Mastering Lightning Network

Protocollo di Secondo Livello per Pagamenti Bitcoin Istantanei

Andreas M. Antonopoulos,
Olaoluwa Osuntokun e René Pickhardt

Traduzione e pubblicazione a cura di Riccardo Masutti

Mastering Lightning Network - Protocollo di Secondo Livello per Pagamenti Bitcoin Istantanei

Prima Edizione – Pubblicata il 25 Febbraio 2023

Versione: PDF

Basato e tradotto dalla 1st edition, 1st print di Mastering the Lightning Network pubblicata da aantonop Books LLC, René Pickhardt, and uuddlrlrbas LLC.

Dedica

La traduzione italiana di questo libro è dedicata a William Rees-Mogg (1928 – 2012), giornalista britannico, redattore del Times dal 1967 al 1981 e co-autore del libro "The Sovereign Individual". Ha previsto lo sviluppo del XXI secolo; concentrandosi sull'ascesa di Internet e del cyberspazio, delle valute ed economie digitali, della proprietà individuale e decentramento da parte dello Stato.

"Le transizioni non sono mai popolari, perché rendono obsoleto il capitale intellettuale faticosamente acquisito e confondono gli imperativi morali stabiliti".

Traduzione Italiana

Mastering Lightning Network – Protocollo di Secondo Livello per Pagamenti Bitcoin Istantanei di Riccardo Masutti, viene rilasciato in una versione cartacea e una versione digitale. Il contenuto è lo stesso nelle due versioni, ma ci sono alcune differenze importanti.

- La versione cartacea è disponibile su Amazon e, a richiesta, nelle varie librerie online e locali. La versione cartacea è a sua volta suddivisa in due formati diversi.
    - Una versione con copertina a colori flessibile e testo su carta leggera con stampante a getto d'inchiostro bianco e nero
    - Una versione con copertina a colori rigida e testo su carta più spessa con stampante a toner a colori e immagini di nitidezza e qualità altissima
- La versione digitale è disponibile su Amazon (formati ePub, Kindle) e in formato PDF. Tutte queste modalità sono prive di DRM.

“Il consiglio che voglio darti è quello di leggere la versione cartacea di questo libro.

Spero che tutte le persone che possono permettersi di acquistare il libro scelgano la versione cartacea rispetto alla versione digitale, poichè l'esperienza di lettura è molto più immersiva, ma anche per sostenere finanziariamente tutto il lavoro che c’è stato dietro questo libro.

Se preferisci comunque la versione digitale, spero tu possa optare per l’acquisto del formato ePub/Kindle su Amazon, anziché usufruire di una versione PDF gratuita nel web. Se sei membro Amazon Prime, puoi anche leggerlo gratuitamente in tali formati e comunque sostenere il lavoro di traduzione e pubblicazione.

Se stai leggendo questo libro gratuitamente sul web in formato PDF, non è un problema. Quando le persone hanno l'opportunità di sfogliarne il contenuto prima di acquistare il libro, possono decidere meglio se questo libro fa per loro o meno. A prescindere che tu stia leggendo una versione acquistata o una versione libera sul web, voglio che tutti abbiano accesso a informazioni ed educazione di prim'ordine circa Bitcoin. Ciò rafforzerà Bitcoin come sistema e lo renderà sempre più resiliente.

Se stai leggendo questo libro gratuitamente, considera l’idea di lasciare una recensione sul libro, ad esempio su Amazon (non serve acquistare il libro per lasciare una recensione) o sui

tuoi social preferiti. Se sei un creatore di contenuti, considera l'idea di creare un post o un video a riguardo. Questo permetterà al libro di crescere nelle classifiche, avere visibilità ed aiutare sempre più persone a comprendere la rivoluzione che tutti noi conosciamo: Bitcoin."

- Riccardo Masutti

Prima Edizione – Pubblicata il 25 Febbraio 2023 in occasione del 5° anniversario del Lightning Pizza Day(*).

Ultimo aggiornamento: 20 Marzo 2023

Contattare Riccardo

Puoi contattare l'autore della versione italiana su https://riccardomasutti.com

Chiave PGP: 9E18 3EAD 5CCF 7D0D 88E5 BC47 CA0E 200F AE9F 2AA0

Se non hai già perso tutti i tuoi amati BTC in un terribile incidente nautico e stai leggendo questo libro gratuitamente, puoi anche effettuare una donazione e supportare l'autore della versione italiana inviando la quantità di sats che desideri a questo indirizzo:

https://dona.riccardomasutti.com

Puoi effettuare una donazione in BTC (on-chain, via PayJoin, via Lightning Network o via Liquid Network) oppure in qualsiasi altra criptovaluta e stablecoin.

(*) Il 25 Febbraio 2018, otto anni dopo aver acquistato la pizza per 10mila BTC, andando ad effettuare il primo acquisto di beni fisici con bitcoin nel mondo reale, Laszlo Hanyecz torna a comprare la pizza tramite Lightning Network, andando a dimostrare all'intero mondo la rivoluzione che porta in ottica di pagamenti istantanei, privati e con basse commissioni.

Indice

Prefazione

Lightning Network (LN) è una rete peer-to-peer di secondo livello che ci consente di effettuare pagamenti Bitcoin "off-chain", ovvero senza dover generare, inoltrare e scrivere transazioni sulla blockchain di Bitcoin.

Lightning Network ci permette di effettuare pagamenti Bitcoin sicuri, economici, veloci e molto più privati, anche per somme molto basse.

Basandosi sull'idea dei canali di pagamento proposta per la prima volta dall'inventore di Bitcoin, Satoshi Nakamoto, Lightning Network è una rete instradata di canali di pagamento in cui i pagamenti stessi "saltano" attraverso un percorso dal mittente al destinatario.

L'idea iniziale di Lightning Network è stata proposta nel 2015 nel documento rivoluzionario "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments", di Joseph Poon e Thaddeus Dryja. Nel 2017 già esisteva una rete Lightning "di prova", poiché diversi gruppi crearono implementazioni compatibili tra di loro e si coordinarono per definire alcuni standard di interoperabilità. Nel 2018, Lightning Network è diventato "attivo" e i pagamenti hanno iniziato a divenire via via sempre più frequenti.

Nel 2019, Andreas M. Antonopoulos, Olaoluwa Osuntokun e René Pickhardt hanno deciso di collaborare per scrivere questo libro. Nel 2022, il libro è stato pubblicato in lingua inglese e Riccardo Masutti ha iniziato a tradurne i contenuti e programmare la edizione cartacea e digitale in lingua italiana. A Febbraio 2023 il libro è stato pubblicato in italiano.

Pubblico Previsto

Questo libro è destinato principalmente a lettori con interessi tecnici e con una comprensione dei fondamenti di Bitcoin.

Convenzioni Usate in Questo Libro

In questo libro vengono utilizzate le seguenti convenzioni tipografiche:

Corsivo

Indica nuovi termini, URL, indirizzi e-mail, nomi di file ed estensioni di file.

`Larghezza costante`

Utilizzato per elenchi di programmi, nonché all'interno dei paragrafi per fare riferimento a elementi di programma come nomi di variabili o funzioni, database, tipi di dati, variabili di ambiente, istruzioni e parole chiave.

`Larghezza costante grassetto`

Mostra i comandi o altro testo che dovrebbe essere digitato letteralmente dall'utente.

`Corsivo a larghezza costante`

Mostra il testo che deve essere sostituito con valori forniti dall'utente o con valori determinati dal contesto.

SUGGERIMENTO	Questo elemento indica un suggerimento da seguire.

NOTA	Questo elemento indica una nota generale.

ATTENZIONE	Questo elemento indica un avviso cui fare particolarmente attenzione.

Esempi di Codice

Gli esempi sono illustrati in Go, C++, Python e utilizzando la riga di comando di un sistema operativo simile a Unix. Tutti gli esempi di codice sono disponibili su GitHub nella sottocartella relativa al codice. Puoi forkare il codice del libro, testare gli esempi e/o inviare le correzioni tramite GitHub.

Tutti gli esempi di codice possono essere replicati sulla maggior parte dei sistemi operativi con un'installazione minima di compilatori, interpreti e librerie per i linguaggi corrispondenti. Ove necessario, verranno fornite istruzioni di installazione di base ed esempi passo-passo dell'output di tali istruzioni.

Alcuni esempi di codice e output di codice sono stati riformattati per la stampa. In tutti questi casi, le righe sono state divise dal carattere “barra rovesciata” (\ ), seguito da un carattere di nuova riga. Quando trascrivi gli esempi, rimuovi quei due caratteri e unisci di nuovo le righe. Così facendo dovresti vedere risultati identici a quelli mostrati nell'esempio.

Tutti gli esempi di codice utilizzano valori reali e calcoli, ove possibile, in modo da permetterti di testare e vedere gli stessi risultati per calcolare gli stessi valori. Ad esempio, le chiavi private e le chiavi pubbliche e gli indirizzi corrispondenti sono tutti reali.

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. 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 non 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 non richiede un'autorizzazione.

Apprezziamo un'attribuzione. Un’attribuzione include solitamente Il titolo, l’autore e l’ISBN. Per esempio: "Mastering Lightning Network - Protocollo di Secondo Livello per Pagamenti Bitcoin Istantanei di Riccardo Masutti, ISBN 9798351252315".

Riferimenti ad Aziende e Prodotti

Tutti i riferimenti ad aziende e prodotti sono destinati a scopi didattici, dimostrativi e di riferimento. Gli autori non approvano nessuna delle società o dei prodotti citati. Non abbiamo testato il funzionamento o la sicurezza di nessuno dei prodotti, progetti o segmenti di codice mostrati in questo libro. Usali a tuo rischio!

Indirizzi e Transazioni 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 navigare nella blockchain, guardare le transazioni offerte come esempi, recuperarle con i tuoi script o programmi, ecc.

Tuttavia, nota che le chiavi private utilizzate per generare gli indirizzi stampati in questo libro sono state "bruciate". Ciò significa che se invii denaro a uno di questi indirizzi, il denaro sarà perso per sempre o (più probabilmente) confiscato, poiché chiunque legga il libro può appropriarsene utilizzando le chiavi private qui stampate.

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

Contattare gli Autori del Libro

Contattare Andreas

Puoi contattare Andreas M. Antonopoulos sul suo sito personale: https://aantonop.com

Iscriviti al canale di Andreas su YouTube: https://www.youtube.com/aantonop

Mi piace alla pagina Facebook di Andreas:
https://www.facebook.com/AndreasMAntonopoulos

Segui Andreas su Twitter: https://twitter.com/aantonop

Connettiti con Andreas su LinkedIn: https://linkedin.com/company/aantonop

Andreas desidera anche ringraziare i sostenitori che supportano il suo lavoro attraverso donazioni mensili. Puoi supportare Andreas su Patreon su https://patreon.com/aantonop.

Contattare Renè

Puoi contattare René Pickhardt sul suo sito personale: https://ln.rene-pickhardt.de

Iscriviti al canale di René su YouTube: https://www.youtube.com/user/RenePickhardt

Segui René su Twitter: https://twitter.com/renepichardt

Connettiti con René su LinkedIn: https://www.linkedin.com/in/rene-pickhardt-80313744

René desidera anche ringraziare tutti i mecenati che sostengono il suo lavoro attraverso donazioni mensili. Puoi supportare René su Patreon su https://patreon.com/reneickhardt.

Oppure puoi supportare il suo lavoro direttamente con Bitcoin (anche tramite Lightning Network) su https://donate.ln.rene-pickhardt.de per il quale René è grato tanto quanto per i suoi patreon.

Contattare Olaoluwa Osuntokun

Puoi contattare Olaoluwa Osuntokun al suo indirizzo email professionale: laolu@lightning.engineering

Segui Olaoluwa su Twitter: https://twitter.com/roasbeef

Ringraziamenti degli Autori del Libro e del Traduttore ed Editore dell'Edizione Italiana

Ringraziamenti di Andreas

Devo il mio amore per le parole e per i libri a mia madre, Theresa, che mi ha cresciuto in una casa con i libri allineati su ogni parete. Mia madre comprò anche il mio primo computer nel 1982, nonostante mi definissi un tecnofobo. Mio padre, Menelaos, un ingegnere civile che ha pubblicato il suo primo libro a 80 anni, è stato colui che mi ha insegnato il pensiero logico e analitico e l'amore per la scienza e l'ingegneria.

Grazie a tutti per avermi supportato durante questo viaggio.

Ringraziamenti di René

Voglio ringraziare il sistema educativo tedesco attraverso il quale ho acquisito le conoscenze su cui si basa il mio lavoro. È uno dei regali più grandi che mi è stato fatto. Allo stesso modo voglio ringraziare il sistema sanitario pubblico tedesco e ogni persona che dedica il proprio tempo a lavorare in quel settore. Il loro sforzi e la loro resistenza fanno di loro i miei eroi

personali e non dimenticherò mai l'aiuto, la pazienza e il supporto che ho ricevuto quando ne avevo bisogno. Un ringraziamento va a tutti gli studenti cui mi è stato permesso di insegnare e che si sono impegnati in discussioni e domande interessanti. Da loro ho imparato di più. Sono anche grato alla comunità di Bitcoin e Lightning Network che mi ha accolto calorosamente, agli appassionati e a tutte le persone che hanno sostenuto finanziariamente e continuano a sostenere il mio lavoro. In particolare sono grato a tutti gli sviluppatori open source (non solo Bitcoin e Lightning Network) e alle persone che li finanziano per rendere possibile quella tecnologia. Un ringraziamento speciale va ai miei coautori per aver cavalcato con me attraverso la tempesta. Ultimo ma non meno importante, sono grato ai miei cari.

Ringraziamenti di Olaoluwa Osuntokun

Vorrei ringraziare lo straordinario team di Lightning Labs, poiché senza tutti loro non ci sarebbe LND. Vorrei anche ringraziare il gruppo originale di autori delle specifiche BOLT: Rusty Russell, Fabrice Drouin, Conner Fromnkchet, Pierre-Marie Padiou, Lisa Neigut e Christian Decker. Ultimo ma non meno importante, vorrei ringraziare Joseph Poon e Tadge Dryja, gli autori dell'articolo originale di Lightning Network, poiché senza di loro non ci sarebbe un Lightning Network di cui scrivere un libro.

Ringraziamenti di Riccardo Masutti

Ringrazio tutti i visionari, sognatori, appassionati, sviluppatori, educatori, designer, creatori di contenuti e studenti Bitcoin-only. Senza di voi Bitcoin non sarebbe dove si trova oggi. Voglio anche ringraziare tutte le persone che mi hanno supportato in tutti gli anni del mio lavoro sul mondo di Bitcoin. Ciò mi ha permesso di continuare a dedicare parte del mio tempo libero ad educare su questi temi, alla traduzione e pubblicazione di questo nuovo capitolo su Lightning Network e a lavorare su moltissimi progetti FOSS su Bitcoin.

Copertina della Traduzione Italiana

Uno speciale ringraziamento a Yegor Petrov, artista ed illustratore, che ha ideato e disegnato il "fulmine" di Lightning Network presente nella copertina di questo libro.

Contributi

Molte persone hanno commentato, corretto ed effettuato aggiunte al libro poiché è stato scritto in collaborazione su GitHub. In questa pagina è riportato un elenco in ordine alfabetico di tutti i contributori di GitHub, inclusi i loro ID GitHub tra parentesi:

https://github.com/lnbook/lnbook/blob/develop/github_contributors.asciidoc

Senza l'aiuto offerto da tutte le persone elencate, la pubblicazione di questo libro non sarebbe stata possibile. I vostri contributi dimostrano il potere dell'open source e della cultura aperta e saremo eternamente grati per il vostro aiuto.

Grazie.

Fonti

Parte del materiale in questo libro è stato tratto da una varietà di fonti di dominio pubblico, fonti con licenza aperta o con autorizzazione.

Parte 1: Comprendere Lightning Network

Una panoramica di Lightning Network adatta a chiunque sia interessato a comprenderne i concetti base e l'utilizzo.

Capitolo 1. Introduzione

Benvenuto in Mastering *Lightning Network*!

Lightning Network (spesso abbreviato in LN), sta cambiando il modo in cui le persone scambiano valore online ed è uno dei progressi più entusiasmanti avvenuti nella storia di Bitcoin. Oggi, nel 2023, Lightning Network è ancora agli inizi. Lightning Network è un protocollo e tecnologia di secondo livello per utilizzare Bitcoin in modo intelligente.

Il concetto di Lightning Network è stato proposto nel 2015 e la prima implementazione è stata lanciata nel 2018. Oggi stiamo solamente iniziando a vedere le opportunità che Lightning Network offre a Bitcoin, incluse migliorie a privacy, velocità e scalabilità. Conoscendo le basi di Lightning Network, potrai contribuire a plasmare il futuro della rete creando allo stesso tempo opportunità economiche e professionali per te stesso.

Partiamo dal presupposto che tu abbia già alcune conoscenze di base su Bitcoin, ma in caso contrario, non preoccuparti: spiegheremo i concetti Bitcoin più importanti, quelli che devi conoscere per comprendere Lightning Network nell'Appendice A, in fondo a questo libro. Se vuoi saperne di più su Bitcoin, puoi leggere "Mastering Bitcoin: Traduzione italiana della guida completa al mondo di bitcoin e della blockchain"[1] di Riccardo Masutti, disponibile in formato cartaceo su Amazon e gratuitamente in formato digitale online.

Sebbene la maggior parte di questo libro sia scritta per programmatori, i primi capitoli sono scritti per essere accessibili a chiunque, indipendentemente dall'esperienza tecnica. In questo capitolo inizieremo con un po' di terminologia, per poi passare ai concetti di fiducia e alla sua applicazione in questi sistemi decentralizzati, infine discuteremo la storia ed il futuro di Lightning Network.

Iniziamo!

Concetti Base di Lightning Network

Mentre esploriamo come funziona effettivamente Lightning Network, andremo in contro a dei termini tecnici che potrebbero inizialmente creare un po' di confusione. Mentre tutti

questi concetti e termini saranno spiegati in dettaglio man mano che avanzeremo nel libro e saranno definiti nel glossario, alcune definizioni di base spiegate già ora renderanno più facile comprendere i concetti nei prossimi due capitoli. Se non capisci ancora tutte le parole in queste definizioni, non è un problema. Comprenderai il tutto mentre leggerai il resto del libro.

Blockchain

> Un registro delle transazioni distribuito, prodotto da una rete di computer. Bitcoin, ad esempio, è un sistema che produce una blockchain. Lightning Network non è di per sé una blockchain, né produce una blockchain. È una rete che si basa su una blockchain esterna già esistente per la sua sicurezza (ovvero quella di Bitcoin).

Firma digitale

> Una firma digitale è uno schema matematico per verificare l'autenticità di messaggi o documenti digitali. Una firma digitale valida dà al destinatario il motivo di ritenere che il messaggio sia stato creato da un mittente noto, che il mittente non possa negare di aver inviato il messaggio e che il messaggio non sia stato alterato durante il transito.

Funzione di hash

> Una funzione hash crittografica è un algoritmo matematico che mappa dati di dimensioni arbitrarie su una stringa di bit di dimensione fissa (un hash) ed è progettata per essere una funzione unidirezionale, ovvero una funzione che non è possibile invertire.

Nodo

> Un computer che partecipa a una rete. Un nodo Lightning è un computer che partecipa alla rete Lightning. Un nodo Bitcoin è un computer che partecipa alla rete Bitcoin. In genere un utente LN eseguirà un nodo Lightning e un nodo Bitcoin.

On-chain vs off-chain

Un pagamento è *on-chain* se viene registrato come transazione sulla blockchain di Bitcoin. I pagamenti inviati tramite canali di pagamento tra i nodi Lightning, e che quindi non sono visibili nella blockchain sottostante, sono chiamati pagamenti *off-chain* (fuori catena). Solitamente su Lightning Network le uniche transazioni on-chain sono quelle utilizzate per aprire e chiudere un canale di pagamento Lightning. Esiste un terzo tipo di transazione di modifica del canale, chiamata splicing, che può essere utilizzata per aumentare/diminuire l'importo dei fondi impegnati in un canale.

Pagamento

Quando il valore viene scambiato sul Lightning Network, lo chiamiamo "pagamento"; mentre quando avviene sulla blockchain di Bitcoin lo chiamiamo "transazione".

Canale di pagamento

Una *relazione finanziaria* tra due nodi sulla Lightning Network, tipicamente implementata da transazioni Bitcoin multi signature (multifirma) che condividono il controllo sui bitcoin gestiti tra i due nodi Lightning.

Routing (instradamento) ed invio

A differenza di Bitcoin in cui le transazioni vengono "inviate" trasmettendole a tutta la rete, Lightning è una rete in cui i pagamenti vengono "instradati" attraverso uno o più canali di pagamento seguendo un percorso dal mittente al destinatario.

Transazione

Una struttura dati che registra il trasferimento del controllo su alcuni fondi (ad esempio, alcuni bitcoin). Lightning Network si basa sulle transazioni Bitcoin per tracciare il controllo dei fondi.

Definizioni più dettagliate e molti altri termini possono essere trovate nel Glossario a fine del libro. Durante la lettura di questo libro spiegheremo capitolo per capitolo cosa significano

questi concetti e come funzionano effettivamente queste tecnologie.

SUGGERIMENTO	In questo libro vedrai "Bitcoin" con la prima lettera maiuscola, che si riferisce al *protocollo Bitcoin* ed è un nome proprio. Vedrai anche "bitcoin", con una *b* minuscola, che si riferisce all'unità di valuta. Ogni bitcoin è ulteriormente suddiviso in 100 milioni di unità ciascuna denominata "satoshi/sat" (singolare) o "satoshis"/"sats" (plurale).

Ora che hai familiarità con questi termini di base, passiamo a un concetto con cui ti senti già a tuo agio: fiducia.

La Fiducia Nelle Reti Decentralizzate

Sentirai spesso dire che Bitcoin e Lightning Network sono "trustless" (senza fiducia). Inizialmente cosa potrebbe confonderti. Dopotutto, la fiducia non è una buona cosa? Le banche usano la parola "fiducia" persino nei loro nomi! Un sistema "trustless", ovvero un sistema privo di fiducia, non è una cosa negativa?

L'uso della parola "trustless" ha lo scopo di trasmettere la capacità di operare *senza aver bisogno* di fiducia negli altri partecipanti del sistema. In un sistema decentralizzato come Bitcoin, puoi sempre scegliere di effettuare transazioni con qualcuno di cui ti fidi. Tuttavia, il sistema garantisce che nessuno possa fregarti anche se non ti fidi dell'altra parte in una transazione. La fiducia è una proprietà piacevole da avere ma non necessaria in un sistema.

Bitcoin e Lightning Network sono in contrasto con i sistemi tradizionali come le banche in cui devi riporre la tua fiducia in una terza parte, poiché controlla i tuoi soldi. Se la banca viola la tua fiducia, potresti fare ricorso in tribunale, con un enorme costo di tempo, denaro e fatica.

Trustless non significa privo di fiducia. Significa che la fiducia non è un prerequisito necessario per tutte le transazioni e che puoi negoziare anche con persone di cui non ti fidi perché il sistema impedisce che ti imbroglino.

Prima di spiegare in che modo funziona Lightning Network, è importante comprendere un concetto che sta alla base di Bitcoin, LN e molti altri sistemi simili: qualcosa che chiamiamo "fairness protocol" (protocollo di equità). Un protocollo di equità è un modo per ottenere risultati equi tra i partecipanti, che non hanno bisogno di fidarsi l'uno dell'altro, senza la

necessità di un'autorità centrale. È la spina dorsale di sistemi decentralizzati come Bitcoin.

Equità Senza Autorità Centrale

Quando le persone hanno interessi contrastanti, come possono stabilire una fiducia sufficiente per impegnarsi in un comportamento cooperativo o transazionale? La risposta a questa domanda è al centro di diverse discipline scientifiche e umanistiche, come l'economia, la sociologia, la psicologia comportamentale e la matematica. Alcune di queste discipline ci danno risposte "morbide" che dipendono da concetti come reputazione, equità, moralità e persino religione. Altre discipline ci danno risposte concrete che dipendono solo dal presupposto che i partecipanti a queste interazioni agiranno razionalmente, con il proprio interesse personale come obiettivo principale.

In generale, ci sono una manciata di modi per garantire risultati equi nelle interazioni tra individui che potrebbero avere interessi contrastanti:

Richiedendo fiducia

> Interagisci solo con persone di cui ti fidi già, a causa di interazioni precedenti, reputazione o relazioni familiari. Questo funziona abbastanza bene su piccola scala, specialmente all'interno di famiglie e piccoli gruppi, da essere la base più comune per un comportamento cooperativo. Sfortunatamente, non si ridimensiona e soffre di pregiudizi tribali (in gruppo).

Stato di Diritto

> Stabilisci regole per le interazioni che vengono applicate da un'istituzione. Questo si ridimensiona meglio, ma non può farlo a livello globale a causa delle differenze di costumi e tradizioni, nonché dell'incapacità di ridimensionare le istituzioni. Un brutto effetto collaterale di questa soluzione è che le istituzioni diventano sempre più potenti man mano che crescono e ciò può portare alla corruzione.

Terze parti fidate

> Inserisci un intermediario in ogni interazione per imporre l'equità. Combinato con lo

"stato di diritto" per garantire il controllo degli intermediari, questo scala meglio, ma soffre dello stesso squilibrio di potere: gli intermediari diventano molto potenti e possono attirare la corruzione. La concentrazione del potere porta al rischio sistemico e al fallimento sistemico ("troppo grande per fallire").

Protocolli di equità teorica del gioco

Quest'ultima categoria emerge dalla combinazione di internet e crittografia ed è oggetto di questa sezione. Vediamo come funziona e quali sono i vantaggi e svantaggi.

Protocolli Affidabili Senza Intermediari

Bitcoin e Lightning Network sono sistemi crittografici che ti consentono di effettuare transazioni con persone (e computer) di cui non ti fidi. Tale operazione viene spesso definita "trustless", anche se in realtà non è completamente esente da fiducia. Devi fidarti del software che esegui e che il protocollo implementato dal software si tradurrà in risultati equi.

La grande distinzione tra un sistema crittografico come questo e un sistema finanziario tradizionale è che nella finanza tradizionale hai una *terza parte fidata*, ad esempio una banca, che garantisce che i risultati siano equi. Un problema significativo con tali sistemi è che danno troppa potenza a terzi e sono vulnerabili a un *singolo punto di fallimento*. Se la stessa terza parte fidata viola la fiducia o tenta di imbrogliare, la base della fiducia si rompe.

Mentre studierai i sistemi crittografici, noterai un certo schema: invece di affidarsi a una terza parte fidata, questi sistemi tentano di prevenire risultati ingiusti utilizzando un sistema di incentivi e disincentivi. Nei sistemi crittografici riponi fiducia nel *protocollo*, che è effettivamente un sistema con un insieme di regole che, se adeguatamente progettate, applicheranno correttamente gli incentivi e i disincentivi desiderati. Il vantaggio di questo approccio è duplice: non solo eviti di fidarti di una terza parte, ma riduci anche la necessità di imporre risultati equi. Finché i partecipanti seguono il protocollo concordato e rimangono all'interno di esso, il meccanismo di incentivazione ottiene risultati equi senza applicazione.

L'uso di incentivi e disincentivi per ottenere risultati equi è un aspetto di una branca della matematica chiamata *teoria dei giochi*, che studia "modelli di interazione strategica tra

decisori razionali"[1]. Sistemi crittografici che controllano le interazioni finanziarie tra i partecipanti, come Bitcoin e Lightning Network, fanno molto affidamento sulla teoria dei giochi per impedire ai partecipanti di barare e consentire ai partecipanti che non si fidano l'uno dell'altro di ottenere risultati equi.

Sebbene la teoria dei giochi e il suo utilizzo nei sistemi crittografici possano sembrare inizialmente confusionari e poco familiari, è probabile che tu abbia già familiarità con questi sistemi nella tua vita quotidiana; semplicemente non li riconosci ancora. Nella sezione seguente useremo un semplice esempio dell'infanzia per aiutarci a identificare lo schema di base. Una volta compreso lo schema di base, lo vedrai ovunque nel mondo di Bitcoin e delle blockchain e arriverai a riconoscerlo in modo rapido e intuitivo.

In questo libro chiamiamo tale modello *protocollo di equità*, definito come un processo che utilizza un sistema di incentivi e/o disincentivi per garantire risultati equi per i partecipanti che non si fidano l'uno dell'altro. L'applicazione di un protocollo di equità è necessaria solo per garantire che i partecipanti non possano sfuggire agli incentivi o ai disincentivi.

Un Protocollo di Equità in Azione

Diamo un'occhiata a un esempio di protocollo di equità che potresti già conoscere.

Immagina un pranzo in famiglia, con un genitore e due figli. I bambini sono schizzinosi e l'unica cosa che accetteranno di mangiare sono le patate fritte. Il genitore ha preparato una ciotola di patate fritte. I due fratelli devono condividere il piatto di patatine. Il genitore deve garantire un'equa distribuzione a ciascun figlio; in caso contrario, il genitore dovrà sentire continue lamentele (forse per tutto il giorno) e c'è sempre la possibilità che una situazione ingiusta si trasformi in violenza. Cosa deve fare un genitore?

Ci sono diversi modi in cui l'equità può essere raggiunta in questa interazione strategica tra due fratelli che non si fidano l'uno dell'altro e hanno interessi contrastanti. Il metodo ingenuo ma comunemente usato è che i genitori usino la loro autorità come terza parte fidata: dividono la ciotola di patatine in due porzioni. Questo è simile alla finanza tradizionale, in cui una banca, un contabile o un avvocato agisce come una terza parte fidata per prevenire qualsiasi imbroglio tra due parti che vogliono effettuare transazioni.

Il problema con questo scenario è che conferisce molto potere e responsabilità nelle mani di

una terza parte fidata. In questo esempio, il genitore è pienamente responsabile dell'equa assegnazione delle patatine e le parti si limitano ad aspettare, guardare e lamentarsi. I bambini accusano il genitore di fare favoritismi e di non distribuire equamente le patatine. I fratelli litigano per le patatine, urlando "quella patatina è più grande!" e trascinando il genitore nella loro lotta. Suona orribile, vero? Il genitore dovrebbe urlare più forte? Togliere tutte le patatine? Minacciare e non permettere di mangiare più patatine e lasciare che quei bambini ingrati abbiano fame?

Esiste una soluzione molto migliore: ai fratelli viene insegnato a giocare a un gioco chiamato "dividi e scegli". Ad ogni pranzo un fratello divide la ciotola di patatine in due porzioni e *l'altro* fratello può scegliere quale porzione desidera. Quasi immediatamente, i fratelli capiscono la dinamica di questo gioco. Se uno che divide commette un errore o cerca di imbrogliare, l'altro fratello può "punirlo" scegliendo la ciotola più grande. È nell'interesse di entrambi i fratelli, ma soprattutto di quello che divide la ciotola per primo, giocare lealmente. Solo l'imbroglione perde in questo scenario. Il genitore non deve nemmeno usare la propria autorità o far rispettare l'equità. Tutto ciò che il genitore deve fare è *far rispettare il protocollo*; fintanto che i fratelli non possono sottrarsi ai ruoli assegnati di "divisore" e "selettore", il protocollo stesso garantisce un risultato equo senza la necessità di alcun intervento. Il genitore in questo caso non può fare favoritismi o distorcere il risultato.

Primitive di Sicurezza come Elementi Costitutivi

Affinché un protocollo di equità come questo funzioni, devono esserci determinate garanzie, o *primitive di sicurezza*, che possono essere combinate per garantirne l'applicazione. La prima primitiva di sicurezza è l'*ordinamento/sequenziamento temporale rigoroso* (strict time ordering/sequencing): l'azione di "divisione" deve avvenire prima dell'azione di "scelta". Non è immediatamente ovvio, ma a meno che tu non possa garantire che l'azione A avvenga prima dell'azione B, il protocollo va in frantumi. La seconda primitiva di sicurezza è *l'impegno con il non ripudio* (commitment with nonrepudiation). Ogni fratello deve impegnarsi nella scelta del ruolo: divisore o selezionatore. Inoltre, una volta completata la divisione, il divisore si impegna nella divisione che ha creato: non può ripudiare quella scelta e riprovare.

I sistemi crittografici offrono una serie di primitive di sicurezza che possono essere combinate in diversi modi per costruire un protocollo di equità. Oltre al sequenziamento e

all'impegno, possiamo utilizzare anche molti altri strumenti:

- Funzioni di hash per generare impronte digitali dei dati sotto osservazione, come forma di impegno o come base per una firma digitale
- Firme digitali per l'autenticazione, il non ripudio e la prova della proprietà di un segreto
- Crittografia/decrittografia per limitare l'accesso alle informazioni ai soli partecipanti autorizzati

Questo è solo un piccolo assaggio del più grande elenco di primitive di sicurezza e crittografia in uso. Ogni giorno vengono inventate sempre più primitive e combinazioni di base.

Nel nostro esempio di vita reale, abbiamo visto una forma di protocollo di equità chiamato "dividi e scegli". Questo è solamente un esempio di una miriade di protocolli di equità diversi che possono essere costruiti combinando gli elementi costitutivi delle primitive di sicurezza in modi diversi; ma lo schema di base è sempre lo stesso: due o più partecipanti interagiscono senza fidarsi l'uno dell'altro impegnandosi in una serie di passaggi che fanno parte di un protocollo concordato. I passaggi del protocollo prevedono incentivi e disincentivi per garantire che se i partecipanti sono razionali, ingannare è controproducente e l'equità è un risultato automatico. L'applicazione non è necessaria per ottenere risultati equi, è solo necessaria per impedire ai partecipanti di rompere il protocollo concordato.

Ora che hai compreso questo schema di base, inizierai a vederlo ovunque in Bitcoin, Lightning Network e molti altri sistemi simili. Diamo un'occhiata ad alcuni esempi specifici...

Esempio del Protocollo di Equità

L'esempio più importante di un protocollo di equità è l'algoritmo di consenso di Bitcoin, il Proof of Work (PoW). In Bitcoin, i miner competono per verificare le transazioni e aggregarle in blocchi. Per garantire che i miner non imbroglino, senza porre la fiducia nel concetto di autorità, Bitcoin utilizza un sistema di incentivi e disincentivi. I miner devono utilizzare l'elettricità e hardware specifici per "lavorare" e "provare" il loro lavoro all'interno di ogni blocco. Ciò si ottiene grazie a una proprietà delle funzioni di hash in cui il valore di output è distribuito casualmente sull'intero intervallo di possibili output. Se i miner riescono a produrre un blocco valido abbastanza velocemente, vengono ricompensati guadagnando la

ricompensa del blocco. Costringere i miner a utilizzare molta elettricità prima che la rete consideri il loro blocco significa che hanno un incentivo nel convalidare correttamente le transazioni nel blocco stesso. Se imbrogliano o commettono qualsiasi tipo di errore, il loro blocco viene rifiutato e l'elettricità che hanno usato per "dimostrarlo" viene essenzialmente "sprecata". Nessuno deve obbligare i miner a produrre blocchi validi; la ricompensa e la punizione è ciò che li incentiva a farlo. Tutto ciò che il protocollo deve fare è garantire che vengano accettati solo blocchi validi tramite la Proof of Work.

Il modello del protocollo di equità può essere trovato anche in molti aspetti diversi di LN:

- Coloro che finanziano i canali si assicurano di avere una transazione di rimborso firmata prima di pubblicare la transazione di finanziamento.
- Ogni volta che un canale viene modificato e ne cambia lo stato, il vecchio stato viene "revocato", garantendo che se qualcuno tenta di trasmetterlo (nonostante sia appunto obsoleto), perda l'intero saldo del canale e venga punito.
- Chi inoltra pagamenti sa che se si impegna a muovere fondi verso il nodo successivo può ottenere un rimborso o essere pagato dal nodo che lo precede.

Questo modello è onnipresente. I risultati equi non sono imposti da alcuna autorità. Emergono come la naturale conseguenza di un protocollo che premia l'equità e punisce l'inganno, un protocollo che sfrutta l'interesse personale dirigendolo verso risultati equi.

Bitcoin e Lightning Network sono entrambe implementazioni di protocolli di equità. Allora perché abbiamo bisogno di Lightning? Bitcoin non è abbastanza?

Motivazioni per Lightning Network

Bitcoin è un sistema che registra le transazioni su un registro pubblico replicato a livello globale. Ogni transazione viene vista, convalidata e archiviata da ogni computer partecipante. Come puoi immaginare, questo genera molti dati ed è difficile da scalare.

Con l'aumento di Bitcoin e della domanda di transazioni, il numero di transazioni in ciascun blocco è aumentato fino a raggiungere il limite di dimensione del blocco stesso. Una volta che i blocchi sono "pieni", le transazioni in eccesso vengono messe in attesa. Molti utenti

aumenteranno le commissioni che sono disposti a pagare per acquistare spazio per le loro transazioni nel blocco successivo e permettere lo spostamento di bitcoin.

Se la domanda continua a superare la capacità della rete, un numero crescente di transazioni degli utenti rimane in attesa di conferma. All'aumentare della concorrenza delle transazioni, aumentano anche le commissioni, quindi anche il costo di ciascuna transazione, rendendo molte transazioni di valore inferiore (ad esempio microtransazioni) completamente antieconomiche durante periodi dove la domanda è particolarmente elevata.

Per risolvere questo problema, potremmo aumentare il limite della dimensione del blocco per creare spazio per più transazioni. Un aumento dell'"offerta" di spazio in blocchi porterebbe in teoria a una media di prezzo inferiore per le commissioni di ogni transazione.

Tuttavia, l'aumento della dimensione del blocco sposta il costo sugli operatori del nodo e richiede loro di spendere più risorse per convalidare e archiviare la blockchain. Poiché le blockchain sono protocolli basati sul "gossip", ogni nodo è tenuto a conoscere e convalidare ogni singola transazione che si verifica sulla rete. Inoltre, una volta convalidata, ogni transazione e blocco devono essere propagati ai "vicini" del nodo, moltiplicando i requisiti di larghezza di banda. Pertanto, maggiore è la dimensione del blocco, maggiori sono i requisiti di larghezza di banda, elaborazione e archiviazione per ogni singolo nodo. Cercare di aumentare la capacità dei blocchi per permettere un maggior numero di transazioni ha l'effetto indesiderato di centralizzare il sistema riducendo il numero di nodi ed operatori. Poiché gli operatori non sono compensati nel gestire nodi, se gli stessi nodi sono molto costosi da mantenere, solo pochi operatori ben finanziati continueranno a gestirli.

Scalare le Blockchain

Gli effetti collaterali dell'aumento della dimensione del blocco o della diminuzione del tempo di blocco rispetto alla centralizzazione della rete sono estremamente gravi, come dimostrato da alcuni semplici calcoli matematici.

Supponiamo che l'utilizzo di Bitcoin cresca in modo che la rete debba elaborare 40.000 transazioni al secondo, che è il livello approssimativo di elaborazione delle transazioni della rete Visa durante un tipico picco di utilizzo.

Supponendo 250 byte in media per transazione, ciò comporterebbe un flusso di dati di 10 megabyte al secondo (MBps) o 80 megabit al secondo (Mbps) solo per poter ricevere tutte le transazioni. Ciò non include il sovraccarico di traffico dovuto all'inoltro delle informazioni

sulla transazione ad altri nodi. Sebbene 10 MBps non sembri un requisito così grande nel contesto della fibra ottica ad alta velocità e delle velocità delle reti 5G, escluderebbero di fatto chiunque non possa soddisfare questo requisito nel lanciare ed eseguire un nodo, specialmente nei paesi in cui Internet ad alte prestazioni non è conveniente o disponibile .

Le persone hanno inoltre molte altre richieste che gravano sulla loro larghezza di banda (basti pensare all'utilizzo quotidiano di una connessione, che comprende videochiamate, streaming di film e serie tv, e navigazione sui social media), e non ci si può aspettare che spendano così tanto solo per ricevere e propagare transazioni.

Inoltre, la memorizzazione di queste informazioni localmente comporterebbe 864 gigabyte al giorno. Si tratta di circa un terabyte di dati, che equivale a circa un disco rigido al giorno.

Anche la verifica di 40.000 firme dell'algoritmo di firma digitale a curva ellittica (ECDSA) al secondo è a malapena fattibile (vedi questo articolo su StackExchange: https://bitcoin.stackexchange.com/questions/95339/how-many-bitcoin-transactions-can-be-verified-per-second), rendendo quasi impossibile il *download del blocco iniziale* (IBD, Initial Block Download) della blockchain di Bitcoin (che equivale a scaricare, sincronizzare e verificare l'intera storia e transazioni a partire dal blocco di genesi) senza un hardware molto costoso e performante.

Sebbene 40.000 transazioni al secondo sembrino molte, tale numero riporta esattamente ciò che accade nelle tradizionali reti di pagamento come Visa e Mastercard solo nelle ore di punta. È probabile che le innovazioni nei pagamenti tra computer come accade nell'IoT, Internet of Things, nelle microtransazioni e in altre applicazioni spingano la domanda ad un ordine di grandezza nettamente superiore.

In poche parole: non puoi scalare una blockchain per convalidare le transazioni del mondo intero in modo decentralizzato.

Ma cosa accadrebbe se a ogni nodo non fosse richiesto di conoscere e convalidare ogni singola transazione? E se ci fosse un modo per permettere transazioni off-chain scalabili, senza perdere la sicurezza della rete Bitcoin?

Nel febbraio 2015, Joseph Poon e Thaddeus Dryja hanno proposto una possibile soluzione al problema della scalabilità di Bitcoin, con la pubblicazione di "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments"[2].

Nel whitepaper (ormai obsoleto), Poon e Dryja stimano che affinché Bitcoin raggiunga le 47.000 transazioni al secondo elaborate da Visa, sarebbero necessari 8 GB di grandezza di

blocco. Ciò renderebbe l'esecuzione di un nodo completamente insostenibile per chiunque tranne per grandi imprese ed entità industriali. Il risultato sarebbe una rete in cui solo pochi utenti potrebbero effettivamente convalidare lo stato della blockchain. L'intero concetto di Bitcoin si basa sul fatto che gli utenti possano convalidarne la blockchain da soli, senza doversi fidare di terze parti, e mantenerne la decentralizzazione. L'aumento del costo per eseguire un nodo, che ricadrebbe sulle spalle degli utenti stessi, costringerebbe la persona media a fidarsi di terze parti per validare lo stato della blockchain e, conseguentemente, delle transazioni proprie e di altri, infrangendo in definitiva il modello di fiducia di Bitcoin.

Lightning Network propone una nuova rete, un secondo livello, in cui gli utenti possono effettuare pagamenti tra loro in maniera peer-to-peer (da persona a persona), senza la necessità di pubblicare una transazione sulla blockchain di Bitcoin per ogni pagamento. Gli utenti possono pagarsi a vicenda su Lightning Network tutte le volte che vogliono, senza creare transazioni Bitcoin aggiuntive o dover pagare delle commissioni per scrivere tali sulla blockchain. Si avvalgono della blockchain di Bitcoin solo per caricare inizialmente dei bitcoin su LN e per accordarsi (*settlement*), ovvero per rimuovere bitcoin da LN. Il risultato è che molti più pagamenti Bitcoin possono essere effettuati fuori catena (*off chain*), mentre solo il caricamento iniziale e le transazioni di regolamento finale devono essere scritte, convalidate e archiviate dai nodi Bitcoin. Oltre a ridurre il carico sui nodi, i pagamenti su LN sono più economici poichè gli utenti non devono pagare commissioni per aggiungere informazioni sulla blockchain e, contemporaneamente, sono più privati perché non vengono pubblicati a tutti i partecipanti della rete e non vengono archiviati in modo permanente.

Sebbene Lightning Network sia stato inizialmente concepito per Bitcoin, può essere implementato su qualsiasi blockchain che soddisfi alcuni requisiti tecnici di base. Altre blockchain, come Litecoin, supportano già Lightning Network. Inoltre, molte altre blockchain stanno sviluppando soluzioni simili di *secondo livello* ("layer 2") per scalare.

Le Caratteristiche Distintive di Lightning Network

Lightning Network è una rete che opera come protocollo di secondo livello sopra Bitcoin ed altre blockchain. LN consente pagamenti veloci, sicuri, privati, trustless e senza autorizzazione. Ecco alcune delle sue caratteristiche:

- Gli utenti di Lightning Network possono instradare pagamenti tra di loro a basso costo e in tempo reale.

- Gli utenti che scambiano valore tramite Lightning Network non devono attendere le conferme di blocco.
- Una volta che un pagamento su Lightning Network è completato, di solito entro pochi secondi, è definitivo e non può essere annullato. Come una transazione Bitcoin, un pagamento su Lightning Network può essere rimborsato solo dal destinatario.
- Mentre le transazioni Bitcoin on-chain vengono trasmesse e verificate da tutti i nodi della rete, i pagamenti instradati su Lightning Network vengono trasmessi tra coppie di nodi e non sono visibili a tutti, con una conseguente maggior privacy.
- A differenza delle transazioni sulla rete Bitcoin, i pagamenti instradati su Lightning Network non devono essere archiviati in modo permanente. LN utilizza meno risorse ed è quindi più economico. Questa proprietà ha anche vantaggi per la privacy.
- Lightning Network utilizza il concetto di onion routing, simile al protocollo utilizzato dalla rete di anonimizzazione The Onion Router (Tor), in modo che i nodi coinvolti nell'instradamento di un pagamento siano solo a conoscenza del loro predecessore e successore nel percorso verso il destinatario. Ciò comporta che gli stessi nodi non siano in grado di conoscere quale sia il mittente e quale sia il destinatario.
- Se utilizzato al di sopra di Bitcoin, Lightning Network fa uso di bitcoin reali, che comporta di essere sempre in possesso (custodia) del proprio valore ed il pieno controllo di tale è in mano all'utente che ne fa uso. I bitcoin che gravitano all’interno di LN non sono un token o una moneta separata, sono bitcoin veri e propri.

Lightning Network: Casi D'Uso, Utenti e le Loro Storie

Per capire meglio come funziona effettivamente Lightning Network e perché le persone lo usano, elencheremo alcuni esempi circa un certo numero di utenti e le loro storie.

Nei nostri esempi, alcune persone hanno già utilizzato Bitcoin e altre sono completamente nuove nel mondo di Bitcoin. Le persone e le loro storie, come elencate di seguito, illustrano uno o più casi d'uso specifici. Li rivisiteremo in questo libro:

Consumatore

Alice è un utente Bitcoin che desidera effettuare pagamenti veloci, sicuri, economici e privati per piccoli acquisti al dettaglio. Compra il caffè in bitcoin, usando Lightning Network.

Commerciante

Bob possiede una caffetteria, "Bob's Cafe". I pagamenti Bitcoin on-chain non scalano per piccole quantità come una tazza di caffè, quindi usa Lightning Network per accettare pagamenti in bitcoin quasi istantaneamente e con commissioni basse.

Aziende di servizi software

Chan è un imprenditore cinese che vende servizi software relativi a Lightning Network, oltre a Bitcoin ed altre criptovalute. Chan vende questi servizi su Internet implementando micropagamenti tramite LN. Ha inoltre avviato un servizio di fornitura di liquidità (liquidity provider service) che permette il noleggio di capacità del canale in entrata (inbound channel capacity) sul LN, addebitando una piccola commissione per ogni periodo di noleggio.

Gamer

Dina è una gamer adolescente russa. Gioca a diversi giochi per computer, ma i suoi preferiti sono quelli che hanno una "economia interna al gioco" basata su soldi veri. Mentre gioca, guadagna anche denaro acquisendo e vendendo oggetti virtuali. Lightning Network le consente di effettuare transazioni di basso valore per oggetti di gioco virtuali e di guadagnare piccole somme per il completamento di missioni specifiche.

Conclusione

In questo capitolo abbiamo parlato del concetto fondamentale che sta alla base sia di Bitcoin che di Lightning Network: il protocollo di equità.

Abbiamo esaminato la storia di Lightning Network e le motivazioni alla base delle soluzioni di *scaling* di secondo livello per Bitcoin e altre reti basate sul concetto di blockchain.

Abbiamo appreso la terminologia di base che comprende nodi, canali di pagamento, transazioni on-chain e pagamenti off-chain.

Infine, abbiamo parlato dei casi d’uso di Alice, Bob, Chan e Dina, che seguiremo per tutto il resto del libro. Nel prossimo capitolo, ci concentreremo su Alice e seguiremo il suo processo di pensiero mentre sceglie un portafoglio (*wallet*) Lightning e si prepara ad effettuare il suo primo pagamento su LN per comprare una tazza di caffè da Bob.

1. La pagina di Wikipedia sulla teoria dei giochi fornisce maggiori informazioni.
2. Joseph Poon e Thaddeus Dryja. "The Bitcoin Lightning Network: pagamenti istantanei fuori catena scalabili". BOZZA Versione 0.5.9.2. 14 gennaio 2016. https://lightning.network/lightning-network-paper.pdf.

Capitolo 2. Inizio

In questo capitolo, inizieremo da dove la maggior parte delle persone inizia quando si addentra per la prima volta in Lightning Network, scegliendo il software per parteciparvi. Esamineremo le scelte di due utenti che rappresentano un caso d'uso comune per Lightning Network e impareremo esempio per esempio. Alice, una cliente di una caffetteria, utilizzerà un wallet Lightning sul suo dispositivo mobile per acquistare caffè da Bob's Cafe. Bob, un commerciante, utilizzerà un nodo Lightning ed un wallet per gestire un sistema di punti vendita nel suo bar, in modo che possa accettare pagamenti tramite Lightning Network.

Il Primo Wallet Lightning di Alice

Alice è una utilizzatrice di Bitcoin di lunga data. Abbiamo incontrato Alice per la prima volta nel capitolo 1 di *Mastering Bitcoin*,[1] quando ha comprato una tazza di caffè dal bar di Bob usando una transazione Bitcoin. Se non hai ancora familiarità con il funzionamento delle transazioni Bitcoin o hai bisogno di un aggiornamento, leggi Mastering BItcoin o l'Appendice A in fondo a questo libro.

Alice ha recentemente appreso che Bob's Cafe ha appena iniziato ad accettare pagamenti LN! Alice non vede l'ora di conoscere e sperimentare Lightning Network; vuole essere uno dei primi clienti LN di Bob. Per fare ciò, innanzitutto, Alice deve scegliere un wallet Lightning che soddisfi le sue esigenze.

Alice non vuole affidare a terzi la custodia dei suoi bitcoin. Ha imparato abbastanza per sapere come usare un wallet. Vuole anche un wallet mobile in modo da poterlo utilizzare per piccoli importi, quindi sceglie il portafoglio *Eclair*, un popolare wallet mobile Lightning non-custodial. Cerchiamo di capire come e perché ha fatto questa scelta.

Nodi Lightning

Si può accede a Lightning Network tramite software in grado di parlare il protocollo LN. Un *nodo Lightning Network* (nodo LN, o semplicemente nodo) è un software che presenta tre caratteristiche importanti. Innanzitutto, i nodi Lightning sono wallet, che inviano e ricevono

pagamenti sulla rete Lightning e sulla rete Bitcoin. In secondo luogo, i nodi devono comunicare in modalità peer-to-peer con altri nodi Lightning che compongono la rete. Infine, i nodi Lightning devono poter anche accedere alla blockchain di Bitcoin (o ad altre blockchain) per proteggere i fondi utilizzati per i pagamenti.

Gli utenti hanno il più alto grado di controllo eseguendo il proprio nodo Bitcoin e il proprio nodo Lightning. Tuttavia, i nodi Lightning possono anche utilizzare un client Bitcoin leggero, comunemente indicato come Simplified Payment Verification (SPV), per interagire con la blockchain di Bitcoin.

Explorer Lightning

Gli explorer LN sono strumenti che mostrano statistiche di nodi, canali e capacità della rete.

Di seguito elenchiamo gli explorer più famosi:

- Explorer Lightning di 1ML
- Explorer Lightning di ACINQ, con una migliore interfaccia
- Explorer Lightning di Amboss, con metriche della community e visualizzazioni intuitive
- Explorer Lightning di Fiatjaf, con molti diagrammi
- Explorer Lightning di hashXP

NOTA	Tieni presente che quando utilizzi Explorer Lightning, proprio come con altri block explorer, la gestione della tua privacy può diventare un problema. Se gli utenti sono negligenti, il sito Web può tracciare i loro indirizzi IP e tracciare il loro comportamento (ad esempio, i nodi a cui gli utenti sono interessati). Inoltre, va notato che, poiché non esiste un consenso globale sull'attuale grafo Lightning o sullo stato corrente di qualsiasi criterio di canali esistenti, gli utenti non dovrebbero mai fare affidamento sugli Explorer Lightning per recuperare informazioni aggiornate. Inoltre, man mano che gli utenti aprono, chiudono e aggiornano i canali, il grafo cambierà e i singoli Explorer Lightning potrebbero non essere aggiornati. Utilizza un Explorer per visualizzare la rete o raccogliere informazioni, ma non

	trattarlo come fonte autorevole di ciò che sta accadendo su Lightning Network. Per avere una visione autorevole della rete, utilizza il tuo nodo Lightning, che creerà un grafo dei canali e raccoglierà varie statistiche, che puoi visualizzare con un'interfaccia grafica.

Wallet Lightning

Il termine *Wallet Lightning* (che, nel corso di questo libro, chiameremo anche *portafoglio*) è alquanto ambiguo perché può descrivere un'ampia varietà di componenti combinati con alcune interfacce utente. I componenti più comuni dei wallet Lightning includono:

- Un keystore che contiene segreti, come le chiavi private
- Un nodo LN che comunica sulla rete peer-to-peer, come descritto in precedenza
- Un nodo Bitcoin che memorizza i dati della blockchain e comunica con altri nodi
- Una "mappa" di nodi e canali annunciati su Lightning Network
- Un gestore di canali in grado di aprire e chiudere i canali LN
- Un sistema in grado di trovare un percorso di canali collegati dal mittente al destinatario di un pagamento

Un wallet Lightning può permettere tutte queste cose, agendo come un portafoglio "completo", senza fare affidamento su servizi di terze parti. Uno o più componenti possono fare affidamento (parzialmente o interamente) su terze parti che mediano tali funzioni.

Una distinzione *chiave* (gioco di parole) è se la funzione keystore (gestione delle chiavi) è interna o esternalizzata. Nelle blockchain, il controllo delle chiavi determina la custodia dei fondi, come ricorda la frase "your keys, your coins; not your keys, not your coins." (le tue chiavi, le tue monete; non le tue chiavi, non le tue monete). Ogni portafoglio che esternalizza la gestione delle chiavi è chiamato wallet custodial poiché una terza parte che agisce come custode ha il controllo dei fondi dell'utente. Un wallet *non-custodial o di autocustodia*, in confronto, è quello in cui il keystore fa parte del wallet stesso e le chiavi sono controllate direttamente dall'utente. Il termine wallet non-custodial implica semplicemente che il keystore è locale e sotto il controllo dell'utente. Tuttavia, uno o più componenti del wallet

stesso possono o meno essere esternalizzati e fare affidamento a terze parti fidate.

Le blockchain, in particolare le blockchain aperte come Bitcoin, tentano di ridurre al minimo o di eliminare la fiducia in terze parti. Questo è spesso chiamato un modello "trustless" (senza fiducia), sebbene "fiducia ridotta al minimo" sia un termine migliore. In tali sistemi, l'utente si fida delle regole del software, non di terze parti. Pertanto, la questione del controllo sulle chiavi è una considerazione principale nella scelta di un portafoglio Lightning.

Ogni altro componente di un wallet Lightning porta simili considerazioni di fiducia. Se tutti i componenti sono sotto il controllo dell'utente, la quantità di fiducia in terze parti viene ridotta al minimo, portando la massima libertà e tutela all'utente. Naturalmente, questo comporta un compromesso diretto perché con quel potere deriva la corrispondente responsabilità di gestire software complessi.

Ogni utente deve considerare le proprie capacità tecniche prima di decidere quale tipo di wallet Lightning utilizzare. Chi ha forti competenze tecniche dovrebbe utilizzare un wallet Lightning che metta tutti i componenti sotto il controllo diretto dell'utente. Quelli con meno competenze tecniche, ma con il desiderio di controllare i propri fondi, dovrebbero scegliere un wallet Lightning non-custodial. Spesso la fiducia in questi casi riguarda la privacy. Se si decide di esternalizzare alcune funzionalità ad una terza parte, solitamente si rinuncia ad un po' di privacy poiché la terza parte apprenderà una serie di informazioni private.

Infine, chi cerca semplicità e convenienza, anche a scapito del controllo e della sicurezza, può scegliere un portafoglio Lightning "custodial". Questa è l'opzione tecnicamente meno impegnativa, ma *mina il modello di fiducia di Bitcoin* e dovrebbe quindi essere considerato solo come un trampolino di lancio verso un maggiore controllo e fiducia in se stessi.

Esistono molti modi in cui i wallet possono essere caratterizzati o classificati. Le domande più importanti da porre su un portafoglio specifico sono:

1. Questo wallet Lightning può essere collegato ad un nodo Lightning personale o utilizza un nodo Lightning di terze parti?
2. Questo wallet Lightning può essere collegato ad un nodo Bitcoin o utilizza un nodo Bitcoin di terze parti?
3. Questo wallet Lightning memorizza le chiavi sotto il controllo dell'utente (autocustodia) o le chiavi sono detenute da un custode/terza parte?

SUGGERIMENTO	Se un wallet Lightning utilizza un nodo Lightning di terze parti, è questo

	nodo Lightning di terze parti che decide come comunicare con Bitcoin. Pertanto, l'utilizzo di un nodo Lightning di terze parti implica che stai utilizzando anche un nodo Bitcoin di terze parti. Solo quando il wallet Lightning utilizza il proprio nodo Lightning, viene data la possibilità tra lo scegliere il proprio nodo Bitcoin o un nodo Bitcoin terzo.

Al più alto livello di astrazione, le domande 1 e 3 sono le più elementari. Da queste due domande possiamo ricavare quattro possibili categorie. Possiamo inserire queste quattro categorie in un quadrante; ma ricorda che questo è solo un modo per classificare i wallet.

Tabella 2-1. Quadrante dei wallet Lightning

	Nodo Lightning completo	Nodo Lightning terzo
Non-Custodial	Q1: Elevata competenza tecnica, meno fiducia in terze parti, permissionless (senza autorizzazione)	Q2: Al di sotto delle competenze tecniche medie, al di sotto della media fiducia in terze parti, sono necessarie alcune autorizzazioni
Custodial	Q3: Competenze tecniche superiori alla media, fiducia in terze parti superiori alla media, richiede alcune autorizzazioni	Q4: Scarse competenze tecniche, alta fiducia in terze parti, meno autorizzazioni

Il quadrante 3 (Q3), in cui viene utilizzato un nodo Lightning completo, in cui però le chiavi sono detenute da un custode, non è attualmente comune. I futuri wallet relativi a quel quadrante potrebbero consentire a un utente di preoccuparsi degli aspetti operativi del proprio nodo, ma poi delegare l'accesso alle chiavi ad una terza parte che utilizza principalmente la conservazione a freddo (cold storage).

I wallet Lightning possono essere installati su una gran varietà di dispositivi, inclusi laptop, server e smartphone. Per eseguire un nodo Lightning completo, dovrai utilizzare un server o un desktop, poiché i dispositivi mobili e i laptop in genere non sono sufficientemente potenti

in termini di capacità, elaborazione, durata della batteria e connettività.

La categoria di nodi Lightning di terze parti può essere nuovamente suddivisa:

Lightweight (leggero)

> Ciò significa che il wallet non gestisce un nodo Lightning e quindi deve ottenere informazioni sulla rete Lightning tramite Internet, dal nodo Lightning di qualcun altro.

Nessuno

> Ciò significa che non solo il nodo Lightning è gestito da una terza parte, ma la maggior parte del wallet stesso è gestita da una terza parte nel cloud. Questo è un wallet custodial dove qualcun altro ha il controllo dei fondi.

Queste sottocategorie sono utilizzate nella Tabella 2-2.

Altri termini che richiedono una spiegazione in Esempi di portafogli Lightning popolari nella colonna "Nodo Bitcoin" sono:

Neutrino

> Questo wallet non gestisce un nodo Bitcoin. Invece, accede ad un nodo Bitcoin gestito da qualcun altro (una terza parte) tramite il protocollo Neutrino.

Electrum

> Questo wallet non gestisce un nodo Bitcoin. Invece, accede a un nodo Bitcoin gestito da qualcun altro (una terza parte) tramite il protocollo Electrum.

Bitcoin Core

> Questa è un'implementazione di un nodo Bitcoin.

btcd

> Questa è un'altra implementazione di un nodo Bitcoin.

Nella Tabella 2-2, vediamo alcuni esempi di implementazioni di nodi e wallet Lightning popolari suddivisi per diversi tipi di dispositivi. L'elenco è ordinato per tipo di dispositivo e successivamente in ordine alfabetico.

Tabella 2-2. Esempi di wallet Lightning popolari

Applicazione	Device	Nodo Lightning	Nodo Bitcoin	Custodia
Blue Wallet	Mobile	Nessuno	Nessuno	Custodial
Breez Wallet	Mobile	Full node	Neutrino	Non custodial
Eclair Mobile	Mobile	Leggero	Electrum	Non custodial
lntxbot	Mobile	Nessuno	None	Custodial
Muun	Mobile	Leggero	Neutrino	Non custodial
Phoenix Wallet	Mobile	Leggero	Electrum	Non custodial
Zeus	Mobile	Full node	Bitcoin Core/btcd	Non custodial
Electrum	Desktop	Full node	Bitcoin Core/Electrum	Non custodial
Zap Desktop	Desktop	Full node	Neutrino	Non custodial
c-lightning	Server	Full node	Bitcoin Core	Non custodial
Eclair Server	Server	Full node	Bitcoin Core/Electrum	Non custodial
lnd	Server	Full node	Bitcoin Core/btcd	Non custodial

Testnet Bitcoin

Bitcoin offre una blockchain alternativa per scopi di test chiamata *testnet*, in contrasto con la "normale" blockchain di Bitcoin che viene chiamata *mainnet*. Su testnet, la valuta è *testnet bitcoin* (*tBTC*), che è una copia senza valore di bitcoin utilizzata esclusivamente per i test. Ogni funzione di Bitcoin viene replicata egualmente, ma le unità di bitcoin non valgono nulla, quindi non hai letteralmente nulla da perdere!

Alcuni wallet Lightning possono operare anche su testnet, permettendoti di effettuare pagamenti Lightning tramite testnet bitcoin, senza rischiare fondi reali. Questo è un ottimo modo per sperimentare Lightning in sicurezza. Eclair Mobile, che Alice utilizza in questo capitolo, è un esempio di wallet Lightning che supporta il funzionamento tramite testnet.

Puoi ottenere alcuni tBTC con cui giocare da un *faucet bitcoin testnet*, che distribuisce tBTC gratuiti su richiesta. Ecco alcuni faucet testnet:

- https://coinfaucet.eu/en/btc-testnet
- https://testnet-faucet.mempool.co
- https://bitcoinfaucet.uo1.net
- https://testnet.help/en/btcfaucet/testnet

Tutti gli esempi in questo libro possono essere replicati esattamente su testnet con tBTC, quindi puoi seguirli ed applicarli senza rischiare soldi veri.

Compromesso tra Complessità e Controllo

I portafogli Lightning devono garantire un attento equilibrio tra complessità e controllo da parte dell'utente. Quelli che danno all'utente il massimo controllo sui propri fondi, il più alto grado di privacy e la massima indipendenza dai servizi di terze parti sono necessariamente più complessi e difficili da gestire. Con l'avanzare della tecnologia, alcuni di questi compromessi diventeranno meno netti e gli utenti potrebbero essere in grado di ottenere un maggiore controllo senza una maggiore complessità. Tuttavia, per ora, diverse aziende e progetti stanno esplorando posizioni diverse lungo questo spettro di complessità e del controllo, sperando di trovare il "punto debole" per gli utenti a cui si rivolgono.

Quando scegli un portafoglio, tieni presente che anche se non vedi questi compromessi, esistono ancora. Ad esempio, molti portafogli cercheranno di rimuovere l'onere della gestione dei canali dai propri utenti. Per far ciò, introducono *nodi hub centrali* a cui si connettono automaticamente tutti i loro portafogli. Sebbene questo compromesso semplifichi l'interfaccia e l'esperienza utente, introduce un singolo punto di errore (SpoF – Single Point of Failure) poiché questi nodi hub diventano indispensabili per il funzionamento del portafoglio. Inoltre, affidarsi a un "hub" come questo può ridurre la privacy dell'utente poiché l'hub conosce il mittente e potenzialmente (se costruisce il percorso di pagamento per conto dell'utente) anche il destinatario di ogni pagamento effettuato dal wallet stesso.

Nella sezione successiva, torneremo al nostro primo utente e illustreremo la sua prima configurazione di un wallet Lightning. Ha scelto un wallet più sofisticato rispetto ai più semplici portafogli custodial. Questo ci consente di mostrare parte della complessità sottostante e introdurre alcuni dei meccanismi interni di un portafoglio avanzato. Potresti scoprire che il tuo primo portafoglio ideale è orientato alla facilità d'uso, accettando alcuni dei compromessi di controllo e privacy. O forse sei un utente esperto e desideri gestire i tuoi nodi Lightning e Bitcoin come parte della tua strategia e scelta di wallet.

Download e Installazione di un Wallet Lightning

Quando cerchi un wallet, devi fare attenzione a scegliere una fonte sicura per il software.

Sfortunatamente, molte applicazioni false ruberanno i tuoi soldi e alcune di queste trovano persino la loro strada su siti di software affidabili e presumibilmente controllati come gli store di applicazioni Apple e Google. Sia che tu stia installando il tuo primo o il tuo decimo wallet, fai sempre molta attenzione. Un'app malevola potrebbe non solo rubare i soldi che le affidi, ma potrebbe anche essere in grado di rubare chiavi e password da altre applicazioni compromettendo il sistema operativo del tuo dispositivo mobile.

Alice utilizza un dispositivo Android e utilizzerà il Google Play Store per scaricare e installare il portafoglio Eclair. Cercando su Google Play, trova una voce per "Eclair Mobile", come mostrato nella Figura 2-1.

Figura 2-1. Eclair Mobile nel Google Play Store

SUGGERIMENTO	È possibile sperimentare e testare tutti i software di Bitcoin a rischio zero (tranne che per il proprio tempo) utilizzando la testnet. Puoi anche scaricare il wallet testnet Eclair per provare Lightning (su testnet) andando su Google Play Store.

Alice nota alcuni elementi diversi in questa pagina che l'aiutano ad accertare che questo è molto probabilmente il wallet "Eclair Mobile" corretto che sta cercando. In primo luogo, l'azienda ACINQ[2] è elencata come lo sviluppatore di questo wallet mobile. Alice, grazie alla propria ricerca, sa che è lo sviluppatore corretto. In secondo luogo, il portafoglio è stato installato più di 10.000 volte e ha più di 320 recensioni positive. È improbabile che si tratti di un'app malevola che si è intrufolata nel Google Play Store. Come terzo passo, va sul sito di ACINQ. Verifica che la pagina Web sia protetta controllando che l'indirizzo inizi con https o sia preceduto da un lucchetto in alcuni browser. Sul sito Web, va alla sezione Download o cerca il collegamento al Google Play Store. Trova il collegamento e ci clicca. Paragona che questo collegamento la porti alla stessa app nel Google Play Store. Soddisfatta di questi risultati, Alice installa l'app Eclair sul suo dispositivo mobile.

ATTENZIONE	Presta sempre molta attenzione durante l'installazione di software su qualsiasi dispositivo. Esistono molti falsi wallet che non solo ruberanno i tuoi soldi, ma potrebbero anche compromettere tutte le altre applicazioni sul tuo dispositivo.

Creazione di un Nuovo Wallet

Quando Alice apre l'app Eclair Mobile per la prima volta, le viene presentata una scelta tra "Crea un nuovo portafoglio" o "Importa un portafoglio esistente". Alice creerà un nuovo portafoglio, ma discutiamo prima perché queste opzioni sono presentate in questo modo e cosa significa importare un portafoglio esistente.

Responsabilità Relativa alla Custodia delle Chiavi

Come accennato all'inizio di questa sezione, Eclair è un portafoglio non custodial, il che significa che Alice ha la custodia esclusiva delle chiavi utilizzate per controllare i suoi bitcoin. Significa anche che Alice è responsabile della protezione e del backup di tali chiavi. Se Alice perde le chiavi, nessuno può aiutarla a recuperare i bitcoin, e andranno persi per sempre.

ATTENZIONE	Presta sempre molta attenzione durante l'installazione di software su qualsiasi dispositivo. Esistono molti falsi wallet che non solo ruberanno i tuoi soldi, ma potrebbero anche compromettere tutte le altre applicazioni sul tuo dispositivo.

Parole Mnemoniche

Simile alla maggior parte dei portafogli Bitcoin, Eclair Mobile fornisce una *frase mnemonica* (a volte chiamata anche "seed" o "seedphrase") per il backup di Alice. La frase mnemonica è composta da 24 parole (solitamente in lingua inglese), selezionate casualmente dal software e utilizzate come base per le chiavi che vengono generate dal wallet. Alice può utilizzare la frase mnemonica per ripristinare tutte le transazioni e i fondi nel portafoglio Eclair Mobile in caso di smarrimento di un dispositivo, bug del software o danneggiamento della memoria.

SUGGERIMENTO	Il termine corretto per queste parole di backup è "frase mnemonica". Evitiamo l'uso del termine "seed" (seme) per riferirci a una frase mnemonica perché anche se il suo uso è comune, non è corretto.

Quando Alice sceglie di creare un nuovo portafoglio, vedrà una schermata con la sua frase mnemonica, che assomiglia allo screenshot nella Figura 2-2.

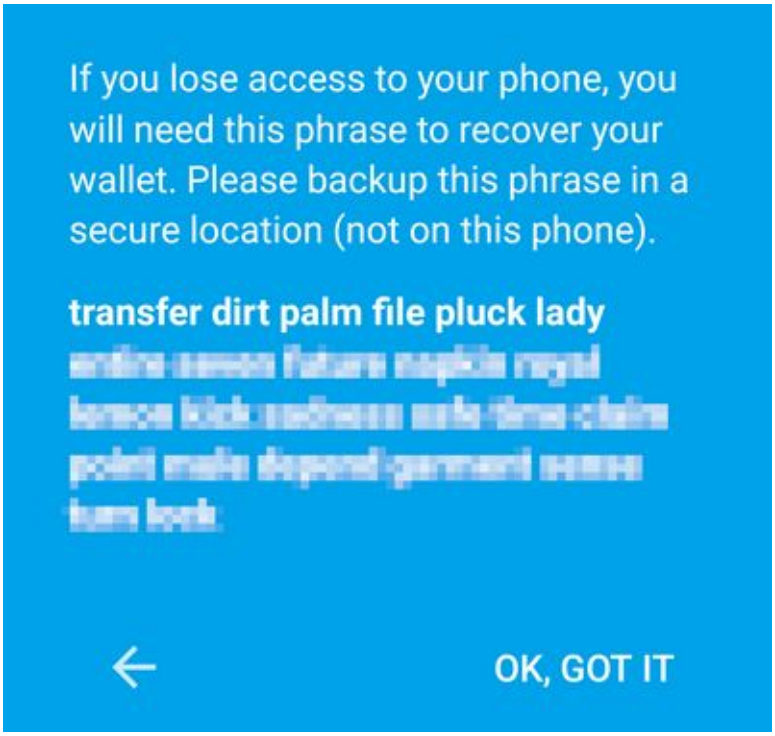

Figura 2-2. Nuova frase mnemonica del wallet appena creato

Nella Figura 2-2, abbiamo volutamente oscurato parte della frase mnemonica per impedire ai lettori di questo libro di riutilizzare la stessa serie di parole.

Memorizzare la Mnemonica in Modo Sicuro

Alice deve stare attenta a memorizzare la frase mnemonica in modo da prevenirne il furto ma anche la perdita accidentale. Il modo consigliato per bilanciare adeguatamente questi rischi è scrivere due copie della frase mnemonica su carta, con ciascuna delle parole numerate: l'ordine è importante.

Una volta che Alice ha registrato la frase mnemonica, dopo aver toccato "OK GOT IT" sul suo schermo, le verrà presentato un quiz per assicurarsi di aver registrato correttamente la mnemonica. Il quiz richiederà tre o quattro parole a caso. Alice non si aspetta un quiz, ma dal momento che ha registrato correttamente la mnemonica, lo supera senza alcuna difficoltà.

Una volta che Alice ha registrato la frase mnemonica e superato il quiz, deve conservare ogni

copia in un luogo sicuro, come un cassetto chiuso a chiave o una cassaforte ignifuga.

ATTENZIONE	Non tentare mai uno schema di sicurezza "fai da te" che si discosti in alcun modo dalla raccomandazioni di questo capitolo. Non tagliare a metà la tua mnemonica, fare uno screenshot, archiviarlo su una USB o sul cloud, crittografarlo o provare qualsiasi altro metodo non standard. Aggiungerai un grado di complessità tale da rischiare una perdita permanente dei fondi. Molte persone hanno perso fondi, non per furto, ma perché hanno provato una soluzione non standard senza avere l'esperienza per bilanciare i rischi connessi. La raccomandazione sulle migliori pratiche è attentamente valutata dagli esperti e adatta alla stragrande maggioranza degli utenti.

Dopo che Alice ha inizializzato il suo portafoglio Eclair Mobile, vedrà un breve tutorial che mostra gli elementi dell'interfaccia utente. Non replicheremo qui il tutorial, ma esploreremo tutti questi elementi mentre seguiamo il tentativo di Alice di comprare una tazza di caffè!

Caricamento di Bitcoin sul portafoglio

Alice ora ha un wallet Lightning... Ma è vuoto! Ora affronta uno degli aspetti più impegnativi di questo esperimento: deve trovare un modo per acquisire dei bitcoin e caricarli sul suo portafoglio Eclair.

SUGGERIMENTO	Se Alice ha già bitcoin in un altro portafoglio, potrebbe scegliere di inviare quei bitcoin al suo portafoglio Eclair invece di acquisirne di nuovi.

Acquisire Bitcoin

Esistono diversi modi in cui Alice può acquisire bitcoin:

- Può scambiare la sua valuta nazionale (ad es. USD) su un exchange di criptovalute.
- Può comprarne da un amico o estraneo in un meetup di Bitcoin, in cambio di contanti.
- Può trovare un *ATM Bitcoin* nella sua zona, che funge da distributore automatico,

scambiando bitcoin per contanti.

- Può offrire le sue competenze o vendere un prodotto ed accettare bitcoin come forma di pagamento.
- Può chiedere al suo datore di lavoro o ai suoi clienti di pagarla in bitcoin.

Tutti questi metodi hanno vari gradi di difficoltà e molti comportano il pagamento di una commissione. Alcuni richiederanno ad Alice dei documenti di identificazione per conformarsi alle normative bancarie locali. Tuttavia, con questi metodi, Alice potrà acquisire dei bitcoin.

Ricevere Bitcoin

Supponiamo che Alice abbia trovato un ATM Bitcoin e abbia deciso di acquistare dei bitcoin in cambio di contanti. Un esempio di ATM Bitcoin, costruito dalla società Lamassu, è mostrato nella Figura 2-3. Tali ATM Bitcoin accettano valuta nazionale (contanti) attraverso una fessura ed inviano bitcoin a un indirizzo Bitcoin scansionato dal portafoglio di un utente utilizzando una fotocamera integrata.

Figura 2-3. Un ATM Bitcoin "Lamassu"

Per ricevere bitcoin nel suo portafoglio Eclair Lightning, Alice dovrà presentare un indirizzo Bitcoin dal portafoglio Eclair Lightning all'ATM. L'ATM invierà quindi i bitcoin appena acquistati da Alice all'indirizzo Bitcoin condiviso.

Per vedere un indirizzo Bitcoin sul portafoglio Eclair, Alice deve scorrere fino alla colonna di sinistra intitolata YOUR BITCOIN ADDRESS (vedi la Figura 2-4), dove vedrà un codice quadrato (chiamato *codice QR*) ed una stringa di lettere e numeri.

Il codice QR contiene la stessa stringa di lettere e numeri mostrata sotto di esso, in un formato facile da scansionare. In questo modo, Alice non deve digitare l'indirizzo Bitcoin. Nello screenshot (Figura 2-4), abbiamo volutamente sfocato entrambi, per impedire ai lettori di inviare erroneamente dei bitcoin a questo indirizzo.

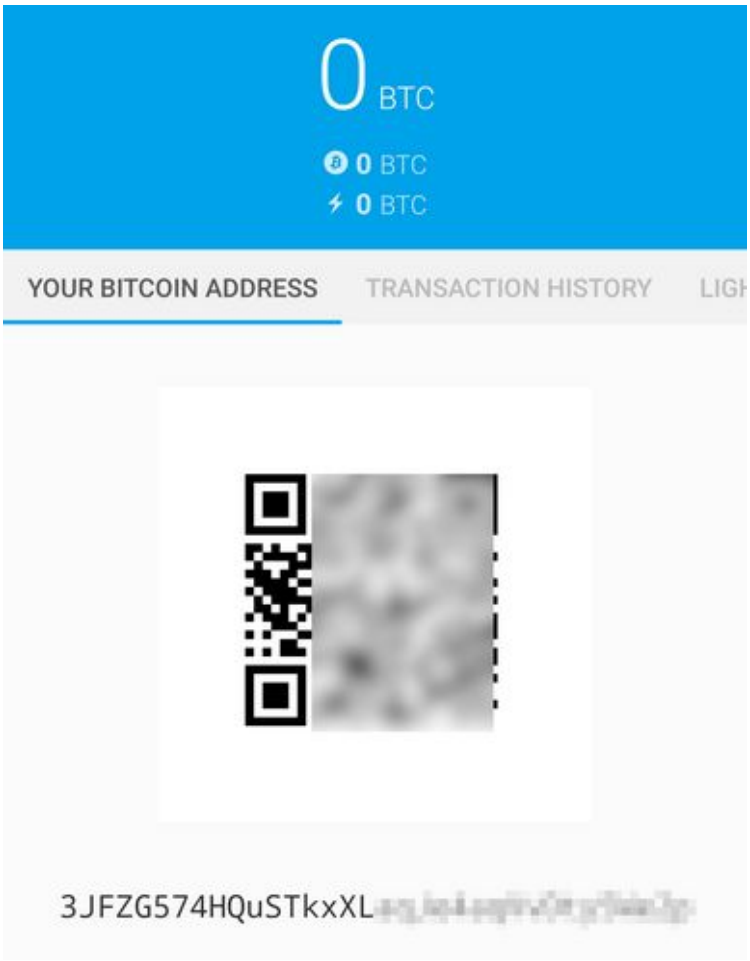

Figura 2-4. L'indirizzo Bitcoin di Alice, mostrato in Eclair

NOTA	Sia gli indirizzi Bitcoin che i codici QR contengono informazioni di rilevamento degli errori che impediscono eventuali errori di digitazione o scansione di produrre un indirizzo Bitcoin "sbagliato". Se c'è un errore nell'indirizzo, qualsiasi portafoglio Bitcoin noterà l'errore e rifiuterà di accettare l'indirizzo Bitcoin come valido.

Alice può portare il suo dispositivo all'ATM e mostrare il codice alla fotocamera integrata, come mostrato nella Figura 2-5. Dopo aver inserito i contanti, riceverà i bitcoin in Eclair!

Figura 2-5. L'ATM Bitcoin scansiona il codice QR

Alice vedrà la transazione dell'ATM nella scheda TRANSACTION HISTORY di Eclair. Sebbene Eclair rileverà la transazione in pochi secondi, ci vorrà del tempo perché la transazione stessa venga "confermata" sulla blockchain di Bitcoin. Come puoi vedere in Figura 2-6, il wallet Eclair di Alice mostra "6+ conf" sotto la transazione, ad indicare che la transazione ha ricevuto il minimo richiesto di sei conferme e che i suoi fondi sono ora pronti per essere utilizzati.

SUGGERIMENTO	Il numero di conferme su una transazione è il numero di blocchi estratti a partire da (e incluso) il blocco che conteneva quella transazione. Sei conferme sono la migliore pratica, ma diversi portafogli Lightning possono considerare un canale aperto dopo un numero qualsiasi di conferme. Alcuni portafogli aumentano persino il numero di conferme previste in base al valore monetario del canale.

Sebbene in questo esempio Alice abbia utilizzato un ATM per acquisire i suoi primi bitcoin, gli stessi concetti di base si applicherebbero anche se utilizzasse uno degli altri metodi del capitolo Acquisire Bitcoin. Ad esempio, se Alice volesse vendere un prodotto o fornire un servizio professionale in cambio di bitcoin, i suoi clienti potrebbero scansionare l'indirizzo Bitcoin con i loro portafogli e pagarla in bitcoin.

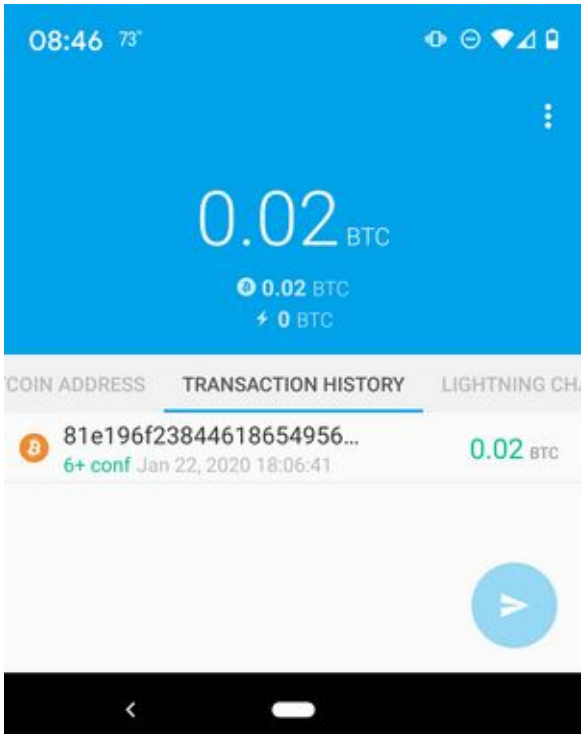

Figura 2-6. Alice riceve i bitcoin

Allo stesso modo, se ha fatturato ad un cliente un servizio offerto su Internet, Alice potrebbe inviargli un'e-mail o un messaggio istantaneo con l'indirizzo Bitcoin o il codice QR, e lo stesso potrebbe incollare o scansionare le informazioni in un portafoglio Bitcoin per pagarla .

Alice potrebbe persino stampare il codice QR ed attaccarlo ad un cartello e mostrarlo pubblicamente per ricevere donazioni. Ad esempio, potrebbe avere un codice QR apposto sulla sua chitarra e ricevere donazioni mentre si esibisce per strada![3]

Infine, se Alice ha acquistato bitcoin da un exchange di criptovalute, potrebbe (e dovrebbe) "ritirare" i bitcoin incollando il suo indirizzo Bitcoin nel sito web dell'exchange. L'exchange invierà quindi i bitcoin direttamente al suo indirizzo.

Da Bitcoin a Lightning Network

Il bitcoin di Alice sono ora controllati tramite il suo wallet Eclair e la transazione è stata registrata sulla blockchain di Bitcoin. A questo punto, i bitcoin di Alice sono *on-chain*, il che

significa che la transazione è stata trasmessa all'intera rete Bitcoin, verificata da tutti i nodi Bitcoin e *minata* (registrata) sulla blockchain.

Finora, il portafoglio Eclair Mobile si è comportato solo come un portafoglio Bitcoin e Alice non ha utilizzato le funzionalità di Lightning Network di Eclair. Come nel caso di molti portafogli Lightning, Eclair collega Bitcoin e Lightning Network in un'unica soluzione, fungendo sia da portafoglio Bitcoin che da portafoglio Lightning.

Ora Alice è pronta per iniziare ad utilizzare LN portando i suoi bitcoin *off-chain* per sfruttare i pagamenti veloci, economici e privati offerti da Lightning Network.

Canali di Lightning Network

Scorrendo verso destra, Alice accede alla sezione LIGHTNING CHANNELS di Eclair. Qui può gestire i canali che collegheranno il suo portafoglio a Lightning Network.

Rivediamo la definizione di un canale LN, per rendere le cose più chiare. In primo luogo, la parola "canale" è una metafora di una *relazione finanziaria* tra il portafoglio Lightning di Alice ed un altro portafoglio Lightning. Lo chiamiamo canale perché è un mezzo per il portafoglio di Alice e questo altro portafoglio per scambiare molti pagamenti tra loro su Lightning Network (off-chain) senza scrivere transazioni sulla blockchain di Bitcoin (on-chain).

Il portafoglio o *nodo* su cui Alice apre un canale è chiamato *peer di canale*. Una volta "aperto", un canale può essere utilizzato per inviare molti pagamenti avanti e indietro tra il portafoglio di Alice e quello del suo peer di canale.

Inoltre, il peer di canale di Alice può *inoltrare* ulteriormente i pagamenti tramite altri canali su LN . In questo modo, Alice può *instradare* un pagamento a qualsiasi portafoglio (ad esempio, il portafoglio Lightning di Bob) purché il portafoglio di Alice possa trovare una *strada* praticabile, letteralmente "saltando" da un canale all'altro, fino al portafoglio di Bob.

SUGGERIMENTO	Non tutti i peer di canale sono *buoni* peer per l'instradamento dei pagamenti. I peer ben collegati saranno in grado di instradare i pagamenti su percorsi più brevi verso la destinazione, aumentando le possibilità di successo. I peer di canale con ampie capacità di fondi saranno in grado di instradare pagamenti più grandi.

In altre parole, Alice ha bisogno di uno o più canali che la colleghino a uno o più altri nodi sulla Rete Lightning. Non ha bisogno di un canale per collegare il suo portafoglio direttamente al Bob's Cafe per inviare a Bob un pagamento, sebbene possa anche scegliere di aprire un canale diretto. Qualsiasi nodo su Lightning Network può essere utilizzato per il primo canale di Alice. Più un nodo è ben connesso, più persone può raggiungere Alice. In questo esempio, poiché vogliamo dimostrare anche l'instradamento dei pagamenti, non faremo in modo che Alice apra un canale direttamente al portafoglio di Bob. Invece, faremo in modo che Alice apra un canale verso un nodo ben connesso e successivamente utilizzi quel nodo per inoltrare il suo pagamento, instradandolo attraverso qualsiasi altro nodo necessario per raggiungere Bob.

All'inizio non ci sono canali aperti, quindi come vediamo nella Figura 2-7, la scheda LIGHTNING CHANNELS mostra un elenco vuoto. Se noti, nell'angolo in basso a destra c'è un simbolo più (+), che è un pulsante per aprire un nuovo canale.

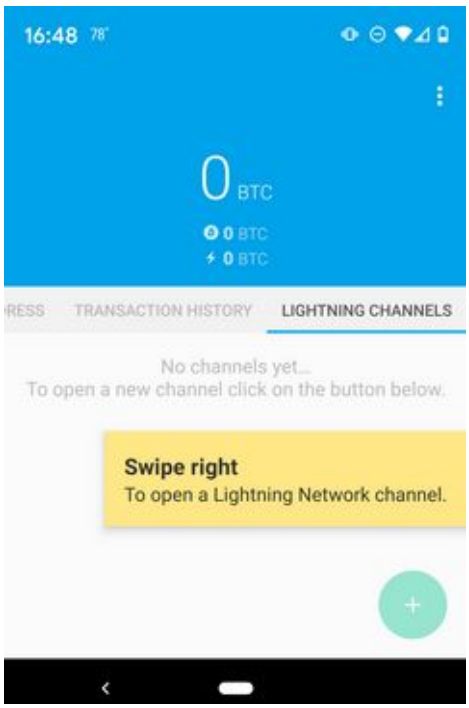

Figura 2-7. Scheda LIGHTNING CHANNELS

Alice preme il simbolo +, che presenta quattro possibili modi per aprire un canale:

- Incollando l'URI di un nodo

- Analizza l'URI di un nodo
- Nodo casuale
- nodo ACINQ

Un "node URI" sta a significare un Universal Resource Identifier (URI), che identifica uno specifico nodo Lightning. Alice può incollare tale URI dai suoi appunti o scansionare un codice QR contenente le stesse informazioni. Un esempio di URI di nodo viene mostrato come codice QR nella Figura 2-8 e successivamente come stringa di testo.

Figura 2-8. Node URI visualizzato tramite un codice QR

02b8c29e92f4842d06c5f6a5e8c53be29f5a6cf4bfd3c4d57d12aae3b0f2d9e10c@51.178.82.201:9735

Sebbene Alice possa selezionare un nodo Lightning specifico o utilizzare l'opzione "Nodo casuale" per fare in modo che il portafoglio Eclair selezioni un nodo a caso, selezionerà l'opzione Nodo ACINQ per connettersi a uno dei nodi Lightning ben collegati di ACINQ.

La scelta del nodo ACINQ ridurrà leggermente la privacy di Alice, perché darà ad ACINQ la possibilità di vedere tutte le transazioni di Alice. Creerà anche un singolo punto di fallimento, poiché Alice avrà un solo canale e se il nodo ACINQ non è disponibile, Alice non sarà in grado di effettuare pagamenti. Per mantenere le cose semplici all'inizio, accetteremo questi compromessi. Nei capitoli successivi impareremo gradualmente come ottenere maggiore indipendenza e fare meno compromessi!

Alice seleziona ACINQ Node ed è pronta ad aprire il suo primo canale su Lightning Network.

Apertura di un Canale Lightning

Quando Alice seleziona un nodo per aprire un nuovo canale, le viene chiesto di selezionare quanti bitcoin vuole allocare su questo specifico canale. Nei capitoli successivi discuteremo le implicazioni di queste scelte, ma per ora Alice destinerà quasi tutti i suoi fondi al canale. Dal momento che dovrà pagare le commissione di transazione per aprire il canale, selezionerà un importo leggermente inferiore al suo saldo totale.[4]

Alice assegna 0.018 BTC dei suoi 0.020 BTC totali al suo canale e accetta la commissione predefinita, come mostrato nella Figura 2-9.

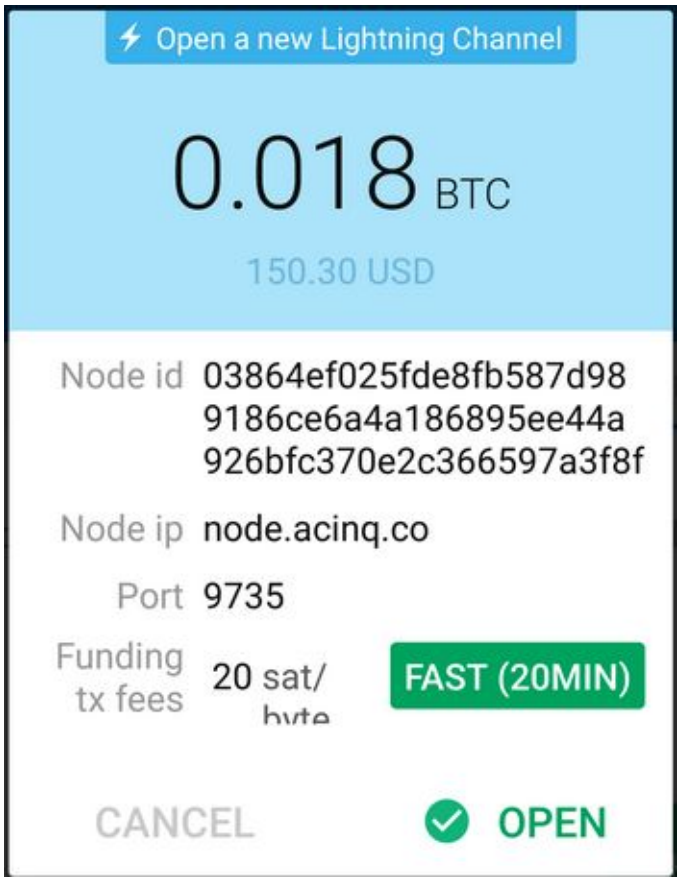

Figura 2.9. Apertura di un Canale Lightning

Dopo aver fatto clic su OPEN, il suo portafoglio crea la speciale transazione Bitcoin che apre un canale Lightning, noto come *funding transaction* (*transazione di finanziamento*). La transazione di finanziamento on-chain viene inviata alla rete Bitcoin per essere confermata.

Alice ora deve attendere nuovamente (vedi Figura 2-10) affinché la transazione venga registrata sulla blockchain di Bitcoin. Come per la transazione Bitcoin iniziale che ha utilizzato per acquisire i suoi primi bitcoin, deve attendere sei o più conferme (circa un'ora).

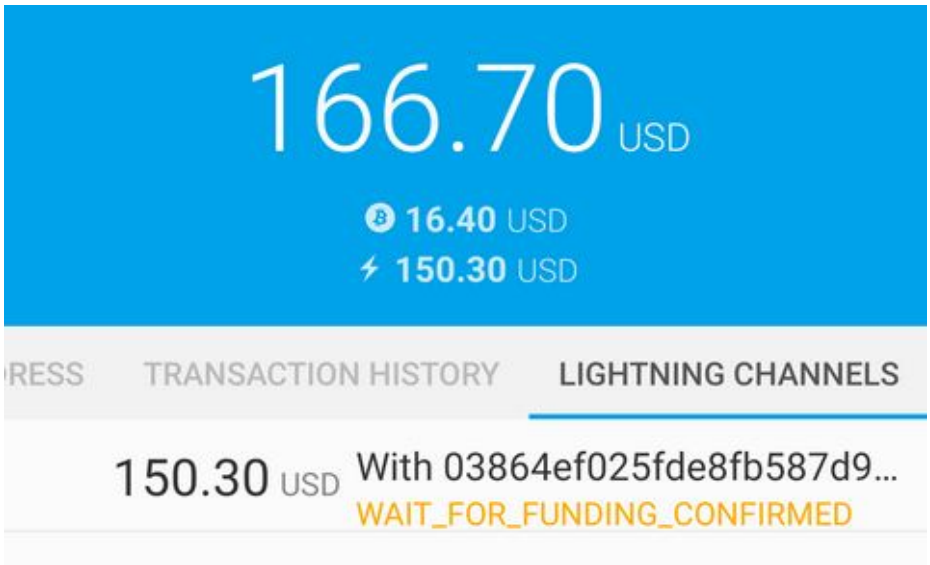

Figura 2-10. In attesa della funding transaction per aprire il canale su LN

Una volta confermata la transazione di finanziamento, il canale di Alice verso il nodo ACINQ è aperto, finanziato e pronto, come mostrato nella Figura 2-11.

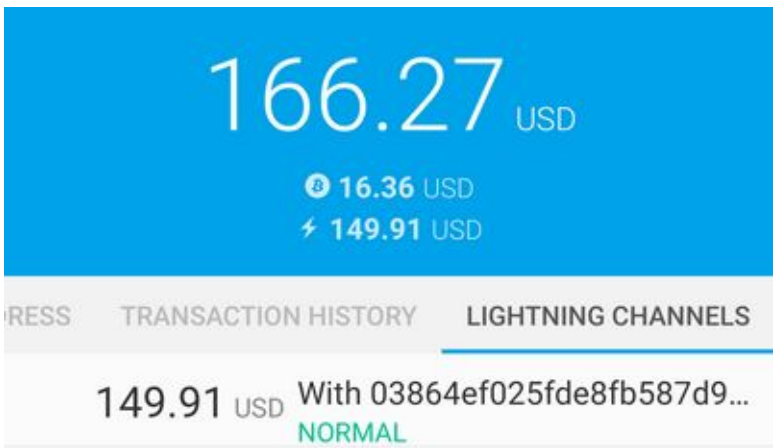

Figura 2-11. Il Canale è finalmente aperto

SUGGERIMENTO	Hai notato che l'importo del canale sembra essere cambiato? Non è così:

	il canale contiene 0.018 BTC, ma nel tempo tra gli screenshot il tasso di cambio di BTC è cambiato, quindi il valore in USD è diverso. Puoi scegliere di mostrare i saldi in BTC o USD, ma tieni presente che i valori in USD sono calcolati in tempo reale e cambieranno sempre!

Acquistare una Tazza di Caffè Utilizzando Lightning Network

Alice ora ha tutto pronto per iniziare a utilizzare Lightning Network. Come puoi vedere, c'è voluto un po' di lavoro e un po' di tempo in attesa di conferme. Tuttavia, ora le azioni successive sono veloci e facili. Lightning Network consente pagamenti senza dover attendere conferme, poiché i fondi vengono liquidati in pochi secondi.

Alice prende il suo dispositivo mobile e corre da Bob's Cafe. È entusiasta di provare il suo nuovo portafoglio Lightning e usarlo per comprare qualcosa!

Bob’s Cafe

Bob ha una semplice applicazione Point-of-Sale (PoS) per permettere a qualsiasi cliente che desidera pagare con bitcoin tramite LN, di poterlo fare. Come vedremo nel prossimo capitolo, Bob utilizza la piattaforma open source *BTCPay Server* che contiene i componenti necessari per una soluzione di e-commerce o vendita al dettaglio, come ad esempio:

- Un nodo Bitcoin che utilizza il software Bitcoin Core
- Un nodo Lightning che utilizza il software c-lightning
- Una semplice applicazione PoS su un tablet

BTCPay Server semplifica l'installazione di tutto il software necessario, carica immagini e prezzi dei prodotti e permette di avviare e gestire rapidamente un negozio offline o online.

Sul bancone del Bob's Cafe c'è un tablet che mostra ciò che vedi nella Figura 2-12.

Figura 2-12. il PoS del Bob's Cafe

Lightning Invoice

Alice seleziona l'opzione Cafe Latte dallo schermo e riceve una *Lightning Invoice* (nota anche come "*richiesta di pagamento*"), come mostrato nella Figura 2-13.

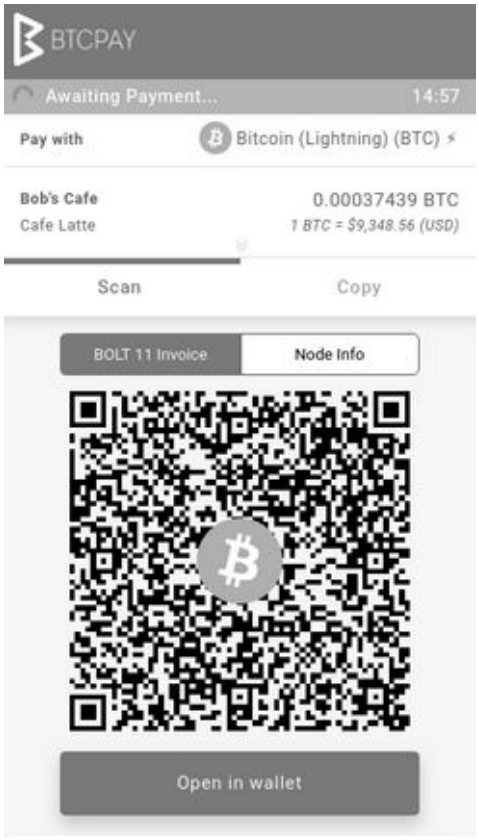

Figura 2-13. L'invoice Lightning per il pagamento del Cafe Latte

Per pagare la fattura, Alice apre il suo wallet Eclair e seleziona il pulsante Send (che assomiglia ad una freccia rivolta verso l'alto) nella scheda TRANSACTION HISTORY, come mostrato nella Figura 2-14.

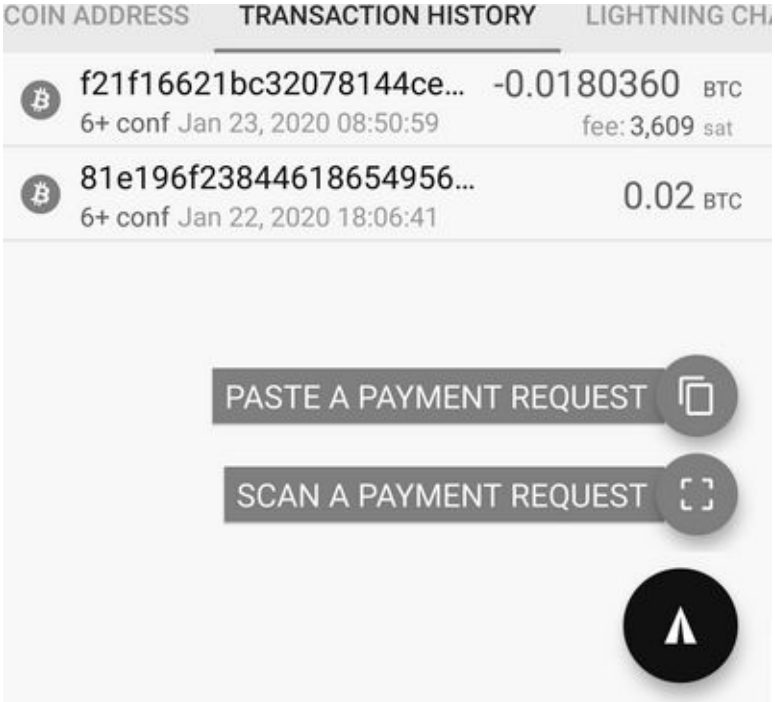

Figura 2-14. Alice clicca Send

SUGGERIMENTO	Il termine "richiesta di pagamento" può riferirsi ad una richiesta di pagamento Bitcoin o a una fattura Lightning, ed i termini "invoice/fattura" e "richiesta di pagamento" sono spesso usati in modo intercambiabile. Il termine tecnico corretto è "Lightning Invoice", indipendentemente da come viene denominata nel wallet.

Alice seleziona l'opzione per "scansionare una richiesta di pagamento", scansiona il codice QR visualizzato sullo schermo del tablet (vedi Figura 2-13) e le viene chiesto di confermare il suo pagamento, come mostrato nella Figura 2-15.

Alice preme PAY e, un secondo dopo, il tablet di Bob mostra un pagamento andato a buon fine. Alice ha completato il suo primo pagamento LN! È stato veloce, economico e facile. Ora può godersi il suo caffè latte che è stato acquistato utilizzando bitcoin attraverso un sistema di pagamento veloce, economico e decentralizzato. D'ora in poi Alice potrà semplicemente

selezionare un prodotto sullo schermo del tablet di Bob, scansionare il codice QR e fare clic su PAY e farsi servire un caffè, il tutto in pochi secondi e senza una transazione on-chain.

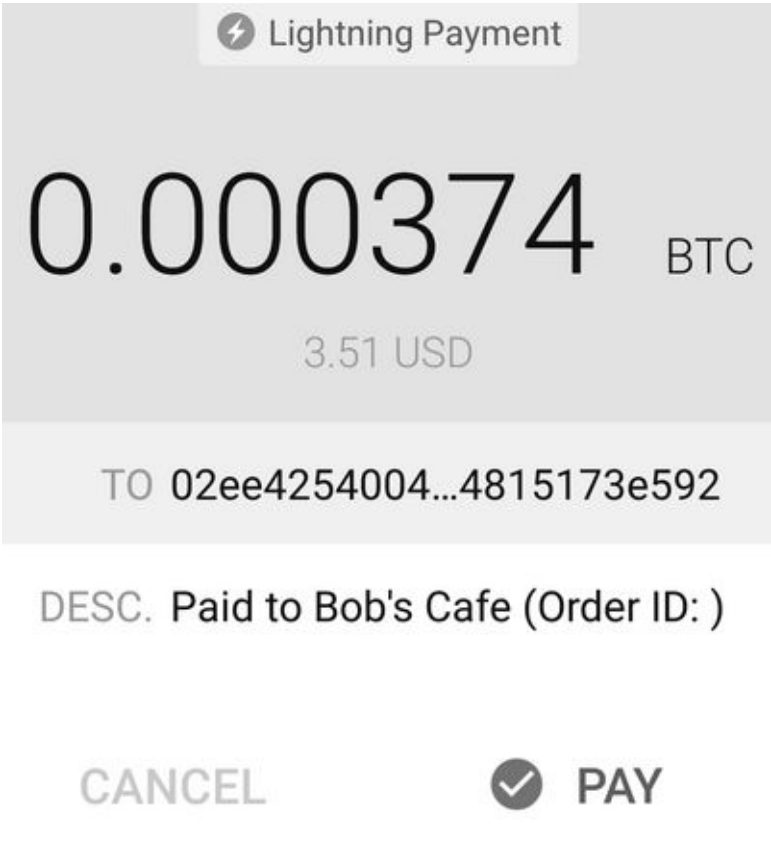

Figura 2-15. La conferma del pagamento di Alice a Bob

I pagamenti su LN sono migliori anche per Bob. È fiducioso che verrà pagato da Alice senza dover aspettare una conferma sulla blockchain. In futuro, ogni volta che Alice avrà voglia di bere un caffè al Bob's Cafe, potrà scegliere di pagare con bitcoin sulla rete Bitcoin o sulla rete Lightning. Quale pensi che sceglierà?

Conclusione

In questo capitolo, abbiamo seguito Alice mentre ha scaricato ed installato il suo primo wallet Lightning, acquisito e trasferito dei bitcoin, aperto il suo primo canale Lightning ed acquistato una tazza di caffè effettuando il suo primo pagamento su Lightning Network. Nei capitoli

seguenti, vedremo dettagliatamente il funzionamento di ogni componente di LN e come il pagamento di Alice ha raggiunto Bob's Cafe.

---

1. Riccardo Masutti - *Mastering Bitcoin: Traduzione italiana della guida completa al mondo di bitcoin e della blockchain*, Capitolo 1
2. ACINQ: sviluppatori del Eclair Mobile Lightning Wallet.
3. In genere non è consigliabile riutilizzare lo stesso indirizzo Bitcoin per più pagamenti perché tutte le transazioni Bitcoin sono pubbliche. Una persona curiosa potrebbe scansionare il codice QR di Alice e vedere quante transazioni Alice ha già ricevuto su questo indirizzo tramite la blockchain di Bitcoin. Fortunatamente, Lightning Network offre soluzioni più private, che discuteremo più avanti nel libro!
4. Il wallet Eclair non offre un'opzione per calcolare automaticamente le commissioni necessarie ed allocare l'importo massimo di fondi ad un canale, quindi Alice deve calcolarlo da sola.

Capitolo 3. Come Funziona Lightning Network

Ora che abbiamo seguito Alice mentre creava un portafoglio Lightning e acquistava un caffè da Bob, daremo un'occhiata nello specifico ai diversi componenti di Lightning Network coinvolti in quel processo. Questo capitolo fornirà una panoramica di alto livello e non approfondirà tutti i dettagli tecnici. L'obiettivo è quello di aiutarti ad acquisire consapevolezza sui concetti e sugli elementi costitutivi più importanti di Lightning Network.

Se hai esperienza in informatica, crittografia, Bitcoin e sviluppo di protocolli, allora questo capitolo dovrebbe essere sufficiente per essere in grado di comprenderlo pienamente. Se sei meno esperto, questo capitolo ti fornirà una panoramica abbastanza buona in modo che tu possa comprendere più facilmente le specifiche formali del protocollo, note come BOLT (Basis of Lightning Technology). Se sei un principiante, questo capitolo ti aiuterà a comprendere meglio i capitoli tecnici del libro.

Se hai bisogno di un ripasso sui fondamenti di Bitcoin, puoi trovare un riassunto nell’Appendice A:

- Chiavi e indirizzi
- Funzioni hash
- Firme digitali
- Struttura delle transazioni
- Input e output delle transazioni
- Concatenamento delle transazioni
- Bitcoin Script
- Indirizzi e script multifirma
- Timelock
- Script complessi

Inizieremo con una frase che definisce il concetto di Lightning Network e la analizzeremo nel resto di questo capitolo.

LN è una rete peer-to-peer di *canali di pagamento* implementati come smart contract (contratti intelligenti) sulla *blockchain di Bitcoin*, nonché un protocollo di comunicazione che definisce il modo in cui i partecipanti configurano ed eseguono questi contratti intelligenti.

Che cos'è un Canale di Pagamento?

Esistono diversi modi per descrivere un canale di pagamento, a seconda del contesto. Cominciamo da un livello alto e poi aggiungiamo qualche dettaglio in più.

Un canale di pagamento è una *relazione finanziaria* tra due nodi su Lightning Network, chiamati *partner di canale*. La relazione finanziaria alloca un *saldo di fondi* (denominato in millisatoshi) tra i due partner di canale.

Il canale di pagamento è gestito da un *protocollo crittografico*, ovvero un processo predefinito basato sulla crittografia che viene utilizzato dai partner di canale per ridistribuire il saldo del canale a favore dell'uno o dell'altro. Il protocollo crittografico garantisce che un partner di canale non possa imbrogliare l'altro, in modo che i partner non debbano fidarsi l'uno dell'altro.

Il protocollo crittografico è stabilito dal finanziamento di un *indirizzo multifirma* 2-di-2 che richiede la cooperazione tra i due partner di canale ed impedisce ad entrambi di spendere i fondi unilateralmente.

Riassumendo: un canale di pagamento è una relazione finanziaria tra nodi, che alloca fondi da un indirizzo multifirma attraverso un protocollo crittografico rigorosamente definito.

Nozioni Base sui Canali di Pagamento

Alla base del canale di pagamento c'è semplicemente un indirizzo multifirma 2-di-2 sulla blockchain Bitcoin, del quale tu detieni una chiave ed il tuo partner di canale detiene l'altra.

Tu e il tuo partner di canale negoziate una sequenza di transazioni che spendono da questo indirizzo con più firme. Invece di trasmettere e registrare queste transazioni sulla blockchain di Bitcoin, entrambi le tenete segrete tra di voi, come se non fossero ancora spese (unspent).

L'ultima transazione in quella sequenza codifica il saldo del canale e definisce come tale saldo viene suddiviso tra te e il tuo partner di canale.

Pertanto, l'aggiunta di una nuova transazione a questa sequenza equivale a spostare una parte del saldo del canale da un partner di canale all'altro, senza che la rete Bitcoin debba esserne a conoscenza. Negoziando ogni nuova transazione, modificando quindi l'allocazione dei fondi nel canale, si revoca anche la transazione precedente, in modo che nessuna delle parti possa regredire ad uno stato precedente.

Ogni transazione nella sequenza utilizza il linguaggio di scripting di Bitcoin, e quindi la negoziazione di fondi tra te e il tuo partner di canale è gestita da uno smart contract Bitcoin. Il contratto intelligente è impostato per penalizzare un membro del canale se tenta di inviare uno stato del canale precedentemente revocato.

NOTA	Se disponi di una transazione non pubblicata da un indirizzo multifirma 2-di-2 che ti paga parte del saldo, una firma dell'altra parte garantisce che tu possa pubblicare in modo indipendente questa transazione in qualsiasi momento aggiungendo semplicemente la tua firma. La possibilità di mantenere una transazione parzialmente firmata, offline e non pubblicata, con la possibilità di pubblicare e possedere quel saldo in qualsiasi momento, è la base di Lightning Network.

Inoltro dei Pagamenti Attraverso i Canali

Una volta che diversi partecipanti dispongono di canali da una parte all'altra, i pagamenti possono anche essere "inoltrati" o "instradati" (routing) da un canale di pagamento all'altro impostando un *percorso* attraverso la rete che collega insieme diversi canali di pagamento.

Ad esempio, Alice può inviare denaro a Charlie, se Alice ha un canale con Bob e Bob ha un canale con Charlie.

Grazie al design di LN, è possibile estendere i contratti intelligenti che gestiscono il canale in modo che Bob non possa rubare i fondi che vengono inoltrati attraverso il suo canale.

Allo stesso modo in cui il contratto intelligente protegge i partner di canale in modo che non debbano fidarsi l'uno dell'altro, l'intera rete protegge i partecipanti in modo che possano inoltrare i pagamenti senza fidarsi di nessuno degli altri partecipanti.

Poiché i canali sono costruiti da indirizzi multifirma e le transazioni di aggiornamento del saldo sono transazioni Bitcoin prefirmate, tutta la fiducia necessaria per far funzionare Lightning Network proviene dalla fiducia nella rete decentralizzata di Bitcoin!

Tali innovazioni sono certamente le principali caratteristiche che hanno permesso la creazione di Lightning Network. Tuttavia, LN è molto più dei protocolli crittografici in cima al linguaggio Bitcoin Script. È un protocollo di comunicazione completo che consente ai partecipanti di scambiarsi messaggi Lightning per per trasferire bitcoin. Il protocollo di comunicazione definisce il modo in cui i messaggi vengono crittografati e scambiati.

Lightning Network utilizza anche un protocollo di gossip per distribuire informazioni pubbliche sui canali (topologia di rete) a tutti i partecipanti.

Alice necessita delle informazioni sulla topologia di rete per essere a conoscenza del canale tra Bob e Charlie, in modo da poter costruire un percorso verso Charlie.

Ultimo ma non meno importante, è necessario capire che Lightning Network non è altro che un'applicazione creata al di sopra di Bitcoin, che utilizza transazioni Bitcoin e linguaggio Bitcoin Script. Non esiste una "criptovaluta Lightning" o una "blockchain di Lightning". Al di là di tutte le primitive tecniche, il protocollo LN è un modo creativo per ottenere maggiori benefici da Bitcoin consentendo una quantità arbitraria di pagamenti istantanei con *settlement* istantanei senza la necessità di doversi fidare di nessuno tranne che di Bitcoin.

Canali di Pagamento

Come abbiamo visto nel capitolo precedente, Alice ha usato il suo software wallet per creare un canale di pagamento tra lei e un altro partecipante di LN.

Un canale è limitato solo da tre cose:

- Innanzitutto, il tempo impiegato da Internet per trasferire le poche centinaia di byte di dati che il protocollo richiede per spostare i fondi da un'estremità all'altra del canale
- In secondo luogo, la capacità del canale, ovvero la quantità di bitcoin impegnata nel canale quando viene aperto
- In terzo luogo, il limite massimo di dimensione di una transazione Bitcoin limita anche il numero di pagamenti incompleti (in corso) inoltrati che possono essere effettuati contemporaneamente su un canale

I canali di pagamento hanno alcune proprietà molto interessanti e utili:

- Poiché il tempo per aggiornare un canale è principalmente legato alla velocità di comunicazione di Internet, effettuare un pagamento su un canale di pagamento può essere quasi istantaneo.
- Se il canale è aperto, effettuare un pagamento non richiede la conferma dei blocchi Bitcoin. Infatti, fintanto che tu ed il tuo partner di canale seguite il protocollo, non è richiesta alcuna interazione con la rete Bitcoin o con chiunque altro.
- Il protocollo crittografico è costruito in modo tale che tra te e il tuo partner di canale sia necessaria poca o nessuna fiducia. Se il tuo partner non risponde o cerca di imbrogliarti, puoi chiedere al sistema di agire come un "tribunale", risolvendo il contratto intelligente che tu e il tuo partner avete concordato in precedenza.
- I pagamenti effettuati in un canale di pagamento sono noti solo a te e al tuo partner. In tal senso, guadagni privacy rispetto alla blockchain Bitcoin, dove ogni transazione è pubblica. Solo il saldo finale, che è la somma di tutti i pagamenti di quello specifico canale, diventerà visibile sulla blockchain di Bitcoin.

Bitcoin aveva circa cinque anni quando sviluppatori di talento capirono per la prima volta come si potevano costruire canali di pagamento bidirezionali, a vita indefinita ed instradabili, e ormai ci sono almeno tre diversi metodi conosciuti per raggiungere ciò.

Questo capitolo si concentrerà sul metodo di costruzione del canale descritto per la prima volta nel Lightning Network whitepaper di Joseph Poon e Thaddeus Dryja nel 2015. Questi sono noti come canali *Poon-Dryja* e sono il metodo di costruzione del canale attualmente utilizzato su Lightning Network. Gli altri due metodi proposti sono i canali *Duplex Micropayment*, introdotti da Christian Decker più o meno nello stesso periodo dei canali Poon-Dryja e dei canali *eltoo*, introdotti in "eltoo: A Simple Layer2 Protocol for Bitcoin" da Christian Decker, Rusty Russel e (coautore di questo libro) Olaoluwa Osuntokun nel 2018.

I canali eltoo hanno alcune proprietà interessanti e semplificano l'implementazione dei canali di pagamento. Tuttavia, i canali eltoo richiedono una modifica del linguaggio Bitcoin Script e pertanto non possono ancora essere implementati sulla rete principale.

Indirizzo Multifirma

I canali di pagamento si basano su indirizzi multifirma (multisig) 2-di-2.

In sintesi, in un indirizzo multifirma i bitcoin sono bloccati in modo tale che lo stesso richieda più firme per sbloccarlo e spenderlo. In un indirizzo multifirma 2-di-2, come quello utilizzato su LN, ci sono due firmatari partecipanti ed *entrambi* devono firmare per spendere i fondi.

Spiegheremo più dettagliatamente gli script e gli indirizzi multifirma nel resto del libro.

Transazione di Finanziamento

L'elemento fondamentale di un canale di pagamento è un indirizzo multifirma 2-di-2. Uno dei due partner di canale finanzierà il canale di pagamento inviando bitcoin all'indirizzo multifirma. Questa transazione è chiamata *transazione di finanziamento* (funding transaction) ed è registrata sulla blockchain di Bitcoin.[1]

Anche se la transazione di finanziamento è pubblica, non è ovvio che si tratti di un canale di pagamento Lightning finché non viene chiuso, a meno che il canale non sia pubblicizzato pubblicamente. I canali sono in genere annunciati pubblicamente dai nodi di routing che desiderano inoltrare i pagamenti. Tuttavia, esistono anche canali non pubblicizzati, e di solito sono creati da nodi mobile che non partecipano attivamente all'instradamento dei pagamenti degli altri utenti di LN. Inoltre, i pagamenti di canale non sono visibili a nessuno al di fuori dei partner di canale, né lo è la distribuzione del saldo del canale tra di loro.

L'importo depositato nell'indirizzo multifirma è chiamato *capacità del canale* e stabilisce l'importo massimo che può essere inviato attraverso il canale di pagamento. Tuttavia, poiché i fondi possono essere inviati avanti e indietro, la capacità del canale non è il limite massimo di quanto valore può fluire attraverso il canale. Questo perché, se la capacità del canale si esaurisce con i pagamenti in una sola direzione, può essere utilizzata per inviare nuovamente i pagamenti nella direzione opposta.

NOTA	I fondi inviati all'indirizzo con più firme nella transazione di finanziamento vengono talvolta definiti "bloccati in un canale Lightning". Tuttavia i fondi in un canale Lightning non sono "bloccati" ma piuttosto "liberati". I fondi del canale Lightning sono più liquidi di quelli sulla blockchain di Bitcoin, in quanto possono essere spesi più velocemente, in modo più economico e in modo più privato. Ci sono alcuni svantaggi nel trasferire fondi su LN (come la necessità di tenerli in un portafoglio "caldo"), ma l'idea di "bloccare i fondi" in Lightning è fuorviante.

Esempio di una cattiva procedura di apertura del canale

Se pensi attentamente agli indirizzi multifirma 2-di-2, ti renderai conto che mettere i tuoi fondi in un tale indirizzo sembrerebbe comportare dei rischi. Cosa succede se il tuo partner di canale si rifiuta di firmare una transazione per sbloccare i fondi? Sono bloccati per sempre? Diamo ora un'occhiata a quello scenario e a come il protocollo LN evita tutto ciò.

Alice e Bob vogliono creare un canale di pagamento. Ciascuno crea una coppia di chiavi pubblica/privata e poi si scambia le chiavi pubbliche. Ora possono costruire un multifirma 2-di-2 con le due chiavi pubbliche, costituendo la base per il loro canale di pagamento.

Successivamente, Alice costruisce una transazione Bitcoin inviando pochi mBTC all'indirizzo multifirma creato dalle chiavi pubbliche di Alice e Bob. Se Alice non esegue ulteriori passaggi e trasmette semplicemente questa transazione, deve fidarsi che Bob fornirà la sua firma per spendere dall'indirizzo multifirma. Bob, d'altra parte, ha la possibilità di ricattare Alice trattenendo la sua firma e negando ad Alice l'accesso ai suoi fondi.

Per evitare ciò, Alice dovrà creare una transazione aggiuntiva che spenda dall'indirizzo multifirma, rimborsando i suoi mBTC. Alice quindi fa firmare a Bob la transazione di rimborso *prima* di trasmettere la sua transazione di finanziamento alla rete Bitcoin. In questo modo, Alice può ottenere un rimborso anche se Bob scompare o non collabora.

La transazione di "rimborso" che protegge Alice è la prima di una classe di transazioni chiamate *transazioni di impegno* (commitment transactions).

Commitment Transaction

Una *transazione di impegno* è una transazione che paga a ciascun partner di canale il proprio saldo di canale e garantisce che i due partner non debbano fidarsi l'uno dell'altro. Firmando una transazione di impegno, ciascun partner di canale "si impegna" per il saldo corrente e offre all'altro partner di canale la possibilità di recuperare i propri fondi quando lo desidera.

Mantenendo una transazione di impegno firmata, ciascun partner di canale può ottenere i propri fondi anche senza la collaborazione dell'altro partner. Questo li protegge dalla scomparsa dell'altro partner, dal rifiuto di collaborare o dal tentativo di imbrogliare violando il protocollo del canale di pagamento.

La transazione di impegno che Alice ha preparato nell'esempio precedente era un rimborso del suo pagamento iniziale all'indirizzo multifirma. Più in generale, tuttavia, una transazione

di impegno divide i fondi del canale di pagamento, pagando i due partner di canale in base alla distribuzione (saldo) che detengono. All'inizio Alice trattiene tutto il saldo, quindi si tratta di un semplice rimborso. Ma man mano che i fondi fluiscono da Alice a Bob, si scambieranno firme per nuove transazioni di impegno che rappresentano la nuova distribuzione del saldo, con una parte dei fondi pagata ad Alice e una parte pagata a Bob.

Supponiamo che Alice apra un canale con una capacità di 100.000 satoshi con Bob. Inizialmente, Alice detiene 100.000 satoshi, equivalenti la totalità dei fondi nel canale. Ecco come funziona il protocollo del canale di pagamento:

- Alice crea una nuova coppia di chiavi private/pubbliche ed informa Bob che desidera aprire un canale tramite il messaggio `open_channel` (messaggio del protocollo LN).
- Bob crea a sua volta una nuova coppia di chiavi private/pubbliche e accetta un canale da Alice, inviando la sua chiave pubblica tramite il messaggio `accept_channel`.
- Alice in seguito crea una transazione di finanziamento dal suo wallet, che invia 100k satoshi all'indirizzo multifirma con uno script di blocco: `2 <PubKey Alice> <PubKey Bob> 2 CHECKMULTISIG`.
- Alice non trasmette ancora questa transazione di finanziamento, ma invia a Bob l'ID della transazione in un messaggio `funding_created` allegando la sua firma per la transazione di impegno di Bob.
- Sia Alice che Bob creano la loro versione di una transazione di impegno. Questa transazione spenderà dalla transazione di finanziamento e rimanderà tutti i bitcoin a un indirizzo controllato da Alice.
- Alice e Bob non hanno bisogno di scambiare queste transazioni di impegno, poiché ognuno di loro sa come sono costruite e può costruirle indipendentemente (perché hanno concordato un ordinamento canonico degli input e degli output). Devono solo scambiarsi le firme.
- Bob fornisce una firma per la transazione di impegno di Alice e la invia ad Alice tramite il messaggio `funding_signed`.
- Ora che le firme sono state scambiate, Alice trasmetterà la transazione di finanziamento alla rete Bitcoin.

Seguendo questo protocollo, Alice non rinuncia alla proprietà dei suoi 100.000 satoshi anche se i fondi vengono inviati a un indirizzo multifirma 2-di-2 per il quale Alice controlla solo una

chiave. Se Bob smettesse di rispondere ad Alice, lei sarebbe comunque in grado di trasmettere la sua transazione di impegno e ricevere indietro i suoi fondi. I suoi unici costi sono le commissioni per le transazioni on-chain. Finché segue il protocollo, questo è il suo unico rischio quando decide di aprire un canale.

Dopo questo scambio iniziale, le transazioni di impegno vengono create ogni volta che il saldo del canale cambia. In altre parole, ogni volta che viene inviato un pagamento tra Alice e Bob, vengono create nuove transazioni di impegno e vengono scambiate le firme. Ogni nuova transazione di impegno codifica l'ultimo saldo tra Alice e Bob.

Se Alice volesse inviare 30k satoshi a Bob, entrambi creerebbero una nuova versione delle loro transazioni di impegno, che ora pagherebbero 70k satoshi ad Alice e 30k satoshi a Bob. Codificando un nuovo saldo per Alice e Bob, le nuove transazioni di impegno sono il mezzo con cui un pagamento viene "inviato" attraverso il canale.

Ora che conosciamo le transazioni di impegno, diamo un'occhiata ad alcuni dei dettagli più sottili. Potresti notare che questo protocollo lascia la possibilità ad Alice o Bob di imbrogliare.

Barare con uno Stato Precedente

Quante transazioni di impegno detiene Alice dopo aver pagato 30.000 satoshi a Bob? Ne detiene due: quella originale che le paga 100k satoshi e quella più recente, che le paga 70k satoshi mentre altri 30k satoshi vanno a Bob.

Nel protocollo di canale che abbiamo visto finora, nulla impedisce ad Alice di pubblicare una precedente transazione di impegno. Un'Alice malevola potrebbe pubblicare la transazione di impegno che le garantisce 100.000 satoshi anziché 30.000. Poiché quella transazione di impegno è stata firmata anche da Bob, nulla può impedire ad Alice di trasmetterla.

È necessario un meccanismo per impedire ad Alice di pubblicare una vecchia transazione di impegno. Scopriamo quindi come ciò può essere ottenuto e come Lightning Network può funzionare senza richiedere alcuna fiducia tra Alice e Bob.

Poiché Bitcoin è resistente alla censura, nessuno può impedire a qualcuno di pubblicare una vecchia transazione di impegno. Per prevenire questa forma di imbroglio, le transazioni di impegno sono costruite in modo tale che se ne viene trasmessa una vecchia, l'imbroglione può essere punito. Rendendo la penalità abbastanza grande, creiamo un forte incentivo contro gli imbrogli, e questo rende il sistema sicuro.

La penalità funziona in modo tale da dare alla parte ingannata l'opportunità di reclamare il saldo dell'imbroglione. Se qualcuno tenta di imbrogliare trasmettendo una vecchia transazione di impegno in cui viene pagato un saldo superiore a quello dovuto, l'altra parte può reclamare sia il proprio saldo che quello dell'imbroglione. Chi imbroglia perde tutto.

SUGGERIMENTO	Potresti notare che se Alice prosciuga quasi completamente il saldo del suo canale, potrebbe provare a barare con pochi rischi. La penalità di Bob non sarebbe così dolorosa se il suo saldo del canale fosse basso. Per evitare ciò, il protocollo Lightning richiede a ciascun partner di canale di mantenere un saldo minimo nel canale (chiamato *riserva*) in modo da avere sempre "skin in the game".

Esaminiamo nuovamente lo scenario di costruzione del canale, aggiungendo un meccanismo di penalità per proteggerci dagli imbrogli:

- Alice crea un canale con Bob da 100.000 satoshi.
- Alice invia 30k satoshi a Bob.
- Alice cerca di truffare Bob dei suoi 30.000 satoshi pubblicando una vecchia transazione di impegno che rivendica per sé l'intero importo di 100.000 sats.
- Bob rileva la frode e punisce Alice prendendo per sé l'intero importo da 100.000 sats.
- Bob riceve 100k satoshi, guadagnando 70k satoshi per aver sorpreso Alice a barare.
- Alice rimane con 0 satoshi.
- Cercando di imbrogliare Bob per 30k satoshi, Alice perde i 70k satoshi che possedeva.

Con un forte meccanismo di penalità, Alice non ha la tentazione di imbrogliare pubblicando una vecchia transazione di impegno, poiché rischia di perdere l'intero saldo.

NOTA	Nel capitolo 12 di *Mastering Bitcoin*, Andreas Antonopoulos (il coautore di questo libro) afferma quanto segue: "Una caratteristica chiave di Bitcoin è che una volta che una transazione è valida, rimane valida e non scade. L'unico modo per annullare una transazione è spendendo due volte i suoi input con un'altra transazione prima che venga minata."

Ora che abbiamo capito *perché* è necessario un meccanismo di penalità e come impedisce le truffe, vediamo *come* funziona in dettaglio.

Di solito la transazione di impegno ha almeno due output e paga ciascun partner di canale. Lo modifichiamo per aggiungere un *ritardo temporale* (*timelock delay*) e un *segreto di revoca* (*revocation secret*) a uno dei pagamenti. Il timelock impedisce al proprietario dell'output di spenderlo immediatamente una volta che la transazione di impegno è inclusa in un blocco. Il segreto di revoca consente ad entrambe le parti di spendere immediatamente quel pagamento, aggirando il timelock.

Quindi, nel nostro esempio, Bob detiene una transazione di impegno che paga Alice *immediatamente*, ma il suo pagamento è ritardato e revocabile. Anche Alice detiene una transazione di impegno, ma la sua è l'opposto: paga immediatamente Bob ma il suo pagamento è ritardato e revocabile.

I due partner di canale detengono metà del segreto di revoca, in modo che nessuno dei due conosca l'intero segreto. Se condividono la loro metà, l'altro partner di canale ha il segreto completo e può usarlo per esercitare la condizione di revoca. Quando si firma una nuova transazione di impegno, ciascun partner di canale revoca l'impegno precedente fornendo all'altra parte la metà del segreto di revoca.

Nel resto del libro, esamineremo il meccanismo di revoca in modo più dettagliato ed apprenderemo i dettagli di come vengono costruiti e utilizzati i segreti di revoca.

In termini semplici, Alice firma la nuova transazione di impegno di Bob solo se Bob offre la sua metà del segreto di revoca per l'impegno precedente. Bob firma la nuova transazione di impegno di Alice solo se lei gli dà la sua metà del segreto di revoca dall'impegno precedente.

Ad ogni nuovo impegno si scambiano il segreto di "punizione" che consente loro di *revocare* di fatto l'operazione di impegno precedente rendendola non redditizia da trasmettere. In sostanza, distruggono la capacità di utilizzare vecchi impegni mentre firmano quelli nuovi. Intendiamo dire che mentre è ancora tecnicamente possibile utilizzare vecchi impegni, il meccanismo sanzionatorio lo rende economicamente irrazionale.

Il timelock è impostato su un numero di blocchi fino a 2.016 (circa due settimane). Se uno dei partner di canale pubblica una transazione di impegno senza cooperare con l'altro partner, dovrà attendere quel numero di blocchi (ad esempio, due settimane) per richiedere il proprio saldo. L'altro partner di canale può richiedere il proprio saldo in qualsiasi momento. Inoltre, se l'impegno da loro pubblicato è stato precedentemente revocato, il partner di canale può

anche rivendicare immediatamente il saldo della controparte malevola, aggirando il timelock e punendo il truffatore.

Il timelock è regolabile e può essere negoziato tra i partner. Di solito è più lungo per i canali di maggiore capacità e più corto per i canali piccoli, allineando gli incentivi al valore dei fondi.

Per ogni nuovo aggiornamento del saldo del canale, devono essere create e salvate nuove transazioni di impegno e nuovi segreti di revoca. Finché un canale rimane aperto, *tutti i segreti di revoca creati* per il canale devono essere conservati perché potrebbero essere necessari in futuro. Fortunatamente, i segreti sono piuttosto piccoli e sono solamente i partner di canale a doverli mantenere, non l'intera rete. Inoltre, a causa di un meccanismo intelligente utilizzato per derivare i segreti di revoca, abbiamo solo bisogno di memorizzare il segreto più recente, perché i segreti precedenti possono essere derivati da esso.

Tuttavia, la gestione e l'archiviazione dei segreti di revoca è una delle parti più elaborate dei nodi Lightning, che richiede agli operatori dei nodi di mantenere i backup.

NOTA	Tecnologie come le watchtower (torri di controllo) o la modifica del protocollo di costruzione del canale con il protocollo eltoo potrebbero essere strategie future per mitigare questi problemi e ridurre la necessità di segreti di revoca, transazioni punitive e backup del canale.

Alice può chiudere il canale in qualsiasi momento se Bob non risponde, reclamando la sua parte del saldo. Dopo aver pubblicato *l'ultima* transazione di impegno on-chain, Alice deve attendere la scadenza del timelock prima di poter spendere i suoi fondi. Come vedremo più avanti, c'è un modo più semplice per chiudere un canale senza aspettare, a patto che Alice e Bob siano entrambi online e collaborino per chiudere il canale con la corretta allocazione del saldo. Le transazioni di impegno memorizzate da ciascun partner di canale fungono da sicurezza, assicurando che non perdano fondi se c'è un problema con il proprio partner.

Annuncio del Canale

I partner di canale possono decidere di annunciare il proprio canale all'intera rete, rendendolo un canale *pubblico*. Per annunciare il canale, usano il protocollo di gossip di LN per comunicare agli altri nodi l'esistenza, la capacità e le commissioni del canale.

L'annuncio pubblico dei canali consente ad altri nodi di utilizzarli per l'instradamento dei pagamenti, generando così anche commissioni di instradamento per i partner di canale.

Al contrario, i partner di canale possono decidere di non annunciare il canale, rendendolo un canale *non annunciato*.

NOTA	Potresti sentire il termine "canale privato" usato per descrivere un canale non annunciato. Evitiamo di usare questo termine perché è fuorviante e crea un falso senso di privacy. Sebbene un canale non annunciato non sarà noto agli altri mentre è in uso, la sua esistenza e la sua capacità saranno rivelate quando il canale si chiude perché quei dettagli saranno visibili on-chain nella transazione di regolamento finale. La sua esistenza può anche trapelare in una varietà di altri modi, quindi evitiamo di chiamarlo "privato".

I canali non annunciati vengono ancora utilizzati per instradare i pagamenti, ma solo dai nodi che sono a conoscenza della loro esistenza o che ricevono "suggerimenti di instradamento" su un percorso che include un canale non annunciato.

Quando un canale e la sua capacità vengono annunciati pubblicamente utilizzando il protocollo gossip, l'annuncio può includere anche informazioni sul canale (metadati), come le tariffe di instradamento e la durata del timelock.

Quando nuovi nodi si uniscono a LN, raccolgono gli annunci di canale propagati tramite il protocollo gossip dai loro pari, costruendo una mappa del Network. Questa mappa può quindi essere utilizzata per trovare percorsi per i pagamenti, collegando i canali end-to-end.

Chiusura del Canale

Il modo migliore per chiudere un canale è... non chiuderlo! L'apertura e la chiusura dei canali richiede una transazione on-chain, che comporterà commissioni di transazione. È quindi meglio mantenere i canali aperti il più a lungo possibile. Puoi continuare a utilizzare il tuo canale per effettuare e inoltrare pagamenti, a condizione che tu disponga di capacità sufficiente sul tuo lato del canale. Ma anche se invii tutto il saldo all'altra estremità del canale, puoi utilizzare il canale per ricevere pagamenti dal tuo partner di canale. Questo concetto di usare un canale in una direzione e poi usarlo nella direzione opposta è chiamato

"ribilanciamento" e lo esamineremo più dettagliatamente in un altro capitolo. Ribilanciando un canale, questo può essere mantenuto aperto quasi indefinitamente e utilizzato per un numero sostanzialmente illimitato di pagamenti.

Tuttavia, a volte è auspicabile o necessario chiudere un canale. Per esempio:

- Vuoi ridurre il saldo trattenuto sui tuoi canali Lightning per motivi di sicurezza e desideri inviare fondi ad un "cold storage".
- Il tuo partner non risponde per molto tempo e non puoi più utilizzare il canale.
- Il canale non viene utilizzato spesso perché il tuo partner non è ben collegato ad altri nodi, quindi vuoi utilizzare i fondi per un altro canale con un nodo connesso meglio.
- Il tuo partner di canale ha violato il protocollo a causa di un bug del software o di proposito, costringendoti a chiudere il canale per proteggere i tuoi fondi.

Esistono tre modi per chiudere un canale di pagamento:

- Chiusura reciproca (il miglior modo)
- Chiusura forzata (il modo sbagliato)
- Violazione del protocollo (il peggior modo)

Ciascuno di questi metodi è utile per circostanze diverse, che esploreremo nelle prossime sezioni di questo capitolo. Ad esempio, se il tuo partner di canale è offline, non sarai in grado di seguire "il miglior modo" perché una chiusura reciproca non può essere effettuata senza un partner che collabora. Di solito, il tuo software Lightning selezionerà automaticamente il miglior meccanismo di chiusura disponibile in base alle circostanze.

Chiusura reciproca (il miglior modo)

La chiusura reciproca è quando entrambi i partner di canale concordano di chiudere un canale. Tale modalità è il modo migliore per chiudere un canale.

Quando decidi di voler chiudere un canale, il tuo nodo Lightning informerà il tuo partner di canale della tua intenzione. Sia il tuo nodo che il nodo del partner di canale lavorano insieme per chiudere il canale. Non verranno accettati nuovi tentativi di instradamento da parte di

entrambi i partner di canale e tutti i tentativi di instradamento in corso verranno risolti o rimossi dopo il timeout. La finalizzazione dei tentativi di instradamento richiede tempo, quindi anche una chiusura reciproca può richiedere del tempo per essere completata.

Una volta che non ci sono tentativi di instradamento in sospeso, i nodi cooperano per preparare una *transazione di chiusura*. Questa transazione è simile alla transazione di impegno: codifica l'ultimo saldo del canale, ma gli output NON sono gravati da un timelock.

