精通閃電網路

Andreas M. Antonopoulos
Olaoluwa Osuntokun
Rene Pickhardt

關於本書

本書是《Mastering the Lightning Network》的繁體中文翻譯版本。

原書由 Andreas M. Antonopoulos、Olaoluwa Osuntokun 和 Rene Pickhardt 共同撰寫，於 2021 年 12 月由 O’ Reilly Media 出版。

繁體中文翻譯：Dr. Awesome Doge

下載電子書

- EPUB 格式 — 適用於 Apple Books、Kobo、Readmoo 等電子書閱讀器
- PDF 格式 — 適用於列印與桌面閱讀
- 原始碼：https://github.com/lnbook/lnbook[GitHub]

第一部分：基礎篇

1. 簡介

歡迎閱讀《精通閃電網路》！

閃電網路（Lightning Network，常簡稱為 LN）正在改變人們在線上交換價值的方式，它是比特幣歷史上最令人興奮的進展之一。 如今，在 2021 年，閃電網路仍處於起步階段。閃電網路是一種以智慧且非顯而易見的方式使用比特幣的協議。它是建立在比特幣之上的第二層技術。

閃電網路的概念於 2015 年被提出，第一個實作於 2018 年推出。截至 2021 年，我們才剛開始看到閃電網路為比特幣帶來的機會，包括改進的隱私性、速度和可擴展性。 憑藉對閃電網路的核心知識，您可以幫助塑造網路的未來，同時也為自己創造機會。

我們假設您已經具備一些比特幣的基礎知識，但如果沒有，不用擔心——我們將在 比特幣基礎回顧 中解釋最重要的比特幣概念，這些是理解閃電網路所必須知道的。如果您想了解更多關於比特幣的知識，可以閱讀 Andreas M. Antonopoulos 的《Mastering Bitcoin》第二版（O’ Reilly 出版），可在 網路上免費取得 (https://github.com/bitcoinbook/bitcoinbook)。

雖然本書的大部分內容是為程式設計師撰寫的，但前幾章的寫法讓任何人都能理解，無論技術經驗如何。在本章中，我們將從一些術語開始，然後探討信任及其在這些系統中的應用，最後我們將討論閃電網路的歷史和未來。讓我們開始吧。

1.1. 閃電網路基本概念

當我們探索閃電網路的實際運作方式時，我們會遇到一些技術術語，起初可能會有點令人困惑。雖然所有這些概念和術語將在我們閱讀本書的過程中詳細解釋，並在術語表中定義，但現在了解一些基本定義將使接下來兩章的概念更容易理解。如果您還不理解這些定義中的所有詞彙，沒關係。隨著您閱讀文本，您會理解更多。

區塊鏈（Blockchain）

一個由電腦網路產生的分散式交易帳本。例如，比特幣是一個產生區塊鏈的系統。閃電網路本身不是區塊鏈，也不產生區塊鏈。它是一個依賴現有外部區塊鏈來確保其安全性的網路。

數位簽章（Digital signature）

數位簽章是一種用於驗證數位訊息或文件真實性的數學方案。有效的數位簽章讓接收者有理由相信訊息是由已知的發送者創建的，發送者不能否認已發送該訊息，且訊息在傳輸過程中未被篡改。

雜湊函數（Hash function）

加密雜湊函數是一種將任意大小的資料映射到固定大小位元字串（雜湊值）的數學演算法，並被設計為單向函數，即一個在實務上無法反向運算的函數。

節點（Node）

參與網路的電腦。閃電網路節點是參與閃電網路的電腦。比特幣節點是參與比特幣網路的電腦。通常，閃電網路使用者會同時運行閃電網路節點和比特幣節點。

鏈上 vs 鏈下（On-chain versus off-chain）

如果支付被記錄為比特幣（或其他底層）區塊鏈上的交易，則該支付是「鏈上」的。透過閃電網路節點之間的支付通道發送的支付，不會顯示在底層區塊鏈上，這些被稱為「鏈下」支付。通常在閃電網路中，唯一的鏈上交易是用於開啟和關閉閃電網路支付通道的交易。還存在第三種通道修改交易，稱為拼接（splicing），可用於增加/減少通道中承諾的資金量。

