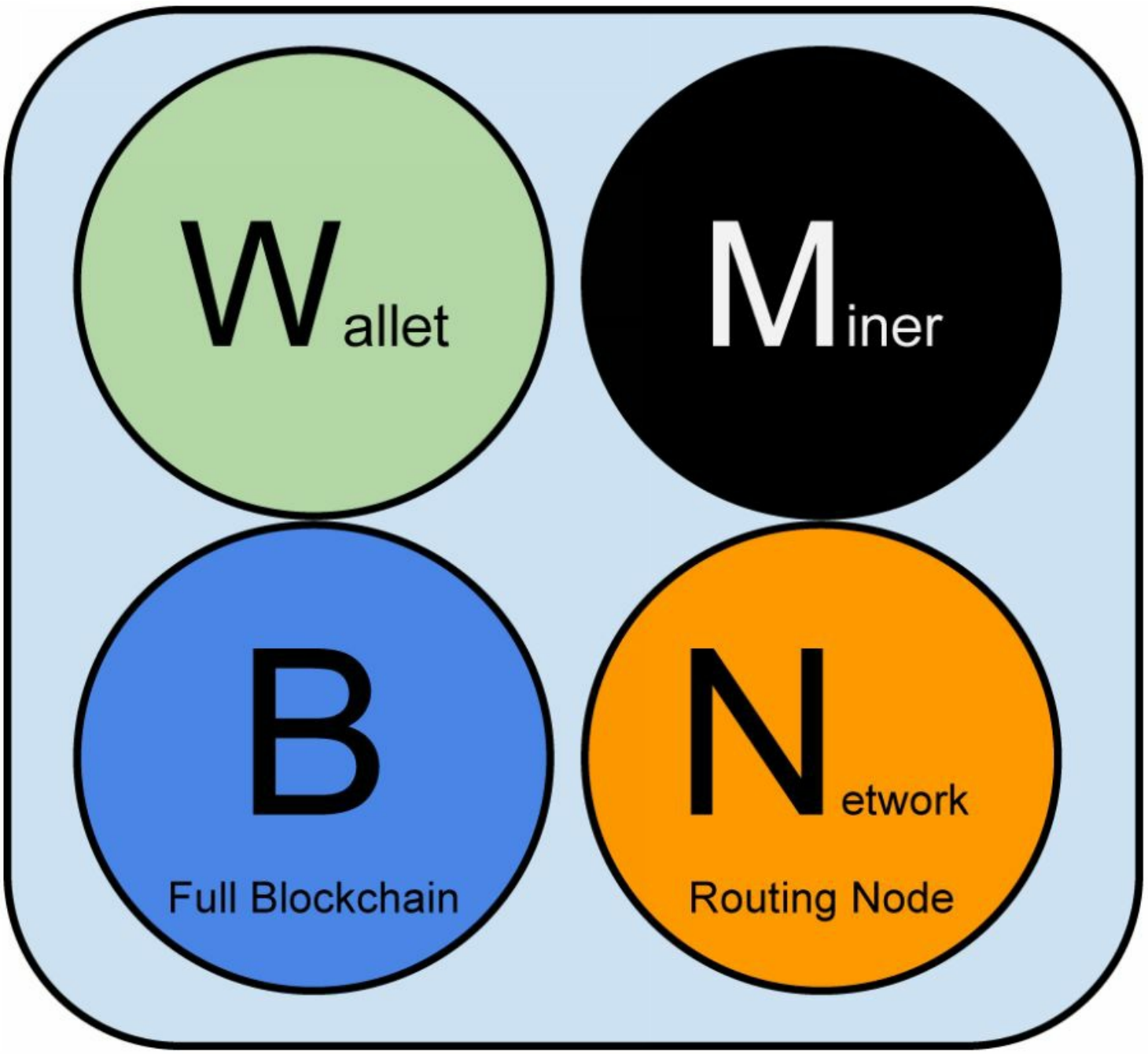


Figura 45. Un nodo della rete bitcoin con tutte e quattro le funzioni: wallet, miner, database blockchain completo, e network routing

Tutti i nodi hanno la funzionalità di routing per partecipare al network ma possono anche avere anche altre funzionalità. Tutti i nodi convalidano e propagano le transazioni ed i blocchi, inoltre individuano e mantengono collegamenti peer-to-peer con altri nodi. Nell'esempio del full-node in Un nodo della rete bitcoin con tutte e quattro le funzioni: wallet, miner, database blockchain completo, e

network routing, la funzione di routing è indicata da un cerchio "N" denominato "Network Routing Node".

Alcuni nodi, detti full-node (nodi completi), mantengono anche una completa e aggiornata copia della blockchain. I full-node possono verificare autonomamente e autoritativamente ogni transazione senza bisogno di riferimenti esterni. Alcuni nodi mantengono solo un sottoinsieme della blockchain e verificano le transazioni utilizzando un metodo chiamato *simplified payment verification*, o SPV (verifica semplificata del pagamento). Questi nodi sono noti come SPV o nodi lightweight (light, leggeri).

Nell'esempio del full-node mostrato nell'immagine, la funzione di database blockchain del full-node è indicata da un "B" chiamato "Full Blockchain." In La rete estesa di bitcoin che mostra vari tipi di nodi, gateway, e protocolli, i nodi SPV sono disegnati senza il "B", per mettere in evidenza che non hanno una copia completa della blockchain.

I nodi addetti al "mining" competono per creare nuovi blocchi utilizzando hardware specializzato per risolvere l'algoritmo di proof-of-work. Alcuni nodi di mining sono anche nodi completi e mantengono una copia completa della blockchain, mentre altri sono nodi leggeri che partecipano ad

un gruppo di mining e si basano su un server di mining per mantenere un nodo completo. La funzione di mining è indicata nel nodo completo come un cerchio "M" denominato "Miner."

I portafogli utente possono far parte di un nodo completo, come spesso è il caso per i client bitcoin desktop. Ma sempre più di frequente, molti portafogli utente, specialmente quelli dedicati a dispositivi con risorse limitate, sono nodi di tipo SPV. Il portafoglio è indicato nel Un nodo della rete bitcoin con tutte e quattro le funzioni: wallet, miner, database blockchain completo, e network routing come un cerchio verde denominato "Wallet" o con la lettera

"W".

In aggiunta ai tipi di nodo principali del protocollo bitcoin P2P, ci sono server e nodi che eseguono altri protocolli, come protocolli di mining pool specializzati e protocolli di accesso dei light client per accedere più efficacemente ad essi.

Diversi tipi di nodi della rete estesa di bitcoin mostra il tipi di nodi più comuni sulla rete bitcoin estesa.

Il Network Bitcoin Esteso

Il principale network bitcoin, l'esecuzione del protocollo bitcoin P2P, consiste in 8000 - 10000 nodi in

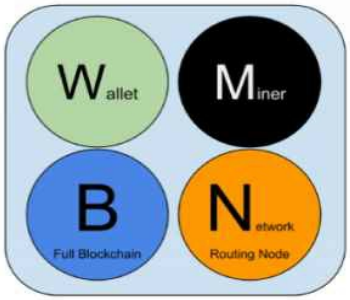
ascolto che eseguono le varie versioni del client di riferimento bitcoin (Bitcoin Core) e qualche centinaio di nodi che eseguono varie altre implementazioni del protocollo Bitcoin P2P, come ad esempio BitcoinJ, Libbitcoin, e btcd. Una piccola percentuale dei nodi della rete P2P Bitcoin sono anche nodi del mining, quindi, li ritroviamo nel processo di estrazione, nel convalidare le transazioni e nel creare nuovi blocchi. Varie grandi compagnie si interfacciano con il network bitcoin eseguendo i client full-node basati sul client di Bitcoin Core, con copie piene della blockchain e di un nodo del network, ma senza mining o funzioni

da wallet. Questi nodi agiscono come un network edge router, consentendo a vari altri servizi (exchange, wallet, block explorer, merchant payment processing) di essere implementati su sulla rete stessa.

L'estensione del network bitcoin include il network di esecuzione del protocollo bitcoin P2P, descritto facilmente, così come i nodi eseguono i protocolli specializzati. Un certo numero di pool server e di protocolli gateway sono collegati alla rete principale bitcoin P2P e connettono i nodi che eseguono altri protocolli. Questi altri nodi dei protocolli sono perlopiù nodi di mining pool (vedi Il Mining e Il Consenso) e wallet leggeri,

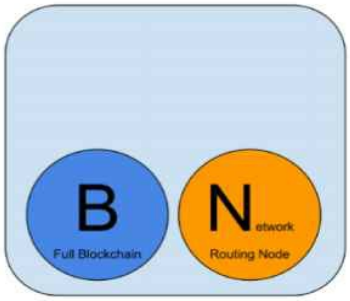
che non scaricano una copia completa della blockchain.

La rete estesa di bitcoin che mostra vari tipi di nodi, gateway, e protocolli mostra la rete estesa di bitcoin con i vari tipi di nodi, gateway server, edge router, e client wallet e i vari protocolli che essi usano per connettersi l'un l'altro.



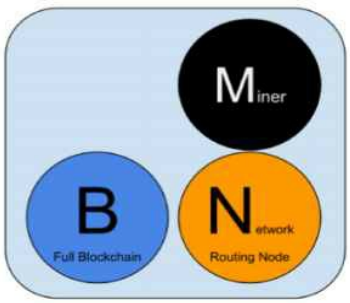
Reference Client (Bitcoin Core)

Contains a Wallet, Miner, full Blockchain database, and Network routing node on the bitcoin P2P network.



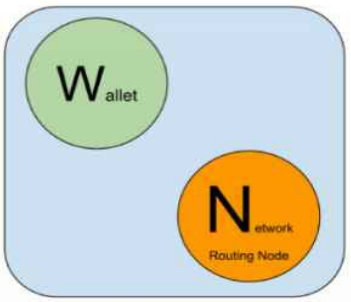
Full Block Chain Node

Contains a full Blockchain database, and Network routing node on the bitcoin P2P network.



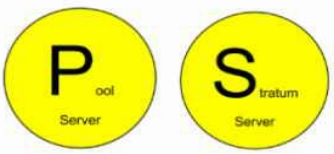
Solo Miner

Contains a mining function with a full copy of the blockchain and a bitcoin P2P network routing node.



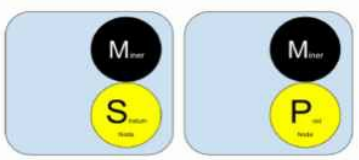
Lightweight (SPV) wallet

Contains a Wallet and a Network node on the bitcoin P2P protocol, without a blockchain.



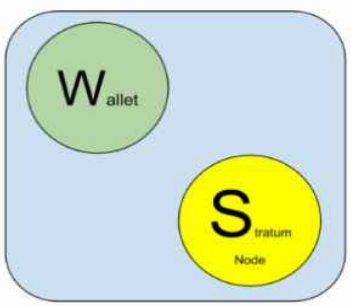
Pool Protocol Servers

Gateway routers connecting the bitcoin P2P network to nodes running other protocols such as pool mining nodes or Stratum nodes.



Mining Nodes

Contain a mining function, without a blockchain, with the Stratum protocol node (S) or other pool (P) mining protocol node.



Lightweight (SPV) Stratum wallet

Contains a Wallet and a Network node on the Stratum protocol, without a blockchain.

Figura 46. Diversi tipi di nodi della rete estesa di bitcoin

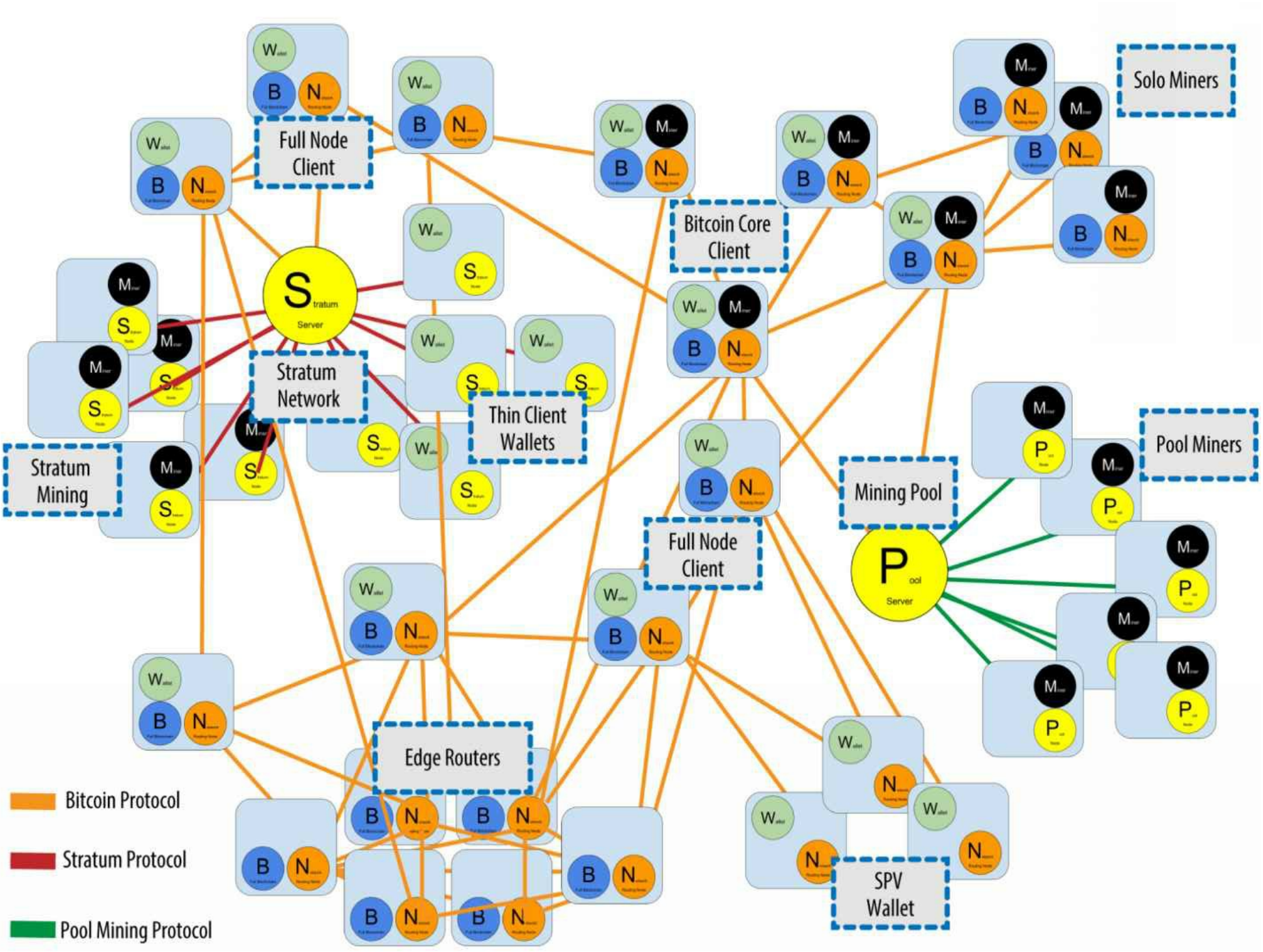

Solo Miners
Full Node Client
Bitcoin Core Client
Stratum Server
Stratum Network
Thin Client Wallets
Stratum Mining
Pool Miners
Mining Pool
Pool Server
Full Node Client
Edge Routers
SPV Wallet
Bitcoin Protocol
Stratum Protocol
Pool Mining Protocol

Figura 47. La rete estesa di bitcoin che mostra vari tipi di nodi, gateway, e protocolli

Bitcoin Relay Networks

Mentre la rete bitcoin P2P soddisfa le esigenze generali di un'ampia varietà di tipi di nodi, presenta una latenza di rete troppo elevata per le esigenze specializzate dei nodi di mining bitcoin.

I miner di Bitcoin sono impegnati in una competizione che richiede tempo per risolvere il problema del Proof-of-Work ed estendere la blockchain (vedi

Il Mining e Il Consenso). Durante la partecipazione a questa competizione, i miner bitcoin devono minimizzare il tempo tra la propagazione di un blocco vincente e l'inizio del prossimo round di competizione. Nel settore del mining, la latenza della rete è direttamente correlata ai margini di profitto.

Bitcoin Relay Network è una rete che tenta di minimizzare la latenza nella trasmissione dei blocchi tra i minatori. Il Bitcoin Relay Network originale è stato creato dal core developer Matt Corallo nel 2015 per consentire una rapida sincronizzazione dei blocchi tra i minatori con latenza molto bassa. La rete era composta da diversi nodi

specializzati ospitati nell'infrastruttura di Amazon Web Services in tutto il mondo e serviva a collegare la maggior parte dei minatori e dei pool minerari.

Il Bitcoin Relay Network originale è stato sostituito nel 2016 con l'introduzione del *Fast Internet Bitcoin Relay Engine* o *FIBRE*, anch'esso creato dal core developer Matt Corallo. FIBRE è una rete di relay basata su UDP che rilascia blocchi all'interno di una rete di nodi. FIBRE implementa l'ottimizzazione del *blocco compatto* per ridurre ulteriormente la quantità di dati trasmessi e la latenza della rete.

Un'altra rete relay (ancora in fase di

proposta) è *Falcon*, basata sulla ricerca di alcuni membri della Cornell University. Falcon utilizza "cut-through-routing" invece di "store-and-forward" per ridurre la latenza propagando parti di blocchi non appena vengono ricevuti, piuttosto che attendere il ricevimento di un blocco completo.

Le reti di inoltro vanno a sostituire la rete P2P di bitcoin. Sono invece reti sovrapposte che forniscono connettività aggiuntiva tra nodi con esigenze specializzate. Le autostrade non sono rimpiazzi per le strade rurali, ma piuttosto scorciatoie tra due punti con traffico intenso; occorrono comunque piccole strade per

collegarsi alle autostrade.

Network Discovery

Quando un nuovo nodo viene avviato, deve scoprire altri nodi bitcoin presenti sulla rete per partecipare a essa. Per iniziare questo processo, un nuovo nodo deve scoprire almeno un nodo esistente sulla rete e connettervisi. La locazione geografica degli altri nodi è irrilevante; la topologia del network bitcoin non è definita geograficamente. Perciò, ogni nodo bitcoin esistente può essere selezionato in modo casuale.

Per connettersi ad un peer conosciuto, i nodi stabiliscono una connessione TCP, solitamente dalla porta 8333 (la

porta generalmente conosciuta come l'unica usata da bitcoin) o una porta alternativa se previsto. Nel momento in cui una connessione sarà stabilita. il nodo inizierà un "handshake" (vedi L'handshake iniziale tra peers) trasmettendo un version message, che conterrà le informazioni di identificazione di base, compreso:

nVersion
: La versione del protocollo bitcoin P2P che il client "parla" (es. 70002)

nLocalServices
: Una lista di servizi locali supportati dal nodo, attualmente solo NODE_NETWORK

nTime

Il tempo attuale

addrYou

L'indirizzo IP del nodo remoto come è visto dal nodo corrente

addrMe

L'indirizzo IP del nodo locale, visto dal nodo locale

subver

Una sotto-versione che mostra il tipo di software in esecuzione su questo nodo (es. "/Satoshi:0.9.2.1/")

BestHeight

L'altezza del blocco della catena di blocchi di questo nodo

(Vedi GitHub per un esempio del messaggio di versione del network.)

Il messaggio di versione è sempre il primo messaggio inviato da qualsiasi peer a un altro peer. Il peer locale che riceve un messaggio di versione esaminerà la versione precedente del peer remoto e deciderà se il peer remoto è compatibile. Se il peer remoto è compatibile, il peer locale confermerà il messaggio di versione e stabilirà una connessione inviando un verack.

Come fa un nuovo nodo a trovare dei peers (ndt altri nodi)? Il primo metodo è è quello di query DNS utilizzando un numero di "seed DNS", che sono i server DNS che forniscono un elenco di indirizzi IP di nodi Bitcoin. Alcuni di quei seed DNS forniscono un'

elenco statico di indirizzi Ip per i nodi stabili di ascolto di bitcoin. Alcuni seeds DNS sono implementazioni personalizzate di BIND (Berkeley Internet Name Daemon) che restituiscono un sottoinsieme casuale da un elenco di indirizzi dei nodi bitcoin raccolti da un crawler o da un nodo bitcoin di lunga durata. Il client Bitcoin Core contiene i nomi dei cinque diversi seeds DNS. La diversità delle proprietà e la diversità di attuazione dei diversi semi DNS offre un livello alto o affidabilità per il processo di bootstrap iniziale. Nel client Bitcoin core, la possibilità di utilizzare i seed DNS è controllato da il parametro di opzione -dnsseed

(impostato a 1 per default ed utilizzare il seed DNS).

Alternativamente, un nodo bootstrapping che non conosce niente del network deve dare l'indirizzo IP dell'ultimo nodo bitcoin, dopo di che è possibile stabilire connessioni attraverso ulteriori presentazioni (ndt di nuovi nodi). L'argomento della riga di comando -seednode può essere utilizzato per connettersi a un nodo solo per le presentazioni, usandolo come un seed. Dopo aver usato il nodo seed per iniziare e per "determinare" un'introduzione, il client si disconnetterà e userà i peer che ha nuovamente scoperto.

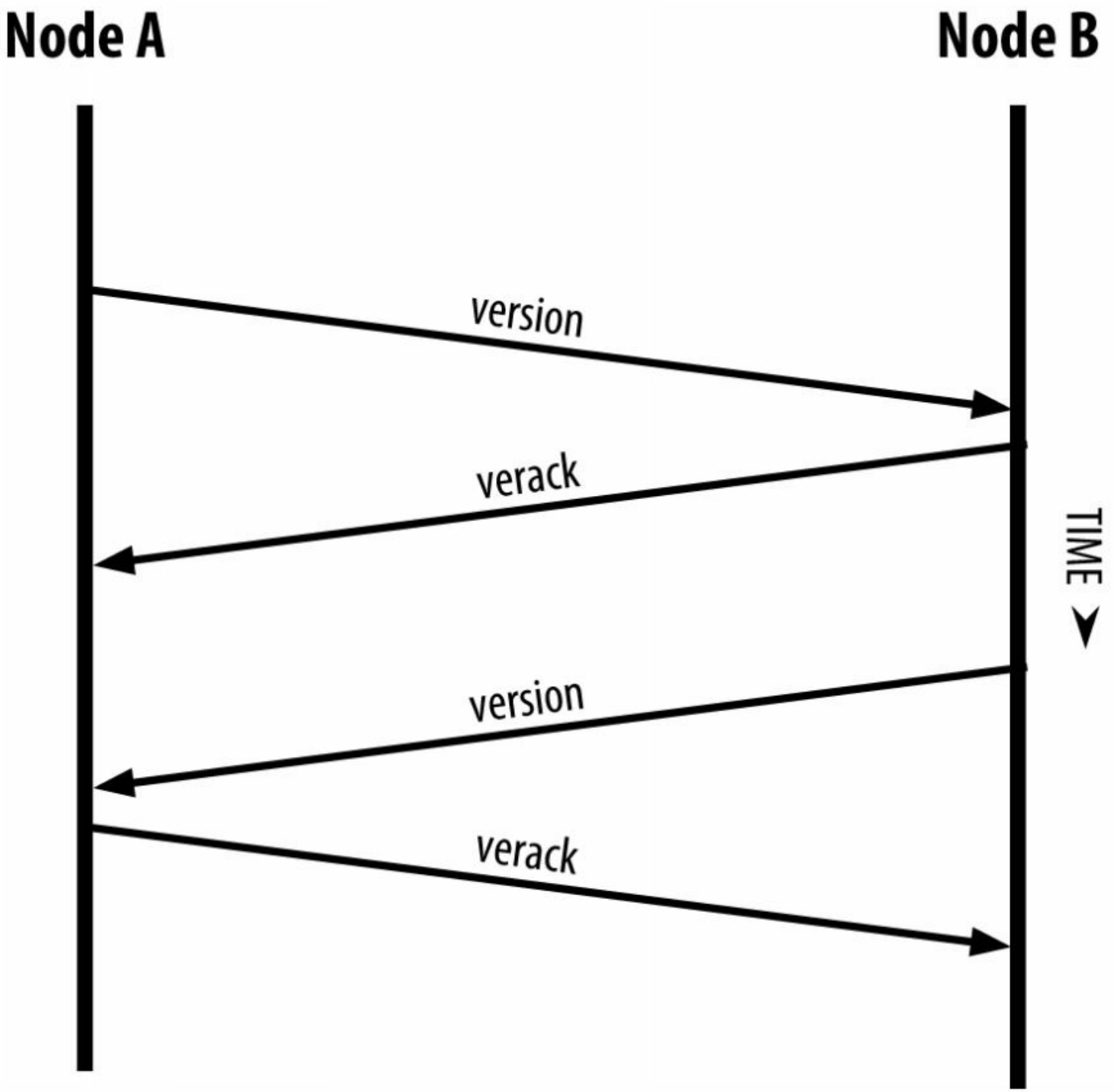


Figura 48. L'handshake iniziale tra peers

Una volta che una o più connessioni

sono stabilite, il nuovo nodo invierà un addr messaggio contenente il proprio indirizzo IP per i suoi vicini. I vicini dovranno, a loro volta, inoltrare il messaggio *addr* ai loro vicini, assicurando che il nodo appena collegato diventa ben noto e meglio collegato. Inoltre, il nodo appena collegato può inviare *getaddr* per i vicini, chiedendo loro di restituire un elenco di indirizzi IP di altri peer. In questo modo, un nodo può trovare peer per connettersi e pubblicizzare la sua esistenza sulla rete per agevolare altri nodi a trovarlo. La propagazione e la "scoperta" degli indirizzi mostra il protocollo di rilevamento dell'indirizzo.

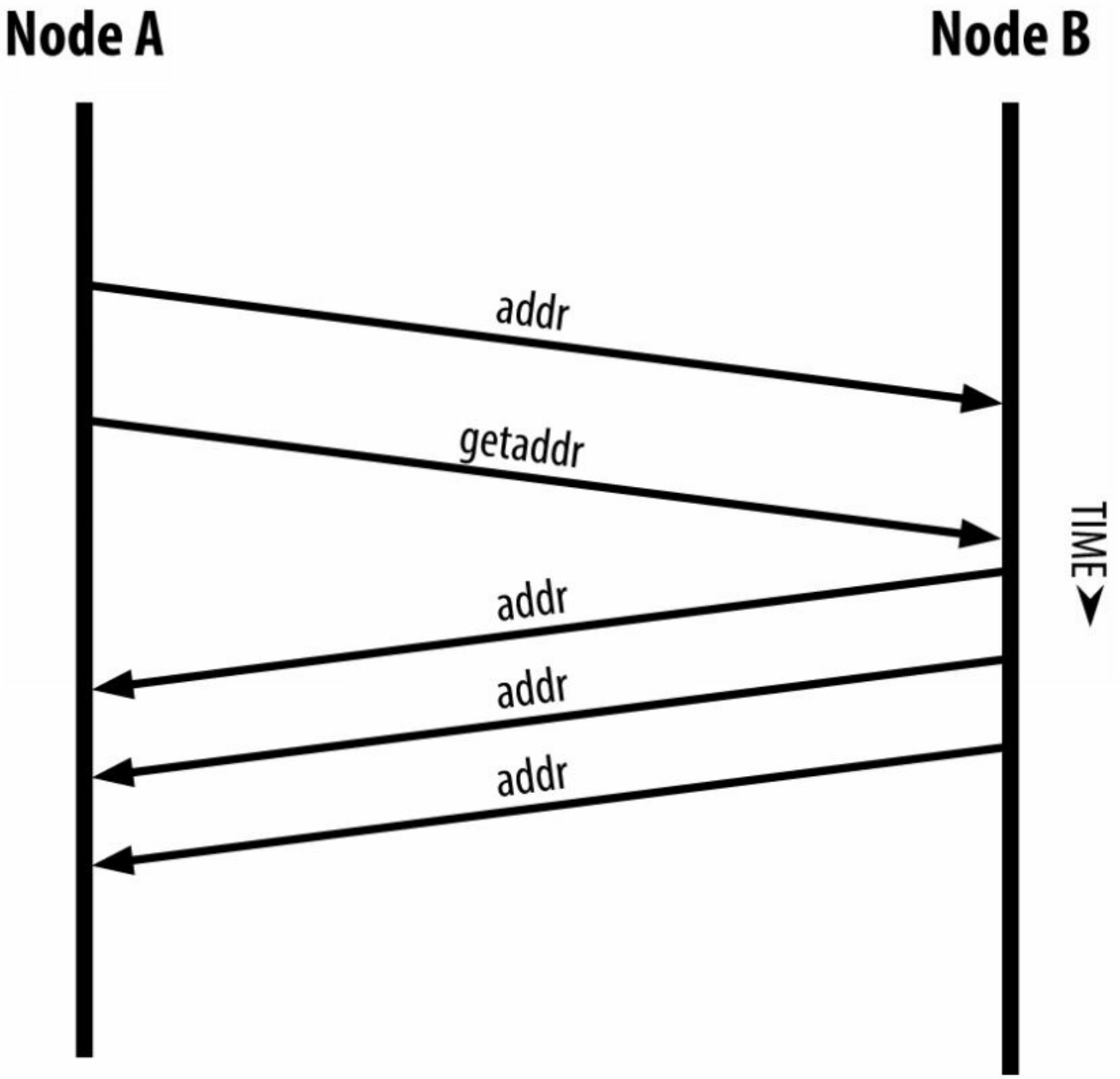


Figura 49. La propagazione e la "scoperta" degli indirizzi

Un nodo deve collegarsi a un paio di peers diversi al fine di stabilire

percorsi diversi nella rete bitcoin. I percorsi non sono affidabili - i nodi vanno e vengono - e quindi il nodo deve continuare a scoprire nuovi nodi dato che perde le vecchie connessioni e deve anche aiutare gli altri nodi quando effettuano il bootstrap. Per il bootstrap basterà solo una connessione è necessario perché il primo nodo è in grado di offrire introduzioni ai suoi nodi peer e quei peers sono in grado di offrire ulteriori introduzioni. E' inutile ed uno spreco di risorse di rete connettersi a più di una manciata di nodi. Dopo il bootstrap, un nodo si ricorderà le sue più recenti connessioni peer di successo, in modo che qualora venisse riavviato potrà

ristabilire rapidamente le connessioni con la sua ex rete peer. Se nessuno degli ex peers dovesse rispondere alla sua richiesta di connessione, il nodo può utilizzare i nodi del seed per avviarsi nuovamente.

Su un nodo che sta eseguendo il client Bitcoin Core, puoi elencare le connessioni dei peer con il comando getpeerinfo:

```
$ bitcoin-cli getpeerinfo
```


```
[
   {
      "addr" : "85.213.199.39:8333",
      "services" : "00000001",
      "lastsend" : 1405634126,
      "lastrecv" : 1405634127,
      "bytessent" : 23487651,
      "bytesrecv" : 138679099,
```

```
    "conntime" : 1405021768,
    "pingtime" : 0.00000000,
    "version" : 70002,
    "subver" : "/Satoshi:0.9.2.1/",
    "inbound" : false,
    "startingheight" : 310131,
    "banscore" : 0,
    "syncnode" : true
},
{
    "addr" : "58.23.244.20:8333",
    "services" : "00000001",
    "lastsend" : 1405634127,
    "lastrecv" : 1405634124,
    "bytessent" : 4460918,
    "bytesrecv" : 8903575,
    "conntime" : 1405559628,
    "pingtime" : 0.00000000,
    "version" : 70001,
    "subver" : "/Satoshi:0.8.6/",
    "inbound" : false,
```

```
        "startingheight" : 311074,
        "banscore" : 0,
        "syncnode" : false
    }
]
```

Per ignorare la gestione automatica dei peer e per specificare un elenco di indirizzi IP, gli utenti possono fornire l'opzione -connect = <IPAddress> e specificare uno o più indirizzi IP. Se si utilizza questa opzione, il nodo si collegherà solo agli indirizzi IP selezionati, invece di scoprire e mantenere automaticamente le connessioni peer.

Se non c'è traffico su una connessione, i nodi periodicamente invieranno messaggi per mantenere la

connessione. Se un nodo non ha comunicato su una connessione per più di 90 minuti, verrà disconnesso e un nuovo peer verrà ricercato. Così, il network dinamicamente regola i nodi transitori e problemi di rete, e può organicamente crescere e ridursi a seconda delle necessità, senza alcun controllo centrale.

Nodi Completi (Full Node)

I nodi completi sono quelli che mantengono una blockchain completa di tutte le transazioni . Più precisamente, dovrebbero essere chiamati "nodi blockchain completi."

Nei primi anni di Bitcoin, tutti i nodi erano completi e attualmente il client Bitcoin Core è un nodo blockchain completo. Negli ultimi due anni, tuttavia, sono state introdotte nuove forme di client bitcoin che non mantengono una blockchain completa ma eseguono una versione più leggera. Li esamineremo con maggiore dettaglio nella prossima sezione.

I full nodes della blockchain mantengono una completa e up-to-date copia della blockchain bitcoin con tutte le transazioni, che si formano indipendentemente e sono verificate, partendo con il primo blocco (blocco genesi) e costruendo l'ultimo blocco noto nella rete. Una blockchain full

node può autonomamente e autorevolmente verificare ogni transazione senza ricorrere o affidarsi a qualsiasi altro nodo o fonte di informazioni. La blockchain full node si basa sulla rete per ricevere aggiornamenti sui nuovi blocchi di operazioni, che poi verifica e incorpora nella propria copia locale della blockchain.

Eseguire un full node della blockchain ti dà un'esperienza pura di bitcoin: verifica indipendente di tutte le transazioni senza la necessità di basarsi o fidarsi di qualche altro sistema. E' facile capire se si sta eseguendo un full node perché sono richiesti più di cento gigabytes di

storage persistente (spazio del disco) per memorizzare tutta la blockchain. Se avete molta memoria nel disco e due o tre giorni per la sincronizzazione con la rete, potete eseguire un nodo completo. Questo è il prezzo di una completa indipendenza e la libertà da un'autorità centrale.

Ci sono un po' di implementazioni alternative al client full node della blockchain di bitcoin, costruite utilizzando diversi linguaggi di programmazione e architetture software. Tuttavia, l'implementazione più comune è il client di riferimento Bitcoin Core, noto anche come il client di Satoshi. Più del 75% dei nodi della rete bitcoin eseguono varie versioni di

Bitcoin Core. Si è identificato come "Satoshi" nella stringa sub-versione inviata nella versione messaggio e mostrato dal comando getpeerinfo come abbiamo visto in precedenza; per esempio, / Satoshi: 0.8.6 / .

Exchanging "repertorio" più comunemente conosciuto come exchanging inventory

La prima cosa che un full node farà, una volta essersi connesso ai peers sarà cercare di costruire una

blockchain completa. Se si tratta di un nuovo nodo, non avrà la blockchain, conoscerà un solo blocco, il blocco di Genesis, che è incorporato in modo statico nel software client. Partendo dal blocco 0 (blocco genesi), il nuovo nodo dovrà scaricare centinaia di migliaia di blocchi per la sincronizzarsi con la rete e ristabilire la blockchain competa.

l processo di sincronizzazione della blockchain inizia con il messaggio di versione, perché quello contiene *BestHeight*, l'altezza della blockchain corrente di un nodo (numero di blocchi). Un nodo vedrà il messaggio di versione dei suoi peer, conoscerà quanti blocchi ognuno ha, e sarà in

grado di confrontare il numero di blocchi che ha nella propria blockchain. I nodi peer si scambieranno un messaggio *getblocks* che contiene l'hash (impronta digitale) del blocco superiore sulla loro blockchain locale. Il nodo sarà in grado di identificare l'hash ricevuto come appartenente ad un blocco che non è in alto, ma appartenente ad un blocco vecchio, deducendo così che la propria blockchain locale è più lunga di quella del suo peer.

Il peer che ha la blockchain più lunga ha più blocchi dell'altro nodo e può identificare quali sono i blocchi di cui l'altro nodo hanno bisogno (ndt per completare la blockchain).

Identificherà i primi 500 blocchi da condividere e trasmetterà i suoi hash usando un inv (inventory) message. Il nodo a cui mancheranno questi blocchi li richiamerà mediante l'emissione di una serie di messaggi gedata che richiedono i tutti i dati del blocco, identificando i blocchi richiesti utilizzando gli hash dal messaggio inv .

Assumiamo, per esempio, che un nodo abbia solamente il genesis block. Riceverà, allora, un messaggio dai suoi pari contenente gli hash dei successivi 500 blocchi nella catena. Inizierà a richiedere blocchi da tutti i suoi pari connessi, distribuendo il carico e garantendo che non travolga qualsiasi pari di richieste. Il nodo

tiene traccia di come molti blocchi sono "in transito" per ciascuna connessione tra pari, ovvero blocchi che ha richiesto ma non ricevuto, controllando che non ecceda il limite (MAX_BLOCKS_IN_TRANSIT_PER_ In questo modo, se necessita molti blocchi, richiederà solamente dei nuovi a seguito del soddisfacimento delle precedenti richieste, consentendo ai pari di controllare il flusso degli aggiornamenti, evitando di sovraccaricare la rete. Quando ciascun blocco arriva, viene aggiunto alla blockchain, come vedremo in La Blockchain. Man mano che la blockchain si forma, sempre piu blocchi vengono richiesti e ricevuti, ed

il processo continua fino a quando il nodo raggiunge il resto della rete.

Questo processo di confrontare la blockchain locale con i pari e recuperare qualsiasi blocco mancante avviene ogni volta che un nodo si disconnette per qualsiasi periodo di tempo. Sia che un nodo sia stato offline per qualche minuto e abbia perso qualche blocco, o per un mese e abbia perso qualche migliaia di blocchi, inizia inviando getblocks, ottenendo un inv response, ed inizia scaricando i blocchi mancanti. I nodi sincronizzano la blockchain scaricando i blocchi da un altro nodo (peer) mostra l'inventario ed il protocollo di propagazione dei

blocchi.

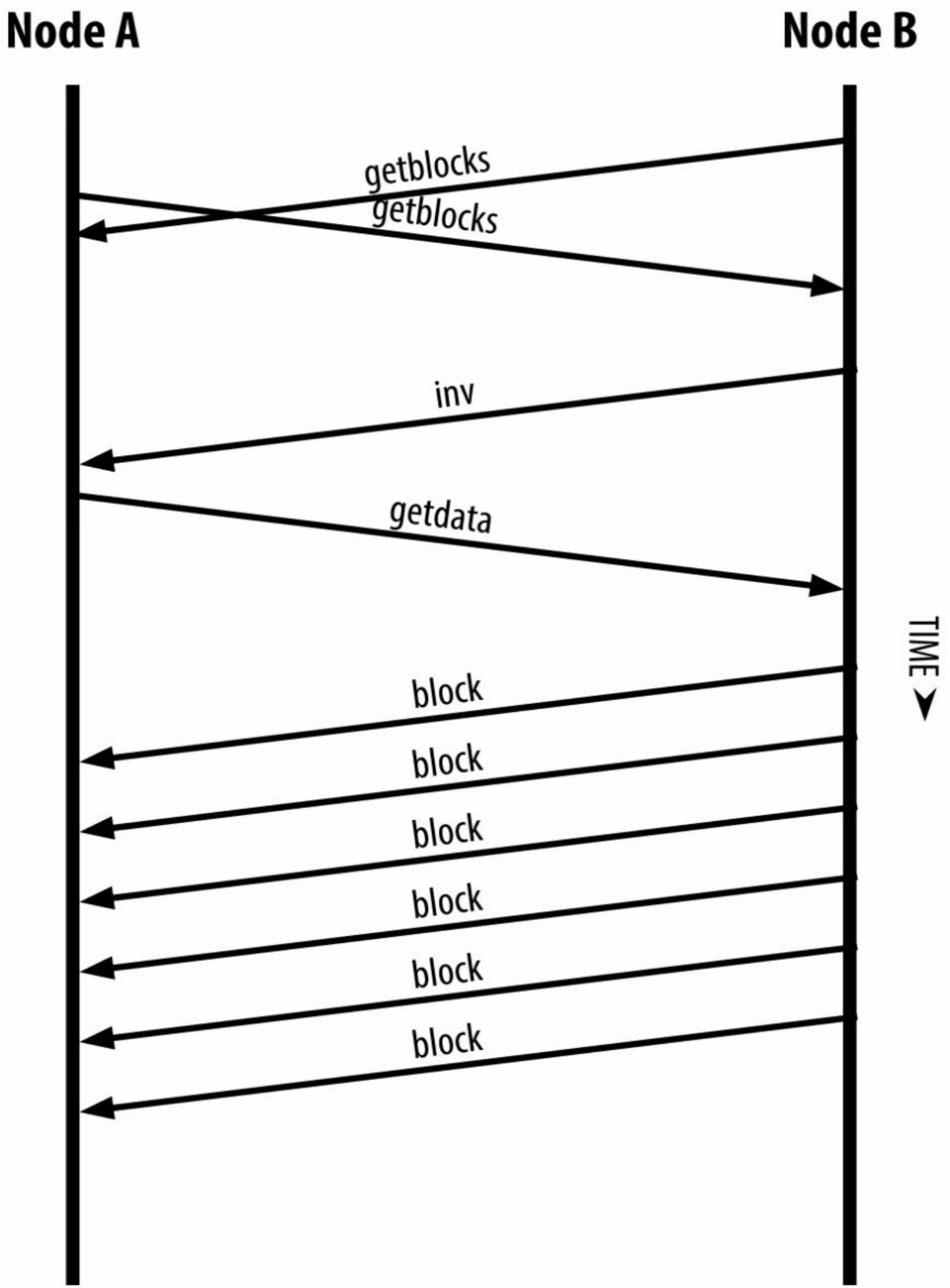

Node A
Node B
getblocks
getblocks
inv
getdata
block
block
block
block
block
block
TIME

Figura 50. I nodi sincronizzano la blockchain scaricando i blocchi da un altro nodo (peer)

Nodi di Simplified Payment Verification (SPV)

Non tutti i nodi hanno la capacità di memorizzare l'intera blockchain. Molti bitcoin client sono progettati per girare su dispositivi con potenza e spazio limitati, come smartphone, tablet, o sistemi integrati. Per questi dispositivi, un metodo simplified

payment verification (SPV) è impiegato per consentirgli di operare senza memorizzare l'intera blockchain. Questi tipi di client sono chiamati SPV client o client leggeri. Con l'aumento dell'adozione di bitcoin, il nodo SPV sta divenendo la più comune forma di nodo, specialmente per i bitcoin wallet.

I nodi SPV scaricano solo gli header del blocco e non scaricano le transazioni incluse in ogni blocco. La catena di blocchi risultante, senza transazioni, è 1.000 volte più piccola della blockchain completa. I nodi SPV non possono avere una visione completa di tutte le UTXO che sono disponibili per essere spese perché

non conoscono tutte le transazioni presenti sul network. I nodi SPV verificano le transazioni usando una metodologia leggermente diversa che si appoggia a nodi per ottenere una visione parziale dei componenti rilevanti della blockchain quando ne hanno bisogno.

Come analogia, un full node e' come un turista in una città straniera, equipaggiato di una mappa dettagliata con ogni strada ed indirizzo. A confronto, un nodo SPV e' come un turista in una città straniera che chiede a casuali sconosciuti dettagliate indicazioni, conoscendo solamente la via principale. Sebbene entrambi i turisti possano verificare l’esistenza di

una strada visitandola, il turista senza mappa non conosce cosa si celi dietro ogni vicolo e non e' a conoscenza di quali altre vie esistono. Posizionato di fronte all'indirizzo, Church Street, il turista sprovvisto di mappa non può sapere se ci sono decine di altri indirizzi, "Church Street", nella città e se questo sia quello corretto. La migliore speranza, per il turista sprovvisto di mappa, e' di chiedere alle persone e sperare che alcune di esse non stiano provando a prendersi gioco di lui.

La verifica semplificata del pagamento verifica le transazioni facendo riferimento alla loro *profondità* nella blockchain invece della loro *altezza*.

Mentre un nodo blockchain completo costruisce una catena completamente verificata di migliaia di blocchi e transazioni che raggiungono la blockchain (andando indietro nel tempo) fino al blocco genesis, un nodo SPV verificherà la catena di tutti i blocchi (ma non tutte le transazioni) e collega tale catena alla transazione di interesse.

Ad esempio, quando si esamina una transazione nel blocco 300.000, un nodo completo collega tutti i 300.000 blocchi al blocco di genesi e crea un database completo di UTXO, stabilendo la validità della transazione confermando che l'UTXO rimane non speso. Un nodo SPV non può validare

se UTXO non è speso. Invece, il nodo SPV stabilirà un collegamento tra la transazione e il blocco che lo contiene, usando un *merkle path* (vedi I Merkle Tree). Quindi, il nodo SPV attende fino a quando vede i sei blocchi da 300,001 a 300,006 impilati sopra il blocco contenente la transazione e lo verifica stabilendone la profondità sotto i blocchi da 300,006 a 300,001. Il fatto che altri nodi sulla rete hanno accettato il blocco 300.000 e poi sia stato fatto il lavoro necessario per produrre altri sei blocchi sopra di esso è la prova, per procura, che la transazione non era una doppia spesa.

Un nodo SPV non può essere persuaso che una transazione esiste in un blocco

quando la transazione non esiste di fatto. Il nodo SPV stabilisce l'esistenza di una transazione in un blocco richiedendo una prova del merkle path e convalidando la prova di lavoro nella catena di blocchi. Tuttavia, l'esistenza di una transazione può essere "nascosta" da un nodo SPV. Un nodo SPV può sicuramente provare che esiste una transazione ma non può verificare che una transazione, come una doppia spesa dello stesso UTXO, non esiste perché non ha un record di tutte le transazioni. Questa vulnerabilità può essere utilizzata in un attacco denial-of-service o per un attacco a doppia spesa contro i nodi SPV. Per difendersi da questo, un nodo

SPV deve connettersi casualmente a diversi nodi, per aumentare la probabilità che sia in contatto con almeno un nodo onesto. Questa necessità di connettersi in modo casuale significa che i nodi SPV sono anche vulnerabili agli attacchi di partizionamento di rete o agli attacchi Sybil, dove i nodi sono connessi a nodi falsi o reti fittizie e non hanno accesso a nodi onesti o alla vera rete di bitcoin.

Per scopi più pratici, i nodi SPV ben connessi sono abbastanza sicuri, trovando il giusto equilibrio di risorse necessarie, praticità e sicurezza. Per una sicurezza infallibile, tuttavia, niente batte l'eseguire un full

blockchain node.

TIP	Un nodo completo blockchain, verifica una transazione controllando l'intera catena di migliaia di blocchi al di sotto di essa al fine di garantire che l'UTXO non sia già stato speso mentre un nodo SPV controlla che un blocco sia sepolto da una manciata di blocchi successivi ad esso.

Per ottenere le intestazioni dei blocchi,

i nodi SPV usano un getheaders message invece di getblocks. Il peer rispondente invierà fino a 2.000 intestazioni di blocco usando un singolo messaggio di header . Il processo è uguale a quello utilizzato da un nodo completo per recuperare blocchi completi. I nodi SPV impostano anche un filtro sulla connessione ai peer, per filtrare il flusso di futuri blocchi e transazioni inviati dai peer. Qualsiasi transazione di interesse viene recuperata utilizzando una richiesta getdata. Il peer genera un messaggio tx contenente le transazioni, in risposta. Nodo di SPV nella fase di sincronizzazione degli header del blocco mostra la

sincronizzazione delle intestazioni dei blocchi.

Poiché i nodi SPV devono recuperare transazioni specifiche per verificarli in modo selettivo, creano anche un rischio per la privacy. A differenza dei full blockchain nodes, che raccolgono tutte le transazioni all'interno di ciascun blocco, le richieste di dati specifici del nodo SPV possono rivelare inavvertitamente gli indirizzi nel loro portafoglio. Ad esempio, una terza parte che controlla una rete può tenere traccia di tutte le transazioni richieste da un portafoglio su un nodo SPV e utilizzare quelle per associare gli indirizzi bitcoin all'utente di quel portafoglio, distruggendo la privacy

dell’utente.

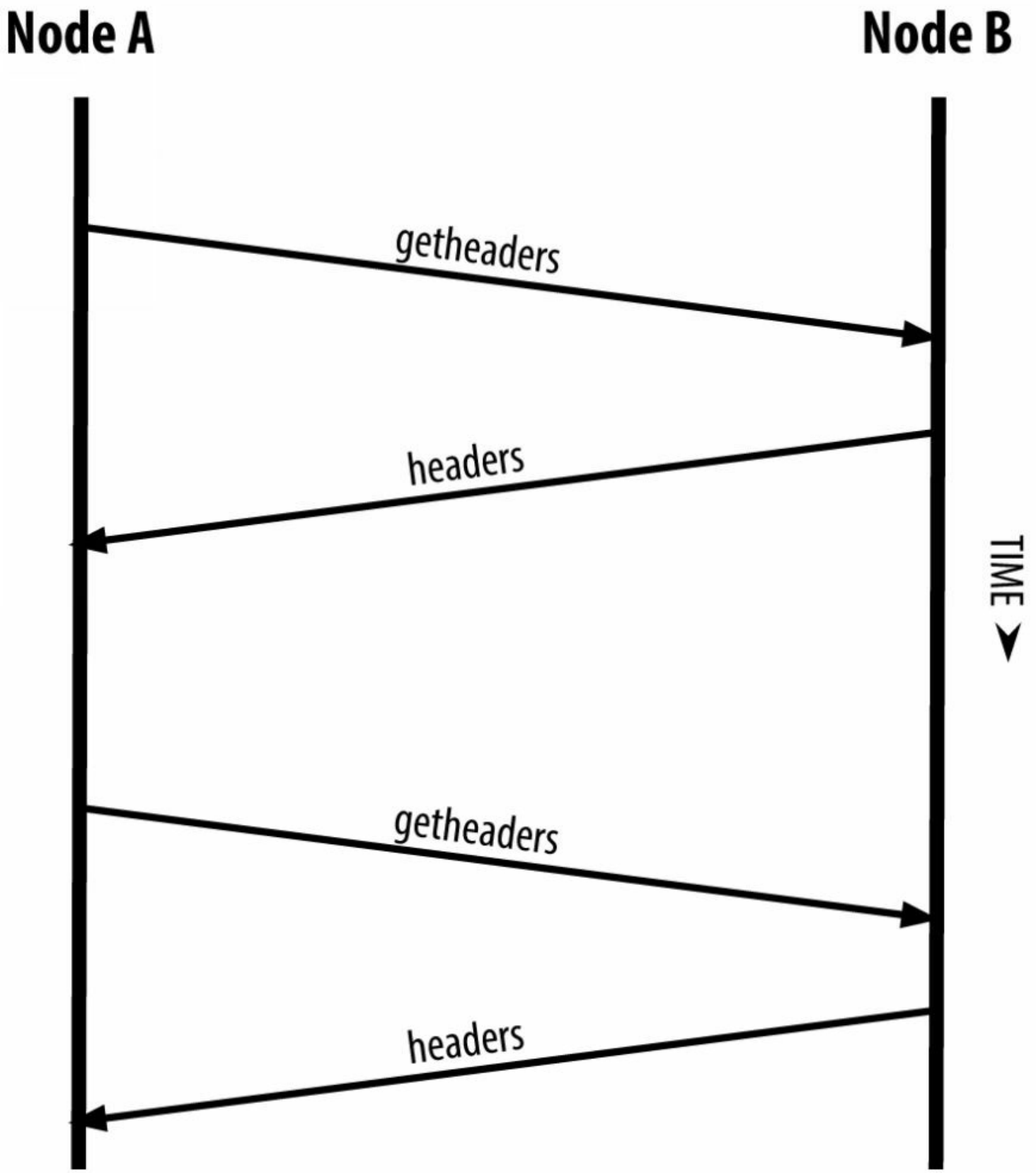


Figura 51. Nodo di SPV nella fase di sincronizzazione degli

header del blocco

Poco dopo l'introduzione dei nodi SPV/lightweight, gli sviluppatori di bitcoin hanno aggiunto una funzione chiamata *bloom filters* per affrontare i rischi per la privacy dei nodi SPV. I filtri Bloom consentono ai nodi SPV di ricevere un sottoinsieme di transazioni senza rivelare con precisione gli indirizzi a cui sono interessati, attraverso un meccanismo di filtraggio che utilizza probabilità anziché pattern fissi.

Filtri di Bloom (Bloom Filter)

Un filtro Bloom è un filtro di ricerca

probabilistico, un modo per descrivere uno schema desiderato senza specificarlo esattamente. I filtri Bloom offrono un modo efficace per esprimere un modello di ricerca proteggendo al contempo la privacy. Vengono utilizzati dai nodi SPV per chiedere ai loro peer transazioni corrispondenti a un modello specifico, senza rivelare esattamente quali indirizzi stanno cercando.

Nella nostra precedente analogia, un turista senza una mappa sta chiedendo indicazioni per un indirizzo specifico, "23 Church St." Se chiede agli estranei le indicazioni per questa strada, rivela inavvertitamente la sua destinazione. Un filtro Bloom è come chiedere: "Ci

sono strade in questo quartiere il cui nome finisce in R-C-H?" Una domanda del genere rivela meno della destinazione desiderata che chiedere "23 Church St." Usando questa tecnica, un turista potrebbe specificare l'indirizzo desiderato in modo più dettagliato come "che termina con U-R-C-H" o meno come "che finisce con H.". Variando la precisione della ricerca, il turista rivela più o meno informazioni, a scapito di ottenere risultati più o meno specifici. Se chiede uno schema meno specifico, ottiene molti più indirizzi possibili e una migliore privacy, ma molti risultati sono irrilevanti. Se chiede uno schema molto specifico, ottiene meno risultati

ma perde la privacy.

I filtri Bloom servono a questa funzione consentendo a un nodo SPV di specificare un modello di ricerca per le transazioni che possono essere regolate in base alla precisione o alla privacy. Un filtro bloom più specifico produrrà risultati accurati, ma a scapito di rivelare quali indirizzi vengono utilizzati nel portafoglio dell'utente. Un filtro meno specifico produrrà più dati su più transazioni, molti irrilevanti per il nodo, ma consentirà al nodo di mantenere una migliore privacy.

Come funzionano i Bloom Filters

I filtri Bloom vengono implementati come una matrice di dimensioni variabili di N cifre binarie (un campo di bit) e un numero variabile di funzioni di hash M. Le funzioni di hash sono progettate per produrre sempre un'uscita compresa tra 1 e N, corrispondente alla matrice di cifre binarie. Le funzioni di hash vengono generate in modo deterministico, in modo che qualsiasi nodo che implementa un filtro bloom utilizzi sempre le stesse funzioni di hash e ottenga gli stessi risultati per un input specifico. Scegliendo filtri bloom di lunghezza diversa (N) e un numero diverso (M) di funzioni di hash, il filtro può essere regolato, variando il

livello di precisione e quindi la privacy.

I n Un esempio di un filtro bloom semplice, con un campo a 16-bit e tre funzioni di hashing, usiamo un array molto piccolo di 16 bit e una serie di tre funzioni di hashing per dimostrare come funzionano i bloom filter.

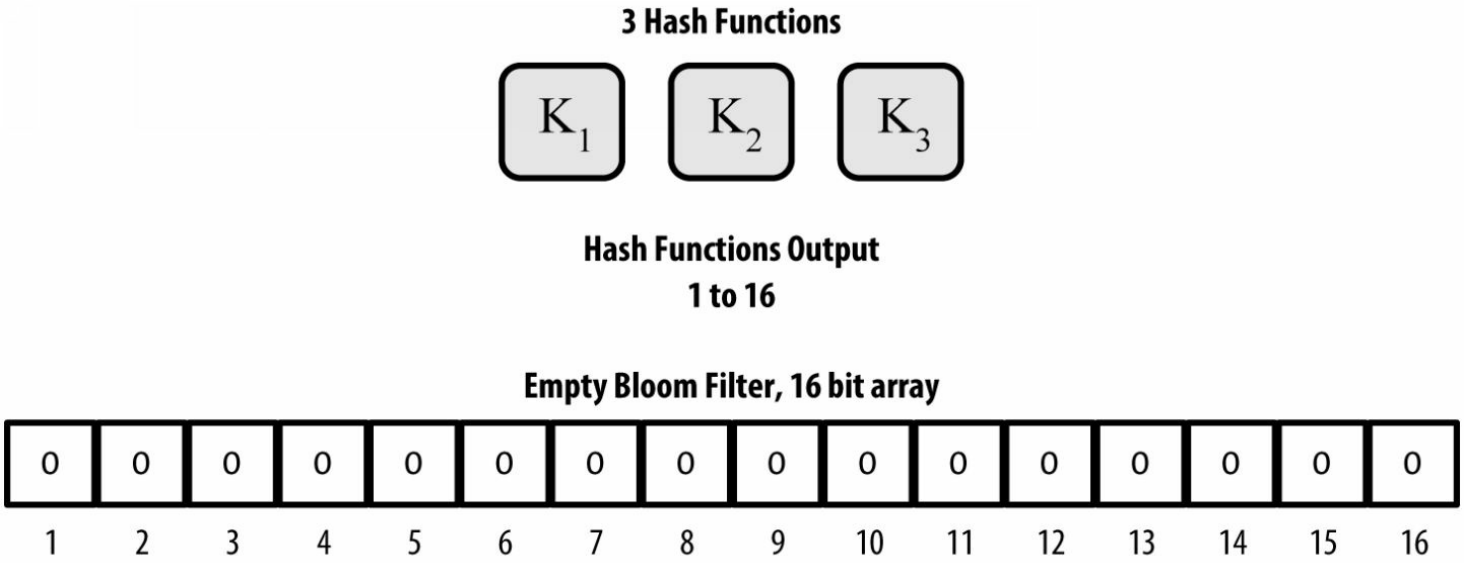


Figura 52. Un esempio di un filtro bloom semplice, con un campo a 16-bit e tre funzioni

di hashing

Il filtro bloom viene inizializzato in modo tale che l'array di bit sia valorizzato tutto a zero. Per aggiungere un pattern al filtro bloom, il pattern viene sostituito a turno da ciascuna funzione di hash. Applicando la prima funzione di hash all'input si ottiene un numero compreso tra 1 e N. Il bit corrispondente nell'array (indicizzato da 1 a N) viene trovato e impostato su 1, registrando in tal modo l'output della funzione di hash. Quindi, la funzione di hash successiva viene utilizzata per impostare un altro bit e così via. Una volta applicate tutte le funzioni di hash M, il pattern di ricerca verrà "registrato" nel filtro bloom

come bit M che sono stati modificati da 0 a 1.

Aggiungendo un pattern "A" al nostro filtro bloom semplice è un esempio di aggiunta di un pattern "A" al semplice filtro bloom mostrato in Un esempio di un filtro bloom semplice, con un campo a 16-bit e tre funzioni di hashing.

Aggiungere un secondo pattern è semplice basta ripetere questo processo. Il motivo viene alternato a turno da ciascuna funzione di hash e il risultato viene registrato impostando i bit su 1. Si noti che come un filtro bloom è pieno di più pattern, un risultato di funzione hash potrebbe coincidere con un bit che è già

impostato su 1, nel qual caso il bit non viene modificato. In sostanza, poiché più pattern registrano su bit sovrapposti, il filtro bloom inizia a saturarsi con più bit impostati su 1 e la precisione del filtro diminuisce. Questo è il motivo per cui il filtro è una struttura di dati probabilistica: diventa meno preciso man mano che vengono aggiunti più modelli. L'accuratezza dipende dal numero di pattern aggiunti rispetto alla dimensione dell'array di bit (N) e dal numero di funzioni hash (M). Più un array di bit è grande, più funzioni di hash possono registrare più pattern con maggiore precisione. Un array di bit più piccolo o un minor numero di

funzioni di hash registreranno meno pattern e produrranno meno precisione.