Le commissioni di transazione on-chain per la transazione di chiusura sono pagate dal partner di canale che ha aperto il canale e non da colui che ha avviato la procedura di chiusura. Calcolando le eventuali commissioni on-chain, i partner di canale concordano la commissione appropriata ed entrambi firmano la transazione di chiusura.

Una volta che la transazione di chiusura è stata trasmessa e confermata dalla rete Bitcoin, il canale è effettivamente chiuso e ciascun partner di canale ha ricevuto la propria quota del saldo del canale. Nonostante il tempo di attesa, una chiusura reciproca è in genere più veloce di una chiusura forzata.

Chiusura forzata (il modo sbagliato)

Una chiusura forzata si verifica quando un partner di canale tenta di chiudere un canale senza il consenso dell'altro partner.

Questo di solito accade quando uno dei partner di canale non è raggiungibile, quindi non è possibile una chiusura reciproca. In questo caso, avvieresti una chiusura forzata per chiudere unilateralmente il canale e "liberarne" i fondi.

Per avviare una chiusura forzata, puoi semplicemente pubblicare l'ultima commitment transaction del tuo nodo. Dopotutto, è a questo che servono le transazioni di impegno: offrono una garanzia che non devi fidarti del tuo partner per recuperare il saldo del canale.

Dopo aver trasmesso l'ultima transazione di impegno alla rete Bitcoin ed è stata confermata, la stessa creerà due output spendibili, uno per te e uno per il tuo partner. Come abbiamo discusso in precedenza, la rete Bitcoin non ha modo di sapere se questa è stata la transazione di impegno più recente o una vecchia che è stata pubblicata per truffare il tuo partner. Quindi questa transazione di impegno darà un leggero vantaggio al tuo partner. Il partner che ha avviato la chiusura forzata avrà il suo output gravato da un timelock e l'output dell'altro partner sarà spendibile immediatamente. Nel caso in cui tu abbia trasmesso una

precedente transazione di impegno, il ritardo temporale offre al tuo partner l'opportunità di contestare la transazione utilizzando il segreto di revoca e punirti per aver imbrogliato.

Quando si pubblica una transazione di impegno durante una chiusura forzata, le commissioni on-chain saranno superiori rispetto a una chiusura reciproca per diversi motivi:

1. Quando la transazione di impegno è stata creata, i partner di canale non sapevano di quanto sarebbero state le commissioni on-chain nel momento futuro in cui la transazione sarebbe stata trasmessa. Poiché le commissioni non possono essere modificate senza modificare gli output della transazione di impegno (che richiede entrambe le firme) e poiché la chiusura forzata si verifica quando un partner di canale non è disponibile a firmare, gli sviluppatori del protocollo hanno deciso di essere molto generosi con la tariffa inclusa nelle operazioni di impegno. Può essere fino a cinque volte superiore a quanto suggerito dalle stime delle commissioni della blockchain al momento della negoziazione della transazione di impegno.
2. La transazione di impegno include output aggiuntivi per qualsiasi tentativo di instradamento in sospeso degli hash time-locked contracts (HTLC), il che rende la transazione di impegno più grande (in termini di byte) rispetto a una transazione di chiusura reciproca. Le transazioni più grandi comportano più commissioni.
3. Eventuali tentativi di instradamento in sospeso dovranno essere risolti on-chain, causando ulteriori transazioni on-chain.

NOTA	Gli hash time-locked contracts (HTLC) saranno trattati in dettaglio nel resto del libro. Per ora, supponi che si tratti di pagamenti instradati attraverso LN, piuttosto che pagamenti effettuati direttamente tra i due partner di canale. Questi HTLC vengono trasportati come output aggiuntivi nelle transazioni di impegno, aumentando così le dimensioni della transazione e le commissioni on-chain.

In generale, una chiusura forzata è sconsigliabile a meno che non sia assolutamente necessaria. I tuoi fondi saranno bloccati per un periodo più lungo e chi ha aperto il canale dovrà pagare commissioni più elevate. Inoltre, potresti dover pagare commissioni on-chain per interrompere o risolvere tentativi di instradamento anche se non hai aperto il canale.

Se conosci il partner di canale, potresti prendere in considerazione l'idea di contattare quella

persona o azienda per chiedere perché il suo nodo Lightning è inattivo e richiedere che lo riavvii in modo da poter ottenere una chiusura reciproca del canale.

Dovresti considerare una chiusura forzata solo come ultima strada.

Violazione del protocollo (il peggior modo)

Una violazione del protocollo si verifica quando il tuo partner di canale cerca di imbrogliarti, deliberatamente o meno, pubblicando una transazione di impegno obsoleta sulla blockchain di Bitcoin, essenzialmente avviando una chiusura forzata (disonesta).

Il tuo nodo deve essere online e tenere sotto osservazione i nuovi blocchi e le nuove transazioni sulla blockchain di Bitcoin per rilevarlo.

Poiché il pagamento del tuo partner di canale sarà gravato da un timelock, il tuo nodo ha del tempo per agire per rilevare una violazione del protocollo e pubblicare una *transazione di punizione (punishment transaction)* prima che scada il timelock.

Se rilevi correttamente la violazione del protocollo e applichi la sanzione, riceverai tutti i fondi nel canale, inclusi i fondi del tuo partner di canale.

In tale scenario, la chiusura del canale sarà rapida. Dovrai pagare commissioni on-chain per pubblicare la transazione di punizione, ma il tuo nodo può stimare le commissioni medie e non pagare più del dovuto. In genere vorrai pagare tariffe più elevate per garantire la conferma il prima possibile. Tuttavia, poiché alla fine riceverai tutti i fondi dell'imbroglione, è essenzialmente l'imbroglione che pagherà per questa transazione.

Se non riesci a rilevare la violazione del protocollo e il timelock scade, riceverai solo i fondi che ti sono stati assegnati dalla transazione di impegno pubblicata dal tuo partner. Tutti i fondi che hai ricevuto dopo tale verranno rubati dal tuo partner. Se ti è stato assegnato un saldo, dovrai pagare commissioni on-chain per riscuoterlo.

Come con una chiusura forzata, anche tutti i tentativi di instradamento in sospeso dovranno essere risolti nella transazione di impegno.

Una violazione del protocollo può essere eseguita più velocemente di una chiusura reciproca perché non aspetti nel negoziare una chiusura con il tuo partner, e più veloce di una chiusura forzata perché non devi aspettare che scada il tuo timelock.

La teoria dei giochi prevede che l'imbroglio non sia una strategia allettante perché è facile

individuare un imbroglione e tale rischia di perdere *tutti* i propri fondi. Inoltre, man mano che LN matura e le watchtower (torri di controllo) diventano più disponibili, gli imbroglioni saranno rilevabili da una terza parte anche se il partner di canale truffato è offline.

Pertanto, non consigliamo di imbrogliare. Raccomandiamo, tuttavia, a chiunque scopra un imbroglione, di punirlo prendendo i suoi fondi.

Quindi, come fai a cogliere un imbroglio o una violazione del protocollo nelle tue attività quotidiane? Lo fai eseguendo un software che monitora la blockchain pubblica di Bitcoin per le transazioni sulla blockchain che corrispondono a qualsiasi transazione di impegno per uno qualsiasi dei tuoi canali. Questo software è di tre tipi:

- Un nodo Lightning gestito correttamente, in esecuzione 24 ore su 24, 7 giorni su 7
- Un nodo watchtower a scopo unico che gestisci per osservare i tuoi canali
- Un nodo watchtower di terze parti che paghi per osservare i tuoi canali

Ricorda che la transazione di impegno ha un periodo di timeout specificato in un determinato numero di blocchi, fino a un massimo di 2016. Finché avvii il tuo nodo Lightning prima che venga raggiunto il periodo di timeout, rileverai tutti i tentativi di frode. Non è consigliabile correre questo tipo di rischio; è importante mantenere un nodo correttamente e tenerlo in esecuzione continuamente.

Invoice

La maggior parte dei pagamenti su Lightning Network inizia con una invoice (fattura), generata dal destinatario del pagamento. Nel nostro esempio precedente, Bob crea una fattura per richiedere un pagamento da Alice.

NOTA	C'è un modo per inviare un pagamento non richiesto senza una fattura, utilizzando una soluzione alternativa nel protocollo chiamata keysend. Lo esamineremo nel capitolo "Keysend: Pagamenti Spontanei".

Una fattura è una semplice istruzione di pagamento contenente informazioni come un identificatore di pagamento univoco (chiamato hash di pagamento), un destinatario, un importo e una descrizione testuale facoltativa.

La parte più importante della fattura è l'hash di pagamento, che consente al pagamento di viaggiare attraverso più canali in modo atomico. Atomico, in informatica, indica qualsiasi azione o cambiamento di stato completato con successo o per niente: non c'è possibilità di uno stato intermedio o di un'azione parziale. Nel Lightning Network, ciò significa che il pagamento percorre l'intero percorso o fallisce. Non può essere parzialmente completato in modo tale che un nodo intermedio sul percorso possa ricevere il pagamento e tenerlo. Non esiste un "pagamento parziale" o un "pagamento parzialmente riuscito".

Le fatture non vengono comunicate tramite LN. Vengono invece comunicate "al di fuori", utilizzando qualsiasi altro meccanismo di comunicazione. Questo è simile al modo in cui gli indirizzi Bitcoin vengono comunicati ai mittenti al di fuori della rete Bitcoin: come codice QR, tramite e-mail o messaggio di testo. Ad esempio, Bob può presentare una fattura Lightning ad Alice come codice QR, tramite e-mail o tramite qualsiasi altro canale di messaggistica.

Le fatture sono generalmente codificate come una lunga stringa con codifica *bech32* o come codice QR per essere scansionate da un portafoglio Lightning per smartphone. La fattura contiene la quantità di bitcoin richiesta e una firma del destinatario. Il mittente utilizza la firma per estrarre la chiave pubblica (nota anche come ID del nodo) del destinatario in modo che il mittente sappia dove inviare il pagamento.

Noti come questo contrasta con Bitcoin e come vengono usate termini e parole diverse? In Bitcoin, il destinatario passa un indirizzo al mittente. In Lightning, il destinatario crea una fattura e la invia al mittente. In Bitcoin, il mittente invia fondi a un indirizzo. In Lightning, il mittente paga una fattura e il pagamento viene indirizzato al destinatario. Bitcoin si basa sul concetto di "indirizzo" e Lightning è una rete di pagamenti basata sul concetto di "fattura". In Bitcoin, creiamo una "transazione", mentre in Lightning inviamo un "pagamento".

Hash del Pagamento e Preimage

La parte più importante della fattura è l'hash del pagamento. Durante la creazione della fattura, Bob eseguirà un hash di pagamento come segue:

1. Bob sceglie un numero casuale *r*. Questo numero casuale è chiamato *preimmagine* (*preimage*) *o pagamento segreto* (*payment secret*).
2. Bob usa SHA-256 per calcolare l'hash *H* di *r* chiamato *hash del pagamento* (payment hash): *H = SHA-256(r)*.

NOTA	Il termine *preimmagine* deriva dalla matematica. In qualsiasi funzione $y = f(x)$, l'insieme di input che produce un certo valore y è chiamato preimmagine di y. In questo caso, la funzione è l'algoritmo hash SHA-256 e qualsiasi valore r che produce l'hash H è chiamato preimage.

Non esiste un modo noto per trovare l'inverso di SHA-256 (ovvero, calcolare una preimmagine da un hash). Solo Bob conosce il valore r, quindi è il segreto di Bob. Ma una volta che Bob rivela r, chiunque abbia l'hash H può verificare che r sia il segreto corretto, calcolando SHA-256(r) e vedendo che corrisponde a H.

Il processo di pagamento di LN è sicuro solo se r viene scelto in modo completamente casuale e non prevedibile. Questa sicurezza si basa sul fatto che le funzioni hash non possono essere invertite o forzate e, pertanto, nessuno può trovare r a partire da H.

Metadati Aggiuntivi

Le fatture possono facoltativamente includere altri metadati utili come una breve descrizione testuale. Se un utente ha diverse fatture da pagare, l'utente può leggere la descrizione e ricordarsi di cosa tratta la fattura.

La fattura può anche includere alcuni *suggerimenti di instradamento* (*routing hints*), che consentono al mittente di utilizzare canali non annunciati per costruire un percorso verso il destinatario. I suggerimenti di instradamento possono essere utilizzati anche per suggerire canali pubblici, ad esempio canali noti al destinatario per avere una capacità in entrata sufficiente per instradare il pagamento.

Nel caso in cui il nodo Lightning del mittente non sia in grado di inviare il pagamento tramite LN, le fatture possono facoltativamente includere un indirizzo Bitcoin on-chain come fallback.

NOTA	Sebbene sia possibile "ripiegare" su una transazione on-chain, è spesso meglio aprire un nuovo canale. Se devi sostenere commissioni on-chain per effettuare un pagamento, potresti anche sostenere tali commissioni per aprire un canale ed effettuare il pagamento tramite Lightning. Dopo che il pagamento è stato effettuato, ti rimane un canale aperto che ha liquidità sul lato del destinatario e può essere utilizzato per instradare i

	pagamenti al tuo nodo Lightning in futuro. Tale transazione on-chain ti offre un pagamento e un canale per un utilizzo futuro.

Le fatture Lightning contengono una data di scadenza. Poiché il destinatario deve conservare la preimage *r* per ogni fattura emessa, è utile far scadere le fatture in modo che queste preimage non debbano essere conservate per sempre. Una volta che una fattura è scaduta o è stata pagata, il destinatario può scartare la preimage.

Consegna del Pagamento

Abbiamo visto come il destinatario crea una fattura che contiene un hash di pagamento. Questo hash di pagamento verrà utilizzato per spostare il pagamento attraverso una serie di canali di pagamento, dal mittente al destinatario, anche se non hanno un canale di pagamento diretto tra di loro.

A breve, ci immergeremo nelle idee e nei metodi utilizzati per effettuare un pagamento tramite Lightning Network e utilizzeremo tutti i concetti che abbiamo presentato finora.

Innanzitutto, diamo un'occhiata al protocollo di comunicazione di Lightning Network.

Il Protocollo di Gossip peer-to-peer

Come visto in precedenza, quando viene creato un canale di pagamento, i partner di canale hanno la possibilità di renderlo pubblico, annunciandone l'esistenza e i dettagli all'intera rete.

Gli annunci di canale vengono comunicati tramite un *protocollo di gossip* peer-to-peer. Un protocollo peer-to-peer è un protocollo di comunicazione in cui ciascun nodo si connette a una selezione casuale di altri nodi nella rete, solitamente su TCP/IP. Ogni nodo che è direttamente connesso (su TCP/IP) al tuo nodo è chiamato *peer*. Il tuo nodo, a sua volta, è uno dei loro peer. Quando diciamo che il tuo nodo è connesso ad altri peer, non intendiamo che hai canali di pagamento, ma solo che sei connesso tramite il protocollo gossip.

Dopo aver aperto un canale, un nodo può scegliere di inviare un annuncio del canale tramite il messaggio `channel_announcement` ai suoi peer. Ogni peer convalida le informazioni dal messaggio `channel_announcement` e verifica che la transazione di finanziamento sia

confermata sulla blockchain di Bitcoin. Dopo la verifica, il nodo inoltrerà il messaggio di gossip ai propri peer, che lo inoltreranno ai propri peer e così via, diffondendo l'annuncio su tutta la rete. Per evitare una comunicazione eccessiva, l'annuncio del canale viene inoltrato da ciascun nodo solo se non ha già inoltrato quell'annuncio in precedenza.

Il protocollo di gossip viene utilizzato anche per annunciare informazioni sui nodi noti con il messaggio `node_announcement`. Perché questo messaggio venga inoltrato, un nodo deve avere almeno un canale pubblico annunciato sul protocollo gossip, sempre per evitare un eccessivo traffico di comunicazione.

I canali di pagamento hanno vari metadati utili per altri partecipanti alla rete. Questi metadati vengono utilizzati principalmente per prendere decisioni di routing. Poiché i nodi potrebbero occasionalmente modificare i metadati dei propri canali, queste informazioni vengono condivise in un messaggio `channel_update`. Questi messaggi verranno inoltrati solo circa quattro volte al giorno (per canale) per evitare comunicazioni eccessive. Il protocollo gossip ha anche una varietà di query e messaggi per sincronizzare inizialmente un nodo con la rete o per aggiornare la vista del nodo dopo essere stato offline per un po'.

Una sfida importante per i partecipanti di LN è che le informazioni sulla topologia condivise dal protocollo di gossip sono solo parziali. Ad esempio, la capacità dei canali di pagamento viene condivisa sul protocollo gossip tramite il messaggio `channel_announcement`. Tuttavia, questa informazione non è utile quanto l'effettiva distribuzione della capacità in termini di equilibrio è conosciuta solo tra i due partner di canale. Un nodo può inoltrare solo la quantità di bitcoin che effettivamente possiede (saldo locale) all'interno di quel canale.

Il design di LN avrebbe potuto includere la condivisione delle informazioni sul bilanciamento dei canali e su una topologia precisa, tuttavia ciò non è stato fatto per diversi motivi:

- Per proteggere la privacy degli utenti, non comunica ogni transazione finanziaria e pagamento. Gli aggiornamenti del saldo del canale rivelerebbero che un pagamento è stato spostato attraverso il canale. Queste informazioni potrebbero essere correlate per rivelare tutte le fonti e le destinazioni di pagamento.
- Scalare l'importo dei pagamenti che possono essere effettuati con Lightning Network. Ricorda che LN è stato creato in primo luogo perché la notifica a tutti i partecipanti di ogni pagamento non permette di scalare. Pertanto, LN non può essere progettato in modo da condividere gli aggiornamenti del bilanciamento del canale tra i partecipanti.
- LN è un sistema dinamico. Cambia costantemente e frequentemente. Si aggiungono

nodi, si disattivano altri nodi, cambiano i saldi, ecc. Anche se tutto venisse comunicato, le informazioni sarebbero valide solo per un breve lasso di tempo. È un dato di fatto: le informazioni sono spesso obsolete quando vengono ricevute.

Esamineremo i dettagli del protocollo del gossip in un capitolo successivo.

Per ora, è importante solo sapere che esiste il protocollo di gossip e che viene utilizzato per condividere le informazioni sulla topologia di LN. Queste informazioni sulla topologia sono fondamentali per la consegna dei pagamenti attraverso la rete dei canali di pagamento.

Pathfinding e Routing

I pagamenti su Lightning Network vengono inoltrati lungo un *percorso* (*path*) fatto di canali che collegano un partecipante all'altro, dalla fonte del pagamento alla destinazione del pagamento. Il processo di ricerca di un percorso dall'origine alla destinazione è chiamato *ricerca del percorso* (*pathfinding*). Il processo di utilizzo di tale percorso per effettuare il pagamento è chiamato *instradamento* (*routing*).

NOTA	Una critica frequente verso LN è che il problema del routing non sia risolto, o addirittura che si tratti di un problema "irrisolvibile". In realtà, il routing è insignificante. Il pathfinding, d'altra parte, è un problema difficile. I due termini sono spesso confusi e devono essere chiaramente definiti per identificare quale problema stiamo tentando di risolvere.

Come vedremo, LN utilizza attualmente un protocollo *basato sull'origine* (*source-based*) per il pathfinding e un protocollo con *instradamento a cipolla* (*onion-routed*) per l'instradamento dei pagamenti. Basato sull'origine significa che il mittente del pagamento deve trovare un percorso attraverso la rete fino alla destinazione prevista. A cipolla significa che gli elementi del percorso sono stratificati (come una cipolla), con ogni strato crittografato in modo tale che possa essere visto solo da un nodo alla volta. Parleremo del routing a cipolla nella prossima sezione.

Source-Based Pathfinding

Se conoscessimo gli esatti saldi di ogni canale, potremmo facilmente calcolare un percorso di

pagamento utilizzando uno qualsiasi degli algoritmi standard di ricerca del percorso insegnati in qualsiasi classe di informatica. Inoltre, in questo modo, potremmo ottimizzare le commissioni pagate ai nodi per l'inoltro del pagamento.

Tuttavia, le informazioni sul saldo di tutti i canali non sono e non possono essere note a tutti i partecipanti alla rete. Abbiamo bisogno di strategie di pathfinding più innovative.

Con solo informazioni parziali sulla topologia di rete, l'individuazione del percorso è una vera sfida e la ricerca attiva è ancora in corso in questa parte del Lightning Network. Il fatto che il problema del pathfinding non sia "completamente risolto" su LN è un importante punto critico nei confronti di questa tecnologia.

NOTA	Una critica comune al pathfinding su LN è che sia irrisolvibile perché è equivalente al *problema del commesso viaggiatore* NP-completo (TSP), un problema fondamentale nella teoria della complessità computazionale. In effetti, il pathfinding su Lightning non è equivalente a TSP e rientra in una diversa classe di problemi. Risolviamo con successo questo tipo di problemi (pathfinding in grafi con informazioni incomplete) ogni volta che chiediamo a Google di darci indicazioni stradali con evitamento del traffico. Risolviamo con successo anche questo problema ogni volta che inoltriamo un pagamento su LN.

Pathfinding e routing possono essere implementati in diversi modi e più algoritmi di pathfinding e routing possono coesistere su Lightning Network, proprio come esistono più algoritmi di pathfinding e routing su Internet. Il pathfinding basato sull'origine è una delle tante soluzioni possibili e ha successo nell'attuale Lightning Network.

La strategia di ricerca dei percorsi attualmente implementata dai nodi Lightning consiste nel provare iterativamente i percorsi finché non ne viene trovato uno con liquidità sufficiente per inoltrare il pagamento. Questo è un processo iterativo di tentativi ed errori, fino a quando non si ottiene il successo o non si trova alcun percorso. L'algoritmo attualmente non risulta necessariamente nel percorso con le commissioni più basse. Anche se ciò non è ottimale e certamente può essere migliorato, questa strategia semplicistica funziona abbastanza bene.

Questo "sondaggio" viene eseguito dal nodo Lightning o dal portafoglio e non viene visto direttamente dall'utente. L'utente potrebbe rendersi conto che il sondaggio è in corso solo se il pagamento non viene completato all'istante.

NOTA	Su Internet, utilizziamo il protocollo Internet e un algoritmo di inoltro IP per inoltrare i pacchetti Internet dal mittente alla destinazione. Sebbene questi protocolli abbiano la proprietà di consentire agli host di trovare in modo collaborativo un percorso per il flusso di informazioni attraverso Internet, non possiamo riutilizzare e adottare questo protocollo per l'inoltro dei pagamenti su Lightning Network. A differenza di Internet, i pagamenti Lightning devono essere atomici e i saldi dei canali devono rimanere privati. Inoltre, la capacità del canale in Lightning cambia frequentemente, a differenza di Internet dove la capacità di connessione è relativamente statica. Questi vincoli richiedono quindi nuove strategie.

Ovviamente, la ricerca del percorso è banale se vogliamo pagare il nostro partner di canale diretto e abbiamo abbastanza fondi dalla nostra parte del canale. Negli altri casi, il nostro nodo usa le informazioni del protocollo di gossip per trovare il percorso. Ciò include canali di pagamento pubblici attualmente noti, nodi noti, topologia nota (come sono collegati i nodi), capacità del canale note e politiche tariffarie note impostate dai proprietari dei nodi.

Onion Routing

Lightning Network utilizza un *protocollo di instradamento a cipolla* simile alla famosa rete Tor (The Onion Router). Il protocollo di onion routing utilizzato in Lightning è chiamato *SPHINX Mix Format*,[2] che verrà spiegato in dettaglio in un capitolo successivo.

NOTA	L'onion routing di Lightning, SPHINX Mix Format, è simile al routing della rete Tor solo nel concetto, ma sia il protocollo che l'implementazione sono completamente diversi da quelli utilizzati nella rete Tor.

Un pacchetto di pagamento utilizzato per l'instradamento è chiamato "cipolla".[3]

Usiamo l'analogia della cipolla per seguire un pagamento instradato: nel suo percorso dal mittente del pagamento (pagante) alla destinazione del pagamento (beneficiario), la cipolla passa da nodo a nodo lungo il percorso. Il mittente costruisce l'intera cipolla, dal centro verso l'esterno. Innanzitutto, il mittente crea le informazioni di pagamento per il destinatario (finale) del pagamento e le crittografa con un livello di crittografia che solo il destinatario può

decrittografare. Quindi, il mittente avvolge quel livello con le istruzioni per il nodo nel percorso *immediatamente precedente al destinatario finale* e crittografa con un livello che solo quel nodo può decrittografare.

I livelli sono costruiti con istruzioni, lavorando a ritroso finché l'intero percorso non è codificato in livelli. Il mittente quindi consegna la cipolla completa al primo nodo del percorso, che può leggere solo il livello più esterno. Ogni nodo sbuccia uno strato, trova le istruzioni all'interno che rivelano il nodo successivo nel percorso e passa la cipolla. Poiché ogni nodo sbuccia uno strato, non può leggere il resto della cipolla. Tutto ciò che sa è da dove è appena arrivata la cipolla e dove andrà dopo, senza alcuna indicazione su chi sia il mittente originale o il destinatario finale.

Questo continua fino a quando la cipolla raggiunge la destinazione del pagamento (beneficiario). Quindi, il nodo di destinazione apre la cipolla e scopre che non ci sono ulteriori livelli da decrittografare e può leggere le informazioni di pagamento all'interno.

NOTA	A differenza di una vera cipolla, quando si sbuccia ogni strato, i nodi aggiungono un riempimento crittografato per mantenere la stessa dimensione della cipolla per il nodo successivo. Come vedremo, questo rende impossibile per qualsiasi nodo intermedio sapere qualcosa sulla dimensione (lunghezza) del percorso, quanti nodi sono coinvolti nell'instradamento, quanti nodi li hanno preceduti o quanti ne seguono. Ciò aumenta la privacy prevenendo banali attacchi di analisi del traffico.

Il protocollo di onion routing utilizzato in Lightning ha le seguenti proprietà:

- Un nodo intermedio può solo vedere su quale canale ha ricevuto la cipolla e su quale canale inoltrarla. Ciò significa che nessun nodo di instradamento può sapere chi ha avviato il pagamento ed a chi è destinato. Questa è la proprietà più importante, che si traduce in un elevato grado di privacy.
- Le cipolle sono abbastanza piccole da entrare in un singolo pacchetto TCP/IP e persino in un frame a livello di collegamento (ad esempio, Ethernet). Ciò rende notevolmente più difficile l'analisi del traffico dei pagamenti, aumentando ulteriormente la privacy.
- Le cipolle sono costruite in modo tale da avere sempre la stessa lunghezza indipendentemente dalla posizione del nodo di lavorazione lungo il percorso. Man

mano che ogni strato viene "sbucciato", la cipolla viene riempita con dati "spazzatura" crittografati per mantenere le stesse dimensioni della cipolla. Ciò impedisce ai nodi intermedi di conoscere la loro posizione nel percorso.

- Le cipolle hanno un HMAC (codice di autenticazione dei messaggi basato su hash) a ogni livello in modo che le manipolazioni delle cipolle siano prevenute e impossibili.
- Le cipolle possono avere fino a circa 26 passaggi/strati di cipolla. Ciò consente percorsi sufficientemente lunghi. L'esatta lunghezza del percorso disponibile dipende dalla quantità di byte allocati al payload di instradamento ad ogni hop.
- La crittografia della cipolla per ogni hop utilizza diverse chiavi di crittografia effimere. Se una chiave (in particolare, la chiave privata di un nodo) dovesse fuoriuscire in un determinato momento, un utente malintenzionato non potrebbe decifrarla. Le chiavi non vengono mai riutilizzate per ottenere una maggiore sicurezza.
- Gli errori possono essere rispediti dal nodo in errore al mittente originale, utilizzando lo stesso protocollo con instradamento onion. Le cipolle di errore sono indistinguibili dalle cipolle di instradamento verso osservatori esterni e nodi intermedi. L'instradamento degli errori abilita il metodo di "probing" (sondaggio) per tentativi ed errori utilizzato per trovare un percorso che abbia una capacità sufficiente per instradare correttamente un pagamento.

Il routing a cipolla sarà esaminato in dettaglio nel Capitolo 10.

Algoritmo di Inoltro del Pagamento

Una volta che il mittente di un pagamento trova un possibile percorso attraverso la rete e costruisce una cipolla, il pagamento viene inoltrato da ogni nodo nel percorso. Ogni nodo elabora uno strato della cipolla e lo inoltra al nodo successivo nel percorso.

Ogni nodo intermedio riceve un messaggio Lightning chiamato `update_add_htlc` con un hash di pagamento e una cipolla. Il nodo intermediario esegue una serie di passaggi, chiamati *algoritmo di inoltro del pagamento* (*payment forwarding algorithm*):

1. Il nodo decifra lo strato esterno della cipolla e controlla l'integrità del messaggio.
2. Conferma di poter soddisfare i suggerimenti di instradamento in base alle commissioni ed alla capacità disponibile sul canale in uscita.

3. Collabora con il suo partner sul canale in entrata per aggiornare lo stato dello stesso.
4. Aggiunge di dati spazzatura alla fine della cipolla per mantenerne una lunghezza costante (poiché ha rimosso alcuni dati) e mantenere la privacy della cipolla.
5. Segue i suggerimenti di instradamento per inoltrare il pacchetto "cipolla" modificato sul suo canale di pagamento in uscita inviando anche un messaggio `update_add_htlc` che include lo stesso hash di pagamento e la cipolla.
6. Collabora con il suo partner sul canale in uscita per aggiornare lo stato del canale.

Naturalmente, questi passaggi vengono interrotti se viene rilevato un errore, e viene inviato un messaggio di errore al mittente del messaggio `update_add_htlc`. Il messaggio di errore è formattato come cipolla e inviato all'indietro sul canale in entrata.

Man mano che l'errore si propaga all'indietro su ciascun canale lungo il percorso originario, i partner di canale rimuovono il pagamento in sospeso, annullando il pagamento nel modo opposto rispetto a quello iniziale.

Sebbene la probabilità di un errore di pagamento sia elevata se non si viene effettuato velocemente, un nodo non dovrebbe mai avviare un altro tentativo di pagamento lungo un percorso diverso prima che la cipolla ritorni con un errore. Il mittente pagherebbe due volte se entrambi i tentativi di pagamento alla fine andassero a buon fine.

Crittografia delle Comunicazioni Peer-to-Peer

Il protocollo LN è principalmente un protocollo peer-to-peer tra i suoi partecipanti. Come abbiamo visto nelle sezioni precedenti, ci sono due funzioni sovrapposte nella rete, che formano due reti logiche che insieme sono *Il Lightning Network*:

1. Un'ampia rete peer-to-peer che utilizza un protocollo di gossip per propagare informazioni sulla topologia, in cui i peer si connettono casualmente tra loro. I peer non hanno necessariamente canali di pagamento tra di loro, quindi non sono sempre partner di canale.
2. Una rete di canali di pagamento tra partner di canale. Anche i partner di canale comunicano e "fanno gossip" tra di loro sulla topologia, nel senso che sono nodi peer nel protocollo di gossip.

Tutte le comunicazioni tra peer vengono inviate tramite messaggi chiamati *messaggi*

Lightning. Questi messaggi sono tutti crittografati utilizzando un framework di comunicazioni crittografiche chiamato *Noise Protocol Framework*. NPF consente la costruzione di protocolli di comunicazione crittografici che offrono autenticazione, crittografia, segretezza di inoltro e privacy dell'identità. NPF viene utilizzato anche in una serie di popolari sistemi di comunicazione crittografati end-to-end come WhatsApp, WireGuard e I2P. Ulteriori informazioni sono disponibili sul sito web del Noise Protocol Framework.

L'uso di NPF in Lightning Network garantisce che ogni messaggio sulla rete sia autenticato e crittografato, aumentando la privacy della rete e la sua resistenza all'analisi del traffico, all'ispezione dei pacchetti e all'intercettazione degli stessi. Tuttavia, come effetto collaterale, questo rende lo sviluppo e il test del protocollo complicato perché non si può semplicemente osservare la rete con uno strumento di acquisizione dei pacchetti o di analisi della rete come Wireshark. Invece, gli sviluppatori devono utilizzare plug-in specializzati che decrittano il protocollo dal punto di vista di un nodo, come *lightning dissector*, un plug-in di Wireshark.

Pensieri sulla Fiducia

Finché una persona segue il protocollo e protegge il proprio nodo, non vi è alcun rischio di perdere fondi quando si partecipa al (Lightning) Network. Tuttavia, esiste il costo di pagamento delle commissioni on-chain quando si apre un canale. Qualsiasi costo dovrebbe precedere un vantaggio corrispondente. Nel nostro caso, la ricompensa per Alice nell'aver sostenuto il costo dell'apertura di un canale è che Alice può inviare e, dopo aver spostato una parte dei fondi all'altra estremità del canale, ricevere pagamenti in bitcoin su LN in qualsiasi momento; non solo, Alice può anche guadagnare commissioni in bitcoin inoltrando pagamenti per altre persone. Alice sa che Bob può chiudere il canale subito dopo l'apertura, con conseguenti costi di chiusura on-chain a carico di Alice. Alice dovrà porre della fiducia in Bob. Alice è stata al Bob's Cafe e chiaramente Bob è interessato a vendere il suo caffè, quindi Alice può fidarsi di Bob in tal senso. Ci sono vantaggi reciproci sia per Alice che per Bob. Alice decide che la ricompensa è sufficiente per sostenere il costo on-chain per la creazione di un canale con Bob. Al contrario, Alice non aprirà un canale a qualche sconosciuto che le ha semplicemente inviato un'e-mail chiedendole di aprire un nuovo canale.

Confronto con Bitcoin

Sebbene LN sia costruito su Bitcoin ed erediti molte delle sue caratteristiche e proprietà, ci sono differenze importanti cui gli utenti di entrambe le reti devono essere consapevoli.

Alcune di queste differenze sono terminologiche. Ci sono anche differenze architettoniche e differenze nell'esperienza utente. Nelle prossime sezioni esamineremo le differenze e le somiglianze, spiegheremo la terminologia e adatteremo le nostre aspettative.

Indirizzi Rispetto a Fatture, Transazioni Rispetto a Pagamenti

In un tipico pagamento tramite Bitcoin, un utente riceve un indirizzo Bitcoin (ad esempio, scansionando un codice QR su una pagina Web o ricevendolo in un messaggio istantaneo o tramite e-mail da un amico). Quindi usa il proprio portafoglio Bitcoin per creare una transazione ed inviare fondi a tale indirizzo.

Su LN, il destinatario di un pagamento crea una fattura. Una fattura Lightning può essere considerata analoga a un indirizzo Bitcoin. Il destinatario previsto fornisce la fattura Lightning al mittente come codice QR o stringa di caratteri, proprio come un indirizzo Bitcoin.

Il mittente utilizza il portafoglio Lightning per pagare la fattura, copiando il testo o scansionando il codice QR. Un pagamento Lightning è analogo a una "transazione" di Bitcoin.

Tuttavia, ci sono alcune differenze nell'esperienza utente. Un indirizzo Bitcoin è *riutilizzabile*. Gli indirizzi Bitcoin non scadono mai e se il proprietario dell'indirizzo detiene ancora le chiavi, i fondi detenuti all'interno sono sempre accessibili. Un mittente può inviare qualsiasi quantità di bitcoin a un indirizzo utilizzato in precedenza e un destinatario può pubblicare un singolo indirizzo statico per ricevere molti pagamenti. Sebbene ciò vada contro le migliori pratiche per motivi di privacy, è tecnicamente possibile e in effetti abbastanza comune.

In Lightning, tuttavia, ciascuna fattura può essere utilizzata solo una volta per un importo di pagamento specifico. Non puoi pagare di più o di meno, non puoi riutilizzare una fattura e la fattura ha un tempo di scadenza incorporato. In Lightning, un destinatario deve generare una nuova fattura per ogni pagamento, specificandone in anticipo l'importo. C'è un'eccezione a ciò, un meccanismo chiamato *keysend*, che esamineremo successivamente.

Selezione degli Output Rispetto alla Ricerca di un Percorso

Per effettuare un pagamento sulla rete Bitcoin, un mittente deve spendere uno o più output di transazione non spesi (UTXO). Se un utente ha più UTXO, lui (o meglio, il suo portafoglio) dovrà selezionare quale/i UTXO inviare. Ad esempio, un utente che effettua un pagamento di 1 BTC può utilizzare un singolo output con valore 1 BTC, due output con valore 0.25 BTC e 0.75 BTC, oppure quattro output con valore 0.25 BTC ciascuno.

Su Lightning, i pagamenti non richiedono il consumo di input. Invece, ogni pagamento si traduce in un aggiornamento del saldo del canale, ridistribuendolo tra i due partner. Il mittente vede uno "spostamento" del saldo del canale dalla propria estremità all'altra, verso il proprio partner di canale. I pagamenti Lightning utilizzano una serie di canali per instradare fondi dal mittente al destinatario. Ciascuno di questi canali deve avere una capacità sufficiente per instradare il pagamento.

Poiché molti possibili canali e percorsi possono essere utilizzati per effettuare un pagamento, la scelta di canali e percorsi da parte dell'utente Lightning è in qualche modo analoga alla scelta di UTXO da parte dell'utente Bitcoin.

Con tecnologie come i pagamenti multipath atomici (AMP, Atomic Multipath Payments) e i pagamenti multipart (MPP, MultiPart Payments), che esamineremo successivamente, diversi percorsi Lightning possono essere aggregati in un singolo pagamento atomico, proprio come diversi UTXO di Bitcoin possono essere aggregati in una singola transazione Bitcoin atomica.

Output di Resto su Bitcoin Rispetto a Nessun Resto su Lightning

Per effettuare un pagamento sulla rete Bitcoin, il mittente deve consumare uno o più output di transazione non spesi (UTXO). Gli UTXO possono essere spesi solo per intero; non possono essere divisi e parzialmente spesi. Quindi, se un utente desidera spendere 0.8 BTC, ma ha solamente un UTXO da 1 BTC, deve spendere l'intero UTXO da 1 BTC inviando 0.8 BTC al destinatario e 0.2 BTC a se stesso come resto. Il resto di di 0.2 BTC crea un nuovo UTXO chiamato "output di resto".

Su Lightning, la transazione di finanziamento spende un po' di UTXO di Bitcoin, creando un UTXO multifirma per aprire il canale. Una volta che i bitcoin sono bloccati all'interno di quel canale, porzioni di essi possono essere inviati avanti e indietro all'interno del canale, senza la necessità di creare alcun resto. Questo perché i partner di canale aggiornano semplicemente

il saldo del canale e creano un nuovo UTXO solo quando il canale viene chiuso utilizzando la transazione di chiusura del canale.

Commissioni di Mining Rispetto a Commissioni di Routing

Sulla rete Bitcoin, gli utenti pagano commissioni ai miner per includere le loro transazioni in un blocco. Queste commissioni vengono pagate al minatore che estrae quel particolare blocco. L'importo della commissione si basa sulla *dimensione* in *byte* che la transazione utilizza in un blocco, nonché sulla velocità con cui l'utente desidera che la transazione venga confermata. Poiché i miner in genere estraggono prima le transazioni più redditizie, un utente che desidera che la transazione venga estratta immediatamente pagherà una tariffa più *alta* per byte, mentre un utente che non ha fretta pagherà una tariffa più *bassa* per byte.

Su Lightning Network, gli utenti pagano commissioni ad altri utenti (nodi intermedi) per instradare i pagamenti attraverso i loro canali. Per instradare un pagamento, un nodo intermediario dovrà spostare i fondi in due o più canali di sua proprietà, oltre a trasmettere i dati per il pagamento del mittente. In genere, i nodi di instradamento addebiteranno al mittente una commissione in base al valore che spostano, avendo stabilito una *base fee* minima (una tariffa fissa per ogni pagamento) e una *fee rate* (una tariffa proporzionale al valore del pagamento). I pagamenti di valore più elevato costeranno quindi di più per l'instradamento e ciò forma un mercato di liquidità, in cui utenti diversi addebitano commissioni diverse per l'instradamento attraverso i loro canali.

Commissioni Variabili a Seconda della Congestione Rispetto alle Tariffe Annunciate

Sulla rete Bitcoin, i miner sono alla ricerca di profitti e in genere includono quante più transazioni possibile in un blocco, pur rimanendo all'interno della capacità del blocco chiamata *peso del blocco* (*block weight*).

Se ci sono più transazioni in coda (chiamata *mempool*) di quelle che possono stare in un blocco, inizieranno minando le transazioni che pagano le commissioni più alte per unità (byte) di *peso della transazione* (*transaction weight*). Pertanto, quando ci sono molte transazioni in coda, gli utenti devono pagare una fee più alta per essere inclusi nel blocco successivo, oppure devono aspettare fino a quando ci sono meno transazioni in coda. Ciò

porta naturalmente all'emergere di un mercato a pagamento in cui gli utenti pagano in base all'urgenza con cui hanno bisogno che la loro transazione sia inclusa nel blocco successivo.

La risorsa scarsa sulla rete Bitcoin è lo spazio nei blocchi. Gli utenti di Bitcoin competono per lo spazio di blocco e il mercato delle commissioni Bitcoin si basa sullo spazio di blocco disponibile. Le scarse risorse su LN sono la *liquidità del canale* (capacità di fondi disponibili per l'instradamento nei canali) e la *connettività del canale* (quanti nodi ben collegati possono raggiungere i canali). Gli utenti Lightning competono per capacità e connettività; pertanto, il mercato delle tariffe Lightning è guidato dalla capacità e dalla connettività.

Sulla rete Lightning, gli utenti pagano commissioni ai nodi che instradano i loro pagamenti. Instradare un pagamento, in termini economici, non è altro che fornire e assegnare capacità al mittente. Naturalmente, gli instradatori (router) che applicano tariffe inferiori per la stessa capacità saranno più attraenti. Pertanto esiste un mercato delle commissioni in cui i router sono in concorrenza tra loro per le tariffe che addebitano per instradare i pagamenti attraverso i loro canali.

Transazioni Pubbliche in Bitcoin Rispetto a Pagamenti Lightning Privati

Sulla rete Bitcoin, ogni transazione è pubblicamente visibile sulla blockchain. Sebbene gli indirizzi coinvolti siano pseudonimi e in genere non siano legati a un'identità, sono comunque visti e convalidati da ogni altro utente sulla rete. Inoltre, le società di sorveglianza blockchain raccolgono e analizzano questi dati in massa e li vendono a parti interessate come aziende private, governi e agenzie di intelligence.

I pagamenti su LN, invece, sono quasi completamente privati. In genere, solo il mittente e il destinatario sono pienamente consapevoli dell'origine, della destinazione e dell'importo oggetto di transazione in un determinato pagamento. Inoltre, il destinatario potrebbe anche non conoscere la fonte del pagamento. Poiché i pagamenti sono instradati via onion, gli utenti che instradano il pagamento sono a conoscenza solo dell'importo del pagamento e non possono determinare né l'origine né la destinazione.

In sintesi, le transazioni Bitcoin vengono trasmesse pubblicamente e archiviate per sempre. I pagamenti Lightning vengono invece eseguiti tra pochi peer e le informazioni su di essi vengono archiviate privatamente solo fino alla chiusura del canale. La creazione di strumenti di sorveglianza e analisi equivalenti a quelli utilizzati su Bitcoin sarà molto più difficile su LN.

Attesa di Conferme Rispetto A Liquidazioni Istantanee

Sulla rete Bitcoin, le transazioni vengono regolate solo dopo essere state incluse in un blocco, nel qual caso si dice che sono "confermate" in quel blocco. Man mano che vengono minati più blocchi, la transazione acquisisce più "conferme" ed è considerata più sicura.

Su Lightning Network, le conferme contano solo per l'apertura e la chiusura dei canali on-chain. Una volta che un'operazione di finanziamento ha raggiunto un numero adeguato di conferme (ad esempio 3), i partner di canale considerano il canale aperto. Poiché i bitcoin nel canale sono protetti da uno smart contract che gestisce quel canale, i pagamenti vengono regolati *istantaneamente* una volta ricevuti dal destinatario finale. In termini pratici, il regolamento istantaneo significa che i pagamenti richiedono solo pochi secondi per essere eseguiti e liquidati. Come con Bitcoin, i pagamenti Lightning non sono reversibili.

Infine, quando il canale viene chiuso, viene effettuata una transazione sulla rete Bitcoin; una volta confermata la transazione, il canale è considerato chiuso.

Invio di Importi Arbitrari Rispetto a Limitazioni di Capacità

Sulla rete Bitcoin, un utente può inviare qualsiasi quantità di bitcoin che possiede a un altro utente, senza limiti di capacità. Una singola transazione può teoricamente inviare fino a 21 milioni di bitcoin come pagamento.

Su Lightning Network, un utente può inviare a un partner di canale solo la quantità di bitcoin attualmente esistente sul proprio lato di un determinato canale. Ad esempio, se un utente possiede un canale con 0.4 BTC dalla sua parte e un altro canale con 0.2 BTC dalla sua parte, il massimo che può inviare con un pagamento è 0.4 BTC. Questo è vero indipendentemente dalla quantità di bitcoin che l'utente ha attualmente nel proprio portafoglio Bitcoin.

I pagamenti multiparte (MPP) sono una funzionalità che, nell'esempio precedente, consente all'utente di combinare entrambi i canali da 0.4 BTC e 0.2 BTC per inviare un massimo di 0.6 BTC con un unico pagamento. Parleremo dettagliatamente di MPP nei capitoli successivi.

Se il pagamento viene instradato, ogni nodo di instradamento lungo il percorso deve disporre di canali con capacità almeno pari all'importo del pagamento. Questo deve valere per ogni singolo canale attraverso il quale viene instradato il pagamento. La capacità di canale più bassa in un percorso imposta il limite per la capacità dell'intero percorso.

Pertanto, la capacità e la connettività sono risorse critiche e scarse su Lightning Network.

Incentivi per Pagamenti di Importo Elevato Rispetto a Pagamenti di Importo Ridotto

La struttura delle commissioni in Bitcoin è indipendente dal valore della transazione. Una transazione da 1 milione di dollari ha la stessa commissione di una transazione da $1 su Bitcoin, assumendo una dimensione della transazione simile in byte (più specificamente byte "virtuali" con SegWit - protocollo Segregated Witness). In Lightning la commissione è una commissione a base fissa più una percentuale del valore della transazione. Pertanto, su LN, la commissione di pagamento aumenta con il valore del pagamento. Queste strutture di commissioni opposte creano incentivi diversi e portano a un utilizzo diverso per quanto riguarda il valore della transazione. Una transazione di maggior valore sarà più economica su Bitcoin. Gli utenti preferiranno Bitcoin per transazioni di valore elevato. Allo stesso modo, dall'altra parte della scala, gli utenti preferiranno Lightning per transazioni di piccolo valore.

Usare la Blockchain Come Libro Mastro Rispetto a un Sistema Giudiziario

Sulla rete Bitcoin, ogni transazione viene registrata in un blocco sulla blockchain. La blockchain costituisce quindi una cronologia completa di ogni transazione dalla creazione di Bitcoin e un modo per controllare completamente ogni bitcoin esistente. Una volta che una transazione è inclusa nella blockchain, è definitiva. Pertanto, non possono sorgere controversie ed è inequivocabile quanti bitcoin siano controllato da un particolare indirizzo in un particolare punto della blockchain.

Su LN, il saldo in un canale in un determinato momento è noto solo ai due partner di canale e viene reso visibile al resto della rete solo quando il canale è chiuso. Quando il canale viene chiuso, il saldo finale del canale viene condiviso alla blockchain Bitcoin e ogni partner riceve la propria quota di bitcoin di quel canale. Ad esempio, se il saldo iniziale era di 1 BTC pagato da Alice e Alice ha effettuato un pagamento di 0.3 BTC a Bob, il saldo finale del canale sarà di 0.7 BTC per Alice e di 0.3 BTC per Bob. Se Alice tenta di imbrogliare inviando lo stato di apertura del canale alla blockchain con 1 BTC per Alice e 0 BTC per Bob, allora Bob può vendicarsi inviando il vero stato finale del canale, oltre a creare una transazione di penalità che gli dà tutti i bitcoin nel canale. Per Lightning Network, la blockchain di Bitcoin funge da sistema giudiziario. Come un giudice robotico, Bitcoin registra i saldi iniziali e finali di ciascun canale e approva sanzioni se una delle parti cerca di imbrogliare.

Offline Rispetto Online, Asincrono Rispetto Sincrono

Quando un utente Bitcoin invia fondi a un indirizzo, non è necessario che sappia nulla del destinatario. Il destinatario potrebbe essere offline o online e non è necessaria alcuna interazione tra mittente e destinatario. L'interazione è tra il mittente e la blockchain Bitcoin. Ricevere bitcoin sulla blockchain di Bitcoin è un'attività *passiva* e *asincrona* che non richiede alcuna interazione da parte del destinatario o che il destinatario sia online in qualsiasi momento. Gli indirizzi Bitcoin possono anche essere generati offline e non sono mai "registrati" con la rete Bitcoin. Solo spendere bitcoin richiede interazione.

In Lightning, il destinatario deve essere online per completare il pagamento prima che scada. Il destinatario deve gestire un nodo o avere qualcuno che esegue un nodo per suo conto (un custode di terze parti). Per la precisione entrambi i nodi, quello del mittente e quello del destinatario, devono essere online al momento del pagamento e devono coordinarsi. La ricezione di un pagamento Lightning è un'attività *attiva* e *sincrona* tra mittente e destinatario, senza la partecipazione della maggior parte della rete Lightning o della rete Bitcoin (ad eccezione degli eventuali nodi di instradamento intermedi).

La natura sincrona e sempre online di LN è probabilmente la più grande differenza nell'esperienza dell'utente, e questo spesso confonde gli utenti che sono abituati a Bitcoin.

Satoshi Rispetto a Millisatoshi

Sulla rete Bitcoin, l'unità più piccola è un *satoshi*, che non può essere ulteriormente suddiviso. Lightning è più flessibile e i nodi Lightning funzionano con i *millisatoshi* (millesimi di satoshi). Ciò consente di inviare piccoli pagamenti tramite Lightning. Un singolo pagamento in millisatoshi può essere inviato attraverso un canale di pagamento, un importo così piccolo che dovrebbe essere propriamente definito un *nanopagamento*.

L'unità di millisatoshi non può, ovviamente, essere stabilita sulla blockchain Bitcoin vista la granularità. Alla chiusura del canale, i saldi vengono arrotondati al satoshi più vicino. Nel corso della vita di un canale, sono possibili milioni di nanopagamenti a livello di millisatoshi. Lightning Network rompe la barriera dei micropagamenti.

Similarità tra Bitcoin e Lightning Network

Sebbene LN differisca da Bitcoin in molti modi, anche nell'architettura e nell'esperienza dell'utente, è costruito su Bitcoin e conserva molte delle caratteristiche principali di Bitcoin.

Unità Monetaria

Sia la rete Bitcoin che la rete Lightning utilizzano le stesse unità monetarie: bitcoin. I pagamenti Lightning utilizzano gli stessi bitcoin delle transazioni Bitcoin. Di conseguenza, poiché l'unità monetaria è la stessa, il limite monetario è lo stesso: meno di 21 milioni di bitcoin. Dei 21 milioni di bitcoin totali di Bitcoin, alcuni sono già assegnati a indirizzi multifirma 2-di-2 come parte dei canali di pagamento su Lightning Network.

Irreversibilità e Definitività dei Pagamenti

Sia le transazioni Bitcoin che i pagamenti Lightning sono irreversibili e immutabili. Non esiste alcuna operazione di "annullamento" o "storno". Come mittente devi agire in modo responsabile; ma anche come destinatario ti viene garantita la finalità delle tue transazioni.

Fiducia e Rischio di Controparte

Come con Bitcoin, Lightning richiede all'utente solo di fidarsi della matematica, della crittografia e che il software non abbia bug critici. Né Bitcoin né Lightning richiedono all'utente di fidarsi di una persona, un'azienda, un'istituzione o un governo. Poiché Lightning si trova sopra Bitcoin e fa affidamento su Bitcoin come livello di base sottostante, è chiaro che il modello di sicurezza di Lightning si riduce alla sicurezza di Bitcoin. Ciò significa che Lightning offre sostanzialmente la stessa sicurezza di Bitcoin nella maggior parte dei casi, con solo una leggera riduzione della sicurezza in alcune circostanze ristrette.

Operazione Senza Autorizzazione

Sia Bitcoin che Lightning possono essere utilizzati da chiunque abbia accesso a Internet e al software appropriato, ad esempio nodo e portafoglio. Nessuna delle due reti richiede agli utenti di ottenere il permesso, il controllo o l'autorizzazione da terze parti, aziende, istituzioni

o un governo. I governi possono mettere fuori legge Bitcoin o Lightning all'interno della loro giurisdizione, ma non possono impedirne l'uso globale.

Open Source e Sistema Aperto

Sia Bitcoin che Lightning sono sistemi software open source creati da una comunità globale decentralizzata di volontari, con licenze aperte. Entrambi sono basati su protocolli aperti e interoperabili che operano come sistemi aperti e reti aperte. Globali, aperti e gratuiti.

Conclusione

In questo capitolo abbiamo visto come funziona effettivamente LN e tutti i componenti che lo costituiscono. Abbiamo esaminato ogni passaggio nella costruzione, gestione e chiusura di un canale. Abbiamo esaminato come vengono instradati i pagamenti e, infine, abbiamo confrontato Lightning con Bitcoin e analizzato le loro differenze e punti in comune.

Nei prossimi capitoli rivisiteremo tutti questi argomenti, ma in modo molto più dettagliato.

---

1. Mentre il whitepaper Lightning originale descriveva i canali finanziati da entrambi i partner di canale, la specifica attuale, a partire dal 2020, presuppone che un solo partner impegni fondi per il canale. A partire da maggio 2021, i canali Lightning a doppio finanziamento sono sperimentali nell'implementazione di LN c-Lightning.
2. George Danezis e Ian Goldberg, "Sphinx: A Compact and Provably Secure Mix Format", in *IEEE Symposium on Security and Privacy* (New York: IEEE, 2009), 269–282.
3. Il termine "cipolla" (onion) è stato originariamente utilizzato dal progetto Tor. Inoltre, la rete Tor è anche chiamata rete Onion ed il progetto utilizza una cipolla come logo. Il nome di dominio di primo livello utilizzato dai servizi Tor su Internet è onion.

Capitolo 4. Software di Lightning Network

Come abbiamo visto nei capitoli precedenti, un nodo Lightning è un sistema informatico che partecipa al Lightning Network. LN non è un prodotto o un'azienda; è un insieme di standard aperti che definiscono una linea di base per l'interoperabilità. Pertanto, il software del nodo Lightning è stato creato da una varietà di aziende e comunità. La maggioranza del software Lightning è *open source*, il che significa che il codice sorgente è aperto e concesso in licenza in modo da consentire la collaborazione, condivisione e partecipazione della comunità al processo di sviluppo. Allo stesso modo, le implementazioni del nodo Lightning che presenteremo in questo capitolo sono tutte open source e sono sviluppate in collaborazione.

A differenza di Bitcoin, dove lo standard è definito da un'*implementazione di riferimento* nel software (Bitcoin Core), in LN lo standard è definito da una serie di documenti standard chiamati *Basis of Lightning Technology* (BOLT), che si trovano nel *repository lightning-rfc*.

Non esiste un'implementazione di riferimento di Lightning Network, ma esistono diverse implementazioni concorrenti, conformi a BOLT e interoperabili. sviluppate da diversi team e organizzazioni. Anche i team che sviluppano software per Lightning Network contribuiscono allo sviluppo e all'evoluzione degli standard BOLT.

Un'altra importante differenza tra il software del nodo Lightning e il software del nodo Bitcoin è che i nodi Lightning non devono operare in sincronia con le regole di consenso e possono avere funzionalità estese oltre la linea di base dei BOLT. Pertanto, diversi team possono perseguire varie funzionalità sperimentali che, se hanno successo e vengono ampiamente implementate, possono diventare parte dei BOLT in seguito.

In questo capitolo imparerai come configurare ciascuno dei pacchetti software per le implementazioni dei nodi Lightning più popolari. Li abbiamo presentati in ordine alfabetico per sottolineare che generalmente non preferiamo o approviamo l'uno rispetto all'altro. Ognuno ha i suoi punti di forza e di debolezza e la scelta di uno dipenderà da una varietà di fattori. Poiché sono sviluppati in diversi linguaggi di programmazione (ad es. Go, C, ecc.), la tua scelta potrebbe dipendere anche dal tuo livello di familiarità ed esperienza con un linguaggio specifico e un set di strumenti di sviluppo.

Ambiente di Sviluppo Lightning

Se sei uno sviluppatore, vorrai configurare un ambiente di sviluppo con tutti gli strumenti, le librerie e il software di supporto per la scrittura e l'esecuzione del software Lightning. In questo capitolo altamente tecnico, illustreremo questo processo passo dopo passo. Se il materiale diventa troppo denso o non stai effettivamente impostando un ambiente di sviluppo, sentiti libero di passare al capitolo successivo, che è meno tecnico.

Utilizzo della Riga di Comando

Gli esempi in questo capitolo, e più in generale nella maggior parte di questo libro, utilizzano un terminale a riga di comando. Ciò significa che digiti i comandi in un terminale e ricevi risposte di testo. Inoltre, gli esempi sono dimostrati su un sistema operativo basato sul kernel Linux e sul sistema software GNU, in particolare l'ultima versione stabile a lungo termine di Ubuntu (Ubuntu 20.04 LTS). La maggior parte degli esempi è replicabile su altri sistemi operativi come Windows o macOS, con piccole modifiche ai comandi. La più grande differenza tra i sistemi operativi è il *gestore di pacchetti* che installa le varie librerie software e i loro prerequisiti. Negli esempi forniti, useremo `apt`, che è il gestore di pacchetti per Ubuntu. Su macOS, un gestore di pacchetti comune utilizzato per lo sviluppo open source è Homebrew, a cui si accede tramite il comando `brew`.

Nella maggior parte degli esempi, costruiremo il software direttamente dal codice sorgente. Anche se questo può essere piuttosto impegnativo, ci dà il massimo potere e controllo. Puoi scegliere di utilizzare contenitori Docker, pacchetti precompilati o altri meccanismi di installazione se rimani bloccato!

SUGGERIMENTO	In molti degli esempi di questo capitolo utilizzeremo l'interfaccia della riga di comando del sistema operativo (nota anche come *shell*), a cui si accede tramite un'applicazione *terminale*. La shell visualizzerà prima un prompt come indicatore che è pronta per il tuo comando. Quindi digiti un comando e premi il tasto Invio, a cui la shell risponde con del testo e un nuovo prompt per il tuo prossimo comando. Il prompt potrebbe avere un aspetto diverso sul tuo sistema, ma negli esempi seguenti è indicato dal simbolo $. Negli esempi, quando vedi del testo dopo un simbolo $, non digitarlo, ma digita il comando immediatamente

	successivo. Subito dopo premi Invio per eseguire il comando. Negli esempi, le righe che seguono ogni comando sono le risposte del sistema operativo a quel comando. Quando vedrai il prossimo prefisso $, sappi che si tratta di un nuovo comando e dovresti ripetere il processo.

Per mantenere le cose coerenti, usiamo la shell `bash` in tutti gli esempi della riga di comando. Mentre altre shell si comporteranno in modo simile e sarai in grado di eseguire tutti gli esempi senza di essa, alcuni degli script di shell sono scritti specificamente per la shell `bash` e potrebbero richiedere alcune modifiche o personalizzazioni per essere eseguiti in un'altra shell. Per coerenza, puoi installare la shell `bash` su Windows e macOS e viene installata per impostazione predefinita sulla maggior parte dei sistemi Linux.

Download del Repository del Libro

Tutti gli esempi di codice sono disponibili nel repository online del libro. Poiché il repository verrà mantenuto aggiornato il più possibile, dovresti sempre cercare l'ultima versione nel repository online invece di copiarla dal libro stampato o dall'ebook.

Puoi scaricare il repository come ZIP visitando GitHub e selezionando il pulsante Code.

In alternativa, puoi utilizzare il comando `git` per creare una copia dalla versione del repository sul tuo computer locale. Git è un sistema di controllo della versione distribuito utilizzato dalla maggior parte degli sviluppatori per collaborare allo sviluppo del software e tenere traccia delle modifiche ai repository software. Scarica e installa `git` seguendo le istruzioni del progetto Git.

Per creare una copia locale del repository sul tuo computer, esegui il comando git:

```
$ git clone https://github.com/lnbook/lnbook.git
```

Ora hai una copia completa del repository del libro in una cartella chiamata `lnbook`. Dovrai passare alla directory appena scaricata eseguendo:

```
$ cd lnbook
```

Tutti gli esempi successivi presumeranno che tu stia eseguendo i comandi dall'interno di questa cartella.

Contenitori Docker

Molti sviluppatori utilizzano un *container*, che è un tipo di macchina virtuale, per installare un sistema operativo preconfigurato e applicazioni con tutte le dipendenze necessarie. Gran parte del software Lightning può anche essere installato utilizzando un sistema di container come *Docker* che si trova nella home page di Docker. Le installazioni di container sono molto più semplici, soprattutto per coloro che non sono abituati a un ambiente a riga di comando.

Il repository del libro contiene una raccolta di contenitori Docker che possono essere utilizzati per impostare un ambiente di sviluppo consistente per esercitarsi e replicare gli esempi su qualsiasi sistema. Poiché il contenitore è un sistema operativo completo che viene eseguito con una configurazione consistente, puoi essere certo che gli esempi funzioneranno sul tuo computer senza doverti preoccupare di dipendenze, versioni della libreria o differenze nella configurazione.

I contenitori Docker sono spesso ottimizzati per essere piccoli, ovvero occupano lo spazio minimo su disco. Tuttavia, in questo libro utilizziamo i contenitori per *standardizzare* l'ambiente e renderlo uguale per tutti i lettori. Inoltre, questi contenitori non sono pensati per essere utilizzati per eseguire servizi in background. Invece, sono pensati per essere utilizzati per testare gli esempi e apprendere interagendo con il software. Per questi motivi, i contenitori sono piuttosto grandi e vengono forniti con molti strumenti e utilità di sviluppo. Comunemente, la distribuzione Alpine viene utilizzata per i container Linux a causa delle loro dimensioni ridotte. Tuttavia, forniamo contenitori basati su Ubuntu perché più sviluppatori hanno familiarità con Ubuntu e questa familiarità è più importante delle dimensioni.

L'installazione e l'uso di Docker e dei suoi comandi sono descritti in dettaglio in Appendice B. Se non hai familiarità con Docker, dovresti rivedere tale sezione.

Puoi trovare le ultime definizioni di container e creare configurazioni nel repository del libro nella cartella *code/docker*. Ogni contenitore si trova in una cartella separata, come puoi vedere di seguito:

```
$ tree -F --charset=asciii code/docker
code/docker
|-- bitcoind/
|   |-- bashrc
|   |-- bitcoind/
|   |   |-- bitcoin.conf
```

```
|   |   `-- keys/
|   |       |-- demo_address.txt
|   |       |-- demo_mnemonic.txt
|   |       `-- demo_privkey.txt
|   |-- bitcoind-entrypoint.sh
|   |-- cli
|   |-- Dockerfile
|   `-- mine.sh*
|-- c-lightning/
|   |-- bashrc
|   |-- cli
|   |-- c-lightning-entrypoint.sh
|   |-- devkeys.pem
|   |-- Dockerfile
|   |-- fund-c-lightning.sh
|   |-- lightningd/
|   |   `-- config
|   |-- logtail.sh
|   `-- wait-for-bitcoind.sh
|-- eclair/
|   |-- bashrc
|   |-- cli
|   |-- Dockerfile
|   |-- eclair/
|   |   `-- eclair.conf
|   |-- eclair-entrypoint.sh
|   |-- logtail.sh
|   `-- wait-for-bitcoind.sh
|-- lnd/
|   |-- bashrc
```

```
|   |-- cli
|   |-- Dockerfile
|   |-- fund-lnd.sh
|   |-- lnd/
|   |   `-- lnd.conf
|   |-- lnd-entrypoint.sh
|   |-- logtail.sh
|   `-- wait-for-bitcoind.sh
|-- check-versions.sh
|-- docker-compose.yml
|-- Makefile
`-- run-payment-demo.sh*
```

Come vedremo nelle prossime sezioni, puoi creare questi contenitori localmente oppure puoi estrarli dal repository del libro su Docker Hub. Le sezioni seguenti presumono che tu abbia installato Docker e che tu abbia familiarità con l'uso di base del comando `docker`.

Bitcoin Core e Regtest

La maggior parte delle implementazioni del nodo Lightning necessita dell'accesso a un full node Bitcoin per funzionare.

L'installazione di un nodo Bitcoin completo e la sincronizzazione della blockchain esulano dallo scopo di questo libro ed è di per sé un'impresa relativamente complessa. Se vuoi provarlo, fai riferimento a *Mastering Bitcoin*, "Capitolo 3: Bitcoin Core: L'implementazione di Riferimento", che discute l'installazione e il funzionamento di un nodo Bitcoin.

Un nodo Bitcoin può essere utilizzato in modalità `regtest`, in cui il nodo crea una blockchain locale a scopo di test. Negli esempi, utilizzeremo la modalità `regtest` per consentirci di dimostrare Lightning senza dover sincronizzare un nodo Bitcoin o rischiare fondi.

Il contenitore per Bitcoin Core è `bitcoind`. È configurato per eseguire Bitcoin Core in modalità `regtest` e per estrarre 6 nuovi blocchi ogni 10 secondi. La sua porta RPC (Remote Procedure Call) è esposta sulla porta 18443 ed è accessibile per le chiamate RPC con il nome utente `regtest` e la password `regtest`. Puoi anche accedervi con una shell interattiva ed

eseguire i comandi `bitcoin-cli` localmente.

Costruire il Bitcoin Core Container

Prepariamo il contenitore `bitcoind` estraendo l'ultimo contenitore da *Docker Hub*:

```
$ docker pull lnbook/bitcoind
Using default tag: latest
latest: Pulling from lnbook/bitcoind
35807b77a593: Pull complete
e1b85b9c5571: Pull complete
[...]
288f1cc78a00: Pull complete
Digest: sha256:861e7e32c9ad650aa367af40fc5acff894e89e47aff4bd400691ae18f1b550e2
Status: Downloaded newer image for lnbook/bitcoind:latest
docker.io/lnbook/bitcoind:latest
```

In alternativa, puoi creare tu stesso il contenitore dalla definizione del contenitore locale che si trova in *code/docker/bitcoind/Dockerfile*.

NOTA	Non è necessario creare il contenitore se in precedenza hai utilizzato il comando `pull` per estrarlo da Docker Hub.

La compilazione del contenitore in locale utilizzerà meno larghezza di banda, ma richiederà più tempo della CPU. Usiamo il comando `docker build` per compilarlo:

```
$ cd code/docker
$ docker run -it --name bitcoind lnbook/bitcoind
Starting bitcoind...
Bitcoin Core starting
Waiting for bitcoind to start
bitcoind started
==============================================
Imported demo private key
```

```
Bitcoin address:  2NBKgwSWY5qEmfN2Br4WtMDGuamjpuUc5q1
Private key:  cSaejkcWwU25jMweWEewRSsrVQq2FGTij1xjXv4x1XvxVRF1ZCr3
================================================
================================================
Balance: 0.00000000
================================================
Mining 101 blocks to unlock some bitcoin
[
  "34c744207fd4dd32b70bac467902bd8d030fba765c9f240a2e98f15f05338964",
  "64d82721c641c378d79b4ff2e17572c109750bea1d4eddbae0b54f51e4cdf23e",
 [...]
  "7a8c53dc9a3408c9ecf9605b253e5f8086d67bbc03ea05819b2c9584196c9294",
  "39e61e50e34a9bd1d6eab51940c39dc1ab56c30b21fc28e1a10c14a39b67a1c3",
  "4ca7fe9a55b0b767d2b7f5cf4d51a2346f035fe8c486719c60a46dcbe33de51a"
]
Mining 6 blocks every 10 seconds
Balance: 50.00000000
[
  "5ce76cc475e40515b67e3c0237d1eef597047a914ba3f59bbd62fc3691849055",
  "1ecb27a05ecfa9dfa82a7b26631e0819b2768fe5e6e56c7a2e1078b078e21e9f",
  "717ceb8b6c329d57947c950dc5668fae65bddb7fa03203984da9d2069e20525b",
  "185fc7cf3557a6ebfc4a8cdd1f94a8fa08ed0c057040cdd68bfb7aee2d5be624",
  "59001ae237a3834ebe4f6e6047dcec8fd67df0352ddc70b6b02190f982a60384",
  "754c860fe1b9e0e7292e1de96a65eaa78047feb4c72dbbde2a1d224faa1499dd"
]
```

Come puoi vedere, `bitcoind` si avvia ed estrae 101 blocchi simulati per avviare la catena. Questo perché secondo le regole di consenso Bitcoin, i bitcoin appena estratti non sono spendibili fino a quando non sono trascorsi 100 blocchi. Estraendo 101 blocchi, rendiamo spendibile la coinbase del primo blocco. Dopo quell'attività di mining iniziale, vengono estratti 6 nuovi blocchi ogni 10 secondi per far avanzare la catena.

Per ora non ci sono transazioni. Ma abbiamo alcuni bitcoin di prova che sono stati estratti nel

portafoglio ed sono disponibili per essere spesi. Quando colleghiamo alcuni nodi Lightning a questa catena, invieremo alcuni bitcoin ai loro portafogli in modo da poter aprire alcuni canali Lightning tra i nodi.

Interagire con il contenitore bitcoin core

Nel frattempo, possiamo anche interagire con il contenitore `bitcoind` inviandogli comandi shell. Il contenitore sta inviando un file di log al terminale e visualizzando il processo di mining del processo `bitcoind`. Per interagire con la shell possiamo impartire comandi in un altro terminale, utilizzando il comando `docker exec`. Poiché in precedenza abbiamo denominato il contenitore in esecuzione con l'argomento `name`, possiamo fare riferimento ad esso con quel nome quando eseguiamo il comando `docker exec`. Innanzitutto, eseguiamo una shell bash interattiva:

```
$ docker exec -it bitcoind /bin/bash
root@e027fd56e31a:/bitcoind# ps x
  PID TTY      STAT   TIME COMMAND
    1 pts/0    Ss+    0:00 /bin/bash /usr/local/bin/mine.sh
    7 ?        Ssl    0:03 bitcoind -datadir=/bitcoind -daemon
   97 pts/1    Ss     0:00 /bin/bash
  124 pts/0    S+     0:00 sleep 10
  125 pts/1    R+     0:00 ps x
root@e027fd56e31a:/bitcoind#
```

L'esecuzione della shell interattiva ci mette "dentro" il contenitore. Accede come utente `root`, come possiamo vedere dal prefisso `root@` nel nuovo prompt della shell `root@e027fd56e31a:/bitcoind#`. Se eseguiamo il comando ps x per vedere quali processi sono in esecuzione, vediamo che sia `bitcoind` che lo script `mine.sh` sono in esecuzione in background. Per uscire da questa shell, premi Ctrl-D o digita **`exit`** e verrai riportato al prompt del tuo sistema operativo.

Invece di eseguire una shell interattiva, possiamo anche emettere un singolo comando che viene eseguito all'interno del contenitore. Per comodità, il comando `bitcoin-cli` ha un alias "cli" che passa la configurazione corretta. Quindi eseguiamolo per chiedere a Bitcoin Core informazioni sulla blockchain. Eseguiamo `getblockchaininfo`:

```
$ docker exec bitcoind cli getblockchaininfo
{
  "chain": "regtest",
  "blocks": 131,
  "headers": 131,
  "bestblockhash":
"2cf57aac35365f52fa5c2e626491df634113b2f1e5197c478d57378e5a146110",
[...]
  "warnings": ""
}
```

Il comando `cli` nel contenitore `bitcoind` ci consente di inviare comandi RPC al nodo Bitcoin Core e ottenere risultati codificati JavaScript Object Notation (JSON).

Inoltre, tutti i nostri container Docker hanno un codificatore/decodificatore JSON della riga di comando preinstallato e denominato jq. jq ci aiuta a elaborare i dati in formato JSON tramite la riga di comando o da script interni. Puoi inviare l'output JSON di qualsiasi comando a `jq` utilizzando il carattere `|`. Questo carattere così come questa operazione è chiamato "pipe". Applichiamo un `pipe` e jq al comando precedente come segue:

```
$ docker exec bitcoind bash -c "cli getblockchaininfo | jq .blocks"
197
```

`jq .blocks` indica al decodificatore jq JSON di estrarre il campo `blocks` dal risultato `getblockchaininfo`. Nel nostro caso, mostra il valore di 197 che potremmo usare in un comando successivo.

Come vedrai nelle sezioni seguenti, possiamo eseguire diversi contenitori contemporaneamente e quindi interagire con essi singolarmente. Possiamo emettere comandi per estrarre informazioni come la chiave pubblica del nodo Lightning o per eseguire azioni come l'apertura di un canale Lightning verso un altro nodo. I comandi `docker run` e `docker exec`, insieme a `jq` per la decodifica JSON, sono tutto ciò di cui abbiamo bisogno per creare una rete Lightning funzionante che combini molte diverse implementazioni di nodi. Questo ci permette di provare diversi esperimenti sul nostro computer.

Il Progetto C-Lightning Lightning Node

`c-lightning` è un'implementazione leggera, altamente personalizzabile e conforme agli standard del protocollo LN, sviluppata da Blockstream come parte del progetto Elements. Il progetto è open source e sviluppato in collaborazione su GitHub.

Ora creeremo un contenitore Docker che esegue un nodo `c-lightning` che si collega al contenitore `bitcoind` che abbiamo creato in precedenza. Ti mostreremo anche come configurare e costruire il software `c-lightning` direttamente dal codice sorgente.

Compilare c-lightning come Container Docker

La distribuzione del software `c-lightning` ha un contenitore Docker, ma è progettata per l'esecuzione di `c-lightning` nei sistemi di produzione e assieme ad un nodo `bitcoind`. Useremo un contenitore configurato per eseguire `c-lightning` a scopo dimostrativo.

Estraiamo il contenitore `c-lightning` dal repository Docker Hub del libro:

```
$ docker pull lnbook/c-lightning
Using default tag: latest
latest: Pulling from lnbook/c-lightning
[...]
Digest: sha256:bdefcefe8a9712e7b3a236dcc5ab12d999c46fd280e209712e7cb649b8bf0688
Status: Downloaded image for lnbook/c-lightning:latest
docker.io/lnbook/c-lightning:latest
```

In alternativa, possiamo compilare il contenitore Docker `c-lightning` dai file del libro che hai precedentemente scaricato in una directory chiamata `lnbook`. Come prima, utilizzeremo il comando `docker build` nella sottodirectory `code/docker`. Taggheremo l'immagine del contenitore con il tag `lnbook/c-lightning`, in questo modo:

```
$ cd code/docker
$ docker build -t lnbook/c-lightning c-lightning
Sending build context to Docker daemon  91.14kB
Step 1/34 : ARG OS=ubuntu
Step 2/34 : ARG OS_VER=focal
```

```
Step 3/34 : FROM ${OS}:${OS_VER} as os-base
 ---> fb52e22af1b0
 [...]
Step 34/34 : CMD ["/usr/local/bin/logtail.sh"]
 ---> Running in 8d3d6c8799c5
Removing intermediate container 8d3d6c8799c5
 ---> 30b6fd5d7503
Successfully built 30b6fd5d7503
Successfully tagged lnbook/c-lightning:latest
```

Il nostro container è ora compilato e pronto per funzionare. Tuttavia, prima di eseguire il contenitore `c-lightning`, dobbiamo avviare il contenitore `bitcoind` in un altro terminale perché c-lightning dipende da `bitcoind`. Dovremo anche configurare una rete Docker che consenta ai container di connettersi tra loro come se risiedessero sulla stessa rete locale.

SUGGERIMENTO	I container Docker possono "parlare" tra loro su una rete locale virtuale gestita dal sistema Docker. Ogni contenitore può avere un nome personalizzato e altri contenitori possono utilizzare tale nome per risolvere il relativo indirizzo IP e connettersi facilmente ad esso.

Configurazione di una Rete Docker

Una volta configurata una rete Docker, Docker attiverà la rete sul nostro computer locale ogni volta che Docker si avvia, ad esempio dopo il riavvio. Quindi abbiamo solo bisogno di configurare una rete una sola volta usando il comando `docker network create`. Il nome della rete in sé non è importante, ma deve essere univoco sul nostro computer. Per impostazione predefinita, Docker ha tre reti denominate `host`, `bridge` e `none`. Chiameremo il nostro network `lnbook` e lo creeremo in questo modo:

```
$ docker network create lnbook
ad75c0e4f87e5917823187febedfc0d7978235ae3e88eca63abe7e0b5ee81bfb
$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
7f1fb63877ea        bridge              bridge              local
```

```
4e575cba0036       host         host         local
ad75c0e4f87e       lnbook       bridge       local
ee8824567c95       none         null         local
```

Come puoi vedere, l'esecuzione di `docker network ls` ci fornisce un elenco delle reti Docker. La nostra rete `lnbook` è stata creata. Possiamo ignorare l'ID di rete, perché viene gestito automaticamente.

Esecuzione dei Contenitori bitcoind e c-lightning

Il passaggio successivo consiste nell'avviare i contenitori `bitcoind` e `c-lightning` e connetterli alla rete `lnbook`. Per eseguire un contenitore in una rete specifica, dobbiamo passare l'argomento network a `docker run`. Per facilitare la ricerca reciproca dei contenitori, assegneremo a ciascuno anche un nome con l'argomento nome. Iniziamo `bitcoind` in questo modo:

```
$ docker run -it --network lnbook --name bitcoind lnbook/bitcoind
```

Dovresti vedere `bitcoind` avviarsi e iniziare a estrarre blocchi ogni 10 secondi. Lascialo in esecuzione e apri una nuova finestra di terminale per avviare `c-lightning`. Usiamo un comando `docker run` simile con gli argomenti network e name per avviare `c-lightning`:

```
$ docker run -it --network lnbook --name c-lightning lnbook/c-lightning
Waiting for bitcoind to start...
Waiting for bitcoind to mine blocks...
Starting c-lightning...
2021-09-12T13:14:50.434Z UNUSUAL lightningd: Creating configuration
directory /lightningd/regtest
Startup complete
Funding c-lightning wallet
8a37a183274c52d5a962852ba9f970229ea6246a096ff1e4602b57f7d4202b31
lightningd: Opened log file /lightningd/lightningd.log
lightningd: Creating configuration directory /lightningd/regtest
lightningd: Opened log file /lightningd/lightningd.log
```

Il contenitore `c-lightning` si avvia e si connette al contenitore `bitcoind` tramite la rete Docker. Ora il nostro nodo `c-lightning` aspetterà l'avvio di `bitcoind`, poi attenderà fino a

quando `bitcoind` non avrà minato dei bitcoin nel suo wallet. Infine, come parte dell'avvio del contenitore, uno script invierà un comando RPC al nodo `bitcoind`, che crea una transazione che finanzia il portafoglio `c-lightning` con 10 BTC di prova. Ora il nostro nodo `c-lightning` non è solo in esecuzione, ma ha anche alcuni bitcoin di prova con cui giocare!

Come abbiamo dimostrato con il contenitore `bitcoind`, possiamo inviare comandi al nostro contenitore `c-lightning` in un altro terminale per estrarre informazioni, aprire canali, ecc. Il comando che ci consente di inviare istruzioni da riga di comando al nodo `c-lightning` è chiamato `lightning-cli`. Questo comando `lightning-cli` è anche alias di `cli` all'interno di questo contenitore. Per ottenere le informazioni del nodo `c-lightning`, utilizza il seguente comando `docker exec` in un'altra finestra di terminale:

```
$ docker exec c-lightning cli getinfo
{
   "id": "026ec53cc8940df5fed5fa18f8897719428a15d860ff4cd171fca9530879c7499e",
   "alias": "IRATEARTIST",
   "color": "026ec5",
   "num_peers": 0,
   "num_pending_channels": 0,
[...]
   "version": "0.10.1",
   "blockheight": 221,
   "network": "regtest",
   "msatoshi_fees_collected": 0,
   "fees_collected_msat": "0msat",
   "lightning-dir": "/lightningd/regtest"
}
```

Ora abbiamo il nostro primo nodo Lightning in esecuzione su una rete virtuale e che comunica con una blockchain di Bitcoin di prova. Più avanti in questo capitolo avvieremo più nodi e li collegheremo tra loro per effettuare alcuni pagamenti Lightning.

Nella prossima sezione vedremo anche come scaricare, configurare e compilare `c-lightning` direttamente dal codice sorgente. Questo è un passaggio facoltativo e avanzato

che ti insegnerà ad utilizzare gli strumenti di compilazione e ti consentirà di apportare modifiche al codice sorgente `c-lightning`. Con questa conoscenza puoi scrivere del codice, correggere alcuni bug o creare un plug-in per `c-lightning`.

NOTA	Se non hai intenzione di immergerti nel codice sorgente o nella programmazione di un nodo Lightning, puoi saltare completamente la sezione successiva. Il contenitore Docker che abbiamo appena creato è sufficiente per la maggior parte degli esempi nel libro.

Installazione di c-lightning dal Codice Sorgente

Gli sviluppatori di `c-lightning` hanno fornito istruzioni dettagliate per compilare `c-lightning` dal codice sorgente. Seguiremo le istruzioni di GitHub.

Installazione di Librerie e Pacchetti Prerequisiti

Queste istruzioni di installazione presumono che tu stia compilando `c-lightning` su un sistema Linux o simile con gli strumenti di compilazione GNU. In caso contrario, cerca le istruzioni per il tuo sistema operativo nel repository di Elements Project.

Il primo passaggio comune è l'installazione delle librerie dei prerequisiti. Usiamo il gestore di pacchetti `apt` per fare ciò:

```
$ sudo apt-get update
Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:2 http://eu-north-1b.clouds.archive.ubuntu.com/ubuntu bionic InRelease
Get:3 http://eu-north-1b.clouds.archive.ubuntu.com/ubuntu bionic-updates
InRelease [88.7 kB]
[...]
Fetched 18.3 MB in 8s (2,180 kB/s)
Reading package lists... Done
$ sudo apt-get install -y \
  autoconf automake build-essential git libtool libgmp-dev \
  libsqlite3-dev python python3 python3-mako net-tools zlib1g-dev \
```

```
  libsodium-dev gettext
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  autotools-dev binutils binutils-common binutils-x86-64-linux-gnu cpp cpp-7
dpkg-dev fakeroot g++ g++-7 gcc gcc-7 gcc-7-base libalgorithm-diff-perl
 [...]
Setting up libsigsegv2:amd64 (2.12-2) ...
Setting up libltdl-dev:amd64 (2.4.6-14) ...
Setting up python2 (2.7.17-2ubuntu4) ...
Setting up libsodium-dev:amd64 (1.0.18-1) ...
[...]
$
```

Dopo pochi minuti e molta attività sullo schermo, avrai installato tutti i pacchetti e le librerie necessarie. Molte di queste librerie sono utilizzate anche da altri pacchetti Lightning e sono necessarie per lo sviluppo del software in generale.

Copia del Codice Sorgente di c-lightning

Successivamente, copiamo l'ultima versione di `c-lightning` dal repository del codice sorgente. Per fare ciò, utilizzeremo il comando `git clone`, che clona una copia controllata dalla versione sulla tua macchina locale, permettendoti così di mantenerla sincronizzata con le modifiche successive senza dover scaricare nuovamente l'intero repository:

```
$ git clone --recurse https://github.com/ElementsProject/lightning.git
Cloning into 'lightning'...
remote: Enumerating objects: 24, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 53192 (delta 5), reused 5 (delta 2), pack-reused 53168
Receiving objects: 100% (53192/53192), 29.59 MiB | 19.30 MiB/s, done.
```

```
Resolving deltas: 100% (39834/39834), done.
$ cd lightning
```

Ora abbiamo una copia di `c-lightning` clonata nella sottocartella *lightning* e abbiamo usato il comando `cd` (change directory) per entrare in quella sottocartella.

Compilazione del Codice Sorgente di c-lightning

Successivamente, usiamo una serie di *script di compilazione* disponibili in molti progetti open source. Questi script utilizzano i comandi configure e make, che ci consentono di:

- Selezionare opzioni di compilazione controllando dipendenze necessarie (`configure`)
- Compilare e installare gli eseguibili e le librerie (`make`)

L'esecuzione di `configure` con l'opzione `help` ci mostrerà tutte le opzioni disponibili:

```
$ ./configure --help
Usage: ./configure [--reconfigure] [setting=value] [options]
Options include:
  --prefix= (default /usr/local)
    Prefix for make install
  --enable/disable-developer (default disable)
    Developer mode, good for testing
  --enable/disable-experimental-features (default disable)
    Enable experimental features
  --enable/disable-compat (default enable)
    Compatibility mode, good to disable to see if your software breaks
  --enable/disable-valgrind (default (autodetect))
    Run tests with Valgrind
  --enable/disable-static (default disable)
    Static link sqlite3, gmp and zlib libraries
  --enable/disable-address-sanitizer (default disable)
    Compile with address-sanitizer
```

Non è necessario modificare nessuna delle impostazioni predefinite per questo esempio.

Eseguiamo nuovamente `configure` senza alcuna opzione per utilizzare i valori predefiniti:

```
$ ./configure
Compiling ccan/tools/configurator/configurator...done
checking for python3-mako... found
Making autoconf users comfortable... yes
checking for off_t is 32 bits... no
checking for __alignof__ support... yes
[...]
Setting COMPAT... 1
PYTEST not found
Setting STATIC... 0
Setting ASAN... 0
Setting TEST_NETWORK... regtest
$
```

Successivamente, utilizziamo il comando `make` per compilare le librerie, i componenti e gli eseguibili del progetto `c-lightning`. Questa parte richiederà diversi minuti per essere completata e utilizzerà pesantemente la CPU e il disco del tuo computer. Esegui `make`:

```
$ make
cc -DBINTOPKGLIBEXECDIR="\"../libexec/c-lightning\"" -Wall -Wundef -Wmis...
[...]
cc   -Og  ccan-asort.o ccan-autodata.o ccan-bitmap.o ccan-bitops.o ccan-...
```

Se tutto va bene non vedrai alcun messaggio `ERROR` che interrompe l'esecuzione del comando precedente. Il software `c-lightning` è stato compilato dal sorgente e ora siamo pronti per installare i componenti eseguibili che abbiamo creato nel passaggio precedente:

```
$ sudo make install
mkdir -p /usr/local/bin
mkdir -p /usr/local/libexec/c-lightning
mkdir -p /usr/local/libexec/c-lightning/plugins
mkdir -p /usr/local/share/man/man1
mkdir -p /usr/local/share/man/man5
```

```
mkdir -p /usr/local/share/man/man7
mkdir -p /usr/local/share/man/man8
mkdir -p /usr/local/share/doc/c-lightning
install cli/lightning-cli lightningd/lightningd /usr/local/bin
[...]
```

Per verificare che i comandi `lightningd` e `lightning-cli` siano stati installati correttamente, chiederemo a ciascun eseguibile le informazioni sulla sua versione:

```
$ lightningd --version
v0.10.1-34-gfe86c11
$ lightning-cli --version
v0.10.1-34-gfe86c11
```

La versione è costituita dall'ultima versione di rilascio (v0.10.1), seguita dal numero di modifiche dal rilascio (34) e infine da un hash che identifica la revisione (fe86c11). Potresti vedere una versione diversa da quella mostrata in precedenza poiché il software continua ad evolversi molto tempo dopo la pubblicazione di questo libro. Tuttavia, indipendentemente dalla versione visualizzata, il fatto che i comandi vengano eseguiti e rispondano con le informazioni sulla versione significa che sei riuscito a compilare il software `c-lightning`.

Il Progetto Lightning Network Daemon Node

Il Lightning Network Daemon (LND) è un'implementazione completa di un nodo LN di Lightning Labs. Il progetto LND fornisce una serie di applicazioni eseguibili, tra cui `lnd` (il demone stesso) e `lncli` (l'utilità della riga di comando). LND ha diversi servizi di back-end collegabili, tra cui btcd (un nodo completo), `bitcoind` (Bitcoin Core) e Neutrino (un nuovo client leggero sperimentale). LND è scritto nel linguaggio di programmazione Go. Il progetto è open source e sviluppato in collaborazione su GitHub.

Nelle prossime sezioni creeremo un contenitore Docker per eseguire LND, compileremo LND dal codice sorgente e impareremo come configurarlo ed eseguirlo.

Il Docker Container LND

Possiamo ottenere il contenitore Docker LND di esempio dal repository Docker Hub del libro:

```
$ docker pull lnbook/lnd
Using default tag: latest
latest: Pulling from lnbook/lnd
35807b77a593: Already exists
e1b85b9c5571: Already exists
52f9c252546e: Pull complete
[...]
Digest: sha256:e490a0de5d41b781c0a7f9f548c99e67f9d728f72e50cd4632722b3ed3d85952
Status: Downloaded newer image for lnbook/lnd:latest
docker.io/lnbook/lnd:latest
```

Possiamo anche compilare localmente il container LND che si trova in *code/docker/lnd*. Cambiamo la cartella di lavoro in *code/docker* ed eseguiamo `docker build`:

```
$ cd code/docker
$ docker build -t lnbook/lnd lnd
Sending build context to Docker daemon  9.728kB
Step 1/29 : FROM golang:1.13 as lnd-base
 ---> e9bdcb0f0af9
Step 2/29 : ENV GOPATH /go
[...]
Step 29/29 : CMD ["/usr/local/bin/logtail.sh"]
 ---> Using cache
 ---> 397ce833ce14
Successfully built 397ce833ce14
Successfully tagged lnbook/lnd:latest
```

Il nostro container è ora pronto per funzionare. Come per il contenitore `c-lightning` che abbiamo creato in precedenza, anche il contenitore LND dipende da un'istanza in esecuzione di Bitcoin Core. Come prima, dobbiamo avviare il contenitore `bitcoind` in un altro terminale e connettere LND ad esso tramite una rete Docker. Abbiamo già creato una rete Docker chiamata `lnbook` e la useremo di nuovo qui.

SUGGERIMENTO	Normalmente, ogni operatore di nodo esegue il proprio nodo Lightning

	e il proprio nodo Bitcoin sul proprio server. Nel nostro caso, un singolo contenitore `bitcoind` può servire molti nodi Lightning. Sulla nostra rete simulata possiamo eseguire diversi nodi Lightning, tutti collegati a un singolo nodo Bitcoin in modalità `regtest`.

Esecuzione dei contenitori bitcoind e LND

Come prima, avviamo il contenitore `bitcoind` in un terminale e LND in un altro. Se hai già il contenitore `bitcoind` in esecuzione, non è necessario riavviarlo. Basta lasciarlo in esecuzione e saltare il passaggio successivo. Per avviare `bitcoind` nella rete `lnbook`, usiamo `docker run` in questo modo:

```
$ docker run -it --network lnbook --name bitcoind lnbook/bitcoind
```

Successivamente, avviamo il contenitore LND che abbiamo appena creato. Come fatto prima, dobbiamo collegarlo alla rete `lnbook` e dargli un nome:

```
$ docker run -it --network lnbook --name lnd lnbook/lnd

Waiting for bitcoind to start...

Waiting for bitcoind to mine blocks...

Starting lnd...

Startup complete

Funding lnd wallet

{
"result":"dbd1c8e2b224e0a511c11efb985dabd84d72d935957ac30935ec4211d28beacb","er
ror":null,"id":"lnd-run-container"}

[INF] LTND: Version: 0.13.1-beta commit=v0.13.1-beta, build=production,
logging=default, debuglevel=info

[INF] LTND: Active chain: Bitcoin (network=regtest)

[INF] RPCS: Generating TLS certificates...
```

Il contenitore LND si avvia e si connette al contenitore `bitcoind` tramite la rete Docker. Innanzitutto, il nostro nodo LND aspetterà l'avvio di `bitcoind`, quindi attenderà fino a quando `bitcoind` non avrà minato dei bitcoin nel suo portafoglio. Infine, come parte dell'avvio del contenitore, uno script invierà un comando RPC al nodo `bitcoind`, creando così una transazione che finanzia il portafoglio LND con 10 BTC di prova.

Come abbiamo dimostrato in precedenza, possiamo inviare comandi al nostro contenitore in un altro terminale per estrarre informazioni, aprire canali, ecc. Il comando che ci consente di inviare istruzioni da riga di comando al demone `lnd` è chiamato `lncli`. Ancora una volta, in questo contenitore abbiamo fornito l'alias `cli` che esegue `lncli` con tutti i parametri appropriati. Otteniamo le informazioni sul nodo utilizzando il comando `docker exec` in un'altra finestra di terminale:

```
$ docker exec lnd cli getinfo
{
    "version": "0.13.1-beta commit=v0.13.1-beta",
    "commit_hash": "596fd90ef310cd7abbf2251edaae9ba4d5f8a689",
    "identity_pubkey":
"02d4545dccbeda29a10f44e891858940f4f3374b75c0f85dcb7775bb922fdeaa14",
[...]
}
```

Ora abbiamo un altro nodo Lightning in esecuzione sulla rete `lnbook` e che comunica con `bitcoind`. Se stai ancora eseguendo il contenitore `c-lightning`, ora ci sono due nodi in esecuzione. Non sono ancora collegati tra loro, ma presto li collegheremo.

Se desideri, è possibile eseguire qualsiasi combinazione di nodi LND e `c-lightning` sulla stessa rete Lightning. Ad esempio, per eseguire un secondo nodo LND devi eseguire il comando docker run con un nome contenitore diverso, in questo modo:

```
$ docker run -it --network lnbook --name lnd2 lnbook/lnd
```

Nel comando precedente, avviamo un altro contenitore LND chiamandolo `lnd2`. I nomi che scegli devono essere unici. Se non fornisci un nome, Docker creerà un nome univoco combinando casualmente due parole inglesi.

Nella prossima sezione vedremo come scaricare e compilare LND direttamente dal codice sorgente. Questo è un passaggio facoltativo e avanzato che ti insegnerà come utilizzare gli strumenti di compilazione del linguaggio Go e ti consentirà di apportare modifiche al codice sorgente LND. Con questa conoscenza puoi scrivere del codice o correggere alcuni bug.

NOTA	Se non hai intenzione di immergerti nel codice sorgente o nella programmazione di un nodo Lightning, puoi saltare completamente la sezione successiva. Il contenitore Docker che abbiamo appena creato è

	sufficiente per la maggior parte degli esempi nel libro.

Installazione di LND dal Codice Sorgente

In questa sezione compileremo LND da zero. LND è scritto nel linguaggio di programmazione Go. Se vuoi saperne di più su Go, cerca `golang` invece di `go` per evitare risultati irrilevanti. Poiché è scritto in Go e non in C o C++, utilizza un framework di "build" diverso rispetto al framework GNU autotools/make che abbiamo visto utilizzato in precedenza in `c-lightning`. Non preoccuparti però, è abbastanza facile installare e utilizzare gli strumenti golang e mostreremo ogni passaggio. Go è un linguaggio fantastico per lo sviluppo di software collaborativo perché produce codice molto coerente, preciso e di facile lettura indipendentemente dal numero di autori. Go è "minimalista" in un modo che incoraggia la coerenza tra le versioni. Come linguaggio compilato, è anche abbastanza efficiente.

Seguiremo le istruzioni di installazione presenti nella documentazione del progetto LND.

Innanzitutto, installeremo il pacchetto `golang` e le librerie associate. Richiediamo Go versione 1.13 o successiva. I pacchetti Go ufficiali sono distribuiti come binari dal Go Project. Per comodità sono anche presenti come pacchetti Debian disponibili tramite il comando apt. Puoi seguire le istruzioni del progetto Go o utilizzare i seguenti comandi apt su un sistema Debian/Ubuntu Linux come descritto nella pagina wiki di GitHub di Go:

```
$ sudo apt install golang-go
```

Verifica di avere la versione corretta installata e pronta per l'uso eseguendo:

```
$ go version
```

```
go version go1.13.4 linux/amd64
```

Abbiamo la versione 1.13.4, quindi siamo pronti a partire! Ora dobbiamo dire a tutti i programmi dove trovare il codice Go. Otteniamo ciò impostando la variabile di ambiente `GOPATH`. Di solito il codice Go si trova in una directory denominata *gocode* direttamente nella home directory dell'utente. Con i seguenti due comandi impostiamo costantemente `GOPATH` e ci assicuriamo che la tua shell lo aggiunga al tuo `PATH` eseguibile. Nota che la directory home dell'utente viene indicata come ~ nella shell.

```
$ export GOPATH=~/gocode
```

```
$ export PATH=$PATH:$GOPATH/bin
```

Per evitare di dover impostare queste variabili d'ambiente ogni volta che si apre una shell, è possibile aggiungere queste due righe alla fine del file di configurazione della shell bash *.bashrc* nella directory home, utilizzando il tuo editor preferito.

Copia del Codice Sorgente di LND

Come molti progetti open source, il codice sorgente di LND è su GitHub (*www.github.com*). Il comando `go get` può recuperarlo direttamente utilizzando il protocollo Git:

```
$ go get -d github.com/lightningnetwork/lnd
```

Una volta terminato `go get`, avrai una sottodirectory sotto `GOPATH` che contiene il codice sorgente di LND.

Compilazione del Codice Sorgente di LND

LND utilizza il sistema `make` build. Per costruire il progetto, cambiamo la directory nel codice sorgente di LND e poi usiamo `make` in questo modo:

```
$ cd $GOPATH/src/github.com/lightningnetwork/lnd
```

```
$ make && make install
```

Dopo alcuni minuti avrai due nuovi comandi, `lnd` e `lncli`, installati. Provali e controlla la loro versione per assicurarti che siano installati:

```
$ lnd --version
```

```
lnd version 0.10.99-beta commit=clock/v1.0.0-106-
gc1ef5bb908606343d2636c8cd345169e064bdc91
```

```
$ lncli --version
```

```
lncli version 0.10.99-beta commit=clock/v1.0.0-106-
gc1ef5bb908606343d2636c8cd345169e064bdc91
```

Probabilmente vedrai una versione diversa da quella mostrata in precedenza, poiché il software continua ad evolversi molto tempo dopo la pubblicazione di questo libro. Tuttavia, indipendentemente dalla versione visualizzata, il fatto che i comandi vengano eseguiti e mostrino le informazioni sulla versione significa che sei riuscito a compilare il software LND.

Il progetto Eclair Lightning Node

Eclair (fulmine in francese) è un'implementazione Scala di Lightning Network realizzata da ACINQ. Eclair è anche uno dei portafogli Lightning mobile più popolari e pionieristici, che abbiamo utilizzato per dimostrare un pagamento Lightning nel Capitolo 2. In questa sezione esaminiamo il progetto del server Eclair, che esegue un nodo Lightning. Eclair è un progetto open source e può essere trovato su GitHub.

Nelle prossime sezioni compileremo un contenitore Docker per eseguire Eclair, come abbiamo fatto in precedenza con `c-lightning` e LND. Compileremo anche Eclair direttamente dal codice sorgente.

Il Contenitore Docker Eclair

Estraiamo il contenitore Eclair del libro dal repository Docker Hub:

```
$ docker pull lnbook/eclair
Using default tag: latest
latest: Pulling from lnbook/eclair
35807b77a593: Already exists
e1b85b9c5571: Already exists
[...]
c7d5d5c616c2: Pull complete
Digest: sha256:17a3d52bce11a62381727e919771a2d5a51da9f91ce2689c7ecfb03a6f028315
Status: Downloaded newer image for lnbook/eclair:latest
docker.io/lnbook/eclair:latest
```

In alternativa, possiamo invece costruire il contenitore localmente. Ormai sei quasi un esperto nelle operazioni di base di Docker! In questa sezione ripeteremo molti dei comandi visti in precedenza per costruire il container Eclair. Il contenitore si trova in *code/docker/eclair*. Iniziamo in un terminale cambiando la directory di lavoro in *code/docker* ed emettendo il comando `docker build`:

```
$ cd code/docker
$ docker build -t lnbook/eclair eclair
Sending build context to Docker daemon  11.26kB
```

```
Step 1/27 : ARG OS=ubuntu
Step 2/27 : ARG OS_VER=focal
Step 3/27 : FROM ${OS}:${OS_VER} as os-base
 ---> fb52e22af1b0
[...]
Step 27/27 : CMD ["/usr/local/bin/logtail.sh"]
 ---> Running in fe639120b726
Removing intermediate container fe639120b726
 ---> e6c8fe92a87c
Successfully built e6c8fe92a87c
Successfully tagged lnbook/eclair:latest
```

La nostra immagine è ora pronta per essere eseguita. Il contenitore Eclair dipende anche da un'istanza in esecuzione di Bitcoin Core. Come prima, dobbiamo avviare il contenitore `bitcoind` in un altro terminale e connettere Eclair ad esso tramite una rete Docker. Abbiamo già creato una rete Docker chiamata `lnbook` e la riutilizzeremo qui.

Una notevole differenza tra Eclair e LND o `c-lightning` è che Eclair non contiene un portafoglio bitcoin separato ma si basa invece direttamente sul portafoglio in Bitcoin Core. Ricordiamo che utilizzando LND abbiamo finanziato il suo portafoglio bitcoin eseguendo una transazione per trasferire bitcoin dal portafoglio di Bitcoin Core al portafoglio bitcoin di LND. Questo passaggio non è necessario utilizzando Eclair. Quando si esegue Eclair, il portafoglio Bitcoin Core viene utilizzato direttamente come fonte di fondi per aprire i canali. Di conseguenza, a differenza dei contenitori LND o `c-lightning`, il contenitore Eclair non contiene uno script per trasferire bitcoin nel suo portafoglio all'avvio.

Esecuzione dei contenitori bitcoind ed Eclair

Come prima, avviamo il contenitore `bitcoind` in un terminale e il contenitore Eclair in un altro. Se hai già il contenitore `bitcoind` in esecuzione, non è necessario riavviarlo. Basta lasciarlo in esecuzione e saltare il passaggio successivo. Per avviare `bitcoind` nella rete `lnbook`, usiamo `docker run` in questo modo:

```
$ docker run -it --network lnbook --name bitcoind lnbook/bitcoind
```

Ora avviamo il contenitore Eclair che abbiamo appena creato. Dovremo collegarlo alla rete

lnbook e dargli un nome, proprio come abbiamo fatto con gli altri contenitori:

```
$ docker run -it --network lnbook --name eclair lnbook/eclair
Waiting for bitcoind to start...
Waiting for bitcoind to mine blocks...
Starting eclair...
Eclair node started
INFO  o.b.Secp256k1Context - secp256k1 library successfully loaded
INFO  fr.acinq.eclair.Plugin - loading 0 plugins
INFO  a.e.slf4j.Slf4jLogger - Slf4jLogger started
INFO  fr.acinq.eclair.Setup - hello!
INFO  fr.acinq.eclair.Setup - version=0.4.2 commit=52444b0
[...]
```

Il contenitore Eclair si avvia e si connette al contenitore `bitcoind` tramite la rete Docker. Innanzitutto, il nostro nodo Eclair attenderà l'avvio di `bitcoind`, quindi attenderà fino a quando `bitcoind` non avrà minato dei bitcoin nel suo portafoglio.

Come abbiamo dimostrato in precedenza, possiamo inviare comandi al nostro contenitore in un altro terminale per estrarre informazioni, aprire canali, ecc. Il comando che ci consente di inviare istruzioni da riga di comando al demone `eclair` è chiamato `eclair-cli`. Come prima, in questo contenitore abbiamo fornito un alias a `eclair-cli`, chiamato semplicemente `cli`, che offre gli argomenti ed i parametri necessari. Utilizzando il comando `docker exec` in un'altra finestra di terminale, otteniamo le informazioni sul nodo da Eclair:

```
$ docker exec eclair cli getinfo
{
  "version": "0.4.2-52444b0",
  "nodeId":
"02fa6d5042eb8098e4d9c9d99feb7ebc9e257401ca7de829b4ce757311e0301de7",
  "alias": "eclair",
  "color": "#49daaa",
  "features": {
[...]
  },
```

```
  "chainHash":
"06226e46111a0b59caaf126043eb5bbf28c34f3a5e332a1fc7b2b73cf188910f",
  "network": "regtest",
  "blockHeight": 779,
  "publicAddresses": [],
  "instanceId": "01eb7a68-5db0-461b-bdd0-29010df40d73"
}
```

Ora abbiamo un altro nodo Lightning in esecuzione sulla rete `lnbook` e che comunica con `bitcoind`. Puoi eseguire qualsiasi numero e qualsiasi combinazione di nodi Lightning sulla stessa rete Lightning. Qualsiasi numero di nodi Eclair, LND e `c-lightning` può coesistere. Ad esempio, per eseguire un secondo nodo Eclair devi eseguire il comando `docker run` con un nome di contenitore diverso, come segue:

```
$ docker run -it --network lnbook --name eclair2 lnbook/eclair
```

Nel comando precedente avviamo un altro contenitore Eclair denominato `eclair2`.

Nella prossima sezione vedremo anche come scaricare e compilare Eclair direttamente dal codice sorgente. Questo è un passaggio facoltativo e avanzato che ti insegnerà come utilizzare gli strumenti di creazione del linguaggio Scala e Java e ti consentirà di apportare modifiche al codice sorgente di Eclair. Con questa conoscenza, puoi scrivere del codice o correggere alcuni bug.

NOTA	Se non hai intenzione di immergerti nel codice sorgente o nella programmazione di un nodo Lightning, puoi saltare completamente la sezione successiva. Il contenitore Docker che abbiamo appena creato è sufficiente per la maggior parte degli esempi nel libro.

Installazione di Eclair dal Codice Sorgente

In questa sezione compileremo Eclair da zero. Eclair è scritto nel linguaggio di programmazione Scala, che viene compilato utilizzando il compilatore Java. Per eseguire Eclair, dobbiamo prima installare Java e i suoi strumenti di compilazione. Seguiremo le istruzioni nel documento BUILD.md del progetto Eclair.

Il compilatore Java richiesto fa parte di OpenJDK 11. Avremo anche bisogno di un framework

di compilazione chiamato Maven, versione 3.6.0 o successiva.

Su un sistema Debian/Ubuntu Linux, possiamo utilizzare il comando `apt` per installare sia OpenJDK 11 che Maven, come mostrato di seguito:

```
$ sudo apt install openjdk-11-jdk maven
```

Verifica di avere la versione corretta installata eseguendo:

```
$ javac -version
```

```
javac 11.0.7
```

```
$ mvn -v
```

```
Apache Maven 3.6.1
```

```
Maven home: /usr/share/maven
```

```
Java version: 11.0.7, vendor: Ubuntu, runtime: /usr/lib/jvm/java-11-openjdk-amd64
```

Abbiamo OpenJDK 11.0.7 e Maven 3.6.1, quindi siamo pronti.

Copiare il Codice Sorgente di Eclair

Il codice sorgente per Eclair è su GitHub. Il comando `git clone` permette di creare una copia locale. Passiamo alla nostra home directory ed eseguiamolo lì:

```
$ cd ~
```

```
$ git clone https://github.com/ACINQ/eclair.git
```

Al termine di `git clone`, avrai una sottodirectory `eclair` contenente il codice sorgente per il server Eclair.

Compilazione del Codice Sorgente di Eclair

Eclair utilizza il sistema di compilazione `Maven`. Per compilare il progetto, cambiamo la cartella di lavoro su Eclair e quindi utilizziamo il pacchetto `mvn` in questo modo:

```
$ cd eclair
```

```
$ mvn package
```

```
[INFO] Scanning for projects...
```

```
[INFO] ------------------------------------------------------------------------
```

```
[INFO] Reactor Build Order:
[INFO]
[INFO] --------------------< fr.acinq.eclair:eclair_2.13 >---------------------
[INFO] Building eclair_2.13 0.4.3-SNAPSHOT                                [1/4]
[INFO] --------------------------------[ pom ]---------------------------------
[...]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:06 min
[INFO] Finished at: 2020-12-12T09:43:21-04:00
[INFO] ------------------------------------------------------------------------
```

Dopo alcuni minuti, la compilazione del pacchetto Eclair dovrebbe essere completata. Tuttavia, l'azione eseguirà anche dei test e alcuni di questi creano una connessione a Internet e potrebbero fallire. Se vuoi saltare i test, aggiungi `-DskipTests` al comando sopra.

Ora estrai ed esegui il pacchetto build seguendo le istruzioni per l'installazione di Eclair.

Congratulazioni! Hai compilato Eclair dal codice sorgente e sei pronto per sviluppare, testare, correggere bug e contribuire a questo progetto!

Costruire una Rete Completa Composta da Diversi Nodi Lightning

L'esempio finale, presentato in questa sezione, riunirà tutti i vari contenitori che abbiamo creato per formare una rete Lightning composta da diverse implementazioni di nodi (LND, `c-lightning`, Eclair). Costruiremo la rete collegando i nodi insieme e aprendo i canali da un nodo all'altro. Come passaggio finale, instraderemo un pagamento attraverso questi canali!

In questo esempio, creeremo una rete Lightning dimostrativa composta da quattro nodi Lightning denominati Alice, Bob, Chan e Dina. Collegheremo Alice a Bob, Bob a Chan e Chan a Dina. Questo è mostrato in Figura 4-1.

Figura 4-1. Una piccola rete dimostrativa composta da quattro nodi

Infine, Dina creerà una fattura e la farà pagare ad Alice. Poiché Alice e Dina non sono collegate direttamente, il pagamento verrà instradato come HTLC attraverso tutti i canali di pagamento.

Utilizzo di docker-compose per Orchestrare i Contenitori Docker

Per far funzionare questo esempio, utilizzeremo uno strumento di *orchestrazione del contenitore* disponibile come comando denominato `docker-compose`. Questo comando ci consente di specificare un'applicazione composta da diversi contenitori ed eseguire l'applicazione avviando insieme tutti i contenitori cooperanti.

Innanzitutto, installiamo docker-compose. Le istruzioni dipendono dal sistema operativo.

Una volta completata l'installazione, puoi verificare l'installazione eseguendo docker-compose in questo modo:

```
$ docker-compose version
docker-compose version 1.21.0, build unknown
[...]
```

I comandi `docker-compose` comuni che useremo sono `up` e `down`, es. docker-compose up.

Configurazione docker-compose

Il file di configurazione per `docker-compose` si trova nella directory *code/docker* ed è denominato *docker-compose.yml*. Contiene una specifica per una rete e ciascuno dei quattro contenitori. La parte superiore si presenta così:

```
version: "3.3"
networks:
  lnnet:
services:
  bitcoind:
    container_name: bitcoind
    build:
        context: bitcoind
    image: lnbook/bitcoind:latest
    networks:
      - lnnet
    expose:
      - "18443"
      - "12005"
      - "12006"
  Alice:
    container_name: Alice
```

Il frammento precedente definisce una rete chiamata `lnnet` e un contenitore chiamato `bitcoind` che si collegherà alla rete `lnnet`. Il contenitore è lo stesso che abbiamo costruito all'inizio di questo capitolo. Esponiamo tre delle porte del contenitore, permettendoci di inviargli comandi e monitorare blocchi e transazioni. Successivamente, la configurazione specifica un contenitore LND chiamato "Alice". Più in basso vedrai anche le specifiche per i contenitori chiamati "Bob" (`c-lightning`), "Chan" (Eclair) e "Dina" (di nuovo LND).

Poiché tutte queste diverse implementazioni seguono la specifica BOLT e sono state ampiamente testate per l'interoperabilità, non hanno difficoltà a lavorare insieme per costruire una rete Lightning.

Avvio della Rete Lightning di Esempio

Prima di iniziare, dovremmo assicurarci che non stiamo già eseguendo nessuno dei container. Se un nuovo contenitore condivide lo stesso nome di uno già in esecuzione, non verrà avviato. Se necessario, utilizza `docker ps`, `docker stop` e `docker rm` per arrestare e rimuovere tutti i contenitori attualmente in esecuzione!

SUGGERIMENTO	Poiché utilizziamo gli stessi nomi per questi contenitori Docker orchestrati, potrebbe essere necessario "ripulire" per evitare conflitti.

Per iniziare, passiamo alla directory che contiene il file di configurazione *docker-compose.yml* e diamo il comando `docker-compose up`:

```
$ cd code/docker
$ docker-compose up
Creating Chan     ... done
Creating Dina     ... done
Creating bitcoind ... done
Creating Bob      ... done
Creating Alice    ... done
Attaching to Chan, Dina, Alice, bitcoind, Bob
Alice       | Waiting for bitcoind to start...
Bob         | Waiting for bitcoind to start...
Dina        | Waiting for bitcoind to start...
Chan        | Waiting for bitcoind to start...
bitcoind    | Starting bitcoind...
bitcoind    | Waiting for bitcoind to start
bitcoind    | bitcoind started
bitcoind    | ================================================
[...]
Chan        | Starting eclair...
Dina        | Starting lnd...
Chan        | Eclair node started
```

```
Alice       | ...Waiting for bitcoind to mine blocks...
Bob         | ...Waiting for bitcoind to mine blocks...
Alice       | Starting lnd...
Bob         | Starting c-lightning...
[...]
```

Dopo l'avvio, vedrai un intero flusso di log mentre ogni nodo si avvia e riporta il suo progresso. Potrebbe sembrare piuttosto confuso sullo schermo, ma ogni riga di output è preceduta dal nome del contenitore, come visto in precedenza. Se desideri visualizzare i log di un solo contenitore, è possibile farlo in un'altra finestra di terminale utilizzando il comando `docker-compose logs` con il flag f (*follow*) e il nome del contenitore specifico:

```
$ docker-compose logs -f Alice
```

Apertura di Canali e Instradamento di un Pagamento

La nostra rete Lightning dovrebbe ora essere in esecuzione. Come abbiamo visto nelle sezioni precedenti di questo capitolo, possiamo inviare comandi a un contenitore Docker in esecuzione con il comando `docker exec`. Indipendentemente dal fatto che abbiamo avviato il contenitore con `docker run` o avviato un gruppo di essi con `docker-compose up`, possiamo comunque accedere ai contenitori individualmente utilizzando i comandi Docker.

La demo di pagamento è contenuta in uno script della shell Bash chiamato `run-payment-demo.sh`. Per eseguire questa demo devi avere la shell Bash installata sul tuo computer. La maggior parte dei sistemi simili a Linux e Unix (es. macOS) hanno `bash` preinstallato. Gli utenti Windows possono installare il sottosistema Windows for Linux e utilizzare una distribuzione Linux come Ubuntu per avere un comando `bash` nativo sul proprio computer.

Eseguiamo lo script per vederne l'effetto, per poi analizzare come funziona internamente. Usiamo `bash` per eseguirlo come comando:

```
$ cd code/docker
$ bash run-payment-demo.sh
Starting Payment Demo
======================================================
Waiting for nodes to startup
- Waiting for bitcoind startup...
```

```
- Waiting for bitcoind mining...
- Waiting for Alice startup...
- Waiting for Bob startup...
- Waiting for Chan startup...
- Waiting for Dina startup...
All nodes have started
=======================================================
Getting node IDs
- Alice:  0335e200756e156f1e13c3b901e5ed5a28b01a3131cd0656a27ac5cc20d4e71129
- Bob:    033e9cb673b641d2541aaaa821c3f9214e8a11ada57451ed5a0eab2a4afbce7daa
- Chan:   02f2f12182f56c9f86b9aa7d08df89b79782210f0928cb361de5138364695c7426
- Dina: 02d9354cec0458e0d6dee5cfa56b83040baddb4ff88ab64960e0244cc618b99bc3
=======================================================
[...]
Setting up connections and channels
- Alice to Bob
- Open connection from Alice node to Bob's node
- Create payment channel Alice->Bob
[...]
Get 10k sats invoice from Dina
- Dina invoice:
lnbcrt100u1psnuzzrpp5rz5dg4wy27973yr7ehwns5ldeusceqdaq0hguu8c29n4nsqkznjsdqqcqz
pgxqyz5vqsp5vdpehw33fljnmmexa6ljk55544f3syd8nfttqlm3ljewu4r0q20q9qyyssqxh5nhkpj
gfm47yxn4p9ecvndz7zddlsgpufnpyjl0kmnq227tdujlm0acdv39hcuqp2vhs40aav70c9yp0tee6t
gzk8ut79mr877q0cpkjcfvr
=======================================================
Attempting payment from Alice to Dina
Successful payment!
```

Come puoi vedere dall'output, lo script ottiene prima gli ID dei nodi (chiavi pubbliche) per ciascuno dei quattro nodi. Quindi, connette i nodi e imposta un canale di 1.000.000 di satoshi da ciascun nodo a quello successivo nella rete. Infine, emette una fattura per 10.000 satoshi dal nodo di Dina e paga la fattura dal nodo di Alice.

SUGGERIMENTO	Se lo script fallisce, puoi provare ad eseguirlo di nuovo dall'inizio. Puoi anche lanciare manualmente i comandi trovati nello script uno per uno e guardare i risultati.

C'è molto da rivedere in quello script, ma man mano che acquisisci comprensione della tecnologia sottostante, le informazioni diventeranno sempre più chiare...

Ovviamente, puoi fare molto di più con questa rete di test rispetto a un pagamento composto da tre canali e quattro nodi. Ecco alcune idee da sperimentare:

- Crea una rete più complessa avviando più nodi diversi. Modifica il file *docker-compose.yml* e copia le sezioni, rinominando i contenitori secondo le necessità.
- Connetti i nodi in topologie complesse: percorsi circolari, hub-and-spoke o full mesh.
- Fai molti pagamenti per esaurire la capacità del canale. Fai pagamenti nella direzione opposta per riequilibrare i canali. Guarda come si adatta l'algoritmo di routing.
- Modifica le commissioni dei canale per vedere come l'algoritmo di routing negozia più percorsi e quali ottimizzazioni applica. Un percorso lungo ed economico è meglio di un percorso breve e costoso?
- Esegui un pagamento circolare da un nodo a se stesso per ribilanciare i canali. Guarda come ciò influisce su tutti gli altri canali e nodi.
- Genera centinaia o migliaia di piccole fatture in un ciclo e poi pagale il più velocemente possibile in un altro ciclo. Misura quante transazioni al secondo puoi effettuare in questa rete di test.

SUGGERIMENTO	Lightning Polar ti consente di visualizzare graficamente la rete che hai creato utilizzando Docker.

Conclusione

In questo capitolo abbiamo esaminato vari progetti che implementano le specifiche BOLT. Abbiamo creato container per eseguire una rete Lightning di esempio e imparato a compilare ogni progetto dal codice sorgente. Ora sei pronto per esplorare ulteriormente e scendere più a fondo nella tana del bianconiglio di Lightning Network!

Capitolo 5. Gestione di un Nodo su Lightning Network

Avendo letto fin qui, probabilmente hai già creato un portafoglio Lightning. In questo capitolo, faremo un ulteriore passo avanti e creeremo un nodo Lightning completo. Oltre a configurarne uno, impareremo a farlo funzionare e mantenerlo nel tempo.

Ci sono molti motivi per cui potresti voler configurare il tuo nodo Lightning:

- Per essere un partecipante attivo e completo su LN, non solo un utente finale
- Per gestire un e-commerce o ricevere pagamenti tramite pagamenti Lightning
- Per guadagnare dalle commissioni di instradamento o affittare la liquidità dei canali
- Per sviluppare nuovi servizi, applicazioni o plug-in per Lightning Network
- Per aumentare la tua privacy finanziaria durante l'utilizzo di Lightning Network
- Per utilizzare alcune app basate su Lightning, come app di messaggistica istantanea
- Per la libertà finanziaria, l'indipendenza e la sovranità

Sono previsti costi associati all'esecuzione di un nodo LN. Hai bisogno di un computer, una connessione Internet permanente, molto spazio su disco e molto tempo! I costi operativi includeranno inoltre le spese per l'energia elettrica.

Ricorda comunque che le abilità che imparerai da questa esperienza sono preziose e possono essere applicate anche a una varietà di altri campi. Iniziamo!

NOTA	È importante che imposti correttamente le tue aspettative su fatti concreti. Se prevedi di gestire un nodo Lightning esclusivamente per guadagnare entrate instradando pagamenti di altri, per prima cosa studia. Avviare un'attività redditizia gestendo un nodo Lightning non è sicuramente facile. Calcola tutti i tuoi costi iniziali e correnti in un foglio di calcolo. Studia attentamente le statistiche di LN. Qual è l'attuale volume di pagamento? Qual è il volume per nodo? Quali sono le commissioni di instradamento medie attuali? Consulta i forum e chiedi consigli o feedback ad altri membri della comunità che hanno già

	acquisito esperienza nel mondo reale. Abbi una opinione solo dopo aver svolto questo esercizio di due diligence. La maggior parte delle persone troverà la motivazione per gestire un nodo non nel guadagno finanziario, ma da qualche altra parte.

Scegli la Tua Piattaforma

Esistono molti modi per eseguire un nodo Lightning, che vanno da un piccolo PC ospitato in casa, passando da un server dedicato, fino ad arrivare a un server ospitato nel cloud. Il metodo che scegli dipenderà dalle risorse che hai e da quanti soldi vuoi spendere.

Perché l'Affidabilità è Importante per l'Esecuzione di un Nodo Lightning?

In Bitcoin l'hardware non è particolarmente importante a meno che non si stia eseguendo specificamente un nodo di mining. Il software Bitcoin Core può essere eseguito su qualsiasi macchina che soddisfi i requisiti minimi e non ha bisogno di essere online per ricevere pagamenti, ma solo per inviarli. Se un nodo Bitcoin si interrompe per diversi tempo, l'utente può semplicemente riavviare il nodo e, una volta connesso alla rete, si risincronizzerà.

In Lightning, invece, l'utente deve essere online sia per inviare che per ricevere pagamenti. Se il nodo Lightning è offline, non può ricevere alcun pagamento da nessuno e quindi le sue fatture aperte non possono essere pagate. Inoltre, i canali aperti di un nodo offline non possono essere utilizzati per instradare i pagamenti. I tuoi partner di canale noteranno che sei offline e non possono usarti per inoltrare un pagamento. Se sei offline troppo spesso, potrebbero considerare i bitcoin bloccati nei vostri canali come capacità inutilizzata e potrebbero chiuderli. Abbiamo già discusso il caso di un attacco di protocollo in cui il tuo partner di canale tenta di imbrogliarti inviando una transazione di impegno precedente. Se sei offline e i tuoi canali non vengono monitorati, il tentativo di furto potrebbe avere successo e non potrai fare ricorso una volta scaduto il timelock. Pertanto, l'affidabilità del nodo è estremamente importante per un nodo Lightning.

Ci sono anche problemi di guasti hardware e perdita di dati. In Bitcoin, un guasto hardware può essere un problema banale se l'utente dispone di un backup della propria frase mnemonica o delle chiavi private. Il portafoglio Bitcoin ed i bitcoin all'interno del portafoglio possono essere facilmente ripristinati dalle chiavi private su un nuovo computer. La maggior

parte delle informazioni può essere riscaricata dalla blockchain.

Al contrario, in Lightning le informazioni sui canali dell'utente, comprese le transazioni di impegno e i segreti di revoca, non sono di dominio pubblico e vengono memorizzate solo sull'hardware del singolo utente. Pertanto, i guasti software e hardware nella rete Lightning possono facilmente comportare la perdita di fondi.

Tipi di Hardware di Nodi Lightning

Esistono tre tipi principali di hardware di nodi Lightning:

Computer generici

> Un nodo LN può essere eseguito su un computer di casa o un laptop con Windows, macOS o Linux. In genere questo viene eseguito insieme a un nodo Bitcoin.

Hardware dedicato

> Un nodo Lightning può anche essere eseguito su hardware dedicato come un Raspberry Pi, Rock64 o un mini PC. Questa configurazione di solito esegue uno stack software, incluso un nodo Bitcoin e altre applicazioni. Questa configurazione è popolare perché l'hardware è dedicato all'esecuzione e alla manutenzione del solo nodo Lightning e di solito è configurato con un "helper" di installazione.

Hardware preconfigurato

> Un nodo LN può anche essere eseguito su hardware specifico, appositamente selezionato e configurato per esso. Ciò includerebbe soluzioni di nodi "pronte all'uso" che possono essere acquistate come kit o come sistema chiavi in mano.

Gestione nel "Cloud"

Il *Virtual Private Server* (VPS) e i servizi di cloud computing come Microsoft Azure, Google Cloud, Amazon Web Services (AWS) o DigitalOcean sono abbastanza convenienti e possono essere configurati molto rapidamente. Un nodo Lightning costa tra $20 e $40 al mese su questo tipi di servizi.

Tuttavia, come si suol dire, "'Cloud' è solo il computer di altre persone". Utilizzare questi servizi significa eseguire il tuo nodo su computer di altre persone. Ciò comporta diversi vantaggi e svantaggi. I principali vantaggi sono la convenienza, l'efficienza, il tempo di attività e forse anche il costo. L'operatore cloud gestisce ed esegue il nodo ad alto livello, fornendoti automaticamente comodità ed efficienza. Fornisce velocità e disponibilità eccellenti, spesso molto migliori di quello che un individuo può ottenere a casa. Se consideri che solo il costo dell'elettricità per far funzionare un server in molti paesi occidentali è di circa $10 al mese, aggiungendo il costo della larghezza di banda della rete e dell'hardware stesso, l'offerta VPS diventa finanziariamente competitiva. Infine, con un VPS non hai bisogno di spazio per un PC a casa e non hai problemi con il rumore o il calore del PC. D'altra parte, ci sono diversi notevoli svantaggi. Un nodo Lightning in esecuzione nel "cloud" sarà sempre meno sicuro e meno privato di uno in esecuzione sul tuo computer. Inoltre, questi servizi di cloud computing sono molto centralizzati. La stragrande maggioranza dei nodi Bitcoin e Lightning in esecuzione su tali servizi si trova in una manciata di data center in Virginia, Sunnyvale, Seattle, Londra e Francoforte. Quando le reti o i data center di questi provider hanno problemi di servizio, ciò interessa migliaia di nodi sulle cosiddette reti "decentralizzate".

Se hai la possibilità e la capacità di eseguire un nodo sul tuo computer a casa o in ufficio, allora potrebbe essere preferibile piuttosto che eseguirlo nel cloud. Tuttavia, se l'esecuzione del proprio server non è un'opzione, è assolutamente consigliabile eseguirne uno su un VPS.

Esecuzione di un Nodo a Casa

Se disponi di una connessione Internet di capacità ragionevole a casa o in ufficio, puoi sicuramente eseguire un nodo Lightning. Qualsiasi connessione "a banda larga" è sufficiente allo scopo di eseguire un nodo leggero e una connessione veloce ti consentirà di eseguire anche un nodo completo Bitcoin.

Anche se puoi eseguire un nodo Lightning (e persino un nodo Bitcoin) sul tuo laptop, diventerà "stretto" col passare de tempo. Questi programmi consumano le risorse del tuo computer e devono essere eseguiti 24 ore su 24, 7 giorni su 7. Le applicazioni come il tuo browser competeranno con i servizi in background di Lightning per le risorse del tuo computer. In altre parole, il tuo browser e altri programmi saranno rallentati. Quando la tua app blocca il tuo laptop, anche il tuo nodo Lightning si interromperà, lasciandoti incapace di ricevere transazioni e trovandosi potenzialmente vulnerabile ad attacchi. Inoltre, non dovresti mai spegnere il tuo laptop. Tutto questo si traduce in una configurazione non ideale.

Lo stesso vale per il tuo desktop personale.

La maggior parte degli utenti sceglierà di eseguire un nodo su un computer dedicato. Fortunatamente, non hai bisogno di un "server" per farlo. Puoi eseguire un nodo Lightning su un computer a scheda singola, come un Raspberry Pi o su un mini PC (solitamente commercializzato come PC home theater). Si tratta di semplici computer comunemente utilizzati come hub domotico o media server. Sono più economici rispetto a un PC o un laptop. Il vantaggio di un dispositivo dedicato come piattaforma per i nodi Lightning e Bitcoin è che può funzionare in modo continuo, silenzioso e discreto sulla rete domestica, nascosto dietro il router o la TV. Nessuno saprà nemmeno che questa piccola scatola fa effettivamente parte di un sistema monetario globale!

ATTENZIONE	Non è consigliabile utilizzare un nodo su un sistema operativo a 32 bit e/o una CPU a 32 bit, poiché il software del nodo potrebbe incorrere in problemi di risorse, causando un arresto anomalo e potenzialmente una perdita di fondi.

Quale Hardware è Necessario per Eseguire un Nodo Lightning?

Come minimo, per eseguire un nodo Lightning è necessario quanto segue:

CPU

È necessaria una potenza di elaborazione sufficiente per eseguire un nodo Bitcoin, che scaricherà e convaliderà continuamente nuovi blocchi. L'utente deve anche considerare il download del blocco iniziale (IBD – Initial Block Download) durante la configurazione di un nuovo nodo Bitcoin, che può richiedere da diverse ore a diversi giorni. Si consiglia una CPU a 2 o 4 core.

RAM

Un sistema con 2 GB di RAM eseguirà *a malapena* entrambi i nodi Bitcoin e Lightning. Funzionerà molto meglio con almeno 4 GB di RAM. L'IBD sarà particolarmente impegnativo con meno di 4 GB di RAM. Più di 8 GB di RAM non sono necessari perché la CPU è il collo di bottiglia maggiore per questo tipo di servizi, a causa di operazioni crittografiche come la convalida delle firme.

Unità di archiviazione

Può trattarsi di un disco rigido (HDD) o di un'unità a stato solido (SSD). Un SSD sarà significativamente più veloce (ma più costoso) per l'esecuzione di un nodo. La maggior parte dello spazio di archiviazione viene utilizzata per la blockchain Bitcoin, che ha una dimensione di centinaia di gigabyte. Un giusto compromesso (costo per complessità) è acquistare un piccolo SSD da cui avviare il sistema operativo e un HDD più grande per archiviare dati di grandi dimensioni (principalmente database).

NOTA	I Raspberry Pi sono una scelta comune per l'esecuzione del software del nodo, a causa del costo e della disponibilità delle parti. Il sistema operativo in esecuzione sul dispositivo di solito si avvia da una scheda SD (Secure Digital). Per la maggior parte dei casi d'uso, questo non è un problema, ma Bitcoin Core è noto per essere pesante lato I/O. Dovresti assicurarti di posizionare la blockchain Bitcoin e la directory dei dati Lightning su un'unità diversa perché l'I/O intensivo a lungo termine può causare il fallimento di una scheda SD.

Connessione internet

È necessaria una connessione Internet affidabile per scaricare nuovi blocchi Bitcoin e per comunicare con altri peer Lightning. Durante il funzionamento, l'utilizzo stimato dei dati varia da 10 a 100 GB al mese, a seconda della configurazione. All'avvio, un nodo completo Bitcoin scarica l'intera blockchain.

Alimentazione elettrica

È necessario un alimentatore affidabile in quanto i nodi Lightning devono essere sempre online. Un'interruzione di corrente causerà il fallimento dei pagamenti in corso. Per i nodi di instradamento, un gruppo di continuità (UPS) è utile in caso di interruzioni di corrente. Dovresti collegare anche il tuo router Internet a questo UPS.

Backup

Il backup è fondamentale perché un errore può comportare la perdita di dati e quindi la perdita di fondi. Ti consigliamo di prendere in considerazione una sorta di soluzione

di backup dei dati. Potrebbe trattarsi di un backup automatico basato su cloud su un server o un servizio Web che controlli. In alternativa, potrebbe trattarsi di un backup hardware locale automatizzato, ad esempio un secondo disco rigido. Per ottenere i migliori risultati, è possibile combinare sia il backup locale che quello remoto.

Cambio della Configurazione del Server nel Cloud

Quando si affitta un server cloud, è spesso conveniente modificare la configurazione tra due diverse fasi operative. Durante l'IBD (ad esempio, il primo giorno) saranno necessarie una CPU più veloce e una memoria più veloce. Una volta che la blockchain è stata sincronizzata, i requisiti di CPU e velocità di archiviazione sono molto inferiori, quindi le prestazioni possono essere ridotte a un livello più conveniente.

Ad esempio, sul cloud di Amazon, utilizzeremmo una RAM da 8-16 GB, una CPU a 8 core (ad es. t3-large o m3.large) e un SSD da 400 GB (che permette oltre 1000 operazioni di input/output al secondo [IOPS]) per l'IBD, riducendone il tempo a sole 6-8 ore. Una volta completato, passeremo all'istanza del server con una RAM da 2 GB, una CPU a 2 core (ad es. t3.small) e gestiremo l'archiviazione con un HDD generico da 1 TB. Tutto ciò ti permette di risparmiare soldi e metterà i tuoi nodi in completa funzione in meno di un giorno anziché dover aspettare quasi una settimana per l'IBD.

Archiviazione permanente dei dati (unità)

Se utilizzi un mini PC o noleggi un server, l'archiviazione può essere la parte più costosa, poiché costa quanto il computer e la connettività (dati) sommati.

Esistono due tipi principali di unità, HDD e SSD. Gli HDD sono più economici e gli SSD sono più veloci, ma entrambi fanno il loro lavoro egregiamente.

Gli SSD più veloci oggi disponibili utilizzano l'interfaccia Non-Volatile Memory Express (NVMe). Gli SSD NVMe sono più veloci nelle macchine di fascia alta, ma anche più costosi. I tradizionali SSD basati su SATA sono più economici, ma non così veloci. Gli SSD SATA funzionano sufficientemente bene per la configurazione del tuo nodo. I computer più piccoli potrebbero non essere in grado di sfruttare gli SSD NVMe. Ad esempio, il Raspberry Pi 4 non può trarne vantaggio a causa della larghezza di banda limitata della sua porta USB.

Per scegliere la dimensione corretta, diamo un'occhiata alla blockchain di Bitcoin. A Febbraio

2023, la sua dimensione è di circa 520 GB, incluso l'indice delle transazioni, e cresce di circa 60 GB all'anno. Se desideri avere un margine disponibile per la crescita futura o per installare altri dati sul tuo nodo, acquista almeno un'unità da 1 TB o, meglio ancora, un'unità da 2 TB.

Utilizzare un Installer o un Assistente

L'installazione di un nodo Lightning o di un nodo Bitcoin può essere scoraggiante se non si ha familiarità con un ambiente a riga di comando. Fortunatamente, ci sono un certo numero di progetti che creano degli "assistenti", cioè software che installano e configurano i vari componenti richiesti per te. Avrai ancora bisogno di imparare alcuni comandi da terminale per interagire con il tuo nodo, ma la maggior parte del lavoro iniziale viene svolto per te.

RaspiBlitz

Uno degli "assistenti" più popolari e completi è *RaspiBlitz* (Figura 5-1), un progetto realizzato da Christian Rotzoll. È pensato per essere installato su un Raspberry Pi 4. RaspiBlitz viene fornito con un kit hardware consigliato che puoi costruire in poche ore o al massimo in un weekend. Se partecipi ad un "hackathon" di LN nella tua città, è probabile che vedrai molte persone lavorare alla configurazione di RaspiBlitz, scambiarsi suggerimenti e aiutarsi a vicenda. Puoi trovare il progetto RaspiBlitz su GitHub.

Figura 5-1. Un nodo RaspiBlitz

Mynode

myNode è un altro popolare "assistente" open source che include molti software relativi a Bitcoin. È facile da installare: basta effettuare il "flash" del programma di installazione su una scheda SD e avviare il tuo mini PC dalla stessa. Non è necessario alcun monitor per utilizzare myNode perché gli strumenti di amministrazione sono accessibili in remoto da un browser. Se il tuo mini PC non ha monitor, mouse o tastiera, puoi gestirlo da un altro computer o anche dal tuo smartphone. Una volta installato, ti basterà andare su http://mynode.local e potrai creare un portafoglio e un nodo Lightning in due click.

Umbrel

Famoso per la sua UX/UI (mostrata in Figura 5-2), *Umbrel* fornisce un modo molto semplice e veloce per mettere in piedi il tuo nodo Bitcoin e Lightning in pochissimo tempo, specialmente per i principianti. Una caratteristica molto distintiva è che Umbrel utilizza Neutrino/SPV durante l'IBD in modo che tu possa iniziare sin da subito ad utilizzare il tuo nodo. Una volta che Bitcoin Core è completamente sincronizzato, disabilita in automatico la modalità SPV. Umbrel OS supporta Raspberry Pi 4 e può essere installato su qualsiasi sistema operativo basato su Linux o su una macchina virtuale su macOS o Windows. Puoi anche connettere qualsiasi wallet che supporti Bitcoin Core P2P, Bitcoin Core RPC, Electrum o lndconnect.

Puoi andare direttamente su Umbrel per saperne di più.