支付（Payment）

當價值在閃電網路上交換時，我們稱之為「支付」，相對於比特幣區塊鏈上的「交易」。

支付通道（Payment channel）

閃電網路上兩個節點之間的「財務關係」，通常透過多重簽名比特幣交易實現，在兩個閃電網路節點之間共享對比特幣的控制權。

路由 vs 發送（Routing versus sending）

與比特幣透過向所有人廣播來「發送」交易不同，閃電網路是一個路由網路，支付沿著從發送者到接收者的「路徑」，透過一個或多個支付通道進行「路由」。

交易（Transaction）

記錄對某些資金（例如一些比特幣）控制權轉移的資料結構。閃電網路依賴比特幣交易（或其他區塊鏈的交易）來追蹤資金的控制權。

這些術語以及許多其他術語的更詳細定義可以在 術語表 中找到。在本書中，我們將解釋這些概念的含義以及這些技術的實際運作方式。

> 在本書中，您會看到首字母大寫的「Bitcoin」，這是指「比特幣系統」，是一個專有名詞。您也會看到小寫的「bitcoin」，這是指貨幣單位。每個 bitcoin 進一步細分為 1 億個單位，每個單位稱為「satoshi」（單數）或「satoshis」（複數）。

現在您已經熟悉了這些基本術語，讓我們轉向一個您已經熟悉的概念：信任。

1.2. 去中心化網路中的信任

您經常會聽到人們將比特幣和閃電網路稱為「無需信任的」（trustless）。乍看之下這令人困惑。畢竟，信任不是一件好事嗎？銀行甚至將它用在名字中！一個「無需信任的」系統，一個沒有信任的系統，難道不是一件壞事嗎？

「無需信任」一詞旨在傳達不需要信任系統中其他參與者就能運作的能力。在像比特幣這樣的去中心化系統中，您始終可以選擇與您信任的人進行交易。然而，即使您無法信任交易中的另一方，該系統也能確保您不會被欺騙。信任是系統的一個可有可無的特性，而不是必須具備的特性。

相比之下，傳統系統如銀行則要求您必須信任第三方，因為他們控制著您的資金。如果銀行違背了您的信任，您可能可以從監管機構或法院尋求一些補救，但需要付出巨大的時間、金錢和精力成本。

無需信任並不意味著沒有信任。它意味著信任不是所有交易的必要前提，即使與您不信任的人交易，系統也能防止欺騙。

在我們深入了解閃電網路的運作方式之前，重要的是理解一個構成比特幣、閃電網路和許多其他類似系統基礎的基本概念：我們稱之為「公平協議」。公平協議是一種在不需要信任彼此的參與者之間，無需中央權威機構就能實現公平結果的方式，它是比特幣等去中心化系統的骨幹。

1.3. 無需中央權威的公平

當人們有競爭性利益時，他們如何建立足夠的信任來進行某些合作或交易行為？這個問題的答案位於多個科學和人文學科的核心，如經濟學、社會學、行為心理學和數學。其中一些學科給出了依賴聲譽、公平、道德甚至宗教等概念的「軟性」答案。其他學科則給出了具體的答案，僅依賴於這些互動參與者會理性行事、以自身利益為主要目標的假設。

廣義而言，有幾種方法可以確保可能有競爭性利益的個人之間互動的公平結果：

需要信任

> 您只與您已經信任的人互動，這種信任基於先前的互動、聲譽或家族關係。這在小規模範圍內運作得很好，特別是在家庭和小團體內，它是合作行為最常見的基礎。不幸的是，它無法擴展，而且會受到部落主義（內群體）偏見的影響。

法治

> 建立由機構執行的互動規則。這種方式更具可擴展性，但由於風俗和傳統的差異，以及執法機構無法擴展，它無法在全球範圍內擴展。這種解決方案的一個令人不快的副作用是，隨著機構變得越來越大，它們變得越來越強大，這可能導致腐敗。

受信任的第三方

> 在每次互動中放置一個中介來執行公平。結合「法治」來對中介進行監督，這種方式更具可擴展性，但會遭受同樣的權力失衡問題：中介變得非常強大，可能會吸引腐敗。權力集中會導致系統性風險和系統性失敗（「大到不能倒」）。