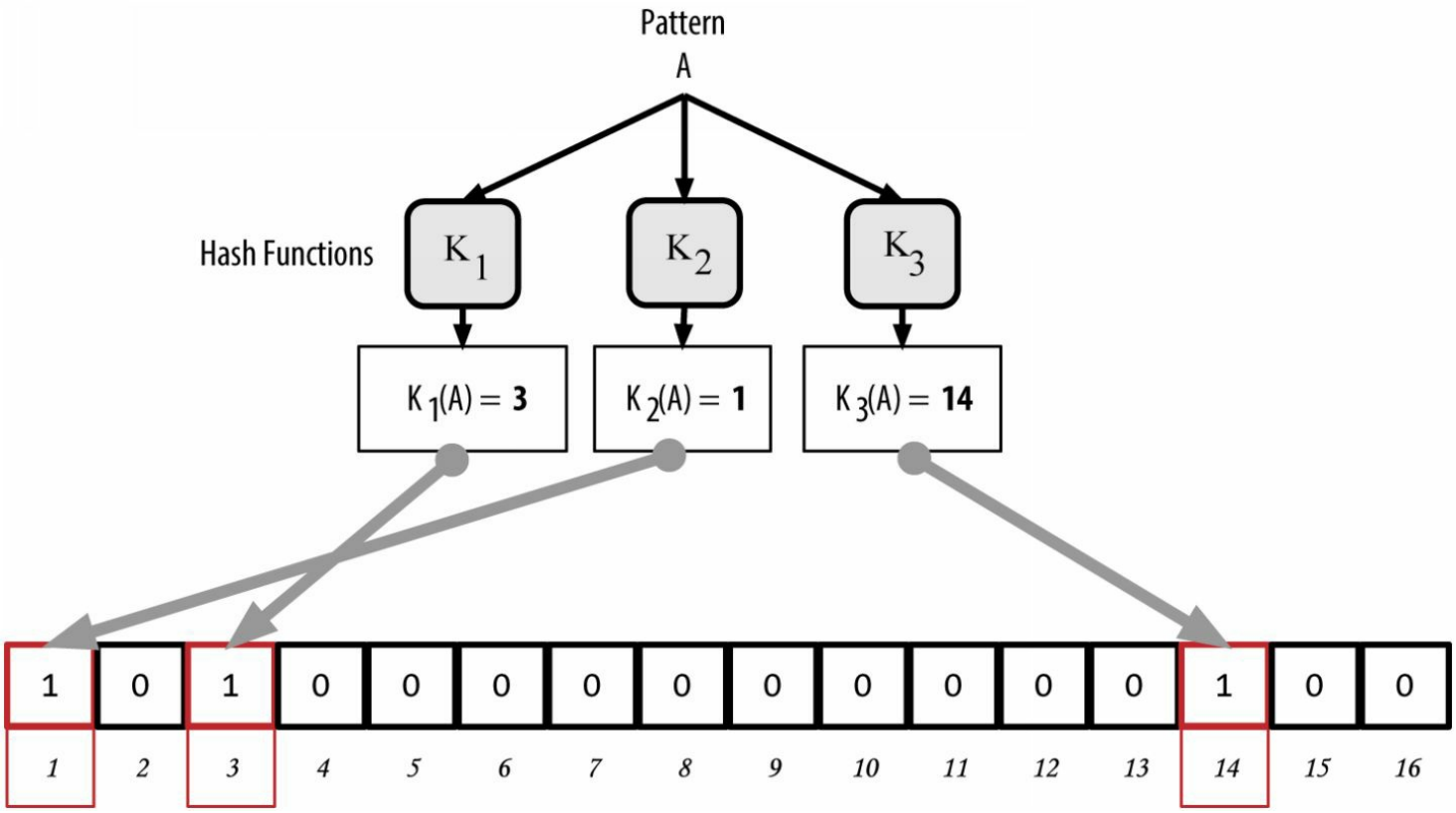


Figura 53. Aggiungendo un pattern "A" al nostro filtro bloom semplice

Aggiungendo un secondo pattern "B" al nostro filtro bloom semplice è un esempio di aggiunta di un secondo

pattern "B" al filtro bloom semplice.

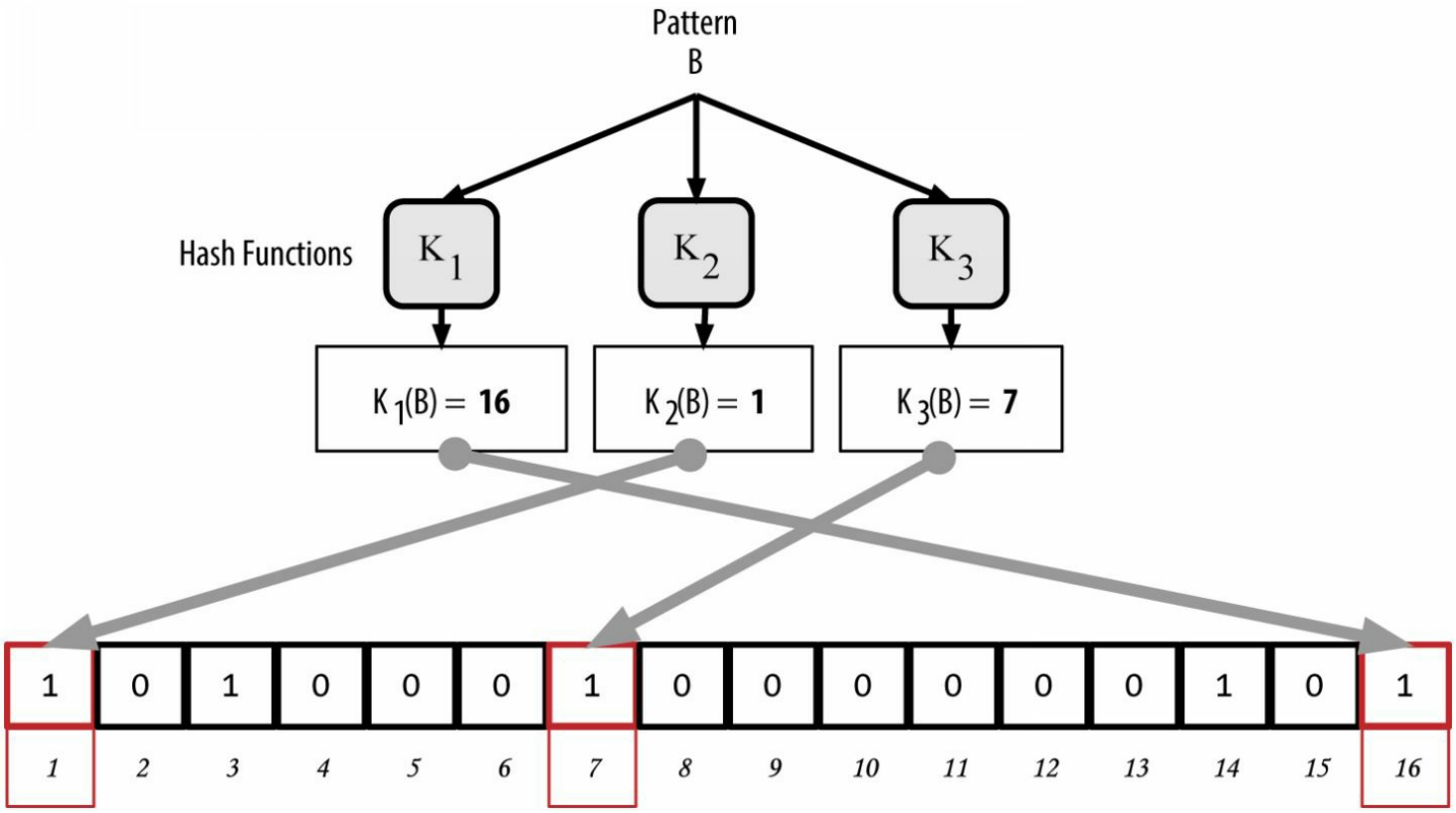


Figura 54. Aggiungendo un secondo pattern "B" al nostro filtro bloom semplice

Per testare se un pattern e' parte di un bloom filter, il pattern è hashato da ogni funzione di hash ed il risultante bit pattern e' testato rispetto al bit array. Se tutti i bit indicizzati dalle

funzioni di hash sono impostati su 1, allora il pattern e' *probablimente* registrato nel bloom filter. Poiché i bit possono essere impostati (ndt ad 1) a causa della sovrapposizione di più schemi, la risposta non è certa, ma è piuttosto probabilistica. In termini semplici, una corrispondenza positiva del filtro bloom è "Forse, Sì".

Testando l'esistenza del pattern "X" nel bloom filter. Il risultato è una corrispondenza probabilistica positiva, che sta a significare "Possibile." è un esempio di controllo dell'esistenza di un pattern "X" nel filtro bloom semplice. I bit corrispondenti sono impostati a 1, quindi il pattern probabilmente è una

corrispondenza.

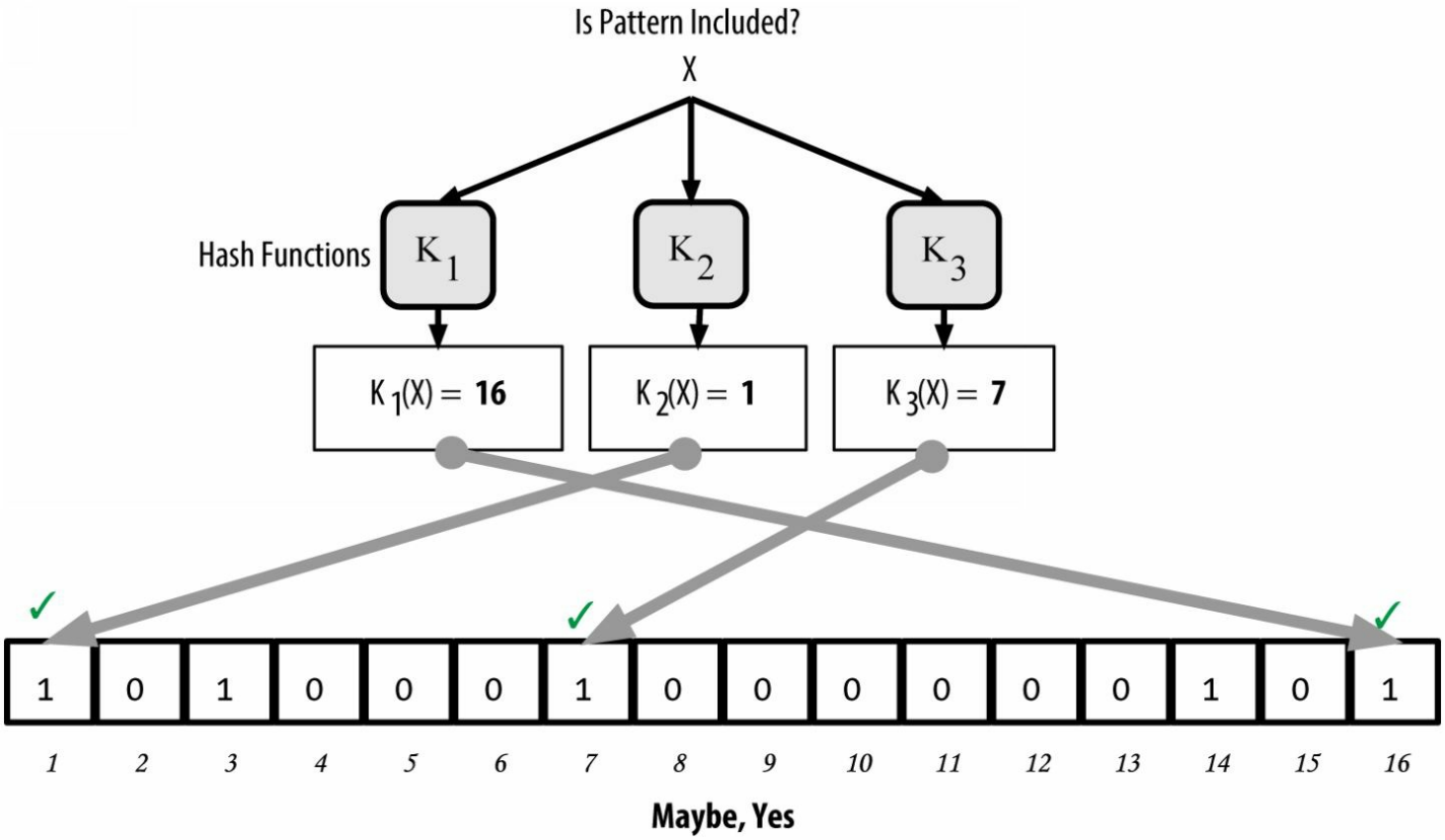


Figura 55. Testando l'esistenza del pattern "X" nel bloom filter. Il risultato è una corrispondenza probabilistica positiva, che sta a significare "Possibile."

Al contrario, se un pattern viene testato rispetto al filtro bloom e uno qualsiasi dei bit è impostato su 0, ciò dimostra che il pattern non è stato registrato nel filtro bloom. Un risultato negativo non è una probabilità, è una certezza. In termini semplici, una corrispondenza negativa su un filtro bloom è "Assolutamente no!"

Testando l'esistenza del pattern "Y" nel filtro bloom. Il risultato sarà una corrispondenza definitiva negativa, che sta a significare "Sicuramente No!" è un esempio di test dell'esistenza del pattern "Y" nel filtro bloom semplice. Uno dei bit corrispondenti è impostato su 0, quindi il pattern non è sicuramente una

corrispondenza.

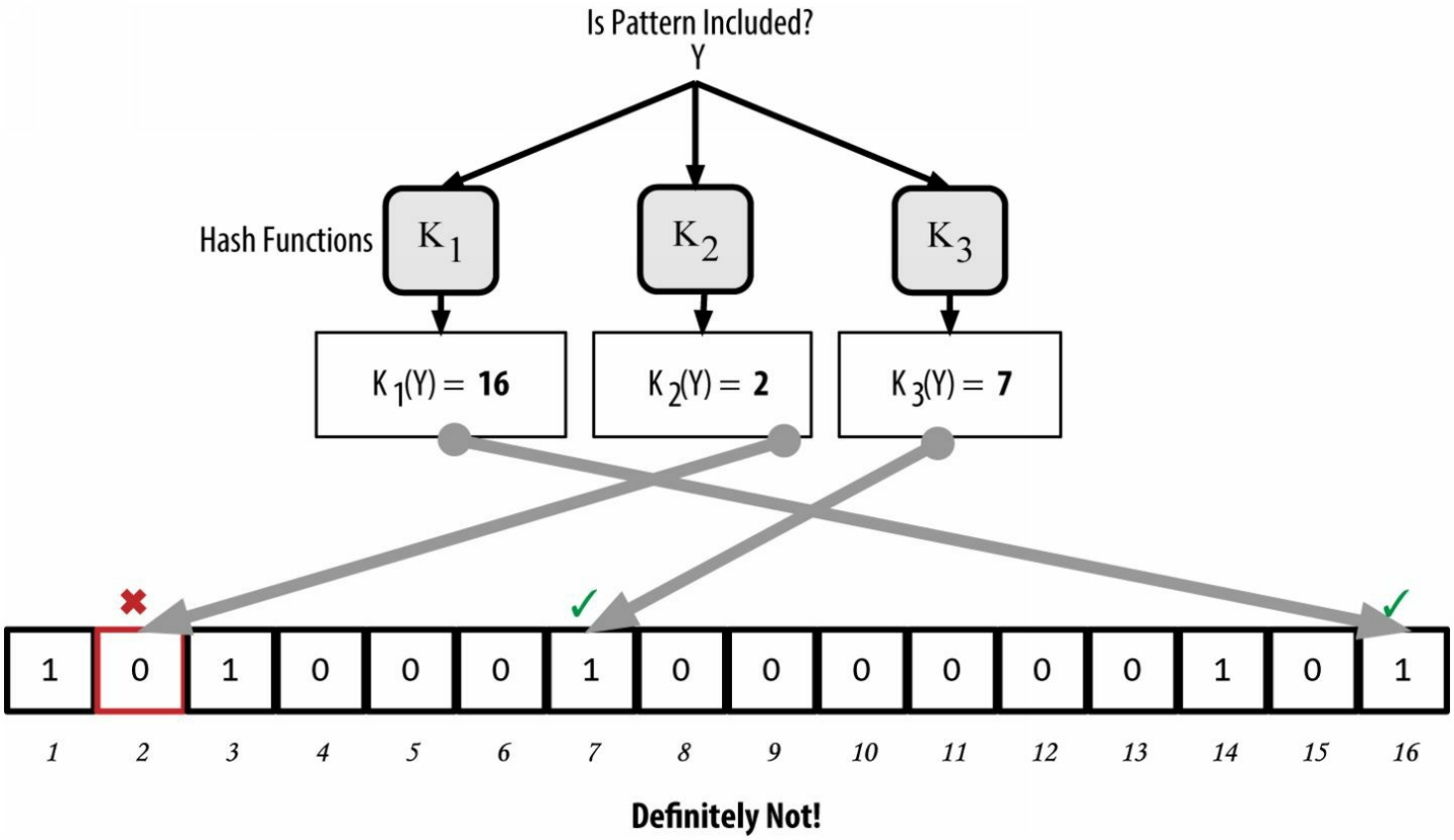


Figura 56. Testando l'esistenza del pattern "Y" nel filtro bloom. Il risultato sarà una corrispondenza definitiva negativa, che sta a significare "Sicuramente

No!"

Come vengono utilizzati i Bloom Filters dai nodi SPV

I filtri bloom vengono utilizzati per filtrare le transazioni (e i blocchi che le contengono) che un nodo SPV riceve dai suoi pari. I nodi SPV creeranno un filtro che corrisponde solo agli indirizzi contenuti nel portafoglio del nodo SPV senza rivelare a quali indirizzi o chiavi è interessato.

Un nodo SPV inizializzerà un filtro bloom come "vuoto" e in tale stato il

filtro non corrisponderà a nessun pattern (n.d.r. modello). Il nodo SPV creerà quindi un elenco di tutti gli indirizzi nel suo portafoglio e creerà un pattern di ricerca corrispondente all'output della transazione corrispondente a ciascun indirizzo. Lo fa estraendo l'hash della chiave pubblica, l'hash dello script e gli ID delle transazioni da qualsiasi UTXO controllato dal suo portafoglio. Il nodo SPV aggiunge quindi ciascuno di questi al filtro di bloom, in modo che il filtro si "abbini" se questi modelli sono presenti in una transazione, senza rivelare i modelli stessi.

Il nodo SPV invierà quindi un messaggio di caricamento del filtro al

peer, contenente il filtro bloom da utilizzare sulla connessione. Sul peer, i filtri di bloom vengono controllati rispetto a ciascuna transazione in entrata. Il nodo completo controlla diverse parti della transazione rispetto al filtro bloom, cercando una corrispondenza che includa:

- L'ID della transazione
- I componenti di dati dagli script di blocco di ciascuno degli output di transazione (ogni chiave e hash nello script)
- Ognuno degli input della transazione
- Ciascuno dei componenti di dati

della firma di input (o script di witness)

Controllando tutti questi componenti, è possibile utilizzare i filtri di bloom per associare gli hash delle chiavi pubbliche, gli script, i valori OP_RETURN, le chiavi pubbliche nelle firme o qualsiasi componente futuro di un contratto intelligente o di uno script complesso.

Dopo aver stabilito un filtro, il peer testerà quindi le uscite di ogni transazione con il filtro bloom. Solo le transazioni che corrispondono al filtro vengono inviate al nodo.

In risposta ad un messaggio getdata dal nodo, i peer invieranno un messaggio

merkleblock che contiene solo gli headers del blocco che hanno dato esito positivo per il filtro indicato ed un merkle path (vedi I Merkle Tree) per ogni transazione accoppiata. il peer invierà poi anche messaggi tx contenenti le transazioni corrispondenti al filtro.

Quando il nodo completo invia le transazioni al nodo SPV, il nodo SPV elimina eventuali falsi positivi e utilizza le transazioni correttamente abbinate per aggiornare il set UTXO e il saldo del wallet. Poiché aggiorna la propria vista del set UTXO, modifica anche il filtro bloom in modo che corrisponda a tutte le transazioni future che fanno riferimento all'UTXO

appena trovato. Il nodo completo utilizza quindi il nuovo filtro di bloom per abbinare le nuove transazioni e l'intero processo si ripete.

Il nodo che imposta il filtro bloom può aggiungere in modo interattivo pattern al filtro inviando un messaggio filteradd. Per cancellare il filtro bloom, il nodo può inviare un messaggio filterclear. Poiché non è possibile rimuovere un pattern da un filtro di bloom, un nodo deve cancellare e inviare nuovamente un nuovo filtro bloom se un pattern non è più desiderato.

Il protocollo di rete e il meccanismo dei filtri bloom per i nodi SPV sono definiti in BIP-37 (Peer Services).

Nodi SPV e Privacy

I nodi che implementano SPV hanno una privacy più debole rispetto a un nodo completo. Un nodo completo riceve tutte le transazioni e quindi non rivela alcuna informazione sul fatto che stia utilizzando un determinato indirizzo del proprio portafoglio. Un nodo SPV riceve un elenco filtrato di transazioni relative agli indirizzi che si trovano nel portafoglio. Di conseguenza, riduce la privacy del proprietario che ne fa uso.

I filtri Bloom sono un modo per ridurre i rischi relativi alla privacy derivanti dall'utilizzo di SPV. Senza di essi, un nodo SPV dovrebbe

elencare esplicitamente gli indirizzi a cui è interessato, creando un serio problema relativo alla privacy. Tuttavia, nonostante l'utilizzo di filtri bloom, un avversario che monitora il traffico di un client SPV o connesso direttamente come nodo nella rete P2P può raccogliere abbastanza informazioni nel tempo per apprendere gli indirizzi nel portafoglio del client SPV.

Connessioni crittografate e autenticate

La maggior parte dei nuovi utenti di bitcoin presuppone che le

comunicazioni di rete di un nodo bitcoin siano crittografate, sbagliato. L'implementazione originale di bitcoin comunica interamente in chiaro. Anche se questo non è un problema di privacy importante per i full node, è un grosso problema per i nodi SPV.

Per aumentare la privacy e la sicurezza della rete P2P bitcoin, esistono due soluzioni che forniscono la crittografia delle comunicazioni: *Tor Transport* e *P2P Authentication and Encryption* con BIP-150/151.

Tor Transport

Tor, che sta per *The Onion Routing network*, è un progetto software e una rete che offre la crittografia e

l'incapsulamento dei dati attraverso percorsi di rete randomizzati che offrono anonimato, non-tracciabilità e privacy.

Bitcoin Core offre diverse opzioni di configurazione che consentono di eseguire un nodo bitcoin ed il relativo traffico trasportato sulla rete Tor. Inoltre, Bitcoin Core può anche offrire un hidden service di Tor che consente ad altri nodi Tor di connettersi al tuo nodo direttamente via Tor.

A partire da Bitcoin Core v0.12, un nodo offrirà automaticamente un hiden Tor service se è in grado di connettersi ad un servizio Tor locale. Se hai installato Tor e viene eseguito Bitcoin Core come utente con

autorizzazioni adeguate per accedere al cookie di autenticazione Tor, il tutto dovrebbe funzionare automaticamente. Usa il flag di debug per attivare il debug di Bitcoin Core per il servizio Tor in questo modo:

```
$ bitcoind --daemon --debug=tor
```

Dovresti vedere "tor: ADD_ONION successful" nei log, il che sta ad indicare che Bitcoin Core ha aggiunto un hidden service alla rete Tor.

Puoi trovare ulteriori istruzioni sull'esecuzione di Bitcoin Core come servizio nascosto di Tor nella documentazione di Bitcoin Core (*docs/tor.md*) e vari tutorial online.

Autenticazione e Crittografia

Peer-to-Peer

Due proposte di miglioramento Bitcoin, BIP-150 e BIP-151, aggiungono supporto per l'autenticazione P2P e la crittografia nella rete P2P di bitcoin. Questi due BIP definiscono servizi opzionali che possono essere offerti da nodi bitcoin compatibili. BIP-151 consente la crittografia negoziata per tutte le comunicazioni tra due nodi che supportano BIP-151. BIP-150 offre un'autenticazione peer opzionale che consente ai nodi di autenticare l'identità di altri nodi utilizzando ECDSA e chiavi private. BIP-150 richiede che prima dell'autenticazione i due nodi abbiano stabilito

comunicazioni crittografate come da BIP-151.

A gennaio 2017, BIP-150 e BIP-151 non sono ancora implementati in Bitcoin Core. Tuttavia, le due proposte sono state implementate da almeno un client bitcoin alternativo denominato bcoin.

BIP-150 e BIP-151 consentono agli utenti di eseguire un client SPV connettendosi ad un nodo completo fidato, utilizzando la crittografia e l'autenticazione per proteggere la privacy del client SPV.

Inoltre, l'autenticazione può essere utilizzata per creare reti di nodi bitcoin affidabili e prevenire attacchi

Man-in-the-Middle. Infine, la crittografia P2P, se applicata ampiamente, rafforzerebbe la resistenza del bitcoin all'analisi del traffico e alla conseguente erosione della privacy, specialmente nei paesi totalitari in cui l'uso di Internet è pesantemente controllato e monitorato.

Gli standard sono definiti in BIP-150 (Peer Authentication) e BIP-151 (Peer-to-Peer Communication Encryption).

Gruppo di Transazioni (Transaction Pools)

Quasi tutti i nodi della rete bitcoin mantengono un elenco temporaneo di

transazioni non confermate chiamato *memory pool*, *mempool* o *transaction pool*. I nodi utilizzano questo pool per tenere traccia delle transazioni note alla rete ma non ancora incluse nella blockchain. Ad esempio, un nodo che contiene il wallet di un utente utilizzerà il pool di transazioni per tenere traccia dei pagamenti in entrata sul wallet dell'utente che sono stati ricevuti sulla rete ma non ancora confermati.

Man mano che le transazioni sono ricevute e verificate, sono aggiunte alla pool delle transazioni e trasmesse ai nodi vicini che le propagano sul network.

Alcune implementazioni di nodi

mantengono un pool separato per le transazioni orfane. Se gli input di una transazione fanno riferimento ad una transazione non ancora nota, ad esempio senza la transazione che l'ha generata, la transazione orfana verrà memorizzata temporaneamente nel pool delle transazioni orfane finché non arriva la transazione padre.

Quando una transazione viene aggiunta al pool di transazioni, il pool orfano viene controllato per tutti gli orfani che fanno riferimento agli output di questa transazione (i relativi figli). Tutti gli orfani corrispondenti vengono quindi convalidati. Se validi, vengono rimossi dal pool orfano e aggiunti al pool di transazioni, completando la

catena avviata con la transazione principale. Alla luce della transazione appena aggiunta, che non è più orfana, il processo viene ripetuto ricorsivamente alla ricerca di ulteriori discendenti, fino a quando non vengono trovati più discendenti. Attraverso questo processo, l'arrivo di una transazione padre innesca una ricostruzione a cascata di un'intera catena di transazioni interdipendenti, riunendo gli orfani con i loro genitori lungo tutta la catena.

Sia il pool di transazioni sia il pool orfano (dove implementato) sono memorizzati nella memoria locale e non vengono salvati nella memoria permanente; piuttosto, sono popolati

dinamicamente dai messaggi in arrivo dalla rete. All'avvio di un nodo, entrambi i pool sono vuoti e vengono gradualmente popolati con nuove transazioni ricevute dalla rete.

Alcune implementazioni del client bitcoin mantengono anche un database UTXO o un pool di UTXO, che è l'insieme di tutti gli output non spesi sulla blockchain. Sebbene il nome "UTXO pool" suona simile al pool di transazioni, rappresenta un diverso insieme di dati. A differenza della transazione e dei pool orfani, l'UTXO pool non è inizializzato vuoto ma contiene invece milioni di voci di output di transazione non spesi, tutto ciò che non è stato speso fino al

blocco di genesi. Il pool UTXO può essere ospitato nella memoria locale o come tabella di database indicizzata su storage persistente.

Mentre i pool di transazioni e i pool orfani rappresentano la prospettiva locale di un singolo nodo e potrebbero variare significativamente da nodo a nodo a seconda di quando il nodo è stato avviato o riavviato, il pool UTXO rappresenta il consenso emergente della rete e pertanto varierà poco tra i nodi. Inoltre, la transazione e i pool orfani contengono solo transazioni non confermate, mentre il pool UTXO contiene solo output confermati.

La Blockchain

Introduzione

I dati della struttura dati blockchain sono un'ordinata lista di blocchi di transazioni back-linked (ndt ogni blocco è collegato al blocco precedente). La blockchain può essere salvata come un file piatto, o in un semplice database. Il client Bitcoin Core salva i metadati della blockchain usando il database LevelDB (libreria C scritta dai programmatori che lavorano a Google). I blocchi sono collegati "indietro", ognuno si riferisce al blocco precedente presente nella catena. La blockchain è spesso

visualizzata come una pila verticale, con blocchi stratificati l'uno sopra l'altro e il primo blocco serve da fondamenta della pila. La visualizzazione dei blocchi impilati uno sopra l'altro provoca l'uso di terminologie come "altezza" (height) per riferirsi alla distanza dal primo blocco, e "top" o "tip" (cima o punta) per riferirsi al blocco aggiunto più recentemente.

Ogni blocco contenuto nella blockchain è identificato da un hash, generato usando l'algoritmo crittografico di hashing SHA256 sull'header del blocco. Ogni blocco inoltre referenzia il blocco precedente, conosciuto anche come *parent* block,

attraverso il campo "previous block hash" nel block header. La sequenza di hash che collegano ogni blocco al proprio parent crea una catena che si collega, blocco per blocco fino al primo blocco creato, conosciuto con il nome di *genesis block* , ovvero blocco di genesi.

Anche se un blocco ha solo un genitore, può temporaneamente avere multipli figli. Ognuno dei figli si riferisce allo stesso blocco del padre e contiene lo stesso hash (padre) nel campo "previous block hash" (hash del blocco precedente). Multipli figli emergono durante un "fork" (biforcazione) della blockchain, una situazione temporanea che accade

quando differenti blocchi sono scoperti quasi simultaneamente da miner differenti (vedi I Fork della Blockchain). Eventualmente, solo un blocco figlio diventa parte della blockchain e la "biforcazione" viene risolta. Anche se un blocco può avere più di un figlio, ogni blocco può avere solo un genitore. Questo perchè un blocco ha un singolo campo "hash del blocco precedente" (previous block hash) che si riferisce al suo singolo genitore.

Il campo "previous block hash" è dentro l'header del blocco e per questo influenza l'hash del blocco *attuale*. L'identità stessa del figlio cambia nel caso in cui cambi l'identità

del genitore. Quando il genitore viene modificato in qualsiasi modo, l'hash del genitore cambia. L'hash modificato del genitore necessita un cambio nel puntatore "previous block hash" del figlio. Questo a sua volta causa il cambio dell'hash del figlio, che richiede il cambio nel puntatore dell'hash nipote, che a sua volta cambia il puntamento al nipote, e così via… Questo effetto a cascata assicura che fino a quando un blocco ha tante generazioni di blocchi che lo seguono, non può essere modificato senza forzare un ricalcolo su tutti i blocchi seguenti. Visto che per questo ricalcolo servirebbe un'enorme potenza computazionale, l'esistenza di

una lunga catena di blocchi fa si che la storia più profonda della blockchain sia immutabile, che è una degli elementi chiave della sicurezza di bitcoin.

Un modo di pensare alla blockchain è come ai livelli di una stratificazione geologica, o a una carota di ghiaccio. Gli strati superficiali possono cambiare con le stagioni o anche essere eliminati prima che abbiano il tempo di depositarsi, ma se si scende di pochi centimetri gli strati geologici diventano sempre più stabili. Al punto che arrivi a guardare poche centinaia di metri sotto stai vedendo una foto del passato che è stata lasciata intatta da milioni di anni. Nella blockchain i

blocchi più recenti potrebbero cambiare se c'è un fork. Gli ultimi sei blocchi sono come alcuni centimetri del suolo. Ma se vai più in profondità nella blockchain, oltre sei blocchi, i blocchi hanno sempre meno probabilità di essere modificati. Dopo 100 blocchi c'è così tanta solidità che la transazione di coinbase - una transazione contenente i nuovi bitcoin minati - può essere spesa. Alcune migliaia di blocchi indietro (un mese), e la blockchain è saldata nella storia, per qualunque tipo di dato. Mentre il protocollo permette sempre che una catena possa essere sostituita da una catena più lunga ed esiste sempre la possibilità che una catena diventi

reversibile, la possibilità che questo accada diminuisce coi blocchi fino a diventare infinitesima.

Struttura del Blocco

Un blocco è un contenitore di una struttura dati che riunisce le transazioni da includere nel pubblico registro, la blockchain. Il blocco è composto da un'intestazione, contenente i metadati, seguito da una lunga lista di transazioni che costituiscono la maggior parte delle sue dimensioni. L'intestazione del blocco è di 80 byte, mentre la media delle transazioni è di almeno 400 byte e in media un blocco contiene più di 1900 transazioni. Un blocco completo,

con tutte le transazioni, è perciò 10000 volte più grande di un'intestazione del blocco. La struttura di un blocco descrive la struttura di un blocco.

Tabella 23. La struttura di un blocc

Dimensione	Campo	Descri
4 byte	Dimensione del Blocco (Block Size)	La dimens del bl in byte
80 bytes	Header del Blocco (Block Header)	Multip campi dall'he del blc
1–9 bytes	Contatore di	Quante

(VarInt)	transazione (Transaction Counter)	transaz seguon
Variabile	Transazioni	Le transaz registra nel blo

Header del Blocco

L'intestazione del blocco consiste in tre gruppi di metadata. Nel primo, c'è un riferimento al precedente hash, il quale connette questo blocco al precedente blocco nella blockchain. Il secondo set di metadata, cioè il

difficulty,*timestamp*, e *nonce*,riguarda la competizione del mining, come dettagliato in Il Mining e Il Consenso. Il terzo pezzo di metadata è il merkle tree root, una struttura dati usata per riassumere efficientemente tutte le transazioni nel blocco.. La struttura di un block header descrive la struttura di un'intestazione del blocco (block header).

Tabella 24. La struttura di un header

Dimensione	Campo	Descrizi
4 byte	Versione	Un nume versione tracciare

		upgrade software al protoc
32 byte	Hash del Blocco Precedente	Un riferimen all'hash blocco precedent (genitore nella cha
32 byte	Merkle Root	Un'hash radice merkle delle transazio questo b

4 byte	Timestamp	Il approssi della creazion blocco corrente (secondi Unix Epo
4 bytes	Target di Difficoltà (Difficulty Target)	Il targe difficolta dell'algo di pro work questo b
4 bytes	Nonce	Un con utilizzato l'algorit

		proof-of

Il nonce, il target di difficoltà, e il timestamp sono usati nel processo di mining e saranno discussi in maggiore dettaglio nel Il Mining e Il Consenso.

Identificatori di blocco: Hash del Block Header e Altezza del Blocco (Block Height)

Il primo identificatore di un blocco è il suo hash criptografico, un'impronta digitale, creata eseguendo due volte l'hash di un block header attraverso

l'algoritmo SHA256. Il risultato di 32-byte di un hash è chiamato il block hask ma più specificatamente il block header hash ,perchè solo il block header è usato per calcolarlo. Per esempio, 000000000019d6689c085ae165831e93 è il block hash del primo blocco bitcoin che sia stato mai creato. Il block hash identifica un blocco unico e senza ambiguità e può essere indipendentemente derivato da ogni nodo da un semplice hashing del block header.

Nota che il block hash non è attualmente incluso all'interno della struttura dati dei blocchi, nè quando il blocco è trasmesso sul network nè

quando è memorizzato sullo storage persistente di un nodo come parte della blockchain. invece, l'hash dei blocchi è computato da ogni nodo che il blocco ha ricevuto dal network. Il block hash viene memorizzato in un separato database come una parte dei metadata dei blocchi, per facilitare l'indicizzazione e il più rapido recupero dei blocchi dal disco.

Una seconda via di identificazione di un blocco è dalla sua posizione nella blockchain, chiamata *block height*. Il primo blocco che sia mai stato creato ha un altezza di blocco 0 (zero) ed è la lo stesso blocco sarà precedentemente preso di riferimento dal seguente block hash

000000000019d6689c085ae165831e93
Un blocco può così essere identificato in due modi: da un riferimento del block hash o dal riferimento dell'altezza di blocco. Ogni susseguente blocco si aggiunge in cima al primo blocco che è in una posizione più alta nella blockchain, come scatole impilate una sopra l'altra. L'altezza del blocco di Gennaio 2014 era approssimativamente 278,000 e significa che ci sono stati 278,000 blocchi accatastati in cima al primo blocco creato a Gennaio 2009.

A differenza del block hash, l'altezza del blocco non è un identificatore unico. Nonostante un singolo blocco avrà sempre una specifica ed

invariabile altezza di blocco, l'inverso non è reale - l'altezza del blocco non è sempre identificata da un singolo blocco. Due o più blocchi possono avere la stessa altezza di blocco, competendo per la stessa posizione nella blockchain. Questo scenario è dibattuto nei dettagli della sezione I Fork della Blockchain. L'altezza del blocco non è anche una parte della struttura dati del blocco; non è memorizzata all'interno del blocco. Ogni nodo dinamicamente identifica una posizione di blocco (altezza) nella blockchain quando è ricevuto dal network bitcoin. L'altezza del blocco può essere anche memorizzata come metadata in un database indicizzato per

rapidi recuperi.

TIP	I l *block hash* di un blocco identifica sempre univocamente un singolo blocco. Un blocco ha inoltre sempre una *block height* specifica. Però non è sempre il caso che una block height specifica identifichi un singolo blocco. Al contrario, due o più blocchi possono competere per una singola posizione nella blockchain.

Il Genesis Block

Il primo blocco nella blockchain è chiamato genesis block ed è stato creato nel 2009. Esso è l'antenato comune a tutti i blocchi della blockchain, questo significa che se inizi a seguire ogni blocco e la chain all'indietro nel tempo, probabilmente risalirai al genesis block.

Ogni nodo inizia sempre con una blockchain di almeno un blocco perchè il genesis block è staticamente codificato all'interno del client software bitcoin, in modo che non possa essere alterato. Ogni nodo

"conosce" sempre l'hash del genesis block e la struttura, il tempo in cui è stato creato e almeno la singola transazione. Quindi, ogni nodo ha il punto iniziale della blockchain, una sicura "radice" da cui costruire una blockchain affidabile.

Visualizza il genesis block staticamente encoded nel client Bitcoin Core, in *chainparams.cpp*.

Il seguente hash identificativo appartiene al genesis block:

```
000000000019d6689c085ae165831e934ff7
```

Puoi cercare quel block hash in un qualsiasi sito block explorer, come blockchain.info, e troverai una pagina che descrive il contenuto di questo

blocco, con un'URL contenente quell'hash:

https://blockchain.info/block/000000000

Usando il client di riferimento Bitcoin Core da riga di comando:

```
$ bitcoin-cli getblock
000000000019d6689c085ae165831e934ff
```


```
{
  "hash":
"000000000019d6689c085ae165831e934ff
  "confirmations": 308321,
  "size": 285,
  "height": 0,
  "version": 1,
  "merkleroot":
"4a5e1e4baab89f3a32518a88c31bc87f618f
  "tx": [

"4a5e1e4baab89f3a32518a88c31bc87f618f
```

```
    ],
    "time" : 1231006505,
    "nonce" : 2083236893,
    "bits" : "1d00ffff",
    "difficulty" : 1.00000000,
    "nextblockhash" :
"000000000839a8e6886ab5951d76f4114754
}
```


Il genesis block contiene un messaggio nascosto in esso. L'input della transazione di coinbase contiene il testo " Dal Times del 03 gennaio 2009, Il Cancelliere sull'orlo del secondo bailout per le banche" Questo messaggio serviva per provare la prima data del blocco che era stato creato, riferendosi al quotidiano britannico *Il Times*. Serve anche come un promemoria ironico

dell'importanza di un sistema monetario indipendente, con il lancio di bitcoin avvenuto contemporaneamente ad una crisi monetaria mondiale senza precedenti. Il messaggio è stato incorporato nel primo blocco di Satoshi Nakamoto, creatore di bitcoin.

Collegando i Blocchi nella Blockchain

Il Bitcoin full node conserva una copia locale della blockchain, partendo dal genesis block. La copia locale della blockchain è costantemente aggiornata ai nuovi blocchi che vengono trovati e usati per estendere la chain. Quando un

nodo invia ai blocchi dal network, esso validerà questi blocchi e dopo li collegherà alla blockchain esistente. Per stabilire un collegamento, un nodo esaminerà l'inizio del block header e guarderà al precedente block hash.

Assumiamo, per esempio, che il nodo abbia 277,314 blocchi nella copia locale della blockchain. L'ultimo blocco del quale il nodo sa qualcosa è il blocco 277,314, con un hash dell'header del blocco di:

```
00000000000000027e7ba6fe7bad39faf3b5a
```

Il nodo bitcoin in seguito riceve un nuovo blocco dalla rete, che lo analizza come riportato qui:

```
{
```

```
    "size" : 43560,
    "version" : 2,
    "previousblockhash" :

"000000000000027e7ba6fe7bad39faf3b5
    "merkleroot" :

"5e049f4030e0ab2debb92378f53c0a6e0954
    "time" : 1388185038,
    "difficulty" : 1180923195.25802612,
    "nonce" : 4215469401,
    "tx" : [

"257e7497fb8bc68421eb2c7b699dbab2348

 #[... molte altre transazioni omesse ...]


"05cfd38f6ae6aa83674cc99e4d75a1458c16
    ]
}
```

Osservando questo nuovo blocco, possiamo notare che il nodo trova il campo previousblockhash, che contiene l'hash del blocco genitore. È un hash conosciuto al nodo, come l'ultimo blocco della chain all'altezza (block height) 277,314. Quindi questo nuovo blocco è un figlio dell'ultimo blocco della chain e estende la blockchain esistente. Il nodo aggiunge questo nuovo blocco alla fine della catena (chain), rendendo la blockchain più lunga con una nuova height di 277,315. Blocchi collegati in una catena facendo riferimento al precedente hash dell'intestazione del blocco mostra la chain di tre blocchi, collegati da referenze nel campo

previousblockhash.

I Merkle Tree

Ogni blocco della blockchain di bitcoin contiene un sommario di tutte le transazione nel blocco, usando un *merkle tree*.

U n *merkle tree*, conosciuto anche come un *binary hash tree*, è una struttura dati usata per indicizzare efficientemente e verificare l'integrità di un grande gruppo di dati. Merkle trees sono binari trees contenenti gli hashes crittografici. Il termine "tree" è usato nella scienza informatica per descrivere una ramificazione della struttura dati, ma questi trees sono visualizzati solitamente sottosopra con

la "radice" in alto e il "leaves" in fondo al diagramma, come vedrete negli esempi che seguono.

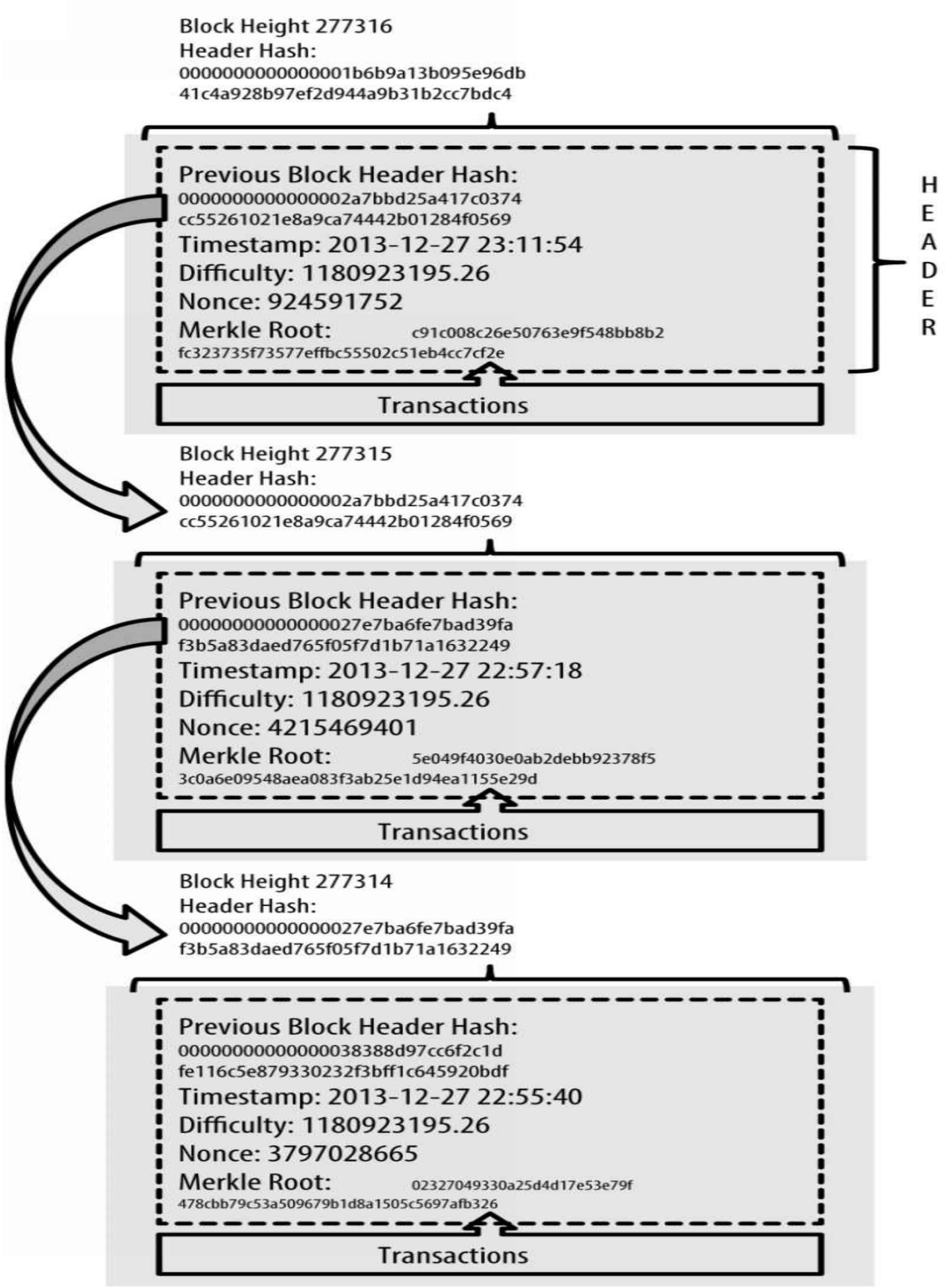

Block Height 277316
Header Hash:
000000000000001b6b9a13b095e96db
41c4a928b97ef2d944a9b31b2cc7bdc4
Previous Block Header Hash:
0000000000000002a7bbd25a417c0374
cc55261021e8a9ca74442b01284f0569
Timestamp: 2013-12-27 23:11:54
Difficulty: 1180923195.26
Nonce: 924591752
Merkle Root: c91c008c26e50763e9f548bb8b2
fc323735f73577effbc55502c51eb4cc7cf2e
Transactions
HEADER
Block Height 277315
Header Hash:
0000000000000002a7bbd25a417c0374
cc55261021e8a9ca74442b01284f0569
Previous Block Header Hash:
0000000000000027e7ba6fe7bad39fa
f3b5a83daed765f05f7d1b71a1632249
Timestamp: 2013-12-27 22:57:18
Difficulty: 1180923195.26
Nonce: 4215469401
Merkle Root: 5e049f4030e0ab2debb92378f5
3c0a6e09548aea083f3ab25e1d94ea1155e29d
Transactions
Block Height 277314
Header Hash:
0000000000000027e7ba6fe7bad39fa
f3b5a83daed765f05f7d1b71a1632249
Previous Block Header Hash:
000000000000000038388d97cc6f2c1d
fe116c5e879330232f3bff1c645920bdf
Timestamp: 2013-12-27 22:55:40
Difficulty: 1180923195.26
Nonce: 3797028665
Merkle Root: 02327049330a25d4d17e53e79f
478cbb79c53a509679b1d8a1505c5697afb326
Transactions

Figura 57. Blocchi collegati in una catena facendo riferimento al precedente hash dell'intestazione del blocco

I merkle tree sono usati in bitcoin per indicizzare tutte le transazioni in un blocco, producendo in sostanza un'impronta digitale dell'intero set di transazioni,provvedendo ad un efficientissimo processo di verifica se una transazione è inclusa nel blocco. Un Il Merkle tree è costruito dal continuo hashing delle coppie di nodi finché c'è un solo hash, chiamato il *root* oppure *merkle root*. L'hash

dell'algoritmo crittografico usato nel merkle tree di bitcoin è SHA256 applicato due volte, conosciuto anche come doppio-SHA256.

Quando N elementi di dati sono sottoposti ad hashing e indicizzati nel merke tree, si possono verificare se ogni singolo data element è incluso nel tree con al massimo 2*log~2~(N) calcoli, ottenendo così un efficientissima struttura dati.

Il merkle tree è costruito dal basso verso l'alto. Nel seguente esempio, partiamo con quattro transazioni, A, B, C e D, che formano le *foglie* del Merkle tree, come mostrato in Calcolando i nodi in un merkle tree. Le transazioni non sono salvate nel

merkle tree; invece, i loro dati sono sottoposti ad hash e l'hash risultante è salvato in ogni nodo figlia come HA, HB, HC, e HD:

$$H_A = \text{SHA256}(\text{SHA256}(\text{Transactior}$$

A))

Le consecutive coppie di nodi leaf sono poi indicizzate in un nodo padre che concatena i due hash ed effettua hashing in combinazione. Per esempio, per costruire un nodo padre H_{AB}, i due 32-byte degli hash dei bambini sono concatenati per creare una stringa di 64-bytes. Questa stringa è poi doppiamente sottoposta ad hash per

produrre l'hash del nodo padre:

$$H_{AB} = SHA256(SHA256(H_A + H_B))$$

Il processo continua finché c'è solo un nodo in cima, il nodo chiamato Merkle root. Questo hash da 32-byte è memorizzato in un block header e indicizza tutti i dati in tutte di tutte le quattro transazioni. Calcolando i nodi in un merkle tree mostra come viene calcolata la radice mediante gli hash a coppie dei nodi.

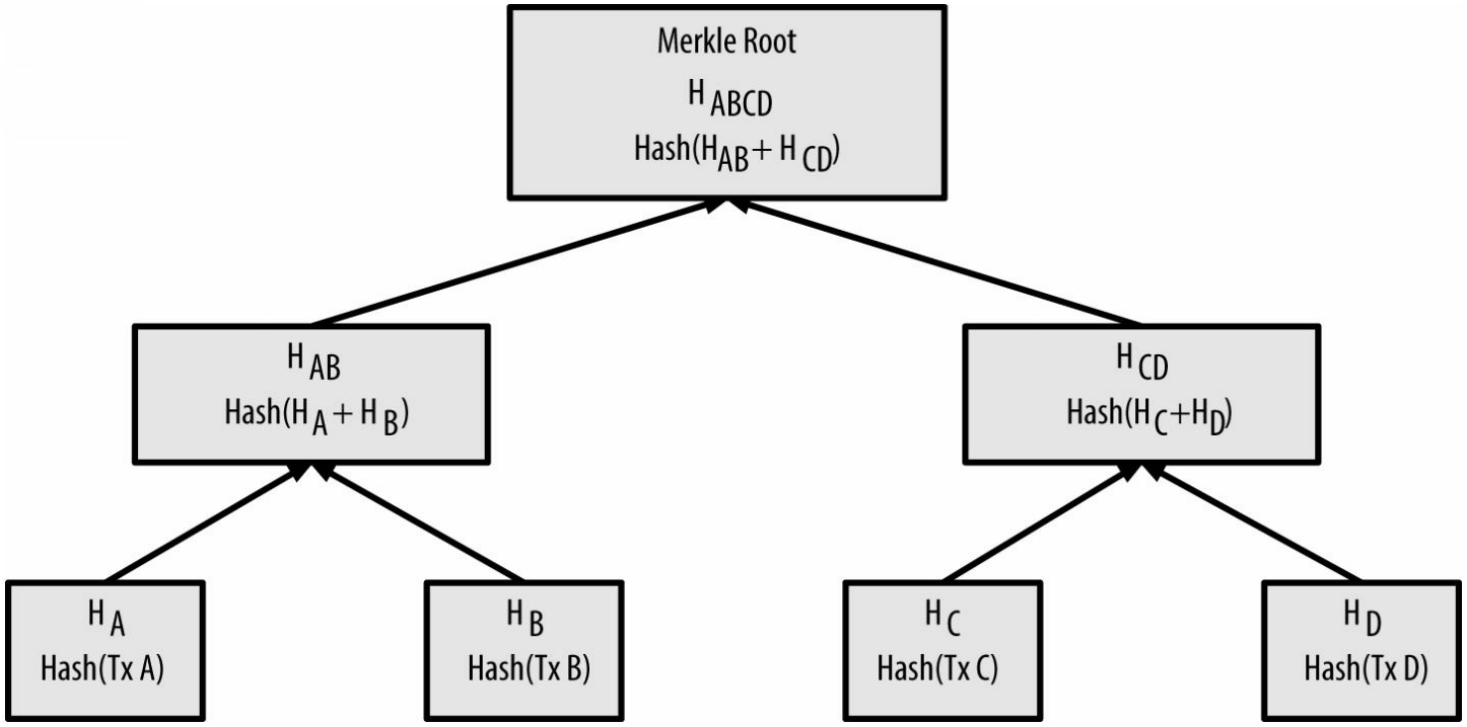


Figura 58. Calcolando i nodi in un merkle tree

Visto che il merkle tree è un albero binario, necessita di un numero pari di nodi foglia. Se il numero di transazioni da sintetizzare è un numero pari, l'ultimo hash di transazione sarà duplicato per creare un numero pari di nodi foglia, questo tipo di alberi sono conosciuti come *alberi bilanciati*. Il

tutto è mostrato in Duplicando un elemento (data element) ottiene un numero di elementi pari, dove la transazione C è duplicata.

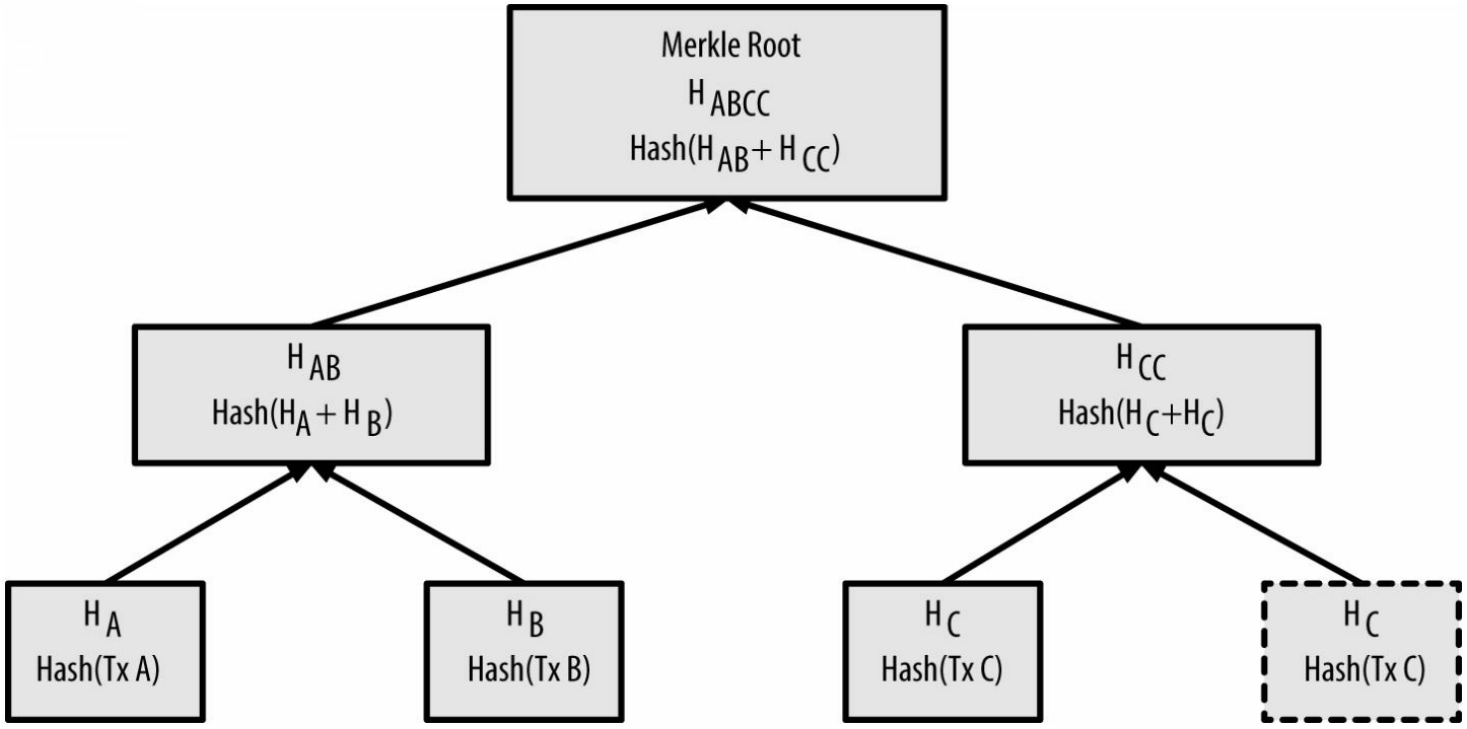


Figura 59. Duplicando un elemento (data element) ottiene un numero di elementi pari

Lo stesso metodo per costruire un tree da quattro transazioni può essere generalizzato per costruire tree di ogni dimensione. In bitcoin è comune avere diverse centinai di migliaia transazioni in un singolo blocco, che sarà indicizzato nel medesimo modo, producendo i 32 byte di dati come un singolo merkle root. Nel Un merkle tree che riassume tanti data element, possiamo vedere un tree costituito da 16 transazioni. Nota che sebbene il root sembra più grande rispetto ai nodi nel diagramma, esso è esattamente della stessa dimensione, cioè 32 byte. Se c'è una transazione o un centinaio di migliaia di transazioni nel blocco, il merkle root li indicizza sempre in 32

byte.

Per provare che una specifica transazione è inclusa nel blocco, un nodo ha bisogno solo di produrre degli hash di 32-byte log~2~(N) , costituendo un *percorso di autenticazione* o percorso merke connettendo la specifica transazione alla radice del tree. Ciò è particolarmente importante in quanto aumenta il numero di transazioni, poiché il logaritmo in base 2 del numero di transazioni aumenta molto più lentamente. Questo consente ai nodi bitcoin di produrre efficientemente il percorso di 10 o 12 hashes (320-384bytes), i quali possono provvedere alla prova della

singola transazione da più di un migliaio di transazioni in un blocco di megabyte-size.

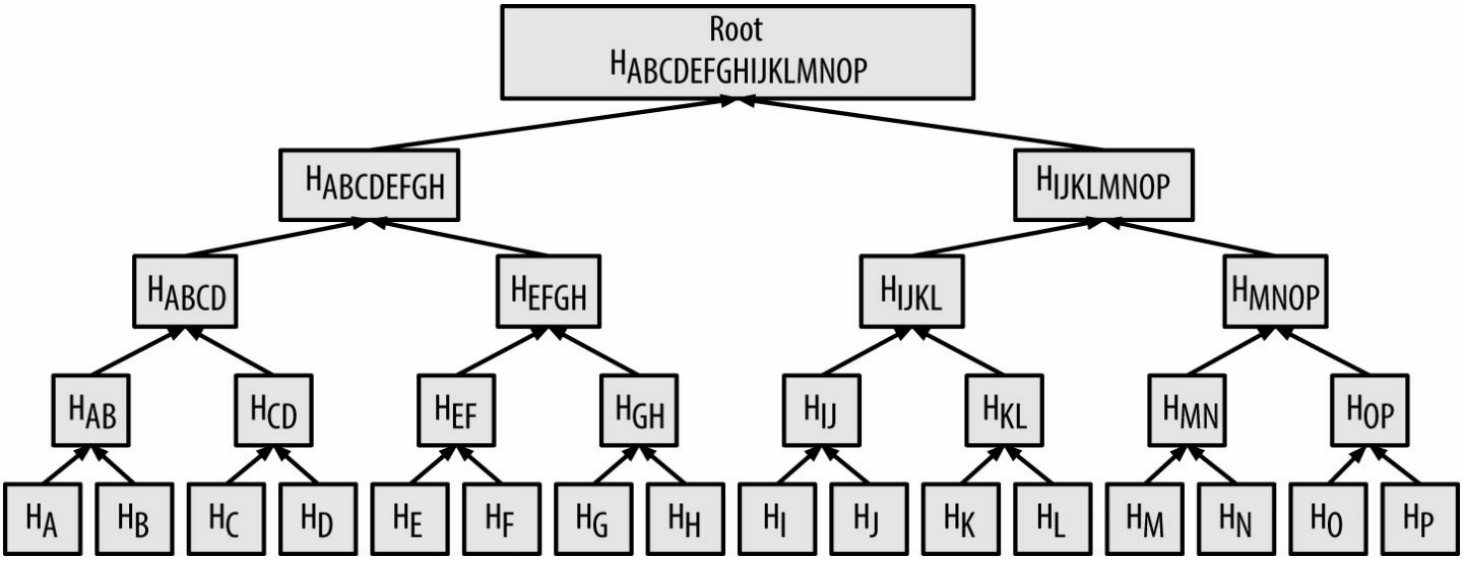


Figura 60. Un merkle tree che riassume tanti data element

In Un merkle path usato per provare l'inclusione di un data element, un nodo può dimostrare che una transazione K è inclusa nel blocco con la produzione di un percorso Merkle

che è solo 32 byte di lunghezza dell'hash (128 byte totali). Il percorso si compone di quattro hash (indicato con uno sfondo ombreggiato in Un merkle path usato per provare l'inclusione di un data element) HL, HIJ, HMNOP, e HABCDEFGH. Con questi quattro hash forniti come un percorso di autenticazione, ogni nodo può dimostrare che HK (con uno sfondo nero nella parte inferiore del diagramma) è incluso nella radice Merkle calcolando quattro ulteriori pair-wise hash HKL, HIJKL, HIJKLMNOP, e la radice del Merkle tree root (delineato in una linea tratteggiata nel diagramma).