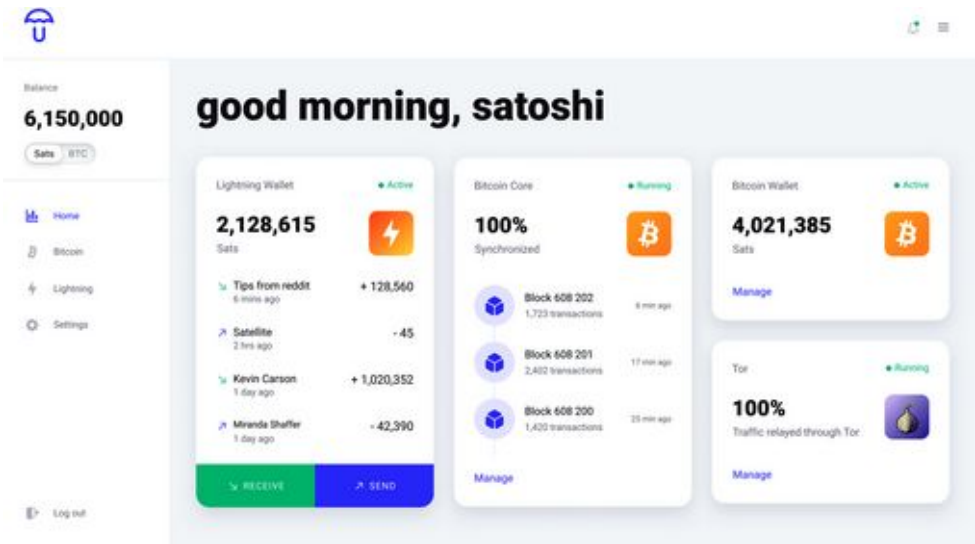

Figura 5-2. L'interfaccia web di Umbrel

BTCPay Server

Sebbene inizialmente non sia stato progettato come "assistente", la piattaforma di e-commerce e pagamento *BTCPay Server* ha un sistema di installazione incredibilmente semplice che utilizza contenitori Docker e `docker-compose` per installare un nodo Bitcoin, un nodo Lightning e un gateway di pagamento, tra molti altri servizi. Può essere installato su una varietà di piattaforme hardware, da un semplice Raspberry Pi 4 (consigliati 4 GB di RAM) a un mini PC o vecchio laptop, desktop o server.

BTCPay Server è un software di e-commerce self-hosted che può essere integrato con molte piattaforme come WordPress, WooCommerce e Shopify. Sebbene originariamente sviluppato come sostituzione del payment processor BitPay, si è evoluto per diventare una piattaforma e-commerce completa per accettare Bitcoin, LN ed altre criptovalute. Per la maggior parte di venditori e negozi è una soluzione e-commerce chiavi in mano.

Oltre a un nodo Bitcoin e Lightning, *RaspiBlitz*, *Mynode*, *Umbrel* e *BTCPay Server* permettono di installare una serie di servizi aggiuntivi, come:

- Tor (eseguito come onion service)
- ElectRS (server Electrum in Rust)
- BTC RPC Explorer (Bitcoin blockchain explorer)
- Ride The Lightning (interfaccia grafica di gestione di nodi Lightning)
- LNbits (wallet/account Lightning)
- Spectre Desktop (wallet multisig per Trezor, Ledger, Coldcard e Spectre-DIY)
- lndmanage (interfaccia a riga di comando per la gestione avanzata dei canali)
- Loop (servizio di submarine swaps)
- JoinMarket (servizio/mercato di liquidità CoinJoin)
- OpenVPN (supporto per rete privata virtuale [VPN] per gestione remota)
- Lightning Terminal (interfaccia di gestione della liquidità, Loop In e Loop Out)
- ThunderHub (per monitorare e gestire il proprio nodo)
- Sphinx Relay (per gestire la connettività e l'archiviazione per la chat Sphinx)
- mempool.space (visualizzatore mempool e block explorer)

Nodo Bitcoin o Nodo Lightning Leggero

Una scelta fondamentale per la tua configurazione sarà la scelta del nodo Bitcoin e la sua configurazione. *Bitcoin Core*, l'implementazione di riferimento, è la scelta più comune ma non l'unica disponibile. Una scelta alternativa è `btcd`, che è un'implementazione in linguaggio Go di un nodo Bitcoin. btcd supporta alcune funzionalità utili per l'esecuzione di un nodo Lightning LND che non sono disponibili in Bitcoin Core.

Una seconda considerazione è se gestirai un nodo Bitcoin di archiviazione (*archival*) con una copia completa della blockchain (circa 520 GB a Febbraio 2023) o una blockchain sfoltita (*pruned*) che conserva solo i blocchi più recenti. Un nodo pruned può farti risparmiare spazio su disco, ma dovrai comunque scaricare l'intera blockchain almeno una volta (durante l'IBD). Ciò non ti farà risparmiare lato rete. L'utilizzo di un nodo pruned per eseguire un nodo Lightning è ancora una funzionalità sperimentale e potrebbe non permettere ogni azione standard. Tuttavia, molte persone gestiscono con successo nodi pruned.

Infine, hai anche la possibilità di non eseguire affatto un nodo Bitcoin. Puoi infatti utilizzare il nodo Lightning LND in modalità "leggera" utilizzando il protocollo Neutrino per recuperare informazioni sulla blockchain da nodi Bitcoin pubblici gestiti da altri. Ciò significa che stai prendendo risorse dalla rete Bitcoin senza offrirne alcuna in cambio. Stai però offrendo le tue risorse e contribuendo alla comunità LN. Per i nodi Lightning più piccoli, questo ridurrà generalmente il traffico di rete rispetto all'esecuzione di un nodo Bitcoin completo.

Tieni presente che il funzionamento di un nodo Bitcoin ti consente di supportare altri servizi oltre a un nodo Lightning. Questi altri servizi potrebbero richiedere un nodo Bitcoin di archiviazione (non pruned) e spesso non possono essere eseguiti senza un nodo Bitcoin. Considera in anticipo quali altri servizi potresti voler eseguire ora o in futuro per prendere una decisione corretta sul tipo di nodo Bitcoin che vuoi lanciare e gestire.

La linea di fondo per questa decisione è: se puoi permetterti un disco più grande di 500 GB, esegui un full node Bitcoin archival. Contribuirai con risorse al sistema Bitcoin e aiuterai gli altri che non possono permetterselo. Se non puoi permetterti un disco così grande, esegui un nodo pruned. Se non puoi permetterti né disco né larghezza di banda nemmeno per un nodo pruned, lancia un nodo LND leggero tramite Neutrino.

Scelta del Sistema Operativo

Il prossimo passo è selezionare un sistema operativo per il tuo nodo. La stragrande maggioranza dei server Internet funziona su qualche variante di Linux. Linux è la piattaforma preferita per Internet perché è un potente sistema operativo open source. Linux, tuttavia, ha una curva di apprendimento molto lunga e richiede familiarità con un ambiente a riga di comando; una cosa che potrebbe sembrare intimidatoria per i nuovi utenti.

A prescindere, la maggior parte dei servizi può essere eseguita su qualsiasi sistema operativo POSIX moderno, inclusi macOS, Windows e, naturalmente, Linux. La tua scelta dovrebbe essere guidata maggiormente dalla tua familiarità e comodità con un sistema operativo e dai tuoi obiettivi di apprendimento. Se vuoi espandere le tue conoscenze e imparare come utilizzare un sistema Linux, questa è una grande opportunità per farlo con un progetto specifico e un obiettivo chiaro. Se vuoi solamente lanciare un nodo, fallo con quello che sai.

Al giorno d'oggi, molti servizi vengono erogati anche sotto forma di contenitori, solitamente basati sul sistema Docker. Questi contenitori possono essere distribuiti su una varietà di sistemi operativi, astraendo il sistema operativo sottostante. Tuttavia, potrebbe essere necessario imparare alcuni comandi della CLI (command line interface) di Linux, poiché la maggior parte dei contenitori esegue all'interno una variante di Linux.

Scegliere una Implementazione di Nodo Lightning

Come per la scelta del sistema operativo, la scelta dell'implementazione del nodo Lightning dovrebbe dipendere principalmente dalla familiarità con il linguaggio di programmazione e gli strumenti di sviluppo utilizzati dai progetti. Sebbene ci siano alcune piccole differenze nelle funzionalità tra le varie implementazioni dei nodi, la maggior parte delle implementazioni convergono sugli standard comuni definiti dai BOLT.

La familiarità con il linguaggio di programmazione e il sistema di compilazione, invece, è una buona base per la scelta di un nodo. Questo perché l'installazione, la configurazione, la manutenzione continua e la risoluzione dei problemi implicano l'interazione con i vari strumenti utilizzati dal sistema di compilazione. Ciò comprende:

- Make, Autotools e utilities GNU per `c-lightning`

- Utilities Go per LND
- Java/Maven per Eclair

Il linguaggio di programmazione influenza non solo la scelta del sistema di compilazione ma anche molti altri aspetti del programma. Ogni linguaggio di programmazione è dotato di un'intera filosofia di progettazione e influisce su molti altri aspetti, come:

- Formato e sintassi dei file di configurazione
- Posizioni dei file (nel filesystem)
- Argomenti della riga di comando e la loro sintassi
- Formattazione dei messaggi di errore
- Librerie richieste
- Interfacce di chiamata di procedura remota (RPC)

Quando scegli il tuo nodo, pensa a tutte le caratteristiche elencate. La tua familiarità con questi strumenti e le filosofie di progettazione semplificheranno l'esecuzione di un nodo.

Se questa è la tua prima volta nel mondo della riga di comando e dei server, ti ritroverai a non avere familiarità con alcuna implementazione e avrai l'opportunità di imparare qualcosa di nuovo. In tal caso potresti decidere in base a una serie di altri fattori, come ad esempio:

- Qualità dei forum di supporto, dei gruppi delle chat
- Qualità della documentazione
- Grado di integrazione con altri strumenti che vuoi e vorrai eseguire

Come considerazione finale, potresti voler esaminare le prestazioni e l'affidabilità delle diverse implementazioni di nodi. Questo è particolarmente importante se utilizzerai questo nodo in un ambiente di produzione e ti aspetti un traffico intenso e requisiti di alta affidabilità. Questo potrebbe essere il caso se prevedi di eseguire il nodo come sistema di pagamento per il tuo negozio.

Installazione di un Nodo Bitcoin o Lightning

Hai deciso di non utilizzare un "assistente" di installazione ma di tuffarti nella riga di comando di un sistema operativo Linux? Questa è una decisione coraggiosa e cercheremo di

aiutarti. Se preferisci non farlo manualmente, prendi in considerazione l'utilizzo di un'applicazione che ti aiuti a installare il software del nodo o una soluzione basata su contenitori, come descritto nei capitoli precedenti.

ATTENZIONE	Questa sezione approfondirà l'argomento avanzato dell'amministrazione di sistema tramite riga di comando. L'amministrazione di Linux è un insieme di competenze che esula dagli scopi di questo libro. È un argomento complicato e ci sono molti rischi. Procedi con cautela!

Nelle prossime sezioni descriveremo brevemente come installare e configurare un nodo Bitcoin e Lightning su un sistema operativo Linux. Dovrai rivedere le istruzioni di installazione per le specifiche applicazioni del nodo Bitcoin e Lightning che hai deciso di utilizzare. Di solito puoi trovarli in un file chiamato *INSTALL* o nella sottodirectory *docs* di ogni progetto. Descriveremo solo alcuni dei passaggi comuni che si applicano a tutti questi servizi e le istruzioni che offriamo saranno per forza di cose incomplete.

Servizi in Background

Per chi è abituato a eseguire applicazioni sul proprio desktop o smartphone, un'applicazione ha sempre un'interfaccia utente grafica anche se a volte può essere eseguita in background. Le applicazioni dei nodi Bitcoin e Lightning, tuttavia, sono molto diverse. Queste applicazioni non dispongono di un'interfaccia utente grafica incorporata. Invece, vengono eseguite come servizi in background *headless*, il che significa che operano sempre in background e non interagiscono direttamente con l'utente.

Ciò può creare confusione per gli utenti che non sono abituati a eseguire servizi in background. Come fai a sapere se tale servizio è attualmente in esecuzione? Come lo avvii e lo fermi? Come interagisci con esso? Le risposte a queste domande dipendono dal sistema operativo in uso. Per ora supponiamo che tu stia utilizzando una variante di Linux.

Isolamento dei Processi

I servizi in background di solito vengono eseguiti con un account utente specifico per isolarli dal sistema operativo e tra di loro. Ad esempio, Bitcoin Core è configurato per funzionare

come utente `bitcoin`. Dovrai utilizzare la riga di comando per creare un utente per ciascuno dei servizi che esegui.

Inoltre, se hai collegato un'unità esterna, dovrai dire al sistema operativo di riposizionare la home directory dell'utente su quell'unità. Questo perché un servizio come Bitcoin Core creerà file nella home directory dell'utente. Se lo stai configurando per scaricare l'intera blockchain, tali file occuperanno diverse centinaia di gigabyte. Supponiamo che tu abbia collegato l'unità esterna e che si trovi nel percorso */external_drive/* del sistema operativo.

Sulla maggior parte dei sistemi Linux puoi creare un nuovo utente con il comando `useradd`:

```
$ sudo useradd -m -d /external_drive/bitcoin -s /dev/null bitcoin
```

I flag `m` e `d` creano la home directory dell'utente come specificato da */external_drive/bitcoin*. Il flag `s` assegna la shell interattiva dell'utente. Lo impostiamo su */dev/null* per disabilitare l'uso interattivo della shell. L'ultimo argomento è il nome `bitcoin` del nuovo utente.

Avvio del Nodo

Per entrambi i servizi Bitcoin e Lightning, l'"installazione" implica anche la creazione di un cosiddetto *script di avvio* per assicurarsi che il nodo si avvii all'avvio del computer. L'avvio e l'arresto dei servizi in background è gestito da un processo del sistema operativo, che in Linux è chiamato `init` o `systemd`. Di solito puoi trovare uno script di avvio del sistema nella sottodirectory `contrib` di ogni progetto. Ad esempio, se utilizzi un sistema operativo Linux che utilizza `systemd`, troverai uno script chiamato *bitcoind.service* che può avviare e arrestare il servizio del nodo Bitcoin Core.

Ecco un esempio di come appare lo script di avvio di un nodo Bitcoin, tratto dal repository di codice Bitcoin Core (bitcoin/contrib/init/bitcoind.service):

```
[Unit]
Description=Bitcoin daemon
After=network.target
[Service]
ExecStart=/usr/bin/bitcoind -daemon \
                            -pid=/run/bitcoind/bitcoind.pid \
                            -conf=/etc/bitcoin/bitcoin.conf \
```

```
                                -datadir=/var/lib/bitcoind
# Make sure the config directory is readable by the service user
PermissionsStartOnly=true
ExecStartPre=/bin/chgrp bitcoin /etc/bitcoin
# Process management
####################
Type=forking
PIDFile=/run/bitcoind/bitcoind.pid
Restart=on-failure
TimeoutStopSec=600
# Directory creation and permissions
####################################
# Run as bitcoin:bitcoin
User=bitcoin
Group=bitcoin
# /run/bitcoind
RuntimeDirectory=bitcoind
RuntimeDirectoryMode=0710
# /etc/bitcoin
ConfigurationDirectory=bitcoin
ConfigurationDirectoryMode=0710
# /var/lib/bitcoind
StateDirectory=bitcoind
StateDirectoryMode=0710
[...]
[Install]
WantedBy=multi-user.target
```

Come utente `root`, installa lo script copiandolo nella cartella del servizio `systemd` */lib/systemd/system/* e quindi ricarica `systemd`:

```
$ sudo systemctl daemon-reload
```

Successivamente, abilita il servizio:

```
$ sudo systemctl enable bitcoind
```

Ora puoi avviare e interrompere il servizio. Non avviarlo ancora, poiché non abbiamo configurato il nodo Bitcoin.

```
$ sudo systemctl start bitcoind
```

```
$ sudo systemctl stop bitcoind
```

Configurazione del Nodo

Per configurare il tuo nodo, devi creare e fare riferimento a un file di configurazione. Per convenzione, questo file viene solitamente creato in */etc*, in una directory con il nome del programma. Ad esempio, le configurazioni di Bitcoin Core e LND vengono solitamente memorizzate rispettivamente in */etc/bitcoin/bitcoin.conf* e */etc/lnd/lnd.conf*.

Questi file di configurazione sono file di testo con ogni riga che esprime un'opzione di configurazione e il suo valore. I valori predefiniti vengono assunti per tutto ciò che non è definito nel file di configurazione. Puoi vedere quali opzioni possono essere impostate nella configurazione in due modi. Innanzitutto, l'esecuzione dell'applicazione del nodo seguito dal comando `help` mostrerà le opzioni che possono essere definite sulla riga di comando. Queste stesse opzioni possono essere definite nel file di configurazione. In secondo luogo, di solito è possibile trovare un file di configurazione di esempio, con tutte le opzioni predefinite, nel repository di codice del software.

Puoi trovare un esempio di configurazione in ciascuna delle immagini Docker che abbiamo utilizzato nel Capitolo 4. Ad esempio, il file *code/docker/bitcoind/bitcoind/bitcoin.conf*:

```
regtest=1
server=1
debuglogfile=debug.log
debug=1
txindex=1
printtoconsole=0
[regtest]
fallbackfee=0.000001
port=18444
```

```
noconnect=1
dnsseed=0
dns=0
upnp=0
onlynet=ipv4
rpcport=18443
rpcbind=0.0.0.0
rpcallowip=0.0.0.0/0
rpcuser=regtest
rpcpassword=regtest
zmqpubrawblock=tcp://0.0.0.0:12005
zmqpubrawtx=tcp://0.0.0.0:12006
```

Quel particolare file configura Bitcoin Core per funzionare come nodo `regtest` e fornisce un nome utente e una password deboli per l'accesso remoto, quindi non dovresti usarlo per la configurazione del tuo nodo. Tuttavia, serve per illustrare la sintassi di un file di configurazione e puoi apportarvi modifiche nel contenitore Docker per sperimentare diverse opzioni. Verifica se puoi utilizzare il comando `bitcoind -help` per capire cosa fa ciascuna delle opzioni nel contesto della rete Docker che abbiamo creato nel Capitolo 4.

Spesso le impostazioni predefinite sono sufficienti e, con poche modifiche, il software del nodo può essere configurato rapidamente. Per far funzionare un nodo Bitcoin Core con una personalizzazione minima, sono necessarie solo quattro righe di configurazione:

```
server=1
daemon=1
txindex=1
rpcuser=USERNAME
rpcpassword=PASSWORD
```

Anche l'opzione `txindex` non è strettamente necessaria, sebbene assicurerà che il tuo nodo Bitcoin crei un indice di tutte le transazioni, che è richiesto per alcune applicazioni. In ogni caso, l'opzione `txindex` non è necessaria per eseguire un nodo Lightning.

Anche un nodo `c-lightning` in esecuzione sullo stesso server richiede solo poche righe nella configurazione:

```
network=mainnet
bitcoin-rpcuser=USERNAME
bitcoin-rpcpassword=PASSWORD
```

In generale, è una buona idea ridurre al minimo la quantità di personalizzazione di questi sistemi. La configurazione predefinita è attentamente progettata per supportare le distribuzioni più comuni. Se modifichi un valore predefinito, potrebbe causare problemi in seguito o ridurre le prestazioni del tuo nodo. Insomma, modificare solo quando necessario!

Configurazione di Rete

La configurazione di rete normalmente non è un problema quando si configura una nuova applicazione. Tuttavia, le reti peer-to-peer come Bitcoin e Lightning Network presentano alcune sfide uniche per la configurazione della rete.

In un servizio centralizzato, il tuo computer si connette ai "grandi server" di qualche azienda, e non viceversa. La tua connessione Internet domestica è in realtà configurata partendo dal presupposto che tu sia semplicemente un consumatore di servizi forniti da altri. Ma in un sistema peer-to-peer, ogni peer consuma e fornisce servizi ad altri nodi. Se stai eseguendo un nodo Bitcoin o Lightning a casa tua, stai fornendo un servizio ad altri computer su Internet. Il tuo servizio Internet per impostazione predefinita non è configurato per consentirti di eseguire dei server e potrebbe richiedere una configurazione aggiuntiva per consentire ad altri di raggiungere il tuo nodo.

Se desideri eseguire un nodo Bitcoin o Lightning, devi consentire ad altri nodi su Internet di connettersi a te. Ciò significa abilitare le connessioni TCP in entrata alla porta Bitcoin (8333 per impostazione predefinita) o alla porta Lightning (9735 per impostazione predefinita). Mentre puoi eseguire un nodo Bitcoin senza connettività in entrata, non puoi farlo con un nodo Lightning. Un nodo Lightning deve essere accessibile ad altri dall'esterno della tua rete.

Per impostazione predefinita, il router Internet di casa non si aspetta connessioni in entrata dall'esterno e infatti le connessioni in entrata sono bloccate. L'indirizzo IP del tuo router Internet è l'unico indirizzo IP accessibile dall'esterno e tutti i computer che esegui all'interno della tua rete domestica condividono quel singolo indirizzo IP. Ciò si ottiene grazie a un meccanismo chiamato *Network Address Translation* (*NAT*), che consente al router Internet di fungere da intermediario per tutte le connessioni in uscita. Se si desidera consentire una connessione in entrata, è necessario configurare il *port forwarding*, che indica al router

Internet che le connessioni in entrata su porte specifiche devono essere inoltrate a computer specifici all'interno della rete. Puoi farlo manualmente modificando la configurazione del tuo router Internet o, se il tuo router lo supporta, attraverso un meccanismo di port forwarding automatico chiamato *Universal Plug and Play* (*UPnP*).

Un meccanismo alternativo al port forwarding consiste nell'abilitare The Onion Router (Tor), che fornisce una sorta di overlay di rete privata virtuale che consente le connessioni in entrata a un *indirizzo onion*. Se esegui Tor, non è necessario eseguire il port forwarding o abilitare le connessioni in entrata alle porte Bitcoin o Lightning. Se esegui i tuoi nodi utilizzando Tor, tutto il traffico passa attraverso Tor e non vengono utilizzate altre porte.

Diamo un'occhiata ai diversi modi in cui puoi consentire ad altri di connettersi al tuo nodo. Esamineremo queste alternative in ordine, dalla più semplice alla più difficile.

Funziona e basta!

È possibile che il tuo provider di servizi Internet o router sia configurato per supportare UPnP per impostazione predefinita e tutto funzioni automaticamente. Proviamo prima questo approccio, nel caso fossimo fortunati.

Supponendo che tu abbia già un nodo Bitcoin o Lightning in esecuzione, proveremo a vedere se sono accessibili dall'esterno.

NOTA	Affinché questo test funzioni, devi avere un nodo Bitcoin o Lightning (o entrambi) attivo e funzionante sulla tua rete domestica. Se il tuo router supporta UPnP, il traffico in entrata verrà automaticamente inoltrato alle porte corrispondenti sul computer che esegue il nodo.

Puoi utilizzare alcuni siti Web per scoprire qual è il tuo indirizzo IP esterno e se consente e inoltra le connessioni in entrata a una porta nota. Eccone due affidabili:

- https://canyouseeme.org
- https://www.whatismyip.com/port-scanner

Per impostazione predefinita, questi servizi ti consentono solo di controllare le connessioni in entrata all'indirizzo IP da cui ti stai connettendo. Questo viene fatto per impedirti di utilizzare il servizio per scansionare reti e computer di altre persone. Vedrai l'indirizzo IP esterno del

tuo router e un campo per inserire una porta. Se non hai modificato le porte predefinite nella configurazione del tuo nodo, prova la porta 8333 (Bitcoin) e/o 9735 (Lightning).

Nella Figura 5-3 puoi vedere il risultato del controllo della porta 9735 su un server che esegue Lightning, utilizzando lo strumento di scansione delle porte *whatismyip.com*. Mostra che il server accetta connessioni in entrata alla porta Lightning.

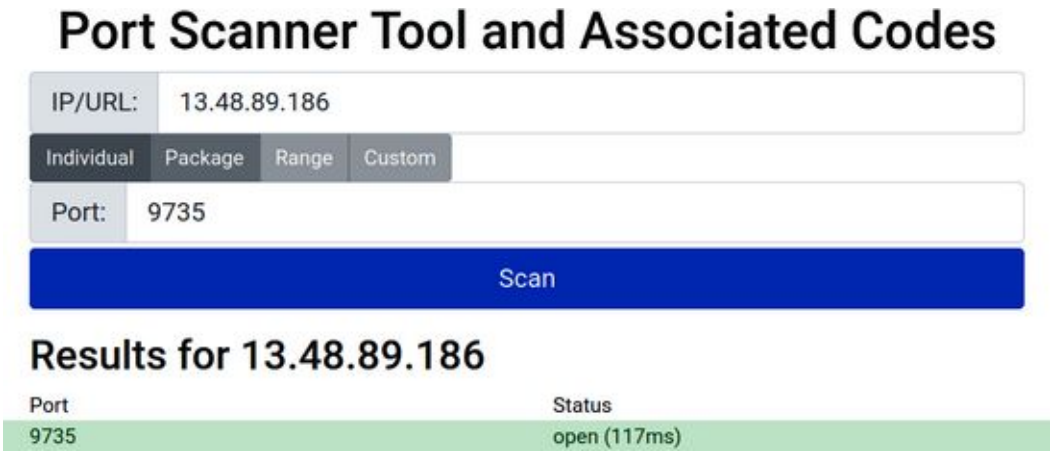

Figura 5-3. Verifica della porta in entrata 9735

Port forwarding automatico tramite UPnP

A volte, anche se il tuo router Internet supporta UPnP, potrebbe essere disattivato per impostazione predefinita. In tal caso è necessario modificare la configurazione del router Internet dalla sua interfaccia di amministrazione web:

1. Connettiti alla pagina di configurazione del tuo router. Di solito questo può essere fatto collegandosi all'*indirizzo del gateway* della rete utilizzando un browser web. Puoi trovare l'*indirizzo del gateway* guardando la configurazione IP di qualsiasi computer sulla tua rete. Spesso è il primo indirizzo in una delle reti non instradabili, come 192.168.0.1 o 10.0.0.1. Controlla anche gli adesivi sul router per trovare l'indirizzo del gateway. Una volta trovato, apri un browser e inserisci l'indirizzo IP nell'URL/casella di ricerca del browser, ad esempio "192.168.0.1" o "http://192.168.0.1".
2. Trova il nome utente e la password dell'amministratore per il pannello di configurazione web del router. Questo è spesso scritto su un adesivo sul router stesso e/o può essere semplice come "admin" e "password". Anche una rapida ricerca sul web per il tuo ISP e il modello di router può aiutarti a trovare queste informazioni.

3. Trova un'impostazione per UPnP e attivala.

Riavvia il tuo nodo Bitcoin e/o Lightning e ripeti il test della porta aperta con uno dei siti Web che abbiamo utilizzato nella sezione precedente.

Utilizzo di Tor per le connessioni in entrata

The Onion Router (*Tor*) è una VPN con la proprietà speciale di crittografare le comunicazioni tra i vari passaggi, in modo tale che qualsiasi nodo intermedio non possa determinare l'origine o la destinazione di un pacchetto. Entrambi i nodi Bitcoin e Lightning supportano il funzionamento su Tor, che ti consente di gestire un nodo senza rivelare il tuo indirizzo IP o la tua posizione geografica. Pertanto, fornisce un elevato livello di privacy al traffico di rete. Un ulteriore vantaggio dell'esecuzione di Tor è che, poiché funziona come VPN, risolve il problema del port forwarding dal tuo router Internet. Le connessioni in entrata vengono ricevute tramite il tunnel Tor e il tuo nodo può essere trovato tramite un *indirizzo onion* generato ad hoc anziché un indirizzo IP.

Abilitare Tor richiede due passaggi. Innanzitutto, devi installare il router Tor e il proxy sul tuo computer. Poi devi abilitare l'uso del proxy Tor nella tua configurazione Bitcoin o Lightning.

Per installare Tor su un sistema Ubuntu Linux che utilizza il gestore di pacchetti `apt`, esegui:

```
sudo apt install tor
```

Successivamente, configuriamo il nostro nodo Lightning per utilizzare Tor per la sua connettività esterna. Ecco un esempio di configurazione per LND:

```
[Tor]
tor.active=true
tor.v3=true
tor.streamisolation=true
listen=localhost
```

Ciò abiliterà Tor (`tor.active`), stabilirà un servizio onion v3 (`tor.v3=true`), utilizzerà un flusso onion diverso per ogni connessione (`tor.streamisolation`) e limiterà l'ascolto delle connessioni solo all'host locale, per evitare divulgare il tuo indirizzo IP (`listen=localhost`).

Puoi verificare se Tor è installato correttamente e funziona eseguendo un semplice comando di una riga. Questo comando dovrebbe funzionare sulla maggior parte delle versioni di Linux:

```
curl --socks5 localhost:9050 --socks5-hostname localhost:9050 -s
https://check.torproject.org/ | cat | grep -m 1 Congratulations | xargs
```

Se tutto funziona correttamente, la risposta di questo comando dovrebbe essere "`Congratulations. This browser is configured to use Tor.`".

A causa della natura di Tor, non puoi utilizzare facilmente un servizio esterno per verificare se il tuo nodo è raggiungibile tramite un indirizzo onion. Tuttavia, dovresti vedere il tuo indirizzo Tor onion nei log del tuo nodo Lightning. È una lunga stringa di lettere e numeri seguita dal suffisso `.onion`. Il tuo nodo ora dovrebbe essere raggiungibile da Internet, con l'ulteriore vantaggio della privacy!

Inoltro manuale della porta

Questo è il processo più complesso e richiede un po' di abilità tecnica. I dettagli dipendono dal tipo di router Internet che possiedi, dalle impostazioni e dai criteri del tuo fornitore di servizi e da molti altri contesti. Prova inizialmente UPnP o Tor, prima di testare questo meccanismo molto più complicato.

I passaggi fondamentali sono i seguenti:

1. Trova l'indirizzo IP del computer su cui si trova il tuo nodo. Questo di solito è allocato dinamicamente dal protocollo DHCP (Dynamic Host Configuration Protocol) ed è spesso da qualche parte nell'intervallo di IP 192.168.x.x o 10.x.x.x.
2. Trova l'indirizzo MAC (Media Access Control) dell'interfaccia di rete del tuo nodo. Questo può essere trovato nelle impostazioni Internet di quel computer.
1. Assegna un indirizzo IP statico al tuo nodo in modo che sia sempre lo stesso. Puoi usare l'indirizzo IP che ha attualmente. Sul tuo router Internet, cerca "Static Leases" sotto la configurazione DHCP. Associa l'indirizzo MAC all'indirizzo IP selezionato. Ora al tuo nodo sarà sempre assegnato quell'indirizzo IP. In alternativa, puoi guardare la configurazione DHCP del tuo router e scoprire qual è il suo intervallo di indirizzi DHCP. Selezionare un indirizzo inutilizzato *al di fuori* dell'intervallo di indirizzi DHCP. Quindi, sul server, configurare la rete per interrompere l'utilizzo di DHCP e codificare l'indirizzo IP non DHCP selezionato nella configurazione di rete del sistema operativo.
2. Infine, imposta "Port Forwarding" sul tuo router Internet per instradare il traffico in entrata su porte specifiche all'indirizzo IP selezionato del tuo server.

Al termine della riconfigurazione, ripeti il controllo della porta.

Sicurezza del tuo Nodo

Un nodo Lightning è, per definizione, un *hot wallet*. Ciò significa che i fondi (sia on-chain che off-chain) controllati da un nodo Lightning sono controllati direttamente da chiavi che vengono caricate nella memoria del nodo o archiviate sull'hard disk del nodo. Se un nodo Lightning viene compromesso, è facile creare transazioni on-chain o off-chain per drenarne i fondi. È quindi di fondamentale importanza proteggerlo da accessi non autorizzati.

La sicurezza è uno sforzo olistico, il che significa che devi proteggere ogni livello di un sistema. Come si suol dire: la catena è forte quanto l'anello più debole. Questo è un concetto importante nella sicurezza delle informazioni e lo applicheremo al nostro nodo.

Nonostante tutte le misure di sicurezza che adotterai, ricorda che Lightning Network è una tecnologia sperimentale in fase iniziale ed è probabile che ci siano bug sfruttabili nel codice di qualsiasi progetto che utilizzi. *Non mettere più denaro di quanto sei disposto a perdere*.

Sicurezza del Sistema Operativo

La protezione di un sistema operativo è un argomento vasto che va oltre lo scopo di questo libro. Tuttavia, possiamo stabilire alcuni principi di base.

Per proteggere il tuo sistema operativo, ecco alcuni degli elementi principali da considerare:

Provenienza

> Inizia assicurandoti di scaricare l'immagine del sistema operativo corretta e verifica eventuali firme o checksum prima di installarlo. Estendilo a qualsiasi software che installi. Ricontrolla qualsiasi fonte o URL da cui scarichi. Verifica l'integrità e la correttezza del software scaricato tramite la verifica della firma e del checksum.

Manutenzione

> Assicurati di mantenere aggiornato il tuo sistema operativo. Abilita l'installazione automatica degli aggiornamenti di sicurezza. Privilegio minimo: imposta gli utenti per processi specifici e concedi loro l'accesso minimo necessario per eseguire un servizio.

Non eseguire processi con privilegi di amministratore (ad es. `root`).

Isolamento del processo

Utilizza le funzionalità del sistema operativo per isolare i processi gli uni dagli altri.

Autorizzazioni del filesystem

Configura attentamente il filesystem in base al principio del privilegio minimo. Non rendere i file leggibili o scrivibili da tutti.

Autenticazione forte

Utilizza password complesse generate casualmente o, quando possibile, autenticazione a chiave pubblica. Ad esempio, è più sicuro utilizzare Secure Shell (SSH) con una coppia di chiavi crittografiche anziché una password.

Autenticazione a due fattori (2FA)

Utilizza l'autenticazione a due fattori ove possibile, incluso Universal 2nd Factor (U2F) con chiavi di sicurezza hardware. Questo vale per tutti i servizi esterni che potresti utilizzare, come il tuo fornitore di servizi cloud. Puoi applicarlo anche alla tua configurazione, come la tua configurazione SSH. Usa 2FA anche per i servizi indiretti. Immagina di utilizzare un servizio cloud. Hai fornito al tuo fornitore di servizi cloud un indirizzo e-mail, quindi dovresti anche proteggere il tuo indirizzo e-mail con 2FA.

Backup

Crea backup del tuo sistema e assicurati di proteggere anche i backup con la crittografia. Esegui questi backup periodicamente. Almeno una volta, verifica se puoi ripristinare il backup e che il backup sia completo e accessibile. Se possibile, conserva una copia dei backup su un disco diverso per evitare che un singolo errore del disco distrugga sia il nodo attivo che le copie di backup.

Vulnerabilità e gestione dell'esposizione

> Usa la scansione remota per assicurarti di aver ridotto al minimo la superficie di attacco del tuo sistema. Chiudi eventuali servizi o porte non necessarie. Installa solo software e pacchetti di cui hai veramente bisogno e che usi. Disinstalla i pacchetti che non usi più. Non utilizzare il computer nodo per attività al di fuori del tuo nodo che puoi eseguire su un altro computer. Soprattutto, se puoi, non utilizzare il computer del tuo nodo per navigare su Internet o leggere la tua posta elettronica.

Questo è un elenco non esaustivo delle misure di sicurezza più basilari.

Accesso al Nodo

Il tuo nodo Lightning esporrà un'API RPC (Remote Procedure Call). Ciò significa che il tuo nodo può essere controllato da remoto tramite comandi inviati a una specifica porta TCP. Il controllo dell'accesso a tale API RPC viene ottenuto mediante una qualche forma di autenticazione dell'utente. A seconda del tipo di nodo Lightning configurato, ciò avverrà tramite l'autenticazione con nome utente/password o tramite un meccanismo chiamato *macaroon* di autenticazione. Come suggerisce il nome, un macaroon (amaretto) è un tipo di cookie (biscotto) più sofisticato. A differenza di un cookie, è firmato crittograficamente e può esprimere un insieme di capacità di accesso.

Ad esempio, LND utilizza i macaroon per concedere l'accesso all'API RPC. Per impostazione predefinita, il software LND crea tre macaroon con diversi livelli di accesso, denominati `admin`, `invoice` e `readonly`. A seconda del macaroon che copi e utilizzi nel tuo client RPC, hai accesso in *sola lettura*, accesso alle *fatture* (che include le funzionalità di sola lettura) o accesso *amministratore*, che ti dà pieno controllo. C'è anche una funzione di `bakery` in LND che può costruire macaroon con qualsiasi combinazione con un controllo molto granulare.

Se utilizzi un modello di autenticazione con nome utente/password, assicurati di selezionare una password lunga e casuale. Non dovrai digitare spesso questa password, perché verrà memorizzata nei file di configurazione. Dovresti quindi sceglierne una che non può essere indovinata. Molti degli esempi che vedrai includono password deboli e spesso le persone le copiano nei propri sistemi, fornendo un facile accesso a chiunque. Non farlo! Usa un gestore di password per generare una lunga password alfanumerica casuale. Poiché alcuni caratteri speciali come `$?/!*\&%`"'` possono interferire con la riga di comando, è meglio evitarli per

le password che verranno utilizzate in un ambiente shell. Per evitare problemi, attieniti a caratteri alfanumerici casuali lunghi per le tue password.

Di solito è sufficiente una semplice sequenza alfanumerica più lunga di 12 caratteri e generata casualmente. Se prevedi di archiviare grandi quantità di denaro sul tuo nodo Lightning e sei preoccupato per gli attacchi di forza bruta remoti, seleziona una password di lunghezza superiore a 20 caratteri per rendere tali attacchi praticamente irrealizzabili.

Backup di Nodi e Canali

Una considerazione molto importante quando si esegue un nodo Lightning è il problema dei backup. A differenza di un portafoglio Bitcoin, dove una frase mnemonica BIP-39 può recuperare tutto lo stato del portafoglio, in Lightning *non* è così.

I portafogli Lightning utilizzano un backup della frase mnemonica BIP-39, ma solo per il portafoglio on-chain. Tuttavia, a causa del modo in cui sono costruiti i canali, la frase mnemonica *non* è sufficiente per ripristinare un nodo Lightning. È necessario un ulteriore livello di backup, denominato *backup del canale statico* (SCB – *Static Channel Backup*). Senza un SCB, un operatore del nodo Lightning potrebbe perdere tutti i fondi presenti nei canali se perde l'archivio dati del nodo stesso.

ATTENZIONE	*Non* finanziare i canali fino a quando non avrai messo in atto un sistema per il backup continuo dello stato dei tuoi canali. I tuoi backup dovrebbero essere spostati "fuori sede" su un sistema e un luogo diversi dal tuo nodo, in modo che possano sopravvivere a una serie di guasti del sistema (mancanza di corrente, danneggiamento dei dati, ecc.) o disastri naturali (inondazioni, incendi, ecc.).

Gli SCB non sono una panacea. Innanzitutto, è necessario eseguire il backup dello stato di ciascun canale ogni volta che si verifica una nuova transazione di impegno. In secondo luogo, il ripristino da un backup del canale è pericoloso. Se non disponi dell'ultima transazione di impegno e trasmetti accidentalmente un vecchio impegno (revocato), il tuo partner di canale presumerà che stai tentando di imbrogliarlo e richiederà l'intero saldo del canale con una transazione di penalità. Per assicurarti di chiudere il canale, devi eseguire una *chiusura cooperativa*. Un peer malevolo potrebbe indurre in errore il tuo nodo a trasmettere un

vecchio impegno revocato durante quella chiusura cooperativa, facendo sì che il tuo nodo provi inavvertitamente a imbrogliare.

Inoltre, i backup dei tuoi canali devono essere crittografati per mantenere la tua privacy e la sicurezza degli stessi. Altrimenti, chiunque trovi i backup non solo può vedere tutti i tuoi canali, ma potrebbe anche utilizzare i backup per chiudere tutti i tuoi canali in modo da consegnare il saldo ai tuoi partner di canale. In altre parole, una persona malevola che ottiene l'accesso ai tuoi backup può farti perdere tutti i fondi dei tuoi canali.

Puoi vedere che gli SCB non sono una salvaguardia infallibile. Sono un compromesso debole perché scambiano un tipo di rischio (corruzione o perdita di dati) con un altro tipo di rischio (peer malevolo). Per ripristinare da un SCB, devi interagire con i tuoi partner del canale e sperare che non tentino di imbrogliarti dandoti un vecchio impegno o ingannando il tuo nodo facendogli trasmettere un impegno revocato in modo che possano penalizzarti. Nonostante i punti deboli di SCB, hanno senso e dovresti sfruttarli. Se non salvi gli SCB e perdi i dati del tuo nodo, perderai per sempre i fondi dei tuoi canali. Tuttavia, se esegui il backup dei SCB e perdi i dati del tuo nodo, allora hai una ragionevole possibilità che alcuni dei tuoi partner di canale siano onesti e che tu possa recuperare parte dei fondi dei tuoi canali. Se sei fortunato, potresti recuperare tutti i tuoi fondi. In conclusione, è consigliabile eseguire SCB continui su un disco diverso dal disco rigido del nodo primario.

I meccanismi di backup del canale sono ancora un lavoro in corso e un punto debole nella maggior parte delle implementazioni Lightning.

I backup basati su file dei database dei nodi Lightning sono nella migliore delle ipotesi una soluzione parziale perché si corre il rischio di eseguire il backup di uno stato del database incoerente. Inoltre, potresti non cogliere in modo affidabile gli ultimi impegni. È molto meglio disporre di un meccanismo di backup che viene attivato ogni volta che si verifica un cambiamento di stato in un canale, garantendo così la coerenza dei dati.

Per configurare gli SCB in LND, imposta il parametro `backupfilepath` sulla riga di comando o nel file di configurazione. LND salverà quindi un file SCB in quella cartella. Successivamente imposta un meccanismo che monitori questo file. Ogni volta che il file cambia, il meccanismo di backup deve copiare questo file su un altro disco, preferibilmente fuori sede. Tali meccanismi di backup vanno oltre lo scopo di questo libro. Tuttavia, qualsiasi sofisticata soluzione di backup dovrebbe essere in grado di gestire questo scenario. Ricorda, anche i file di backup dovrebbero essere crittografati.

Rischi degli Hot Wallet

Come abbiamo discusso in precedenza, Lightning Network è costituito da una rete di *hot wallet* (portafogli caldi). I fondi che conservi in un portafoglio Lightning sono sempre online. Questo li rende vulnerabili. Pertanto, non dovresti conservarne grandi quantità. Grandi quantità dovrebbero essere conservate in un *cold* wallet (portafoglio freddo) che *non* è online e che può effettuare solo transazioni on-chain.

Anche se inizi con poco, con il passare del tempo potresti comunque scoprire di avere una notevole quantità di denaro in un portafoglio Lightning. Questo è uno scenario tipico per i proprietari di negozi. Se utilizzi un nodo Lightning per il tuo e-commerce, il tuo portafoglio probabilmente riceverà spesso fondi, ma li invierà raramente. Finirai quindi per avere due problemi: innanzitutto, i tuoi canali saranno sbilanciati, con grandi saldi locali che superano i piccoli saldi remoti; in secondo luogo, avrai troppi soldi nel portafoglio. Fortunatamente, puoi risolvere entrambi questi problemi contemporaneamente.

Vediamo alcune delle soluzioni che puoi utilizzare per ridurre i fondi esposti in un hot wallet.

Sweep dei Fondi

Se il saldo del tuo portafoglio Lightning diventa troppo grande per la tua tolleranza al rischio, dovrai "spazzolare" (sweep) i fondi dal portafoglio. Puoi farlo in tre modi: on-chain, off-chain e Loop Out. Diamo un'occhiata a ciascuna di queste opzioni nelle prossime sezioni.

Sweep on-chain

Lo sweep di fondi on-chain si ottiene spostando i fondi dal portafoglio Lightning a un portafoglio Bitcoin. Lo fai chiudendo i canali. Quando chiudi un canale, tutti i fondi del tuo saldo locale vengono "spostati" a un indirizzo Bitcoin. L'indirizzo Bitcoin per i fondi on-chain è solitamente generato dal tuo portafoglio Lightning, quindi è ancora un portafoglio caldo. Potrebbe essere necessario eseguire un'ulteriore transazione on-chain per trasferire i fondi a un indirizzo più sicuro, come quello generato sul tuo hardware wallet.

La chiusura dei canali comporterà una commissione on-chain e ridurrà la capacità e la connettività del nodo Lightning. Tuttavia, se gestisci un e-commerce popolare, non ti mancherà la capacità in entrata e potrai chiudere strategicamente i canali con grandi saldi locali; essenzialmente "raggruppando" i tuoi fondi on-chain. Potrebbe essere necessario

utilizzare alcune tecniche di ribilanciamento dei canali prima di chiudere gli stessi, per massimizzare i vantaggi di questa strategia.

Sweep off-chain

Un'altra tecnica che puoi utilizzare prevede l'esecuzione di un secondo nodo Lightning non pubblicizzato sulla rete. Puoi stabilire canali di grande capacità dal tuo nodo pubblico (ad esempio, quello che gestisce il tuo negozio) al tuo nodo non pubblicizzato (nascosto). Su base regolare, "raccogli" i fondi effettuando un pagamento Lightning al tuo nodo nascosto.

Il vantaggio di questa tecnica sta nel fatto che il nodo Lightning che riceve i pagamenti per il tuo negozio sarà di dominio pubblico. Questo lo rende un bersaglio per gli hacker, poiché si presume che qualsiasi nodo Lightning associato a un negozio abbia un saldo elevato. Un secondo nodo non associato al tuo negozio non sarà facilmente identificato come un target.

Come ulteriore misura di sicurezza, puoi rendere il tuo secondo nodo un servizio Tor nascosto in modo che il suo indirizzo IP non sia noto. Ciò riduce ulteriormente la possibilità di attacchi e aumenta la tua privacy.

Sarà necessario impostare uno script che venga eseguito a intervalli regolari. Lo scopo di questo script è creare una invoice sul tuo nodo nascosto e pagare quella fattura dal nodo del tuo negozio, spostando così i fondi sul tuo nodo nascosto.

Tieni presente che questa tecnica non sposta i fondi in un cold wallet. Entrambi i nodi Lightning sono hot wallet. L'obiettivo di questo sweep è spostare fondi da un hot wallet noto ad un hot wallet privato.

Submarine swap

Un altro modo per ridurre il saldo del tuo hot wallet Lightning è utilizzare una tecnica chiamata submarine swap. I submarine swap, concettualizzati da Olaoluwa Osuntokun e Alex Bosworth, consentono lo scambio di bitcoin on-chain con pagamenti Lightning e viceversa. I submarine swap sono scambi atomici tra fondi off-chain Lightning e fondi on-chain Bitcoin.

Un operatore di nodo può avviare un submarine swap e inviare tutti i saldi di canale disponibili all'altra parte, che in cambio invierà bitcoin on-chain.

Oggi questo servizio viene anche offerto a pagamento dai nodi di Lightning Network, che

condividono i propri tassi di cambio e addebitano una commissione per la conversione.

Il vantaggio di un submarine swap è che non è necessario chiudere alcun canale. Ciò significa che preserviamo i nostri canali, ribilanciandoli attraverso questa operazione. Quando effettuiamo un pagamento Lightning, spostiamo parte del saldo da locale a remoto su uno o più dei nostri canali. Ciò non solo riduce il saldo esposto nell'hot wallet del nostro nodo, ma aumenta anche il saldo disponibile per futuri pagamenti in entrata.

Potresti farlo affidandoti a un intermediario che funga da gateway, ma questo presuppone fiducia. Tuttavia, nel caso di un submarine swap, l'operazione non richiede fiducia. I submarine swap sono operazioni atomiche non detentive. Ciò significa che la controparte nel tuo submarine swap non può rubare i tuoi fondi perché il pagamento on-chain dipende dal completamento del pagamento off-chain e viceversa.

Submarine swap con Loop

Un esempio di servizio di submarine swap è *Loop* di Lightning Labs, la stessa società che costruisce LND. Loop è disponibile in due varianti: Loop In e Loop Out. *Loop In* accetta una transazione on-chain Bitcoin e la converte in un pagamento off-chain Lightning. *Loop Out* converte un pagamento Lightning in una transazione Bitcoin.

NOTA	Per utilizzare il servizio Loop, devi eseguire un nodo Lightning LND.

Allo scopo di ridurre il saldo del tuo hot wallet Lightning, utilizzeresti il servizio Loop Out. Per utilizzare Loop, devi installare alcuni software aggiuntivi sul tuo nodo. Loop viene eseguito insieme al tuo nodo LND e fornisce alcuni strumenti da riga di comando (e grafici) per effettuare submarine swap. Puoi trovare il software e le istruzioni di installazione su GitHub.

Dopo aver installato e avviato il software, un'operazione Loop Out è eseguibile con:

```
loop out --amt 501000 --conf_target 400
```

```
Max swap fees for 501000 sat Loop Out: 25716 sat
```

```
Regular swap speed requested, it might take up to 30m0s for the swap to be
executed.
```

```
CONTINUE SWAP? (y/n), expand fee detail (x): x
```

```
Estimated on-chain sweep fee:        149 sat
```

```
Max on-chain sweep fee:              14900 sat
```

```
Max off-chain swap routing fee:      10030 sat
Max no show penalty (prepay):        1337 sat
Max off-chain prepay routing fee:    36 sat
Max swap fee:                        750 sat
CONTINUE SWAP? (y/n): y
Swap initiated
Run `loop monitor` to monitor progress.
```

Tieni presente che la commissione massima, che rappresenta lo scenario peggiore, dipenderà dall'obiettivo di conferma selezionato.

Uptime e Disponibilità di Nodi Lightning

A differenza di Bitcoin, i nodi Lightning devono essere online quasi continuamente. Il tuo nodo deve essere online per ricevere pagamenti, aprire canali, chiudere canali (cooperativamente) e monitorare le violazioni del protocollo. La disponibilità/uptime dei nodi è un requisito estremamente importante su LN, tanto che è una metrica utilizzata da vari strumenti di gestione automatica dei canali (es. `autopilot`) per decidere con quali nodi aprire i canali. Puoi vedere l'uptime come metrica dei nodi su popolari explorer come 1ML.

La disponibilità dei nodi è particolarmente importante per mitigare e risolvere potenziali violazioni del protocollo (ad esempio impegni revocati). Mentre puoi permetterti brevi interruzioni da un'ora fino a uno o due giorni, non puoi avere il tuo nodo offline per periodi di tempo più lunghi senza rischiare la perdita di fondi.

Mantenere un nodo online non è facile, poiché bug e limitazioni delle risorse potrebbero occasionalmente causare tempi di inattività. Se gestisci un nodo grosso, ti imbatterai in limiti di memoria, spazio swap, numero di file aperti, spazio su disco e così via. Tutta una serie di problemi diversi causerà il crash del tuo nodo o del tuo server.

Tollera Errori e Automatizza

Se hai il tempo e le competenze, dovresti testare alcuni scenari di errore di base di LN. Sulla testnet avrai la possibilità di testare e imparare senza rischiare fondi. Qualsiasi passaggio eseguito per automatizzare il sistema migliorerà la tua disponibilità:

Riavvio automatico del server/computer

Cosa succede quando il tuo server o il sistema operativo va in crash? Cosa succede quando c'è un'interruzione di corrente? Simula questo guasto premendo il pulsante "spegni" sul tuo PC o scollegando il cavo di alimentazione. Dopo un arresto anomalo, un ripristino o un'interruzione di corrente, il computer dovrebbe riavviarsi automaticamente. Alcuni computer hanno un'impostazione nel loro BIOS per specificare come il computer dovrebbe reagire in caso di interruzioni di corrente. Provalo per assicurarti che il computer si riavvii davvero automaticamente in caso di interruzione di corrente senza intervento umano.

Riavvio automatico del nodo

Cosa succede quando il tuo nodo o uno dei tuoi nodi va in crash? Simula questo errore chiudendo i processi del nodo corrispondente. Se un nodo crasha, dovrebbe riavviarsi automaticamente. Assicurati che il nodo si riavvii davvero automaticamente in caso di errore senza intervento umano. In caso contrario, molto probabilmente il tuo nodo non è impostato correttamente come servizio del sistema operativo.

Riconnessione automatica alla rete

Cosa succede se la tua rete si interrompe? Cosa succede quando il tuo ISP si interrompe? Cosa succede quando il tuo ISP assegna un nuovo indirizzo IP al tuo router o al tuo computer? Quando la rete torna, i nodi in esecuzione si riconnettono automaticamente alla rete? Simula questo errore scollegando e successivamente ricollegando il cavo Ethernet dal dispositivo che ospita i tuoi nodi. I nodi dovrebbero riconnettersi automaticamente e continuare a funzionare senza intervento umano.

Configura i tuoi file di log

Tutti gli errori precedenti dovrebbero lasciare voci testuali nei file log corrispondenti. Aumenta la verbosità della registrazione se necessario. Trova queste voci di errore nei file di registro e usale per il monitoraggio.

Monitoraggio della Disponibilità del Nodo

Il monitoraggio del tuo nodo è una parte importante per mantenerlo in esecuzione. È necessario monitorare non solo la disponibilità del computer stesso, ma anche la disponibilità e il corretto funzionamento del software del nodo Lightning.

Esistono diversi modi per farlo, ma la maggior parte richiede una certa personalizzazione. È possibile utilizzare strumenti generici di monitoraggio dell'infrastruttura o di monitoraggio delle applicazioni, ma è necessario personalizzarli in modo specifico per interrogare l'API del nodo Lightning per garantire che il nodo sia in esecuzione, sincronizzato con la blockchain e connesso ai partner di canale.

Lightning.watch fornisce un servizio specializzato che offre il monitoraggio del nodo Lightning. Utilizza un bot di Telegram per avvisarti di eventuali interruzioni del servizio. Questo è un servizio gratuito, anche se puoi pagare per ricevere avvisi più veloci.

Col tempo, prevediamo che più servizi di terze parti forniscano un monitoraggio specializzato dei nodi Lightning usufruibile tramite micropagamenti. Forse tali servizi e le loro API diventeranno col tempo degli standard e un giorno saranno supportati direttamente dal software del tuo nodo Lightning.

Watchtower

Le *watchtower* (torri di controllo) sono un meccanismo per esternalizzare il monitoraggio e la risoluzione delle sanzioni in caso di violazioni del protocollo Lightning.

Come accennato nei capitoli precedenti, il protocollo Lightning mantiene la sicurezza attraverso un meccanismo di penalità. Se uno dei tuoi partner di canale trasmette una vecchia transazione di impegno, il tuo nodo dovrà esercitare la clausola di revoca e trasmettere una transazione di penalità per evitare di perdere denaro. Se il tuo nodo è inattivo durante la violazione del protocollo, potresti perdere soldi.

Per risolvere questo problema, possiamo utilizzare una o più watchtower per esternalizzare il compito di monitorare le violazioni del protocollo ed emettere transazioni di penalità. Ci sono due parti nella configurazione di una watchtower: un server watchtower (o semplicemente watchtower) che monitora la blockchain e un client watchtower che chiede al server watchtower questo servizio di monitoraggio.

Il software LND include sia un server watchtower che un client watchtower. Puoi attivare il

server watchtower aggiungendo le seguenti opzioni di configurazione:

```
[watchtower]
watchtower.active=1
watchtower.towerdir=/path_to_watchtower_data_directory
```

Puoi utilizzare il client watchtower di LND attivandolo nella configurazione e quindi utilizzando la riga di comando per connetterlo a un server watchtower. La configurazione è:

```
[wtclient]
wtclient.active=1
```

Il client della riga di comando `lncli` di LND mostra le seguenti opzioni per la gestione del client watchtower:

```
$ lncli wtclient
NAME:
   lncli wtclient - Interact with the watchtower client.
USAGE:
   lncli wtclient command [command options] [arguments...]
COMMANDS:
     add     Register a watchtower to use for future sessions/backups.
     remove  Remove a watchtower to prevent its use for future
sessions/backups.
     towers  Display information about all registered watchtowers.
     tower   Display information about a specific registered watchtower.
     stats   Display the session stats of the watchtower client.
     policy  Display the active watchtower client policy configuration.
OPTIONS:
   --help, -h  show help
```

Infine, una watchtower famosa è *The Eye of Satoshi* (TEOS), che puoi trovare su GitHub.

Gestione dei Canali

In qualità di operatore di un nodo Lightning, una delle attività ricorrenti che dovrai eseguire è

la gestione dei tuoi canali. Ciò significa aprire canali in uscita verso ad altri nodi, oltre a far sì che altri nodi aprano canali in entrata col tuo. È già possibile costruire canali cooperativi, ovvero canali simmetrici che hanno fondi impegnati su entrambe le estremità al momento della creazione. Tuttavia, la maggior parte dei nuovi canali hanno fondi solo da un lato, ovvero da quello di chi li crea. Per rendere il tuo nodo bilanciato con capacità sia in entrata che in uscita, devi aprire canali con altri e invogliare altri ad aprire canali verso il tuo nodo.

Apertura di Canali in Uscita (Outbound Channels)

Non appena il tuo nodo Lightning è attivo e funzionante, puoi finanziarne il portafoglio Bitcoin on-chain relativo e quindi iniziare ad aprire canali con tali fondi.

Devi scegliere attentamente i partner di canale perché la capacità del tuo nodo di inviare pagamenti dipende da chi sono i tuoi partner di canale e da quanto sono ben collegati al resto del Lightning Network. Dovresti volere anche più di un canale per evitare di essere suscettibile ad un singolo punto di fallimento. Poiché Lightning supporta i pagamenti in più parti, puoi dividere i tuoi fondi iniziali in più canali e instradare pagamenti più grandi combinando la loro capacità. Allo stesso tempo, evita di rendere i tuoi canali troppo piccoli. Poiché è necessario pagare commissioni per aprire e chiudere un canale on-chain, il saldo del canale non dovrebbe essere troppo piccolo. Si tratta di equilibrio!

Per riassumere:

- Connettiti ad alcuni nodi ben collegati
- Apri più di un canale
- Non aprire troppi canali
- Non aprire canali troppo piccoli

Un modo per trovare nodi ben collegati è aprire un canale a un commerciante popolare che vende prodotti su Lightning Network. Questi nodi tendono ad essere ben finanziati e ben collegati. Quando sei pronto per acquistare qualcosa online tramite Lightning, puoi aprire un canale direttamente al nodo del commerciante. L'ID del nodo del commerciante sarà nella fattura che riceverai quando proverai ad acquistare qualcosa.

Un altro modo per trovare nodi ben connessi consiste nell'utilizzare un Lightning Explorer come 1ML e sfoglia l'elenco dei nodi ordinati per capacità del canale e numero di canali. Non

scegliere i nodi più grandi, perché ciò incoraggia la centralizzazione. Scegli un nodo nel mezzo dell'elenco in modo da poterli aiutare a crescere. Un altro fattore da considerare potrebbe essere il periodo di tempo in cui un nodo è stato operativo. È probabile che i nodi stabiliti per più di un anno siano più affidabili e meno rischiosi dei nodi che hanno iniziato a funzionare una settimana fa.

Autopilot

L'attività di apertura dei canali può essere parzialmente automatizzata con l'uso di un *autopilot*, ovvero un software che apre automaticamente i canali in base ad alcune euristiche. Il software autopilot è ancora relativamente nuovo e non sempre seleziona i migliori partner di canale. Soprattutto all'inizio, potrebbe essere meglio aprire i canali manualmente. Gli autopilot esistono attualmente in tre forme:

- `lnd` incorpora un autopilot completamente integrato e funziona in background.
- `lib_autopilot.py` offre autopilot per qualsiasi implementazione del nodo in base al gossip e ai dati del canale.
- Un plug-in `c-lightning` basato su `lib_autopilot.py` che fornisce un'interfaccia facile da usare per gli utenti `c-lightning`.

Tieni presente che l'autopilot di `lnd` inizierà a funzionare in background non appena viene attivato tramite il file di configurazione. Di conseguenza, inizierà immediatamente ad aprire i canali se hai fondi on-chain nel tuo portafoglio. Se vuoi avere il pieno controllo sulle transazioni bitcoin che effettui e sui canali che apri, assicurati di disattivare autopilot *prima* di caricare il tuo portafoglio `lnd` con fondi bitcoin. Se l'autopilot è stato precedentemente attivato, potresti dover riavviare `lnd` prima di ricaricare il tuo portafoglio con una transazione on-chain o prima di chiudere i canali, il che ti restituisce di fatto i fondi on-chain. È fondamentale configurare alcune variabili di autopilot manualmente se vuoi che venga eseguito in automatico. Dai un'occhiata a questa configurazione di esempio:

```
[lnd-autopilot]
autopilot.active=1
autopilot.maxchannels=40
autopilot.allocation=0.70
autopilot.minchansize=500000
```

```
autopilot.maxchansize=5000000
autopilot.heuristic=top_centrality:1.0
```

Questo file di configurazione attiva autopilot. Aprirà canali purché siano soddisfatte le seguenti due condizioni:

1. Il tuo nodo ha attualmente meno di 40 canali aperti.
2. Meno del 70% dei tuoi fondi totali è off-chain in canali di pagamento (già aperti).

I numeri 40 e 0.7 sono scelti in modo del tutto arbitrario perché non possiamo fornire una raccomandazione che sia valida per tutti. L'autopilot in `lnd` non terrà conto delle commissioni on-chain. In altre parole, non ritarderà l'apertura dei canali a un periodo di tempo in cui la mempool è vuota e le commissione sono basse. Per ridurre le commissioni, puoi aprire manualmente i canali durante un periodo di tempo in cui le stesse sono basse, ad esempio durante il fine settimana. L'autopilot consiglierà canali ogni volta che le condizioni sono soddisfatte e tenterà immediatamente di aprirne uno utilizzando le fee correnti necessarie. Secondo il file di configurazione precedente, i canali avranno una dimensione compresa tra 5 mBTC (`minchansize` = 500.000 satoshi) e 50 mBTC (`maxchansize` = 5.000.000 satoshi). Come è comune, gli importi nel file di configurazione sono enumerati in satoshi. Attualmente, canali inferiori a 1 mBTC non sono molto utili e ti sconsigliamo di aprire canali troppo piccoli e inferiori a tale importo. Con l'adozione di pagamenti multipath, i canali più piccoli saranno comunque utili; ma per il momento, questa è la nostra raccomandazione.

Il plug-in `c-lightning`, originariamente scritto da René Pickhardt (coautore di questo libro), funziona in modo molto diverso rispetto all'autopilot `lnd`. Innanzitutto, differisce negli algoritmi utilizzati per formulare le raccomandazioni (argomento che non tratteremo) e, in secondo luogo, differisce nella sua interfaccia utente. Sarà necessario scaricare il *plug-in autopilot* dal repository dei plug-in di c-lightning e attivarlo.

NOTA	Per attivare un plug-in in `c-lightning`, posizionalo nella directory *~/.lightning/plugins*, assicurati che sia eseguibile (es. `chmod +x ~/.lightning/plugins/autopilot.py`) e riavvia `lightningd`. In alternativa, se non vuoi che un plug-in si attivi automaticamente quando avvii `lightningd`, puoi posizionarlo in una directory diversa e attivarlo manualmente con la variabile `--plugin`: `lightningd --plugin=~/lightning-plugins/autopilot.py`

L'autopilot in `c-lightning` è controllato tramite tre valori che possono essere impostati nel file di configurazione o come argomenti della riga di comando all'avvio di `lightningd`:

```
[c-lightning-autopilot]
autopilot-percent=75
autopilot-num-channels=10
autopilot-min-channel-size-msat=100000000msat
```

Questi valori sono predefiniti e non è necessario impostarli manualmente.

L'autopilot non funzionerà automaticamente in background come in `lnd`. Dovrai infatti avviarlo tramite il comando `lightning-cli autopilot-run-once`, se vuoi che l'autopilot apra i canali consigliati. Se invece vuoi che fornisca solo consigli, per poi valutare se selezionare manualmente dei nodi, puoi aggiungere l'argomento opzionale `dryrun`.

Una differenza fondamentale tra l'autopilot `lnd` e `c-lightning` è che l'autopilot `c-lightning` fornirà anche una raccomandazione per la dimensione del canale. Ad esempio, se l'autopilot consiglia di aprire un canale con un nodo piccolo che ha solo canali piccoli, non consiglierà di aprire un canale grande. Tuttavia, se apre un canale con un nodo ben connesso che ha molti canali di grandi dimensioni, consiglierà una dimensione del canale maggiore.

Come puoi vedere, l'autopilot di `c-lightning` non è automatico come quello di `lnd`, ma ti dà maggiore controllo. Queste differenze riflettono le preferenze personali e potrebbero effettivamente essere il fattore decisivo per scegliere un'implementazione rispetto all'altra.

Tieni presente che gli attuali autopilot utilizzano principalmente le informazioni pubbliche del protocollo di gossip sull'attuale topologia di LN. È ovvio che le tue esigenze personali possono riflettersi solo in una certa misura. Gli autopilot più avanzati userebbero le informazioni storiche e l'utilizzo del tuo nodo durante un'esecuzione nel tempo, incluse le informazioni sui successi di instradamento, chi hai pagato in passato e chi ti ha pagato. Questi autopilot avanzati possono anche utilizzare tali dati raccolti per formulare raccomandazioni sulla chiusura di canali e sulla corretta riallocazione dei fondi.

In generale, fai attenzione a non dipendere o fare troppo affidamento sugli autopilot.

Ottenere Liquidità in Entrata (Inbound Liquidity/Channel)

Nell'attuale progettazione di LN, per gli utenti è più semplice ottenere liquidità in uscita

piuttosto che liquidità in entrata. Tale liquidità si ottiene aprendo un canale con un altro nodo e, solitamente, gli utenti potranno solamente inviare bitcoin prima di poterne ricevere tramite i propri canali. Esistono tre modi per ottenere liquidità in entrata:

- Apri un canale con liquidità in uscita e poi spendi parte di quei fondi. A questo punto il saldo è dall'altra parte del canale, il che significa che puoi ricevere pagamenti.
- Chiedi a qualcuno di aprire un canale verso il tuo nodo. Offriti di ricambiare, in modo che entrambi i tuoi nodi diventino meglio connessi ed equilibrati.
- Usa un submarine swap (es. Loop In) per scambiare BTC on-chain con un canale in entrata verso il tuo nodo.
- Paga un servizio di terze parti per aprire un canale con te. Esistono molti di questi servizi. Alcuni applicano una commissione per fornire liquidità, altri sono gratuiti.

Ecco un elenco di fornitori di liquidità che apriranno un canale al tuo nodo a pagamento:

- Il servizio Thor di Bitrefill
- Lightning To Me
- LNBig
- Lightning Conductor

Ottenere liquidità in entrata è impegnativo sia dal punto di vista pratico che dall'esperienza dell'utente. La liquidità in entrata non è un processo magico e automatico, anzi, devi trovare il modo di ottenerla per il tuo nodo. Questa asimmetria dei canali di pagamento non è per nulla intuitiva. Nella maggior parte degli altri sistemi di pagamento, vieni pagato per primo (in entrata) prima di pagare gli altri (in uscita). Su LN è il contrario.

La sfida di creare liquidità in entrata è più evidente se sei un commerciante o vendi i tuoi servizi per pagamenti Lightning. In tal caso, devi essere vigile ed assicurarti di avere liquidità in entrata sufficiente per poter continuare a ricevere pagamenti. Cosa succede se c'è un'ondata di acquirenti nel tuo negozio, ma non possono effettivamente pagarti perché non hai più capacità in entrata?

Queste sfide possono essere parzialmente mitigate tramite la creazione di canali a doppio finanziamento (dual funded channels), che sono finanziati da entrambe le parti e offrono una capacità bilanciata in entrata (inbound) e in uscita (outbound). L'onere potrebbe anche essere mitigato da un autopilot più sofisticato, che potrebbe richiedere e pagare la capacità

in entrata secondo necessità.

In definitiva, gli utenti di LN devono essere strategici e proattivi nella gestione dei canali per garantire che sia disponibile capacità in entrata sufficiente per soddisfare le loro esigenze.

Chiusura dei Canali

Come discusso in precedenza, una *chiusura reciproca* è il modo preferito per chiudere un canale tra partner. Tuttavia, ci sono casi in cui è necessaria una *chiusura forzata*.

Qualche esempio:

- Il partner è offline e non può essere contattato per avviare una chiusura reciproca.
- Il partner è online, ma non risponde alle richieste di avviare una chiusura reciproca.
- Il partner è online e i tuoi nodi stanno negoziando una chiusura reciproca, ma si bloccano e non riescono a raggiungere una risoluzione.

Ribilanciare i Canali

Mentre effettuerai pagamenti personali ed instraderai pagamenti di altri su Lightning, l'equilibrio tra capacità in entrata e in uscita può diventare sbilanciato.

Ad esempio, se uno dei tuoi partner di canale instrada frequentemente i pagamenti attraverso il tuo nodo, esaurirai la capacità in entrata su quel canale ed esaurirai anche la capacità in uscita (outbound) sui canali in uscita (outgoing). Una volta che ciò accade, non puoi più instradare i pagamenti attraverso quel percorso.

Esistono molti modi per ribilanciare i canali, ognuno con diversi vantaggi e svantaggi. Un modo è sfruttando un submarine swap (es. Loop Out) come descritto in precedenza in questo capitolo. Un altro modo per riequilibrare è semplicemente attendere dei pagamenti instradati che viaggino nella direzione opposta. Se il tuo nodo è ben connesso, quando un percorso specifico si esaurisce in una direzione, lo stesso percorso diventa disponibile nella direzione opposta. Altri nodi potrebbero "scoprire" quel percorso nella direzione opposta ed usarlo come parte del loro percorso di pagamento, riequilibrando così nuovamente i fondi.

Un terzo modo per ribilanciare i canali è creare intenzionalmente un *percorso circolare* (*circular route*) che invii un pagamento dal tuo nodo al tuo nodo. Inviando un pagamento su

un canale con grande capacità locale e organizzando il percorso in modo che ritorni al tuo nodo su un canale con grande capacità remota, entrambi i canali diventeranno più bilanciati. Un esempio di strategia di ribilanciamento del percorso circolare è nella Figura 5-4.

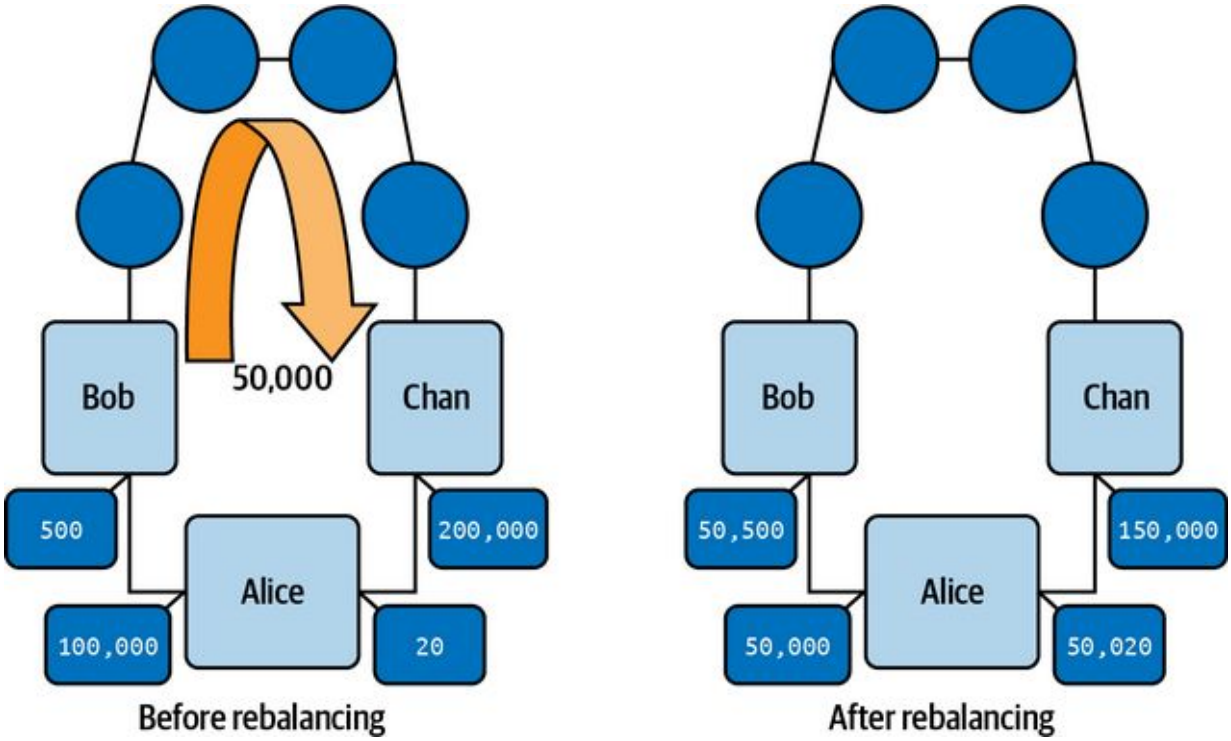

Figura 5-4. Ribilanciamento tramite percorso circolare

Il ribilanciamento circolare è supportato dalla maggior parte delle implementazioni di nodi Lightning e può essere eseguito sulla riga di comando o tramite una delle interfacce grafiche. Il ribilanciamento dei canali è una questione complessa oggetto di ricerca attiva.

Commissioni di Instradamento (Routing Fees)

L'esecuzione di un nodo Lightning ti consente di guadagnare commissioni instradando i pagamenti attraverso i tuoi canali. Le commissione di instradamento non sono generalmente una fonte di reddito significativa e sono sminuite dal costo di gestione di un nodo. Ad esempio, su un nodo relativamente grosso che instrada una dozzina di pagamenti al giorno, le commissioni ammontano a non più di 2000 satoshi.

I nodi competono per le commissioni di instradamento impostando la tariffa desiderata su ciascun canale. Le commissioni di instradamento sono stabilite da due parametri su ciascun canale: una *tariffa base fissa* (*fixed base fee*) che viene addebitata per qualsiasi pagamento e una *tariffa variabile proporzionale* all'importo del pagamento (*variable fee rate*).

Quando si invia un pagamento Lightning, un nodo selezionerà un percorso in modo da ridurre al minimo le commissioni, ridurre al minimo gli hop o entrambe le cose. Di conseguenza, da queste interazioni emerge un mercato delle commissioni di instradamento. Attualmente ci sono molti nodi che applicano tariffe molto basse o pari a zero, creando una pressione al ribasso sul mercato delle commissioni di instradamento.

Se non effettui alcuna scelta, il tuo nodo Lightning imposterà una tariffa base e una tariffa predefinita per ogni nuovo canale. I valori predefiniti dipendono dall'implementazione del nodo utilizzata. La tariffa base è fissata nell'unità di *millisatoshi* (millesimi di satoshi). L'aliquota proporzionale è stabilita nell'unità di *milionesimi* e viene applicata all'importo del pagamento. L'unità di milionesimi è spesso abbreviata con *ppm* (parti per milione). Ad esempio, una tariffa base di 1000 (millisatoshi) e una tariffa di 1000 ppm (milionesimi) comporterebbe i seguenti addebiti per un pagamento di 100000 satoshi:

P = 100000 satoshi

F_{base} = 1000 millisatoshi = 1 satoshi

F_{rate} = 1000 ppm = 1000/1000000 = 1/1000 = 0.001 = 0. 1%

$F_{total} = F_{base} + (P * F_{rate})$

$\Rightarrow F_{total}$ = 1 satoshi + (100000/1000) satoshi

$\Rightarrow F_{total}$ = 1 satoshi + 100 satoshi = 101 satoshi

In generale, puoi adottare uno dei due approcci per le tue commissioni di instradamento. Puoi instradare molti pagamenti con commissioni basse, compensando le commissioni basse con un volume elevato. In alternativa, puoi scegliere di addebitare commissioni più elevate. Se scegli di impostare commissioni più elevate, il tuo nodo verrà selezionato solo quando non esistono altri percorsi più economici. Pertanto, eseguirai l'instradamento meno frequentemente ma guadagnerai di più per ogni instradamento riuscito.

Per la maggior parte dei nodi, di solito è meglio utilizzare i valori delle tariffe di instradamento predefinite. In questo modo, il tuo nodo compete in condizioni di parità con altri nodi che utilizzano i valori predefiniti.

Puoi anche utilizzare le commissioni di instradamento per ribilanciare i tuoi canali. Se la maggior parte dei tuoi canali ha commissioni predefinite ma desideri ribilanciare un particolare canale, ti basterà ridurre le commissioni su quel canale specifico (es. impostandole a zero o valori molti bassi). Ti basterà aspettare che qualcuno instradi un pagamento sul tuo percorso "economico" e riequilibri i tuoi canali come effetto collaterale.

Gestione dei Nodi

Gestire il tuo nodo tramite terminale non è facile. Ti offre la piena flessibilità del nodo e la possibilità di scrivere i tuoi script personalizzati per soddisfare le tue esigenze personali; ma se non vuoi affrontare la complessità della riga di comando e hai solo bisogno di alcune funzionalità di gestione del nodo di base, dovresti prendere in considerazione l'installazione di un'interfaccia utente che semplifichi la gestione dei nodi. Vediamo qualche esempio...

Ride The Lightning

Ride The Lightning (RTL) è un'interfaccia web per aiutare gli utenti a gestire i propri nodi Lightning. È disponibile per le tre principali implementazioni (LND, `c-lightning` ed Eclair). RTL è un progetto open source sviluppato da Shahana Farooqui e molti altri contributori. Puoi scaricare RTL su GitHub. La Figura 5-5 mostra uno screenshot dell'interfaccia di RTL.

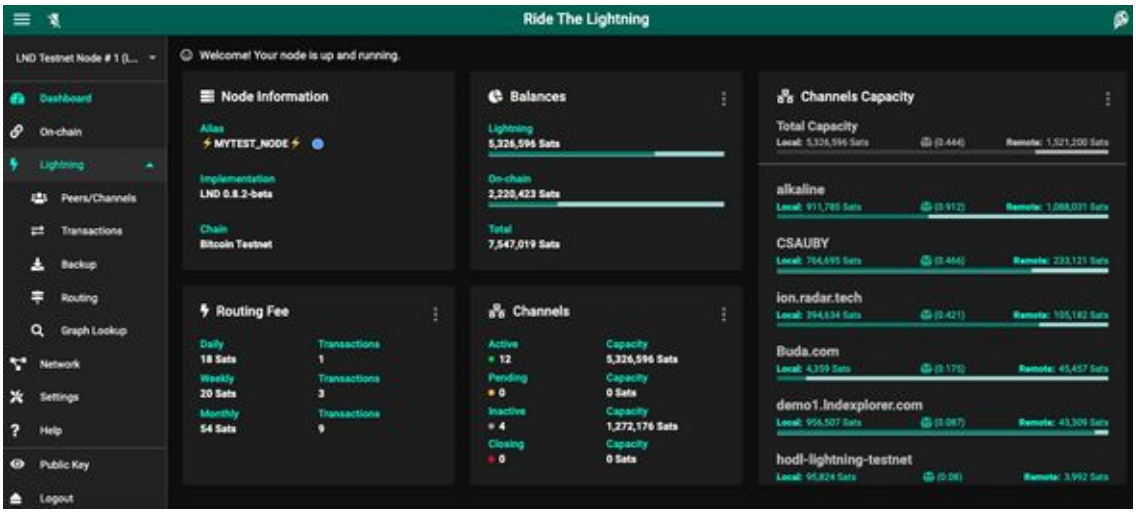

Figura 5-5. Interfaccia di Ride The Lightning

lndmon

Lightning Labs, i creatori di LND, forniscono un'interfaccia grafica chiamata `lndmon` per monitorare le varie metriche di un nodo LND. È un'interfaccia di sola lettura e non consente di gestire attivamente il nodo. Puoi trovare e scaricare `lndmon` su GitHub.

ThunderHub

ThunderHub è un'interfaccia grafica per LND simile a RTL. Può essere utilizzata per effettuare pagamenti, ribilanciare i canali e gestire il nodo attraverso una varietà di funzionalità.

Conclusione

Man mano che mantieni il tuo nodo e acquisisci esperienza, imparerai molto su Lightning Network. Gestire un nodo è un compito impegnativo ma gratificante. Padroneggiare queste abilità ti consentirà di contribuire alla crescita e allo sviluppo di questa tecnologia e di LN. Potrai inoltre acquisire la possibilità di inviare e ricevere pagamenti Lightning con il massimo grado di controllo e facilità. Giocherai un ruolo centrale nell'infrastruttura di Lightning Network e non sarai solo un partecipante ai margini di esso.

Parte 2: Lightning Network in Dettaglio

Una spiegazione dettagliata di tutti i componenti di Lightning Network e di come funzionano. Questa parte è altamente tecnica e richiede che il lettore abbia una certa esperienza di programmazione e informatica.

Capitolo 6. Architettura di Lightning Network

Nella prima parte di questo libro abbiamo introdotto i concetti principali di Lightning Network attraverso un esempio di instradamento di un pagamento e su come configurare determinati software e strumenti. Nella seconda parte del libro esploreremo Lightning Network in modo molto più dettagliato, analizzando ciascuno degli elementi costitutivi. In questa sezione delineeremo i componenti di Lightning Network e forniremo una prospettiva "generale" per guidarti attraverso i capitoli successivi.

La Suite di Protocolli di Lightning Network

LN è composto da una complessa raccolta di protocolli che vengono eseguiti su Internet. Possiamo classificare ampiamente questi protocolli in cinque livelli distinti che costituiscono uno stack, in cui ogni livello si basa e utilizza i protocolli nel livello sottostante. Inoltre, ogni livello di protocollo astrae i livelli sottostanti e "nasconde" parte della complessità.

Il diagramma in Figura 6-1 fornisce una panoramica di questi livelli e dei loro componenti.

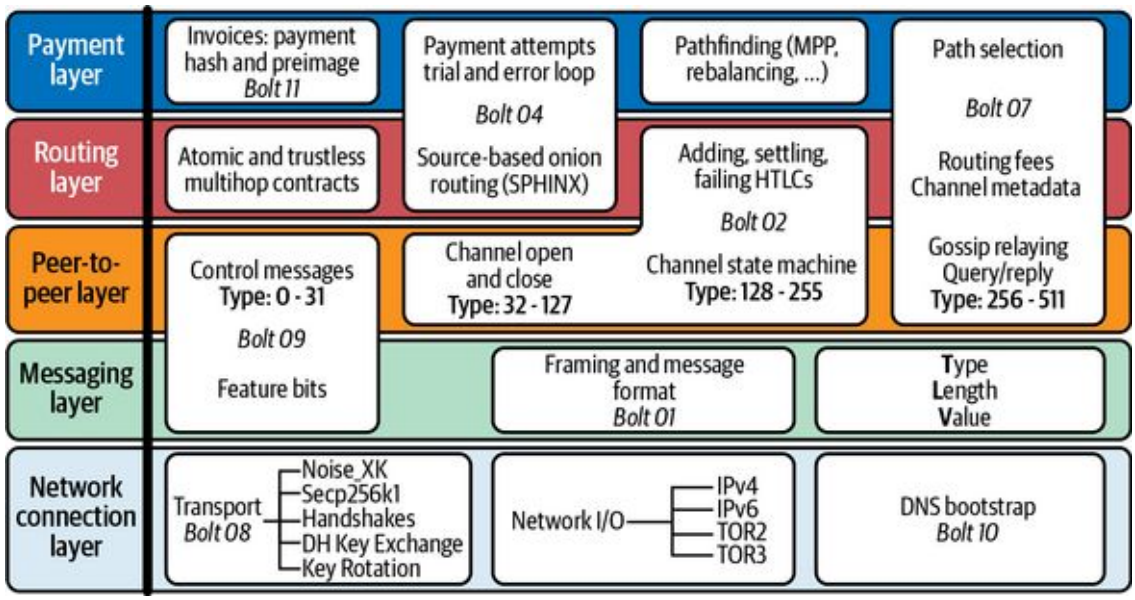

Figura 6-1. La Suite di Protocolli di Lightning Network

I cinque livelli di Lightning Network, dal basso verso l'alto, sono:

Network connection layer

> Contiene i protocolli che interagiscono direttamente con i protocolli principali di Internet (TCP/IP), i protocolli di sovrapposizione (Tor v2/v3) e i servizi Internet (DNS). Contiene anche i protocolli di trasporto crittografico che proteggono i messaggi su LN.

Messaging layer

> Questo livello contiene i protocolli utilizzati dai nodi per negoziare caratteristiche, formattare i messaggi e codificare i campi dei messaggi.

Peer-to-Peer (P2P) layer

> Questo è il livello di protocollo principale per la comunicazione tra i nodi Lightning e contiene tutti i diversi messaggi scambiati tra i nodi.

Routing layer

> Questo livello contiene i protocolli utilizzati per instradare i pagamenti tra i nodi, end-to-end e in modo atomico. Questo livello contiene la funzionalità principale di Lightning Network: i pagamenti instradati.

Payment layer

> Il livello più alto della rete, che presenta un'interfaccia di pagamento affidabile per le applicazioni.

Lightning Network in Dettaglio

Nei prossimi capitoli analizzeremo la suite di protocolli ed in dettaglio Lightning Network.

Abbiamo passato un bel po' di tempo a cercare di decidere l'ordine migliore per presentare tutto ciò. Non è stata una scelta facile perché c'è così tanta interdipendenza tra i diversi

componenti... Quando inizi a spiegarne uno, scopri che ne coinvolge parecchi altri. Invece di un approccio dall'alto verso il basso o dal basso verso l'alto, abbiamo finito per scegliere un percorso più tortuoso che inizia con gli elementi costitutivi fondamentali che sono unici circa i canali di pagamento di Lightning Network e da lì pian piano spostarci verso l'esterno. Purtroppo, siccome questo percorso non è ovvio e facilmente comprensibile, useremo la suite di protocolli Lightning mostrata nella Figura 6-1 come mappa logica. Ogni capitolo si concentrerà su uno o più componenti correlati e li vedrai evidenziati nella suite di protocolli. Immaginalo come un navigatore in tempo reale che ti dice "Ti trovi qui!"

Capitolo 7. "Canali di Pagamento"

In questo capitolo esamineremo come funzionano i canali di pagamento, in modo molto più approfondito rispetto a quanto visto nelle parti precedenti del libro. Esamineremo la struttura e il Bitcoin Script delle transazioni di finanziamento ed impegno, oltre ad il processo utilizzato dai nodi per negoziare ogni fase del protocollo.

Capitolo 8. "Routing su una Rete di Canali di Pagamento"

Successivamente, metteremo insieme diversi canali di pagamento in una rete e instraderemo un pagamento da un'estremità all'altra. Ci immergeremo negli hash time-locked contract (HTLC) e nel linguaggio Bitcoin Script che usiamo per costruirli.

Capitolo 9. "Funzionamento del Canale e Inoltro dei Pagamenti"

Mettendo insieme i concetti di un semplice canale di pagamento e di un pagamento instradato utilizzando HTLC, esamineremo ora come gli HTLC fanno parte della transazione di impegno di ciascun canale. Esamineremo anche il protocollo per l'aggiunta, la risoluzione, il fallimento e la rimozione di HTLC dai commitments.

Capitolo 10. "Onion Routing"

Successivamente, esamineremo come le informazioni HTLC vengono propagate attraverso la rete all'interno del protocollo di onion routing. Esamineremo il meccanismo di crittografia e decrittografia a più livelli che conferisce a Lightning Network alcune delle sue caratteristiche di privacy.

Capitolo 11. "Gossip e Grafo dei Canali"

In questo capitolo esamineremo come i nodi Lightning si conoscono l'un l'altro e studieremo come vengono sfruttati i canali annunciati per costruire un grafo dei canali che (i nodi) possono utilizzare per trovare percorsi attraverso la rete.

Capitolo 12. "Pathfinding e Consegna dei Pagamenti"

Successivamente, vedremo come le informazioni del protocollo di gossip vengono utilizzate da ciascun nodo per costruire una "mappa" dell'intera rete, che può essere utilizzata per trovare percorsi da un punto all'altro per instradare pagamenti. Esamineremo anche le innovazioni del pathfinding, come i pagamenti multipart.

Capitolo 13. "Protocollo Wire: Framing ed Estensibilità"

Alla base di Lightning Network c'è il protocollo peer-to-peer che i nodi utilizzano per scambiare messaggi sulla rete e sui loro canali. In questo capitolo esaminiamo come sono costruiti questi messaggi e le capacità di estensione integrate nei messaggi con bit di funzionalità e codifica TLV (Type-Length-Value).

Capitolo 14. "Trasporto di Messaggi Crittografati"

Nel livello inferiore della rete, esamineremo il sistema di trasporto crittografato che garantisce la segretezza e l'integrità di tutte le comunicazioni tra i nodi.

Capitolo 15. "Richieste di Pagamento Lightning"

Una parte fondamentale di Lightning Network sono le richieste di pagamento, note anche come fatture/invoice Lightning. In questo capitolo ne analizzeremo la struttura.

Iniziamo!

Capitolo 7. Canali di Pagamento

In questo capitolo tratteremo i canali di pagamento e vedremo come sono costruiti. Inizieremo con il nodo di Alice che apre un canale verso il nodo di Bob, basandoci sugli esempi presentati all'inizio di questo libro.

I messaggi scambiati dai nodi di Alice e Bob sono definiti in "BOLT #2: Peer Protocol for Channel Management". Le transazioni create dai nodi di Alice e Bob sono definite in "BOLT #3: Bitcoin Transaction and Script Formats". In questo capitolo ci concentriamo sulle parti "Channel open and close" e "Channel state machine" dell'architettura del protocollo Lightning, evidenziate da uno schema al centro (livello peer-to-peer) nella Figura 7-1 .

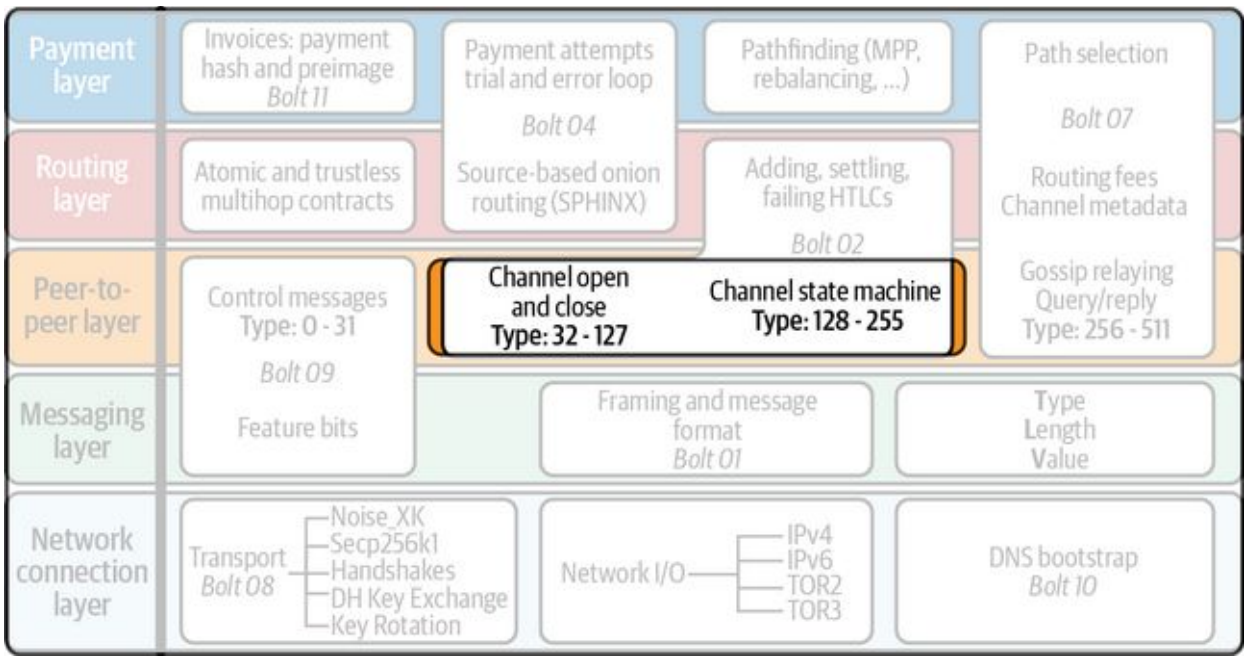

Figura 7-1. Canali di pagamento nella suite di protocolli Lightning

Un Modo Diverso di Utilizzare Bitcoin

Lightning Network è spesso descritto come un "protocollo Bitcoin di livello 2 (o secondo

livello)", il che lo rende distinto da Bitcoin. Un altro modo per descrivere Lightning è come un "modo più intelligente di usare Bitcoin" o semplicemente come "un'applicazione su Bitcoin".

Le transazioni Bitcoin vengono trasmesse a tutti e registrate sulla blockchain per essere considerate valide. Come vedremo, tuttavia, se qualcuno detiene una transazione Bitcoin prefirmata che spende un output multisig 2-di-2 che gli dà la possibilità esclusiva di spendere quei bitcoin, effettivamente possiede quei bitcoin anche se non trasmette la transazione.

Puoi pensare alla transazione Bitcoin prefirmata come un assegno postdatato, che può essere incassato in qualsiasi momento. A differenza del sistema bancario tradizionale, tuttavia, questa transazione non è una "promessa" di pagamento (nota anche come IOU), ma uno strumento al portatore verificabile che equivale al contante. Finché il bitcoin a cui si fa riferimento nella transazione non è già stato speso al momento del riscatto (o nel momento in cui si tenta di "incassare" l'assegno), il protocollo Bitcoin garantisce che questa transazione prefirmata possa essere trasmessa e registrata in qualsiasi momento . Questo è vero solo se questa è l'unica transazione prefirmata. All'interno di LN esistono contemporaneamente due o più di tali transazioni prefirmate; pertanto, abbiamo bisogno di un meccanismo più sofisticato per avere ancora la funzionalità di tale strumento al portatore verificabile.

Lightning Network è semplicemente un modo diverso e creativo di usare Bitcoin. In LN una combinazione di transazioni registrate (on-chain) e prefirmate ma trattenute (off-chain) forma uno "strato" (layer) di pagamenti che è un modo più veloce, più economico e più privato per utilizzare Bitcoin. Puoi vedere la relazione tra transazioni Bitcoin on-chain e off-chain in Figura 7-2.

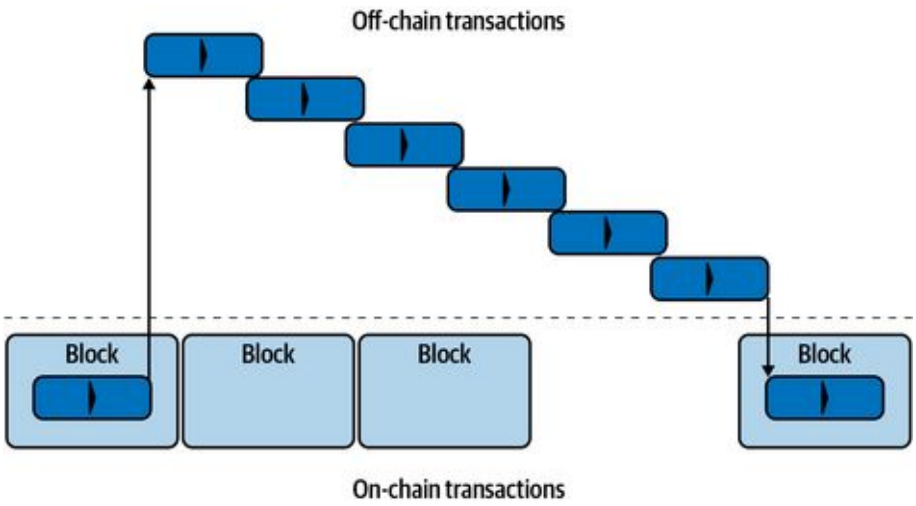

Figura 7-2. Canale di pagamento Lightning fatto di transazioni on-chain e off-chain

Lightning Network è Bitcoin. È un modo diverso di usarlo.

Proprietà e Controllo di Bitcoin

Prima di comprendere i canali di pagamento, dobbiamo fare un piccolo passo indietro e capire come funzionano la proprietà ed il controllo in Bitcoin.

Quando qualcuno dice di "possedere" bitcoin, in genere significa che conosce la chiave privata di un indirizzo Bitcoin che ha alcuni output di transazione non spesi (vedi Appendice A). La chiave privata consente di firmare una transazione per spendere quei bitcoin trasferendoli a un indirizzo diverso. In Bitcoin la "proprietà" può essere definita come la capacità di spendere determinati bitcoin.

Tieni presente che il termine "proprietà" utilizzato in Bitcoin è distinto dal termine "proprietà" utilizzato in senso legale. Un ladro che ha le chiavi private e può spendere bitcoin è *de facto* il proprietario di quei bitcoin anche se non è un legittimo proprietario.

SUGGERIMENTO	La proprietà di Bitcoin riguarda solo il controllo delle chiavi e la possibilità di spendere i bitcoin con quelle chiavi. Come dice il popolare detto Bitcoin: "Le tue chiavi, le tue monete, non le tue chiavi, non le tue monete" ("Your keys, your coins, not your keys, not your coins.")

Diversità di Proprietà (indipendente) e Multisig

La proprietà e il controllo delle chiavi private non sono sempre nelle mani di una sola persona. È qui che le cose si fanno interessanti e complicate. Sappiamo che più di una persona può venire a conoscenza della stessa chiave privata, per furto o perché il detentore originale della chiave ne fa una copia e la consegna a qualcun altro. Tutte queste persone sono proprietarie? In pratica lo sono, perché chiunque conosca la chiave privata può spendere bitcoin senza l'approvazione di nessun altro.

Bitcoin ha anche indirizzi multifirma in cui sono necessarie più chiavi private per firmare prima di spendere. Da un punto di vista pratico, la proprietà in un indirizzo con più firme dipende dal quorum (*K*) e dal totale (*N*) definiti nello schema *K*-di-*N*. Uno schema multifirma 1-di-10 consentirebbe a qualsiasi firmatario 1 (*K*) su 10 (*N*) di spendere un importo in bitcoin bloccato in quell'indirizzo. Questo è simile allo scenario in cui 10 persone hanno una copia

della stessa chiave privata e ognuna di loro può spenderla in modo indipendente.

Proprietà Congiunta Senza Controllo Indipendente

C'è anche lo scenario in cui *nessuno* ha il quorum. In uno schema 2-di-2 come quello utilizzato in LN, nessuno dei due firmatari può spendere bitcoin senza ottenere una firma dall'altra parte. Chi possiede i bitcoin in questo caso? Nessuno ha davvero la proprietà perché nessuno ha il controllo. Ciascuno possiede l'equivalente di una quota di voto nella decisione, ma sono necessari entrambi i voti. Un problema chiave (gioco di parole) in uno schema 2-di-2, sia in Bitcoin che nella legge, è: cosa succede se una delle parti non è disponibile, o se c'è una situazione di stallo del voto e una delle parti si rifiuta di collaborare?

Prevenire il Problema di Bitcoin "bloccati" e Non Spendibili

Se uno dei due firmatari di un multisig 2-di-2 non può o non vuole firmare, i fondi diventano non spendibili. Questo scenario non solo può verificarsi accidentalmente (perdita delle chiavi), ma può essere utilizzato come forma di ricatto da entrambe le parti: "Non firmerò a meno che tu non mi paghi una parte dei fondi".

I canali di pagamento in Lightning si basano su un indirizzo multisig 2-di-2, con i due partner di canale come firmatari nel multisig. Al momento, i canali sono finanziati solo da uno dei due partner di canale: quando scegli di "aprire" un canale, depositi fondi nell'indirizzo multisig 2-di-2 con una transazione. Una volta che la transazione è stata minata e i fondi sono nel multisig, non puoi recuperarli senza la collaborazione del tuo partner di canale, perché hai bisogno (anche) della sua firma per spendere i bitcoin.

Nella sezione successiva, mentre esaminiamo come aprire (creare) un canale Lightning, vedremo come possiamo prevenire la perdita di fondi o qualsiasi scenario di ricatto tra i due partner implementando un protocollo di equità per la costruzione del canale con l'aiuto di transazioni prefirmate. Transazioni che spendono l'output multisig in un modo che offre ad entrambi i partner di canale la possibilità esclusiva di spendere uno degli output che codifica la quantità di bitcoin che essi possiedono nel canale.

Costruzione di un Canale di Pagamento

Precedentemente abbiamo descritto i canali di pagamento come una *relazione finanziaria* tra

due nodi di LN, stabilita finanziando un indirizzo multifirma 2-di-2 dai due partner di canale.

Supponiamo che Alice voglia creare un canale di pagamento che le consenta di connettersi al negozio di Bob. Innanzitutto, i due nodi (quello di Alice e quello di Bob) devono stabilire una connessione Internet tra loro, in modo da poter negoziare un canale di pagamento.

Chiavi Private e Pubbliche del Nodo

Ogni nodo su Lightning Network è identificato da una *chiave pubblica del nodo* (*node public key).* La chiave pubblica identifica in modo univoco il nodo e viene solitamente presentata come codifica esadecimale. Ad esempio, Riccardo Masutti attualmente esegue un nodo Lightning identificato dalla seguente chiave pubblica del nodo:

```
02b8c29e92f4842d06c5f6a5e8c53be29f5a6cf4bfd3c4d57d12aae3b0f2d9e10c
```

Ogni nodo genera una chiave privata root quando viene inizializzato per la prima volta. La chiave privata viene sempre mantenuta privata (mai condivisa) e archiviata in modo sicuro nel portafoglio del nodo. Da quella chiave privata, il nodo deriva una chiave pubblica che è l'identificatore del nodo e condivisa con la rete. Poiché lo spazio delle chiavi è enorme, fintanto che ogni nodo genera la chiave privata in modo casuale, avrà una chiave pubblica univoca, che potrà quindi identificarlo univocamente sulla rete.

Indirizzo di Rete del Nodo

Inoltre, ogni nodo pubblicizza anche un indirizzo di rete dove può essere raggiunto, in uno dei diversi formati possibili:

TCP/IP

: Un indirizzo IPv4 o IPv6 e un numero di porta TCP

TCP/Tor

: Un indirizzo Tor "onion" e un numero di porta TCP

L'identificatore dell'indirizzo di rete è scritto come `Indirizzo:Porta`, che è coerente con gli standard internazionali per gli identificatori di rete utilizzati sul web.

Ad esempio, il nodo di Riccardo con la chiave pubblica `02b8c29...2d9e10c` attualmente pubblicizza tre diversi indirizzi di rete: TCP/IP (v4 e v6) e TCP/Tor (onion):

`51.178.82.201:9735`

`[2001:41d0:404:200::5082]:9735`

`wstxzgbzof5ecpnkacm6b4ljgr5nx5xwucxc2rw53t5cqygv5c23mqad.onion:9735`

SUGGERIMENTO	La porta TCP predefinita per Lightning Network è 9735, ma un nodo può scegliere di utilizzare qualsiasi porta TCP.

Identificatori di Nodo

Assieme, la chiave pubblica del nodo e l'indirizzo di rete sono scritti nel seguente formato, separati da un simbolo @, come `IDNodo@Indirizzo:Porta`.

Quindi gli identificatori completi per il nodo di Riccardo sono:

`02b8c29e92f4842d06c5f6a5e8c53be29f5a6cf4bfd3c4d57d12aae3b0f2d9e10c@51.178.82.201:9735`

`02b8c29e92f4842d06c5f6a5e8c53be29f5a6cf4bfd3c4d57d12aae3b0f2d9e10c@[2001:41d0:404:200::5082]:9735`

`02b8c29e92f4842d06c5f6a5e8c53be29f5a6cf4bfd3c4d57d12aae3b0f2d9e10c@wstxzgbzof5ecpnkacm6b4ljgr5nx5xwucxc2rw53t5cqygv5c23mqad.onion:9735`

SUGGERIMENTO	L'alias del nodo possono anche essere specificati come domini (es. ln.dominio.com), per permettere una migliore leggibilità. Ogni operatore di nodo può annunciare qualunque alias desideri e non esiste alcun meccanismo che impedisca agli operatori di nodo di selezionare un alias che è già in uso. Pertanto, per fare riferimento a un nodo, è necessario utilizzare lo schema `IDNodo@Indirizzo:Porta`.

Gli identificatori precedenti sono spesso codificati in un codice QR, rendendone più facile agli utenti la scansione nel caso desiderino connettere il proprio nodo al nodo identificato da quell'identificatore.

Proprio come i nodi Bitcoin, i nodi Lightning pubblicizzano la loro presenza sulla rete Lightning "spettegolando" (ricordiamoci il concetto di gossip p2p) la chiave pubblica del nodo

e l'indirizzo di rete. In questo modo, altri nodi possono trovarli e mantenere un inventario (database) di tutti i nodi noti a cui possono connettersi e scambiare i messaggi definiti nel protocollo di messaggistica Lightning P2P.

Connessione Diretta a Nodi come Peer

Affinché il nodo di Alice possa connettersi al nodo di Bob, avrà bisogno della chiave pubblica del nodo di Bob o dell'indirizzo completo contenente la chiave pubblica, l'indirizzo IP o Tor e la porta. Poiché Bob gestisce un negozio, l'indirizzo del nodo di Bob può essere recuperato da una fattura o da una pagina di pagamento del suo negozio sul Web. Alice può scansionare un codice QR che contiene l'indirizzo e istruire il suo nodo a connettersi al nodo di Bob.

Una volta che Alice è connessa al nodo di Bob, i loro nodi sono ora connessi direttamente.

SUGGERIMENTO	Per aprire un canale di pagamento, due nodi devono prima essere collegati come peer diretti aprendo una connessione su Internet (o Tor).

Costruzione del Canale

Ora che i nodi Lightning di Alice e Bob sono connessi, possono iniziare il processo di costruzione di un canale di pagamento. In questa sezione esamineremo le comunicazioni tra i loro nodi, noto come Lightning Peer Protocol for Channel Management, e il protocollo crittografico che usano per costruire transazioni Bitcoin.

SUGGERIMENTO	Descriveremo due diversi protocolli in questo scenario. Innanzitutto, esiste un protocollo di messaggio che stabilisce come i nodi Lightning comunicano su Internet e quali messaggi si scambiano tra loro. In secondo luogo, c'è un protocollo crittografico, che stabilisce come i due nodi costruiscono e firmano le transazioni Bitcoin.

Peer Protocol per la Gestione dei Canali

Il Lightning Peer Protocol for Channel Management è definito in BOLT #2: Peer Protocol for Channel Management. In questo capitolo esamineremo più dettagliatamente le sezioni "Creazione del canale" e "Chiusura del canale" di BOLT #2.

Flusso di Messaggi di Creazione del Canale

La creazione del canale è ottenuta mediante lo scambio di sei messaggi tra i nodi di Alice e Bob (tre da ciascun peer): `open_channel`, `accept_channel`, `funding_created`, `funding_signed`, `funding_locked` e `funding_locked`. I sei messaggi sono mostrati come un diagramma di sequenza temporale nella Figura 7-3.

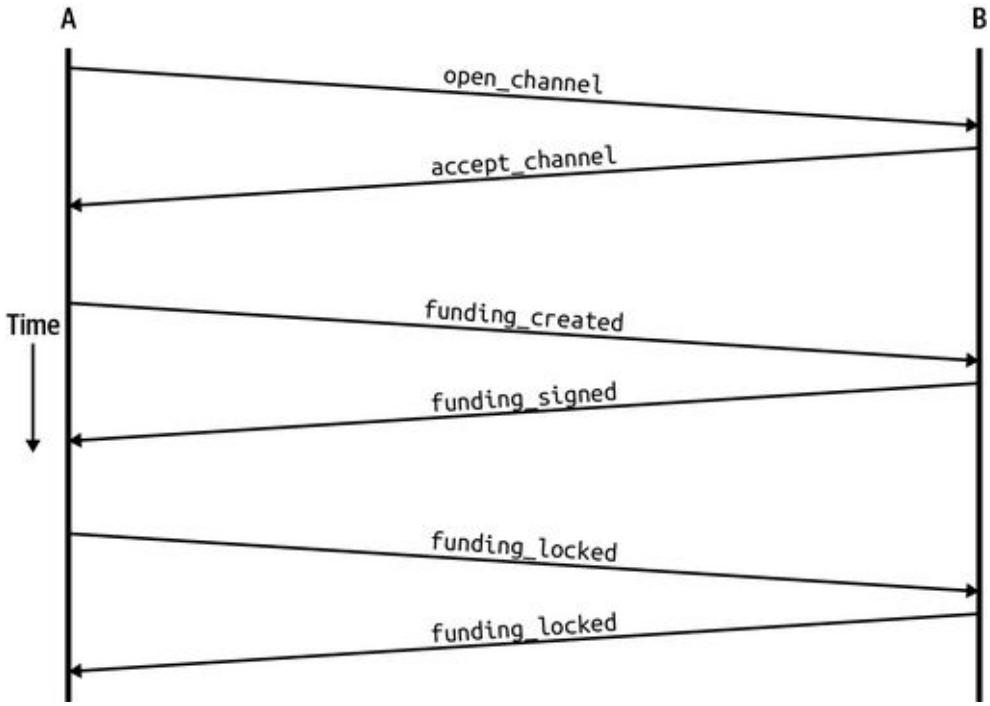

Figura 7-3. Il flusso di messaggi di creazione del canale

Nella Figura 7-3, i nodi di Alice e Bob sono rappresentati dalle linee verticali "A" e "B" su entrambi i lati del diagramma. Un diagramma di sequenza temporale come questo mostra il tempo che scorre verso il basso e i messaggi che scorrono da un lato all'altro tra i due partner che comunicano tra di loro. Le linee sono inclinate verso il basso per rappresentare il tempo trascorso necessario per trasmettere ciascun messaggio e la direzione del messaggio è indicata da una freccia alla fine di ogni riga.

La creazione del canale comprende tre parti. Innanzitutto, i due peer comunicano le proprie capacità e aspettative, con Alice che avvia una richiesta tramite `open_channel` e Bob che accetta la richiesta di canale tramite `accept_channel`.

In secondo luogo, Alice costruisce le transazioni di finanziamento e rimborso (come vedremo più avanti in questa sezione) e invia `funding_created` a Bob. Un altro nome per la transazione di "rimborso" è una transazione di "impegno", in quanto si impegna alla distribuzione corrente dei saldi nel canale. Bob risponde restituendo le firme necessarie con `funding_signed`. Questa interazione è la base per il *protocollo crittografico* per proteggere il canale e prevenire il furto. Alice ora trasmetterà la transazione di finanziamento (on-chain) per stabilire e ancorare il canale di pagamento. La transazione dovrà essere confermata sulla blockchain di Bitcoin.

SUGGERIMENTO	Il nome del messaggio `funding_signed` può creare della confusione. Questo messaggio non contiene una firma per la transazione di finanziamento ma la firma di Bob per la transazione di rimborso che consente ad Alice di richiedere indietro i suoi bitcoin dal multisig.

Una volta che la transazione ha conferme sufficienti (come definito dal campo `minimum_depth` nel messaggio `accept_channel`), Alice e Bob si scambiano messaggi `funding_locked` e il canale entra in modalità operativa.

Il messaggio open_channel

Il nodo di Alice richiede un canale di pagamento con il nodo di Bob inviando un messaggio `open_channel`. Il messaggio contiene informazioni sulle *aspettative* di Alice per la configurazione del canale, che Bob può accettare o rifiutare.

La struttura del messaggio `open_channel` (presa da BOLT #2) è mostrata in Esempio 7-1.

Esempio 7-1. Il messaggio open_channel

```
[chain_hash:chain_hash]
[32*byte:temporary_channel_id]
[u64:funding_satoshis]
[u64:push_msat]
[u64:dust_limit_satoshis]
[u64:max_htlc_value_in_flight_msat]
[u64:channel_reserve_satoshis]
[u64:htlc_minimum_msat]
```

```
[u32:feerate_per_kw]
[u16:to_self_delay]
[u16:max_accepted_htlcs]
[point:funding_pubkey]
[point:revocation_basepoint]
[point:payment_basepoint]
[point:delayed_payment_basepoint]
[point:htlc_basepoint]
[point:first_per_commitment_point]
[byte:channel_flags]
[open_channel_tlvs:tlvs]
```

I campi contenuti in questo messaggio specificano i parametri del canale desiderati da Alice, nonché varie impostazioni di configurazione dai nodi di Alice che riflettono le aspettative di sicurezza per il funzionamento del canale.

Alcuni dei parametri di costruzione del canale sono elencati di seguito:

`chain_hash`

> Questo identifica quale blockchain (ad es. Bitcoin mainnet) verrà utilizzata per questo canale. Di solito è l'hash del blocco di genesi di quella blockchain.

`funding_satoshis`

> L'importo che Alice utilizzerà per finanziare il canale, ovvero la capacità totale del canale.

`channel_reserve_satoshis`

> Il saldo minimo, in satoshi, riservato su ciascun lato di un canale. Torneremo su questo punto quando parleremo di penalità.

`push_msat`

> Un importo facoltativo che Alice "spingerà" immediatamente a Bob come pagamento

al momento del finanziamento del canale. Impostare questo valore su qualsiasi cosa tranne 0 significa effettivamente regalare denaro al tuo partner di canale e dovrebbe essere usato con cautela.

to_self_delay

Un parametro di sicurezza molto importante per il protocollo. Il valore nel messaggio `open_channel` viene utilizzato nella transazione di impegno del risponditore e `accept_channel` in quella dell'iniziatore. Questa asimmetria esiste per consentire a ciascuna parte di esprimere quanto tempo l'altra parte deve attendere per richiedere unilateralmente i fondi in una transazione di impegno. Se Bob in qualsiasi momento chiude unilateralmente il canale contro la volontà di Alice, si impegna a non accedere ai propri fondi per il ritardo qui definito. Più alto è questo valore, maggiore è la sicurezza di Alice, ma più a lungo Bob potrebbe avere i suoi fondi bloccati.

funding_pubkey

La chiave pubblica che Alice contribuirà al multisig 2-di-2 che ancora questo canale.

X_basepoint

Chiavi madre, utilizzate per derivare chiavi figlie per varie parti dell'impegno, revoca, pagamenti instradati (HTLC) e transazioni di chiusura. Queste verranno utilizzate e spiegate nei capitoli successivi.

SUGGERIMENTO	Se vuoi comprendere gli altri campi e messaggi del protocollo peer che non vengono discussi in questo libro, ti suggeriamo di cercarli nelle specifiche BOLT. Questi messaggi e campi sono importanti, ma non possono essere trattati in modo sufficientemente dettagliato nell'ambito di questo libro. Vogliamo che tu comprenda i principi fondamentali abbastanza bene da poter inserire i dettagli leggendo le specifiche del protocollo effettivo (BOLT).

Il messaggio accept_channel

In risposta al messaggio `open_channel` di Alice, Bob restituisce il messaggio `accept_channel` mostrato in Esempio 7-2.

Esempio 7-2. Il messaggio accept_channel

```
[32*byte:temporary_channel_id]
[u64:dust_limit_satoshis]
[u64:max_htlc_value_in_flight_msat]
[u64:channel_reserve_satoshis]
[u64:htlc_minimum_msat]
[u32:minimum_depth]
[u16:to_self_delay]
[u16:max_accepted_htlcs]
[point:funding_pubkey]
[point:revocation_basepoint]
[point:payment_basepoint]
[point:delayed_payment_basepoint]
[point:htlc_basepoint]
[point:first_per_commitment_point]
[accept_channel_tlvs:tlvs]
```

Come puoi vedere, questo è simile al messaggio `open_channel` e contiene le aspettative del nodo e i valori di configurazione di Bob.

I due campi più importanti in `accept_channel` che Alice utilizzerà per costruire il canale di pagamento sono:

`funding_pubkey`

> La chiave pubblica del nodo di Bob che contribuisce all'indirizzo multisig 2-di-2 ed ancora il canale.

`minimum_depth`

> Il numero di conferme che il nodo di Bob si aspetta per la transazione di finanziamento prima di considerare il canale "aperto" e pronto per l'uso.

La Transazione di Finanziamento

Una volta che il nodo di Alice riceve il messaggio `accept_channel` di Bob, dispone delle informazioni necessarie per costruire la *transazione di finanziamento* che ancora il canale alla blockchain di Bitcoin. Come discusso nei capitoli precedenti, un canale di pagamento Lightning è ancorato a un indirizzo multifirma 2-di-2. Innanzitutto, dobbiamo generare quell'indirizzo multifirma per consentirci di costruire la transazione di finanziamento (e la transazione di rimborso come descritto successivamente).

Generazione di un Indirizzo Multifirma

La transazione di finanziamento invia una certa quantità di bitcoin (`funding_satoshis` dal messaggio `open_channel`) a un output multifirma 2-di-2 costruito dalle chiavi pubbliche `funding_pubkey` di Alice e Bob.

Il nodo di Alice costruisce uno script multisignature come mostrato di seguito:

```
2 <Alice_funding_pubkey> <Bob_funding_pubkey> 2 CHECKMULTISIG
```

Nota che, in pratica, le chiavi di finanziamento sono *ordinate* in modo deterministico (utilizzando l'ordine lessicografico della forma compressa serializzata delle chiavi pubbliche) prima di essere inserite nello script di witness. Accettando questo ordine ordinato in anticipo, ci assicuriamo che entrambe le parti costruiscano un output di transazione di finanziamento identico, che è firmato dalla firma della transazione di impegno scambiata.

Questo script è codificato come un indirizzo Bitcoin Pay-to-Witness-Script-Hash (P2WSH), che assomiglia a questo:

```
bc1q89ju02heg32yrqdrnqghe6132wek25p6sv6e564znvrvez7tq5zqt4dn02
```

Costruzione della Transazione di Finanziamento

Il nodo di Alice può ora costruire una transazione di finanziamento inviando l'importo concordato con Bob (`funding_satoshis`) all'indirizzo multisig 2-di-2. Supponiamo che `funding_satoshis` fosse 140.000 e che Alice stia spendendo un output di 200.000 satoshi e creando un resto di 60.000 satoshi. La transazione sarà simile a quella in Figura 7-4.

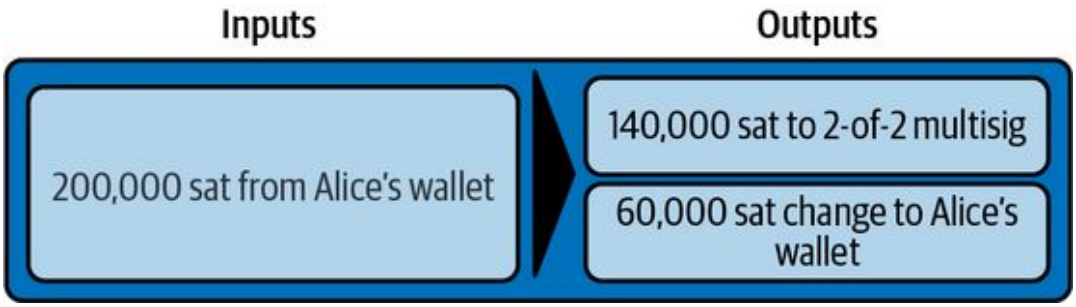

Figura 7-4. Alice costruisce la transazione di finanziamento

Alice non trasmette questa transazione perché così facendo metterebbe a rischio i suoi 140.000 satoshi. Una volta spesi verso multisig 2-di-2, non c'è modo per Alice di recuperare i suoi soldi senza la firma di Bob.

Canali di pagamento a doppio finanziamento

In un canale a doppio finanziamento, sia Alice che Bob contribuirebbero alla transazione di finanziamento. I canali a doppio finanziamento richiedono un flusso di messaggi e un protocollo crittografico leggermente più complicato.

Conservare Transazioni Firmate Senza Trasmetterle

Un'importante caratteristica di Bitcoin che rende possibile Lightning è la capacità di costruire e firmare transazioni, ma di non trasmetterle. La transazione è *valida* in ogni modo, ma finché non viene trasmessa e confermata sulla blockchain di Bitcoin non viene riconosciuta e i suoi output non sono spendibili perché non sono stati creati sulla blockchain. Utilizzeremo questa funzionalità molte volte su LN e il nodo di Alice utilizza la funzionalità durante la costruzione della transazione di finanziamento: trattenendola e non trasmettendola.

Rimborso Prima del Finanziamento

Per evitare la perdita di fondi, Alice non può mettere i suoi bitcoin in un multisig 2-di-2 fino a quando non avrà un modo per ottenere un rimborso se le cose vanno male. Essenzialmente, deve pianificare l'"uscita" dal canale prima di entrare in questo accordo.

Considera il costrutto legale di un accordo prematrimoniale. Quando due persone si

sposano, il loro denaro è vincolato per legge (a seconda della giurisdizione). Prima di sposarsi, possono firmare un accordo che specifichi come separare i propri beni in caso di scioglimento del matrimonio mediante divorzio.

Possiamo creare un accordo simile in Bitcoin. Ad esempio, possiamo creare una transazione di rimborso, che funziona come un contratto prematrimoniale, consentendo alle parti di decidere come dividere i fondi nel loro canale prima che i loro fondi siano effettivamente bloccati nell'indirizzo di finanziamento multifirma.

Costruzione della Transazione di Rimborso Prefirmata

Alice costruirà la transazione di rimborso immediatamente dopo aver costruito (ma non trasmesso) la transazione di finanziamento. La transazione di rimborso spende il multisig 2-di-2 nel portafoglio di Alice. Chiamiamo questa transazione di rimborso una *transazione di impegno* perché impegna entrambi i partner di canale a distribuire equamente il saldo del canale. Poiché Alice ha finanziato il canale da sola, ottiene l'intero saldo e sia Alice che Bob si impegnano a rimborsare Alice con questa transazione.

In pratica è un po' più complicato come vedremo nei capitoli successivi, ma per ora manteniamo le cose semplici e assumiamo che assomigli alla Figura 7-5.

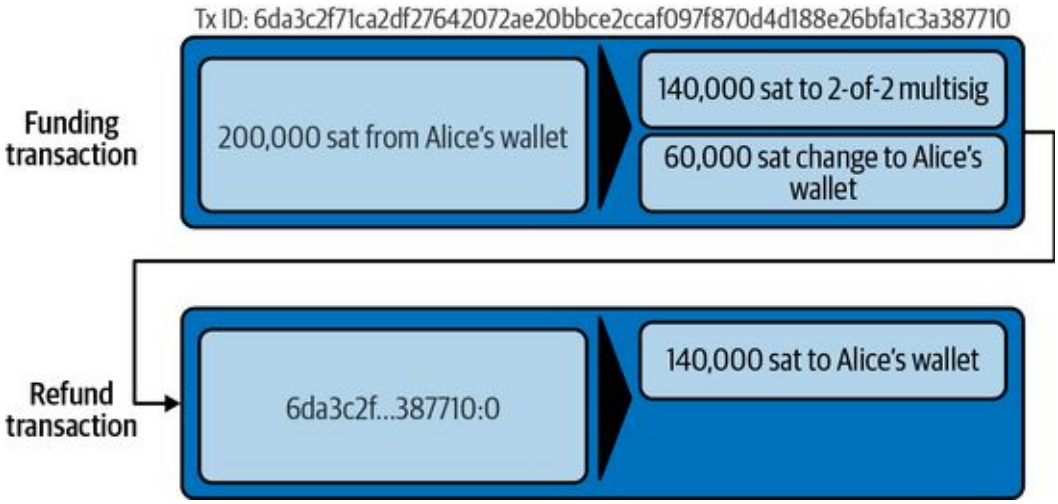

Figura 7-5. Alice costruisce anche la transazione di rimborso

Più avanti in questo capitolo vedremo come è possibile effettuare più transazioni di impegno per distribuire il saldo del canale in importi diversi.

Concatenare Transazioni Senza Trasmetterle

Come mostrato in Figura 7-5, Alice ha costruito due transazioni concatenate. Ti starai chiedendo come sia possibile, giusto? Alice non ha trasmesso la transazione di finanziamento alla blockchain di Bitcoin. Fino a prova contraria, sulla blockchain, quella transazione non esiste. L'operazione di rimborso è costruita in modo da spendere uno degli output dell'operazione di finanziamento, anche se quell'output esiste ancora. Come puoi spendere un output che non è stato confermato sulla blockchain di Bitcoin?

La transazione di rimborso non è ancora una transazione valida. Perché diventi una transazione valida devono accadere due cose:

- La transazione di finanziamento deve essere trasmessa alla rete di Bitcoin. (Per garantire la sicurezza di LN, richiederemo anche che sia confermata sulla blockchain, sebbene ciò non sia strettamente necessario per concatenare le transazioni).
- L'input della transazione di rimborso richiede le firme di Alice e Bob.

Anche se queste due cose non sono accadute, e anche se il nodo di Alice non ha trasmesso la transazione di finanziamento, può ancora costruire la transazione di rimborso. Può farlo

perché può calcolare l'hash della transazione di finanziamento e farvi riferimento come input nella transazione di rimborso.

Nota come Alice ha calcolato `6da3c2...387710` come hash della transazione di finanziamento? Se e quando la transazione di finanziamento viene trasmessa, tale hash verrà registrato come ID transazione della transazione di finanziamento. Pertanto, l'output `0` della transazione di finanziamento (l'output dell'indirizzo 2-di-2) verrà quindi indicato come ID output `6da3c2...387710:0`. La transazione di rimborso può essere costruita per spendere l'output della transazione di finanziamento anche se non esiste ancora, perché Alice sa quale sarà il suo identificatore una volta confermato.

Ciò significa che Alice può creare una transazione concatenata facendo riferimento a un output che non esiste ancora, sapendo che il riferimento sarà valido se la transazione di finanziamento viene confermata, rendendo valida anche la transazione di rimborso. Come vedremo nella prossima sezione, questo "trucco" di concatenare le transazioni prima che vengano trasmesse richiede una caratteristica molto importante di Bitcoin che è stata introdotta nell'agosto del 2017: *Segregated Witness*.

Risolvere la Malleabilità (Segregated Witness)

Alice deve dipendere dal fatto che l'ID della transazione di finanziamento sia noto prima della conferma. Ma prima dell'introduzione di Segregated Witness (SegWit) nell'agosto 2017, questo non era sufficiente per proteggere Alice. A causa del modo in cui le transazioni sono state costruite con le firme (witnesses - testimoni) incluse nell'ID transazione, era possibile per una terza parte (ad esempio, Bob) trasmettere una versione alternativa di una transazione con un ID transazione *malleato* (modificato). Questo è noto come *malleabilità delle transazioni* e, prima di SegWit, questo problema rendeva difficile l'implementazione sicura di canali di pagamento a durata indefinita.

Se Bob potesse modificare la transazione di finanziamento di Alice prima che venga confermata e produrre una replica con un ID di transazione diverso, Bob potrebbe rendere non valida la transazione di rimborso di Alice e bloccare i suoi bitcoin. Alice sarebbe alla mercé di Bob per ottenere una firma per sbloccare i suoi fondi e potrebbe essere facilmente ricattata. Bob non poteva rubare i fondi, ma poteva impedire ad Alice di riaverli indietro.

L'introduzione di SegWit ha reso gli ID transazione non confermati immutabili dal punto di vista di terze parti, il che significa che Alice poteva essere certa che l'ID transazione della

transazione di finanziamento non sarebbe cambiato. Di conseguenza, Alice può essere sicura che se ottiene la firma di Bob sulla transazione di rimborso, ha un modo per recuperare i suoi soldi. Ora ha un modo per implementare l'equivalente Bitcoin di un "accordo prematrimoniale" prima di bloccare i suoi fondi nel multisig.

SUGGERIMENTO	Potresti esserti chiesto come Bob potrebbe alterare (malleare) una transazione creata e firmata da Alice. Bob certamente non ha le chiavi private di Alice. Tuttavia le firme ECDSA per un messaggio non sono univoche. Conoscere una firma (che è inclusa in una transazione valida) consente di produrre molte firme dall'aspetto diverso che sono ancora valide. Prima che SegWit rimuovesse le firme dall'algoritmo digest della transazione, Bob poteva sostituire la firma con una firma valida equivalente che producesse un ID transazione diverso, interrompendo la catena tra la transazione di finanziamento e la transazione di rimborso.

Il messaggio funding_created

Ora che Alice ha costruito le transazioni necessarie, il flusso di messaggi di costruzione del canale continua. Alice trasmette il messaggio `funding_created` a Bob. Puoi vedere il contenuto di questo messaggio di seguito.

```
[32*byte:temporary_channel_id]
[sha256:funding_txid]
[u16:funding_output_index]
[signature:signature]
```

Con questo messaggio, Alice fornisce a Bob le informazioni importanti sulla transazione di finanziamento che ancora il canale di pagamento:

`funding_txid`

Questo è l'ID transazione (TxID) della transazione di finanziamento e viene utilizzato per creare l'ID canale una volta stabilito il canale.

`funding_output_index`

Questo è l'indice di output, quindi Bob sa quale output della transazione (ad esempio,

l'output `0`) è l'output multisig 2-di-2 finanziato da Alice. Viene utilizzato anche per formare l'ID del canale.

Infine, Alice invia anche la `signature` corrispondente al `funding_pubkey` di Alice e utilizzato per spendere dal multisig 2-di-2. Questo è necessario a Bob perché dovrà anche creare la propria versione di una transazione di impegno. Quella transazione di impegno richiede una firma di Alice, che lei gli fornisce. Nota che le transazioni di impegno di Alice e Bob hanno un aspetto leggermente diverso, quindi le firme saranno diverse. Sapere come appare la transazione di impegno dell'altra parte è fondamentale e fa parte del protocollo per fornire la firma valida.

SUGGERIMENTO	Nel protocollo Lightning vediamo spesso nodi che inviano firme invece di intere transazioni firmate. Questo perché entrambe le parti possono ricostruire la stessa transazione e quindi è necessaria solo la firma per renderla valida. L'invio della sola firma e non dell'intera transazione consente di risparmiare molta larghezza di banda della rete.

Il messaggio funding_signed

Dopo aver ricevuto il messaggio `funding_created` da Alice, Bob ora conosce l'ID della transazione di finanziamento e l'indice di output. L'ID del canale è costituito da un "exclusive or" (XOR) dell'ID della transazione di finanziamento e dell'indice di output:

```
channel_id = funding_txid XOR funding_output_index
```

Più precisamente, un `channel_id`, che è la rappresentazione a 32 byte di un UTXO di finanziamento, viene generato mediante XORing dei 2 byte inferiori del TxID di finanziamento con l'indice dell'output di finanziamento.

Bob dovrà anche inviare ad Alice la sua firma per la transazione di rimborso, basata sulla `funding_pubkey` di Bob che ha formato il multisig 2-di-2. Sebbene Bob abbia già la sua transazione di rimborso locale, ciò consentirà ad Alice di completare la transazione di rimborso con tutte le firme necessarie e assicurarsi che i suoi soldi siano rimborsabili nel caso qualcosa vada storto.

Bob costruisce un messaggio `funding_signed` e lo invia ad Alice. Di seguito vediamo il contenuto di questo messaggio:

`[channel_id:channel_id]`

`[signature:signature]`

Trasmissione della Transazione di Finanziamento

Dopo aver ricevuto il messaggio `funding_signed` da Bob, Alice ora ha entrambe le firme necessarie per firmare la transazione di rimborso. Il suo "piano di uscita" è ora sicuro e quindi può trasmettere la transazione di finanziamento senza timore che i suoi fondi siano bloccati. Se qualcosa va storto, Alice può semplicemente trasmettere la transazione di rimborso e recuperare i suoi soldi, senza ulteriore aiuto da parte di Bob.

Alice ora invia la transazione di finanziamento alla rete Bitcoin in modo che possa essere inclusa in un blocco e minata nella blockchain. Sia Alice che Bob staranno attenti a questa transazione e aspetteranno conferme secondo il `minimum_depth` (ad esempio, sei conferme) sulla blockchain di Bitcoin.

SUGGERIMENTO	Ovviamente Alice utilizzerà il protocollo Bitcoin per verificare che la firma che Bob le ha inviato sia effettivamente valida. Questo passaggio è molto cruciale. Se per qualche motivo Bob avesse inviato dati errati ad Alice, il suo "piano di uscita" sarebbe stato sabotato.

Il messaggio funding_locked

Non appena la transazione di finanziamento ha raggiunto il numero richiesto di conferme, sia Alice che Bob si scambiano il messaggio `funding_locked` e il canale è pronto per l'uso.

Invio di Pagamenti Attraverso il Canale

Il canale è stato impostato, ma nel suo stato iniziale tutta la capacità (140.000 satoshi) è dalla parte di Alice. Ciò significa che Alice può inviare pagamenti a Bob attraverso il canale, ma Bob non ha ancora fondi da inviare ad Alice.

Nelle prossime sezioni mostreremo come vengono effettuati i pagamenti attraverso il canale di pagamento e come viene aggiornato lo *stato del canale* (*channel state*).

Supponiamo che Alice voglia inviare 70.000 satoshi a Bob per pagare il conto al bar di Bob.

Suddivisione del Saldo

In linea di principio, l'invio di un pagamento da Alice a Bob è semplicemente una questione di ridistribuzione del saldo del canale. Prima che il pagamento venga inviato, Alice ha 140.000 satoshi e Bob non ne ha. Dopo che il pagamento di 70.000 satoshi è stato inviato, Alice ha 70.000 satoshi e Bob ha 70.000 satoshi.

Pertanto, tutto ciò che Alice e Bob devono fare è creare e firmare una transazione che spenda il multisig 2-di-2 in due output pagando ad Alice e Bob i saldi corrispondenti. Chiamiamo questa transazione aggiornata una *transazione di impegno*.

Alice e Bob gestiscono il canale di pagamento facendo avanzare lo stato del canale attraverso una serie di impegni. Ogni impegno aggiorna i saldi per riflettere i pagamenti che sono passati attraverso il canale. Sia Alice che Bob possono avviare un nuovo impegno per aggiornare il canale.

Nella Figura 7-6 vediamo diverse transazioni di impegno...

La prima transazione di impegno mostrata in Figura 7-6 è la transazione di rimborso che Alice ha costruito prima di finanziare il canale. Nel diagramma, questo è il Commitment #0. Dopo che Alice ha pagato a Bob 70.000 satoshi, la nuova transazione di impegno (Commitment #1) ha due output che pagano ad Alice e Bob i rispettivi saldi. Abbiamo incluso due successive transazioni di impegno (Commitment #2 e Commitment #3) che rappresentano Alice che paga a Bob altri 10.000 satoshi e poi altri 20.000 satoshi.

Ogni transazione di impegno firmata e valida può essere utilizzata da entrambi i partner di canale in qualsiasi momento per chiudere il canale trasmettendolo alla rete Bitcoin. Poiché entrambi dispongono della transazione di impegno più recente e possono utilizzarla in qualsiasi momento, possono anche semplicemente trattenerla e non trasmetterla. È la loro garanzia di una giusta uscita dal canale.

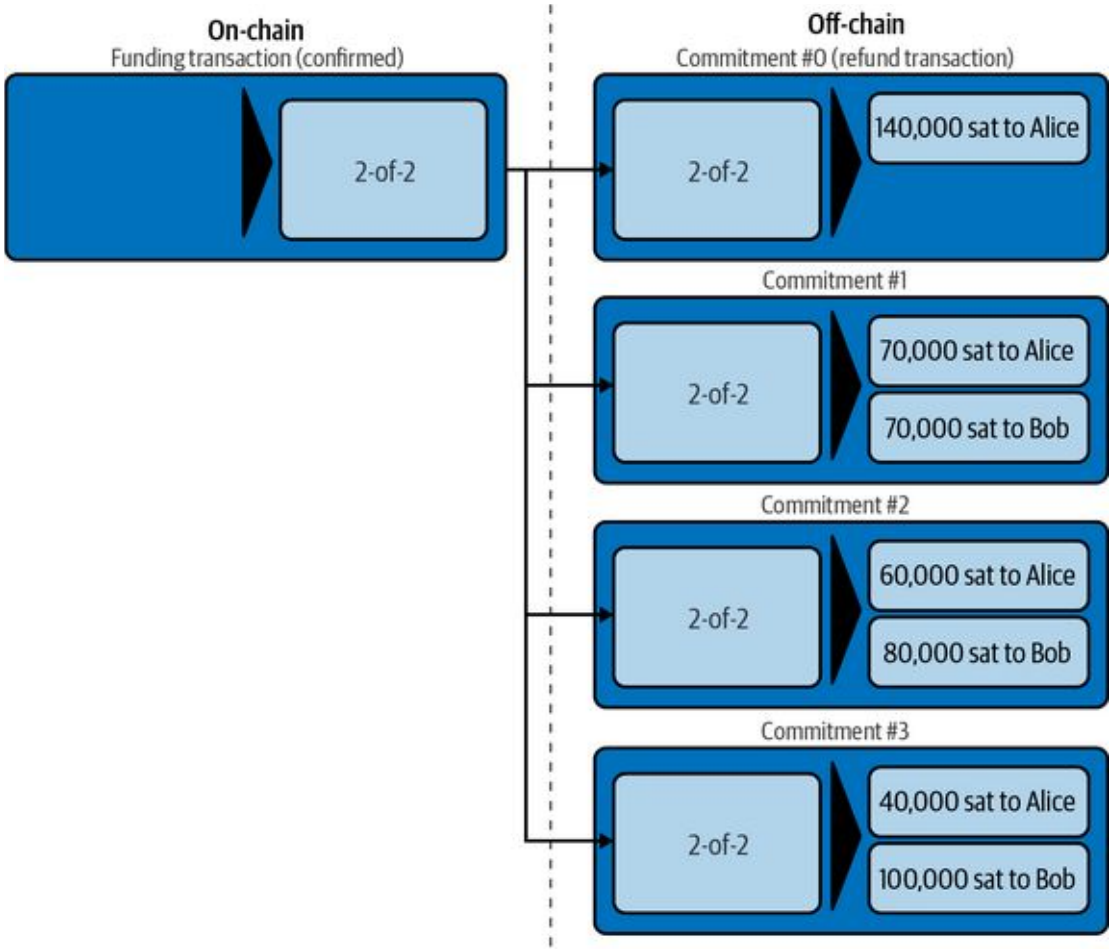

Figura 7-6. Più transazioni di impegno

Impegni in Competizione

Ci si potrebbe chiedere come sia possibile che Alice e Bob abbiano più transazioni di impegno, tentando tutte di spendere lo stesso output 2-di-2 dalla transazione di finanziamento. Queste transazioni di impegno non sono in conflitto? Non è questa una "doppia spesa" che il sistema Bitcoin intende impedire?