賽局理論公平協議

最後一類源自網際網路和密碼學的結合，是本節的主題。讓我們看看它是如何運作的，以及它的優缺點。

1.3.1. 無需中介的受信任協議

像比特幣和閃電網路這樣的密碼學系統是允許您與不信任的人（和電腦）進行交易的系統。這通常被稱為「無需信任」的操作，儘管它實際上並非完全無需信任。您必須信任您運行的軟體，並且您必須信任該軟體實作的協議將產生公平的結果。

這種密碼學系統與傳統金融系統的最大區別在於，在傳統金融中，您有一個「受信任的第三方」，例如銀行，來確保結果公平。這種系統的一個重大問題是它們將太多權力交給了第三方，而且它們也容易受到「單點故障」的影響。如果受信任的第三方本身違背信任或試圖欺騙，信任的基礎就會崩潰。

當您研究密碼學系統時，您會注意到一個特定的模式：這些系統不是依賴受信任的第三方，而是試圖透過使用激勵和懲罰系統來防止不公平的結果。在密碼學系統中，您將信任放在「協議」中，協議實際上是一個具有一組規則的系統，如果設計得當，將正確應用所需的激勵和懲罰。這種方法的優勢是雙重的：您不僅避免了信任第三方，還減少了執行公平結果的需要。只要參與者遵循約定的協議並留在系統內，該協議中的激勵機制就能在不需要強制執行的情況下實現公平結果。

使用激勵和懲罰來實現公平結果是數學一個分支——「賽局理論」的一個方面，它研究「理性決策者之間戰略互動的模型」。[1]像比特幣和閃電網路這樣控制參與者之間金融互動的密碼學系統，嚴重依賴賽局理論來防止參與者欺騙，並讓不信任彼此的參與者實現公平結果。

雖然賽局理論及其在密碼學系統中的使用起初可能看起來令人困惑和陌生，但您很可能在日常生活中已經熟悉這些系統；只是您還沒有認出它們。在下一節中，我們將使用一個童年的簡單例子來幫助我們識別基本模式。一旦您理解了基本模式，您將在區塊鏈領域到處看到它，並會快速直觀地認出它。

在本書中，我們稱這種模式為「公平協議」，定義為使用激勵和/或懲罰系統來確保不信任彼此的參與者獲得公平結果的過程。公平協議的強制執行僅在確保參與者無法逃避激勵或懲罰時才是必要的。

1.3.2. 公平協議實例

讓我們看一個您可能已經熟悉的公平協議例子。

想像一個家庭午餐，有一位家長和兩個孩子。孩子們是挑食者，他們唯一同意吃的東西是炸薯條。家長準備了一碗炸薯條（根據您使用的英語方言，稱為「french fries」或「chips」）。兩個兄弟姐妹必須分享這盤薯條。家長必須確保薯條公平分配給每個孩子；否則，家長將不得不聽不斷的抱怨（可能整天），而且總是有可能不公平的情況升級為暴力。家長該怎麼辦？

在兩個不信任彼此且有競爭性利益的兄弟姐妹之間的這種戰略互動中，有幾種不同的方法可以實現公平。天真但常用的方法是讓家長使用他們作為受信任第三方的權威：他們將薯條分成兩份。這類似於傳統金融，銀行、會計師或律師作為受信任的第三方來防止雙方之間的欺騙。

這種情況的問題是它將大量權力和責任交給了受信任的第三方。在這個例子中，家長完全負責薯條的平等分配，而各方只是等待、觀看和抱怨。孩子們指責家長偏心，沒有公平分配薯條。兄弟姐妹為薯條爭吵，大喊「那塊薯條更大！」並把家長拖入他們的爭吵。聽起來很糟糕，不是嗎？家長應該更大聲地喊？把所有薯條都拿走？威脅再也不做薯條，讓那些不知感恩的孩子挨餓？