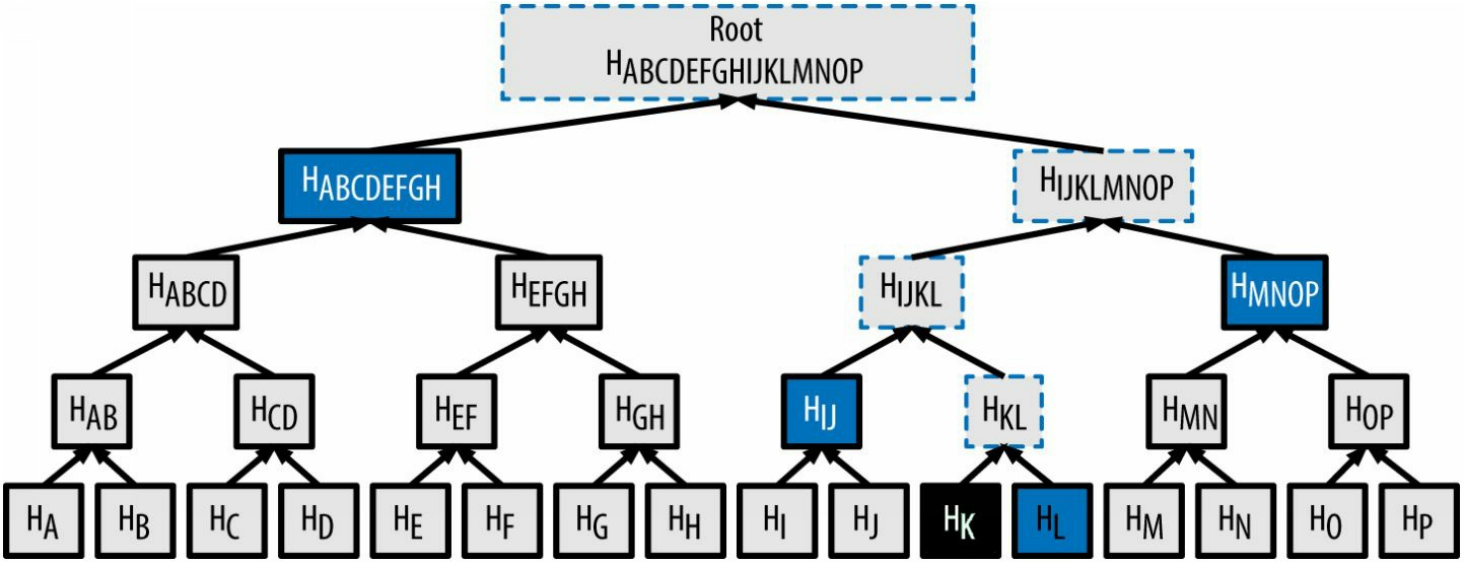


Figura 61. Un merkle path usato per provare l'inclusione di un data element

Il codice nel Costruendo un merkle tree dimostra il processo di creazione di un merkle tree dagli hash del nodo-foglia fino alla radice, usando la libreria libbitcoin per qualche funzione helper.

Esempio 18. Costruendo un

merkle tree

```
#include <bitcoin/bitcoin.hpp>
```

```
bc::hash_digest
create_merkle(bc::hash_list& merkle)
{
    // Stop if hash list is empty.
    if (merkle.empty())
        return bc::null_hash;
    else if (merkle.size() == 1)
        return merkle[0];

    // While there is more than 1 hash in
the list, keep looping...
    while (merkle.size() > 1)
    {
        // If number of hashes is odd,
duplicate last hash in the list.
```

```
        if (merkle.size() % 2 != 0)

merkle.push_back(merkle.back());
        // List size is now even.
        assert(merkle.size() % 2 == 0);

        // New hash list.
        bc::hash_list new_merkle;
        // Loop through hashes 2 at a time.
        for (auto it = merkle.begin(); it !=
merkle.end(); it += 2)
        {
            // Join both current hashes
together (concatenate).
            bc::data_chunk
concat_data(bc::hash_size * 2);
            auto concat = bc::serializer<

decltype(concat_data.begin())>
(concat_data.begin());
            concat.write_hash(*it);
```

```
            concat.write_hash(*(it + 1));
            // Hash both of the hashes.
            bc::hash_digest new_root =
bc::bitcoin_hash(concat_data);
            // Add this to the new list.

new_merkle.push_back(new_root);
        }
        // This is the new list.
        merkle = new_merkle;

        // DEBUG output ----------------
----------------
        std::cout << "Current merkle hash
list:" << std::endl;
        for (const auto& hash: merkle)
            std::cout << "  " <<
bc::encode_base16(hash) << std::endl;
        std::cout << std::endl;
        // ---------------------------------
------------
```

```
    }
    // Finally we end up with a single
item.
    return merkle[0];
}

int main()
{
    // Replace these hashes with ones
from a block to reproduce the same
merkle root.
    bc::hash_list tx_hashes{{

bc::hash_literal("0000000000000000000

bc::hash_literal("0000000000000000000

bc::hash_literal("0000000000000000000
    }};
    const bc::hash_digest merkle_root =
create_merkle(tx_hashes);
```

```
    std::cout << "Result: " <<
bc::encode_base16(merkle_root) <<
std::endl;
    return 0;
}
```

Compilando e eseguendo il codice di esempio dell'esempio merkle mostra il risultato della compilazione e esecuzione del codice merkle.

Esempio 19. Compilando e eseguendo il codice di esempio dell'esempio merkle

```
$ # Compila il codice in merkle.cpp

$ g++ -o merkle merkle.cpp $(pkg-
```

```
config --cflags --libs libbitcoin)
$ # Esegui l'eseguibile merkle
$ ./merkle
Lista di hash merkle corrente:

32650049a0418e4380db0af81788635d8b

30861db96905c8dc8b99398ca1cd5bd5b84

Lista di hash merkle corrente:

d47780c084bad3830bcdaf6eace035e4c6cb

Risultato:
d47780c084bad3830bcdaf6eace035e4c6cb
```

L'efficienza dei merkle tree diventa ovvia man mano che la scala aumenta. Efficenza del merkle tree mostra la quantità di dati necessari che devono essere scambiati come merkle path per

provare che la transazione sia facente parte di un blocco.

Tabella 25. Efficenza del merkle tre

Numero di transazioni	**Dimensione approssimativa del blocco**	**Dim del (has**
16 transazioni	4 kilobyte	4 ha
512 transazioni	128 kilobyte	9 ha
2048 transazioni	512 kilobyte	11 h
65,535 transazioni	16 megabyte	16 h

Come si può vedere dalla tabella, mentre la dimensione del blocco aumenta rapidamente, da 4 KB con 16 transazioni a una dimensione di blocco di 16 MB per adattarsi 65.535 transazioni, il percorso Merkle necessario per provare l'inserimento della transazione aumenta molto più lentamente, da 128 bytes a soli 512bytes. Con il merke trees, un nodo può scaricare soli i block headers (80 bytes per blocco) e può essere ancora in grado di identificare l'inserimento della transazione nel blocco recuperando una parte del percorso merkle da un full node, senza memorizzare o trasmettere la

stragrande maggioranza della blockchain, che potrebbe essere di diversi gigabyte di dimensione. I nodi che non contengono una blockchain intera, chiamati simplified payment verification (SPV nodes), usano il merkle path per verificare le transazioni senza scaricare gli interi blocchi.

Merkle Tree e Simplified Payment Verification (SPV)

I merkle tree sono usati estensivamente dai nodi SPV (o VSP in italiano - Verifica Semplificata del Pagamento, ndt). I nodi SPV non hanno tutte le

transazioni e non scaricano blocchi completi, ma solo header dei blocchi. Per verificare che una transazione sia inclusa in un blocco, senza dover scaricare tutte le transazioni nel blocco, essi utilizzano un percorso di autenticazione, chiamato anche merkle path.

Consideriamo, per esempio, un nodo SPV che è interessato ai pagamenti in entrata all'indirizzo contenuto nel suo wallet. Il nodo SPV stabilirà un filtro bloom (vedi Filtri di Bloom (Bloom Filter)) sulle sue connessioni ai peer per limitare le transazioni ricevute ai soli che contengono indirizzi di interesse. Quando un peer vede una transazione che corrisponde al filtro

bloom, invierà quel blocco utilizzando un merkleblock messaggio. Il merkleblock messaggio che contiene il block header, così come un percorso Merkle che collega la transazione di interesse alla radice Merkle nel blocco. Il nodo SPV può utilizzare questo percorso Merkle per collegare la transazione al blocco e verificare che l'operazione è inclusa nel blocco. Il nodo SPV utilizza anche il block header per collegare il blocco al resto della blockchain. La combinazione di questi due link, tra la transazione e il blocco, e tra il blocco e blockchain, dimostra che l'operazione è registrata nella blockchain. Tutto sommato, il nodo SPV avrà ricevuto meno di un

kilobyte di dati per l'header di blocco e il merkle path, una quantità di dati che è più di mille volte inferiore a un blocco completo (circa 1 megabyte al momento).

Le Blockchain di Test di Bitcoin

Potresti essere sorpreso di apprendere che esiste più di una blockchain bitcoin. La blockchain di bitcoin "principale", quella creata da Satoshi Nakamoto il 3 gennaio 2009, con il blocco di genesi che abbiamo studiato in questo capitolo, è chiamata *mainnet*. Esistono altre blockchain di bitcoin che vengono utilizzate a scopo

di test: in questo momento esistono *testnet*, *segnet* e *regtest*. Diamo un'occhiata ad ognuna di queste.

Testnet—Campo da gioco di Bitcoin

Testnet è il nome della blockchain, della rete e della valuta utilizzati a scopo di test. Il testnet è una rete P2P live con funzionalità complete, con portafogli, bitcoin di test (monete testnet), mining e tutte le altre funzionalità di mainnet. Ci sono davvero solo due differenze: le monete testnet sono inutili e la difficoltà di estrazione deve essere abbastanza bassa da permettere a chiunque di estrarre le monete testnet in modo

relativamente semplice (mantenendole inutili).

Qualsiasi sviluppo di software destinato alla produzione su mainnet di bitcoin deve essere prima testato su testnet con monete di prova. Ciò protegge gli sviluppatori dal perdere bitcoin reali causa possibili bug e la rete da comportamenti non voluti causati sempre da possibili bug.

Mantenere le monete prive di utilità e permettere un'estrazione veloce, tuttavia, non è semplice. Nonostante le richieste di utilizzare equipaggiamento low-end da parte degli sviluppatori, alcune persone usano attrezzature da mining avanzate (GPU e ASIC) per estrarre bitcoin su testnet. Ciò aumenta

la difficoltà rendendo impossibile minare con una CPU, e di conseguenza rende altrettanto difficile ottenere monete di prova e le persone hanno iniziato a darne un valore, rendendole non realmente inutili e prive di valore. Di conseguenza, di tanto in tanto, il testnet deve essere scartato e riavviato da un nuovo blocco di genesi, ripristinando la difficoltà.

L'attuale testnet è chiamato *testnet3*, la terza iterazione di testnet, riavviata a febbraio 2011 per ripristinare la difficoltà dal precedente testnet.

Tieni presente che testnet3 è una blockchain di grandi dimensioni, che ad inizio 2017 pesa oltre 20 GB. Ci vuole circa un giorno per sincronizzare

completamente la blockchain e sfruttare le risorse del tuo computer per minare. Non è proprio come mainnet, ma nemmeno così "leggera". Un buon modo per eseguire un nodo testnet è attraverso un'immagine caricata su macchina virtuale (ad es. VirtualBox, Docker, Cloud Server, ecc.) dedicata a tale scopo.

Utilizzare testnet

Bitcoin Core, come quasi tutti gli altri software bitcoin, supporta pienamente il funzionamento su testnet oltre che su mainnet. Tutte le funzioni di Bitcoin Core funzionano su testnet, incluso il portafoglio, l'estrazione di monete testnet e la sincronizzazione di un nodo

testnet completo.

Per avviare Bitcoin Core su testnet anziché su mainnet, devi utilizzare l'opzione testnet:

```
$ bitcoind -testnet
```

Nei log dovresti vedere che bitcoind sta creando una nuova blockchain nella sottodirectory testnet3 della directory bitcoind predefinita:

```
bitcoind: Using data directory /home/username/.bitcoin/testnet3
```

Per connetterti a bitcoind, utilizza lo strumento da riga di comando bitcoin-cli, ricorda che è necessario passare alla modalità testnet:

```
$ bitcoin-cli -testnet getblockchaininfo
{
```

```
  "chain": "test",
  "blocks": 1088,
  "headers": 139999,
  "bestblockhash":
"0000000063d29909d475a1c4ba26da64b36
  "difficulty": 1,
  "mediantime": 1337966158,
  "verificationprogress":
0.0016440659140997 59,
  "chainwork":
"0000000000000000000000000000000(
  "pruned": false,
  "softforks": [

  [...]
```

Puoi eseguire su testnet3 altre implementazioni a livello di full node, come btcd (scritto in Go) e bcoin (scritto in JavaScript), per sperimentare e apprendere in altri

linguaggi e framework di programmazione.

All'inizio del 2017, testnet3 supporta tutte le funzionalità di mainnet, tra cui Segregated Witness (vedi Segregated Witness). Pertanto, testnet3 può anche essere utilizzato per testare le funzionalità di SegWit.

Segnet—La Testnet di Segregated Witness

Nel 2016 è stata lanciata una testnet speciale per aiutare lo sviluppo e il test di Segregated Witness (conosciuto come segwit; vedi Segregated Witness). Questa blockchain di prova è chiamata segnet e vi si può partecipare eseguendo una versione

speciale (ramo) di Bitcoin Core.

Poiché segwit è stato aggiunto a testnet3, non è più necessario utilizzare segnet per il test delle funzionalità segwit.

In futuro è probabile che vedremo altre blockchain testnet specificamente progettate per testare una singola funzione o una modifica architettonica importante, come segnet.

Regtest—La Blockchain Locale

Regtest, che sta per "Regression Testing", è una funzionalità di Bitcoin Core che consente di creare una blockchain locale a scopo di test. A differenza di testnet3, che è una

blockchain di test pubblica e condivisa, le blockchain regtest sono concepite per essere eseguite su sistemi chiusi per test locali. Permette di lanciare una blockchain regtest da zero, creando un blocco genesis locale. È possibile inoltre aggiungere altri nodi alla rete o eseguirla con un solo nodo per testare unicamente il software Bitcoin Core.

Per avviare Bitcoin Core in modalità regtest, utilizza il flag regtest:

```
$ bitcoind -regtest
```

Proprio come con testnet, Bitcoin Core inizializzerà una nuova blockchain sotto la sottodirectory regtest della directory predefinita bitcoind:

```
bitcoind: Using data directory
/home/username/.bitcoin/regtest
```

Per utilizzare lo strumento da riga di comando, è necessario specificare anche il flag regtest. Proviamo il comando getblockchaininfo per ispezionare la blockchain regtest:

```
$ bitcoin-cli -regtest getblockchaininfo
{
 "chain": "regtest",
 "blocks": 0,
 "headers": 0,
 "bestblockhash":
"0f9188f13cb7b2c71f2a335e3a4fc328bf5be
 "difficulty": 4.656542373906925e-10,
 "mediantime": 1296688602,
 "verificationprogress": 1,
 "chainwork":
"000000000000000000000000000000000
 "pruned": false,
```

```
  [...]
```

Come puoi vedere, non ci sono ancora blocchi. Proviamo a minarne un paio (500 blocchi) e diamo un'occhiata alla ricompensa:

```
$ bitcoin-cli -regtest generate 500
[

"7afed70259f22c2bf11e406cb12ed5c0657b

"1aca2f154a80a9863a9aac4c72047a6d3f385

"4334ecf6fb022f30fbd764c3ee778fabbd53b

"5f951d34065efeaf64e54e91d00b260294fc

"43744b5e77c1dfece9d05ab5f0e6796ebe62
  [...]

"6c31585a48d4fc2b3fd25521f4515b18aefb3
```

```
]
```

Bastano pochi secondi per estrarre tutti questi blocchi, il che rende estremamente facile effettuare il test. Se controlli il saldo del tuo portafoglio, vedrai che hai guadagnato un premio per i primi 400 blocchi (per i premi coinbase devi attendere 100 blocchi prima di poterli spendere):

```
$ bitcoin-cli -regtest getbalance
12462.50000000
```

Utilizzare le Blockchain di Testing per lo Sviluppo

Le varie blockchain di Bitcoin (regtest, segnet, testnet3, mainnet)

offrono una gamma di ambienti di test per lo sviluppo di bitcoin. Utilizza sempre le blockchain di prova sia che tu stia sviluppando per Bitcoin Core, sia che si tratti di un altro client full node, un'applicazione come un wallet, un'exchange, un sito di e-commerce o anche lo sviluppo di nuovi smart contract e script complessi.

È possibile utilizzare i blocchi di prova per stabilire una pipeline di sviluppo. Prova il tuo codice localmente su regtest mentre sviluppi. Una volta che sei pronto a provarlo su una rete pubblica, passa a testnet per esporre il tuo codice a un ambiente più dinamico con più varietà di codice e applicazioni. Infine, una volta che sei

certo che il codice funziona come previsto, passa a mainnet per distribuirlo in produzione. Quando apporti modifiche, miglioramenti, correzioni di errori, ecc., avvia di nuovo la pipeline, distribuendo ogni modifica prima su regtest, poi su testnet e infine in produzione.

Il Mining e Il Consenso

Introduzione

La parola "mining" è in qualche modo ingannevole. Evocando l'estrazione di metalli preziosi, concentra la nostra attenzione nella ricompensa del mining, i nuovi bitcoin in ogni nuovo blocco. Anche se il mining è incentivato da questa ricompensa, il primo scopo del mining non è la ricompensa o la generazione di nuova moneta. Se vedi il mining solo come il processo nel quale la moneta è creata, ne stai confondendo i mezzi (gli

incentivi) come se fossero il traguardo del processo. Il mining è il meccanismo che sta alla base della di compensazione decentralizzata, attraverso la quale le transazioni vengono convalidate e liquidate. Il mining è l'invenzione che rende bitcoin speciale, un meccanismo di sicurezza che è la base del contante digitale peer-to-peer.

Il mining rende protetto il sistema bitcoin e abilita l'emergenza di un consenso su tutta la rete *senza il bisogno di un'autorità centrale.* La ricompensa delle nuove monete coniate e delle commissioni di transazione è uno schema di incentivi che allinea le azioni dei minatori con

la sicurezza della rete, implementando contemporaneamente l'offerta monetaria.

TIP	Lo scopo del mining non è la creazione di nuovi bitcoin. Questo è un sistema di incentivi. Il mining è il meccanismo con cui la *sicurezza di bitcoin* è *decentralizzata*.

I miner validano nuove transazioni e le registrano sulla ledger globale (un libro mastro globale). Un nuovo blocco, contenente le transazioni che sono avvenute dopo la scoperta

dell'ultimo blocco, è minato in media ogni 10 minuti, e inoltre accodando queste transazioni alla blockchain. Le transazioni che sono diventate parte di un blocco e aggiunte alla blockchain sono considerate "confermate", questo permette ai nuovi proprietari dei bitcoin di spendere i bitcoin ricevuti nelle suddette transazioni.

I miner ricevono due tipi di ricompense per il lavoro di mining: i nuovi bitcoin creati ogni nuovo blocco, e le commissioni (fee) di transazione da tutte le transazioni incluse nel blocco. Per ottenere questa ricompensa, i miner devono competere nel risolvere un difficile problema matematico basato su un algoritmo

crittografico di hashing. La soluzione al problema, chiamato proof of work, è incluso nel nuovo blocco e agisce come prova che il miner ha impiegato uno sforzo computazionale adeguato. La gara a risolvere l'algoritmo di proof-of-work per ottenere la ricompensa e il diritto di registrare le transazioni nella blockchain è alla base del modello di sicurezza di bitcoin.

Il processo si chiama mining perché la ricompensa (nuova generazione di monete) è progettata per simulare rendimenti decrescenti, proprio come l'estrazione di metalli preziosi. L'offerta di moneta di Bitcoin viene creata attraverso il mining, in modo

simile a come una banca centrale emette nuovi soldi stampando banconote. La quantità massima di bitcoin appena creata che un minatore può aggiungere a un blocco diminuisce approssimativamente ogni quattro anni (o precisamente ogni 210.000 blocchi). È iniziato con 50 bitcoin per blocco nel gennaio del 2009 e si è dimezzato a 25 bitcoin per blocco nel novembre del 2012. Si è nuovamente dimezzato a 12,5 bitcoin nel luglio 2016. Sulla base di questa formula, i premi del mining di bitcoin diminuiscono esponenzialmente fino a circa l'anno 2140, anno in cui tutti i bitcoin (20.99999998 milioni) saranno stati emessi. Dopo il 2140, non verrà

emesso alcun nuovo bitcoin.

Il processo di generazione di nuova moneta è chiamato mining perché la ricompensa è fatta per simulare il rendimento decrescente, con lo stesso funzionamento dell'estrazione (mining - ndt) di metalli preziosi. La massa monetaria di Bitcoin è creata attraverso appunto questa estrazione, il mining, nello stesso modo in cui una banca centrale emette nuova moneta con la stampa delle banconote. La somma di nuovi bitcoin creati che un miner può aggiungere in un blocco decresce approssimativamente ogni quattro anni (o precisamente ogni 210.000 blocchi). È iniziata con 50 bitcoin per blocco nel Gennaio del

2009 e si è dimezzata a 25 bitcoin per blocco nel Novembre del 2012 e nuovamente a 12.5 bitcoin nel 2016. Basato su questa formula, la ricompensa del mining di bitcoin decresce esponenzialmente fino approssimativamente all'anno 2140, quando tutti i bicoin (20,99999998 milioni) saranno stati emessi. Dopo il 2140, nessun nuovo bicoin verrà emesso.

I miner bitcoin inoltre guadagnano un compenso dalle transazioni. Ogni transazione può includere una fee di transazione, nella forma di un surplus di bitcoin tra gli input e gli output della transazione. Il miner bitcoin vincente può "tenere il resto" della

transazione inclusa nel blocco vincente. Ad oggi, le fee rappresentano 0.5% o meno delle entrate di un miner bitcoin, la maggior parte derivanti dal conio dei nuovi bitcoin. In ogni caso, visto che la ricompensa diminuisce nel tempo e il numero di transazioni per blocco aumenta, una maggiore proporzione degli introiti ottenuti con il mining di bitcoin proverrà dalle fee. Dopo il 2140, tutti i guadagni dei miner bitcoin saranno nella forma di fee di transazione.

In questo capitolo, per iniziare esamineremo il mining come sistema di fornitura monetaria e poi osserveremo la funzione più importante del mining: il meccanismo

di consenso decentralizzato emergente che è alla base della sicurezza di bitcoin.

Per comprendere il mining e il consenso, seguiremo la transazione di Alice così come viene ricevuta e aggiunta a un blocco dall'attrezzatura mineraria di Jing. Seguiremo il blocco mentre viene estratto, aggiunto alla blockchain e accettato dalla rete bitcoin attraverso il processo di consenso emergente.

L'economia di Bitcoin e la Creazione di Valuta

I bitcoin sono "coniati" durante la creazione di ogni blocco con un andamento prefissato e in diminuzione.

Ogni blocco, generato in media ogni 10 minuti, contiene bitcoin completamente nuovi, creati dal niente. Ogni 210.000 blocchi o approssimativamente ogni quattro anni, il tasso di emissione di moneta è abbassato del 50%. Per i primi quattro anni di vita del network, ogni blocco conteneva 50 nuovi bitcoin.

Nel Novembre 2012, il nuovo di tasso di emissione di bitcoin è stato diminuito a 25 bitcoin per blocco. Nel Luglio 2016 è stato diminuito a 12.5 bitcoin per blocco. Si dividerà nuovamente a 6.25 bitcoin al blocco 630.000, che verrà estratto nel 2020. Il tasso di creazione di nuove monete scende esponenzialmente in questo

modo in 32 "dimezzamenti" fino al blocco 6,720,000 (che verrà minato approssimativamente nell'anno 2137), quando arriverà ad avere il valore minimo di unità di 1 satoshi. Finalmente, dopo 6.93 milioni di blocchi, approssimativamente nel 2140, circa 2.099.999.997.690.000 satoshi, o circa 21 milioni di bitcoin, saranno stati emessi. In seguito, i blocchi non conterranno più nuovi bitcoin, e i miner saranno ricompensati solamente attraverso le commissioni (fee) di transazione. La riserva monetaria di bitcoin nel tempo basata su di un ritmo di emissione geometricamente decrescente mostra il totale dei bitcoin in circolazione del

tempo, e il decrescente andamento d'emissione della moneta.

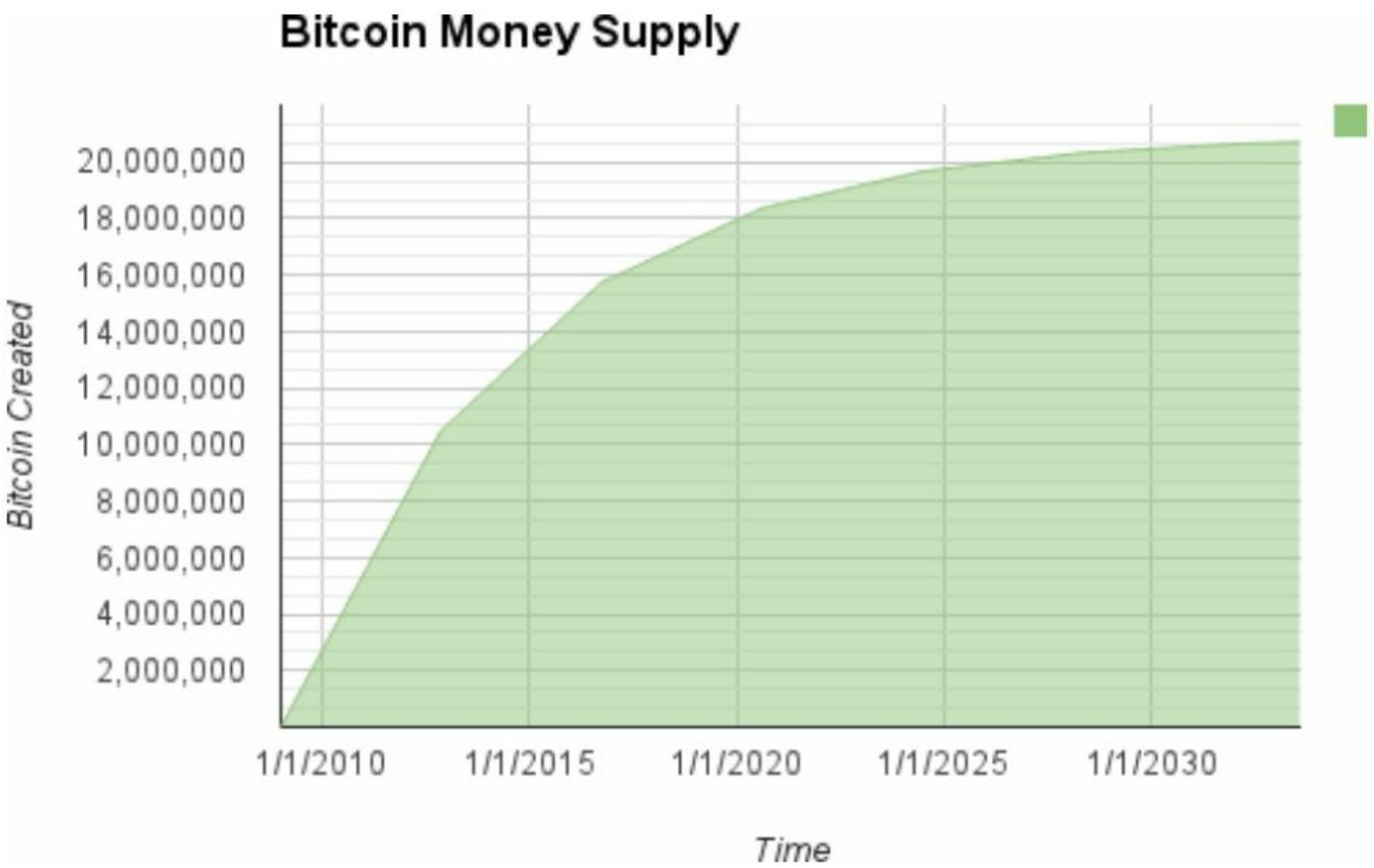


Figura 62. La riserva monetaria di bitcoin nel tempo basata su di un ritmo di emissione geometricamente decrescente

NOTA

Il numero
massimo di
monete estratto è
il *limite massimo*
dei possibil
premi minabil
per bitcoin. In
pratica, un
minatore può
intenzionalmente
estrarre un
blocco prendendo
meno della piena

	ricompensa (che gli spetterebbe). Tali blocchi sono già stati estratti e altri potrebbero essere estratti in futuro, determinando una minore emissione totale della valuta.

Nel codice di esempio in Uno script per calcolare quanti bitcoin totali saranno emessi, calcoliamo il numero

totale di bitcoin che saranno mai emessi.

Esempio 20. Uno script per calcolare quanti bitcoin totali saranno emessi

```
# Original block reward for miners
was 50 BTC
```

```
start_block_reward = 50
# 210000 is around every 4 years with a
10 minute block interval
reward_interval = 210000


def max_money():
    # 50 BTC = 50 0000 0000 Satoshis
    current_reward = 50 * 10**8
    total = 0
```

```
    while current_reward > 0:
        total += reward_interval *
current_reward
        current_reward /= 2
    return total


print("Total BTC to ever be created:",
max_money(), "Satoshis")
```

Eseguendo lo script max_money.py mostra l'output prodotto eseguendo questo script.

Esempio 21. Eseguendo lo script max_money.py

```
$ python max_money.py
```

```
I BTC totali che saranno mai creati:
2099999997690000 Satoshis
```

L'emissione limitata e decrescente crea un approvvigionamento monetario fisso che resiste all'inflazione. A differenza di una moneta fiat, che può essere stampata in numeri infiniti da una banca centrale, il bitcoin non può mai essere gonfiato con la stampa (ndt di nuovi bitcoin).

Moneta Deflazionaria

La più importante e dibattuta conseguenza di una emissione monetaria fissa e in diminuzione è il fatto che la valuta tenderà ad essere intrinsecamente *deflazionaria*. La

deflazione è il fenomeno di apprezzamento di del valore dato da un disaccoppiamento tra la domanda e l'offerta che aumenta il valore (e il valore di scambio) di una moneta. L'opposto dell'inflazione, la deflazione dei prezzi sta a significare che la valuta ha più potenza d'acquisto nel tempo.

Molti economisti sostengono che un'economia deflazionaria è un disastro che dovrebbe essere evitato ad ogni costo. Questo perché in un periodo di deflazione rapida, le persone tenono a mettere da parte la moneta invece di spenderla, sperando che i prezzi cadranno.

Questo fenomeno si è verificato durante il "Decennio Perduto" del Giappone, quando un totale collasso della domanda ha spinto la moneta in una spirale deflativa.

Gli esperti di bitcoin sostengono che la deflazione non è in se per se un effetto negativo. Di solito, la deflazione è associata con un collasso di domanda perché quello è l'unico esempio di deflazione che usualmente viene studiata. In una valuta fiat con la possibilità di stampare moneta ad libitum, è molto difficile entrare in una spirale deflazionaria a meno che non ci sia un completo collasso nella domanda e la mancanza di volontà di

stampare nuova moneta. La deflazione in bitcoin non è causata da un collasso nella domanda ma da una massa monetaria sempre prevedibile.

L'aspetto positivo della deflazione, ovviamente, è che è l'opposto dell'inflazione. L'inflazione causa una lenta ma inevitabile svalutazione della moneta, che si traduce in una forma di tassazione nascosta che punisce i risparmiatori al fine di salvare i debitori (inclusi i maggiori debitori, i governi stessi). Le valute sotto controllo governativo risentono del rischio morale di un'emissione facile del debito che può essere cancellata in

seguito a svalutazione a spese dei risparmiatori.

Resta da vedere se l'aspetto deflazionistico della moneta è un problema quando non è guidato da una rapida ritrattazione economica, o da un vantaggio perché la protezione da inflazione e svalutazione supera di gran lunga i rischi di deflazione.

Consenso Decentralizzato

Nel capitolo precedente abbiamo esaminato la blockchain, il libro mastro globale (elenco) di tutte le transazioni, che tutti nella rete bitcoin

accettano come registrazione autorevole di proprietà.

Ma come possono tutti i partecipanti del network concordare su una singola unica "verità" riguardo a chi possiede cosa, senza avere il bisogno di fidarsi di nessuno? Tutti i sistemi di pagamento tradizionale dipendono su di un modello di fiducia che ha un'autorità centrale che fornisce il servizio di camera di compensazione (clearing house), praticamente verificando e liquidando tutte le transazioni. Bitcoin non ha un'autorità centrale, ma in qualche modo ogni full node ha la copia completa di un libro mastro a cui possa affidarsi usandolo come registro autoritativo. La

blockchain non è stata creata da un'autorità centrale, tuttavia è assemblata indipendentemente da ogni nodo nella rete. In qualche maniera, ogni nodo del network, agendo sulle informazioni trasmesse attraverso connessioni di rete non sicure, può arrivare alla stessa conclusione e ad assemblare una copia dello stesso libro mastro come tutti gli altri nodi. Questo capitolo esamina il processo con cui la rete bitcoin ottiene un consenso globale senza aver bisogno di un'autorità centrale.

L'invenzione più importante di Satoshi Nakamoto è il meccanismo decentralizzato di *consenso emergente*. Emergente, perché il

consenso non è raggiunto esplicitamente- non c'è un'elezione o un momento fisso nel quale occorre. Invece, il consenso viene creato dalla interazione asincrona di migliaia di nodi indipendenti, che seguono tutti semplici regole. Tutte le proprietà dei bitcoin, includendo la valuta, transazioni, pagamenti e il modello di sicurezza che non dipende da un'autorità centrale o di fiducia, deriva da questa invenzione.

Il consenso decentralizzato di bitcoin emerge dalla coordinazione di quattro processi che avvengono indipendentemente sui nodi del network:

- La verifica indipendente di ogni transazione, da ogni full node, basata su una lista comprensiva di criteri
- L'aggregazione indipendente di tali transazioni in nuovi blocchi mediante nodi di data mining, accoppiata a calcolo dimostrato tramite un algoritmo di Proof-of-Work
- Verifica indipendente dei nuovi blocchi da ogni nodo e l'assemblaggio in una catena di blocchi
- Selezione indipendente, da ogni

nodo, della catena con più calcolo cumulativo, dimostrato attraverso la prova di lavoro

Nelle prossime poche sezioni, esamineremo questi processi e come essi interagiscano per creare la proprietà emergente del consenso di rete che consente a ciascun nodo bitcoin di assemblare la propria copia dell'autorevole, fidato, pubblico, registro globale.

Verifica Indipendente delle Transazioni

In Transazioni, abbiamo visto come il software del portafoglio crea

transazioni raccogliendo UTXO, fornendo lo script di sblocco appropriato e quindi la costruzione di nuovi output assegnati a un nuovo proprietario. La transazione risultante viene quindi inviata ai nodi adiacenti nella rete bitcoin in modo che possa essere propagata attraverso l'intera rete bitcoin.

Invece, prima di inoltrare le transazioni ai suoi vicini, ogni nodo bitcoin che riceve una transazione per prima cosa verificherà la transazione. Questo assicura che solo le transazioni valide siano propagate sulla rete, mentre le transazioni non valide saranno scartate al primo nodo che le riceve.

Ogni nodo verifica ogni transazione seguendo una lunga lista di criteri:

- La sintassi della transazione e la struttura dati devono essere corrette.
- Ne la lista degli input ne quella degli output devono essere vuote.
- La dimensione della transazione in byte è inferiore di MAX_BLOCK_SIZE.
- I valori di ogni output, come per il totale, devono essere entro il range di valori consentito (meno di 21 milioni di bitcoin, maggiori di zero)

- Nessuno degli input hanno hash=0, N=–1 (le transazioni coinbase non devono essere trasmesse).
- nLockTime è inferiore o uguale a INT_MAX, oppure nLocktime e nSequence sono soddisfatti in base a MedianTimePast.
- La dimensione della transazione in byte è maggiore o uguale a 100.
- Il numero di operazioni di firma (SIGOPS) contenute nella transazione è inferiore al limite dell'operazione di firma.

- L’unlocking script (scriptSig) puo solo aggiungere numeri allo stack, e il locking script (scriptPubkey) deve corrispondere al form isStandard (questo scarta le transazioni "nonstandard").
- Deve esistere una transazione simile nella transaction pool, o in un blocco nel ramo principale.
- Per ogni input, se l’output referenziato esiste in qualsiasi altra transazione nella pool, la transazione deve essere respinta.

- Per ogni input, cercare il ramo principale e il pool di transazioni per trovare la transazione di output referenziata. Se la transazione di output manca per qualsiasi input, questa sarà una transazione orfana. Aggiungerla al pool di transazioni orfane, se una transazione corrispondente non è già nel pool.
- Per ogni input, se l'output di transazione referenziato è un output coinbase, dovrà avere almeno COINBASE_MATURITY (100)

conferme.

- Per ogni input, l'output referenziato deve esistere e non può essere già stato speso.
- Utilizzando le transazioni di output referenziate per ottenere i valori di input, verificare che ciascun valore di input, nonché la somma, siano compresi nell'intervallo di valori consentito (meno di 21 milioni di monete, più di 0).
- Respingi se la somma dei valori di input è meno della somma dei valori

di output.

- Rifiuta se la commissione di transazione sarà troppo bassa (minRelayTxFee) per entrare in un blocco vuoto.
- Gli script di sblocco per ogni input devono essere validi rispetto agli script di blocco dell'output corrispondenti.

Queste condizioni possono essere viste in dettaglio nelle funzioni AcceptToMemoryPool, CheckTransaction e CheckInputs nel client di riferimento bitcoin. Si noti che le condizioni cambiano nel tempo,

per affrontare nuovi tipi di attacchi denial-of-service o talvolta per allentare le regole in modo da includere più tipi di transazioni.

Verificando indipendentemente ogni transazione così come viene ricevuta e prima di propagarla, ogni nodo crea un pool di transazioni valide (ma non confermate) note come *transaction pool*, *memory pool* o *mempool*.

Nodi di Mining

Alcuni nodi della rete bitcoin sono nodi specializzati chiamati *miners*. In Introduzione abbiamo introdotto Jing, uno studente di ingegneria informatica a Shanghai, in Cina, che è un minatore bitcoin. Jing guadagna bitcoin

eseguendo un "mining rig", che è un sistema computer-hardware specializzato progettato per estrarre bitcoin. L'hardware di mining specializzato di Jing è connesso a un server che esegue un intero nodo bitcoin. A differenza di Jing, alcuni minatori non hanno un nodo completo, come vedremo in Le Mining Pool. Come ogni altro nodo completo, il nodo di Jing riceve e propaga transazioni non confermate sulla rete bitcoin. Il nodo di Jing, tuttavia, aggrega anche queste transazioni in nuovi blocchi.

Il nodo di Jing è in attesa di ricevere nuovi blocchi, propagati sulla rete bitcoin, come fanno tutti i nodi.

Tuttavia, l'arrivo di un nuovo blocco ha un significato speciale per un nodo di data mining. La competizione tra i minatori termina efficacemente con la propagazione di un nuovo blocco che funge da annuncio di un vincitore. Per i minatori, ricevere un nuovo blocco significa che qualcun altro ha vinto la competizione e che i riceventi hanno perso. Tuttavia, la fine di un round di una competizione è anche l'inizio del prossimo round. Il nuovo blocco non è solo una bandiera a scacchi, che segna la fine della corsa; è anche la pistola di partenza nella corsa per il prossimo blocco.

Aggregare le

transazioni in blocchi

Dopo aver convalidato le transazioni, un nodo bitcoin le aggiungerà al *memory pool* o *transaction pool*, dove le transazioni attendono fino a quando non possono essere incluse (minate) in un blocco. Il nodo di Jing raccoglie, convalida ed inoltra nuove transazioni proprio come qualsiasi altro nodo. A differenza degli altri nodi, tuttavia, il nodo di Jing aggregherà queste transazioni in un *candidate block* (ndt blocco candidato).

Seguiamo i blocchi che sono stati creati da quando Alice ha comprato un caffè dal Bar di Bob (vedi Pagare un Caffè). La transazione di Alice è stata

inclusa nel blocco 277.316. Per dimostrare alcuni concetti propri di questo capitolo, assumiamo che quel blocco sia stato minato dal sistema di mining di Jing e abbia seguito la transazione di Alice fino a quando questa non sia divenuta parte di questo nuovo blocco.

Il nodo di mining di Jing mantiene una copia locale della blockchain, la lista di tutti i blocchi creati dall'inizio del sistema bitcoin nel 2009. Quando Alice compra la tazza di caffè, il nodo di Jing ha assemblato una catena fino al blocco 277.314. Il nodo di Jing è in attesa di ricevere le transazioni, cercando di estrarre un nuovo blocco e anche attendendo l'arrivo di eventuali

blocchi scoperti da altri nodi. Poiché il nodo di Jing è un nodo di mining, riceve il blocco 277.315 attraverso la rete bitcoin. L'arrivo di questo blocco indica la fine della competizione per il blocco 277.315 e l'inizio della competizione per creare il blocco 277.316.

Durante i precedenti 10 minuti, mentre il nodo di Jing stava cercando una soluzione per il blocco 277.315, stava anche raccogliendo le transazioni in preparazione per il prossimo blocco. Ormai ha raccolto alcune centinaia di transazioni nel pool di memoria. Dopo aver ricevuto il blocco 277.315 e averlo convalidato, il nodo di Jing controllerà anche tutte le transazioni

nel pool di memoria e rimuoverà quelle che erano incluse nel blocco 277.315. Qualsiasi transazione rimanga nel pool di memoria non è confermata e attende di essere registrata in un nuovo blocco.

Il nodo di Jing costruisce subito un nuovo blocco vuoto, un candidato per il blocco 277.316. Questo blocco è detto blocco candidato perché non è ancora un blocco valido, in quanto non contiene una valida proof of work. Il blocco diventa valido solo se il miner riesce a trovare una soluzione per l'algoritmo di proof-of-work.

Quando il nodo di Jing aggrega tutte le transazioni dal pool di memoria, il nuovo blocco candidato ha 418

transazioni con commissioni di transazione totali pari a 0,09094928 bitcoin. Puoi vedere questo blocco nella blockchain usando l'interfaccia della riga di comando del client Bitcoin Core, come mostrato in Utilizzando la riga di comando per recuperare il blocco 277,316.

Esempio 22. Utilizzando la riga di comando per recuperare il blocco 277,316

```
$ bitcoin-cli getblockhash 277316

0000000000000001b6b9a13b095e96db41

$ bitcoin-cli getblock
0000000000000001b6b9a13b095e96db41
44a9b31b2cc7bdc4
```

```
{
   "hash" :
"000000000000001b6b9a13b095e96db4
   "confirmations" : 35561,
   "size" : 218629,
   "height" : 277316,
   "version" : 2,
   "merkleroot" :
"c91c008c26e50763e9f548bb8b2fc32373
   "tx" : [

"d5ada064c6417ca25c4308bd158c34b77e

"b268b45c59b39d759614757718b9918caf

      ... 417 more transactions ...

      ],
   "time" : 1388185914,
   "nonce" : 924591752,
   "bits" : "1903a30c",
```

```
   "difficulty" :
1180923195.25802612,
   "chainwork" :
"000000000000000000000000000000000
   "previousblockhash" :
"0000000000000002a7bbd25a417c0374c
}
```


La transazione generatrice (Coinbase)

La prima transazione aggiunta al blocco è una transazione speciale, denominata *generation transaction* o *coinbase transaction*. Questa transazione è costruita dal nodo di Jing ed è la sua *ricompensa* per lo sforzo di mining.

	Quando il blocco

NOTA	277.316 è stato estratto, la ricompensa era di 25 bitcoin per blocco. Da allora è trascorso un periodo di "dimezzamento". La ricompensa per il blocco è passata a 12,5 bitcoin a luglio 2016. Sarà dimezzata di nuovo in 210.000 blocchi, nell'anno 2020.

Il nodo di Jing crea la transazione di generazione come pagamento sul proprio portafoglio: "Pagare Jing all'indirizzo inviando 25.09094928 bitcoin." L'ammontare totale della ricompensa che Jing raccoglie per il

mining di un blocco è la somma della ricompensa di coinbase (25 nuovi bitcoin) e le commissioni di transazione (0.09094928) la somma di tutte le transazioni incluse nel blocco come mostrato in Transazione generatrice.

Esempio 23. Transazione generatrice

```
$ bitcoin-cli getrawtransaction
d5ada064c6417ca25c4308bd158c34b77e1
1

{
  "hex" :
"01000000010000000000000000000000
  "txid" :
"d5ada064c6417ca25c4308bd158c34b77e
```

```
   "version" : 1,
   "locktime" : 0,
   "vin" : [
      {
         "coinbase" :
"03443b0403858402062f503253482f",
         "sequence" : 4294967295
      }
   ],
   "vout" : [
      {
         "value" : 25.09094928,
         "n" : 0,
         "scriptPubKey" : {
            "asm" :
"02aa970c592640d19de03ff6f329d6fd2ee
            "hex" :
"2102aa970c592640d19de03ff6f329d6fd2
            "reqSigs" : 1,
            "type" : "pubkey",
            "addresses" : [
```

```
"1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2
            ]
        }
    }
  ]
}
```

A differenza delle normali transazioni, la transazione di generazione non consuma (spende) UTXO come input. Invece, ha solo un input, chiamato coinbase, che crea bitcoin dal nulla. La transazione di generazione ha un output, pagabile all'indirizzo bitcoin del minatore stesso. L'output della transazione di generazione invia il valore di 25.09094928 bitcoin all'indirizzo bitcoin del minatore, in questo caso

1MxTkeEP2PmHSMze5tUZ1hAV3YTK

Ricompensa Coinbase e Fee

Per costruire la transazione di generazione, il nodo di Jing calcola innanzitutto l'ammontare totale delle commissioni di transazione aggiungendo tutti gli input e gli output delle 418 transazioni che sono state aggiunte al blocco. Le tariffe sono calcolate come:

```
Fee Totali = Somma(Input) - Somma(Output)
```

Nel blocco 277.316, le fee di transazione totali sono di 0.09094928 bitcoin.

Successivamente, il nodo di Jing calcola la corretta ricompensa per il

nuovo blocco. La ricompensa è calcolata tramite la block height, che inizia a 50 bitcoin per blocco ed è ridotta della metà ogni 210.000 blocchi. Visto che questo blocco è alla height 277.316, la ricompensa corretta è di 25 bitcoin.

Il calcolo si può vedere nella funzione GetBlockSubsidy nel client Bitcoin Core, come mostrato in Calcolo del premio del blocco—Funzione GetBlockSubsidy, Bitcoin Core Client, main.cpp.

Esempio 24. Calcolo del premio del blocco—Funzione GetBlockSubsidy,

Bitcoin Core Client, main.cpp

```
CAmount GetBlockSubsidy(int nHeight, const Consensus::Params& consensusParams)
{
    int halvings = nHeight / consensusParams.nSubsidyHalvingInterval;
    // Forza la ricompensa del blocco a zero quando il right shift è undefined.
    if (halvings >= 64)
        return 0;

    CAmount nSubsidy = 50 * COIN;
    // La ricompensa è dimezzata ogni 210.000 blocchi, questo occorre approssimativamente ogni 4 anni.
    nSubsidy >>= halvings;
```

```
    return nSubsidy;
}
```

Il sussidio iniziale è calcolato in satoshi moltiplicando 50 con la costante COIN (100,000,000 satoshi). Questo imposta la ricompensa iniziale (nSubsidy) a 5 miliardi di satoshi.

Successivamente, la funzione calcola il numero di halving che si sono verificati dividendo l'altezza del blocco corrente per l'intervallo di dimezzamento (SubsidyHalvingInterval). Nel caso del blocco 277.316, con un intervallo di dimezzamento ogni 210.000 blocchi, il risultato è 1 dimezzamento.

Il numero massimo di dimezzamenti consentito è 64, dopo questo il codice

impone una ricompensa di zero (ritorna solo le fee) se il 64esimo dimezzamento viene superato.

Successivamente, la funzione utilizza l'operatorebinary-right-shift per dividere la ricompensa (nSubsidy) di due per ogni ciclo di dimezzamento. Nel caso del blocco 277.316, questo farebbe binary-right-shift sulla ricompensa di 5 miliardi di satoshi una volta (un dimezzamento) per ottenere 2,5 miliardi di satoshi, o 25 bitcoin. L'operatore binary-right-shift viene utilizzato perché è più efficiente per la divisione per due rispetto alla divisione di un intero o in virgola mobile. Per evitare un potenziale bug, l'operazione di spostamento viene

saltata dopo 63 fermi e il sussidio è impostato su 0.

Infine, la ricompensa coinbase (nSubsidy) è aggiunta alle fee di transazione (nFees), e viene ritornata la somma.

TIP	Se il nodo di mining di Jing scrive la transazione coinbase, cosa impedisce a Jing di "ricompensare" se stesso 100 o 1000 bitcoin? La risposta è che una ricompensa errata provocherebbe che il blocco fosse ritenuto non valido da tutti gli altri

	sprecando l'elettricità di Jing usata per il Proof-of-Work. Jing può solo spendere la ricompensa se il blocco è accettato da tutti.

Struttura della Transazione Coinbase

Con questi calcoli, il nodo di Jing costruisce la transazione generativa per pagarsi 25.09094928 bitcoin.

Come puoi vedere in Transazione generatrice, a transazione di generazione ha un formato speciale. Invece di un input di transazione che

specifica un UTXO precedente da spendere, ha un input "coinbase". Abbiamo esaminato gli input delle transazioni in Serializzazione dell'input della transazione. Confrontiamo un input di transazione normale con un input di transazione generativa. La struttura di un "normale" input di transazione mostra la struttura di una transazione normale, mentre La struttura dell'input di una transazione coinbase mostra la struttura dell'input della transazione di generativa.

Tabella 26. La struttura di un "nor di transazione

Dimensione	Campo	Des

32 byte	Hash della Transazione	Pun tran con l'U and
4 byte	Indice dell'Output	Il dell dell spe prir
1–9 byte (VarInt)	Dimensione dell'Unlocking-Script	Lun dell Scr seg
Variabile	Unlocking-	Unc

	Script	con con locl dell
4 byte	Sequence Number	Sol imp 0xF vist BIP

Tabella 27. La struttura dell'input d
transazione coinbase

Dimensione	Campo	Descri
32 byte	Hash di Transazione	Tutti sono a Non hash

		referen una transaz
4 byte	Indice dell'Output	Tutti sono 0xFFF
1–9 byte (VarInt)	Dimensione dei Dati Coinbase	Lunghe dei coinbas 2 a 100
Variabile	Dati Coinbase	Dati ar usati nonce e per ta mining blocchi

		deve ir con la height
4 byte	Numero di Sequenza	Imposta 0xFFFF

In una transazione di generazione, i primi due campi sono impostati su valori che non rappresentano un riferimento UTXO. Invece di un "Transaction Hash", il primo campo viene riempito con 32 byte tutti impostati su zero. L' "Output Index" è riempito con 4 byte tutti impostati su 0xFF (255 decimale). Lo "Unlocking Script" (scriptSig) è sostituito dai dati di coinbase, un campo dati arbitrario

usato dai minatori.

Coinbase Data

Le transazioni di generazione non hanno un campo script di sblocco (a.k.a., scriptSig). Invece, questo campo viene sostituito dai dati di coinbase, che devono essere compresi tra 2 e 100 byte. Tranne che per i primi pochi byte, il resto dei dati di coinbase può essere usato dai minatori in qualsiasi modo essi vogliano; sono dati arbitrari.

Nel genesis block, ad esempio, Satoshi Nakamoto ha aggiunto il testo "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks" nei dati di coinbase, utilizzandolo come

prova della data e per trasmettere un messaggio. Attualmente, i minatori usano i dati della base di monete per includere valori extra nonce e stringhe che identificano la mining pool.

I primi pochi byte della coinbase erano arbitrari, ma non è più così. Come da Bitcoin Improvement Proposal 34 (BIP0034), i blocchi della versione 2 (blocchi con il campo di versione impostato su 2) devono contenere l'indice di altezza del blocco come un'operazione di script "push" all'inizio del campo coinbase.

Nel blocco 277.316 vediamo che il coinbase (vedi Transazione generatrice (Coinbase)), che si trova nel campo "Unlocking Script" o

scriptSig dell'ingresso della transazione, contiene il valore esadecimale 03443b0403858402062f503253482f. Decodifichiamo questo valore.

Il primo byte, 03, indica al motore di esecuzione dello script di inserire i tre byte successivi nello stack di script (vedi Mette un valore nello stack). I successivi tre byte, 0x443b04, sono l'altezza del blocco codificata nel formato little-endian (inizia dal byte meno significativo per finire col più significativo). Invertire l'ordine dei byte e il risultato è 0x043b44, che è 277.316 in decimale.

Le poche prossime cifre esadecimali (0385840206 2) sono usate per

codificare un nonce extra (vedi La Soluzione del Nonce Extra), o valore casuale, usato per trovare una soluzione adatta alla proof of work.

La parte finale dei dati di coinbase (2f503253482f) è la stringa codificata ASCII /P2SH/, che indica che il nodo di mining che ha estratto questo blocco supporta miglioramento pay-to-script-hash (P2SH) definito in BIP-16. L'introduzione della funzionalità P2SH richiedeva un "voto" da parte dei minatori per approvare BIP-16 o BIP-17. Coloro che avallano l'implementazione di BIP-16 dovevano includere /P2SH/ nei loro dati di coinbase. Coloro che approvavano l'implementazione BIP-

17 di P2SH dovevano includere la stringa p2sh/CHV nei loro dati di coinbase. Il BIP-16 è stato eletto come il vincitore e molti minatori hanno continuato a includere la stringa /P2SH/ nella loro coinbase per indicare il supporto per questa funzione.

Estrae i dati coinbase dal genesis block utilizza la libreria libbitcoin introdotta in Client Alternativi, Librerie, e Toolkits per estrarre i dati della coinbase dal genesis block, visualizzando il messaggio di Satoshi. Si noti che la libreria libbitcoin contiene una copia statica del blocco genesis, quindi il codice di esempio può recuperare il blocco genesi

direttamente dalla libreria.

Esempio 25. Estrae i dati coinbase dal genesis block

```
/*
 Display the genesis block message by Satoshi.
*/
#include <iostream>
#include <bitcoin/bitcoin.hpp>

int main()
{
    // Create genesis block.
    bc::chain::block block = bc::chain::block::genesis_mainnet();
    // Genesis block contains a single coinbase transaction.
```

```
    assert(block.transactions().size() ==
1);
    // Get first transaction in block
(coinbase).
    const bc::chain::transaction&
coinbase_tx = block.transactions()[0];
    // Coinbase tx has a single input.
    assert(coinbase_tx.inputs().size() ==
1);
    const bc::chain::input&
coinbase_input = coinbase_tx.inputs()
[0];
    // Convert the input script to its raw
format.
    const auto prefix = false;
    const bc::data_chunk& raw_message
=
coinbase_input.script().to_data(prefix);
    // Convert this to a std::string.
    std::string
message(raw_message.begin(),
```

```
raw_message.end());
    // Display the genesis block message.
    std::cout << message << std::endl;
    return 0;
}
```

Compiliamo il codice con il compilatore GNU C ++ ed eseguiamo l'eseguibile risultante, come mostrato in Compilando e lanciando il codice di esempio satoshi-words.

Esempio 26. Compilando e lanciando il codice di esempio satoshi-words

```
$ # Compila il codice

$  g++ -o satoshi-words satoshi-words.cpp $(pkg-config --cflags --libs
```

```
libbitcoin)
$ # Esegue l'eseguibile
$ ./satoshi-words
The Times 03/Jan/2009 Chancellor on brink of second bailout for banks --- Il New York Times - 3 Gennaio 2009 - Il Cancelliere sull'orlo del secondo salvataggio per le banche
```

Costruendo l'Header del Blocco

Per costruire il block header, il nodo di mining deve completare sei campi, come mostrato in La struttura di un block header.

Tabella 28. La struttura di un header

Dimensione	Campo	Descrizi

4 byte	Versione	Un nume versione tracciare upgrade software al protoc
32 byte	Hash del Blocco Precedente	Un riferimer all'hash blocco preceder (genitore nella cha
32 byte	Merkle Root	Un'hash radice merkle

		delle transazio questo b
4 byte	Timestamp	Il approssi della creazion blocco corrente (secondi Unix Epo
4 byte	Target	Il targe difficolta dell'algo di pro work questo b

4 byte	Nonce	Un cor utilizzat l’algorit proof-of

Nel tempo nel quale è stato effettuato mining sul blocco 277.316, il numero di versione che descrive la struttura del blocco è 2, che è codificato nel formato little-endian in 4 byte come 0x02000000.

Successivamente, il nodo di mining deve aggiungere il "Previous Block Hash" (ndt l’hash del blocco precedente). Questo è l’hash dell’header del blocco 277.315, il

blocco precedente ricevuto dalla rete, che il nodo di Jing ha accettato e selezionato come *genitore* del blocco candidato 277.316. L'hash dell'intestazione del blocco per il blocco 277.315 è:

```
00000000000000002a7bbd25a417c0374cc5
```

TIP	Selezionando il blocco *genitore* specifico indicato dal campo Hash di blocco precedente nell'intestazione del blocco candidato, Jing sta usando il suo potere di estrazione per estendere la catena che termina in quel blocco

	specifico. In sostanza questo è il modo in cui Jing "vota" con il suo potere di estrazione per la catena valida più a lungo.

Il prossimo passo è quello di riepilogare tutte le transazioni con un merkle tree, in modo da aggiungere la radice del merkle tree, all'intestazione del blocco. La transazione di generazione è elencata come prima transazione nel blocco. Quindi, dopo di esso vengono aggiunte altre 418 transazioni, per un totale di 419 transazioni nel blocco. Come abbiamo visto ne I Merkle Tree, ci deve essere

un numero pari di nodi "foglia" nell'albero, quindi l'ultima transazione viene duplicata, creando 420 nodi, ciascuno contenente l'hash di una transazione. Gli hash della transazione vengono quindi combinati, in coppie, creando ogni livello dell'albero, fino a quando tutte le transazioni sono riepilogate in un nodo nella "radice" dell'albero. La radice del merkle tree riassume tutte le transazioni in un singolo valore di 32 byte, che puoi vedere elencato come "merkle root" in Utilizzando la riga di comando per recuperare il blocco 277,316, e qui:

```
c91c008c26e50763e9f548bb8b2fc323735f
```

Il nodo di data mining aggiungerà quindi un timestamp a 4 byte,

codificato come timestamp "Epoch" di Unix, che si basa sul numero di secondi trascorsi dalla mezzanotte del 1 ° gennaio 1970 UTC/GMT. Il tempo 1388185914 è uguale a venerdì, 27 dicembre 2013, 23:11:54 UTC/GMT.

Il nodo quindi riempie l'obiettivo di difficoltà, che definisce la difficoltà richiesta di prova del lavoro per rendere questo un blocco valido. La difficoltà è memorizzata nel blocco come una metrica "difficulty bits", che è una codifica mantissa-exponent del bersaglio. La codifica ha un esponente da 1 byte, seguito da un coefficiente di mantissa di 3 byte. Ad esempio, nel blocco 277.316, il valore dei bit di difficoltà è 0x1903a30c. La prima

parte 0x19 è un esponente esadecimale, mentre la parte successiva, 0x03a30c, è il coefficiente. Il concetto di un obiettivo di difficoltà è spiegato in Il Target di Difficoltà e Il Retargeting e la rappresentazione del "difficulty bits" è spiegata in Rappresentazione della Difficoltà.

Il campo finale è il nonce, che è inizializzato a zero.

Con tutti gli altri campi riempiti, l'intestazione del blocco è ora completa e può iniziare il processo di estrazione. L'obiettivo è ora di trovare un valore per il nonce che generi un hash di intestazione del blocco, inferiore all'obiettivo di difficoltà. Il

nodo di mining dovrà testare miliardi o trilioni di valori nonce prima che venga trovato un nonce che soddisfi i requisiti.

Effettuando Mining sul Blocco

Ora che un blocco candidato è stato costruito dal nodo di Jing, è tempo che l'hardware di Jing "mini" il blocco per trovare una soluzione all'algoritmo di proof-of-work (ndt prova di lavoro) che rende valido il blocco. In questo libro abbiamo studiato le funzioni hash crittografiche utilizzate in vari aspetti del sistema bitcoin. La funzione hash SHA256 è la funzione utilizzata nel

processo di mining di bitcoin.

In termini più semplici, il mining è la ripetizione del processo di hashing dell'header del blocco, modificando un parametro (ndt nonce), fino a quando l'hash risultante corrisponde ad un target specifico. Il risultato della funzione di hash non può essere determinato in anticipo, né può essere creato uno schema che produca un valore hash specifico. Questa caratteristica delle funzioni di hash significa che l'unico modo per produrre un risultato di hash corrispondente ad un obiettivo specifico è di provare ancora e ancora, modificando casualmente l'input fino a quando il risultato

dell'hash desiderato appare per caso.