Lo è! In effetti, ci affidiamo alla capacità di Bitcoin di *impedire* una doppia spesa per far funzionare Lightning. Indipendentemente dal numero di transazioni di impegno che Alice e Bob costruiscono e firmano, solo una di esse può effettivamente essere confermata.

Finché Alice e Bob mantengono queste transazioni e non le trasmettono, l'output del finanziamento non viene speso. Ma se una transazione di impegno viene trasmessa e

confermata, spenderà l'output del finanziamento. Se Alice o Bob tentano di trasmettere più di una transazione di impegno, solo una di esse verrà confermata e le altre verranno rifiutate in quanto tentativi (falliti) di doppia spesa.

Se viene trasmessa più di una transazione di impegno, ci sono molti fattori che determineranno quale verrà confermata per prima: l'importo delle commissioni incluse, la velocità di propagazione di queste transazioni concorrenti, la topologia della rete, ecc. Essenzialmente diventa una corsa senza un prevedibile risultato. Non sembra molto sicuro... C'è la possibilità che qualcuno possa imbrogliare?

Barare con Transazioni di Vecchi Impegni

Esaminiamo più attentamente le transazioni di impegno nella Figura 7-6. Tutte e quattro le transazioni di impegno sono firmate e valide... ma solo l'ultima riflette accuratamente i saldi più recenti del canale. In questo particolare scenario, Alice ha l'opportunità di imbrogliare trasmettendo un vecchio impegno e ottenendone la conferma sulla blockchain di Bitcoin. Supponiamo che Alice trasmetta il Commitment #0 e che tale viene poi confermato. In questo caso chiuderebbe il canale e prenderebbe tutti i 140.000 satoshi. Infatti, in questo particolare esempio qualsiasi impegno diverso dal Commitment #3 migliora la posizione di Alice e le consente di "annullare" almeno una parte dei pagamenti riflessi nel canale.

Nella sezione successiva vedremo come Lightning Network risolve questo problema, impedendo che le transazioni di impegno precedenti vengano utilizzate dai partner di canale mediante un meccanismo di revoca e sanzioni. Esistono altri modi per impedire la trasmissione di transazioni di impegno precedenti, come i canali eltoo, ma richiedono un aggiornamento a Bitcoin chiamato input rebinding (che tratteremo alla fine del libro).

Revoca delle Vecchie Transazioni di Impegno

Le transazioni Bitcoin non scadono e non possono essere "annullate". Né possono essere fermate o censurate una volta che vengono trasmesse. Quindi, come possiamo "revocare" una transazione che un'altra persona detiene e che è già stata firmata?

La soluzione utilizzata in Lightning è un altro esempio di protocollo di equità. Invece di cercare di controllare la capacità di trasmettere una transazione, esiste un *meccanismo di penalità* integrato che garantisce che non sia nel migliore interesse di un aspirante

imbroglione trasmettere una vecchia transazione di impegno. Potrebbero sempre trasmetterla, ma molto probabilmente perderebbero denaro se lo facessero.

SUGGERIMENTO	La parola "revoca" è un termine improprio perché implica che gli impegni precedenti siano in qualche modo resi non validi e non possano essere trasmessi e confermati. Non è così, poiché le transazioni Bitcoin valide non possono essere revocate. Invece, il protocollo Lightning utilizza un meccanismo di penalità per punire il partner di canale che trasmette un vecchio impegno.

Tre sono gli elementi che compongono il meccanismo di revoca e penalità:

Transazioni con impegni asimmetrici

Le transazioni di impegno di Alice sono leggermente diverse da quelle di Bob.

Spesa ritardata

Il pagamento alla parte titolare dell'operazione di impegno è ritardato (timelocked), mentre il pagamento alla controparte può essere richiesto immediatamente.

Chiavi di revoca

Utilizzato per sbloccare un'opzione di penalità per vecchi impegni.

Diamo un'occhiata a questi tre elementi uno per uno...

Transazioni con impegni asimmetrici

Alice e Bob gestiscono transazioni di impegno leggermente diverse. Osserviamo il Commitment #2 della Figura 7-6, mostrato più dettagliatamente nella Figura 7-7 di seguito.

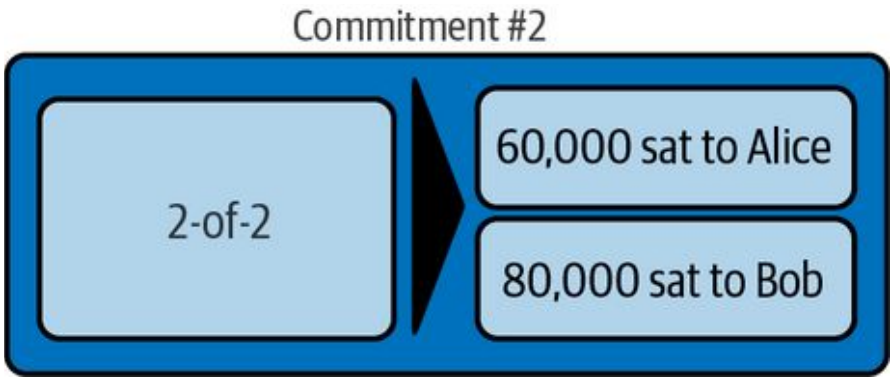

Figura 7-7. Transazione di impegno n.2

Alice e Bob detengono due diverse varianti di questa transazione, come da Figura 7-8.

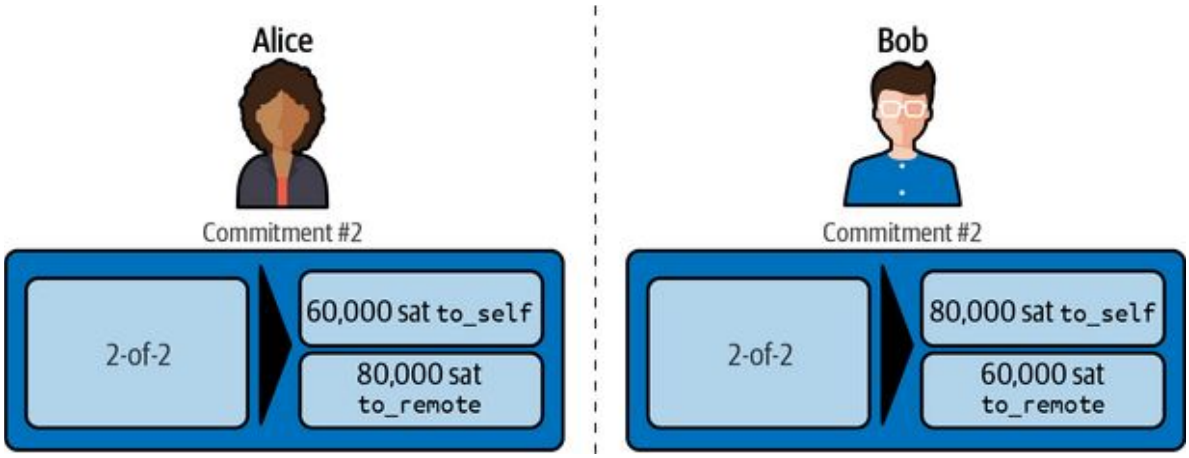

Figura 7-8. Transazioni con impegni asimmetrici

Per convenzione, all'interno del protocollo Lightning, ci riferiamo ai due partner di canale come `self` (noto anche come `local`) e `remote`, a seconda di quale lato stiamo guardando. Gli output che pagano ciascun partner di canale sono chiamati rispettivamente `to_local` e `to_remote`.

Nella Figura 7-8 vediamo che Alice detiene una transazione che paga 60.000 satoshi `to_self` (possono essere spesi dalle chiavi di Alice) e 80.000 satoshi `to_remote` (possono essere spesi dalle chiavi di Bob).

Bob detiene l'immagine speculare di quella transazione, dove il primo output è 80.000 satoshi `to_self` (può essere speso dalle chiavi di Bob) e 60.000 satoshi `to_remote` (può

essere speso dalle chiavi di Alice).

Spesa Ritardata (Timelocked) Verso Se Stessi

L'utilizzo di transazioni asimmetriche consente al protocollo di *attribuire* facilmente la colpa alla parte che tradisce. Un'invariante che la parte *emittente* deve sempre attendere garantisce che la parte "onesta" abbia il tempo di confutare la richiesta e revocare i propri fondi. Questa asimmetria si manifesta sotto forma di output diversi per ciascun lato: l'output `to_local` è sempre timelocked e non può essere speso immediatamente, mentre l'output `to_remote` non è timelocked e può essere speso immediatamente.

Nella transazione di impegno detenuta da Alice, ad esempio, l'output `to_local` che la paga è timelocked per 432 blocchi, mentre l'output `to_remote` che paga Bob può essere speso immediatamente (vedi la Figura 7-9). La transazione di impegno di Bob per il Commitment #2 è speculare: il suo output (`to_local`) è timelocked e l'output `to_remote` di Alice può essere speso immediatamente.

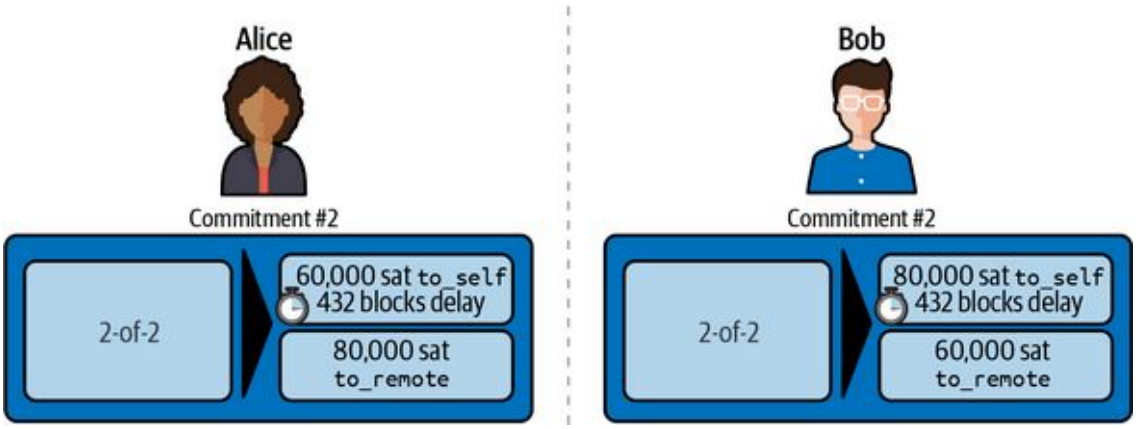

Figura 7-9. Transazioni di impegno asimmetriche e ritardate

Ciò significa che se Alice chiude il canale trasmettendo e confermando la transazione di impegno che detiene, non può spendere il suo saldo per 432 blocchi, ma Bob può richiedere immediatamente il suo saldo. Se Bob chiude il canale utilizzando la transazione di impegno che detiene, non può spendere il suo output per 432 blocchi mentre Alice può spendere immediatamente il suo.

Il ritardo c'è per un motivo: consentire al soggetto *remoto* di esercitare un'opzione di

penalità se un vecchio impegno (revocato) dovesse essere trasmesso dall'altro partner di canale. Diamo un'occhiata alle chiavi di revoca e all'opzione di penalità successiva.

Il ritardo viene negoziato da Alice e Bob durante il flusso iniziale di messaggi di costruzione del canale, come campo chiamato `to_self_delay`. Per garantire la sicurezza del canale, il ritardo viene ridimensionato in base alla capacità del canale, il che significa che un canale con più fondi ha ritardi più lunghi negli output di sé stessi negli impegni. Il nodo di Alice include un `to_self_delay` desiderato nel messaggio `open_channel`. Se Bob lo trova accettabile, il suo nodo include lo stesso valore per `to_self_delay` nel messaggio `accept_channel`. Se non sono d'accordo, il canale viene rifiutato (vedremo questo rifiuto a breve).

Chiavi di Revoca

Come abbiamo discusso in precedenza, la parola "revoca" è un po' fuorviante perché implica che la transazione "revocata" non può essere utilizzata.

In effetti, la transazione revocata può essere utilizzata, ma se viene utilizzata ed è stata revocata, uno dei partner di canale può prelevare tutti i fondi del canale creando una transazione di penalità.

Il modo in cui funziona è che l'output `to_local` non è solo timelocked, ma ha anche due condizioni di spesa nello script: può essere speso da *se stessi* dopo il ritardo timelock *o* può essere speso da *remoto* immediatamente con una chiave di revoca per questo impegno .

Quindi, nel nostro esempio, ciascuna parte detiene una transazione di impegno che include un'opzione di revoca nell'output `to_local`, come mostrato nella Figura 7-10.

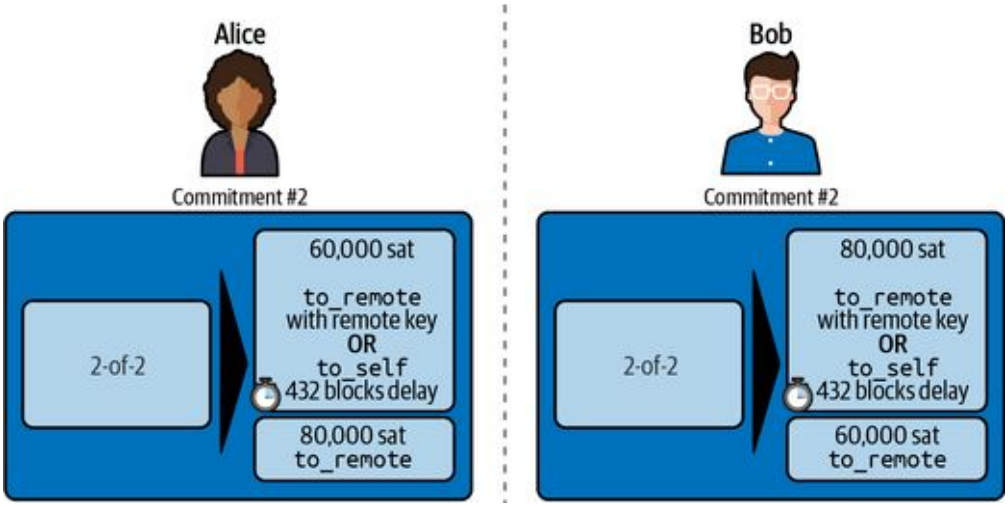

Figura 7-10. Impegni asimmetrici, differiti e revocabili

La Transazione di Impegno

Ora che comprendiamo la struttura delle transazioni di impegno e perché abbiamo bisogno di impegni asimmetrici, ritardati e revocabili, diamo un'occhiata al Bitcoin Script che lo implementa.

Il primo output (`to_local`) di una transazione di commit è definito in BOLT #3: Commitment Transaction, `to_local` Output, come segue:

```
OP_IF
    # Penalty transaction
    <revocationpubkey>
OP_ELSE
    <to_self_delay>
    OP_CHECKSEQUENCEVERIFY
    OP_DROP
    <local_delayedpubkey>
OP_ENDIF
OP_CHECKSIG
```

Questo è uno script condizionale, il che significa che l'output può essere speso *se una delle due* condizioni è soddisfatta. La prima clausola consente all'output di essere speso da

chiunque possa firmare per <revocationpubkey>. La seconda clausola è timelocked per <to_self_delay> *blocchi* e può essere spesa solo dopo quel numero di blocchi da chiunque possa firmare per <local_delayedpubkey>. Nel nostro esempio, avevamo impostato il timelock <to_self_delay> su 432 blocchi, ma si tratta di un ritardo configurabile che viene negoziato dai due partner di canale. La durata del timelock to_self_delay viene solitamente scelta in proporzione alla capacità del canale, il che significa che i canali di maggiore capacità (più fondi) hanno timelock to_self_delay più lunghi per proteggere le parti.

La prima clausola consente all'output di essere speso da chiunque possa firmare per <revocationpubkey>. Un requisito fondamentale per la sicurezza di questo script è che la parte remota *non possa* firmare unilateralmente con la revocationpubkey. Per capire perché ciò è importante, si consideri lo scenario in cui la parte remota viola un impegno revocato in precedenza. Se riesce a firmare con questa chiave, può *impossessarsi* della clausola di revoca e rubare tutti i fondi nel canale. Invece, deriviamo la revocationpubkey per *ogni* stato in base alle informazioni provenienti *sia* dalla parte locale che da quella remota. Viene sfruttata una combinazione intelligente di crittografia simmetrica e asimmetrica per consentire a entrambe le parti di calcolare la chiave pubblica di revocationpubkey, ma consentire solo alla parte locale onesta di calcolare la chiave privata in base alle proprie informazioni segrete, come spiegato di seguito.

Derivazioni Segrete di Revoca e Impegno

Ciascuna parte invia un revocation_basepoint durante i messaggi iniziali di negoziazione del canale e un first_per_commitment_point. Il revocation_basepoint è statico per la durata del canale, mentre ogni nuovo stato del canale sarà basato su un nuovo first_per_commitment_point.

Date queste informazioni, la revocationpubkey per ogni stato del canale viene derivata tramite la seguente serie di curve ellittiche e operazioni di hashing:

```
revocationpubkey = revocation_basepoint * sha256(revocation_basepoint ||
per_commitment_point) + per_commitment_point * sha256(per_commitment_point ||
revocation_basepoint)
```

A causa della proprietà commutativa dei gruppi abeliani su cui sono definite le curve ellittiche, una volta che il per_commitment_secret (la chiave privata per il per_commitment_point) viene rivelato dalla parte remota, la parte locale può derivare la

chiave privata per la `revocationpubkey` con la seguente operazione:

```
revocation_priv = (revocationbase_priv * sha256(revocation_basepoint ||
per_commitment_point)) + (per_commitment_secret * sha256(per_commitment_point
|| revocation_basepoint)) mod N
```

Per vedere perché funziona in pratica, si noti che possiamo riordinare (commutare) ed espandere il calcolo della chiave pubblica della formula originale per `revocationpubkey`:

```
revocationpubkey = G*(revocationbase_priv * sha256(revocation_basepoint ||
per_commitment_point) + G*(per_commitment_secret * sha256(per_commitment_point
|| revocation_basepoint)) = revocation_basepoint * sha256(revocation_basepoint
|| per_commitment_point) + per_commitment_point * sha256(per_commitment_point
|| revocation_basepoint))
```

In altre parole, `revocationbase_priv` può essere derivato (e utilizzato per firmare per `revocationpubkey`) solo dalla parte che conosce *sia* `revocationbase_priv` che `per_commitment_secret`. Questo piccolo trucco è ciò che rende sicuro il sistema di revoca basato su chiave pubblica utilizzato nel Lightning Network.

SUGGERIMENTO	Il timelock utilizzato nella transazione di impegno con `CHECKSEQUENCEVERIFY` è un *timelock relativo*. Conta i blocchi trascorsi dalla conferma di questo output. Ciò significa che non sarà spendibile fino a quando il blocco `to_self_delay` *dopo* questa transazione di impegno non sarà trasmesso e confermato.

Il secondo output di output (`to_remote`) della transazione di impegno è definito in BOLT #3: Commitment Transaction, `to_remote` Output e nella forma più semplice è un Pay-to-Witness-Public-Key-Hash (P2WPKH) per `<remote_pubkey>`, nel senso che paga semplicemente il proprietario che può firmare per `<remote_pubkey>`.

Ora che abbiamo definito in dettaglio le transazioni di impegno, vediamo come Alice e Bob fanno avanzare lo stato del canale, creano e firmano nuove transazioni di impegno e revocano le vecchie transazioni di impegno.

Avanzamento dello Stato del Canale

Per far avanzare lo stato del canale, Alice e Bob si scambiano due messaggi: `commitment_signed` e `revoke_and_ack`. Il messaggio `commitment_signed` può essere

inviato da entrambi i partner di canale quando dispongono di un aggiornamento allo stato del canale. L'altro partner di canale può quindi rispondere con `revoke_and_ack` per revocare il vecchio impegno e riconoscere il nuovo impegno.

Nel Figura 7-11 vediamo Alice e Bob che si scambiano due coppie di `commitment_signed` e `revoke_and_ack`. Il primo flusso mostra un aggiornamento dello stato avviato da Alice (da sinistra a destra `commitment_signed`), a cui Bob risponde (da destra a sinistra `revoke_and_ack`). Il secondo flusso mostra un aggiornamento dello stato avviato da Bob e a cui Alice ha risposto.

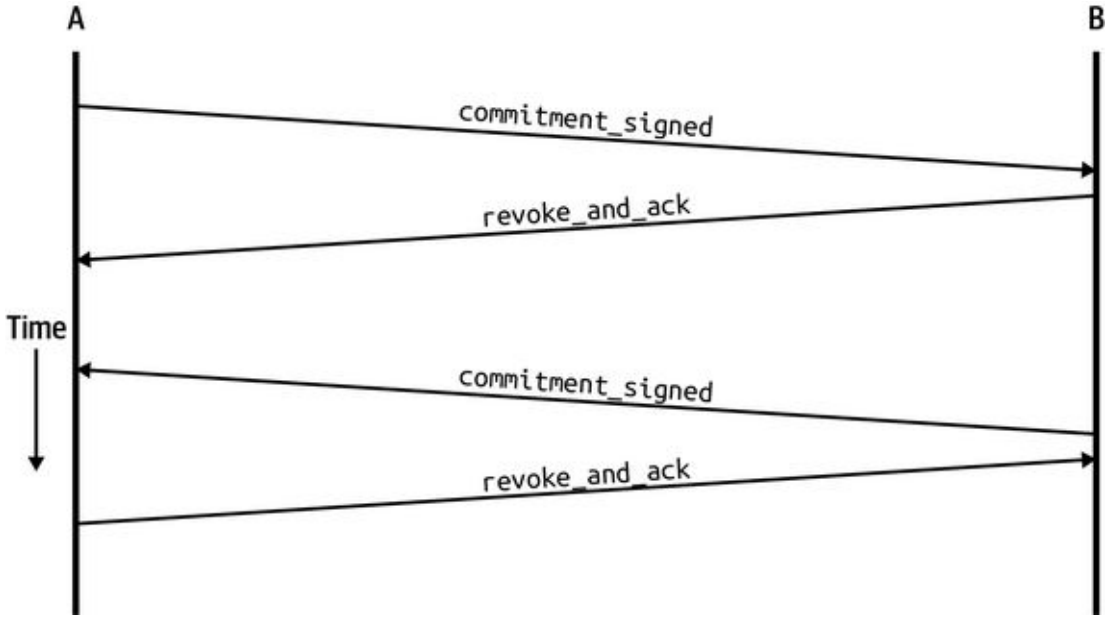

Figura 7-11. Flusso di messaggi di impegno e revoca

Il Messaggio `commitment_signed`

La struttura del messaggio `commitment_signed` è definita in BOLT #2: Peer Protocol, `commitment_signed`, e mostrata di seguito:

```
[channel_id:channel_id]
[signature:signature]
[u16:num_htlcs]
[num_htlcs*signature:htlc_signature]
```

channel_id

L'identificatore del canale

signature

La firma per il nuovo impegno remoto

num_htlcs

Il numero di HTLC aggiornati in questo impegno

htlc_signature

Le firme per gli aggiornamenti

NOTA	L'uso di HTLC per eseguire il commit degli aggiornamenti sarà spiegato in dettaglio nei prossimi capitoli.

Il messaggio `commitment_signed` di Alice fornisce a Bob la firma necessaria (la parte di Alice del 2-di-2) per una nuova transazione di impegno.

Il Messaggio `revoke_and_ack`

Ora che Bob ha una nuova transazione di impegno, può revocare l'impegno precedente dando ad Alice una chiave di revoca e costruire il nuovo impegno con la firma di Alice.

Il messaggio `revoke_and_ack` è definito in BOLT #2: Peer Protocol, `revoke_and_ack`, e mostrato di seguito:

```
[channel_id:channel_id]
[32*byte:per_commitment_secret]
[point:next_per_commitment_point]
```

channel_id

Questo è l'identificatore del canale.

per_commitment_secret

Utilizzato per generare una chiave di revoca per il precedente (vecchio) impegno, revocandolo di fatto.

next_per_commitment_point

Utilizzato per creare un `revocation_pubkey` per il nuovo impegno, in modo che possa essere successivamente revocato.

Revoca e Nuovo Impegno

Diamo un'occhiata più da vicino a questa interazione tra Alice e Bob.

Alice sta dando a Bob i mezzi per creare un nuovo impegno. In cambio, Bob sta revocando il vecchio impegno per assicurare ad Alice che non lo userà. Alice può fidarsi del nuovo impegno solo se ha la chiave di revoca per punire Bob per aver pubblicato il vecchio impegno. Dal punto di vista di Bob, può tranquillamente revocare il vecchio impegno dando ad Alice le chiavi per penalizzarlo, perché ha una firma per un nuovo impegno.

Quando Bob risponde con `revoke_and_ack`, dà ad Alice un `per_commitment_secret`. Questo segreto può essere utilizzato per costruire la chiave di firma della revoca per il vecchio impegno, che consente ad Alice di sequestrare tutti i fondi del canale esercitando una penale.

Non appena Bob avrà rivelato questo segreto ad Alice, non dovrà più trasmettere quel vecchio impegno. Se lo fa, darà ad Alice l'opportunità di penalizzarlo. In sostanza, Bob sta dando ad Alice la possibilità di ritenerlo responsabile per aver trasmesso un vecchio impegno, in quanto ha revocato la sua capacità di utilizzare quel vecchio impegno.

Una volta che Alice ha ricevuto `revoke_and_ack` da Bob, può essere certa che Bob non possa trasmettere il vecchio impegno senza essere penalizzato. Ora ha le chiavi necessarie per creare una transazione di penalità se Bob trasmette un vecchio impegno.

Baro e Penalità in Pratica

In pratica, sia Alice che Bob devono monitorare gli imbrogli. Stanno monitorando la blockchain Bitcoin per qualsiasi transazione di impegno relativa a uno qualsiasi dei canali che stanno operando. Se vedono una transazione di impegno confermata sulla catena, verificheranno se si tratta dell'impegno più recente. Se si tratta di un "vecchio" impegno, devono immediatamente costruire e trasmettere una transazione di penalità. La transazione di penalità spende *entrambi* gli output `to_local` e `to_remote`, chiudendo il canale e inviando entrambi i saldi al partner di canale ingannato.

Per consentire a entrambe le parti di tenere traccia dei numeri di impegno degli impegni di revoca approvati, ogni impegno *codifica* il numero dell'impegno all'interno dei campi del tempo di blocco e della sequenza in una transizione. All'interno del protocollo, questa codifica speciale viene definita *suggerimento di stato*. Supponendo che una parte conosca il numero di impegno corrente, è in grado di utilizzare i suggerimenti di stato per riconoscere facilmente se un impegno trasmesso è stato revocato e, in tal caso, quale numero di impegno è stato violato, poiché quel numero viene utilizzato per cercare facilmente quale segreto di revoca deve essere utilizzato nell'albero segreto di revoca (shachain).

Invece di codificare il suggerimento di stato in bella vista, al suo posto viene utilizzato un suggerimento di stato *offuscato*. Questo offuscamento viene ottenuto eseguendo prima lo XOR del numero di impegno corrente con un insieme di byte casuali generati in modo deterministico utilizzando le chiavi pubbliche di finanziamento di entrambi i lati del canale. Un totale di 6 byte attraverso il tempo di blocco e la sequenza (24 bit del tempo di blocco e 24 bit della sequenza) vengono utilizzati per codificare il suggerimento di stato all'interno della transazione di impegno, quindi sono necessari 6 byte casuali da utilizzare per effettuare lo XORing. Per ottenere questi 6 byte, entrambe le parti ottengono l'hash SHA-256 della chiave di finanziamento dell'iniziatore concatenata alla chiave di finanziamento del risponditore. Prima di codificare l'altezza di impegno corrente, il numero intero viene sottoposto a XOR con questo offuscatore di suggerimenti sullo stato e quindi codificato nei 24 bit inferiori del tempo di blocco e nei 64 bit superiori della sequenza.

Rivediamo il nostro canale tra Alice e Bob e mostriamo un esempio specifico di transazione di penalità. Nella Figura 7-12 vediamo i quattro impegni sul canale di Alice e Bob. Alice ha effettuato tre pagamenti a Bob:

- 70.000 satoshi pagati e impegnati con Bob con il Commitment #1

- 10.000 satoshi pagati e impegnati con Bob con il Commitment #2
- 20.000 satoshi pagati e impegnati con Bob con il Commitment #3

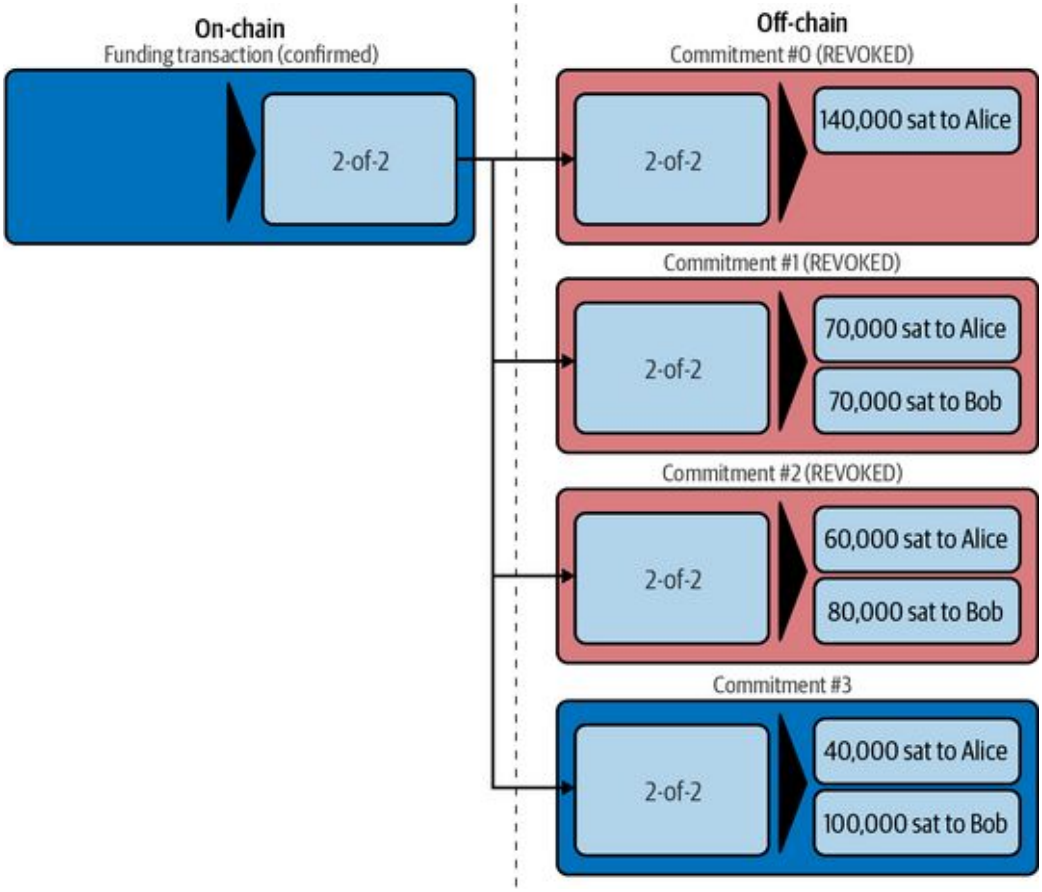

Figura 7-12. Impegni revocati e attuali

Con ogni impegno, Alice ha revocato l'impegno precedente (più vecchio). Lo stato attuale del canale e il corretto equilibrio sono rappresentati dal Commitment #3. Tutti gli impegni precedenti sono stati revocati e Bob ha le chiavi necessarie per emettere transazioni di penalità nei loro confronti, nel caso in cui Alice tenti di trasmetterne uno.

Alice potrebbe avere un incentivo a imbrogliare perché tutte le precedenti transazioni di impegno le darebbero una percentuale maggiore del saldo del canale rispetto a quella a cui ha diritto. Fingiamo ad esempio che Alice abbia provato a trasmettere Commitment #1. Quella transazione di impegno pagherebbe ad Alice 70.000 satoshi e a Bob 70.000 satoshi. Se Alice fosse in grado di trasmettere e spendere il suo output `to_local`, ruberebbe

effettivamente 30.000 satoshi a Bob ripristinando i suoi ultimi due pagamenti verso Bob.

Alice decide di correre un grosso rischio e trasmettere il Commitment #1 revocato per rubare 30.000 satoshi a Bob. Nella Figura 7-13 vediamo il vecchio impegno che Alice trasmette alla blockchain di Bitcoin.

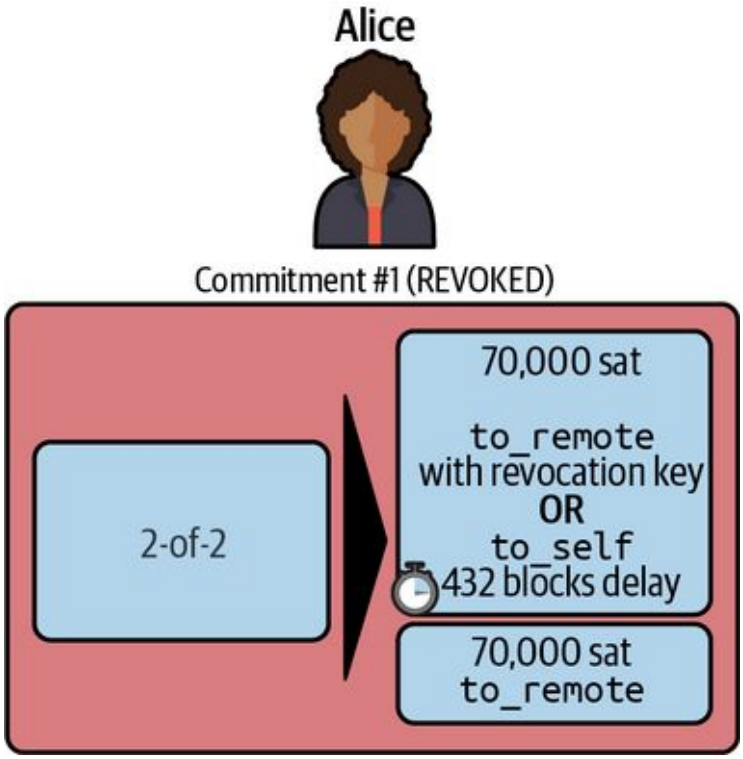

Figura 7-13. Alice bara

Come puoi vedere, il vecchio impegno di Alice ha due output, uno che paga a se stessa 70.000 satoshi (output `to_local`) e uno che paga a Bob 70.000 satoshi. Alice non può ancora spendere il suo output di 70.000 `to_local` perché ha un timelock di 432 blocchi (3 giorni). Ora deve solo sperare che Bob non se ne accorga per tre giorni.

Sfortunatamente per Alice, il nodo di Bob sta monitorando diligentemente la blockchain e vede una vecchia transazione di impegno trasmessa e (alla fine) confermata sulla catena.

Il nodo di Bob trasmetterà immediatamente una transazione di penalità. Poiché questo vecchio impegno è stato revocato da Alice, Bob ha il `per_commitment_secret` che Alice gli ha inviato. Usa quel segreto per costruire una firma per `revocation_pubkey`. Mentre Alice deve attendere 432 blocchi, Bob può spendere immediatamente *entrambi* gli output. Può

spendere l'output `to_remote` con le sue chiavi private perché era destinato a pagarlo comunque. Può anche spendere l'output destinato ad Alice con una firma dalla chiave di revoca. Il suo nodo trasmette la transazione di penalità mostrata nella Figura 7-14.

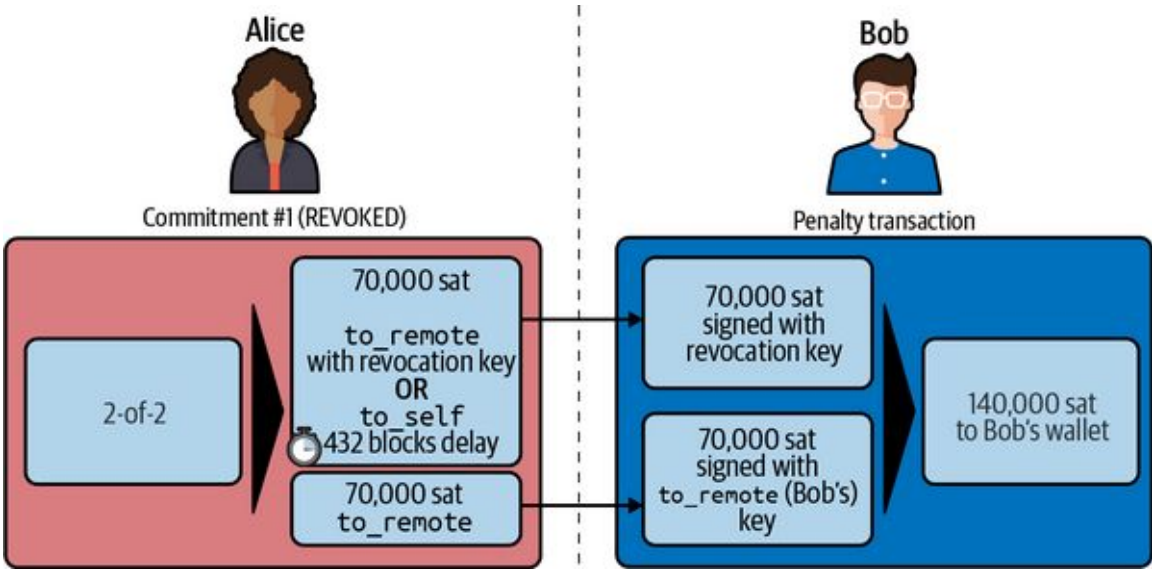

Figura 7-14. Truffa e Penalità

La transazione di penalità di Bob paga 140.000 sats al proprio wallet prendendo l'intera capacità del canale. Alice non solo non è riuscita a imbrogliare, ma ha perso tutto!

La Riserva di Canale: Garanzia di Avere "Skin in The Game"

Potresti aver notato che c'è una situazione speciale che deve essere affrontata. Se Alice potesse continuare a spendere il suo saldo fino a quando non sarà pari a zero, sarebbe in grado di chiudere il canale trasmettendo una vecchia transazione di impegno senza rischiare una penalità: o la transazione di impegno revocata va a buon fine dopo il ritardo, oppure l'imbroglione viene scoperto ma non c'è nessuna conseguenza perché la penalità è pari a zero. Dal punto di vista della teoria dei giochi, è letteralmente gratis tentare di imbrogliare in questa situazione. Questo è il motivo per cui è in gioco la riserva del canale, quindi un potenziale imbroglione corre sempre il (minimo) rischio di una penalità.

Chiusura del Canale (Chiusura Cooperativa)

Finora abbiamo considerato le transazioni di impegno come un possibile modo per chiudere un canale unilateralmente. Questo tipo di chiusura del canale non è l'ideale perché impone un timelock al partner di canale che lo utilizza.

Un modo migliore per chiudere un canale è una chiusura cooperativa. In una chiusura cooperativa, i due partner di canale negoziano una transazione di impegno finale chiamata transazione di chiusura, che paga immediatamente a ciascuna parte il proprio saldo nel portafoglio di destinazione di sua scelta. Quindi, il partner che ha avviato il flusso di chiusura del canale trasmetterà la transazione di chiusura.

Il flusso di messaggi di chiusura è definito in BOLT #2: Peer Protocol, Channel Close, ed è mostrato nella Figura 7-15.

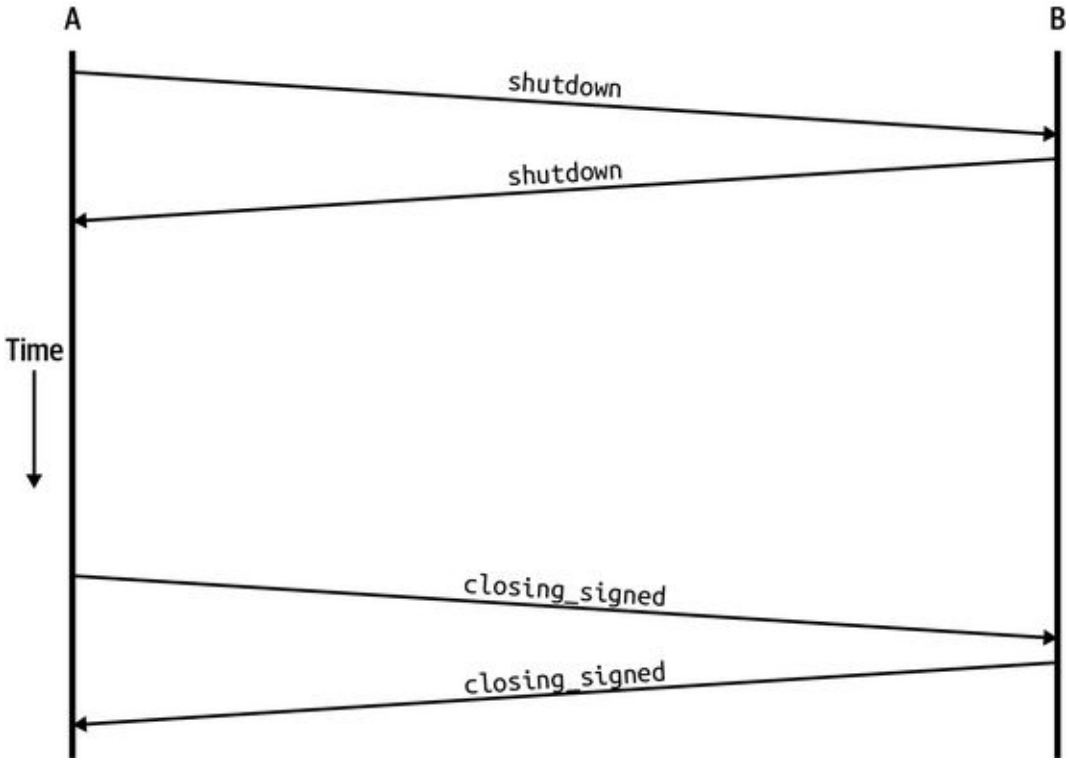

Figura 7-15. Il flusso di messaggi di chiusura del canale

Il Messaggio di Arresto

La chiusura del canale inizia con l'invio del messaggio di `shutdown` da parte di uno dei due

partner di canale. Il contenuto del messaggio è mostrato di seguito:

```
[channel_id:channel_id]
[u16:len]
[len*byte:scriptpubkey]
```

`channel_id`

L'identificatore del canale che vogliamo chiudere

`len`

La lunghezza dello script del portafoglio di destinazione cui il partner di canale vuole ricevere il proprio saldo

`scriptpubkey`

Uno script Bitcoin del portafoglio di destinazione, in uno dei formati di indirizzo Bitcoin "standard" (P2PKH, P2SH, P2WPKH, P2WSH, ecc.)

Supponiamo che Alice invii il messaggio di `shutdown` a Bob per chiudere il canale. Alice specificherà uno script Bitcoin che corrisponde all'indirizzo Bitcoin del suo wallet. Sta dicendo a Bob: facciamo una transazione di chiusura che paghi il mio saldo a questo wallet.

Bob risponderà con il proprio messaggio di `shutdown` indicando che accetta di chiudere in modo cooperativo il canale. Il suo messaggio di `shutdown` include lo script per l'indirizzo del suo wallet.

Ora sia Alice che Bob hanno l'indirizzo del wallet preferito l'uno dell'altro e possono costruire transazioni di chiusura identiche per saldare il saldo del canale.

Il Messaggio `closing_signed`

Supponendo che il canale non abbia impegni o aggiornamenti in sospeso e che i partner di canale si siano scambiati i messaggi di chiusura mostrati nella sezione precedente, ora possono terminare questa chiusura cooperativa.

Il *finanziatore* del canale (Alice nel nostro esempio) inizia inviando un messaggio `closing_signed` a Bob. Questo messaggio propone una commissione di transazione per la transazione on-chain e la firma di Alice (il multisig 2-di-2) per la transazione di chiusura. Il messaggio `closing_signed` è mostrato di seguito:

```
[channel_id:channel_id]
[u64:fee_satoshis]
[signature:signature]
```

`channel_id`

: L'identificatore del canale

`fee_satoshis`

: La commissione di transazione on-chain proposta, in satoshi

`signature`

: La firma del mittente per la transazione di chiusura

Quando Bob lo riceve, può rispondere con un proprio messaggio `closing_signed`. Se è d'accordo con il compenso, restituisce semplicemente lo stesso compenso proposto e la propria firma. Se non è d'accordo, deve proporre una `fee_satoshis` differente.

Questa negoziazione può continuare con messaggi di `closing_signed` avanti e indietro finché i due partner di canale non concordano una tariffa.

Una volta che Alice riceve un messaggio `closing_signed` con la stessa tariffa proposta nel suo ultimo messaggio, la negoziazione è completa. Alice firma e trasmette la transazione di chiusura e il canale viene chiuso.

La Transazione di Chiusura Cooperativa

La transazione di chiusura cooperativa è simile all'ultima transazione di impegno concordata da Alice e Bob. Tuttavia, a differenza dell'ultima transazione di impegno, non ha timelock o

chiavi di revoca e penalità negli output. Poiché entrambe le parti cooperano per produrre questa transazione e non assumeranno ulteriori impegni, non sono necessari gli elementi asimmetrici, ritardati e revocabili in questa transazione.

In genere gli indirizzi utilizzati in questa transazione di chiusura cooperativa vengono generati nuovi per ogni canale chiuso. Tuttavia, è anche possibile per entrambe le parti *bloccare* un indirizzo di "consegna" da utilizzare per inviare i propri fondi regolati in modo cooperativo. All'interno dello spazio dei nomi TLV di entrambi i messaggi `open_channel` e `accept_channel`, entrambe le parti sono libere di specificare uno "script di spegnimento anticipato". In genere, questo indirizzo deriva da chiavi che risiedono in cold wallet. Questa pratica serve ad aumentare la sicurezza dei canali: se un partner di canale viene in qualche modo violato, l'hacker non è in grado di chiudere cooperativamente il canale utilizzando un indirizzo che controlla. Invece, il partner di canale onesto e senza compromessi si rifiuterà di cooperare alla chiusura di un canale se non viene utilizzato l'indirizzo di chiusura iniziale specificato. Questa funzione crea effettivamente un "circuito chiuso", limitando il flusso di fondi in uscita da un determinato canale.

Alice trasmette una transazione, come mostrato in Figura 7-16, per chiudere il canale.

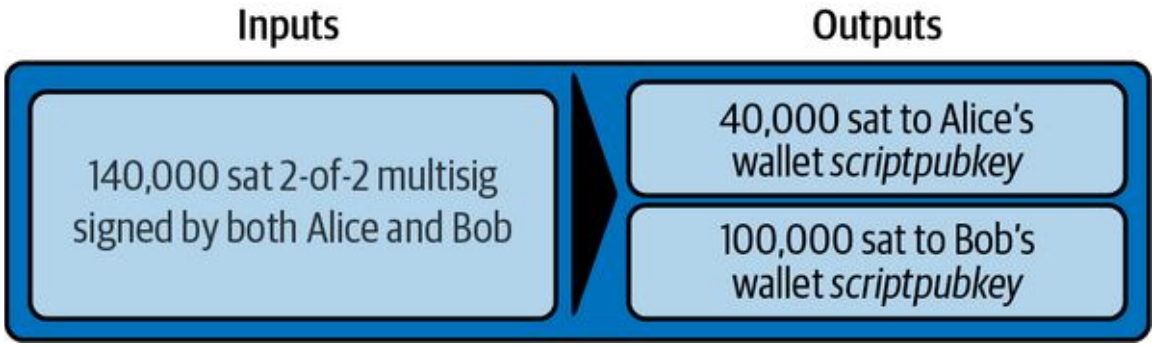

Figura 7-16. La transazione di chiusura cooperativa

Non appena questa transazione di chiusura viene confermata sulla blockchain di Bitcoin, il canale viene chiuso. Ora, Alice e Bob possono spendere le loro uscite a loro piacimento.

Conclusione

In questa sezione abbiamo esaminato i canali di pagamento in modo molto più dettagliato.

Abbiamo esaminato tre flussi di messaggi utilizzati da Alice e Bob per negoziare finanziamenti, impegni e chiusura del canale. Abbiamo anche mostrato la struttura delle transazioni di finanziamento, impegno e chiusura e abbiamo esaminato i meccanismi di revoca e sanzione.

Come vedremo nei prossimi capitoli, gli HTLC vengono utilizzati anche per i pagamenti locali tra partner di canale. Non sono necessari, ma il protocollo è molto più semplice se i pagamenti locali (un canale) e instradati (molti canali) vengono effettuati nello stesso modo.

In un singolo canale di pagamento, il numero di pagamenti al secondo è limitato solo dalla capacità della rete tra Alice e Bob. Finché i partner di canale sono in grado di inviare avanti e indietro alcuni byte di dati per concordare un nuovo saldo del canale, hanno effettivamente effettuato un pagamento. Questo è il motivo per cui possiamo ottenere un throughput di pagamenti molto maggiore su Lightning Network (off-chain) rispetto al throughput delle transazioni che può essere gestito dalla blockchain di Bitcoin (on-chain).

Nei prossimi capitoli discuteremo l'instradamento, gli HTLC e il loro utilizzo nelle operazioni di canale.

Capitolo 8. Routing su una Rete di Canali di Pagamento

In questo capitolo spiegheremo come i canali possono essere collegati per formare una rete di canali di pagamento tramite un processo chiamato *routing*. In particolare, esamineremo la prima parte del livello di routing, il protocollo "Contratti multihop atomici e trustless". Ciò è evidenziato da uno schema nella suite di protocolli, mostrato nella Figura 8-1.

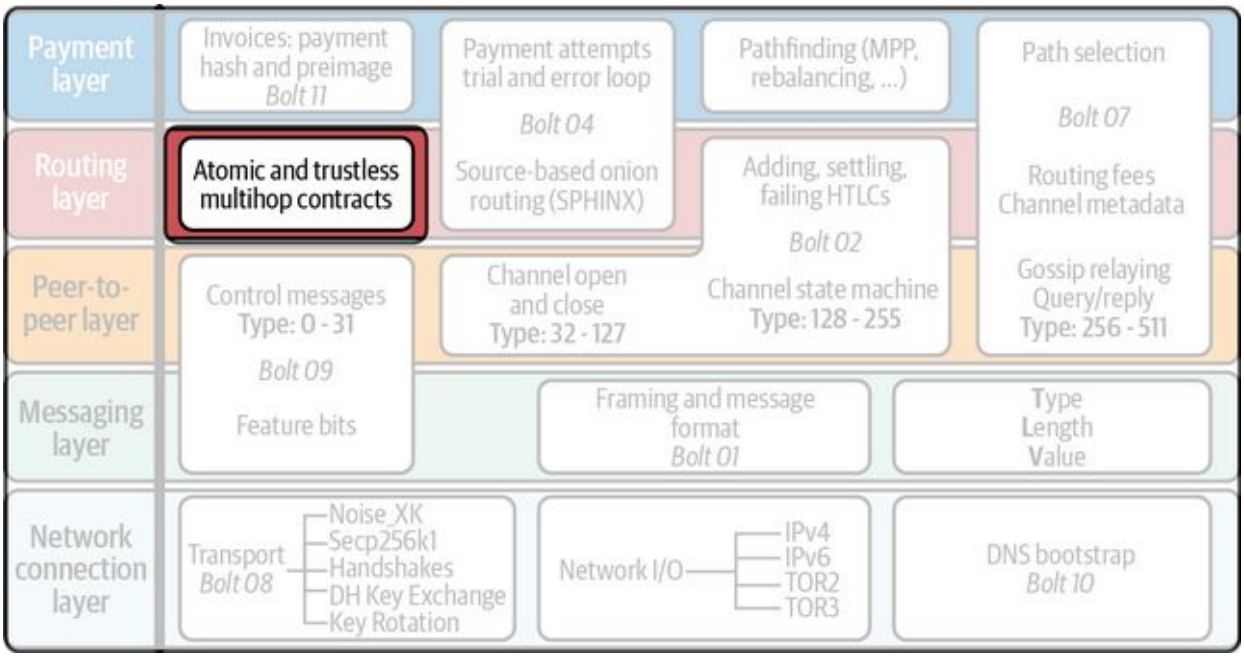

Figura 8-1. Instradamento atomico dei pagamenti nella suite di protocolli Lightning

Instradamento di un Pagamento

In questa sezione esamineremo il routing dal punto di vista di Dina, una player che riceve donazioni dai suoi fan mentre trasmette in streaming le sue sessioni di gioco.

L'innovazione dei canali di pagamento instradati consente a Dina di ricevere donazioni senza mantenere un canale separato con ognuno dei suoi fan. Finché esiste un percorso di canali ben finanziati da uno specifico spettatore a Dina, lei potrà ricevere il pagamento.

In Figura 8-2 vediamo un possibile schema di rete creato da vari canali di pagamento tra i nodi Lightning. Tutti in questo diagramma possono inviare a Dina un pagamento costruendo un percorso. Immagina che il Fan n.4 voglia inviare a Dina un pagamento. Vedi il percorso che potrebbe permettere che ciò accada? Il Fan n.4 potrebbe indirizzare un pagamento a Dina tramite il Fan n.3, Bob e Chan. Allo stesso modo, Alice potrebbe instradare un pagamento a Dina tramite Bob e Chan.

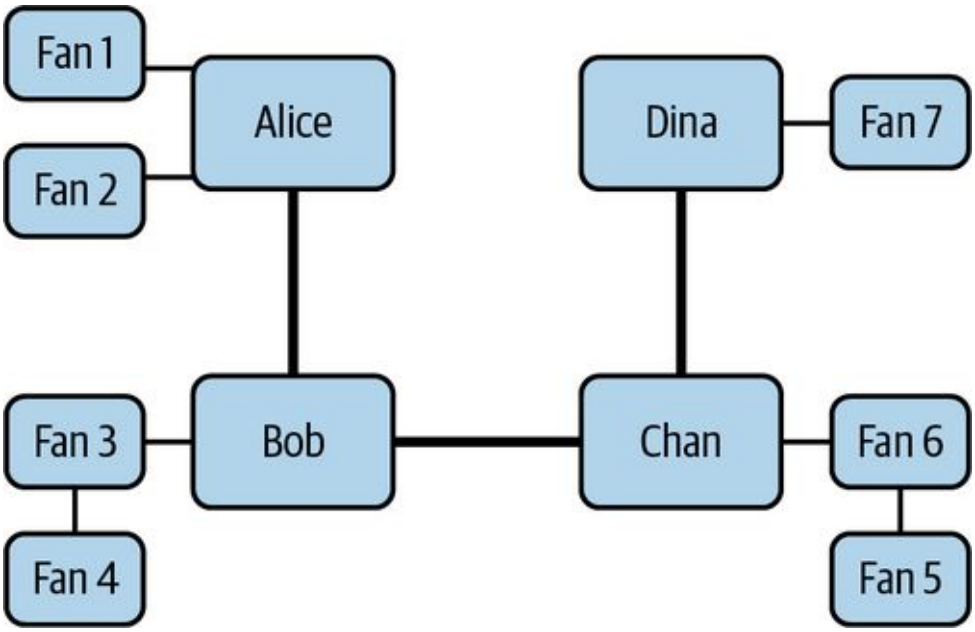

Figura 8-2. I fan si sono collegati (in)direttamente a Dina su Lightning Network

I nodi lungo il percorso dal fan a Dina sono intermediari chiamati *nodi di instradamento* (*routing nodes*) nel contesto dell'instradamento di un pagamento. Non c'è alcuna differenza funzionale tra i nodi di instradamento e i nodi gestiti dai fan di Dina. Qualsiasi nodo Lightning è in grado di instradare i pagamenti attraverso i propri canali di pagamento.

È importante sottolineare che i nodi di instradamento non sono in grado di rubare i fondi mentre instradano un pagamento da un fan a Dina. Inoltre, i nodi di instradamento non possono perdere denaro durante la partecipazione al processo di instradamento. I nodi di instradamento possono addebitare una commissione per agire come intermediari, sebbene non debbano farlo e possano scegliere di instradare i pagamenti gratuitamente.

Un altro dettaglio importante è che, a causa dell'uso dell'onion routing, i nodi intermedi sono esplicitamente consapevoli solo del nodo che li precede e del nodo che li segue nel percorso.

Non sapranno necessariamente chi è il mittente e il destinatario del pagamento. Ciò consente ai fan di utilizzare i nodi intermediari per pagare Dina, senza far trapelare informazioni private e senza rischiare furti.

Questo processo di collegamento di una serie di canali di pagamento con sicurezza end-to-end e la struttura di incentivi per i nodi per *l'inoltro* dei pagamenti è una delle principali innovazioni di Lightning Network.

In questo capitolo, ci addentreremo nel meccanismo di instradamento su Lightning Network, descrivendo in dettaglio il modo preciso in cui i pagamenti fluiscono attraverso la rete. Innanzitutto, chiariremo il concetto di routing e lo confronteremo con quello di pathfinding, perché questi sono spesso confusi e usati in modo intercambiabile. Successivamente, costruiremo il protocollo di equità: un protocollo atomico, trustless, multihop utilizzato per instradare i pagamenti. Per dimostrare come funziona questo protocollo di equità, utilizzeremo un equivalente fisico del trasferimento di monete d'oro tra quattro persone. Infine, esamineremo l'implementazione del protocollo atomico, trustless e multihop attualmente utilizzato su LN, che è chiamato contratto hash time-locked (HTLC).

Routing contro Pathfinding

È importante notare che separiamo il concetto di *routing* dal concetto di *pathfinding*. Questi due concetti sono spesso confusi e il termine routing viene spesso utilizzato per descrivere entrambi i concetti. Eliminiamo l'ambiguità prima di procedere...

Il pathfinding, trattato in dettaglio nel Capitolo 12, è il processo di ricerca e scelta di un percorso contiguo costituito da canali di pagamento che collega il mittente A al destinatario B. Il mittente di un pagamento esegue il pathfinding esaminando il *grafo dei canali* che ha assemblato da annunci di canale condivisi (gossip p2p) da altri nodi.

Il routing si riferisce alla serie di interazioni attraverso la rete che tentano di inoltrare un pagamento da un punto A a un altro punto B, attraverso il percorso precedentemente selezionato dal pathfinding. Il routing è il processo attivo di invio di un pagamento su un percorso, che prevede la cooperazione di tutti i nodi intermedi lungo tale percorso.

Una regola empirica importante è che è possibile che esista un *percorso* tra Alice e Bob (o più di uno), tuttavia potrebbe non esserci un *percorso disponibile* su cui inviare il pagamento. Un esempio è lo scenario in cui tutti i nodi che collegano Alice e Bob sono

offline. In tale esempio, si può esaminare il grafi dei canali e collegare una serie di canali di pagamento da Alice a Bob, quindi esiste un *percorso*. Tuttavia, poiché i nodi intermedi sono offline, il pagamento non può essere inviato e quindi non esiste alcun *percorso disponibile*.

Creazione di una Rete di Canali di Pagamento

Prima di addentrarci nel concetto di pagamento atomico multihop senza fiducia, analizziamo un esempio. Pensiamo ad Alice che, nei capitoli precedenti, ha acquistato un caffè da Bob con il quale ha aperto un canale. Ora Alice sta guardando una diretta streaming di Dina, la giocatrice, e vuole inviare a Dina una donazione di 50.000 satoshi tramite Lightning Network. Alice non ha un canale diretto con Dina. Cosa può fare Alice in questo caso?

Alice potrebbe aprire un canale diretto con Dina; tuttavia, ciò richiederebbe liquidità e commissioni on-chain che potrebbero essere superiori al valore della donazione stessa. Invece, Alice può utilizzare i suoi canali aperti esistenti per inviare una donazione a Dina *senza* la necessità di aprire un canale direttamente con lei. Ciò è possibile, purché esista un percorso di canali da Alice a Dina con capacità sufficiente per instradare la donazione.

Come puoi vedere nella Figura 8-3, Alice ha un canale aperto con Bob, il proprietario del bar. Bob, a sua volta, ha un canale aperto con lo sviluppatore di software Chan che lo aiuta con il sistema del punto vendita che usa nel suo bar. Chan è anche il proprietario di una grande società di software che sviluppa il gioco di Dina, e hanno già un canale aperto che Dina usa per pagare la licenza del gioco e gli oggetti virtuali di gioco.

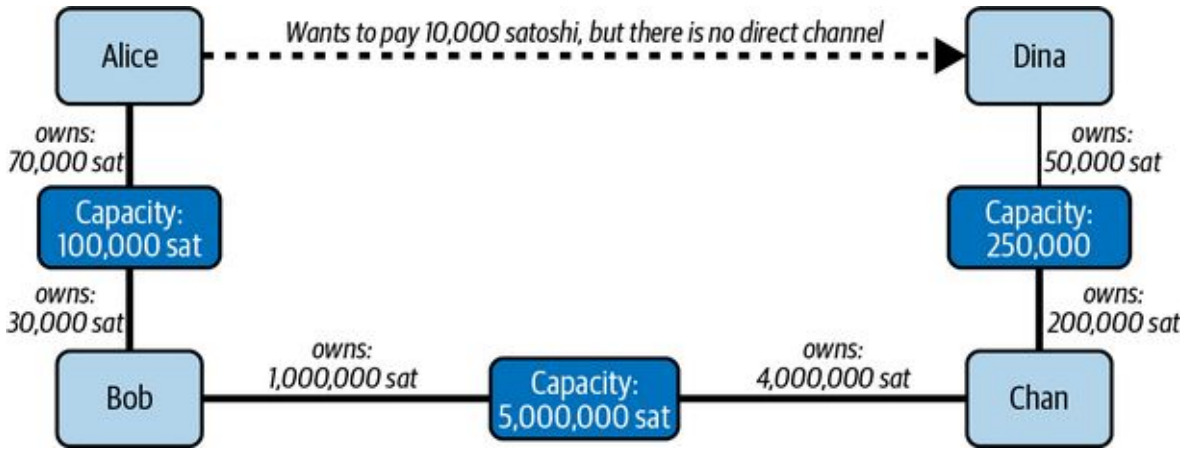

Figura 8-3. Una rete di canali di pagamento tra Alice e Dina

È possibile *tracciare un percorso* da Alice a Dina, che utilizza Bob e Chan come nodi di instradamento intermedi. Alice può quindi *creare un percorso* ed usarlo per inviare una donazione di qualche migliaio di satoshi a Dina, con il pagamento *inoltrato* da Bob e Chan. In sostanza, Alice pagherà Bob, che pagherà Chan, che pagherà Dina. Non è richiesto alcun canale diretto da Alice a Dina.

La sfida principale è farlo in modo da impedire a Bob e Chan di rubare i soldi che Alice vuole donare a Dina.

Un Esempio Fisico di "Routing"

Per capire in che modo LN protegge durante l'instradamento, possiamo confrontarlo con un esempio di instradamento di pagamenti fisici con monete d'oro nel mondo reale.

Supponiamo che Alice voglia dare 10 monete d'oro a Dina, ma non abbia accesso diretto a Dina. Tuttavia, Alice conosce Bob, che conosce Chan, che conosce Dina, quindi decide di chiedere aiuto a Bob e Chan. Questo è mostrato in Figura 8-4.

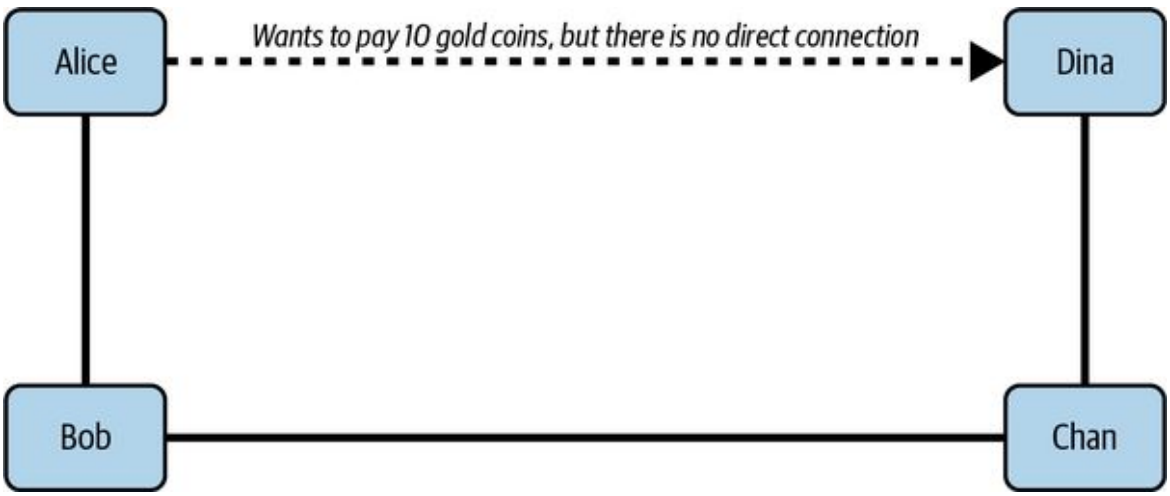

Figura 8-4. Alice vuole pagare a Dina 10 monete d'oro

Alice può pagare Bob per pagare Chan per pagare Dina, ma come fa ad assicurarsi che Bob o Chan non scappino con le monete dopo averle ricevute? Nel mondo fisico, i contratti potrebbero essere utilizzati per eseguire in sicurezza una serie di pagamenti.

Alice potrebbe negoziare un contratto con Bob, che recita:

"Io, Alice, darò a te, Bob, 10 monete d'oro se le consegni a Chan."

Sebbene questo contratto sia decente, nel mondo reale Alice corre il rischio che Bob possa violare il contratto e sperare di non essere scoperto. Anche se Bob viene catturato e perseguito, Alice corre il rischio che possa andare in bancarotta e non essere in grado di restituire le sue 10 monete d'oro. Supponendo che questi problemi siano magicamente risolti, non è ancora chiaro come sfruttare un tale contratto per ottenere il risultato desiderato: far consegnare le monete a Dina.

Miglioriamo il nostro contratto per incorporare queste considerazioni:

"Io, Alice, ti rimborserò, Bob, di 10 monete d'oro se puoi dimostrarmi (ad esempio, tramite una ricevuta) che hai consegnato 10 monete d'oro a Chan."

Potresti chiederti perché Bob dovrebbe firmare un contratto del genere. Deve pagare Chan ma alla fine non ottiene nulla dallo scambio e corre il rischio che Alice non lo rimborsi. Bob potrebbe offrire a Chan un contratto simile per pagare Dina, ma allo stesso modo anche Chan non avrebbe motivo di accettarlo.

Anche mettendo da parte il rischio, Bob e Chan devono *già aver*e 10 monete d'oro da inviare; in caso contrario, non sarebbero in grado di partecipare al contratto.

Pertanto Bob e Chan affrontano sia il rischio che il costo per aver accettato questo contratto, e dovrebbero essere ricompensati per accettarlo.

Alice può quindi renderlo attraente sia per Bob che per Chan offrendo loro una commissione di una moneta d'oro ciascuno, se trasmettono il suo pagamento a Dina.

Il contratto quindi recita:

"Io, Alice, ti rimborserò, Bob, con 12 monete d'oro se puoi dimostrarmi (ad esempio, tramite una ricevuta) che hai consegnato 11 monete d'oro a Chan."

Alice ora promette a Bob 12 monete d'oro. Sono 10 da consegnare a Dina e 2 di commissione. Promette 12 a Bob se può dimostrare di aver inoltrato 11 a Chan. La differenza di una moneta d'oro è la commissione che Bob guadagnerà per aiutare la buona riuscita di questo particolare pagamento. Nella Figura 8-5 vediamo come questa disposizione farebbe ottenere 10 monete d'oro a Dina tramite Bob e Chan.

Figura 8-5. Alice paga Bob, Bob paga Chan, Chan paga Dina

Poiché c'è ancora il problema della fiducia e il rischio che Alice o Bob non onorino il contratto, tutte le parti decidono di utilizzare un servizio di deposito a garanzia. All'inizio dello scambio, Alice potrebbe "bloccare" queste 12 monete d'oro in garanzia che verranno pagate a Bob solo una volta che avrà dimostrato di aver pagato 11 monete d'oro a Chan.

Questo servizio di deposito a garanzia (escrow) è idealizzato in modo tale da non introdurre altri rischi (es. rischio di controparte). Più avanti vedremo come sostituire l'escrow con uno smart contract di Bitcoin. Supponiamo per ora che tutti si fidino di questo servizio di deposito a garanzia.

Su Lightning Network, la ricevuta (prova di pagamento) potrebbe assumere la forma di un segreto che solo Dina conosce. In pratica, questo segreto sarebbe un numero casuale abbastanza grande da impedire ad altri di indovinarlo (tipicamente un numero *molto*, *molto* grande, codificato usando 256 bit!).

Dina genera questo valore segreto R da un generatore di numeri casuali.

Il segreto potrebbe quindi essere impegnato nel contratto includendo l'hash SHA-256 del segreto nel contratto stesso, come segue:

H = SHA-256(R)

Chiamiamo questo hash del segreto del pagamento *hash del pagamento* (*payment hash*). Il segreto che "sblocca" il pagamento è chiamato *segreto di pagamento* (*payment secret*).

Per ora, manteniamo le cose semplici e assumiamo che il segreto di Dina sia semplicemente la riga di testo: `segreto di Dina`. Questo messaggio segreto è chiamato *segreto di pagamento* o *preimmagine di pagamento* (*payment secret* / *payment preimage*).

Per "impegnarsi" in questo segreto, Dina calcola l'hash SHA-256, che una volta codificato in esadecimale, può essere visualizzato come segue:

```
0575965b3b44be51e8057d551c4016d83cb1fba9ea8d6e986447ba33fe69f6b3
```

Per facilitare il pagamento di Alice, Dina creerà il segreto di pagamento e l'hash di pagamento e invierà l'hash di pagamento ad Alice. Nella Figura 8-6 vediamo che Dina invia l'hash del pagamento ad Alice tramite un canale esterno (linea tratteggiata), come un'e-mail o un messaggio di testo.

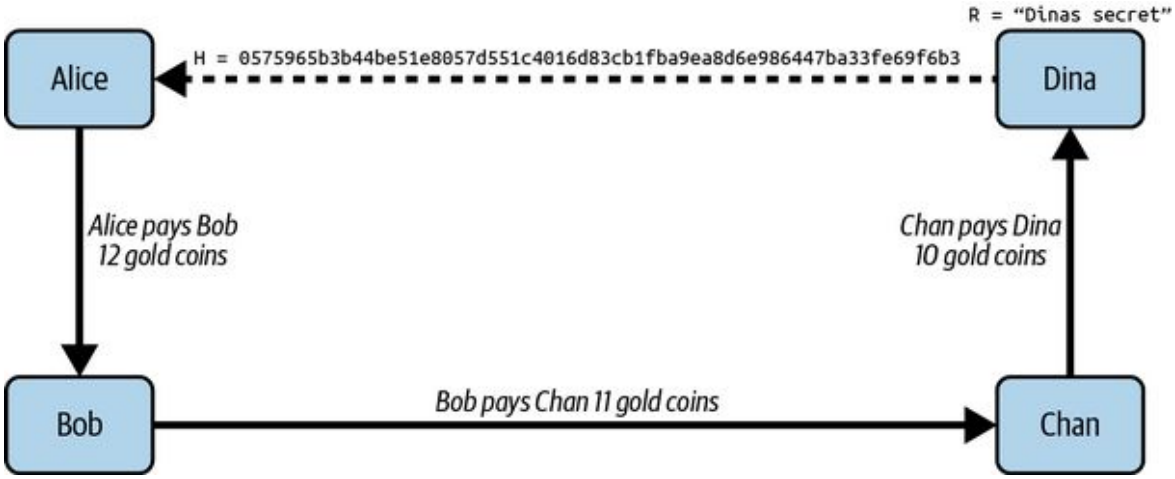

Figura 8-6. Dina invia il segreto con hash ad Alice

Alice non conosce il segreto, ma può riscrivere il suo contratto per utilizzare l'hash del segreto come prova di pagamento:

Io, Alice, ti rimborserò, Bob, con 12 monete d'oro se puoi mostrarmi un messaggio valido che corrisponde all'hash `057596`... Puoi acquisire questo messaggio stabilendo un contratto simile con Chan che deve stipulare un contratto simile con Dina. Per assicurarti che verrai rimborsato, fornirò le 12 monete d'oro a un deposito a garanzia di fiducia prima che tu stabilisca il tuo prossimo contratto.

Questo nuovo contratto ora protegge Alice dal mancato inoltro di Bob a Chan, protegge Bob dal mancato rimborso da parte di Alice e garantisce che ci sarà la prova che Dina è stata

infine pagata tramite l'hash del segreto di Dina.

Dopo che Bob e Alice hanno accettato il contratto e Bob ha ricevuto il messaggio dell'impegno che Alice ha depositato le 12 monete d'oro, Bob può ora negoziare un contratto simile con Chan.

Nota che poiché Bob sta prendendo una commissione di servizio di 1 moneta, inoltrerà solo 11 monete d'oro a Chan una volta che Chan mostrerà la prova di aver pagato Dina. Allo stesso modo, anche Chan richiederà una commissione e si aspetterà di ricevere 11 monete d'oro una volta dimostrato di aver pagato a Dina le 10 monete d'oro promesse.

Il contratto di Bob con Chan reciterà:

Io, Bob, ti rimborserò, Chan, con 11 monete d'oro se puoi mostrarmi un messaggio valido che corrisponde all'hash `057596`... Puoi acquisire questo messaggio stipulando un contratto simile con Dina. Per assicurarti che verrai rimborsato, fornirò le 11 monete d'oro a un deposito a garanzia di fiducia prima che tu stabilisca il tuo prossimo contratto.

Una volta che Chan riceve il messaggio dell'impegno che Bob ha depositato le 11 monete d'oro, Chan stabilisce un contratto simile con Dina:

Io, Chan, ti rimborserò, Dina, con 10 monete d'oro se puoi mostrarmi un messaggio valido che corrisponde all'hash `057596`... Per assicurarti che verrai rimborsata dopo aver rivelato il segreto, fornirò le 10 monete d'oro a un deposito a garanzia di fiducia.

Adesso è tutto a posto. Alice ha un contratto con Bob e ha messo in garanzia 12 monete d'oro. Bob ha un contratto con Chan e ha messo in garanzia 11 monete d'oro. Chan ha un contratto con Dina e ha depositato 10 monete d'oro in garanzia. Tocca ora a Dina svelare il segreto, che è l'anticipazione dell'hash che ha stabilito come prova di pagamento.

Dina ora invia il `segreto di Dina` a Chan.

Chan controlla che il `segreto di Dina` corrisponda all'hash `057596`... Chan ora ha la prova del pagamento e quindi ordina al servizio di deposito a garanzia di rilasciare le 10 monete d'oro a Dina.

Chan ora fornisce il segreto a Bob. Bob lo controlla e ordina al servizio di deposito a garanzia di rilasciare le 11 monete d'oro a Chan.

Bob ora fornisce il segreto ad Alice. Alice lo controlla e ordina al servizio di deposito a garanzia di rilasciare 12 monete d'oro a Bob.

Tutti i contratti sono ora risolti. Alice ha pagato un totale di 12 monete d'oro, 1 delle quali è stata ricevuta da Bob, 1 da Chan e 10 da Dina. Con una catena di contratti come questa in atto, Bob e Chan non potevano scappare con i soldi perché li avevano prima depositati in un deposito a garanzia.

Tuttavia, rimane ancora un problema. Se Dina si rifiutasse di rilasciare la sua preimmagine segreta, allora Chan, Bob e Alice avrebbero tutte le loro monete bloccate in garanzia ma non verrebbero rimborsate. Allo stesso modo, se qualcun altro lungo la catena non fosse riuscito a trasmettere il segreto, sarebbe successa la stessa cosa. Quindi, mentre nessuno può rubare denaro ad Alice, tutti vedrebbero comunque i propri soldi bloccati in garanzia in modo permanente.

Fortunatamente, questo può essere risolto aggiungendo una scadenza al contratto.

Potremmo modificare il contratto in modo che se non viene rispettato entro una certa scadenza, il contratto scade e il servizio di deposito a garanzia restituisce il denaro alla persona che ha effettuato il deposito originale. Chiamiamo questa scadenza un *timelock*.

Il deposito è bloccato con il servizio di deposito a garanzia per un certo periodo di tempo e alla fine viene rilasciato anche se non è stata fornita alcuna prova di pagamento.

Per tenere conto di ciò, il contratto tra Alice e Bob viene nuovamente modificato con una nuova clausola:

Bob ha 24 ore per mostrare il segreto dopo la firma del contratto. Se Bob non fornisce il segreto entro questo tempo, il deposito di Alice verrà rimborsato dal servizio di deposito a garanzia e il contratto perderà validità.

Bob, ovviamente, ora deve assicurarsi di ricevere la prova del pagamento entro 24 ore. Anche se paga Chan con successo, se non riceve la prova del pagamento entro 24 ore, non verrà rimborsato. Per non rischiare, Bob deve dare a Chan una scadenza ancora più breve.

A sua volta, Bob modificherà il suo contratto con Chan come segue:

Chan ha 22 ore per mostrare il segreto dopo la firma del contratto. Se non fornisce il segreto entro questo tempo, il deposito di Bob verrà rimborsato dal servizio di deposito a garanzia e il contratto non sarà più valido.

Come avrai intuito, Chan modificherà anche il suo contratto con Dina:

Dina ha 20 ore per mostrare il segreto dopo la firma del contratto. Se non fornisce il segreto entro questo tempo, il deposito di Chan sarà rimborsato dal servizio di deposito a garanzia e

il contratto non sarà più valido.

Con una tale catena di contratti possiamo garantire che, dopo 24 ore, il pagamento andrà con successo da Alice a Bob a Chan a Dina, o fallirà e tutti saranno rimborsati. O il contratto fallisce o ha successo, non c'è via di mezzo.

Nel contesto di Lightning Network, chiamiamo questa proprietà "tutto o niente", *atomicità*.

Finché l'impegno è affidabile e svolge fedelmente il proprio dovere, a nessuna parte verranno rubate le monete durante il processo.

Il prerequisito affinché questo *percorso* funzioni è che tutte le parti del percorso abbiano denaro sufficiente per soddisfare la serie di depositi richiesta.

Anche se questo sembra un dettaglio minore, vedremo più avanti in questo capitolo che questo requisito è in realtà uno dei problemi più difficili per i nodi LN. Diventa progressivamente più difficile con l'aumentare della dimensione del pagamento. Inoltre, le parti non possono utilizzare il proprio denaro mentre è bloccato in garanzia.

Pertanto, gli utenti che inoltrano i pagamenti affrontano un costo opportunità per bloccare il denaro, che viene infine rimborsato attraverso le commissioni di instradamento, come abbiamo visto nell'esempio precedente.

Ora che abbiamo visto un esempio di instradamento di pagamento fisico, vedremo come questo può essere implementato sulla blockchain di Bitcoin, senza bisogno di deposito a garanzia di terze parti. Per fare ciò, configureremo i contratti tra i partecipanti utilizzando Bitcoin Script. Sostituiamo l'impegno di terze parti con *smart contract* che implementano un protocollo di equità. Analizziamo questo concetto e implementiamolo!

Protocollo di Equità

Come abbiamo visto nel primo capitolo di questo libro, l'innovazione di Bitcoin è la capacità di utilizzare primitive crittografiche per implementare un protocollo di equità che sostituisce la fiducia in terze parti (intermediari) con un protocollo fidato.

Nel nostro esempio di moneta d'oro, avevamo bisogno di un servizio di deposito a garanzia per impedire a una delle parti di venir meno ai propri obblighi. L'innovazione dei protocolli di equità crittografica ci consente di sostituire il servizio di escrow con un protocollo.

Le proprietà del protocollo di equità che vogliamo creare sono:

Operazione senza fiducia

I partecipanti a un pagamento instradato non devono fidarsi l'uno dell'altro, o di qualsiasi intermediario o terza parte. Invece, si fidano del protocollo per proteggersi dagli imbrogli.

Atomicità

O il pagamento viene eseguito completamente o fallisce e tutti vengono rimborsati. Non è possibile che un intermediario raccolga un pagamento instradato e non lo inoltri all'hop successivo. Pertanto, gli intermediari non possono imbrogliare o rubare.

.

Multihop

La sicurezza del sistema si estende end-to-end per i pagamenti instradati attraverso più canali di pagamento, proprio come avviene per un pagamento tra le due estremità di un singolo canale di pagamento.

Una proprietà aggiuntiva facoltativa è la possibilità di suddividere i pagamenti in più parti mantenendo l'atomicità per l'intero pagamento. Questi sono chiamati *multipart payments* (MPP – pagamenti multiparte) e sono descritti dettagliatamente in un prossimo capitolo.

Implementazione di Atomic Trustless Multihop Payments

Bitcoin Script è abbastanza flessibile e ci sono dozzine di modi per implementare un protocollo di equità che ha le proprietà di atomicità, funzionamento senza fiducia e sicurezza multihop. La scelta di un'implementazione specifica dipende da determinati compromessi tra privacy, efficienza e complessità.

Il protocollo di equità per l'instradamento utilizzato oggi su Lightning Network è chiamato contratto hash time-locked (HTLC). Gli HTLC usano l'hash di una preimmagine come segreto che sblocca un pagamento, come abbiamo visto nell'esempio della moneta d'oro in questo capitolo. Il destinatario di un pagamento genera un numero segreto casuale e ne calcola l'hash. L'hash diventa la condizione di pagamento e, una volta svelato il segreto, tutti i partecipanti possono riscattare i pagamenti in entrata. Gli HTLC offrono atomicità,

funzionamento senza fiducia e sicurezza multihop.

Un altro meccanismo proposto per implementare l'instradamento è un *Point Time-Locked Contract* (PTLC). I PTLC raggiungono anche l'atomicità, il funzionamento senza fiducia e la sicurezza multihop, ma lo fanno con maggiore efficienza e migliore privacy. L'implementazione efficiente dei PTLC dipende da un nuovo algoritmo di firma digitale chiamato *firme di Schnorr*, che sono state aggiunte in Bitcoin nel 2021.

Rivisitazione dell'Esempio della Donazione

Rivisitiamo l'esempio dalla prima parte di questo capitolo. Alice vuole donare a Dina tramite LN. Supponiamo che Alice voglia inviare a Dina 50.000 satoshi in donazione.

Affinché Alice possa pagare Dina, Alice avrà bisogno del nodo di Dina per generare una fattura Lightning. Ne discuteremo più dettagliatamente in seguito... Per ora, supponiamo che Dina abbia un sito Web in grado di produrre una fattura Lightning per le donazioni.

SUGGERIMENTO	I pagamenti LN possono essere inviati senza fattura utilizzando una funzione chiamata *keysend*, di cui parleremo più dettagliatamente in un capitolo dedicato. Per ora, spiegheremo il flusso di pagamento più semplice utilizzando una fattura.

Alice visita il sito di Dina, inserisce l'importo di 50.000 sats in un modulo e, in risposta, il nodo Lightning di Dina genera una richiesta di pagamento per 50.000 sats sotto forma di fattura Lightning. Questa interazione avviene sul Web e all'esterno di Lightning Network, come mostrato nella Figura 8-7.

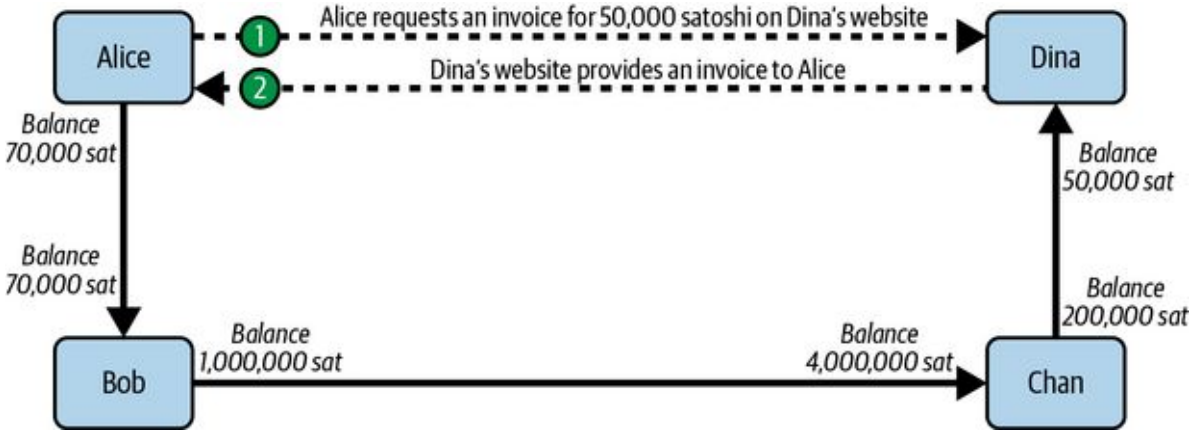

Figura 8-7. Alice richiede una fattura dal sito web di Dina

Come abbiamo visto negli esempi precedenti, assumiamo che Alice non abbia un canale di pagamento diretto verso Dina. Invece, Alice ha un canale con Bob, Bob ha un canale con Chan e Chan ha un canale con Dina. Per pagare Dina, Alice deve trovare un percorso che la colleghi a Dina. Discuteremo questo passaggio in modo più dettagliato nel Capitolo 12. Per ora, supponiamo che Alice sia in grado di raccogliere informazioni sui canali disponibili e veda che c'è un percorso da lei a Dina, attraverso Bob e Chan.

NOTA	Ricordi come Bob e Chan potrebbero aspettarsi un piccolo compenso per l'instradamento del pagamento attraverso i loro nodi? Alice vuole pagare a Dina 50.000 satoshi, ma come vedrai nelle sezioni seguenti invierà a Bob 50.200 satoshi. I 200 satoshi extra pagheranno Bob e Chan 100 satoshi ciascuno, come commissione di instradamento.

Ora, il nodo di Alice può costruire un pagamento Lightning. Nelle prossime sezioni vedremo come il nodo di Alice costruisce un HTLC per pagare Dina e come tale HTLC viene inoltrato lungo il percorso da Alice a Dina.

Liquidazione di HTL On-Chain vs Off-Chain

Lo scopo di Lightning Network è di permettere transazioni *off-chain* che sono attendibili esattamente come le transazioni on-chain perché nessuno può imbrogliare. Il motivo per cui nessuno può imbrogliare è perché in qualsiasi momento, qualsiasi partecipante può portare

le proprie transazioni off-chain, on-chain. Ogni transazione off-chain è pronta per essere inviata alla blockchain di Bitcoin in qualsiasi momento. Pertanto, la blockchain di Bitcoin funge da meccanismo di risoluzione delle controversie e risoluzione finale, se necessario.

Il semplice fatto che qualsiasi transazione possa essere portata on-chain in qualsiasi momento è precisamente il motivo per cui tutte quelle transazioni possono essere mantenute off-chain. Se sai di poter fare ricorso, puoi continuare a collaborare con gli altri partecipanti ed evitare la necessità di regolamenti on-chain e costi aggiuntivi.

In tutti gli esempi che seguono, assumeremo che ognuna di queste transazioni possa essere effettuata on-chain in qualsiasi momento. I partecipanti sceglieranno di tenere i propri fondi off-chain, ma non vi è alcuna differenza nella funzionalità del sistema oltre alle commissioni più elevate e al ritardo imposto dal mining on-chain delle transazioni. L'esempio funziona allo stesso modo se tutte le transazioni sono on-chain o off-chain.

Hash Time-Locked Contracts

In questa sezione spieghiamo come funzionano gli HTLC.

La prima parte di un HTLC è l'*hash*. Questo si riferisce all'uso di un algoritmo crittografico di hashing per impegnarsi in un segreto generato casualmente. La conoscenza del segreto consente il rimborso del pagamento. La funzione hash crittografica garantisce che mentre è impossibile per chiunque indovinare la preimmagine segreta, è facile per chiunque verificare l'hash e c'è solo una possibile preimmagine che risolve la condizione di pagamento.

Nella Figura 8-8 vediamo Alice ricevere una fattura Lightning da Dina. All'interno di quella fattura Dina ha codificato un *hash di pagamento*, che è l'hash crittografico di un segreto prodotto dal nodo di Dina. Il segreto di Dina si chiama *preimmagine del pagamento*. L'hash del pagamento funge da identificatore che può essere utilizzato per instradare il pagamento a Dina. La preimmagine di pagamento funge da ricevuta e prova di pagamento una volta completato il pagamento.

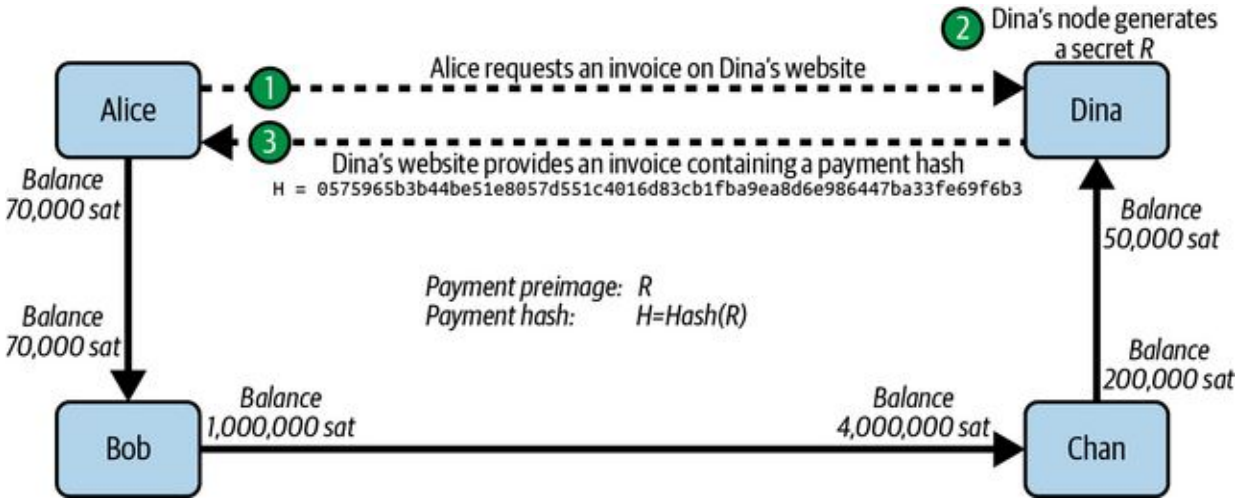

Figura 8-8. Alice riceve un hash di pagamento da Dina

Su Lightning Network, la preimmagine del pagamento di Dina non sarà una frase come `segreto di Dina` ma un numero casuale generato dal nodo di Dina. Chiamiamo queesto numero casuale R.

Il nodo di Dina calcolerà un hash crittografico di R, tale che:

H = SHA-256(R)

In questa equazione, H è l'hash, o *hash di pagamento,* e R è il segreto, o *preimmagine di pagamento*.

L'uso di una funzione hash crittografica è un elemento che garantisce il *funzionamento senza fiducia*. Gli intermediari dei pagamenti non hanno bisogno di fidarsi l'uno dell'altro perché sanno che nessuno può indovinare il segreto o falsificarlo.

HTLC in Bitcoin Script

Nel nostro esempio di moneta d'oro, Alice aveva un contratto imposto da un deposito a garanzia come questo:

Alice rimborserà Bob con 12 monete d'oro se può mostrare un messaggio valido che corrisponda all'hash `0575...f6b3`. Bob ha 24 ore per mostrare il segreto dopo la firma del contratto. Se Bob non fornisce il segreto entro questo momento, il deposito di Alice verrà rimborsato dal servizio di deposito a garanzia e il contratto perderà validità.

Vediamo come lo implementeremmo come HTLC in Bitcoin Script. Nell'Esempio 8-1 vediamo un HTLC in Bitcoin Script attualmente utilizzato su Lightning Network. Puoi trovare questa definizione in BOLT # 3, Transactions.

Esempio 8-1. HTLC implementato in Bitcoin Script (BOLT #3)

```
# To remote node with revocation key
OP_DUP OP_HASH160 <RIPEMD160(SHA256(revocationpubkey))> OP_EQUAL
OP_IF
    OP_CHECKSIG
OP_ELSE
    <remote_htlcpubkey> OP_SWAP OP_SIZE 32 OP_EQUAL
    OP_IF
        # To local node via HTLC-success transaction.
        OP_HASH160 <RIPEMD160(payment_hash)> OP_EQUALVERIFY
        2 OP_SWAP <local_htlcpubkey> 2 OP_CHECKMULTISIG
    OP_ELSE
        # To remote node after timeout.
        OP_DROP <cltv_expiry> OP_CHECKLOCKTIMEVERIFY OP_DROP
        OP_CHECKSIG
    OP_ENDIF
OP_ENDIF
```

Complicatissimo! Non preoccuparti, lo esamineremo e semplificheremo un passo alla volta...

Il Bitcoin Script attualmente utilizzato su LN è piuttosto complesso perché è ottimizzato per l'efficienza dello spazio on-chain, il che lo rende molto compatto ma difficile da leggere.

Nelle sezioni seguenti, ci concentreremo sugli elementi principali dello script e presenteremo script semplificati leggermente diversi da quelli effettivamente utilizzati in Lightning.

La parte principale dell'HTLC è nella riga 10 dell'Esempio 8-1. Vediamolo assieme!

Preimmagine di Pagamento e Verifica dell'Hash

Il nucleo di un HTLC è l'hash, in cui è possibile effettuare il pagamento se il destinatario

conosce la preimmagine del pagamento. Alice blocca il pagamento su uno specifico hash di pagamento e Bob deve presentare una preimmagine del pagamento per richiedere i fondi. Il sistema Bitcoin può verificare che l'immagine preliminare del pagamento di Bob sia corretta eseguendo l'hashing e confrontando il risultato con l'hash del pagamento utilizzato da Alice per bloccare i fondi.

Questa parte di un HTLC può essere implementata in Bitcoin Script come segue:

```
OP_SHA256 <H> OP_EQUAL
```

Alice può creare un output di transazione che paga, 50.200 satoshi con uno script di blocco sopra citato, sostituendo `<H>` con il valore hash `0575...f6b3` fornito da Dina. Quindi, Alice può firmare questa transazione e offrirla a Bob:

```
OP_SHA256 0575...f6b3 OP_EQUAL
```

Bob non può spendere questo HTLC finché non conosce il segreto di Dina, quindi spendere l'HTLC è subordinato all'adempimento del pagamento da parte di Bob fino a Dina.

Una volta che Bob ha il segreto di Dina, Bob può utilizzare questo output con uno script di sblocco contenente il valore segreto della preimmagine *R*.

Lo script di sblocco combinato con lo script di blocco produrrebbe:

```
<R> OP_SHA256 <H> OP_EQUAL
```

Il motore di Bitcoin Script valuterebbe questo script come segue:

- R viene messo in stack.
- L'operatore `OP_SHA256` prende il valore `R` dallo stack e ne calcaola l'hash, spingendo il risultato H_R nello stack.
- H viene messo in stack.
- L'operatore `OP_EQUAL` confronta H e H_R. Se sono uguali, il risultato è `TRUE`, lo script è completo e il pagamento è verificato.

Estendere gli HTLC da Alice a Dina

Alice ora estenderà l'HTLC attraverso la rete in modo che raggiunga Dina.

Nella Figura 8-9, vediamo l'HTLC propagato attraverso la rete da Alice a Dina. Alice ha dato a Bob un HTLC di 50.200 sats. Bob ora può creare un HTLC per 50.100 sats e darlo a Chan.

Bob sa che Chan non può riscattare l'HTLC di Bob senza trasmettere il segreto, a quel punto Bob può anche usare il segreto per riscattare l'HTLC di Alice. Questo è un punto davvero importante perché garantisce l'*atomicità* end-to-end dell'HTLC. Per spendere l'HTLC, bisogna svelare il segreto, che poi rende possibile anche ad altri di spendere il proprio HTLC. O tutti gli HTLC sono spendibili, o nessuno degli HTLC è spendibile: atomicità!

Poiché l'HTLC di Alice è di 100 sats in più rispetto all'HTLC che Bob ha dato a Chan, Bob guadagnerà 100 sats come commissione di routing se questo pagamento viene completato.

Bob non sta correndo rischi e non si fida di Alice o Chan. Invece, Bob confida che una transazione firmata insieme al segreto sarà riscattabile sulla blockchain di Bitcoin.

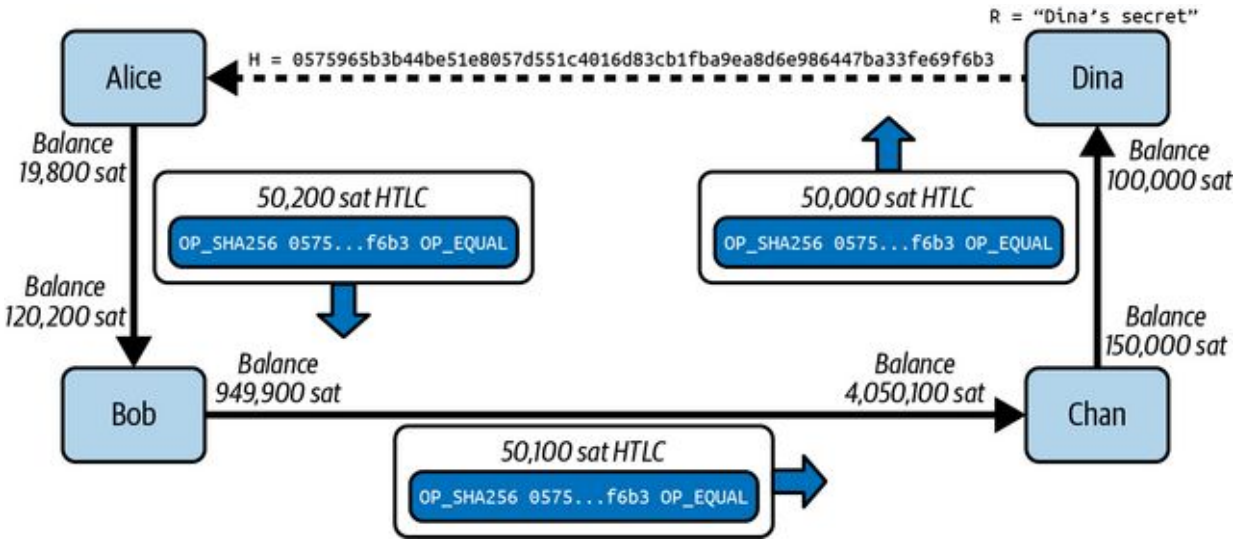

Figura 8-9. Propagazione dell'HTLC attraverso la rete

Allo stesso modo, Chan può estendere un HTLC di 50.000 sats a Dina. Non sta rischiando nulla né si fida di Bob o Dina. Per riscattare l'HTLC, Dina dovrebbe trasmettere il segreto, che Chan potrebbe usare per riscattare l'HTLC di Bob. Chan guadagnerebbe 100 satoshi come commissione di instradamento.

Propagazione Posteriore del Segreto

Dopo che Dina riceve l'HTLC di 50.000 da Chan, può essere pagata. Dina potrebbe semplicemente impegnare questo HTLC on-chain e spenderlo rivelando il segreto nella

transazione di spesa. Oppure Dina potrebbe aggiornare il saldo del canale con Chan rivelandogli il segreto. Non c'è motivo di sostenere una commissione di transazione e andare on-chain. Quindi Dina invia il segreto a Chan, e accettano di aggiornare i saldi dei loro canali per riflettere un pagamento Lightning di 50.000 satoshi a Dina. In Figura 8-10 vediamo Dina dare il segreto a Chan, adempiendo così l'HTLC.

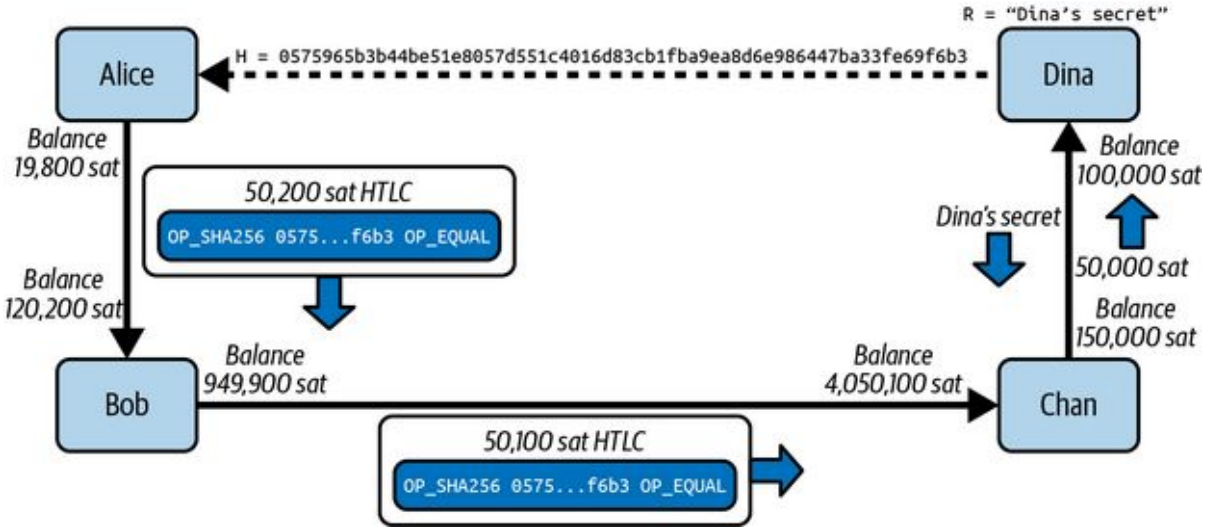

Figura 8-10. Dina risolve l'HTLC off-chain di Chan

Nota che il saldo del canale di Dina va da 50.000 satoshi a 100.000 satoshi. Il saldo del canale di Chan è ridotto da 200.000 satoshi a 150.000 satoshi. La capacità del canale non è cambiata, ma 50.000 si sono spostati dal lato del canale di Chan al lato del canale di Dina.

Chan ora ha il segreto e ha pagato a Dina 50.000 satoshi. Può farlo senza alcun rischio, perché il segreto consente a Chan di riscattare i 50.100 HTLC da Bob. Chan ha la possibilità di impegnare quell'HTLC on-chain e spenderlo rivelando il segreto sulla blockchain di Bitcoin. Ma, come Dina, preferisce evitare le spese di transazione. Quindi invia il segreto a Bob in modo che possano aggiornare i loro saldi di canale per riflettere un pagamento Lightning di 50.100 satoshi da Bob a Chan. In Figura 8-11 vediamo Chan che invia il segreto a Bob e riceve un pagamento in cambio.

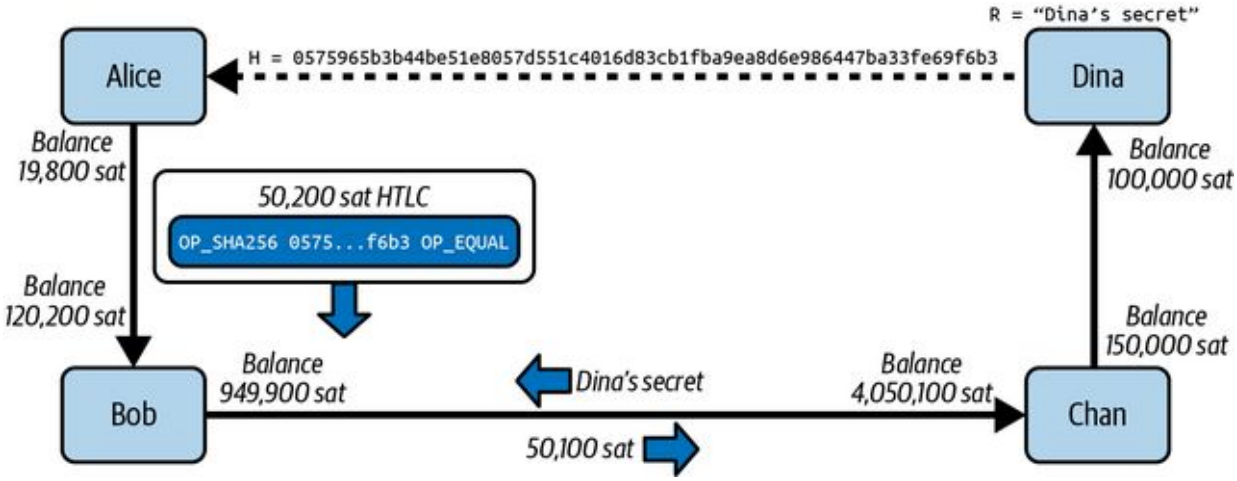

Figura 8-11. Chan risolve l'HTLC di Bob off-chain

Chan ha pagato a Dina 50.000 sats e ha ricevuto 50.100 sats da Bob. Quindi Chan ha 100 sats in più nei suoi saldi di canale, che ha guadagnato come commissione di instradamento.

Bob ora ha anche il segreto. Può usarlo per spendere l'HTLC di Alice on-chain. Oppure può evitare le spese di transazione saldando l'HTLC nel canale con Alice. In Figura 8-12 vediamo che Bob invia il segreto ad Alice ed assieme aggiornano il saldo del canale per riflettere un pagamento Lightning di 50.200 satoshi da Alice a Bob.

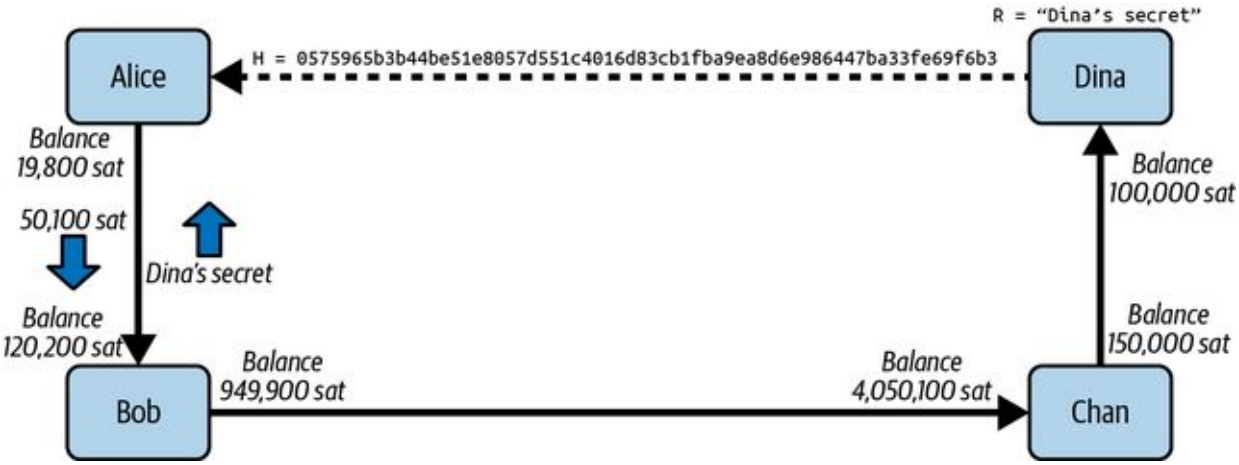

Figura 8-12. Bob risolve l'HTLC off-chain di Alice

Bob ha ricevuto 50.200 satoshi da Alice e ha pagato 50.100 satoshi a Chan, quindi ha 100 satoshi in più nei suoi saldi di canale derivati dalle commissioni di instradamento.

Alice riceve il segreto e salda l'HTLC da 50.200 satoshi. Il segreto può essere utilizzato come *ricevuta* per dimostrare che Dina è stata pagata per quello specifico hash di pagamento.

I saldi finali del canale riflettono il pagamento di Alice a Dina e le commissioni di instradamento pagate a ogni passaggio, come mostrato nella figura 8-13.

Figura 8-13. Saldi del canale dopo il pagamento

Signature Binding: Prevenzione del Furto di HTLC

C'è un problema... L'hai notato?

Se Alice, Bob e Chan creano gli HTLC come mostrato in Figura 8-13, devono affrontare un piccolo ma non trascurabile rischio di perdita. Ognuno di questi HTLC può essere riscattato (speso) da chiunque conosca il segreto. All'inizio solo Dina conosce il segreto. Dina dovrebbe spendere solo l'HTLC da Chan. Ma Dina potrebbe spendere tutti e tre gli HTLC contemporaneamente, o anche in un'unica transazione di spesa! Dopotutto, Dina conosce il segreto prima di chiunque altro. Allo stesso modo, una volta che Chan conosce il segreto, dovrebbe spendere solo l'HTLC offerto da Bob. Ma cosa succede se Chan spende anche l'HTLC offerto da Alice?

Questa cosa non è *trustless*! Fallisce la caratteristica di sicurezza più importante: dobbiamo non doverci fidare. Come facciamo a risolvere questo problema?

Lo script HTLC deve avere una condizione aggiuntiva che associ ogni HTLC a un destinatario specifico. Lo facciamo richiedendo una firma digitale che corrisponda alla chiave pubblica di ciascun destinatario, impedendo così a chiunque altro di spendere quell'HTLC. Poiché solo il destinatario designato ha la capacità di produrre una firma digitale corrispondente a quella

chiave pubblica, solo il destinatario designato può spendere quell'HTLC.

Diamo un'occhiata di nuovo agli script con questa modifica in mente. L'HTLC di Alice per Bob viene modificato per includere la chiave pubblica di Bob e l'operatore `OP_CHECKSIG`.

Ecco lo script HTLC modificato:

```
OP_SHA256 <H> OP_EQUALVERIFY <Bob's Pub> OP_CHECKSIG
```

SUGGERIMENTO	Nota che abbiamo anche modificato `OP_EQUAL` in `OP_EQUALVERIFY`. Quando un operatore ha il suffisso `VERIFY`, non restituisce `TRUE` o `FALSE` nello stack. Invece, *interrompe* l'esecuzione e fallisce lo script se il risultato è falso e continua senza alcun output dello stack se è vero.

Per riscattare questo HTLC, Bob deve presentare uno script di sblocco che include una firma dalla chiave privata di Bob e la preimage del pagamento segreta, come di seguito:

```
<Bob's Signature> <R>
```

Gli script di sblocco/blocco vengono combinati e valutati dal motore di scripting, come segue:

```
<Bob's Sig> <R> OP_SHA256 <H> OP_EQUALVERIFY <Bob's Pub> OP_CHECKSIG
```

1. `<Bob's Sig>` viene messo in stack.
2. `R` viene messo in stack.
3. `OP_SHA256` esegue l'hashing di `R` dalla cima dello stack e inserisce H_R nello stack.
4. H viene messo in stack.
5. OP_EQUALVERIFY estrae `H` e H_R e li confronta. Se non sono uguali, l'esecuzione si interrompe. Altrimenti, continuiamo senza output nello stack.
6. La chiave `<Bob's Pub>` viene inserita nello stack.
7. `OP_CHECKSIG` visualizza `<Bob's Sig>` e `<Bob's Pub>` e verifica la firma. Il risultato (`TRUE`/`FALSE`) viene inserito nello stack.

Come puoi vedere, questo è leggermente più complicato; ma ora abbiamo corretto l'HTLC e ci siamo assicurati che solo il destinatario previsto possa spenderlo.

Ottimizzazione Hash

Diamo un'occhiata alla prima parte dello script HTLC:

```
OP_SHA256 <H> OP_EQUALVERIFY
```

Se osserviamo questo nella precedente rappresentazione simbolica, sembra che gli operatori `OP_` occupino la maggior parte dello spazio... ma non è così. Bitcoin Script è codificato in binario, con ogni operatore che rappresenta un byte. Nel frattempo, il valore `<H>` che usiamo come segnaposto per l'hash di pagamento è un valore di 32 byte (256 bit). Puoi trovare un elenco di tutti gli operatori di Bitcoin Script e della loro codifica binaria ed esadecimale in Bitcoin Wiki: Script, o nell'Appendice D, "Operatori, Costanti e Simboli del Linguaggio di Script di Transazione", in Mastering Bitcoin.

Rappresentato in esadecimale, il nostro script HTLC sarebbe simile a questo:

```
a8 0575965b3b44be51e8057d551c4016d83cb1fba9ea8d6e986447ba33fe69f6b3 88
```

Nella codifica esadecimale, `OP_SHA256` è `a8` e `OP_EQUALVERIFY` è `88`. La lunghezza totale di questo script è di 34 byte, di cui 32 byte sono l'hash.

Come accennato in precedenza, qualsiasi partecipante al Lightning Network dovrebbe essere in grado di prendere una transazione off-chain che detiene e inserirla on-chain se ha bisogno di far valere la propria richiesta di fondi. Per accettare una transazione on-chain, dovrebbero pagare commissioni di transazione ai miner e queste commissioni sono proporzionali alla dimensione in byte della transazione.

Pertanto, vogliamo trovare modi per ridurre al minimo il "peso" delle transazioni on-chain ottimizzando il più possibile lo script. Un modo per farlo è aggiungere un'altra funzione di hash sopra l'algoritmo SHA-256, una che produce hash più piccoli. Il linguaggio Bitcoin Script fornisce l'operatore `OP_HASH160` che esegue il "doppio hash" di una preimage: prima la preimage viene sottoposta ad hashing con SHA-256, quindi l'hash risultante viene nuovamente sottoposto ad hashing con l'algoritmo hash RIPEMD160. L'hash risultante da RIPEMD160 è di 160 bit o 20 byte, molto più compatto. In Bitcoin Script questa è un'ottimizzazione molto comune che viene utilizzata in molti dei formati di indirizzi comuni.

Usiamo quindi questo tipo di ottimizzazione. Il nostro hash SHA-256 è `057596...69f6b3`. Facendogli fare un altro giro di hashing con RIPEMD160 ci dà il risultato:

```
R = "segreto di Dina"

H256 = SHA256(R)
```

```
H256 = 0575965b3b44be51e8057d551c4016d83cb1fba9ea8d6e986447ba33fe69f6b3
```

```
H160 = RIPEMD160(H256)
```

```
H160 = 9e017f6767971ed7cea17f98528d5f5c0ccb2c71
```

Alice può calcolare l'hash RIPEMD160 dell'hash di pagamento fornito da Dina e utilizzare l'hash più piccolo nel suo HTLC, così come Bob e Chan!

Lo script HTLC "ottimizzato" sarebbe simile a questo:

```
OP_HASH160 <H160> OP_EQUALVERIFY
```

Codificato in esadecimale:

```
a9 9e017f6767971ed7cea17f98528d5f5c0ccb2c71 88
```

Dove `OP_HASH160` è `a9` e `OP_EQUALVERIFY` è `88`. Questo script è lungo solo 22 byte! Abbiamo risparmiato 12 byte da ogni transazione che riscatta un HTLC on-chain.

Con questa ottimizzazione, arriviamo allo script HTLC mostrato nella riga 10 dell'Esempio 8.1:

```
...
    # To local node via HTLC-success transaction.
    OP_HASH160 <RIPEMD160(payment_hash)> OP_EQUALVERIFY…
```

Cooperazione HTLC e Errore di Timeout

Finora abbiamo esaminato la parte "hash" di HTLC e come funzionerebbe se tutti collaborassero e fossero online al momento del pagamento.

Cosa succede se qualcuno va offline o non collabora? Cosa succede se il pagamento non va a buon fine?

Dobbiamo garantire un modo per "fallire con garbo", perché errori di instradamento occasionali sono inevitabili. Ci sono due modi per fallire: cooperativamente e con un rimborso time-locked.

Il fallimento cooperativo è relativamente semplice: l'HTLC viene riavvolto da ogni partecipante del percorso, rimuovendo l'output HTLC dalle transazioni di impegno senza modificare il saldo. Vedremo come funziona in dettaglio nel Capitolo 9.

Diamo un'occhiata a come possiamo invertire un HTLC senza la collaborazione di uno o più partecipanti. Dobbiamo assicurarci che se uno dei partecipanti non collabora, i fondi non

siano semplicemente bloccati *per sempre* nell'HTLC. Ciò darebbe a qualcuno l'opportunità di riscattare i fondi di un altro partecipante: "Lascerò i tuoi fondi bloccati per sempre se non mi paghi il riscatto".

Per evitare ciò, ogni script HTLC include una clausola di rimborso collegata a un timelock. Ricordi il nostro contratto di deposito a garanzia originale? "Bob ha 24 ore per mostrare il segreto dopo la firma del contratto. Se Bob non fornisce il segreto entro questo termine, il deposito di Alice verrà rimborsato."

Il rimborso a tempo è una parte importante dello script che garantisce l'*atomicità*, in modo che l'intero pagamento end-to-end abbia esito positivo o negativo. Non esiste uno stato "pagato a metà" o un limbo di cui preoccuparsi. Se si verifica un errore, ogni partecipante può sciogliere l'HTLC in collaborazione con il proprio partner di canale o inserire unilateralmente la transazione di rimborso time-locked on-chain per riavere i propri soldi.

Per implementare questo rimborso in Bitcoin Script, utilizziamo un operatore speciale `OP_CHECKLOCKTIMEVERIFY` noto anche come `OP_CLTV`. Ecco lo script, come visto in precedenza nella riga 13 dell'Esempio 8.1:

```
...
      OP_DROP <cltv_expiry> OP_CHECKLOCKTIMEVERIFY OP_DROP
      OP_CHECKSIG
...
```

L'operatore `OP_CLTV` richiede un tempo di scadenza definito come l'altezza del blocco dopo la quale questa transazione è valida. Se il timelock della transazione non è impostato come `<cltv_expiry>`, la valutazione dello script non riesce e la transazione non è valida. In caso contrario, lo script continua senza alcun output nello stack. Ricorda, il suffisso `VERIFY` significa che questo operatore non restituisce `TRUE` o `FALSE`, ma si interrompe o continua senza l'output dello stack.

Essenzialmente, `OP_CLTV` funge da "guardiano" impedendo allo script di procedere ulteriormente se l'altezza del blocco `<cltv_expiry>` non è stata raggiunta sulla blockchain di Bitcoin.

L'operatore `OP_DROP` rilascia semplicemente l'elemento più in alto nello stack di script. Questo è necessario all'inizio perché c'è un elemento "rimanente" dalle righe di script precedenti. Subito dopo `OP_CLTV` bisogna rimuovere l'elemento `<cltv_expiry>` dalla parte superiore dello stack perché non più necessario.

Infine, una volta ripulito lo stack, dovrebbero essere rimaste una chiave pubblica e una firma che `OP_CHECKSIG` può verificare. Come abbiamo visto in precedenza, questo è necessario per garantire che solo il legittimo proprietario dei fondi possa rivendicarli, legando questo output alla loro chiave pubblica e richiedendo una firma.

Diminuzione dei Blocchi Temporali

Poiché gli HTLC vengono estesi da Alice a Dina, la clausola di rimborso temporizzato in ogni HTLC ha un *diverso* valore `cltv_expiry`. Lo vedremo in dettaglio nel Capitolo 10. Ma basti dire che per garantire uno riavvolgimento ordinato di un pagamento che fallisce, ogni hop deve attendere un po' meno tempo per il proprio rimborso. La differenza tra i timelock per ogni hop è chiamata `cltv_expiry_delta`, ed è impostata da ciascun nodo e annunciata alla rete, come vedremo in dettaglio nel Capitolo 11.

Ad esempio, Alice imposta il timelock del rimborso sul primo HTLC ad un'altezza del blocco corrente +500 blocchi. Bob imposterebbe quindi il timelock `cltv_expiry` sull'HTLC a Chan sul blocco corrente +450 blocchi. Chan imposterà il timelock a +400 blocchi dall'altezza del blocco corrente. In questo modo, Chan può ottenere un rimborso sull'HTLC che ha offerto a Dina *prima* che Bob ottenga un rimborso sull'HTLC che ha offerto a Chan. Bob può ottenere un rimborso dell'HTLC che ha offerto a Chan prima che Alice possa ottenere un rimborso per l'HTLC che ha offerto a Bob. Il timelock decrementale previene gare tra partecipanti e assicura che la catena HTLC venga riavvolta all'indietro, dalla destinazione verso l'origine.

Conclusione

In questo capitolo abbiamo visto come Alice può pagare Dina anche se non ha un canale di pagamento diretto. Alice può trovare un percorso che la collega a Dina e instradare un pagamento attraverso diversi canali in modo da raggiungerla.

Per garantire che il pagamento sia atomico e affidabile su più passaggi, Alice deve implementare un protocollo di equità in collaborazione con tutti i nodi intermedi nel percorso. Il protocollo di equità è attualmente implementato come HTLC, che impegna i fondi in un hash di pagamento derivato da una preimmagine di pagamento segreta.

Ciascuno dei partecipanti al percorso di pagamento può estendere un HTLC al partecipante successivo, senza preoccuparsi di furti o fondi bloccati. L'HTLC può essere riscattato

rivelando la preimmagine del pagamento segreta. Una volta che un HTLC raggiunge Dina, lei rivela la preimmagine, che scorre all'indietro, risolvendo tutti gli HTLC offerti.

Infine, abbiamo visto come una clausola di rimborso a tempo determinato completa l'HTLC, garantendo che ogni partecipante possa ottenere un rimborso se il pagamento fallisce ma per qualsiasi motivo uno dei partecipanti non collabora allo scioglimento degli HTLC stessi. Avendo sempre la possibilità di andare on-chain per un rimborso, l'HTLC raggiunge l'obiettivo di equità dell'atomicità e del funzionamento senza fiducia (trustless).

Capitolo 9. Funzionamento del Canale e Inoltro dei Pagamenti

In questo capitolo uniremo canali di pagamento e contratti hash time-locked (HTLC). Nel Capitolo 7, abbiamo spiegato il modo in cui Alice e Bob costruiscono un canale di pagamento tra i loro nodi. Abbiamo anche esaminato i meccanismi di impegno e penalità che proteggono il canale di pagamento. Nel Capitolo 8, abbiamo esaminato gli HTLC e come questi possono essere utilizzati per instradare un pagamento attraverso un percorso composto da più canali. In questo capitolo riuniamo quindi i due concetti esaminando come vengono gestiti gli HTLC su ciascun canale di pagamento, come gli HTLC sono impegnati nello stato del canale e come vengono regolati per aggiornare i saldi del canale.

Nello specifico, discuteremo di "Adding, settling, failing HTLC" e "Channel state machine" che formano la sovrapposizione tra il livello peer-to-peer e il livello di instradamento, come evidenziato da uno schema in Figura 9-1.

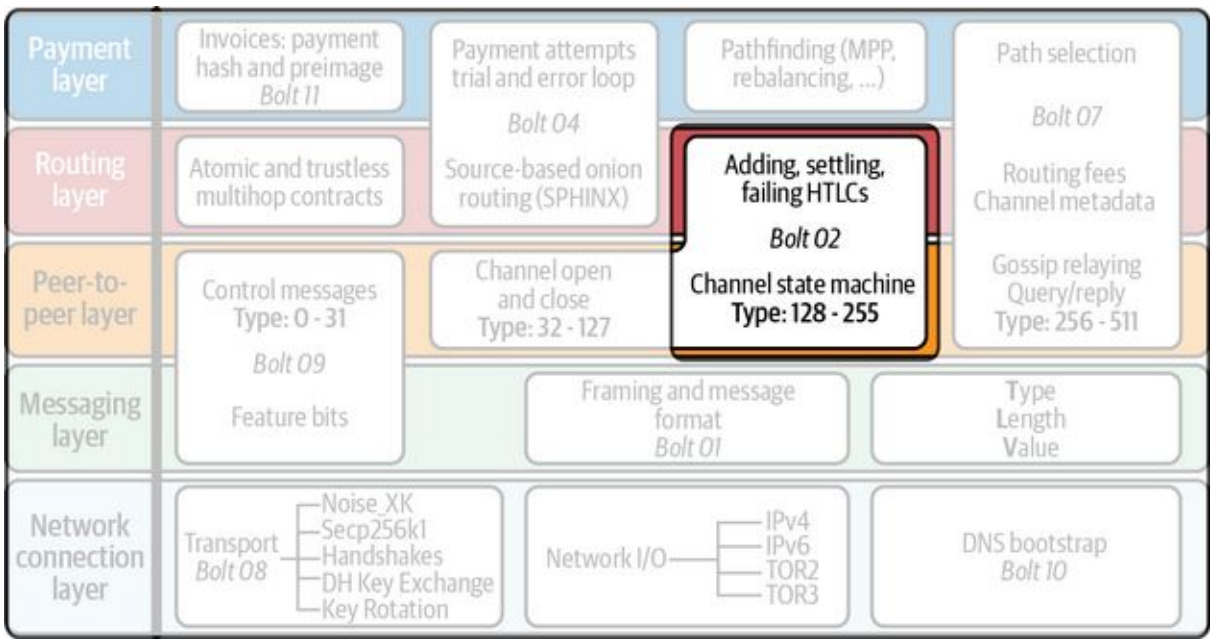

Figura 9-1. Funzionamento del canale e inoltro dei pagamenti nella suite di protocolli Lightning

Locale (Canale Singolo) vs Instradato (Canali Multipli)

Anche se è possibile inviare pagamenti attraverso un canale di pagamento aggiornando i saldi del canale e creando nuove transazioni di impegno, il protocollo Lightning utilizza HTLC anche per i pagamenti "locali" attraverso un canale. La ragione di ciò è mantenere lo stesso design del protocollo indipendentemente dal fatto che un pagamento sia solo un hop (su un singolo canale di pagamento) o più hop (instradato su più canali di pagamento).

Mantenendo la stessa astrazione sia per il locale che per il remoto semplifichiamo la progettazione del protocollo e miglioriamo la privacy. Per il destinatario di un pagamento non vi è alcuna differenza tra un pagamento effettuato direttamente dal proprio partner di canale e un pagamento inoltrato dal proprio partner di canale per conto di qualcun altro.

Inoltro dei Pagamenti e Aggiornamento degli Impegni con gli HTLC

Rivisiteremo l'esempio del Capitolo 8 per dimostrare come gli HTLC da Alice a Dina vengono impegnati in ciascun canale di pagamento. Come ricorderai nell'esempio, Alice sta pagando a Dina 50.000 satoshi indirizzando un HTLC tramite Bob e Chan, come da Figura 9-2.

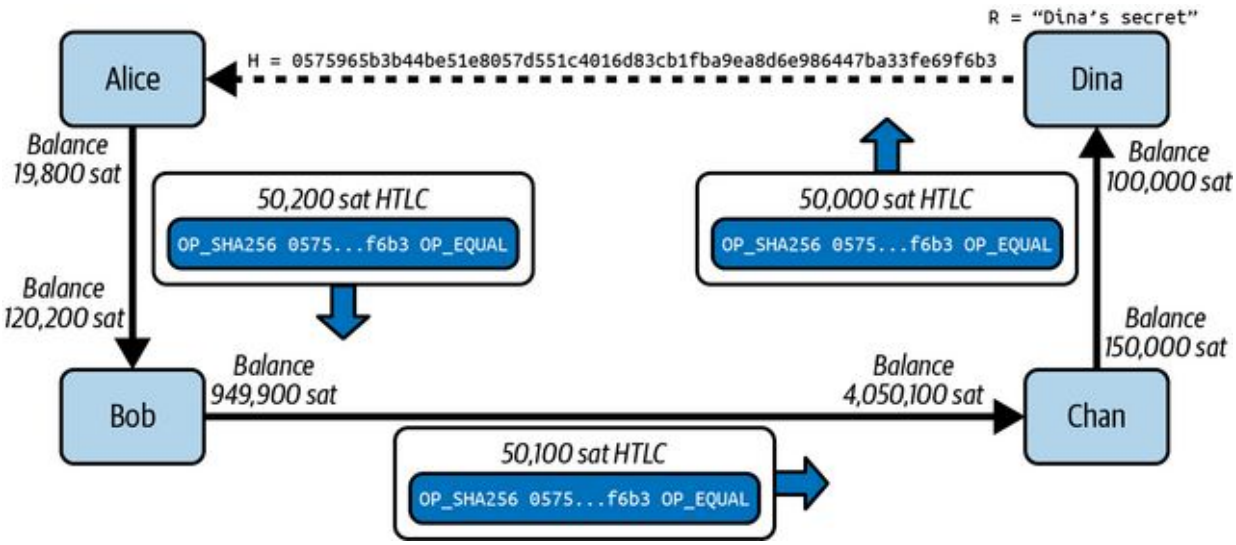

Figura 9-2. Alice paga Dina con un HTLC indirizzato tramite Bob e Chan

Ci concentreremo sul canale di pagamento tra Alice e Bob e rivedremo i messaggi e le transazioni che utilizzano per elaborare questo HTLC.

HTLC e Flusso di Messaggi di Impegno

Il flusso di messaggi tra Alice e Bob (o tra qualsiasi coppia di partner di canale) è mostrato nella Figura 9-3.

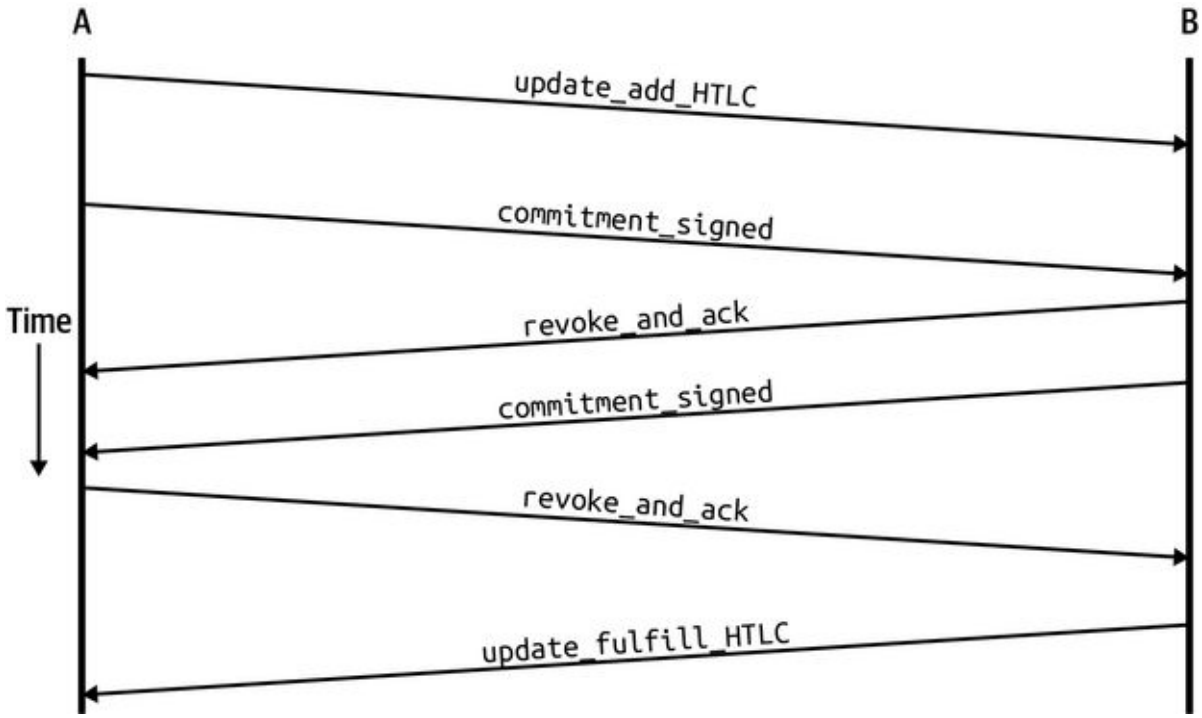

Figura 9-3. Il flusso di messaggi per l'impegno HTLC tra i partner di canale

Abbiamo già visto il `commitment_signed` e il `revoke_and_ack` nel Capitolo 7. Ora vedremo come gli HTLC si inseriscono nello schema. I due nuovi messaggi sono update_add_htlc, che Alice usa per chiedere a Bob di aggiungere un HTLC, e `update_fulfill_htlc`, che Bob usa per riscattare l'HTLC una volta ricevuto il segreto di pagamento (il segreto di Dina).

Inoltro di Pagamenti con HTLC

Alice e Bob creano un canale di pagamento che ha un saldo di 70.000 satoshi su ciascun lato.

Come abbiamo visto nel Capitolo 7, ciò significa che Alice e Bob hanno negoziato e ciascuno ha trattenuto transazioni di impegno. Queste transazioni di impegno sono asimmetriche, ritardate e revocabili e assomigliano all'esempio in Figura 9-4.

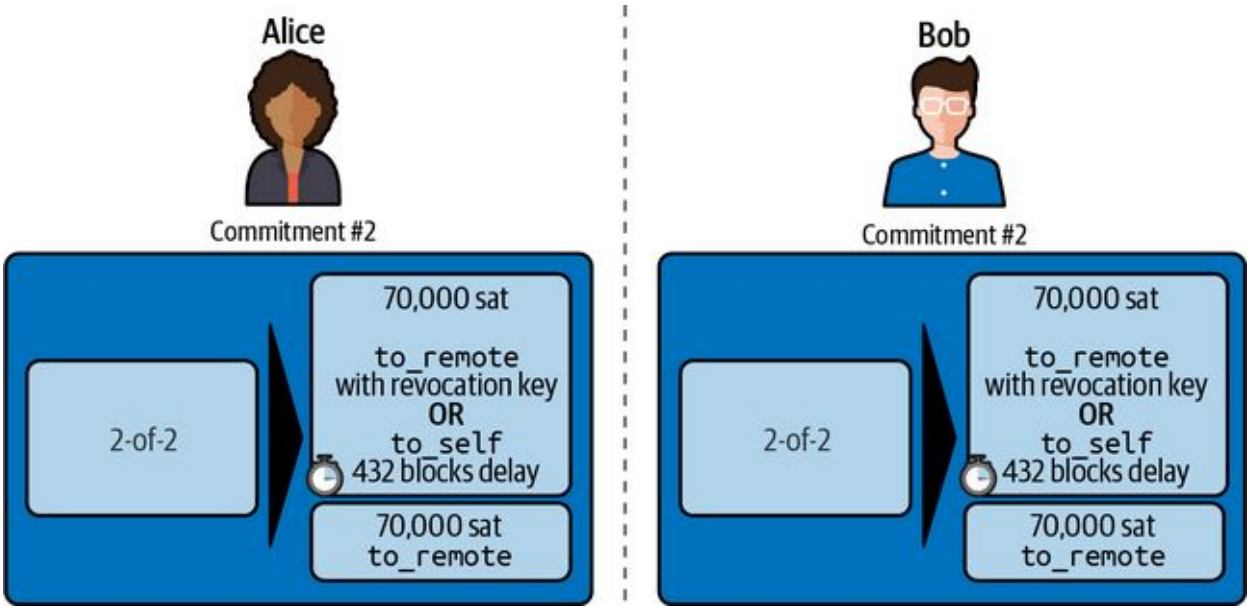

Figura 9-4. Transazioni iniziali di impegno di Alice e Bob

Aggiunta di un HTLC

Alice vuole che Bob accetti un HTLC del valore di 50.200 sats da inoltrare a Dina. Per far ciò, Alice deve inviare i dettagli di questo HTLC, inclusi l'hash e l'importo del pagamento, a Bob. Bob dovrà anche sapere dove inoltrarlo, cosa di cui parleremo in dettaglio nel Capitolo 10.

Per aggiungere l'HTLC, Alice avvia il flusso che abbiamo visto in Figura 9-3 inviando il messaggio `update_add_htlc` a Bob.

Il Messaggio `update_add_HTLC`

Alice invia il messaggio Lightning `update_add_HTLC` a Bob. Questo messaggio è definito in

BOLT #2: Peer Protocol, `update_add_HTLC`, ed è mostrato nell'Esempio 9-1.

Esempio 9-1. Il messaggio `update_add_HTLC`

```
[channel_id:channel_id]
[u64:id]
[u64:amount_msat]
[sha256:payment_hash]
[u32:cltv_expiry]
[1366*byte:onion_routing_packet]
```

`channel_id`

Questo è il canale che Alice ha con Bob e dove vuole aggiungere l'HTLC. Ricorda che Alice e Bob possono avere più canali tra loro.

`id`

Questo è un contatore HTLC e parte da `0` per il primo HTLC offerto a Bob da Alice e viene incrementato per ogni successivo HTLC.

`amount_msat`

Questa è la quantità (valore) dell'HTLC in millisatoshi. Nel nostro esempio è 50.200.000 millisatoshi (ovvero 50.200 satoshi).

`payment_hash`

Questo è l'hash di pagamento calcolato dalla fattura di Dina. È H = RIPEMD160(SHA-256(R)), dove R è il segreto di Dina che è conosciuto solo da Dina e verrà rivelato se Dina viene pagata.

`cltv_expiry`

Questo è il tempo di scadenza per questo HTLC, che sarà codificato come rimborso timelocked nel caso in cui l'HTLC non riesca a raggiungere Dina.

`onion_routing_packet`

Questo è un percorso cifrato a cipolla che dice a Bob dove inoltrare questo HTLC successivo (a Chan). L'onion routing viene trattato in dettaglio nel Capitolo 10.

SUGGERIMENTO	La contabilità all'interno di Lightning Network è in unità di millisatoshi (millesimi di satoshi), mentre la contabilità Bitcoin è in satoshi. Tutti gli importi in HTLC sono in millisatoshi, che vengono poi arrotondati al satoshi più vicino nelle transazioni di impegno in Bitcoin.