Algoritmo di Proof-Of-Work

Un algoritmo hash prende un input di dati di lunghezza arbitraria e produce un risultato deterministico a lunghezza fissa, un'impronta digitale digitale dell'input. Per ogni input specifico, l'hash risultante sarà sempre lo stesso e può essere facilmente calcolato e verificato da chiunque implementa lo stesso algoritmo di hash. La caratteristica chiave di un algoritmo di hash crittografico è che è praticamente impossibile trovare due input diversi che producono la stessa impronta digitale (conosciuta come *collision*). Come corollario, è anche praticamente

impossibile selezionare un input in modo tale da produrre un'impronta digitale desiderata, oltre a provare input casuali.

Con SHA256, l'output è sempre lungo 256 bit, indipendentemente dalla dimensione dell'input. In Esempio di SHA256, utilizzeremo l'interprete Python per calcolare l'hash SHA256 della frase "I am Satoshi Nakamoto."

Esempio 27. Esempio di SHA256

```
$ python

Python 2.7.1
>>> import hashlib
>>> print hashlib.sha256("I am Satoshi Nakamoto").hexdigest()
```

5d7c7ba21cbbcd75d14800b100252d5b428

Esempio di SHA256 mostra il risultato del calcolo dell'hash di "I am Satoshi Nakamoto":
5d7c7ba21cbbcd75d14800b100252d5b
Questo numero a 256 bit è l' *hash* o *digest* della frase e dipende da ogni parte della frase. L'aggiunta di una singola lettera, un segno di punteggiatura o qualsiasi altro carattere produrrà un diverso hash.

Adesso, se cambiano la frase, dovremo aspettarci hash completamente differenti. Proviamolo aggiungendo un numero alla fine della nostra frase, usando il semplice script python Script SHA256 per generare molti hash ripetendo su un nonce.

Esempio 28. Script SHA256 per generare molti hash ripetendo su un nonce

```
# example of iterating a nonce in a
hashing algorithm's input
```

```
from __future__ import print_function
import hashlib

text = "I am Satoshi Nakamoto"

# iterate nonce from 0 to 19
for nonce in range(20):

    # add the nonce to the end of the text
    input_data = text + str(nonce)
```

```
    # calculate the SHA-256 hash of the
input (text+nonce)
    hash_data =
hashlib.sha256(input_data).hexdigest()

    # show the input and hash result
    print(input_data, '=>', hash_data)
```

Eseguendo questo, verranno prodotti gli hash di varie frasi, ottenute ognuna diversa dall'altra tramite l'aggiunta di un numero alla fine del testo. Incrementando il numero, possiamo ottenere hash differenti, come mostrato i n Output SHA256 di uno script per generare molti hash iterando su di un nonce.

Esempio 29. Output SHA256 di uno script per generare

molti hash iterando su di un nonce

```
$ python hash_example.py

I am Satoshi Nakamoto0 =>
a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 =>
f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 =>
ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 =>
bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 =>
bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 =>
eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 =>
4a2fd48e3be420d0d28e202360cfbaba...
I am Satoshi Nakamoto7 =>
790b5a1349a5f2b909bf74d0d166b17a...
```

I am Satoshi Nakamoto8 =>
702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 =>
7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 =>
c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 =>
7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 =>
60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 =>
0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 =>
27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 =>
394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 =>
8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 =>
dca9b8b4f8d8e1521fa4eaa46f4f0cd...
I am Satoshi Nakamoto18 =>

```
9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 =>
cda56022ecb5b67b2bc93a2d764e75f...
```

Ogni frase produce un risultato hash completamente diverso. Sembrano completamente casuali, ma è possibile riprodurre i risultati esatti in questo esempio su qualsiasi computer con Python e vedere gli stessi hash esatti.

Il numero usato come variabile in tale scenario è chiamato *nonce*. Il nonce viene utilizzato per variare l'output di una funzione crittografica, in questo caso per variare l'impronta digitale SHA256 della frase.

Per creare una sfida a questo algoritmo, impostiamo un obiettivo arbitrario: trova una frase che produce

un hash esadecimale che inizia con uno zero. Fortunatamente, ciò non è difficile Output SHA256 di uno script per generare molti hash iterando su di un nonce mostra che la frase "I am Satoshi Nakamoto13" produce l'hash 0ebc56d59a34f5082aaef3d66b37a6616 che corrisponde ai nostri criteri. Ci sono voluti 13 tentativi per trovarlo. In termini di probabilità, se l'output della funzione di hash è distribuito uniformemente ci aspetteremmo di trovare un risultato con uno 0 come prefisso esadecimale una volta ogni 16 hash (una su 16 cifre esadecimali da 0 a F). In termini numerici, ciò significa trovare un valore hash inferiore a 0x10000000000000000000000000000

Chiamiamo questa soglia il *target* e l'obiettivo è trovare un hash che sia numericamente più piccolo rispetto al target. Se riduciamo il bersaglio, il compito di trovare un hash che è inferiore all'obiettivo diventa sempre più difficile.

Per fare una semplice analogia, immagina un gioco in cui i giocatori lanciano ripetutamente un paio di dadi, cercando di fare un numero inferiore ad un target specificato. Nel primo round, l'obiettivo è 12. A meno che non si lanci doppio-sei, si vince. Nel turno successivo l'obiettivo è 11. I giocatori devono fare 10 o meno per vincere, ancora una volta un compito facile. Diciamo alcuni round dopo che

l'obiettivo è sceso a 5. Ora, più della metà dei tiri di dadi si sommano a più di 5 e quindi non validi. Ci vogliono più tiri per vincere, il numero di tiri cresce in modo esponenziale, più basso diventa il target. Alla fine, quando il bersaglio è 2 (il minimo possibile), solo un tiro su ogni 36, o il 2% di loro, produrrà un risultato vincente.

Dal punto di vista di un osservatore che sa che il bersaglio del gioco dei dadi è 2, se qualcuno è riuscito a lanciare un tiro vincente si può presumere che abbia tentato, in media, 36 tiri. In altre parole, si può stimare la quantità di lavoro necessaria per riuscire nella difficoltà imposta

dall'obiettivo. Quando l'algoritmo è basato su una funzione deterministica come SHA256, l'input stesso costituisce la prova che è stata eseguita una certa quantità di lavoro per produrre un risultato al di sotto del target. Appunto, prova di lavoro.

TIP	Anche se ogni tentativo produce un risultato casuale, la probabilità di qualsiasi risultato possibile può essere calcolata in anticipo. Pertanto, un risultato di difficoltà specificato costituisce la prova di una quantità specifica di lavoro.

In Output SHA256 di uno script per generare molti hash iterando su di un nonce, il "nonce" vincente è 13 e questo risultato può essere confermato da chiunque in modo indipendente. Chiunque può aggiungere il numero 13 come suffisso alla frase "I am Satoshi Nakamoto" e calcolare l'hash, verificando che sia inferiore all'obiettivo. Il risultato positivo è anche la prova del lavoro, perché dimostra che abbiamo fatto il lavoro per trovare quel nonce. Mentre ci vuole solo un calcolo hash per verificare, ci sono voluti 13 calcoli hash per trovare un nonce che ha

funzionato. Se avessimo un obiettivo inferiore (difficoltà più alta) ci vorrebbero molti più calcoli hash per trovare un nonce adatto, ma solo un calcolo hash per chiunque da verificare. Inoltre, conoscendo l'obiettivo, chiunque può stimare la difficoltà usando le statistiche e quindi sapere quanto lavoro è stato necessario per trovare un tale nonce.

TIP	La prova di lavoro deve produrre un hash *che è inferiore* all'obiettivo. Un bersaglio più alto significa che è meno difficile trovare un hash che si trova sotto il bersaglio. Un bersaglio

	più basso significa che è più difficile trovare un hash sotto il bersaglio. L'obiettivo e la difficoltà sono inversamente correlati.

La dimostrazione del lavoro di Bitcoin è molto simile alla sfida mostrata in Output SHA256 di uno script per generare molti hash iterando su di un nonce. Il minatore costruisce un blocco candidato pieno di transazioni. Successivamente, il minatore calcola l'hash dell'intestazione di questo blocco e vede se è più piccolo dell'attuale *target*. Se l'hash non è

inferiore all'obiettivo, il minatore modificherà il nonce (di solito solo incrementandolo di uno) e riprova. Alla difficoltà attuale nella rete bitcoin, i minatori devono provare quadrilioni di volte prima di trovare un nonce che si traduce in un hash dell'intestazione di blocco abbastanza basso.

Una versione molto semplificata dell'algoritmo di proof-of-lavoro è implementata in Python in Implementazione proof-of-work semplificata..

Esempio 30. Implementazione proof-of-work semplificata

```
#!/usr/bin/env python
```

```
# example of proof-of-work algorithm

import hashlib
import time

try:
    long        # Python 2
    xrange
except NameError:
    long = int  # Python 3
    xrange = range

max_nonce = 2 ** 32  # 4 billion


def proof_of_work(header,
difficulty_bits):
    # calculate the difficulty target
```

```
    target = 2 ** (256 - difficulty_bits)

    for nonce in xrange(max_nonce):
        hash_result =
hashlib.sha256(str(header) +
str(nonce)).hexdigest()

        # check if this is a valid result,
below the target
        if long(hash_result, 16) < target:
            print("Success with nonce %d"
% nonce)
            print("Hash is %s" %
hash_result)
            return (hash_result, nonce)

    print("Failed after %d (max_nonce)
tries" % nonce)
    return nonce
```

```
if __name__ == '__main__':
    nonce = 0
    hash_result = ''

    # difficulty from 0 to 31 bits
    for difficulty_bits in xrange(32):
        difficulty = 2 ** difficulty_bits
        print("Difficulty: %ld (%d bits)" %
(difficulty, difficulty_bits))
        print("Starting search...")

        # checkpoint the current time
        start_time = time.time()

        # make a new block which includes
the hash from the previous block
        # we fake a block of transactions -
just a string
        new_block = 'test block with
transactions' + hash_result
```

```
        # find a valid nonce for the new
block
        (hash_result, nonce) =
proof_of_work(new_block,
difficulty_bits)

        # checkpoint how long it took to
find a result
        end_time = time.time()

        elapsed_time = end_time -
start_time
        print("Elapsed Time: %.4f
seconds" % elapsed_time)

        if elapsed_time > 0:

            # estimate the hashes per second
            hash_power = float(long(nonce)
/ elapsed_time)
            print("Hashing Power: %ld
```

```
hashes per second" % hash_power)
```

Eseguendo questo codice, puoi scegliere la difficoltà (in bit, quanti dei primi bit devono avere valore zero) e vedere quanto tempo impiega il tuo computer per trovare una soluzione. In Eseguendo l'esempio di proof of work a varie difficoltà, puoi vedere come lavora su un laptop di fascia media.

Esempio 31. Eseguendo l'esempio di proof of work a varie difficoltà

```
$ python proof-of-work-example.py*

Difficulty: 1 (0 bits) --- Difficoltà: 1
(0 bits)
```

[...]

Difficulty: 8 (3 bits) --- Difficoltà: 8 (3 bits)
Starting search... --- Inizio la ricerca...
Success with nonce 9 --- Trovato con nonce 9
L'hash è 1c1c105e65b47142f028a8f93ddf3dabb926
Tempo Trascorso: 0.0004 secondi
Hashing Power: 25065 hashes per second
Difficulty: 16 (4 bits)
Starting search... --- Inizio la ricerca...
Success with nonce 25
Hash is 0f7becfd3bcd1a82e06663c97176add89e7c
Elapsed Time: 0.0005 seconds
Hashing Power: 52507 hashes per second

```
Difficulty: 32 (5 bits)
Starting search... --- Inizio la ricerca...
Success with nonce 36
Hash is
029ae6e5004302a120630adcbb80845234
Elapsed Time: 0.0006 seconds
Hashing Power: 58164 hash per
secondo

[...]

Difficulty: 4194304 (22 bits)
Starting search... --- Inizio la ricerca...
Success with nonce 1759164
Hash is
0000008bb8f0e731f0496b8e530da984e85
Elapsed Time: 13.3201 seconds
Hashing Power: 132068 hashes per
second
Difficulty: 8388608 (23 bits)
Starting search... --- Inizio la ricerca...
```

```
Success with nonce 14214729
Hash is
000001408cf12dbd20fcba6372a223e098d
Tempo Trascorso: 110.1507 secondi
Hashing Power: 129048 hashes per
second
Difficulty: 16777216 (24 bits)
Starting search... --- Inizio la ricerca...
Success with nonce 24586379
Hash is
0000002c3d6b370fccd699708d1b7cb4a94
Elapsed Time: 195.2991 seconds
Hashing Power: 125890 hashes per
second

[...]

Difficulty: 67108864 (26 bits)
Starting search... --- Inizio la ricerca...
Success with nonce 84561291
Hash is
```

```
000001f0ea21e676b6dde5ad429b9d131a
Elapsed Time: 665.0949 seconds
Hashing Power: 127141 hashes per
second
```

Come puoi vedere, l'aumento della difficoltà di 1 bit provoca un aumento esponenziale del tempo necessario per trovare una soluzione. Se si pensa all'intero spazio di 256 bit, ogni volta che si limita un altro bit a zero, si riduce lo spazio di ricerca della metà. I n Eseguendo l'esempio di proof of work a varie difficoltà, ci vogliono 84 milioni di tentativi di hash per trovare un nonce che produce un hash con 26 bit iniziali come zero. Anche a una velocità di oltre 120.000 hash al secondo, richiede ancora 10 minuti su

un laptop consumer per trovare questa soluzione.

Al momento della scrittura, la rete sta tentando di trovare un blocco il cui intestazione hash è inferiore a:

```
0000000000000029AB900000000000
```

Come puoi vedere, all'inizio di tale hash ci sono molti zeri, il che significa che l'intervallo accettabile di hash è molto più piccolo, quindi è più difficile trovare un hash valido. Ci vorranno in media più di 1.8 zeta-hash (miliardi di hash) al secondo per consentire alla rete di scoprire il prossimo blocco. Sembra un compito impossibile, ma fortunatamente la rete ha toccato i 3 exa-hash al secondo

(EH/sec) di potenza di elaborazione da sopportare, per trovare un blocco in media in circa 10 minuti.

Rappresentazione della Difficoltà

In Utilizzando la riga di comando per recuperare il blocco 277,316, abbiamo visto che il blocco contiene l'obiettivo di difficoltà, in una notazione chiamata "difficulty bits" o semplicemente "bits", che nel blocco 277.316 ha il valore di 0x1903a30c. Questa notazione esprime l'obiettivo di difficoltà come formato coefficiente/esponente, con le prime due cifre esadecimali per l'esponente e le successive sei cifre esadecimali

come coefficiente. In questo blocco, quindi, l'esponente è 0x19 e il coefficiente è 0x03a30c.

La formula per calcolare il target di difficoltà da questa rappresentazione è:

- target = coefficient * $2^{(8*(exponent-3))}$

Usando quella formula, e i bit di difficoltà con valore 0x1903a30c, otteniamo:

- target = 0x03a30c * $2^{0x08*(0x19-0x03)}$
- => target = 0x03a30c * $2^{(0x08*0x16)}$
- => target = 0x03a30c * 2^{0xB0}

il quale decimale è:

- => target = 238,348 * 2176
- => target = 22,829,202,948,393,929,850,749,706,

ri-convertendolo in esadecimale:

- => target = 0x00000000000003A30C0000000

Ciò significa che un blocco valido per l'altezza 277.316 è uno che ha un hash di intestazione di blocco inferiore alla destinazione. In binario, quel numero avrebbe più di 60 bit impostati a zero. Con questo livello di difficoltà, un singolo minatore che elabora 1 trilione di hash al secondo (1 tera-hash al secondo o 1 TH/sec) troverà una soluzione solo una volta ogni 8.496 blocchi o una volta ogni 59 giorni, in

media.

Il Target di Difficoltà e Il Retargeting

Come abbiamo visto, il target determina la difficoltà e quindi influenza il tempo necessario per trovare una soluzione da parte dell'algoritmo di proof of work. Questo porta alle domande ovvie: perché la difficoltà è regolabile, chi la regola e come?

I blocchi di Bitcoin vengono generati ogni 10 minuti, in media. Questo è il battito (ndt del cuore) del bitcoin e sostiene la frequenza dell'emissione di valuta e la velocità del regolamento delle transazioni. Deve rimanere

costante non solo nel breve periodo, ma per un periodo di molti decenni. In questo periodo, si prevede che la potenza dei computer continuerà ad aumentare a un ritmo rapido. Inoltre, anche il numero di partecipanti nel settore del mining e i computer che usano cambierà costantemente. Per mantenere il tempo di generazione del blocco a 10 minuti, la difficoltà di estrazione deve essere regolata per tenere conto di questi cambiamenti. Infatti, la difficoltà è un parametro dinamico che verrà periodicamente regolato per raggiungere un target di blocco di 10 minuti. In termini semplici, l'obiettivo di difficoltà farà si che qualsiasi potenza di mining sarà

disponibile, genererà un blocco ogni 10 minuti.

In che modo, quindi, tale adeguamento viene effettuato in una rete completamente decentralizzata? Il retargeting di difficoltà si verifica automaticamente e su ogni nodo indipendentemente. Ogni 2.016 blocchi, tutti i nodi ritarano la difficoltà proof of work. L'equazione per la difficoltà di retargeting misura il tempo impiegato per trovare gli ultimi 2.016 blocchi e li confronta con il tempo previsto di 20.160 minuti (due settimane in base al tempo di blocco desiderato di 10 minuti). Viene calcolato il rapporto tra il periodo di tempo effettivo e il periodo di tempo

desiderato e viene effettuata una correzione corrispondente (in su o giù) alla difficoltà. In termini semplici: se la rete trova blocchi più velocemente di ogni 10 minuti, la difficoltà aumenta. Se la scoperta dei blocchi è più lenta del previsto, la difficoltà diminuisce.

L'equazione può essere riassunta in:

```
New Target = Old Target * (Actual Time of Last 2016 Blocks / 20160 minutes)
```

Retargeting della difficoltà di prova di lavoro — CalculateNextWorkRequired () in pow.cpp mostra il codice utilizzato nel client Bitcoin Core.

Esempio 32. Retargeting

della difficoltà di prova di lavoro — CalculateNextWorkRequired() in pow.cpp

```
    // Step di aggiustamento del limite
```

```
    int64_t nActualTimespan = pindexLast->GetBlockTime() - nFirstBlockTime;
    LogPrintf("  nActualTimespan = %d before bounds\n", nActualTimespan);
    if (nActualTimespan < params.nPowTargetTimespan/4)
        nActualTimespan = params.nPowTargetTimespan/4;
    if (nActualTimespan > params.nPowTargetTimespan*4)
        nActualTimespan =
```

```
params.nPowTargetTimespan*4;

    // Retarget
    const arith_uint256 bnPowLimit =
UintToArith256(params.powLimit);
    arith_uint256 bnNew;
    arith_uint256 bnOld;
    bnNew.SetCompact(pindexLast-
>nBits);
    bnOld = bnNew;
    bnNew *= nActualTimespan;
    bnNew /=
params.nPowTargetTimespan;

    if (bnNew > bnPowLimit)
        bnNew = bnPowLimit;
```

	Mentre la ricalibrazione del target dovrebbe avvenire ogni 2.016

NOTA	blocchi, a causa di un errore "off-by-one" nel client Bitcoin Core originale, si basa invece sul tempo totale dei precedenti 2.015 blocchi (non a 2.016 come dovrebbe essere), determinando un bias di retargeting verso maggiore difficoltà dello 0,05%

I parametri Intervallo (2.016 blocchi) e TargetTimespan (due settimane come 1.209.600 secondi) sono definiti in *chainparams.cpp*.

Per evitare la volatilità estrema nella difficoltà, la regolazione del retargeting deve essere inferiore a un fattore di quattro (4) per ciclo. Se la regolazione della difficoltà richiesta è maggiore di un fattore di quattro, sarà regolata al massimo e non di più. Qualsiasi ulteriore aggiustamento verrà effettuato nel prossimo periodo di retargeting, in quanto lo squilibrio persisterà nei successivi 2.016 blocchi. Pertanto, grandi discrepanze tra la potenza di hash e la difficoltà potrebbero richiedere diversi cicli da 2.016 blocchi per bilanciare.

	La difficoltà nel trovare un blocco bitcoin è approssivativamente '10

TIP	minuti di calcolo' per l'intero network, basato sul tempo che ci vuole per trovare i precedenti 2.016 blocchi, aggiustata ogni 2.016 blocchi.

Si noti che la difficoltà di destinazione è indipendente dal numero di transazioni o dal valore delle transazioni. Ciò significa che la quantità di potenza di hash, e quindi di energia spesa per proteggere bitcoin, è del tutto indipendente dal numero di transazioni. Bitcoin può scalare, ottenere un'adozione più ampia e rimanere al sicuro senza alcun aumento

della potenza di hashing dal livello attuale. L'aumento della potenza di hash rappresenta le forze del mercato quando nuovi minatori entrano nel mercato per competere per la ricompensa. Fintanto che un sufficiente potere di hashing è sotto il controllo dei minatori che agiscono onestamente alla ricerca della ricompensa, è sufficiente per prevenire attacchi di "takeover" e, quindi, è sufficiente per proteggere i bitcoin.

La difficoltà dell'obiettivo è strettamente correlata al costo dell'elettricità e al tasso di cambio del bitcoin rispetto alla valuta utilizzata per pagare l'elettricità. I sistemi di estrazione ad alte prestazioni sono

quanto più efficienti possibile con l'attuale generazione di device al silicio, convertendo l'elettricità in calcolo dell'hashing alla massima velocità possibile. L'influenza primaria sul mercato dei miner è il prezzo di un chilowattora in bitcoin, perché questo determina la redditività del settore mining e quindi gli incentivi per entrare o uscire dal mercato del mining.

Effettuando Mining del Blocco con Successo

Come abbiamo visto prima, il nodo di Jing ha costruito un blocco candidato e lo ha preparato per il mining. Jing ha

diverse piattaforme hardware mining circuiti integrati specifici dell'applicazione, dove centinaia di migliaia di circuiti integrati eseguono l'algoritmo SHA256 in parallelo a velocità incredibili. Queste macchine specializzate sono collegate al suo nodo di mining via USB. Successivamente, il nodo di mining in esecuzione sul desktop di Jing trasmette l'intestazione del blocco al suo hardware di data mining, che inizia a testare trilioni di nonce al secondo. Poiché il nonce ha solo 32 bit, dopo aver esaurito tutte le possibilità nonce (circa 4 miliardi), l'hardware di data mining modifica l'intestazione del blocco (aggiustando

lo spazio nonce o timestamp della base di monete) e reimposta il contatore nonce, testando nuove combinazioni.

Quasi 11 minuti dopo l'avvio del mio blocco 277.316, una delle macchine di mining hardware trova una soluzione e la rimanda al nodo di data mining.

Quando inserito nell'intestazione del blocco, il nonce 4.215.469.401 produce un hash di blocco di:

```
0000000000000001b6b9a13b095e96db41c
```

che è inferiore al target:

```
00000000000003A30C0000000000000
```

Immediatamente, il nodo di mining di Jing trasmette il blocco a tutti i suoi pari. Essi ricevono, convalidano e quindi propagano il nuovo blocco.

Mentre il blocco si diffonde attraverso la rete, ogni nodo lo aggiunge alla propria copia della blockchain, estendendolo a una nuova altezza di 277.316 blocchi. Mentre i nodi di mining ricevono e convalidano il blocco, abbandonano i loro sforzi per trovare un blocco alla stessa altezza e iniziano immediatamente a calcolare il prossimo blocco della catena, usando il blocco di Jing come "genitore". Costruendo in cima al blocco appena scoperto di Jing, gli altri minatori stanno essenzialmente "votando" con il loro potere minerario e approvando il blocco di Jing e la catena che estende.

Nella prossima sezione, osserveremo il processo usato da ogni nodo per

validare un blocco e selezionare la catena più lunga, creando il consenso che forma la blockchain decentralizzata.

Validando un Nuovo Blocco

Il terzo passo nel meccanismo di consenso di bitcoin è la convalida indipendente di ogni nuovo blocco da parte di ogni nodo della rete. Mentre il blocco appena risolto si propaga attraverso la rete, ciascun nodo esegue una serie di test per convalidarlo prima di propagarlo ai suoi pari. Ciò garantisce che solo i blocchi validi vengano propagati sulla rete. La

convalida indipendente assicura anche che i minatori che agiscono in modo onesto ottengano i loro blocchi incorporati nella blockchain, guadagnando così la ricompensa. Quei minatori che agiscono disonestamente hanno i loro blocchi respinti e non solo perdono la ricompensa, ma anche sprecano lo sforzo speso per trovare una soluzione di proof of work, incorrendo così nel costo dell'elettricità senza compensazione.

Quando un nodo riceve un nuovo blocco, convaliderà il blocco stesso controllandolo da una lunga lista di criteri che devono essere soddisfatti tutti; in caso contrario, il blocco viene rifiutato. Questi criteri possono essere

visualizzati nel client Bitcoin Core nelle funzioni CheckBlock e CheckBlockHeader e include:

- La struttura del blocco è sintatticamente valida
- L'hash del block header è inferiore della difficoltà del target (impone la proof of work)
- Il timestamp del blocco è inferiore di due ore nel futuro (permettendo errori temporali) *La dimensione del blocco è entro i limiti accettati
- La prima transazione (e solo la prima) è una transazione di generazione coinbase
- Tutte le transazioni nel blocco sono valide usando la checklist di

transazione introdotta in Verifica Indipendente delle Transazioni

La convalida indipendente di ogni nuovo blocco da parte di ogni nodo della rete garantisce che i minatori non possano imbrogliare. Nelle sezioni precedenti abbiamo visto come i minatori possono scrivere una transazione che assegna loro i nuovi bitcoin creati all'interno del blocco e rivendicare le commissioni di transazione. Perché i minatori non scrivono una transazione per un migliaio di bitcoin invece della ricompensa giusta? Perché ogni nodo convalida i blocchi secondo le stesse regole. Una transazione di coinbase non valida renderebbe l'intero blocco

non valido, il che comporterebbe il rifiuto del blocco e, pertanto, tale transazione non diventerebbe mai parte del libro mastro. I minatori devono costruire un blocco perfetto, basato sulle regole condivise seguite da tutti i nodi, e collegarlo con una soluzione corretta alla prova di lavoro. Per fare ciò, spendono molta elettricità nel mining e, se imbrogliano, tutta l'elettricità e lo sforzo sono sprecati. Questo è il motivo per cui la convalida indipendente è una componente chiave del consenso decentralizzato.

Assemblando e Selezionando Catene di

Blocchi

Il passo finale nel meccanismo di consenso decentralizzato di bitcoin è l'assemblaggio di blocchi in catene e la selezione della catena con la maggior prova di lavoro. Una volta che un nodo ha convalidato un nuovo blocco, tenterà quindi di assemblare una catena collegando il blocco alla blockchain esistente.

I nodi mantengono tre tipi di blocchi: quelli connessi alla blockchain principale, quelli che formano rami che partono dalla blockchain principale (chain secondarie), ed infine, blocchi che non hanno un genitore presente nelle chain

conosciute (orfani). I blocchi non validi sono rifiutati non appena uno dei criteri di validazione fallisce e quindi non vengono inclusi in nessuna chain.

La "catena principale" in qualsiasi momento è la catena di blocchi valida a cui è associata la prova di lavoro più cumulativa. Nella maggior parte dei casi questa è anche la catena con il maggior numero di blocchi, a meno che non ci siano due catene di uguale lunghezza e una abbia più Proof-of-Work. La catena principale avrà anche rami con blocchi che sono "fratelli" ai blocchi sulla catena principale. Questi blocchi sono validi ma non fanno parte della catena principale. Sono

conservati per riferimento futuro, nel caso in cui una di quelle catene sia estesa per superare la catena principale in corso di lavoro. Nella prossima sezione (I Fork della Blockchain), vedremo come si verificano le catene secondarie a seguito di un'estrazione quasi simultanea di blocchi alla stessa altezza.

Quando viene ricevuto un nuovo blocco, un nodo proverà a inserirlo nella blockchain esistente. Il nodo esaminerà il campo "hash blocco precedente" del blocco, che è il riferimento al genitore del blocco. Quindi, il nodo tenterà di trovare quel genitore nella blockchain esistente. Il

più delle volte, il genitore sarà la "punta" della catena principale, il che significa che questo nuovo blocco estende la catena principale. Ad esempio, il nuovo blocco 277.316 ha un riferimento all'hash del suo blocco padre 277.315. La maggior parte dei nodi che ricevono 277.316 avranno già il blocco 277.315 come punta della loro catena principale e quindi collegheranno il nuovo blocco ed estenderanno uella catena.

A volte, come vedremo in I Fork della Blockchain, il nuovo blocco estende una catena che non è la catena principale. In tal caso, il nodo collegherà il nuovo blocco alla catena secondaria che si estende e quindi

confronterà il lavoro della catena secondaria con la catena principale. Se la catena secondaria ha un lavoro più cumulativo rispetto alla catena principale, il nodo si ricongiungerà sulla catena secondaria, il che significa che selezionerà la catena secondaria come nuova catena principale, rendendo la vecchia catena principale una catena secondaria. Se il nodo è un miner, costruirà un blocco estendendo questa nuova catena più lunga.

Se viene ricevuto un blocco valido e nessun genitore viene trovato nelle catene esistenti, quel blocco viene considerato un "orfano". I blocchi orfani vengono salvati nel pool di

blocchi orfani dove rimarranno fino alla ricezione del genitore. Una volta che il genitore è stato ricevuto e collegato nelle catene esistenti, l'orfano può essere estratto dal pool orfano e collegato al genitore, rendendolo parte di una catena. I blocchi orfani di solito si verificano quando due blocchi che sono stati estratti in un breve lasso di tempo sono ricevuti in ordine inverso (figlio prima del genitore).

Selezionando la catena con la maggior difficoltà, tutti i nodi raggiungono infine un consenso a livello di rete. Le discrepanze temporanee tra le catene sono risolte con l'aggiunta di ulteriori prove di lavoro, estendendo una delle

possibili catene. I nodi di mining "votano" con la loro potenza di calcolo scegliendo quale catena estendere estraendo il blocco successivo. Quando estraggono un nuovo blocco ed estendono la catena, il nuovo blocco rappresenta il loro voto.

Nella prossima sezione daremo un'occhiata a come discrepanze tra catene concorrenti (fork) sono risolte dalla selezione indipendente della catena di difficoltà piu lunga.

I Fork della Blockchain

Visto che la blockchain è una struttura dati decentralizzata, le diverse copie di essa non sono sempre identiche. I

blocchi potrebbero arrivare ai vari nodi in momenti differenti, questo fa si che i nodi abbiano diverse prospettive della blockchain. Per risolvere questo problema, ogni nodo seleziona e cerca di estendere sempre la catena dei blocchi che rappresenta la maggiore proof-of-work, conosciuta anche come la longest chain (la catena più lunga) o la catena più grande e con difficoltà cumulativa più alta. Sommando la difficoltà registrata in ogni blocco in una chain, un nodo può calcolare la quantità totale di proof-of-work che è stata spesa per creare quella chain. Fino a quando tutti i nodi selezionano la catena con difficoltà cumulativa più lunga, la rete globale bitcoin

eventualmente convergerà verso uno stato consistente. Le ramificazioni (fork) avvengono come inconsistenze temporanee tra le varie versioni della blockchain, che sono risolte da eventuali riconvergenze mano a mano che più blocchi sono aggiunti a uno dei rami.

TIP	Le fork della blockchain descritte in questa sezione si verificano naturalmente a causa dei ritardi di trasmissione nella rete globale. Vedremo anche le fork deliberatamente indotte più avanti in questo capitolo.

Nei prossimi diagrammi, seguiamo l’andamento di un evento "fork" attraverso la rete. Il diagramma è una rappresentazione semplificata della rete bitcoin. A scopo illustrativo, i diversi blocchi sono mostrati come forme diverse (stella, triangolo, triangolo capovolto, rombo), diffondendosi attraverso la rete. Ogni nodo nella rete è rappresentato come un cerchio.

Ogni nodo ha la propria prospettiva della blockchain globale. Poiché ogni nodo riceve blocchi dai suoi vicini, aggiorna la propria copia della blockchain, selezionando la catena di

lavoro più cumulativo-lavorativo. A scopo illustrativo, ciascun nodo contiene una forma che rappresenta il blocco che ritiene sia attualmente il vertice della catena principale. Quindi, se vedi una stella nel nodo, significa che il blocco a stella è la punta della catena principale, per quanto riguarda quel nodo.

Nel primo diagramma (Prima del fork—tutti i nodi hanno la stessa prospettiva), la rete ha una prospettiva unificata della blockchain, con il blocco a stella come la punta della catena principale.

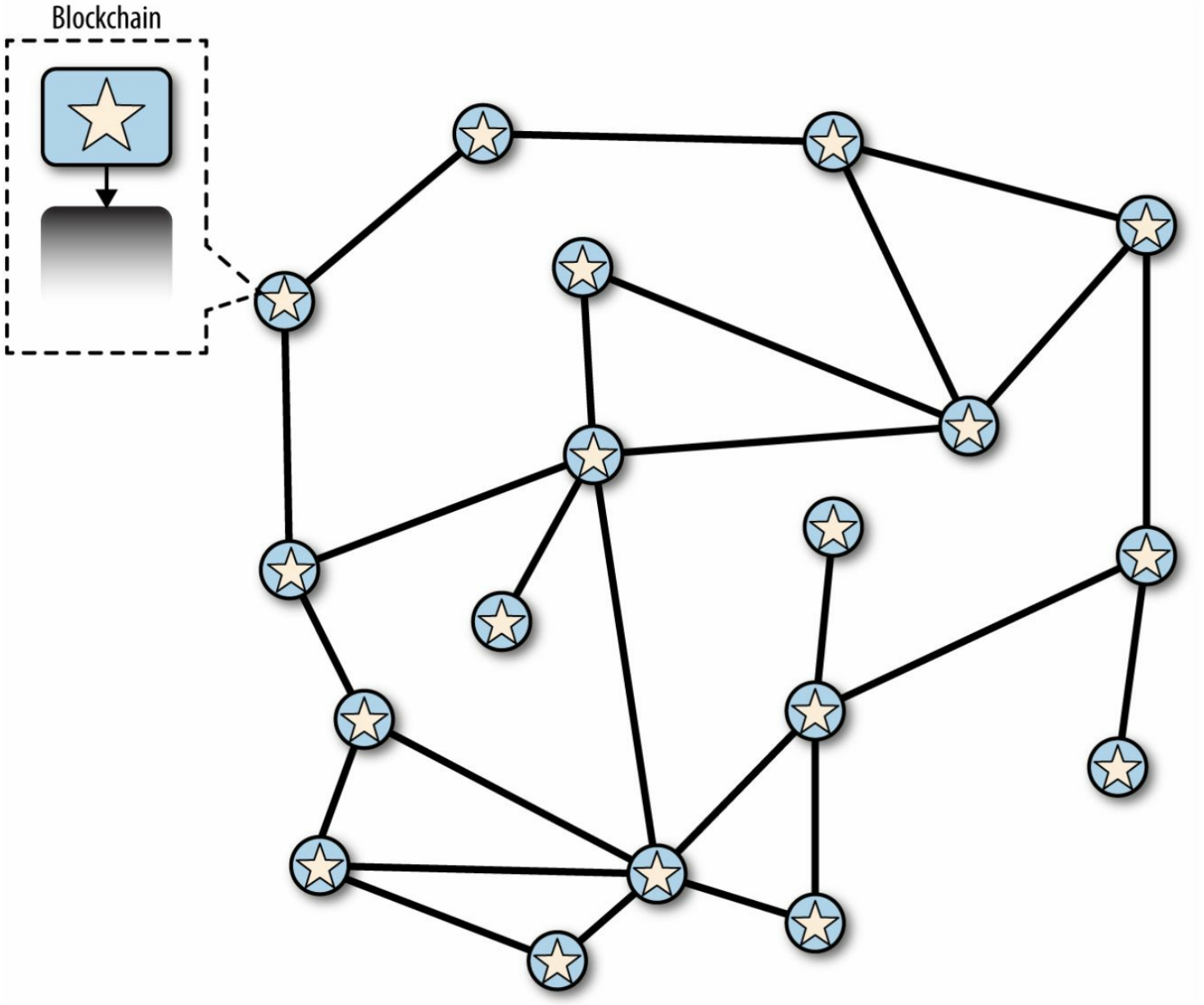


Figura 63. Prima del fork—tutti i nodi hanno la stessa prospettiva

Un "fork" si verifica ogni volta che ci sono due blocchi candidati in

competizione per formare la blockchain più lunga. Ciò si verifica in condizioni normali ogni volta che due minatori risolvono l'algoritmo di prova di lavoro entro un breve periodo di tempo l'uno dall'altro. Quando entrambi i minatori scoprono una soluzione per i rispettivi blocchi candidati, trasmettono immediatamente il proprio blocco "vincente" ai loro vicini immediati che iniziano a propagare il blocco attraverso la rete. Ogni nodo che riceve un blocco valido lo incorporerà nella sua blockchain, estendendo la blockchain di un blocco. Se in seguito tale nodo vede un altro blocco candidato che estende lo stesso genitore, collega il secondo candidato

su una catena secondaria. Di conseguenza, alcuni nodi "vedranno" prima un blocco candidato, mentre altri nodi vedranno emergere l'altro blocco candidato e due versioni concorrenti della blockchain.

In Visualizzazione di un'evento di fork di blockchain: due blocchi trovati simultaneamente, vediamo due minatori (Nodo X e Nodo Y) che estraggono due blocchi diversi quasi simultaneamente. Entrambi questi blocchi sono figli del blocco stella ed estendono la catena costruendo in cima al blocco a stella. Per aiutarci a rintracciarlo, uno viene visualizzato come un blocco a triangolo che origina dal Nodo X e l'altro è mostrato come

un blocco a triangolo capovolto originato dal Nodo Y.

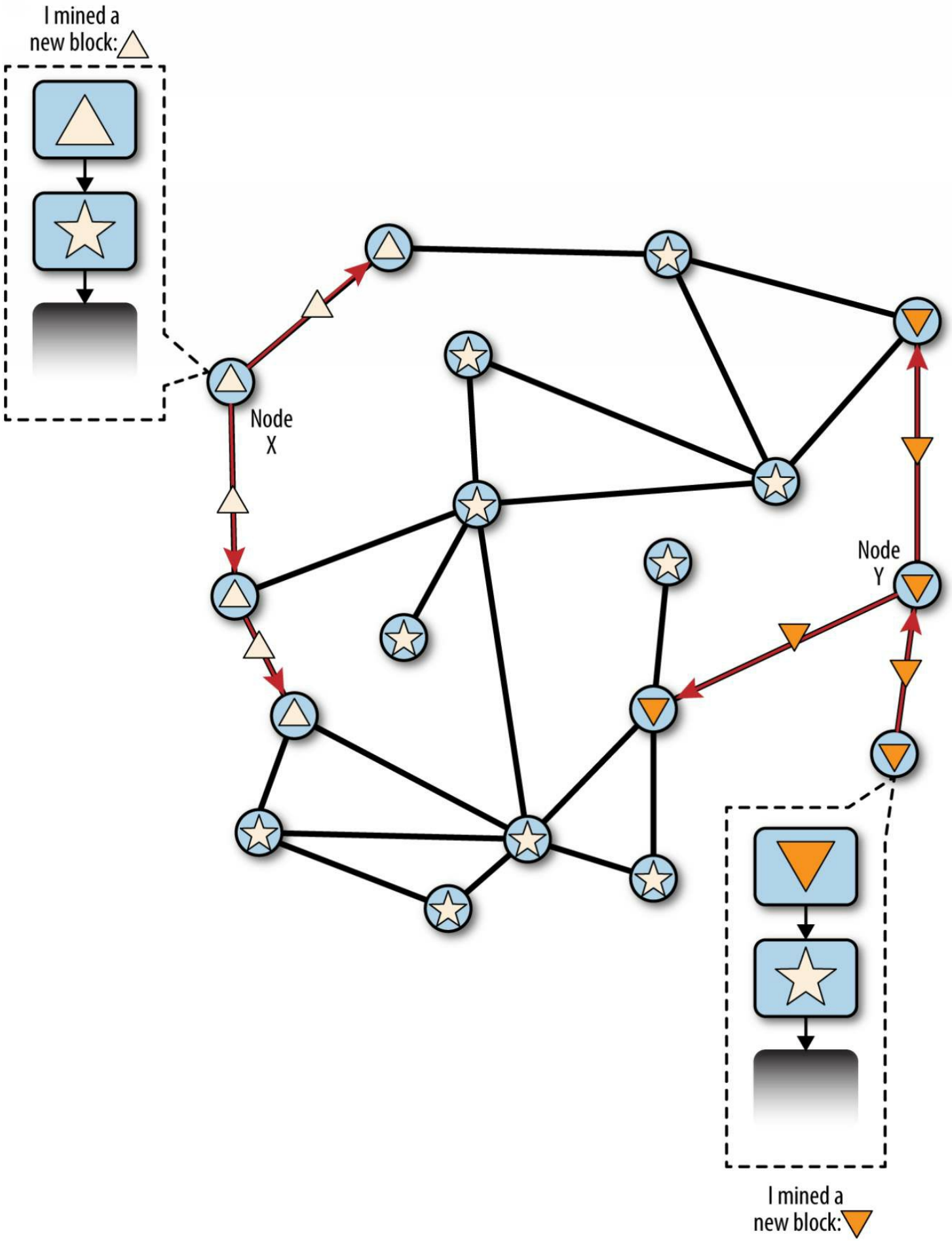

I mined a new block:
Node X
Node Y
I mined a new block:

Figura 64. Visualizzazione di un'evento di fork di blockchain: due blocchi trovati simultaneamente

Supponiamo, per esempio, che un Nodo minatore X trovi una soluzione Proof-of-Work per un "triangolo" di blocco che estende la blockchain, costruendo sopra la "stella" del blocco principale. Quasi contemporaneamente, il Nodo minatore Y che stava anch'esso estendendo la catena dal blocco "stella" trova una soluzione per il blocco "triangolo rovesciato", il suo blocco candidato. Ora, ci sono due possibili blocchi; uno che chiamiamo

"triangolo", originario nel nodo X; e uno che chiamiamo "triangolo rovesciato", originario nel nodo Y. Entrambi i blocchi sono validi, entrambi i blocchi contengono una soluzione valida per la Prova di lavoro, ed entrambi i blocchi estendono lo stesso genitore (blocco "stella"). Entrambi i blocchi probabilmente contengono la maggior parte delle stesse transazioni, con solo forse alcune differenze nell'ordine delle transazioni.

Mentre i due blocchi si propagano, alcuni nodi ricevono prima il blocco "triangolo" e alcuni ricevono prima il blocco "triangolo rovesciato". Come mostrato in Visualizzazione di

un’evento di fork di blockchain: due blocchi propagati, dividendo il network, la rete si divide in due diversi punti di vista della blockchain, un lato con un blocco rosso, l’altro con un blocco verde.

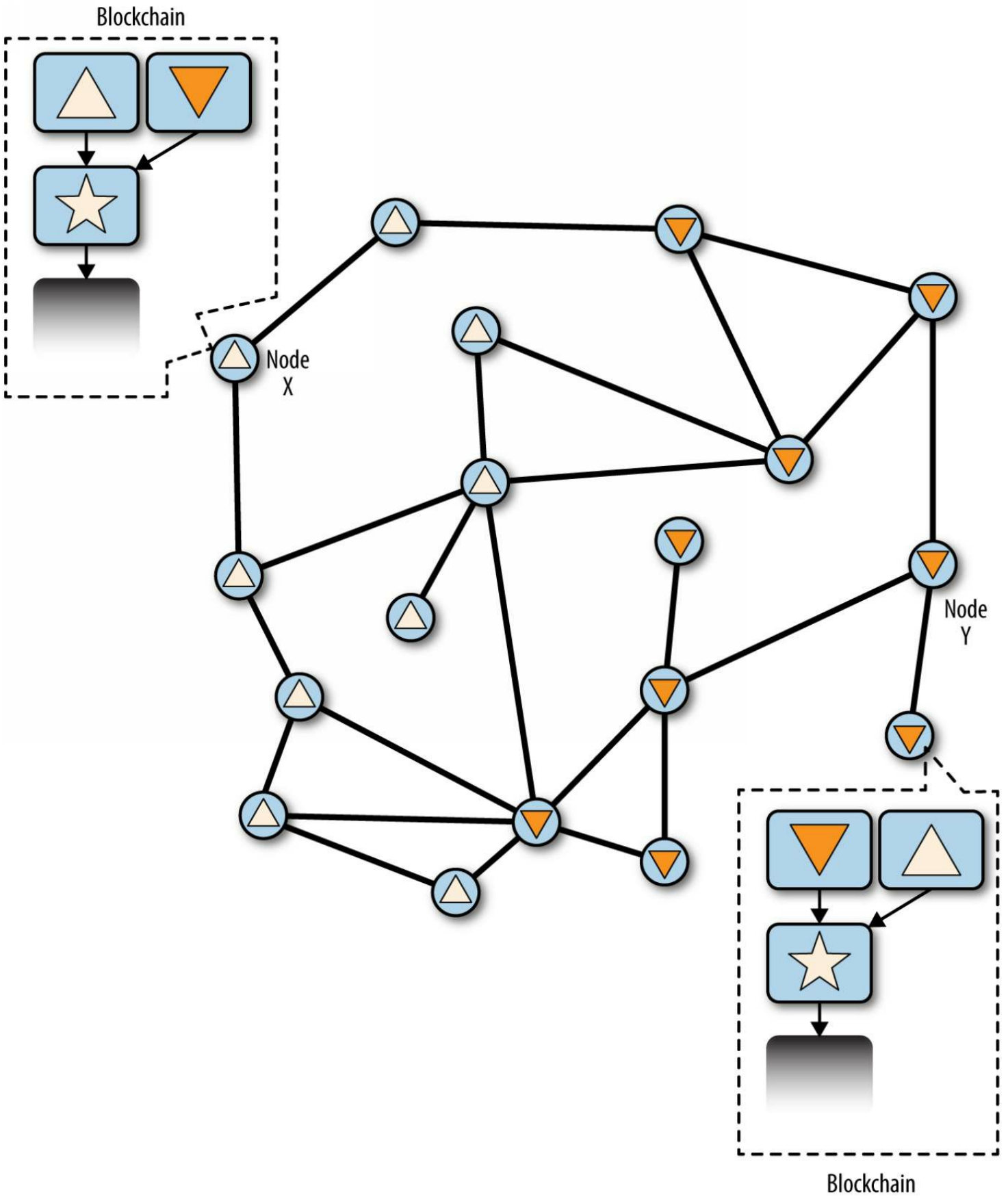


Figura 65. Visualizzazione di

un'evento di fork di blockchain: due blocchi propagati, dividendo il network

Nel diagramma, un "Nodo X" scelto a caso ha ricevuto per primo il blocco a triangolo e ha esteso la catena a stella con esso. Il nodo X ha selezionato la catena con il blocco "triangolo" come catena principale. Successivamente, il nodo X ha anche ricevuto il blocco "triangolo rovesciato". Poiché è stato ricevuto secondo, si presume che abbia "perso" la gara. Tuttavia, il blocco "triangolo rovesciato" non viene scartato. È collegato al genitore del blocco "stella" e forma una catena

secondaria. Mentre il Nodo X presuppone di aver correttamente selezionato la catena vincente, mantiene la catena "perdente" in modo che abbia le informazioni necessarie per ricostituire se la catena "perdente" finisce per "vincere".

Dall'altro lato della rete, il nodo Y costruisce una blockchain in base alla propria prospettiva della sequenza di eventi. Ha ricevuto per primo "triangolo capovolto" ed ha eletto quella catena come "vincitrice". Quando in seguito ha ricevuto il blocco "triangolo", lo ha collegato al genitore del blocco "stella" come catena secondaria.

Nessuno dei due lati è "corretto" o

"errato". Entrambe sono prospettive valide della blockchain. Solo col senno di poi prevarrà, in base a come queste due catene concorrenti vengono estese con un lavoro aggiuntivo.

I mining node la cui prospettiva assomiglia al Nodo X inizieranno immediatamente ad estrarre un blocco candidato che estende la catena con "triangolo" come suggerimento. Collegando "triangolo" come genitore del blocco candidato, votano con il loro potere di hashing. Il loro voto sostiene la catena che hanno eletto come la catena principale.

Qualsiasi nodo di mining la cui prospettiva assomigli al Nodo Y inizierà a costruire un nodo candidato

con "triangolo rovesciato" come genitore, estendendo la catena che credono sia la catena principale. E così via, la gara ricomincia.

I fork sono quasi sempre risolti in un blocco. Come parte della potenza di hashing della rete è dedicato alla costruzione di "triangolo" come genitore, un'altra parte della potenza di hashing è incentrata sulla costruzione di "triangolo rovesciato". Anche se il potere di hashing è diviso in modo quasi uniforme, è probabile che un gruppo di minatori troverà una soluzione e la propagherà prima che l'altro gruppo di minatori abbia trovato qualche soluzione. Diciamo, per esempio, che i minatori che si

trovano in cima a "triangolo" trovano un nuovo blocco "rombo" che estende la catena (ad esempio, stella-triangolo-rombo). Propagano immediatamente questo nuovo blocco e l'intera rete lo vede come una soluzione valida come mostrato in Visualizzazione di un'evento di fork di blockchain: un nuovo blocco estende un fork.

Tutti i nodi che hanno scelto "triangolo" come vincitore nel round precedente estenderanno semplicemente un ulteriore blocco alla catena. I nodi che hanno scelto "triangolo rovesciato" come vincitore, tuttavia, vedranno ora due catene: stella-triangolo-rombo e stella-

triangolo rovesciato. La catena stella-triangolo-rombo è ora più lunga (lavoro più cumulativo) rispetto all'altra catena. Di conseguenza, questi nodi imposteranno la catena triangolo-stella-rombo come catena principale e trasformeranno la catena stella-triangolo rovesciato in catena secondaria, come mostrato in Visualizzazione di un'evento di fork di blockchain: il network riconverge su di una nuova longest chain. Questa è una riconversione a catena, perché quei nodi sono costretti a rivedere la loro visione della blockchain per incorporare la nuova evidenza di una catena più lunga. Tutti i miner che lavorano sull'estensione della catena

stella-triangolo rovesciato ora interromperanno il lavoro perché il loro blocco candidato è un "orfano", poiché il suo genitore "triangolo rovesciato" non è più sulla catena più lunga. Le transazioni all'interno di "triangolo rovesciato" che non seguono "triangolo" vengono reinserite nel mempool per essere incluse nel blocco successivo e diventare parte della catena principale. L'intera rete riconfigura una singola blockchain a stella-triangolo-rombo, con "rombo" come ultimo blocco della catena. Tutti i minatori iniziano immediatamente a lavorare su blocchi candidati che fanno riferimento a "rombo" come genitore per estendere la catena stella-

triangolo-rombo.

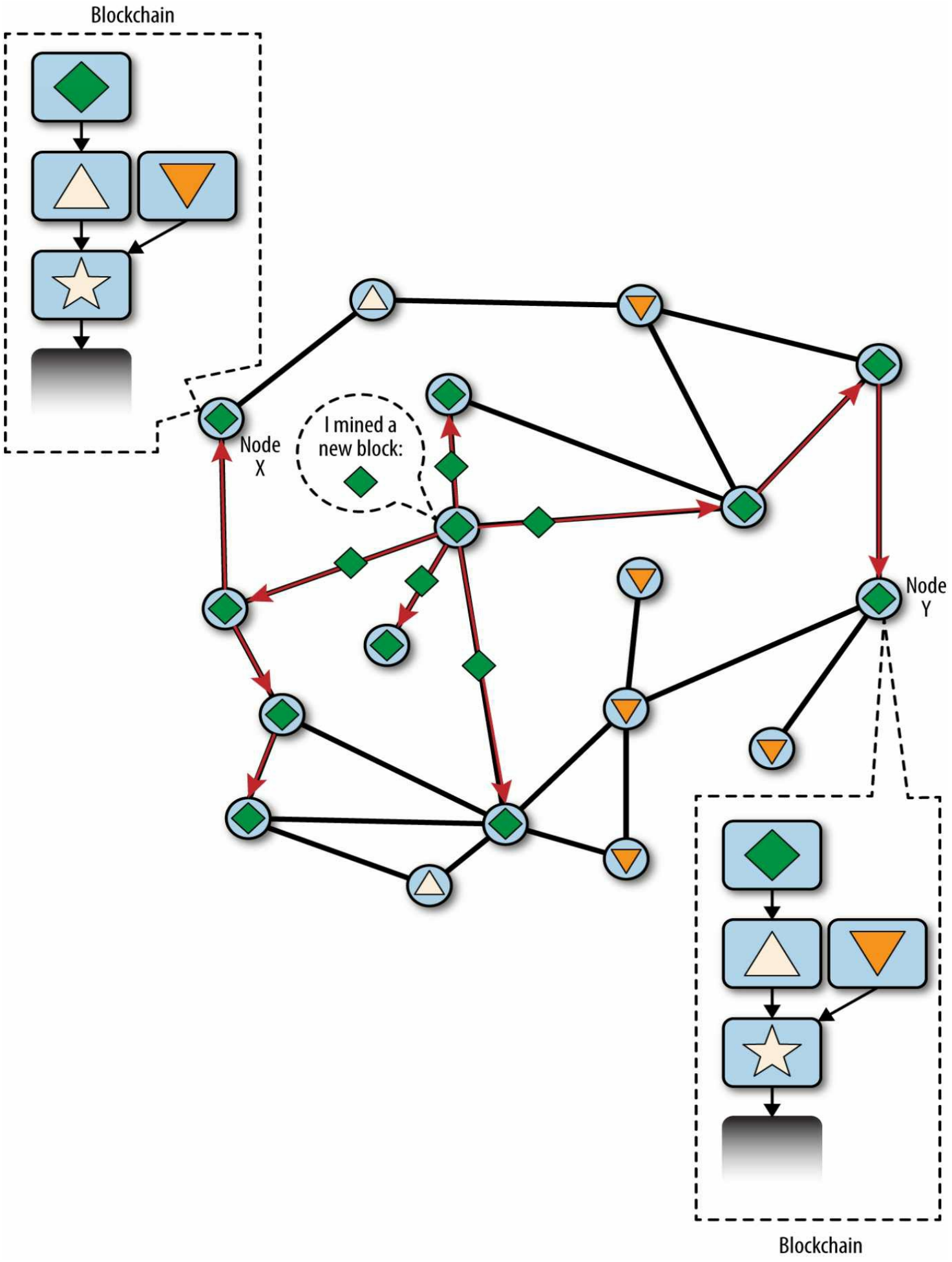

Blockchain
Node X
I mined a new block:
Node Y
Blockchain

Figura 66. Visualizzazione di un'evento di fork di blockchain: un nuovo blocco estende un fork

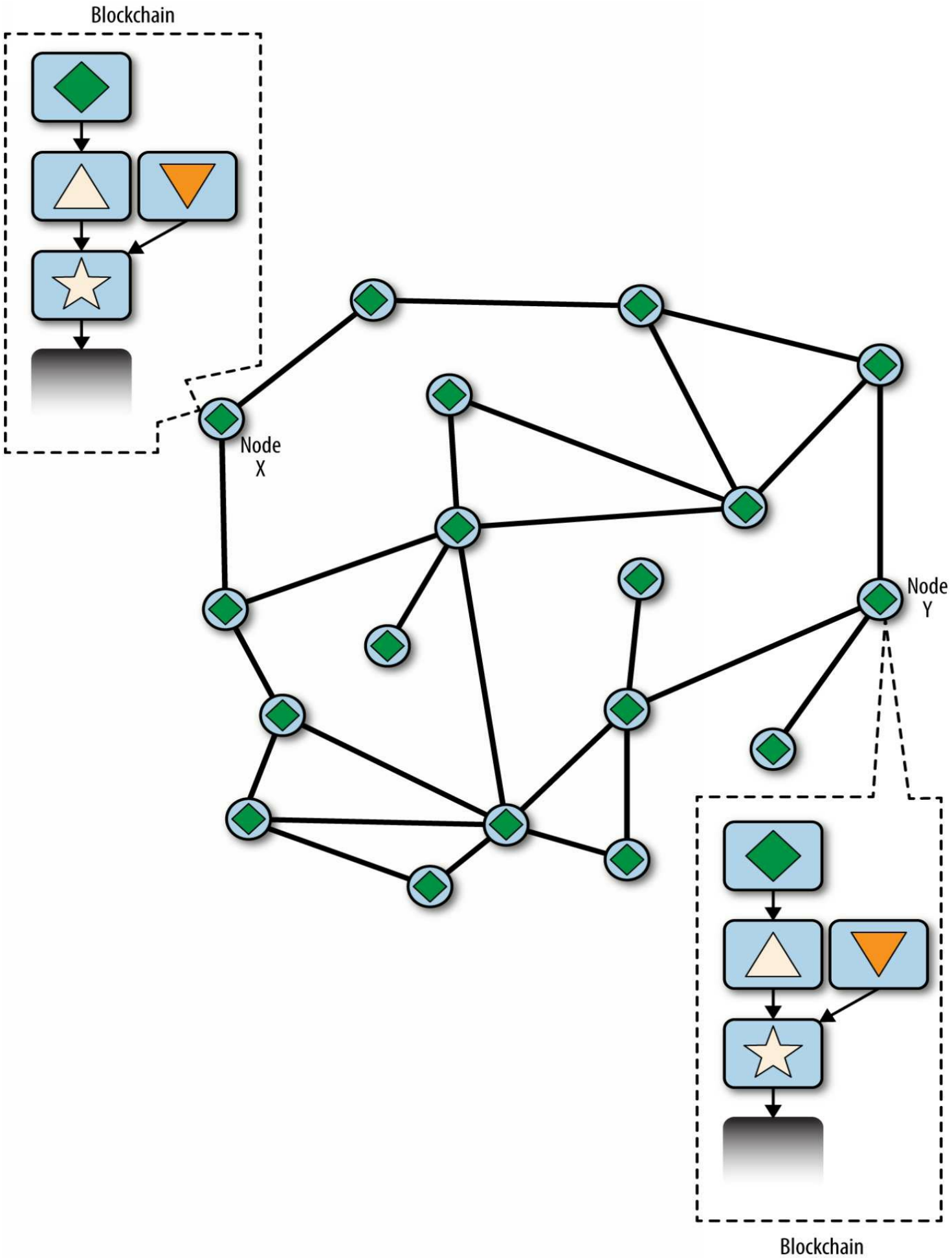

Blockchain
Node
X
Node
Y
Blockchain

Figura 67. Visualizzazione di un'evento di fork di blockchain: il network riconverge su di una nuova longest chain

È teoricamente possibile che un fork si estenda fino a due blocchi, se due blocchi si trovano quasi contemporaneamente da minatori su "lati" opposti di un fork precedente. Tuttavia, la possibilità che ciò accada è molto bassa. Mentre un fork a un blocco potrebbe verificarsi ogni settimana, una fork a due blocchi è

estremamente raro.

L'intervallo di blocco di Bitcoin di 10 minuti è un compromesso di progettazione tra i tempi di conferma rapidi (liquidazione delle transazioni) e la probabilità di un fork. Un tempo di blocco più veloce renderebbe le transazioni più veloci, ma porterà a fork sulla blockchain più frequenti, mentre un tempo di blocco più lento ridurrebbe il numero di fork, ma rallenterebbe le transazioni più lente.

Il Mining e la Gara di Hashing

Il mining di Bitcoin è un settore estremamente competitivo. Il potere di

hashing è aumentato esponenzialmente ogni anno dell'esistenza di bitcoin. Da qualche anno la crescita ha riflesso un cambiamento completo della tecnologia, come nel 2010 e nel 2011, quando molti minatori passarono dall'uso del mining della CPU a al mining con le GPU e successivamente al mining field programmable gate array (FPGA). Nel 2013 l'introduzione di il mining con ASIC ha portato a un altro gigantesco balzo in avanti nel settore del mining, collocando la funzione SHA256 direttamente su chip di silicio specializzati ai fini del mining. I primi chip di questo tipo potrebbero fornire più potenza di estrazione in un singolo

box rispetto all'intera rete bitcoin nel 2010.

La lista seguente mostra la potenza totale di hashing del network bitcoin, durante i primi cinque anni di operatività:

2009

0.5 MH/sec–8 MH/sec (16x di crescita)

2010

8 MH/sec–116 GH/sec (14,500x di crescita)

2011

116 GH/sec–9 TH/sec (78× di crescita)

2012

9 TH/sec–23 TH/sec (2.5x di crescita)

2013

23 TH/sec–10 PH/sec (450x di crescita)

2014

10 PH/sec–300 PH/sec (30× di crescita)

2015

300 PH/sec-800 PH/sec (2.66× di crescita)

2016

800 PH/sec-2.5 EH/sec (3.12× di crescita)

Nel grafico in Potenza di hashing totale, terahash per secondo, nel corso

di due anni, vediamo l'aumento della potenza hashing della rete bitcoin negli ultimi due anni. Come potete vedere, la competizione tra i minatori e la crescita del bitcoin ha comportato un aumento esponenziale della potenza di hashing (hash totale al secondo attraverso la rete).

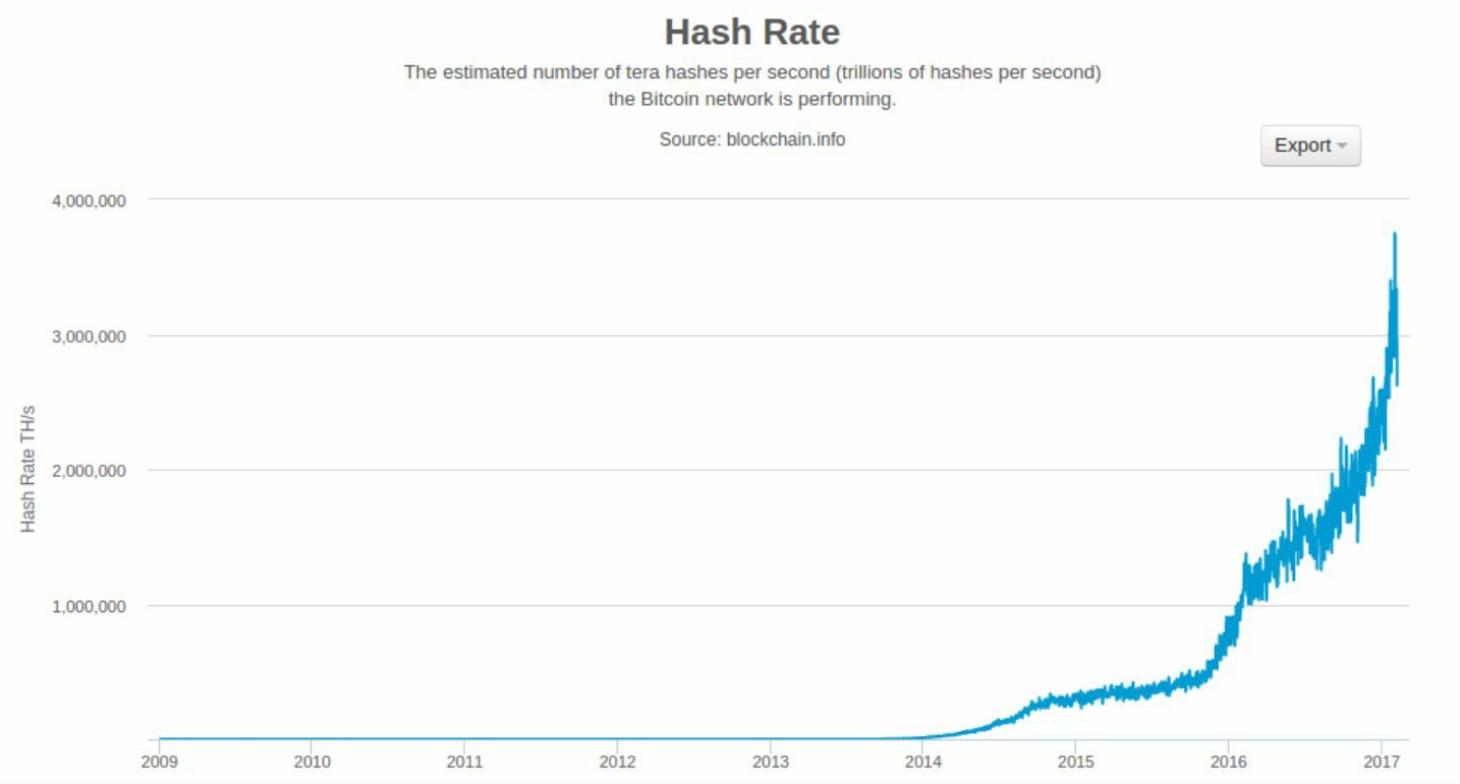


Figura 68. Potenza di hashing totale, terahash per

secondo, nel corso di due anni

Con l'esplosione della quantità di potenza di hashing applicata al mining bitcoin, la difficoltà è aumentata per controbilanciare. La metrica di difficoltà nella tabella mostrata in La metrica di difficoltà di mining di Bitcoin è misurata come rapporto tra la difficoltà attuale e la difficoltà minima (la difficoltà del primo blocco).

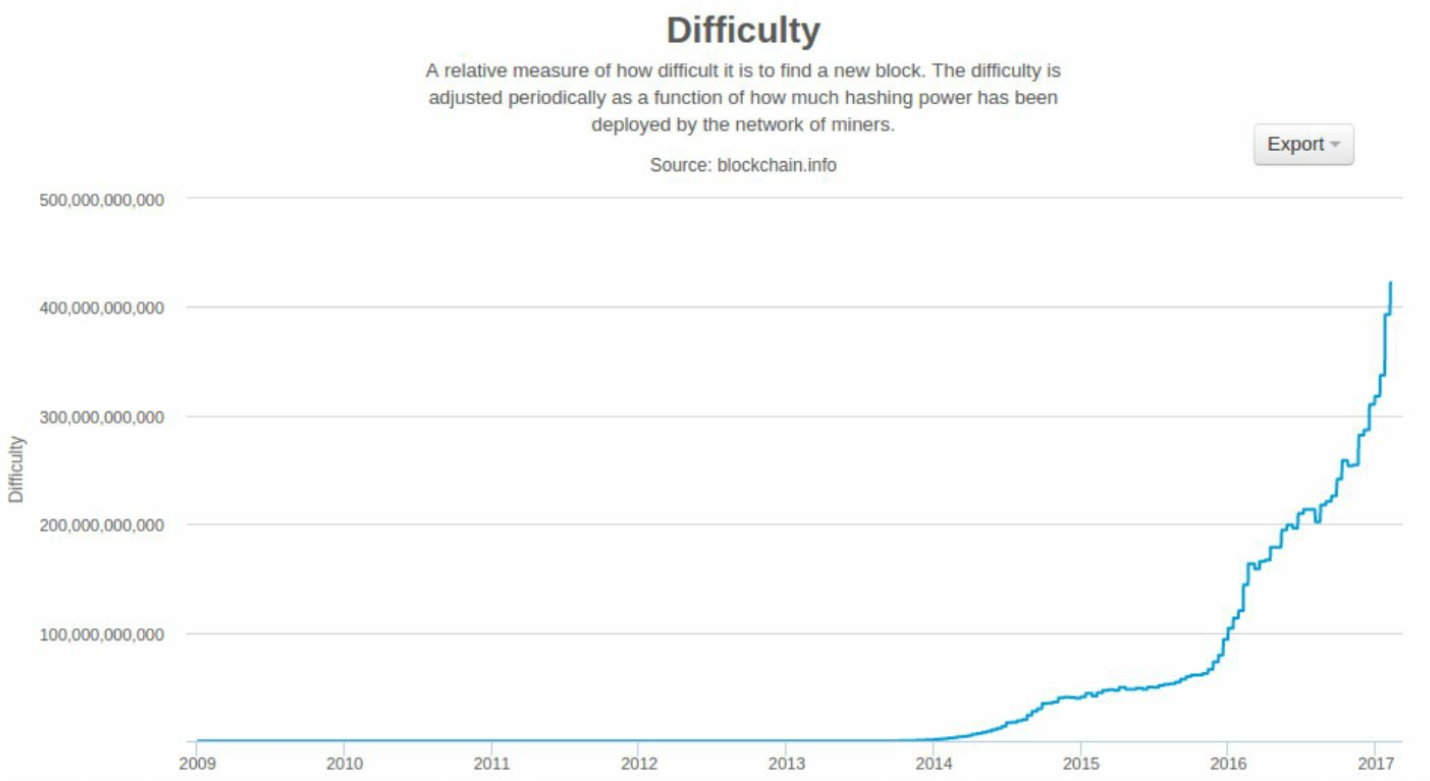


Figura 69. La metrica di difficoltà di mining di Bitcoin

Negli ultimi due anni, i chip ASIC per il mining sono diventati sempre più densi, avvicinandosi alla tecnologia di fabbricazione del silicio con una dimensione (risoluzione) di 16 nanometri (nm). Attualmente, i

produttori ASIC puntano a sorpassare i produttori di chip CPU generici, progettando chip con dimensioni di 14 nm, perché la redditività del settore del mining sta guidando questo settore ancora più rapidamente del calcolo generale. Non ci sono più balzi giganteschi nell'industria del mining del bitcoin, perché l'industria ha raggiunto la prima linea della Legge di Moore, che stabilisce che la densità di calcolo raddoppierà all'incirca ogni 18 mesi. Tuttavia, la potenza di mining della rete continua ad avanzare ad un ritmo esponenziale mentre la corsa per i chip ad alta densità è abbinata ad una gara per data center ad alta densità dove migliaia di questi chip possono

essere schierati. Non si tratta più di quanta estrazione può essere fatta con un solo chip, ma di quanti chip possono essere stipati in un edificio, continuando a dissipare il calore e fornendo una potenza adeguata.

La Soluzione del Nonce Extra

Dal 2012, il bitcoin mining si è evoluto per risolvere una limitazione fondamentale nella struttura dell'intestazione del blocco. Agli albori del bitcoin, un miner poteva trovare un blocco iterando attraverso il nonce fino a quando l'hash risultante era inferiore all'obiettivo. A mano a mano che aumentava la difficoltà, i

miner spesso ciclavano tutti i 4 miliardi di valori del nonce senza trovare un blocco. Tuttavia, questo è stato facilmente risolto aggiornando il timestamp del blocco per tenere conto del tempo trascorso. Poiché il timestamp fa parte dell'intestazione, la modifica consentirebbe ai miner di scorrere nuovamente i valori del nonce con risultati diversi. Una volta che la potenza di calcolo dell'hardware di mining ha superato i 4 GH/sec, tuttavia, questo approccio è diventato sempre più difficile perché i valori nonce sono stati esauriti in meno di un secondo. Poiché le attrezzature di mining ASIC iniziarono a spingere e quindi a superare la frequenza hash di

1 TH/sec, il software di mining aveva bisogno di più spazio per i valori nonce al fine di trovare blocchi validi. Il timestamp potrebbe essere allungato un po', ma spostandolo troppo lontano nel futuro potrebbe rendere il blocco invalido. Nell'intestazione del blocco era necessaria una nuova fonte di "modifica". La soluzione era usare la transazione coinbase come fonte di valori extra nonce. Poiché lo script coinbase può archiviare tra 2 e 100 byte di dati, i minatori hanno iniziato a utilizzare quello spazio come spazio extra nonce, consentendo loro di esplorare una gamma molto più ampia di valori di intestazione di blocco per trovare blocchi validi. La transazione

coinbase è inclusa nel merkle tree, il che significa che qualsiasi modifica nello script coinbase fa cambiare la merkle root. Otto byte di extra nonce, più i 4 byte di "standard" nonce permettono ai miner di esplorare un totale di 296 (8 seguito da 28 zeri) possibilità *al secondo* senza dover modificare il timestamp. Se, in futuro, i minatori potrebbero attraversare tutte queste possibilità, potrebbero quindi modificare il timestamp. C'è anche più spazio nello script coinbase per l'espansione futura dello spazio extra nonce.

Le Mining Pool (piscine)

In questo ambiente altamente

competitivo, i singoli miner che lavorano da soli (noti anche come minatori solisti) non hanno alcuna possibilità. La probabilità che trovino un blocco per compensare i costi dell'elettricità e dell'hardware è talmente bassa da rappresentare un gioco d'azzardo, come giocare alla lotteria. Persino il più più veloce sistema di estrazione ASIC, non riesce a stare al passo con i sistemi commerciali che accumulano decine di migliaia di questi chip in magazzini giganteschi vicino a centrali idroelettriche. I miner ora collaborano per formare mining pools, unendo il loro potere di hashing e condividendo la ricompensa tra migliaia di

partecipanti. Partecipando ad un pool, i miner ricevono una quota minore della ricompensa complessiva, ma in genere vengono premiati ogni giorno, riducendo l'incertezza.

Diamo un'occhiata ad un esempio specifico. Supponiamo che un minatore abbia acquistato hardware di mining con una velocità di hashing combinata di 14.000 gigahash al secondo (GH/s) o 14 TH/s. Nel 2017 questa attrezzatura costa circa $2,500 USD. L'hardware consuma 1375 watt (1,3 kW) di elettricità quando è in funzione, 33 kW-ore al giorno, a un costo da $1 a $2 al giorno a tariffe elettriche molto basse. Con l'attuale difficoltà di bitcoin, il minatore sarà in

grado di estrarre un blocco da solo circa una volta ogni 4 anni. Come calcoliamo questa probabilità? Si basa su un tasso di hashing a livello di rete di 3 EH/sec (nel 2017) e sulla velocità del minatore di 14 TH/sec:

- $P = (14 * 10^{12} / 3 * 10^{18}) * 210240 = 0.98$

... dove 21240 è il numero di blocchi in quattro anni. Il minatore ha una probabilità del 98% di trovare un blocco ogni quattro anni, in base al tasso di hash globale all'inizio del periodo.

Se il minatore trova un blocco unico in quel lasso di tempo, il pagamento di 12.5 bitcoin, a circa $1.000 per

bitcoin, si tradurrà in un unico pagamento di $12.500, che produrrà un profitto netto di circa $7.000. Tuttavia, la possibilità di trovare un blocco in un periodo di 4 anni dipende dalla fortuna del minatore. Potrebbe trovare due blocchi in 4 anni e realizzare un profitto molto grande. Oppure potrebbe non trovare un blocco per 5 anni e soffrire una perdita finanziaria più grande. Ancora peggio, è probabile che la difficoltà dell'algoritmo bitcoin Proof-of-Work aumenti significativamente in quel periodo, al ritmo attuale di crescita della potenza di hashing, il che significa che il minatore ha, al massimo, un anno per arrivare a break

even prima che l'hardware diventi effettivamente obsoleto e debba essere sostituito da hardware da mining più potente. Se questo minatore partecipa a una mining pool, invece di aspettare un guadagno eccezionale di 12.500 $ una volta ogni quattro anni, sarà in grado di guadagnare circa dai $50 ai $60 a settimana. I pagamenti regolari da una mining pool lo aiuteranno ad ammortizzare il costo dell'hardware e dell'elettricità nel tempo senza correre un enorme rischio finanziario. L'hardware sarà ancora obsoleto in uno o due anni e il rischio è ancora elevato, ma le entrate sono regolari e affidabili per tutto quel periodo. Dal punto di vista finanziario ciò ha senso

solo con un costo dell'elettricità molto basso (meno di 1 centesimo per kW-ora) e solo su una scala molto ampia.

Le mining pool coordinano molte centinaia o migliaia di miner, attraverso protocolli specializzati per le mining pool. I singoli miner configurano le proprie apparecchiature di mining per connettersi a un server della pool, dopo aver creato un account. Il loro hardware di mining rimane connesso al server della pool durante l'estrazione, sincronizzando i loro sforzi con gli altri miner. Così, i miner della medesima pool condividono lo sforzo di estrazione di un blocco e quindi ne condividono la ricompensa.

I blocchi vincenti pagano la ricompensa a un indirizzo bitcoin della pool, piuttosto che a singoli minatori. Il server della pool eseguirà periodicamente i pagamenti agli indirizzi bitcoin dei singoli miner, una volta che la loro quota dei premi abbia raggiunto una determinata soglia. In genere, il server di pool addebita una percentuale dei premi per fornire il servizio di mining pool.

I minatori che partecipano a un pool dividono il lavoro di ricerca di una soluzione per un blocco candidato, guadagnando "azioni" per il loro contributo di calcolo. Il pool di mining imposta un obiettivo di difficoltà inferiore per guadagnare una

condivisione, in genere più di 1.000 volte più semplice della difficoltà della rete bitcoin. Quando qualcuno nel pool riesce a estrarre un blocco, il premio viene versato al pool e poi condiviso con tutti i miner in proporzione al numero di azioni che hanno ottenuto contribuendo allo sforzo.

Le piscine sono aperte a tutti i miner, grandi o piccoli, professionisti o dilettanti. Un pool avrà quindi alcuni partecipanti con una singola piccola macchina, e altri con un garage pieno di hardware di fascia alta. Alcuni eseguiranno attività di mining con poche decine di kilowatt di elettricità, altri gestiranno un data center che

consuma un megawatt di energia. Come fa un pool minerario a misurare i singoli contributi, in modo da distribuire equamente i premi, senza la possibilità di barare? La risposta è usare l'algoritmo di proof of work di bitcoin per misurare il contributo di ciascun miner, ma impostarlo a una difficoltà inferiore in modo che anche i minatori della pool vincano una quota abbastanza frequentemente in modo che valga la pena contribuire alla pool. Impostando una difficoltà inferiore per guadagnare quote, la pool misura la quantità di lavoro svolto da ciascun miner. Ogni volta che un minatore della piscina trova un hash dell'header del blocco inferiore alla

difficoltà del pool, dimostra di aver eseguito il lavoro di hashing per trovare quel risultato. Ancora più importante, il lavoro per trovare le azioni contribuisce, in un modo statisticamente misurabile, allo sforzo complessivo di trovare un hash più basso rispetto all'obiettivo della rete bitcoin. Migliaia di miner che cercano di trovare gli hash di basso valore alla fine ne troveranno uno abbastanza basso da soddisfare il target richiesto della rete bitcoin.

Torniamo all'analogia di un gioco di dadi. Se i giocatori lanciano i dadi con l'obiettivo di ottenere un numero inferiore a quattro (la difficoltà generale della rete), una pool

imposterà un obiettivo più facile, contando quante volte i giocatori della pool sono riusciti a ottenere meno di otto. Quando i giocatori della pool lanciano meno di otto (l'obiettivo di condivisione della pool), guadagnano delle quote, ma non vincono la partita perché non raggiungono l'obiettivo di gioco (meno di quattro). I giocatori raggiungeranno più facilmente l'obiettivo della pool, guadagnandoli in modo molto regolare, anche quando non raggiungono l'obiettivo più difficile, cioè vincere la partita. Ogni tanto, uno dei giocatori della pool lancia i dadi ottenendo un numero inferiore a quattro e la pool ottiene quindi la ricompensa. I guadagni

possono essere distribuiti ai giocatori della pool in base alle azioni che hanno guadagnato. Anche se l'obiettivo inferiore a otto non permette di vincere, è un modo giusto per misurare i lanci di dadi dei singoli giocatori, e occasionalmente produce un tiro di meno di quattro (ndt che permette alla pool di vincere la partita).

In modo analogo, una mining pool imposterà una difficoltà di pool che garantirà che, un singolo miner del pool, possa trovare l'hash del blocco abbastanza spesso, con una difficoltà del pool inferiore (ndt a quella della rete bitcoin), guadagnando delle quote. Di tanto in tanto, uno di questi tentativi

produrrà un hash del blocco inferiore al target della rete bitcoin, rendendolo un blocco valido con la conseguente vittoria dell'intero pool.

Le Pool Gestite

Molti mining pool sono "gestiti", ovvero c'è una società o individuo che gira un pool server. Il proprietario del pool server e' chiamato il *pool operator* che addebita una una commissione percentuale sui guadagni ai miner del pool.

Il server della pool esegue un software specializzato e un protocollo di mining pool che coordina le attività dei miner appartenteni alla stessa pool. Il server di pool è inoltre connesso a uno o più

full node della rete bitcoin e ha accesso diretto a una copia completa del database della blockchain. Ciò consente al server di pool di convalidare blocchi e transazioni per conto dei miner della pool, sollevandoli dal carico di esecuzione di un nodo completo. Per i miner che partecipano ad una pool, questa è un fattore importante, poiché un nodo completo richiede un computer dedicato con almeno 100-150 GB di memoria permanente (hard disk) e almeno 2-4 GB di memoria (RAM). Inoltre, il software bitcoin in esecuzione sul nodo completo deve essere monitorato, gestito e aggiornato frequentemente. Qualsiasi fermo

macchina causato da una mancanza di manutenzione o mancanza di risorse danneggerà la redditività del miner. Per molti miner, la possibilità di estrarre senza un nodo completo è un altro grande vantaggio che si ottiene unendosi ad una pool.

I miner della pool si collegano al server della della pool alla quale appartengono utilizzando un protocollo di mining come Stratum (STM) o GetBlockTemplate (GBT). Un vecchio standard chiamato GetWork (GWK) è diventato obsoleto dalla fine del 2012, perché non supporta facilmente l'estrazione a tassi di hash superiori a 4 GH/s. Entrambi i protocolli STM e GBT creano blocchi *templates* che

contengono un modello di un'intestazione di blocco candidato. Il server di pool costruisce un blocco candidato aggregando le transazioni, aggiungendo una transazione coinbase (con spazio extra nonce), calcolando la merkle root e collegando l'hash del blocco precedente. L'intestazione del blocco candidato viene quindi inviata a ciascuno dei miner del pool come modello. Ogni pool minatore quindi utilizza il modello di blocco, a una difficoltà inferiore rispetto alla difficoltà della rete bitcoin, e invia eventuali risultati di successo al server della pool per guadagnare azioni.

Peer-to-peer mining pool

(P2Pool)

Queste pool creano la possibilità di cheating da parte dell'operatore del pool, che potrebbe indirizzare lo sforzo del pool verso le transazioni a doppia spesa o i blocchi invalidati (vedere Attacchi al Consenso). Inoltre, i server di pool centralizzati rappresentano un singolo punto di errore. Se il server della piscina è inattivo o viene rallentato da un attacco denial-of-service, i minatori della piscina non possono estrarlo. Nel 2011, per risolvere questi problemi di centralizzazione, è stato proposto e implementato un nuovo metodo di mining pool: P2Pool è un pool di mining peer-to-peer, senza un

operatore centrale.

P2Pool funziona decentralizzando le funzioni del server di pool, implementando un sistema parallelo simile a blockchain chiamato a catena condivisa. Una catena di azioni è una blockchain in esecuzione ad una difficoltà inferiore rispetto alla blockchain bitcoin. La catena condivisa consente ai pool di miner di collaborare in un pool decentralizzato, estraendo le azioni della catena ad un tasso di un blocco ogni 30 secondi. Ciascuno dei blocchi della catena di azioni registra un premio proporzionale per i minatori del pool che contribuiscono al lavoro, portando avanti le azioni dal precedente blocco.

Quando uno dei blocchi condivisi raggiunge anche l'obiettivo di difficoltà della rete bitcoin, viene propagato e incluso nella blockchain bitcoin, ricompensando tutti i minatori del pool che hanno contribuito a tutte le azioni che hanno preceduto il blocco azionario vincente. Essenzialmente, invece di un server di pool che tiene traccia delle azioni e dei premi del miner, la catena di condivisione consente a tutti i minatori di pool di tenere traccia di tutte le azioni utilizzando un meccanismo di consenso decentralizzato come il meccanismo di consenso blockchain del bitcoin.

Il mining di P2Pool è più complesso

del mining pool (ndt tradizionale) perché richiede che i miner abbiano un computer dedicato con sufficiente spazio su disco, memoria e larghezza di banda Internet per supportare un intero nodo bitcoin e il software del nodo P2Pool. I miniport P2Pool collegano il loro hardware di mining al loro nodo P2Pool locale, che simula le funzioni di un server di pool inviando modelli di blocchi all'hardware di mining. Su P2Pool, i singoli miner del pool costruiscono i propri blocchi candidati, aggregando le transazioni in modo molto simile ai singoli miner, ma in seguito collaborano alla catena di condivisione. P2Pool è un approccio

ibrido che ha il vantaggio di pagamenti molto più granulari rispetto al mining solista, ma senza dare troppo controllo a un operatore di pool come i pool gestiti.

Anche se P2Pool riduce la concentrazione della potenza da parte degli operatori del pool di mining, è presumibilmente vulnerabile agli attacchi del 51% contro la stessa catena di azioni. Un'adozione molto più ampia di P2Pool non risolve il problema di attacco del 51% per bitcoin stesso. Piuttosto, P2Pool rende il bitcoin più solido nel complesso, come parte di un ecosistema minerario diversificato.

Attacchi al Consenso

Il meccanismo di consenso di Bitcoin è, almeno teoricamente, vulnerabile agli attacchi di miner (o pool) che tentano di usare il loro potere di hashing in modo disonesto o distruttivo. Come abbiamo visto, il meccanismo del consenso dipende dal fatto che la maggioranza dei miner agisca onestamente per interesse personale. Tuttavia, se un miner o un gruppo di miner, può raggiungere una quota significativa del potere di calcolo, essi possono attaccare il meccanismo del consenso in modo da interrompere la sicurezza e la disponibilità della rete bitcoin.

È importante notare che gli attacchi di consenso possono influenzare solo il consenso futuro o, nel migliore dei casi, il passato più recente (decine di blocchi). Il registro di Bitcoin diventa sempre più immutabile con il passare del tempo. Mentre in teoria, un fork può essere raggiunto a qualsiasi profondità, nella pratica, la potenza di calcolo necessaria per forzare un fork molto profondo è immensa, rendendo i vecchi blocchi praticamente immutabili. Anche gli attacchi di consenso non influiscono sulla sicurezza delle chiavi private e dell'algoritmo di firma (ECDSA). Un attacco di consenso non può sottrarre bitcoin, spendere bitcoin senza firme,

reindirizzare bitcoin o modificare in altro modo transazioni passate o informazioni di proprietà. Gli attacchi di consenso possono interessare solo i blocchi più recenti e causare interruzioni di tipo denial-of-service sulla creazione di blocchi futuri.

Uno scenario di attacco contro il meccanismo del consenso è chiamato "attacco del 51%". In questo scenario un gruppo di minatori, controllando una maggioranza (51%) del potere di hashing della rete totale, collude per attaccare bitcoin. Con la possibilità di estrarre la maggior parte dei blocchi, i minatori che tentano un attacco possono causare "fork" deliberati nelle transazioni blockchain e double-

spending o eseguire attacchi denial-of-service contro specifiche transazioni o indirizzi. Un fork/double spending attack è quello in cui l'attaccante fa sì che i blocchi precedentemente confermati vengano invalidati mediante la biforcazione sotto di essi e la ricomposizione in una catena alternativa. Con una potenza di calcolo sufficiente, un utente malintenzionato può invalidare sei o più blocchi di fila, causando l'annullamento delle transazioni considerate immutabili (sei conferme). Tieni presente che una doppia spesa può essere eseguita solo sulle transazioni dell'hacker, per le quali l'attaccante può produrre una firma valida. Doppio-spendere le

proprie transazioni è redditizio se invalidando una transazione l'attaccante può ottenere un pagamento non reversibile o un prodotto senza pagarlo.

Esaminiamo un esempio pratico di un attacco del 51%. Nel primo capitolo, abbiamo esaminato una transazione tra Alice e Bob per una tazza di caffè. Bob, il proprietario del caffè, è disposto ad accettare pagamenti per tazze di caffè senza attendere conferma (estrazione in un blocco), perché il rischio di una doppia spesa per una tazza di caffè è basso rispetto alla convenienza del servizio clienti rapido . Questo è simile alla pratica dei coffee shop che accettano pagamenti

con carta di credito senza firma per importi inferiori a $ 25, perché il rischio di un chargeback di una carta di credito è basso mentre il costo di ritardare la transazione per ottenere una firma è relativamente più grande. Al contrario, la vendita di un articolo più costoso per bitcoin comporta il rischio di un attacco a doppia spesa, in cui l'acquirente trasmette una transazione concorrente che spende gli stessi input (UTXO) e annulla il pagamento al commerciante. Un attacco a doppia spesa può avvenire in due modi: o prima che una transazione sia confermata, o se l'attaccante sfrutta un fork blockchain per annullare diversi blocchi. Un attacco del 51%

consente agli aggressori di raddoppiare le proprie transazioni nella nuova catena, annullando la transazione corrispondente nella vecchia catena.

Nel nostro esempio, l'aggressore malvagio Mallory si reca alla galleria di Carol e acquista un bellissimo dipinto trittico raffigurante Satoshi Nakamoto come Prometeo. Carol vende dipinti "The Great Fire" per $ 250.000 in bitcoin, a Mallory. Invece di aspettare sei o più conferme sulla transazione, Carol avvolge e consegna i dipinti a Mallory dopo una sola conferma. Mallory lavora con un complice, Paul, che gestisce una grande mining pool e il complice

lancia un attacco del 51% non appena la transazione di Mallory è inclusa in un blocco. Paul ordina al pool di miner di minare nuovamente un blocco con la stessa altezza del blocco contenente la transazione di Mallory, sostituendo il pagamento di Mallory a Carol con una transazione che duplica lo stesso input del pagamento di Mallory. La transazione a doppia spesa consuma lo stesso UTXO e la ripaga sul portafoglio di Mallory, invece di pagarla a Carol, essenzialmente permettendo a Mallory di mantenere il bitcoin. Paul quindi indirizza la pool a estrarre un ulteriore blocco, in modo da rendere la catena contenente la transazione a doppia

spesa più lunga della catena originale (causando una fork sotto il blocco contenente la transazione di Mallory). Quando la biforcazione della blockchain si risolve a favore della nuova (più lunga) catena, la transazione a doppio spesa sostituisce il pagamento originale a Carol. A Carol mancano ora i tre dipinti e inoltre non ha ottenuto il pagamenti con bitcoin. Durante tutta questa attività, i partecipanti alla pool di Paul potrebbero rimanere beatamente inconsapevoli del tentativo di doppia spesa, perché sono miner automatizzati e non possono monitorare ogni transazione o blocco.

Per proteggersi da questo tipo di

attacco, un commerciante che vende oggetti di grande valore deve attendere almeno sei conferme prima di dare il prodotto all'acquirente. In alternativa, il commerciante deve utilizzare un conto di deposito a garanzia multi-firma, in attesa di diverse conferme dopo che l'account di deposito a garanzia è stato finanziato. Più conferme si accumulano, più diventa difficile invalidare una transazione con un attacco del 51%. Per gli articoli di valore elevato, il pagamento tramite bitcoin sarà comunque conveniente ed efficiente anche se l'acquirente deve attendere 24 ore per la consegna, il che corrisponderebbe a circa 144 conferme.

Oltre a un attacco a doppia spesa, l'altro scenario per un attacco di consenso consiste nel negare il servizio a partecipanti specifici di bitcoin (specifici indirizzi bitcoin). Un attaccante con la maggioranza del potere di mining può semplicemente ignorare transazioni specifiche. Se sono inclusi in un blocco estratto da un altro miner, l'utente malintenzionato può deliberatamente eseguire il fork e minare nuovamente quel blocco, escludendo di nuovo le transazioni specifiche. Questo tipo di attacco può comportare un rifiuto prolungato del servizio contro un indirizzo specifico o un insieme di indirizzi per tutto il tempo in cui l'attaccante controlla la

maggior parte della potenza di hashing.

Nonostante il suo nome, lo scenario di attacco del 51% non richiede in realtà il 51% della potenza di hashing. In realtà, un simile attacco può essere tentato con una percentuale minore della potenza di hashing. La soglia del 51% è semplicemente il livello al quale un tale attacco è quasi garantito per avere successo. Un attacco di consenso è essenzialmente un tiro alla fune per il prossimo blocco e il gruppo "più forte" ha più probabilità di vincere. Con meno potere di hashing, la probabilità di successo è ridotta, perché altri minatori controllano la generazione di alcuni blocchi con la loro "onesta" potenza di calcolo. Un

modo per vederlo è che più potere di hashing ha un attaccante, più lungo è il fork che può creare deliberatamente, più blocchi nel recente passato possono essere invalidati, o più blocchi in futuro potranno essere controllati. I gruppi di ricerca sulla sicurezza hanno utilizzato modelli statistici per affermare che vari tipi di attacchi di consenso sono possibili con un minimo del 30% della potenza di hashing.

Il massiccio aumento della potenza di hashing totale ha probabilmente reso i bitcoin impermeabili agli attacchi di un singolo miner. Non esiste un modo per un miner solitario di controllare più di una piccola percentuale della

potenza di estrazione totale. Tuttavia, la centralizzazione del controllo causata dalle mining pool ha introdotto il rischio di attacchi di profitto da parte di un gestore di pool. L'operatore del pool, in un pool gestito, controlla la costruzione dei blocchi candidati e controlla anche quali transazioni sono incluse. Ciò conferisce al gestore del pool, il potere di escludere le transazioni o introdurre transazioni a doppia spesa. Se tale abuso di potere è fatto in modo limitato e sottile, un operatore di pool potrebbe teoricamente trarre profitto da un attacco di consenso senza essere notato.

Tuttavia, non tutti gli aggressori

saranno motivati dal profitto. Uno scenario di potenziale attacco è dove un attaccante intende disturbare la rete bitcoin senza la possibilità di trarre profitto da tale disturbo. Un attacco malevolo mirato a paralizzare bitcoin richiederebbe enormi investimenti e pianificazione segreta, ma potrebbe essere probabilmente lanciato da un attaccante ben finanziato e molto probabilmente sponsorizzato dallo stato. In alternativa, un aggressore ben finanziato potrebbe attaccare il consenso di bitcoin accumulando simultaneamente hardware di data mining, compromettendo gli operatori di pool e attaccando altri pool con denial-of-service. Tutti questi scenari

sono teoricamente possibili, ma sempre più impraticabili in quanto la potenza di hashing complessiva della rete bitcoin continua a crescere esponenzialmente.

Indubbiamente, un serio attacco di consenso eroderebbe la fiducia nel bitcoin a breve termine, causando probabilmente un significativo calo dei prezzi. Tuttavia, la rete bitcoin e il software sono in continua evoluzione, quindi gli attacchi di consenso verrebbero affrontati con contromisure immediate da parte della comunità bitcoin, rendendo il bitcoin più solido, subdolo e robusto che mai.

Cambiando le Regole

di Consenso

Le regole del consenso determinano la validità delle transazioni e dei blocchi. Queste regole sono la base per la collaborazione tra tutti i nodi bitcoin e sono responsabili della convergenza di tutte le prospettive locali in un'unica blockchain coerente attraverso l'intera rete.

Mentre le regole di consenso sono invariabili a breve termine e devono essere coerenti su tutti i nodi, non sono invariabili a lungo termine. Al fine di evolvere e sviluppare il sistema bitcoin, le regole devono cambiare di volta in volta per adattarsi a nuove funzionalità, miglioramenti o

correzioni di bug. A differenza dello sviluppo di software tradizionale, tuttavia, l'aggiornamento a un sistema di consenso è molto più difficile e richiede il coordinamento tra tutti i partecipanti.

Hard Fork

In I Fork della Blockchain abbiamo osservato come la rete bitcoin possa divergere brevemente, con due parti della rete che seguono due rami diversi della blockchain per un breve periodo. Abbiamo visto come questo processo si verifica in modo naturale, come parte del normale funzionamento della rete e come la rete riconquista una blockchain comune dopo che uno o

più blocchi sono stati estratti.

C'è un altro scenario in cui la rete può divergere nelle seguenti due catene: un cambiamento nelle regole di consenso. Questo tipo di fork è chiamato *hard fork*, perché dopo la fork, la rete non ritorna su una singola catena. In questo caso le due catene si evolvono indipendentemente. Le hard fork si verificano quando una parte della rete funziona con una serie di regole di consenso diverse rispetto al resto della rete. Ciò può verificarsi a causa di un bug o a causa di un cambiamento intenzionale nell'implementazione delle regole di consenso.

Le hard fork possono essere utilizzate per cambiare le regole del consenso,

ma richiedono un coordinamento tra tutti i partecipanti al sistema. Qualsiasi nodo che non si aggiorna alle nuove regole di consenso non è in grado di partecipare al meccanismo di consenso stesso e viene forzato su una catena separata al momento dell'hard fork. Pertanto, un cambiamento introdotto da un hard fork può essere considerato non "forward compatible", in quanto i sistemi non aggiornati non possono più elaborare le nuove regole di consenso.

Esaminiamo la meccanica di un'hard fork con un esempio specifico.

Una blockchain con fork mostra una blockchain con due fork. All'altezza del blocco 4, si verifica una fork a un

blocco. Questo è il tipo di fork spontanea che abbiamo visto ne I Fork della Blockchain. Con l'estrazione del blocco 5, la rete riconverge su una catena e la fork viene risolta.

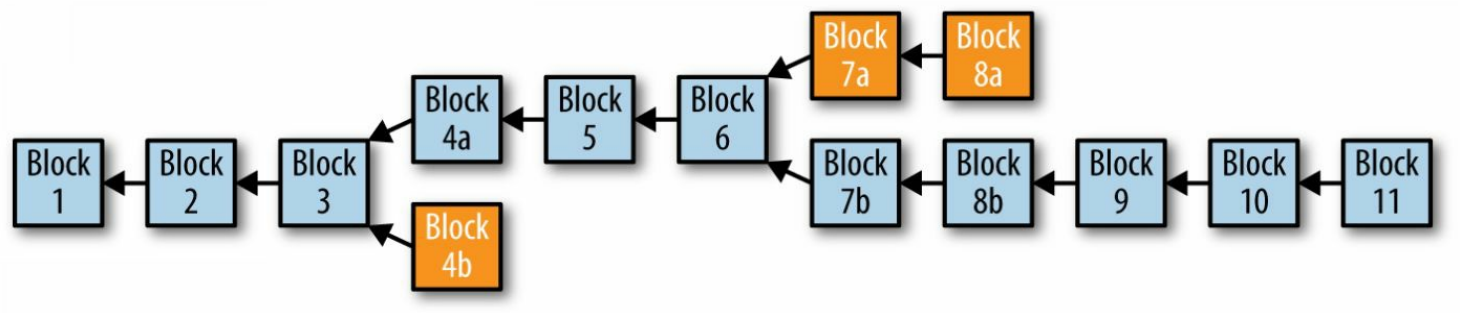


Figura 70. Una blockchain con fork

Successivamente, tuttavia, all'altezza del blocco 6 si verifica un'hard fork. Supponiamo che venga rilasciata una nuova implementazione del client con una modifica delle regole di consenso. A partire dall'altezza del blocco 7, i minatori che eseguono questa nuova

implementazione accetteranno un nuovo tipo di firma digitale, chiamiamola firma "Smores", che non è basata su ECDSA. Subito dopo, un nodo che esegue la nuova implementazione crea una transazione che contiene una firma di Smores e un minatore con la versione del software aggiornato mina il blocco 7b contenente questa transazione.

Nessun nodo o minatore che non ha aggiornato il software per convalidare le firme di Smores ora non è in grado di elaborare il blocco 7b. Dal loro punto di vista, sia la transazione che conteneva una firma di Smores che il blocco 7b che conteneva tale transazione non sono validi, perché li

stanno valutando in base alle vecchie regole di consenso. Questi nodi rifiuteranno la transazione, il blocco e non li propagheranno al resto della rete. Qualsiasi minatore che utilizza le vecchie regole non accetterà il blocco 7b e continuerà a estrarre un blocco candidato il cui genitore è il blocco 6. Infatti, i minatori che usano le vecchie regole potrebbero non ricevere nemmeno il blocco 7b se tutti i nodi a cui sono connessi stanno ancora seguendo le vecchie regole e quindi non propagheranno il blocco. Alla fine, saranno in grado di minare il blocco 7a, che è valido secondo le vecchie regole e non contiene alcuna transazione con le firme Smores.

Le due catene continuano a divergere da questo punto. I minatori sulla catena "b" continueranno ad accettare e a estrarre le transazioni contenenti le firme Smores, mentre i minatori sulla catena "a" continueranno a ignorare queste transazioni. Anche se il blocco 8b non contiene transazioni firmate Smores, i minatori della catena "a" non potranno elaborarlo. Per loro risulterà essere un blocco orfano, in quanto il suo genitore "7b" non viene riconosciuto come blocco valido.

Hard Fork: Software, Network, Mining e Chain

Per gli sviluppatori di software, il termine "fork" ha un altro significato,

andando ad aggiungere ulteriore confusione al termine "hard fork". Nel software open source, un fork si verifica quando un gruppo di sviluppatori sceglie di seguire una diversa roadmap del software avviando un'implementazione in competizione col progetto open source originale. Abbiamo già discusso due circostanze che porteranno a un bivio: un bug nelle regole di consenso e una modifica deliberata delle regole di consenso. Nel caso di un cambiamento intenzionale delle regole di consenso, il software fork precede l'hard fork. Tuttavia, per questo tipo di hard fork, è necessario sviluppare, adottare e lanciare una nuova implementazione

software delle regole di consenso.

Esempi di software fork che hanno tentato di modificare le regole di consenso includono Bitcoin XT, Bitcoin Classic e, più recentemente, Bitcoin Unlimited. Tuttavia, nessuna di queste software fork ha provocato un hard fork. Mentre un fork del software è una condizione preliminare necessaria, non è di per sé sufficiente per un hard fork. Affinché un hard fork si verifichi, è necessario che venga adottata l'implementazione rivale e che vengano attivate le nuove regole da minatori, portafogli e nodi intermediari. Viceversa, ci sono numerose implementazioni alternative di Bitcoin Core e persino le software

fork, che non cambiano le regole di consenso e bloccano un bug, possono coesistere sulla rete e interoperare senza causare un hard fork.

Le regole di consenso possono differire in modi ovvi ed espliciti, nella convalida di transazioni o blocchi. Le regole possono anche differire in modi più sottili, nell'implementazione delle regole di consenso e come si applicano agli script bitcoin o alle funzionalità crittografiche come le firme digitali. Infine, le regole di consenso possono differire in modi imprevisti a causa di vincoli impliciti di consenso imposti da limitazioni di sistema o dettagli di implementazione. Un esempio di

quest'ultimo è stato visto nell'imprevisto hard fork durante l'aggiornamento di Bitcoin Core da 0.7 a 0.8, che è stato causato da una limitazione dell'implementazione del DB Berkley utilizzato per memorizzare i blocchi.

Concettualmente, possiamo definire lo sviluppo di un hard fork in quattro fasi: un fork software, una fork di rete, una fork di mining e una fork di catena.

Il processo inizia quando un'implementazione alternativa del client, con regole di consenso modificate, viene creata dagli sviluppatori.

Quando questa implementazione

forkata viene implementata nella rete, una determinata percentuale di miner, utenti di wallet e nodi intermedi potrebbe adottare ed eseguire questa implementazione. Un fork risultante dipenderà dal fatto che le nuove regole di consenso si applichino a blocchi, transazioni o altri aspetti del sistema. Se le nuove regole di consenso riguardano le transazioni, un portafoglio che crea una transazione con le nuove regole potrebbe determinare un fork di rete, seguito da un hard fork quando la transazione viene estratta in un blocco. Se le nuove regole riguardano i blocchi, il processo di hard fork inizierà quando un blocco viene estratto secondo le

nuove regole.

In primo luogo, la rete verrà forkata. I nodi basati sull'implementazione originale delle regole di consenso rifiuteranno tutte le transazioni e i blocchi creati con le nuove regole. Inoltre, i nodi che seguono le regole di consenso originali verranno temporaneamente esclusi e disconnessi da tutti i nodi che inviano loro transazioni e blocchi non validi. Di conseguenza, la rete si dividerà in due: i vecchi nodi rimarranno collegati ai vecchi nodi e i nuovi nodi saranno collegati solo ai nuovi nodi. Una singola transazione o blocco basato sulle nuove regole si propagherà attraverso la rete e determinerà la

partizione in due reti.

Una volta che un miner che usa le nuove regole mina un blocco, anche la potenza di mining e la catena forkeranno. Nuovi miner mineranno in cima al nuovo blocco, mentre i vecchi miner mineranno una catena separata basata sulle vecchie regole. La rete partizionata farà in modo che i miner che operano su regole di consenso separate non riceveranno i blocchi l'uno dell'altro, poiché sono collegati a due reti separate.

Divergere Miner e Difficoltà

Mentre i miner divergono nell'estrazione di due diverse catene, il potere di hashing viene anch'esso

diviso tra le catene. La potenza di mining può essere suddivisa in qualsiasi proporzione tra le due catene. Le nuove regole possono essere seguite solo da una minoranza o dalla stragrande maggioranza della potenza di mining.

Supponiamo, ad esempio, una divisione dell'80% -20%, con la maggior parte della potenza di mining che utilizza le nuove regole di consenso. Supponiamo inoltre che la fork si verifichi immediatamente dopo un periodo di retargeting.

Le due catene eroderebbero ciascuna la difficoltà dal periodo di retargeting. Le nuove regole di consenso avrebbero l'80% della potenza di

mining precedentemente disponibile. Dal punto di vista di questa catena, la potenza di mining è improvvisamente diminuita del 20% rispetto al periodo precedente. I blocchi si troverebbero in media ogni 12,5 minuti, rappresentando il 20% di diminuzione della potenza di mining disponibile per estendere questa catena. Questo tasso di blocchi continuerà (salvo eventuali modifiche alla potenza di hashing) fino all'estrazione dei successivi 2016 blocchi, che richiederà circa 25.200 minuti (a 12,5 minuti per blocco) o 17,5 giorni. Dopo 17,5 giorni, si verificherà un nuovo retarget e la difficoltà si adatterà (ridotta del 20%) per produrre

nuovamente blocchi di 10 minuti, in base alla ridotta quantità di potenza di hashing in questa catena.

La catena minore, invece, in cui viene effettuato mining sotto le vecchie regole con solo il 20% della potenza di hashing, affronterà un cambiamento molto più dirompente. Su questa catena i blocchi verranno ora estratti ogni 50 minuti in media. La difficoltà non verrà modificata per i successivi 2016 blocchi, che impiegheranno 100.800 minuti, o circa 10 settimane per essere minati. Supponendo una capacità fissa per blocco, ciò comporterà anche una riduzione della capacità di transazione di un fattore 5, in quanto vi sono meno blocchi/ora

disponibili in cui inserire e registrare le transazioni.

Hard Fork Contenziosi

Questa è l'alba dello sviluppo del software del consenso. Proprio come lo sviluppo open source ha cambiato sia i metodi che i prodotti del software e ha creato nuove metodologie, nuovi strumenti e nuove comunità, lo sviluppo del software del consenso rappresenta anch'esso una nuova frontiera dell'informatica. Fuori dai dibattiti, dagli esperimenti e dalle tribolazioni della roadmap dello sviluppo di bitcoin, vedremo emergere nuovi strumenti di sviluppo, pratiche, metodologie e comunità.

Le hard fork sono considerate rischiose perché costringono una minoranza ad effettuare l'aggiornamento o a rimanere su una catena minoritaria. Il rischio di dividere l'intero sistema in due sistemi concorrenti è visto da molti come un rischio inaccettabile. Di conseguenza, molti sviluppatori sono riluttanti a utilizzare il meccanismo di hard fork per implementare gli aggiornamenti alle regole di consenso, a meno che non vi sia un supporto quasi unanime dell'intera rete. Tutte le proposte di hard fork che non hanno un sostegno quasi unanime sono considerate troppo "controverse" per tentare di effettuare cambiamenti senza rischiare una

partizione del sistema.

Il problema delle hard fork è molto controverso nella comunità di sviluppo di bitcoin, in particolare per quanto riguarda le modifiche proposte alle regole di consenso che controllano il limite massimo di dimensioni del blocco. Alcuni sviluppatori si oppongono a qualsiasi forma di hard fork, considerandolo troppo rischioso. Altri vedono il meccanismo di hard fork come uno strumento essenziale per l'aggiornamento delle regole di consenso in modo da evitare il "debito tecnico" e creare una netta rottura con il passato. Infine, alcuni sviluppatori vedono le hard fork come un meccanismo che dovrebbe essere

usato raramente, con molta pianificazione anticipata e solo con un consenso quasi unanime.

Abbiamo già visto l'emergere di nuove metodologie per affrontare i rischi delle hard fork. Nella prossima sezione vedremo le soft fork ed i metodi BIP-34 e BIP-9 per la segnalazione e l'attivazione delle modifiche del consenso.

Soft Fork

Non tutte le modifiche alle regola di consenso determinano una hard fork. Solo le modifiche di consenso che sono in "forward-incompatible" (in avanti) causano una fork. Se la modifica è implementata in modo tale

che un client non modificato vede ancora la transazione o il blocco come valido secondo le regole precedenti, la modifica può avvenire senza fork.

Il termine "*soft fork*" è stato introdotto per distinguere questo metodo di aggiornamento da una "hard fork". In pratica, una soft fork non è affatto una fork. Una soft fork è una modifica forward-compatible con le regole di consenso che consente ai client non aggiornati di continuare a operare in accordo con le nuove regole.

Un aspetto delle soft fork che non è immediatamente ovvio è che gli aggiornamenti delle soft fork possono essere utilizzati solo per vincolare le regole di consenso, non per estenderle.

Per essere compatibili con il futuro, le transazioni e i blocchi creati con le nuove regole devono essere validi anche in base alle vecchie regole, ma non viceversa. Le nuove regole possono limitare solo ciò che è valido; in caso contrario, attivano una hard fork quando vengono respinti in base alle vecchie regole.

Le soft fork possono essere implementate in diversi modi. Il termine non specifica un particolare metodo, piuttosto un insieme di metodi che hanno tutti una cosa in comune: non richiedono che tutti i nodi eseguino l'aggiornamento e non forzano i nodi non aggiornati fuori dal consenso.

Soft fork che ridefiniscono gli opcode NOP

In bitcoin è stato implementato un numero di fork flessibili, basato sulla reinterpretazione degli opcode NOP. Bitcoin Script aveva dieci opcode riservati per uso futuro, da NOP1 a NOP10. In base alle regole di consenso, la presenza di questi opcode in uno script viene interpretata come un operatore impotente, nel senso che non ha alcun effetto. L'esecuzione continua dopo l'opcode NOP come se non fosse lì.

Una soft fork può quindi modificare la semantica di un codice NOP per dargli un nuovo significato. Ad esempio,

BIP-65 (CHECKLOCKTIMEVERIFY) ha reinterpretato l'opcode NOP2. I client che implementano BIP-65 interpretano NOP2 come OP_CHECKLOCKTIMEVERIFY e impongono una regola di consenso assoluta di blocco su UTXO che contiene questo codice operativo negli script di blocco. Questa modifica è una soft fork poiché una transazione valida in BIP-65 è valida anche su qualsiasi client che non implementa (o che non è a conoscenza di) BIP-65. Per i vecchi client, lo script contiene un codice NOP, che viene ignorato.

Altri modi per effettuare un

aggiornamento tramite soft fork

La reinterpretazione degli opcode NOP è stata pianificata come meccanismo per gli aggiornamenti del consenso. Di recente, tuttavia, è stato introdotto un altro meccanismo di soft fork che non si basa su opcode NOP per un tipo molto specifico di modifica del consenso. Ciò è esaminato in modo più dettagliato in Segregated Witness. Segwit è una modifica architettonica alla struttura di una transazione che sposta lo script di sblocco (testimone) dall'interno della transazione a una struttura di dati esterna (che la segrega). Inizialmente, Segwit era

stato concepito come un aggiornamento hard fork, in quanto modificava una struttura fondamentale (la transazione). Nel novembre 2015, uno sviluppatore che lavorava su Bitcoin Core ha proposto un meccanismo attraverso il quale la segwit poteva essere introdotto come una soft fork. Il meccanismo utilizzato per rendere ciò possibile è tramite una modifica dello script di blocco di UTXO creato con regole segwit, in modo tale che i client non modificati vedessero lo script di blocco come riscattabile con qualsiasi script di sblocco. Di conseguenza, è possibile introdurre segwit senza che ogni nodo debba aggiornare o dividere la catena: una soft fork.

È probabile che vi siano altri meccanismi, ancora da scoprire, in base ai quali gli aggiornamenti possono essere fatti in modo compatibile con il futuro come con una soft fork.

Critiche alle Soft Fork

Le soft fork basate sugli opcode NOP sono relativamente incontrovertibili. Gli opcode NOP sono stati inseriti in Bitcoin Script con l'obiettivo esplicito di consentire aggiornamenti senza interruzioni.

Tuttavia, molti sviluppatori sono preoccupati del fatto che altri metodi di aggiornamento di soft fork comportino compromessi inaccettabili.

Le critiche comuni alle modifiche tramite soft fork morbida includono:

Debito tecnico

Poiché le soft fork sono tecnicamente più complesse rispetto a un aggiornamento hard fork, introducono un *technical debt* (debito tecnico). Tale termine si riferisce all'aumento del costo futuro della manutenzione del codice a causa dei compromessi di progettazione effettuati in passato. La complessità del codice a sua volta aumenta la probabilità di bug e vulnerabilità della sicurezza.

Rilassamento di convalida

I client non modificati vedono le transazioni come valide, senza

valutare le regole di consenso modificate. In effetti, i client non modificati non convalidano utilizzando l'intera gamma delle regole di consenso, poiché sono ignari delle nuove regole. Questo si applica agli aggiornamenti basati su NOP e ad altri aggiornamenti di soft fork.

Aggiornamenti irreversibili

Poiché le soft fork creano transazioni con vincoli di consenso aggiuntivi, nella pratica diventano

aggiornamenti irreversibili. Se un upgrade soft fork dovesse essere invertito dopo averlo correttamente avviato, qualsiasi transazione creata secondo le nuove regole potrebbe comportare una perdita di fondi in base alle vecchie regole. Ad esempio, se una transazione CLTV viene valutata secondo le vecchie regole, non vi è alcun

vincolo di timelock e può essere spesa in qualsiasi momento. Pertanto, i critici sostengono che una soft fork fallita che dovrebbe essere annullata ed invertita a causa di un bug, porterebbe quasi certamente alla perdita di fondi.

Segnalazione Soft Fork con versione a blocchi

Poiché le soft fork consentono ai client non aggiornati di continuare a operare

all'interno del consenso, il meccanismo di "attivazione" di una soft fork è rappresentato dai miner che segnalano prontezza: la maggioranza dei minatori deve essere d'accordo sul fatto che siano pronti e disposti a far rispettare le nuove regole di consenso. Per coordinare le loro azioni, c'è un meccanismo di segnalazione che consente loro di mostrare il loro supporto per un cambiamento della regola del consenso. Questo meccanismo è stato introdotto con l'attivazione di BIP-34 a marzo 2013 e sostituito dall'attivazione di BIP-9 a luglio 2016.

Segnalazione e Attivazione

BIP-34

La prima implementazione, in BIP-34, utilizzava il campo della versione a blocchi per consentire ai minatori di segnalare la disponibilità per una specifica modifica della regola del consenso. Prima di BIP-34, la versione del blocco era impostata su "1" per *convenzione* non forzata per *consenso*.

BIP-34 ha definito una modifica della regola del consenso che richiedeva il campo coinbase (input) della transazione coinbase per contenere l'altezza del blocco. Prima di BIP-34, il coinbase di monete poteva contenere qualsiasi dato arbitrario che i minatori sceglievano di includere. Dopo

l'attivazione di BIP-34, i blocchi validi dovevano contenere una specifica altezza di blocco all'inizio del coinbase e essere identificati con un numero di versione maggiore o uguale a "2."

Per segnalare la modifica e l'attivazione di BIP-34, i miner impostano la versione del blocco su "2", anziché su "1." Questo non ha reso immediatamente i blocchi della versione "1" non validi. Una volta attivati, i blocchi di versione "1" non sarebbero più stati validi e tutti i blocchi di versione "2" sarebbero stati necessari per contenere l'altezza del blocco nel coinbase per essere validi.

BIP-34 ha definito un meccanismo di

attivazione in due fasi, basato su una finestra a rotazione di 1000 blocchi. Un minatore segnalerebbe la sua prontezza individuale per BIP-34 costruendo blocchi con "2" come numero di versione. A rigor di termini, questi blocchi non dovevano ancora conformarsi alla nuova regola del consenso di includere l'altezza del blocco nella transazione di coinbase perché la regola del consenso non era ancora stata attivata. Le regole di consenso sono attivate in due fasi:

- Se il 75% (750 dei più recenti 1000 blocchi) sono contrassegnati con la versione "2", i blocchi di versione "2" devono contenere l'altezza del

blocco nella transazione coinbase oppure vengono rifiutati come non validi. I blocchi della versione "1" sono ancora accettati dalla rete e non devono contenere l'altezza del blocco. Le vecchie e nuove regole di consenso coesistono durante questo periodo.

- Quando il 95% (950 dei più recenti 1000 blocchi) sono versione "2", i blocchi versione "1" non sono più considerati validi. I blocchi di versione "2" sono validi solo se

contengono l'altezza del blocco nel coinbase (come per la soglia precedente). Successivamente, tutti i blocchi devono essere conformi alle nuove regole di consenso e tutti i blocchi validi devono contenere l'altezza del blocco nella transazione coinbase.

Dopo la segnalazione ed attivazione con successo sotto le regole BIP-34, tale meccanismo è stato utilizzato due volte per attivare le soft fork:

- BIP-66 Strict DER Encoding of Signatures è stata attivato dalla

segnalazione in stile BIP-34 con una versione a blocchi "3" e blocchi di versione "2" invalidanti.

- BIP-65 CHECKLOCKTIMEVERIFY è stato attivato dalla segnalazione in stile BIP-34 con una versione di blocco "4" e blocchi di versione "3" invalidanti.

Dopo l'attivazione di BIP-65, il meccanismo di segnalazione e attivazione di BIP-34 è stato ritirato e sostituito con il meccanismo di segnalazione BIP-9 descritto in seguito.

Lo standard è definito in BIP-34

(Block v2, Height in Coinbase).

BIP-9 Segnalazione e Attivazione

Il meccanismo utilizzato da BIP-34, BIP-66 e BIP-65 ha avuto successo nell'attivazione di tre soft fork. Tuttavia, è stato sostituito perché aveva diverse limitazioni:

- Utilizzando il valore intero della versione a blocchi, è possibile attivare solo una soft fork per volta, quindi è necessario un coordinamento tra le proposte di soft e un accordo sulle priorità e sulla

sequenza.

- Inoltre, poiché la versione dei blocchi è stata incrementata, il meccanismo non ha offerto un modo semplice per rifiutare una modifica e quindi proporne una diversa. Se i vecchi client sono ancora in esecuzione, potrebbero scambiare segnali per una nuova modifica come segnalazione per la modifica precedentemente rifiutata.
- Ogni nuova modifica ha irrevocabilmente ridotto le versioni

di blocco disponibili per future modifiche.

È stato proposto BIP-9 proprio per superare queste sfide e migliorare il tasso e la facilità di implementazione di possibili cambiamenti futuri.

BIP-9 interpreta la versione del blocco come un campo di bit invece di un intero. Poiché la versione del blocco era originariamente utilizzata come un numero intero, versioni da 1 a 4, restano disponibili solo 29 bit da utilizzare come campo di bit. Ciò rende disponibili 29 bit che possono essere utilizzati per segnalare in modo indipendente e simultaneamente 29 proposte diverse.

BIP-9 imposta anche un tempo massimo per la segnalazione e l'attivazione. In questo modo i minatori non hanno bisogno di effettuare una segnalazione perenne. Se una proposta non viene attivata entro il periodo TIMEOUT (definito nella proposta), la proposta viene considerata respinta. La proposta può essere rinviata per la segnalazione con un bit diverso, rinnovando il periodo di attivazione.

Inoltre, dopo il che TIMEOUT viene superato e una funzione è stata attivata o rifiutata, il bit di segnalazione può essere riutilizzato per un'altra funzione senza creare confusione. Pertanto, possono essere segnalati fino a 29 cambiamenti in parallelo e dopo

TIMEOUT i bit possono essere "riciclati" per proporre nuove modifiche.

NOTA	Mentre i bit di segnalazione possono essere riutilizzati o riciclati, purché il periodo di votazione non si sovrapponga gli autori di BIP-9 raccomandano che i bit vengano riutilizzati solo quando necessario; potrebbe verificarsi un comportamento imprevisto a causa di errori nel software

	precedente. In breve non dovremmo aspettarci di vederne il riutilizzo finché tutti e 29 i bit non saranno stati usati almeno una volta.

Le modifiche proposte sono identificate da una struttura dati che contiene i seguenti campi:

name

Una breve descrizione usata per distinguere le varie proposte. Molto spesso il BIP che descrive la proposta, come "bipN", dove N è il numero BIP.

bit

Da 0 a 28, il bit nella versione del blocco che i minatori usano per segnalare l'approvazione di tale proposta.

starttime

Il tempo (basato su Median Time Past o MTP) che della segnalazione d'inizio dopo il quale il valore del bit viene interpretato come segnalazione di prontezza per la proposta.

endtime

Il tempo (basato su MTP) dopo il quale la modifica viene considerata rifiutata se non viene raggiunta la soglia di attivazione.

A differenza di BIP-34, BIP-9 conta la segnalazione di attivazione a intervalli interi in base al periodo di retargeting di difficoltà di 2016 blocchi. Per ogni periodo di retarget, se la somma dei blocchi che segnalano una proposta supera il 95% (1916 di 2016), la proposta verrà attivata nel successivo periodo di retargeting.

BIP-9 offre un diagramma di stato della proposta per illustrare le varie fasi e transizioni di una proposta, come mostrato in Diagramma di transizione di stato di BIP-9.

Le proposte iniziano nello stato DEFINED, una volta che i loro parametri sono noti (defined) nel software bitcoin. Per i blocchi con

MTP dopo l'ora di inizio, lo stato della proposta passa a STARTED. Se la soglia di voto viene superata entro un periodo di retarget e il timeout non è stato superato, lo stato della proposta passa a LOCKED_IN. Dopo un periodo di retargeting, la proposta diventa ACTIVE. Le proposte rimangono nello stato ACTIVE perennemente quando raggiungono tale stato. Se il timeout è trascorso prima del raggiungimento della soglia di voto, lo stato della proposta passa a FAILED, che indica una proposta respinta. Le proposte FAILED rimangono in quello stato perennemente.