HTLC nelle Transazioni di Impegno

Le informazioni ricevute sono sufficienti a Bob per creare una nuova transazione di impegno. La nuova transazione di impegno ha gli stessi due output `to_self` e `to_remote` per il saldo di Alice e Bob, e un nuovo output che rappresenta l'HTLC offerto da Alice.

Abbiamo già visto la struttura di base di un HTLC nel Capitolo 8. Lo script completo di un HTLC offerto è definito in BOLT #3: Transactions, Offered HTLC Output ed è mostrato nell'Esempio 9-2.

Esempio 9-2. Script di output HTLC offerto

```
# Revocation (1)
OP_DUP OP_HASH160 <RIPEMD160(SHA256(revocationpubkey))> OP_EQUAL
OP_IF
    OP_CHECKSIG
OP_ELSE
    <remote_HTLCpubkey> OP_SWAP OP_SIZE 32 OP_EQUAL
    OP_IF
        # Redemption (2)
        OP_HASH160 <RIPEMD160(payment_hash)> OP_EQUALVERIFY
        2 OP_SWAP <local_HTLCpubkey> 2 OP_CHECKMULTISIG
```

```
    OP_ELSE
        # Refund (3)
        OP_DROP <cltv_expiry> OP_CHECKLOCKTIMEVERIFY OP_DROP
        OP_CHECKSIG
    OP_ENDIF
OP_ENDIF
```

1. La prima clausola del condizionale `OP_IF` è redimibile da Alice con una chiave di revoca. Se questo impegno viene successivamente revocato, Alice avrà una chiave di revoca per richiedere questo output in una transazione di penalità, prendendo l'intero saldo del canale.
2. La seconda clausola è riscattabile dalla preimage (segreto di pagamento o, nel nostro esempio, segreto di Dina) se viene rivelata. Ciò consente a Bob di richiedere questo output se ha il segreto di Dina, il che significa che ha consegnato con successo il pagamento a Dina.
3. La terza è un rimborso dell'HTLC ad Alice se l'HTLC scade senza raggiungere Dina. È bloccato nel tempo con la scadenza `cltv_expiry`. Ciò garantisce che il saldo di Alice non sia "bloccato" in un HTLC che non può essere indirizzato a Dina.

Ci sono tre modi per rivendicare questo output. Prova a leggere lo script e vedi se riesci a capirlo (ricorda, è un linguaggio basato su stack, quindi le cose appaiono "all'indietro").

Nuovo Impegno con l'Output HTLC

Bob dispone ora delle informazioni necessarie per aggiungere questo script HTLC come output aggiuntivo e creare una nuova transazione di impegno. Il nuovo impegno di Bob avrà 50.200 satoshi nell'output HTLC. Tale importo proverrà dal saldo del canale di Alice, quindi il nuovo saldo di Alice sarà di 19.800 satoshi (70.000 – 50.200 = 19.800). Bob costruisce questo impegno come un tentativo di "Commitment #3", mostrato nella Figura 9-5.

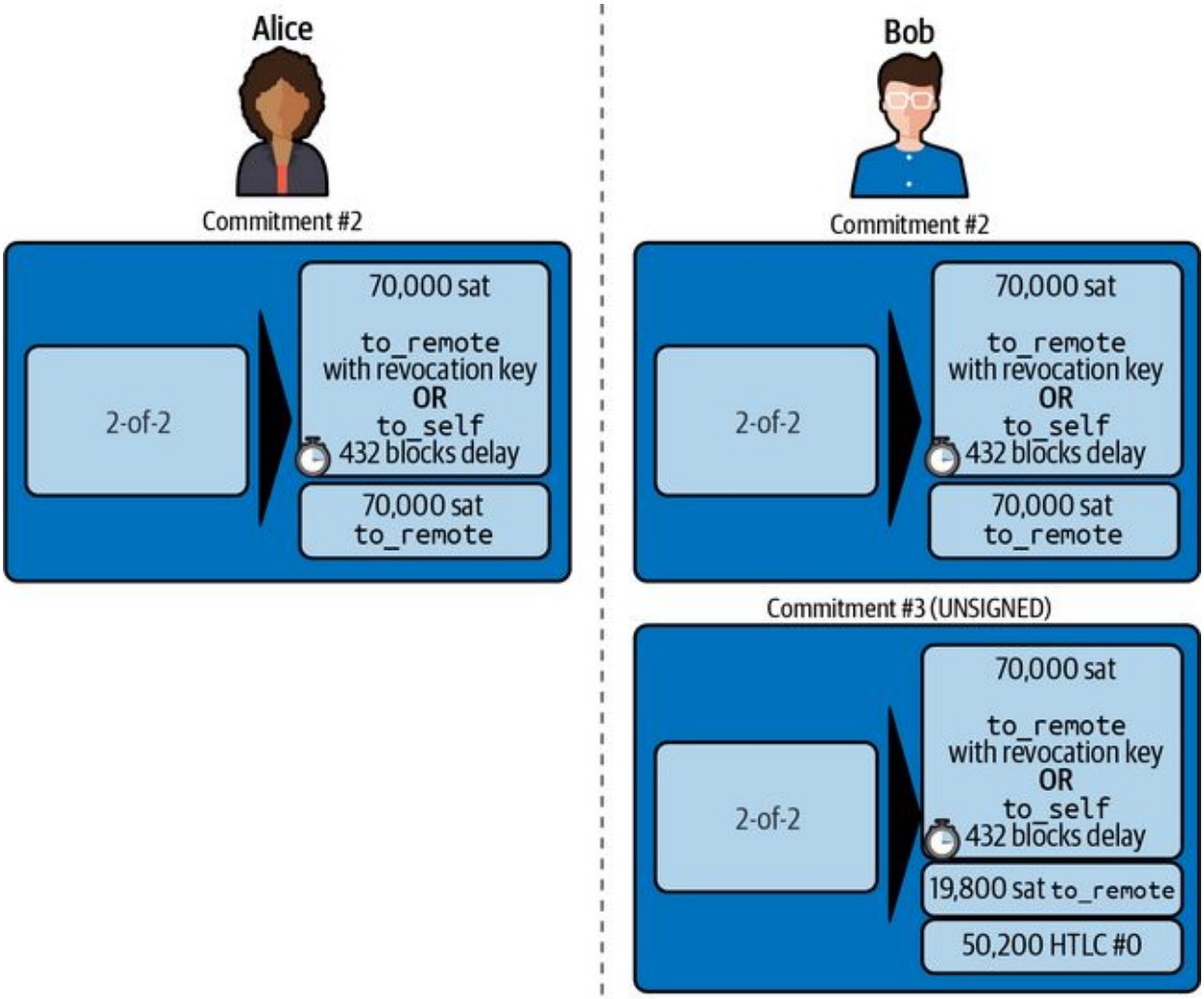

Figura 9-5. Il nuovo impegno di Bob con un output HTLC

Alice si Impegna

Poco dopo aver inviato il messaggio `update_add_htlc`, Alice si impegnerà nel nuovo stato del canale, in modo che l'HTLC possa essere aggiunto in modo sicuro da Bob. Bob ha le informazioni HTLC e ha costruito un nuovo impegno ma non ha ancora questo nuovo impegno firmato da Alice.

Alice invia `commitment_signed` a Bob con la firma per il nuovo impegno e per l'HTLC all'interno. Abbiamo visto il messaggio `commitment_signed` nel Capitolo 7 ed ora possiamo capire il resto dei campi. Come promemoria, lo mostriamo nuovamente nell'Esempio 9.3,

Esempio 9-3. Il messaggio `commitment_signed`

```
[channel_id:channel_id]
[signature:signature]
[u16:num_htlcs]
[num_htlcs*signature:htlc_signature]
```

I campi `num_htlcs` e `htlc_signature` ora hanno più senso:

`num_htlcs`

> Questo è il numero di HTLC in sospeso nella transazione di impegno. Nel nostro esempio, un solo HTLC, quello offerto da Alice.

`htlc_signature`

> Si tratta di un array di firme (`num_htlcs` in lunghezza), contenente le firme per gli output HTLC.

Alice può inviare queste firme senza esitazione: può sempre ottenere un rimborso se l'HTLC scade senza essere indirizzato a Dina.

Ora, Bob ha una nuova transazione di impegno firmata, come mostrato nella Figura 9-6.

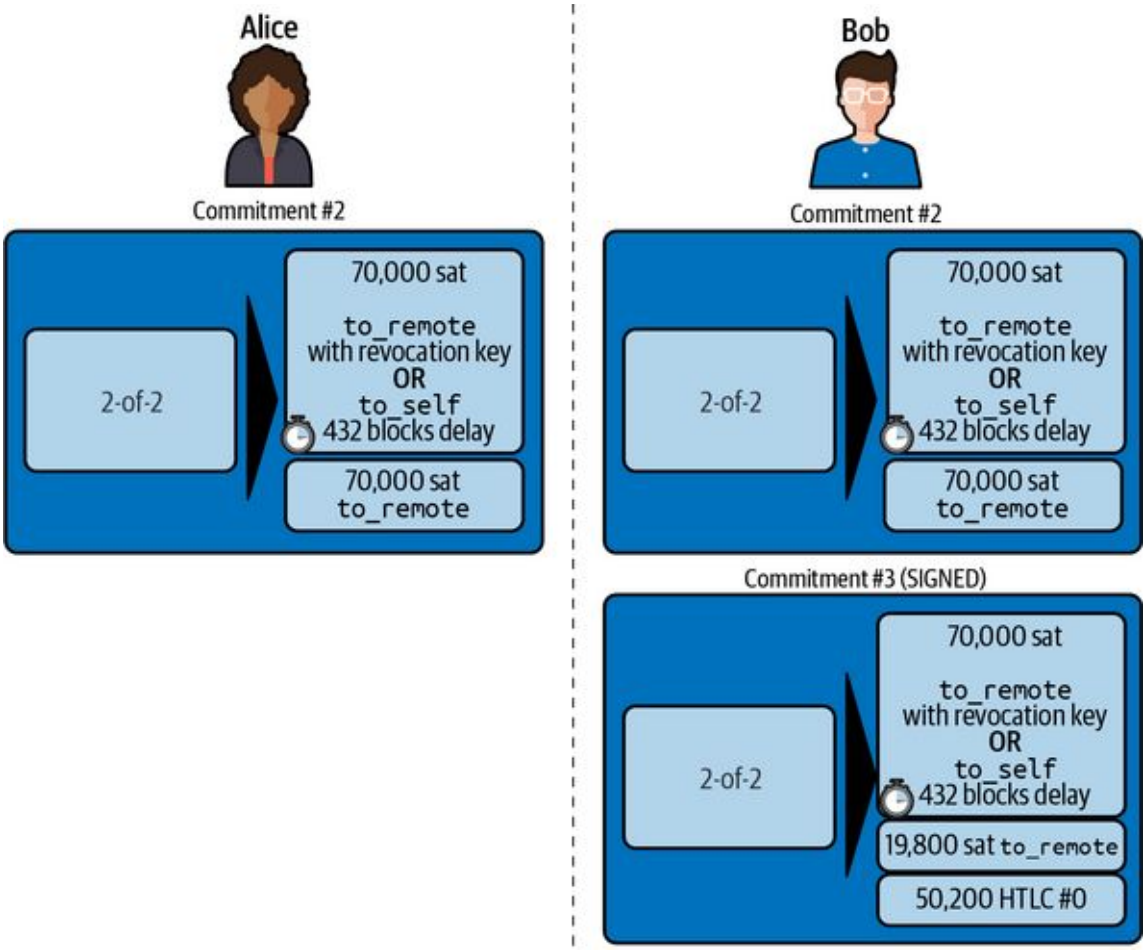

Figura 9-6. Bob ha un nuovo impegno firmato

Bob Riconosce il Nuovo Impegno e Revoca Quello Vecchio

Ora che Bob ha un nuovo impegno firmato, deve riconoscerlo e revocare il vecchio impegno. Lo fa inviando il messaggio `revoke_and_ack`, come abbiamo visto in precedenza nel Capitolo 7. Come promemoria, tale messaggio viene mostrato nell'Esempio 9-4..

Esempio 9-4. Il messaggio `revoke_and_ack`

```
[channel_id:channel_id]
[32*byte:per_impegno_segreto]
[point:next_per_commitment_point]
```

Bob invia il `per_commitment_secret` che consente ad Alice di costruire una chiave di

revoca per creare una transazione di penalità spendendo il vecchio impegno di Bob. Una volta che Bob lo ha inviato, non potrà mai pubblicare "Commitment #2" senza rischiare una transazione di penalità e perdere tutti i soldi. Quindi, il vecchio impegno è di fatto revocato.

Bob ha effettivamente spostato in avanti lo stato del canale, come mostrato nella Figura 9-7.

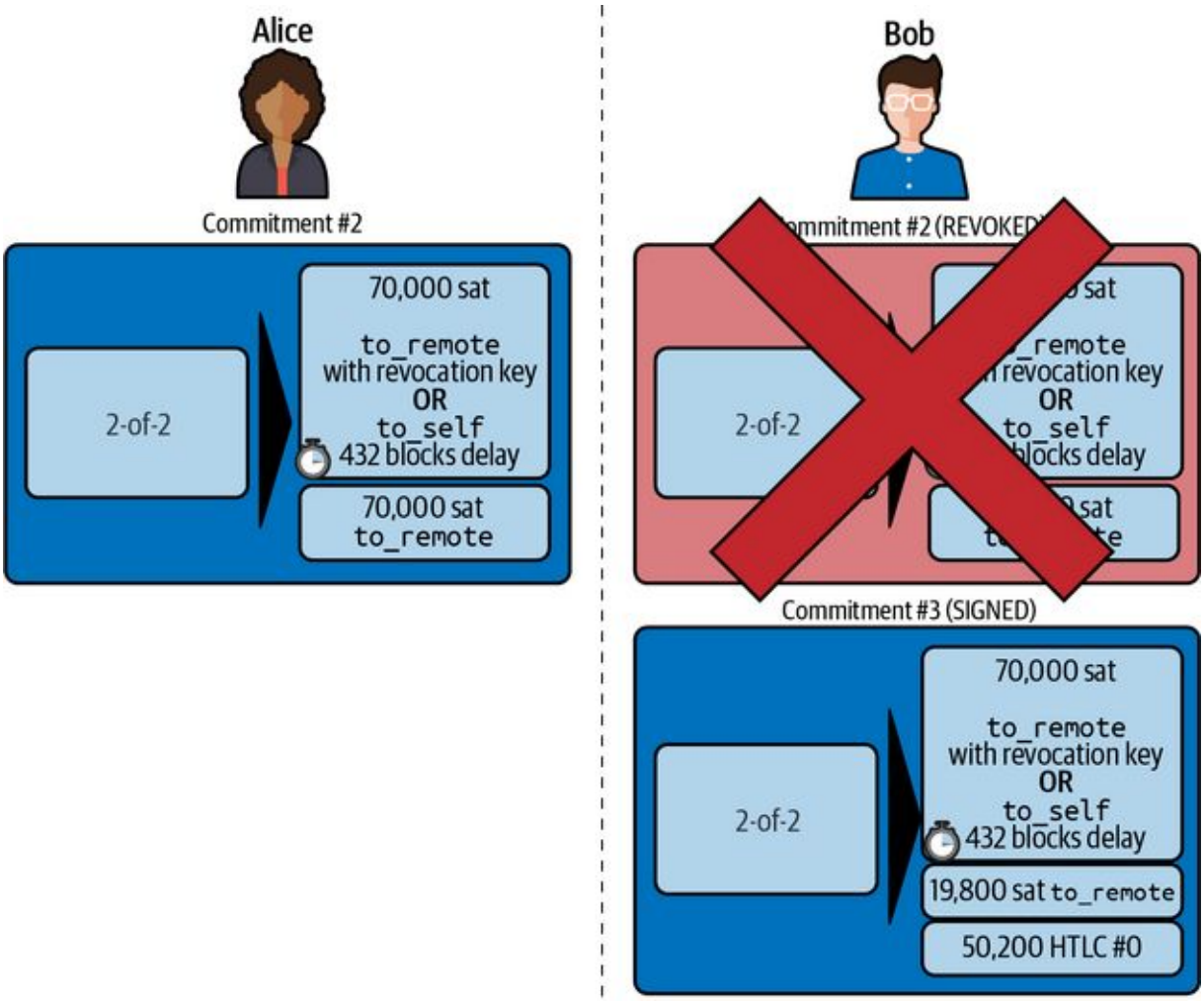

Figura 9-7. Bob ha revocato il vecchio impegno

Nonostante Bob abbia una nuova transazione di impegno (firmata) e un output HTLC all'interno, non può considerare che il suo HTLC sia stato impostato correttamente.

Prima ha bisogno che Alice revochi il suo vecchio impegno, perché altrimenti Alice può riportare il suo saldo a 70.000 satoshi. Bob deve assicurarsi che anche Alice abbia una

transazione di impegno contenente l'HTLC e abbia revocato il vecchio impegno.

Ecco perché, se Bob non è il destinatario finale dei fondi HTLC, non dovrebbe ancora inoltrare l'HTLC offrendo un HTLC sul canale successivo con Chan.

Alice ha costruito una nuova transazione di impegno speculare contenente il nuovo HTLC, ma deve ancora essere firmata da Bob. Possiamo vederlo nella Figura 9-8.

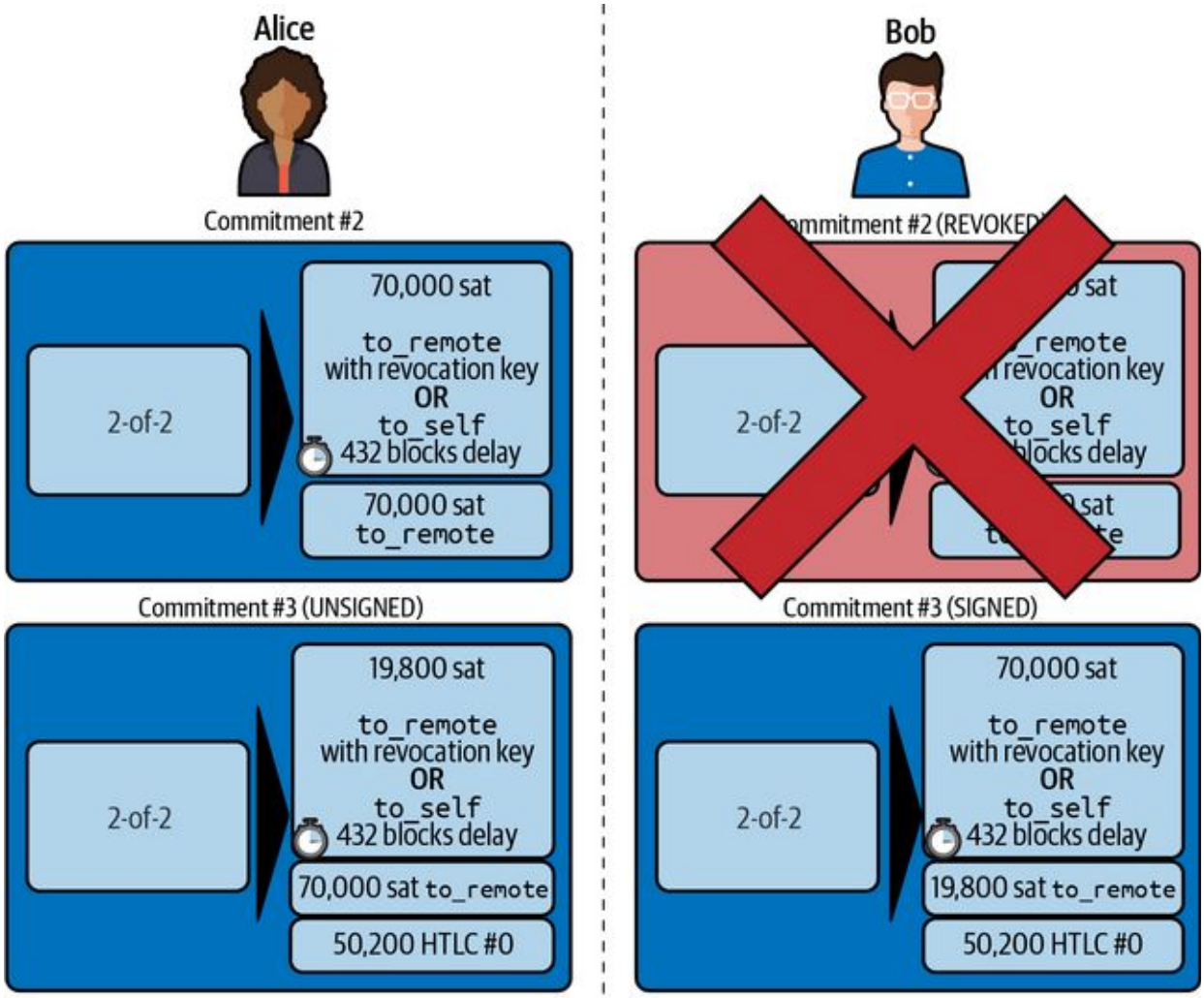

Figura 9-8. Il nuovo impegno di Alice con un output HTLC

Come abbiamo visto nel Capitolo 7, l'impegno di Alice è speculare a quello di Bob, in quanto contiene il costrutto asimmetrico, ritardato e revocabile per la revoca e sanzione di vecchi impegni. Il saldo di 19.800 satoshi di Alice (dopo aver dedotto il valore HTLC) è ritardato e

revocabile. Il saldo di 70.000 satoshi di Bob è immediatamente riscattabile.

Successivamente, il flusso di messaggi per `commitment_signed` e `revoke_and_ack` viene ora ripetuto, ma nella direzione opposta. Bob invia `commitment_signed` per firmare il nuovo impegno di Alice, e Alice risponde revocando il suo vecchio impegno.

Per completezza, esaminiamo rapidamente le transazioni di impegno man mano che si verifica questo ciclo di impegno/revoca.

Bob si Impegna

Bob ora invia un `commitment_signed` ad Alice, con le sue firme per la nuova transazione di impegno di Alice, incluso l'output HTLC che ha aggiunto.

Ora Alice ha la firma per la nuova transazione di impegno. Lo stato del canale è mostrato nella Figura 9-9.

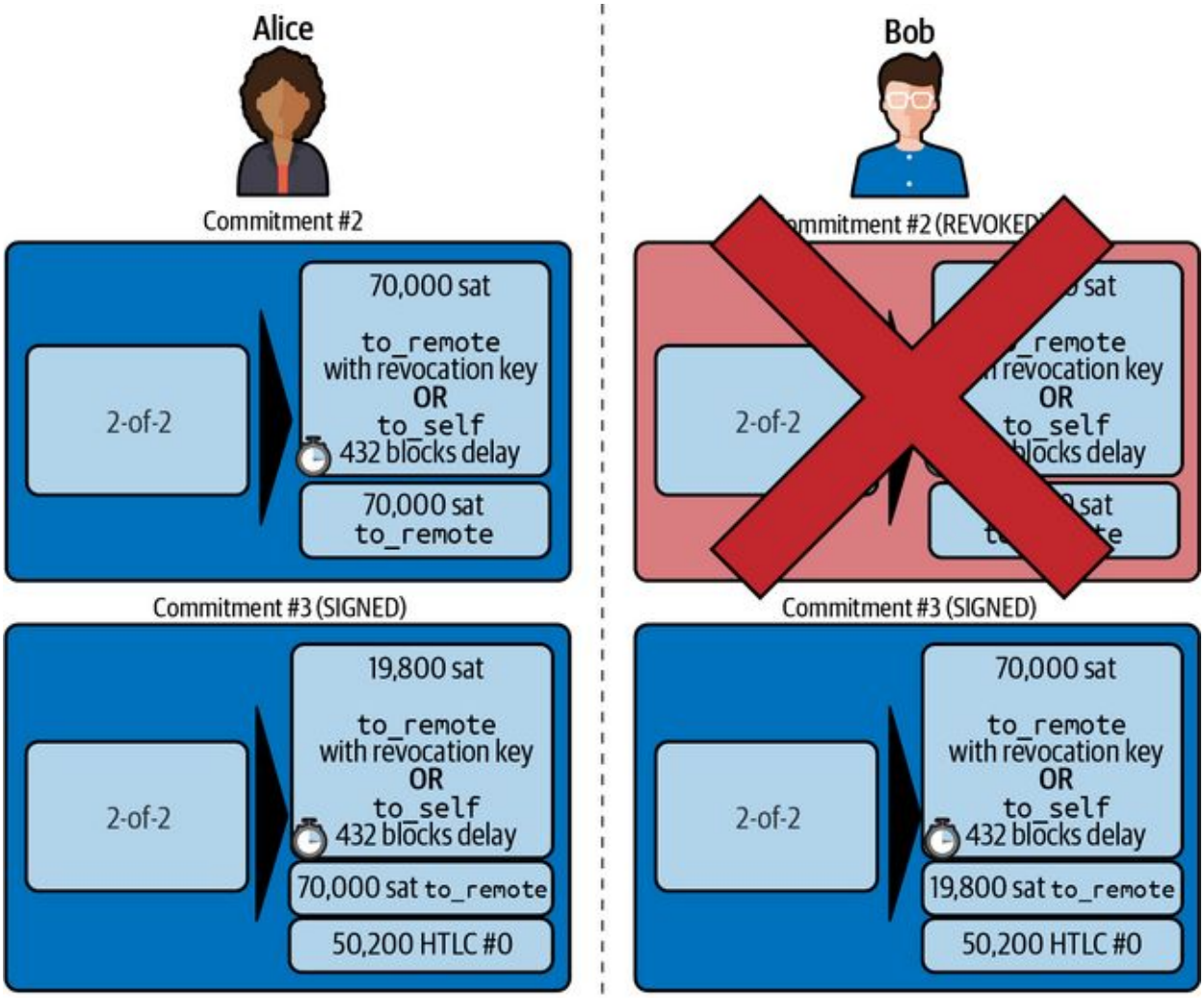

Figura 9-9. Alice ha un nuovo impegno firmato

Alice può ora riconoscere il nuovo impegno revocando quello vecchio. Alice invia il messaggio `revoke_and_ack` contenente il `per_commitment_point` che consentirà a Bob di costruire una chiave di revoca e una transazione di penalità. Così, Alice revoca il suo vecchio impegno.

Lo stato del canale è mostrato nella Figura 9-10.

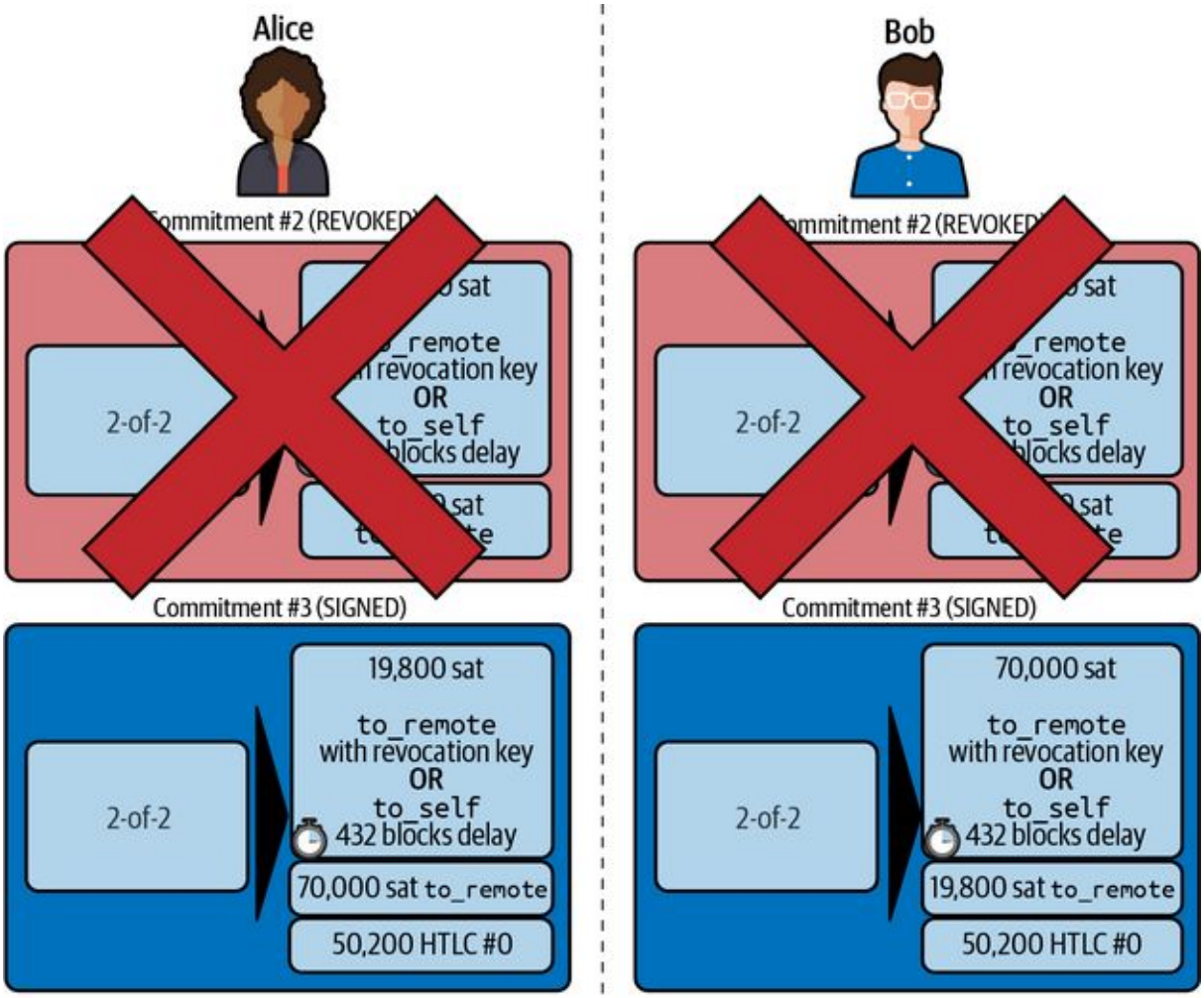

Figura 9-10. Alice ha revocato il vecchio impegno

HTLC Multipli

In qualsiasi momento, Alice e Bob possono avere dozzine o addirittura centinaia di HTLC su un singolo canale. Ogni HTLC viene offerto e aggiunto alla transazione di impegno come output aggiuntivo. Una transazione di impegno ha quindi sempre due output per i saldi dei partner di canale e qualsiasi numero di output HTLC, uno per ogni HTLC.

Come abbiamo visto nel messaggio `commitment_signed`, esiste un array per le firme HTLC in modo che più impegni HTLC possano essere trasmessi contemporaneamente.

L'attuale numero massimo di HTLC consentiti su un canale è di 483 HTLC per tenere conto della dimensione massima della transazione Bitcoin e garantire che le transazioni di impegno

continuino a essere transazioni Bitcoin valide.

Come vedremo nella prossima sezione, il massimo è solo per gli HTLC in *sospeso* perché, una volta che un HTLC è adempiuto (o fallisce a causa di timeout/errore), viene rimosso dalla transazione di impegno.

Adempimento degli HTLC

Ora Bob e Alice hanno entrambi una nuova transazione di impegno con un output HTLC aggiuntivo e abbiamo compiuto un passo importante verso l'aggiornamento di un canale di pagamento.

Il nuovo bilancio di Alice e Bob non riflette ancora che Alice abbia inviato con successo 50.200 satoshi a Bob.

Tuttavia, gli HTLC sono ora impostati in modo tale da rendere possibile un regolamento sicuro in cambio della prova del pagamento.

Propagazione degli HTLC

Supponiamo che Bob continui la catena e organizzi un HTLC con Chan per 50.100 satoshi. Il processo sarà esattamente lo stesso che abbiamo appena visto tra Alice e Bob. Bob invierà `update_add_htlc` a Chan, poi si scambieranno messaggi `commitment_signed` e `revoke_and_ack` in due turni, facendo progredire il proprio canale allo stato successivo.

Successivamente, Chan farà lo stesso con Dina: offre un HTLC da 50.000 satoshi, si impegna, revoca, ecc. Tuttavia, Dina è il destinatario finale dell'HTLC. Dina è l'unica a conoscere il segreto del pagamento (la preimage dell'hash del pagamento). Pertanto, Dina può soddisfare immediatamente l'HTLC con Chan!

Dina Soddisfa l'HTLC con Chan

Dina può risolvere l'HTLC inviando un messaggio `update_fulfill_htlc` a Chan. Il messaggio `update_fulfill_htlc` è definito in BOLT #2: Peer Protocol,

`update_fulfill_htlc` ed è mostrato di seguito:

```
[channel_id:channel_id]
[u64:id]
[32*byte:payment_preimage]
```

È un messaggio molto semplice:

`channel_id`

L'ID del canale su cui è eseguito il commit dell'HTLC.

`id`

L'ID dell'HTLC (abbiamo iniziato con `0` e incrementato per ogni HTLC sul canale).

`payment_preimage`

Il segreto che prova che il pagamento è stato effettuato e che riscatta l'HTLC. Questo è il valore `R` sottoposto ad hashing da Dina per produrre l'hash del pagamento nella fattura ad Alice.

Quando Chan riceve questo messaggio, controllerà immediatamente se `payment_preimage` (chiamiamolo *R*) produce l'hash di pagamento (chiamiamolo *H*) nell'HTLC che ha offerto a Dina. Lo fa in questo modo:

H = RIPEMD160(SHA-256 (*R*))

Se il risultato *H* corrisponde all'hash del pagamento nell'HTLC, Chan può celebrare. Questo segreto tanto atteso può essere utilizzato per riscattare l'HTLC e verrà ritrasmesso lungo la catena di canali di pagamento fino ad Alice, risolvendo ogni HTLC che faceva parte di questo pagamento a Dina.

Torniamo al canale di Alice e Bob e guardiamoli scogliere l'HTLC. Per arrivarci, supponiamo che Dina abbia inviato `update_fulfill_htlc` a Chan, Chan abbia inviato `update_fulfill_htlc` a Bob e Bob abbia inviato `update_fulfill_htlc` ad Alice. La preimmagine del pagamento si è propagata fino ad Alice.

Bob Risolve l'HTLC con Alice

Quando Bob invia `update_fulfill_htlc` ad Alice, conterrà la stessa `payment_preimage` che Dina ha selezionato per la sua fattura. Quel `payment_preimage` ha viaggiato a ritroso lungo il percorso di pagamento. Ad ogni passaggio, `channel_id` sarà diverso e `id` (HTLC ID) potrebbe essere diverso. Ma la preimmagine è la stessa!

Alice convaliderà anche il `payment_preimage` ricevuto da Bob. Confronterà il suo hash con l'hash del pagamento HTLC nell'HTLC che ha offerto a Bob. Scoprirà che questa preimmagine corrisponde all'hash nella fattura di Dina. Questa è la prova che Dina è stata pagata.

Il flusso di messaggi tra Alice e Bob è mostrato nella Figura 9-11.

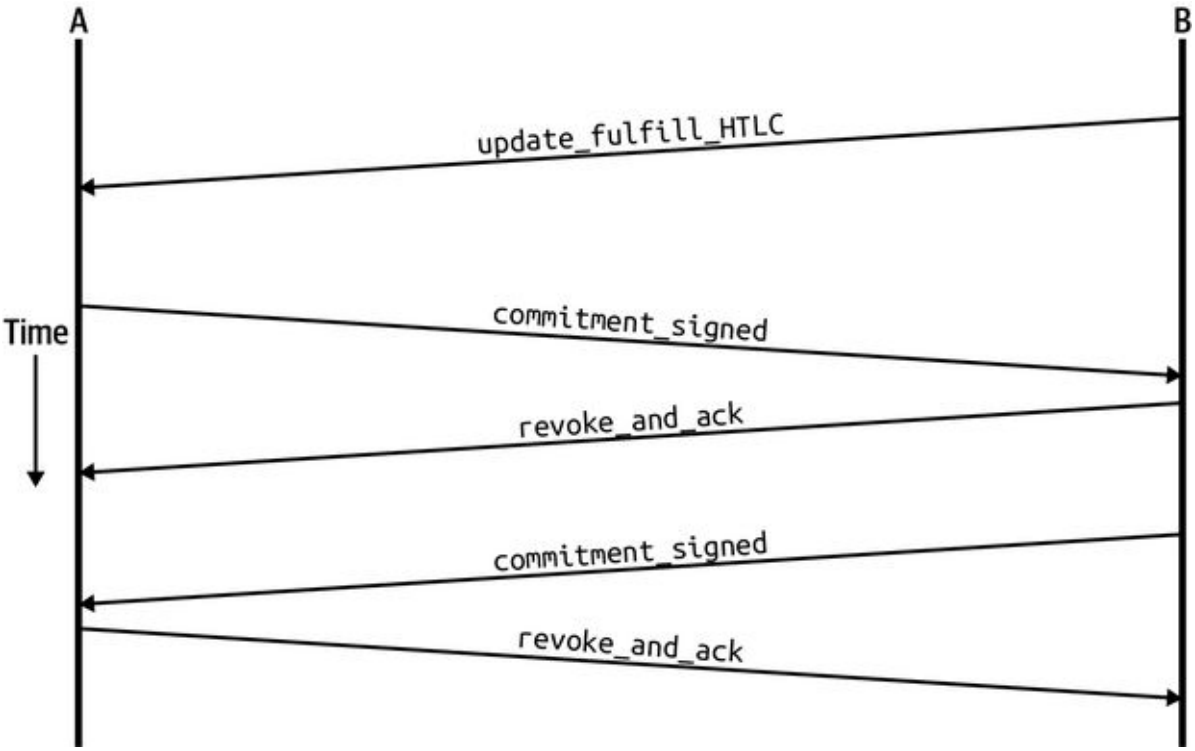

Figura 9-11. Il flusso di messaggi di compimento dell'HTLC

Sia Alice che Bob possono ora rimuovere l'HTLC dalle transazioni di impegno e aggiornare i loro saldi di canale.

Creano nuovi impegni (Commitment #4), come mostrato nella Figura 9-12.

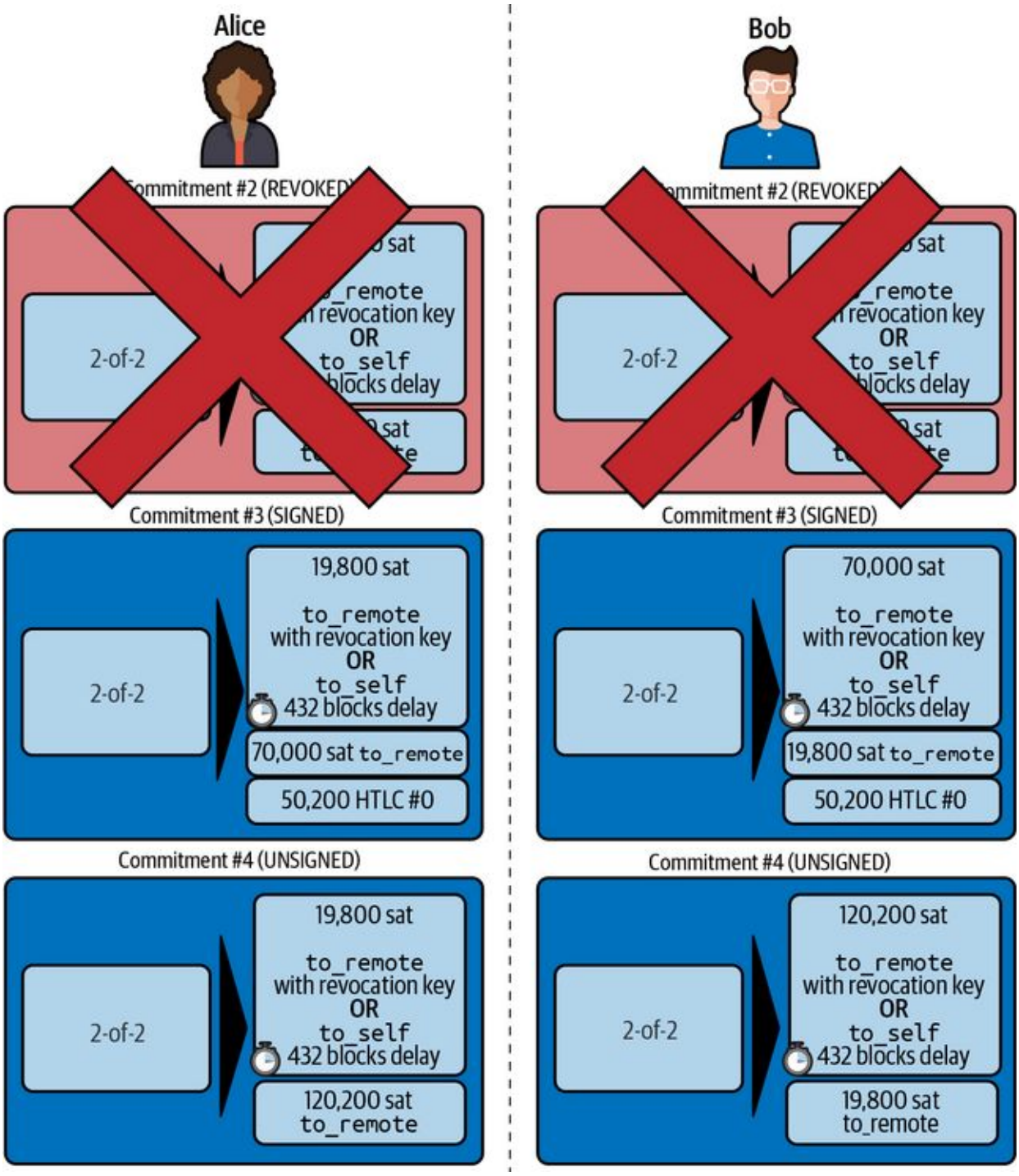

Figura 9-12. L'HTLC viene rimosso e i saldi vengono aggiornati in nuovi impegni

Successivamente, completano due round di impegno e revoca. Innanzitutto, Alice invia `commitment_signed` per firmare la nuova transazione di impegno di Bob. Bob risponde con `revoke_and_ack` per revocare il suo vecchio impegno. Una volta che Bob ha spostato in avanti lo stato del canale, gli impegni hanno l'aspetto che vediamo nella Figura 9-13.

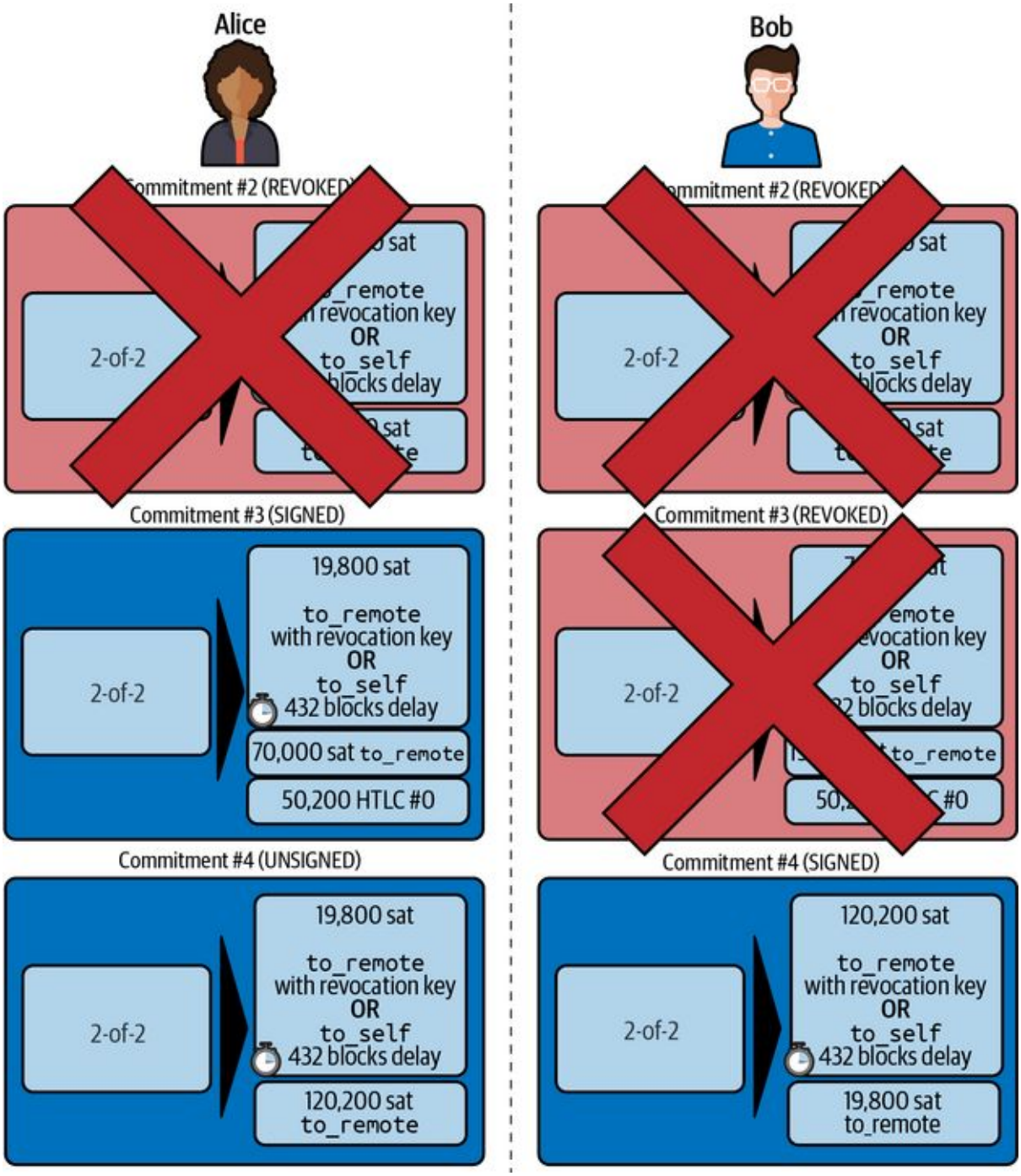

Figura 9-13. Alice firma il nuovo impegno di Bob e Bob revoca quello vecchio

Infine, Bob firma l'impegno di Alice inviandole un messaggio `commitment_signed`. Quindi Alice riconosce e revoca il suo vecchio impegno inviando `revoke_and_ack` a Bob. Il risultato

finale è che sia Alice che Bob hanno spostato lo stato del loro canale sul Commitment #4, hanno rimosso l'HTLC e hanno aggiornato i loro saldi. Il loro stato di canale corrente è rappresentato dalle transazioni di impegno mostrate nella Figura 9-14.

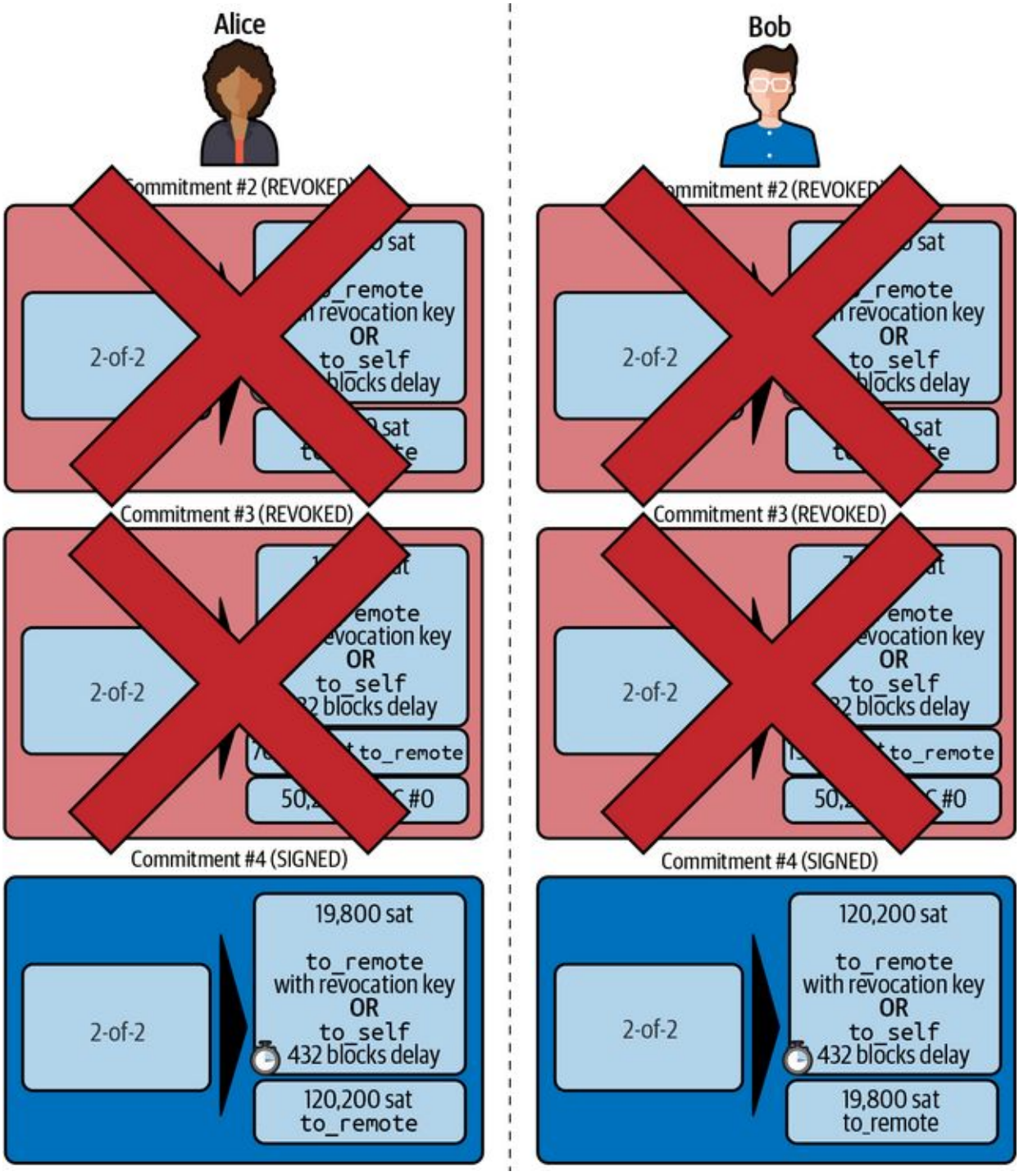

Figura 9-14. Alice e Bob regolano l'HTLC e aggiornano i saldi

Rimozione di un HTLC a Causa di Errore o Scadenza

Se un HTLC non può essere soddisfatto, può essere rimosso dall'impegno del canale utilizzando lo stesso processo di impegno e revoca.

Invece di `update_fulfill_htlc`, Bob invierebbe un `update_fail_htlc` o `update_fail_malformed_htlc`. Questi due messaggi sono definiti in BOLT #2: Peer Protocol, Removing an HTLC.

Il messaggio `update_fail_htlc` è mostrato di seguito:

```
[channel_id:channel_id]
[u64:id]
[u16:len]
[len*byte:reason]
```

È abbastanza autoesplicativo. Il campo `reason` multibyte è definito in BOLT #4: Onion Routing, che descriveremo nel Capitolo 10.

Se Alice ricevesse un `update_fail_htlc` da Bob, il processo si svilupperebbe più o meno allo stesso modo: i due partner di canale rimuoverebbero l'HTLC, creerebbero transazioni di impegno aggiornate e passerebbero attraverso due cicli di impegno/revoca per spostare lo stato del canale al successivo impegno. L'unica differenza: i saldi finali torneranno a quelli che erano senza HTLC, essenzialmente rimborsando Alice per il valore HTLC.

Effettuare un Pagamento Locale

A questo punto, capirai facilmente perché gli HTLC vengono utilizzati sia per i pagamenti remoti che per quelli locali. Quando Alice paga Bob per un caffè, non si limita ad aggiornare il saldo del canale e ad impegnarsi in un nuovo stato. Invece il pagamento viene effettuato con un HTLC, nello stesso modo in cui Alice ha pagato Dina. Il fatto che ci sia un solo salto di canale non fa differenza. Funzionerebbe così:

1. Alice ordina un caffè dal negozio di Bob.
2. Il negozio di Bob invia una fattura con un hash di pagamento.
3. Alice costruisce un HTLC da quell'hash di pagamento.

4. Alice offre l'HTLC a Bob con `update_add_htlc`.
5. Alice e Bob si scambiano impegni e revoche aggiungendo l'HTLC alle loro transazioni di impegno.
6. Bob invia `update_fulfill_htlc` ad Alice con la preimage del pagamento.
7. Alice e Bob si scambiano impegni e revoche rimuovendo l'HTLC e aggiornando i saldi del canale.

Se un HTLC viene inoltrato attraverso molti canali o semplicemente eseguito in un singolo canale/"hop", il processo è esattamente lo stesso

Conclusione

In questo capitolo abbiamo visto come funzionano assieme le transazioni di impegno (dal Capitolo 7) e gli HTLC (dal Capitolo 8). Abbiamo visto come un HTLC viene aggiunto a una transazione di impegno e come viene soddisfatto. Abbiamo visto come il sistema asimmetrico, ritardato e revocabile per imporre lo stato del canale viene esteso agli HTLC.

Abbiamo anche visto come un pagamento locale e un pagamento instradato multihop vengono gestiti in modo identico: utilizzando HTLC.

Nel prossimo capitolo esamineremo il sistema di instradamento dei messaggi cifrati chiamato *onion routing*.

Capitolo 10. Onion Routing

In questo capitolo descriveremo il meccanismo di instradamento a cipolla di LN. L'invenzione dell'*onion routing* precede di 25 anni Lightning Network! L'onion routing è stato inventato dai ricercatori della Marina degli Stati Uniti come protocollo di sicurezza delle comunicazioni. L'onion routing è famoso perché utilizzato da Tor, l'overlay Internet con instradamento a cipolla che consente a ricercatori, attivisti, agenti dell'intelligence e chiunque altro di utilizzare Internet in modo privato e anonimo.

In questo capitolo ci concentreremo sulla parte "Source-based Onion Routing (SPHINX)" dell'architettura del protocollo Lightning, evidenziata da un contorno al centro (Routing layer) della Figura 10-1

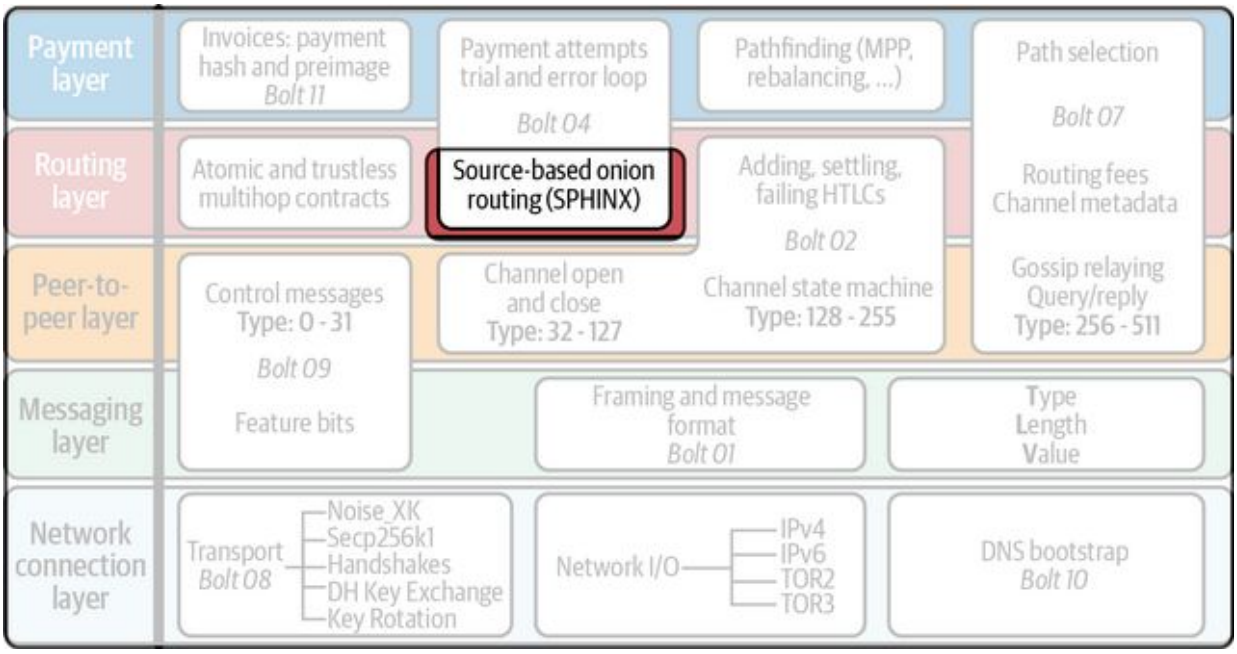

Figura 10-1. L'onion routing nella suite di protocolli Lightning

L'*onion routing* descrive un metodo di comunicazione crittografata in cui un mittente del messaggio crea successivi *livelli nidificati di crittografia* che vengono "sbucciati" da ciascun

nodo intermedio, fino a quando il livello più interno non viene consegnato al destinatario previsto. Il nome "onion routing" descrive questo uso della crittografia a più livelli che viene aperta uno strato alla volta, come la buccia di una cipolla.

Ciascuno dei nodi intermedi può solo "sbucciare" un livello e vedere chi è il prossimo nel percorso di comunicazione. L'onion routing garantisce che nessuno, tranne il mittente, conosca la destinazione o la lunghezza del percorso di comunicazione. Ogni intermediario conosce solo l'hop precedente e successivo.

Lightning Network utilizza un'implementazione del protocollo di onion routing basato su *Sphinx*,[1] sviluppato nel 2009 da George Danezis e Ian Goldberg.

L'implementazione dell'onion routing su LN è definita in BOLT #4: Onion Routing Protocol.

Un Esempio Fisico che Illustra l'Onion Routing

Esistono molti modi per descrivere l'onion routing, ma uno dei più semplici è utilizzare l'equivalente fisico delle buste sigillate. Una busta rappresenta un livello di crittografia, che consente solo al destinatario indicato di aprirla e leggerne il contenuto.

Diciamo che Alice vuole inviare una lettera segreta a Dina indirettamente tramite alcuni intermediari.

Selezione di un Percorso

Lightning Network utilizza il *source routing* (origine, il che significa che il percorso di pagamento viene selezionato e specificato solo dal mittente. In questo esempio, la lettera segreta di Alice a Dina è l'equivalente di un pagamento. Per assicurarsi che la lettera raggiunga Dina, Alice crea un percorso da lei a Dina, usando Bob e Chan come intermediari.

SUGGERIMENTO	Ci possono essere molti percorsi che permettono ad Alice di raggiungere Dina. Spiegheremo il processo di selezione del percorso ottimale nel Capitolo 12. Per ora, supponiamo che il percorso scelto da Alice utilizzi Bob e Chan come intermediari per arrivare a Dina.

Come promemoria, il percorso selezionato da Alice è mostrato nella Figura 10-2.

Figura 10-2. Percorso: da Alice a Bob a Chan a Dina

Vediamo come Alice può utilizzare questo percorso senza rivelare informazioni agli intermediari Bob e Chan.

Instradamento Basato sull'Origine

Il routing basato sull'origine non è il modo in cui i pacchetti vengono generalmente instradati su Internet oggi, sebbene il source routing fosse possibile sin dai primi giorni. L'instradamento su Internet si basa sulla *commutazione dei pacchetti* in ciascun nodo di instradamento intermedio. Un pacchetto IPv4, ad esempio, include gli indirizzi IP del mittente e del destinatario e ogni altro nodo di routing IP decide come inoltrare ciascun pacchetto verso la destinazione. Tuttavia, la mancanza di privacy in un tale meccanismo di instradamento, in cui ogni nodo intermedio vede il mittente e il destinatario, rende questa scelta sbagliata per l'utilizzo in una rete di pagamento.

Costruzione dei Livelli

Alice inizia scrivendo una lettera segreta a Dina. Quindi sigilla la lettera all'interno di una busta e scrive "A Dina" all'esterno (vedi la Figura 10-3). La busta rappresenta la crittografia con la chiave pubblica di Dina in modo che solo Dina possa aprire la busta e leggere la lettera.

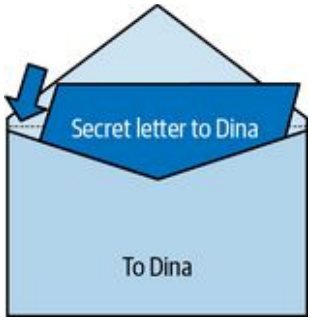

Figura 10-3. La lettera segreta di Dina, sigillata in una busta

La lettera di Dina sarà consegnata a Dina da Chan, che è immediatamente prima di Dina nel "percorso". Quindi, Alice mette la busta di Dina all'interno di una busta indirizzata a Chan (vedi Figura 10-4). L'unica parte che Chan può leggere è la destinazione (istruzioni di routing/instradamento): "A Dina". Sigillarlo all'interno di una busta indirizzata a Chan rappresenta la crittografia con la chiave pubblica di Chan in modo che solo Chan possa leggere l'indirizzo della busta. Chan non riesce ancora ad aprire la busta di Dina. Tutto ciò che vede sono le istruzioni all'esterno (l'indirizzo).

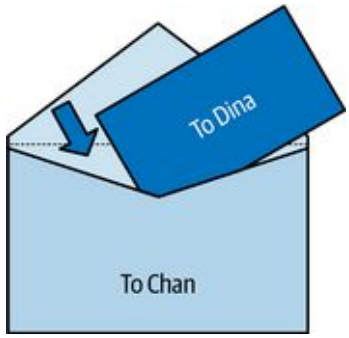

Figura 10-4. La busta di Chan, contenente la busta sigillata di Dina

Ora, questa lettera sarà consegnata a Chan da Bob. Quindi Alice la mette in una busta indirizzata a Bob (vedi Figura 10-5). Come prima, la busta rappresenta un messaggio cifrato per Bob che solo Bob può leggere. Bob può solo leggere l'esterno della busta di Chan (l'indirizzo), quindi sa di doverlo inviare a Chan.

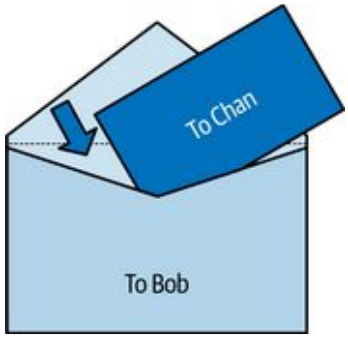

Figura 10-5. La busta di Bob, contenente la busta sigillata di Chan

Ora, se potessimo guardare attraverso le buste (con i raggi X!) vedremmo le buste annidate una dentro l'altra, come mostrato in Figura 10-6.

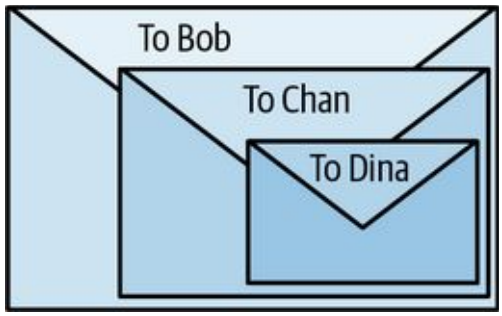

Figura 10-6. Buste annidate

Sbucciamento degli Strati

Alice ora ha una busta con la scritta "A Bob" all'esterno. Rappresenta un messaggio crittografato che solo Bob può aprire (decrittografare). Alice ora inizierà il processo inviandolo a Bob. L'intero processo è mostrato nella Figura 10-7.

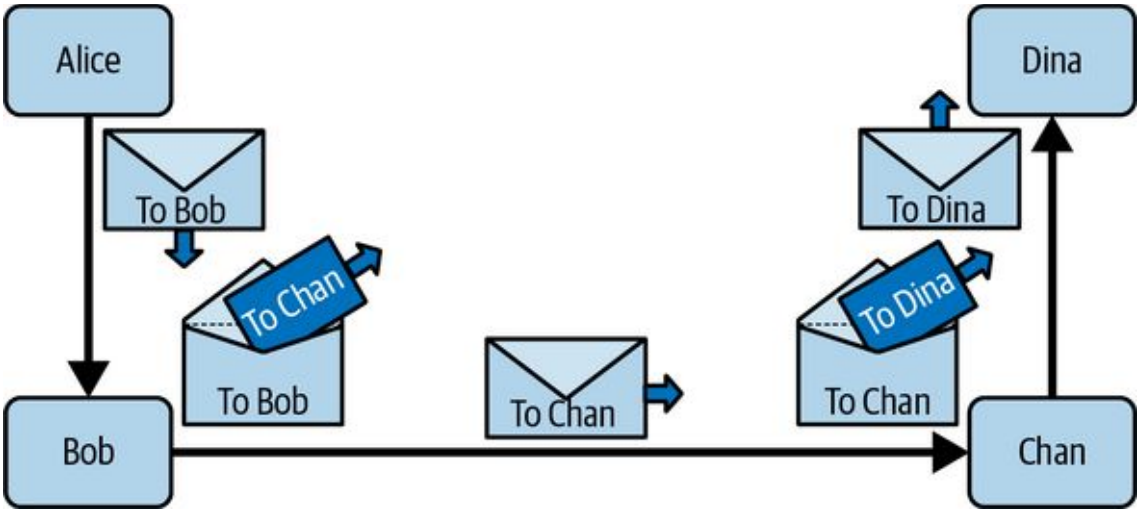

Figura 10-7. Invio delle buste

Come puoi vedere, Bob riceve la busta da Alice. Sa che proviene da Alice, ma non sa se Alice è il mittente originale o solo qualcuno che inoltra le buste. La apre per trovare una busta all'interno che dice "A Chan". Poiché tale è indirizzata a Chan, Bob non può aprirla. Non sa cosa c'è dentro e non sa se Chan sta ricevendo una lettera o un'altra busta da inoltrare. Bob non sa se Chan è il destinatario finale o meno. Bob inoltra la busta a Chan.

Chan riceve la busta da Bob. Non sa che proviene da Alice. Non sa se Bob è un intermediario o il mittente di una lettera. Chan apre la busta e trova un'altra busta all'interno indirizzata "A Dina", che non può aprire. Chan la inoltra a Dina, non sapendo se Dina è il destinatario finale.

Dina riceve una busta da Chan. Aprendola trova una lettera all'interno, quindi ora sa di essere il destinatario previsto di questo messaggio. Legge la lettera, sapendo che nessuno degli intermediari sa da dove proviene e nessun altro ha letto la sua lettera segreta!

Questa è l'essenza dell'onion routing. Il mittente avvolge un messaggio in livelli, specificando esattamente come verrà instradato e impedendo a qualsiasi intermediario di ottenere informazioni sul percorso o sul payload. Ogni intermediario rimuove un livello, vede solo un indirizzo di inoltro e non conosce nient'altro che l'hop precedente e successivo nel percorso.

Ora, diamo un'occhiata ai dettagli dell'implementazione del'onion routing su LN.

Introduzione all'Onion Routing di HTLC

L'onion routing su Lightning Network sembra complesso a prima vista, ma una volta compreso il concetto di base, è davvero molto semplice.

Da un punto di vista pratico, Alice sta dicendo a ogni nodo intermedio quale HTLC impostare con il nodo successivo nel percorso.

Il primo nodo, che è il mittente del pagamento, Alice nel nostro esempio, è chiamato *nodo di origine*. L'ultimo nodo, che è il destinatario del pagamento, Dina nel nostro esempio, è chiamato *nodo finale*.

Ogni nodo intermedio, Bob e Chan nel nostro esempio, è chiamato hop. Ogni hop deve impostare un *HTLC in uscita* per l'hop successivo. Le informazioni comunicate a ogni hop da Alice sono chiamate *hop payload* o *hop data*. Il messaggio che viene instradato da Alice a Dina è chiamato *onion* ed è costituito da messaggi *hop payload* o *hop data* crittografati su ciascun hop.

Ora che conosciamo la terminologia utilizzata nell'onion routing di Lightning Network, riformuliamo il compito di Alice: Alice deve costruire una cipolla (onion) con dati hop, dicendo a ogni hop come costruire un HTLC in uscita per inviare un pagamento al nodo finale (Dina).

Alice Seleziona il Percorso

Dal Capitolo 8 sappiamo che Alice invierà un pagamento di 50.000 satoshi a Dina tramite Bob e Chan. Questo pagamento viene trasmesso tramite una serie di HTLC, come mostrato in Figura 10-8.

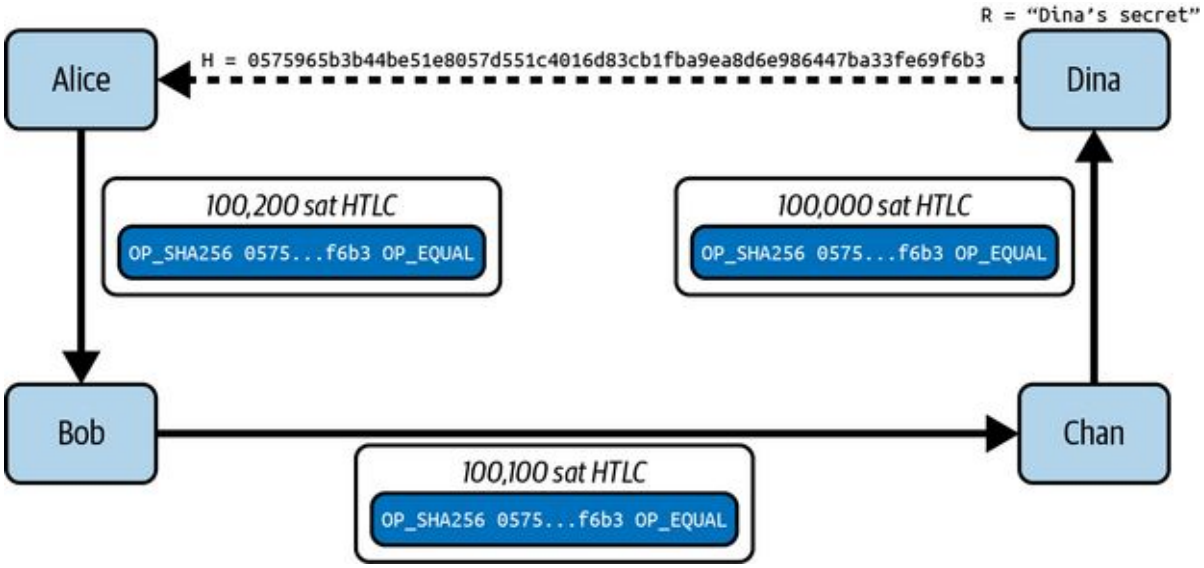

Figura 10-8. Percorso di pagamento con HTLC da Alice a Dina

Come vedremo nel Capitolo 11, Alice è in grado di costruire questo percorso verso Dina perché i nodi Lightning annunciano i loro canali all'intero Lightning Network utilizzando il Lightning Gossip Protocol. Dopo l'annuncio iniziale del canale, Bob e Chan hanno inviato ciascuno un ulteriore messaggio `channel_update` con la loro tariffa di instradamento e le aspettative di timelock per l'instradamento dei pagamenti.

Dagli annunci e dagli aggiornamenti, Alice conosce le seguenti informazioni sui canali tra Bob, Chan e Dina:

- Un `short_channel_id` (ID canale breve) per ogni canale, che Alice può utilizzare per fare riferimento al canale durante la costruzione del percorso
- Un `cltv_expiry_delta` (timelock delta), che Alice può aggiungere al tempo di scadenza per ogni HTLC
- `Fee_base_msat` e `fee_proportional_millionths`, che Alice utilizza per calcolare la tariffa di instradamento totale prevista da quel nodo per l'inoltro su quel canale.

In pratica, vengono scambiate anche altre informazioni, come gli HTLC più grandi (`htlc_maximum_msat`) e più piccoli (`htlc_minimum_msat`) trasportati da un canale, ma questi non vengono utilizzati direttamente durante la costruzione del percorso onion come lo sono i campi precedenti.

Queste informazioni vengono utilizzate da Alice per identificare i nodi, i canali, le tariffe e i timelock per il seguente percorso dettagliato, mostrato nella Figura 10-9.

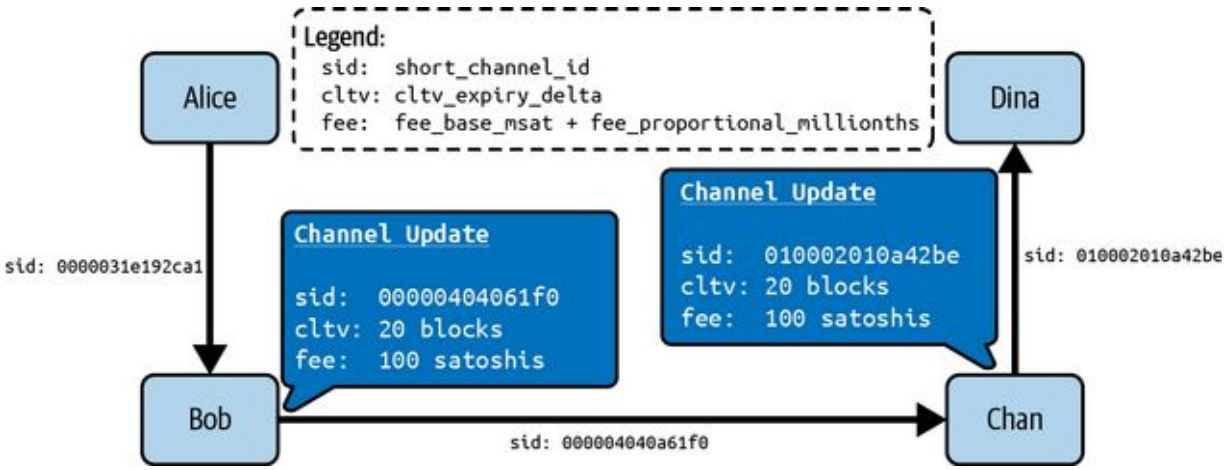

Figura 10-9. Un percorso dettagliato costruito da gossip di informazioni su canali e nodi

Alice conosce già il proprio canale per Bob e quindi non ha bisogno di queste informazioni per costruire il percorso. Nota anche che Alice non aveva bisogno di un aggiornamento del canale da Dina perché ha l'aggiornamento da Chan per l'ultimo canale nel percorso.

Alice Costruisce i Payload

Esistono due possibili formati che Alice può utilizzare per le informazioni comunicate a ciascun hop: un formato legacy a lunghezza fissa chiamato *hop data* e un formato più flessibile basato su Type-Length-Value (TLV) chiamato *hop payload*. Il formato del messaggio TLV verrà spiegato in dettaglio successivamente. Per ora, ci basta sapere che offre flessibilità consentendo di aggiungere campi al protocollo a piacere.

NOTA	Entrambi i formati sono specificati in BOLT #4: Onion Routing Protocol, Packet Structure.

Alice inizierà a costruire i dati del payload dalla fine del percorso all'indietro: Dina, Chan, poi Bob.

Payload del nodo finale per Dina

Alice costruisce prima il payload che verrà consegnato a Dina. Dina non costruirà un "HTLC in uscita", perché Dina è il nodo finale e il destinatario del pagamento. Per questo motivo, il payload per Dina è diverso da tutti gli altri (usa tutti zeri per `short_channel_id`), ma solo Dina lo saprà perché sarà crittografato nello strato più interno della cipolla. Essenzialmente, questa è la "lettera segreta a Dina" che abbiamo visto nel nostro esempio di busta fisica.

L'hop payload per Dina deve corrispondere alle informazioni nella fattura generata da Dina per Alice e conterrà (almeno) i seguenti campi in formato TLV:

`amt_to_forward`

> L'importo di questo pagamento in millisatoshi. Se si tratta solo di una parte di un pagamento in più parti, l'importo è inferiore al totale. In caso contrario, si tratta di un unico pagamento completo ed è pari all'importo della fattura e al valore `total_msat`.

`outgoing_cltv_value`

> Il timelock di scadenza del pagamento impostato sul valore `min_final_cltv_expiry` nella fattura.

`payment_secret`

> Uno speciale valore segreto a 256 bit dalla fattura, che consente a Dina di riconoscere questo pagamento in entrata. Ciò impedisce anche una classe di attacchi di probing che in precedenza rendeva insicure le fatture di valore zero. Il probing da parte dei nodi intermedi è mitigato poiché questo valore è crittografato solo per il destinatario, il che significa che non possono ricostruire un pacchetto finale che "sembra" legittimo.

`total_msat`

> L'importo totale corrispondente alla fattura. Questo può essere omesso se c'è solo una parte, nel qual caso si presume che corrisponda a `amt_to_forward` e deve essere uguale all'importo della fattura.

La fattura che Alice ha ricevuto da Dina specificava l'importo di 50.000 satoshi, ovvero

50.000.000 di millisatoshi. Dina ha specificato la scadenza minima per il pagamento `min_final_cltv_expiry` in 18 blocchi (3 ore, dati 10 minuti in media dei blocchi Bitcoin). Nel momento in cui Alice sta tentando di effettuare il pagamento, diciamo che la blockchain di Bitcoin ha registrato 700.000 blocchi. Quindi Alice deve impostare `outgoing_cltv_value` su un'altezza minima del blocco di 700.018.

Alice costruisce il payload hop per Dina come segue:

```
amt_to_forward : 50,000,000
outgoing_cltv_value: 700,018
payment_secret: fb53d94b7b65580f75b98f10...03521bdab6d519143cd521d1b3826
total_msat: 50,000,000
```

Alice lo serializza in formato TLV, come mostrato (semplificato) nella Figura 10-10.

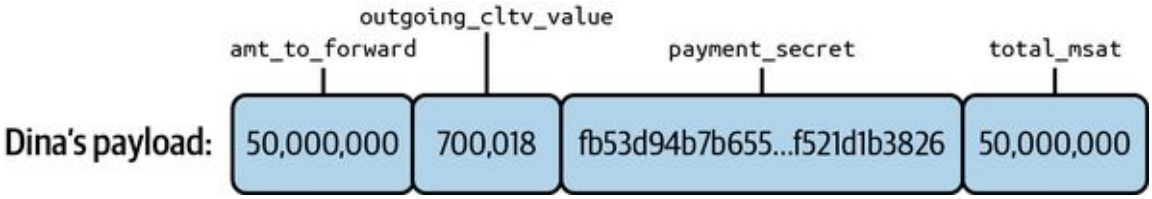

Figura 10-10. Il payload di Dina è costruito da Alice

Hop payload per Chan

Successivamente, Alice costruisce l'hop payload per Chan. Questo dirà a Chan come impostare un HTLC in uscita per Dina.

L'hop payload per Chan include tre campi: `short_channel_id`, `amt_to_forward` e `outgoing_cltv_value`:

```
short_channel_id: 010002010a42be
amt_to_forward: 50,000,000
outgoing_cltv_value: 700,018
```

Alice serializza questo payload in formato TLV, come mostrato (semplificato) nella Figura 10-11.

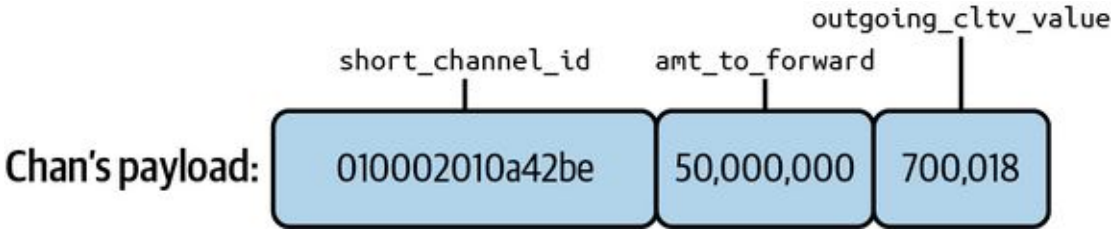

Figura 10-11. Il payload di Chan è costruito da Alice

Hop payload per Bob

Infine, Alice costruisce l'hop payload per Bob, che contiene anche gli stessi tre campi del payload hop per Chan, ma con valori diversi:

```
short_channel_id: 000004040a61f0
amt_to_forward: 50,100,000
outgoing_cltv_value: 700,038
```

Come puoi vedere, il campo `amt_to_forward` è 50.100.000 millisatoshi o 50.100 satoshi. Questo perché Chan si aspetta una commissione di 100 satoshi per instradare un pagamento a Dina. Affinché Chan possa "guadagnare" quella tariffa di instradamento, l'HTLC in entrata di Chan deve essere di 100 satoshi in più rispetto all'HTLC in uscita di Chan. Poiché l'HTLC in entrata di Chan è l'HTLC in uscita di Bob, le istruzioni a Bob riflettono la quota che Chan guadagna. In termini semplici, a Bob deve essere detto di inviare 50.100 satoshi a Chan, in modo che Chan possa inviare 50.000 satoshi e tenere 100 satoshi.

Allo stesso modo, Chan si aspetta un delta timelock di 20 blocchi. Quindi l'HTLC in entrata di Chan deve scadere 20 blocchi dopo l'HTLC in uscita di Chan. Per raggiungere questo obiettivo, Alice dice a Bob di far scadere il suo HTLC in uscita a Chan all'altezza del blocco 700.038-20 blocchi dopo l'HTLC di Chan a Dina.

SUGGERIMENTO	Le commissioni e le aspettative delta timelock per un canale sono stabilite dalla differenza tra HTLC in entrata e in uscita. Poiché l'HTLC in entrata viene creato dal *nodo precedente*, la commissione e il delta timelock vengono impostati nel payload onion su quel nodo precedente. A Bob viene detto come realizzare un HTLC che soddisfi le aspettative di tariffa e timelock di Chan.

Alice serializza questo payload in formato TLV, come mostrato (semplificato) nella Figura 10-12.

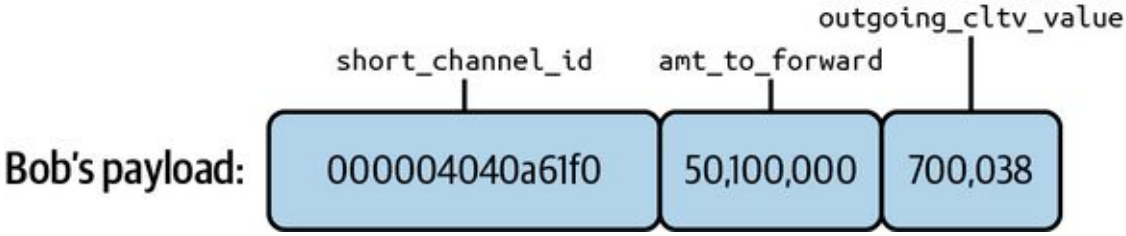

Figura 10-12. Il payload di Bob è costruito da Alice

Hop payload finalizzati

Alice ha ora costruito i tre hop payload che saranno avvolti in una cipolla. Una vista semplificata dei payload è mostrata nella Figura 10-13.

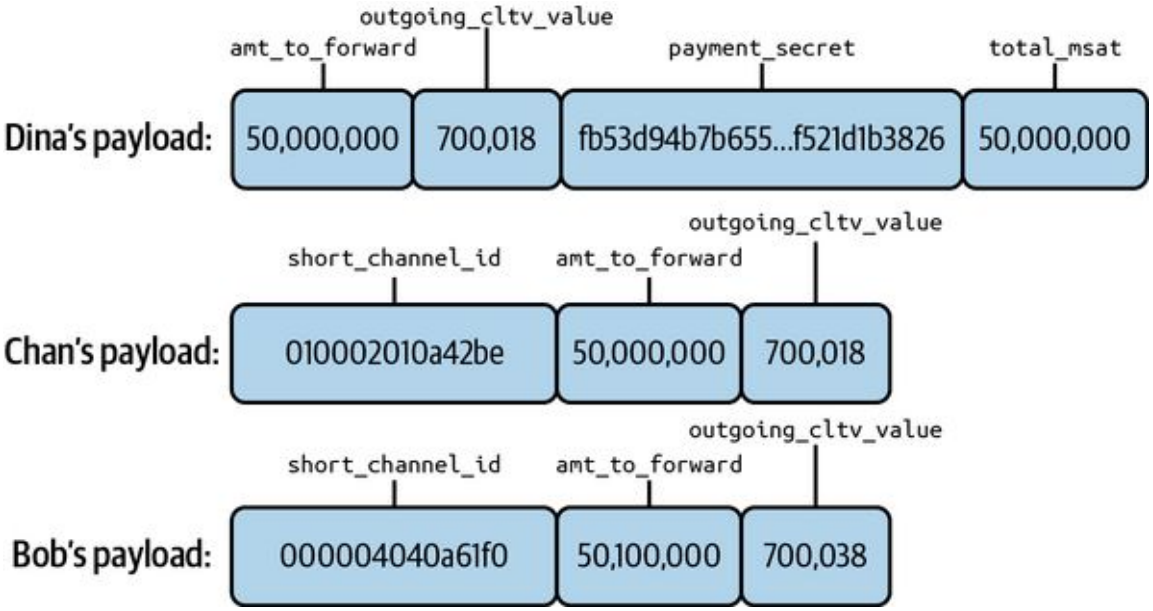

Figura 10-13. Hop payload per tutti i passaggi (hop)

Generazione delle Chiavi

Alice deve ora generare diverse chiavi che verranno utilizzate per crittografare i vari livelli della cipolla.

Con queste chiavi, Alice può raggiungere un elevato grado di privacy e integrità:

- Alice può crittografare ogni strato della cipolla in modo che solo il destinatario previsto possa leggerlo.
- Ogni intermediario può verificare che il messaggio non venga modificato.
- Nessuno sul percorso saprà chi ha inviato questa cipolla o dove sta andando. Alice non rivela la sua identità come mittente o l'identità di Dina come destinataria del pagamento.
- Ogni intermediario apprende solo il passaggio precedente e successivo.
- Nessuno può sapere quanto è lungo il percorso o dove si trova nel percorso.

ATTENZIONE	Come una cipolla tritata, i successivi dettagli tecnici possono far venire le lacrime agli occhi. Sentiti libero di saltare alla sezione successiva se ti senti perso. Torna in futuro su questo paragrafo e leggi BOLT #4: Onion Routing, Packet Construction, se vuoi saperne di più.

La base per tutte le chiavi utilizzate nella cipolla è un segreto condiviso che Alice e Bob possono entrambi generare indipendentemente utilizzando l'algoritmo Elliptic Curve Diffie-Hellman (ECDH). Dal segreto condiviso (shared secret - ss), possono generare indipendentemente quattro chiavi aggiuntive denominate `rho`, `mu`, `um` e `pad`:

`rho`

Utilizzato per generare un flusso di byte casuali da un cifrario a flusso (utilizzato come CSPRNG). Questi byte vengono utilizzati per crittografare/decrittografare il corpo del messaggio nonché i byte di riempimento zero durante l'elaborazione dei pacchetti Sphinx.

`mu`

Utilizzato nel codice di autenticazione dei messaggi basato su hash (HMAC) per la verifica di integrità/autenticità.

`ehm`

Utilizzato nella segnalazione degli errori.

pad

Utilizzato per generare byte di riempimento della cipolla a una lunghezza fissa.

La relazione tra le varie chiavi e il modo in cui vengono generate è illustrata in Figura 10-14.

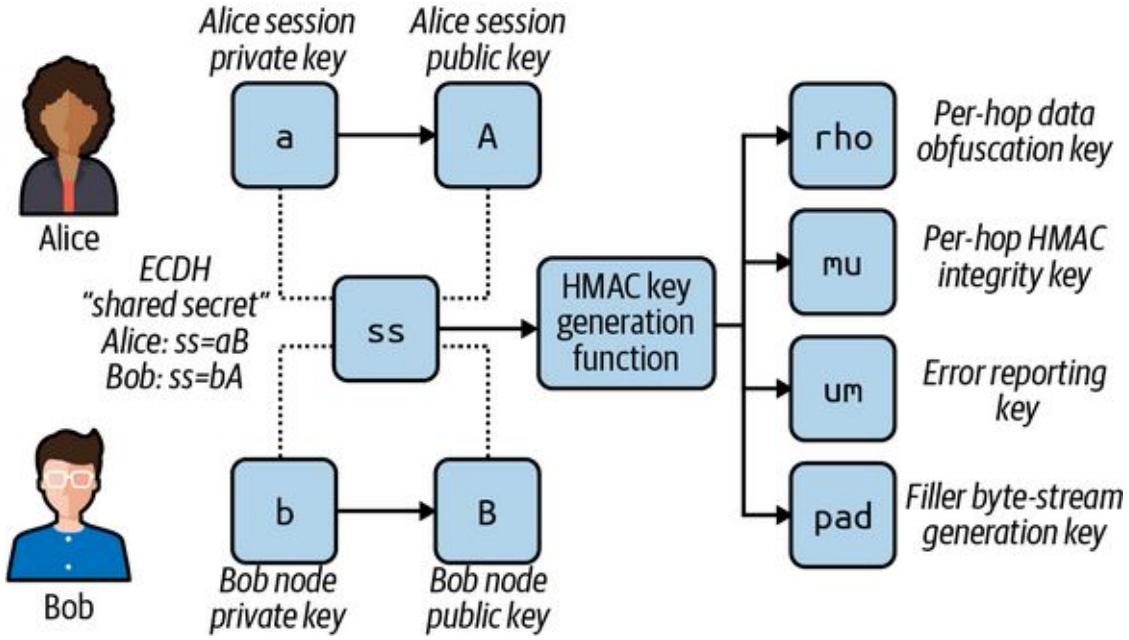

Figura 10-14. Generazione di chiavi onion

La chiave di sessione di Alice

Per evitare di rivelare la sua identità, Alice non usa la chiave pubblica del proprio nodo per costruire la cipolla. Invece, Alice crea una chiave temporanea a 32 byte (256 bit) denominata *chiave privata di sessione* e corrispondente *chiave pubblica di sessione*. Questo funge da "identità" temporanea e chiave *solo per questa cipolla*. Da questa chiave di sessione, Alice costruirà tutte le altre chiavi che verranno utilizzate in questa cipolla.

Dettagli sulla generazione delle chiavi

La generazione di chiavi, la generazione di byte casuali, le chiavi temporanee e il modo in cui vengono utilizzate nella costruzione dei pacchetti sono specificate in tre sezioni di BOLT #4:

- Key Generation
- Random Byte Stream
- Packet Construction

Per semplicità e per evitare di diventare troppo tecnici, non li abbiamo inclusi nel libro.

Generazione segreto condiviso

Un dettaglio importante che sembra quasi magico è la capacità di Alice di creare un *segreto condiviso* con un altro nodo semplicemente conoscendo le loro chiavi pubbliche. Questo si basa sull'invenzione dello scambio di chiavi Diffie-Hellman (DH) negli anni '70 che ha rivoluzionato la crittografia. L'onion routing di Lightning utilizza Elliptic Curve Diffie–Hellman (ECDH) sulla curva `secp256k1` di Bitcoin.

Spiegazione di Diffie-Hellman sulla curva ellittica

Supponiamo che la chiave privata di Alice sia *a* e la chiave privata di Bob sia *b*. Utilizzando la curva ellittica, Alice e Bob moltiplicano ciascuno la propria chiave privata per il punto generatore *G* per produrre rispettivamente le proprie chiavi pubbliche *A* e *B*:

$$A = aS$$

$$B = bG$$

Ora Alice e Bob possono utilizzare *Elliptic Curve Diffie–Hellman Key Exchange* per creare un segreto condiviso *ss*, un valore che entrambi possono calcolare indipendentemente senza scambiarsi alcuna informazione

Il segreto condiviso *ss* viene calcolato da ciascuno moltiplicando la propria chiave privata con la chiave pubblica dell'altro, in modo tale che:

$$ss = aB = bA$$

Ma perché queste due moltiplicazioni darebbero lo stesso valore *ss*? Segui, mentre dimostriamo la matematica che dimostra che ciò è possibile:

ss

= aB

calcolato da Alice che conosce sia *a* (la sua chiave privata) che *B* (la chiave pubblica di Bob)

= a(bG)

poiché sappiamo che *B = bG*, sostituiamo

= (ab)G

a causa dell'associatività, possiamo spostare le parentesi

= (b)G

perché *xy = yx* (la curva è un gruppo abeliano)

= b(aG)

a causa dell'associatività, possiamo spostare le parentesi

= bA

e possiamo sostituire *aG* con *A*.

Il risultato *bA* può essere calcolato indipendentemente da Bob che conosce *b* (la sua chiave privata) e *A* (la chiave pubblica di Alice).

Abbiamo quindi dimostrato che:

ss = aB (Alice può calcolarlo)

ss = bA (Bob può calcolarlo)

Pertanto, ognuno può calcolare in modo indipendente *ss* che può utilizzare come chiave condivisa per crittografare simmetricamente i segreti tra loro due senza comunicare il segreto condiviso.

Una caratteristica unica di Sphinx come formato di pacchetto mix-net è che invece di includere una chiave di sessione distinta per ogni salto nel percorso, che aumenterebbe notevolmente la dimensione del pacchetto mix-net, viene utilizzato uno schema intelligente di *accecamento* (*blinding*) per randomizzare in modo deterministico la chiave di sessione ad ogni hop.

In pratica, questo piccolo accorgimento ci permette di mantenere il pacchetto di cipolle il più

compatto possibile pur conservando le proprietà di sicurezza desiderate.

La chiave di sessione per l'hop i viene derivata utilizzando la chiave pubblica del nodo e il segreto condiviso derivato dell'hop `i – 1`:

```
session_key_i = session_key_{i-1} * SHA-256(node_pubkey_{i-1} ||
shared_secret_{i-1})
```

In altre parole, prendiamo la chiave di sessione dell'hop precedente e la moltiplichiamo per un valore derivato dalla chiave pubblica e dal segreto condiviso derivato per quell'hop.

Poiché la moltiplicazione della curva ellittica può essere eseguita su una chiave pubblica senza conoscere la chiave privata, ogni hop è in grado di ri-randomizzare la chiave di sessione per l'hop successivo in modo deterministico.

Il creatore del pacchetto onion conosce tutti i segreti condivisi (poiché ha crittografato il pacchetto in modo univoco per ogni hop) e quindi è in grado di derivare tutti i fattori di blinding.

Questa conoscenza consente di ricavare tutte le chiavi di sessione utilizzate in anticipo durante la generazione dei pacchetti.

Nota che il primissimo hop utilizza la chiave di sessione originale generata perché questa chiave viene utilizzata per avviare il blinding della chiave di sessione da ogni hop successivo.

Avvolgere gli Strati della Cipolla

Il processo di confezionamento della cipolla è dettagliato in BOLT #4: Onion Routing, Packet Construction.

In questa sezione descriveremo questo processo ad un livello elevato e un po' semplificato, omettendo alcuni dettagli.

Cipolle a Lunghezza Fissa

Abbiamo accennato al fatto che nessuno dei nodi "hop" sa quanto è lungo o dove si trova nel percorso. Com'è possibile?

Se hai una serie di indicazioni stradali, anche se crittografate, non puoi dire quanto sei lontano dall'inizio o dalla fine semplicemente guardando *dove* ti trovi nella serie?

Il trucco utilizzato nell'onion routing è quello di rendere sempre il percorso (la serie di direzioni) della stessa lunghezza per ogni nodo. Ciò si ottiene mantenendo il pacchetto cipolla della stessa lunghezza ad ogni passaggio.

Ad ogni salto, l'hop payload appare all'inizio dell'onion payload, seguito da quelli che *sembrano essere* altri 19 payload. Ogni hop vede se stessi come il primo di 20 hop.

SUGGERIMENTO	Il payload della cipolla (onion payload) è di 1.300 byte. Ogni hop payload è di 65 byte o meno (riempito a 65 byte se inferiore). Il payload totale della cipolla può contenere 20 hop payload (20 × 65 = 1300). Il percorso di instradamento massimo della cipolla è quindi di 20 salti (hops).

Man mano che ogni strato viene "sbucciato", vengono aggiunti dati di riempimento (dati spazzatura) alla fine del payload della cipolla in modo che l'hop (passaggio) successivo ottenga una cipolla della stessa dimensione ed è sempre il "primo hop" della cipolla.

La dimensione della cipolla è di 1.366 byte, strutturata come mostrato in Figura 10-15.

1 byte

Un byte di versione

33 byte

Una chiave di sessione pubblica compressa da cui è possibile generare il segreto condiviso per hop senza rivelare l'identità di Alice

1.300 byte

L'onion payload effettivo contenente le istruzioni per ciascun hop

32 byte

Un checksum di integrità HMAC

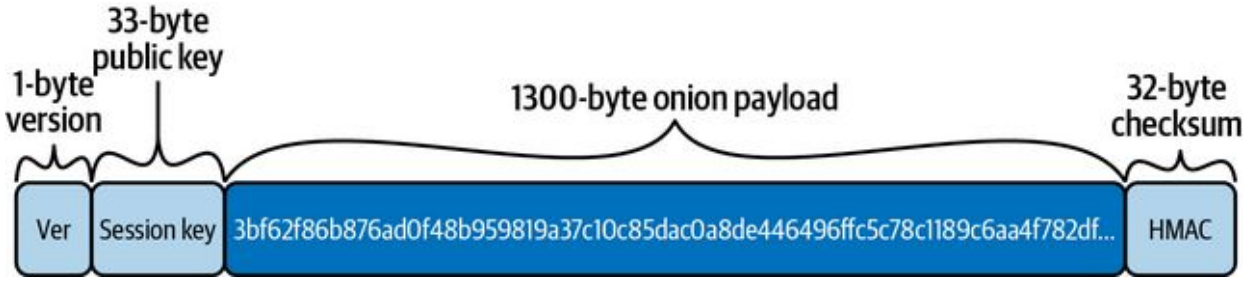

Figura 10-15. Il pacchetto onion / la cipolla

Avvolgere la Cipolla (passo per passo)

Descriveremo ora, passo per passo, il processo di avvolgimento della cipolla.

Per ogni hop, il mittente (Alice) ripete lo stesso processo:

1. Alice genera il segreto condiviso per-hop e le chiavi `rho`, `mu` e `pad`.
2. Alice genera 1.300 byte di riempimento e riempie il campo dell'onion payload di 1.300 byte con il tale.
3. Alice calcola l'HMAC per il payload dell'hop (zero per l'hop finale).
4. Alice calcola la lunghezza dell'hop payload + HMAC + spazio per memorizzare la lunghezza stessa.
5. Alice *sposta a destra* l'onion payload secondo lo spazio necessario calcolato per adattarsi all'hop payload. I dati "di riempimento" più a destra vengono scartati, lasciando spazio sufficiente a sinistra per il payload.
6. Alice inserisce la lunghezza + hop payload + HMAC nella parte anteriore del campo payload nello spazio creato dallo spostamento del filler.
7. Alice utilizza la chiave `rho` per generare un one-time pad da 1.300 byte.
8. Alice offusca l'intero onion payload tramite XORing con i byte generati da `rho`.
9. Alice calcola l'HMAC dell'onion payload, utilizzando la chiave `mu`.
10. Alice aggiunge la chiave pubblica di sessione (in modo che l'hop possa calcolare il segreto condiviso).
11. Alice aggiunge il numero di versione.

12. Alice esegue nuovamente il blind deterministico della chiave di sessione utilizzando un valore derivato dall'hashing del segreto condiviso e della chiave pubblica dell'hop precedente.

Successivamente, Alice ripete il processo. Le nuove chiavi vengono calcolate, l'onion payload viene spostato (eliminando più spazzatura), il nuovo hop payload viene aggiunto in primo piano e l'intero onion payload viene crittografato con il flusso di byte `rho` per l'hop successivo.

Per l'hop finale, l'HMAC incluso nel passaggio n. 3, corrisponde a *zero*. L'hop finale utilizza questo segnale per determinare che si tratta effettivamente dell'ultimo hop del percorso. In alternativa, può essere utilizzato anche il fatto che lo `short_chan_id` incluso nel payload per denotare il "next hop" sia zero.

Nota che in ogni fase la chiave `mu` viene utilizzata per generare un HMAC sul pacchetto onion *crittografato* (dal punto di vista del nodo che elabora il payload), nonché sul contenuto del pacchetto con un singolo livello di crittografia rimosso. Questo HMAC esterno consente al nodo che elabora il pacchetto di verificare l'integrità del pacchetto onion (che nessun byte sia stato modificato). L'HMAC interno viene quindi rivelato durante l'inverso della routine "shift and encrypt" descritta in precedenza, che funge da HMAC *esterno* per l'hop successivo.

Avvolgimento dell'Hop Payload di Dina

Come promemoria, la cipolla viene avvolta partendo dalla fine del percorso, da Dina, il nodo o destinatario finale. Quindi il percorso viene ricostruito al contrario fino al mittente, Alice.

Alice inizia con un campo vuoto di 1.300 byte, l'onion payload di lunghezza fissa. Quindi riempie l'onion payload con un "riempimento" di flusso di byte pseudocasuale generato dalla chiave `pad`.

Ciò è mostrato in Figura 10-16.

NOTA	La generazione di flussi di byte casuali utilizza l'algoritmo ChaCha20 come generatore di numeri pseudocasuali crittografici sicuri (CSPRNG). Tale algoritmo genererà un flusso deterministico, lungo e non ripetitivo di byte apparentemente casuali da un seme iniziale. I dettagli sono specificati in BOLT #4: Onion Routing, Pseudo Random Byte Stream.

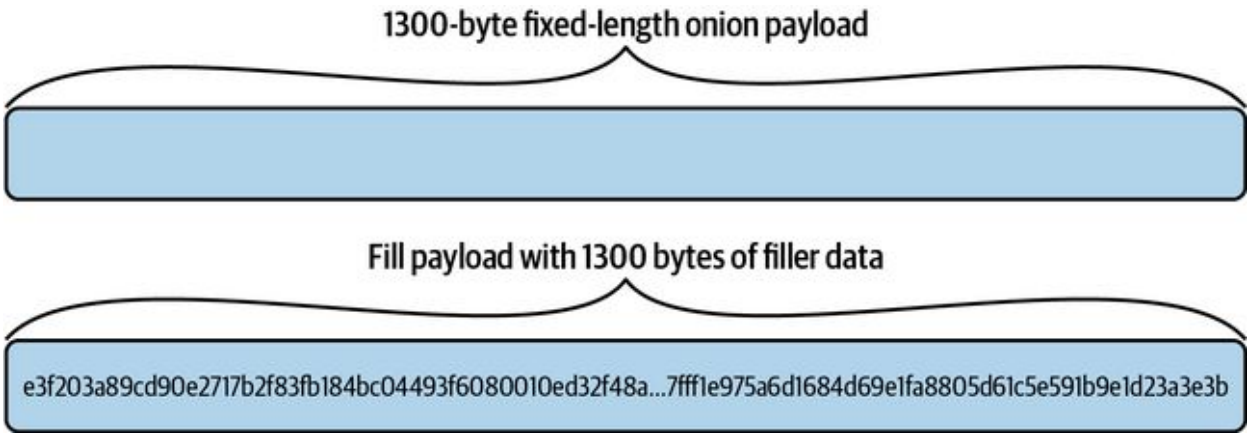

Figura 10-16. Riempimento delll'onion payload con un flusso di byte casuale

Alice ora inserirà l'hop payload di Dina nel lato sinistro dell'array da 1.300 byte, spostando il riempimento a destra e scartando il restante. Ciò è visualizzato nella Figura 10-17.

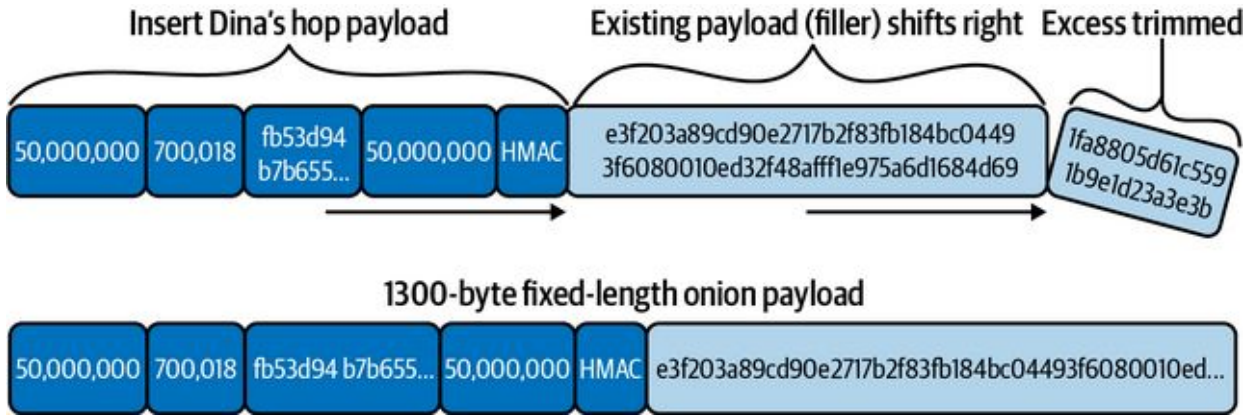

Figura 10-17. Aggiunta delll'hop payload di Dina

Alice misura la lunghezza dell'hop payload di Dina, sposta il riempimento a destra per creare uno spazio uguale nella parte sinistra dell'onion payload e inserisce il payload di Dina in quello spazio.

Nella riga successiva vediamo il risultato: il payload onion da 1.300 byte contiene il payload hop di Dina e il flusso di byte di riempimento che riempie il resto dello spazio disponibile.

Successivamente, Alice offusca l'intero payload della cipolla in modo che *solo Dina* possa leggerlo.

Per fare ciò, Alice genera un flusso di byte utilizzando la chiave `rho` (che anche Dina conosce). Alice utilizza una disgiunzione esclusiva "o" bit per bit (XOR) tra i bit dell'onion payload e il flusso di byte creato da `rho`. Il risultato appare come un flusso di byte casuale (o crittografato) della lunghezza di 1.300 byte. Questo passaggio è mostrato nella Figura 10-18.

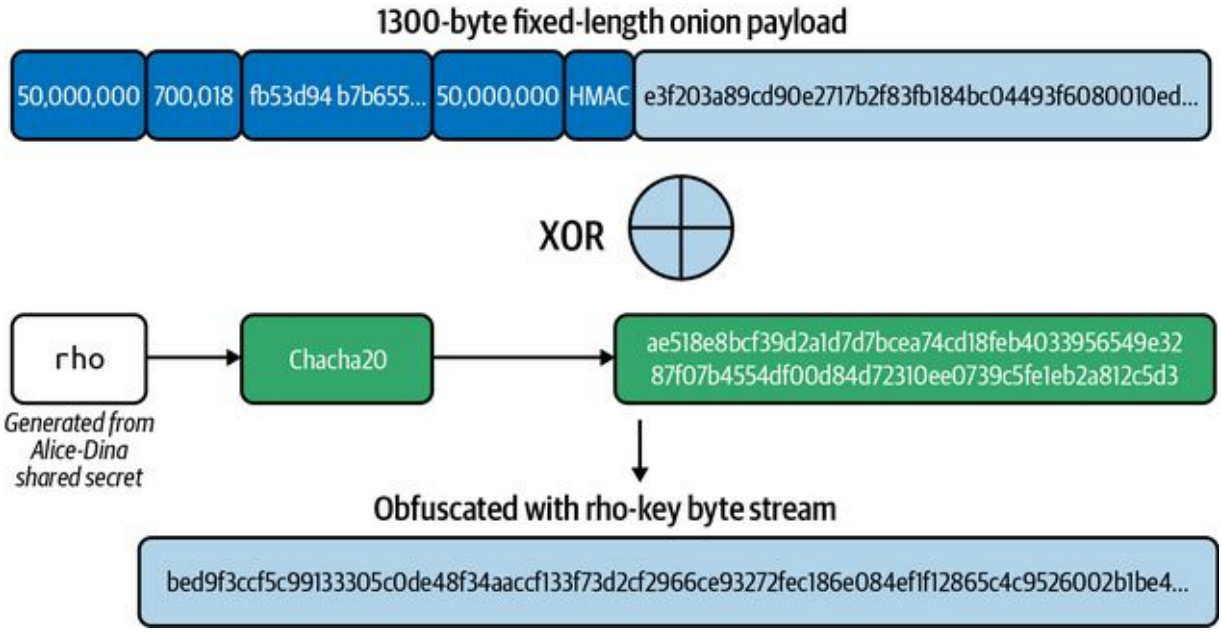

Figura 10-18. Offuscamento del payload della cipolla

Una delle proprietà di XOR è che se lo fai due volte, torni ai dati originali. Come vedremo tra poco, se Dina applica la stessa operazione XOR con il flusso di byte generato da `rho`, rivelerà il payload originale della cipolla.

NOTA	XOR è una funzione *involutoria*, il che significa che se viene applicata due volte, si annulla da sola. Nello specifico XOR(XOR(*a*, *b*), *b*) = *a*. Questa proprietà è ampiamente utilizzata nella crittografia a chiave simmetrica.

Poiché solo Alice e Dina hanno la chiave `rho` (derivata dal segreto condiviso di Alice e Dina), solo loro possono farlo. Ciò crittografa il payload della cipolla solo per gli occhi di Dina.

Infine, Alice calcola un codice di autenticazione dei messaggi basato su hash (HMAC) per il payload di Dina, che utilizza la chiave `mu` come chiave di inizializzazione. Questo è mostrato nella Figura 10-19.

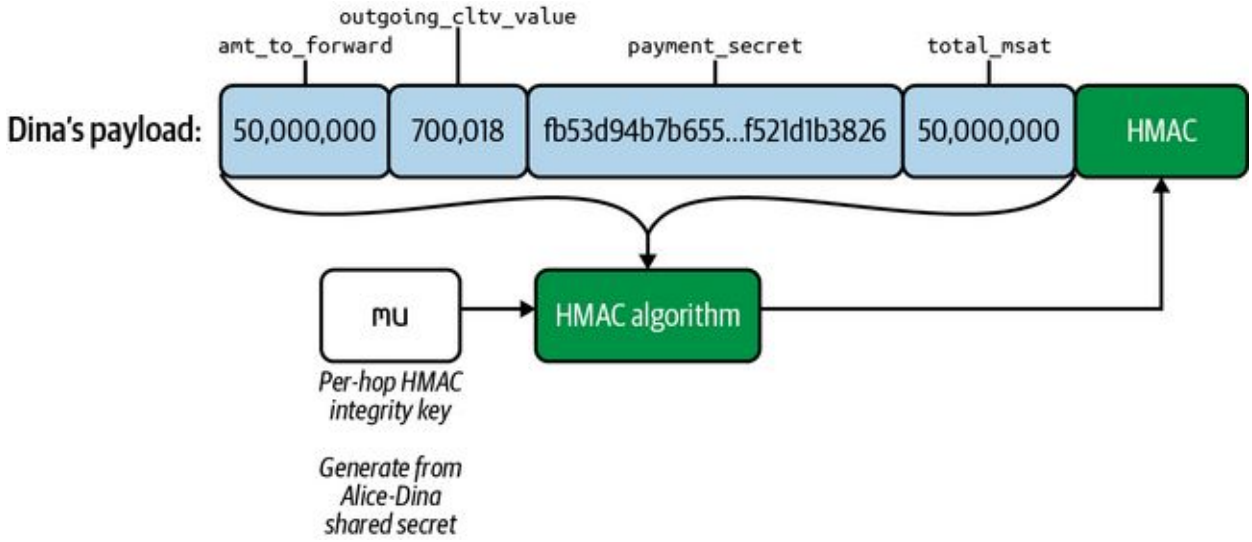

Figura 10-19. Aggiunta di un checksum di integrità HMAC all' hop payload di Dina

Protezione e rilevamento del replay dell'onion routing

L'HMAC funge da checksum e aiuta Dina a verificare l'integrità dell'hop payload. L'HMAC a 32 byte viene aggiunto all'hop payload di Dina. Nota che calcoliamo l'HMAC sui dati *crittografati* piuttosto che sui dati in chiaro. Questo è noto come *encrypt-then-mac* ed è il modo consigliato per utilizzare un MAC, in quanto fornisce sia l'integrità del testo normale *che* quella del testo cifrato.

La moderna crittografia autenticata consente inoltre di autenticare anche l'uso di un set facoltativo di byte di testo in chiaro, noto come *dati associati*. In pratica, di solito si tratta di qualcosa di simile all'intestazione di un pacchetto in testo normale o ad altre informazioni ausiliarie. Includendo questi dati associati nel payload da autenticare, il verificatore del MAC garantisce che questi dati associati non siano stati manomessi (ad esempio, scambiando l'intestazione di testo normale su un pacchetto crittografato).

Nel contesto di Lightning Network, questi dati associati vengono utilizzati per *rafforzare* la protezione dalla riproduzione (replay) di questo schema. Come impareremo di seguito, la protezione dalla riproduzione garantisce che un utente malintenzionato non possa

ritrasmettere (riprodurre) un pacchetto nella rete e osservarne il percorso risultante. Invece, i nodi intermedi sono in grado di utilizzare le misure di protezione dalla riproduzione per rilevare e rifiutare un pacchetto riprodotto. Il formato del pacchetto Sphinx di base utilizza un registro di tutte le chiavi segrete effimere utilizzate per rilevare i replay. Se una chiave segreta viene utilizzata di nuovo, il nodo può rilevarla e rifiutare il pacchetto.

La natura degli HTLC su Lightning Network ci consente di rafforzare ulteriormente la protezione del replay aggiungendo un ulteriore incentivo *economico*. Ricorda che l'hash di pagamento di un HTLC può essere utilizzato in sicurezza solo una volta (per un pagamento completo). Se un hash di pagamento viene utilizzato di nuovo e attraversa un nodo che ha già visto il segreto di pagamento per quell'hash, i nodi possono semplicemente ritirare i fondi e riscuotere l'intero importo del pagamento senza inoltrarlo! Possiamo utilizzare questo fatto per rafforzare la protezione del replay richiedendo che l'*hash di pagamento* sia incluso nel nostro calcolo HMAC come dato associato. Con questo passaggio aggiuntivo, anche il tentativo di riprodurre un pacchetto onion richiede al mittente di impegnarsi a utilizzare lo *stesso* hash di pagamento. Di conseguenza, oltre alla normale protezione del replay, un utente malintenzionato rischia anche di perdere l'intero importo dell'HTLC riprodotto.

Una considerazione relativa al set sempre crescente di chiavi di sessione archiviate per la protezione dal replay è: i nodi sono in grado di recuperare questo spazio? Nel contesto del Lightning Network, la risposta è: sì! Ancora una volta, grazie agli attributi unici del costrutto HTLC, possiamo apportare un ulteriore miglioramento rispetto al protocollo Sphinx di base. Dato che gli HTLC sono contratti *time-locked* basati sull'altezza assoluta del blocco, una volta che un HTLC è scaduto, il contratto è effettivamente chiuso definitivamente. Di conseguenza, i nodi possono utilizzare questa altezza di scadenza CLTV (operatore CHECKLOCKTIMEVERIFY) come indicatore per sapere quando è sicuro scartare una voce nel registro anti-replay.

Avvolgimento dell'Hop Payload di Chan

Nella Figura 10-20 vediamo i passaggi utilizzati per avvolgere l'hop payload di Chan nella cipolla. Questi sono gli stessi passaggi che Alice ha usato per avvolgere l'hop payload di Dina.

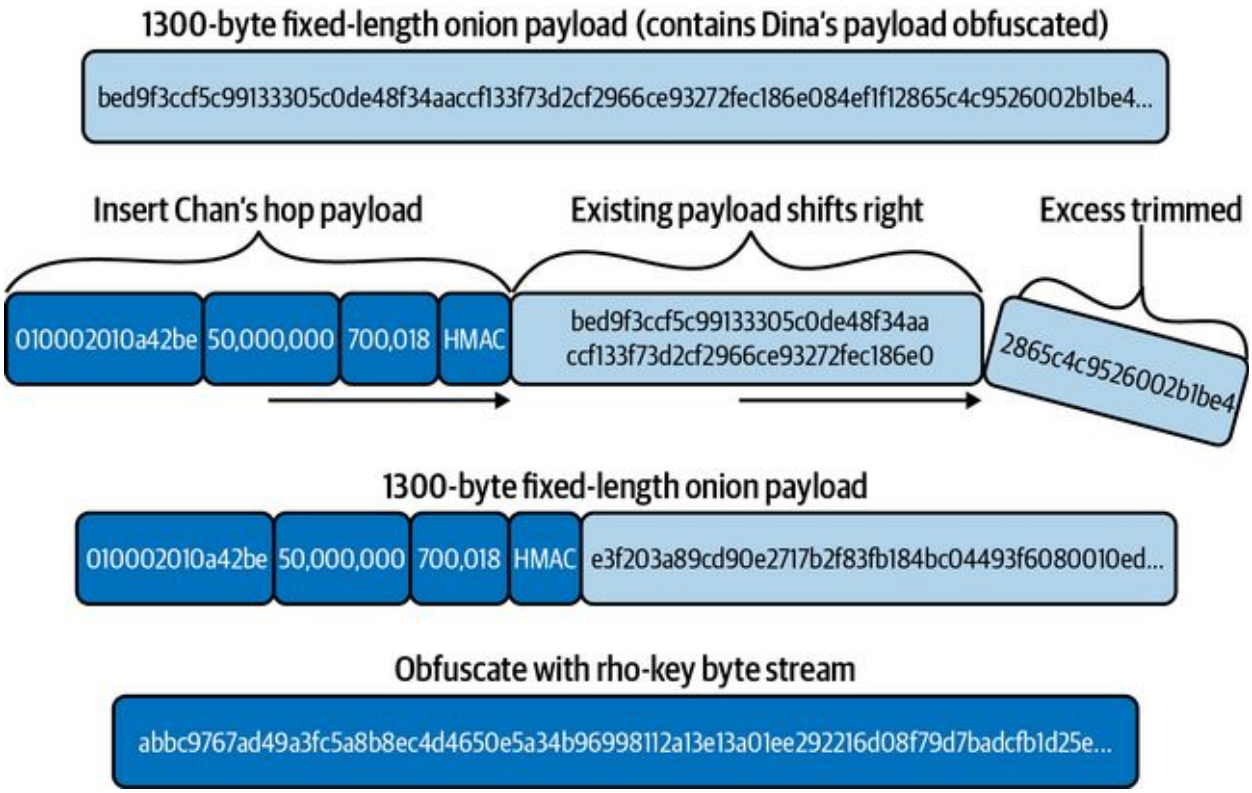

Figura 10-20. Avvolgimento della cipolla per Chan

Alice inizia con l'onion payload di 1.300 creato per Dina. I primi 65 (o meno) byte sono il payload offuscato di Dina offuscato e il resto è riempimento. Alice deve stare attenta a non sovrascrivere il payload di Dina.

Successivamente, Alice deve individuare la chiave pubblica temporanea (che è stata generata all'inizio di ogni hop) che verrà anteposta al pacchetto di instradamento in questo hop.

Ricorda che invece di includere una chiave pubblica effimera univoca (che il mittente e il nodo intermedio utilizzano in un'operazione ECDH per generare un segreto condiviso), Sphinx utilizza una singola chiave pubblica effimera che viene randomizzata in modo deterministico ad ogni hop.

Durante l'elaborazione del pacchetto, Dina utilizzerà il suo segreto condiviso e la sua chiave pubblica per derivare il valore di blinding (`b_dina`) e lo utilizzerà per ri-randomizzare la chiave pubblica effimera, in un'operazione identica a quella eseguita da Alice durante la costruzione iniziale del pacchetto.

Alice aggiunge un checksum HMAC interno al payload di Chan e lo inserisce nella "parte anteriore" (lato sinistro) del payload della cipolla, spostando il payload esistente a destra di una quantità uguale. Ricorda che ci sono effettivamente due HMAC utilizzati nello schema: l'HMAC esterno e l'HMAC interno. In questo caso, l'HMAC *interno* di Chan è in realtà l'HMAC *esterno* di Dina.

Ora il payload di Chan è nella parte anteriore della cipolla. Quando Chan lo vede, non ha idea di quanti payload siano arrivati prima o dopo. Sembra sempre il primo di 20 passaggi/hop!

Successivamente, Alice offusca l'intero payload tramite XOR con il flusso di byte generato dalla chiave Alice-Chan `rho`. Solo Alice e Chan hanno questa chiave `rho` e solo loro possono produrre il flusso di byte per offuscare e de-offuscare la cipolla. Infine, come abbiamo fatto nel passaggio precedente, calcoliamo l'HMAC esterno di Chan, che è quello che userà per verificare l'integrità del pacchetto onion crittografato.

Avvolgimento dell'Hop Payload di Bob

In Figura 10-21 vediamo i passaggi utilizzati per avvolgere l'hop payload di Bob nella cipolla.

Iniziamo con il l'onion payload (offuscato) contenente gli hop payload di Chan e Dina.

Otteniamo la chiave di sessione per questo hop derivata dal fattore di blinding generato dall'hop precedente. Include l'HMAC esterno dell'hop precedente come HMAC interno di questo hop. Inseriamo l'hop payload di Bob all'inizio e spostiamo tutto il resto a destra, togliendo una frazione uguale alle dimensioni dell'hop payload di Bob a partire dalla parte finale (erano comunque dati spazzatura di riempimento, quindi non ci preoccupa).

Offuschiamo l'intero XOR con la chiave `rho` dal segreto condiviso Alice-Bob in modo che solo Bob possa scartarlo.

Calcoliamo l'HMAC esterno e lo appendiamo all'estremità dell'hop payload di Bob.

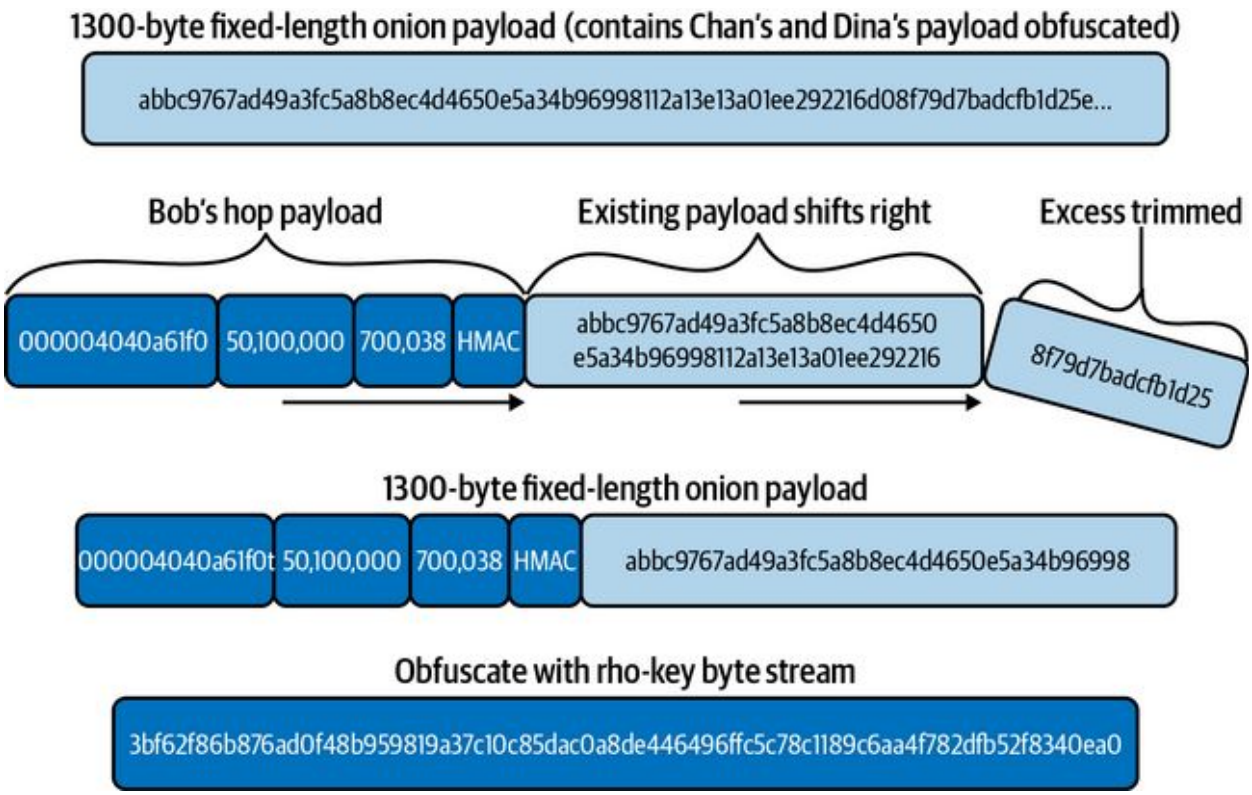

Figura 10-21. Avvolgimento della cipolla per Bob

Il Pacchetto Onion Finale

L'onion payload è pronto per essere inviato a Bob. Alice non ha bisogno di aggiungere altri hop payload.

Alice calcola quindi un HMAC per l'onion payload per proteggerlo crittograficamente con un checksum che Bob può verificare.

Successivamente Alice aggiunge una chiave di sessione pubblica a 33 byte che verrà utilizzata da ciascun hop per generare un segreto condiviso e le chiavi `rho`, `mu` e `pad`.

Infine Alice inserisce il numero di versione onion (attualmente `0`) all'inizio. Ciò consente futuri aggiornamenti del formato del pacchetto onion.

Il risultato può essere visto nella Figura 10-22.

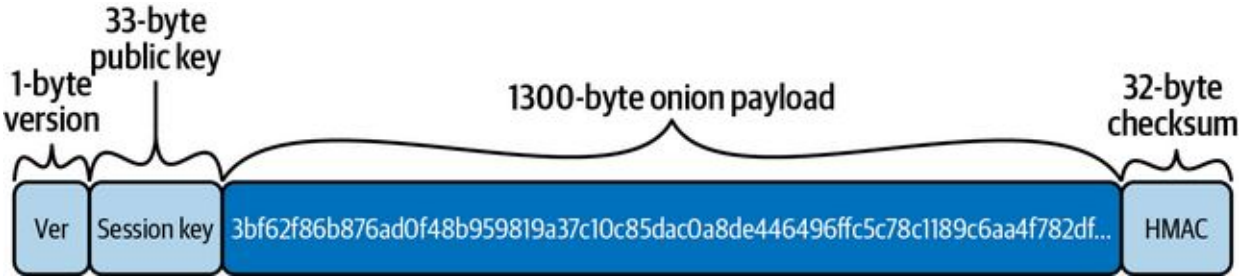

Figura 10-22. Il pacchetto onion finalizzato

Invio Della Cipolla

In questa sezione vedremo come viene inoltrato il pacchetto onion e come vengono distribuiti gli HTLC lungo il percorso.

Il Messaggio update_add_htlc

I pacchetti onion vengono inviati come parte del messaggio `update_add_htlc`. Nel Capitolo 9 abbiamo visto che i contenuti del messaggio `update_add_htlc` sono i seguenti:

```
[channel_id:channel_id]
[u64:id]
[u64:amount_msat]
[sha256:payment_hash]
[u32:cltv_expiry]
[1366*byte:onion_routing_packet]
```

Ricorderai che questo messaggio viene inviato da un partner di canale per chiedere all'altro partner di aggiungere un HTLC. È così che Alice chiederà a Bob di aggiungere un HTLC per pagare Dina. Ora capisci lo scopo dell'ultimo campo, `onion_routing_packet`, che è lungo 1.366 byte. È il pacchetto onion, la nostra cipolla completamente avvolta in più strati che abbiamo appena costruito!

Alice Manda la Cipolla a Bob

Alice invierà il messaggio `update_add_htlc` a Bob. Vediamo cosa contiene:

`channel_id`

Questo campo contiene l'ID del canale Alice-Bob, che nel nostro esempio è `0000031e192ca1` (rivedi la Figura 10-9).

`id`

L'ID di questo HTLC in questo canale, a partire da 0.

`amount_msat`

L'importo dell'HTLC: 50.200.000 millisatoshi.

`payment_hash`

L'hash di pagamento RIPEMD160(SHA-256):

`9e017f6767971ed7cea17f98528d5f5c0ccb2c71`.

`cltv_expiry`

Il timelock di scadenza per l'HTLC sarà 700.058. Alice aggiunge 20 blocchi alla scadenza impostata nel payload di Bob in base al `cltv_expiry_delta` negoziato da Bob.

`onion_routing_packet`

L'ultimo pacchetto onion che Alice ha costruito con tutti gli hop payload!

Bob Controlla la Cipolla

Come abbiamo visto nel Capitolo 9, Bob aggiungerà l'HTLC alle transazioni di impegno e aggiornerà lo stato del canale con Alice.

Bob sbuccia la cipolla che ha ricevuto da Alice:

1. Prende la chiave di sessione dal pacchetto onion e ne ricava il segreto condiviso Alice-

Bob.

2. Genera la chiave `mu` dal segreto condiviso e la utilizza per verificare il checksum HMAC del pacchetto onion.

Ora che Bob ha generato la chiave condivisa e verificato l'HMAC, può iniziare a scartare l'onion payload da 1.300 byte all'interno del pacchetto onion. L'obiettivo è che Bob recuperi il proprio hop payload e quindi inoltri la cipolla rimanente all'hop successivo.

Se Bob estrae e rimuove il suo hop payload, la cipolla rimanente non sarà di 1.300 byte, sarà più piccola! Quindi l'hop successivo saprà che non è il primo hop e sarà in grado di rilevare quanto è lungo il percorso. Per evitare ciò, Bob deve aggiungere più dati di riempimento per riempire la cipolla.

Bob Genera il Riempimento

Bob genera il riempimento in modo leggermente diverso rispetto ad Alice, ma seguendo lo stesso principio generale.

Innanzitutto, Bob estende l'onion payload di 1.300 byte e li riempie con valori uguali a `0`. Ora il pacchetto onion è lungo 2.600 byte, con la prima metà contenente i dati inviati da Alice e la metà successiva contenente zeri. Questa operazione è mostrata nella Figura 10-23.

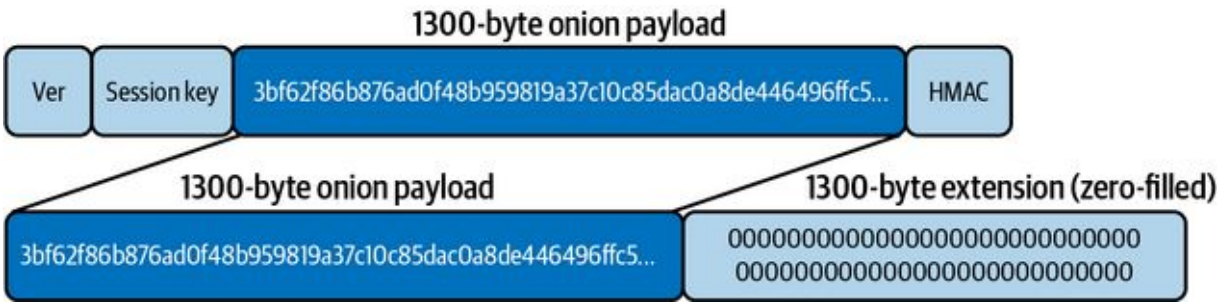

Figura 10-23. Bob estende l'onion payload di 1.300 byte (riempendolo di zeri).

Questo spazio vuoto diventerà offuscato e si trasformerà in "riempimento" mediante lo stesso processo che Bob utilizza per de-offuscare il proprio hop payload. Vediamo come funziona...

Bob De-Offusca il Suo Hop Payload

Successivamente Bob genererà la chiave `rho` dalla chiave condivisa Alice-Bob. La userà per generare un flusso di 2.600 byte, utilizzando l'algoritmo ChaCha20.

NOTA	I primi 1.300 byte del flusso di byte generato da Bob sono esattamente gli stessi di quelli generati da Alice utilizzando la chiave `rho`.

Successivamente Bob applica i 2.600 byte del flusso di byte `rho` all'onion payload di 2.600 byte con un'operazione XOR bit a bit.

I primi 1.300 byte verranno de-offuscati da questa operazione XOR, perché è la stessa operazione applicata da Alice e XOR è involutivo. Quindi Bob rivelerà il suo hop payload seguito da alcuni dati che sembrano criptati.

Allo stesso tempo, l'applicazione del flusso di byte `rho` ai 1.300 zeri che sono stati aggiunti all'onion payload li trasformerà in dati di riempimento apparentemente casuali. Questa operazione è mostrata nella Figura 10-24.

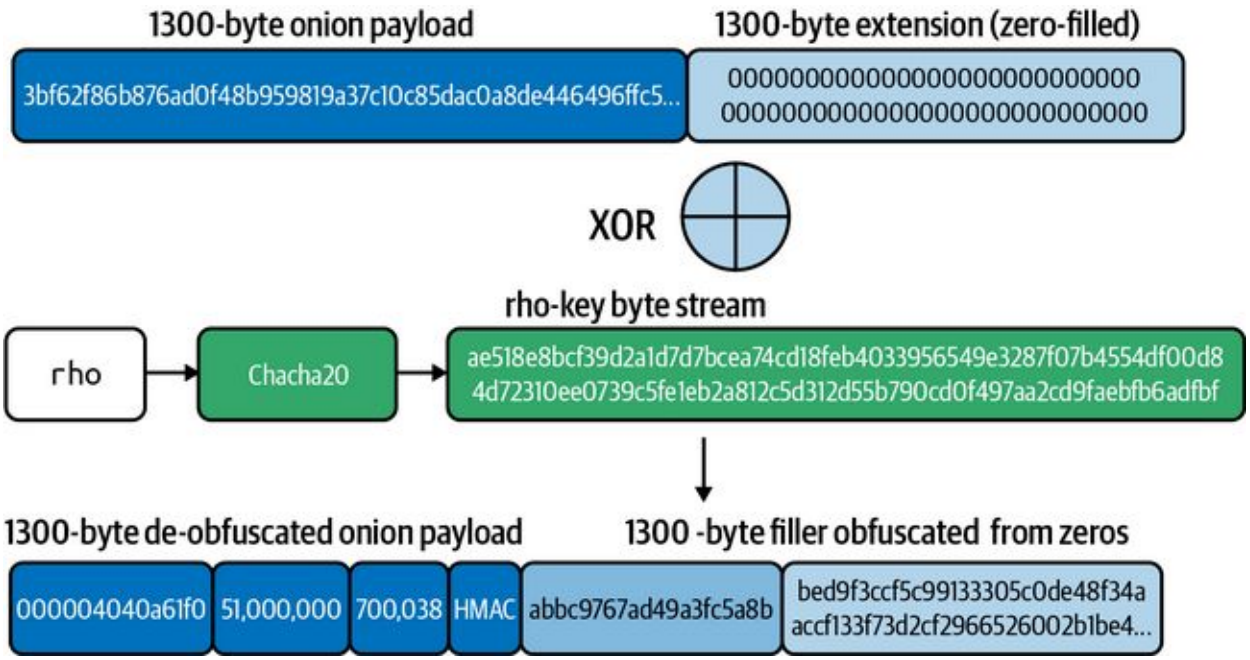

Figura 10-24. Bob de-offusca la cipolla e offusca il riempimento

Bob Estrae l'HMAC Esterno per l'Hop Successivo

Ricorda che per ogni salto (hop) è incluso un HMAC interno, che diventerà quindi l'HMAC esterno per il salto successivo. In questo caso, Bob estrae l'HMAC interno (ha già verificato l'integrità del pacchetto cifrato con l'HMAC esterno), e lo mette da parte perché lo aggiungerà al pacchetto de-offuscato per consentire a Chan di verificare l'HMAC del suo pacchetto crittografato.

Bob Rimuove il Suo Hop Payload e Sposta a Sinistra la Cipolla

Ora Bob può rimuovere il suo hop payload dalla parte anteriore della cipolla e spostare a sinistra i dati rimanenti. Una quantità di dati pari all'hop payload di Bob dalla seconda metà di 1.300 byte di riempimento verrà ora spostata nello spazio del payload della cipolla. Questo è mostrato nella Figura 10-25.

Ora Bob può mantenere la prima metà di 1.300 byte e scartare i 1.300 byte estesi (di riempimento).

Bob ora ha un pacchetto onion da 1.300 byte da inviare all'hop successivo. È quasi identico all'onion payload che Alice aveva creato per Chan, tranne per il fatto che gli ultimi 65 byte circa di riempimento sono stati inseriti da Bob e saranno diversi.