Figura 71. Diagramma di transizione di stato di BIP-9

BIP-9 è stato implementato per l'attivazione di CHECKSEQUENCEVERIFY e BIP associati (68, 112, 113). La proposta denominata "csv" è stata attivata correttamente a luglio 2016.

Lo standard è definito in BIP-9 (Version bits with timeout and delay).

Sviluppo del Software di Consenso

Il software di consenso continua ad evolversi e vi sono molte discussioni sui vari meccanismi per modificare le

regole di consenso. Per la sua stessa natura, bitcoin stabilisce un livello molto alto di coordinamento e consenso per i cambiamenti. Come sistema decentralizzato, non ha "autorità" che possa imporre la sua volontà ai partecipanti della rete. La potenza è diffusa tra più collegi elettorali come minatori, sviluppatori principali, sviluppatori di portafogli, exchange, commercianti e utenti finali. Le decisioni non possono essere prese unilateralmente da nessuno di questi collegi elettorali. Ad esempio, mentre i minatori possono teoricamente cambiare le regole semplicemente per maggioranza (51%), sono vincolati dal consenso degli altri collegi elettorali.

Se agiscono unilateralmente, il resto dei partecipanti può semplicemente rifiutarsi di seguirli, mantenendo l'attività economica su una catena minoritaria. Senza attività economica (transazioni, mercanti, portafogli, exchange), i minatori estraggono una moneta senza valore con blocchi vuoti. Questa diffusione del potere significa che tutti i partecipanti devono coordinarsi, o non è possibile apportare modifiche. Lo status quo è lo stato stabile di questo sistema con solo pochi cambiamenti possibili e se vi è un forte consenso da parte di una maggioranza molto ampia. La soglia del 95% per le soft fork riflette questa realtà.

È importante riconoscere che non esiste una soluzione perfetta per lo sviluppo del consenso. Sia le hard fork che le soft fork implicano compromessi. Per alcuni tipi di modifiche, le soft fork possono essere una scelta migliore; per altri, le hard fork potrebbero essere una scelta migliore. Non c'è scelta perfetta; entrambe comportano rischi. L'unica caratteristica costante dello sviluppo del software di consenso è che il cambiamento è difficile e le forze dietro al consenso lo compromettono.

Alcuni vedono ciò come una debolezza dei sistemi di consenso. Col tempo, potresti pensare allo stesso modo in cui lo penso io: come la più grande

forza del sistema stesso.

Sicurezza Bitcoin

La sicurezza di bitcoin è difficile perché bitcoin non è un riferimento astratto al valore, come il saldo in un conto bancario. Il bitcoin è molto simile al denaro digitale o all'oro. Probabilmente hai sentito l'espressione: "Il possesso è nove decimi della legge". Bene, in bitcoin, il possesso è dieci decimi della legge. Il possesso delle chiavi per sbloccare il bitcoin equivale al possesso di denaro o di un pezzo di metallo prezioso. Puoi perderlo, smarrirlo, rubarlo o dare per sbaglio una somma sbagliata a qualcuno. In ognuno di questi casi, gli utenti non hanno alcun

diritto di ricorso, proprio come se avessero perso denaro sulla strada pubblica.

Tuttavia, bitcoin ha una capacità che contanti, oro, e conti bancari non hanno. Un portafoglio bitcoin, contenente le chiavi, può essere sottoposto a backup come un qualsiasi file. Può essere conservato in copie multiple, o addirittura stampato su carta in copia fisica. Non è possibile effettuare un "backup" di contanti, oro, o conti bancari. Bitcoin è così diverso da tutto ciò è esistito fino ad ora che si presenta la necessità di concepire la sua sicurezza in modo completamente nuovo.

Principi sulla Sicurezza

Il principio fondamentale di bitcoin è il decentramento ed ha importanti implicazioni per la sicurezza. Un modello centralizzato, come una banca tradizionale o una rete di pagamento, dipende dal controllo degli accessi e dal controllo per tenere fuori dal sistema i cattivi attori. In confronto, un sistema decentralizzato come bitcoin spinge la responsabilità ed il controllo agli utenti. Poiché la sicurezza della rete è basata su Proof of Work, non sul controllo degli accessi, la rete può essere aperta e nessuna crittografia è richiesta per al traffico bitcoin.

Su una rete di pagamento tradizionale,

come un sistema di carte di credito, il pagamento è a tempo indeterminato perché contiene l'identificativo privato dell'utente (il numero della carta di credito). Dopo la carica iniziale, chiunque abbia accesso all'identificativo può "prelevare" fondi e addebitarli al proprietario più volte. Pertanto, la rete di pagamento deve essere protetta "end-to-end" con la crittografia e deve garantire che nessun intercettatore o intermediario possa compromettere il traffico dei pagamenti in transito o quando è archiviato (a riposo). Se un malintenzionato ottiene l'accesso al sistema, può compromettere le transazioni correnti e i token di

pagamento che possono essere utilizzati per creare nuove transazioni. Peggio ancora, quando i dati dei clienti vengono compromessi, i clienti sono esposti al furto di identità e devono intervenire per prevenire l'uso fraudolento degli account compromessi.

Il sistema bitcoin è sensibilmente diverso. Una transazione bitcoin autorizza esclusivamente un valore specifico ad un destinatario specifico, e non può essere contraffatta o modificata. La transazione infatti non rivela alcuna informazione privata, come l'identità delle parti, e non può essere utilizzata per autorizzare pagamenti successivi. Pertanto, una

rete di pagamento bitcoin non ha bisogno di essere criptata o protetta da intercettazioni. Anzi, è possibile persino trasmettere le transazioni bitcoin su un canale pubblico aperto, come reti Wi-Fi o Bluetooth, senza alcuna rischio in termini di sicurezza.

Il modello di sicurezza decentralizzato di Bitcoin mette molto potere nelle mani degli utenti. Questo potere significa la responsabilità di mantenere la segretezza delle chiavi. Per la maggior parte degli utenti non è facile da fare, specialmente su dispositivi informatici per uso generico come smartphone o laptop connessi a Internet. Sebbene il modello decentralizzato di bitcoin

prevenga il tipo di compromesso di massa riscontrato con le carte di credito, molti utenti non sono in grado di proteggere adeguatamente le proprie chiavi e di essere hackerate, una per una.

Sviluppare Sistemi Bitcoin Sicuri

Il principio più importante per gli sviluppatori di bitcoin è il decentramento. La maggior parte degli sviluppatori avrà familiarità con i modelli di sicurezza centralizzati e potrebbe essere tentata di applicare questi modelli alle proprie applicazioni bitcoin, con risultati disastrosi.

La sicurezza di Bitcoin si basa sul controllo decentralizzato delle chiavi e sulla convalida indipendente delle transazioni da parte dei minatori. Se si desidera sfruttare la sicurezza di bitcoin, è necessario assicurarsi di rimanere all'interno del modello di sicurezza bitcoin. In termini semplici: non prendere il controllo delle chiavi lontano dagli utenti e non togliere le transazioni dalla blockchain.

Ad esempio, molti dei primi scambi in bitcoin concentravano tutti i fondi degli utenti in un unico portafoglio "caldo" (chiamato cold wallet ndt) con chiavi memorizzate su un unico server. Tale struttura rimuove il controllo da parte degli utenti e centralizza il

controllo delle chiavi in un unico sistema. Molti di questi sistemi sono stati attaccati, con conseguenze disastrose per i loro clienti.

Un altro errore comune è quello di portare le transazioni "fuori blockchain", uno sforzo errato per ridurre le spese di transazione o accelerare l'elaborazione delle transazioni. Un sistema "off chain" registra le transazioni su un registro interno centralizzato e solo occasionalmente le sincronizza con il la blockchain di bitcoin. Questa pratica, di nuovo, sostituisce la sicurezza decentralizzata di bitcoin con un approccio proprietario e centralizzato. Quando le transazioni

sono fuori dalla blockchain, i registri centralizzati protetti in modo improprio possono essere falsificati, deviando fondi e esaurendo le riserve, senza essere notati.

A meno che non siate disposti a investire pesantemente nella sicurezza operativa, in più livelli di controllo degli accessi e audit (come fanno le banche tradizionali) dovreste riflettere attentamente prima di prendere fondi al di fuori del contesto di sicurezza decentralizzato di bitcoin. Anche se si dispone dei fondi e della disciplina per attuare un solido modello di sicurezza, un tale progetto si limita a replicare il fragile modello delle reti finanziarie tradizionali, afflitto da furto

di identità, corruzione e appropriazione indebita. Per sfruttare l'esclusivo modello di sicurezza decentralizzata di bitcoin, è necessario evitare la tentazione di architetture centralizzate che potrebbero sembrare familiari ma che in ultima analisi sovvertono la sicurezza di bitcoin.

La Radice della Fiducia

L'architettura di sicurezza tradizionale si basa su un concetto chiamato root of trust, un nucleo di fiducia utilizzato come base per la sicurezza dell'intero sistema o applicazione. L'architettura di sicurezza si sviluppa intorno alla root of trust come una serie di cerchi

concentrici, simili agli strati di una cipolla, estendendo la fiducia dal centro verso l'esterno. Ogni livello si basa sul livello interno più attendibile utilizzando controlli di accesso, firma digitale, crittografia ed altre informazioni primitive di sicurezza. Più i sistemi software diventano complessi, più sono propensi a contenere bug, che li rendono vulnerabili alla compromissione della sicurezza. Come risultato di ciò, più un sistema software diventa complesso, più è difficile proteggerlo. Il concetto di root of trust assicura che la maggior parte della fiducia sia posizionata all'interno della parte meno complessa del sistema, e quindi meno

vulnerabile, mentre l'architettura del software più complessa è stratificata intorno ad esso. Questa architettura di sicurezza è ripetuta a scale diverse, in primo luogo stabilendo una root of trust all'interno dell'hardware di un unico sistema, quindi estendendo quella root of trust, attraverso il sistema operativo, a servizi di sistema di livello superiore, ed infine a molti server stratificati in cerchi concentrici in di fiducia che diminuisce andando verso l'esterno.

L'architettura di sicurezza bitcoin è diversa. In bitcoin, il sistema di consenso crea un registro pubblico di fiducia completamente decentralizzato. Una blockchain la cui validità è

verificata correttamente utilizza il genesis block (blocco genesi) come root of trust, costruendo una catena di fiducia che si estende fino al blocco attuale. I sistemi bitcoin possono, e devono, utilizzare la blockchain come root of trust. Se si progetta un'applicazione bitcoin complessa, che consiste in servizi su diversi sistemi, è necessario esaminare attentamente l'architettura di sicurezza, in modo da poter verificare dove viene posta la fiducia. In definitiva, l'unica cosa a cui si può dare fiducia esplicitamente è una blockchain la cui validità è verificata completamente. Se nella vostra applicazione viene conferita fiducia, in modo esplicito o

implicito, a qualsiasi componente che non sia la blockchain, la si dovrebbe considerare come una potenziale fonte di problemi, dato che introduce vulnerabilità. Un buon metodo per valutare l'architettura di sicurezza della vostra applicazione è quello di considerare ogni singolo componente e valutare un ipotetico scenario in cui tale componente sia completamente compromesso e sotto il controllo di un malintenzionato. Considerare ogni componente della vostra applicazione e, uno alla volta, valutare l'impatto sulla sicurezza complessiva se tale componente fosse compromesso. Se la vostra applicazione non può più essere considerata sicura quando uno o più

componenti sono compromessi, ciò mostra una collocazione errata della fiducia in tali componenti. Un'applicazione bitcoin senza vulnerabilità dovrebbe essere vulnerabile solo come compromesso del meccanismo di consenso bitcoin, il che significa che la sua root of trust si basa sulla parte più forte della architettura di sicurezza bitcoin.

I numerosi esempi di exchange bitcoin hackerati, evidenziano l'importanza di questo punto, perché la loro architettura di sicurezza ed il loro design fallisce anche le verifiche più superficiali. Queste implementazioni centralizzate avevano dato fiducia in modo esplicito a numerosi componenti

di fuori della blockchain bitcoin, come gli hot wallet, registri database centralizzati, chiavi di crittografia vulnerabili, e altri schemi simili.

Le "Best Practices" sulla Sicurezza livello Utente

Gli esseri umani hanno utilizzato controlli di sicurezza fisici per migliaia di anni. In confronto ciò, in quanto a sicurezza digitale abbiamo meno di 50 anni di esperienza. I sistemi operativi general purpose moderni non sono molto sicuri e non sono particolarmente adatti alla memorizzazione di denaro digitale. I

nostri computer sono costantemente esposti a minacce esterne attraverso connessioni internet sempre online. Essi gestiscono migliaia di componenti software prodotti da centinaia di sviluppatori diversi, spesso con accesso senza vincoli ai file dell'utente. Un unico componente software malevolo, tra le molte migliaia installati sul computer, può compromettere la tua tastiera e i tuoi file, rubando ogni bitcoin "contenuto" in applicazioni wallet. Il livello di manutenzione del computer necessario per mantenere un computer privo di virus e trojan è attuabile da una piccola minoranza di utenti, in quanto va ben oltre il livello di competenza

informatica di un utente medio.

Nonostante decenni di ricerca e progresso nel campo della sicurezza delle informazioni, i beni digitali sono purtroppo ancora vulnerabili a nemici determinati. Persino i sistemi più altamente protetti ad accesso limitato, in società di servizi finanziari, agenzie di intelligence, e industria della difesa, sono spesso violati. Bitcoin crea risorse digitali che hanno un valore intrinseco e possono essere rubati e dirottati a nuovi proprietari immediatamente ed in maniera irrevocabil. Questo crea un incentivo enorme per gli hacker. Fino ad ora, gli hacker hanno dovuto convertire informazioni di identità o token di

account—ad esempio carte di credito, e conti bancari—in beni dopo averli violati. Nonostante la difficoltà dei processi di schermaggio e di riciclaggio delle informazioni finanziarie, la quantità di furti che si verificano è sempre crescente. Bitcoin diminuisce questo problema perché non ha bisogno di essere schermato o riciclato; è un valore intrinseco di un bene digitale.

Fortunatamente, bitcoin crea anche gli incentivi per migliorare la sicurezza informatica. Mentre in passato i rischi di compromissione del computer erano vaghi ed indiretti, bitcoin rende questi rischi chiari ed evidenti. Il possesso di bitcoin su computer serve a far

meditare l'utente sulla necessità di migliorare la sicurezza del computer. Come diretta conseguenza della proliferazione e dell'aumento di utilizzatori di bitcoin e di altre valute digitali, si è potuta riscontrare un'escalation sia di tecniche di hacking che di soluzioni di sicurezza. In parole semplici, gli hacker hanno adesso un obiettivo molto allettante mentre gli utenti hanno un chiaro incentivo per difendersi.

Nel corso degli ultimi tre anni, come risultato diretto dell'aumento di utilizzatori di bitcoin, si è verificato un enorme progresso nel campo della sicurezza delle informazioni nelle forme di criptaggio hardware,

salvataggio delle chiavi e wallet fisici (hardware wallet), nella tecnologia multi-signature, e in digital escrow. Nelle sezioni seguenti esamineremo diverse procedure consigliate per la sicurezza utente.

Archiviazione Fisica dei Bitcoin

Dato che la maggior parte degli utenti si trova molto più a suo agio con sicurezza fisica piuttosto che con sicurezza di informazioni, un metodo molto efficace per la protezione dei bitcoin è il convertirli in forma fisica. Le chiavi bitcoin non sono altro che lunghi numeri. Ciò significa che possono essere memorizzate in forma

fisica, come ad esempio stampate su carta o incise su una moneta metallica. La sicurezza delle chiavi diventa a quel punto altrettanto semplice quanto il custodire la copia stampata delle chiavi bitcoin. Un set di chiavi bitcoin che viene stampato su carta è chiamato "paper wallet", ed esistono molti strumenti gratuiti che possono essere utilizzati per crearli. Per quanto mi riguarda, custodisco la stragrande maggioranza dei miei bitcoin (99% o più) su dei paper wallet, cifrati con BIP-38, con più copie custodite in casseforti. Custodire bitcoin offline è definito come *cold storage* ed è una delle tecniche di sicurezza più efficaci. Un sistema cold storage è un

sistema in cui le chiavi vengono generate da un sistema offline (non si è mai connesso a Internet) e memorizzate offline su carta o su supporti digitali, come ad esempio una chiavetta USB.

Wallet Hardware

Con l'avanzare del tempo, la sicurezza bitcoin adotterà in modo sempre crescente la forma di hardware wallet a prova di manomissione. A differenza di uno smartphone o di un computer desktop, un hardware wallet bitcoin ha un solo scopo: custodire i bitcoin in modo sicuro. Senza alcun software general-purpose a rischio di compromissioni e con interfacce limitate, gli hardware wallet sono in

grado di offrire un livello quasi infallibile di sicurezza ad utenti non esperti. Prevedo che questi diventeranno il metodo più diffuso di custodire i bitcoin. Per un esempio di hardware wallet, si veda Trezor.

Bilanciamento del Rischio

Anche se la maggior parte degli utenti sono comprensibilmente preoccupati dai furti bitcoin, esiste un rischio ancor maggiore. I file di dati vengono smarriti continuamente. Se questi contengono bitcoin, la perdita può essere ancor più gravosa. Nel tentativo di proteggere i loro portafogli bitcoin, gli utenti devono stare molto attenti a non oltrepassare il limite, rischiando

di perdere i propri bitcoin. Nel luglio del 2011, un progetto di sensibilizzazione ed educazione al bitcoin molto conosciuto perse quasi 7000 bitcoin. Cercando di impedire furti, i proprietari implementarono una serie complessa di backup crittografati. Alla fine di ciò, essi persero accidentalmente le chiavi di crittografia, rendendo i backup inutili e perdendo un'intera fortuna. Così come nascondere i soldi seppellendoli nel deserto, se si proteggono i propri bitcoin 'troppo' bene si rischia di non essere in grado di ritrovarli.

Diversificazione del Rischio

Porteresti il tuo intero patrimonio netto

in contanti nel tuo portafoglio? Quasi tutti considererebbero una cosa del genere piuttosto spericolata, tuttavia spesso gli utenti bitcoin tengono tutti i loro bitcoin in un unico wallet. Al contrario, gli utenti bitcoin dovrebbero frazionare il rischio suddividendo i propri possedimenti in diversi wallet bitcoin. Gli utenti prudenti manterranno solo una piccola frazione, forse meno del 5%, dei loro bitcoin in un wallet online o mobile wallet, come "spiccioli". Il resto dovrebbe essere diviso su diversi sistemi di deposito, come ad esempio un desktop wallet e offline (cold storage).

Multisig e Amministrazione

Ogni volta che una società o un individuo custodisce una grossa quantità di bitcoin, essi dovrebbero considerare la possibilità di utilizzare un indirizzo bitcoin multi-signature. Gli indirizzi multi-signature proteggono i fondi richiedendo più di una firma per effettuare il pagamento. Le chiavi di firma sono conservate in luoghi diversi e sotto il controllo di persone diverse. In un ambiente aziendale, per esempio, le chiavi sono generate in modo indipendente e custodite da diversi dirigenti aziendali, per essere sicuri che nessuno possa compromettere i fondi aziendali da solo. Gli indirizzi multi-signature sono anche in grado di

offrire ridondanza, con una persona singola che detiene diverse chiavi custodite in luoghi diversi.

Sopravvivenza

Un'importante considerazione riguardante la sicurezza che viene spesso trascurata è la disponibilità, specialmente nel contesto di incapacità o morte del proprietario delle chiavi. Spesso viene detto agli utenti bitcoin di utilizzare password complesse e di mantenere le loro chiavi al sicuro, non condividendole con nessun altro. Purtroppo, questa pratica rende quasi impossibile per la famiglia dell'utente di recuperare i fondi se l'utente non è più in grado di sbloccarli. Nella

maggior parte dei casi, infatti, le famiglie degli utenti bitcoin potrebbero essere completamente all'oscuro dell'esistenza di tali fondi bitcoin.

Se possiedi molti bitcoin, dovresti considerare condividere i dettagli di accesso con un parente fidato o un avvocato. Uno schema di sopravvivenza più complesso potrebbe essere quello di impostare un accesso multi-signature e una pianificazione immobiliare dei beni attraverso un un avvocato specializzato come "digital asset executor."

Conclusioni

Bitcoin è una tecnologia completamente nuova, senza precedenti e relativamente complessa. Man mano che passa il tempo svilupperemo strumenti migliori per la sicurezza e metodologie più semplici da applicare per i non esperti. Per ora, gli utenti di bitcoin possono utilizzare molti dei suggerimenti discussi in questo libro per godere di un'esperienza sicura e senza problemi.

Applicazioni Blockchain

Andiamo a capire il funzionamento di bitcoin guardando la sua *application platform*. Oggigiorno, molte persone utilizzano il termine "blockchain"per riferirsi a qualsiasi piattaforma che condivide i principi di funzionamento del bitcoin. Il termine è spesso usato in modo sbagliato per riferirsi a molte cose che non offrono le stesse funzioni primarie che offre la blockchain di bitcoin.

In questo capitolo andremo a vedere le funzionalità offerte dalla piattaforma di applicazioni della blockchain di

bitcoin. Considereremo la costruzione delle applicazioni *primitives*, la quale crea la costruzione di blocchi di ogni applicazioni blockchain. Vedremo molte applicazioni importanti che usano questi *primitives*, come *colored coins*, *payment (state) channels*, e *routed payment channels (Lightning Network)*.

Introduzione

Il sistema bitcoin è stato progettato come sistema di pagamento e valuta decentralizzato. Tuttavia, la maggior parte delle sue funzionalità deriva da costrutti di livello molto basso che possono essere utilizzati per applicazioni molto più ampie. Bitcoin

non è stato creato con componenti come account, utenti, saldi e pagamenti. Al contrario, utilizza un linguaggio di scripting transazionale con funzioni crittografiche di basso livello, come abbiamo visto in Transazioni. Proprio come i concetti di livello superiore come conti, bilanci e pagamenti possono essere derivati da questi primitivi di base, così possono molte altre applicazioni complesse. Pertanto, la blockchain di bitcoin può diventare una piattaforma applicativa che offre servizi di fiducia alle applicazioni, come i contratti intelligenti, che superano di gran lunga lo scopo originario della moneta digitale e dei pagamenti.

Costruendo Blocchi (Primitivi)

Quando operato in modo corretto e per un lungo periodo, il sistema bitcoin offre certe garanzie, che possono essere usate per costruire blocchi per creare applicazioni. Queste includono:

No alla Doppia Spesa

La garanzia più importante del algoritmo decentralizzato di bitcoin assicura che nessun UTXO possa essere speso due volte.

Immutabilità

Una volta che una transazione viene registrata nella blockchain ed è stata garantita abbastanza potenza

computazionale con i blocchi successivi, i dati della transazione diventano immutabili. L'immutabilità è sottoscritta dall'energia, poiché la riscrittura della blockchain richiede il dispendio di energia per produrre la Prova di Lavoro (Proof of Work). L'energia richiesta e quindi il grado di immutabilità aumenta con la quantità di lavoro impegnata in cima al blocco contenente una transazione.

Neutralità

La rete bitcoin decentralizzata propaga transazioni valide ignorando l'origine o il contenuto di queste transazioni. Questo significa che chiunque può creare transazioni valide con abbastanza fees e sarà

abile di trasmettere queste transazioni ed averle incluse nella blockchain in qualsiasi momento.

Timestamping Sicuro

Le consensus rules respingono ogni blocco la cui timestamp è troppo indietro nel passato o troppo avanti nel futuro. Questo assicura che la timestamp nei blocchi è affidabile. Il timestamp in un blocco implica che un unspent-before garantisca per tutti gli inputs di tutte le transazioni incluse.

Autorizzazione

Le firme digitali, validate in una rete decentralizzata, offrono autorizzazioni garantite. Gli script

che contengono la richiesta della firma digitale non possono essere eseguiti senza l'autorizzazione del proprietario della chiave privata implicata nello script.

Verificabilità

Tutte le transazioni sono pubbliche e possono essere controllate. Tutte le transazioni e i blocchi sono collegati all'indietro in una catena ininterrotta fino al blocco *genesis*.

Contabilità

In ogni transazione (tranne le transazioni coinbase) il valore degli inputs uguale il valore degli outputs più fees. Non è possibile creare o distruggere il valore del bitcoin in

una transazione. Gli outputs non possono eccedere gli inputs.

Non Scadenza

Una transazione valida non scade. Se è valida oggi, sarà valida anche nel prossimo futuro, fintanto che gli inputs non vengano spesi e le regole del consenso non cambino.

Integrità

Una transazione bitcoin firmata con SIGHASH_ALL o parti di una transazione firmate da un altro SIGHASH non può essere modificata senza invalidare la firma, invalidando in questo modo la transazione stessa.

Atomicità di Transazione

Le transazioni di Bitcoin sono atomiche. Sono valide e confermate (minate) o meno. Le transazioni parziali non possono essere minate e non esiste uno stato provvisorio per una transazione. In qualsiasi momento una transazione viene minata o meno.

Unità di Valore Discrete (Indivisibili)

Gli output di transazione sono unità di valore discrete e indivisibili. Possono essere spesi o non spesi, per intero. Non possono essere divisi o parzialmente spesi.

Quorum di Controllo

I vincoli multifirma negli script impongono un quorum di autorizzazione, predefinito nello

schema multisignature. Il requisito M-di-N è applicato dalle regole di consenso.

Timelock/Invecchiamento

Qualsiasi clausola di script contenente un timelock relativo o assoluto può essere eseguita solo dopo che la sua età supera il tempo specificato.

Replicazione

L'archiviazione decentralizzata della blockchain garantisce che quando una transazione viene minata, dopo sufficienti conferme, viene replicata attraverso la rete e diventa duratura e resiliente alla mancanza di elettricità, alla perdita di dati, ecc.

Protezione dalla Contraffazione

Una transazione può solo spendere output esistenti e convalidati. Non è possibile creare o contraffare il valore.

Consistenza

In assenza di partizioni di miner, i blocchi che sono registrati nella blockchain sono soggetti a riorganizzazione o disaccordo con probabilità di diminuzione esponenziale, in base alla profondità alla quale sono registrati. Una volta registrato abbastanza profondamente, il calcolo e l'energia necessarie per cambiare tale dato rendono praticamente impossibile il cambiamento.

Registrazione dello Stato Esterno

Una transazione può impegnare un valore di dati, tramite OP_RETURN, che rappresenta una transizione di stato in una macchina di stato esterna.

Emissione Prevedibile

Verranno emessi meno di 21 milioni di bitcoin, a un ritmo prevedibile.

L'elenco dei blocchi predefiniti non è completo e altri verranno aggiunti con ogni nuova funzionalità introdotta in bitcoin.

Applicazioni dagli Elementi Costitutivi

Gli elementi costitutivi offerti da bitcoin sono elementi di una

piattaforma di fiducia che può essere utilizzata per comporre applicazioni. Ecco alcuni esempi di applicazioni che esistono oggi e gli elementi costitutivi che usano:

Proof-of-Existence (Notaio Digitale)

Immutabilità + Timestamp + Durata. Un'impronta digitale può essere impegnata con una transazione sulla blockchain, a dimostrazione dell'esistenza di un documento (Timestamp) nel momento in cui è stato registrato. L'impronta digitale non può essere modificata ex-post-facto (immutabilità) e la prova verrà archiviata in modo permanente (durabilità).

Kickstarter (Faro)

Consistenza + Atomicità + Integrità. Se si firma un input e l'output (Integrità) di una transazione per una raccolta fondi, altri possono contribuire alla raccolta stessa ma i bitcoin non possono essere spesi (Atomicità) finché l'obiettivo (valore di output) non viene finanziato completamente (Consistenza).

Canali di Pagamento

Quorum di Controllo + Timelock + No Doppia Spesa + No Scadenza + Resistenza alla Censura + Autorizzazione. Un multisig 2-di-2 (Quorum) con un timelock (Timelock) utilizzato come transazione "settlement" di un canale di pagamento può essere trattenuto

(No Scadenza) e speso in qualsiasi momento (Resistenza alla censura) da una delle parti (Autorizzazione). Le due parti possono quindi creare transazioni di impegno che raddoppiano (No Doppia Spesa) la liquidazione su un timelock più breve (Timelock).

Colored Coins (Monete Colorate)

La prima applicazione di blockchain di cui parleremo sono le *colored coins* (monete colorate).

Le monete colorate si riferiscono a una serie di tecnologie simili che utilizzano le transazioni bitcoin per

registrare la creazione, la proprietà e il trasferimento di risorse estrinseche diverse dal bitcoin. Per "estrinseco" intendiamo attività che non sono archiviate direttamente sulla blockchain bitcoin, al contrario del bitcoin stesso, che è un asset intrinseco alla blockchain.

Le monete colorate vengono utilizzate per tenere traccia delle risorse digitali e delle attività fisiche detenute da terze parti e scambiate tramite certificati di proprietà di monete colorate. Le monete colorate delle risorse digitali possono rappresentare beni immateriali come un certificato azionario, una licenza, una proprietà virtuale (elementi di gioco) o la

maggior parte di qualsiasi forma di proprietà intellettuale concessa in licenza (marchi, diritti d'autore, ecc.). Le monete colorate di beni tangibili possono rappresentare certificati di proprietà delle merci (oro, argento, petrolio), titolo di terra, automobili, barche, aerei, ecc.

Il termine deriva dall'idea di "colorare" o contrassegnare una quantità nominale di bitcoin, ad esempio un singolo satoshi, per rappresentare qualcosa di diverso dal valore bitcoin stesso. Per analogia, prendi in considerazione la possibilità di timbrare una banconota da $ 1 con un messaggio che dice "questo è un certificato azionario di ACME" o

"questa nota può essere riscattata per 1 oncia di argento" e poi scambiare la banconota da $ 1 come certificato di proprietà di quest'altro bene. La prima implementazione di monete colorate, denominata *Enhanced Padded-Order-Based Coloring* o *EPOBC*, ha assegnato risorse estrinseche a un output di 1 satoshi. In questo modo, è stata una vera "moneta colorata", poiché ogni asset è stato aggiunto come attributo (colore) di un singolo satoshi.

Implementazioni più recenti di monete colorate utilizzano l'opcode dello script OP_RETURN per archiviare i metadati in una transazione, in combinazione con archivi di dati

esterni che associano i metadati a risorse specifiche.

Le due più importanti implementazioni di monete colorate oggi sono *Open Assets* e *Colored Coins di Colu*. Questi due sistemi utilizzano approcci diversi alle monete colorate e non sono compatibili. Le monete colorate create in un sistema non possono essere viste o utilizzate nell'altro sistema.

Utilizzando Monete Colorate

Le monete colorate vengono create, trasferite e generalmente visualizzate in appositi portafogli in grado di interpretare i metadati del protocollo delle monete colorate allegati alle

transazioni bitcoin. È necessario prestare particolare attenzione per evitare l'uso di una chiave correlata alla moneta colorata in un normale portafoglio bitcoin, in quanto il normale portafoglio potrebbe distruggere i metadati. Allo stesso modo, le monete colorate non devono essere inviate ad indirizzi gestiti da normali portafogli, ma solo a indirizzi gestiti da portafogli che sono a conoscenza delle monete colorate. Entrambi i sistemi Colu e Open Assets utilizzano speciali indirizzi di monete colorate per mitigare questo rischio e per garantire che le monete colorate non vengano inviate a portafogli non a conoscenza di esse.

Le monete colorate non sono visibili anche per la maggior parte degli blockchain explorer di uso generale. Invece, devi usare un explorer di monete colorate per interpretare i metadati di una transazione di colored coin.

Un'applicazione per wallet e blockchain explorer compatibili con Open Assets è disponibile su coinprism.

Un portafoglio compatibile con le monete colorate Colu e un blockchain explorer è disponibile su Blockchain Explorer.

Un plug-in wallet Copay è disponibile su Colored Coins Copay Addon.

Emissione di monete colorate

Ciascuna delle implementazioni delle colored coin ha un modo diverso di creare monete colorate, ma tutte offrono funzionalità simili. Il processo di creazione di un asset di monete colorate è chiamato *issuance*. Una *transazione iniziale*, la transazione di emissione registra l'asset sulla blockchain di bitcoin e crea un ID di *asset* che viene utilizzato come riferimento alla risorsa. Una volta emesse, le risorse possono essere trasferite tra gli indirizzi utilizzando le *transazioni di trasferimento*.

Le risorse emesse come monete colorate possono avere proprietà

multiple. Possono essere *divisibili* o *indivisibili*, nel senso che la quantità di asset in un trasferimento può essere un numero intero (ad es. 5) o una suddivisione decimale (ad esempio, 4.321). Gli asset possono anche avere un'*emissione fissa*, ovvero un determinato importo viene emesso una sola volta o può essere *emesso nuovamente*, il che significa che le nuove quote dell'attività possono essere emesse dall'emittente originario dopo l'emissione iniziale.

Infine, alcune monete colorate consentono di distribuire *dividendi*, consentendo la distribuzione di pagamenti bitcoin ai proprietari di una moneta colorata in proporzione alla

loro proprietà.

Transazioni di Monete Colorate

I metadati che danno significato a una transazione di monete colorate vengono solitamente memorizzati in uno degli output usando l'opcode OP_RETURN. Diversi protocolli di monete colorate utilizzano codifiche diverse per il contenuto dei dati OP_RETURN. L'output contenente OP_RETURN è chiamato *marker output*.

L'ordine degli output e la posizione del marker output possono avere un significato speciale nel protocollo delle monete colorate. In Open Assets,

ad esempio, qualsiasi output prima del marker output rappresenta l'emissione di asset. Qualsiasi output dopo il marker rappresenta il trasferimento di risorse. Il marker output assegna valori e colori specifici agli altri output facendo riferimento al loro ordine nella transazione.

In Colored Coins (Colu), in confronto, il marker output codifica un opcode che determina come vengono interpretati i metadati. Gli opcode da 0x01 a 0x0F indicano una transazione di emissione. Un codice operativo di emissione viene in genere seguito da un ID dell'asset o un altro identificatore che può essere utilizzato per recuperare le informazioni

sull'asset da una fonte esterna (ad es. Bittorrent). Gli opcode da 0x10 a 0x1F rappresentano una transazione di trasferimento. I metadati delle transazioni di trasferimento contengono semplici script che trasferiscono quantità specifiche di asset dagli input agli output, facendo riferimento al loro indice. L'ordinamento di input e output è quindi importante nell'interpretazione dell script.

Se i metadati sono troppo lunghi per rientrare in OP_RETURN, il protocollo delle monete colorate può utilizzare altri "trucchi" per memorizzare i metadati in una transazione. Gli esempi includono

mettere metadati in un redeem script, seguito da opcode OP_DROP per assicurare che lo script ignori i metadati. Un altro meccanismo utilizzato è uno script multisig 1-di-N in cui solo la prima chiave pubblica è una vera chiave pubblica in grado di spendere l'output e le "chiavi" successive vengono sostituite da metadati codificati.

Per interpretare correttamente i metadati in una transazione di monete colorate devi usare un portafoglio compatibile o un block explorer. Altrimenti, la transazione sembrerà una normale transazione bitcoin con un output OP_RETURN.

Ad esempio, ho creato e rilasciato un

asset MasterBTC usando monete colorate. L'asset MasterBTC rappresenta un buono per una copia gratuita di questo libro. Questi buoni possono essere trasferiti, scambiati e riscattati utilizzando un portafoglio compatibile con le monete colorate.

Per questo particolare esempio, ho utilizzato il wallet and explorer su *https://coinprism.info*, che utilizza il protocollo delle monete colorate Open Asset.

La transazione di emissione vista su coinprism.info mostra la transazione di emissione utilizzando il block explorer di Coinprism:

https://www.coinprism.info/tx/10d7c4e

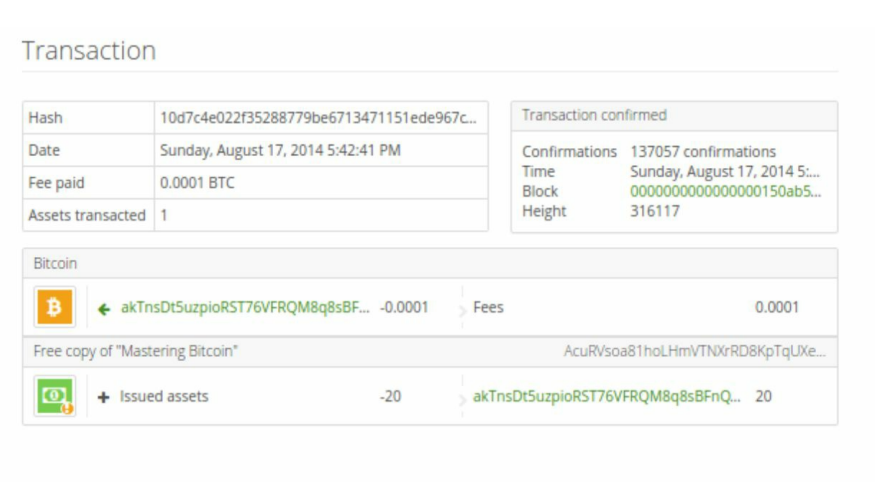


Figura 72. La transazione di emissione vista su coinprism.info

Come puoi vedere, coinprism mostra l'emissione di 20 unità di "Free copy of Mastering Bitcoin", l'asset MasterBTC, in uno speciale indirizzo di monete colorate:

akTnsDt5uzpioRST76VFRQM8q8sBFnQiwcx

	Qualsiasi importo c moneta

ATTENZIONE	colorata inviata a questo indirizzo andrà persa per sempre Non inviare valore a questo indirizzo di esempio!

L'ID transazione della transazione di emissione è un ID transazione bitcoin "normale". La transazione di emissione su un block explorer che non decodifica le monete colorate mostra la stessa transazione in un block

explorer che non decodifica le monete colorate. Useremo *blockchain.info*:

https://blockchain.info/tx/10d7c4e022f3

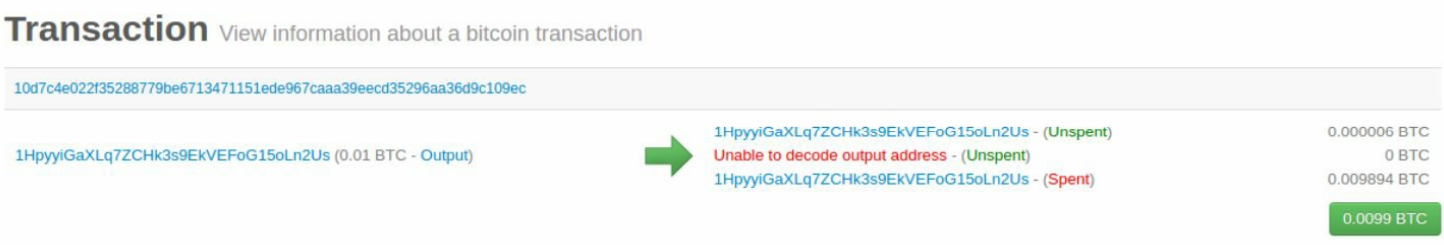


Figura 73. La transazione di emissione su un block explorer che non decodifica le monete colorate

Come puoi vedere, *blockchain.info* non riconosce la tx come una transazione di monete colorate. In effetti, segna il secondo output con "Impossibile decodificare l'indirizzo di output" in lettere rosse.

Se si seleziona "Mostra script e coinbase" in quella schermata, è possibile visualizzare ulteriori dettagli sulla transazione (Gli script nella transazione di emissione).

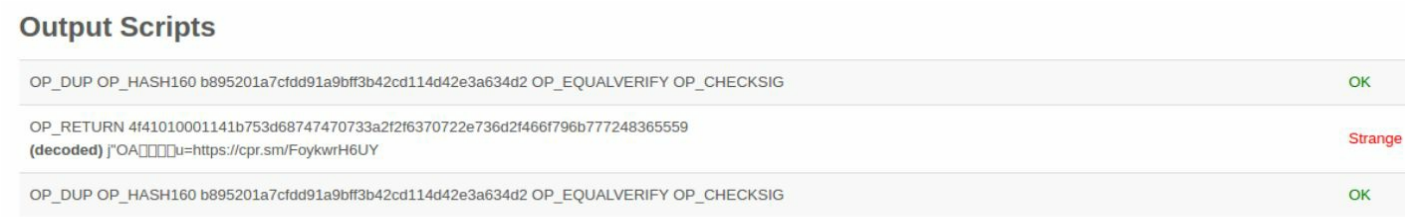


Figura 74. Gli script nella transazione di emissione

Ancora una volta, *blockchain.info* non comprende il secondo output. Lo contrassegna con "Strano" in lettere rosse. Tuttavia, possiamo vedere che alcuni dei metadati nell'output del marcatore sono leggibili dall'uomo:

```
OP_RETURN
4f41010001141b753d68747470733a2f2f63
(decoded)
"OA____u=https://cpr.sm/FoykwrH6UY
```

Recuperiamo la transazione usando bitcoin-cli:

```
$ bitcoin-cli decoderawtransaction
`bitcoin-cli getrawtransaction
10d7c4e022f35288779be6713471151ede9
```

Eliminando il resto della transazione, il secondo output si presenta così:

```
{
"value": 0.00000000,
"n": 1,
"scriptPubKey": "OP_RETURN
4f41010001141b753d68747470733a2f2f63


}
```

Il prefisso 4F41 rappresenta le lettere

"OA", che sta per "Open Assets" e ci aiuta a identificare che quanto segue sono i metadati definiti dal protocollo Open Assets. La stringa codificata ASCII che segue è un collegamento a una definizione di asset:

```
u=https://cpr.sm/FoykwrH6UY
```

Se recuperiamo questo URL, otteniamo una definizione di asset con codifica JSON, come mostrato qui:

```
{
"asset_ids": [
"AcuRVsoa81hoLHmVTNXrRD8KpTqUXeq
],
"contract_url": null,
"name_short": "MasterBTC",
"name": "Copia gratuita di \"Mastering
Bitcoin\"",
"issuer": "Andreas M. Antonopoulos",
```

```
  "description": "This token is redeemable for a free copy of the book \"Mastering Bitcoin\"",
  "description_mime": "text/x-markdown; charset=UTF-8",
"type": "Other",
"divisibility": 0,
"link_to_website": false,
"icon_url": null,
"image_url": null,
"version": "1.0"
}
```


Counterparty (Controparte)

Counterparty è uno strato di protocollo costruito su bitcoin. Il Counterparty Protocol, simile alle monete colorate, offre la possibilità di creare e

scambiare beni virtuali e token. Inoltre, Counterparty offre un exchange decentralizzato di risorse. Counterparty sta inoltre implementando contratti intelligenti, basati su Ethereum Virtual Machine (EVM).

Come i protocolli delle monete colorate, Counterparty incorpora i metadati nelle transazioni bitcoin usando l'opcode OP_RETURN o gli indirizzi multifirma 1-di-N che codificano i metadati al posto delle chiavi pubbliche. Usando questi meccanismi, Counterparty implementa un livello di protocollo codificato in transazioni bitcoin. Il livello del protocollo aggiuntivo può essere

interpretato da applicazioni che sono in grado di riconoscere Counterparty, ad esempio portafogli e blockchain explorer o qualsiasi applicazione creata utilizzando le librerie Counterparty.

Counterparty può essere utilizzato a sua volta come piattaforma per altre applicazioni e servizi. Ad esempio, Tokenly è una piattaforma costruita su Counterparty che consente a creatori di contenuti, artisti e aziende di emettere token che esprimono la proprietà digitale e possono essere utilizzati per noleggiare, accedere, scambiare o acquistare contenuti, prodotti e servizi. Altre applicazioni che sfruttano Counterparty includono giochi (Spells

of Genesis) e progetti di grid computing (Folding Coin).

Maggiori dettagli su Counterparty possono essere trovati su https://counterparty.io. Il progetto open source può essere trovato su https://github.com/CounterpartyXCP.

Canali di Pagamento e Canali di Stato

I *canali di pagamento* sono un meccanismo trustless per lo scambio di transazioni bitcoin tra due parti, al di fuori della blockchain bitcoin. Queste transazioni, che sarebbero valide se basate sulla blockchain bitcoin, sono invece detenute fuori

catena (offchain), e fungono da *cambiali* per l'eventuale liquidazione del lotto. Poiché le transazioni non sono regolate, possono essere scambiate senza la normale latenza di liquidazione, consentendo un throughput di transazione estremamente elevato, una latenza bassa (submillisecond) e una granularità fine (satoshi).

In realtà, il termine *canale* è una metafora. I canali di stato sono costrutti virtuali rappresentati dallo scambio di stato tra due parti, al di fuori della blockchain. Non ci sono "canali" di per sé e il meccanismo di trasporto dati sottostante non è il canale. Usiamo il termine canale per

rappresentare la relazione e lo stato condiviso tra due parti, al di fuori della blockchain.

Per spiegare ulteriormente questo concetto, pensiamo ad un flusso TCP. Dal punto di vista dei protocolli di livello superiore è un "socket" che collega due applicazioni su Internet. Ma se si osserva il traffico di rete, uno stream TCP è solo un canale virtuale su pacchetti IP. Ciascun endpoint delle sequenze di flusso TCP elabora i pacchetti IP per creare l'illusione di un flusso di byte. Sotto, sono tutti i pacchetti disconnessi. Allo stesso modo, un canale di pagamento è solo una serie di transazioni. Se correttamente sequenziati e connessi,

creano obblighi riscattabili di cui ti puoi fidare anche se non ti fidi dell'altro lato del canale.

In questa sezione vedremo varie forme di canali di pagamento. Innanzitutto, esamineremo i meccanismi utilizzati per costruire un canale di pagamento unidirezionale per un servizio di micropagamento con contatore, ad esempio lo streaming di video. Quindi, espanderemo questo meccanismo e introdurremo canali di pagamento bidirezionali. Infine, vedremo come i canali bidirezionali possono essere collegati end-to-end per formare canali multihop in una rete indirizzata, per prima proposta sotto il nome di *Lightning Network*.

I canali di pagamento fanno parte del più ampio concetto di canale di stato, che rappresenta un'alterazione di stato off-chain, assicurata dall'eventuale soluzione in una blockchain. Un canale di pagamento è un canale di stato in cui lo stato che viene modificato è il saldo di una valuta virtuale.

Canali di Stato: Concetti di Base e Terminologia

Un canale di stato viene stabilito tra due parti, attraverso una transazione che blocca uno stato condiviso sulla blockchain. Questa è chiamata *transazione di finanziamento* o *transazione di ancoraggio*. Questa singola transazione deve essere

trasmessa alla rete e estratta per stabilire il canale. Nell'esempio di un canale di pagamento, lo stato bloccato è il saldo iniziale (in valuta) del canale.

Le due parti scambiano quindi le transazioni firmate, chiamate *transazioni di impegno*, che modificano lo stato iniziale. Queste transazioni sono transazioni valide in quanto potrebbero essere presentate per il pagamento da entrambe le parti, ma invece sono tenute fuori catena da ciascuna parte in attesa della chiusura del canale. Gli aggiornamenti di stato possono essere creati con la stessa rapidità con cui ciascuna parte può creare, firmare e trasmettere una

transazione all'altra parte. In pratica ciò significa che possono essere effettuate migliaia di transazioni al secondo possono.

Quando si scambiano le transazioni di impegno, le due parti invalidano anche gli stati precedenti, in modo che la transazione di impegno più aggiornata sia sempre l'unica che può essere riscattata. Ciò impedisce ad entrambe le parti di barare chiudendo unilateralmente il canale con uno stato precedente scaduto che è più favorevole a loro rispetto allo stato corrente. Esamineremo i vari meccanismi che possono essere utilizzati per invalidare lo stato precedente nel resto di questo

capitolo.

Infine, il canale può essere chiuso in modo cooperativo, inviando una *transazione di regolamento finale* alla blockchain, o unilateralmente, da entrambe le parti che inviano l'ultima transazione di impegno alla blockchain. È necessaria un'opzione di chiusura unilaterale nel caso in cui una delle parti si disconnetta inaspettatamente. La transazione di regolamento rappresenta lo stato finale del canale e viene stabilita sulla blockchain.

Durante l'intero ciclo di vita del canale, è necessario presentare solo due transazioni per il mining sulla blockchain: le transazioni di

finanziamento e di regolamento. Tra questi due stati, le due parti possono scambiare qualsiasi numero di transazioni di impegno che non sono mai viste da nessun altro, né sottoposte alla blockchain.

Un canale di pagamento tra Bob e Alice, che mostra le operazioni di finanziamento, di impegno e di regolamento illustra un canale di pagamento tra Bob e Alice, che mostra le operazioni di finanziamento, impegno e regolamento.

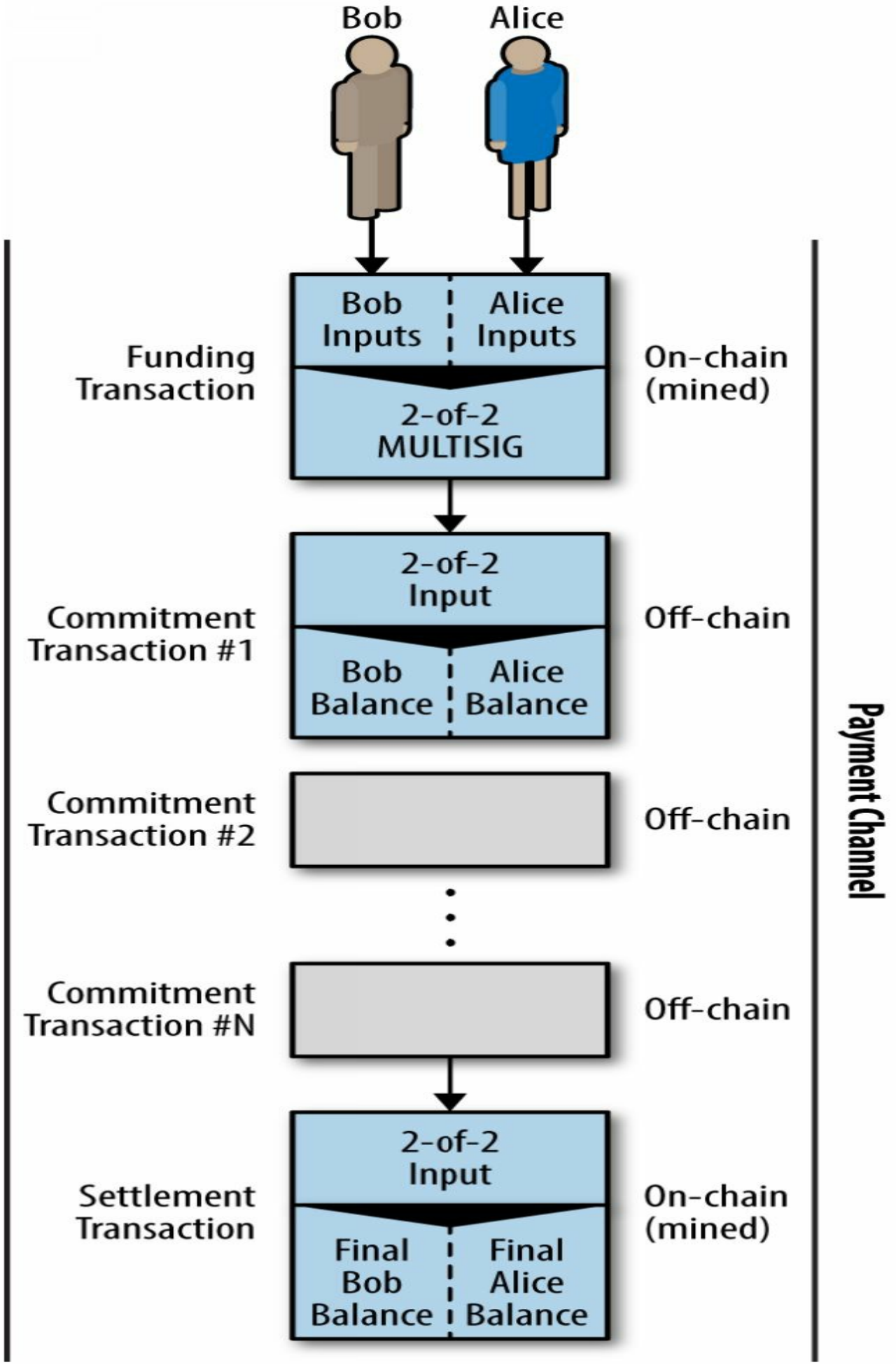

Bob
Alice
Bob Inputs
Alice Inputs
Funding Transaction
On-chain (mined)
2-of-2 MULTISIG
2-of-2 Input
Commitment Transaction #1
Off-chain
Bob Balance
Alice Balance
Commitment Transaction #2
Off-chain
Commitment Transaction #N
Off-chain
2-of-2 Input
Settlement Transaction
On-chain (mined)
Final Bob Balance
Final Alice Balance
Payment Channel

Figura 75. Un canale di pagamento tra Bob e Alice, che mostra le operazioni di finanziamento, di impegno e di regolamento

Esempio di Canale di Pagamento Semplice

Per spiegare i canali di stato, iniziamo con un esempio molto semplice. Dimostriamo un canale a senso unico, il che significa che il valore scorre solo in una direzione. Inizieremo anche con l'assunzione ingenua che nessuno sta cercando di imbrogliare, mantenendo le cose semplici. Una

volta che avremo spiegato l'idea del canale di base, vedremo cosa ci vuole per renderlo trustless, in modo che nessuna delle parti possa imbrogliare, anche tentandoci.

Per questo esempio assumeremo due partecipanti: Emma e Fabian. Fabian offre un servizio di streaming video che viene fatturato al secondo utilizzando un canale di micropagamento. Fabian fa pagare 0.01 millibit (0.00001 BTC) al secondo di video, equivalenti a 36 millibit (0.036 BTC) all'ora di video. Emma è un utente che acquista questo servizio di streaming video da Fabian. Emma acquista in streaming video da Fabian con un canale di pagamento,

pagando per ogni secondo di video mostra Emma acquistare un servizio di video streaming da Fabian tramite un canale di pagamento.

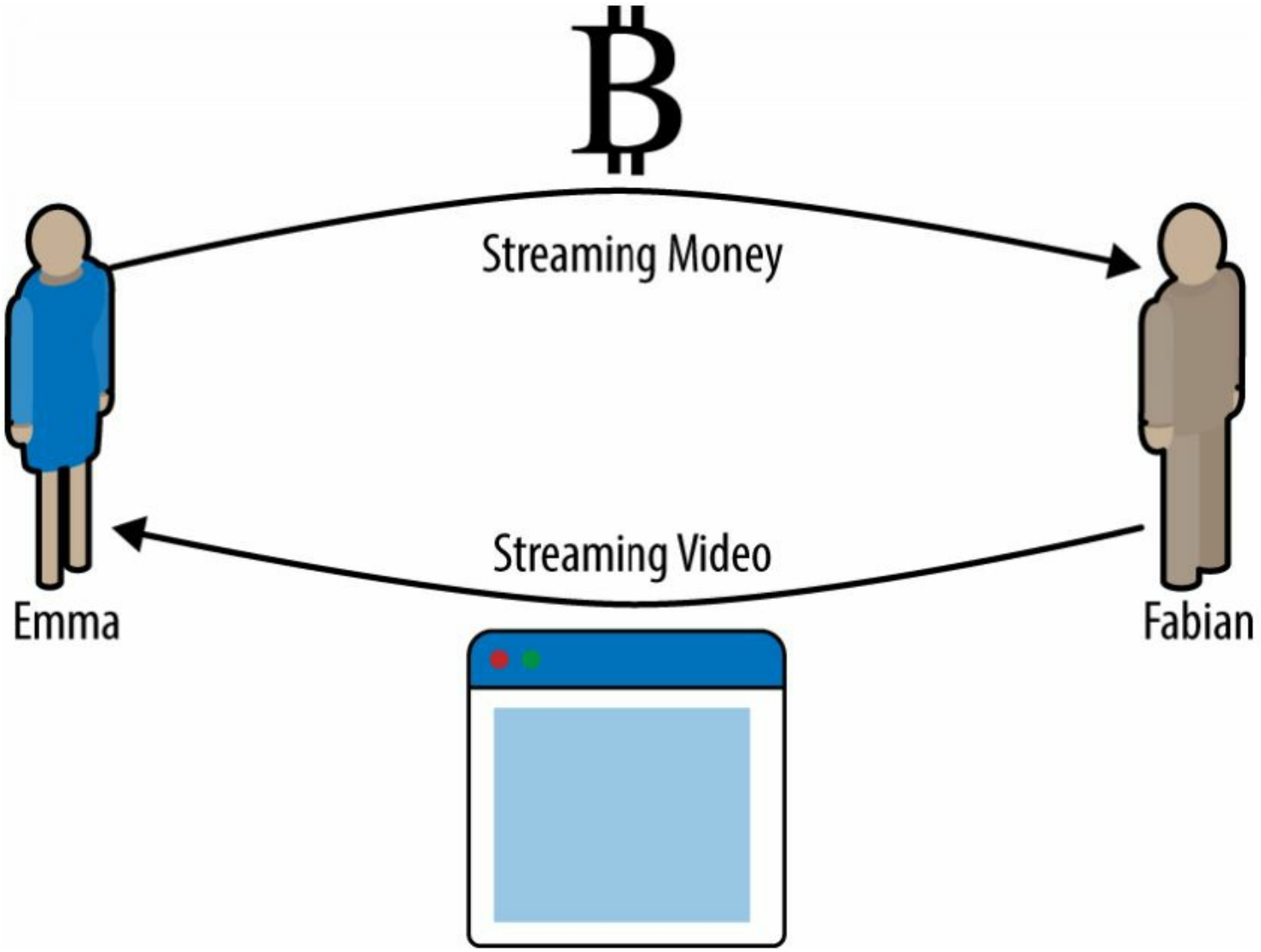


Figura 76. Emma acquista in streaming video da Fabian

con un canale di pagamento, pagando per ogni secondo di video

In questo esempio, Fabian ed Emma utilizzano un software speciale che gestisce sia il canale di pagamento che lo streaming video. Emma sta eseguendo il software nel suo browser, Fabian lo sta eseguendo su un server. Il software include funzionalità di base di un portafoglio bitcoin e può creare e firmare transazioni bitcoin. Sia il concetto che il termine "canale di pagamento" sono completamente nascosti agli utenti. Quello che vedono è un video che viene pagato dal secondo.

Per impostare il canale di pagamento, Emma e Fabian stabiliscono un indirizzo multifirma 2-di-2, in cui ciascuno è in possesso di una delle chiavi. Dal punto di vista di Emma, il software nel suo browser presenta un codice QR con un indirizzo P2SH (che inizia con "3"), che le chiede di inviare un "deposito" per un massimo di 1 ora di video. L'indirizzo è quindi finanziato da Emma. La transazione di Emma, che paga l'indirizzo di multisignature, è la transazione di finanziamento o di ancoraggio per il canale di pagamento.

Per questo esempio, diciamo che Emma finanzia il canale con 36 millibit (0.036 BTC). Ciò consentirà a

Emma di consumare fino a 1 ora di streaming video. La transazione di finanziamento in questo caso imposta la quantità massima che può essere trasmessa in questo canale, impostando la *capacità del canale*.

La transazione di finanziamento consuma uno o più input dal portafoglio di Emma, reperendo i fondi. Crea un output con un valore di 36 millibit pagato per l'indirizzo multisig 2-di-2 controllato congiuntamente da Emma e Fabian. Potrebbe avere output aggiuntivi per il ristorno al portafoglio di Emma.

Una volta confermata la transazione di finanziamento, Emma può avviare lo streaming di video. Il software di

Emma crea e firma una transazione di impegno che modifica il saldo del canale per accreditare 0.01 millibit all'indirizzo di Fabian e rimborsare 35.99 millibit a Emma. La transazione firmata da Emma consuma l'output di 36 millibit creato dalla transazione di finanziamento e crea due output: uno per il rimborso, l'altro per il pagamento di Fabian. La transazione è solo parzialmente firmata, richiede due firme (2-di-2), ma ha solo la firma di Emma. Quando il server di Fabian riceve questa transazione, aggiunge la seconda firma (per l'input 2-di-2) e la restituisce ad Emma insieme ad 1 secondo di video. Ora entrambe le parti hanno una transazione di impegno

pienamente firmata che può riscattare, rappresentando il corretto equilibrio aggiornato del canale. Nessuna delle due parti trasmette questa transazione alla rete.

Successivamente il software di Emma crea e firma un'altra transazione di impegno (impegno #2) che consuma lo stesso output 2-di-2 dalla transazione di finanziamento. La seconda transazione di impegno alloca un output di 0.02 millibit all'indirizzo di Fabian e un output di 35.98 millibit all'indirizzo di Emma. Questa nuova transazione è il pagamento di due secondi cumulativi di video. Il software di Fabian firma e restituisce la seconda transazione di impegno,

insieme a un altro secondo di video.

In questo modo, il software di Emma continua a inviare transazioni di impegno al server di Fabian in cambio di video in streaming. L'equilibrio del canale si accumula gradualmente a favore di Fabian, dato che Emma consuma più secondi di video. Diciamo che Emma guarda 600 secondi (10 minuti) di video, creando e firmando 600 transazioni di impegno. L'ultima transazione di impegno (n. 600) avrà due risultati, dividendo il saldo del canale, uguali a 6 millibit per Fabian e 30 millibit per Emma.

Infine, Emma fa clic su "Stop" per interrompere lo streaming video. Fabian o Emma possono ora

trasmettere la transazione di stato finale per il pagamento. Quest'ultima transazione è la *transazione di regolamento* e paga a Fabian tutti i video consumati da Emma, rimborsando il resto della transazione di finanziamento a Emma.

Il canale di pagamento di Emma con Fabian, che mostra le transazioni di impegno che aggiornano il saldo del canale mostra il canale tra Emma e Fabian e le transazioni di impegno che aggiornano il saldo del canale.

Alla fine, sulla blockchain sono state registrate solo due transazioni: la transazione di finanziamento che ha stabilito il canale e una transazione di regolamento che ha assegnato

correttamente il saldo finale tra i due partecipanti.

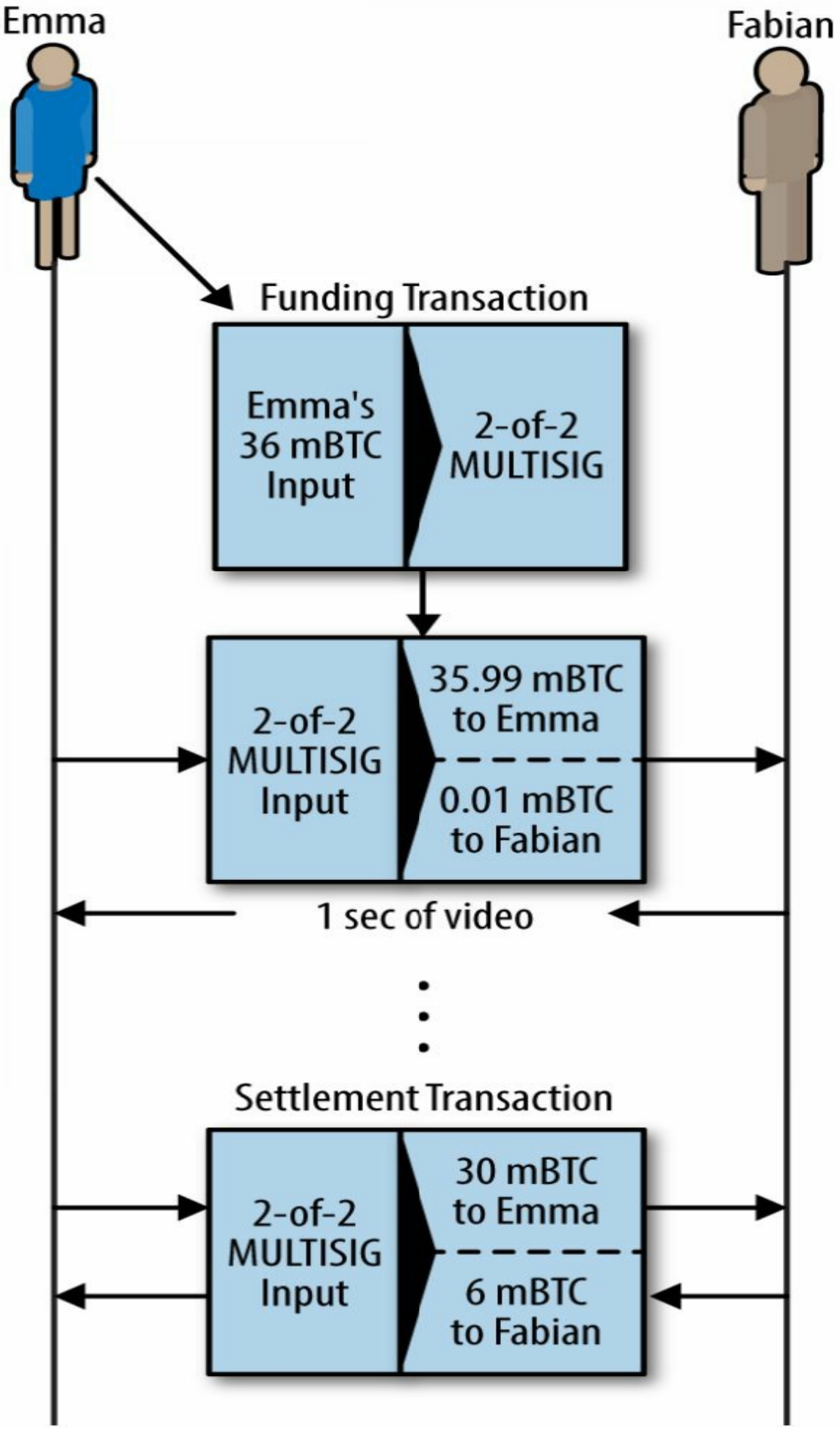

Emma
Fabian
Funding Transaction
Emma's 36 mBTC Input
2-of-2 MULTISIG
2-of-2 MULTISIG Input
35.99 mBTC to Emma
0.01 mBTC to Fabian
1 sec of video
Settlement Transaction
2-of-2 MULTISIG Input
30 mBTC to Emma
6 mBTC to Fabian

Figura 77. Il canale di pagamento di Emma con Fabian, che mostra le transazioni di impegno che aggiornano il saldo del canale

Creare Canali Trustless

Il canale che abbiamo appena descritto funziona, ma solo se entrambe le parti cooperano, senza fallimenti o tentativi di imbroglio. Diamo un'occhiata ad alcuni degli scenari che rompono tale canale e vediamo cosa è necessario fare per evitarli:

- Una volta avvenuta la transazione di

finanziamento, Emma ha bisogno della firma di Fabian per ottenere qualsiasi rimborso. Se Fabian scompare, i fondi di Emma sono bloccati in un multisig 2-di-2 e quindi persi. Questo canale, per come costruito, porta a una perdita di fondi se una delle parti si disconnette prima che vi sia almeno una transazione di impegno firmata da entrambe le parti.

- Mentre il canale è in esecuzione, Emma può prendere qualsiasi delle

transazioni di impegno che Fabian ha contato e trasmesso alla blockchain. Perché pagare per 600 secondi di video, se è in grado di trasmettere la transazione di impegno n. 1 e pagare solo per 1 secondo di video? Il canale fallisce perché Emma può imbrogliare trasmettendo un precedente impegno a suo favore.

Entrambi questi problemi possono essere risolti con i timelock: vediamo come utilizzare i timelock a livello di transazione (nLocktime).

Emma non può rischiare il finanziamento di un 2-di-2 multisig se

non ha un rimborso garantito. Per risolvere questo problema, Emma costruisce la transazione di finanziamento e rimborso allo stesso tempo. Firma la transazione di finanziamento ma non la trasmette a nessuno. Emma trasmette solo la transazione di rimborso a Fabian e ottiene la sua firma.

La transazione di rimborso funge da prima transazione di impegno e il suo timelock stabilisce il limite superiore per la vita del canale. In questo caso, Emma potrebbe impostare il nLocktime su 30 giorni o 4320 blocchi nel futuro. Tutte le transazioni di impegno successive devono avere un timelock più breve, in modo che

possano essere riscattate prima della transazione di rimborso.

Ora che Emma ha una transazione di rimborso completamente firmata, può tranquillamente trasmettere la transazione di finanziamento firmata sapendo che alla fine, dopo la scadenza del timelock, potrà riscattare la transazione di rimborso anche se Fabian scompare.

Ogni transazione di impegno che le parti scambieranno durante la vita del canale verrà timbrata (timelocked) nel futuro. Ma il ritardo sarà leggermente più breve per ogni impegno, quindi l'impegno più recente può essere riscattato prima che l'impegno precedente venga invalidato. A causa

del nLockTime, nessuna delle parti può propagare correttamente nessuna delle transazioni di impegno fino alla scadenza del timeout. Se tutto andrà bene, coopereranno e chiuderanno il canale con una transazione di regolamento, rendendo non necessario trasmettere una transazione di impegno intermedio. In caso contrario, la transazione di impegno più recente può essere propagata per regolare l'account e invalidare tutte le precedenti transazioni di impegno.

Ad esempio, se la transazione di impegno n. 1 ha un timelock a 4320 blocchi in futuro, la transazione di impegno n. 2 ha un timelock a 4319 blocchi in futuro. La transazione di

impegno n. 600 può essere spesa 600 blocchi prima che la transazione di impegno n. 1 diventi valida.

Ogni impegno imposta un intervallo di tempo più breve, permettendogli di essere speso prima che gli impegni precedenti diventino validi mostra ogni impostazione della transazione di impegno in un intervallo di tempo più breve, consentendone la spesa prima che diventino validi gli impegni precedenti.

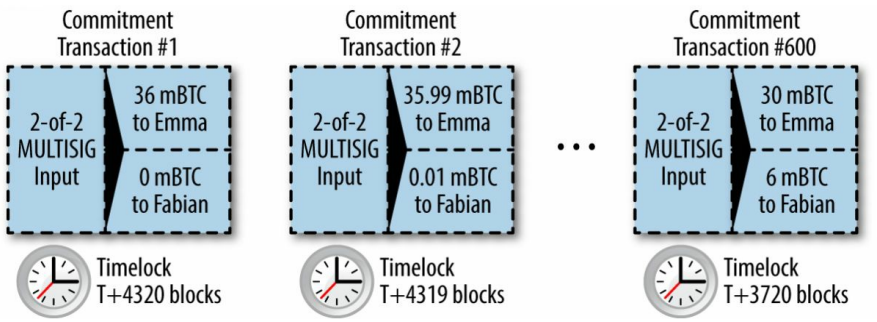


Figura 78. Ogni impegno imposta un intervallo di tempo più breve,

permettendogli di essere speso prima che gli impegni precedenti diventino validiOgni transazione di impegno successiva deve avere un timelock più breve in modo che possa essere trasmessa prima dei suoi predecessori e prima della transazione di rimborso. La possibilità di trasmettere un impegno in precedenza garantisce che sia in grado di spendere l'output di finanziamento e impedire che qualsiasi altra transazione di impegno venga riscattata spendendo l'output. Le garanzie offerte dalla blockchain di bitcoin, impedendo la doppia spesa e rafforzando i timelock, consentono efficacemente a ciascuna transazione di impegno di invalidare i suoi

predecessori.

I canali di stato utilizzano i timelock per imporre contratti intelligenti in una dimensione temporale. In questo esempio abbiamo visto come la dimensione temporale garantisce che la transazione di impegno più recente diventi valida prima di qualsiasi impegno precedente. Pertanto, la transazione di impegno più recente può essere trasmessa, spendendo gli input e invalidando le precedenti transazioni di impegno. L'applicazione di smart contract con scadenze assolute protegge da imbrogli da parte di una delle parti. Questa implementazione non richiede nient'altro che i timelock assoluti a livello di transazione

(nLocktime). Successivamente, vedremo come i timelock a livello di script, CHECKLOCKTIMEVERIFY e CHECKSEQUENCEVERIFY, possono essere utilizzati per costruire canali di stato più flessibili, utili e sofisticati.

La prima forma di canale di pagamento unidirezionale è stata dimostrata attraverso un'applicazione di streaming video nel 2015 da un team di sviluppatori argentino. Puoi ancora vederlo su *streamium.io*.

I timelock non sono l'unico modo per invalidare le precedenti transazioni di impegno. Nelle prossime sezioni vedremo come utilizzare una chiave di revoca per ottenere lo stesso risultato.

I timelock sono efficaci ma presentano due svantaggi distinti. Stabilendo un timelock massimo, quando il canale viene aperto per la prima volta, limita la durata del canale stesso. Peggio ancora, costringono le implementazioni dei canali a trovare un equilibrio tra l'abilitazione dei canali longevi e l'obbligatorietà di uno dei partecipanti ad attendere per un lungo periodo un rimborso in caso di chiusura anticipata. Ad esempio, se si consente al canale di rimanere aperto per 30 giorni, impostando il timelock del rimborso su 30 giorni, se una delle parti scompare immediatamente, l'altra parte deve attendere 30 giorni per un rimborso. Più il punto finale è distante,

più il rimborso è distante.

Il secondo problema è che, dal momento che ogni transazione di impegno successiva deve ridurre il timelock, esiste un limite esplicito al numero di transazioni di impegno che possono essere scambiate tra le parti. Ad esempio, un canale di 30 giorni, impostando un timelock di 4320 blocchi nel futuro, può contenere solo 4320 transazioni di impegno intermedio prima di essere chiuso. C'è un pericolo nell'impostare l'intervallo di transazione dell'impegno di timelock a 1 blocco. Impostando l'intervallo di tempo tra le transazioni di impegno su 1 blocco, uno sviluppatore sta creando un onere

molto elevato per i partecipanti al canale che devono essere vigili, restare online e tenere tutto sott'occhio; pronti a trasmettere la transazione di impegno corretta in qualsiasi momento.

Ora che comprendiamo come utilizzare i timelock per invalidare gli impegni precedenti, possiamo vedere la differenza tra chiudere il canale in modo cooperativo e chiuderlo unilateralmente trasmettendo una transazione di impegno. Tutte le transazioni di impegno sono timelocked, pertanto la trasmissione di una transazione di impegno implica sempre l'attesa fino alla scadenza del timelock. Ma se le due parti

concordano su quale sia il bilancio finale e sanno che entrambe detengono transazioni di impegno che alla fine renderanno tale equilibrio una realtà, possono costruire una transazione di regolamento senza un timelock che rappresenta lo stesso equilibrio. In una chiusura cooperativa, ciascuna delle parti prende la transazione di impegno più recente e crea una transazione di regolamento identica in tutto tranne per il fatto che omette il timelock. Entrambe le parti possono firmare questa transazione di regolamento sapendo che non c'è modo di imbrogliare e ottenere un equilibrio più favorevole. Firmando e trasmettendo in modo cooperativo la

transazione di regolamento, possono chiudere il canale e riscattare il proprio saldo immediatamente. Nel peggiore dei casi, una delle parti può essere meschina, rifiutarsi di cooperare e costringere l'altra parte a chiudere unilateralmente con la transazione di impegno più recente. Ma se venisse fatto ciò, anche l'altra parte dovrebbe aspettare i propri fondi.

Impegni Revocabili Asimmetrici

Un modo migliore per gestire gli stati di impegno precedenti è revocandoli esplicitamente. Tuttavia, questo ciò è facile da raggiungere. Una

caratteristica chiave di bitcoin è che una volta che una transazione è valida, rimane valida e non ha scadenza. L'unico modo per annullare una transazione è spendere due volte i suoi input con un'altra transazione prima che venga minata. Ecco perché abbiamo utilizzato i timelock nell'esempio di canale di pagamento sopra riportato per garantire che gli impegni più recenti sarebbero potuto essere spesi prima che gli impegni più vecchi fossero validi. Tuttavia, gli impegni di sequencing nel tempo creano una serie di vincoli che rendono i canali difficili da usare.

Anche se una transazione non può essere annullata, può essere costruita

in modo tale da renderla indesiderabile da usare. Il modo in cui lo facciamo è dando a ciascuna delle parti una *chiave di revoca* che può essere usata per punire l'altra parte se cerca di imbrogliare. Questo meccanismo di revoca delle precedenti transazioni di impegno è stato inizialmente proposto come parte di Lightning Network.

Per spiegare le chiavi di revoca, costruiremo un canale di pagamento più complesso tra due exchange gestiti da Hitesh e Irene. Hitesh e Irene gestiscono exchange di bitcoin rispettivamente in India e negli Stati Uniti. I clienti dell'exchange indiano di Hitesh spesso inviano i pagamenti

all'exchange degli statunitense di Irene e viceversa. Attualmente, queste transazioni si verificano sulla blockchain di bitcoin, ma ciò significa pagare le commissioni ed attendere diversi blocchi per le conferme. La creazione di un canale di pagamento tra gli exchange ridurrà significativamente i costi e accelererà il flusso delle transazioni.

Hitesh e Irene avviano il canale costruendo collaborativamente una transazione di finanziamento, con cui entrambi finanziano il canale con 5 bitcoin. Il bilancio iniziale è di 5 bitcoin per Hitesh e 5 bitcoin per Irene. La transazione di finanziamento blocca lo stato del canale in un

multisig 2-di-2, proprio come nell'esempio di un canale semplice.

La transazione di finanziamento può avere uno o più input da Hitesh (aggiungendo fino a 5 bitcoin o più) e uno o più input da Irene (aggiungendo fino a 5 bitcoin o più). Gli input devono superare leggermente la capacità del canale per coprire le commissioni di transazione. La transazione ha un output che blocca i 10 bitcoin totali in un indirizzo multisig 2-di-2 controllato da Hitesh e Irene. La transazione di finanziamento può anche avere uno o più output che restituiscono il resto a Hitesh e Irene se i loro input superano il loro contributo previsto al canale. Questa è

una singola transazione con input offerti e firmati da due parti. Deve essere costruita in collaborazione e firmata da ciascuna parte prima che venga trasmessa.

Ora, invece di creare una singola transazione di impegno firmata da entrambe le parti, Hitesh e Irene creano due diverse transazioni di impegno che sono *asimmetriche*.

Hitesh ha una transazione di impegno con due output. Il primo output paga a Irene i 5 bitcoin che le sono *dovuti immediatamente*. Il secondo output paga a Hitesh i 5 bitcoin che gli sono dovuti, ma solo dopo un timelock di 1000 blocchi. Gli output della transazione hanno questo aspetto:

```
Input: 2-of-2 funding output, signed by
Irene

Output 0 <5 bitcoin>:
   <Irene's Public Key> CHECKSIG

Output 1:
   <1000 blocks>
CHECKSEQUENCEVERIFY
DROP
   <Hitesh's Public Key> CHECKSIG
```

Irene ha una transazione di impegno diversa con due output. Il primo output paga a Hitesh i 5 bitcoin che gli sono dovuti immediatamente. Il secondo output paga a Irene i 5 bitcoin che deve, ma solo dopo un timelock di 1000 blocchi. La transazione di impegno che Irene detiene (firmata da

Hitesh) si presenta così:

```
Input: 2-of-2 funding output, signed by Hitesh

Output 0 <5 bitcoin>:
    <Hitesh's Public Key> CHECKSIG

Output 1:
    <1000 blocks>
CHECKSEQUENCEVERIFY
DROP
    <Irene's Public Key> CHECKSIG
```

In questo modo, ciascuna parte ha una transazione di impegno spendendo l'output di fondi 2-di-2. Questo input è firmato dall'altra parte. In qualsiasi momento la parte che detiene la transazione può anche firmare (completando il 2-di-2) e trasmettere.

Tuttavia, se viene trasmessa la transazione di impegno, paga immediatamente l'altra parte mentre attende la scadenza di un breve periodo di tempo. Imponendo un ritardo sulla redenzione di uno degli output, poniamo un lieve svantaggio in ciascuna delle parti quando scelgono di trasmettere unilateralmente una transazione di impegno. Ma un ritardo di tempo da solo non è sufficiente per incoraggiare una condotta equa.

Due transazioni di impegno asimmetriche con pagamento ritardato per la parte che detiene la transazione mostra due transazioni di impegno asimmetriche, in cui l'output che paga il titolare dell'impegno viene ritardato.

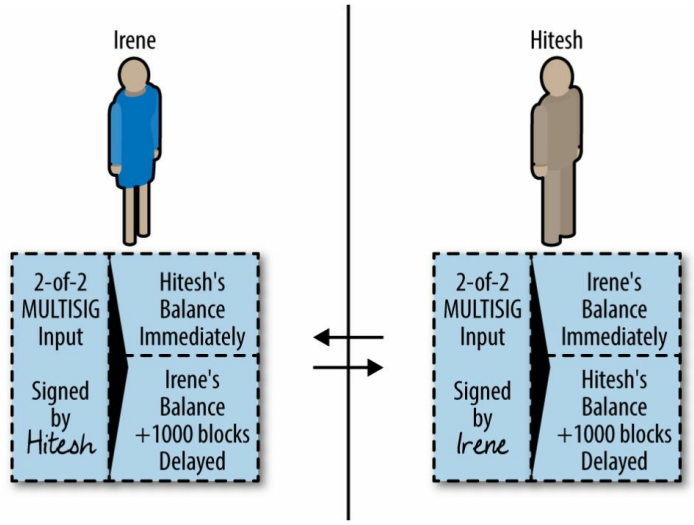


Figura 79. Due transazioni di impegno asimmetriche con pagamento ritardato per la parte che detiene la transazioneOra introduciamo l'elemento finale di questo schema: una chiave di revoca che impedisce a un imbroglione di trasmettere un impegno scaduto. La chiave di revoca consente alla parte ingiusta di punire l'imbroglione riportando l'intero equilibrio del canale.

La chiave di revoca è composta da due segreti, ognuno generato in modo indipendente da ciascun partecipante al canale. È simile a un multisig 2-di-2, ma costruito utilizzando l'aritmetica della curva ellittica, in modo che entrambe le parti conoscano la chiave pubblica di revoca, ma ciascuna parte conosce solo metà della chiave segreta di revoca.

In ogni turno, entrambe le parti rivelano metà del segreto di revoca all'altra parte, dando così all'altra parte (che ora ha entrambe le metà) i mezzi per reclamare l'output di penalità se tale transazione revocata viene trasmessa.

Ciascuna transazione di impegno ha un

output "ritardato". Lo script di redeem per quell'output consente a una parte di riscattarlo dopo 1000 blocchi, o all'altra parte di riscattarlo se ha in possesso della chiave di revoca, penalizzando la trasmissione di un impegno revocato.

Quindi, quando Hitesh crea una transazione di impegno per Irene e la sua firma, rende il secondo output pagabile a se stesso solo dopo 1000 blocchi, o tramite la chiave pubblica di revoca (di cui conosce solo metà del segreto). Hitesh costruisce questa transazione. Invierà la propria metà del segreto di revoca a Irene solo quando sarà pronto a trasferirsi in un nuovo canale di stato e desidera

revocare questo impegno.

Lo script del secondo output si presenta così:

```
Output 0 <5 bitcoin>:
   <Irene's Public Key> CHECKSIG

Output 1 <5 bitcoin>:
IF
   # Revocation penalty output
   <Revocation Public Key>
ELSE
   <1000 blocks>
CHECKSEQUENCEVERIFY
DROP
   <Hitesh's Public Key>
ENDIF
CHECKSIG
```

Irene può tranquillamente firmare questa transazione, poiché se

trasmessa le pagherà immediatamente ciò che le è dovuto. Hitesh detiene la transazione, ma sa che se la trasmette in un canale unilaterale di chiusura, dovrà attendere 1000 blocchi per essere pagato.

Quando il canale passa allo stato successivo, Hitesh deve *revocare* questa transazione di impegno prima che Irene accetti di firmare la prossima transazione di impegno. Tutto ciò che deve fare è inviare la sua metà della *chiave di revoca* a Irene. Una volta che Irene ha entrambe le parti della chiave segreta di revoca per questo impegno, può firmare il prossimo impegno con fiducia. Sa che se Hitesh cerca di imbrogliare

pubblicando l'impegno precedente, può usare la chiave di revoca per riscattare l'output ritardato di Hitesh. *Se Hitesh imbroglia, Irene ottiene ENTRAMBI gli output*. Nel frattempo, Hitesh ha solo metà del segreto di revoca per quella chiave pubblica di revoca e non può riscattare l'output fino a 1000 blocchi. Irene sarà in grado di riscattare l'output e punire Hitesh prima che siano trascorsi i 1000 blocchi.

Il protocollo di revoca è bilaterale, il che significa che in ogni round, mentre lo stato del canale avanza, le due parti si scambiano nuovi impegni, scambiano i segreti di revoca per gli impegni precedenti e firmano

reciprocamente le nuove operazioni di impegno. Quando accettano un nuovo stato, rendono impossibile l'uso dello stato precedente, inviandosi reciprocamente i segreti di revoca necessari per punire qualsiasi imbroglio.

Facciamo un esempio per capire come funziona tutto ciò: uno dei clienti di Irene vuole inviare 2 bitcoin a uno dei clienti di Hitesh. Per trasmettere 2 bitcoin attraverso il canale, Hitesh e Irene devono far avanzare lo stato del canale per riflettere il nuovo equilibrio. Si impegneranno in un nuovo stato (stato numero 2) in cui i 10 bitcoin del canale sono suddivisi rispettivamente in 7 bitcoin per Hitesh

e 3 bitcoin per Irene. Per far avanzare lo stato del canale, ciascuno di essi creerà nuove transazioni di impegno che riflettono il nuovo equilibrio del canale.

Come prima, queste transazioni di impegno sono asimmetriche in modo che la transazione di impegno di ciascuna parte le imponga di aspettare se vogliono riscattarla. Fondamentalmente, prima di firmare nuove operazioni di impegno, devono prima scambiarsi le chiavi di revoca per invalidare l'impegno precedente. In questo caso particolare, gli interessi di Hitesh sono allineati con lo stato reale del canale e quindi non ha motivo di trasmettere uno stato

precedente. Tuttavia, per Irene, lo stato numero 1 la lascia con un bilancio più alto dello stato 2. Quando Irene consegna a Hitesh la chiave di revoca per la sua precedente transazione di impegno (stato numero 1) sta effettivamente revocando la sua capacità di trarre profitto dalla regressione del canale a un precedente stato perché con la chiave di revoca Hitesh può riscattare entrambi gli output della transazione di impegno precedente senza ritardi. Nel senso che se Irene trasmette lo stato precedente, Hitesh può esercitare il suo diritto di prendere tutti gli output.

È importante sottolineare che la revoca non avviene automaticamente.

Mentre Hitesh ha la capacità di punire Irene se prova a imbrogliare, deve tenere sempre gli occhi puntati sulla blockchain per scovare eventuali tentativi di baro. Se vede trasmessa una transazione di impegno precedente, ha 1000 blocchi per agire e utilizzare la chiave di revoca per contrastare l'imbroglio di Irene e punirla prendendo l'intero equilibrio, ovvero tutti i 10 bitcoin.

Gli impegni asimmetrici revocabili con timelock relativi (CSV) sono un modo migliore per implementare i canali di pagamento e un'innovazione molto significativa in questa tecnologia. Con questo costrutto, il canale può rimanere aperto

indefinitamente e può avere miliardi di transazioni di impegno intermedio. Nelle implementazioni prototipo di Lightning Network, lo stato di impegno è identificato da un indice a 48 bit, che consente più di 281 trilioni (2.8 x 1014) di transzioni di stato in ogni singolo canale!

Hash Time Lock Contract (HTLC)

I canali di pagamento possono essere ulteriormente ampliati con un tipo speciale di contratto intelligente che consente ai partecipanti di impegnare fondi per un segreto rimborsabile, con una scadenza. Questa funzione è denominata *Hash Time Lock Contract*

o *HTLC* e viene utilizzata in entrambi i canali di pagamento bidirezionali e instradati.

Per prima cosa spieghiamo la parte "hash" dell'HTLC. Per creare un HTLC, il destinatario previsto del pagamento creerà innanzitutto un R segreto. Quindi calcola l'hash di questo H segreto:

```
H = Hash(R)
```

Ciò produce un hash H che può essere incluso nello script di blocco di un output. Chi conosce il segreto può usarlo per riscattare l'output. Il segreto R viene anche definito come *preimage* della funzione hash. Il preimage è solo il dato che viene utilizzato come input

per una funzione hash.

La seconda parte di un HTLC è il componente "time lock". Se il segreto non viene rivelato, il mittente dell'HTLC può ottenere un "rimborso" dopo un po' di tempo. Ciò si ottiene con un blocco temporale assoluto usando CHECKLOCKTIMEVERIFY.

Lo script che implementa un HTLC potrebbe assomigliare a questo:

```
IF
   # Payment if you have the secret R
   HASH160 <H> EQUALVERIFY
ELSE
   # Refund after timeout.
   <locktime>
CHECKLOCKTIMEVERIFY DROP
   <Payer Public Key> CHECKSIG
ENDIF
```

Chiunque conosca il segreto R, che quando viene sottoposto a hash è uguale a H, può riscattare l'output esercitando la prima clausola del flusso IF.

Se il segreto non viene rivelato, dopo un certo numero di blocchi il mittente può richiedere un rimborso utilizzando la seconda clausola nel flusso IF.

Questa è un'implementazione di base di un HTLC. Questo tipo di HTLC può essere riscattato *da chiunque* abbia il segreto R. Un HTLC può assumere molte forme diverse con leggere variazioni dello script. Ad esempio, aggiungendo un operatore CHECKSIG e una chiave pubblica nella prima

clausola si limita la redenzione dell'hash a un destinatario con nome, che deve conoscere anche la R segreta.

Canali di Pagamento Instradati (Lightning Network)

Lightning Network è una proposta di rete formata da canali di pagamento bidirezionali collegati end-to-end. Una rete come questa può consentire a qualsiasi partecipante di instradare un pagamento da un canale all'altro senza affidarsi a nessuno degli intermediari. Lightning Network è stato descritto per la prima volta da Joseph Poon e Thadeus Dryja a febbraio 2015,

basandosi sul concetto di canali di pagamento proposto ed elaborato da molti altri.

"Lightning Network" si riferisce a un design specifico di una rete di canali di pagamento instradati, che è stato ora implementato da almeno cinque diversi team open source. Le implementazioni indipendenti sono coordinate da una serie di standard di interoperabilità descritti nel *Basics of Lightning Technology (BOLT)* paper.

Prototipi di implementazioni di Lightning Network sono stati rilasciati da diversi team.

Lightning Network è una delle modalità pensate per implementare

canali di pagamento instradati su bitcoin. Ci sono molti altri progetti che mirano a raggiungere obiettivi simili, come Teechan e Tumblebit.

Esempio di Base di Lightning Network

Vediamo come funziona.

In questo esempio, abbiamo cinque partecipanti: Alice, Bob, Carol, Diana ed Eric. Questi cinque partecipanti hanno aperto canali di pagamento tra loro, in coppia. Alice ha un canale di pagamento con Bob. Bob è collegato a Carol, Carol a Diana e Diana a Eric. Per semplicità supponiamo che ogni canale sia finanziato con 2 bitcoin da

ciascun partecipante, per una capacità totale di 4 bitcoin in ciascun canale.

Una serie di canali di pagamento bidirezionali collegati per formare una rete Lightning che può instradare un pagamento da Alice a Eric mostra cinque partecipanti di una rete Lightning, collegati da canali di pagamento bidirezionali che possono essere collegati per effettuare un pagamento da Alice a Eric (Canali di pagamento indirizzati (Lightning Network)).

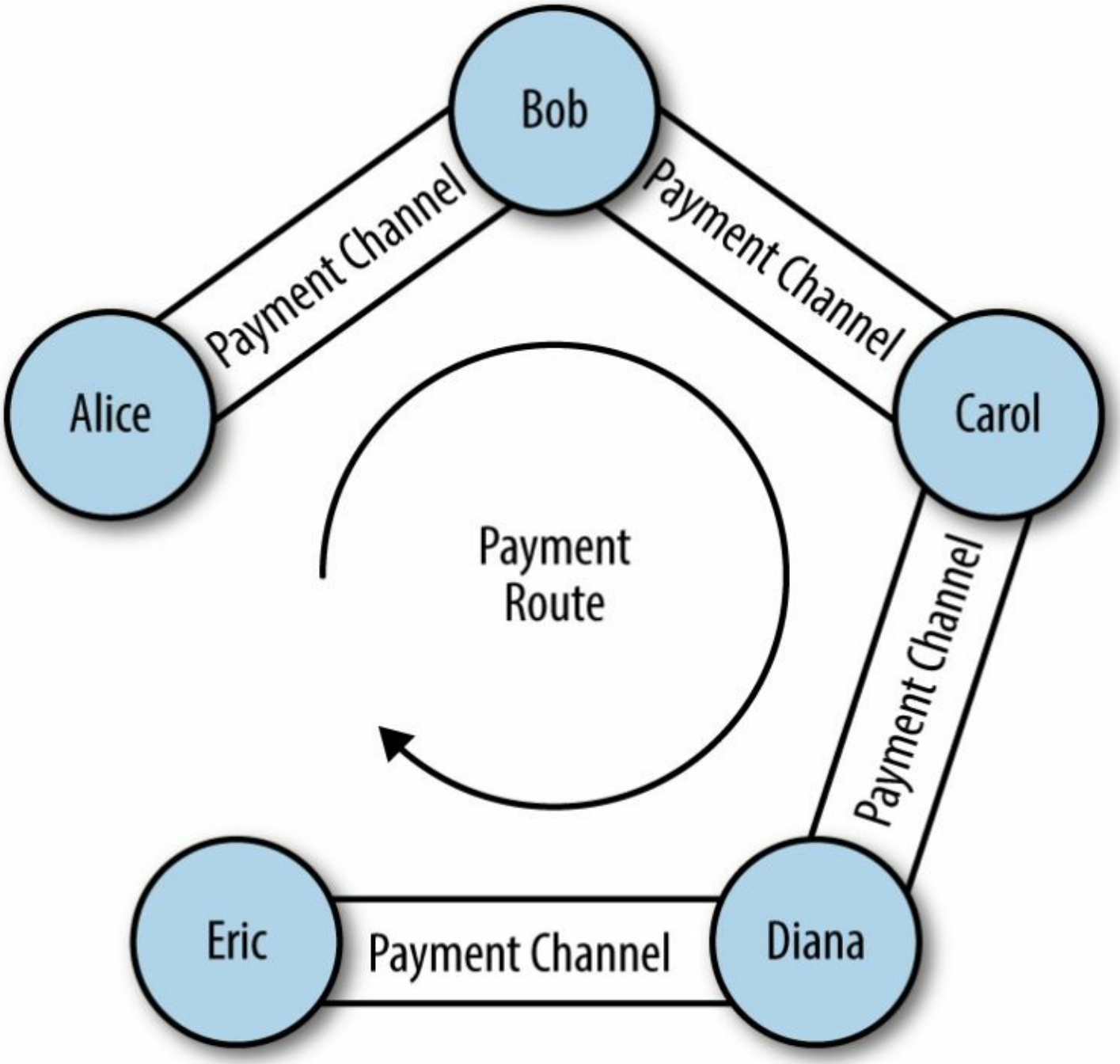


Figura 80. Una serie di canali di pagamento bidirezionali collegati per formare una rete Lightning

che può instradare un pagamento da Alice a Eric

Alice vuole pagare 1 bitcoin a Eric. Tuttavia, Alice non è collegata a Eric da un canale di pagamento. La creazione di un canale di pagamento richiede una transazione di finanziamento, che deve essere impegnata nella blockchain di bitcoin. Alice non vuole aprire un nuovo canale di pagamento e impegnare più fondi. C'è un modo per pagare Eric indirettamente?

Instradamento dei pagamenti passo-passo attraverso Lightning Network mostra il processo passo-passo per instradare un pagamento da Alice a

Eric, attraverso una serie di impegni HTLC sui canali di pagamento che collegano i partecipanti.

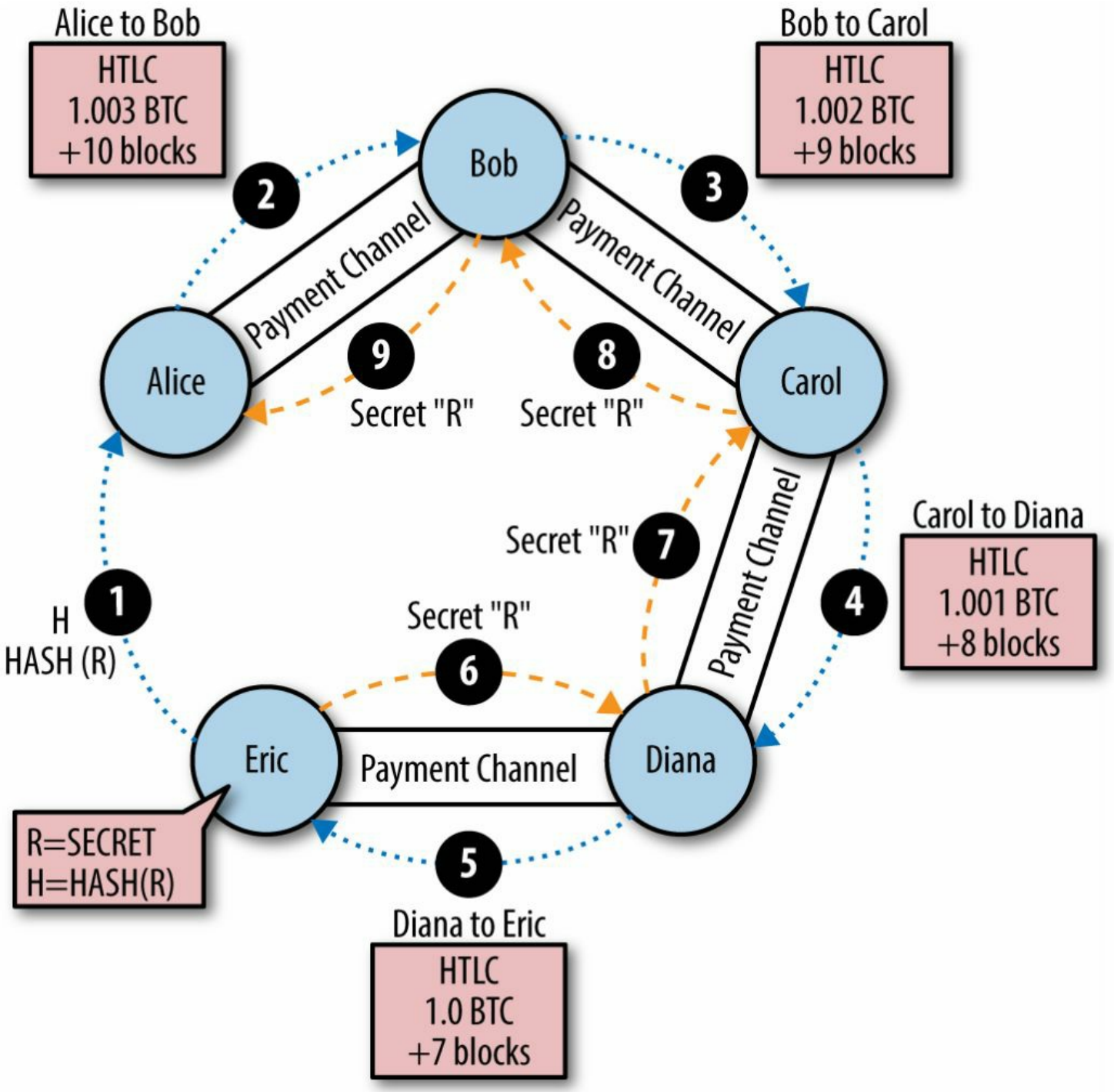


Figura 81. Instradamento dei pagamenti passo-passo attraverso Lightning Network

Alice sta eseguendo un nodo Lightning Network (LN) che sta tenendo traccia del suo canale di pagamento con Bob e ha la possibilità di scoprire percorsi tra i canali di pagamento. Il nodo LN di Alice ha anche la possibilità di connettersi via Internet al nodo LN di Eric. Il nodo LN di Eric crea una R segreta usando un generatore di numeri casuali. Il nodo di Eric non rivela questo segreto a nessuno. Al contrario, il nodo di Eric calcola un hash H della R segreta e trasmette questo hash al nodo di Alice (vedi passaggio 1 di Instradamento dei pagamenti passo-passo attraverso Lightning Network).

Ora il nodo LN di Alice costruisce una rotta tra il nodo LN di Alice e il nodo

LN di Eric. L'algoritmo di routing utilizzato verrà esaminato più dettagliatamente in seguito, ma per ora supponiamo che il nodo di Alice possa trovare un percorso efficiente.

Il nodo di Alice quindi costruisce un HTLC, pagabile all'hash H, con un timeout di rimborso di 10 blocchi (blocco corrente + 10), per un importo di 1.003 bitcoin (vedi passaggio 2 di Instradamento dei pagamenti passo-passo attraverso Lightning Network). Lo 0.003 extra verrà utilizzato per compensare i nodi intermedi per la loro partecipazione a questa rotta di pagamento. Alice invia questo HTLC a Bob, sottraendo 1.003 bitcoin dal suo saldo di canale con Bob e

impegnandolo in HTLC. L'HTLC ha il seguente significato: "*Alice sta impegnando 1.003 del saldo del suo canale da pagare a Bob se Bob conosce il segreto o effettuando un rimborso al saldo di Alice se trascorrono 10 blocchi*". Il saldo del canale tra Alice e Bob è ora espresso da transazioni di impegno con tre output: 2 saldo bitcoin per Bob, 0.997 saldo bitcoin per Alice, 1.003 bitcoin impegnati nell'HTLC di Alice. Il saldo di Alice è ridotto dell'importo impegnato nell'HTLC.

Bob ora ha l'impegno per cui se è in grado di ottenere la R segreta entro i prossimi 10 blocchi, può rivendicare i 1.003 bitcoin bloccati da Alice. Con

questo impegno in mano, il nodo di Bob crea un HTLC sul suo canale di pagamento con Carol. L'HTLC di Bob impegna 1.002 bitcoin a hash H per 9 blocchi, che Carol può riscattare se possiede R segreta (vedi passaggio 3 di Instradamento dei pagamenti passo-passo attraverso Lightning Network). Bob sa che se Carol vuole rivendicare il suo HTLC, deve produrre R. Se Bob ha R in nove blocchi, può usarlo per rivendicare l'HTLC di Alice. Inoltre, guadagna 0.001 bitcoin per aver impegnato il suo saldo di canale per nove blocchi. Se Carol non è in grado di rivendicare il suo HTLC e non è in grado di rivendicare l'HTLC di Alice, tutto torna ai precedenti saldi dei

canali e nessuno è in perdita. Il bilanciamento del canale tra Bob e Carol è ora: 2 a Carol, 0.998 a Bob, 1.002 impegnati da Bob nell'HTLC.

Carol ora ha l'impegno e se ottiene R entro i prossimi nove blocchi, può rivendicare 1.002 bitcoin bloccati da Bob. Ora può assumere un impegno HTLC sul suo canale con Diana. Crea un HTLC di 1.001 bitcoin nell'hash H, per otto blocchi, che Diana può riscattare se ha R segreta (vedi il passaggio 4 di Instradamento dei pagamenti passo-passo attraverso Lightning Network). Dal punto di vista di Carol, se tutto funziona guadagna 0.001 bitcoin e se non funziona non perde nulla. Il suo HTLC a Diana è

praticabile solo se viene rivelato R, a quel punto può rivendicare l'HTLC da Bob. Il bilanciamento del canale tra Carol e Diana è ora: 2 a Diana, 0.999 a Carol, 1.001 impegnati da Carol nell'HTLC.

Infine, Diana può offrire un HTLC a Eric, impegnando 1 bitcoin per sette blocchi nell'hash H (vedi il passaggio 5 di Instradamento dei pagamenti passo-passo attraverso Lightning Network). Il bilanciamento del canale tra Diana ed Eric è ora: 2 a Eric, 1 a Diana, 1 impegnato da Diana nell'HTLC.

Tuttavia, a questo punto del percorso, Eric *ha* il segreto R. Può quindi rivendicare l'HTLC offerto da Diana.

Invia R a Diana e rivendica 1 bitcoin, aggiungendolo al suo saldo di canale (vedi il passaggio 6 di Instradamento dei pagamenti passo-passo attraverso Lightning Network). Il saldo del canale è ora: 1 a Diana, 3 a Eric.

Ora Diana ha un segreto R. Pertanto, può reclamare l'HTLC da Carol. Diana trasmette R a Carol e aggiunge 1.001 bitcoin al suo saldo di canale (vedi il passaggio 7 di Instradamento dei pagamenti passo-passo attraverso Lightning Network). Ora l'equilibrio del canale tra Carol e Diana è: 0.999 per Carol, 3.001 per Diana. Diana ha "guadagnato" 0.001 per aver partecipato a questa via di pagamento.

Tornando indietro lungo il percorso, la

R segreta consente a ciascun partecipante di rivendicare gli HTLC in sospeso. Carol rivendica 1.002 da Bob, impostando il saldo sul proprio canale a: 0.998 a Bob, 3.002 a Carol (vedi il passaggio 8 di Instradamento dei pagamenti passo-passo attraverso Lightning Network. Infine, Bob rivendica l'HTLC da Alice (vedi il passaggio 9 di Instradamento dei pagamenti passo-passo attraverso Lightning Network). Il saldo di canale viene aggiornato così: 0.997 ad Alice, 3.003 a Bob.

Alice ha pagato 1 bitcoin a Eric senza aver avuto la necessità di aprire un canale diretto con lui. Nessuna delle parti intermedie nel percorso di

pagamento doveva fidarsi reciprocamente l'uno dell'altro. Per aver impegnato dei loro fondi nel canale sono stati tutti in grado di guadagnare una piccola commissione, con il solo rischio di un piccolo ritardo nel rimborso se il canale fosse stato stato chiuso o il pagamento indirizzato non fosse riuscito.

Trasporto e Routing di Lightning Network

Tutte le comunicazioni tra i nodi LN sono crittografate punto a punto. Inoltre, i nodi hanno una chiave pubblica a lungo termine che usano come identificatore e per autenticarsi reciprocamente..

Ogni volta che un nodo desidera inviare un pagamento a un altro nodo, deve prima costruire un *percorso* attraverso la rete collegando i canali di pagamento con capacità sufficiente. I nodi pubblicizzano le informazioni di routing, inclusi i canali aperti, la capacità di ciascun canale e le tariffe applicate per instradare i pagamenti. Le informazioni di routing possono essere condivise in vari modi ed è probabile che emergano protocolli di routing diversi man mano che la tecnologia di Lightning Network avanza. Alcune implementazioni di Lightning Network utilizzano il protocollo IRC come meccanismo conveniente per i nodi per annunciare

le informazioni di routing. Un'altra implementazione del rilevamento delle rotte utilizza un modello P2P in cui i nodi propagano gli annunci dei canali ai loro pari, in un modello "flooding", simile al modo in cui bitcoin propaga le transazioni. I piani futuri includono una proposta chiamata Flare, che è un modello di routing ibrido con "quartieri" di nodi locali e nodi beacon a lungo raggio.

Nell'esempio precedente, il nodo di Alice utilizza uno di questi meccanismi di individuazione del percorso per trovare uno o più percorsi che collegano il suo nodo al nodo di Eric. Una volta che il nodo di Alice crea un percorso, inizializzerà

quel percorso attraverso la rete, propagando una serie di istruzioni crittografate e nidificate per connettere ciascuno dei canali di pagamento adiacenti.

È importante sottolineare che questo percorso è noto solo al nodo di Alice. Tutti gli altri partecipanti al percorso di pagamento vedono solo i nodi adiacenti. Dal punto di vista di Carol, questo sembra un pagamento da Bob a Diana. Carol non sa che Bob sta effettivamente trasmettendo un pagamento da Alice. Inoltre, non sa che Diana invierà un pagamento a Eric.

Questa è una caratteristica fondamentale di Lightning Network,

perché garantisce la riservatezza dei pagamenti e rende molto difficile applicare tecniche di sorveglianza, di censura o blacklist. Ma come fa Alice a stabilire questo percorso di pagamento, senza rivelare nulla ai nodi intermedi?

Lightning Network implementa un protocollo onion-routed basato su uno schema chiamato Sphinx. Questo protocollo di routing garantisce che un mittente del pagamento possa costruire e comunicare un percorso attraverso la rete Lightning in modo tale che:

- I nodi intermedi possono verificare e decrittografare la loro porzione di informazioni sul percorso e trovare

l'hop successivo.

- A parte il nodo precedente e successivo, non possono conoscere altri nodi che fanno parte del percorso.
- Non possono identificare la lunghezza del percorso di pagamento o la propria posizione in tale percorso.
- Ogni parte del percorso è crittografata in modo tale che un utente malintenzionato a livello di rete non possa associare i pacchetti

da parti diverse del percorso tra loro.

- A differenza di Tor (un protocollo di anonimizzazione onion-routed su Internet), non esistono "nodi di uscita" che possono essere posti sotto sorveglianza. Non è necessario che i pagamenti vengano trasmessi alla blockchain bitcoin; i nodi aggiornano semplicemente i saldi dei canali.

Usando questo protocollo onion-routed, Alice avvolge ogni elemento del percorso in un livello di

crittografia, iniziando dalla fine e procedendo all'indietro. Cripta un messaggio a Eric con la chiave pubblica di Eric. Questo messaggio è racchiuso in un messaggio crittografato a Diana, identificando Eric come destinatario successivo. Il messaggio a Diana è racchiuso in un messaggio crittografato con la chiave pubblica di Carol che identifica Diana come destinatario successivo. Il messaggio a Carol è crittografato sulla chiave di Bob. Pertanto, Alice ha costruito questa "cipolla" multistrato crittografata di messaggi. Invia questa a Bob, che può solo decifrare e scartare lo strato esterno. All'interno, Bob trova un messaggio indirizzato a

Carol che può inoltrare a Carol ma che non può decifrare da solo. Seguendo il percorso, i messaggi vengono inoltrati, decrittografati, inoltrati, ecc., fino a Eric. Ogni partecipante conosce solo il nodo precedente e successivo in ciascun hop.

Ogni elemento del percorso contiene informazioni sull'HTLC che devono essere estese all'hop successivo, l'importo che viene inviato, la tariffa da includere e la scadenza del blocco CLTV (in blocchi) dell'HTLC. Man mano che le informazioni sul percorso si propagano, i nodi assumono impegni HTLC per l'hop successivo.

A questo punto, ti starai chiedendo come sia possibile che i nodi non

conoscano la lunghezza del percorso e la loro posizione in quel percorso. Dopo tutto, ricevono un messaggio e lo inoltrano all'hop successivo. Non si tratta alla fine di consentire loro di dedurre le dimensioni del percorso e la loro posizione? Per evitare ciò, il percorso è sempre fissato a 20 hop e riempito con dati casuali. Ogni nodo vede l'hop successivo e un messaggio crittografato a lunghezza fissa da inoltrare. Solo il destinatario finale vede che non c'è hop successivo. A tutti gli altri sembra che ci siano sempre altri 20 hop da fare.

Benefici Di Lightning Network

Lightning Network è una tecnologia di routing di secondo livello. Può essere applicato a qualsiasi blockchain che supporti alcune funzionalità di base, come transazioni multisignature, timelock e smart contract di base.

Se Lightning Network è sovrapposta alla rete bitcoin, la rete bitcoin stessa può ottenere un aumento significativo di capacità, privacy, granularità e velocità, senza sacrificare i principi di funzionamento della fiducia senza intermediari:

Privacy

I pagamenti di Lightning Network sono molto più privati dei pagamenti

sulla blockchain bitcoin, in quanto non sono pubblici. Mentre i partecipanti di un percorso possono vedere i pagamenti propagati attraverso i loro canali, non conoscono il mittente o il destinatario.

Fungibilità

Una rete Lightning rende molto più difficile applicare tecniche di sorveglianza e le blacklist che sono possibili su bitcoin, aumentando la fungibilità della valuta.

Velocità

Le transazioni bitcoin che utilizzano Lightning Network sono regolate in millisecondi, anziché in minuti,

poiché gli HTLC vengono cancellati senza impegnare le transazioni in un blocco.

Granularità

Una rete Lightning può consentire pagamenti tanto piccoli quanto il limite di "dust" di bitcoin, forse anche più piccoli. Alcune proposte consentono incrementi subsatoshi.

Capacità

Una rete Lightning aumenta la capacità del sistema bitcoin di diversi ordini di grandezza. Non esiste un limite superiore pratico al numero di pagamenti al secondo che possono essere instradati su una rete Lightning, poiché dipende solo dalla

capacità e dalla velocità di ciascun nodo.

Operazioni senza fiducia (trustless)

Una rete Lightning effettua transazioni bitcoin tra nodi che funzionano come peer senza fidarsi l'uno dell'altro. Pertanto, una rete Lightning preserva i principi del sistema bitcoin, espandendo significativamente i suoi parametri operativi.

Naturalmente, come accennato in precedenza, il protocollo Lightning Network non è l'unico modo per implementare i canali di pagamento indirizzati. Altri sistemi proposti includono Tumblebit e Teechan. Al momento, tuttavia, la rete Lightning è già stata implementata su mainnet.

Diversi team hanno sviluppato implementazioni concorrenti di LN e stanno lavorando verso uno standard di interoperabilità comune (chiamato BOLT).

Conclusione

Abbiamo esaminato solo alcune delle applicazioni emergenti che possono essere create utilizzando la blockchain di bitcoin come piattaforma di trust. Queste applicazioni estendono l'ambito del bitcoin oltre i pagamenti e oltre gli strumenti finanziari, per includere molte altre applicazioni in cui la fiducia è fondamentale. Decentrando le basi della fiducia, la blockchain di bitcoin è una piattaforma

che genererà molte applicazioni rivoluzionarie in un'ampia varietà di settori.

The Bitcoin Whitepaper di Satoshi Nakamoto

NOTA	Questo è l'originale Whitepaper, riprodotto integralmente come fu pubblicato da Satoshi Nakamoto nell'Ottobre 2008.

Bitcoin - Un Sistema di moneta elettronica Peer-to-Peer

Satoshi Nakamoto

satoshin@gmx.com

www.bitcoin.org

Estratto

Una versione puramente peer-to-peer di denaro elettronico potrebbe permettere di inviare direttamente pagamenti online da una parte all'altra senza passare attraverso una istituzione finanziaria. Le firme digitali forniscono una parte della soluzione, ma i principali benefici sono persi se una terza parte fidata è ancora necessaria per evitare la doppia spesa. Proponiamo una soluzione al problema della doppia spesa utilizzando una rete peer-to-peer. Le transazioni di data e ora della rete sono suddivise in una catena continua di prove di lavoro basate su hash, che formano un record

che non può essere modificato senza ripetere la prova di lavoro. La catena più lunga non serve solo come prova della sequenza di eventi certificati, ma prova che proviene dal più grande pool di potenza delle CPU. Finché la maggior parte della potenza della CPU è controllata da nodi che non cooperano per attaccare la rete, genereranno la catena più lunga e oltrepassano gli attaccanti. La rete stessa richiede una struttura minima. I messaggi vengono trasmessi in modo ottimale ed i nodi possono disconnettersi e riconnettersi alla rete a loro piacimento, accettando la più lunga catena di prove di lavoro come prova di ciò che è accaduto mentre non

erano presenti

Introduzione

Il commercio su Internet ha finito per basarsi quasi esclusivamente su istituzioni finanziarie che servono come terze parti di fiducia per elaborare i pagamenti elettronici. Mentre il sistema funziona abbastanza bene per la maggior parte delle transazioni, soffre ancora delle debolezze intrinseche del modello basato sulla fiducia. Le transazioni completamente non reversibili non sono realmente possibili, dal momento che le istituzioni finanziarie non possono evitare di mediare le controversie. Il costo della mediazione

aumenta i costi di transazione, limitando le dimensioni minime delle transazioni pratiche ed escludendo la possibilità di piccole transazioni occasionali, e vi è un costo più ampio nella perdita di capacità di effettuare pagamenti non reversibili per servizi non reversibili. Con la possibilità di inversione, si diffonde la necessità di fiducia. I commercianti devono diffidare dei loro clienti, infastidendoli per avere più informazioni di quelle che altrimenti avrebbero bisogno. Una certa percentuale di frodi è accettata come inevitabile. Questi costi ed incertezze di pagamento possono essere evitati di persona utilizzando la valuta fisica, ma

non esiste alcun meccanismo per effettuare pagamenti su un canale di comunicazione senza una parte fidata.

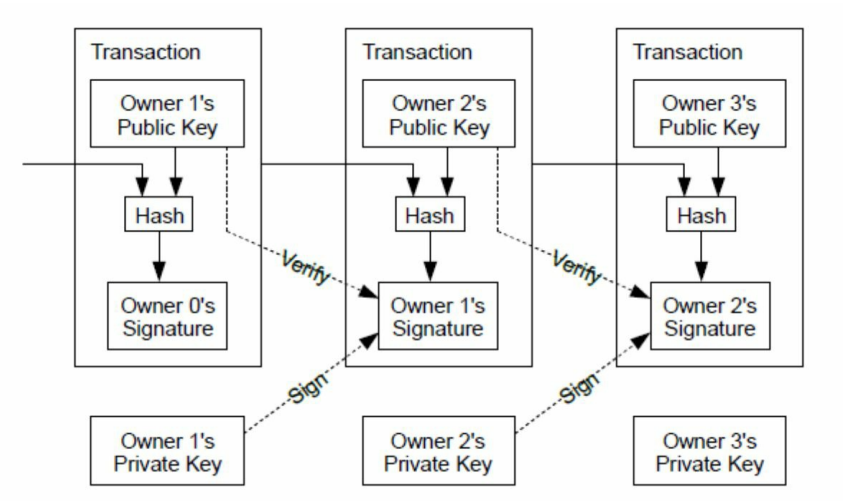
Ciò di cui abbiamo bisogno è un sistema di pagamento elettronico basato su prove crittografiche anziché sulla fiducia, che consente a due parti volontarie di negoziare direttamente tra loro senza la necessità di una terza parte fidata. Le transazioni che sono computazionalmente impraticabili da invertire proteggerebbero i venditori dalle frodi e potrebbero essere facilmente implementati meccanismi con depositi di garanzia per proteggere gli acquirenti. In questo documento, proponiamo una soluzione al problema della doppia spesa utilizzando un

"timestamp server" distribuito peer-to-peer per generare prove computazionali dell'ordine cronologico delle transazioni. Il sistema è sicuro fino a quando i nodi onesti controllano collettivamente più potenza di calcolo rispetto a qualsiasi gruppo cooperante di nodi attaccanti.

Transazioni

Definiamo una moneta elettronica come una catena di firme digitali. Ogni proprietario trasferisce la moneta alla successiva firmando digitalmente un hash della transazione precedente e la chiave pubblica del prossimo proprietario e aggiungendoli alla fine della moneta. Un beneficiario può

verificare le firme per verificare la catena di proprietà.



Il problema, naturalmente, è che il beneficiario non può verificare che uno dei proprietari non abbia speso il doppio della moneta. Una soluzione comune è l'introduzione di un'autorità centrale affidabile, o Zecca, che controlla che ogni transazione non sia stata spesa due volte. Dopo ogni transazione, la moneta deve essere restituita alla zecca per emettere una nuova moneta, e solo le monete emesse

direttamente dalla zecca sono attendibili per non essere spese due volte. Il problema con questa soluzione è che il destino dell'intero sistema monetario dipende dalla società che gestisce la zecca, con ogni transazione che deve attraversarli, proprio come una banca.

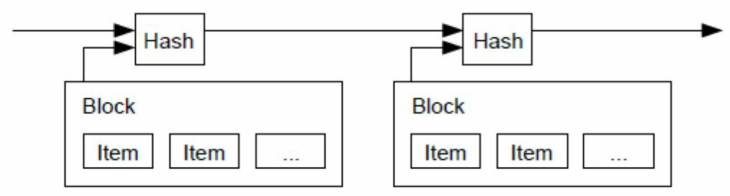
Abbiamo bisogno di un modo per il beneficiario di sapere che i precedenti proprietari non hanno firmato alcuna transazione precedente. Per i nostri scopi, la prima transazione è quella che conta, quindi non ci interessano i successivi tentativi di doppia spesa. L'unico modo per confermare l'assenza di una transazione è essere a conoscenza di tutte le transazioni. Nel

modello basato sulla zecca, la zecca era a conoscenza di tutte le transazioni e decideva quale arrivò prima. Per ottenere ciò senza una parte fidata, le transazioni devono essere annunciate pubblicamente [1] e abbiamo bisogno di un sistema che permetta ai partecipanti di concordare una singola cronologia dell'ordine in cui sono state ricevute. Il beneficiario deve dimostrare che al momento di ogni transazione, la maggior parte dei nodi concordava che era la prima ricevuta.

Timestamp Server

La soluzione che proponiamo inizia con un server timestamp. Un server timestamp funziona prendendo un hash

di un blocco di elementi per essere timestamped e pubblicando l'hash a tutta la rete, ad esempio in un giornale o in un post di Usenet [2-5]. Il timestamp dimostra che i dati devono essere esistiti al momento, ovviamente, per entrare nell'hash. Ogni timestamp include il timestamp precedente nel suo hash, formando una catena, con ogni timestamp aggiuntivo che conferma quelli precedenti.

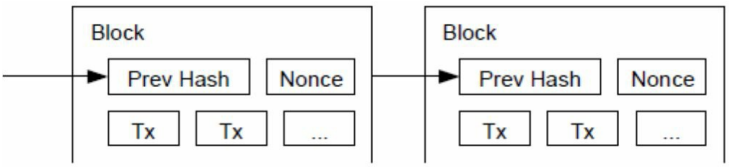


Proof-of-Work (Prova di lavoro)

Per implementare un server timestamp distribuito su base peer-to-peer, sarà

necessario utilizzare un sistema di proof-of-work simile all'Hashcash di Adam Back [6], piuttosto che ai giornali o ai post Usenet. Il proof-of-work implica la scansione di un valore che quando "hashed", ad esempio con SHA-256, l'hash inizia con un numero di bit a zero. Il lavoro medio richiesto è esponenziale nel numero di bit zeri richiesti e può essere verificato eseguendo un singolo hash. Per la nostra rete timestamp, implementiamo il proof-of-work incrementando un nonce nel blocco finché non viene trovato un valore che fornisce all'hash del blocco gli zeri richiesti. Una volta esaurito lo sforzo della CPU per soddisfare le prove di lavoro, il

blocco non può essere modificato senza ripetere il lavoro. Poiché i blocchi successivi sono concatenati, il lavoro di modifica del blocco include la ripetizione di tutti i blocchi dopo di esso.