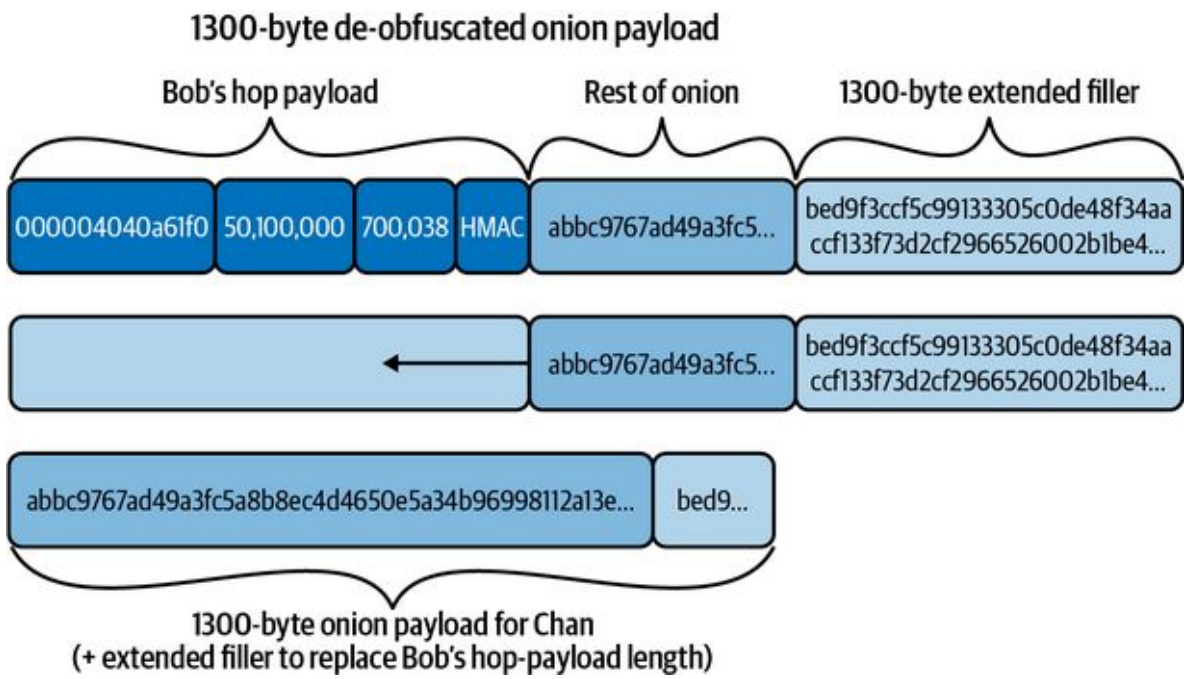

Figura 10-25. Bob rimuove l'hop payload e sposta a sinistra il resto, riempiendo ciò che è vuoto

Nessuno può dire la differenza tra il riempimento messo lì da Alice e il riempitivo messo lì da Bob. Sono comunque tutti byte casuali. Nota che se Bob (o uno degli altri alias di Bob) è presente nel percorso in due posizioni distinte, allora possono notare la differenza perché il protocollo di base utilizza sempre lo stesso hash di pagamento lungo l'intero percorso. I pagamenti atomic multipath (AMP) e i Point Time-Locked Contracts (PTLC) eliminano il vettore di correlazione randomizzando l'identificatore del pagamento su ogni percorso/hop.

Bob Costruisce il Nuovo Pacchetto Onion

Bob ora copia l'onion payload nel pacchetto onion, aggiunge l'HMAC esterno per Chan, ri-randomizza la chiave di sessione (nello stesso modo in cui lo fa Alice, il mittente) con l'operazione di moltiplicazione della curva ellittica e aggiunge un nuovo byte di versione.

Per ri-randomizzare la chiave di sessione, Bob prima calcola il fattore di blinding per il suo hop, usando la sua chiave pubblica del nodo e il segreto condiviso che ha derivato:

```
b_bob = SHA-256(P_bob || shared_secret_bob)
```

Bob ora ri-randomizza la chiave di sessione eseguendo una moltiplicazione EC usando la sua chiave di sessione e il fattore di blinding:

```
session_key_chan = session_key_bob * b_bob
```

La chiave pubblica `session_key_chan` verrà quindi aggiunta all'inizio del pacchetto onion per l'elaborazione da parte di Chan.

Bob Verifica i Dettagli HTLC

L'hop payload di Bob contiene le istruzioni necessarie per creare un HTLC per Chan.

Nell'hop payload, Bob trova `short_channel_id`, `amt_to_forward` e `cltv_expiry`.

Per prima cosa, Bob controlla se ha un canale con quell'ID breve. Scopre di avere un canale con Chan.

Successivamente Bob conferma che l'importo in uscita (50.100 satoshi) è inferiore all'importo in entrata (50.200 satoshi) e quindi le aspettative sulle commissioni di Bob sono soddisfatte.

Allo stesso modo, Bob verifica che il `cltv_expiry` in uscita sia inferiore al `cltv_expiry` in entrata, dando a Bob tempo sufficiente per richiedere l'HTLC in arrivo in caso di violazione.

Bob Invia `update_add_htlc` a Chan

Bob ora costruisce e HTLC da inviare a Chan, come segue:

`channel_id`

> Questo campo contiene l'ID del canale Bob-Chan, che nel nostro esempio è `000004040a61f0`.

`id`

> L'ID di questo HTLC in questo canale, a partire da 0.

`amount_msat`

> L'importo dell'HTLC: 50.100.000 millisatoshi.

payment_hash

L'hash di pagamento RIPEMD160(SHA-256):

`9e017f6767971ed7cea17f98528d5f5c0ccb2c71`.

Questo è lo stesso dell'hash di pagamento dall'HTLC di Alice.

cltv_expiry

Il timelock di scadenza per l'HTLC sarà 700.038.

onion_routing_packet

L'onion packet che Bob ha ricostruito dopo aver rimosso il suo hop payload.

Chan Inoltra la Cipolla

Chan ripete esattamente lo stesso processo di Bob:

1. Chan riceve `update_add_htlc` ed elabora la richiesta HTLC, aggiungendola alle transazioni di impegno.
2. Chan genera la chiave condivisa Alice-Chan e la sottochiave `mu`.
3. Chan verifica l'HMAC del pacchetto onion ed estrae l'onion payload da 1.300 byte.
4. Chan estende l'onion payload di 1.300 byte extra, riempiendolo di zeri.
5. Chan utilizza la chiave `rho` per produrre 2.600 byte.
6. Chan utilizza il flusso di byte generato da XOR e de-offusca l'onion payload. Allo stesso tempo, l'operazione XOR offusca i 1.300 zeri extra, trasformandoli in riempimento.
7. Chan estrae l'HMAC interno nel payload, che diventerà l'HMAC esterno per Dina.
8. Chan rimuove il suo hop payload e sposta a sinistra l'onion payload. Parte del riempimento generato nei 1.300 byte estesi si sposta nella prima metà dei 1.300 byte, diventando parte dell'onion payload.
9. Chan costruisce il pacchetto onion per Dina con questo onion payload.

10. Chan crea un messaggio `update_add_htlc` per Dina e vi inserisce il pacchetto onion
11. Chan invia `update_add_htlc` a Dina.
12. Chan ri-randomizza la chiave della sessione come ha fatto Bob nell'hop precedente per Dina.

Dina Riceve il Payload Finale

Quando Dina riceve il messaggio `update_add_htlc` da Chan, sa dal `payment_hash` che questo è un pagamento per lei. Sa di essere l'ultimo hop/passaggio/strato della cipolla.

Dina segue esattamente lo stesso processo di Bob e Chan per verificare e sbucciare la cipolla, tranne per il fatto che non costruisce un nuovo strato e non inoltra nulla. Invece, Dina risponde a Chan con `update_fulfill_htlc` per riscattare l'HTLC. `update_fulfill_htlc` scorrerà all'indietro lungo il percorso fino a raggiungere Alice. Tutti gli HTLC vengono riscattati e i saldi dei canali vengono aggiornati. Il pagamento è completo!

Errori di Restituzione

Finora abbiamo esaminato la propagazione in avanti della cipolla che stabilisce gli HTLC e la propagazione all'indietro del segreto di pagamento che riavvolge gli HTLC una volta che il pagamento è andato a buon fine.

C'è un'altra funzione molto importante dell'onion routing: gli *errori di restituzione*. Se c'è un problema con il pagamento, la cipolla o l'hop, dobbiamo propagare un errore all'indietro per informare tutti i nodi dell'errore e rimuovere eventuali HTLC.

Gli errori rientrano in tre categorie: errori di cipolla, errori di nodo e errori di canale. Questi inoltre possono essere suddivisi in errori permanenti e transitori. Alcuni errori contengono aggiornamenti del canale per facilitare futuri tentativi di consegna dei pagamenti.

NOTA	A differenza dei messaggi nel protocollo peer-to-peer (definito in BOLT #2: Peer Protocol for Channel Management), gli errori non vengono inviati come messaggi P2P ma vengono racchiusi all'interno di pacchetti onion di ritorno e seguono il percorso inverso (retropropagazione).

La restituzione dell'errore è definita in BOLT #4: Onion Routing, Returning Errors.

Gli errori sono codificati dal nodo di ritorno (quello che ha scoperto un errore) in un *pacchetto di ritorno* come segue:

```
[32*byte:hmac]
[u16:failure_len]
[failure_len*byte:failuremsg]
[u16:pad_len]
[pad_len*byte:pad]
```

Il checksum di verifica HMAC del pacchetto di ritorno viene calcolato con la chiave `um`, generata dal segreto condiviso stabilito dalla cipolla.

SUGGERIMENTO	Se noti, `um` è il contrario di `mu`, indicando lo stesso uso ma nella direzione opposta (propagazione all'indietro).

Il nodo di ritorno genera quindi una chiave `ammag` (l'inverso di "gamma") e offusca il pacchetto di ritorno utilizzando un'operazione XOR con un flusso di byte generato da `ammag`.

Infine il nodo invia il pacchetto di ritorno all'hop da cui ha ricevuto la cipolla originale.

Ogni hop che riceve un errore genererà una chiave `ammag` e offuscherà nuovamente il pacchetto di ritorno utilizzando un'operazione XOR con il flusso di byte da `ammag`.

Alla fine, il mittente (nodo di origine) riceve un pacchetto di ritorno. Quindi genererà le chiavi `ammag` e `um` per ogni hop e deoffuscherà tramite XOR l'errore di ritorno in modo iterativo fino a quando non rivelerà il pacchetto di ritorno.

Messaggi di Errore

Il `failuremsg` è definito in BOLT #4: Onion Routing, Failure Messages.

Un messaggio di errore consiste in un codice di errore a due byte seguito dai dati applicabili a quel tipo di errore.

Il byte superiore del `failure_code` è un insieme di flag binari che possono essere combinati (con OR binario):

0x8000 (BADONION)

Cipolla non analizzabile crittografata inviata da un peer

0x4000 (PERM)

Guasto permanente (altrimenti transitorio)

0x2000 (NODE)

Guasto del nodo (altrimenti canale)

0x1000 (UPDATE)

Nuovo aggiornamento del canale

Vediamo assieme le tipologie di errore, mostrate in Tabella 10-1.

Table 10-1. Tipi di errori onion

Tipo	Nome simbolico	Significato
PERM|1	invalid_realm	Il byte realm non è stato compreso dal nodo di elaborazione
NODE|2	temporary_node_failure	Guasto temporaneo generale del nodo di elaborazione
PERM|NODE|2	permanent_node_failure	Guasto permanente generale del nodo di elaborazione
PERM|NODE|3	required_node_feaure_missing	Il nodo di elaborazione ha una funzionalità richiesta che non era presente in questa cipolla

Tipo	Nome simbolico	Significato
BADONION|PERM|4	invalid_onion_version	Il byte della versione non è stato compreso dal nodo di elaborazione
BADONION|PERM|5	invalid_onion_hmac	L'HMAC della cipolla non era corretto quando ha raggiunto il nodo di elaborazione
BADONION|PERM|6	invalid_onion_key	La chiave temporanea non era analizzabile dal nodo di elaborazione
UPDATE|7	temporary_channel_failure	Il canale dal nodo di elaborazione non è stato in grado di gestire questo HTLC, ma potrebbe essere in grado di gestirlo in seguito
PERM|8	permanent_channel_failure	Il canale dal nodo di elaborazione non è in grado di gestire alcun HTLC
PERM|9	required_channel_feature_missing	Il canale dal nodo di elaborazione richiede funzionalità non presenti nella cipolla
PERM|10	unknown_next_peer	La cipolla ha specificato un short_channel_id che non corrisponde a nessuna parte iniziale del nodo di elaborazione
UPDATE|11	amount_below_minimum	La quantità HTLC era inferiore a htlc_minimum_msat del canale dal nodo di elaborazione
UPDATE|12	fee_insufficient	L'importo della commissione era inferiore a quello richiesto dal

Tipo	Nome simbolico	Significato
		canale dal nodo di elaborazione
UPDATE|13	incorrect_cltv_expiry	Il cltv_expiry non è conforme al cltv_expiry_delta richiesto dal canale dal nodo di elaborazione
UPDATE|14	expiry_too_soon	La scadenza CLTV è troppo vicina all'altezza del blocco corrente per una gestione sicura da parte del nodo di elaborazione
PERM|15	incorrect_or_unknown_payment_details	Il payment_hash è sconosciuto al nodo finale, il payment_secret non corrisponde al payment_hash, l'importo per tale payment_hash non è corretto o la scadenza CLTV dell'HTLC è troppo vicina all'altezza del blocco corrente per una gestione sicura
18	final_incorrect_cltv_expiry	La scadenza CLTV nell'HTLC non corrisponde al valore nella cipolla
19	final_incorrect_htlc_amount	La quantità nell'HTLC non corrisponde al valore nella cipolla
UPDATE|20	channel_disabled	Il canale dal nodo di elaborazione è stato disabilitato
21	expiry_too_far	La scadenza del CLTV nell'HTLC è troppo lontana nel futuro
PERM|22	invalid_onion_payload	Il payload onion-per-hop decrittografato non è stato compreso dal nodo di

Tipo	Nome simbolico	Significato
		elaborazione o è incompleto
23	mpp_timeout	L'intero importo del pagamento in più parti non è stato ricevuto entro un termine ragionevole

Pagamenti bloccati

Nell'attuale implementazione di Lightning Network, esiste la possibilità che un tentativo di pagamento si *blocchi*: non venga né eseguito né annullato da un errore. Ciò può accadere a causa di un bug su un nodo intermedio, un nodo che va offline durante la gestione di HTLC o un nodo dannoso che detiene HTLC senza segnalare un errore. In tutti questi casi, l'HTLC non può essere risolto fino alla sua scadenza. Il timelock (CLTV) impostato su ogni HTLC aiuta a risolvere questa condizione (tra gli altri possibili errori HTLC di routing e di canale).

Tuttavia, ciò significa che il mittente dell'HTLC deve attendere fino alla scadenza e i fondi impegnati in tale HTLC rimangono non disponibili fino alla scadenza dell'HTLC. Inoltre, il mittente *non può riprovare* lo stesso pagamento, perché se lo fa, corre il rischio che sia il pagamento originale *sia* quello ritentato vadano a buon fine: il destinatario viene pagato due volte. Questo perché, una volta inviato, un HTLC non può essere "annullato" dal mittente: deve fallire o scadere. I pagamenti bloccati, sebbene rari, creano un'esperienza utente indesiderata, in cui il portafoglio dell'utente non può pagare o annullare un pagamento.

Una soluzione proposta a questo problema è chiamata *stuckless payments* e dipende dai Point Time-Locked Contracts (PTLC), che sono contratti di pagamento che utilizzano una primitiva crittografica diversa dagli HTLC (ovvero, aggiunta di punti sulla curva ellittica invece di un hash e immagine segreta). I PTLC sono ingombranti utilizzando ECDSA ma molto più facili da gestire con le funzionalità di firma Taproot e Schnorr di Bitcoin, che sono state attivate tra il 2021 e il 2022.

Pagamenti Spontanei Keysend

Nel flusso di pagamento descritto in precedenza nel capitolo, abbiamo ipotizzato che Dina

avesse ricevuto una fattura da Alice "fuori banda" o l'avesse ottenuta tramite qualche meccanismo non correlato al protocollo (tipicamente copia/incolla o scansione del codice QR). Questa caratteristica significa che il processo di pagamento richiede sempre due fasi: in primo luogo, il mittente ottiene una fattura e, in secondo luogo, utilizza l'hash di pagamento (codificato nella fattura) per instradare correttamente un HTLC. Il viaggio di andata e ritorno extra necessario per ottenere una fattura prima di effettuare un pagamento può costituire un collo di bottiglia nelle applicazioni che prevedono lo streaming di micropagamenti su Lightning. E se potessimo semplicemente "spingere" un pagamento spontaneamente, senza dover prima ottenere una fattura dal destinatario? Il protocollo `keysend` è un'estensione end-to-end (solo il mittente e il destinatario sono a conoscenza) del protocollo Lightning che consente pagamenti spontanei.

Record Onion TLV Personalizzati

Il moderno protocollo Lightning utilizza la codifica TLV (Type-Length-Value) nella cipolla per codificare le informazioni che indicano a ciascun nodo *dove* e *come* inoltrare il pagamento. Sfruttando il formato TLV, a ogni informazione di instradamento (come il nodo successivo a cui passare l'HTLC) viene assegnato un tipo specifico (o chiave) codificato come numero intero di lunghezza variabile `BigSize` (dimensione massima come numero intero a 64 bit). Questi tipi "essenziali" (valori invertiti inferiori a `65536`) sono definiti in BOLT #4, insieme al resto dei dettagli dell'onion routing. Le tipologie di cipolla con un valore maggiore di `65536` devono essere utilizzate da portafogli e applicazioni con "record personalizzati".

I record personalizzati consentono alle applicazioni di pagamento di allegare metadati o contesti aggiuntivi a un pagamento come coppie chiave/valore nella cipolla. Poiché i record personalizzati sono inclusi nell'onion payload, come tutti gli altri contenuti hop, i record sono crittografati end-to-end. Poiché i record personalizzati consumano effettivamente una parte del pacchetto onion da 1300 byte a dimensione fissa, la codifica di ciascuna chiave e valore di ciascun record personalizzato riduce la quantità di spazio disponibile per la codifica del resto del percorso. In pratica, ciò significa che maggiore è lo spazio onion utilizzato per i record personalizzati, più breve può essere il percorso. Dato che ogni pacchetto HTLC ha dimensioni fisse, i record personalizzati non aggiungono dati aggiuntivi a un HTLC; piuttosto, riallocano byte che altrimenti sarebbero stati riempiti con dati casuali.

Invio e Ricezione di Pagamenti Keysend

Un pagamento keysend inverte il flusso tipico di un HTLC in cui il destinatario rivela una preimage segreta al mittente. Invece, il mittente include la preimmagine *all'interno* della cipolla al destinatario e instrada l'HTLC al destinatario. Il destinatario quindi decrittografa l'onion payload e utilizza la preimage inclusa (che *deve* corrispondere all'hash di pagamento dell'HTLC) per saldare il pagamento. Di conseguenza, i pagamenti keysend possono essere effettuati senza prima ottenere una fattura dal destinatario, in quanto l'immagine preliminare viene "spinta" verso il destinatario. Un pagamento keysend utilizza un tipo di record personalizzato TLV di 5482373484 per codificare un valore preimage di 32 byte.

Keysend e Record Personalizzati nelle Applicazioni Lightning

Molte applicazioni Lightning in streaming utilizzano il protocollo keysend per trasmettere continuamente satoshi a una destinazione identificata dalla sua chiave pubblica. In genere, un'applicazione includerà anche metadati come una nota di donazione o altre informazioni a livello di applicazione oltre al record keysend.

Conclusione

Il protocollo di onion routing di LN è stato adattato dal protocollo Sphinx per soddisfare al meglio le esigenze di una rete di pagamento. Offre un enorme miglioramento della privacy e della contro-sorveglianza rispetto alla blockchain pubblica e trasparente di Bitcoin.

Nel Capitolo 12 vedremo come la combinazione di source routing e onion routing viene utilizzata da Alice per trovare un percorso e instradare il pagamento a Dina. Per trovare un percorso, Alice deve conoscere la topologia della rete, che è l'argomento del Capitolo 11.

---

1. George Danezis e Ian Goldberg, “Sphinx: A Compact and Provably Secure Mix Format,” in *IEEE Symposium on Security and Privacy* (New York: IEEE, 2009), 269–282.

Capitolo 11. Gossip e Grafo dei Canali

In questo capitolo descriveremo il protocollo di gossip di Lightning Network e come viene utilizzato dai nodi per costruire e mantenere un grafo dei canali. Esamineremo anche il meccanismo di bootstrap DNS utilizzato per trovare peer con cui "fare gossip".

La sezione "Routing fees and Gossip relaying" è evidenziata da uno schema che si trova tra il livello di instradamento e il livello peer-to-peer della Figura 11-1.

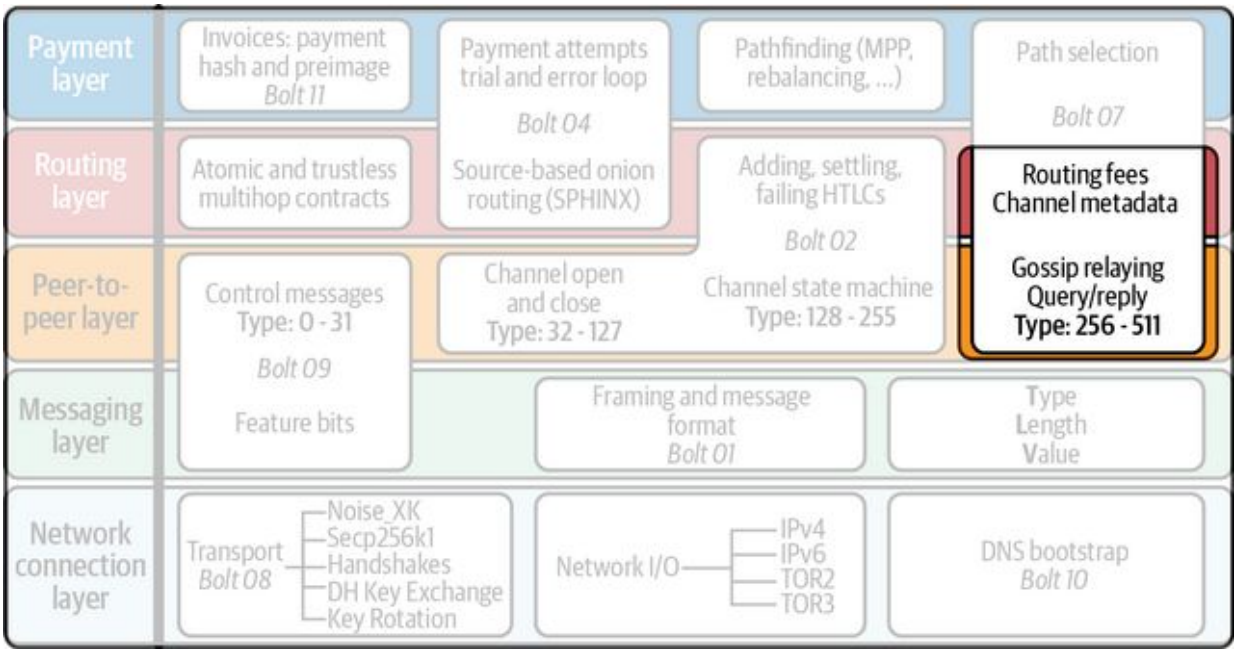

Figura 11-1. Protocollo Gossip nella suite di protocolli Lightning

Come abbiamo già appreso, Lightning Network utilizza un protocollo di onion routing basato sull'origine per consegnare un pagamento da un mittente al destinatario. Per fare ciò, il nodo mittente deve essere in grado di costruire un percorso di canali di pagamento che lo connetta con il destinatario, come vedremo nel Capitolo 12. Pertanto, il mittente deve essere in grado di mappare LN costruendo un *grafo dei canali*. Il grafo dei canali è l'insieme interconnesso di canali anunciato pubblicamente e dei nodi che questi canali collegano.

Poiché i canali sono supportati da una transazione di finanziamento che sta avvenendo on-chain, potresti erroneamente credere che i nodi Lightning possano semplicemente estrarre i canali esistenti dalla blockchain di Bitcoin. Tuttavia questo è possibile solo fino a un certo punto. Le transazioni di finanziamento sono indirizzi Pay-to-Witness-Script-Hash (P2WSH) e la natura dello script (un multisig 2-di-2) verrà rivelata solo una volta speso l'output della transazione di finanziamento. Anche se la natura dello script fosse nota, è importante ricordare che non tutti gli script multisig 2-di-2 corrispondono a canali di pagamento.

Ci sono ancora più motivi per cui guardare la blockchain di Bitcoin potrebbe non essere utile. Ad esempio, su Lightning Network, le chiavi Bitcoin utilizzate per la firma vengono ruotate dai nodi per ogni canale e aggiornamento. Pertanto, anche se potessimo rilevare in modo affidabile le transazioni di finanziamento sulla blockchain di Bitcoin, non sapremmo quali nodi del Lightning Network possiedono quel particolare canale.

Lightning Network risolve questo problema implementando un *protocollo di gossip*. I protocolli di gossip sono tipici delle reti peer-to-peer (P2P) e consentono ai nodi di condividere informazioni con l'intera rete con poche connessioni dirette ai peer. I nodi Lightning aprono tra loro connessioni peer-to-peer crittografate e condividono informazioni (gossip) che hanno ricevuto da altri peer. Non appena un nodo desidera condividere alcune informazioni, ad esempio su un canale appena creato, invia un messaggio a tutti i suoi peer. Alla ricezione di un messaggio, un nodo decide se il messaggio ricevuto era nuovo e, in tal caso, inoltra le informazioni ai suoi pari. In questo modo, se la rete peer-to-peer è ben connessa, tutte le nuove informazioni necessarie al funzionamento della rete verranno eventualmente propagate a tutti gli altri peer.

Ovviamente, se un nuovo peer si unisce alla rete per la prima volta, deve conoscere altri peer sulla rete, in modo da potersi connettere ad altri e partecipare alla rete.

In questo capitolo, esploreremo esattamente *come* i nodi Lightning si scoprono a vicenda, aggiornano lo stato dei nodi e comunicano tra loro.

Quando le persone si riferiscono alla parte di *rete* del Lightning Network, in realtà si riferiscono al *grafo dei canali* che a sua volta è una struttura di dati autenticata univoca *ancorata* nella blockchain di Bitcoin.

Tuttavia, Lightning Network è anche una rete peer-to-peer di nodi che trasmettono informazioni su canali e nodi di pagamento. Di solito, per mantenere un canale di pagamento, due peer devono parlare direttamente tra loro, il che significa che ci sarà una connessione tra loro. Ciò suggerisce che il grafo dei canali è una sottorete della rete peer-to-

peer. Tuttavia, questo non è vero perché i canali di pagamento possono rimanere aperti anche se uno o entrambi i peer vanno temporaneamente offline.

Rivediamo parte della terminologia che abbiamo utilizzato nel libro, osservando in particolare il significato in termini di grafo dei canali e rete peer-to-peer (vedi la Tabella 11-1).

Tabella 11-1. Terminologia delle diverse reti

Grafo dei canali	Rete peer-to-peer
canale	connessione
apertura	connessione
chiusura	disconnessione
transazione di finanziamento	connessione TCP/IP crittografata
invio	trasmissione
pagamento	messaggio

Poiché LN è una rete peer-to-peer, è necessario un bootstrap iniziale affinché i peer possano rilevarsi a vicenda. In questo capitolo seguiremo la storia di un nuovo peer che si connette alla rete per la prima volta ed esamineremo ogni fase del processo di bootstrap, dalla scoperta iniziale del peer alla sincronizzazione e convalida del grafo dei canali.

Come primo passo, il nostro nuovo nodo deve in qualche modo *scoprire* almeno *un* peer che è già connesso alla rete e ha un grafo completo dei canali (come vedremo più avanti, non esiste una versione canonica del grafo dei canali). Utilizzando uno dei tanti protocolli iniziali di bootstrap per trovare quel primo peer, dopo che è stata stabilita una connessione, il nostro nuovo peer deve ora *scaricare* e *convalidare* il grafo dei canali. Una volta che il grafo dei canali è stato completamente convalidato, il nostro nuovo peer è pronto per iniziare ad

aprire canali e inviare pagamenti sulla rete.

Dopo il bootstrap iniziale, un nodo sulla rete deve continuare a mantenere la sua visualizzazione del grafo dei canali elaborando nuovi aggiornamenti delle policy di instradamento del canale, scoprendo e convalidando nuovi canali, rimuovendo i canali che sono stati chiusi on-chain e infine eliminando i canali che falliscono nell'inviare un vero e proprio "battito cardiaco" ogni due settimane circa.

Al termine di questo capitolo, capirai un componente chiave del Lightning Network peer-to-peer: come i peer si scoprono a vicenda e mantengono una copia locale (soggettiva) del grafo dei canali. Inizieremo esplorando la storia di un nuovo nodo che si è appena avviato e ha bisogno di trovare altri peer a cui connettersi sulla rete.

Peer Discovery

In questa sezione, inizieremo a seguire un nuovo nodo Lightning che desidera unirsi alla rete attraverso tre passaggi:

1. Scopre un set di peer di bootstrap
2. Scarica e convalida il grafo dei canali
3. Inizia il processo di manutenzione continua del grafo dei canali

P2P Bootstrap

Prima di fare qualsiasi altra cosa, il nostro nuovo nodo deve prima scoprire un insieme di peer che fanno già parte della rete. Chiamiamo questo processo iniziale peer bootstrap ed è qualcosa che ogni rete peer-to-peer deve implementare correttamente per garantire una rete solida e sana.

Il bootstrap di nuovi peer su reti peer-to-peer esistenti è un problema molto ben studiato con diverse soluzioni note, ciascuna con i propri distinti compromessi. La soluzione più semplice a questo problema è semplicemente quella di impacchettare un set di peer di bootstrap *hardcoded* nel software del nodo P2P impacchettato. Questo è semplice in quanto ogni nuovo nodo ha un elenco di peer di bootstrap nel software che sta eseguendo, ma piuttosto fragile dato che se l'insieme di peer di bootstrap va offline, nessun nuovo nodo

sarà in grado di unirsi alla rete. A causa di questa fragilità, questa opzione viene solitamente utilizzata come fallback nel caso in cui nessuno degli altri meccanismi di bootstrap P2P funzioni correttamente.

Piuttosto che codificare l'insieme di peer di bootstrap all'interno del software/binario stesso, possiamo invece consentire ai peer di ottenere dinamicamente un nuovo set di peer di bootstrap che possono utilizzare per unirsi alla rete. Chiameremo questo processo *scoperta tra peer iniziale* (*initial peer discovery*). In genere sfrutteremo i protocolli Internet esistenti per mantenere e distribuire un set di peer di bootstrap. Un elenco non esaustivo di protocolli che sono stati utilizzati in passato per realizzare l'iniziale peer discovery include:

- Domain Name Service (DNS)
- Internet Relay Chat (IRC)
- Hypertext Transfer Protocol (HTTP)

Simile al protocollo Bitcoin, il principale meccanismo iniziale di peer discovery utilizzato su Lightning Network avviene tramite DNS. Poiché la scoperta peer iniziale è un'attività critica e universale per la rete, il processo è stato *standardizzato* in BOLT #10: DNS Bootstrap.

Bootstrap DNS

Il documento BOLT #10 descrive un modo standardizzato di implementare la peer discovery utilizzando il DNS. Il bootstrap di LN basato su DNS utilizza fino a tre tipi di record distinti:

- Record SRV per l'individuazione di un set di *chiavi pubbliche del nodo*.
- Record A per mappare la chiave pubblica di un nodo al suo attuale indirizzo `IPv4`.
- Record AAA per mappare la chiave pubblica di un nodo al suo attuale indirizzo `IPv6`.

Coloro che hanno una certa familiarità con il protocollo DNS potrebbero già avere familiarità con i tipi di record A (da nome a indirizzo IPv4) e AAA (da nome a indirizzo IPv6), ma non con il tipo SRV. Il tipo di record SRV viene utilizzato dai protocolli basati su DNS per *determinare* la posizione di un servizio specifico. Nel nostro contesto, il servizio in questione è un determinato nodo Lightning e la posizione è il suo indirizzo IP. Abbiamo bisogno di utilizzare questo tipo di record aggiuntivo perché, a differenza dei nodi all'interno del protocollo Bitcoin, abbiamo bisogno *sia* di una chiave pubblica che di un indirizzo IP per connetterci a un nodo. Come vedremo nel Capitolo 13, il protocollo di crittografia del trasporto utilizzato

su Lightning Network richiede la conoscenza della chiave pubblica di un nodo prima di connettersi, in modo da implementare l'occultamento dell'identità per i nodi nella rete.

Il flusso di bootstrap di un nuovo peer

Prima di approfondire le specifiche di BOLT #10, delineeremo innanzitutto il flusso di alto livello di un nuovo nodo che desidera utilizzare BOLT #10 per unirsi alla rete.

Innanzitutto, un nodo deve identificare un singolo server DNS o un insieme di server DNS che comprendano BOLT #10 in modo che possano essere utilizzati per il bootstrap P2P.

Sebbene BOLT #10 utilizzi *lseed.bitcoinstats.com* come seed server, non esiste un set "ufficiale" di seed DNS, ma ciascuna delle principali implementazioni mantiene il proprio seed DNS e interrogano reciprocamente i seed dell'altro per fini di ridondanza. Nella tabella 11-2 viene mostrato un elenco non esaustivo di alcuni noti seed server DNS.

Tabella 11-2. Tabella dei seed server Lightning DNS noti

Server DNS	Manutentore
lseed.bitcoinstats.com	Christian Decker
nodes.lightning.directory	Lightning Labs (Olaoluwa Osuntokun)
soa.nodes.lightning.directory	Lightning Labs (Olaoluwa Osuntokun)
lseed.darosior.ninja	Antoine Poinsot

Esistono seed DNS sia per la mainnet che per la testnet di Bitcoin. Per il nostro esempio, assumeremo l'esistenza di un seed DNS BOLT #10 valido in *nodes.lightning.directory*.

Successivamente, il nostro nuovo nodo emetterà una query SRV per ottenere un *set di peer bootstrap candidati*. La risposta alla nostra domanda sarà una serie di chiavi pubbliche codificate bech32. Poiché il DNS è un protocollo basato su testo, non possiamo inviare dati binari grezzi, quindi è necessario uno schema di codifica. BOLT #10 specifica una codifica bech32 a causa del suo utilizzo nel più ampio ecosistema Bitcoin. Il numero di chiavi

pubbliche codificate restituite dipende dal server che restituisce la query, nonché da tutti i resolver che si trovano tra il client e il server autorevole.

Utilizzando lo strumento da riga di comando `dig` ampiamente disponibile, possiamo interrogare la versione *testnet* del seed DNS menzionato in precedenza:

```
$ dig @8.8.8.8 test.nodes.lightning.directory SRV
```

Utilizziamo l'argomento @ per forzare la risoluzione tramite il server dei nomi di Google (con indirizzo IP 8.8.8.8) perché non filtra le risposte alle query SRV di grandi dimensioni. Alla fine del comando, specifichiamo che vogliamo che vengano restituiti solo i record SRV. Una risposta di esempio è simile all'Esempio 11-1.

Esempio 11-1. Interrogazione del seed DNS per i nodi raggiungibili

```
$ dig @8.8.8.8 test.nodes.lightning.directory SRV

; <<>> DiG 9.10.6 <<>> @8.8.8.8 test.nodes.lightning.directory SRV

; (1 server found)

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43610

;; flags: qr rd ra; QUERY: 1, ANSWER: 25, AUTHORITY: 0, ADDITIONAL: 1

;; QUESTION SECTION:

;test.nodes.lightning.directory.    IN    SRV

;; ANSWER SECTION:

test.nodes.lightning.directory.     59 IN SRV   10 10 9735 (1)

ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test.nodes.light
ning.directory. (2)

test.nodes.lightning.directory.     59 IN SRV   10 10 15735
ln1qtgsl3efj8verd4z27k44xu0a59kncvsarxatahm334exgnuvwhnz8dkhx8.test.nodes.light
ning.directory.

 [...]

;; Query time: 89 msec

;; SERVER: 8.8.8.8#53(8.8.8.8)

;; WHEN: Thu Dec 31 16:41:07 PST 2020
```

1. Numero di porta TCP in cui è possibile raggiungere il nodo LN.
2. Chiave pubblica (ID) del nodo codificata come nome di dominio virtuale.

Abbiamo troncato la risposta per semplicità e mostriamo solo due delle risposte restituite. Le risposte contengono un nome di dominio "virtuale" per un nodo target, poi a sinistra abbiamo la *porta TCP* dove questo nodo può essere raggiunto. La prima risposta utilizza la porta TCP standard per Lightning Network: `9735`. La seconda risposta utilizza una porta personalizzata, consentita dal protocollo.

Successivamente, tenteremo di ottenere l'altra informazione di cui abbiamo bisogno per connetterci a un nodo: il suo indirizzo IP. Prima di poterlo interrogare, tuttavia, *decodificheremo* la codifica bech32 della chiave pubblica dal nome di dominio virtuale:

```
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7
```

Decodificando questa stringa bech32 otteniamo la seguente chiave pubblica `secp256k1`:

```
026c64f5a7f24c6f7f0e1d6ec877f23b2f672fb48967c2545f227d70636395eaf3
```

Ora che abbiamo la chiave pubblica grezza, chiederemo al server DNS di *risolvere* l'host virtuale fornito in modo da poter ottenere le informazioni IP (record `A`) per il nodo, come mostrato in Esempio 11.2.

Esempio 11-2. Ottenere l'indirizzo IP più recente per un nodo

```
$ dig
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test.nodes.light
ning.directory A

; <<>> DiG 9.10.6 <<>>
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test.nodes.light
ning.directory A

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41934

;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:

; EDNS: version: 0, flags:; udp: 4096
```

```
;; QUESTION SECTION:

;ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test.nodes.lightning.directory. IN A

;; ANSWER SECTION:

ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test.nodes.lightning.directory. 60 IN A X.X.X.X (1)

;; Query time: 83 msec

;; SERVER: 2600:1700:6971:6dd0::1#53(2600:1700:6971:6dd0::1)

;; WHEN: Thu Dec 31 16:59:22 PST 2020
```

1. Il server DNS restituisce un indirizzo IP `X.X.X.X`. L'abbiamo sostituito con delle X per evitare di presentare un vero indirizzo IP.

Nel comando precedente, abbiamo interrogato il server in modo da poter ottenere un indirizzo IPv4 (record `A`) per il nostro nodo di destinazione (sostituito da `X.X.X.X` nell'esempio precedente). Ora che abbiamo la chiave pubblica non elaborata, l'indirizzo IP e la porta TCP, possiamo connetterci al protocollo di trasporto del nodo:

```
026c64f5a7f24c6f7f0e1d6ec877f23b2f672fb48967c2545f227d70636395eaf3@x.x.x.x:9735
```

Interrogazione del record DNS A corrente per un determinato nodo può essere utilizzato anche per cercare l'*ultimo* set di indirizzi. Tali query possono essere utilizzate per sincronizzare più rapidamente le informazioni di indirizzamento più recenti per un nodo, rispetto all'attesa degli aggiornamenti degli indirizzi sulla rete gossip.

A questo punto del nostro viaggio, il nostro nuovo nodo Lightning ha trovato il suo primo peer e ha stabilito la sua prima connessione! Ora possiamo iniziare la seconda fase del nuovo bootstrap peer: sincronizzazione e convalida del grafo dei canali.

Esploreremo le complessità di BOLT # 10 stesso per vederlo più approfonditamente.

Opzioni Query SRV

Lo standard BOLT #10 è altamente estensibile grazie al suo utilizzo di sottodomini nidificati come livello di comunicazione per ulteriori opzioni di query. Il protocollo di bootstrap consente ai client di specificare ulteriormente il tipo di nodi per i quali stanno tentando di interrogare rispetto all'impostazione predefinita di ricevere un sottoinsieme casuale di nodi

nelle risposte alla query.

Lo schema del sottodominio dell'opzione di query utilizza una serie di coppie chiave-valore in cui la chiave stessa è una *singola lettera* e l'insieme di testo rimanente è il valore stesso. I seguenti tipi di query esistono nella versione corrente dello standard BOLT #10:

r

Il byte *realm* utilizzato per determinare per quali query della catena o *realm* devono essere restituite. Così com'è, l'unico valore per questa chiave è `0` che denota "Bitcoin".

a

Consente ai client di filtrare i nodi restituiti in base ai *tipi* di indirizzi che pubblicizzano. Ad esempio, questo può essere utilizzato per ottenere solo nodi che pubblicizzano un indirizzo IPv6 valido. Il valore che segue questo tipo si basa su un campo di bit che esegue l'*indicizzazione* nell'insieme di *tipi* di indirizzo specificati definiti in BOLT #7. Il valore predefinito per questo campo è 6, che rappresenta sia IPv4 che IPv6 (i bit 1 e 2 sono impostati).

l

Una chiave pubblica del nodo serializzata in formato compresso. Ciò consente a un client di interrogare un nodo specifico anziché ricevere un insieme di nodi casuali.

n

Il numero di record da restituire. Il valore predefinito per questo campo è 25.

Una query di esempio con opzioni di query aggiuntive è simile alla seguente:

```
r0.a2.n10.nodes.lightning.directory
```

Analizzando la query una coppia chiave-valore alla volta, otteniamo le seguenti informazioni:

r0

La query si rivolge al ream Bitcoin

a2

La query richiede solo la restituzione degli indirizzi IPv4

n10

La richiesta di query

Prova tu stesso alcune combinazioni dei vari flag utilizzando il comando `dig`:

```
dig @8.8.8.8 r0.a6.nodes.lightning.directory SRV
```

Il Grafo dei Canali

Ora che il nostro nuovo nodo è in grado di utilizzare il protocollo di bootstrap DNS per connettersi al suo primo peer, può iniziare a sincronizzare il grafo dei canali! Tuttavia, prima di sincronizzare il grafo dei canali, dobbiamo imparare esattamente *cosa* intendiamo per grafo dei canali. In questa sezione esploreremo la *struttura* precisa del grafo dei canali ed esamineremo gli aspetti unici del grafo dei canali rispetto alla tipica struttura dati "grafica" astratta che è ben nota/usata nel campo dell'informatica.

Un Grafo Orientato

Un *grafo* in informatica è una struttura di dati speciale composta da vertici (tipicamente indicati come nodi) e bordi (noti anche come collegamenti). Due nodi possono essere collegati da uno o più bordi. Anche il grafo dei canali è *orientato* dato che un pagamento è in grado di fluire in entrambe le direzioni su un dato bordo (un canale). Un esempio di *grafo orientato* è mostrato nella Figura 11-2.

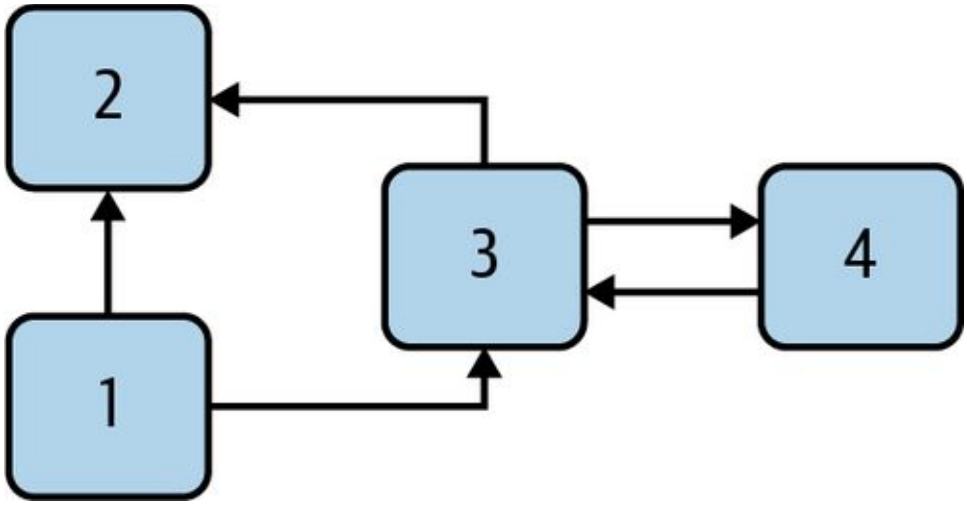

Figura 11-2. Un grafo orientato

Nel contesto di Lightning Network, i nostri vertici sono gli stessi nodi Lightning, mentre i nostri bordi sono i canali di pagamento che collegano questi nodi. Poiché ci occupiamo di *instradare i pagamenti*, nel nostro modello un nodo senza bordi (nessun canale di pagamento) non è considerato parte del grafo poiché non è utile.

Poiché i canali stessi sono UTXO (indirizzi multisig finanziati 2-di-2), possiamo visualizzare il grafo dei canali come un sottoinsieme speciale del set di UTXO di Bitcoin, in cima al quale possiamo aggiungere alcune informazioni aggiuntive (i nodi, ecc.) per arrivare alla struttura di sovrapposizione finale, che è il grafo dei canali. Questo ancoraggio di componenti fondamentali del grafo dei canali nella blockchain di Bitcoin significa che è impossibile *falsificare* un grafo dei canali valido, che ha proprietà utili quando si tratta di prevenzione dello spam, come vedremo più avanti.

Messaggi del Protocollo di Gossip

Le informazioni del grafo dei canali vengono propagate attraverso la rete Lightning P2P come tre messaggi, descritti in BOLT #7:

`node_announcement`

> Il vertice nel nostro grafico che comunica la chiave pubblica di un nodo, nonché come raggiungere il nodo su Internet e alcuni metadati aggiuntivi che descrivono l'insieme

di *funzionalità* supportate dal nodo.

`channel_announcement`

Una prova ancorata alla blockchain dell'esistenza di un canale tra due singoli nodi. Qualsiasi terza parte può verificare questa prova per garantire che un canale *reale* sia effettivamente pubblicizzato. Simile a `node_announcement`, questo messaggio contiene anche informazioni che descrivono le *capacità* del canale, utili quando si tenta di instradare un pagamento.

`channel_update`

Una coppia di strutture che descrive l'insieme delle politiche di instradamento per un determinato canale. I messaggi `channel_update` vengono in *coppia* perché un canale è un bordo diretto, quindi ogni lato del canale è in grado di specificare la propria politica di instradamento personalizzata.

È importante notare che ogni componente del grafo del canali è *autenticato*, consentendo a una terza parte di garantire che il proprietario di un canale/aggiornamento/nodo sia effettivamente quello che invia un aggiornamento. Ciò rende effettivamente il grafo dei canali un tipo unico di *struttura dati autenticata* che non può essere contraffatta. Per l'autenticazione, utilizziamo una firma digitale ECDSA `secp256k1` (o una serie di esse) sopra il digest serializzato del messaggio stesso. In questo capitolo non entreremo nello specifico del framing/serializzazione della messaggistica utilizzato in Lightning Network, poiché tratteremo tali informazioni in dettaglio nel Capitolo 13.

Dopo aver delineato la struttura di alto livello del grafo dei canali, ci addentreremo nella struttura precisa di ciascuno dei tre messaggi utilizzati per farne gossip. Spiegheremo anche come si può anche verificare ogni messaggio e componente del grafo dei canali.

Il Messaggio `node_announcement`

Innanzitutto, il messaggio `node_announcement`, ha due scopi principali:

1. Pubblicizzare le informazioni di connessione in modo che altri nodi possano

connettersi a un nodo per eseguire il bootstrap o per tentare di stabilire un nuovo canale di pagamento con quel nodo.

2. Comunicare l'insieme di funzionalità (capacità) a livello di protocollo che un nodo comprende/supporta. La negoziazione delle funzionalità tra i nodi consente agli sviluppatori di aggiungere nuove funzionalità in modo indipendente e supportarle con qualsiasi altro nodo su base opt-in.

A differenza degli annunci di canale, gli annunci di nodo non sono ancorati alla blockchain. Pertanto, gli annunci di nodo sono considerati validi solo se si sono propagati con un corrispondente annuncio di canale. In altre parole, rifiutiamo sempre i nodi senza canali di pagamento per garantire che un peer malevolo non possa inondare la rete con nodi fasulli che non fanno parte del grafo dei canali.

La struttura del messaggio `node_announcement`

Il `node_announcement` è composto dai seguenti campi:

`signature`

Una firma ECDSA valida che copre il digest serializzato di tutti i campi elencati di seguito. Questa firma deve corrispondere alla chiave pubblica del nodo pubblicizzato.

`features`

Un vettore di bit che descrive l'insieme di caratteristiche del protocollo che questo nodo comprende. Ad alto livello, questo campo contiene una serie di bit che rappresentano le caratteristiche di un nodo. Ad esempio, un nodo può segnalare che comprende l'ultimo tipo di canale.

`timestamp`

Un timestamp codificato in un'epoca Unix. Ciò consente ai client di applicare un ordinamento parziale sugli aggiornamenti all'annuncio di un nodo.

`node_id`

La chiave pubblica `secp256k1` a cui appartiene questo annuncio del nodo. Può esserci un singolo `node_announcement` per un dato nodo nel grafo dei canali in uno specifico momento. Di conseguenza, un `node_announcement` può sostituire un `node_announcement` precedente per lo stesso nodo se ha un timestamp più alto.

`rgb_color`

Un campo che consente a un nodo di specificare un colore RGB da associare ad esso, spesso utilizzato nelle visualizzazioni dei grafici dei canali e nelle directory dei nodi.

`alias`

Una stringa UTF-8 che funge da soprannome per un dato nodo. Tieni presente che questi alias non devono essere univoci a livello globale, né sono verificati in alcun modo. Di conseguenza, non dovrebbero essere considerati una forma di identità: possono essere facilmente falsificati.

`addresses`

Un insieme di indirizzi Internet pubblici raggiungibili che devono essere associati a un dato nodo. Nella versione attuale del protocollo, sono supportati quattro tipi di indirizzo: IPv4 (tipo: 1), IPv6 (tipo: 2), Tor v2 (tipo: 3) e Tor v3 (tipo: 4). Nel messaggio `node_announcement`, ciascuno di questi tipi di indirizzo è indicato da un tipo intero che è incluso tra parentesi dopo il tipo di indirizzo.

Convalida degli annunci dei nodi

La convalida di un `node_announcement` in arrivo è semplice. Le seguenti affermazioni dovrebbero essere confermate quando si esamina un annuncio di nodo:

- Se un `node_announcement` esistente per quel nodo è già noto, il `timestamp` di un nuovo `node_announcement` in arrivo deve essere maggiore di quello precedente. Con questo vincolo, applichiamo un livello forzato di "freschezza".
- Se non esiste alcun `node_announcement` per il dato nodo, allora un `channel_announcement` esistente che fa riferimento al dato nodo (ne parleremo più

avanti) deve già esistere nel grafo dei canali locale.

- La `signature` inclusa deve essere una firma ECDSA valida verificata utilizzando la chiave pubblica `node_id` inclusa e il digest double–SHA-256 della codifica del messaggio non elaborato (meno la firma e l'intestazione del frame) come messaggio.
- Tutti gli `addresses` (indirizzi) inclusi devono essere ordinati in ordine crescente in base all'identificatore dell'indirizzo.
- I byte `alias` inclusi devono essere una stringa UTF-8 valida.

Il Messaggio `channel_announcement`

Il messaggio `channel_announcement` viene utilizzato per *annunciare* un nuovo canale *pubblico* alla rete. Tieni presente che l'annuncio di un canale è *facoltativo*. Un canale deve essere annunciato solo se è destinato a essere utilizzato per l'instradamento su LN. I nodi di routing potrebbero voler annunciare tutti i loro canali. Tuttavia, alcuni nodi come i nodi mobile probabilmente non hanno uptime o non vogliono essere un nodo di routing. Di conseguenza, questi nodi mobile (che in genere utilizzano client leggeri per connettersi alla rete Bitcoin P2P) possono invece avere canali (privati) puramente *non annunciati*.

Canali non annunciati (privati)

Un canale non annunciato non fa parte del grafo dei canali pubblici noti, ma può comunque essere utilizzato per inviare/ricevere pagamenti. Un lettore astuto potrebbe ora chiedersi come un canale che non fa parte del grafo dei canali pubblici sia in grado di ricevere pagamenti. La soluzione a questo problema è un insieme di "aiutanti di ricerca del percorso" che chiamiamo suggerimenti di instradamento (routing hints). Come vedremo nel Capitolo 15, le fatture create da nodi con canali non pubblicizzati includeranno informazioni per aiutare il mittente a instradarle, supponendo che il nodo abbia almeno un singolo canale con un nodo di instradamento pubblico esistente.

A causa dell'esistenza di canali non pubblicizzati, la dimensione reale del grafo dei canali (sia la componente pubblica che quella privata) è sconosciuta.

Individuazione di un canale sulla blockchain di Bitcoin

Come accennato in precedenza, il grafo dei canali è autenticato grazie al suo utilizzo della crittografia a chiave pubblica, nonché della blockchain Bitcoin come sistema di prevenzione dello spam. Per fare in modo che un nodo accetti un nuovo `channel_announcement`, l'annuncio deve dimostrare che il canale esiste effettivamente nella blockchain di Bitcoin. Questo sistema di prova aggiunge un costo iniziale all'aggiunta di una nuova voce al grafo dei canali (le commissioni on-chain che si devono pagare per creare l'UTXO del canale). Di conseguenza, mitighiamo lo spam e assicuriamo che un nodo disonesto sulla rete non possa riempire gratuitamente la memoria di un nodo onesto con canali fasulli.

Dato che abbiamo bisogno di costruire una prova dell'esistenza di un canale, una domanda naturale che ci si pone è: come possiamo "indicare" o fare riferimento a un dato canale per il verificatore? Dato che un canale di pagamento è ancorato all'output di una transazione non spesa, un primo pensiero potrebbe essere quello di tentare di pubblicizzare l'outpoint completo (`txid:index`) del canale. Dato che l'outpoint è globalmente unico e confermato nella catena, sembra una buona idea; tuttavia, presenta uno svantaggio: il verificatore deve conservare una copia completa dell'UTXO impostato per verificare i canali. Funziona bene per i nodi completi Bitcoin, ma i client che si affidano a una verifica leggera in genere non mantengono un set UTXO completo. Poiché vogliamo assicurarci di poter supportare i nodi mobile su Lightning Network, siamo costretti a cercare un'altra soluzione.

E se invece di fare riferimento a un canale tramite il suo UTXO, facessimo riferimento in base alla sua "posizione" nella catena? Per fare ciò, avremo bisogno di uno schema che ci consenta di fare riferimento a un determinato blocco, quindi a una transazione all'interno di quel blocco e infine a un output specifico creato da quella transazione. Tale identificatore è descritto in BOLT #7 ed è indicato come *short channel ID*, o `scid`. Lo scid viene utilizzato in `channel_announcement` (e `channel_update`) così come all'interno del pacchetto di routing cifrato onion incluso negli HTLC, come abbiamo appreso nel Capitolo 10.

L'ID del canale breve

Sulla base delle informazioni precedenti, abbiamo tre informazioni che dobbiamo codificare per fare riferimento in modo univoco a un determinato canale. Poiché vogliamo una rappresentazione compatta, tenteremo di codificare le informazioni in un *singolo* numero intero. Il nostro formato preferito è un intero senza segno a 64 bit, composto da 8 byte.

Innanzitutto, l'altezza del blocco. Usando 3 byte (24 bit) possiamo codificare 16.777.216 blocchi. Ciò lascia 5 byte disponibili per codificare rispettivamente l'indice di transazione e

l'indice di output. Useremo i successivi 3 byte per codificare l'indice della transazione *all'interno* di un blocco. Questo è più che sufficiente dato che è possibile correggere solo decine di migliaia di transazioni in un blocco alle dimensioni attuali del blocco. Ciò ci lascia 2 byte per codificare l'indice di output del canale all'interno della transazione.

Il nostro formato `scid` finale assomiglia a:

```
block_height (3 bytes) || transaction_index (3 bytes) || output_index (2 bytes)
```

Utilizzando le tecniche di impacchettamento dei bit, codifichiamo prima i 3 byte più significativi come altezza del blocco, i successivi 3 byte come indice di transazione e i 2 byte meno significativi come indice di output che crea il canale UTXO.

Un ID canale breve può essere rappresentato come un singolo numero intero (`695313561322258433`) o come una stringa più intuitiva: `632384x1568x1`. Qui vediamo che il canale è stato minato nel blocco `632384`, era la `1568`a transazione nel blocco, con l'output del canale come secondo output (gli UTXO sono indicizzati a zero) prodotto dalla transazione.

Ora che siamo in grado di indicare un determinato output di finanziamento del canale on-chain, possiamo esaminare l'intera struttura del messaggio `channel_announcement`, oltre a vedere come verificare la prova di esistenza inclusa nel messaggio.

La struttura del messaggio `channel_announcement`

Un `channel_announcement` comunica principalmente due cose:

1. Una prova che esiste un canale tra il nodo A e il nodo B con entrambi i nodi che controllano le chiavi multisig nell'output di quel canale.
2. L'insieme delle capacità del canale (quali tipi di HTLC può instradare, ecc.).

Quando descriviamo la prova, in genere faremo riferimento al nodo `1` e al nodo 2. Dei due nodi che un canale connette, il "primo" nodo è il nodo che ha una codifica della chiave pubblica "inferiore" quando confrontiamo la chiave pubblica dei due nodi in formato compresso codificato esadecimale in ordine lessicografico. Di conseguenza, oltre a una chiave pubblica del nodo sulla rete, ogni nodo dovrebbe anche controllare una chiave pubblica all'interno della blockchain di Bitcoin.

Simile al messaggio `node_announcement`, tutte le firme incluse del messaggio `channel_announcement` dovrebbero essere firmate/verificate rispetto alla codifica grezza

del messaggio (meno l'intestazione) che segue *dopo* la firma finale (poiché non è possibile che una firma digitale si firmi da sola).

Detto questo, un messaggio `channel_announcement` ha i seguenti campi:

`node_signature_1`

La firma del primo nodo sul digest del messaggio.

`node_signature_2`

La firma del secondo nodo sul digest del messaggio.

`bitcoin_signature_1`

La firma della chiave multisig (nell'output di finanziamento) del primo nodo sul digest del messaggio.

`bitcoin_signature_2`

La firma della chiave multisig (nell'output di finanziamento) del secondo nodo sul digest del messaggio.

`features`

Un vettore di bit che descrive l'insieme di funzionalità a livello di protocollo supportate da questo canale.

`chain_hash`

Un hash a 32 byte che è in genere l'hash del blocco di genesi della blockchain (ad esempio, la mainnet Bitcoin) in cui è stato aperto il canale.

`short_channel_id`

Lo `scid` che individua in modo univoco il dato output di finanziamento del canale all'interno della blockchain.

node_id_1

La chiave pubblica del primo nodo della rete.

node_id_2

La chiave pubblica del secondo nodo della rete.

bitcoin_key_1

La chiave multisig non elaborata per l'output del finanziamento del canale per il primo nodo della rete.

bitcoin_key_2

La chiave multisig non elaborata per l'output del finanziamento del canale per il secondo nodo della rete.

Convalida dell'annuncio del canale

Ora che sappiamo cosa contiene un `channel_announcement`, possiamo vedere come verificare l'esistenza on-chain del canale.

Armato delle informazioni in `channel_announcement`, qualsiasi nodo Lightning (anche uno senza una copia completa della blockchain di Bitcoin) può verificare l'esistenza e l'autenticità del canale di pagamento.

Innanzitutto, il verificatore utilizzerà l'ID breve del canale per trovare quale blocco Bitcoin contiene l'output del finanziamento del canale. Con le informazioni sull'altezza del blocco, il verificatore può richiedere solo quel blocco specifico da un nodo Bitcoin. Il blocco può quindi essere ricollegato al blocco di genesi seguendo all'indietro la catena dell'intestazione del blocco (verificando la proof of work/prova di lavoro), confermando che si tratta di un blocco appartenente alla blockchain di Bitcoin.

Successivamente, il verificatore utilizza il numero di indice della transazione per identificare l'ID della transazione contenente il canale di pagamento. La maggior parte delle moderne

librerie Bitcoin consentirà l'indicizzazione nella transazione di un blocco in base all'indice della transazione all'interno del blocco più grande.

Poi, il verificatore utilizza una libreria Bitcoin per estrarre la transazione pertinente in base al suo indice all'interno del blocco. Il verificatore convaliderà la transazione (controllando che sia firmata correttamente e produca lo stesso ID di transazione quando sottoposto a hash).

Successivamente, il verificatore estrarrà l'output Pay-to-Witness-Script-Hash (P2WSH) a cui fa riferimento il numero di indice di output dell'ID canale breve. Questo è l'indirizzo dell'output di finanziamento del canale. Inoltre, il verificatore assicurerà che la dimensione del presunto canale corrisponda al valore dell'output prodotto all'indice di output specificato.

Infine, il verificatore ricostruirà lo script multisig da `bitcoin_key_1` e `bitcoin_key_2` e confermerà che produce lo stesso indirizzo dell'output.

Il verificatore ha ora verificato in modo indipendente che il canale di pagamento nell'annuncio è finanziato e confermato sulla blockchain di Bitcoin!

Il Messaggio `channel_update`

Il terzo e ultimo messaggio utilizzato nel protocollo gossip è il messaggio `channel_update`. Due di questi vengono generati per ciascun canale di pagamento (uno da ciascun partner di canale) che annunciano le tariffe di instradamento, le aspettative di timelock e le capacità.

Il messaggio `channel_update` contiene anche un timestamp, che consente a un nodo di aggiornare le sue commissioni di instradamento e altre aspettative e capacità inviando un nuovo messaggio `channel_update` con un timestamp più alto (successivo) che sostituisce qualsiasi aggiornamento precedente.

Il messaggio `channel_update` contiene i seguenti campi:

`signature`

> Una firma digitale corrispondente alla chiave pubblica del nodo, per autenticare l'origine e l'integrità dell'aggiornamento del canale

`chain_hash`

> L'hash del blocco genesi della catena che contiene il canale

short_channel_id

L'ID canale breve per identificare il canale

timestamp

Il timestamp di questo aggiornamento, per consentire ai destinatari di sequenziare gli aggiornamenti e sostituire gli aggiornamenti precedenti

message_flags

Un campo di bit che indica la presenza di campi aggiuntivi nel messaggio `channel_update`

channel_flags

Un campo di bit che mostra la direzione del canale e altre opzioni del canale

cltv_expiry_delta

Le aspettative delta timelock di questo nodo per il routing (vedi il Capitolo 10)

htlc_minimum_msat

L'importo minimo HTLC che verrà instradato

fee_base_msat

La commissione base che verrà addebitata per l'instradamento

fee_proportional_millionths

La commissione proporzionale che verrà addebitata per l'instradamento

`htlc_maximum_msat (option_channel_htlc_max)`

L'importo massimo che verrà instradato

Un nodo che riceve il messaggio `channel_update` può allegare questi metadati al bordo del grafo dei canali per abilitare il pathfinding, come vedremo in nel Capitolo 12.

Manutenzione Continua del Grafo dei Canali

La costruzione di un grafo di canali non è un evento una tantum, ma piuttosto un'attività continua. Quando un nodo esegue il bootstrap nella rete, inizierà a ricevere "pettegolezzi", sotto forma di tre messaggi di aggiornamento. Utilizzerà questi messaggi per iniziare immediatamente a costruire un grafo dei canali convalidato.

Più informazioni riceve un nodo, migliore diventa la sua "mappa" della rete Lightning e più efficace può essere nella ricerca del percorso e nella consegna dei pagamenti.

Un nodo non aggiungerà solo informazioni al grafo dei canali. Terrà anche traccia dell'ultima volta che un canale è stato aggiornato ed eliminerà i canali "obsoleti" che non sono stati aggiornati da più di due settimane. Infine, se vede che qualche nodo non ha più canali, rimuoverà anche quel nodo.

Le informazioni raccolte dal protocollo di gossip non sono le uniche informazioni che possono essere memorizzate nel grafo dei canali. Diverse implementazioni di nodi Lightning possono collegare altri metadati a nodi e canali. Ad esempio, alcune implementazioni di nodi calcolano un "punteggio" che valuta la "qualità" di un nodo come peer di instradamento. Questo punteggio viene utilizzato come parte del pathfinding per dare priorità o depriorizzare i percorsi.

Conclusione

In questo capitolo, abbiamo appreso come i nodi Lightning si scoprono a vicenda, scoprono e aggiornano il loro stato di nodo e comunicano tra di loro. Abbiamo imparato come vengono creati e mantenuti i grafi dei canali e abbiamo esplorato alcuni modi in cui Lightning Network scoraggia attori malintenzionati o nodi disonesti dall'inviare spam alla rete.

Capitolo 12. Pathfinding e Consegna dei Pagamenti

La consegna dei pagamenti su Lightning Network dipende dalla ricerca di un percorso dal mittente al destinatario, un processo chiamato *pathfinding* (ricerca del percorso). Poiché l'instradamento viene effettuato dal mittente, il mittente deve trovare un percorso adatto per raggiungere la destinazione. Questo percorso viene quindi codificato in una cipolla, come abbiamo visto nel Capitolo 10.

In questo capitolo esamineremo il problema del pathfinding, capiremo come l'incertezza sui bilanciamenti dei canali complichi questo problema e vedremo come una tipica implementazione del pathfinding tenta di risolverlo.

Pathfinding nella Suite di Protocolli Lightning

Pathfinding, selezione del percorso, pagamenti in più parti (MPP) e il ciclo di prova ed errore del tentativo di pagamento occupano la maggior parte del livello di pagamento nella parte superiore della suite di protocolli.

Questi componenti sono evidenziati da una struttura nella suite di protocolli, mostrata in Figura 12-1.

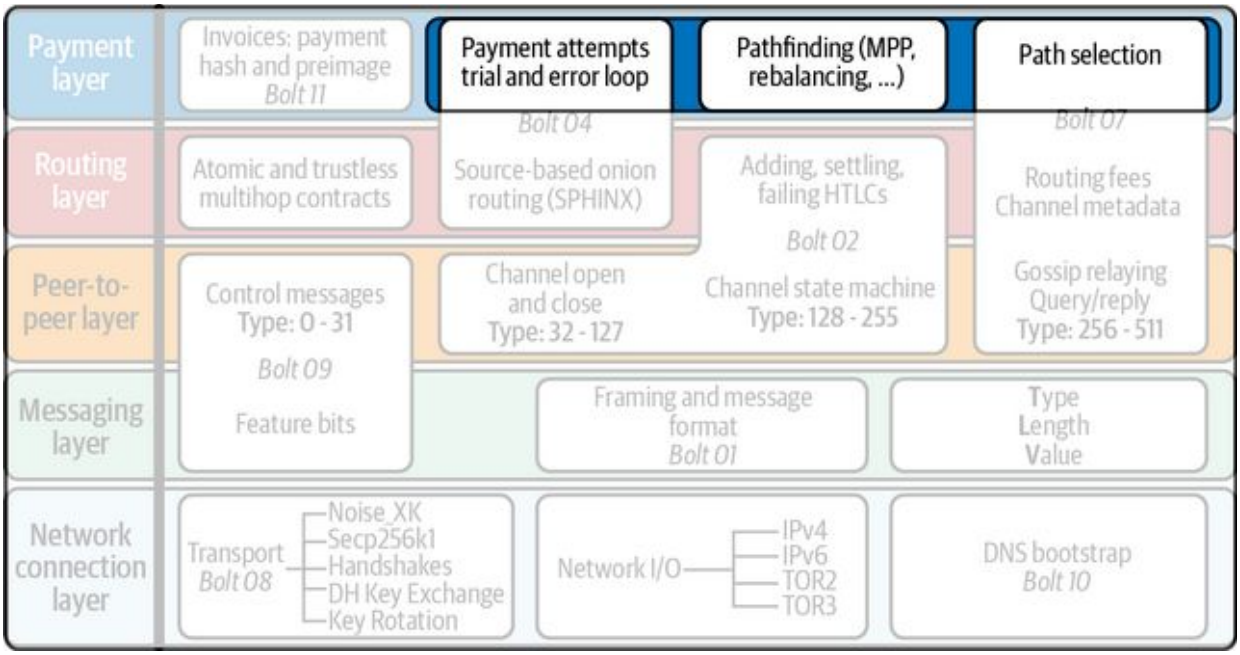

Figura 12-1. Consegna dei pagamenti nella suite di protocolli Lightning

Dov'è BOLT?

Finora abbiamo esaminato diverse tecnologie che fanno parte di Lightning Network e abbiamo visto le loro specifiche esatte come parte di uno standard BOLT. Potresti essere sorpreso di scoprire che il pathfinding non fa parte dei BOLT!

Questo perché il pathfinding non è un'attività che richiede alcuna forma di coordinamento o interoperabilità tra diverse implementazioni. Come abbiamo visto, il percorso è selezionato dal mittente. Anche se i dettagli di instradamento sono specificati in dettaglio nei BOLT, la scoperta e la selezione del percorso sono lasciate interamente al mittente. Ogni implementazione può scegliere una strategia/algoritmo diverso per trovare i percorsi. In effetti, le diverse implementazioni nodo/client e portafoglio possono persino competere e utilizzare il loro algoritmo di ricerca del percorso come punto di differenziazione.

Pathfinding: Quale Problema Stiamo Risolvendo?

Il termine pathfinding può essere in qualche modo fuorviante perché implica la ricerca di un *singolo percorso* che collega due nodi. All'inizio, quando Lightning Network era piccolo e mal

interconnesso, il problema era quello di trovare un modo per unire i canali di pagamento e raggiungere il destinatario.

Poiché LN è cresciuto in modo esplosivo, la natura del problema del pathfinding è cambiata. A Febbraio 2023, data di pubblicazione della traduzione italiana di questo libro, la rete Lightning è composta da 16.000 nodi collegati da almeno 76.500 canali pubblici con una capacità complessiva di quasi 5.500 BTC. Un nodo ha in media 9,5 canali, mentre i primi 10 nodi più connessi hanno tra 1000 e 2.500 canali ciascuno. Una visualizzazione di solo un piccolo sottoinsieme del grafico del canale LN è mostrata in Figura 12-2.

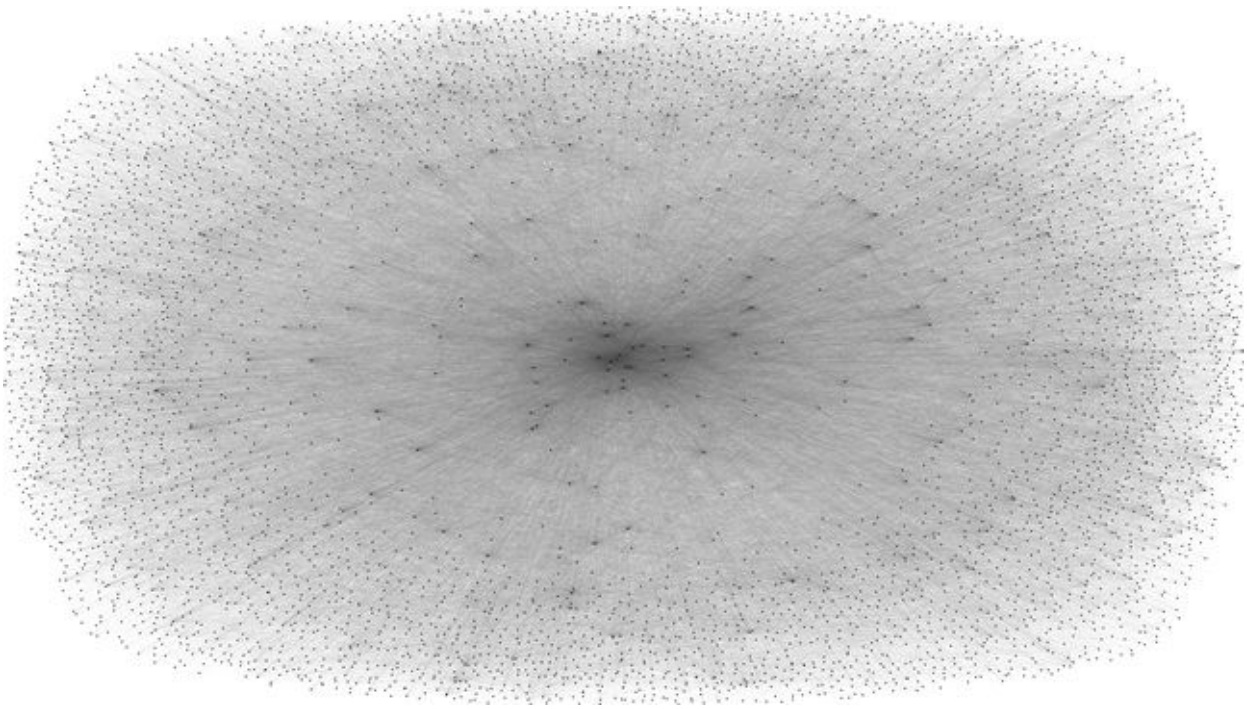

Figura 12-2. Una visualizzazione di una parte della rete Lightning a Febbraio 2023

NOTA	La visualizzazione della rete in Figura 12-2 è stata prodotta con uno script Python che puoi trovare in *code/lngraph* nel repository del libro.

Se il mittente e il destinatario sono collegati ad altri nodi ben collegati e dispongono di almeno un canale con capacità adeguata, ci saranno migliaia di percorsi. Il problema diventa selezionare il percorso *migliore* che avrà successo nella consegna del pagamento, da un

elenco di migliaia di possibili percorsi.

Selezione del Percorso Migliore

Per selezionare il percorso migliore, dobbiamo prima definire cosa intendiamo per "migliore". Ci possono essere molti criteri diversi, come ad esempio:

- Percorsi con sufficiente liquidità. Ovviamente se un percorso non ha abbastanza liquidità per instradare il nostro pagamento, allora non è un percorso adatto.
- Percorsi a basso costo. Se abbiamo più candidati, potremmo voler selezionare quelli con tariffe più basse.
- Percorsi con timelock brevi. Potremmo voler evitare di bloccare i nostri fondi per troppo tempo e quindi selezionare percorsi con timelock più brevi.

Tutti questi criteri possono essere desiderabili in una certa misura e selezionare percorsi favorevoli in molte dimensioni non è un compito facile. Problemi di ottimizzazione come questo possono essere troppo complessi da risolvere per la soluzione "migliore", ma spesso possono essere risolti per qualche approssimazione dell'ottimale, il che è una buona notizia perché altrimenti il pathfinding sarebbe un problema intrattabile.

Pathfinding in Matematica e Informatica

Il pathfinding su Lightning Network rientra in una categoria generale di *teoria dei grafi* in matematica e nella categoria più specifica di *attraversamento di grafi* in informatica.

Una rete come Lightning Network può essere rappresentata come un costrutto matematico chiamato *grafo*, in cui i *nodi* sono collegati tra loro da *bordi* (equivalenti ai canali di pagamento). Lightning Network forma un *grafo orientato* perché i nodi sono collegati in modo *asimmetrico*, poiché il saldo del canale è suddiviso tra i due partner di canale e la liquidità dei pagamenti è diversa in ciascuna direzione. Un grafo orientato con vincoli di capacità numerica sui bordi è chiamato *rete di flusso*, un costrutto matematico utilizzato per ottimizzare il trasporto e altre reti simili. Le reti di flusso possono essere utilizzate come framework quando le soluzioni devono raggiungere un flusso specifico riducendo al minimo i costi, noto come problema di flusso a costo minimo (MCFP).

Capacità, Equilibrio, Liquidità

Per comprendere meglio il problema del trasporto dei sats dal punto A al punto B, dobbiamo definire meglio tre termini importanti: capacità, equilibrio e liquidità. Utilizziamo questi termini per descrivere la capacità di un canale di pagamento di instradare un pagamento.

In un canale di pagamento che collega A←→B:

Capacità

> La quantità complessiva di satoshi finanziati nel multisig 2-di-2 con la transazione di finanziamento. Rappresenta la quantità massima di valore detenuta nel canale. La capacità del canale è annunciata dal protocollo di gossip ed è nota ai nodi.

Saldo

> La quantità di satoshi detenuti da ciascun partner di canale che possono essere inviati all'altro partner. Un sottoinsieme del saldo di A può essere inviato nella direzione (A→B) verso il nodo B. Un sottoinsieme del saldo di B può essere inviato nella direzione opposta (A←B).

Liquidità

> Il saldo disponibile (sottoinsieme) che può effettivamente essere inviato attraverso il canale in una direzione. La liquidità di A è uguale al saldo di A meno la riserva di canale e qualsiasi HTLC in sospeso impegnato da A.

L'unico valore noto alla rete (tramite annunci di gossip) è la capacità aggregata del canale. Una parte sconosciuta di tale capacità viene distribuita come saldo di ciascun partner. Alcuni sottoinsiemi di quel saldo sono disponibili per l'invio attraverso il canale in una direzione:

> capacità = saldo(A) + saldo(B)
>
> liquidità(A) = saldo(A) – channel_reserve(A) – pending_HTLCs(A)

Incertezza degli Saldi

Se conoscessimo gli esatti saldi di ogni canale, potremmo calcolare uno o più percorsi di pagamento utilizzando uno qualsiasi degli algoritmi standard di ricerca dei percorsi insegnati nei programmi di informatica. Non conosciamo i saldi del canale, conosciamo solo la capacità aggregata del canale, che viene pubblicizzata dai nodi negli annunci del canale. Affinché un pagamento vada a buon fine, deve esserci un saldo adeguato sul lato di invio del canale. Se non sappiamo come viene distribuita la capacità tra i partner di canale, non sappiamo se c'è abbastanza equilibrio nella direzione in cui stiamo cercando di inviare il pagamento.

I saldi non vengono annunciati negli aggiornamenti del canale per due motivi: privacy e scalabilità. In primo luogo, l'annuncio dei saldi ridurrebbe la privacy di LN perché consentirebbe la sorveglianza dei pagamenti mediante analisi statistiche delle variazioni dei saldi. In secondo luogo, se i nodi annunciassero saldi (globalmente) ad ogni pagamento, il ridimensionamento di LN sarebbe negativo quanto quello delle transazioni Bitcoin on-chain che vengono trasmesse a tutti i partecipanti. Pertanto, i saldi non vengono annunciati. Per risolvere il problema del pathfinding di fronte all'incertezza dei saldi, abbiamo bisogno di strategie innovative di pathfinding. Queste strategie devono essere strettamente correlate all'algoritmo di routing utilizzato, che è l'onion routing basato sull'origine in cui è responsabilità del mittente trovare un percorso attraverso la rete.

Il problema dell'incertezza può essere descritto matematicamente come un *intervallo di liquidità*, indicando i limiti inferiori e superiori della liquidità sulla base delle informazioni note. Poiché conosciamo la capacità del canale e conosciamo il saldo di riserva del canale (il saldo minimo consentito su ciascuna estremità), la liquidità può essere definita come:

min(liquidità) = channel_reserve

max(liquidità) = capacità – channel_reserve

o come intervallo:

channel_reserve <= liquidità <= (capacità – channel_reserve)

Il nostro intervallo di incertezza sulla liquidità del canale è l'intervallo tra la liquidità minima e massima possibile. Questo è sconosciuto alla rete, ad eccezione dei due partner di canale. Tuttavia, come vedremo, possiamo utilizzare gli HTLC falliti restituiti dai nostri tentativi di pagamento per aggiornare la nostra stima di liquidità e ridurre l'incertezza. Se, ad esempio, otteniamo un codice di errore HTLC che ci dice che un canale non può soddisfare un HTLC inferiore alla nostra stima per la liquidità massima, ciò significa che la liquidità massima può

essere aggiornata all'importo dell'HTLC fallito. Se pensiamo che la liquidità possa gestire un HTLC di *N* satoshi e scopriamo che non riesce a fornire M satoshi (dove *M* è più piccolo), allora possiamo aggiornare la nostra stima a *M*–1 come limite superiore.

Complessità del Pathfinding

Trovare un percorso attraverso un grafo è un problema che i computer moderni possono risolvere in modo piuttosto efficiente. Gli sviluppatori scelgono principalmente la ricerca in ampiezza se i bordi hanno tutti lo stesso peso. Nei casi in cui i bordi non hanno lo stesso peso, viene utilizzato un algoritmo basato sull'algoritmo di Dijkstra, come A* (pronunciato "A-star"). Nel nostro caso i pesi dei bordi possono rappresentare le spese di instradamento. Verranno inclusi nella ricerca solo i bordi con una capacità superiore alla quantità da inviare. In questa forma di base, il pathfinding su Lightning Network è molto semplice e diretto.

Tuttavia, la liquidità del canale è sconosciuta al mittente. Questo trasforma il nostro semplice problema di informatica teorica in un problema del mondo reale piuttosto complesso. Ora dobbiamo risolvere un problema di individuazione del percorso con una conoscenza parziale. Ad esempio, sospettiamo quali bordi potrebbero essere in grado di inoltrare un pagamento perché la loro capacità sembra abbastanza grande. Ma non possiamo esserne certi se non lo proviamo o chiediamo direttamente ai proprietari del canale. Anche se potessimo chiedere direttamente ai proprietari del canale, il loro equilibrio potrebbe cambiare nel momento in cui avremo chiesto ad altri, calcolato un percorso, costruito una cipolla e inviata. Non solo disponiamo di informazioni limitate, ma le informazioni che abbiamo sono dinamiche e potrebbero cambiare in qualsiasi momento a nostra insaputa.

Semplifichiamo

Il meccanismo di ricerca dei percorsi implementato su LN consiste nel creare prima un elenco di percorsi candidati, filtrati e ordinati in base a una funzione. Quindi, il nodo o il wallet esaminerà i percorsi (tentando di consegnare un pagamento) in un ciclo di tentativi ed errori finché non viene trovato un percorso che consegna correttamente il pagamento.

NOTA	Questo sondaggio viene eseguito dal nodo Lightning o dal portafoglio e non viene osservato direttamente dall'utente. Tuttavia, l'utente potrebbe sospettare che sia in corso un sondaggio se il pagamento non viene

	completato immediatamente.

Sebbene il sondaggio cieco non sia ottimale e lasci ampi margini di miglioramento, va notato che anche questa strategia semplicistica funziona sorprendentemente bene per pagamenti più piccoli e nodi ben collegati.

La maggior parte delle implementazioni di nodi e wallet Lightning migliora questo approccio ordinando/ponderando l'elenco dei percorsi candidati. Alcune implementazioni ordinano i percorsi candidati in base al costo (commissioni) o ad una combinazione di costo e capacità.

Pathfinding e Processo di Consegna dei Pagamenti

La ricerca del percorso e la consegna del pagamento comportano diversi passaggi, che elenchiamo di seguito. Implementazioni diverse possono utilizzare algoritmi e strategie diverse, ma è probabile che i passaggi di base effettuati da un software siano molto simili:

1. Crea di un *grafo dei canali* da annunci e aggiornamenti contenente la capacità di ciascun canale e filtra il grafico, ignorando eventuali canali con capacità insufficiente per la quantità che vogliamo inviare.
2. Trova i percorsi che collegano il mittente al destinatario.
3. Ordina i percorsi in base a un certo peso (questo potrebbe far parte dell'algoritmo del passaggio precedente).
4. Prova ogni percorso in ordine fino a quando il pagamento va a buon fine (il ciclo di tentativi ed errori).
5. Facoltativamente, utilizza i ritorni di errore HTLC per aggiornare il nostro grafo, riducendo l'incertezza.

Possiamo raggruppare questi passaggi in tre attività principali:

- Costruzione del grafo dei canali
- Pathfinding (filtrato e ordinato da alcune euristiche)
- Tentativo/i di pagamento

Queste tre attività possono essere ripetute in un *round di pagamento* se utilizziamo i ritorni

di errore per aggiornare il grafo o se stiamo eseguendo pagamenti in più parti.

Nelle prossime sezioni esamineremo ciascuno di questi passaggi in modo più dettagliato, nonché strategie di pagamento più avanzate.

Costruzione del Grafo dei Canali

Nel Capitolo 11 abbiamo visto i tre messaggi principali che i nodi usano nei loro pettegolezzi: `node_announcement`, `channel_announcement` e `channel_update`. Questi tre messaggi consentono a qualsiasi nodo di costruire gradualmente una "mappa" di LN sotto forma di un grafo. Ciascuno di questi messaggi fornisce un'informazione fondamentale per il grafo:

`node_announcement`

> Questo contiene le informazioni su un nodo sulla rete Lightning, come l'ID del nodo (chiave pubblica), l'indirizzo di rete (ad es. IPv4/6 o Tor), capacità/caratteristiche, ecc.

`channel_announcement`

> Contiene la capacità e l'ID del canale di un canale pubblico (annunciato) tra due nodi e la prova dell'esistenza e della proprietà del canale.

`channel_update`

> Contiene le aspettative di pagamento e timelock (CLTV) di un nodo per l'instradamento di un pagamento in uscita (dal punto di vista di quel nodo) su un canale specificato.

In termini di un grafo matematico, il `node_announcement` è l'informazione necessaria per creare i nodi o *vertici* del grafo. Il `channel_announcement` ci permette di creare i *bordi* del grafo che rappresentano i canali di pagamento. Poiché ogni direzione del canale di pagamento ha il proprio saldo, creiamo un grafo orientato. Il `channel_update` ci consente di incorporare commissioni e timelock per impostare il costo o il *peso* dei bordi del grafo.

A seconda dell'algoritmo che useremo per il pathfinding, potremmo stabilire un numero di diverse funzioni di costo per i bordi del grafo.

Per ora, ignoriamo la funzione di costo e stabiliamo semplicemente un grafo dei canali che mostri nodi e canali utilizzando i messaggi `node_announcement` e `channel_announcement`.

In questo capitolo vedremo come Selena tenta di trovare un modo per pagare a Rashid un milione di sats. Per iniziare, Selena sta costruendo un grafo dei canali utilizzando le informazioni dei pettegolezzi di LN per scoprire nodi e canali. Selena esplorerà quindi il suo grafo dei canali per trovare un percorso ed inviare un pagamento a Rashid.

Questo è il grafo dei canali *di Selena*. Non esiste *il* grafo dei canali, c'è sempre e solo *un grafo dei canali*, ed è sempre dalla prospettiva del nodo che lo ha costruito.

NOTA	Selena non costruisce un grafo dei canali solo quando invia un pagamento. Piuttosto, il nodo di Selena costruisce e aggiorna *continuamente* un grafo dei canali. Dal momento in cui il nodo di Selena si avvia e si connette a qualsiasi peer sulla rete, parteciperà al gossip e utilizzerà ogni messaggio per imparare il possibile sulla rete.

La relazione mappa-territorio

Dalla pagina di Wikipedia sulla relazione mappa-territorio, "la relazione mappa-territorio descrive la relazione tra un oggetto e una rappresentazione di quell'oggetto, come nella relazione tra un territorio geografico e una sua mappa".

La relazione mappa-territorio è illustrata al meglio in "Sylvie and Bruno Concluded", un racconto di Lewis Carroll che descrive una mappa fittizia che è una scala 1:1 del territorio che mappa, avendo quindi una precisione perfetta ma diventando completamente inutile in quanto coprirebbe l'intero territorio se dispiegato.

Cosa significa questo per Lightning Network? LN è il territorio, e un grafo dei canali è una mappa di quel territorio.

Mentre potremmo immaginare un grafo del canale teorico (ideale platonico) che rappresenta la mappa completa e aggiornata di Lightning Network, tale mappa è semplicemente il Lightning Network stesso. Ogni nodo ha il proprio grafo dei canali che è costruito dagli annunci ed è necessariamente incompleto, errato e obsoleto!

La mappa non può mai descrivere completamente e accuratamente il territorio.

Selena ascolta i messaggi `node_announcement` e scopre altri quattro nodi (oltre a Rashid, il

destinatario previsto). Il grafo risultante rappresenta una rete di sei nodi: Selena e Rashid sono rispettivamente il mittente e il destinatario; Alice, Bob, Xavier e Yan sono nodi intermedi. Il grafo iniziale di Selena è solo un elenco di nodi, mostrato in Figura 12-3.

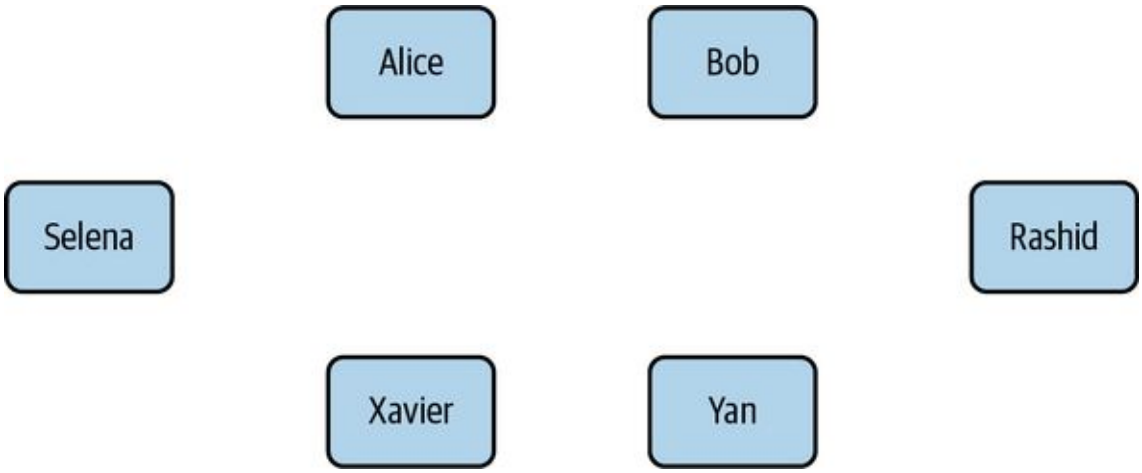

Figura 12-3. Annunci di nodo

Selena riceve anche sette messaggi `channel_announcement` con le corrispondenti capacità di canale, che le consentono di costruire una "mappa" di base della rete, mostrata in Figura 12-4. (I nomi Alice, Bob, Selena, Xavier, Yan e Rashid sono stati sostituiti dalle loro iniziali: A, B, S, X e R, rispettivamente.)

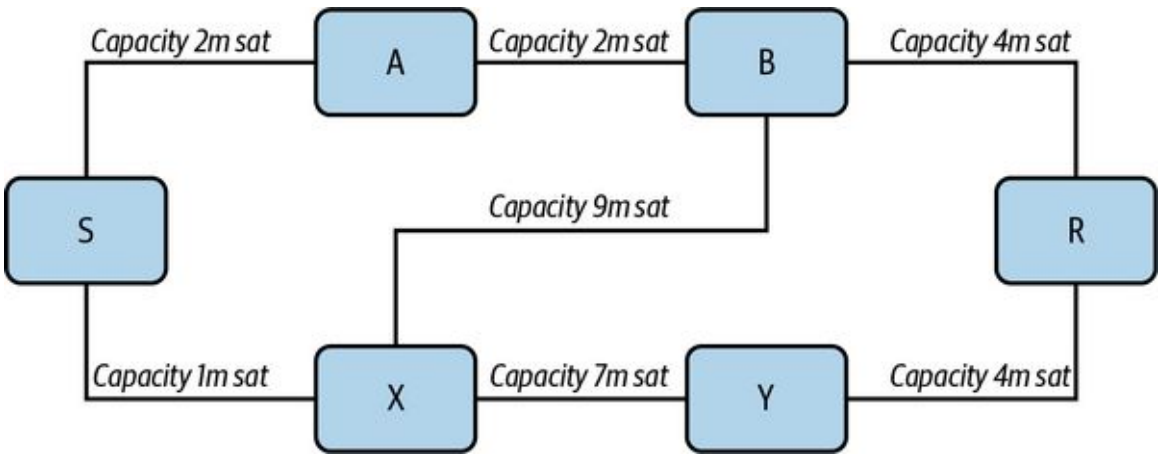

Figura 12-4. Il grafo dei canali

Incertezza nel grafo dei canali

Come puoi vedere in Figura 12-4, Selena non conosce nessuno dei saldi dei canali. Il suo grafo dei canali iniziale contiene il più alto livello di incertezza.

Ma aspetta: Selena conosce *alcuni* saldi del canale! Conosce i saldi dei canali che il suo nodo ha creato con altri nodi. Anche se questo non sembra molto, in realtà è un'informazione molto importante per costruire un percorso: Selena conosce l'effettiva liquidità dei propri canali. Aggiorniamo il grafo dei canali per mostrare queste informazioni. Useremo un simbolo "?" per rappresentare i saldi sconosciuti, come mostrato in Figura 12-5.

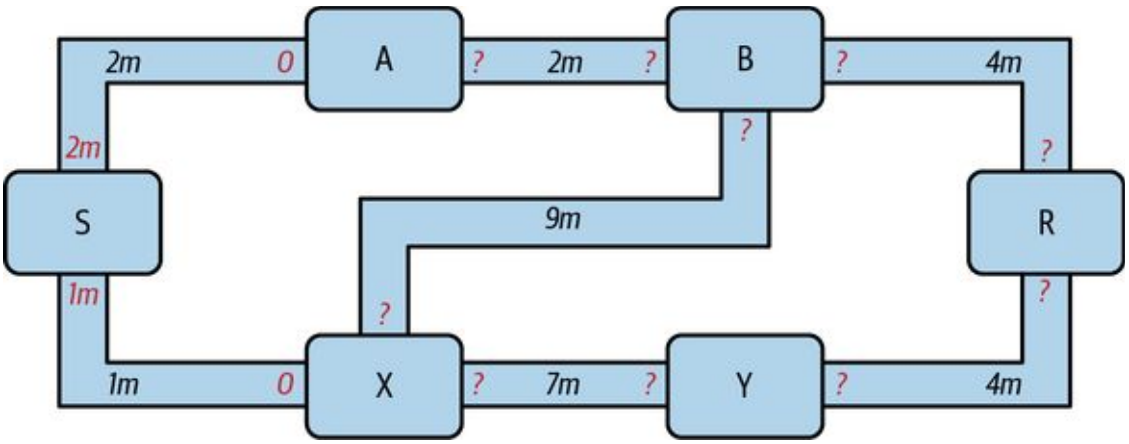

Figura 12-5. Grafo dei canali con saldi noti e sconosciuti

Mentre il simbolo "?" sembra un serio problema, una mancanza di certezza non è la stessa cosa di una completa ignoranza. Possiamo *quantificare* l'incertezza e ridurla aggiornando il grafo con gli HTLC riusciti/falliti che tentiamo.

L'incertezza può essere quantificata, perché conosciamo la liquidità massima e minima possibile e possiamo calcolare le probabilità per intervalli più piccoli (più precisi).

Una volta che tentiamo di inviare un HTLC, possiamo saperne di più sui saldi dei canali: se ci riusciamo, allora il saldo era *almeno* sufficiente per trasportare l'importo specifico. Nel frattempo, se riceviamo un errore di "guasto temporaneo del canale", il motivo più probabile è una mancanza di liquidità per l'importo specifico.

NOTA	Potresti pensare: "Che senso ha imparare da un HTLC di successo?" Dopotutto, se ha avuto successo, abbiamo "finito"... Considera che

	potremmo inviare una parte di un pagamento in più parti. Potremmo anche inviare altri pagamenti in una sola parte entro breve tempo. Tutto ciò che impariamo sulla liquidità è utile per il prossimo tentativo!

Incertezza e Probabilità di Liquidità

Per quantificare l'incertezza della liquidità di un canale, possiamo applicare la teoria della probabilità. Un modello di base della probabilità di consegna del pagamento porterà ad alcune conclusioni piuttosto ovvie, ma importanti:

- I pagamenti più piccoli hanno maggiori possibilità di consegna riuscita attraverso un percorso.
- I canali di maggiore capacità ci daranno maggiori possibilità di consegna del pagamento per un importo specifico.
- Maggiore è il numero di canali (hop), minore è la possibilità di successo.

Anche se questi possono essere ovvi, hanno implicazioni importanti, in particolare per l'uso di pagamenti in più parti. La matematica non è difficile da seguire.

Usiamo la teoria della probabilità per vedere come siamo arrivati a queste conclusioni...

Innanzitutto, supponiamo che un canale con capacità c abbia liquidità su un lato con un valore sconosciuto compreso nell'intervallo di (0, c) o "intervallo tra 0 e c". Ad esempio, se la capacità è 5, la liquidità sarà compresa nell'intervallo (0, 5). Ora, da questo vediamo che se vogliamo inviare 5 satoshi, la nostra possibilità di successo è solo 1 su 6 (16,66%), perché riusciremo solo se la liquidità è esattamente 5.

Più semplicemente, se i possibili valori per la liquidità sono 0, 1, 2, 3, 4 e 5, solo uno di quei sei possibili valori sarà sufficiente per inviare il nostro pagamento. Per continuare questo esempio, se l'importo del nostro pagamento fosse 3, avremmo successo se la liquidità fosse 3, 4 o 5. Quindi le nostre possibilità di successo sono 3 su 6 (50%). Espressa in matematica, la funzione di probabilità di successo per un singolo canale è:

$P_c(a) = (c + 1 - a) / (c + 1)$

dove a è la quantità e c è la capacità.

Dall'equazione vediamo che se l'importo è vicino a 0, la probabilità è vicina a 1, mentre se

l'importo supera la capacità, la probabilità è zero.

In altre parole: "I pagamenti più piccoli hanno maggiori possibilità di consegna riuscita" o "I canali con una capacità maggiore ci danno migliori possibilità di consegna per un importo specifico" e "Non puoi inviare un pagamento su un canale con capacità insufficiente".

Ora pensiamo alla probabilità di successo attraverso un percorso composto da più canali. Supponiamo che il nostro primo canale abbia una probabilità di successo del 50% (P = 0,5). Quindi, se il nostro secondo canale ha una probabilità di successo del 50% (P = 0,5), è intuitivo che la nostra possibilità complessiva è del 25% (P = 0,25).

Possiamo esprimerlo come un'equazione che calcola la probabilità di successo di un pagamento come prodotto delle probabilità per ogni canale nel percorso:

$P_{pagamento} = \prod_{i=1}^{n} P_i$

Dove P_i è la probabilità di successo su un percorso o canale e $P_{pagamento}$ è la probabilità complessiva di un pagamento riuscito su tutti i percorsi/canali.

Dall'equazione vediamo che poiché la probabilità di successo su un singolo canale è sempre minore o uguale a 1, la probabilità su molti canali *diminuirà in modo esponenziale*.

In altre parole, "Più canali (hop) usi, minori sono le possibilità di successo".

NOTA	C'è molta teoria matematica e modellazione dietro l'incertezza della liquidità nei canali. Il lavoro fondamentale sulla modellazione degli intervalli di incertezza della liquidità del canale può essere trovato nel documento "Security and Privacy of Lightning Network Payments with Uncertain Channel Balances" di (coautore di questo libro) Pickhardt et al.

Commissioni e Altre Metriche di Canale

Successivamente, il nostro mittente aggiungerà informazioni al grafo dai messaggi `channel_update` ricevuti dai nodi intermedi. Come promemoria, `channel_update` contiene una grande mole di informazioni su un canale e le aspettative di uno dei partner.

In Figura 12-6 vediamo come Selena può aggiornare il grafo dei canali in base ai messaggi `channel_update` da A, B, X e Y. Nota che l'ID canale e la direzione del canale (inclusi in `channel_flags`) dicono a Selena quale canale e quale direzione a cui si riferisce questo aggiornamento. Ogni partner di canale invia uno o più messaggi `channel_update` per

annunciare le proprie aspettative sulle commissioni e altre informazioni sul canale. Ad esempio, in alto a sinistra vediamo il `channel_update` inviato da Alice per il canale A—B e la direzione A-verso-B. Con questo aggiornamento, Alice dice alla rete quante commissioni addebiterà per instradare un HTLC a Bob su quel canale specifico. Bob potrebbe annunciare un aggiornamento del canale (non mostrato in questo diagramma) per la direzione opposta con aspettative tariffarie completamente diverse. Qualsiasi nodo può inviare un nuovo `channel_update` per modificare le tariffe o le aspettative di timelock in qualsiasi momento.

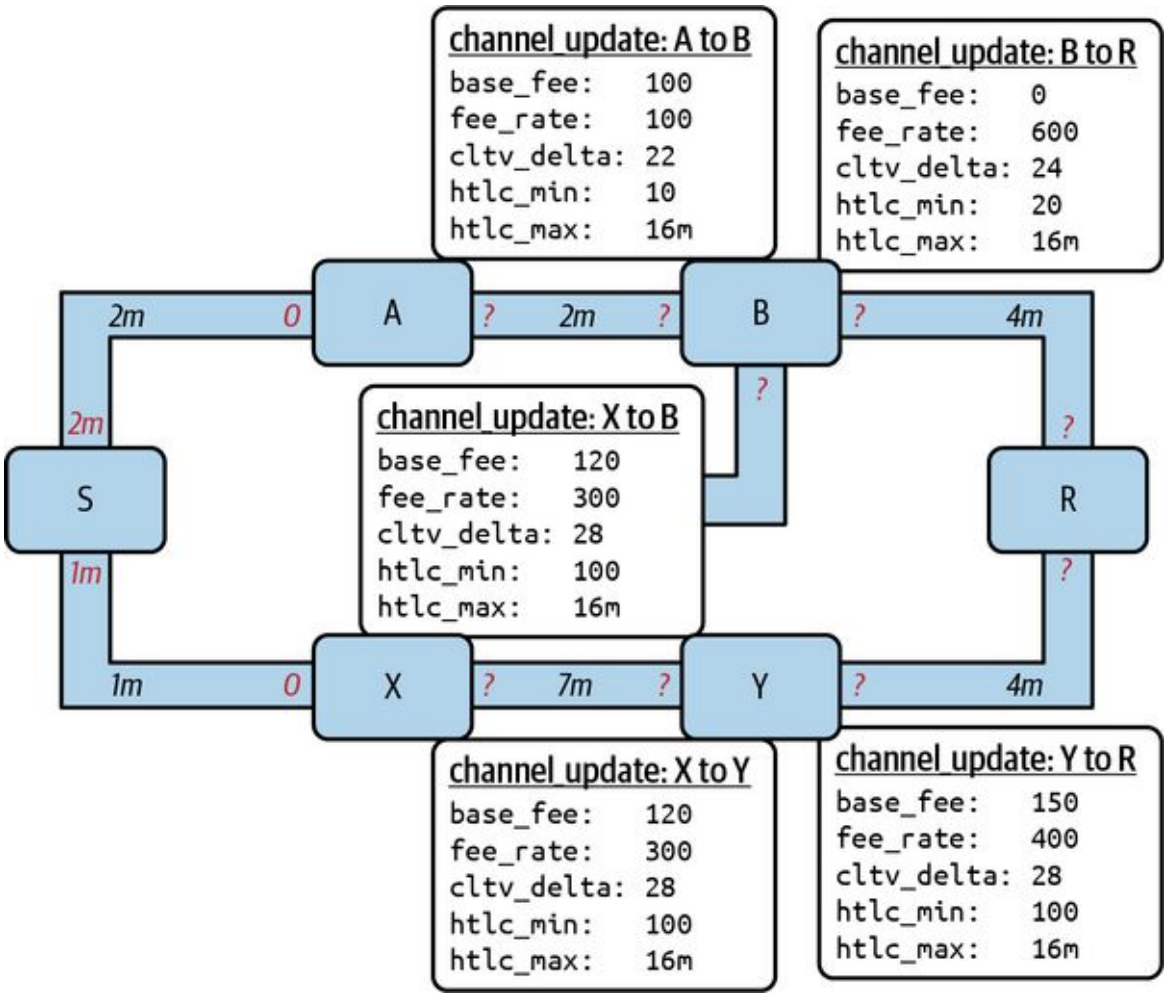

Figura 12-6. Commissioni del grafo dei canali e altre metriche di canale

Le informazioni sulla tariffa e sul timelock sono molto importanti, non solo come metriche di selezione del percorso. Come abbiamo visto nel Capitolo 10, il mittente deve sommare tariffe e timelock (`cltv_expiry_delta`) a ogni hop per creare la cipolla. Il processo di calcolo delle

tariffe avviene dal destinatario al mittente *a ritroso* lungo il percorso perché ogni hop intermediario si aspetta un HTLC in entrata con importo e timelock maggiori rispetto all'HTLC in uscita che invieranno all'hop successivo. Quindi, ad esempio, se Bob desidera 1.000 satoshi in commissioni e 30 blocchi di delta timelock di scadenza per inviare un pagamento a Rashid, allora quell'importo e il delta di scadenza devono essere aggiunti all'HTLC *da Alice*.

È inoltre importante notare che un canale deve avere una liquidità sufficiente non solo per l'importo del pagamento ma anche per le commissioni cumulative di tutti i salti successivi. Anche se il canale di Selena con Xavier (S→X) ha liquidità sufficiente per un pagamento di 1 milione di satoshi, *non ha* liquidità sufficiente se consideriamo le commissioni. Abbiamo bisogno di conoscere le commissioni perché verranno presi in considerazione solo i percorsi che hanno liquidità sufficiente *sia per il pagamento che per tutte le commissioni*.

Trovare Percorsi Candidati

Trovare un percorso adatto attraverso un grafo orientato come questo è un problema di informatica ben studiato (noto in generale come *problema del percorso più breve*), che può essere risolto da una varietà di algoritmi a seconda dell'ottimizzazione desiderata e dei vincoli delle risorse.

L'algoritmo più famoso per risolvere questo problema è stato inventato dal matematico olandese E. W. Dijkstra nel 1956, noto semplicemente come *algoritmo di Dijkstra*. Oltre all'algoritmo originale di Dijkstra, esistono molte varianti e ottimizzazioni, come A* ("A-star"), che è un algoritmo basato sull'euristica.

Come accennato, la "ricerca" deve essere applicata *all'indietro* per tenere conto delle commissioni accumulate dal destinatario al mittente. Pertanto, Dijkstra, A* o altri algoritmi cercherebbero un percorso dal destinatario al mittente, utilizzando commissioni, liquidità stimata e delta timelock (o una combinazione) come funzione di costo per ciascun hop.

Utilizzando uno di questi algoritmi, Selena calcola diversi possibili percorsi per Rashid, ordinati per percorso più breve:

1. S→A→B→R
2. S→X→Y→R
3. S→X→B→R

4. S→A→B→X→Y→R

Ma, come abbiamo visto in precedenza, il canale S→X non ha liquidità sufficiente per un pagamento di 1 milione di satoshi una volta considerate le commissioni. Quindi i percorsi 2 e 3 non sono praticabili. Ciò lascia i percorsi 1 e 4 come possibili percorsi per il pagamento.

Con due possibili percorsi, Selena è pronta a tentare la consegna!

Consegna del Pagamento (Ciclo di Prova ed Errore)

Il nodo di Selena avvia il ciclo di tentativi ed errori costruendo gli HTLC, costruendo la cipolla e tentando di consegnare il pagamento. Per ogni tentativo, ci sono tre possibili esiti:

- Un risultato positivo (`update_fulfill_htlc`)
- Un errore (`update_fail_htlc`)
- Un pagamento "bloccato" senza risposta (né successo né fallimento)

Se il pagamento non va a buon fine, è possibile riprovare un percorso diverso aggiornando il grafo (modificando le metriche di un canale) e ricalcolando un percorso alternativo.

Abbiamo esaminato cosa succede se il pagamento è "bloccato" in precedenza. Il dettaglio importante è che un pagamento bloccato è il risultato peggiore perché non possiamo riprovare con un altro HTLC poiché entrambi (quello bloccato e il nuovo tentativo) potrebbero eventualmente andare a buon fine e causare un doppio pagamento.

Primo Tentativo (Percorso n.1)

Selena tenta il primo percorso (S→A→B→R). Costruisce la cipolla e la invia, ma riceve un codice di errore dal nodo di Bob. Bob riporta un malfunzionamento temporaneo del canale con un `channel_update` che identifica il canale B→R come quello che non è in grado di inviare il pagamento. Questo tentativo è mostrato in Figura 12-7.

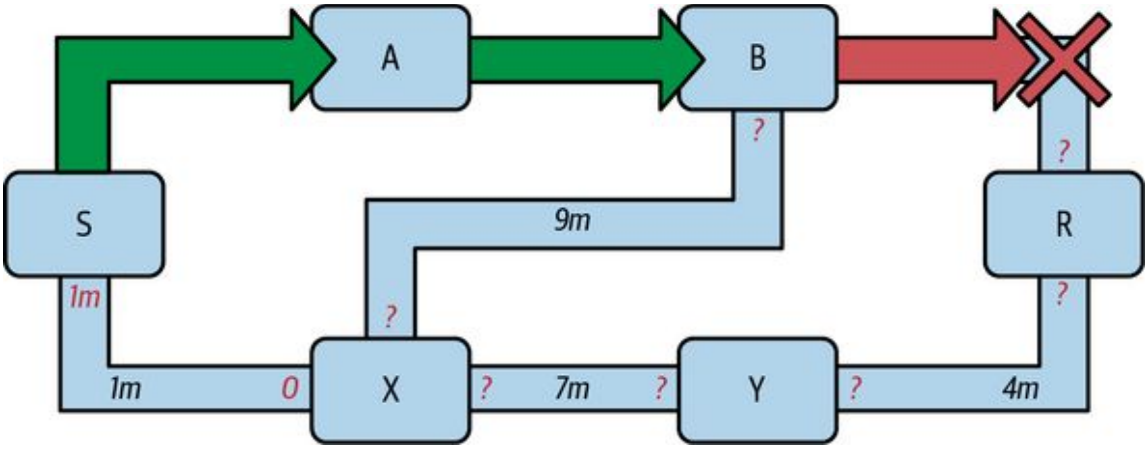

Figura 12-7. Il tentativo del percorso n. 1 fallisce

Imparare dagli errori

Da questo codice di errore, Selena dedurrà che Bob non ha liquidità sufficiente per consegnare il pagamento a Rashid su quel canale. È importante sottolineare che questo fallimento riduce l'incertezza della liquidità di quel canale! In precedenza, il nodo di Selena presumeva che la liquidità sul lato del canale di Bob fosse da qualche parte nell'intervallo (0, 4M). Ora può presumere che la liquidità sia nell'intervallo (0, 999999). Allo stesso modo, Selena può ora presumere che la liquidità di quel canale dalla parte di Rashid sia nell'intervallo (1M, 4M), anziché (0, 4M). Selena ha imparato molto da questo fallimento.

Secondo Tentativo (percorso n. 4)

Ora Selena tenta il quarto percorso candidato (S→A→B→X→Y→R). Questo è un percorso più lungo e comporterà più commissioni, ma è al momento l'opzione migliore per la consegna del pagamento.

Fortunatamente, Selena riceve un messaggio `update_fulfill_htlc` da Alice, che indica che il pagamento è andato a buon fine, come mostrato nella Figura 12-8.

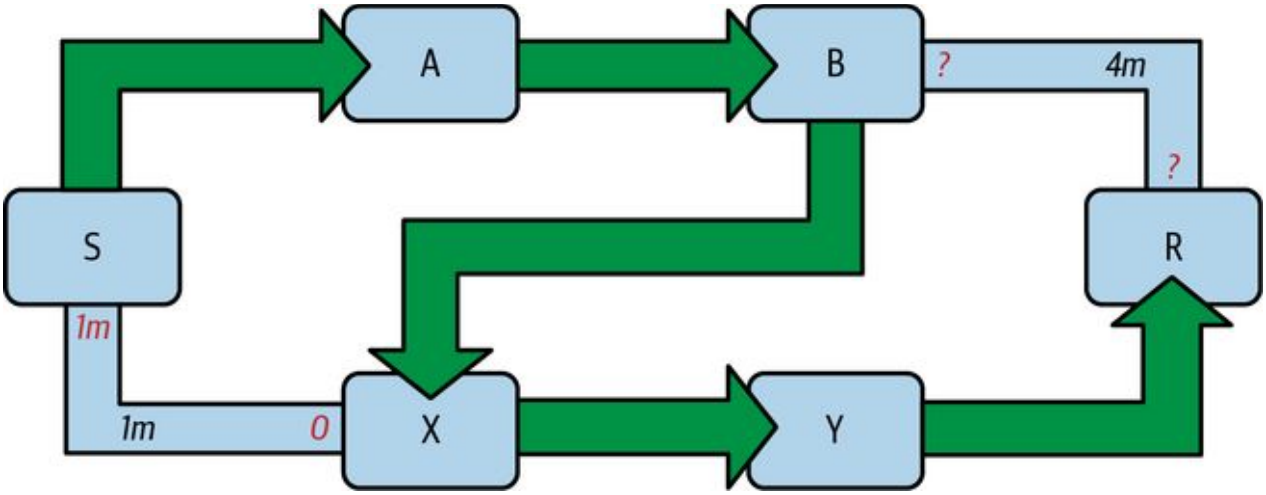

Figura 12-8. Il tentativo del percorso n. 4 ha successo

Imparare dal successo

Anche Selena ha imparato molto da questo pagamento andato a buon fine. Ora sa che tutti i canali sul percorso S→A→B→X→Y→R avevano liquidità sufficiente per consegnare il pagamento. Inoltre, ora sa che ciascuno di questi canali ha spostato l'importo HTLC (1 milione + commissioni) all'altra estremità del canale. Ciò consente a Selena di ricalcolare l'intervallo di liquidità sul lato ricevente di tutti i canali in quel percorso, sostituendo la liquidità minima con commissioni di 1+ milione.

Conoscenza stantia?

Selena ora ha una "mappa" molto migliore di LN (per quanto riguarda questi sette canali). Tale conoscenza sarà utile per eventuali pagamenti che Selena tenterà di effettuare in futuro.

Tuttavia, questa conoscenza diventa in qualche modo "stantia" man mano che gli altri nodi inviano o instradano i pagamenti. Selena non vedrà mai nessuno di questi pagamenti (a meno che non sia lei il mittente). Anche se coinvolta nell'instradamento di pagamenti, l'onion routing permette lei di vedere i cambiamenti solo per un hop (i suoi canali).

Pertanto, il nodo di Selena deve considerare per quanto tempo conservare questa conoscenza prima di presumere che sia stantia e non più utile.

Multipart Payments - Pagamenti in Più Parti

I *multipart payments* (MPP) sono una funzionalità introdotta su Lightning Network nel 2020 e sono già ampiamente disponibili. Gli MPP consentono di suddividere un pagamento in più *parti* che vengono inviate come HTLC su diversi percorsi diversi al destinatario previsto, preservando l'*atomicità* del pagamento complessivo. In questo contesto, l'atomicità significa che tutte le parti HTLC di un pagamento alla fine vengono soddisfatte o l'intero pagamento fallisce e tutte le parti HTLC falliscono. Non è possibile un pagamento parzialmente riuscito.

I pagamenti in più parti rappresentano un miglioramento significativo di Lightning Network perché consentono di inviare importi che non "si adattano" a nessun singolo canale, suddividendoli in importi più piccoli per i quali esiste liquidità sufficiente. Inoltre, è stato dimostrato che i pagamenti in più parti aumentano la probabilità che un pagamento vada a buon fine, rispetto a un pagamento a percorso singolo.

SUGGERIMENTO	Ora che MPP è disponibile, è meglio pensare a un pagamento a percorso singolo come una sotto-categoria di un MPP. In sostanza, un percorso singolo è solo un multipart di dimensione uguale a uno. Tutti i pagamenti possono essere considerati pagamenti in più parti a meno che l'entità del pagamento e la liquidità disponibile non rendano possibile la consegna con una singola parte.

Utilizzare MPP

MPP non è qualcosa che un utente selezionerà, ma piuttosto è una strategia di ricerca del percorso del nodo e di consegna del pagamento. Vengono implementati gli stessi passaggi di base: creazione di un grafo, selezione di percorsi e ciclo di tentativi ed errori. La differenza è che durante la selezione del percorso dobbiamo anche considerare come suddividere il pagamento per ottimizzare la consegna.

Nel nostro esempio possiamo vedere alcuni miglioramenti immediati al nostro problema di ricerca del percorso che diventano possibili con MPP. Innanzitutto, possiamo utilizzare il canale S→X che ha una liquidità conosciuta insufficiente per trasportare 1 milione di satoshi più le commissioni. Inviando una parte più piccola lungo quel canale, possiamo utilizzare percorsi che prima non erano disponibili. In secondo luogo, abbiamo la liquidità sconosciuta del canale B→R, che è insufficiente per trasportare l'importo di 1 milione, ma potrebbe

essere sufficiente per trasportare un importo inferiore.

Suddividere i Pagamenti

La questione fondamentale è come suddividere i pagamenti. Nello specifico, qual è il numero ottimale di frazionamenti e gli importi ottimali per ogni frazionamento?

Questa è un'area di ricerca in corso in cui stanno emergendo nuove strategie. I pagamenti in più parti portano a un approccio algoritmico diverso rispetto ai pagamenti a percorso singolo, anche se soluzioni a percorso singolo possono emergere da un'ottimizzazione in più parti (ovvero, un percorso singolo può essere la soluzione ottimale suggerita da un algoritmo di ricerca del percorso in più parti).

Se ricordi, abbiamo scoperto che l'incertezza della liquidità/saldi porta ad alcune conclusioni (in qualche modo ovvie) che possiamo applicare nel pathfinding MPP, vale a dire:

- I pagamenti più piccoli hanno maggiori possibilità di successo.
- Più canali usi, più le possibilità di successo diminuiscono (esponenzialmente).

Dalla prima di queste intuizioni, potremmo concludere che la suddivisione di un pagamento elevato (ad esempio, 1 milione di satoshi) in piccoli pagamenti aumenta la possibilità che ciascuno di questi pagamenti più piccoli abbia successo. Il numero di percorsi possibili con liquidità sufficiente sarà maggiore se inviamo importi inferiori.

Per portare questa idea all'estremo, perché non dividere il pagamento di 1 milione di satoshi in un milione di parti separate da un satoshi? Bene, la risposta sta nella nostra seconda intuizione: dal momento che useremmo più canali/percorsi per inviare il nostro milione di HTLC di singoli sats, le nostre possibilità di successo diminuirebbero in modo esponenziale.

Le due intuizioni precedenti creano un "punto debole" in cui possiamo massimizzare le nostre possibilità di successo: dividendo in pagamenti più piccoli ma non troppi!

Quantificare questo equilibrio ottimale di dimensione/numero di divisioni per un dato grafo di canali esula dagli scopi di questo libro, ma è un'area di ricerca attiva. Alcune implementazioni attuali utilizzano una strategia molto semplice di dividere il pagamento in due metà, quattro quarti, ecc.

SUGGERIMENTO	Per saperne di più sul problema di ottimizzazione noto come flussi di costo minimo coinvolti quando si suddividono i pagamenti in diverse

	dimensioni e li si allocano ai percorsi, vedi l'articolo "Optimally Reliable & Cheap Payment Flows on the Lightning Network" di (coautore di questo libro) René Pickhardt e Stefan Richter.

Nel nostro esempio, il nodo di Selena tenterà di dividere il pagamento di 1 milione di satoshi in 2 parti rispettivamente di 600.000 e 400.000 satoshi e di inviarle su 2 percorsi diversi. Questo è mostrato in Figura 12-9.

Poiché ora è possibile utilizzare il canale S→X e, fortunatamente per Selena, il canale B→R ha liquidità sufficiente per 600.000 satoshi, le 2 parti hanno successo lungo percorsi che prima non erano possibili.

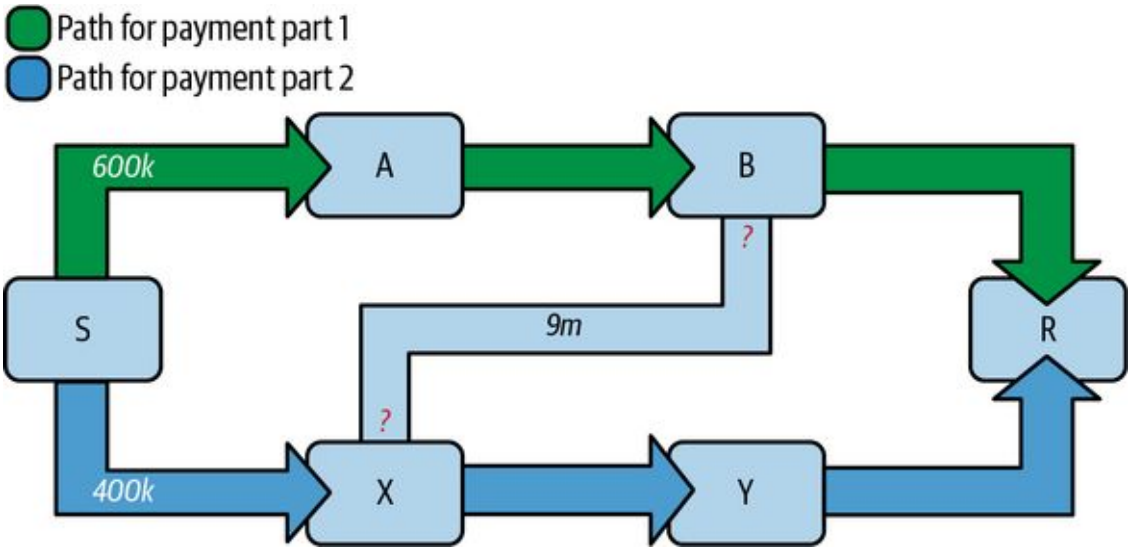

Figura 12-9. Invio di due parti di un pagamento multipart

Prove ed Errori su Più "Round"

I pagamenti in più parti portano a un ciclo di tentativi ed errori in qualche modo modificato per la consegna dei pagamenti. Poiché stiamo tentando più percorsi in ogni tentativo, abbiamo quattro possibili risultati:

- Tutte le parti hanno successo, il pagamento è andato a buon fine
- Alcune parti riescono, altre falliscono con errori restituiti

- Tutte le parti falliscono con errori restituiti
- Alcune parti sono "bloccate", non vengono restituiti errori

Nel secondo caso, in cui alcune parti falliscono con errori restituiti e alcune parti riescono, possiamo *ripetere* il ciclo di tentativi ed errori, ma *solo per l'importo residuo*.

Supponiamo ad esempio che Selena avesse un grafo dei canali molto più ampio con centinaia di possibili percorsi per raggiungere Rashid. Il suo algoritmo di ricerca del percorso potrebbe trovare una suddivisione ottimale dei pagamenti composta da 26 parti di dimensioni diverse. Dopo aver tentato di inviare tutte le 26 parti nel primo round, 3 di quelle parti falliscono con errori.

Se quelle 3 parti consistessero, per esempio, in 155.000 satoshi, allora Selena ricomincerebbe lo sforzo di ricerca del percorso solo per 155.000 satoshi. Il round successivo potrebbe trovare percorsi completamente diversi (ottimizzati per l'importo residuo di 155.000) e suddividere l'importo di 155.000 in frazioni completamente diverse!

SUGGERIMENTO	Anche se sembra che 26 parti divise siano molte, i test su Lightning Network hanno fornito con successo un pagamento di 0.3679 BTC suddividendolo in 345 parti.

Inoltre, il nodo di Selena aggiornerebbe il grafo dei canali utilizzando le informazioni raccolte dai successi e dagli errori del primo round per trovare i percorsi e le divisioni ottimali per il secondo round.

Fingiamo che il nodo di Selena calcoli che il modo migliore per inviare il residuo di 155k è dividendolo in 6 parti da 80k, 42k, 15k, 11k, 6.5k e 500 satoshi. Nel round successivo, Selena ottiene un solo errore, indicando che la parte di 11k satoshi è fallita. Ancora una volta, Selena aggiorna il grafo dei canali in base alle informazioni raccolte ed esegue di nuovo la ricerca per inviare i restanti 11k. Questa volta ci riesce con 2 parti rispettivamente di 6k e 5k satoshi.

Questo esempio multiround di invio di un pagamento tramite MPP è mostrato nella Figura 12-10.

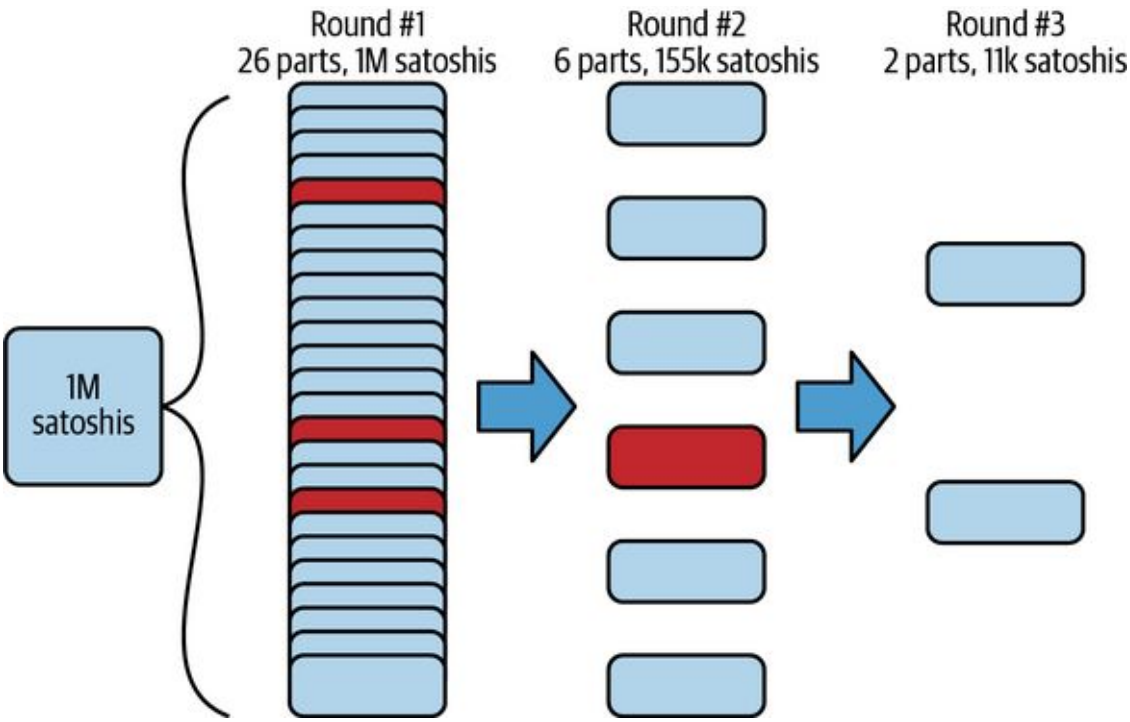

Figura 12-10. Invio di un pagamento in più round con MPP

Alla fine, il nodo di Selena ha utilizzato 3 round di ricerca del percorso per inviare 1 milione di satoshi in 30 parti.

Conclusione

In questo capitolo abbiamo esaminato il pathfinding e la consegna dei pagamenti. Abbiamo visto come utilizzare il grafo dei canali per trovare i percorsi da un mittente a un destinatario. Abbiamo anche visto come il mittente tenterà di consegnare i pagamenti su un percorso candidato e ripetere in un ciclo di tentativi ed errori.

Abbiamo anche esaminato l'incertezza della liquidità del canale (dal punto di vista del mittente) e le implicazioni che ha sul pathfinding. Abbiamo visto come possiamo quantificare l'incertezza e utilizzare la teoria della probabilità per trarre delle conclusioni utili. Abbiamo anche visto come ridurre l'incertezza imparando sia dai pagamenti riusciti che da quelli falliti.

Infine, abbiamo visto come la funzione di pagamento multipart ci consente di suddividere i pagamenti in parti, aumentando la probabilità di successo anche per pagamenti più grandi.

Capitolo 13. Protocollo Wire: Framing ed Estensibilità

In questo capitolo, ci addentreremo nel protocollo wire di LN e tratteremo tutte le varie leve di estensibilità che sono state integrate nel protocollo. Entro la fine di questo capitolo, un lettore ambizioso dovrebbe essere in grado di scrivere il proprio parser di protocollo wire per Lightning Network. Oltre a essere in grado di scrivere un parser di protocollo wire personalizzato, un lettore di questo capitolo acquisirà una profonda comprensione dei vari meccanismi di aggiornamento che sono stati incorporati nel protocollo.

Livello di Messaggistica nella Suite dei Protocolli Lightning

Il livello di messaggistica, descritto in dettaglio in questo capitolo, consiste in "Framing and message format", codifica "Type-Lenght-Value" e "Feature Bits". Questi componenti sono evidenziati da uno schema nella suite di protocolli, mostrata in Figura 13-1.

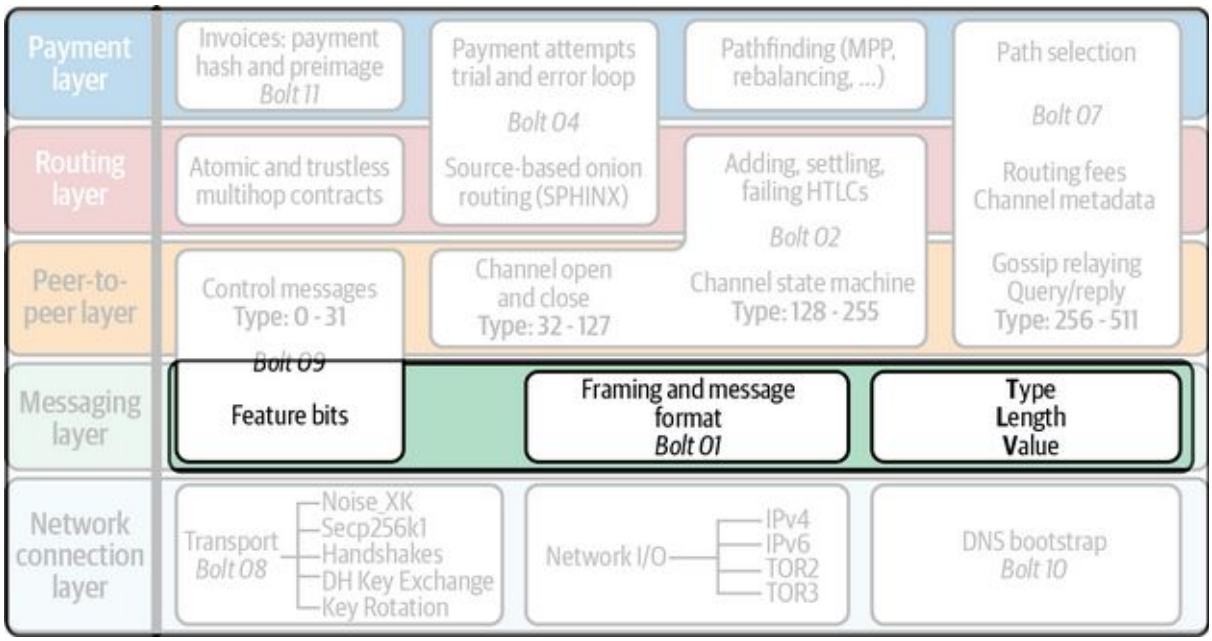

Figura 13-1. Livello di messaggistica nella suite di protocolli Lightning

Wire Framing

Iniziamo descrivendo la struttura di alto livello del *wire framing* all'interno del protocollo. Il termine wire frame (letteralmente "fil di ferro") e/o protocollo wire, nelle reti di computer, si riferisce a un modo per ottenere dati da un punto all'altro. Un protocollo wire è necessario se più di un'applicazione deve interagire. Quando diciamo framing, intendiamo il modo in cui i byte sono impacchettati sul filo per *codificare* un particolare messaggio di protocollo. Senza la conoscenza del sistema di framing utilizzato nel protocollo, una stringa di byte sul filo assomiglierebbe a una serie di byte casuali perché non è stata imposta alcuna struttura. Applicando un framing adeguato per decodificare questi byte sul filo, saremo in grado di estrarre la struttura e infine analizzare questa struttura in messaggi di protocollo all'interno del nostro linguaggio di livello superiore.

È importante notare che Lightning Network è un protocollo crittografato *end-to-end* e il wire framing è esso stesso incapsulato all'interno di un livello di trasporto dei messaggi crittografato. Come vedremo nel Capitolo 14, Lightning Network utilizza una variante personalizzata del protocollo Noise per gestire la crittografia durante il trasporto. All'interno di questo capitolo, ogni volta che forniamo un esempio di wireframing, assumiamo che il livello di crittografia sia già stato rimosso (durante la decodifica) o che non abbiamo ancora crittografato l'insieme di byte prima di inviarli sul filo (codifica) .

Wire Framing di Alto Livello

Detto questo, siamo pronti a descrivere lo schema di alto livello utilizzato per codificare i messaggi in rete:

- I messaggi in transito iniziano con un campo a *2 byte*, seguito da un payload del messaggio.
- Il payload del messaggio stesso può avere una dimensione massima di 65 KB.
- Tutti i numeri interi sono codificati in big-endian (ordine di rete).
- Tutti i byte che seguono un messaggio definito possono essere ignorati.

Poiché il protocollo si basa su un livello di crittografia del protocollo di trasporto *incapsulante*, non è necessaria una lunghezza esplicita per ogni tipo di messaggio. Ciò è dovuto al fatto che la crittografia di trasporto funziona a livello di *messaggio*, quindi quando

siamo pronti a decodificare il messaggio successivo, conosciamo già il numero totale di byte del messaggio stesso. L'utilizzo di 2 byte per il tipo di messaggio (codificato in big-endian) significa che il protocollo può avere fino a 2^{16}–1 o 65.535 messaggi distinti. Poiché sappiamo che tutti i messaggi devono essere inferiori a 65 KB, questo semplifica la nostra analisi in quanto possiamo utilizzare un buffer di dimensioni fisse e mantenere forti limiti sulla quantità totale di memoria richiesta per analizzare un messaggio in arrivo.

L'elenco puntato finale consente un certo grado di *retrocompatibilità* perché i nuovi nodi sono in grado di fornire informazioni nei messaggi di rete che i nodi più vecchi (che potrebbero non capirli) possono tranquillamente ignorare. Vedremo che tale caratteristica, combinata con un formato di estensibilità dei messaggi via filo molto flessibile, consente al protocollo di raggiungere anche la compatibilità con le versioni successive.

Type Encoding

Con questo background ad alto livello, iniziamo ora dal livello più primitivo: l'analisi dei tipi primitivi. Oltre a codificare numeri interi, il protocollo Lightning consente anche la codifica di una vasta gamma di tipi, tra cui byte slice di lunghezza variabile, chiavi pubbliche a curva ellittica, indirizzi Bitcoin e firme. Quando descriveremo la *struttura* dei messaggi via filo più avanti in questo capitolo, ci riferiremo al tipo di alto livello (il tipo astratto) piuttosto che alla rappresentazione di livello inferiore di detto tipo. In questa sezione, eliminiamo questo livello di astrazione per garantire che il nostro futuro wire parser sia in grado di codificare/decodificare correttamente qualsiasi tipo di livello superiore.

In Tabella 13-1, mappiamo il nome di un dato tipo di messaggio alla routine di alto livello utilizzata per codificarne/decodificarne il tipo.

Tabella 13-1. Tipi di messaggi di alto livello

Tipo di Alto Livello	Framing	Commento
`node_alias`	Una porzione di byte a lunghezza fissa di 32 byte	Durante la decodifica, rifiuta se i contenuti non sono una stringa UTF-8 valida
`channel_id`	Un byte slice a lunghezza fissa di	Dato un outpoint, è possibile

Tipo di Alto Livello	Framing	Commento
	32 byte che esegue il mapping di un outpoint a un valore di 32 byte	convertirlo in un `channel_id` prendendo il TxID dell'outpoint e facendo XOR con l'indice (interpretato come i 2 byte inferiori)
`short_chan_id`	Un numero intero senza segno a 64 bit (`uint64`)	Composto da altezza del blocco (24 bit), indice di transazione (24 bit) e indice di output (16 bit) racchiusi in 8 byte
`milli_satoshi`	Un numero intero senza segno a 64 bit (`uint64`)	Rappresenta il millesimo di un satoshi
`satoshi`	Un numero intero senza segno a 64 bit (`uint64`)	L'unità di base di bitcoin
`pubkey`	Una chiave pubblica secp256k1 codificata in formato *compresso*, che occupa 33 byte	Occupa una lunghezza fissa di 33 byte sul filo
`sig`	Una firma ECDSA della curva ellittica secp256k1	Codificata come una porzione di byte fissa da 64 byte, impacchettata come `R || S`
`uint8`	Un numero intero a 8 bit	
`uint16`	Un numero intero a 16 bit	
`uint64`	Un numero intero a 64 bit	
`[]byte`	Una porzione di byte di lunghezza variabile	Preceduto da un numero intero a 16 bit che denota la lunghezza dei byte
`color_rgb`	Codifica colore RGB	Codificato come una serie di numeri

Tipo di Alto Livello	Framing	Commento
		interi a 8 bit
`net_addr`	La codifica di un indirizzo di rete	Codificato con un prefisso di 1 byte che denota il tipo di indirizzo, seguito dal corpo dell'indirizzo

Nella sezione successiva, descriviamo la struttura di ogni messaggio wire, incluso il tipo di prefisso del messaggio assieme al contenuto del corpo del messaggio.

Estensioni Messaggio Type-Lenght-Value

In precedenza in questo capitolo abbiamo accennato al fatto che i messaggi possono avere dimensioni fino a 65 KB e se durante l'analisi di un messaggio rimangono byte extra, allora quei byte devono essere ignorati. A prima vista, questo requisito può sembrare alquanto arbitrario; tuttavia, questo requisito consente l'evoluzione de-sincronizzata disaccoppiata del protocollo Lightning stesso. Ne discuteremo maggiormente verso la fine del capitolo. Per ora, rivolgiamo la nostra attenzione esattamente a cosa possono essere usati quei "byte extra" alla fine di un messaggio.

Il Formato dei Messaggi dei Buffer del Protocollo

Il formato di serializzazione dei messaggi Protocol Buffers (Protobuf) è iniziato come formato interno utilizzato da Google ed è sbocciato in uno dei formati di serializzazione dei messaggi più popolari utilizzati dagli sviluppatori a livello globale. Il formato Protobuf descrive come un messaggio (di solito una sorta di struttura dati correlata ad un'API) viene codificato sul filo e decodificato all'altra estremità. Esistono diversi "compilatori Protobuf" in dozzine di linguaggi che fungono da ponte che consente a qualsiasi linguaggio di codificare un Protobuf, che sarà in grado di essere decodificato mediante una decodifica conforme in un altro linguaggio. Tale compatibilità della struttura dei dati tra linguaggi diverse consente un'enorme innovazione

perché è possibile trasmettere strutture e persino strutture di dati digitate oltre i confini del linguaggio e dell'astrazione.

I Protobuf sono noti anche per la loro flessibilità rispetto al modo in cui gestiscono i cambiamenti nella struttura dei messaggi sottostanti. Finché viene rispettato lo schema di numerazione dei campi, è possibile che una scrittura più recente di Protobuf includa informazioni all'interno che potrebbero essere sconosciute a qualsiasi lettore più vecchio. Quando il vecchio lettore incontra il nuovo formato serializzato, se ci sono tipi/campi che non comprende, semplicemente li *ignora*. Ciò consente ai client vecchi e nuovi di coesistere perché tutti i client possono analizzare una parte del formato del messaggio più recente.