Il proof-of-work risolve anche il problema di determinare la rappresentazione nel processo decisionale a maggioranza. Se la maggioranza era basata su "un indirizzo IP-un voto", poteva essere sovvertita da chiunque fosse in grado di allocare molti indirizzi IP. La prova di lavoro è essenzialmente una CPU-un voto. La decisione maggioritaria è

rappresentata dalla catena più lunga, che ha il maggiore sforzo di lavoro investito in essa. Se la maggior parte della potenza della CPU è controllata da nodi onesti, la catena onesta crescerà più velocemente e oltrepasserà qualsiasi catena concorrente. Per modificare un blocco passato, un utente malintenzionato dovrebbe ripetere le bozze del blocco e tutti i blocchi dopo di esso e quindi recuperare e superare il lavoro dei nodi onesti. Mostreremo in seguito che la probabilità che un attaccante più lento recuperi, diminuisce esponenzialmente man mano che i blocchi successivi vengono aggiunti.

Per compensare l'aumento della

velocità dell'hardware e il diverso interesse per l'esecuzione dei nodi nel tempo, la difficoltà della Proof-of-Work è determinata da una media mobile che mira ad un determinato numero medio di blocchi all'ora. Se vengono generati troppo velocemente, la difficoltà aumenta.

Rete

I passi per avviare la rete sono i seguenti:

1. Nuove transazioni sono spedite a tutti i nodi.
2. Ogni nodo raccoglie le nuove transazioni dentro un blocco.
3. Ogni nodo lavora per trovare la Proof-of-work del proprio blocco.

4. Quando un nodo trova la Proof-of-Work, spedisce il blocco a tutti gli altri nodi.
5. I nodi accettano il blocco solo se tutte le transazioni all'interno sono valide e non sono state già spese.
6. I nodi esprimono la loro accettazione del blocco, lavorando per creare il prossimo blocco nella catena, usando l'hash del blocco accettato come hash precedente.

I nodi considerano sempre la catena più lunga quella corretta e continueranno a lavorare per estenderla. Se due nodi trasmettono contemporaneamente versioni diverse del blocco successivo, alcuni nodi potrebbero ricevere prima uno o

l'altro. In tal caso, lavorano sul primo che hanno ricevuto, ma salvano l'altro ramo nel caso in cui diventi più lungo. Il legame verrà interrotto quando verrà trovata la successiva Prova di Lavoro (PoW) e un ramo si allungherà; i nodi che stavano lavorando sull'altro ramo passeranno quindi a quello più lungo.

Le nuove transazioni trasmesse alla rete non devono necessariamente raggiungere tutti i nodi. Finché raggiungono molti nodi, entreranno prima in un blocco. I blocchi spediti sulla rete sono anche tolleranti per i messaggi omessi. Se un nodo non riceve un blocco, lo richiederà quando riceve il blocco successivo e si rende conto di averne perso uno.

Incentivi

Per convenzione, la prima transazione in un blocco è una transazione speciale che conia una nuova moneta di proprietà del creatore del blocco. Ciò aggiunge un incentivo per i nodi a supportare la rete e fornisce un modo per distribuire inizialmente le monete in circolazione, poiché non esiste un'autorità centrale per emetterle. L'aggiunta costante di una quantità costante di nuove monete è analoga a quella dei minatori d'oro che spendono risorse per aggiungere oro alla circolazione. Nel nostro caso, è il tempo della CPU e l'elettricità che è stata spesa.

L’incentivo può anche essere finanziato con commissioni sulle transazioni. Se il valore di uscita di una transazione è inferiore al suo valore di input, la differenza è una commissione di transazione che viene aggiunta al valore di incentivo del blocco contenente la transazione. Una volta entrato in circolazione un numero prestabilito di monete, l’incentivo può passare interamente alle commissioni di transazione ed essere completamente privo di inflazione.

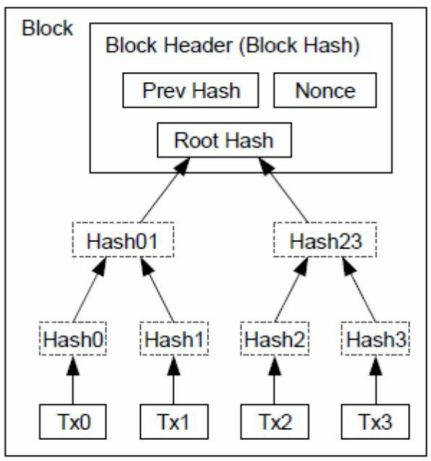
L’incentivo incoraggia i nodi a rimanere onesti. Se un avido attaccante è in grado di raggruppare più potenza di elaborazione rispetto a tutti i nodi onesti, dovrebbe scegliere tra usarlo

per frodare le persone rubando indietro i suoi pagamenti, o usarlo per generare nuove monete. Dovrebbe trovare più proficuo giocare secondo le regole, regole che lo favoriscono con più monete nuove di quelle di chiunque altro, piuttosto che indebolire il sistema e la validità della propria ricchezza.

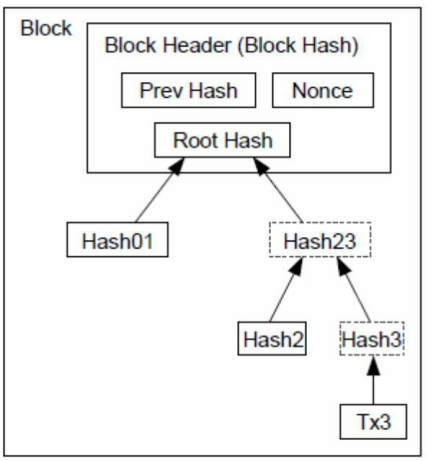
Recuperare Spazio su Disco

Una volta che l'ultima transazione in una moneta è coperta da un numero sufficiente di blocchi, le transazioni spese prima di essa possono essere scartate per risparmiare spazio su disco. Per facilitare ciò senza interrompere l'hash del blocco, le

transazioni vengono sottoposte a hashing in un Merkle Tree [7] [2] [5], con solo la radice inclusa nell'hash del blocco. I vecchi blocchi possono quindi essere compattati rimuovendo i rami dell'albero. Gli hash interni non hanno bisogno di essere memorizzati.



Transactions Hashed in a Merkle Tree



After Pruning Tx0-2 from the Block

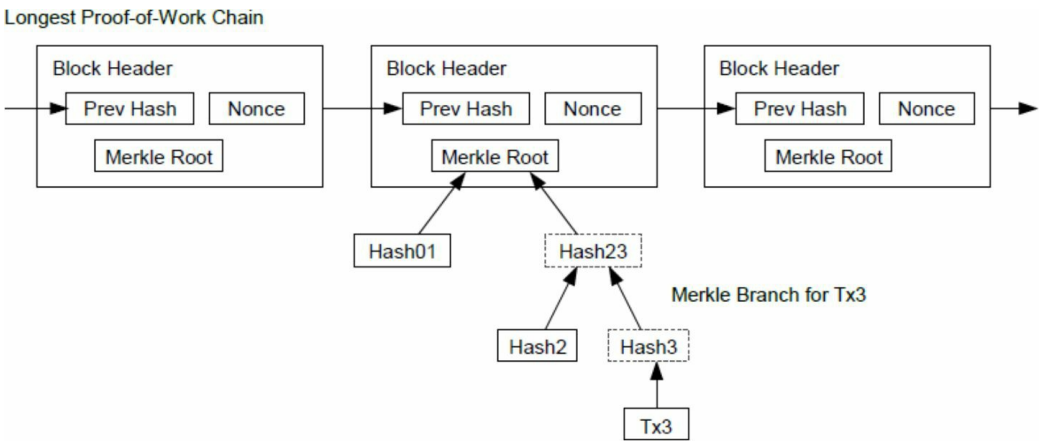
L'intestazione di un blocco senza transazioni occuperebbe circa 80 byte. Se supponiamo che i blocchi vengano generati ogni 10 minuti, 80 byte * 6 *

24 * 365 == 4,2 MB all'anno. Con i sistemi informatici in genere venduti con 2 GB di RAM a partire dal 2008, e la legge di Moore che prevede una crescita attuale di 1,2 GB all'anno, la memorizzazione non dovrebbe rappresentare un problema anche se le intestazioni dei blocchi devono essere conservate in memoria.

Verifica dei pagamenti semplificata

È possibile verificare i pagamenti senza eseguire un nodo di rete completo. Un utente deve solo mantenere una copia delle intestazioni dei blocchi della catena di prove più lunga, che può ottenere interrogando i

nodi della rete finché non è convinto di avere la catena più lunga e ottenere il ramo di Merkle che collega la transazione al blocco che ha il "timestamp" al suo interno. L'utente non può controllare la transazione da solo, ma collegandolo a un punto della catena, può vedere che un nodo della rete l'ha accettato e blocchi aggiunti dopo aver confermato ulteriormente che la rete l'ha accettato.



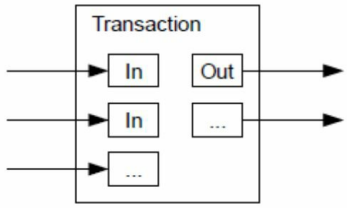
In quanto tale, la verifica è affidabile fintanto che i nodi onesti controllano la rete, ma è più vulnerabile se la rete

viene sopraffatta da un utente malintenzionato. Mentre i nodi di rete possono verificare le transazioni da soli, il metodo semplificato può essere ingannato dalle transazioni inventate da un utente malintenzionato fino a quando l'attaccante può continuare a sopraffare la rete. Una strategia per proteggersi da questo sarebbe accettare gli avvisi dai nodi di rete quando rilevano un blocco non valido, chiedendo al software dell'utente di scaricare il blocco completo e le transazioni sospettate per confermare l'incoerenza. Le aziende che ricevono pagamenti frequenti probabilmente vorranno ancora gestire i propri nodi per una sicurezza più indipendente e

una verifica più rapida.

Combinazione e divisione del valore

Anche se sarebbe possibile gestire le monete singolarmente, sarebbe ingombrante effettuare una transazione separata per ogni centesimo da trasferire. Per consentire la divisione e il raggruppamento dei valori, le transazioni contengono più input e output. Normalmente ci sarà un singolo input da una precedente transazione più grande o più input che combina quantità minori e al massimo due output: uno per il pagamento e uno che restituisce il resto, se presente, al mittente.

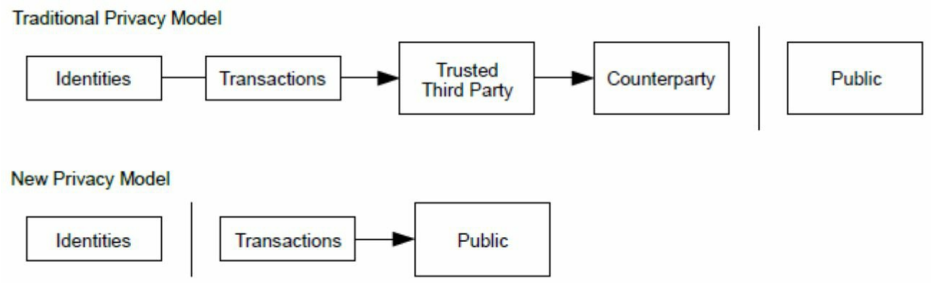


Va notato che il "fan-out", in cui una transazione dipende da diverse transazioni, e quelle transazioni dipendono da molte altre, non è un problema. Non è mai necessario estrarre una copia completa della cronologia di una transazione.

Privacy

Il modello bancario tradizionale raggiunge un livello di privacy limitando l'accesso alle informazioni alle parti coinvolte e alla terza parte fidata. La necessità di annunciare pubblicamente tutte le transazioni preclude questo metodo, ma la privacy

può essere mantenuta interrompendo il flusso di informazioni in un altro punto: mantenendo anonime le chiavi pubbliche. Il pubblico può vedere che qualcuno sta inviando un importo a qualcun altro, ma senza informazioni che collegano la transazione a chiunque. Questo è simile al livello di informazioni diffuse dalle borse, dove il tempo e le dimensioni dei singoli scambi, il "nastro", sono resi pubblici, ma senza dire chi fossero le parti.



Come protezione aggiuntiva, è necessario utilizzare una nuova coppia di chiavi per ogni transazione per

impedire che vengano collegate a uno stesso proprietario. Alcuni collegamenti sono ancora inevitabili con le transazioni multi-input, che rivelano necessariamente che i loro input erano di proprietà dello stesso proprietario. Il rischio è che se viene rivelato il proprietario di una chiave, il collegamento potrebbe rivelare altre transazioni appartenenti allo stesso proprietario.

Calcoli

Consideriamo lo scenario di un aggressore che cerca di generare una catena alternativa più velocemente della catena onesta. Anche se ciò è portato a termine, non apre il sistema a

cambiamenti arbitrari, come la creazione di valore dal nulla o l'assunzione di denaro che non è mai appartenuto all'attaccante. I nodi non accetteranno una transazione non valida come pagamento e i nodi onesti non accetteranno mai un blocco che li contiene. Un utente malintenzionato può solo provare a cambiare una delle sue transazioni per ritirare i soldi che ha recentemente speso.

La corsa tra catena onesta e catena di attaccanti può essere caratterizzata come una passeggiata casuale binomiale. L'evento di successo è che la catena onesta viene estesa di un blocco, aumentando il suo vantaggio di +1, mentre l'evento maligno è che la

catena dell'attaccante viene estesa di un blocco, riducendo il divario di -1.

La probabilità che un attaccante recuperi da un dato deficit è analoga al problema della rovina del giocatore d'azzardo. Supponiamo che un giocatore con un credito illimitato abbia un deficit e giochi potenzialmente un numero infinito di prove per cercare di raggiungere il pareggio. Possiamo calcolare la probabilità che abbia raggiunto il pareggio o che un attaccante raggiunga la catena onesta, come segue [8]:

p == probabilità che un nodo onesto trova il prossimo blocco

q == probabilità che un attaccante

trova il prossimo blocco

qz == probabilità che un attaccante raggiungerà mai da z i blocchi successivi

$$q_z = \begin{cases} 1 & \text{if } p \le q \\ (q/p)^z & \text{if } p > q \end{cases}$$

Data la nostra ipotesi che p> q, la probabilità scende esponenzialmente come il numero di blocchi che l'attaccante deve recuperare con gli aumenti. Con le probabilità contro di lui, se non fa un salto in avanti molto presto, le sue possibilità diventano incredibilmente piccole mentre cade più indietro.

Consideriamo ora per quanto tempo il destinatario di una nuova transazione deve attendere prima di essere

sufficientemente certo che il mittente non possa modificare la transazione. Assumiamo che il mittente sia un aggressore che vuole far credere al destinatario di averlo pagato per un po ', quindi di cambiarlo per ripagarsi dopo un po' di tempo. Il ricevitore verrà avvisato quando ciò accadrà, ma il mittente spera che sarà troppo tardi.

Il ricevitore genera una nuova coppia di chiavi e consegna la chiave pubblica al mittente poco prima della firma. Questo impedisce al mittente di preparare una catena di blocchi in anticipo, lavorando su di esso continuamente fino a quando non è abbastanza fortunato da arrivare abbastanza avanti, quindi eseguendo la

transazione in quel momento. Una volta inviata la transazione, il mittente disonesto inizia a lavorare in segreto su una catena parallela contenente una versione alternativa della sua transazione.

Il destinatario attende che la transazione sia stata aggiunta a un blocco e che i blocchi z siano stati collegati dopo di esso. Non conosce l'esatto ammontare di progressi compiuti dall'attaccante, ma supponendo che i blocchi onesti abbiano impiegato il tempo medio previsto per blocco, il potenziale progresso dell'attaccante sarà una distribuzione di Poisson con il valore atteso:

$$\lambda = z\frac{q}{p}$$

Per ottenere la probabilità che l'attaccante possa ancora recuperare, moltiplichiamo la densità di Poisson per ogni quantità di progresso che avrebbe potuto ottenere dalla probabilità che avrebbe potuto recuperare da quel punto:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & \text{if } k \le z \\ 1 & \text{if } k > z \end{cases}$$

Ottimizzazione per evitare di sommare la coda infinita della distribuzione …

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

Conversione in C code...

```
#include <math.h>
double AttackerSuccessProbability(double
q, int z)
{
   double p == 1.0 - q;
```

```
    double lambda == z * (q / p);
    double sum == 1.0;
    int i, k;
    for (k == 0; k <== z; k++)
    {
        double poisson == exp(-lambda);
        for (i == 1; i <== k; i++)
            poisson *== lambda / i;
        sum -== poisson * (1 - pow(q / p, z -
k));
    }
    return sum;
}
```

Eseguendo alcuni risultati, noi possiamo vedere la probabilità decrescere esponenzialmente con z.

```
q=0.1
z=0 P=1.0000000
z=1 P=0.2045873
z = 2 P = 0.0509779
```

z = 3 P = 0,0131722
z = 4 P = 0,0034552
z = 5 P = 0,0009137
z = 6 P = 0,0002428
z = 7 P = 0,0000647
z = 8 P = 0,0000173
z = 9 P = 0,0000046
z = 10 P = 0,0000012

q = 0,3
z=0 P=1.0000000
z = 5 P = 0,1773523
z = 10 P = 0.0416605
z = 15 P = 0,0101008
z = 20 P = 0,0024804
z = 25 P = 0.0006132
z = 30 P = 0,0001522
z = 35 P = 0,0000379
z = 40 P = 0,0000095
z = 45 P = 0,0000024
z = 50 P = 0,0000006

Risoluzione per P inferione allo 0,1 %
…

P <0,001
q = 0,10 z = 5
q = 0,15 z = 8
q = 0,20 z = 11
q = 0,25 z = 15
q = 0,30 z = 24
q = 0,35 z = 41
q = 0,40 z = 89
q = 0,45 z = 340

Conclusione

Abbiamo proposto un sistema per le transazioni elettroniche senza fare affidamento sulla fiducia. Abbiamo iniziato con il solito quadro di monete ricavate da firme digitali, che fornisce un forte controllo della proprietà, ma è

incompleto senza un modo per evitare la doppia spesa. Per risolvere questo problema, abbiamo proposto una rete peer-to-peer usando una prova di lavoro (proof-of-work) per registrare una cronologia pubblica delle transazioni che diventa rapidamente poco pratica dal punto di vista computazionale per consentire a un utente malintenzionato di alterare se i nodi onesti controllano la maggior parte della potenza di calcolo. La rete è robusta nella sua semplicità non strutturata. I nodi funzionano tutti contemporaneamente con poca coordinazione. Non hanno bisogno di essere identificati, dal momento che i messaggi non sono indirizzati verso un

luogo particolare e devono essere consegnati solo nel miglior modo possibile. I nodi possono disconnettersi e riconnettersi alla rete a loro piacimento, accettando la catena di prove di lavoro come prova di ciò che è accaduto mentre erano disconnessi. Votano con la loro potenza di calcolo, esprimendo la loro accettazione di blocchi validi lavorando per estenderli e rifiutando blocchi non validi, rifiutandosi di lavorare su di essi. Eventuali regole e incentivi necessari possono essere applicati con questo meccanismo di consenso.

Riferimenti

[1] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

[2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In 20th Symposium on Information Theory in the Benelux, May 1999.

[3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In Journal of Cryptology, vol 3, no 2, pages 99-111, 1991.

[4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In Sequences II: Methods in

Communication, Security and Computer Science, pages 329-334, 1993.

[5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In Proceedings of the 4th ACM Conference on Computer and Communications Security, pages 28-35, April 1997.

[6] A. Back, "Hashcash - a denial of service counter-measure," *http://www.hashcash.org/papers/hashc* 2002.

[7] R.C. Merkle, "Protocols for public key cryptosystems," In Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society, pages 122-133, April 1980.

[8] W. Feller, "An introduction to probability theory and its applications," 1957.

License

This whitepaper was published in October 2008 by Satoshi Nakamoto. It was later (2009) added as supporting documentation to the bitcoin software and carries the same MIT license. It has been reproduced in this book, without modification other than formatting, under the terms of the MIT license:

of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO

THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Appendix A: Linguaggio di Scripting delle Transazioni: Operatori, Costanti e Simboli

NOTA	Tabelle e des provenienti https://en.bitcoin.it/wik

Mette un valore nello stack mostra gli

operatori per caricare i valori sullo stack.

Tabella 29. Mette un valore nello st

Simbolo	**Valore (hex)**	**Descr**
OP_0 or OP_FALSE	0x00	Un vuoto imme nello
1-75	0x01-0x4b	Metti pross byte stack N è 75 by

OP_PUSHDATA1	0x4c	Il pr byte contie imme segue byte stack
OP_PUSHDATA2	0x4d	I pr due byte conte N, n segue byte stack
OP_PUSHDATA4	0x4e	I pr quattr

		script conte N, n segue byte stack
OP_1NEGATE	0x4f	Metti il val 1" stack
OP_RESERVED	0x50	Halt Trans non v meno non esegu

		clausc OP_II
OP_1 or OP_TRUE	0x51	Metti valore sullo
OP_2 to OP_16	0x52 to 0x60	For push value onto stack, OP_2 pushe

Flow control condizionale mostra gli operatori di controllo di flusso condizionati.

Tabella 30. Flow control condiziona

Simbolo	Valore (hex)	Descriz
OP_NOP	0x61	Non eseguire niente
OP_VER	0x62	Halt— Invalid transacti unless fo in unexecu OP_IF clause
OP_IF	0x63	Esegui

		istruzion continua se la dello s non è 0
OP_NOTIF	0x64	Esegui istruzion continua se la dello s è 0
OP_VERIF	0x65	Halt— Transaz non vali
OP_VERNOTIF	0x66	Halt— Transaz

		non vali
OP_ELSE	0x67	Esegui se istruzion precede non e state eseguite
OP_ENDIF	0x68	Finel blocco OP_IF, OP_NO OP_ELS
OP_VERIFY	0x69	Controll in cima stack,

		interrom invalida transazi se TRUE
OP_RETURN	0x6a	Interrom invalida transazi

Operazioni Timelock mostra gli operatori usati per timelocks.

Tabella 31. Operazioni Timelock

Simbolo	
OP_CHECKLOCKTIMEVERIFY	

(precedentemente OP_NOP2)

OP_CHECKSEQUENCEVERIFY (previously OP_NOP3)

Stack operations mostra gli operatori usati per manipolare lo stack.

Tabella 32. Stack operations

Simbolo	Valore (hex)
OP_TOALTSTACK	0x6b

OP_FROMALTSTACK	0x6c	
OP_2DROP	0x6d	
OP_2DUP	0x6e	

OP_2DUP	0x6e	
OP_2OVER	0x70	
OP_2ROT	0x71	

OP_2SWAP	0x72	
OP_IFDUP	0x73	
OP_DEPTH	0x74	

OP_DROP	0x75	
OP_DUP	0x76	
OP_NIP	0x77	
OP_OVER	0x78	

OP_PICK	0x79	
OP_ROLL	0x7a	

OP_ROT	0x7b	
OP_SWAP	0x7c	
OP_TUCK	0x7d	

Operazioni di concatenamento di stringhe mostra gli operatori stringa.

Tabella 33. Operazioni concatenamento di stringhe

Simbolo	Valore (hex)	Descrizion
OP_CAT	0x7e	Disabilitat (concatena due eleme in cima)

OP_SUBSTR	0x7f	Disabilitato (ritorna u sottostringa
OP_LEFT	0x80	Disabilitato (ritorna substring sinistra)
OP_RIGHT	0x81	Disabilitato (ritorna substring destra)
OP_SIZE	0x82	Calcola lunghezza della strin dell'elemen

		in cima pusha risultato

Aritmetica binaria e condizionali mostra gli operatori logici booleani e dell'aritmetica binaria.

Tabella 34. Aritmetica binaria e con

Simbolo	**Valore (hex)**	**De**
OP_INVERT	0x83	Dis (Ca bit del in c

OP_AND	0x84	Dis (Bo AN ele cim
OP_OR	0x85	Dis (Bo OR ele cim
OP_XOR	0x86	Dis (Bo XO ele cim
OP_EQUAL	0x87	Me

		(1) ele cim esa ide altr met (0)
OP_EQUALVERIFY	0x88	Co OP ma OP dop fer se
OP_RESERVED1	0x89	Ha

		Tra inv nor una OP ina
OP_RESERVED2	0x8a	Ha Tra inv nor una OP ese

Operatori numerici mostra gli operatori numerici (aritmetici).

Tabella 35. Operatori numerici

Simbolo
OP_1ADD
OP_1SUB
OP_2MUL
OP_2DIV

OP_NEGATE
OP_ABS
OP_NOT

OP_0NOTEQUAL
OP_ADD
OP_SUB

OP_MUL
OP_DIV
OP_MOD

OP_LSHIFT
OP_RSHIFT

OP_BOOLAND
OP_BOOLOR
OP_NUMEQUAL

OP_NUMEQUALVERIFY

OP_NUMNOTEQUAL

OP_LESSTHAN
OP_GREATERTHAN
OP_LESSTHANOREQUAL

OP_GREATERTHANOREQUAL

OP_MIN

OP_MAX
OP_WITHIN

Operazioni crittografiche e di hashing

mostra gli operatori per le funzioni criptografiche.

Tabella 36. Operazioni crittografich

Simbolo	**V (I**
OP_RIPEMD160	0
OP_SHA1	0
OP_SHA256	0
OP_HASH160	0

OP_HASH256	0
OP_CODESEPARATOR	0
OP_CHECKSIG	0
OP_CHECKSIGVERIFY	0

OP_CHECKMULTISIG	0
OP_CHECKMULTISIGVERIFY	0

Nonoperators mostra simboli nonoperator.

Tabella 37. Nonoperators

Simbolo	Valore (hex)	Descrizione
OP_NOP1-OP_NOP10	0xb0-0xb9	Non fa niente, ignorato

OP code riservati per uso interno dal parser mostra codici operatori riservati per l'uso attraverso lo script parser interno.

Tabella 38. OP code riservati per dal parser

Simbolo	Valore (hex)	
OP_SMALLDATA	0xf9	
OP_SMALLINTEGER	0xfa	
OP_PUBKEYS	0xfb	
OP_PUBKEYHASH	0xfd	

OP_PUBKEY	0xfe	
OP_INVALIDOPCODE	0xff	

Appendix B: Bitcoin Improvement Proposals

I Bitcoin improvement proposals (Proposte per il miglioramento di Bitcoin) sono documenti tecnici che forniscono informazioni alla comunità bitcoin sui miglioramenti, o descrivono una nuova funzionalità per Bitcoin o per i suoi processi o per l'ecosistema.

Seguendo gli *Scopi e linee guida* dei BIP contenute nel BIP-01, ci sono tre tipi di BIP:

***Standard* BIP**

Descrive ogni cambiamento che interessa la maggior parte o tutte le implementazioni bitcoin, come un cambio nel protocollo di rete, un cambio nelle regole di validità delle transazioni, o ogni tipo di cambiamento o aggiunta che interessa l'interoperabilità delle applicazioni che usano bitcoin.

***Informational* BIP**

(BIP informativo) Descrive un problema di design di bitcoin, o fornisce linee guida generali o informazioni alla comunità bitcoin, ma non propone una nuova funzionalità. Gli Informational BIP non rappresentano necessariamente un consenso della comunità bitcoin o

un consiglio, in questo modo gli utenti e gli implementatori possono ignorare i BIP informativi o seguire il loro consiglio.

***Process* BIP**

(BIP di processo) Descrive un processo bitcoin, o propone un cambiamento a (o un evento in) un processo. I Process BIP sono come i BIP standard ma si applicano su aree diverse dal protocollo bitcoin stesso. Questi possono proporre un'implementazione, ma non al codice bitcoin; essi richiedono generalmente un consenso della community; e diversamente dagli informational BIP, sono più di semplici consigli, e gli utenti sono

tipicamente consigliati a non ignorarli. Esempi includono procedure, linee guida, cambiamenti al processo di presa delle decisioni, e cambiamenti agli strumenti o alle situazioni usate nello sviluppo di Bitcoin. Ogni meta-BIP è inoltre considerato un BIP di processo.

Le Bitcoin improvement proposals sono registrate in un repository versionato su GitHub: https://github.com/bitcoin/bips. Stato dei BIP mostra una "fotografia" dei BIP ad Aprile 2017. Consulta il repository autoritativo per informazioni aggiornate sui BIP esistenti e i loro contenuti.

Tabella 39. Stato dei BIP

BIP#	Titolo
BIP-1	BIP Purpose and Guidelines
BIP-2	BIP process, revised
BIP-8	Version bits with guarar lock-in
BIP-9	Version bits with timeout delay

BIP-10	Multi-Sig Transa Distribution
BIP-11	M-of-N Standard Transactio
BIP-12	OP_EVAL
BIP-13	Address Format for pay-to-s hash
BIP-14	Protocol Version and User A
BIP-15	Aliases
BIP-	Pay to Script Hash

16	
BIP-17	OP_CHECKHASHVERIFY (CHV)
BIP-18	hashScriptCheck
BIP-19	M-of-N Standard Transac (Low SigOp)
BIP-20	URI Scheme
BIP-21	URI Scheme
BIP-	getblocktemplate - Fundamer

22	
BIP-23	getblocktemplate - Pooled M
BIP-30	Duplicate transactions
BIP-31	Pong message
BIP-32	Hierarchical Determi Wallets
BIP-33	Stratized Nodes
BIP-34	Block v2, Height in Coinbase

BIP-35	mempool message
BIP-36	Custom Services
BIP-37	Connection Bloom filtering
BIP-38	Passphrase-protected private
BIP-39	Mnemonic code for gener deterministic keys

BIP-40	Stratum wire protocol
BIP-41	Stratum mining protocol
BIP-42	A finite monetary supply Bitcoin
BIP-43	Purpose Field for Determi Wallets
BIP-	Multi-Account Hierarchy

44	Deterministic Wallets
BIP-45	Structure for Deterministic I Multisignature Wallets
BIP-47	Reusable Payment Codes Hierarchical Determi Wallets
BIP-49	Derivation scheme for P2WI nested-in-P2SH based accou
BIP-50	March 2013 Chain Fork Mortem

BIP-60	Fixed Length "version" Mes (Relay-Transactions Field)
BIP-61	Reject P2P message
BIP-62	Dealing with malleability
BIP-63	Stealth Addresses
BIP-64	getutxo message
BIP-65	OP_CHECKLOCKTIMEVE

BIP-66	Strict DER signatures
BIP-67	Deterministic Pay-to-script multi-signature addresses th public key sorting
BIP-68	Relative lock-time consensus-enforced sequ numbers
BIP-69	Lexicographical Indexing Transaction Inputs and Outpu

BIP-70	Payment Protocol
BIP-71	Payment Protocol MIME type
BIP-72	bitcoin: uri extensions Payment Protocol
BIP-73	Use "Accept" header for resp type negotiation with Pay Request URLs
BIP-74	Allow zero value OP_RET in Payment Protocol
BIP-75	Out of Band Address Exch using Payment Pro Encryption

BIP-80	Hierarchy for Non-Co Voting Pool Determin Multisig Wallets
BIP-81	Hierarchy for Colored V Pool Deterministic Mu Wallets
BIP-83	Dynamic Hierarc Deterministic Key Trees
BIP-90	Buried Deployments

BIP-99	Motivation and deploymer consensus rule cha ([soft/hard]forks)
BIP-101	Increase maximum block size
BIP-102	Block size increase to 2MB
BIP-103	Block size follo technological growth
BIP-104	'Block75' - Max block size difficulty
BIP-105	Consensus based block retargeting algorithm
BIP-	Dynamically Controlled Bi

106	Block Size Max Cap
BIP-107	Dynamic limit on the block si
BIP-109	Two million byte size limit sigop and sighash limits
BIP-111	NODE_BLOOM service bit
BIP-112	CHECKSEQUENCEVERIFY
BIP-113	Median time-past as endpoi lock-time calculations

BIP-114	Merkelized Abstract Syntax T
BIP-120	Proof of Payment
BIP-121	Proof of Payment URI schem
BIP-122	URI scheme for Block references / exploration
BIP-123	BIP Classification
BIP-124	Hierarchical Deterministic S Templates

BIP-125	Opt-in Full Replace-by Signaling
BIP-126	Best Practices for Heteroger Input Script Transactions
BIP-130	sendheaders message
BIP-131	"Coalescing Transac Specification (wildcard inpu
BIP-132	Committee-based Acceptance Process
BIP-	feefilter message

133	
BIP-134	Flexible Transactions
BIP-140	Normalized TXID
BIP-141	Segregated Witness (Conse layer)
BIP-142	Address Format for Segre Witness
BIP-143	Transaction Sign Verification for Version

	Witness Program
BIP-144	Segregated Witness (Services)
BIP-145	getblocktemplate Updates Segregated Witness
BIP-146	Dealing with signature enc malleability
BIP-147	Dealing with dummy element malleability
BIP-148	Mandatory activation of se deployment

BIP-150	Peer Authentication
BIP-151	Peer-to-Peer Communic Encryption
BIP-152	Compact Block Relay
BIP-171	Currency/exchange information API
BIP-180	Block size/weight fraud proo
BIP-199	Hashed Time-Locked Cor transactions

Appendix C: Bitcore

Bitcore è una suite di strumenti forniti da BitPay. Il suo obiettivo è fornire strumenti facili da usare per gli sviluppatori di Bitcoin. Quasi tutto il codice di Bitcore è scritto in JavaScript. Ci sono alcuni moduli scritti appositamente per NodeJS. Infine, il modulo "nodo" di Bitcore include il codice C ++ di Bitcoin Core. Si prega di consultare https://bitcore.io per ulteriori informazioni.

Bitcore's Feature List

- Bitcoin full node (bitcore-node)
- Block explorer (insight)

- Block, transaction, and wallet utilities (bitcore-lib)
- Comunicazione diretta con la rete P2P di Bitcoin (bitcore-p2p)
- Generazione mnemonica di entropia del seme (bitcore-mnemonico)
- Protocollo di pagamento (bitcore-payment-protocol)
- Verifica e firma dei messaggi (bitcore-message)
- Schema di crittografia integrata a curva ellittica (bitcore-ecies)
- Wallet service (bitcore-wallet-service)
- Wallet client (bitcore-wallet-client)
- Playground (bitcore-playground)
- Integrazione di servizi direttamente

con Bitcoin Core (bitcore-node)

Esempi di libreria Bitcore

Prerequisiti

- NodeJS >= 4.x or use our hosted online playground

Se si utilizza NodeJS e node REPL :

```
$ npm install -g bitcore-lib bitcore-p2p
$ NODE_PATH=$(npm list -g | head -1)/node_modules node
```

Esempi di wallet che utilizzano bitcore-lib

Creazione di un nuovo indirizzo bitcoin con chiave privata associata:

```
> bitcore = require('bitcore-lib')
> privateKey = new bitcore.PrivateKey()
> address = privateKey.toAddress().toString()
```

Creare una private key ed indirizzo gerarchico deterministico:

```
> hdPrivateKey = bitcore.HDPrivateKey()
> hdPublicKey = bitcore.HDPublicKey(hdPrivateKey)
> hdAddress = new bitcore.Address(hdPublicKey.publicKey).toS
```

Creazione e firma di una transazione da un UTXO:

```
> utxo = {
txId: id della transazione contenente un output non speso,
outputIndex: output indexi, ad es. 0,
indirizzo: addressOfUtxo,
 script:
```

```
bitcore.Script.buildPublicKeyHashOut(addre
satoshis: importo inviato all'indirizzo
}
> fee = 3000 // impostato in modo
appropriato per le condizioni sulla rete
> tx = new bitcore.Transaction()
.from(utxo)
.to(address, 35000)
.fee(fee)
.enableRBF()
.sign(privateKeyOfUtxo)
```

Sostituisci l'ultima transazione nel mempool (replace-by-fee):

```
> rbfTx = new Transaction()
.from(utxo)
.to(address, 35000)
.fee(fee*2)
.enableRBF()
.sign(privateKeyOfUtxo);
> tx.serialize();
```

```
> rbfTx.serialize();
```

Trasmissione di una transazione alla rete Bitcoin (nota: trasmetti solo transazioni valide, fai riferimento a https://bitnodes.21.co/nodes [] per host peer):

1. Copia il codice seguente in un file chiamato *broadcast.js*.
2. Le variabili + tx + e + rbfTx + sono l'output di + tx.serialize () + e + rbfTx.serialize () +, rispettivamente.
3. Al fine di sostituire a pagamento, il peer deve supportare l'opzione bitcoind + mempoolreplace + e impostarlo su 1.
4. Eseguire il nodo file _broadcast.js

_ :

```
var p2p = require('bitcore-p2p');
var bitcore = require('bitcore-lib');
var tx = new bitcore.Transaction('output
from serialize function');
var rbfTx = new
bitcore.Transaction('output from serialize
function');
var host = 'ip address'; //use valid peer
listening on tcp 8333
var peer = new p2p.Peer({host: host});
var messages = new p2p.Messages();
peer.on('ready', function() {
  var txs = [messages.Transaction(tx),
messages.Transaction(rbfTx)];
var index = 0;
var interval = setInterval(function() {
peer.sendMessage(txs[index++]);
console.log('tx: ' + index + ' sent');
if (index === txs.length) {
clearInterval(interval);
```

```
console.log('disconnecting from peer: ' +
host);
peer.disconnect();
    }
}, 2000);
});
peer.connect();
```

Appendix D: pycoin, ku, e tx

La libreria Python pycoin, originariamente scritto e gestito da Richard Kiss, è una libreria basata su Python che supporta la manipolazione di chiavi e transazioni bitcoin, supportando anche il linguaggio di scripting a sufficienza per gestire correttamente transazioni non standard.

La pycoin library supporta sia Python 2 (2.7.x) che Python 3 (3.3 e seguenti) e viene fornito con alcune comode utilities command-line, ku and tx.

Key Utility (KU)

La command-line utility ku ("key utility") è come un attrezzo multiuso per manipolare le chiavi. Suporta le chiavi BIP-32, WIF, ed indirizzi (bitcoin and alt coins). In seguito ci sono alcuni esempi.

Creare una chiave BIP-32 usando una sorgente di entropia di default di GPG e */dev/random*:

```
$ ku create

input          : create
network        : Bitcoin
wallet key     :
xprv9s21ZrQH143K3LU5ctPZTBnb9kTjA5S

m2nhnzdvxJf5KJo9vjP2nABX65c5sFsWsV8
public version :
xpub661MyMwAqRbcFpYYiuvZpKjKhnJDZ
```

DGcpfT56AMFeo8M8KPkFMfLUtvwjwb6W
tree depth : 0
fingerprint : 9d9c6092
parent f'print : 00000000
child index : 0
chain code :
80574fb260edaa4905bc86c9a47d30c697c5
private key : yes
secret exponent :
112471538590155650688604752840386
hex :
f8a8a28b28a916e1043cc0aca52033a18a13c
wif :
L5Z54xi6qJusQT42JHA44mfPVZGjyb4XBF
uncompressed :
5KhoEavGNNH4GHKoy2Ptu4KfdNp4r56L5
public pair x :
764606382405464783648433974782784 6
public pair y :
59807879657469774102040120298272207

```
 x come hex      :
a90b300879243206 0fa04365941e09a8e4ad
 y come hex      :
843a0f6ed9c0eb1962c7453379540691 4fe3
y parity        : dispari
key pair as sec :
03a90b300879243206 0fa04365941e09a8e4
 uncompressed   :
04a90b300879243206 0fa04365941e09a8e4

843a0f6ed9c0eb1962c7453379540691 4fe3
hash160         :
9d9c609247174ae323acfc96c852753fe3c8
 uncompressed   :
8870d869800c9b91ce1eb460f4c60540f87c
Indirizzo Bitcoin :
1FNNRQ5fSv1wBi5gyfVBs2rkNheMGt86sp
 uncompressed   :
1DSS5isnH4FsVaLVjeVXewVSpfqktdiQAM
```

Creare una chiave BIP-32 da una

passphrase:

ATTENZIONE	La passphrase in questo esempio è troppo semplice da indovinare.

```
$ ku P:foo

input          : P:foo
network        : Bitcoin
wallet key     :
xprv9s21ZrQH143K31AgNK5pyVvW23gHn

ZoY5eSJMJ2Vbyvi2hbmQnCuHBujZ2WXG
public version :
xpub661MyMwAqRbcFVF9ULcqLdsEa5Wn
```

VYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu
tree depth : 0
fingerprint : 5d353a2e
parent f'print : 00000000
child index : 0
chain code :
5eeb1023fd6dd1ae52a005ce0e73420821e1
private key : yes
esponente segreto (secret exponent) :
6582573054709730571605716043797079
hex :
91880b0e3017ba586b735fe7d04f1790f3c4
wif :
L26c3H6jEPVSqAr1usXUp9qtQJw6NHgAp
uncompressed :
5JvNzA5vXDoKYJdw8SwwLHxUxaWvn9mI
public pair x :
8182198271938110406177734926913041
public pair y :
5899421806960542427832070325068978
x in hex :

```
b4e599dfa44555a4ed38bcfff0071d5af676a8
 y in hex      :
826d8b4d3010aea16ff4c1c1d3ae68541d9a0
parità y       : pari
key pair come sec :
02b4e599dfa44555a4ed38bcfff0071d5af67e
 non-compressa   :
04b4e599dfa44555a4ed38bcfff0071d5af67e

826d8b4d3010aea16ff4c1c1d3ae68541d9a0
hash160        :
5d353a2ecdb262477172852d57a3f11de0c1
 uncompressed   :
e5bd3a7e6cb62b4c820e51200fb1c148d79e
Indirizzo Bitcoin :
19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii
 uncompressed   :
1MwkRkogzBRMehBntgcq2aJhXCXStJTXH
```

Ottieni info come JSON:

```
$ ku P:foo -P -j
```

```
{
  "y_parity": "even",
  "public_pair_y_hex":
"826d8b4d3010aea16ff4c1c1d3ae68541d9a
  "private_key": "no",
  "parent_fingerprint": "00000000",
  "tree_depth": "0",
  "network": "Bitcoin",
  "btc_address_uncompressed":
"1MwkRkogzBRMehBntgcq2aJhXCXStJTXH
  "key_pair_as_sec_uncompressed":
"04b4e599dfa44555a4ed38bcfff0071d5af67
  "public_pair_x_hex":
"b4e599dfa44555a4ed38bcfff0071d5af676a
  "wallet_key":
"xpub661MyMwAqRbcFVF9ULcqLdsEa5Wr
  "chain_code":
"5eeb1023fd6dd1ae52a005ce0e73420821e1
  "child_index": "0",
  "hash160_uncompressed":
"e5bd3a7e6cb62b4c820e51200fb1c148d79e
```

```
  "btc_address":
"19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAi
  "fingerprint": "5d353a2e",
  "hash160":
"5d353a2ecdb2624771728 52d57a3f11de0c
  "input": "P:foo",
  "public_pair_x":
"818219827193811040617734926913041
  "public_pair_y":
"589942180696054242783207032506897
  "key_pair_as_sec":
"02b4e599dfa44555a4ed38bcfff0071d5af67
}
```

Chiave pubblica BIP32:

```
$ ku -w -P P:foo
xpub661MyMwAqRbcFVF9ULcqLdsEa5Wn
```

Genera una sottochiave:

```
$ ku -w -s3/2 P:foo
xprv9wTErTSkjVyJa1v4cUTFMFkWMe5eu8
```

Sottochiave Hardened:

```
$ ku -w -s3/2H P:foo
xprv9wTErTSu5AWGkDeUPmqBcbZWX1xq
```

WIF:

```
$ ku -W P:foo
L26c3H6jEPVSqAr1usXUp9qtQJw6NHgAp
```

Indirizzo:

```
$ ku -a P:foo
19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii
```

Genera una serie di sottochiavi:

```
$ ku P:foo -s 0/0-5 -w
xprv9xWkBDfyBXmZjBG9EiXBpy67KK72f
xprv9xWkBDfyBXmZnzKf3bAGifK593gT7V
xprv9xWkBDfyBXmZqdXA8y4SWqfBdy71g
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv8
xprv9xWkBDfyBXmZv2q3N66hhZ8DAcEnQ
xprv9xWkBDfyBXmZw4jEYXUHYc9fT25k9
```

Genera gli indirizzi corrispondenti:

```
$ ku P:foo -s 0/0-5 -a
1MrjE78H1R1rqdFrmkjdHnPUdLCJALbv3x
1AnYyVEcuqeoVzH96zj1eYKwoWfwte2pxu
1GXr1kZfxE1FcK6ZRD5sqqqs5YfvuzA1Lb
116AXZc4bDVQrqmcinzu4aaPdrYqvuiBEK
1Cz2rTLjRM6pMnxPNrRKp9ZSvRtj5dDUM
1WstdwPnU6HEUPme1DQayN9nm6j7nDVI
```

Genera i WIF corrispondenti:

```
$ ku P:foo -s 0/0-5 -W
L5a4iE5k9gcJKGqX3FWmxzBYQc29PvZ6p
Kyjgne6GZwPGB6G6kJEhoPbmyjMP7D5d3
L4B3ygQxK6zH2NQGxLDee2H9v4Lvwg14
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxb
L2oD6vA4TUyqPF8QG4vhUFSgwCyuuvFZ3
KzChTbc3kZFxUSJ3Kt54cxsogeFAD9CCM4
```

Controlla che funzioni scegliendo una stringa BIP32 (quella corrispondente alla sottochiave 0/3):

```
$ ku -W
```

```
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv8
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxb
$ ku -a
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv8
116AXZc4bDVQrqmcinzu4aaPdrYqvuiBEK
```

Sì, sembra familiare.

Dall'esponente segreto:

```
$ ku 1

input          : 1
network        : Bitcoin
esponente segreto : 1
 esadecimale          : 1
wif            :
KwDiBf89QgGbjEhKnhXJuH7LrciVrZi3qYj
 non-compressa   :
5HpHagT65TZzG1PH3CSu63k8DbpvD8s5ip
coppia pubblica x   :
55066263022773436695787188951685 3
coppia pubblica y   :
```

```
3267051002075881697808308513050704
 x in esadecimale      :
79be667ef9dcbbac55a06295ce870b07029b
 y in esadecimale      :
483ada7726a3c4655da4fbfc0e1108a8fd17b
parità y       : pari
key pair come sec :
0279be667ef9dcbbac55a06295ce870b0702
 non-compressa   :
0479be667ef9dcbbac55a06295ce870b0702

483ada7726a3c4655da4fbfc0e1108a8fd17b
hash160        :
751e76e8199196d454941c45d1b3a323f14
 non-compresso   :
91b24bf9f5288532960ac687abb035127b1d
Indirizzo Bitcoin :
1BgGZ9tcN4rm9KBzDn7KprQz87SZ26SAM
 non-compressa   :
1EHNa6Q4Jz2uvNExL497mE43ikXhwF6kZ
```

Versione Litecoin:

```
$ ku -nL 1

input           : 1
network         : Litecoin
esponente segreto: 1
 hex            : 1
wif             :
T33ydQRKp4FCW5LCLLUB7deioUMoveiw
 non-compressa:
6u823ozcyt2rjPH8Z2ErsSXJB5PPQwK7VV
coppia pubblica x:
55066263022277343669578718895168534
coppia pubblica y    :
32670510020758816978083085130507041
 x in esadecimale       :
79be667ef9dcbbac55a06295ce870b07029bf
 y in esadecimale       :
483ada7726a3c4655da4fbfc0e1108a8fd17b4
parità y        : pari
key pair come sec  :
0279be667ef9dcbbac55a06295ce870b07029
```

```
 non-compresso   :
0479be667ef9dcbbac55a06295ce870b0702

483ada7726a3c4655da4fbfc0e1108a8fd17b
hash160         :
751e76e8199196d454941c45d1b3a323f14
 non-compressa   :
91b24bf9f5288532960ac687abb035127b1d
Indirizzo Litecoin :
LVuDpNCSSj6pQ7t9Pv6d6sUkLKoqDEVUn
 uncompressed   :
LYWKqJhtPeGyBAw7WC8R3F7ovxtzAiubd
```

Dogecoin WIF:

```
$ ku -nD -W 1
QNcdLVw8fHkixm6NNyN6nVwxKek4u7qri
```

Dalla coppia pubblica (sulla Testnet):

```
$ ku -nT
55066263022773436695787188951685
```

input :
5506626302277343669578718895168534
89116729240,pari
network : Bitcoin testnet
coppia pubblica x:
5506626302277343669578718895168534
coppia pubblica y:
3267051002075881697808308513050704
x in esadecimale:
79be667ef9dcbbac55a06295ce870b07029b
y in esadecimale:
483ada7726a3c4655da4fbfc0e1108a8fd17b
parità y: pari
coppia di chiavi come sec:
0279be667ef9dcbbac55a06295ce870b0702
non-compressa:
0479be667ef9dcbbac55a06295ce870b0702

483ada7726a3c4655da4fbfc0e1108a8fd17b
hash160:
751e76e8199196d454941c45d1b3a323f14

```
 non-compressa:
91b24bf9f5288532960ac687abb035127b1d
Indirizzo bitcoin testnet:
mrCDrCybB6J1vRfbwM5hemdJz73FwDBC8
 non-compressa:
mtoKs9V381UAhUia3d7Vb9GNak8Qvmcsm
```

Da hash160:

```
$ ku
751e76e8199196d454941c45d1b3a323f143


input:
751e76e8199196d454941c45d1b3a323f143
network       : Bitcoin
hash160       :
751e76e8199196d454941c45d1b3a323f143
Indirizzo Bitcoin :
1BgGZ9tcN4rm9KBzDn7KprQz87SZ26SAM
```

Come un indirizzo Dogecoin:

```
$ ku -nD
```

```
751e76e8199196d454941c45d1b3a323f14

input       :
751e76e8199196d454941c45d1b3a323f14
network     : Dogecoin
hash160     :
751e76e8199196d454941c45d1b3a323f14
Indirizzo Dogecoin :
DFpN6QqFfUm3gKNaxN6tNcab1FArL9cZL
```

Utilità di Transazione - Transaction Utility (TX)

L'Utility command-line tx visualizzerà la trudizione in forma leggibile, recuperando le transazioni di base dalla cache delle transazioni di pycoin o dai servizi Web (blockchain.info, blockcypher.com, blockr.io e chain.so sono attualmente supportati), unendo le

transazioni, aggiungendo o eliminando input o output e firmando le transazioni.

Di seguito puoi trovare altri esempi.

Vediamo la famosa transazione "pizza":

```
$ tx
49d2adb6e476fa46d8357babf78b1b501fd39
attenzione: considera impostare la
variabile d'ambiente
PYCOIN_CACHE_DIR=~/.pycoin_cache
per abilitare il caching di transazioni
ottenute da servizi web
warning: nessun service providers trovato
per get_tx; considera di settare la variabile
di ambiente PYCOIN_BTC_PROVIDERS
usage: tx [-h] [-t
TRANSACTION_VERSION] [-l
LOCK_TIME] [-n NETWORK] [-a]
```

```
        [-i address (indirizzo)] [-f path-to-
private-keys (percorso alle chiavi private)]
[-g GPG_ARGUMENT (argomento gpg)]
        [--remove-tx-in
tx_in_index_to_delete]
        [--remove-tx-out
tx_out_index_to_delete] [-F transaction-
fee] [-u]
        [-b BITCOIND_URL] [-o path-to-
output-file (path al file di output)]
        argomento [argument ...]
tx: error: can't find Tx with id (non posso
trovare una transazione con id)
49d2adb6e476fa46d8357babf78b1b501fd39
```

Oops! Non abbiamo web services impostati. Impostiamoli adesso:

```
$
PYCOIN_CACHE_DIR=~/.pycoin_cache
$ PYCOIN_BTC_PROVIDERS="block.io
blockchain.info blockexplorer.com"
```

```
$ export PYCOIN_CACHE_DIR
PYCOIN_BTC_PROVIDERS
```

Non è effettuato automaticamente quindi uno strumento da riga di comando non trasmetterà informazioni potenzialmente private riguardo a quale transazione sei interessato a un sito di terza parte. Se questo non ti interessa, potresti mettere queste linee nel tuo *.profilo*.

Proviamo di nuovo:

```
$ tx
49d2adb6e476fa46d8357babf78b1b501fd39
Version: 1 tx hash
49d2adb6e476fa46d8357babf78b1b501fd39
159 bytes
TxIn count: 1; TxOut count: 1
Lock time: 0 (valid anytime)
Input:
```

```
 0:                    (sconosciuta) da
1e133f7de73ac7d074e2746a3d6717dfc99e
Output:
 0:
1CZDM6oTttND6WPdt3D6bydo7DYKzd9Q
receives 10000000.00000 mBTC
Output totale 10000000.00000 mBTC
includndo l'hex dump degli unspent visto
che la transazione non è firmata
completamente
0100000001141045e0ab2b0b82cdefaf9e9a8

** impossibile validare la transazione
perché la fonte delle transazioni è
attualmente non disponibile
```

L'ultima linea appare perché per validare le firme delle transazioni, ti serve tecnicamente la fonte delle transazioni. Quindi aggiungiamo -a per aggiungere alla transazione

informazioni riguardo la fonte:

```
$ tx -a
49d2adb6e476fa46d8357babf78b1b501fd39
attenzione: i suggerimenti riguardo le fee
di transazione calcolate e stimate
casualmente potrebbero essere non
corretti
attenzione: le fee di transazione sono più
basse del (calcolato casualmente) valore
aspettato di 0.1 mBTC, la transazione
potrebbe non propagarsi
Version:  1  tx hash
49d2adb6e476fa46d8357babf78b1b501fd39
159 bytes
TxIn count: 1; TxOut count: 1
Lock time: 0 (valid anytime)
Input:
  0:
17WFx2GQZUmh6Up2NDNCEDk3deYomd
from
1e133f7de73ac7d074e2746a3d6717dfc99e
```

```
10000000.00000 mBTC  sig ok
Output:
  0:
1CZDM6oTttND6WPdt3D6bydo7DYKzd9Q
receives 10000000.00000 mBTC
Input totali  10000000.00000 mBTC
Output totale 10000000.00000 mBTC
Commissioni totali 0.00000 mBTC

0100000001141045e0ab2b0b82cdefaf9e9a8c

tutti i valori delle transazioni in entrata
validati
```

Ora, guardiamo un output non spese per uno specifico indirizzo (UTXO). Nel blocco #1, noi vediamo una transazione coinbase a 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzj Usiamo fetch_unspent per trovare tutte le monete in questo indirizzo:

```
$ fetch_unspent
12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX
a3a6f902a51a2cbebede144e48a88c05e608c
cea36d008badf5c7866894b191d3239de958
065ef6b1463f552f675622a5d1fd2c08d632
a66dddd42f9f2491d3c336ce5527d45cc5c2
ffd901679de65d4398de90cefe68d2c3ef073
d658ab87cc053b8dbcfd4aa2717fd23cc3edf
36ebe0ca3237002acb12e1474a3859bde0ac
fd87f9adebb17f4ebb1673da76ff48ad29e64b
dfdf0b375a987f17056e5e919ee6eadd87dad
cb2679bfd0a557b2dc0d8a616822f3fcbe28
d6be34ccf6edddc3cf69842dce99fe503bf63

0e3e2357e806b6cdb1f70b54c3a3a17b6714
```

Appendix E: Comandi Bitcoin Explorer (bx)

Bitcoin Explorer (bx) è un tool command-line che offre una varietà di comandi per manipolare le chiavi e la costruzione delle transazioni. È parte di libbitcoin bitcoin library.

```
Utilizzo: bx COMMAND [--help]

Info: I comandi bx sono:

address-decode
address-embed
address-encode
address-validate
```

base16-decode
base16-encode
base58-decode
base58-encode
base58check-decode
base58check-encode
base64-decode
base64-encode
bitcoin160
bitcoin256
btc-to-satoshi
ec-add
ec-add-secrets
ec-multiply
ec-multiply-secrets
ec-new
ec-to-address
ec-to-public
ec-to-wif
fetch-balance
fetch-header

fetch-height
fetch-history
fetch-stealth
fetch-tx
fetch-tx-index
hd-new
hd-private
hd-public
hd-to-address
hd-to-ec
hd-to-public
hd-to-wif
help
input-set
input-sign
input-validate
message-sign
message-validate
mnemonic-decode
mnemonic-encode
ripemd160

satoshi-to-btc
script-decode
script-encode
script-to-address
seed
send-tx
send-tx-node
send-tx-p2p
settings
sha160
sha256
sha512
stealth-decode
stealth-encode
stealth-public
stealth-secret
stealth-shared
tx-decode
tx-encode
uri-decode
uri-encode

```
validate-tx
watch-address
wif-to-ec
wif-to-public
wrap-decode
wrap-encode
```

Per maggiori informazioni, vedere Bitcoin Explorer homepage and Bitcoin Explorer user documentation.

Esempi di bx Command Use

Guardiamo alcuni esempi d'uso di Bitcoin Explorer commands per familiarizzare con chiavi ed indirizzi.

Generare a "seed" casuale usando il comando seed, che usa il generatore di numeri casuali del sistema operativo.

Passiamo il seed al comando ec-new per generare una nuova chiave privata. Salviamo l'output standard entro il file *private_key*:

```
$ bx seed | bx ec-new > private_key
$ cat private_key
73096ed11ab9f1db6135857958ece7d73ea7
```

Ora, generiamo la chiave pubblica dalla chiave privata, usando il comando ec-to-public. Passiamo il file *private_key* file nell'input standard e salviamo l'output standard del comando in un nuovo file *public_key*:

```
$ bx ec-to-public < private_key >
public_key
$ cat public_key
02fca46a6006a62dfdd2dbb2149359d0d97a0
```

Possiamo riformattare la public_key

come un indirizzo, usando il comando ec-to-address. Passiamo la *public_key* entro lo standard input:

```
$ bx ec-to-address < public_key
17re1S4Q8ZHyCP8Kw7xQad1Lr6XUzWUn
```

Le chiavi generate in questo mondo producono un wallet non deterministico type-0 Questo significa che ogni chiave è generata da un seed indipendente. I comandi di Bitcoin Explorer possono anche generale chiavi deterministicamente, in accordo con BIP-32. In questo caso, è creata una "master" key da un seed e quindi estesa deterministicamente per produrre un albero di sottochiavi, risultando un wallet deterministico type-2.

Primo, usiamo il comando seed and hd-new per generare una master key che sarà usata come base per derivare una gerarchia di chiavi:

```
$ bx seed > seed
$ cat seed
eb68ee9f3df6bd4441a9feadec179ff1

$ bx hd-new < seed > master
$ cat master
xprv9s21ZrQH143K2BEhMYpNQoUvAgiEj
```

Ora usiamo il comando hd-private per generare a "account" key consolidato e una sequenza di due chiavi private entro l'account:

```
$ bx hd-private --hard < master > account
$ cat account
xprv9vkDLt81dTKjwHB8fsVB5QK8cGnzve
```

```
$ bx hd-private --index 0 < account
xprv9xHfb6w1vX9xgZyPNXVgAhPxSsEkeR

$ bx hd-private --index 1 < account
xprv9xHfb6w1vX9xjc8XbN4GN86jzNAZ6x
```

Quindi, usiamo il comando hd-public per generare la corrispondente sequenza di due chiavi pubbliche:

```
$ bx hd-public --index 0 < account
xpub6BH1zcTuktiFu43rUZ2gXqLgzu5F3tLE

$ bx hd-public --index 1 < account
xpub6BH1zcTuktiFx6CzhPbGjG3UYQ13WR
```

Le chiavi pubbliche possono anche essere derivate dalle loro corrispondenti chiavi private usando il comando hd-to-public :

```
$ bx hd-private --index 0 < account | bx hd-
to-public
```

```
xpub6BH1zcTuktiFu43rUZ2gXqLgzu5F3tLE

$ bx hd-private --index 1 < account | bx hd-
to-public
xpub6BH1zcTuktiFx6CzhPbGjG3UYQ13WR
```

Possiamo generare un numero di chiavi praticamente senza limiti in una catena deterministica, tutte derivate da un singolo seed. Questa tecnica è usata in molte applicazioni wallet per generare chiavi che possono essere archiviate (backup) e ripristinate con un singolo valore seed. E' molto piu facile che dover effettuare il backup del wallet con tutte le chiavi generate randomicamente ogni volta che si crea una nuova chiave.

Il seed può essere codificato usando il

comando mnemonic-encode :

```
$ bx hd-mnemonic < seed > words
adore repeat vision worst especially veil
inch woman cast recall dwell appreciate
```

Il seed può essere decodificato usando il comando mnemonic-dencode :

```
$ bx mnemonic-decode < words
eb68ee9f3df6bd4441a9feadec179ff1
```

La codifica mnemonica può rendere il seed più facile da registrare e da ricordare.

1. "Bitcoin: A Peer-to-Peer Electronic Cash System," Satoshi Nakamoto (https://bitcoin.org/bitcoin.pdf).