Compatibilità Futura e Retrocompatibilità

I Protobuf sono estremamente popolari tra gli sviluppatori perché hanno il supporto integrato per la compatibilità sia in avanti che all'indietro. La maggior parte degli sviluppatori ha probabilmente familiarità con il concetto di retrocompatibilità. In termini semplici, qualsiasi modifica al formato di un messaggio o all'API deve essere eseguita in modo da non interrompere il supporto per i client meno recenti. Nei nostri precedenti esempi di estensibilità di Protobuf, la compatibilità con le versioni precedenti si ottiene assicurando che le nuove aggiunte al formato Protobuf non rompano le parti note dei lettori precedenti. La compatibilità con le versioni successive, d'altra parte, è altrettanto importante per gli aggiornamenti de-sincronizzati; tuttavia, è meno comunemente nota. Affinché una modifica sia compatibile con le versioni successive, i client devono semplicemente ignorare qualsiasi informazione che non comprendono. Si può dire che il meccanismo di soft fork dell'aggiornamento del sistema di consenso Bitcoin sia compatibile sia con le versioni successive che con le versioni precedenti: tutti i client che non si aggiornano possono ancora utilizzare Bitcoin e se incontrano transazioni che non comprendono, semplicemente le ignorano poiché i loro fondi non utilizzano queste nuove funzionalità.

Formato Type-Lenght-Value

Per essere in grado di aggiornare i messaggi in un modo compatibile sia con le versioni future che con le versioni precedenti, oltre ai bit delle funzionalità (cui parleremo più avanti), Lightning Network utilizza un formato di serializzazione dei messaggi personalizzato

chiamato semplicemente Type-Length-Value o TLV. Il formato è stato ispirato dal formato Protobuf ampiamente utilizzato e prende in prestito molti concetti semplificando notevolmente l'implementazione e il software che interagisce con l'analisi dei messaggi. Un lettore curioso potrebbe chiedere: "perché non usare solo Protobuf?" In risposta, gli sviluppatori di Lightning risponderebbero che siamo in grado di avere il meglio dell'estensibilità di Protobuf pur avendo il vantaggio di un'implementazione più piccola e quindi di un attacco più piccolo. A partire dalla versione 3.15.6, il compilatore Protobuf pesa oltre 656.671 righe di codice. In confronto, l'implementazione di LND del formato del messaggio TLV pesa solo 2300 righe di codice (inclusi i test).

Siamo ora pronti per descrivere il formato TLV in dettaglio. Si dice che un'estensione di messaggio TLV sia un flusso di singoli record TLV. Un singolo record TLV ha tre componenti: il tipo di record, la lunghezza del record e infine il valore opaco del record:

`type`

Un numero intero che rappresenta il nome del record da codificare

`lenght`

La lunghezza del record

`value`

Il valore opaco del record

Sia il `type` che la `lenght` sono codificati utilizzando un numero intero di dimensione variabile ispirato al numero intero di dimensione variabile (varint) utilizzato nel protocollo P2P di Bitcoin, chiamato in breve `BigSize`.

Codifica BigSize Integer

Nella sua forma più completa, un numero intero `BigSize` può rappresentare un valore fino a 64 bit. Contrariamente al formato varint di Bitcoin, il formato `BigSize` codifica invece i numeri interi utilizzando un ordinamento di byte big-endian.

La variante `BigSize` ha due componenti: il discriminante e il corpo. Nel contesto dell'intero `BigSize`, il discriminante comunica al decodificatore la dimensione dell'intero di dimensione variabile che segue. Ricorda che l'unicità degli interi di dimensioni variabili è che consentono a un parser di utilizzare meno byte per codificare interi più piccoli rispetto a quelli più grandi, risparmiando spazio. La codifica di un numero intero `BigSize` segue una delle quattro seguenti opzioni:

1. Se il valore è minore di `0xfd` (`253`): allora il discriminante non è effettivamente utilizzato e la codifica è semplicemente il numero intero stesso. Questo ci consente di codificare numeri interi molto piccoli senza costi aggiuntivi.
2. Se il valore è minore o uguale a `0xffff` (`65535`): il discriminante è codificato come `0xfd`, che indica che il valore che segue è maggiore di `0xfff`, ma minore di `0xffff`. Il numero viene quindi codificato come numero intero a 16 bit. Includendo il discriminante, possiamo codificare un valore maggiore di 253, ma minore di 65.535 utilizzando 3 byte.
3. Se il valore è inferiore a `0xffffffff` (`4294967295`): il discriminante è codificato come `0xfe`. Il corpo è codificato utilizzando un numero intero a 32 bit, incluso il discriminante, e possiamo codificare un valore inferiore a `4.294.967.295` utilizzando 5 byte.
4. Altrimenti, codifichiamo semplicemente il valore come numero intero a 64 bit a dimensione intera.

Vincoli di Codifica TLV

Nel contesto di un messaggio TLV, si dice che i tipi di record inferiori a 2^{16} siano *riservati* per un uso futuro. I tipi oltre questo intervallo devono essere utilizzati per le estensioni dei messaggi "personalizzate" utilizzate dai protocolli delle applicazioni di livello superiore.

Il `value` di un record dipende dal `type`. In altre parole, può assumere qualsiasi forma perché i parser tenteranno di interpretarlo a seconda del contesto del tipo stesso.

Codifica Canonica TLV

Un problema con il formato Protobuf è che le codifiche dello stesso messaggio possono

generare un set di byte completamente diverso se codificate da due diverse versioni del compilatore. Tali istanze di una codifica non canonica non sono accettabili nel contesto di Lightning, poiché molti messaggi contengono una firma del digest del messaggio. Se è possibile codificare un messaggio in due modi diversi, allora sarebbe possibile interrompere inavvertitamente l'autenticazione di una firma ricodificando un messaggio utilizzando un insieme leggermente diverso di byte sul filo.

Per garantire che tutti i messaggi codificati siano canonici, durante la codifica vengono definiti i seguenti vincoli:

- Tutti i record all'interno di un flusso TLV devono essere codificati in ordine di tipo strettamente crescente.
- Tutti i record devono codificare come minimo i campi di `type` e `lenght`. In altre parole, deve essere utilizzata sempre la rappresentazione `BigSize` più piccola per un numero intero.
- Ogni `type` può apparire solo una volta all'interno di un determinato flusso TLV.

Oltre a questi vincoli di codifica, viene definita anche una serie di requisiti di interpretazione di livello superiore basati sull'*arietà* di un dato numero intero di `type`. Approfondiremo ulteriormente questi dettagli verso la fine del capitolo, dopo aver descritto come il protocollo Lightning viene aggiornato in pratica e in teoria.

Bit di Funzionalità ed Estensibilità del Protocollo

Poiché Lightning Network è un sistema decentralizzato, nessuna singola entità può imporre una modifica del protocollo a tutti gli utenti del sistema. Questa caratteristica si vede anche in altre reti decentralizzate come Bitcoin. Tuttavia, a differenza di Bitcoin, *non è necessario* un consenso per modificare un sottoinsieme di LN. Lightning è in grado di evolversi a piacimento senza un forte requisito di coordinamento perché, a differenza di Bitcoin, non è richiesto un consenso globale su LN. A causa di ciò e di altri meccanismi di aggiornamento incorporati in LN, solo i partecipanti che desiderano utilizzare queste nuove funzionalità di LN devono eseguire l'aggiornamento e per essere in grado di interagire tra loro.

In questa sezione, esploreremo i vari modi in cui gli sviluppatori e gli utenti sono in grado di progettare e implementare nuove funzionalità in Lightning Network. I creatori dell'originale LN sapevano che c'erano molte possibili direzioni future per la rete e il protocollo

sottostante. Di conseguenza, si sono assicurati di implementare diversi meccanismi di estensibilità all'interno del sistema, che possono essere utilizzati per aggiornarlo parzialmente o completamente in modo disaccoppiato, de-sincronizzato e decentralizzato.

Bit di Funzionalità come Meccanismo di Rilevabilità dell'Aggiornamento

Un lettore astuto potrebbe aver notato le varie posizioni in cui i bit delle funzionalità sono inclusi nel protocollo Lightning. Un *feature bit* è un campo di bit che può essere utilizzato per pubblicizzare la comprensione o l'adesione a un possibile aggiornamento del protocollo di rete. I bit di funzionalità sono comunemente assegnati in coppia, il che significa che ogni potenziale nuova funzionalità/aggiornamento definisce sempre due bit all'interno del campo di bit. Un bit segnala che la funzione annunciata è *facoltativa*, il che significa che il nodo conosce la funzione e può utilizzarla, ma non la considera necessaria per il normale funzionamento. L'altro bit segnala che la funzione è invece *richiesta*, il che significa che il nodo non continuerà a funzionare se un potenziale peer non comprende quella funzione.

Usando questi due bit (facoltativi e richiesti), possiamo costruire una semplice matrice di compatibilità che i nodi/utenti possono consultare per determinare se un peer è compatibile con una caratteristica desiderata, come mostrato in Tabella 13-2.

Tabella 13-2. Matrice di compatibilità dei bit di funzionalità

Tipo di bit	Remoto facoltativo	Remoto richiesto	Remoto sconosciuto
Locale facoltativo	✔	✔	✔
Locale richiesto	✔	✔	✘
Locale sconosciuto	✔	✘	✘

Da questa matrice di compatibilità semplificata, possiamo vedere che fintanto che l'altra parte conosce il nostro bit di funzionalità, allora possiamo interagire con loro utilizzando il protocollo. Se la parte non sa nemmeno a quale parte ci riferiamo *e* richiedono la funzione, allora siamo incompatibili con loro. All'interno della rete, le funzionalità opzionali vengono segnalate utilizzando un *numero di bit dispari*, mentre le funzionalità richieste vengono

segnalate utilizzando un *numero di bit pari*. Ad esempio, se un peer segnala di essere a conoscenza di una funzionalità che utilizza il bit `15`, allora sappiamo che si tratta di una funzionalità opzionale e possiamo interagire con loro o rispondere ai loro messaggi anche se non conosciamo la funzionalità . Se invece hanno segnalato la funzione utilizzando il bit `16`, allora sappiamo che questa è una funzione richiesta e non possiamo interagire con loro a meno che anche il nostro nodo non comprenda quella funzione.

Gli sviluppatori di Lightning hanno escogitato una frase facile da ricordare che codifica questa matrice: "it's OK to be odd" (che, tradotto in italiano significa "va bene essere strani", frase che però è anche un gioco di parole in quanto "odd" significa non solo "strano" ma anche "dispari"). Questa semplice regola consente un ricco insieme di interazioni all'interno del protocollo, poiché una semplice operazione tra due vettori di bit di funzionalità consente ai peer di constatare se determinate interazioni sono compatibili tra loro o meno. In altre parole, i bit di funzionalità vengono utilizzati come meccanismo di rilevabilità dell'aggiornamento: consentono facilmente ai peer di capire se sono compatibili o meno in base ai concetti di feature bit opzionali, richiesti e sconosciuti.

I bit di funzionalità si trovano nei messaggi `node_announcement`, `channel_announcement` e `init` all'interno del protocollo. Di conseguenza, questi tre messaggi possono essere utilizzati per segnalare la conoscenza e/o la comprensione degli aggiornamenti del protocollo all'interno della rete. I bit di funzionalità trovati nel messaggio `node_announcement` possono consentire a un peer di determinare se le loro *connessioni* sono compatibili o meno. I bit di funzionalità all'interno dei messaggi `channel_announcement` consentono a un peer di determinare se un determinato tipo di pagamento o HTLC può transitare attraverso un determinato peer o meno. I bit di funzionalità all'interno del messaggio `init` consentono ai peer di capire se possono mantenere una connessione e anche quali funzionalità vengono negoziate per la durata di una data connessione.

TLV per Compatibilità Futura e Retrocompatibilità

Come abbiamo appreso in precedenza, i record TLV possono essere utilizzati per estendere i messaggi in modo compatibile con le versioni precedenti e successive. Nel corso del tempo, questi record sono stati utilizzati per estendere i messaggi esistenti senza violare il protocollo utilizzando l'area "non definita" all'interno di un messaggio oltre l'insieme di byte conosciuti.

Ad esempio, il protocollo Lightning originale non prevedeva il concetto di "quantità massima

di HTLC" che potesse attraversare un canale come dettato da una politica di instradamento. Successivamente, il campo `max_htlc` è stato aggiunto al messaggio `channel_update` per introdurre gradualmente questo concetto nel tempo. I peer che ricevono un `channel_update` che imposta tale campo ma non sanno che l'aggiornamento è esistito non sono interessati dalla modifica, ma i loro HTLC vengono rifiutati se superano il limite. I peer più recenti sono invece in grado di analizzare, verificare e utilizzare il nuovo campo.

Coloro che hanno familiarità con il concetto di soft fork in Bitcoin potrebbero ora vedere alcune somiglianze tra i due meccanismi. A differenza dei soft fork a livello di consenso Bitcoin, gli aggiornamenti al Lightning Network non richiedono un consenso schiacciante per essere adottati. Invece, almeno due peer all'interno della rete devono comprendere un nuovo aggiornamento per iniziare a usarlo. Comunemente questi due peer possono essere il destinatario e il mittente di un pagamento o possono essere i partner di canale di un nuovo canale di pagamento.

Una Tassonomia dei Meccanismi di Aggiornamento

Piuttosto che un unico meccanismo di aggiornamento ampiamente utilizzato all'interno della rete (come i soft fork per Bitcoin), esistono diversi possibili meccanismi di aggiornamento all'interno di Lightning Network. In questa sezione, enumeriamo questi meccanismi di aggiornamento e forniamo un esempio reale del loro utilizzo in passato.

Aggiornamenti della rete interni

Iniziamo con il tipo di aggiornamento che richiede la maggior coordinazione a livello di protocollo: gli aggiornamenti di rete interni. Un aggiornamento di rete interno è caratterizzato dal fatto che richiede che *ogni singolo nodo* all'interno di un percorso di pagamento potenziale comprenda la nuova funzionalità. Tale aggiornamento è simile a qualsiasi aggiornamento all'interno di Internet che richiede aggiornamenti a livello hardware all'interno della parte core-relay dell'aggiornamento. Nel contesto di Lightning Network, tuttavia, abbiamo a che fare con software puro, quindi tali aggiornamenti sono più facili da implementare, ma richiedono ancora molto più coordinamento rispetto a qualsiasi altro meccanismo di aggiornamento nella rete.

Un esempio di tale aggiornamento all'interno della rete è stata l'introduzione di una codifica TLV per le informazioni di instradamento codificate all'interno dei pacchetti onion. Il formato

precedente utilizzava un formato di messaggio a lunghezza fissa codificato per comunicare informazioni come l'hop successivo. Poiché questo formato era corretto, significava che non erano possibili nuovi aggiornamenti a livello di protocollo. Il passaggio al formato TLV più flessibile significava che dopo questo aggiornamento, qualsiasi tipo di funzionalità che modificava il tipo di informazioni comunicate a ogni hop poteva essere implementata a piacere.

Vale la pena ricordare che l'aggiornamento del TLV onion era una sorta di aggiornamento "soft" della rete interna, in quanto se un pagamento non utilizzava alcuna nuova funzionalità oltre a quella nuova codifica delle informazioni di instradamento, allora un pagamento poteva essere trasmesso utilizzando un insieme misto di nodi .

Aggiornamenti end-to-end

Per contrastare l'aggiornamento della rete interno, in questa sezione viene descritto l'aggiornamento della rete *end-to-end*. Questo meccanismo di aggiornamento differisce dall'aggiornamento della rete interno in quanto richiede solo l'aggiornamento degli "end" del pagamento: il mittente e il destinatario.

Questo tipo di aggiornamento consente un'ampia gamma di innovazione illimitata all'interno della rete. A causa della natura crittografata tramite onion dei pagamenti all'interno della rete, coloro che inoltrano HTLC all'interno della rete potrebbero non sapere nemmeno che vengono utilizzate nuove funzionalità.

Un esempio di aggiornamento end-to-end all'interno della rete è stato il lancio dei pagamenti in più parti (MPP). MPP è una funzionalità a livello di protocollo che consente di suddividere un singolo pagamento in più parti o percorsi, che va successivamente analizzato e gestito dal destinatario di un pagamento. L'implementazione di MPP è stata abbinata a un nuovo bit di funzionalità a livello di `node_announcement` che indica che il destinatario sa come gestire i pagamenti parziali. Supponendo che mittente e destinatario si conoscano l'un l'altro (possibilmente tramite una fattura BOLT #11), sono in grado di utilizzare la nuova funzionalità senza ulteriori negoziazioni.

Un altro esempio di aggiornamento end-to-end implementati all'interno della rete sono i vari tipi di pagamenti *spontanei*. Un tipo di pagamento spontaneo chiamato *keysend* funziona semplicemente inserendo la preimmagine di un pagamento all'interno della cipolla crittografata. Al ricevimento, il destinatario decrittografa l'immagine preliminare, quindi la

utilizza per saldare il pagamento. Poiché l'intero pacchetto è crittografato end-to-end, tale tipologia di pagamento è sicura, poiché nessuno dei nodi intermedi è in grado di sbucciare completamente la cipolla per scoprire la preimage del pagamento.

Aggiornamenti a Livello di Costruzione del Canale

L'ultima ampia categoria di aggiornamenti è quella che avviene a livello di costruzione del canale, ma che non modifica la struttura dell'HTLC ampiamente utilizzato all'interno della rete. Quando diciamo costruzione del canale, intendiamo come il canale viene finanziato o creato. Ad esempio, il tipo di canale eltoo può essere implementato all'interno della rete utilizzando un nuovo bit di funzionalità a livello di `node_announcement` e un bit di funzionalità a livello di `channel_announcement`. Solo i due pari ai lati dei canali devono comprendere e pubblicizzare queste nuove funzionalità. Questa coppia di canali può quindi essere utilizzata per inoltrare qualsiasi tipo di pagamento ammesso che il canale lo supporti.

Un altro è il formato del canale di *output di ancoraggio* che consente di aumentare la commissione di impegno tramite il meccanismo di gestione delle commissioni Child-Pays-For-Parent (CPFP) di Bitcoin.

Conclusione

Il protocollo wire di Lightning è incredibilmente flessibile e consente una rapida innovazione e interoperabilità senza uno stretto consenso. È uno dei motivi per cui Lightning Network sta vivendo uno sviluppo molto più rapido ed è attraente per molti sviluppatori, che altrimenti potrebbero trovare lo stile di sviluppo di Bitcoin troppo conservatore e lento.

Capitolo 14. Trasporto di Messaggi Crittografati

In questo capitolo esamineremo il *trasporto di messaggi crittografati* di Lightning Network, a volte indicato come *Protocollo Brontide*, che consente ai peer di stabilire comunicazioni crittografate end-to-end, autenticazione e controllo dell'integrità.

NOTA	Parte di questo capitolo include alcuni dettagli altamente tecnici sul protocollo e sugli algoritmi di crittografia utilizzati nel trasporto crittografato di Lightning Network. Puoi decidere di saltare quella sezione se non sei interessato a questi dettagli.

Trasporto Crittografato nella Suite di Protocolli Lightning

Il componente di trasporto di Lightning Network e i suoi numerosi componenti sono mostrati nella parte più a sinistra del livello di connessione di rete nella Figura 14-1.

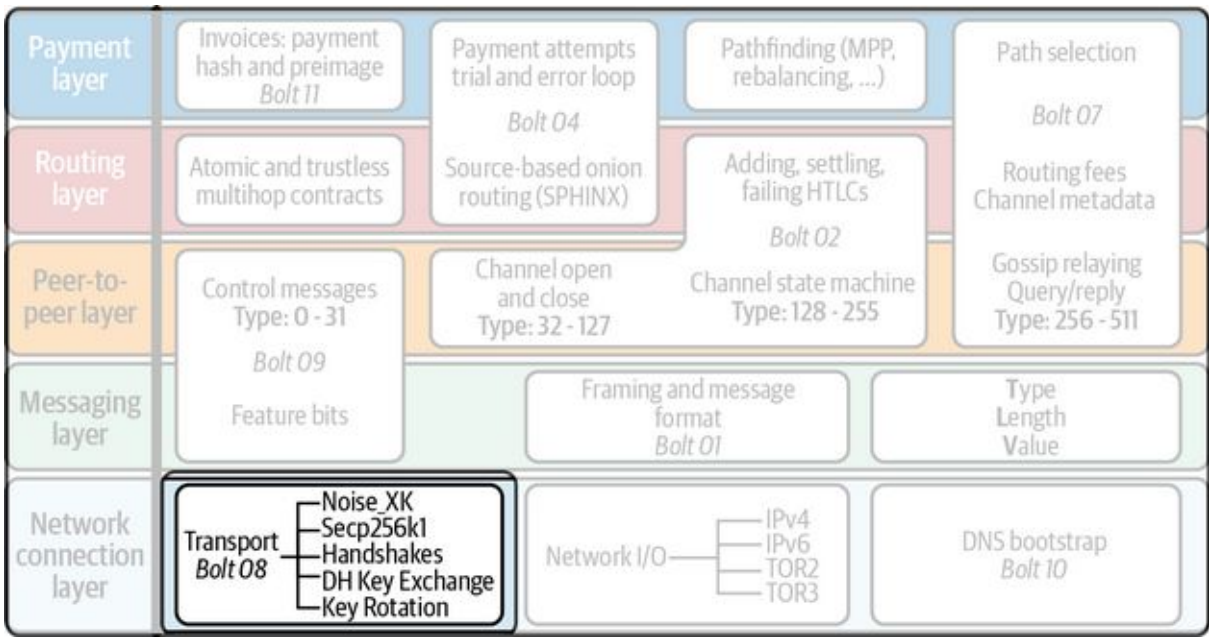

Figura 14-1. Trasporto di messaggi crittografati nella suite di protocolli Lightning

introduzione

A differenza della rete P2P di Bitcoin, ogni nodo della rete Lightning è identificato da una chiave pubblica univoca che funge da identità. Per impostazione predefinita, questa chiave pubblica viene utilizzata per crittografare end-to-end *tutte* le comunicazioni all'interno della rete. La crittografia per impostazione predefinita al livello più basso del protocollo garantisce che tutti i messaggi siano autenticati, immuni da attacchi man-in-the-middle (MITM) e snooping da parte di terze parti e garantisce la privacy a livello di trasporto fondamentale. In questo capitolo, impareremo in dettaglio il protocollo di crittografia utilizzato da Lightning Network. Al termine di questo capitolo, avrai familiarità con lo stato dell'arte dei protocolli di messaggistica crittografati, nonché con le varie proprietà che tale protocollo fornisce alla rete. Vale la pena ricordare che il nucleo del trasporto di messaggi crittografati è *indipendente* dal suo utilizzo nel contesto di Lightning Network. Di conseguenza, il trasporto di messaggi crittografati personalizzato utilizzato da Lightning può essere inserito in qualsiasi contesto che richieda la comunicazione crittografata tra due parti.

Il Grafo Dei Canali Come Infrastruttura Decentralizzata a Chiave Pubblica

Come abbiamo appreso nel Capitolo 8, ogni nodo ha un'identità a lungo termine che viene utilizzata come identificatore di un vertice durante la ricerca del percorso e utilizzata anche nelle operazioni crittografiche asimmetriche relative alla creazione di pacchetti onion routing cifrati. Questa chiave pubblica, che funge da identità a lungo termine di un nodo, è inclusa nella risposta di bootstrap DNS, nonché incorporata nel grafo dei canali. Di conseguenza, prima che un nodo tenti di connettersi a un altro nodo sulla rete P2P, conosce già la chiave pubblica del nodo a cui desidera connettersi.

Inoltre, se il nodo a cui ci si connette ha già una serie di canali pubblici all'interno del grafo, allora il nodo in connessione è in grado di verificare ulteriormente l'identità dello stesso. Poiché l'intero grafo dei canali è completamente autenticato, è possibile visualizzarlo come una sorta di infrastruttura a chiave pubblica (PKI) decentralizzata: per registrare una chiave, è necessario aprire un canale pubblico nella blockchain di Bitcoin e una volta che un nodo non ha più alcun canale pubblico, viene effettivamente rimosso dalla PKI.

Poiché Lightning è una rete decentralizzata, è imperativo che a nessuna parte centrale sia

designato il potere di fornire un'identità a chiave pubblica all'interno della rete. Al posto di un partito centrale, Lightning Network utilizza la blockchain Bitcoin come meccanismo di mitigazione Sybil perché ottenere un'identità sulla rete ha un costo tangibile: la commissione necessaria per creare un canale nella blockchain, così come il costo opportunità del capitale assegnato ai canali. L'attacco di Sybil è un attacco informatico dove i sistemi di reputazione sono sovvertiti falsificando le identità di una persona in una rete p2p. Nel processo di implementazione di una PKI specifica del dominio, LN è in grado di semplificare notevolmente il proprio protocollo di trasporto crittografato in quanto non ha bisogno di affrontare tutte le complessità che derivano da TLS, il protocollo Transport Layer Security.

Perché Non TLS?

I lettori che hanno familiarità con il sistema TLS potrebbero chiedersi a questo punto: perché non è stato utilizzato TLS nonostante gli svantaggi del sistema PKI esistente? È infatti un dato di fatto che i "certificati autofirmati" possono essere utilizzati per eludere efficacemente il sistema PKI globale esistente semplicemente affermando l'identità di una data chiave pubblica tra un insieme di pari. Tuttavia, anche con il sistema PKI esistente, TLS presenta diversi inconvenienti che hanno spinto i creatori di Lightning Network a optare invece per un protocollo di crittografia personalizzato più compatto.

Per cominciare, TLS è un protocollo che esiste da diversi decenni e di conseguenza si è evoluto nel tempo man mano che sono stati compiuti nuovi progressi nel campo della crittografia dei trasporti. Tuttavia, questa evoluzione ha fatto sì che il protocollo aumentasse di dimensioni e complessità. Negli ultimi decenni, diverse vulnerabilità in TLS sono state scoperte e corrette, con ogni evoluzione che aumenta ulteriormente la complessità del protocollo. A causa dell'età del protocollo, esistono diverse versioni e iterazioni, il che significa che un client deve comprendere molte delle iterazioni precedenti del protocollo per comunicare con un'ampia porzione dell'Internet pubblico, aumentando ulteriormente la complessità dell'implementazione.

In passato, sono state scoperte diverse vulnerabilità di sicurezza della memoria in implementazioni ampiamente utilizzate di SSL/TLS. Il confezionamento di un tale protocollo all'interno di ogni nodo Lightning aumenterebbe la superficie di attacco dei nodi esposti alla rete pubblica peer-to-peer. Per aumentare la sicurezza della rete nel suo complesso e ridurre al minimo la superficie di attacco sfruttabile, i creatori di LN hanno invece optato per

l'adozione del Noise Protocol Framework. Il Noise all'interno del protocollo interiorizza molte delle lezioni sulla sicurezza e sulla privacy apprese nel tempo grazie al continuo controllo del protocollo TLS. In un certo senso, l'esistenza di Noise consente alla comunità di "ricominciare" efficacemente, con un protocollo più compatto e semplificato che conserva tutti i vantaggi aggiuntivi di TLS.

Il Noise Protocol Framework

Il Noise Protocol Framework è un protocollo di crittografia dei messaggi moderno, estensibile e flessibile progettato dai creatori del Signal Protocol. Il protocollo Signal è uno dei protocolli di crittografia dei messaggi più utilizzati al mondo. È utilizzato sia da Signal che da Whatsapp, che cumulativamente sono utilizzati da miliardi di persone in tutto il mondo. Il framework Noise è il risultato di decenni di evoluzione sia all'interno del mondo accademico che nell'industria dei protocolli di crittografia dei messaggi. Lightning utilizza il Noise Protocol Framework per implementare un protocollo di crittografia *orientato ai messaggi* utilizzato da tutti i nodi per comunicare tra loro.

Una sessione di comunicazione che utilizza Noise ha due fasi distinte: la fase di stretta di mano e la fase di messaggistica. Prima che due parti possano comunicare tra loro, devono prima arrivare a un segreto condiviso noto solo a loro che verrà utilizzato per crittografare e autenticare i messaggi reciprocamente inviati. Viene utilizzato un accordo chiave autenticato per arrivare a una chiave condivisa finale tra le due parti. Nel contesto del Protocollo Noise, questo accordo chiave autenticato è indicato come *handshake* (stretta di *mano*). Una volta che l'handshake è stato completato, entrambi i nodi possono ora scambiarsi messaggi crittografati. Ogni volta che i peer devono connettersi o riconnettersi tra loro, viene eseguita una nuova iterazione del protocollo di handshake, assicurando che venga raggiunta la segretezza futura (la perdita della chiave di una trascrizione precedente non compromette alcuna trascrizione futura).

Poiché il Protocollo Noise consente a un progettista di protocolli di scegliere tra diverse primitive crittografiche, come la crittografia simmetrica e la crittografia a chiave pubblica, è consuetudine che ogni versione del Protocollo Noise sia indicata con un nome univoco. Nello spirito di "Noise", ogni versione del protocollo seleziona un nome derivato da una sorta di "rumore". Nel contesto del Lightning Network, il tipo di Noise Protocol utilizzato è talvolta indicato come Brontide. Un *brontide* è un basso rumore fluttuante, simile a quello che si

sentirebbe durante un temporale quando si è molto lontani.

Il Trasporto Crittografato di Lightning in Dettaglio

In questa sezione analizzeremo il protocollo di trasporto crittografato di Lightning e approfondiremo i dettagli sugli algoritmi crittografici e del protocollo utilizzato per stabilire comunicazioni crittografate, autenticate e garantite dall'integrità tra peer. Sentiti libero di saltare questa sezione se non sei interessato a questo livello di dettaglio.

Noise_XK: Il Noise Handshake di Lightning Network

Il Noise Protocol è estremamente flessibile in quanto pubblicizza diverse handshake, ciascuna con diverse proprietà di sicurezza e privacy tra cui un potenziale implementatore del protocollo può scegliere. Un'analisi approfondita di ciascuna delle handshake e dei loro vari compromessi esula dallo scopo di questo capitolo. Detto ciò, LN utilizza uno specifico handshake denominato `Noise_XK`. La proprietà univoca fornita da questo handshake è l'*occultamento dell'identità*: affinché un nodo possa avviare una connessione con un altro nodo, deve prima conoscere la sua chiave pubblica. Ciò significa che la chiave pubblica del risponditore non viene effettivamente mai trasmessa durante il contesto dell'handshake. Invece, per autenticare il risponditore viene utilizzata una serie intelligente di controlli Elliptic Curve Diffie-Hellman (ECDH) e del codice di autenticazione del messaggio (MAC).

Handshake Notation e Flusso del Protocollo

Ogni handshake consiste in diversi passaggi. Ad ogni passaggio, del materiale possibilmente crittografato viene inviato alla parte opposta, viene eseguito uno o più ECDH, con il risultato che l'handshake viene "mescolato" in una *trascrizione* del protocollo. Questa trascrizione serve ad autenticare ogni fase del protocollo e aiuta a contrastare un certo tipo di attacchi man-in-the-middle. Al termine dell'handshake, vengono prodotte due chiavi, `ck` e `k`, utilizzate per crittografare i messaggi (`k`) e ruotare le chiavi (`ck`) per tutta la durata della sessione.

Nel contesto di una handshake, `s` è solitamente una chiave pubblica statica a lungo termine. Nel nostro caso, il sistema crittografico a chiave pubblica utilizzato è un sistema a curva ellittica, specificatamente `secp256k1`, che viene utilizzato anche in Bitcoin. Diverse chiavi

effimere vengono generate durante l'handshake. Usiamo e per riferirci a una nuova chiave effimera. Le operazioni ECDH tra due chiavi sono indicate come la concatenazione di due chiavi. Ad esempio, ee rappresenta un'operazione ECDH tra due chiavi temporanee.

Panoramica di Alto Livello

Usando tali presupposti, possiamo descrivere sinteticamente `Noise_XK` come segue:

```
Noise_XK(s, rs):
   <- rs
   ...
   -> e, e(rs)
   <- e, ee
   -> s, se
```

Il protocollo inizia con la "pre-trasmissione" della chiave statica (`rs`) del risponditore all'iniziatore. Prima di eseguire l'handshake, l'iniziatore deve generare la propria chiave statica (s). Durante ogni fase dell'handshake, tutto il materiale inviato attraverso il filo e le chiavi inviate/utilizzate vengono sottoposte ad hashing incrementale in un *digest dell'handshake*, h. Questo digest non viene mai inviato attraverso il filo durante l'handshake e viene invece utilizzato come "dati associati" quando AEAD (crittografia autenticata con dati associati) viene inviato attraverso il filo. I *dati associati* (AD) consentono a un protocollo di crittografia di autenticare informazioni aggiuntive insieme a un pacchetto di testo cifrato. In altri domini, l'AD può essere un nome di dominio o una porzione di testo del pacchetto.

L'esistenza di h garantisce che se una parte di un messaggio di handshake trasmesso viene sostituita, l'altra parte se ne accorgerà. Ad ogni passaggio, viene controllato un digest MAC. Se il controllo MAC ha esito positivo, la parte ricevente sa che l'handshake ha avuto successo fino a quel momento. Altrimenti, se un controllo MAC fallisce, il processo di handshake non è riuscito e la connessione dovrebbe essere terminata.

Il protocollo aggiunge anche un nuovo dato a ogni messaggio di handshake: una versione del protocollo. La versione iniziale del protocollo è `0`. Al momento della scrittura, non sono state create nuove versioni del protocollo. Di conseguenza, se un peer riceve una versione diversa da `0`, dovrebbe rifiutare il tentativo di avvio dell'handshake.

Per quanto riguarda le primitive crittografiche, SHA-256 viene utilizzato come funzione hash

preferita, `secp256k1` come curva ellittica e `ChaChaPoly-130` come costruzione AEAD (crittografia simmetrica).

Ogni variante del Noise Protocol ha una stringa ASCII univoca utilizzata per fare riferimento ad esso. Per garantire che due parti utilizzino la stessa variante di protocollo, la stringa ASCII viene sottoposta ad hashing in un digest, che viene utilizzato per inizializzare lo stato iniziale dell'handshake. Nel contesto di Lightning Network, la stringa ASCII che descrive il protocollo è `Noise_XK_secp256k1_ChaChaPoly_SHA256`.

Handshake in Tre Atti

La parte della stretta di mano può essere suddivisa in tre distinti "atti". L'handshake completa richiede 1,5 round andata e ritorno tra l'iniziatore e il risponditore. Ad ogni atto, viene inviato un singolo messaggio tra le due parti. Il messaggio di handshake è un payload di dimensioni fisse preceduto dalla versione del protocollo.

Il Protocollo Noise utilizza una notazione ispirata agli oggetti per descrivere il protocollo in ogni fase. Durante l'impostazione dello stato di handshake, ciascuna parte inizializzerà le seguenti variabili:

`ck`

: La *chiave del concatenamento*. Questo valore è l'hash accumulato di tutti i precedenti output ECDH. Alla fine dell'handshake, `ck` viene utilizzato per derivare le chiavi di crittografia per i messaggi Lightning.

`h`

: L'*handshake hash*. Questo valore è l'hash accumulato di *tutti* i dati di handshake che sono stati inviati e ricevuti durante il processo di handshake.

`temp_k1, temp_k2, temp_k3`

: Le *chiavi intermedie*. Queste vengono utilizzate per crittografare e decrittografare i payload AEAD di lunghezza zero alla fine di ogni messaggio di handshake.

`e`

La *coppia di chiavi effimere* di un gruppo. Per ogni sessione, un nodo deve generare una nuova chiave temporanea con una forte casualità crittografica.

`s`

La *coppia di chiavi statiche* di un gruppo (`ls` per locale, `rs` per remoto).

Dato questo handshake più lo stato della sessione di messaggistica, definiremo quindi una serie di funzioni che opereranno sullo stato di handshake e messaggistica. Quando descriviamo il protocollo di handshake, useremo queste variabili in modo simile allo pseudocodice per ridurre la verbosità della spiegazione di ogni passaggio del protocollo. Definiremo le primitive *funzionali* dell'handshake come:

`ECDH(k, rk)`

Esegue un'operazione Diffie-Hellman a curva ellittica utilizzando `k`, che è una chiave privata `secp256k1` valida, e `rk`, che è una chiave pubblica valida.

Il valore restituito è SHA-256 del formato compresso del punto generato.

`HKDF(salt,ikm)`

Una funzione definita in `RFC 5869`, valutata con un campo `info` di lunghezza zero.

Tutte le chiamate di `HKDF` restituiscono implicitamente 64 byte di casualità crittografica utilizzando il componente di estrazione ed espansione di `HKDF`.

`encryptWithAD(k, n, ad, plaintext)`

Mostra l'output `encrypt(k, n, ad, plaintext)`.

Dove `encrypt` è una valutazione di `ChaCha20-Poly1305` (variante Internet Engineering Task Force) con gli argomenti passati, con nonce `n` codificato come 32 bit di zeri, seguito da un valore *little-endian* a 64 bit. Nota: questo segue la convenzione del Protocollo Noise, piuttosto che il nostro normale endian.

`decryptWithAD(k, n, ad, ciphertext)`

Mostra l'output `decrypt(k, n, ad, ciphertext)`.

Dove `decrypt` è una valutazione di `ChaCha20-Poly1305` (variante IETF) con gli argomenti passati, con nonce `n` codificato come 32 bit di zeri, seguito da un valore *little-endian* a 64 bit.

`generateKey()`

Genera e restituisce una nuova coppia di chiavi `secp256k1`.

Dove l'oggetto restituito da `generateKey` ha due attributi: `.pub`, che restituisce un oggetto astratto che rappresenta la chiave pubblica e `.priv`, che rappresenta la chiave privata utilizzata per generare la chiave pubblica

Dove l'oggetto ha anche un singolo metodo: `.serializeCompressed()`

`a || b`

Questo denota la concatenazione di due stringhe di byte `a` e `b`.

Inizializzazione dello stato della sessione di handshake

Prima di avviare il processo di handshake, entrambe le parti devono inizializzare lo stato primario che useranno per far avanzare il processo di handshake. Per iniziare, entrambe le parti devono costruire il digest primario della handshake `h`.

1. `h = SHA-256(protocolName)`

   Dove `protocolName = "Noise_XK_secp256k1_ChaChaPoly_SHA256"` codificato come stringa ASCII.

2. `ck = h`

3. `h = SHA-256(h || prologue)`

   Dove `prologue` è la stringa ASCII: `lightning`.

Oltre al nome del protocollo, aggiungiamo anche un "prologo" aggiuntivo utilizzato per associare ulteriormente il contesto del protocollo alla rete Lightning.

Per concludere la fase di inizializzazione, entrambe le parti mescolano la chiave pubblica del risponditore nel digest dell'handshake. Poiché questo digest viene utilizzato mentre vengono inviati i dati associati con un testo cifrato di lunghezza zero (solo il MAC), ciò garantisce che l'iniziatore conosca effettivamente la chiave pubblica del risponditore.

- Il nodo di avvio mescola la chiave pubblica statica del nodo di risposta serializzata nel formato compresso di Bitcoin: `h = SHA-256(h || rs.pub.serializeCompressed())`
- Il nodo che risponde mescola la propria chiave pubblica statica locale serializzata nel formato compresso di Bitcoin: `h = SHA-256(h || ls.pub.serializeCompressed())`

Atti di Handshake

Dopo l'inizializzazione primaria dell'handshake, possiamo iniziare l'effettiva esecuzione del processo. L'handshake è composta da una serie di tre messaggi inviati tra l'iniziatore e il risponditore, d'ora in poi denominati "atti". Poiché ogni atto è un singolo messaggio inviato tra le parti, una stretta di mano viene completata in un totale di 1,5 viaggi di andata e ritorno (0,5 per ogni atto).

Il primo atto completa la parte iniziale del triplo scambio di chiavi incrementali Diffie-Hellman (utilizzando una nuova chiave effimera generata dall'iniziatore) e garantisce anche che l'iniziatore conosca effettivamente la chiave pubblica a lungo termine del risponditore. Durante il secondo atto, il risponditore trasmette all'iniziatore la chiave effimera che desidera utilizzare per la sessione, e ancora una volta mescola in modo incrementale questa nuova chiave nella tripla handshake DH. Durante il terzo e ultimo atto, l'iniziatore trasmette la propria chiave pubblica statica a lungo termine al risponditore ed esegue l'operazione DH finale per mescolarla nel segreto condiviso finale risultante.

Primo Atto

```
-> e, es
```

Il primo atto viene inviato dall'iniziatore al risponditore. Durante il primo atto, l'iniziatore tenta di soddisfare una sfida implicita da parte del rispondente. Per completare questa sfida, l'iniziatore deve conoscere la chiave pubblica statica del risponditore.

Il messaggio di handshake è *esattamente* di 50 byte: 1 byte per la versione di handshake, 33 byte per la chiave pubblica temporanea compressa dell'iniziatore e 16 byte per il tag `poly1305`.

Azioni del mittente:

1. `e = generateKey()`
2. `h = SHA-256(h || e.pub.serializeCompressed())`

   La chiave temporanea appena generata viene accumulata nel digest dell'handshake in esecuzione.

3. `es = ECDH(e.priv, rs)`

   L'iniziatore esegue un ECDH tra la sua chiave temporanea appena generata e la chiave pubblica statica del nodo remoto.

4. `ck, temp_k1 = HKDF(ck, es)`

   Viene generata una nuova chiave di crittografia temporanea, che viene utilizzata per generare il MAC di autenticazione.

5. `c = encryptWithAD(temp_k1, 0, h, zero)`

   Dove `zero` è un testo normale di lunghezza zero.

6. `h = SHA-256(h || c)`

   Infine, il testo cifrato generato viene accumulato nel digest dell'handshake di autenticazione.

7. Viene inviato `m = 0 || e.pub.serializeCompressed() || c` al risponditore tramite il buffer di rete.

Azioni del destinatario:

1. Legge *esattamente* 50 byte dal buffer di rete.
2. Analizza il messaggio letto (`m`) in `v`, `re`, and `c`:

- Dove v è il *primo* byte di m, re sono i successivi 33 byte di m e c sono gli ultimi 16 byte di m.
- I byte grezzi della chiave pubblica effimera (re) della parte remota devono essere de-serializzati in un punto sulla curva utilizzando coordinate affini come codificate dal formato composto serializzato della chiave.

3. Se v è una versione di handshake non riconosciuta, il risponditore deve interrompere il tentativo di connessione.
4. `h = SHA-256(h || re.serializeCompressed())`

   Il risponditore accumula la chiave effimera dell'iniziatore nel digest di autenticazione della stretta di mano.
5. `es = ECDH(s.priv, re)`

   Il risponditore esegue un ECDH tra la sua chiave privata statica e la chiave pubblica temporanea dell'iniziatore.
6. `ck, temp_k1 = HKDF(ck, es)`

   Viene generata una nuova chiave di crittografia temporanea, che verrà utilizzata a breve per controllare il MAC di autenticazione.
7. `p = decryptWithAD(temp_k1, 0, h, c)`

   Se il controllo MAC in questa operazione fallisce, l'iniziatore non conosce la chiave pubblica statica del risponditore. In tal caso, il risponditore deve terminare la connessione senza ulteriori messaggi.
8. `h = SHA-256(h || c)`

   Il testo cifrato ricevuto viene inserito nel digest dell'handshake. Questo passaggio serve a garantire che il payload non sia stato modificato da un MITM.

Secondo Atto

```
<- e, ee
```

Il secondo atto viene inviato dal risponditore all'iniziatore. Il secondo atto avrà luogo *solo se* il primo atto ha avuto successo. Il primo atto ha avuto successo se il risponditore è stato in grado di decrittografare e controllare correttamente il MAC del tag inviato alla fine del primo

atto.

L'handshake è *esattamente* di 50 byte: 1 byte per la versione handshake, 33 byte per la chiave pubblica temporanea compressa del risponditore e 16 byte per il tag `poly1305`.

Azioni del mittente:

1. `e = generateKey()`
2. `h = SHA-256(h || e.pub.serializeCompressed())`

   La chiave temporanea appena generata viene accumulata nel digest dell'handshake in esecuzione.
3. `ee = ECDH(e.priv, re)`

   Dove `re` è la chiave effimera dell'iniziatore, ricevuta durante il primo atto.
4. `ck, temp_k2 = HKDF(ck, ee)`

   Viene generata una nuova chiave di crittografia temporanea, che viene utilizzata per generare il MAC di autenticazione.
5. `c = encryptWithAD(temp_k2, 0, h, zero)`

   Dove `zero` è un testo normale di lunghezza zero.
6. `h = SHA-256(h || c)`

   Infine, il testo cifrato generato viene accumulato nel digest dell'handshake di autenticazione.
7. Invia `m = 0 || e.pub.serializeCompressed() || c` all'iniziatore tramite il buffer di rete.

Azioni del destinatario:

1. Legge esattamente 50 byte dal buffer di rete.
2. Analizza il messaggio letto (`m`) in `v`, `re` e `c`:

   Dove `v` è il primo byte di `m`, `re` sono i successivi 33 byte di `m` e `c` sono gli ultimi 16 byte di `m`.

3. Se v è una versione di handshake non riconosciuta, il risponditore deve interrompere il tentativo di connessione.
4. `h = SHA-256(h || re.serializeCompressed())`
5. `ee = ECDH(e.priv, re)`

   Dove `re` è la chiave pubblica effimera del risponditore.

   I byte grezzi della chiave pubblica effimera (`re`) della parte remota devono essere de-serializzati in un punto sulla curva utilizzando coordinate affini come codificate dal formato composto serializzato della chiave.
6. `ck, temp_k2 = HKDF(ck, ee)`

   Viene generata una nuova chiave di crittografia temporanea, che viene utilizzata per generare il MAC di autenticazione.
7. `p = decryptWithAD(temp_k2, 0, h, c)`

   Se il controllo MAC in questa operazione fallisce, l'iniziatore deve terminare la connessione senza ulteriori messaggi.
8. `h = SHA-256(h || c)`

   Il testo cifrato ricevuto viene inserito nel digest dell'handshake. Questo passaggio serve a garantire che il payload non sia stato modificato da un MITM.

Terzo Atto

```
-> s, se
```

Il terzo atto è la fase finale dell'accordo chiave autenticato descritto in questa sezione. Questo atto viene inviato dall'iniziatore al risponditore come passaggio conclusivo. Il terzo atto viene eseguito *se e solo se* il secondo atto ha avuto successo. Durante il terzo atto, l'iniziatore trasporta la sua chiave pubblica statica al risponditore crittografata con una *forte* segretezza futura, utilizzando la chiave segreta derivata da `HKDF` accumulata a questo punto dell'handshake.

L'handshake è *esattamente* di 66 byte: 1 byte per la versione handshake, 33 byte per la chiave pubblica statica crittografata con il cifrario a flusso `ChaCha20`, 16 byte per il tag della chiave pubblica crittografata generato tramite la costruzione AEAD e 16 byte per un tag di

autenticazione finale .

Azioni del mittente:

1. `c = encryptWithAD(temp_k2, 1, h, s.pub.serializeCompressed())`

   Dove s è la chiave pubblica statica dell'iniziatore.

2. `h = SHA-256(h || c)`
3. `se = ECDH(s.priv, re)`

   Dove `re` è la chiave pubblica temporanea del risponditore.

4. `ck, temp_k3 = HKDF(ck, se)`

   Il segreto condiviso intermedio finale viene inserito nella chiave di concatenamento in esecuzione.

5. `t = encryptWithAD(temp_k3, 0, h, zero)`

   Dove `zero` è un testo normale di lunghezza zero.

6. `sk, rk = HKDF(ck, zero)`

   Dove `zero` è un testo normale di lunghezza zero, `sk` è la chiave che deve essere utilizzata dall'iniziatore per crittografare i messaggi al risponditore e `rk` è la chiave che deve essere utilizzata dall'iniziatore per decrittografare i messaggi inviati dal risponditore.

   Vengono generate le chiavi di cifratura finali, da utilizzare per l'invio e la ricezione dei messaggi per tutta la durata della sessione.

7. `rn = 0, sn = 0`

   I nonce di invio e ricezione sono inizializzati a `0`.

8. Invia `m = 0 || c || t` tramite il buffer di rete.

Azioni del destinatario:

1. Legge esattamente 66 byte dal buffer di rete.

2. Analizza il messaggio letto (`m`) in `v`, `c` e `t`:

   Dove `v` è il primo byte di `m`, `c` sono i successivi 49 byte di `m` e `t` sono gli ultimi 16 byte di `m`.

3. Se `v` è una versione di handshake non riconosciuta, il risponditore deve interrompere il tentativo di connessione.

4. `rs = decryptWithAD(temp_k2, 1, h, c)`

   A questo punto, il risponditore ha recuperato la chiave pubblica statica dell'iniziatore.

5. `h = SHA-256(h || c)`

6. `se = ECDH(e.priv, rs)`

   Dove e è la chiave effimera originale del rispondente.

7. `ck, temp_k3 = HKDF(ck, se)`

8. `p = decryptWithAD(temp_k3, 0, h, t)`

   Se il controllo MAC in questa operazione fallisce, il risponditore deve terminare la connessione senza ulteriori messaggi.

9. `rk, sk = HKDF(ck, zero)`

   Dove `zero` è un testo normale di lunghezza zero, `rk` è la chiave che deve essere utilizzata dal risponditore per decrittografare i messaggi inviati dall'iniziatore e `sk` è la chiave che deve essere utilizzata dal risponditore per crittografare i messaggi all'iniziatore.

   Vengono generate le chiavi di cifratura finali, da utilizzare per l'invio e la ricezione dei messaggi per tutta la durata della sessione.

10. `rn = 0, sn = 0`

    I nonce di invio e ricezione sono inizializzati a `0`.

Crittografia dei messaggi di trasporto

Alla conclusione del terzo atto, entrambe le parti hanno ricavato le chiavi di crittografia, che verranno utilizzate per crittografare e decrittografare i messaggi per il resto della sessione.

I messaggi effettivi del protocollo Lightning sono incapsulati all'interno di testi cifrati AEAD. Ogni messaggio è preceduto da un altro testo cifrato AEAD, che codifica la lunghezza totale del seguente messaggio Lightning (escluso il suo MAC).

La dimensione *massima* di *qualsiasi* messaggio Lightning non deve superare i 65.535 byte. Una dimensione massima di 65.535 semplifica i test, semplifica la gestione della memoria e aiuta a mitigare gli attacchi di esaurimento della memoria.

Per rendere più difficile l'analisi del traffico, viene crittografato anche il prefisso di lunghezza per tutti i messaggi Lightning. Inoltre, al prefisso della lunghezza crittografata viene aggiunto un tag `Poly-1305` da 16 byte per garantire che la lunghezza del pacchetto non sia stata modificata durante il transito e per evitare la creazione di un oracolo di decrittazione.

La struttura dei pacchetti sul filo ricorda il diagramma della Figura 14-2.

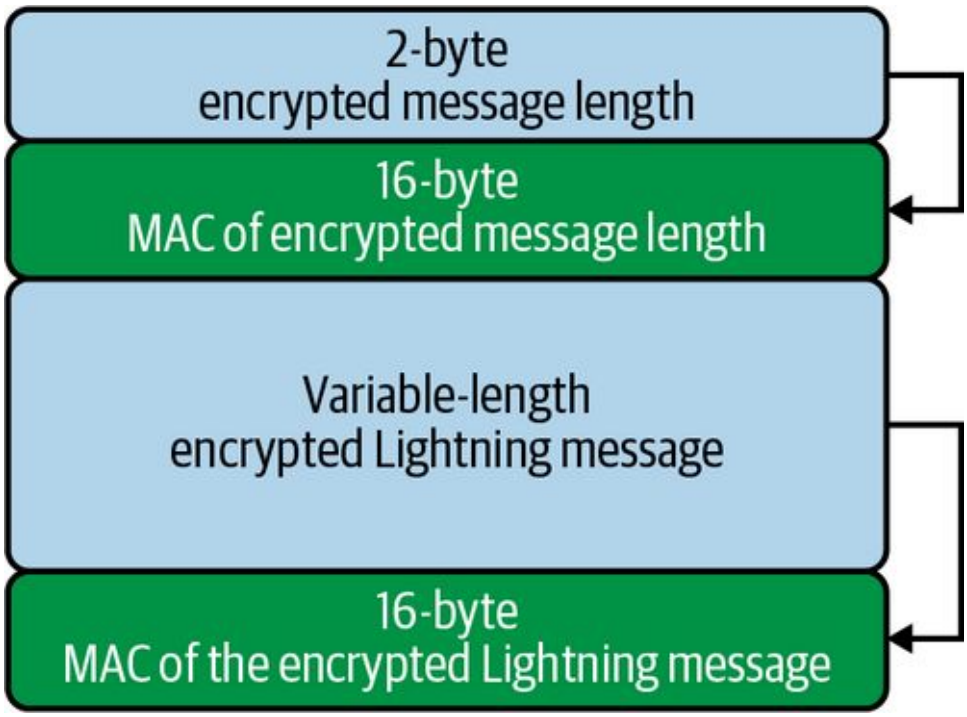

Figura 14-2. Struttura del pacchetto crittografato

La lunghezza del messaggio prefissata è codificata come numero intero big-endian a 2 byte, per una lunghezza massima totale del pacchetto di 2 + 16 + 65.535 + 16 = 65.569 byte.

Crittografia e invio di messaggi

Per crittografare e inviare un messaggio Lightning (`m`) al flusso di rete, data una chiave di invio (`sk`) e un nonce (`sn`), vengono completati i seguenti passaggi:

1. Lasciare che `l = len(m)`.

   Dove `len` ottiene la lunghezza in byte del messaggio Lightning.

2. Serializzazione `l` in 2 byte codificati come numero intero big-endian.
3. Crittografia di `l` (usando `ChaChaPoly-1305`, `sn` e `sk`), per ottenere `lc` (18 byte).
   - Il nonce `sn` è codificato come numero little-endian a 96 bit. Poiché il nonce decodificato è di 64 bit, il nonce a 96 bit è codificato come 32 bit di zeri iniziali seguiti da un valore di 64 bit.
   - Il nonce `sn` deve essere incrementato dopo questo passaggio.
   - Una porzione di byte di lunghezza zero deve essere passata come AD (dati associati).
4. Infine, crittografia del messaggio stesso (`m`) utilizzando la stessa procedura utilizzata per crittografare il prefisso di lunghezza. Questo testo cifrato è noto come `c`.

   Il nonce `sn` deve essere incrementato dopo questo passo.

5. Invio di `lc || c` sul buffer di rete.

Ricevere e decifrare messaggi

Per decrittografare il messaggio successivo nel flusso di rete, vengono completati i seguenti passaggi:

1. Lettura esatta di 18 byte dal buffer di rete.
2. Lasciare che il prefisso di lunghezza crittografato sia noto come `lc`.
3. Decrittografare `lc` (utilizzando `ChaCha20-Poly1305`, `rn` e `rk`) per ottenere la dimensione del pacchetto crittografato `l`.
   - Una porzione di byte di lunghezza zero deve essere passata come AD (dati associati).

- Il nonce `rn` deve essere incrementato dopo questo passo.

4. Leggere *esattamente* `l + 16 byte` dal buffer di rete e lasciare che i byte siano noti come c.
5. Decrittografare c (utilizzando `ChaCha20-Poly1305`, `rn` e `rk`) per ottenere il pacchetto di testo in chiaro decrittografato `p`.

   Il nonce `rn` deve essere incrementato dopo questo passo.

Rotazione della chiave del messaggio Lightning

Cambiare le chiavi regolarmente e dimenticare le chiavi precedenti è utile per prevenire la decrittazione di vecchi messaggi, in caso di successiva fuga di chiavi (segretezza all'indietro).

La rotazione delle chiavi viene eseguita singolarmente per *ogni* chiave (`sk` e `rk`). Una chiave deve essere ruotata dopo che una parte ha crittografato o decrittografato 1.000 volte con essa (ovvero ogni 500 messaggi). Questo può essere adeguatamente spiegato ruotando la chiave una volta che il nonce ad esso dedicato supera 1.000.

La rotazione per una chiave k viene eseguita secondo i seguenti passaggi:

1. Lasciare che `ck` sia la chiave di concatenamento ottenuta alla fine del terzo atto.
2. `ck', k' = HKDF(ck, k)`
3. Reimpostare il nonce per la chiave su `n = 0`.
4. `k = k'`
5. `ck = ck'`

Conclusione

La crittografia di trasporto di Lightning si basa sul Protocollo Noise e offre solide garanzie di sicurezza, privacy, autenticità e integrità per tutte le comunicazioni tra peer.

A differenza di Bitcoin, dove i peer spesso comunicano "in chiaro" (senza crittografia), tutte le comunicazioni Lightning sono crittografate peer-to-peer. Oltre alla crittografia del trasporto (peer-to-peer), su Lightning Network, *anche* i pagamenti vengono crittografati in pacchetti onion (hop-to-hop) e i dettagli del pagamento vengono inviati fuori banda tra il mittente e il

destinatario (end-to-end). La combinazione di tutti questi meccanismi di sicurezza è cumulativa e fornisce una difesa a più livelli contro la de-anonimizzazione, gli attacchi man-in-the-middle e la sorveglianza della rete.

Naturalmente, nessuna sicurezza è perfetta e vedremo nel Capitolo 16 che queste proprietà possono essere degradate e attaccate. Tuttavia, Lightning Network migliora significativamente la privacy di Bitcoin.

Capitolo 15. Richieste di Pagamento Lightning

In questo capitolo esamineremo le *richieste di pagamento Lightning* (*payment requests*) o, come sono più comunemente note, le *fatture Lightning* (*invoices*).

Fatture nella Suite di Protocolli Lightning

Le *richieste di pagamento*, note anche come *fatture*, fanno parte del livello di pagamento e sono mostrate in alto a sinistra nella Figura 15-1.

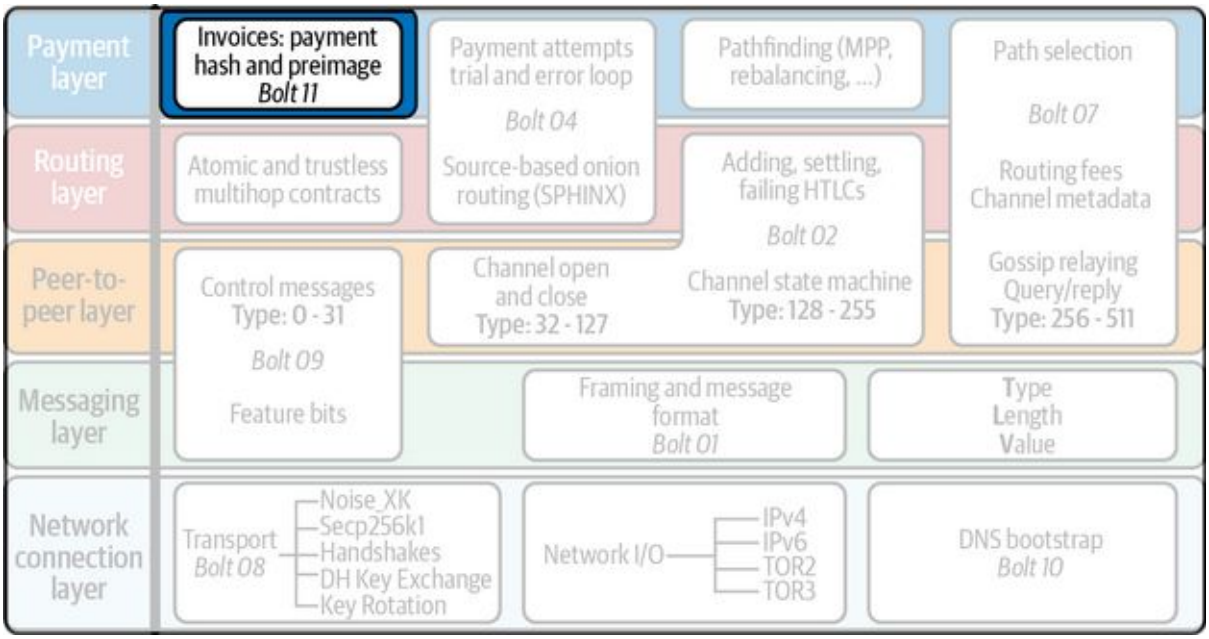

Figura 15-1. Richieste di pagamento nella suite di protocolli Lightning

Introduzione

Come abbiamo appreso nel corso del libro, per completare un pagamento Lightning sono necessari almeno due dati: un hash di pagamento e una destinazione. Poiché SHA-256 viene

utilizzato nella rete Lightning per implementare gli HTLC, queste informazioni richiedono 32 byte essere comunicate. Le destinazioni, invece, sono semplicemente la chiave pubblica `secp256k1` del nodo che desidera ricevere un pagamento. Lo scopo di una richiesta di pagamento nel contesto di Lightning Network è comunicare queste due informazioni dal mittente al destinatario. Il formato compatibile con il codice QR per comunicare le informazioni necessarie per completare un pagamento dal destinatario al mittente è descritto in BOLT #11: Invoice Protocol for Lightning Payments. In pratica, in una richiesta di pagamento vengono comunicati molto più che solamente l'hash e la destinazione del pagamento per rendere la codifica più completa.

Richieste di Pagamento Lightning Rispetto a Indirizzi Bitcoin

Una domanda frequente che la gente si pone quando incontra per la prima volta una richiesta di pagamento Lightning è: perché non è possibile utilizzare un normale formato di indirizzo statico?

Per rispondere a questa domanda, devi prima interiorizzare in che modo Lightning differisce dal livello base Bitcoin come metodo di pagamento. Rispetto a un indirizzo Bitcoin che può essere utilizzato per effettuare un numero potenzialmente illimitato di pagamenti (sebbene il riutilizzo di un indirizzo Bitcoin possa degradare la propria privacy), una richiesta di pagamento Lightning dovrebbe essere utilizzata *solo una volta*. Ciò è dovuto al fatto che l'invio di un pagamento a un indirizzo Bitcoin utilizza essenzialmente un sistema crittografico a chiave pubblica per "codificare" il pagamento in modo che solo il vero "proprietario" di quell'indirizzo Bitcoin possa riscattarlo.

Al contrario, per completare un pagamento Lightning, il destinatario deve rivelare un "segreto" all'intero percorso di pagamento, incluso il mittente. Questo può essere interpretato come l'utilizzo di una sorta di crittografia simmetrica specifica del dominio perché la preimmagine di pagamento è per scopi pratici un nonce (numero utilizzato solo una volta). Se il mittente tenta di effettuare un altro pagamento utilizzando lo stesso hash di pagamento, rischia di perdere fondi perché il pagamento potrebbe non essere effettivamente consegnato alla destinazione. È lecito ritenere che dopo che una preimage è stata rivelata, tutti i nodi nel percorso la manterranno per sempre, quindi invece di inoltrare l'HTLC per riscuotere una tariffa di instradamento se il pagamento è completato, possono semplicemente saldare il pagamento in quell'istanza e ottenere l'intero importo del

pagamento in cambio. Di conseguenza, non è sicuro utilizzare una richiesta di pagamento più di una volta.

Esistono nuove varianti della richiesta di pagamento Lightning originale che consentono al mittente di riutilizzarle tutte le volte che desidera. Queste varianti capovolgono il normale flusso di pagamento poiché il mittente trasmette una preimage all'interno del'onion payload crittografato al destinatario, che è l'unico in grado di decrittografarlo e saldare il pagamento. In alternativa, supponendo un meccanismo che consenta a un mittente di richiedere in genere una nuova richiesta di pagamento al destinatario, è possibile utilizzare un protocollo interattivo per consentire un certo grado di riutilizzo della richiesta di pagamento.

BOLT #11: Serializzazione ed Interpretazione delle Richieste di Pagamento Lightning

In questa sezione, descriveremo il meccanismo utilizzato per codificare il set di informazioni necessarie per completare un pagamento su Lightning Network. Come accennato in precedenza, l'hash e la destinazione del pagamento rappresentano la quantità minima di informazioni necessarie per completare un pagamento. Tuttavia vengono comunicate anche ulteriori informazioni come le informazioni sul timelock, la scadenza della richiesta di pagamento e possibilmente un indirizzo di fallback on-chain. Il documento completo delle specifiche è BOLT #11: Invoice Protocol for Lightning Payments.

Spiegazione della Codifica delle Richieste di Pagamento

Innanzitutto, esaminiamo come si presenta in pratica una vera richiesta di pagamento. Quella che segue è una richiesta di pagamento valida che potrebbe essere stata utilizzata per completare un pagamento sulla mainnet di LN al momento della sua creazione:

```
lnbc2500u1pvjluezpp5qqqsyqcyq5rqwzqfqqqsyqcyq5rqwzqfqqqsyqcyq5rqwzqfqypqdq5xysx
xatsyp3k7enxv4jsxqzpuaztrnwngzn3kdzw5hydlzf03qdgm2hdq27cqv3agm2awhz5se903vruatf
hq77w3ls4evs3ch9zw97j25emudupq63nyw24cg27h2rspfj9srp
```

Un Prefisso Leggibile dall'Uomo

Guardando la stringa, possiamo estrarre una parte che possiamo analizzare con i nostri

occhi, mentre il resto sembra solo un insieme casuale di stringhe. La parte che è in qualche modo analizzabile da un essere umano è indicata come *prefisso leggibile dall'uomo*. Consente a un essere umano di estrarre rapidamente alcune informazioni rilevanti da una richiesta di pagamento a colpo d'occhio. In questo caso, possiamo vedere che questo pagamento è per l'istanza mainnet di Lightning Network (`lnbc`) e richiede 2.500 uBTC (microbitcoin) o 25.000.000 sats. L'ultima parte è indicata come porzione di dati e utilizza un formato estensibile per codificare le informazioni necessarie per completare un pagamento.

Ogni versione dell'istanza di Lightning Network (mainnet, testnet, ecc.) ha il proprio prefisso leggibile dall'uomo (vedi la Tabella 15-1). Ciò consente al software client e anche agli esseri umani di determinare rapidamente se una richiesta di pagamento può essere soddisfatta o meno dal proprio nodo.

Tabella 15-1. Prefissi di rete BOLT #11

Rete	Predisso BOLT #11
mainnet	`lnbc`
testnet	`lntb`
simnet/regtest	`lnbcrt`

La prima parte del prefisso leggibile dall'uomo è un'espressione *compatta* dell'importo della richiesta di pagamento. L'importo compatto è codificato in due sottoparti. Innanzitutto, viene utilizzato un numero intero come importo di *base*. Questo è poi seguito da un moltiplicatore che ci permette di specificare distinti aumenti in ordini di grandezza compensati dall'importo base. Se torniamo al nostro esempio iniziale, possiamo prendere la porzione di `2500u` e ridurla di un fattore 1.000 per utilizzare invece `2500m` (2.500 mBTC). Come regola generale, per accertare a colpo d'occhio l'importo di una fattura, prendi il fattore base e moltiplicalo per il moltiplicatore.

Un elenco completo dei moltiplicatori attualmente definiti è riportato nella Tabella 15-2.

Tabella 15-2. Moltiplicatori di importo di BOLT #11

Moltiplicatore	Unità di bitcoin	Fattore di moltiplicazione
`m`	milli	0.001
`u`	micro	0.000001
`n`	nano	0.000000001
`p`	pico	0.000000000001

bech32 e il Segmento Dati

Se la parte "illeggibile" sembra familiare, è perché utilizza lo stesso schema di codifica utilizzato dagli indirizzi Bitcoin compatibili con SegWit: bech32. La descrizione dello schema di codifica bech32 esula dagli scopi di questo capitolo. In breve, è un modo sofisticato per codificare stringhe brevi che ha un'ottima correzione degli errori e proprietà di rilevamento.

La porzione di dati può essere suddivisa in tre sezioni:

- Il timestamp (marca temporale)
- Zero o più coppie chiave-valore codificate
- La firma dell'intera fattura

Il timestamp è espresso in secondi dall'anno 1970, o Unix Epoch. Questo timestamp consente al mittente di valutare quanti anni ha la fattura e, come vedremo, consente al destinatario di forzare la validità di una fattura solo per un periodo di tempo, se lo desidera.

Simile al formato TLV di cui abbiamo parlato in precedenza, il formato di fattura BOLT #11 utilizza una serie di coppie chiave-valore estensibili per codificare le informazioni necessarie per soddisfare un pagamento. Poiché vengono utilizzate coppie chiave-valore, è facile aggiungere nuovi valori in futuro se viene introdotto un nuovo tipo di pagamento o un requisito/funzionalità aggiuntivo.

Infine, è inclusa una firma che copre l'intera fattura firmata dalla destinazione del pagamento. Questa firma consente al mittente di verificare che la richiesta di pagamento sia stata effettivamente creata dal destinatario del pagamento. A differenza delle richieste di pagamento Bitcoin che non sono firmate, ciò ci consente di garantire che una particolare entità abbia firmato la richiesta di pagamento. La firma stessa è codificata utilizzando un ID di ripristino, che consente di utilizzare una firma più compatta che consente l'estrazione della chiave pubblica. Durante la verifica della firma, l'ID di recupero estrae la chiave pubblica, quindi la confronta con la chiave pubblica inclusa nella fattura.

Campi fattura contrassegnati

I campi della fattura contrassegnati sono codificati nel corpo principale della fattura. Questi campi rappresentano diverse coppie chiave-valore che esprimono informazioni aggiuntive che possono aiutare a completare il pagamento o informazioni *necessarie* per completare il pagamento. Poiché viene utilizzata una leggera variante di bech32, ciascuno di questi campi si trova effettivamente nel dominio "base 5".

Un dato campo tag è composto da tre componenti:

- Il `tipo` di campo (5 bit)
- La `lunghezza` dei dati del campo (10 bit)
- I `dati` stessi, che hanno una dimensione `lunghezza * 5 byte`

Un elenco completo di tutti i campi con tag attualmente definiti è mostrato in Tabella 15-3.

Tabella 15-3. Campi della fattura contrassegnati di BOLT #11

Etichetta	Lunghezza dati	Utilizzo
p	52	L'hash di pagamento SHA-256.
s	52	Un segreto a 256 bit che aumenta la privacy end-to-end di un pagamento mitigando il sondaggio da parte dei nodi intermedi.
d	Variabile	La descrizione, una breve stringa UTF-8 dello scopo del

Etichetta	Lunghezza dati	Utilizzo
		pagamento.
n	53	La chiave pubblica del nodo di destinazione.
h	52	Un hash che rappresenta una descrizione del pagamento stesso. Questo può essere utilizzato per impegnarsi in una descrizione che supera i 639 byte di lunghezza.
x	Variabile	Il tempo di scadenza, in secondi, del pagamento. Il valore predefinito è 1 ora (3.600) se non specificato.
c	Variabile	Il `min_cltv_expiry` da utilizzare per l'hop finale nel percorso. Il valore predefinito è 9 se non specificato.
f	Variabile	Un indirizzo on-chain di riserva da utilizzare per completare il pagamento se tale non può essere completato tramite LN.
r	Variabile	Una o più voci che consentono a un destinatario di fornire al mittente ulteriori dati effimeri per completare il pagamento.
9	Variabile	Un insieme di valori a 5 bit che contengono i bit di funzionalità richiesti per completare il pagamento.

Gli elementi contenuti nel campo `r` sono comunemente indicati come *suggerimenti di instradamento* (*routing hints*). Consentono al destinatario di comunicare una serie aggiuntiva di canali che possono aiutare il mittente a completare il pagamento. I suggerimenti vengono solitamente utilizzati quando il destinatario ha alcuni/tutti i canali privati e desidera guidare il mittente in questa porzione "non mappata" del grafo dei canali. Un routing hint codifica efficacemente le stesse informazioni di un normale messaggio `channel_update`. L'aggiornamento stesso è racchiuso in un singolo valore con i seguenti campi:

- La pubkey del nodo in uscita nel canale (264 bit)

- Lo `short_channel_id` del canale "virtuale" (64 bit)
- La tariffa base (`fee_base_msat`) del canale (32 bit)
- Il compenso proporzionale (`fee_proportional_millionths`) (32 bit)
- Il delta di scadenza CLTV (`cltv_expiry_delta`) (16 bit)

La parte finale del segmento di dati è l'insieme di bit di funzionalità che comunicano al mittente la funzionalità necessaria per completare un pagamento. Ad esempio, se in futuro viene aggiunto un nuovo tipo di pagamento che non è retrocompatibile con il tipo di pagamento originale, il destinatario può impostare un bit di funzionalità *richiesto* per comunicare che il pagatore deve comprendere tale funzionalità per completare il pagamento.

Conclusione

Come abbiamo visto, le fatture sono molto più di una semplice richiesta di denaro. Contengono informazioni critiche su *come* effettuare il pagamento, come i suggerimenti di instradamento, la chiave pubblica del nodo di destinazione, le chiavi temporanee per aumentare la sicurezza e molto altro.

Capitolo 16. Sicurezza e Privacy di Lightning Network

In questo capitolo esaminiamo alcune delle questioni più importanti relative alla sicurezza e alla privacy di Lightning Network. Innanzitutto, parleremo di privacy, cosa significa, come valutarla e alcune cose che puoi fare per proteggere la tua privacy durante l'utilizzo di Lightning Network. Quindi esploreremo alcuni attacchi comuni e tecniche di mitigazione.

Perché la Privacy È Importante?

La proposta di valore delle criptovalute è denaro resistente alla censura. Bitcoin offre ai partecipanti la possibilità di immagazzinare e trasferire la propria ricchezza senza interferenze da parte di governi, banche o società. LN continua questa missione.

A differenza delle banali soluzioni di scalabilità come "banche Bitcoin custodial", Lightning Network mira a ridimensionare Bitcoin senza compromettere l'autocustodia, il che dovrebbe portare a una maggiore resistenza alla censura nell'ecosistema Bitcoin. Tuttavia, LN opera con un modello di sicurezza diverso, che introduce nuove sfide per la sicurezza e la privacy.

Definizioni di Riservatezza

La domanda "Lightning Network è privato?" non ha una risposta diretta. La privacy è un argomento complesso; spesso è difficile definire con precisione si intende per privacy, in particolare se non sei un ricercatore sulla privacy. Fortunatamente, i ricercatori sulla privacy utilizzano processi per analizzare e valutare le caratteristiche di privacy dei sistemi, e anche noi possiamo usarli! Diamo un'occhiata a come un ricercatore di sicurezza potrebbe cercare di rispondere alla domanda "Lightning Network è privato?" in due fasi generali.

In primo luogo, un ricercatore sulla privacy definirebbe un *modello di sicurezza* che specifica ciò di cui un avversario è capace e mira a raggiungere. Quindi, descriverebbe le proprietà rilevanti del sistema e verificherebbe se è conforme ai requisiti.

Processo per Valutare la Privacy

Un modello di sicurezza si basa su una serie di *presupposti di sicurezza* sottostanti. Nei sistemi crittografici, queste ipotesi sono spesso incentrate sulle proprietà matematiche delle primitive crittografiche, come cifrari, firme e funzioni hash. I presupposti di sicurezza di Lightning Network sono che le firme ECDSA, la funzione hash SHA-256 e altre funzioni crittografiche utilizzate nel protocollo si comportino rispettando le loro definizioni di sicurezza. Ad esempio, assumiamo che sia praticamente impossibile trovare una preimmagine (e una seconda preimmagine) di una funzione hash. Ciò consente a Lightning Network di fare affidamento sul meccanismo HTLC (che utilizza la preimmagine di una funzione hash) per l'atomicità dei pagamenti multihop: nessuno, tranne il destinatario finale, può rivelare il segreto di pagamento e risolvere l'HTLC. Assumiamo anche un grado di connettività nella rete, vale a dire che i canali Lightning formino un grafo connesso. Pertanto, è possibile trovare un percorso da qualsiasi mittente a qualsiasi destinatario. Infine, supponiamo che i messaggi di rete vengano propagati entro determinati timeout.

Ora che abbiamo identificato alcuni dei nostri presupposti sottostanti, consideriamo alcuni possibili avversari.

Un nodo di routing "onesto ma curioso" può osservare gli importi dei pagamenti, i nodi immediatamente precedenti e successivi e il grafo dei canali annunciati con le loro capacità. Un nodo molto ben connesso può fare lo stesso, ma in misura maggiore. Ad esempio, considera gli sviluppatori di un wallet popolare che mantengono un nodo cui gli utenti si connettono per impostazione predefinita. Questo nodo sarebbe responsabile dell'instradamento di un'ampia quota di pagamenti da e verso gli utenti di quel wallet. Cosa succede se più nodi sono sotto controllo di un avversario malevolo? Se due nodi in collusione si trovano sullo stesso percorso di pagamento, capirebbero che stanno inoltrando HTLC appartenenti allo stesso pagamento perché gli HTLC hanno lo stesso hash di pagamento.

NOTA	I pagamenti in più parti (multipart) consentono agli utenti di offuscare i loro importi di pagamento date le loro dimensioni divise non uniformi.

Quali possono essere gli obiettivi di un attaccante Lightning? La sicurezza delle informazioni è spesso descritta in termini di tre proprietà principali: riservatezza, integrità e disponibilità.

Riservatezza

Le informazioni arrivano solo ai destinatari previsti.

Integrità

Le informazioni non vengono alterate durante il trasporto.

Disponibilità

Il sistema funziona per la maggior parte del tempo.

Le proprietà importanti di LN sono principalmente incentrate sulla riservatezza e sulla disponibilità. Alcune delle proprietà più importanti da proteggere includono:

- Solo il mittente e il destinatario conoscono l'importo del pagamento.
- Nessuno può collegare mittenti e destinatari.
- A un utente onesto non può essere impedito di inviare e ricevere pagamenti.

Per ogni obiettivo di privacy e modello di sicurezza, esiste una certa probabilità che un utente malintenzionato abbia successo. Questa probabilità dipende da vari fattori, come la dimensione e la struttura della rete. A parità di altre condizioni, è generalmente più facile attaccare con successo una piccola rete piuttosto che una grande. Allo stesso modo, più centralizzata è la rete, più capace può essere un utente malintenzionato se i nodi "centrali" sono sotto il suo controllo. Naturalmente, il termine centralizzazione deve essere definito proprio per costruire attorno ad esso modelli di sicurezza, e ci sono molte possibili definizioni di quanto sia centralizzata una rete. Infine, come rete di pagamento, Lightning Network dipende dagli stimoli economici. La dimensione e la struttura delle commissioni influiscono sull'algoritmo di instradamento e quindi possono aiutare l'attaccante inoltrando la maggior parte dei pagamenti attraverso i propri nodi o impedire che ciò accada.

Anonimity Set

Cosa significa de-anonimizzare qualcuno? In termini semplici, la de-anonimizzazione implica

il collegamento di un'azione con l'identità reale di una persona, come il nome o l'indirizzo fisico. Nella ricerca sulla privacy, la nozione di de-anonimizzazione è più sfumata. Innanzitutto, non stiamo necessariamente parlando di nomi e indirizzi. Anche la scoperta dell'indirizzo IP o del numero di telefono di qualcuno può essere considerata de-anonimizzazione. Un'informazione che consente di collegare l'azione di un utente alle sue azioni precedenti viene definita *identità*. In secondo luogo, la de-anonimizzazione non è binaria; un utente non è né completamente anonimo né completamente de-anonimizzato. Invece, la ricerca sulla privacy esamina l'anonimato rispetto al set di anonimato.

Il *set di anonimato* (*anonimity set*) è un concetto centrale della privacy. Si riferisce all'insieme di identità tali che, dal punto di vista di un attaccante, una data azione potrebbe corrispondere a chiunque nell'insieme. Considera un esempio di vita reale. Immagina di incontrare una persona per strada... Qual è il suo anonimato secondo il tuo punto di vista? Se non lo/la conosci personalmente e senza ulteriori informazioni, il suo anonimato è più o meno uguale alla popolazione della città, compresi i viaggiatori temporanei come i turisti. Se consideri inoltre il suo aspetto, potresti essere in grado di stimare approssimativamente la sua età ed escludere i residenti della città che sono ovviamente più anziani o più giovani della persona in questione dal set di anonimato. Inoltre, se noti che la persona entra nell'ufficio dell'azienda X utilizzando un badge elettronico, l'anonimato si riduce al numero di dipendenti e visitatori dell'azienda X. Infine, potresti notare il numero di targa dell'auto con cui è arrivato/a sul posto. Se sei un osservatore casuale, questo non ti dà molto. Tuttavia, se sei un funzionario comunale e hai accesso al database che abbina i numeri di targa ai nomi, puoi restringere l'anonimato impostato a poche persone: il proprietario/a dell'auto e gli eventuali amici e parenti stretti che potrebbero aver preso in prestito l'auto .

Questo esempio illustra alcuni punti importanti. In primo luogo, ogni piccola informazione può portare l'avversario più vicino al suo obiettivo. Potrebbe non essere necessario arrivare ad un set di anonimato uguale a 1. Ad esempio, se l'avversario pianifica un attacco Denial-of-Service (DoS) mirato e può abbattere 100 server, il set di anonimato di 100 è sufficiente. In secondo luogo, l'avversario può correlare in modo incrociato le informazioni provenienti da fonti diverse. Anche se una violazione della privacy sembra relativamente benigna, non sappiamo mai cosa può ottenere in combinazione con altre fonti di dati. Infine, soprattutto nella crittografia, l'attaccante ha sempre "l'ultima risorsa" di una ricerca a forza bruta (bruteforce). Le primitive crittografiche sono progettate in modo che sia praticamente impossibile indovinare un segreto come una chiave privata. Tuttavia, ogni bit di informazione avvicina l'avversario a questo obiettivo e, a un certo punto, diventa raggiungibile.

Su Lightning Network, de-anonimizzazione significa generalmente derivare una corrispondenza tra i pagamenti e gli utenti identificati dagli ID dei nodi. Ad ogni pagamento può essere assegnato un set di anonimato del mittente e un set di anonimato del destinatario. Idealmente, il set di anonimato è costituito da tutti gli utenti della rete. Questo assicura che l'attaccante non abbia alcuna informazione. Tuttavia, la rete reale perde informazioni che consentono a un utente malintenzionato di restringere la ricerca. Più piccolo è il set di anonimato, maggiore è la possibilità di successo di una de-anonimizzazione.

Differenze tra Lightning Network e Bitcoin in Termini di Privacy

Sebbene sia vero che le transazioni sulla rete Bitcoin non associano le identità del mondo reale agli indirizzi Bitcoin, tutte le transazioni vengono trasmesse in chiaro e possono essere analizzate. Sono state create diverse società per de-anonimizzare gli utenti di Bitcoin e altre criptovalute.

A prima vista, Lightning offre una privacy migliore rispetto a Bitcoin perché i pagamenti Lightning non vengono trasmessi all'intera rete. Sebbene ciò migliori la linea di base della privacy, altre proprietà del protocollo Lightning possono rendere più impegnativi i pagamenti anonimi. Ad esempio, i pagamenti più grandi possono avere meno opzioni di instradamento. Ciò può consentire a un avversario che controlla i nodi ben capitalizzati di instradare i pagamenti più grandi e scoprire gli importi dei pagamenti e probabilmente altri dettagli. Nel tempo, con la crescita di Lightning Network, questo potrebbe diventare un problema minore.

Un'altra differenza rilevante tra Lightning e Bitcoin è che i nodi Lightning mantengono un'identità permanente, mentre i nodi Bitcoin no. Un utente avanzato di Bitcoin può facilmente cambiare i nodi utilizzati per ricevere dati sulla blockchain e trasmettere transazioni. Un utente Lightning, al contrario, invia e riceve pagamenti attraverso i nodi che ha utilizzato per aprire i propri canali di pagamento. Inoltre, il protocollo Lightning presuppone che i nodi di instradamento annuncino il proprio indirizzo IP oltre al proprio ID. Ciò crea un collegamento permanente tra gli ID dei nodi e gli indirizzi IP, che può essere pericoloso, considerando che un indirizzo IP è spesso un passaggio intermedio negli attacchi di anonimato legati alla posizione fisica dell'utente e, nella maggior parte dei casi, all'identità del mondo reale. È possibile utilizzare Lightning con Tor, ma molti nodi non utilizzano questa funzionalità, come puoi vedere dalle statistiche raccolte dagli annunci dei nodi.

Un utente Lightning, quando invia un pagamento, ha i suoi vicini nel suo set di anonimato.

Nello specifico, un nodo di instradamento conosce solo i nodi immediatamente precedenti e successivi. Il nodo di instradamento non sa se i suoi vicini immediati nel percorso di pagamento sono il mittente o il destinatario finale. Pertanto, l'insieme di anonimato di un nodo in Lightning è approssimativamente uguale ai suoi vicini (vedi Figura 16-1).

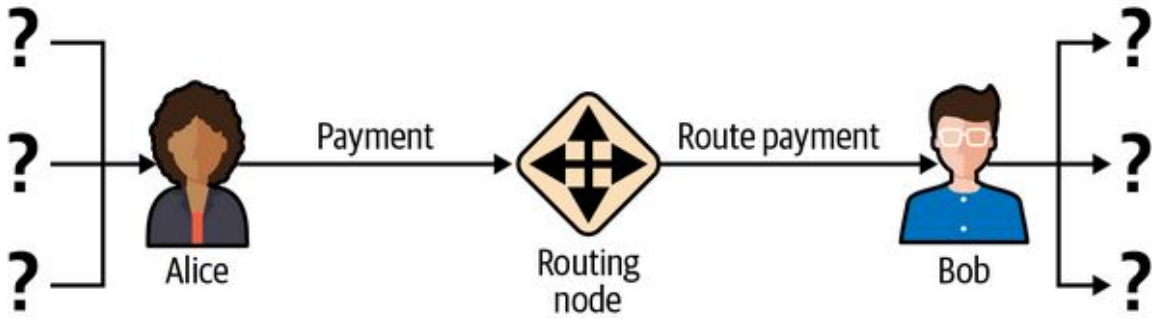

Figura 16-1. L'insieme dell'anonimato di Alice e Bob costituisce i loro vicini

Una logica simile si applica ai destinatari dei pagamenti. Molti utenti aprono solo una manciata di canali di pagamento, limitando quindi i loro set di anonimato. Inoltre, su LN, il set di anonimato è statico o cambia lentamente. Al contrario, è possibile ottenere set di anonimato molto più ampi nelle transazioni CoinJoin on-chain. Le transazioni CoinJoin con set di anonimato superiori a 50 sono piuttosto frequenti. In genere, i set di anonimato in una transazione CoinJoin corrispondono a un insieme di utenti che cambiano dinamicamente.

Infine, agli utenti Lightning può anche essere negato il servizio, bloccando i loro canali o esaurendoli da parte di un utente malevolo. L'inoltro dei pagamenti richiede che il capitale (che è una risorsa scarsa) sia temporaneamente bloccato negli HTLC lungo il percorso. Un utente malevolo può inviare molti pagamenti senza finalizzarli, occupando il capitale degli utenti onesti per lunghi periodi. Questo vettore di attacco non è presente (o almeno non così ovvio) in Bitcoin.

In sintesi, mentre alcuni aspetti dell'architettura di Lightning Network suggeriscono che si tratti di un passo avanti in termini di privacy rispetto a Bitcoin, altre proprietà del protocollo potrebbero facilitare gli attacchi alla privacy. È necessaria una ricerca approfondita per valutare quali garanzie di privacy fornisce Lightning Network e migliorare lo stato delle cose.

Quest'area di ricerca e sviluppo sta crescendo rapidamente ed esistono molti team di ricerca che attualmente lavorano sulla privacy di Lightning e su come migliorarla.

Vediamo ora alcuni degli attacchi alla privacy di LN che sono stati descritti accademicamente.

Attacchi su Lightning Network

Ricerche recenti descrivono vari modi in cui la sicurezza e la privacy di Lightning Network possono essere compromesse.

Osservazione degli Importi dei Pagamenti

Uno degli obiettivi di un sistema di pagamento che tutela la privacy è nascondere l'importo del pagamento alle parti non coinvolte. La rete Lightning è un miglioramento rispetto al layer 1 (Bitcoin) a questo proposito. Mentre le transazioni Bitcoin vengono trasmesse in chiaro e possono essere osservate da chiunque, i pagamenti Lightning viaggiano solo attraverso pochi nodi lungo il percorso di pagamento. Tuttavia, i nodi intermediari vedono l'importo del pagamento, sebbene questo importo potrebbe non corrispondere all'importo totale effettivo del pagamento (ricorda gli MPP – Multipart Payments). Ciò è necessario per creare un nuovo HTLC ad ogni hop. La disponibilità di importi di pagamento ai nodi intermedi non rappresenta una minaccia immediata. Tuttavia, un nodo intermediario *onesto ma curioso* potrebbe usarlo come parte di un attacco più ampio.

Collegamento di Mittenti e Destinatari

Un utente malintenzionato potrebbe essere interessato a conoscere il mittente e/o il destinatario di un pagamento per rivelare determinate relazioni economiche. Questa violazione della privacy potrebbe danneggiare la resistenza alla censura, in quanto un nodo intermediario potrebbe censurare i pagamenti da o verso determinati destinatari o mittenti. Idealmente, collegare i mittenti ai destinatari non dovrebbe essere possibile a nessuno diverso dal mittente e dal destinatario.

Nelle sezioni seguenti, prenderemo in considerazione due tipi di avversari: l'avversario fuori percorso e l'avversario sul percorso. Un avversario fuori percorso tenta di valutare il mittente e il destinatario di un pagamento senza partecipare al processo di instradamento del pagamento. Un avversario sul percorso può sfruttare tutte le informazioni che potrebbe ottenere indirizzando il pagamento degli interessi.

Innanzitutto, consideriamo l'*avversario fuori percorso*. Nella prima fase di questo scenario di attacco, un potente avversario fuori percorso deduce i singoli saldi in ciascun canale di pagamento tramite sondaggio/probing (che descriveremo a breve) e forma un'istantanea della rete al tempo t_1. Per semplicità, rendiamo t_1 uguale a 12:05. Poi sonda nuovamente la rete in un momento successivo all'ora t_2 alle 12:10. L'attaccante confronterebbe quindi le istantanee alle 12:10 e alle 12:05 e utilizzerebbe le differenze tra le due istantanee per dedurre informazioni sui pagamenti avvenuti osservando i percorsi che sono cambiati. Nel caso più semplice, se si verificasse un solo pagamento tra le 12:10 e le 12:05, l'avversario osserverebbe un unico percorso in cui i saldi sono cambiati degli stessi importi. Pertanto, l'avversario apprende quasi tutto su questo pagamento: il mittente, il destinatario e l'importo. Se più percorsi di pagamento si sovrappongono, l'avversario deve applicare l'euristica per identificare tale sovrapposizione e separare i pagamenti.

Ora rivolgiamo la nostra attenzione ad un *avversario sul percorso*. Un tale avversario potrebbe sembrare innocuo. Tuttavia, nel giugno 2020, i ricercatori hanno notato che il singolo nodo più connesso e centrale ha avuto modo di osservare quasi il 50% di tutti i pagamenti LN, mentre i quattro nodi più centrali hanno osservato una media del 72% dei pagamenti. Questi risultati sottolineano la rilevanza del modello dell'attaccante sul percorso. Anche se gli intermediari in un percorso di pagamento apprendono solo il loro successore e predecessore, ci sono diverse fughe di notizie che un intermediario malintenzionato o onesto ma curioso potrebbe utilizzare per dedurre il mittente e il destinatario.

L'avversario sul percorso può osservare l'importo di qualsiasi pagamento instradato così come i delta del timelock (vedi il Capitolo 10). Pertanto, l'avversario può escludere qualsiasi nodo dal set di anonimato del mittente o del destinatario impostato con capacità inferiori alla quantità instradata. Pertanto, osserviamo un compromesso tra privacy e importi di pagamento. In genere, maggiore è l'importo del pagamento, minore è il set di anonimato. Notiamo che questa perdita potrebbe essere ridotta al minimo con pagamenti in più parti o con canali di pagamento di grande capacità. Allo stesso modo, i canali di pagamento con piccoli delta timelock potrebbero essere esclusi da un percorso di pagamento. Più precisamente, un canale di pagamento non può riguardare un pagamento se il tempo rimanente per il quale il pagamento potrebbe essere bloccato è maggiore di quello che il nodo di inoltro sarebbe disposto ad accettare. Questa perdita potrebbe essere contrastata aderendo ai cosiddetti percorsi ombra (shadow routes).

Una delle fughe di notizie più sottili ma potenti che un avversario sul percorso può favorire è l'analisi dei tempi. Un avversario sul percorso può tenere un registro per ogni pagamento

instradato, insieme al tempo impiegato da un nodo per rispondere a una richiesta HTLC. Prima di iniziare l'attacco, l'attaccante apprende le caratteristiche di latenza di ogni nodo di Lightning Network inviando loro richieste. Naturalmente, questo può aiutare a stabilire la posizione precisa dell'avversario nel percorso di pagamento. Ancora peggio, come è stato mostrato, un utente malintenzionato può determinare con successo il mittente e il destinatario di un pagamento da un insieme di possibili mittenti e destinatari utilizzando stime basate sul tempo.

Infine, è importante riconoscere che probabilmente esistono perdite sconosciute o non studiate che potrebbero aiutare i tentativi di de-anonimizzazione. Ad esempio, poiché diversi portafogli Lightning applicano algoritmi di instradamento diversi, anche conoscere l'algoritmo di instradamento applicato potrebbe aiutare a escludere determinati nodi dall'essere mittente e/o destinatario di un pagamento.

Rivelare i Saldi dei Canali (Probing)

I saldi dei canali Lightning dovrebbero essere nascosti per motivi di privacy ed efficienza. Un nodo Lightning conosce solo i saldi dei suoi canali adiacenti. Il protocollo non fornisce un modo standard per interrogare il saldo di un canale remoto.

Tuttavia, un utente malintenzionato può rivelare il saldo di un canale remoto in un *attacco di probing* (sondaggio). Nella sicurezza delle informazioni, probing si riferisce alla tecnica di inviare richieste a un sistema mirato e trarre conclusioni sul suo stato privato in base alle risposte ricevute.

I canali Lightning sono inclini a sondare. Ricorda che un pagamento Lightning standard inizia con il destinatario che crea un segreto di pagamento casuale e invia il suo hash al mittente. Nota che per i nodi intermedi, tutti gli hash sembrano casuali. Non c'è modo di sapere se un hash corrisponde a un vero segreto o se è stato generato in modo casuale.

L'attacco di probing procede come segue. Supponiamo che l'attaccante Mallory voglia rivelare il saldo di Alice nel canale pubblico tra Alice e Bob. Supponiamo che la capacità totale di quel canale sia di 1 milione di sats. Il saldo di Alice potrebbe essere qualsiasi numero, da zero a 1 milione di satoshi (per essere precisi, la stima è un po' più ristretta a causa della riserva del canale, che non consideriamo per semplicità). Mallory apre un canale con Alice con 1 milione di sats e invia 500.000 sats a Bob tramite Alice utilizzando un *numero casuale* come hash di pagamento. Naturalmente, questo numero non corrisponde a nessun segreto

di pagamento noto. Pertanto, il pagamento fallirà. La domanda è: esattamente, come fallirà?

Ci sono due scenari. Se Alice ha più di 500.000 satoshi dalla sua parte del canale con Bob, inoltra il pagamento. Bob decifra la cipolla di pagamento e si rende conto che il pagamento è destinato a lui. Cerca nel suo archivio locale di segreti di pagamento e cerca la preimmagine che corrisponde all'hash di pagamento, ma non ne trova una. Seguendo il protocollo, Bob restituisce l'errore "hash di pagamento sconosciuto" ad Alice, che lo inoltra a Mallory. Di conseguenza, Mallory sa che il pagamento *sarebbe potuto andare a buon fine* se l'hash del pagamento fosse reale. Pertanto, Mallory può aggiornare la sua stima del saldo di Alice da "tra zero e 1 milione" a "tra 500.000 e 1 milione". Un altro scenario si verifica se il saldo di Alice è inferiore a 500.000 satoshi. In tal caso, Alice non è in grado di inoltrare il pagamento e restituisce a Mallory l'errore di "saldo insufficiente". Mallory aggiorna la sua stima da "tra zero e 1 milione" a "tra zero e 500.000".

Nota che in ogni caso, la stima di Mallory diventa due volte più precisa dopo un solo sondaggio! Può continuare a sondare, scegliendo l'importo di sondaggio successivo in modo tale da dividere a metà l'intervallo di stima corrente. Questa ben nota tecnica di ricerca si chiama ricerca binaria. Con la *ricerca binaria*, il numero di sondaggi è *logaritmico* nella precisione desiderata. Ad esempio, per ottenere il saldo di Alice in un canale di 1 milione di satoshi fino a un singolo satoshi, Mallory dovrebbe eseguire solo $\log_2$ (1.000.000) ≈ 20 sondaggi. Se un sondaggio richiede 3 secondi, un canale può essere sondato con precisione in circa un minuto!

Il probing può essere reso ancora più efficiente. Nella sua variante più semplice, Mallory si collega direttamente al canale che vuole sondare. È possibile farlo senza aprire un canale a uno dei nodi coinvolti? Immagina che Mallory voglia sondare un canale tra Bob e Charlie ma non voglia aprire un canale, il che richiede il pagamento di commissioni on-chain e l'attesa di conferme delle transazioni di finanziamento. Invece, Mallory riutilizza il suo canale esistente con Alice e invia una sonda lungo il percorso Mallory → Alice → Bob → Charlie. Mallory può interpretare l'errore "hash di pagamento sconosciuto" allo stesso modo di prima: la sonda ha raggiunto la destinazione; pertanto, tutti i canali lungo il percorso dispongono di saldi sufficienti per inoltrarlo. Ma cosa succede se Mallory riceve l'errore "saldo insufficiente"? Significa che l'equilibrio è insufficiente tra Alice e Bob o tra Bob e Charlie?

Nell'attuale protocollo Lightning, i messaggi di errore segnalano non solo *quale* errore si è verificato, ma anche *dove* si è verificato. Quindi, con una gestione degli errori più precisa, Mallory ora sa quale canale ha fallito. Se questo è il canale target, aggiorna le sue stime; in

caso contrario, sceglie un altro percorso verso il canale di destinazione. Ottiene anche informazioni *aggiuntive* sui saldi dei canali intermediari, oltre a quelli del canale target.

L'attacco di probing può essere ulteriormente utilizzato per collegare mittenti e destinatari, come descritto nella sezione precedente.

A questo punto, potresti chiederti: perché Lightning Network fa un lavoro così scarso nel proteggere i dati privati dei suoi utenti? Non sarebbe meglio non rivelare al mittente perché e dove il pagamento è fallito? In effetti, questa potrebbe essere una potenziale contromisura, ma presenta notevoli inconvenienti. Lightning deve trovare un attento equilibrio tra privacy ed efficienza. Ricorda che i nodi regolari non conoscono le distribuzioni dei saldi nei canali remoti. Pertanto, i pagamenti possono (e spesso accade) fallire a causa di un saldo insufficiente presso un hop intermedio. I messaggi di errore consentono al mittente di escludere il canale in errore dalla considerazione durante la costruzione di un altro percorso. Diversi portafogli Lightning eseguono persino sondaggi interni per verificare se un percorso costruito può davvero gestire un pagamento.

Esistono altre potenziali contromisure contro i sondaggio dei canali. Innanzitutto, è difficile per un utente malintenzionato prendere di mira i canali non annunciati. In secondo luogo, i nodi che implementano il just-in-time routing (JIT) potrebbero essere meno soggetti all'attacco. Infine, poiché i pagamenti in più parti rendono meno grave il problema della capacità insufficiente, gli sviluppatori del protocollo possono prendere in considerazione l'idea di nascondere alcuni dettagli dell'errore senza danneggiarne l'efficienza.

Denial of Service - Negazione del Servizio

Quando le risorse vengono rese disponibili al pubblico, esiste il rischio che gli attaccanti tentino di rendere indisponibili tali risorse eseguendo un attacco Denial of Service (DoS). In genere, ciò viene ottenuto dall'attaccante bombardando una risorsa richieste malevole, che sono indistinguibili dalle richieste legittime. Gli attacchi raramente comportano perdite finanziarie per il bersaglio, a parte la riduzione del costo opportunità del loro servizio, e hanno semplicemente lo scopo di mandare offline il bersaglio.

Le mitigazioni tipiche per gli attacchi DoS richiedono l'autenticazione per le richieste per separare gli utenti legittimi da quelli malintenzionati. Queste mitigazioni comportano un costo irrisorio per gli utenti regolari, ma fungono da deterrente per un utente malevolo che lancia richieste su larga scala. Le misure anti-denial-of-service possono essere viste ovunque

su Internet: i siti Web applicano limiti di velocità per garantire che nessun utente possa consumare tutta la banda del proprio server, i siti di recensioni di film richiedono l'autenticazione per verificarne l'autenticità e alcuni servizi vendono chiavi API per limitare il numero di richieste.

DoS in Bitcoin

In Bitcoin, la larghezza di banda utilizzata dai nodi per inoltrare le transazioni e lo spazio di cui dispongono nella rete sotto forma di mempool sono risorse disponibili pubblicamente. Qualsiasi nodo sulla rete può consumare larghezza di banda e spazio mempool inviando una transazione valida. Se questa transazione viene minata in un blocco valido, pagheranno le commissioni, il che aggiunge un costo all'utilizzo di queste risorse di rete condivise.

In passato, la rete Bitcoin ha affrontato un tentativo di attacco DoS in cui gli aggressori hanno inviato spam alla rete con transazioni a basso costo. Molte di queste transazioni non sono state selezionate dai miner a causa delle loro basse commissioni, quindi gli aggressori potrebbero consumare risorse di rete senza pagare. Per risolvere questo problema, è stata impostata una tariffa minima di inoltro delle transazioni che imposta una commissione di soglia richiesta dai nodi per propagare le transazioni. Questa misura ha ampiamente garantito che le transazioni che consumano risorse di rete alla fine pagheranno le loro commissioni on-chain. La tariffa minima per l'inoltro è accettabile per gli utenti regolari, ma danneggerebbe finanziariamente gli aggressori se tentassero di inviare spam alla rete. Anche se alcune transazioni potrebbero non entrare in blocchi validi durante periodi di commissioni alte, queste misure sono state ampiamente efficaci nel dissuadere questo tipo di spam.

DoS in Lightning

Analogamente a Bitcoin, LN addebita commissioni per l'utilizzo delle sue risorse pubbliche, ma in questo caso le risorse sono canali pubblici e le commissioni si presentano sotto forma di commissioni di instradamento. La capacità di instradare i pagamenti attraverso i nodi in cambio di commissioni fornisce alla rete un grande vantaggio di scalabilità (i nodi che non sono direttamente connessi possono comunque effettuare transazioni), ma ha il costo di esporre una risorsa pubblica che deve essere protetta dagli attacchi DoS.

Quando un nodo Lightning inoltra un pagamento per tuo conto, utilizza i dati e la larghezza di banda del pagamento per aggiornare la sua transazione di impegno e l'importo del

pagamento viene riservato nel saldo del canale fino a quando non viene regolato o non va a buon fine. Nei pagamenti andati a buon fine, questo è accettabile perché al nodo vengono infine pagate le sue commissioni. I pagamenti non riusciti non comportano commissioni nel protocollo attuale. Ciò consente ai nodi di instradare senza costi i pagamenti falliti attraverso qualsiasi canale. Questo è ottimo per gli utenti legittimi, che non vorrebbero pagare per i tentativi falliti, ma consente anche agli aggressori di consumare a costo zero le risorse dei nodi, proprio come le transazioni a basso costo su Bitcoin che non finiscono mai per pagare le commissioni dei miner.

È in corso una discussione sulla mailing list lightning-dev su come affrontare tale problema.

Attacchi DoS noti

Esistono due attacchi DoS noti sui canali LN pubblici che rendono inutilizzabile un canale o un insieme di canali di destinazione. Entrambi gli attacchi comportano l'instradamento dei pagamenti attraverso un canale pubblico, quindi il loro mantenimento fino al loro timeout, massimizzando così la durata dell'attacco. Il requisito di fallire i pagamenti per non pagare le commissioni è abbastanza semplice da soddisfare perché i nodi malevoli possono semplicemente reindirizzare i pagamenti a se stessi. In assenza di commissioni per i pagamenti non riusciti, l'unico costo per l'attaccante è il costo on-chain per l'apertura di un canale per l'invio di questi pagamenti, che può essere irrisorio in periodi di basse fee.

Commitment Jamming – Disturbo del Canale

I nodi Lightning aggiornano il loro stato condiviso utilizzando transazioni di impegno asimmetriche, su cui vengono aggiunti e rimossi HTLC per facilitare i pagamenti. Ogni parte è limitata a un totale di 483 HTLC alla volta nella transazione di impegno. Un attacco di channel jamming consente a un utente malevolo di rendere inutilizzabile un canale instradando 483 pagamenti attraverso il canale di destinazione e trattenendoli fino al timeout.

Va notato che questo limite è stato scelto nella specifica per garantire che tutti gli HTLC possano essere rimossi in un'unica transazione di penalità. Sebbene questo limite *possa* essere aumentato, le transazioni sono ancora limitate dalla dimensione del blocco, quindi è probabile che il numero di slot disponibili rimanga limitato.

Blocco della Liquidità del Canale

Un attacco di blocco della liquidità del canale è paragonabile a un attacco di channel jamming in quanto instrada i pagamenti attraverso un canale e li trattiene in modo che tale sia inutilizzabile. Invece di bloccare gli slot sull'impegno del canale, questo attacco indirizza HTLC di grandi dimensioni attraverso un canale target, consumando tutta la larghezza di banda disponibile del canale. L'impegno di capitale di questo attacco è superiore all'attacco di commitment jamming perché il nodo attaccante ha bisogno di più fondi per instradare i pagamenti falliti attraverso il target.

De-Anonimizzazione Cross-Layer

Le reti di computer sono spesso stratificate. La stratificazione consente la separazione delle preoccupazioni e rende gestibile l'intero sistema. Nessuno potrebbe progettare un sito Web se richiedesse la comprensione di tutto lo stack TCP/IP fino alla codifica fisica dei bit in un cavo ottico. Ogni livello dovrebbe fornire la funzionalità al livello superiore in modo pulito. Idealmente, lo strato superiore dovrebbe percepire uno strato inferiore come una scatola nera. In realtà, però, le implementazioni non sono ideali e i dettagli hanno una *fuga* nello strato superiore. Questo è il problema delle "leaky abstractions".

Nel contesto di Lightning, il protocollo LN si basa sul protocollo Bitcoin e sulla rete LN P2P. Fino a questo punto, abbiamo considerato isolatamente le garanzie di privacy offerte da Lightning Network. Tuttavia, la creazione e la chiusura dei canali di pagamento sono intrinsecamente eseguite sulla blockchain di Bitcoin. Di conseguenza, per un'analisi completa delle disposizioni sulla privacy di Lightning Network, è necessario considerare ogni livello dello stack tecnologico con cui gli utenti potrebbero interagire. Nello specifico, un avversario che vuole de-anonimizzare qualcuno può e utilizzerà dati off-chain e on-chain per raggruppare o collegare i nodi LN ai corrispondenti indirizzi Bitcoin.

Gli aggressori che tentano di de-anonimizzare gli utenti LN possono avere vari obiettivi, in un contesto a più livelli:

- Cluster di indirizzi Bitcoin di proprietà dello stesso utente (Layer 1). Chiamiamo tali "entità Bitcoin".
- Cluster di nodi LN di proprietà dello stesso utente (Layer 2).
- Collegamento univoco di nodi LN agli insiemi di entità Bitcoin che li controllano.

Esistono diverse euristiche e modelli di utilizzo che consentono a un avversario di raggruppare indirizzi Bitcoin e nodi LN di proprietà degli stessi utenti LN. Inoltre, questi cluster possono essere collegati tra più livelli utilizzando altre potenti euristiche di collegamento multi livello. L'ultimo tipo di euristica, le tecniche di collegamento incrociato, sottolinea la necessità di una visione olistica della privacy. Nello specifico, dobbiamo considerare la privacy nel contesto di entrambi i livelli in contemporanea.

Clustering di Entità Bitcoin On-Chain

Le interazioni blockchain di Lightning Network si riflettono in modo permanente nel grafico dell'entità Bitcoin. Anche se un canale è chiuso, un utente malintenzionato può osservare quale indirizzo ha finanziato il canale e dove vengono spese le monete dopo averlo chiuso. Per questa analisi, consideriamo quattro entità separate. L'apertura di un canale provoca un flusso monetario da un'*entità fonte* a un'*entità finanziatrice*; la chiusura di un canale provoca un flusso da un'*entità di regolamento* a un'*entità di destinazione*.

All'inizio del 2021, Romiti et al. Hanno identificato quattro euristiche che consentono il raggruppamento di queste entità. Due di loro trattano fughe di finanziamento (funding) e due descrivono fughe di regolamento (settlement).

Euristica della stella (finanziamento)

> Se un componente contiene un'entità di origine che inoltra i fondi a una o più entità di finanziamento, queste entità di finanziamento sono probabilmente controllate dallo stesso utente.

Euristica del serpente (finanziamento)

> Se un componente contiene un'entità di origine che inoltra fondi a una o più entità, che a loro volta sono utilizzate come entità di origine e di finanziamento, è probabile che tutte queste entità siano controllate dallo stesso utente.

Euristica del collezionista (settlement)

> Se un componente contiene un'entità di destinazione che riceve fondi da una o più

entità di regolamento, queste entità di regolamento sono probabilmente controllate dallo stesso utente.

Euristica proxy (settlement)

Se un componente contiene un'entità di destinazione che riceve fondi da una o più entità, che a loro volta sono utilizzate come entità di regolamento e di destinazione, è probabile che tali entità siano controllate dallo stesso utente.

Vale la pena sottolineare che queste euristiche potrebbero produrre falsi positivi. Ad esempio, se le transazioni di più utenti non correlati vengono combinate in una transazione CoinJoin, l'euristica stella o proxy può produrre falsi positivi. Ciò potrebbe accadere se gli utenti finanziano un canale di pagamento da una transazione CoinJoin. Un'altra potenziale fonte di falsi positivi potrebbe essere che un'entità potrebbe rappresentare diversi utenti se gli indirizzi in cluster sono controllati da un servizio (ad esempio un exchange) o per conto dei propri utenti (custodial wallet). Questi falsi positivi possono essere efficacemente filtrati.

Contromisure

Se gli output delle transazioni di finanziamento non vengono riutilizzati per l'apertura di altri canali, l'euristica del serpente non funziona. Se gli utenti si astengono dai canali di finanziamento da un'unica fonte esterna ed evitano di raccogliere fondi in un'unica entità di destinazione esterna, le altre euristiche non darebbero risultati significativi.

Clustering di Nodi Lightning Off-Chain

I nodi LN pubblicizzano alias, ad esempio *LNBig.com*. Gli alias possono migliorare l'usabilità del sistema. Tuttavia, gli utenti tendono a utilizzare alias simili per i propri nodi diversi. Ad esempio, *LNBig.com Billing* è probabilmente di proprietà dello stesso utente del nodo con alias *LNBig.com*. Alla luce di questa osservazione, è possibile raggruppare i nodi LN applicando i relativi alias di nodo. Nello specifico, si raggruppano i nodi LN in un singolo cluster se i relativi alias sono simili rispetto a una metrica di somiglianza di stringa.

Un altro metodo per raggruppare i nodi LN consiste nel valutare i loro indirizzi IP o Tor. Se gli

stessi indirizzi IP o Tor corrispondono a diversi nodi LN, questi nodi sono probabilmente controllati dallo stesso utente.

Contromisure

Per una maggiore privacy, gli alias dovrebbero essere sufficientemente diversi l'uno dall'altro. Sebbene l'annuncio pubblico degli indirizzi IP possa essere inevitabile per quei nodi che desiderano avere canali in entrata nella rete Lightning, la collegabilità tra i nodi dello stesso utente può essere mitigata se i client per ciascun nodo sono ospitati con diversi fornitori di servizi e quindi indirizzi IP .

Collegamento Cross-Layer: Nodi Lightning ed Entità Bitcoin

L'associazione di nodi LN a entità Bitcoin è una grave violazione della privacy, aggravata dal fatto che la maggior parte dei nodi LN espone pubblicamente i propri indirizzi IP. In genere, un indirizzo IP può essere considerato un identificatore univoco di un utente. Due modelli di comportamento ampiamente osservati rivelano collegamenti tra i nodi LN e le entità Bitcoin:

Riutilizzo dei fondi

> Ogni volta che gli utenti chiudono i canali di pagamento, recuperano i fondi corrispondenti. Tuttavia, molti utenti riutilizzano tali fondi per aprire un nuovo canale. Tali monete fondi essere efficacemente collegati ad un nodo LN comune.

Riutilizzo dell'entità

> In genere, gli utenti finanziano i propri canali di pagamento da indirizzi Bitcoin corrispondenti alla stessa entità Bitcoin.

Questi algoritmi di collegamento cross-layer potrebbero essere sventati se gli utenti possiedono più indirizzi non in cluster o utilizzano più portafogli per interagire con LN.

La possibile de-anonimizzazione delle entità Bitcoin illustra quanto sia importante considerare la privacy di entrambi i livelli contemporaneamente anziché uno alla volta.

Grafo di Lightning Network

Lightning Network, come suggerisce il nome, è una rete peer-to-peer di canali di pagamento. Pertanto, molte delle sue proprietà (privacy, robustezza, connettività, efficienza di routing) sono influenzate e caratterizzate dalla sua natura di rete.

In questa sezione, discutiamo e analizziamo Lightning Network dal punto di vista della scienza delle reti. Siamo particolarmente interessati a comprendere il grafo dei canali LN, la sua robustezza, connettività e altre caratteristiche importanti.

Come Appare il Grafo di Lightning?

Ci si sarebbe potuti aspettare che Lightning Network fosse un grafo casuale, in cui i bordi sono formati casualmente tra i nodi. In tal caso, la distribuzione dei gradi di LN seguirebbe una distribuzione normale gaussiana. In particolare, la maggior parte dei nodi avrebbe approssimativamente lo stesso grado e non ci aspetteremmo nodi con gradi straordinariamente alti. Questo perché la distribuzione normale diminuisce esponenzialmente per i valori al di fuori dell'intervallo intorno al valore medio. La rappresentazione di un grafo casuale sembra una topologia di rete mesh. Sembra decentralizzato e non gerarchico: ogni nodo sembra avere uguale importanza. Inoltre, i grafi casuali hanno un grande diametro. In particolare, l'instradamento in tali grafi è impegnativo perché il percorso più breve tra due nodi qualsiasi è moderatamente lungo.

Tuttavia, in netto contrasto, il grafo di LN è completamente diverso.

Il Grafo di Lightning oggi

Lightning è una rete finanziaria. Pertanto, la crescita e la formazione della rete sono influenzate anche da incentivi economici. Ogni volta che un nodo si unisce a LN, potrebbe voler massimizzare la sua connettività ad altri nodi per aumentare la sua efficienza di instradamento. Questo fenomeno è chiamato attaccamento preferenziale. Questi incentivi economici si traducono in una rete fondamentalmente diversa da un grafo casuale.

Sulla base di istantanee di canali annunciati pubblicamente, la distribuzione dei gradi di Lightning Network segue una funzione di legge di potenza. In un tale grafo, la stragrande maggioranza dei nodi ha pochissime connessioni ad altri nodi, mentre solo una manciata di

nodi ha numerose connessioni. Ad alto livello, questa topologia del grafo assomiglia a una stella: la rete ha un nucleo ben connesso e una parte periferica poco connessa. Le reti con distribuzione dei gradi della legge di potenza sono anche chiamate reti senza scala. Questa topologia è vantaggiosa per instradare i pagamenti in modo efficiente ma soggetta a determinati attacchi basati sulla topologia.

Attacchi basati sulla topologia

Un avversario potrebbe voler interrompere Lightning Network e potrebbe decidere che il suo obiettivo è smantellare l'intera rete in molti componenti più piccoli, rendendo praticamente impossibile l'instradamento dei pagamenti. Un obiettivo meno ambizioso, ma comunque dannoso e severo, potrebbe essere quello di abbattere solo alcuni nodi di rete. Tale interruzione potrebbe verificarsi a livello di nodo o a livello di bordo.

Supponiamo che un avversario possa abbattere qualsiasi nodo di Lightning Network. Ad esempio, può attaccarli con un attacco DDoS (Distributed Denial of Service) o renderli non operativi con qualsiasi mezzo. Scopriamo che se l'avversario sceglie i nodi in modo casuale, le reti scale-free come Lightning Network sono robuste contro gli attacchi di rimozione dei nodi. Questo perché un nodo casuale si trova alla periferia con un piccolo numero di connessioni, svolgendo quindi un ruolo trascurabile nella connettività della rete. Tuttavia, se l'avversario è più prudente, può prendere di mira i nodi meglio collegati. Non sorprende che Lightning Network e altre reti scale-free *non* siano resistenti agli attacchi mirati di rimozione dei nodi.

D'altra parte, l'avversario potrebbe essere più furtivo. Diversi attacchi basati sulla topologia prendono di mira un singolo nodo o un singolo canale di pagamento. Ad esempio, un avversario potrebbe essere interessato a esaurire di proposito la capacità di un determinato canale di pagamento. Più in generale, un avversario può esaurire tutta la capacità in uscita di un nodo per rimuoverlo dal mercato del routing. Ciò potrebbe essere facilmente ottenuto instradando i pagamenti attraverso il nodo vittima con importi pari alla capacità in uscita di ciascun canale di pagamento. Dopo aver completato questo cosiddetto attacco di isolamento del nodo, la vittima non può più inviare o instradare pagamenti a meno che non riceva un pagamento o ribilanci i suoi canali.

Per concludere, anche in base alla progettazione, è possibile rimuovere bordi e nodi dalla rete Lightning instradabile. Tuttavia, a seconda del vettore di attacco utilizzato, l'avversario potrebbe dover fornire più o meno risorse per eseguire l'attacco.

Temporalità del Lightning Network

Lightning Network è una rete che cambia dinamicamente e senza autorizzazione. I nodi possono entrare o uscire liberamente dalla rete, possono aprire e creare canali di pagamento ogni volta che lo desiderano. Pertanto, una singola istantanea statica del grafo di LN è fuorviante. Dobbiamo considerare la temporalità e la natura in continua evoluzione della rete. Per ora, il grafo LN sta crescendo in termini di numero di nodi e canali di pagamento. Anche il suo diametro effettivo si sta restringendo; cioè, i nodi si avvicinano l'uno all'altro, come possiamo vedere nella Figura 16-2.

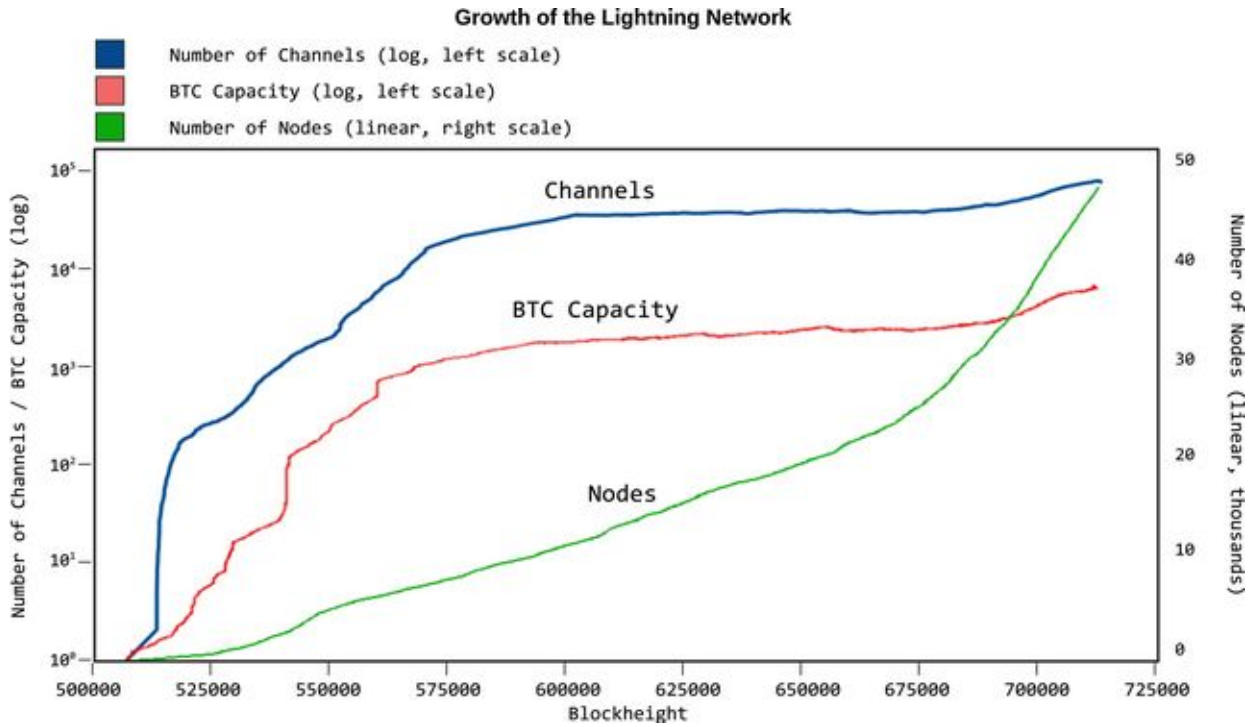

Figura 16-2. La costante crescita di LN in termini di nodi, canali e capacità

Nelle reti sociali, il comportamento di chiusura del triangolo è comune. Nello specifico, in un grafo in cui i nodi rappresentano le persone e le amicizie sono rappresentate come bordi, è in qualche modo previsto che nel grafico emergano dei triangoli. Un triangolo, in questo caso, rappresenta le amicizie a coppie tra tre persone. Ad esempio, se Alice conosce Bob e Bob conosce Charlie, allora è probabile che a un certo punto Bob presenterà Alice a Charlie. Tuttavia, questo comportamento sarebbe strano su Lightning Network. I nodi semplicemente non sono incentivati a chiudere i triangoli perché avrebbero potuto semplicemente

instradare i pagamenti invece di aprire un nuovo canale di pagamento. Sorprendentemente, la chiusura del triangolo è una pratica comune su Lightning Network. Il numero di triangoli era in costante crescita prima dell'implementazione dei pagamenti in più parti. Ciò è controintuitivo e sorprendente dato che i nodi avrebbero potuto semplicemente instradare i pagamenti attraverso i due lati del triangolo invece di aprire il terzo canale. Ciò può significare che le inefficienze di instradamento hanno incentivato gli utenti a chiudere i triangoli e a non ripiegare sull'instradamento.

Centralizzazione di Lightning Network

Una metrica comune per valutare la centralizzazione di un nodo in un grafo è la sua *centralità intermedia*. La dominanza del punto centrale è una metrica derivata dalla centralità intermedia, utilizzata per valutare la centralizzazione di una rete. Per una definizione precisa della dominanza del punto centrale, ti rimandiamo al lavoro di Freeman.

Più grande è il punto centrale di una rete, più centralizzata è la rete. Possiamo osservare che Lightning Network ha una maggiore dominanza del punto centrale (cioè è più centralizzato) di un grafo casuale (grafo di Erdős-Rényi) o di un grafo senza scala (grafo di Barabási-Albert) di uguale dimensione.

In generale, la nostra comprensione della natura dinamica del grafo dei canali di LN è piuttosto limitata. È fruttuoso analizzare come le modifiche al protocollo come i pagamenti in più parti possono influenzare le dinamiche di Lightning Network. Sarebbe utile esplorare la natura temporale del grafico LN in modo più approfondito.

Incentivi Economici e Struttura del Grafo

Il grafo di LN si forma spontaneamente e i nodi si connettono tra loro in base all'interesse reciproco. Di conseguenza, gli incentivi guidano lo sviluppo del grafo stesso. Diamo un'occhiata ad alcuni degli incentivi più rilevanti:

- Incentivi razionali:
    - I nodi stabiliscono canali per inviare, ricevere e instradare i pagamenti (guadagnando commissioni).

    - Cosa rende più probabile che si stabilisca un canale tra due nodi che agiscono razionalmente?
- Incentivi altruistici:
    - I nodi stabiliscono canali "per il bene della rete".
    - Sebbene non dovremmo basare le nostre ipotesi di sicurezza sull'altruismo, in una certa misura, il comportamento altruistico guida Bitcoin (accettando connessioni in entrata, servendo blocchi).
    - Che ruolo ha in Lightning?

Nelle prime fasi di Lightning Network, molti operatori di nodi hanno affermato che le commissioni di instradamento guadagnate non compensano i costi di opportunità derivanti dal blocco della liquidità. Ciò indicherebbe che il funzionamento di un nodo può essere guidato principalmente da incentivi altruistici "per il bene della rete". Ciò potrebbe cambiare in futuro se la rete Lightning dovesse avere un traffico significativamente maggiore o se emergesse un mercato per le tariffe di instradamento. D'altra parte, se un nodo desidera ottimizzare le proprie tariffe di instradamento, ridurrà al minimo la lunghezza media del percorso più breve verso ogni altro nodo. In altre parole, un nodo in cerca di profitto proverà a posizionarsi al centro del grafo del canale o vicino ad esso.

Consigli Pratici per Proteggere la Propria Privacy

Siamo ancora nelle prime fasi di Lightning Network. È probabile che molte delle preoccupazioni elencate in questo capitolo vengano affrontate man mano che matura e cresce. Nel frattempo, ci sono alcune misure che puoi adottare per proteggere il tuo nodo da utenti malintenzionati; qualcosa di semplice come l'aggiornamento dei parametri predefiniti con cui viene eseguito il tuo nodo può fare molto per rafforzare il tuo nodo.

Canali non annunciati

Se intendi utilizzare Lightning Network per inviare e ricevere fondi tra nodi e portafogli che controlli e non sei interessato a instradare i pagamenti di altri utenti, non è necessario annunciare i tuoi canali al resto della rete. Potresti aprire un canale tra, ad esempio, il tuo PC

desktop che esegue un nodo completo e il tuo smartphone che esegue un portafoglio Lightning leggero, e semplicemente rinunciare all'annuncio del canale discusso nel Capitolo 3. Questi sono talvolta chiamati canali "privati"; tuttavia è più corretto chiamarli canali "non annunciati" perché non sono strettamente privati.

I canali non annunciati non saranno noti al resto della rete e normalmente non verranno utilizzati per instradare i pagamenti di altri utenti. Possono ancora essere utilizzati per instradare i pagamenti se altri nodi ne vengono a conoscenza; ad esempio, una fattura potrebbe contenere suggerimenti di instradamento che suggeriscono un percorso tramite un canale non annunciato. Tuttavia, supponendo che tu abbia aperto solamente un canale non annunciato con te stesso, guadagni un certo grado di privacy. Poiché non stai esponendo il tuo canale alla rete, riduci il rischio di un attacco denial-of-service sul tuo nodo. Puoi anche gestire più facilmente la capacità di questo canale, poiché verrà utilizzato solo per ricevere o inviare direttamente al tuo nodo.

Ci sono anche vantaggi nell'aprire un canale non annunciato con una parte nota con cui effettui transazioni frequenti. Ad esempio, se Alice e Bob giocano spesso a poker in bitcoin, potrebbero aprire un canale per scambiarsi le vincite e perdite. In condizioni normali, questo canale non verrà utilizzato per instradare pagamenti da altri utenti e/o riscuotere commissioni. Poiché il canale non sarà noto al resto della rete, eventuali pagamenti tra Alice e Bob non possono essere dedotti tenendo traccia delle modifiche nella capacità di instradamento del canale. Ciò conferisce una certa privacy ad Alice e Bob; tuttavia, se uno di loro decide di far conoscere il canale ad altri utenti, ad esempio includendolo nei suggerimenti di instradamento di una fattura, questa privacy viene persa.

Va inoltre notato che per aprire un canale non annunciato è necessario effettuare una transazione pubblica sulla blockchain di Bitcoin. È quindi possibile dedurre l'esistenza e le dimensioni del canale se una parte malevola sta monitorando la blockchain per le transazioni di apertura dei canali e tentando di abbinarle ai canali sulla rete. Inoltre, quando il canale viene chiuso, il saldo finale del canale verrà reso pubblico una volta che sarà impegnato nella blockchain di Bitcoin. Tuttavia, poiché le transazioni di apertura e impegno sono pseudonime, non sarà semplice ricollegarlo ad Alice o Bob. L'aggiornamento Taproot del 2021 rende difficile distinguere tra transazioni di apertura/chiusura di canali e altri tipi specifici di transazioni Bitcoin. Pertanto, sebbene i canali non annunciati non siano completamente privati, offrono dei vantaggi in termini di privacy se utilizzati con attenzione.

Considerazioni sul Routing

Come spiegato in precedenza, i nodi che aprono canali pubblici si espongono al rischio di una serie di attacchi. Mentre le mitigazioni vengono sviluppate a livello di protocollo, ci sono molti passaggi che un nodo può intraprendere per proteggersi dagli attacchi denial of service sui propri canali pubblici:

Dimensione minima dell'HTLC

Sul canale aperto, il tuo nodo può impostare la dimensione HTLC minima che accetterà. L'impostazione di un valore più elevato garantisce che ciascuno dei tuoi slot di canale disponibili non possa essere occupato da un pagamento molto piccolo.

Limitazione della velocità

Molte implementazioni di nodi consentono ai nodi di accettare o rifiutare dinamicamente gli HTLC che vengono inoltrati attraverso il tuo nodo. Alcune linee guida utili per un limitatore di velocità personalizzato sono le seguenti:

- Limita il numero di slot di impegno che un singolo peer può consumare
- Monitora i tassi di errore da un singolo peer e limita la frequenza se i loro errori aumentano improvvisamente

Shadow channels

I nodi che desiderano aprire grandi canali verso un singolo target possono invece aprire un singolo canale pubblico verso il target e supportarlo con ulteriori canali privati chiamati shadow channel (canali ombra). Questi canali possono ancora essere utilizzati per il routing ma non vengono annunciati a potenziali attaccanti.

Accettare i canali

Al momento, i nodi Lightning faticano ad ottenere della liquidità in entrata. Sebbene esistano alcune soluzioni a pagamento per acquistare liquidità in entrata come exchange, mercati di canali e servizi di apertura di canali a pagamento da hub noti, molti nodi accetteranno volentieri qualsiasi richiesta di apertura di canali dall'aspetto legittimo per aumentare la loro

liquidità in entrata.

Tornando al contesto di Bitcoin, questo può essere paragonato al modo in cui Bitcoin Core tratta le sue connessioni in entrata e in uscita in modo diverso per timore che il nodo possa essere eclissato. Se un nodo apre una connessione in entrata al tuo nodo Bitcoin, non hai modo di sapere se l'iniziatore ti ha selezionato casualmente o se sta prendendo di mira specificamente il tuo nodo con intenti malevoli. Le tue connessioni in uscita non devono essere trattate con tale sospetto perché o il nodo è stato selezionato casualmente da un pool di molti potenziali peer o ti sei connesso intenzionalmente al peer manualmente.

Lo stesso si può dire in Lightning. Quando apri un canale, lo fai intenzionalmente, ma quando una parte remota apre un canale sul tuo nodo, non hai modo di sapere se questo canale verrà utilizzato per attaccare il tuo nodo o meno. Come mostrano diverse ricerche, il costo relativamente basso per lanciare un nodo ed aprire canali verso specifici target è uno dei fattori significativi che rendono gli attacchi facili da eseguire. Se accetti canali in entrata, dovresti valutare l'idea di porre alcune restrizioni sui nodi da cui accetti tali canali. Molte implementazioni espongono politiche di accettazione di canale che ti consentono di specificare le tue personali preferenze.

La questione dell'accettazione e del rifiuto dei canali è filosofica. Cosa succede se finiamo con una rete Lightning in cui i nuovi nodi non possono partecipare perché non possono aprire alcun canale? Il nostro suggerimento non è quello di impostare un elenco esclusivo di "mega-hub" da cui accetterai i canali, ma piuttosto di accettare i canali in un modo che si adattino alle tue preferenze di rischio.

Alcune potenziali strategie sono:

Nessun rischio

: Non accettare alcun canale in entrata.

Basso rischio

: Accetta i canali da un set noto di nodi con cui hai precedentemente aperto i canali con successo.

Medio rischio

Accetta solo canali da nodi che sono stati presenti nel grafo per un periodo più lungo e hanno canali di lunga durata.

Alto rischio

Accetta tutti i canali in entrata e implementa le mitigazioni descritte in precedenza.

Conclusione

In sintesi, la privacy e la sicurezza sono argomenti complessi e mentre molti ricercatori e sviluppatori sono alla ricerca di miglioramenti a livello di rete, è importante che tutti i partecipanti di LN capiscano cosa possono fare per proteggere la propria privacy e aumentare la sicurezza a livello di singolo nodo.

Riferimenti e Ulteriori Letture

In questo capitolo abbiamo utilizzato molti riferimenti tratti da ricerche sulla sicurezza di Lightning. Di seguito trovi questi articoli e documenti utili elencati per argomento.

Attacchi alla privacy e probing

- Jordi Herrera-Joancomartí et al. "On the Difficulty of Hiding the Balance of Lightning Network Channels". *Asia CCS '19: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security* (Luglio 2019): 602–612.
- Utz Nisslmueller et al. "Toward Active and Passive Confidentiality Attacks on Cryptocurrency Off-Chain Networks." arXiv preprint, *https://arxiv.org/abs/2003.00003* (2020).
- Sergei Tikhomirov et al. "Probing Channel Balances in the Lightning Network." arXiv preprint, *https://arxiv.org/abs/2004.00333* (2020).
- George Kappos et al. "An Empirical Analysis of Privacy in the Lightning Network." arXiv preprint, *https://arxiv.org/abs/2003.12470* (2021).

- Codice sorgente di Zap con funzione di sondaggio.

Attacchi di congestione

- Ayelet Mizrahi e Aviv Zohar. "Congestion Attacks in Payment Channel Networks." arXiv preprint, *https://arxiv.org/abs/2002.06564* (2020).

Considerazioni sul routing

- Marty Bent, intervista con Joost Jager, Tales from the Crypt, podcast audio, Ottobre 2, 2020, *https://anchor.fm/tales-from-the-crypt/episodes/197-Joost-Jager-ekghn6*.

Conclusione

In pochi anni, Lightning Network è passato da un white paper a una rete globale in rapida crescita. Come secondo livello di Bitcoin, ha mantenuto la promessa di pagamenti veloci, economici e privati. Inoltre, ha dato il via a uno tsunami di innovazione, poiché libera gli sviluppatori dai vincoli del consenso serrato che esistono nello sviluppo di Bitcoin.

L'innovazione di Lightning Network sta avvenendo su diversi livelli:

- Al protocollo Core di Bitcoin, fornendo l'uso e la domanda di nuovi codici operativi Bitcoin Script, algoritmi di firma e ottimizzazioni
- A livello di protocollo Lightning, con nuove funzionalità implementate rapidamente in tutta la rete
- A livello di canale di pagamento, con nuovi costrutti e miglioramenti del canale
- Come funzionalità opt-in distinte distribuite end-to-end da implementazioni indipendenti che mittenti e destinatari possono utilizzare se lo desiderano
- Con le nuove ed entusiasmanti applicazioni Lightning (LApps) basate su client e protocolli

Diamo un'occhiata a come queste innovazioni stanno cambiando Lightning...

Innovazione Decentralizzata e Asincrona

Lightning non è vincolato dal consenso lockstep come nel caso di Bitcoin. Ciò significa che diversi client Lightning possono implementare funzionalità diverse e negoziare le loro interazioni. Di conseguenza, l'innovazione di LN si sta verificando a un ritmo molto più rapido rispetto a Bitcoin.

Non solo Lightning sta avanzando rapidamente, ma sta creando domanda per nuove funzionalità nel sistema Bitcoin. Molte innovazioni recenti e pianificate in Bitcoin sono sia motivate che giustificate dal loro utilizzo su Lightning Network. In effetti, Lightning Network viene spesso citato come caso d'uso esemplificativo per molte delle nuove funzionalità.

Innovazione del Protocollo Bitcoin e di Bitcoin Script

Il sistema Bitcoin è, per necessità, un sistema conservativo che deve preservare la compatibilità con le regole del consenso per evitare fork non pianificati della blockchain o partizioni della rete P2P. Di conseguenza, le nuove funzionalità richiedono molto coordinamento e test prima di essere implementate in mainnet, il sistema di produzione live.

Ecco alcune delle innovazioni attuali o proposte in Bitcoin che sono motivate da casi d'uso su Lightning Network:

Neutrino

> Un protocollo client leggero con funzionalità di privacy migliorate rispetto al protocollo SPV legacy. Neutrino è utilizzato principalmente dai client Lightning per accedere alla blockchain di Bitcoin.

Firme di Schnorr

> Introdotte come parte del soft fork Taproot, le firme Schnorr consentono contratti flessibili Point Time-Locked (PTLC) per la costruzione di canali Lightning. Ciò permette l'utilizzo di firme revocabili anziché transazioni revocabili.

Taproot

> Parte del soft fork di novembre 2021 che introduce le firme Schnorr, Taproot consente a script complessi di apparire come singoli pagamenti a firma singola, indistinguibili dal tipo più comune di pagamento su Bitcoin. Ciò consente alle transazioni di chiusura cooperativa (reciproca) dei canali Lightning di apparire in modo indistinguibile rispetto ai semplici pagamenti e aumenta la privacy per gli utenti di Lightning Network.

Input rebinding

> Conosciuto anche con i nomi SIGHAS_NOINPUT o SIGHASH_ANYPREVOUT, questo aggiornamento del linguaggio Bitcoin Script è principalmente motivato da contratti intelligenti (smart contract) avanzati come il protocollo del canale eltoo.

Covenants

I covenants consentono alle transazioni di creare output che vincolano le transazioni future che li spendono. Questo meccanismo potrebbe aumentare la sicurezza per i canali Lightning rendendo possibile l'applicazione dell'elenco degli indirizzi consentiti nelle transazioni di impegno.

Innovazione del Protocollo Lightning

Il protocollo Lightning P2P è altamente estensibile e ha subito molti cambiamenti sin dal suo inizio. La regola "It's OK to be odd" utilizzata nei bit delle funzionalità garantisce che i nodi possano negoziare le funzionalità che supportano, consentendo più aggiornamenti indipendenti al protocollo.

Estensibilità TLV

Il meccanismo Type-Length-Value per estendere il protocollo di messaggistica è estremamente potente e ha già consentito l'introduzione di diverse nuove funzionalità in Lightning mantenendo la compatibilità sia con le versioni precedenti che con quelle future. Un esempio importante, che è attualmente in fase di sviluppo e che ne fa uso, è l'accecamento del percorso (path blinding) e i pagamenti trampolino (trampoline payments). Ciò consente a un destinatario di nascondersi dal mittente, consentendo ai client mobile di inviare pagamenti senza la necessità di memorizzare l'intero grafo dei canali sui propri dispositivi utilizzando una terza parte cui non è necessario rivelare il destinatario finale.

Costruzione del Canale di Pagamento

I canali di pagamento sono un'astrazione gestita da due partner di canale. Finché questi due sono disposti a eseguire nuovo codice, possono implementare contemporaneamente una varietà di meccanismi di canale. In effetti, recenti ricerche suggeriscono che i canali potrebbero persino essere aggiornati a un nuovo meccanismo in modo dinamico, senza chiudere il vecchio canale e aprire un nuovo tipo di canale.

eltoo

Un meccanismo di canale che utilizza il rebinding di input per semplificare in modo significativo il funzionamento dei canali di pagamento ed eliminare la necessità del meccanismo di penalità.

Funzionalità Opt-In End-to-End

Contratti a tempo determinato (Point Time-Locked Contracts)

Un approccio diverso agli HTLC, i PTLC possono aumentare la privacy, ridurre le informazioni condivise ai nodi intermedi e operare in modo più efficiente rispetto ai canali basati su HTLC.

Canali più grandi

I canali Large o Wumbo sono stati introdotti in modo dinamico nella rete senza richiedere coordinamento. I canali che supportano pagamenti di grandi dimensioni vengono pubblicizzati come parte dei messaggi di annuncio del canale e possono essere utilizzati in modalità opt-in.

Pagamenti in più parti (MPP)

Anche MPP è stato introdotto in modalità opt-in, ma ancora meglio richiede solo che il mittente e il destinatario di un pagamento siano in grado di eseguire MPP. Il resto della rete instrada semplicemente gli HTLC come se fossero pagamenti singoli.

Instradamento JIT (Jit Routing)

Un metodo facoltativo che può essere utilizzato dai nodi di instradamento per aumentare la loro affidabilità e per proteggersi dallo spam.

Keysend

Un aggiornamento introdotto in modo indipendente dalle implementazioni dei client Lightning, consente al mittente di inviare denaro in modo "non richiesto" e asincrono

senza richiedere prima una fattura.

Fatture HODL[1]

Pagamenti in cui l'HTLC finale non viene riscosso, impegnando il mittente al pagamento, ma consentendo al destinatario di ritardare l'incasso fino a quando non viene soddisfatta qualche altra condizione, o annullare la fattura senza incasso. Anche questo è stato implementato in modo indipendente da diversi client Lightning e può essere utilizzato in modo opt-in.

Servizi di messaggi instradati via onion

Il meccanismo di onion routing e il sottostante database a chiave pubblica dei nodi possono essere utilizzati per inviare dati non correlati ai pagamenti, come messaggi di testo o post di forum. L'uso di Lightning per abilitare la messaggistica a pagamento come soluzione allo spam e agli attacchi Sybil è un'altra innovazione che è stata implementata indipendentemente dal protocollo principale.

Offerte

Attualmente proposto come BOLT #12 ma già implementato da alcuni nodi, si tratta di un protocollo di comunicazione per richiedere fatture (ricorrenti) da nodi remoti tramite messaggi onion.

Applicazioni Lightning (LApp – Lightning Applications)

Stiamo già assistendo all'emergere di interessanti applicazioni Lightning. Definite in generale come un'applicazione che utilizza il protocollo Lightning o un client Lightning come componente, le LApp sono il livello dell'applicazione di Lightning. Un esempio importante è LNURL, che fornisce una funzionalità simile a quella offerta da BOLT #12, ma appena sopra gli indirizzi HTTP e Lightning. Fornisce agli utenti un indirizzo in stile e-mail a cui altri possono inviare fondi mentre il software in background richiede una fattura verso l'endpoint LNURL del nodo. Ulteriori LApp sono in fase di realizzazione per giochi semplici, applicazioni di messaggistica, microservizi, API a pagamento, distributori a pagamento (es. pompe di

carburante), sistemi di trading di derivati e molto altro.

Pronti, Partenza, Via!

Il futuro è estremamente positivo ed incoraggiante. Lightning Network sta portando Bitcoin in nuovi mercati e applicazioni inesplorate. Dotato delle conoscenze contenute in questo libro, puoi esplorare questa nuova frontiera e forse anche unirti come pioniere e creare una nuova idea o futuro rivoluzionario.

---

1. La parola HODL deriva da un eccitato errore di ortografia della parola "HOLD" gridata in un forum per incoraggiare le persone a non vendere bitcoin in preda al panico.

Appendice A. Fondamenti di Bitcoin

Lightning Network è in grado di funzionare su più blockchain, ma è principalmente ancorato a Bitcoin. Per comprendere LN, è necessaria una conoscenza fondamentale di Bitcoin e dei suoi elementi costitutivi.

Ci sono molte risorse che puoi utilizzare per saperne di più su Bitcoin, incluso il libro Mastering Bitcoin: Traduzione italiana della guida completa al mondo di bitcoin e della blockchain, 2ª edizione, di Riccardo Masutti. Non è necessario che tu legga un altro libro intero per capire cos'è Bitcoin... Lo spiegheremo in maniera semplificata in questa sezione.

In questo capitolo abbiamo raccolto i concetti più importanti che devi sapere su Bitcoin e li esplichiamo nel contesto di Lightning Network. In questo modo puoi imparare esattamente ciò che devi sapere per comprendere LN nella sua interezza.

Questo capitolo copre diversi concetti importanti di Bitcoin, tra cui:

- Chiavi e firme digitali
- Funzioni di hash
- Transazioni Bitcoin e loro struttura
- Concatenamento delle transazioni Bitcoin
- Output di transazione
- Bitcoin Script: script di blocco e sblocco
- Script di blocco di base
- Script di blocco complessi e condizionali
- Timelock

Chiavi e Firme Digitali

Potresti aver sentito che Bitcoin si basa sulla *crittografia*, che è una branca della matematica

ampiamente utilizzata nella sicurezza informatica. La crittografia può anche essere utilizzata per provare la conoscenza di un segreto senza rivelarlo (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.

La proprietà di bitcoin viene stabilita tramite *chiavi digitali*, *indirizzi bitcoin* e *firme digitali*. Le chiavi digitali non sono effettivamente memorizzate nella rete, ma vengono invece create e archiviate dagli utenti in un file o database, chiamato *portafoglio* (*wallet*). Le chiavi digitali nel portafoglio di un utente sono completamente indipendenti dal protocollo Bitcoin e possono essere generate e gestite dal software del portafoglio dell'utente senza riferimento alla blockchain o senza necessità di avere accesso a Internet.

La maggior parte delle transazioni Bitcoin richiede l'inclusione di una firma digitale valida nella blockchain, che può essere generata solo con una chiave segreta; pertanto, chiunque abbia una copia di quella chiave ha il controllo dei bitcoin all'interno di un indirizzo. La firma digitale utilizzata per spendere fondi viene anche chiamata *testimone* (*witness*). I dati del testimone in una transazione bitcoin testimoniano la vera proprietà dei fondi spesi. Le chiavi sono disponibili in coppie costituite da una chiave privata (segreta) e una chiave pubblica. Pensa alla chiave pubblica come un IBAN e alla chiave privata come un PIN segreto.

Chiavi Private e Pubbliche

Una chiave privata è semplicemente un numero scelto a caso. In pratica, e per semplificare la gestione di molte chiavi, la maggior parte dei portafogli Bitcoin genera una sequenza di chiavi private da un singolo *seme* (*seed*) casuale utilizzando un algoritmo di derivazione deterministico. In poche parole, un singolo numero casuale viene utilizzato per produrre una sequenza ripetibile di numeri apparentemente casuali che vengono utilizzati come chiavi private. Ciò consente agli utenti di eseguire solo il backup del seme ed essere in grado di *derivare* tutte le chiavi di cui hanno bisogno da quel seme.

Bitcoin, come molte altre criptovalute e blockchain, utilizza *curve ellittiche* per la propria sicurezza. In Bitcoin, la moltiplicazione sulla curva ellittica *secp256k1* viene utilizzata come *funzione unidirezionale*. In poche parole, la natura della matematica della curva ellittica rende banale calcolare la moltiplicazione scalare di un punto ma impossibile calcolare l'inverso (divisione o logaritmo discreto).

Ogni chiave privata ha una *chiave pubblica* corrispondente (che viene calcolata dalla chiave

privata) utilizzando la moltiplicazione scalare sulla curva ellittica. In termini semplici, con una chiave privata k, possiamo moltiplicarla per una costante *G* per produrre una chiave pubblica *K*:

$$K = k*G$$

È impossibile invertire tale calcolo. Data una chiave pubblica *K*, non si può calcolare la chiave privata *k*. La divisione per *G* non è possibile nella matematica della curva ellittica. Invece, si dovrebbero provare tutti i possibili valori di *k* in un processo esaustivo chiamato *attacco di forza bruta*. Poiché *k* è un numero a 256 bit, esaurire tutti i valori possibili con qualsiasi computer richiederebbe più tempo ed energia di quanto disponibile in questo universo.

Hash

Un altro strumento importante ampiamente utilizzato in Bitcoin e su Lightning Network sono le *funzioni hash crittografiche*, e in particolare la funzione hash SHA-256.

Una funzione hash, nota anche come *funzione digest*, è una funzione che prende dati di lunghezza arbitraria e li trasforma in un risultato di lunghezza fissa, chiamato *hash*, *digest* o *fingerprint* (vedi Figura A-1). È importante sottolineare che le funzioni hash sono funzioni unidirezionali, il che significa che non è possibile invertirle e calcolare i dati di input dall'impronta digitale.

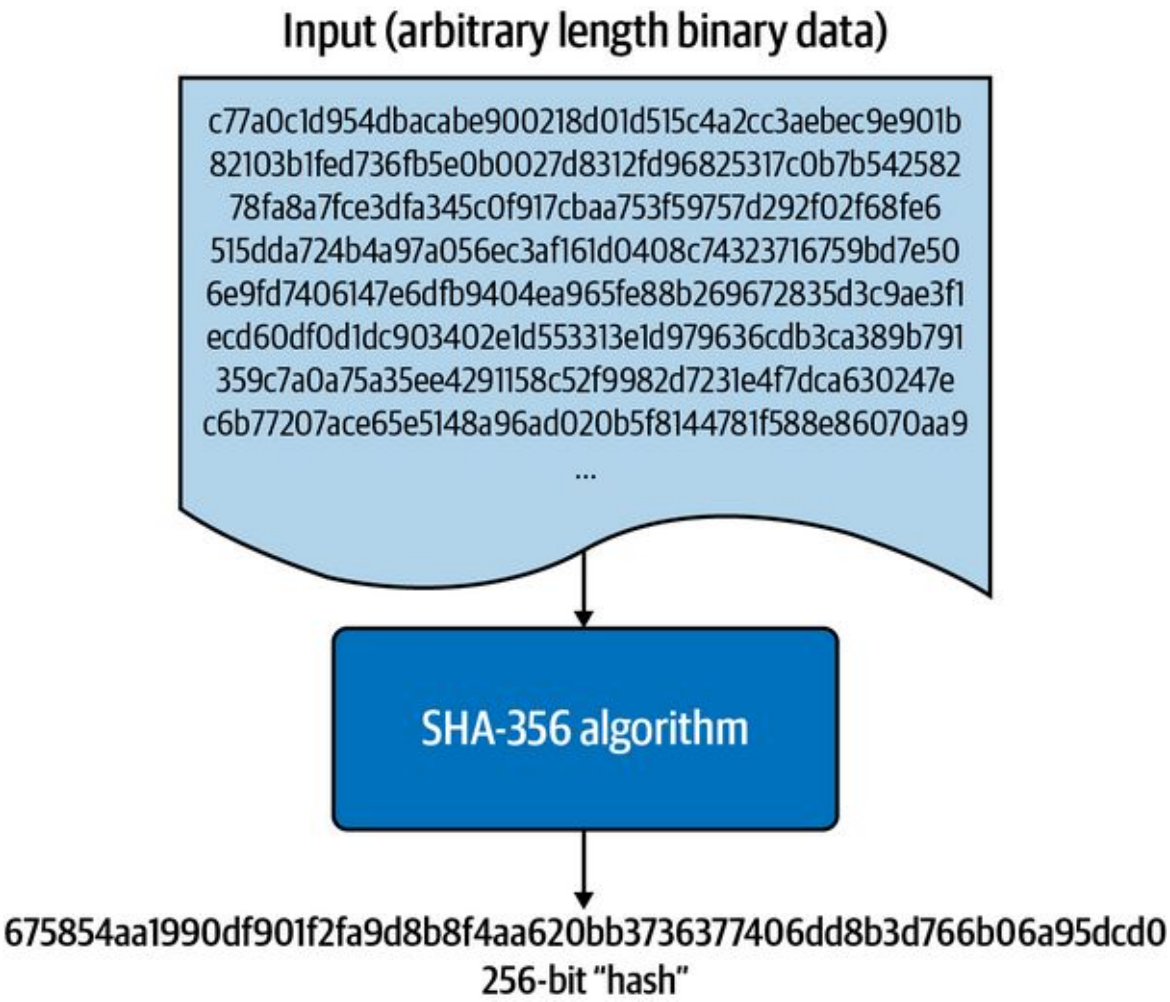

Figura A-1. L'algoritmo hash crittografico SHA-256

Ad esempio, se utilizziamo un terminale a riga di comando per inserire il testo "Mastering the Lightning Network" nella funzione SHA-256, produrrà un'impronta digitale di questo tipo:

```
$ echo -n "Mastering the Lightning Network" | shasum -a 256
ce86e4cd423d80d054b387aca23c02f5fc53b14be4f8d3ef14c089422b2235de  -
```

NOTA	L'input utilizzato per calcolare un hash è anche chiamato *preimage*.

La lunghezza dell'input può essere molto più grande. Proviamo la stessa cosa con il file PDF del whitepaper Bitcoin di Satoshi Nakamoto:

```
$ wget http://bitcoin.org/bitcoin.pdf
$ cat bitcoin.pdf | shasum -a 256
b1674191a88ec5cdd733e4240a81803105dc412d6c6708d53ab94fc248f4f553  -
```

Sebbene richieda più tempo di una singola frase, la funzione SHA-256 elabora il PDF di 9 pagine, "digerendolo" in un'impronta digitale a 256 bit.

Ti starai chiedendo come sia possibile che una funzione che digerisce dati di dimensioni illimitate produca un'impronta digitale univoca che è un numero di dimensioni fisse...

In teoria, poiché esiste un numero infinito di possibili preimage (input) e solo un numero finito di impronte digitali, devono esserci molte preimage che producono la stessa impronta digitale a 256 bit. Quando due preimage producono lo stesso hash, si parla di *collisione*.

In pratica, un numero a 256 bit è così grande che non troverai mai una collisione di proposito. Le funzioni hash crittografiche funzionano sulla base del fatto che la ricerca di una collisione è uno sforzo che richiede così tanta energia e tempo tale da essere impossibile.

Le funzioni hash crittografiche sono ampiamente utilizzate in una varietà di applicazioni perché hanno alcune caratteristiche utili. Sono:

Deterministiche

: Lo stesso input produce sempre lo stesso hash.

Irreversibili

: Non è possibile calcolare la preimmagine di un hash.

A prova di collisione

: È computazionalmente impossibile trovare due messaggi con lo stesso hash.

Non correlati

: Un piccolo cambiamento nell'input produce un cambiamento così grande nell'output che l'output sembra non correlato all'input.

Uniforme/casuale

Una funzione hash crittografica produce hash distribuiti uniformemente nell'intero spazio a 256 bit dei possibili output. L'output di un hash sembra essere casuale, sebbene non sia veramente casuale.

Utilizzando queste funzionalità possiamo creare alcune applicazioni interessanti:

Impronte digitali

Un hash può essere utilizzato per l'impronta digitale di un file o di un messaggio in modo che possa essere identificato in modo univoco. Gli hash possono essere utilizzati come identificatori universali di qualsiasi set di dati.

A prova di integrità

Un'impronta digitale di un file o di un messaggio dimostra la sua integrità perché il file o il messaggio non può essere manomesso o modificato in alcun modo senza cambiare l'impronta digitale. Questo viene spesso utilizzato per garantire che il software non sia stato manomesso prima di installarlo sul computer.

Impegno/non ripudio

Puoi impegnarti in una preimmagine specifica (ad esempio, un numero o un messaggio) senza rivelarlo, pubblicando il suo hash. Successivamente, puoi rivelare il segreto e tutti possono verificare che è la stessa cosa cui ti sei impegnato in precedenza perché produce l'hash pubblicato.

Proof of Work/hash grinding

Puoi usare un hash per dimostrare di aver svolto un lavoro computazionale mostrando uno schema non casuale nell'hash che può essere prodotto solo da ipotesi ripetute su una preimage. Ad esempio, l'hash di un'intestazione di blocco Bitcoin inizia con molti zero bit. L'unico modo per produrlo è modificare una parte dell'intestazione e eseguirne l'hashing trilioni di volte finché non produce quel modello per caso.

Atomicità

> Puoi rendere una preimage segreta un prerequisito per spendere fondi in diverse transazioni collegate. Se una delle parti rivela la preimage per spendere una delle transazioni, anche tutte le altre parti possono spendere le proprie transazioni. Tutto o nulla diventa spendibile, raggiungendo l'atomicità attraverso diverse transazioni.

Firme Digitali

La chiave privata viene utilizzata per creare firme necessarie per spendere bitcoin dimostrando la proprietà dei fondi utilizzati in una transazione.

Una *firma digitale* è un numero che viene calcolato dall'applicazione della chiave privata a un messaggio specifico.

Dato un messaggio m e una chiave privata k, una funzione di firma F_{sign} può produrre una firma S:

$S = F_{sign}(m, k)$

Questa firma S può essere verificata indipendentemente da chiunque abbia la chiave pubblica K (corrispondente alla chiave privata k), e il messaggio:

$F_{verify}(m, K, S)$

Se F_{verify} restituisce un risultato vero, chi verifica può confermare che il messaggio m è stato firmato da qualcuno che aveva accesso alla chiave privata k. È importante sottolineare che la firma digitale prova il possesso della chiave privata k al momento della firma, senza rivelarla.

Le firme digitali utilizzano un algoritmo hash crittografico. La firma viene applicata a un hash del messaggio, in modo che il messaggio m venga "riassunto" in un hash di lunghezza fissa $H(m)$ che funge da impronta digitale.

Applicando la firma digitale sull'hash di una transazione, la firma non solo prova l'autorizzazione, ma anche "blocca" i dati della transazione, assicurandone l'integrità. Una transazione firmata non può essere modificata perché qualsiasi modifica comporterebbe un hash diverso e invaliderebbe la firma.

Tipi di firma

Le firme non vengono sempre applicate all'intera transazione. Per fornire flessibilità di firma, una firma digitale Bitcoin contiene un prefisso chiamato tipo di hash della firma, che specifica quale parte dei dati della transazione è inclusa nell'hash. Ciò consente alla firma di impegnare o "bloccare" tutti o solo alcuni dei dati nella transazione. Il tipo di hash della firma più comune è `SIGHASH_ALL` che blocca tutto nella transazione includendo tutti i dati della transazione nell'hash firmato. In confronto, `SIGHASH_SINGLE` blocca tutti gli input della transazione, ma solo un output (ulteriori informazioni su input e output nella sezione successiva). Diversi tipi di hash della firma possono essere combinati per produrre sei diversi "schemi" di dati di transazione bloccati dalla firma.

Ulteriori informazioni sui tipi di hash della firma sono disponibili nella sezione "Signature Hash Types (SIGHASH)" a pagina 245 (Capitolo 6) di Mastering Bitcoin: Traduzione italiana della guida completa al mondo di bitcoin e della blockchain, 2ª edizione, di Riccardo Masutti.

Transazioni Bitcoin

Le *transazioni* sono strutture di dati che codificano il trasferimento di valore tra i partecipanti al sistema Bitcoin.

Input e Output

Gli *output delle transazioni* sono blocchi indivisibili di bitcoin, registrati sulla blockchain e riconosciuti come validi dal network. Sono l'elemento fondamentale di ogni transazione. Una transazione spende input e crea output. Gli *input delle transazioni* sono riferimenti agli output di transazioni registrate in precedenza. Ogni transazione spende gli output delle transazioni precedenti e crea nuovi output (vedi Figura A-2).

Figura A-2. Una transazione trasferisce valore dagli input agli output

I nodi completi (full node) di Bitcoin tengono traccia di tutti gli output disponibili e spendibili, noti come *output di transazione non spesi* (UTXO, *unspent transaction output*). La raccolta di tutti gli UTXO è nota come UTXO set, che attualmente conta milioni di UTXO. Il set di UTXO cresce man mano che vengono creati nuovi UTXO e si riduce quando gli UTXO vengono consumati. Ogni transazione rappresenta un cambiamento (transizione di stato) nel set di UTXO, consumando uno o più UTXO come *input della transazione* e creando uno o più UTXO come *output della transazione*.

Ad esempio, supponiamo che Alice abbia un UTXO di 100.000 satoshi che può spendere. Alice può inviare a Bob 100.000 satoshi costruendo una transazione con un input (consumando il suo input di 100.000 satoshi esistente) e un output che "paga" Bob 100.000 satoshi. Ora Bob ha un UTXO di 100.000 satoshi che può spendere, creando una nuova transazione che consuma questo nuovo UTXO e lo spende in un altro UTXO come pagamento a un altro utente, e così via (vedi Figura A-3).

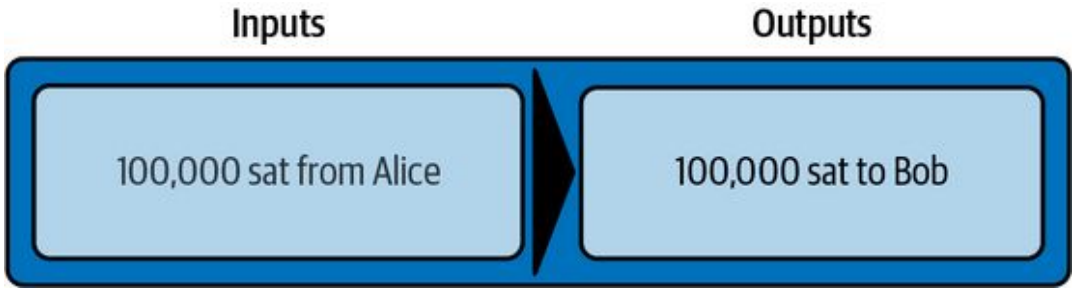

Figura A-3. Alice paga 100.000 satoshi a Bob

L'output di una transazione può avere un valore arbitrario (intero) denominato in satoshi. Proprio come i dollari possono essere divisi fino a due cifre decimali come centesimi, i bitcoin possono essere divisi fino a otto cifre decimali come satoshi. Sebbene un output possa avere qualsiasi valore arbitrario, una volta creato è indivisibile. Questa è una caratteristica importante degli output che deve essere sottolineata: gli output sono unità di valore discrete e indivisibili, denominate in satoshi interi. Un output non speso può essere consumato nella sua interezza solo da una transazione.

Quindi cosa succede se Alice vuole pagare a Bob 50.000 satoshi, ma ha solo un UTXO indivisibile di 100.000 satoshi? Alice dovrà creare una transazione che consumi (come input) l'UTXO da 100.000 satoshi e abbia due output: uno che paga 50.000 satoshi a Bob e uno che *restituisce* 50.000 satoshi ad Alice come "resto" (vedi Figura A-4).

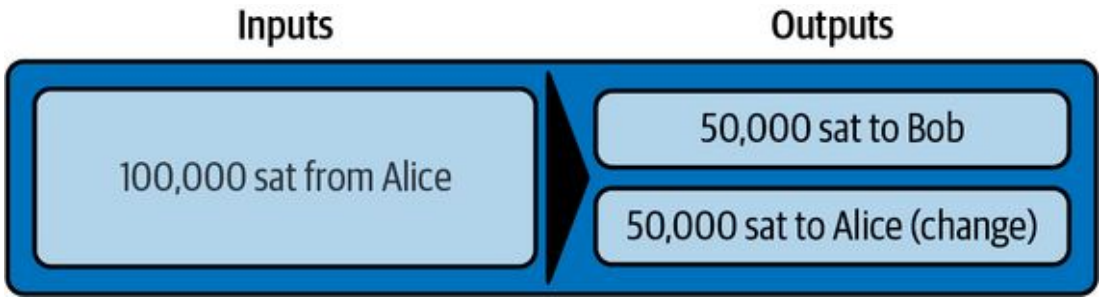

Figura A-4. Alice paga 50k sat a Bob e 50k sat a se stessa come resto

SUGGERIMENTO	Non c'è niente di speciale in un output di resto o un modo per distinguerlo da qualsiasi altro output. Non deve essere l'ultimo output. Potrebbe esserci più di un output di resto o nessun output di resto. Solo il creatore della transazione sa quali output sono per gli altri e quali output sono per gli indirizzi che possiedono e quindi "di resto".

Allo stesso modo, se Alice vuole pagare a Bob 85.000 satoshi ma ha due UTXO da 50.000 satoshi disponibili, deve creare una transazione con due input (consumando entrambi i suoi UTXO da 50.000 satoshi) e due output, pagando Bob 85.000 e inviando a se stessa 15.000 satoshi come resto (vedi Figura A-5).

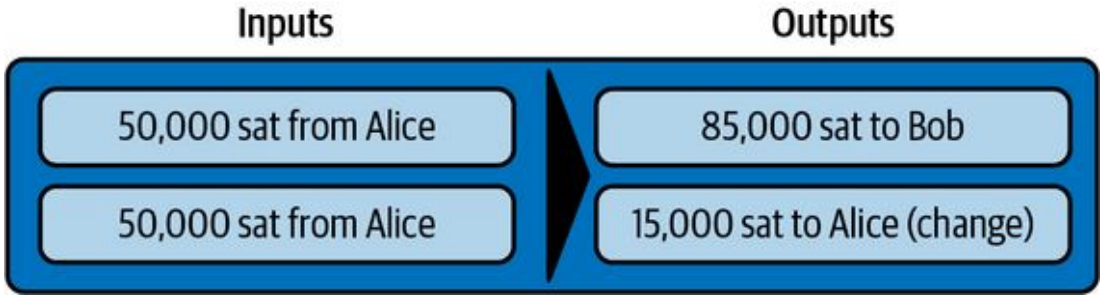

Figura A-5. Alice utilizza due input da 50k per pagare 85k sat a Bob e 15k sat a se stessa come resto

Le illustrazioni e gli esempi precedenti mostrano come una transazione Bitcoin combina (spende) uno o più input e crea uno o più output. Una transazione può avere centinaia o addirittura migliaia di input e output.

SUGGERIMENTO	Sebbene le transazioni create da Lightning Network abbiano output multipli, non hanno "resto" di per sé, perché l'intero saldo disponibile di un canale è suddiviso tra i due partner di canale.

Catene di Transazioni

Ogni output può essere speso come input in una transazione successiva. Quindi se Bob decidesse di spendere 10.000 satoshi in una transazione per pagare Chan, e Chan spendesse 4.000 satoshi per pagare Dina, le cose andrebbero come mostrato nella figura A-6.

Un output è considerato *speso* se viene indicato come input in un'altra transazione registrata sulla blockchain. Un output è considerato *non speso* (e disponibile per la spesa) se nessuna transazione registrata vi fa riferimento.

L'unico tipo di transazione che non ha input è una transazione speciale creata dai miner di Bitcoin chiamata *transazione coinbase*. La transazione coinbase ha solo output e nessun input perché crea nuovi bitcoin attraverso il processo di mining. Ogni altra transazione spende uno o più output precedentemente registrati come input.

Poiché le transazioni sono concatenate, se scegli una transazione a caso, puoi seguire uno qualsiasi dei suoi input all'indietro fino alla transazione precedente che l'ha creata. Se continui a farlo, alla fine raggiungerai una transazione coinbase in cui il bitcoin è stato estratto per la prima volta.

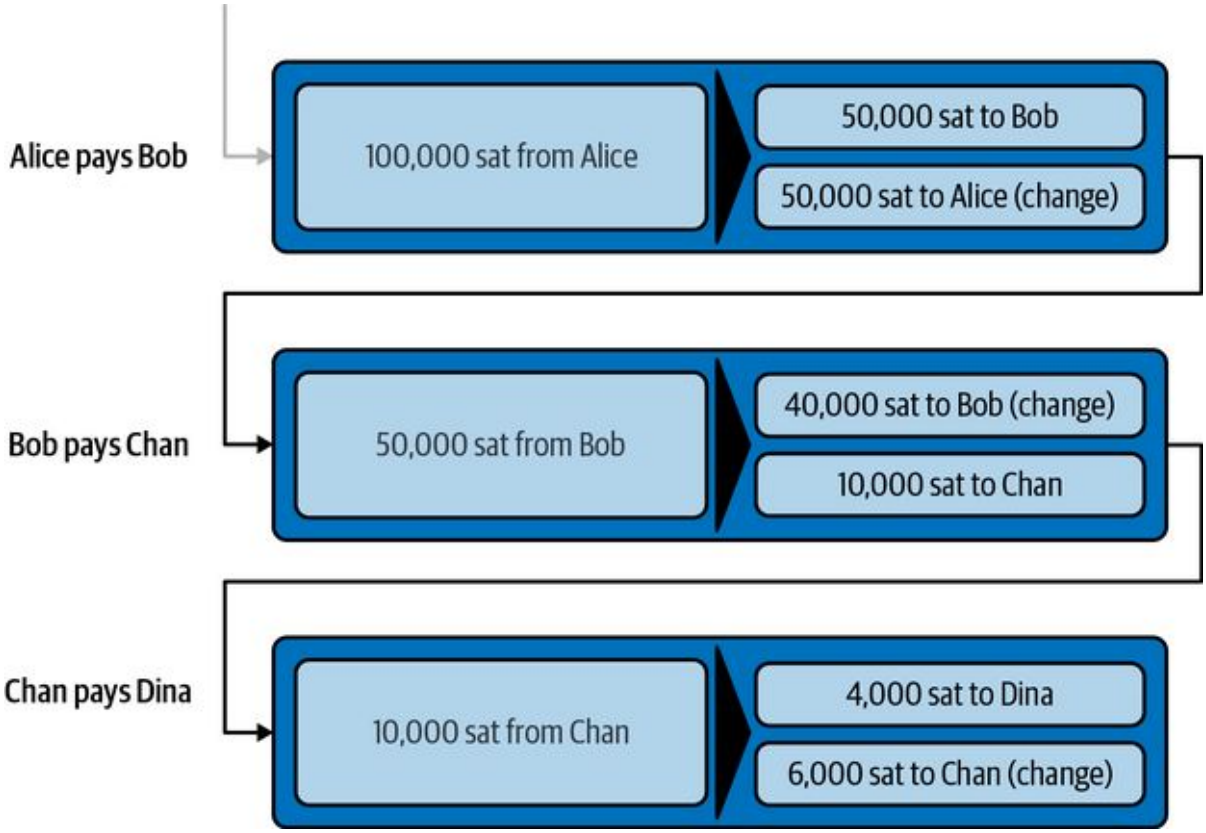

Figura A-6. Alice paga Bob che paga Chan che paga Dina

TxID: Identificatori di Transazione

Ogni transazione nel sistema Bitcoin è identificata da un identificatore univoco (assumendo l'esistenza di BIP-0030), chiamato *ID transazione* o *TxID* in breve. Per produrre un identificatore univoco, utilizziamo la funzione hash crittografica SHA-256 per produrre un hash dei dati della transazione. Questa "impronta digitale" funge da identificatore universale. Una transazione può essere referenziata dal suo ID transazione e una volta che una transazione viene registrata sulla blockchain Bitcoin, ogni nodo nella rete Bitcoin sa che questa transazione è valida.

Ad esempio, un ID transazione potrebbe avere il seguente aspetto:

```
e31e4e214c3f436937c74b8663b3ca58f7ad5b3fce7783eb84fd9a5ee5b9a54c
```

Questa è una transazione reale (creata come esempio per il libro Mastering Bitcoin) che può essere trovata sulla blockchain. Prova a trovarla inserendo questo TxID in un block explorer:

https://blockstream.info/tx/e31e4e214c3f436937c74b8663b3ca58f7ad5b3fce7783eb84fd9a5ee5b9a54c

o utilizza questo link breve:

https://bit.ly/AliceTx

Outpoint: Identificatori di Output

Poiché ogni transazione ha un ID univoco, possiamo anche identificare un output di transazione all'interno di quella transazione in modo univoco facendo riferimento al TxID e al numero di indice di output. Il primo output in una transazione è l'indice di output 0, il secondo output è l'indice di output 1 e così via. Un identificatore di output è comunemente noto come *outpoint*.

Per convenzione scriviamo un outpoint con TxID, due punti e il numero dell'indice di output:

```
7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18:
0
```

Gli identificatori di output (outpoint) sono i meccanismi che collegano le transazioni insieme in una catena. Ogni input di transazione è un riferimento a uno specifico output di una transazione precedente. Tale riferimento è un outpoint: un TxID e un numero di indice di output. Quindi una transazione "spende" un output specifico (per numero di indice) da una transazione specifica (per TxID) per creare nuovi output che possono essere spesi a loro volta facendo riferimento all'outpoint.

La Figura A-7 presenta la catena di transazioni da Alice a Bob a Chan a Dina, questa volta con outpoint in ciascuno degli input.

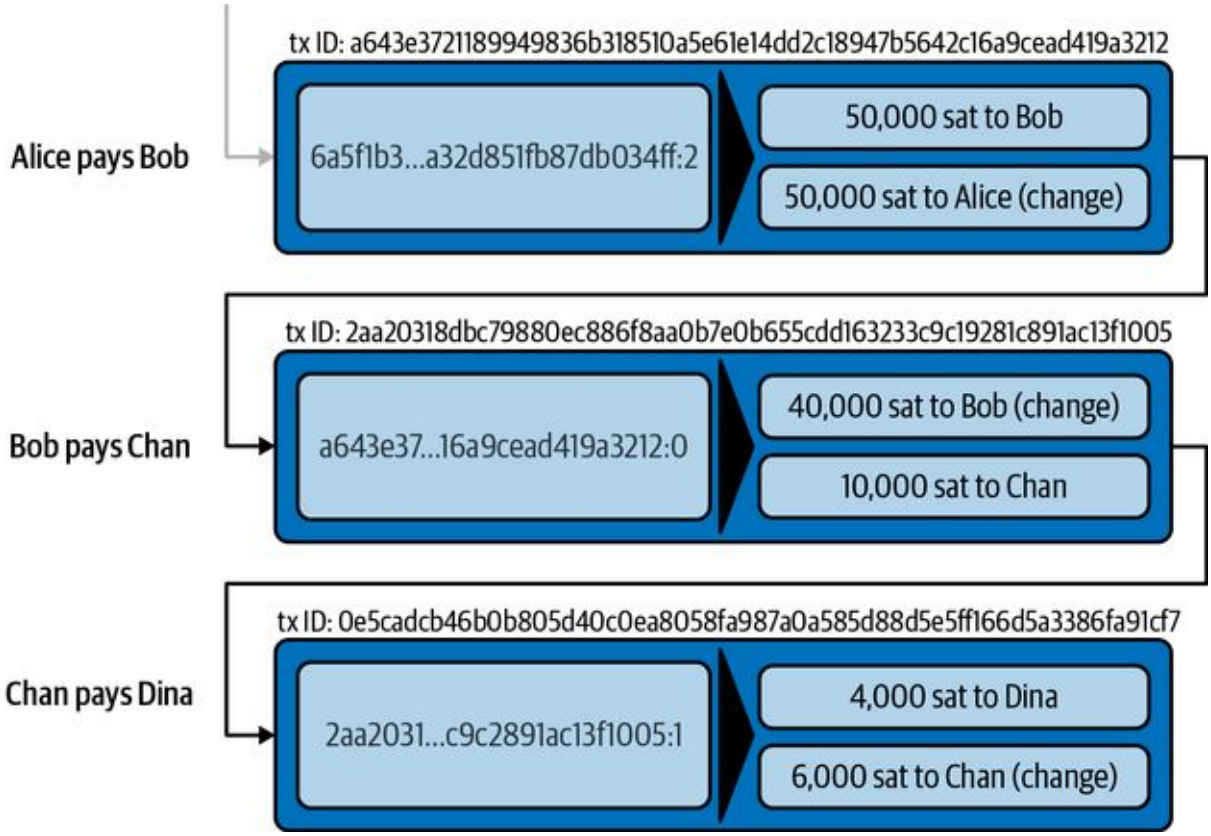

Figura A-7. Gli input di transazione si riferiscono agli outpoint che formano una catena

L'input nella transazione di Bob fa riferimento alla transazione di Alice (tramite TxID) e l'output indicizzato 0.

L'input nella transazione di Chan fa riferimento al TxID della transazione di Bob e al primo output indicizzato, perché il pagamento a Chan è l'output #1. Nel pagamento di Bob a Chan, il resto di Bob è nell'output #0.[1]

Ora, se osserviamo il pagamento di Alice a Bob, possiamo vedere che Alice sta spendendo un outpoint che era il terzo (output index #2) output in una transazione il cui ID è 6a5f1b3[...]. Non vediamo quella transazione referenziata nel diagramma, ma possiamo dedurre questi dettagli dall'outpoint.

Bitcoin Script

L'elemento finale di Bitcoin necessario per completare la nostra comprensione è il linguaggio di scripting che controlla l'accesso agli outpoint. Finora abbiamo semplificato la descrizione dicendo "Alice firma la transazione per pagare Bob". Dietro le quinte, tuttavia, c'è una certa complessità nascosta che rende possibile implementare condizioni di spesa più complesse. La condizione di spesa più semplice e comune è "presentare una firma che corrisponda alla seguente chiave pubblica". Una condizione di spesa come questa viene registrata in ogni output come *script di blocco* scritto in un linguaggio di scripting chiamato *Bitcoin Script*.

Bitcoin Script è un linguaggio di scripting basato su stack estremamente semplice. Non contiene loop o ricorsione e quindi è *Turing incompleto* (il che significa che non può esprimere una complessità arbitraria e ha un'esecuzione prevedibile). Chi ha familiarità con il (ormai antico) linguaggio di programmazione FORTH riconoscerà la sintassi e lo stile.

Esecuzione di Bitcoin Script

In termini semplici, il sistema Bitcoin valuta Bitcoin Script eseguendo lo script su uno stack; se il risultato finale è VERO, considera soddisfatta la condizione di spesa e valida la transazione.

Diamo un'occhiata a un esempio molto semplice di Bitcoin Script, che somma i numeri 2 e 3 e poi confronta il risultato con il numero 5:

```
2 3 ADD 5 EQUAL
```

Nella Figura A-8, vediamo come viene eseguito questo script (da sinistra a destra).

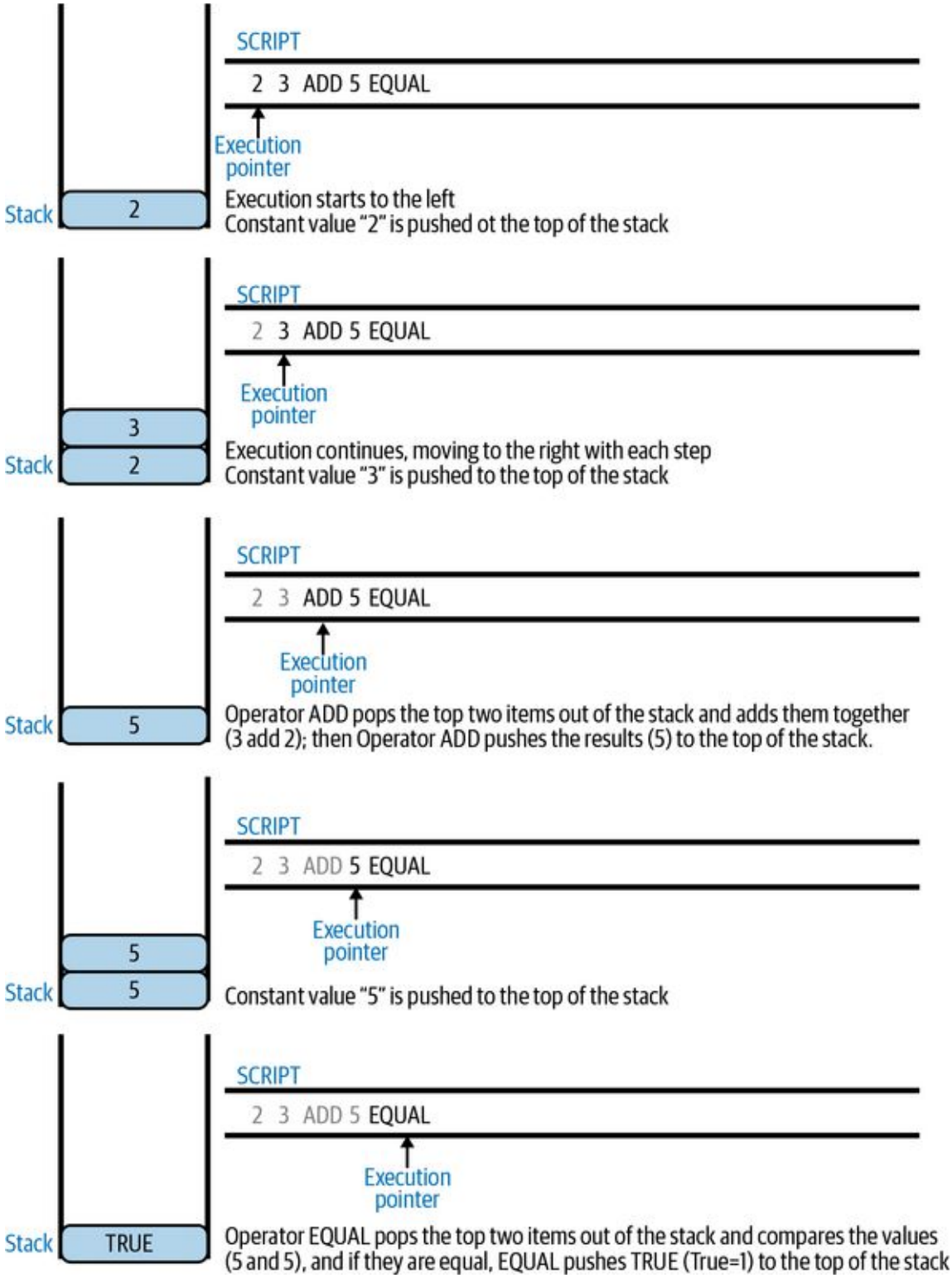

Figura A-8. Esempio di esecuzione di Bitcoin Script

Script di Blocco e Sblocco

Bitcoin Script è composto da due parti:

Script di blocco

> Sono incorporati negli output delle transazioni, stabilendo le condizioni che devono essere soddisfatte per spendere quell'output. Ad esempio, il portafoglio di Alice aggiunge uno script di blocco all'output pagando Bob, che imposta la condizione per cui è necessaria la firma di Bob per spenderlo.

Script di sblocco

> Sono incorporati negli input della transazione, soddisfacendo le condizioni stabilite dallo script di blocco dell'output di riferimento. Ad esempio, Bob può sbloccare l'output precedente fornendo uno script di sblocco contenente una firma digitale.

Utilizzando un modello semplificato, per la convalida, lo script di sblocco e di blocco vengono concatenati ed eseguiti (P2SH e SegWit sono eccezioni). Ad esempio, se qualcuno ha bloccato l'output di una transazione con lo script di blocco "`3 ADD 5 EQUAL`", potremmo spenderlo con lo script di sblocco "2" in un input di transazione. Chiunque convalidi tale transazione concatenerebbe il nostro script di sblocco (2) e lo script di blocco (`3 ADD 5 EQUAL`) ed eseguirebbe il risultato attraverso il motore di esecuzione di Bitcoin Script. La risposta sarebbe `TRUE` e quindi viene data la possibilità di spendere l'output.

Ovviamente, questo esempio semplificato sarebbe una scelta molto sbagliata per bloccare un vero output di Bitcoin perché non c'è segreto, solo aritmetica di base. Chiunque potrebbe spendere l'output fornendo la risposta "2". La maggior parte degli script di blocco richiede quindi la dimostrazione della conoscenza di un segreto.

Blocco su una Chiave Pubblica (Firma)

La forma più semplice di uno script di blocco è quella che richiede una firma. Consideriamo la transazione di Alice che paga a Bob 50.000 satoshi. L'output creato da Alice per pagare Bob avrà uno script di blocco che richiede la firma di Bob e sarà simile a questo:

```
<Bob Public Key> CHECKSIG
```

L'operatore `CHECKSIG` prende due elementi dallo stack: una firma e una chiave pubblica. Come puoi vedere, la chiave pubblica di Bob è nello script di blocco, quindi ciò che manca è la firma corrispondente a quella chiave pubblica. Questo script di blocco può essere speso solo da Bob, perché solo Bob ha la corrispondente chiave privata necessaria per produrre una firma digitale corrispondente alla chiave pubblica. Per sbloccarlo, Bob dovrebbe fornire uno script di sblocco contenente solo la sua firma digitale:

```
<Bob Signature>
```

Nella Figura A-9 puoi vedere lo script di blocco nella transazione di Alice (nell'output che paga Bob) e lo script di sblocco (nell'input che spende quell'output) nella transazione di Bob.

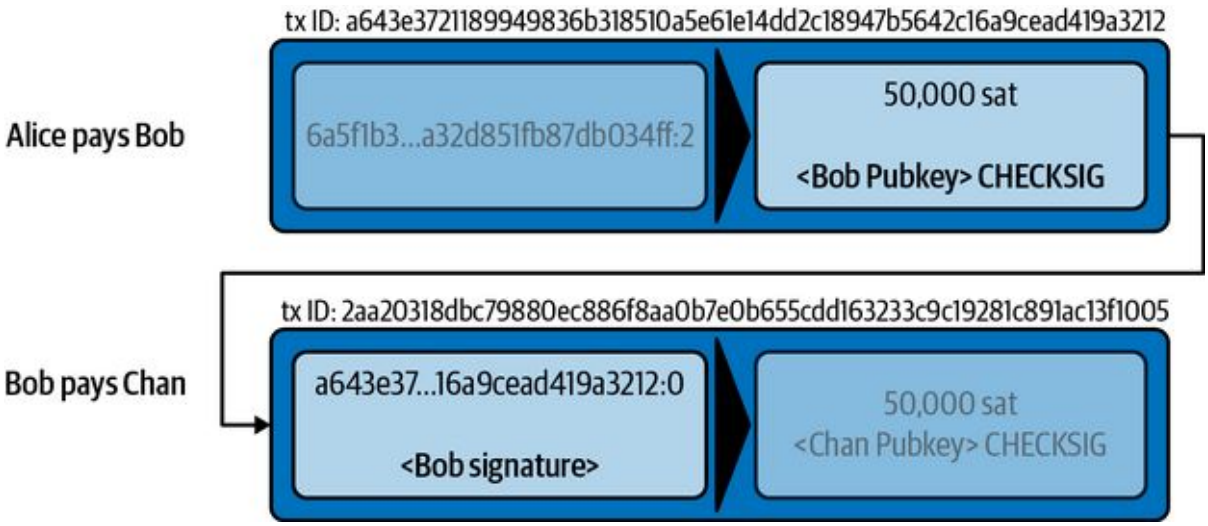

Figura A-9. Catena di transazioni che mostra lo script di blocco (output) e di sblocco (input)

Per convalidare la transazione di Bob, un nodo Bitcoin farebbe quanto segue:

1. Estrae lo script di sblocco dall'input (`<Bob Signature>`).
2. Cerca l'outpoint che sta tentando di spendere (`a643e37...3213:0`). Questa è la transazione di Alice e si trova sulla blockchain.
3. Estrae lo script di blocco da quell'outpoint (`<Bob PubKey> CHECKSIG`).
4. Concatena in uno script, posizionando lo script di sblocco davanti allo script di blocco (`<Bob Signature> <Bob PubKey> CHECKSIG`).
5. Esegue questo script sul motore di esecuzione di Bitcoin Script per vedere quale

risultato viene prodotto.

6. Se il risultato è TRUE, deduce che la transazione di Bob è valida perché è stata in grado di soddisfare la condizione di spesa per spendere quell'outpoint.

Blocco su un Hash (Segreto)

Un altro tipo di script di blocco, utilizzato in Lightning Network, è un *hashlock*. Per sbloccarlo, devi conoscere la *preimmagine* segreta dell'hash.

Per dimostrarlo, chiediamo a Bob di generare un numero casuale R e di tenerlo segreto:

```
R = 1833462189
```

Ora Bob calcola l'hash SHA-256 di questo numero:

```
H = SHA256(R) =>
```

```
H = SHA256(1833462189) =>
```

```
H = 0ffd8bea4abdb0deafd6f2a8ad7941c13256a19248a7b0612407379e1460036a
```

Successivamente Bob invia ad Alice l'hash H che abbiamo calcolato in precedenza, ma mantiene segreto il numero R. Ricorda che a causa delle proprietà degli hash crittografici, Alice non può "invertire" il calcolo dell'hash e indovinare il numero R.

Alice crea un output pagando 50.000 satoshi con lo script di blocco:

```
HASH256 H EQUAL
```

dove H è il valore hash effettivo (`0ffd8...036a`) che Bob ha dato ad Alice.

Spieghiamo questo script:

L'operatore HASH256 estrae un valore dallo stack e calcola l'hash SHA-256 di quel valore. Poi mette il risultato in pila.

Il valore H viene inserito nello stack, quindi l'operatore EQUAL controlla se i due valori sono uguali e inserisce TRUE o FALSE nello stack di conseguenza.

Pertanto, questo script di blocco funzionerà solo se combinato con uno script di sblocco che contiene R, in modo che una volta concatenato avremo:

```
R HASH256 H EQUAL
```

Solo Bob conosce R, quindi solo Bob può produrre una transazione con uno script di sblocco

che rivela il valore segreto R.

È interessante notare che Bob può dare il valore R a chiunque altro, che può quindi spendere quel Bitcoin. Questo rende il valore segreto R quasi come un "voucher" bitcoin, poiché chiunque lo abbia può spendere l'output creato da Alice. Vedremo come questa sia una proprietà utile per Lightning Network!

Script Multifirma

Il linguaggio Bitcoin Script fornisce un multisignature building block (primitiva), che può essere utilizzato per creare servizi di deposito a garanzia e complesse configurazioni di proprietà tra diverse parti interessate. Un accordo che richiede più firme per spendere Bitcoin è chiamato *schema multifirma*, ulteriormente specificato come schema *K-di-N*, dove:

- *N* è il numero totale di firmatari identificati nello schema multifirma
- *K* è il *quorum* o *soglia*: il numero minimo di firme per autorizzare la spesa

Lo script per un multisig *K-di-N* è:

```
K <PubKey1> <PubKey2> ... <PubKeyN> N CHECKMULTISIG
```

dove *N* è il numero totale di chiavi pubbliche elencate (dalla Chiave Pubblica 1 alla Chiave Pubblica *N*) e *K* è la soglia delle firme richieste per spendere l'output.

Lightning Network utilizza uno schema multifirma 2-di-2 per creare un canale di pagamento. Ad esempio, un canale di pagamento tra Alice e Bob verrebbe costruito su un multisig 2-di-2:

```
2 <PubKey Alice> <PubKey Bob> 2 CHECKMULTISIG
```

Lo script di blocco precedente può essere soddisfatto con uno script di sblocco contenente una coppia di firme:[2]

```
0 <Sig Alice> <Sig Bob>
```

I due script insieme formerebbero lo script di convalida combinato:

```
0 <Sig Alice> <Sig Bob> 2 <PubKey Alice> <PubKey Bob> 2 CHECKMULTISIG
```

Uno script di blocco multifirma può essere rappresentato da un indirizzo Bitcoin, che codifica l'hash dello script di blocco. Ad esempio, la transazione di finanziamento iniziale di un canale di pagamento Lightning è una transazione che paga a un indirizzo che codifica uno script di blocco di un multisig 2-di-2 dei due partner di canale.

Script di Timelock

Un altro elemento ampiamente utilizzato in Lightning Network è un *timelock*. Un timelock è una restrizione sulla spesa che richiede che sia trascorso un certo tempo o un'altezza del blocco prima che la spesa sia consentita. È un po' come un assegno postdatato tratto da un conto bancario che non può essere incassato prima della data indicata sull'assegno.

Bitcoin ha due livelli di timelock: timelock a livello di transazione e timelock a livello di output.

Un *timelock a livello di transazione* viene registrato nel campo `nLockTime` della transazione e impedisce l'accettazione dell'intera transazione prima che il timelock sia passato. I timelock a livello di transazione sono il meccanismo di timelock più comunemente utilizzato in Bitcoin.

Un *timelock a livello di output* viene creato da un operatore di script. Esistono due tipi di timelock di output: timelock assoluti e timelock relativi.

I timelock assoluti a livello di output sono implementati dall'operatore `CHECKLOCKTIMEVERIFY`, che viene spesso abbreviato nelle conversazioni come *CLTV*. I timelock assoluti implementano un vincolo temporale con un timestamp assoluto o blockheight, che esprime l'equivalente di "non spendibile prima del blocco 800.000".

I *timelock relativi* a livello di output sono implementati dall'operatore `CHECKSEQUENCEVERIFY`, spesso abbreviato come *CSV*. I timelock relativi implementano un vincolo di spesa relativo alla conferma della transazione, esprimendo l'equivalente di "non può essere speso fino a 1.024 blocchi dopo la conferma".

Script con più Condizioni

Una delle funzionalità più potenti di Bitcoin Script è il controllo del flusso, noto anche come clausole condizionali. Probabilmente hai familiarità con il controllo di flusso in vari linguaggi di programmazione che utilizzano il costrutto `IF...THEN...ELSE`. Le clausole condizionali di Bitcoin hanno un aspetto leggermente diverso, ma sono praticamente lo stesso costrutto.

A livello di base, i codici operativi ci consentono di costruire uno script di blocco che ha due modi per essere sbloccato, a seconda di un risultato `TRUE/FALSE` della valutazione di una condizione logica. Ad esempio, se x è `TRUE`, lo script di blocco è A `ELSE` lo script di blocco è B.

Inoltre, le espressioni condizionali di Bitcoin possono essere *nidificate* indefinitamente, il che significa che una clausola condizionale può contenerne un'altra al suo interno, che ne

contiene un'altra, ecc. Il controllo del flusso di Bitcoin Script può essere utilizzato per costruire script molto complessi con centinaia o addirittura migliaia di possibili percorsi di esecuzione. Non c'è limite all'annidamento, ma le regole di consenso impongono un limite alla dimensione massima, in byte, di uno script.

Bitcoin implementa il controllo del flusso utilizzando i codici operativi `IF`, `ELSE`, `ENDIF` e `NOTIF`. Inoltre, le espressioni condizionali possono contenere operatori booleani come `BOOLAND`, `BOOLOR` e `NOT`.

A prima vista, potresti trovare confusi gli script di controllo del flusso di Bitcoin. Questo perché Bitcoin Script è un linguaggio stack. Allo stesso modo in cui l'operazione aritmetica `1 + 1` sembra "volta all'indietro" quando espressa in Bitcoin Script come `1 1 ADD`, anche le clausole di controllo del flusso in Bitcoin sembrano "volte all'indietro".

Nella maggior parte dei linguaggi di programmazione (procedurali) tradizionali, il controllo di flusso si presenta così:

```
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 lo fa sembrare "all'indietro", in questo modo:

```
condition
IF
  code to run when condition is true
ELSE
  code to run when condition is false
ENDIF
code to run in either case
```

Quando leggi Bitcoin Script, ricorda che la condizione da valutare viene *prima* del codice operativo `IF`.

Utilizzo del Controllo di Flusso negli Script

Un uso molto comune per il controllo del flusso in Bitcoin Script è costruire uno script di blocco che offra più percorsi di esecuzione, ciascuno 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 verrebbe espresso come uno script multisig 1-di-2. Per motivi di dimostrazione, faremo la stessa cosa con una clausola `IF`:

```
IF
 <Alice's Pubkey> CHECKSIG
ELSE
 <Bob's Pubkey> CHECKSIG
ENDIF
```

Guardando questo script di blocco, potresti chiederti: "Dov'è la condizione? Non c'è niente che precede la clausola `IF`!" La condizione non fa parte dello script di blocco. Verrà *offerta nello script di sblocco*, consentendo ad Alice e Bob di "scegliere" quale percorso di esecuzione desiderano.

Alice lo riscatta con lo script di sblocco:

```
<Alice's Sig> 1
```

L'`1` alla fine funge da condizione (*TRUE*) che farà eseguire alla clausola `IF` il primo percorso di riscatto per il quale 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 inserisce uno `0` nello stack, facendo in modo che la clausola `IF` esegua il secondo script (`ELSE`), che richiede la firma di Bob.

Poiché ciascuna delle due condizioni richiede anche una firma, Alice non può utilizzare la seconda clausola e Bob non può utilizzare la prima clausola; non hanno le firme necessarie!

Poiché i flussi condizionali possono essere nidificati, lo stesso vale per i valori `TRUE/FALSE` nello script di sblocco, per navigare in un percorso complesso di condizioni.

Nell'Esempio A-1 puoi vedere uno script complesso utilizzato in LN, con diverse condizioni. Gli script utilizzati in Lightning Network sono altamente ottimizzati e compatti, per ridurre al minimo il peso on-chain, quindi non sono facili da leggere e comprendere. Tuttavia, prova ad

identificare alcuni dei concetti di Bitcoin Script che abbiamo appreso in questo capitolo.

Esempio A-1. Uno script complesso utilizzato in Lightning Network

```
# To remote node with revocation key
DUP HASH160 <RIPEMD160(SHA256(revocationpubkey))> EQUAL
IF
    CHECKSIG
ELSE
    <remote_htlcpubkey> SWAP SIZE 32 EQUAL
    NOTIF
        # To local node via HTLC-timeout transaction (timelocked).
        DROP 2 SWAP <local_htlcpubkey> 2 CHECKMULTISIG
    ELSE
        # To remote node with preimage.
        HASH160 <RIPEMD160(payment_hash)> EQUALVERIFY
        CHECKSIG
    ENDIF
ENDIF
```

---

1. Ricorda che il resto non deve essere l'ultimo output di una transazione ed è infatti indistinguibile dagli altri output.
2. Il primo argomento (0) non ha alcun significato ma è richiesto a causa di un bug nell'implementazione multifirma di Bitcoin. Questo problema è descritto in Mastering Bitcoin, Capitolo 7.

   Da BOLT #3.

Appendice B. Basi di Docker, Installazione e Utilizzo

Questo libro contiene una serie di esempi che vengono eseguiti all'interno dei container Docker per la standardizzazione su diversi sistemi operativi.

Questa sezione ti aiuterà a installare Docker e a familiarizzare con alcuni dei comandi Docker più comunemente utilizzati, in modo da poter eseguire i contenitori di esempio del libro.

Installazione di Docker

Prima di iniziare, dovresti installare il sistema di contenitori Docker sul tuo computer. Docker è un sistema aperto distribuito gratuitamente come *Community Edition* per molti sistemi operativi diversi tra cui Windows, macOS e Linux. Le versioni Windows e Macintosh sono denominate *Docker Desktop* e sono costituite da un'applicazione desktop GUI e strumenti da riga di comando. La versione Linux si chiama *Docker Engine* ed è composta da un demone del server e strumenti da riga di comando. Useremo gli strumenti della riga di comando, che sono identici su tutte le piattaforme.

Vai avanti e installa Docker per il tuo sistema operativo seguendo le istruzioni per "Get Docker" dal sito web di Docker.

Seleziona il tuo sistema operativo dall'elenco e segui le istruzioni di installazione.

SUGGERIMENTO	Se esegui l'installazione su Linux, segui le istruzioni post-installazione per assicurarti di poter eseguire Docker come utente normale anziché come utente root. Altrimenti, dovrai aggiungere il prefisso a tutti i comandi `docker` con `sudo`, eseguendoli come root come: `sudo docker`.

Dopo aver installato Docker, puoi testare la tua installazione eseguendo il container demo `hello-world` in questo modo:

```
$ docker run hello-world
Hello from Docker!
```

```
This message shows that your installation appears to be working correctly.
[...]
```

Comandi Docker di Base

In questa appendice, utilizziamo Docker in modo piuttosto esteso. Useremo i seguenti comandi e argomenti Docker.

Creazione di un Contenitore

```
docker build [-t tag] [directory]
```

`tag` è il modo in cui identifichiamo il contenitore che stiamo costruendo e `directory` è dove si trova il contenitore (cartelle e file) e il file di definizione (`Dockerfile`).

Esecuzione di un Contenitore

```
docker run -it [--network netname] [--name cname] tag
```

`netname` è il nome di una rete Docker, `cname` è il nome che scegliamo per questa istanza di container e `tag` è l'etichetta del nome che abbiamo dato al container durante la creazione.

Esecuzione di un Comando in un Contenitore

```
docker exec cname command
```

`cname` è il nome che abbiamo assegnato al contenitore nel comando di esecuzione e `command` è un eseguibile o uno script che vogliamo eseguire all'interno del contenitore.

Arresto e Avvio di un Contenitore

Nella maggior parte dei casi, se stiamo eseguendo un contenitore in modalità *interattiva* oltre che *terminale*, ovvero con i flag `i` e `t` (combinati come `-it`) impostati, il contenitore può essere fermato semplicemente premendo Ctrl-C o uscendo dalla shell con `exit` o Ctrl-D. Se un contenitore non termina, puoi fermarlo da un altro terminale in questo modo:

```
docker stop cname
```

Per riprendere un contenitore già esistente, utilizza il comando `start` in questo modo:

```
docker start cname
```

Eliminazione di un Contenitore per Nome

Se assegni un nome a un contenitore anziché consentire a Docker di assegnargli un nome casuale, non puoi riutilizzare tale nome fino a quando il contenitore non viene eliminato. Docker restituirà un errore come questo:

```
docker: Error response from daemon: Conflict. The container name "/bitcoind" is already in use...
```

Per risolvere questo problema, elimina l'istanza esistente del contenitore:

```
docker rm cname
```

`cname` è il nome assegnato al contenitore (`bitcoind` nel messaggio di errore di esempio).

Elenco dei Contenitori in Esecuzione

```
docker ps
```

Questo comando mostra i contenitori attualmente in esecuzione ed i relativi nomi.

Elenco delle Immagini Docker

```
docker image ls
```

Questo comando mostra le immagini Docker che sono state create o scaricate sul tuo computer.

Conclusione

Questi comandi Docker di base saranno sufficienti per iniziare e ti permetteranno di eseguire tutti gli esempi in questo libro.

Appendice C. Wire Protocol Messages

Questa appendice elenca tutti i tipi di messaggio attualmente definiti ed utilizzati nel protocollo Lightning P2P. Inoltre, mostriamo la struttura di ciascun messaggio, mostrando i messaggi in raggruppamenti logici basati sui flussi del protocollo.

NOTA	I messaggi del Protocollo Lightning sono estensibili e la loro struttura può cambiare durante gli aggiornamenti a livello di rete. Per informazioni autorevoli, consulta l'ultima versione dei BOLT nel repository GitHub Lightning-RFC.

Tipi di Messaggio

I tipi di messaggio attualmente definiti sono elencati nella Tabella C-1.

Tabella C-1. Tipi di messaggio

Numero Intero	Nome Messaggio	Categoria
16	`init`	Stabilimento della Connessione
17	`error`	Errore di Comunicazione
18	`ping`	Vivacità della Connessione
19	`pong`	Vivacità della Connessione
32	`open_channel`	Finanziamento del Canale
33	`accept_channel`	Finanziamento del Canale

Numero Intero	Nome Messaggio	Categoria
34	`funding_created`	Finanziamento del Canale
35	`funding_signed`	Finanziamento del Canale
36	`funding_locked`	Finanziamento + Operazione del Canale
38	`shutdown`	Chiusura del Canale
39	`closing_signed`	Chiusura del Canale
128	`update_add_htlc`	Operazione del Canale
130	`update_fulfill_hltc`	Operazione del Canale
131	`update_fail_htlc`	Operazione del Canale
132	`commit_sig`	Operazione del Canale
133	`revoke_and_ack`	Operazione del Canale
134	`update_fee`	Operazione del Canale
135	`update_fail_malformed_htlc`	Operazione del Canale
136	`channel_reestablish`	Operazione del Canale
256	`channel_announcement`	Annuncio del Canale
257	`node_announcement`	Annuncio del Canale

Numero Intero	Nome Messaggio	Categoria
258	`channel_update`	Annuncio del Canale
259	`announce_signatures`	Annuncio del Canale
261	`query_short_chan_ids`	Sincronizzazione del Grafo dei Canali
262	`reply_short_chan_ids_end`	Sincronizzazione del Grafo dei Canali
263	`query_channel_range`	Sincronizzazione del Grafo dei Canali
264	`reply_channel_range`	Sincronizzazione del Grafo dei Canali
265	`gossip_timestamp_range`	Sincronizzazione del Grafo dei Canali

Nella Tabella 13-1, il campo `Categoria` ci consente di classificare rapidamente un messaggio in base alla sua funzionalità all'interno del protocollo stesso. Ad alto livello, inseriamo un messaggio in uno degli otto bucket (non esaustivi) tra cui:

Stabilimento della Connessione

Inviato quando viene stabilita per la prima volta una connessione p2p. Utilizzato anche per negoziare il set di funzionalità supportate da una nuova connessione.

Errore di Comunicazione

Utilizzato dai peer per comunicare il verificarsi di errori a livello di protocollo.

Vivacità della connessione

Utilizzato dai peer per verificare che una determinata connessione sia ancora attiva.

Finanziamento del Canale

Utilizzato per creare un nuovo canale di pagamento. Questo processo è anche noto come processo di finanziamento del canale.

Operazione del Canale

L'atto di aggiornare un dato canale off-chain. Ciò include l'invio e la ricezione di pagamenti, nonché l'inoltro di pagamenti all'interno della rete.

Annuncio del Canale

Il processo di annuncio di un nuovo canale pubblico alla rete in modo che possa essere utilizzato per scopi di instradamento.

Sincronizzazione del Grafo dei Canali

Il processo di download e verifica del grafo dei canali.

Nota come i messaggi che appartengono alla stessa categoria in genere condividono anche un *tipo di messaggio* adiacente. Questo viene fatto appositamente per raggruppare messaggi semanticamente simili all'interno della specifica stessa.

Struttura del Messaggio

Descriviamo ora in dettaglio ciascuna categoria di messaggio al fine di definire la struttura e la semantica precise di tutti i messaggi definiti all'interno del protocollo LN.

Messaggi di Creazione della Connessione

Sono i primissimi messaggi inviati tra peer una volta stabilita una connessione. Al momento della stesura di questo capitolo, esiste un solo messaggio all'interno di questa categoria, il messaggio `init`. Il messaggio `init` viene inviato da *entrambi* i partner una volta che è stata stabilita una connessione per la prima volta. Nessun altro messaggio deve essere inviato prima che il messaggio `init` sia stato inviato da entrambe le parti.

Il messaggio init

La struttura del messaggio `init` è definita come segue:

- Tipo: 16
- Campi:
  - `uint16: global_features_len`
  - `global_features_len*byte: global_features`
  - `uint16: features_len`
  - `features_len*byte: features`
  - `tlv_stream_tlvs`

Strutturalmente, il messaggio `init` è composto da due fette di byte di dimensioni variabili che memorizzano ciascuna una serie di *bit di funzionalità*. I feature bit sono una primitiva usata all'interno del protocollo per pubblicizzare l'insieme di caratteristiche del protocollo che un nodo comprende (caratteristiche opzionali) o richiede (caratteristiche richieste).

Nota che le moderne implementazioni dei nodi utilizzeranno solo il campo `features`, con elementi che risiedono all'interno del vettore *global_features* principalmente per motivazioni *storiche* (compatibilità con le versioni precedenti).

Ciò che segue dopo il messaggio principale è una serie di record TLV (Type-Length-Value) che possono essere utilizzati per estendere il messaggio in modo compatibile con le versioni precedenti e successive in futuro. Tratteremo cosa sono i record TLV e come vengono utilizzati più avanti in questa appendice.

Un messaggio `init` viene quindi esaminato da un peer per determinare se la connessione è ben definita in base all'insieme di bit di funzionalità facoltativi e richiesti annunciati da

entrambe le parti.

Una funzionalità opzionale significa che un peer conosce una funzionalità, ma non la considera fondamentale per il funzionamento di una nuova connessione. Un esempio potrebbe essere qualcosa come la capacità di comprendere la semantica di un campo appena aggiunto a un messaggio esistente.

D'altra parte, le funzionalità richieste indicano che se l'altro peer non è a conoscenza della funzionalità, la connessione non è ben definita. Un esempio di tale funzionalità potrebbe essere un nuovo tipo di canale teorico all'interno del protocollo: se il tuo partner non è a conoscenza di questa funzionalità, allora non vuoi mantenere la connessione perché non è in grado di aprire il tuo nuovo tipo di canale preferito .

Messaggi di Comunicazione di Errore

I messaggi in questa categoria vengono utilizzati per inviare errori a livello di connessione tra due peer. Esiste un altro tipo di errore nel protocollo: un errore di livello di inoltro HTLC. Gli errori a livello di connessione possono segnalare cose come l'incompatibilità dei bit di funzionalità o l'intenzione di *forzare la chiusura* (trasmissione unilaterale dell'ultimo impegno firmato).

Il messaggio di errore

L'unico messaggio in questa categoria è il messaggio `error`.

- Tipo: 17
- Campi:
  - `channel_id : chan_id`
  - `uint16 : data_len`
  - `data_len*byte : data`

Un messaggio `error` può essere inviato all'interno dell'ambito di un particolare canale impostando `channel_id` sul `channel_id` del canale sottoposto a questo nuovo stato di errore. In alternativa, se l'errore si applica alla connessione in generale, il campo `channel_id` deve essere impostato su una serie di zeri. Questo `channel_id` composto da

soli zeri è anche noto come identificatore del livello di connessione per un errore.

A seconda della natura dell'errore, l'invio di un `error` a un peer con cui hai un canale può indicare che il canale non può continuare senza intervento manuale, quindi l'unica opzione a quel punto è forzarne la chiusura trasmettendo l'ultimo stato di impegno del canale.

Vivacità della Connessione

I messaggi in questa sezione vengono utilizzati per verificare se una connessione è ancora attiva o meno. Poiché il protocollo LN astrae in qualche modo dal trasporto sottostante utilizzato per trasmettere i messaggi, viene definito un insieme di messaggi `ping` e `pong` a livello di protocollo.

Il messaggio ping

Il messaggio `ping` viene utilizzato per verificare se l'altra parte in una connessione è "attiva". Contiene i seguenti campi:

- Tipo: 18
- Campi:
  - `uint16 : num_pong_bytes`
  - `uint16 : ping_body_len`
  - `ping_body_len*bytes : ping_body`

Successivamente arriva il suo compagno, il messaggio `pong`...

Il messaggio pong

Il messaggio `pong` viene inviato in risposta al messaggio `ping` e contiene i seguenti campi:

- Tipo: 19
- Campi:
  - `uint16 : pong_body_len`
  - `ping_body_len*bytes : pong_body`

Un messaggio `ping` può essere inviato da entrambe le parti in qualsiasi momento.

Il messaggio `ping` include un campo `num_pong_bytes` che viene utilizzato per istruire il nodo ricevente rispetto a quanto è grande il payload che invia nel suo messaggio `pong`. Il messaggio `ping` include anche un set opaco di byte `ping_body` che può essere tranquillamente ignorato. Serve solo a consentire a un mittente di riempire i messaggi `ping` che invia, il che può essere utile nel tentativo di contrastare determinate tecniche di de-anonimizzazione basate sulle dimensioni dei pacchetti sulla rete.

Un messaggio `pong` deve essere inviato in risposta a un messaggio `ping` ricevuto. Il ricevitore dovrebbe leggere un set di `num_pong_bytes` byte casuali da inviare come campo `pong_body`. L'uso intelligente di questi campi/messaggi può consentire a un nodo di instradamento attento alla privacy di tentare di contrastare determinate classi di tentativi di de-anonimizzazione della rete perché possono creare una trascrizione "falsa" che assomiglia ad altri messaggi in base alle dimensioni dei pacchetti inviati. Ricorda che per impostazione predefinita la rete Lightning utilizza un trasporto crittografato, quindi un monitor di rete passivo non può leggere i byte di testo in chiaro e quindi ha solo tempi e dimensioni dei pacchetti da cui partire.

Finanziamento del Canale

Man mano che procediamo, entriamo nel territorio dei messaggi fondamentali che governano la funzionalità e la semantica del protocollo Lightning. In questa sezione, esploriamo i messaggi inviati durante il processo di creazione di un nuovo canale. Descriveremo solo i campi utilizzati, lasciando un'analisi approfondita del processo di finanziamento nel Capitolo 7.

I messaggi inviati durante il flusso di finanziamento del canale appartengono al seguente set di cinque messaggi: `open_channel`, `accept_channel`, `funding_created`, `funding_signed` e `funding_locked`.

Il flusso di protocollo dettagliato che utilizza questi messaggi è descritto nel Capitolo 7.

Il messaggio open_channel

Il messaggio `open_channel` avvia il processo di finanziamento del canale e contiene i seguenti campi:

- Tipo: 32
- Campi:
  - `chain_hash : chain_hash`
  - `32*byte : temp_chan_id`
  - `uint64 : funding_satoshis`
  - `uint64 : push_msat`
  - `uint64 : dust_limit_satoshis`
  - `uint64 : max_htlc_value_in_flight_msat`
  - `uint64 : channel_reserve_satoshis`
  - `uint64 : htlc_minimum_msat`
  - `uint32 : feerate_per_kw`
  - `uint16 : to_self_delay`
  - `uint16 : max_accepted_htlcs`
  - `pubkey : funding_pubkey`
  - `pubkey : revocation_basepoint`
  - `pubkey : payment_basepoint`
  - `pubkey : delayed_payment_basepoint`
  - `pubkey : htlc_basepoint`
  - `pubkey : first_per_commitment_point`
  - `byte : channel_flags`
  - `tlv_stream : tlvs`

Questo è il primo messaggio inviato quando un nodo desidera eseguire un nuovo flusso di finanziamento con un altro nodo. Questo messaggio contiene tutte le informazioni necessarie affinché entrambi i peer costruiscano sia la transazione di finanziamento che la transazione di impegno.

Al momento della stesura di questo capitolo, un singolo record TLV è definito all'interno dell'insieme di record TLV opzionali che possono essere aggiunti alla fine di un messaggio:

- Tipo: 0
- Dati: `upfront_shutdown_script`

L'`upfront_shutdown_script` è una porzione di byte di dimensioni variabili che deve essere uno script di chiave pubblica valido accettato dall'algoritmo di consenso della rete Bitcoin. Fornendo tale indirizzo, il mittente è in grado di creare efficacemente un "circuito chiuso" per il proprio canale, poiché nessuna delle due parti firmerà una transazione di chiusura cooperativa che paga a qualsiasi altro indirizzo. In pratica, questo indirizzo è solitamente quello derivato da un cold wallet (portafoglio freddo).

Il campo `channel_flags` è un bitfield di cui, al momento in cui scriviamo, solo il *primo* bit ha un significato. Se questo bit è impostato, questo canale deve essere annunciato alla rete pubblica come canale instradabile. In caso contrario, il canale è considerato non annunciato, comunemente indicato anche come canale privato.

Il messaggio accept_channel

Il messaggio `accept_channel` è la risposta al messaggio `open_channel`.

- Tipo: 33
- Campi:
  - `32*byte : temp_chan_id`
  - `uint64 : dust_limit_satoshis`
  - `uint64 : max_htlc_value_in_flight_msat`
  - `uint64 : channel_reserve_satoshis`
  - `uint64 : htlc_minimum_msat`
  - `uint32 : minimum_depth`
  - `uint16 : to_self_delay`
  - `uint16 : max_accepted_htlcs`
  - `pubkey : funding_pubkey`
  - `pubkey : revocation_basepoint`
  - `pubkey : payment_basepoint`
  - `pubkey : delayed_payment_basepoint`
  - `pubkey : htlc_basepoint`
  - `pubkey : first_per_commitment_point`
  - `tlv_stream : tlvs`

Il messaggio `accept_channel` è il secondo messaggio inviato durante il processo del flusso di finanziamento. Serve per riconoscere l'intenzione di aprire un canale con un nuovo peer remoto. Il messaggio fa principalmente eco all'insieme di parametri che il risponditore desidera applicare alla propria versione della transazione di impegno. Nel Capitolo 7, quando entriamo nel dettaglio del processo di finanziamento, esploriamo le implicazioni dei vari parametri che possono essere impostati quando si apre un nuovo canale.

Il messaggio funding_created

In risposta, l'iniziatore invierà il messaggio `funding_created`.

- Tipo: 34
- Campi:
  - `32*byte : temp_chan_id`
  - `32*byte : funding_txid`
  - `uint16 : funding_output_index`
  - `sig : commit_sig`

Una volta che l'iniziatore di un canale riceve il messaggio `accept_channel` dal risponditore, dispone di tutto ciò di cui ha bisogno per costruire la transazione di impegno, così come la transazione di finanziamento. Poiché i canali per impostazione predefinita sono finanziatori singoli (solo una parte impegna fondi), solo l'iniziatore deve costruire la transazione di finanziamento. Di conseguenza, per consentire al risponditore di firmare una versione di una transazione di impegno per l'iniziatore, l'iniziatore deve solo inviare l'outpoint di finanziamento del canale.

Il messaggio funding_signed

Per concludere, il risponditore invia il messaggio `funding_signed`.

- Tipo: 34
- Campi:

- `channel_id : channel_id`
- `sig : signature`

Per concludere, dopo che il risponditore riceve il messaggio `funding_created`, ora possiede una firma valida della transazione di impegno da parte dell'iniziatore. Con questa firma sono in grado di uscire dal canale in qualsiasi momento firmando la loro metà dell'output di finanziamento multisig e trasmettendo la transazione. Questo è indicato come una chiusura forzata. Al contrario, per dare all'iniziatore la possibilità di chiudere il canale, il risponditore firma anche la transazione di impegno dell'iniziatore.

Una volta che questo messaggio è ricevuto dall'iniziatore, è sicuro trasmettere la transazione di finanziamento perché ora è in grado di uscire unilateralmente dall'accordo sul canale.

Il messaggio funding_locked

Una volta che la transazione di finanziamento ha ricevuto conferme sufficienti, viene inviato il messaggio funding_locked.

- Tipo: 36
- Campi:
  - `channel_id : channel_id`
  - `pubkey : next_per_commitment_point`

Una volta che la transazione di finanziamento ottiene un numero minimo di conferme, il messaggio `funding_locked` deve essere inviato da entrambe le parti. Solo dopo che questo messaggio è stato ricevuto e inviato, il canale può iniziare ad essere utilizzato.

Chiusura del Canale

La chiusura del canale è un processo in più fasi. Un nodo inizia inviando il messaggio di arresto. I due partner di canale si scambiano quindi una serie di messaggi `closing_signed` per negoziare tariffe reciprocamente accettabili per la transazione di chiusura. Il finanziatore del canale invia il primo messaggio `closing_signed` e l'altra parte può accettare inviando

un messaggio `closing_signed` con gli stessi valori di commissione.

Il messaggio shutdown

Il messaggio `shutdown` avvia il processo di chiusura di un canale e contiene i seguenti campi:

- Tipo: 38
- Campi:
  - `channel_id : channel_id`
  - `u16 : len`
  - `len*byte : scriptpubkey`

Il messaggio closing_signed

Il messaggio `closing_signed` viene inviato da ciascun partner di canale fino a quando non concordano le tariffe. Contiene i seguenti campi:

- Tipo: 39
- Campi:
  - `channel_id : channel_id`
  - `u64 : fee_satoshis`
  - `signature : signature`

Operazione del Canale

In questa sezione, descriviamo brevemente l'insieme di messaggi utilizzati per consentire ai nodi di gestire un canale. Per operazione intendiamo la possibilità di inviare, ricevere e inoltrare pagamenti per un determinato canale.

Per inviare, ricevere o inoltrare un pagamento su un canale, è necessario prima aggiungere un HTLC a entrambe le transazioni di impegno che comprendono un collegamento al canale.

Il messaggio update_add_htlc

Il messaggio `update_add_htlc` consente a entrambe le parti di aggiungere un nuovo HTLC alla transazione di impegno opposta.

- Tipo: 128
- Campi:
  - `channel_id : channel_id`
  - `uint64 : id`
  - `uint64 : amount_msat`
  - `sha256 : payment_hash`
  - `uint32 : cltv_expiry`
  - `1366*byte : onion_routing_packet`

L'invio di questo messaggio consente a una delle parti di avviare l'invio di un nuovo pagamento o l'inoltro di un pagamento esistente arrivato tramite un canale in entrata. Il messaggio specifica l'importo (`amount_msat`) insieme all'hash del pagamento che sblocca il pagamento stesso. Il set di istruzioni di inoltro dell'hop successivo è cifrato onion all'interno del campo `onion_routing_packet`. Nel Capitolo 10, nella parte dell'inoltro di HTLC multihop, trattiamo in dettaglio il protocollo di onion routing utilizzato su Lightning Network.

Nota che ogni HTLC inviato usa un ID a incremento automatico che viene utilizzato da qualsiasi messaggio che modifica un HTLC (regolazione o annullamento) per fare riferimento all'HTLC in un modo univoco e limitato al canale.

Il messaggio update_fulfill_hltc

Il messaggio `update_fulfill_hltc` consente il riscatto (ricevuta) di un HTLC attivo.

- Tipo: 130
- Campi:
  - `channel_id : channel_id`
  - `uint64 : id`
  - `32*byte : payment_preimage`

Questo messaggio viene inviato dal ricevitore dell'HTLC al proponente per riscattare un HTLC attivo. Il messaggio fa riferimento all'`id` dell'HTLC in questione e fornisce anche la preimmagine (che sblocca l'HLTC).

Il messaggio update_fail_htlc

Il messaggio `update_fail_htlc` viene inviato per rimuovere un HTLC da una transazione di impegno.

- Tipo: 131
- Campi:
  - `channel_id : channel_id`
  - `uint64 : id`
  - `uint16 : len`
  - `len*byte : reason`

Il messaggio `update_fail_htlc` è l'opposto del messaggio `update_fulfill_hltc` in quanto consente al destinatario di un HTLC di rimuovere lo stesso. Questo messaggio viene in genere inviato quando un HTLC non può essere instradato correttamente a monte e deve essere rispedito al mittente. Il messaggio contiene un *motivo* dell'errore crittografato (`reason`) che può consentire al mittente di modificare il proprio percorso di pagamento o terminare se l'errore stesso è terminale.

Il messaggio commitment_signed

Il messaggio `commitment_signed` viene utilizzato per contrassegnare la creazione di una nuova transazione di impegno.

- Tipo: 132
- Campi:
  - `channel_id : channel_id`
  - `sig : signature`
  - `uint16 : num_htlcs`
  - `num_htlcs*sig : htlc_signature`

Oltre a inviare una firma per la successiva transazione di impegno, il mittente di questo messaggio deve anche inviare una firma per ogni HTLC presente nella transazione di impegno.

Il messaggio revoke_and_ack

Il `revoke_and_ack` viene inviato per revocare un impegno datato.

- Tipo: 133
- Campi:
  - `channel_id : channel_id`
  - `32*byte : per_commitment_secret`
  - `pubkey : next_per_commitment_point`

Poiché Lightning Network utilizza una transazione di impegno sostitutiva con revoca, dopo aver ricevuto una nuova transazione di impegno tramite il messaggio `commit_sig`, una parte deve revocare l'impegno precedente prima di poterne ricevere un altro. Durante la revoca di una transazione di impegno, il revocatore fornisce anche il punto di impegno successivo necessario per consentire all'altra parte di inviare loro un nuovo stato di impegno.

Il messaggio update_fee

L'`update_fee` viene inviato per aggiornare la commissione sulle transazioni di impegno correnti.

- Tipo: 134
- Campi:
  - `channel_id : channel_id`
  - `uint32 : feerate_per_kw`

Questo messaggio può essere inviato solo dall'iniziatore del canale; sono loro che

pagheranno la commissione di impegno del canale finché sarà aperto.

Il messaggio update_fail_malformed_htlc

L'`update_fail_malformed_htlc` viene inviato per rimuovere un HTLC danneggiato.

- Tipo: 135
- Campi:
  - `channel_id : channel_id`
  - `uint64 : id`
  - `sha256 : sha256_of_onion`
  - `uint16 : failure_code`

Questo messaggio è simile al messaggio `update_fail_htlc`, ma nella pratica è usato raramente. Come accennato in precedenza, ogni HTLC trasporta un pacchetto cifrato via onion routing che copre anche l'integrità di porzioni dell'HTLC stesso. Se una parte riceve un pacchetto onion che è stato in qualche modo corrotto lungo il percorso, non sarà in grado di decifrarlo. Di conseguenza, non può nemmeno inoltrare correttamente l'HTLC; pertanto, invierà questo messaggio per indicare che l'HTLC è stato danneggiato da qualche parte lungo il percorso di ritorno al mittente.

Annuncio del canale

I messaggi in questa categoria vengono utilizzati per annunciare i componenti della struttura dei dati autenticati del grafo dei canali alla rete. Il grafo dei canali ha una serie di proprietà uniche dovute alla condizione che tutti i dati aggiunti al grafo devono essere ancorati alla blockchain di Bitcoin. Di conseguenza, per aggiungere una nuova voce al grafo, è necessaria una commissione di transazione on-chain. Questo funge da deterrente antispam naturale.

Il messaggio channel_announcement

Il messaggio `channel_announcement` viene utilizzato per annunciare un nuovo canale.

- Tipo: 256

- Campi:
  - `sig : node_signature_1`
  - `sig : node_signature_2`
  - `sig : bitcoin_signature_1`
  - `sig : bitcoin_signature_2`
  - `uint16 : len`
  - `len*byte : features`
  - `chain_hash : chain_hash`
  - `short_channel_id : short_channel_id`
  - `pubkey : node_id_1`
  - `pubkey : node_id_2`
  - `pubkey : bitcoin_key_1`
  - `pubkey : bitcoin_key_2`

La serie di firme e chiavi pubbliche nel messaggio serve a creare una *prova* che il canale esista effettivamente nella blockchain di Bitcoin. Ogni canale è identificato in modo univoco da un localizzatore che codifica la sua *posizione* all'interno della blockchain. Tale localizzatore è chiamato `short_channel_id` e può rientrare in un numero intero a 64 bit.

Il messaggio node_announcement

Il messaggio `node_announcement` consente a un nodo di annunciare/aggiornare il proprio vertice all'interno del grafo dei canali.

- Tipo: 257
- Campi:
  - `sig : signature`
  - `uint64 : flen`
  - `flen*byte : features`
  - `uint32 : timestamp`
  - `pubkey : node_id`
  - `3*byte : rgb_color`

    - `32*byte : alias`
    - `uint16 : addrlen`
    - `addrlen*byte : addresses`

Tieni presente che se un nodo non ha alcun canale pubblicizzato all'interno del grafo dei canali, questo messaggio viene ignorato per garantire che l'aggiunta di un elemento al grafo abbia un costo on-chain. In questo caso, il costo on-chain sarà il costo di creazione del canale a cui è connesso questo nodo.

Oltre a pubblicizzare il suo set di funzionalità, questo messaggio consente anche a un nodo di annunciare/aggiornare il set di `addresses` (indirizzi) di rete dove può essere raggiunto.

Il messaggio channel_update

Il messaggio `channel_update` viene inviato per aggiornare le proprietà e le policy di un bordo del canale attivo all'interno del grafo dei canali.

- Tipo: 258
- Campi:
    - `signature : signature`
    - `chain_hash : chain_hash`
    - `short_channel_id : short_channel_id`
    - `uint32 : timestamp`
    - `byte : message_flags`
    - `byte : channel_flags`
    - `uint16 : cltv_expiry_delta`
    - `uint64 : htlc_minimum_msat`
    - `uint32 : fee_base_msat`
    - `uint32 : fee_proportional_millionths`
    - `uint16 : htlc_maximum_msat`

Oltre a poter abilitare/disabilitare un canale, questo messaggio consente a un nodo di aggiornare le sue tariffe di instradamento e altri campi che determinano il tipo di pagamento che può fluire attraverso questo canale.

Il messaggio advertise_signatures

Il messaggio *advertise_signatures* viene scambiato dai peer di canale per assemblare l'insieme di firme necessarie per produrre un messaggio `channel_announcement`.

- Tipo: 259
- Campi:
  - `channel_id : channel_id`
  - `short_channel_id : short_channel_id`
  - `sig : node_signature`
  - `sig : bitcoin_signature`

Dopo che il messaggio `funding_locked` è stato inviato, se entrambe le parti desiderano annunciare il proprio canale sulla rete, ciascuna di esse invierà il messaggio `announce_signatures` che consente a entrambe le parti di posizionare le quattro firme necessarie per generare un messaggio `announce_signatures`.

Sincronizzazione del Grafo dei Canali

I nodi creano una prospettiva locale del grafico del canale utilizzando cinque messaggi: `query_short_chan_ids`, `reply_short_chan_ids_end`, `query_channel_range`, `reply_channel_range` e `gossip_timestamp_range`.

Il messaggio query_short_chan_ids

Il messaggio `query_short_chan_ids` consente a un peer di ottenere le informazioni sul canale relative a una serie di ID canale brevi.

- Tipo: 261
- Campi:
  - `chain_hash : chain_hash`
  - `u16 : len`

- `len*byte : encoded_short_ids`
- `query_short_channel_ids_tlvs : tlvs`

Come apprendiamo nel Capitolo 11, questi ID canale possono essere una serie di canali nuovi per il mittente o non aggiornati, il che consente al mittente di ottenere l'ultimo set di informazioni per un insieme di canali.

Il messaggio reply_short_chan_ids_end

Il messaggio `reply_short_chan_ids_end` viene inviato dopo che un peer ha finito di rispondere a un precedente messaggio `query_short_chan_ids`.

- Tipo: 262
- Campi:
  - `chain_hash : chain_hash`
  - `byte : full_information`

Questo messaggio segnala alla parte ricevente che se desidera inviare un altro messaggio di richiesta, ora può farlo.

Il messaggio query_channel_range

Il messaggio `query_channel_range` consente a un nodo di interrogare l'insieme di canali aperti all'interno di un intervallo di blocchi.

- Tipo: 263
- Campi:
  - `chain_hash : chain_hash`
  - `u32 : first_blocknum`
  - `u32 : number_of_blocks`
  - `query_channel_range_tlvs : tlvs`

Poiché i canali sono rappresentati utilizzando un ID canale breve che codifica la posizione di

un canale on-chain, un nodo sulla rete può utilizzare un'altezza del blocco come una sorta di *cursore* per cercare nella catena al fine di scoprire un insieme di canali appena aperti .

Il messaggio reply_channel_range

Il messaggio `reply_channel_range` è la risposta al messaggio `query_channel_range` e include il set di ID canale brevi per i canali noti all'interno di tale intervallo.

- TIpo: 264
- Campi:
  - `chain_hash : chain_hash`
  - `u32 : first_blocknum`
  - `u32 : number_of_blocks`
  - `byte : sync_complete`
  - `u16 : len`
  - `len*byte : encoded_short_ids`
  - `reply_channel_range_tlvs : tlvs`

In risposta a `query_channel_range`, questo messaggio restituisce il set di canali che sono stati aperti all'interno di tale intervallo. Questo processo può essere ripetuto con il richiedente che fa avanzare il cursore più in basso nella catena per continuare a sincronizzare il grafo dei canali.

Il messaggio gossip_timestamp_range

Il messaggio `gossip_timestamp_range` consente a un peer di iniziare a ricevere nuovi messaggi di gossip in arrivo sulla rete.

- Tipo: 265
- Campi:
  - `chain_hash : chain_hash`
  - `u32 : first_timestamp`
  - `u32 : timestamp_range`

Una volta che un peer ha sincronizzato il grafo dei canali, può inviare questo messaggio se desidera ricevere aggiornamenti in tempo reale sui cambiamenti nel grafo dei canali. Può anche impostare i campi `first_timestamp` e `timestamp_range` se desidera ricevere un arretrato di aggiornamenti che potrebbero essersi persi mentre era inattivo.

Appendice D. Fonti e Avvisi di licenza

Questa appendice contiene gli avvisi di attribuzione e licenza per il materiale incluso con il permesso concesso tramite licenze aperte.

Fonti

Il materiale è stato acquisito da varie fonti pubbliche e con licenze aperte:

- ION Lightning Network Wiki
- "Lightning 101: What Is a Lightning Invoice?" by Suredbits
- Lightning Network In-Progress Specifications GitHub; Creative Commons Attribution (CC-BY 4.0)
- Wikipedia page, "Elliptic-curve Diffie–Hellman"
- Wikipedia page, "Digital signature"
- Wikipedia page, "Cryptographic hash function"
- Wikipedia page, "Onion routing"
- Wikimedia Commons, "Lightning Network Protocol Suite"
- Wikimedia Commons, "Introduction to the Lightning Network Protocol and the Basics of Lightning Technology"

BTCPay Server

Logo, schermate e altre immagini di BTCPay Server sono utilizzate con il permesso della licenza MIT:

MIT License

Lamassu Industries AG

Le immagini del *Gaia* Bitcoin ATM sono utilizzate con il permesso di Lamassu Industries AG. L'uso di queste immagini non è un'approvazione del prodotto o dell'azienda, ma viene fornito come esempio visivo di un ATM Bitcoin.

Glossario

Questo rapido glossario contiene molti dei termini utilizzati in relazione a Bitcoin e LN. Questi termini sono usati in tutto il libro, quindi aggiungilo ai segnalibri per un rapido riferimento.

indirizzo

Gli indirizzi Bitcoin codificano in modo compatto le informazioni necessarie per pagare un destinatario. Un indirizzo è costituito da una stringa di lettere e numeri che inizia con bc1 e assomiglia a `bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4`. Un indirizzo è una scorciatoia per lo script di blocco di un destinatario, che può essere utilizzato da un mittente per firmare fondi verso il destinatario. La maggior parte degli indirizzi rappresenta la chiave pubblica del destinatario o una qualche forma di script che definisce condizioni di spesa più complesse. L'esempio precedente è un indirizzo moderno bech32 che codifica un witness che blocca i fondi sull'hash di una chiave pubblica (vedi *Pay-to-Witness-Public-Key-Hash*). Esistono anche formati di indirizzi meno recenti che iniziano con 1 o 3 che utilizzano la codifica dell'indirizzo Base58Check per rappresentare hash di chiavi pubbliche o hash di script.

asymmetric cryptographic system (sistema crittografico asimmetrico)

La crittografia asimmetrica, o crittografia a chiave pubblica, è un sistema crittografico che utilizza coppie di chiavi: chiavi pubbliche che possono essere ampiamente diffuse e chiavi private note solo al proprietario. La generazione di tali chiavi dipende da algoritmi crittografici basati su problemi matematici per produrre funzioni facili da risolvere in un modo, ma molto difficili da risolvere al contrario. Una sicurezza efficace richiede solo di mantenere privata la chiave privata; la chiave pubblica può essere distribuita apertamente senza compromettere la sicurezza.

autopilot

Un pilota automatico è un motore di raccomandazione per i nodi Lightning che utilizza le statistiche della topologia di LN per suggerire con quali nodi aprire canali. A seconda dell'implementazione dell'autopilot, potrebbe essere consigliata anche la capacità del canale. Un autopilot non fa parte del protocollo LN.

balance (saldo)

Il saldo di un canale di pagamento è la quantità di bitcoin che appartiene a ciascun partner di canale. Ad esempio, Alice potrebbe aprire un canale con Bob per il valore di 1 BTC. Il saldo del canale è quindi di 1 BTC per Alice e di 0 BTC per Bob. Man mano che gli utenti effettuano le transazioni, il saldo del canale verrà aggiornato. Ad esempio, se Alice invia 0.2 BTC a Bob, il saldo ora è di 0.8 BTC per Alice e 0.2 per Bob. Quando il canale viene chiuso, i bitcoin nel canale verranno divisi tra i due partner di canale in base all'ultimo saldo codificato nella transazione di impegno. In Lightning Network, la possibilità di inviare e ricevere pagamenti è limitata dai saldi dei canali. Vedi *capacità*.

bech32

bech32 si riferisce a un formato generico codificato in Base32 con somma di controllo che presenta forti garanzie di rilevamento degli errori. Sebbene bech32 sia stato originariamente sviluppato per essere utilizzato come formato di indirizzo per gli output SegWit nativi (BIP-173), viene utilizzato anche per codificare le fatture Lightning (BOLT #11). Mentre gli output SegWit nativi versione 0 (P2WPKH e P2WSH) utilizzano bech32, le versioni di output SegWit nativi superiori (es. Pay-to-Taproot o P2TR) utilizzano la variante bech32m (BIP-350). gli indirizzi bech32m sono talvolta indicati come indirizzi "bc1", che riflettono il prefisso di tali indirizzi. Gli output SegWit nativi sono più efficienti in termini di spazio rispetto agli indirizzi precedenti e pertanto possono ridurre le commissioni di transazione per il proprietario di tale indirizzo.

Bitcoin Improvement Proposal (BIP)

Una proposta che i membri della comunità hanno presentato per migliorare Bitcoin. Ad esempio, BIP-21 è una proposta per migliorare lo schema URI (Uniform Resource Identifier) di Bitcoin. I BIP sono disponibili su GitHub.

bitcoin, Bitcoin

A seconda del contesto, potrebbe riferirsi al nome dell'unità monetaria (la moneta), alla rete o al protocollo. Scritto con una "b" minuscola si riferisce all'unità monetaria. Bitcoin con una "B" maiuscola di solito si riferisce al protocollo o al sistema.

bitcoin mining

Il mining di Bitcoin è il processo di costruzione di un blocco dalle recenti transazioni e quindi la risoluzione di un problema computazionale richiesto come prova di lavoro (Proof of Work). È il processo mediante il quale il libro mastro condiviso (ovvero la blockchain di Bitcoin) viene aggiornato e mediante il quale le nuove transazioni vengono incluse nel libro mastro. È anche il processo attraverso il quale vengono emessi nuovi bitcoin. Ogni volta che viene creato un nuovo blocco, il nodo di mining riceverà nuovi bitcoin creati all'interno della transazione coinbase di quel blocco.

blocco

Un blocco è una struttura dati nella blockchain che consiste in un'intestazione e un corpo di transazioni Bitcoin. Il blocco è contrassegnato da un timestamp e si lega ad uno specifico blocco predecessore (padre). Quando sottoposto ad hashing, l'intestazione del blocco fornisce la prova del lavoro che rende la blockchain probabilisticamente immutabile. I blocchi devono aderire alle regole applicate dal consenso di rete per estendere la blockchain. Quando un blocco viene aggiunto alla blockchain, si considera che le transazioni incluse abbiano la loro prima conferma.

blockchain

La blockchain è un registro distribuito, o database, di tutte le transazioni Bitcoin. Le transazioni sono raggruppate in aggiornamenti discreti chiamati blocchi, limitati fino a 4 milioni di unità di peso. I blocchi vengono prodotti circa ogni 10 minuti tramite un processo stocastico chiamato mining. Ogni blocco include una "prova di lavoro" computazionalmente intensiva. Il requisito della prova del lavoro viene utilizzato per regolare gli intervalli di blocco e proteggere la blockchain dagli attacchi per riscrivere la cronologia: un utente malintenzionato dovrebbe superare la prova del lavoro

esistente per sostituire i blocchi già pubblicati, rendendo ogni blocco probabilisticamente immutabile poiché è sepolto sotto i blocchi successivi .

BOLT

BOLT, o Basis of Lightning Technology, è la specifica formale della rete Lightning. A differenza di Bitcoin, che ha un'implementazione di riferimento che funge anche da specifica del protocollo, le varie implementazioni LN seguono BOLT in modo che possano lavorare l'una con l'altra per formare la stessa rete. È disponibile su GitHub.

capacità

La capacità di un canale di pagamento è equivalente alla quantità di bitcoin fornita dalla transazione di finanziamento. Poiché la transazione di finanziamento è pubblicamente visibile sulla blockchain e il canale viene annunciato tramite il protocollo di gossip, la capacità è un'informazione pubblica. Non rivela alcuna informazione su quanti bitcoin ciascuno dei partner di canale possiede nel canale, ovvero il saldo. Una capacità elevata non garantisce che il canale possa essere utilizzato per l'instradamento in entrambe le direzioni.

c-lightning

Implementazione del Protocollo LN da parte della società Blockstream con sede a Victoria. È scritto in C. Il codice sorgente è su GitHub.

closing transaction (transazione di chiusura)

Se entrambi i partner di canale accettano di chiudere un canale, creeranno una transazione di liquidazione che riflette la transazione di impegno più recente. Dopo aver scambiato le firme per una transazione di chiusura, non dovrebbero essere effettuati ulteriori aggiornamenti del canale. La chiusura reciproca di un canale con l'aiuto di una transazione di chiusura ha il vantaggio che sono necessarie meno transazioni blockchain per richiedere tutti i fondi, rispetto alla chiusura forzata unilaterale di un canale pubblicando una transazione di impegno. Inoltre, i fondi per

entrambe le parti sono immediatamente spendibili da una transazione di chiusura.

CLTV

CLTV è un acronimo per l'operatore Bitcoin Script OP_CHECKLOCKTIMEVERIFY. Questo definisce un blockheight assoluto prima che un output possa essere speso. L'atomicità del processo di routing dipende fortemente dai valori CLTV negli HTLC. I nodi di instradamento annunciano, tramite il protocollo gossip, i loro delta di scadenza CLTV previsti che desiderano per qualsiasi HTLC in entrata e in uscita.

coinbase

Il coinbase è un campo speciale consentito nel solo input delle transazioni coinbase. La coinbase consente fino a 100 byte di dati arbitrari ma, a partire da BIP-34, deve prima presentare l'altezza del blocco corrente per garantire che le transazioni della coinbase siano univoche. Da non confondere con la transazione coinbase.

coinbase transaction (transazione coinbase)

La prima transazione in un blocco che viene sempre creato da un miner e che include una singola coinbase. La transazione coinbase può richiedere la ricompensa del blocco e assegnarla a uno o più output. La ricompensa del blocco è costituita dal sussidio per il blocco (bitcoin di nuova creazione) e dalla somma di tutte le commissioni di transazione derivanti dalle transazioni incluse nel blocco. Gli output di coinbase possono essere spesi solo dopo essere maturati per 100 blocchi. Se il blocco include transazioni SegWit, la transazione coinbase deve includere un impegno per gli identificatori di transazione witness in un output aggiuntivo.

cold storage

Si riferisce a mantenere una quantità di bitcoin offline. Il cold storage si ottiene quando le chiavi private di Bitcoin vengono create e archiviate in un ambiente offline sicuro. La conservazione a freddo è importante per proteggere le riserve di bitcoin. I computer online sono vulnerabili agli hacker e non dovrebbero essere utilizzati per

archiviare una quantità significativa di bitcoin.

commitment transaction (transazione di impegno)

Una transazione di impegno è una transazione Bitcoin, firmata da entrambi i partner di canale, che codifica l'ultimo saldo di un canale. Ogni volta che viene effettuato o inoltrato un nuovo pagamento utilizzando il canale, il saldo del canale verrà aggiornato e una nuova transazione di impegno verrà firmata da entrambe le parti. È importante sottolineare che, in un canale tra Alice e Bob, sia Alice che Bob mantengono la propria versione della transazione di impegno, anch'essa firmata dall'altra parte. In qualsiasi momento, il canale può essere chiuso da Alice o Bob se inviano la loro transazione di impegno alla blockchain di Bitcoin. L'invio di una transazione di impegno precedente (obsoleta) è considerato imbroglio (ovvero una violazione del protocollo) nel Lightning Network e può essere penalizzato dall'altra parte, rivendicando per sé tutti i fondi nel canale, tramite una transazione di penalità.

conferme

Una volta che una transazione è inclusa in un blocco, ha una conferma. Non appena viene minato un altro blocco sulla blockchain, la transazione ha due conferme e così via. Sei o più conferme sono considerate una prova sufficiente che una transazione non può essere annullata.

contratto

Un contratto è un insieme di transazioni Bitcoin che determinano un certo comportamento desiderato. Esempi sono RSMC per creare un canale di pagamento bidirezionale affidabile o HTLC per creare un meccanismo che consenta l'inoltro affidabile di pagamenti tramite terze parti.

Diffie–Hellman Key Exchange (Scambio di chiavi Diffie-Hellman - DHKE)

In LN viene utilizzato il metodo Elliptic Curve Diffie-Hellman (ECDH). È un protocollo di accordo anonimo che consente a due parti, ciascuna con una coppia di chiavi

pubblica-privata a curva ellittica, di stabilire un segreto condiviso su un canale di comunicazione non sicuro. Questo segreto condiviso può essere utilizzato direttamente come chiave o per derivare un'altra chiave. La chiave, o la chiave derivata, può quindi essere utilizzata per crittografare le comunicazioni successive utilizzando una cifratura a chiave simmetrica. Un esempio della chiave derivata è il segreto condiviso tra la chiave di sessione effimera di un mittente di una cipolla con la chiave pubblica di un nodo hop, come descritto e utilizzato dal formato SPHINX Mix.

digital signature (firma digitale)

Una firma digitale è uno schema matematico per verificare l'autenticità e l'integrità di messaggi o documenti digitali. Può essere visto come un impegno crittografico in cui il messaggio non è nascosto.

double spend (doppia spesa)

La doppia spesa è il risultato di aver speso con successo del denaro più di una volta. Bitcoin protegge dalla doppia spesa verificando che ogni transazione aggiunta alla blockchain aderisca alle regole del consenso; ciò significa verificare che gli input per la transazione non siano stati preventivamente spesi.

Elliptic Curve Digital Signature Algorithm (Algoritmo di firma digitale a curva ellittica - ECDSA)

ECDSA è un algoritmo crittografico utilizzato da Bitcoin per garantire che i fondi possano essere spesi solo dal detentore della chiave privata corretta.

Éclair

Implementazione del Protocollo LN da parte della società parigina ACINQ. È scritto in Scala. Il codice sorgente è su GitHub.

encoding (codifica)

La codifica è il processo di conversione di un messaggio in una forma diversa. Ad

esempio, convertire un numero da decimale a esadecimale.

Electrum server

Un Electrum server è un nodo Bitcoin con un'interfaccia aggiuntiva (API). È spesso richiesto dai portafogli bitcoin che non eseguono un nodo completo. Ad esempio, questi portafogli controllano lo stato di transazioni specifiche o trasmettono transazioni alla mempool utilizzando le API di Electrum server. Alcuni portafogli Lightning utilizzano anche Electrum server.

ephemeral key (chiave effimera)

Le chiavi effimere sono chiavi che vengono utilizzate solo per un breve periodo e non vengono conservate. Sono spesso derivate per l'uso in una sessione da un'altra chiave che viene conservata a lungo termine. Le chiavi effimere vengono utilizzate principalmente all'interno del formato SPHINX Mix e dell'onion routing su LN. Ciò aumenta la sicurezza dei messaggi trasportati o dei pagamenti. Se viene persa una chiave effimera, solo le informazioni su una singola sessione diventano pubbliche.

feature bits (bit di funzionalità)

Una stringa binaria che i nodi Lightning utilizzano per comunicare tra loro quali funzioni supportano. I bit di funzionalità sono inclusi in molti messaggi Lightning e in BOLT #11. Possono essere decodificati utilizzando BOLT #9 e diranno ai nodi quali funzionalità il nodo ha abilitato e se sono compatibili con le versioni precedenti. Sono conosciuti anche come flag di funzionalità (feature flags).

fees (commissioni)

Nel contesto di LN, i nodi addebitano commissioni di instradamento per l'inoltro dei pagamenti di altri utenti. I singoli nodi possono impostare le proprie commissioni che saranno calcolate come la somma di una `base_fee` fissa e una `fee_rate` che dipende dall'importo del pagamento. In Bitcoin, il mittente di una transazione paga una commissione di transazione ai miner per includere la transazione in un blocco. Le

commissioni di transazione Bitcoin non includono una commissione base e dipendono linearmente dal peso della transazione, ma non dall'importo.

funding transaction (transazione di finanziamento)

La transazione di finanziamento viene utilizzata per aprire un canale di pagamento. Il valore (in bitcoin) della transazione di finanziamento è esattamente la capacità del canale di pagamento. L'output della transazione di finanziamento è uno script multifirma 2-di-2 (multisig) in cui ogni partner di canale controlla una chiave. A causa della sua natura, può essere speso solo di comune accordo tra i partner di canale. Alla fine verrà speso da una delle transazioni di impegno o dalla transazione di chiusura.

global features (campo `globalfeatures`)

Le funzionalità globali di un nodo Lightning sono le funzionalità di interesse per tutti gli altri nodi. Più comunemente sono correlati ai formati di routing supportati. Sono annunciati nel messaggio `init` del protocollo peer così come nei messaggi `channel_announcement` e `node_announcement` del protocollo gossip.

gossip protocol (protocollo di gossip)

I nodi Lightning inviano e ricevono informazioni sulla topologia di LN tramite messaggi di gossip che vengono scambiati con i loro peer. Il protocollo gossip è definito principalmente in BOLT #7 e definisce il formato dei messaggi `node_announcement`, `channel_announcement` e `channel_update`. Per prevenire lo spam, i messaggi di annuncio del nodo verranno inoltrati solo se il nodo ha già un canale e i messaggi di annuncio del canale verranno inoltrati solo se la transazione di finanziamento del canale è stata confermata dalla rete Bitcoin. Di solito, i nodi Lightning si connettono con i loro partner di canale, ma va bene connettersi con qualsiasi altro nodo Lightning per elaborare i messaggi di gossip.

hardware wallet (portafoglio hardware)

Un portafoglio hardware è un tipo speciale di portafoglio Bitcoin che memorizza le

chiavi private dell'utente in un dispositivo hardware sicuro. Al momento della stesura del libro, i portafogli hardware non sono disponibili per i nodi LN perché le chiavi utilizzate da Lightning devono essere online per partecipare al protocollo.

hash

Un'impronta digitale di dimensioni fisse di un input binario di lunghezza arbitraria. Conosciuto anche come digest.

hash-based message authentication code (HMAC)

HMAC è un algoritmo per verificare l'integrità e l'autenticità di un messaggio basato su una funzione hash e una chiave crittografica. Viene utilizzato nell'instradamento onion per garantire l'integrità di un pacchetto a ogni salto, nonché all'interno della variante del Noise Protocol utilizzata per la crittografia dei messaggi.

hash function (funzione hash)

Una funzione hash crittografica è un algoritmo matematico che associa dati di dimensioni arbitrarie a una stringa di bit di dimensioni fisse (un hash) ed è progettato per essere una funzione unidirezionale, ovvero una funzione che non è possibile invertire. L'unico modo per ricreare i dati di input dall'output di una funzione hash crittografica ideale è tentare una ricerca a forza bruta di possibili input per vedere se producono una corrispondenza.

hashlock

Un hashlock è una condizione di spesa di Bitcoin Script che limita la spesa di un output fino a quando non viene rivelato un dato specifico. Gli hashlock hanno l'utile proprietà che una volta che un hashlock viene rivelato tramite la spesa, è possibile spendere anche qualsiasi altro hashlock protetto utilizzando la stessa chiave. Ciò rende possibile la creazione di più output che sono tutti gravati dallo stesso hashlock e che diventano tutti spendibili contemporaneamente.

hash time-locked contract (HTLC)

Un contratto hash time-locked (HTLC) è uno script Bitcoin che consiste in hashlock e timelock per richiedere che il destinatario di un pagamento spenda il pagamento prima di una scadenza presentando l'hash della preimmagine o che il mittente possa richiedere un rimborso dopo che il timelock scade. Su Lightning Network, gli HTLC sono output nella transazione di impegno di un canale di pagamento e vengono utilizzati per abilitare l'instradamento affidabile dei pagamenti.

invoice (fattura)

Il processo di pagamento su LN viene avviato dal beneficiario che emette una fattura, nota anche come richiesta di pagamento. Le fatture includono l'hash del pagamento, l'importo, una descrizione e l'ora di scadenza. Le fatture Lightning sono definite in BOLT #11. Le fatture possono anche includere un indirizzo Bitcoin di fallback a cui effettuare il pagamento nel caso in cui non sia possibile trovare un percorso, nonché suggerimenti per instradare un pagamento attraverso un canale privato.

just-in-time (JIT) routing

Il routing just-in-time (JIT) è un'alternativa al routing basato sull'origine, proposto per la prima volta dal coautore René Pickhardt. Con il routing JIT, i nodi intermedi lungo un percorso possono sospendere un pagamento in transito per ribilanciare i propri canali prima di procedere con il pagamento. Ciò potrebbe consentire loro di inoltrare con successo pagamenti che altrimenti sarebbero falliti a causa della mancanza di capacità in uscita.

Lightning message

Un messaggio Lightning è una stringa di dati crittografata che può essere inviata tra due peer su LN. Analogamente ad altri protocolli di comunicazione, i messaggi Lightning sono costituiti da un'intestazione e da un corpo. L'intestazione e il corpo hanno il proprio HMAC. I messaggi Lightning sono l'elemento costitutivo principale del livello di messaggistica.

Lightning Network, Protocollo di Lightning Network, Protocollo Lightning

Il Lightning Network è un protocollo su Bitcoin (o su altre criptovalute). Crea una rete di canali di pagamento che consente l'inoltro senza fiducia dei pagamenti attraverso la rete con l'aiuto di HTLC e onion routing. Altri componenti di Lightning Network sono il protocollo di gossip, il livello di trasporto e le richieste di pagamento.

Lightning Network protocol suite (suite di protocolli di Lightning Network)

La suite di protocolli di LN è composta da cinque livelli responsabili di varie parti del protocollo. Dal basso (il primo livello) all'alto (il quinto livello), sono chiamati livello di comunicazione di rete, livello di messaggistica, livello peer-to-peer, livello di instradamento e livello di pagamento. Vari BOLT definiscono parti di uno o più livelli.

Lightning Network node, Lightning node (Nodo di Lightning Network, nodo Lightning)

Un computer che partecipa a LN, tramite il protocollo peer-to-peer Lightning. I nodi Lightning hanno la capacità di aprire canali con altri nodi, inviare e ricevere pagamenti e instradare pagamenti da altri utenti. In genere, un utente del nodo Lightning eseguirà anche un nodo Bitcoin.

lnd

Implementazione del protocollo LN da parte dell'azienda Lightning Labs con sede a San Francisco. È scritto in Go. Il codice sorgente è su GitHub.

local features (campo: `localfeatures`)

Le caratteristiche locali di un nodo sono caratteristiche configurabili di interesse diretto per i suoi pari. Sono annunciati nel messaggio `init` del protocollo peer e nei messaggi `channel_announcement` e `node_announcement` del protocollo gossip.

locktime

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

alla blockchain.

messaging layer (livello di messaggistica)

Il livello di messaggistica si basa sul livello di connessione di rete della suite di protocolli di Lightning Network. È responsabile di garantire una comunicazione crittografata e sicura e lo scambio di informazioni tramite il protocollo del livello di connessione di rete scelto. Il livello di messaggistica definisce l'inquadratura e il formato dei messaggi Lightning come definito in BOLT #1. Anche i bit delle caratteristiche definiti in BOLT #9 fanno parte di questo livello.

millisatoshi

La più piccola unità di conto su Lightning Network. Un millisatoshi è il centomiliardesimo di un singolo bitcoin. Un millisatoshi è un millesimo di un satoshi. I millisatoshi non esistono, né possono essere stabiliti, sulla rete Bitcoin.

multipart payments (pagamenti in più parti - MPP)

I pagamenti multipart (MPP), spesso indicati anche come pagamenti multipath, sono un metodo per suddividere l'importo del pagamento in più parti e consegnarle lungo uno o più percorsi. Poiché MPP può inviare molte o tutte le parti su un singolo percorso, il termine pagamento in più parti è più accurato del pagamento in più percorsi. In informatica, i pagamenti in più parti sono modellati come flussi di rete.

multisignature (multifirma)

Multisignature (multisig) si riferisce a uno script che richiede più di una firma per autorizzare la spesa. I canali di pagamento sono sempre codificati come indirizzi multisig e richiedono una firma da ciascun partner del canale. Nel caso standard di un canale di pagamento a due parti, viene utilizzato un indirizzo multisig 2-di-2.

nodo

Vedi *Lightning Network node*.

network capacity (capacità di rete)

La capacità LN è la quantità totale di bitcoin bloccati e fatti circolare all'interno della rete. È la somma delle capacità di ciascun canale pubblico. Riflette in una certa misura l'utilizzo di Lightning Network perché ci aspettiamo che le persone inseriscano bitcoin nei canali Lightning per spenderli o inoltrare i pagamenti di altri utenti. Quindi maggiore è la quantità di bitcoin nei canali Lightning, maggiore è l'utilizzo previsto della rete Lightning. Nota che poiché è possibile osservare solo la capacità del canale pubblico, la vera capacità della rete è sconosciuta. Inoltre, poiché la capacità di un canale può consentire un numero illimitato di pagamenti avanti e indietro, la capacità della rete non implica un limite di valore trasferito sulla rete Lightning.

network connection layer (livello di connessione di rete)

Il livello più basso della suite di protocolli di Lightning Network. La sua responsabilità è supportare i protocolli Internet come IPv4, IPv6, TOR2 e TOR3 e utilizzarli per stabilire un canale di comunicazione crittografico sicuro come definito in BOLT #8 o per gestire il DNS per il bootstrap della rete come definito in BOLT #10 .

Noise_XK

Il modello del Noise Protocol Framework per stabilire un canale di comunicazione autenticato e crittografato tra due peer della rete Lightning. X significa che nessuna chiave pubblica deve essere conosciuta dall'iniziatore della connessione. K significa che è necessario conoscere la chiave pubblica del destinatario.

onion routing

L'onion routing è una tecnica per la comunicazione anonima su una rete di computer. In una rete onion, i messaggi sono incapsulati in strati di crittografia, analoghi agli strati di una cipolla. I dati crittografati vengono trasmessi attraverso una serie di nodi di rete chiamati onion router, ognuno dei quali sbuccia un singolo strato, scoprendo la

destinazione successiva dei dati. Quando il livello finale viene decrittografato, il messaggio arriva a destinazione. Il mittente rimane anonimo perché ogni intermediario conosce solo i nodi immediatamente precedenti e successivi.

output

L'output di una transazione Bitcoin, chiamato anche output di transazione non speso (UTXO). Un output è una quantità indivisibile di bitcoin che può essere spesa, nonché uno script che definisce quali condizioni devono essere soddisfatte affinché venga speso. Ogni transazione bitcoin consuma output di transazioni precedentemente registrate e crea nuovi output che possono essere spesi successivamente da transazioni successive. Un tipico output di bitcoin richiederà la spesa di una firma, ma gli output possono richiedere l'esecuzione di script più complessi. Ad esempio, uno script multifirma richiede la firma di due o più detentori di chiavi prima che l'output possa essere speso, che è un elemento fondamentale di Lightning Network.

Pay-to-Public-Key-Hash (P2PKH)

P2PKH è un tipo di output che blocca bitcoin sull'hash di una chiave pubblica. Un output bloccato da uno script P2PKH può essere sbloccato (speso) presentando la chiave pubblica corrispondente all'hash e una firma digitale creata dalla chiave privata corrispondente.

Pay-to-Script-Hash (P2SH)

P2SH è un tipo versatile di output che consente l'uso di script Bitcoin complessi. Con P2SH, lo script complesso che dettaglia le condizioni per spendere l'output (script di riscatto) non è presentato nello script di blocco. Invece, il valore è bloccato sull'hash di uno script, che deve essere presentato e realizzato per spendere l'output.

P2SH address (indirizzo P2SH)

Gli indirizzi P2SH sono codifiche Base58Check dell'hash a 20 byte di uno script. Gli indirizzi P2SH iniziano con un "3". Gli indirizzi P2SH nascondono tutta la complessità,

in modo che il mittente di un pagamento non veda lo script.

Pay-to-Witness-Public-Key-Hash (P2WPKH)

P2WPKH è l'equivalente SegWit di P2PKH, utilizzando un testimone segregato (segregated witness). La firma per utilizzare un output P2WPKH viene inserita nel witness tree (albero testimone) anziché nel campo ScriptSig. Vedi *SegWit*.

P2WPKH address (indirizzo P2WPKH)

Il formato dell'indirizzo "native SegWit v0", gli indirizzi P2WPKH sono codificati bech32 e iniziano con "bc1q".

Pay-to-Witness-Script-Hash (P2WSH)

P2WSH è l'equivalente SegWit di P2SH, utilizzando un segregated witness (testimone segregato). La firma e lo script per utilizzare un output P2WSH vengono inseriti nel witness tree (albero testimone) anziché nel campo ScriptSig. Vedi *SegWit*.

P2WSH address (indirizzo P2WSH)

Il formato dell'indirizzo dello script "native Segwit v0", gli indirizzi P2WSH sono codificati bech32 e iniziano con "bc1q".

Pay-to-taroot (P2TR)

Attivato a novembre 2021, Taproot è un nuovo tipo di output che blocca bitcoin a un albero (tree) di condizioni di spesa o a una singola condizione di radice (root).

P2TR address (indirizzo P2TR)

Il formato dell'indirizzo Taproot, che rappresenta SegWit v1, è un indirizzo con codifica bech32m e inizia con "bc1p".

payment (pagamento)

Un pagamento Lightning si verifica quando i bitcoin vengono trasferiti all'interno di LN. I pagamenti generalmente non sono visibili sulla blockchain di Bitcoin.

payment channel (canale di pagamento)

Un canale di pagamento è una relazione finanziaria tra due nodi sulla rete Lightning, creata utilizzando una transazione bitcoin pagando un indirizzo multifirma. I partner di canale possono utilizzare il canale per inviare bitcoin avanti e indietro tra loro senza impegnare tutte le transazioni sulla blockchain di Bitcoin. In un tipico canale di pagamento solo due transazioni, la transazione di finanziamento e la transazione di impegno, vengono aggiunte alla blockchain. I pagamenti inviati attraverso il canale non vengono registrati nella blockchain e si dice che avvengano "off-chain".

payment layer (livello di pagamento)

Il quinto livello superiore della suite di protocolli di Lightning Network che opera sopra il livello di routing. La sua responsabilità è abilitare il processo di pagamento tramite fatture BOLT #11. Sebbene utilizzi pesantemente il grafo dei canali dal protocollo di gossip come definito in BOLT # 7, le strategie effettive per consegnare un pagamento non fanno parte delle specifiche del protocollo e sono lasciate alle implementazioni. Poiché questo argomento è molto importante per garantire l'affidabilità del processo di consegna dei pagamenti, lo abbiamo incluso in questo libro.

peer

I partecipanti a una rete peer-to-peer. Su LN, i peer si connettono tra loro tramite comunicazioni crittografate e autenticate tramite un socket TCP, su IP o Tor.

peer-to-peer layer (livello peer-to-peer)

Il livello peer-to-peer è il terzo livello della suite di protocolli di Lightning Network e funziona al di sopra del livello di messaggistica. È responsabile della definizione della sintassi e della semantica delle informazioni scambiate tra peer tramite messaggi

Lightning. Consiste in messaggi di controllo come definito in BOLT #9; messaggi di creazione, funzionamento e chiusura del canale come definito in BOLT #2; così come i messaggi di gossip e routing come definiti in BOLT #7.

private channel (canale privato)

Un canale non annunciato al resto della rete. Tecnicamente, "privato" è un termine improprio perché questi canali possono ancora essere identificati tramite suggerimenti di instradamento e transazioni di impegno. Andrebbero definiti canali "non annunciati". In un canale non annunciato, i partner di canale possono inviare e ricevere pagamenti normalmente. Tuttavia, il resto della rete non sarà a conoscenza del canale e quindi in genere non può utilizzarlo per instradare i pagamenti. Poiché il numero e la capacità dei canali non annunciati sono sconosciuti, il numero e la capacità totali dei canali pubblici rappresentano solo una parte del totale di LN.

preimage (preimmagine)

Nel contesto della crittografia e in particolare su Lightning Network, la preimage si riferisce all'input di una funzione hash che produce un hash specifico. Non è possibile calcolare la preimmagine dall'hash (le funzioni hash sono unilaterali). Selezionando un valore casuale segreto come preimage e calcolando il suo hash, possiamo impegnarci con quella preimage e successivamente rivelarla. Chiunque può confermare che la preimage rivelata produce correttamente l'hash.

Proof of Work (prova di lavoro - PoW)

Dati che richiedono un calcolo significativo per essere trovati e che possono essere facilmente verificati da chiunque per dimostrare la quantità di lavoro necessaria per produrli. In Bitcoin, i miner devono trovare una soluzione numerica all'algoritmo SHA-256 che soddisfi un obiettivo a livello di rete, chiamato difficulty target (obiettivo di difficoltà). Vedi *Bitcoin mining* per ulteriori informazioni.

Point Time-Locked Contract (contratto Point Time-Locked - PTLC)

Un Point Time-Locked Contract (PTLC) è uno script Bitcoin che consente una spesa condizionale alla presentazione di un segreto o dopo che è trascorso un certo blockheight (altezza di blocco), simile a un HTLC. A differenza degli HTLC, i PTLC non dipendono da una preimmagine di una funzione hash ma piuttosto dalla chiave privata da un punto della curva ellittica. L'ipotesi di sicurezza si basa quindi sul logaritmo discreto.

relative timelock (timelock relativo)

Un timelock relativo è un tipo di timelock che consente a un input di specificare la prima volta che l'input può essere aggiunto a un blocco. Il tempo è relativo e si basa su quando l'output a cui fa riferimento quell'ingresso è stata registrato in un blocco. I timelock relativi sono impostati dal campo della transazione `nSequence` e dal codice operativo Bitcoin Script `CHECKSEQUENCEVERIFY` (CSV), introdotto da BIP-68/112/113.

Revocable Sequence Maturity Contract (RSMC)

Questo contratto viene utilizzato per costruire un canale di pagamento tra due utenti Bitcoin o LN che non devono fidarsi l'uno dell'altro. Il nome deriva da una sequenza di stati che sono codificati come transazioni di impegno e possono essere revocati se pubblicati e estratti erroneamente dalla rete Bitcoin.

revocation key (chiave di revoca)

Ogni RSMC contiene due chiavi di revoca. Ciascun partner di canale conosce una chiave di revoca. Conoscendo entrambe le chiavi di revoca, l'output dell'RSMC può essere speso all'interno del timelock predefinito. Durante la negoziazione di un nuovo stato del canale, le vecchie chiavi di revoca vengono condivise, "revocando" in tal modo il vecchio stato. Le chiavi di revoca vengono utilizzate per scoraggiare i partner di canale dal trasmettere uno stato di canale precedente.

RIPEMD-160

RIPEMD-160 è una funzione crittografica che produce un hash a 160 bit (20 byte).

routing layer (livello di instradamento)

Il quarto livello della suite di protocolli di Lightning Network che opera al di sopra del livello peer-to-peer. La sua responsabilità è definire le primitive crittografiche e il protocollo di comunicazione necessario per consentire il trasporto sicuro e atomico di bitcoin da un nodo mittente a un nodo destinatario. Mentre BOLT #4 definisce il formato onion utilizzato per comunicare le informazioni di trasporto a peer remoti con i quali non esistono connessioni dirette, il trasporto effettivo delle cipolle e le primitive crittografiche sono definite in BOLT #2.

topology (topologia)

La topologia di Lightning Network descrive la forma della rete come un grafo matematico. I nodi del grafo sono i nodi Lightning (partecipanti/peer della rete). I bordi del grafo sono i canali di pagamento. La topologia di LN viene trasmessa pubblicamente con l'ausilio del protocollo di gossip, ad eccezione dei canali non annunciati. Ciò significa che la rete Lightning potrebbe essere significativamente più grande del numero annunciato di canali e nodi. Conoscere la topologia è di particolare interesse nel processo di instradamento dei pagamenti basato sull'origine in cui il mittente scopre un percorso.

satoshi

Un satoshi è la più piccola unità (denominazione) di bitcoin che può essere registrata sulla blockchain. Un satoshi è 1/100 milionesimo (0.00000001) di un bitcoin e prende il nome dal creatore di Bitcoin, Satoshi Nakamoto.

Satoshi Nakamoto

Satoshi Nakamoto è il nome utilizzato dalla persona o dal gruppo di persone che ha progettato Bitcoin e creato la sua implementazione di riferimento originale. Come parte dell'implementazione, hanno anche ideato il primo database blockchain. Nel processo, sono stati i primi a risolvere il problema della doppia spesa per la valuta digitale. La loro vera identità rimane sconosciuta.

Schnorr Signature (firma di Schnorr)

Un nuovo schema di firma digitale attivato in Bitcoin nel novembre 2021. Consente innovazioni sulla rete Lightning, come PTLC efficienti (un miglioramento degli HTLC).

script, Bitcoin Script

Bitcoin utilizza un sistema di scripting per le transazioni chiamato Bitcoin Script. Simile al linguaggio di programmazione Forth, è semplice, basato su stack ed elaborato da sinistra a destra. È volutamente Turing-incompleto, senza loop o ricorsione.

ScriptPubKey (noto anche come pubkey script)

ScriptPubKey o pubkey script, è uno script incluso negli output che imposta le condizioni che devono essere soddisfatte affinché tali output vengano spesi. I dati per soddisfare le condizioni possono essere forniti in uno script di firma. Vedi *ScriptSig*.

ScriptSig (noto anche come signature script)

ScriptSig o signature script sono i dati generati dallo spenditore, che sono quasi sempre usati come variabili per soddisfare uno script pubkey.

secret key / private key (chiave segreta / chiave privata)

Il numero segreto che sblocca i bitcoin inviati all'indirizzo corrispondente. Ha questo aspetto: `5J76sF8L5jTtzE96r66Sf8cka9y44wdpJjMwCxR3tzLh3ibVPxh`

Segregated Witness (Testimone Segregato - SegWit)

Segregated Witness (SegWit) è un aggiornamento del protocollo Bitcoin introdotto nel 2017 che aggiunge un nuovo testimone per le firme e altre prove di autorizzazione delle transazioni. Questo nuovo campo testimone è esente dal calcolo dell'ID transazione, che risolve la maggior parte delle classi di malleabilità delle transazioni di terze parti. SegWit è stato implementato come soft fork ed è un cambiamento che tecnicamente rende le regole del protocollo di Bitcoin più restrittive.

Secure Hash Algorithm (SHA)

Il Secure Hash Algorithm o SHA è una famiglia di funzioni hash crittografiche pubblicate dal National Institute of Standards and Technology (NIST). Il protocollo Bitcoin attualmente utilizza SHA-256, che produce un hash a 256 bit.

short channel ID (ID canale breve - scid)

Una volta stabilito un canale, l'indice della transazione di finanziamento sulla blockchain viene utilizzato come ID canale breve per identificare in modo univoco il canale. L'ID canale breve è composto da otto byte che si riferiscono a tre numeri. Nella sua forma serializzata, raffigura questi tre numeri come valori decimali separati dalla lettera "x" (ad esempio, `600123x01x00`) Il primo numero (4 byte) è l'altezza del blocco. Il secondo numero (2 byte) è l'indice dell'operazione di finanziamento con i blocchi. L'ultimo numero (2 byte) è l'output della transazione.

simplified payment verification (verifica semplificata del pagamento - SPV)

SPV o simplified payment verification è un metodo per verificare che determinate transazioni siano state incluse in un blocco senza scaricare l'intero blocco. Il metodo è utilizzato da alcuni portafogli Bitcoin e Lightning leggeri (light wallet).

source-based routing (instradamento basato sull'origine)

Su Lightning Network, il mittente di un pagamento decide il percorso del pagamento. Sebbene ciò riduca il tasso di successo del processo di instradamento, aumenta la privacy dei pagamenti. A causa del formato SPHINX Mix utilizzato dall'onion routing, tutti i nodi di instradamento non conoscono l'originatore di un pagamento o il destinatario finale. Il routing basato sull'origine è fondamentalmente diverso da come funziona il routing sul protocollo Internet.

soft fork

Soft fork, o modifica soft-forking, è un aggiornamento del protocollo compatibile con le versioni precedenti e successive, quindi consente sia ai vecchi che ai nuovi nodi di

continuare a utilizzare la stessa catena.

SPHINX Mix Format

Una tecnica particolare per l'onion routing utilizzata su LN e inventata da George Danezis e Ian Goldberg nel 2009. Con il formato SPHINX Mix, ogni messaggio del pacchetto onion viene riempito con alcuni dati casuali in modo che nessun singolo hop possa stimare fino a che punto il percorso che ha percorso. Mentre la privacy del mittente e del destinatario del pagamento è protetta, ogni nodo è ancora in grado di restituire un messaggio di errore lungo il percorso verso l'originatore del messaggio.

submarine swaps (scambi sottomarini)

Un submarine swap è uno scambio atomico senza fiducia tra indirizzi Bitcoin on-chain e pagamenti off-chain su Lightning Network. Proprio come i pagamenti LN utilizzano HTLC che subordinano la richiesta finale sui fondi alla rivelazione di un segreto da parte del destinatario (hash preimage), i submarine swap utilizzano lo stesso meccanismo per trasferire fondi attraverso la barriera on-chain/off-chain con una fiducia minima. I reverse submarine swap consentono scambi nella direzione opposta, da un pagamento LN off-chain a un indirizzo Bitcoin on-chain.

timelock

Un timelock è un tipo di condizione che limita la spesa di alcuni bitcoin fino a un tempo futuro specificato o all'altezza del blocco (block height). I timelock hanno un posto di rilievo in molti contratti Bitcoin, inclusi i canali di pagamento e gli HTLC.

transazione

Le transazioni sono strutture dati utilizzate da Bitcoin per trasferire bitcoin da un indirizzo a un altro. Diverse migliaia di transazioni vengono aggregate in un blocco, che viene poi registrato (minato) sulla blockchain. La prima transazione in ogni blocco, chiamata transazione coinbase, genera nuovi bitcoin.

transaction malleability (malleabilità della transazione)

La malleabilità della transazione è una proprietà dove l'hash di una transazione può essere modificato senza modificare la semantica della transazione. Ad esempio, l'alterazione della firma può modificare l'hash di una transazione. Una transazione di impegno richiede l'hash di una transazione di finanziamento e, se l'hash della transazione di finanziamento cambia, le transazioni che dipendono da esso diventeranno non valide. Ciò impedisce agli utenti di richiedere i rimborsi se ce ne sono. Il soft fork Segregated Witness (SegWit) risolve questo problema ed è stato quindi un aggiornamento importante per permettere Lightning Network.

transport layer (livello di trasporto)

Nelle reti di computer, il livello di trasporto è una divisione concettuale dei metodi utilizzati dai computer (e in definitiva dalle applicazioni) per comunicare tra loro. Il livello di trasporto fornisce servizi di comunicazione tra computer, come il controllo del flusso, la verifica e il multiplexing (per consentire a più applicazioni di funzionare contemporaneamente su un computer).

unspent transaction output (UTXO)

Vedi *Output*.

wallet (portafoglio)

Un portafoglio è un software che contiene le chiavi private di Bitcoin. Viene utilizzato per creare e firmare transazioni Bitcoin. Nel contesto di Lightning Network, contiene anche i segreti di revoca del vecchio stato del canale e le ultime transazioni di impegno prefirmate.

watchtower (torre di controllo)

Le watchtower sono un servizio di sicurezza su Lightning Network che monitora i canali di pagamento per potenziali violazioni del protocollo. Se uno dei partner di canale va offline o perde il backup, una torre di controllo conserva i backup e può

ripristinare le informazioni sul canale.

Le watchtower monitorano anche la blockchain di Bitcoin e possono presentare una transazione di penalità se uno dei partner tenta di "imbrogliare" trasmettendo uno stato obsoleto. Le watchtower possono essere gestite dagli stessi partner di canale o come servizio a pagamento offerto da terzi. Le watchtower non hanno alcun controllo sui fondi nei canali stessi.