存在一個更好的解決方案：教兄弟姐妹玩一個叫做「分割與選擇」的遊戲。每頓午餐，一個兄弟姐妹把薯條分成兩份，另一個兄弟姐妹選擇他們想要哪一份。幾乎立即，兄弟姐妹們就弄清楚了這個遊戲的動態。如果分割的那個人犯了錯誤或試圖欺騙，另一個兄弟姐妹可以通過選擇較大的那碗來「懲罰」他們。讓兩個兄弟姐妹，尤其是分割薯條的那個人，公平地進行遊戲符合他們的最佳利益。在這種情況下，只有欺騙者會輸。家長甚至不必使用他們的權威或強制執行公平。家長所要做的就是「執行協議」；只要兄弟姐妹無法逃脫他們被分配的「分割者」和「選擇者」角色，協議本身就能確保公平結果，而無需任何干預。家長不能偏心或扭曲結果。

> 雖然 1980 年代臭名昭著的薯條戰爭很好地說明了這一點，但前述場景與任何作者與他們表親的實際童年經歷之間的任何相似之處純屬巧合…或者是嗎？

1.3.3. 安全原語作為構建模組

為了讓像這樣的公平協議運作，需要有某些保證，或稱為「安全原語」，可以組合起來確保強制執行。第一個安全原語是「嚴格的時間排序/順序」：「分割」動作必須在「選擇」動作之前發生。這不是立即顯而易見的，但除非您能保證動作 A 在動作 B 之前發生，否則協議就會崩潰。第二個安全原語是「承諾與不可否認性」。每個兄弟姐妹必須承諾他們選擇的角色：分割者或選擇者。此外，一旦分割完成，分割者就承諾了他們創建的分割——他們不能否認該選擇並重新嘗試。

密碼學系統提供了許多安全原語，可以以不同方式組合來構建公平協議。除了順序和承諾之外，我們還可以使用許多其他工具：

- 雜湊函數用於對資料進行指紋識別，作為承諾的一種形式，或作為數位簽章的基礎
- 數位簽章用於身份驗證、不可否認性和秘密所有權證明
- 加密/解密用於將資訊存取限制為僅授權參與者

這只是正在使用的整個安全和密碼學原語「動物園」的一小部分列表。更多基本原語和組合一直在被發明。

在我們的現實生活例子中，我們看到了一種稱為「分割與選擇」的公平協議形式。這只是透過以不同方式組合安全原語構建模組可以構建的無數不同公平協議之一。但基本模式始終相同：兩個或更多參與者在不信任彼此的情況下，透過參與一系列屬於約定協議一部分的步驟來進行互動。協議的步驟安排激勵和懲罰，以確保如果參與者是理性的，欺騙是適得其反的，公平是自動的結果。強制執行不是獲得公平結果所必需的——它只是保持參與者不脫離約定協議所必需的。

現在您已經理解了這個基本模式，您將開始在比特幣、閃電網路和許多其他系統中到處看到它。接下來讓我們看一些具體的例子。

1.3.4. 公平協議範例

最突出的公平協議例子是比特幣的共識演算法：工作量證明（Proof of Work，PoW）。在比特幣中，礦工競爭驗證交易並將它們聚合到區塊中。為了確保礦工不會欺騙，而不將權威託付給他們，比特幣使用激勵和懲罰系統。礦工必須使用電力並專門用硬體進行「工作」，這些工作作為「證明」嵌入到每個區塊中。這是因為雜湊函數的一個特性，即輸出值在整個可能輸出的範圍內隨機分布。如果礦工成功快速產生一個有效區塊，他們會透過獲得該區塊的區塊獎勵來獲得獎勵。強制礦工在網路考慮他們的區塊之前使用大量電力意味著他們有動力正確驗證區塊中的交易。如果他們欺騙或犯任何錯誤，他們的區塊會被拒絕，他們用於「證明」的電力就會浪費。沒有人需要強迫礦工產生有效區塊；獎勵和懲罰激勵他們這樣做。協議需要做的就是確保只有帶有工作量證明的有效區塊被接受。

公平協議模式也可以在閃電網路的許多不同方面找到：

- 為通道提供資金的人確保他們在發布資金交易之前已簽署退款交易。
- 每當通道移動到新狀態時，舊狀態會被「撤銷」，確保如果任何人試圖廣播它，他們會失去整個餘額並受到懲罰。
- 轉發支付的人知道，如果他們承諾向前轉發資金，他們要麼可以獲得退款，要麼從前一個節點獲得付款。

我們一次又一次地看到這種模式。公平結果不是由任何權威強制執行的。它們作為獎勵公平和懲罰欺騙的協議的自然結果而出現，這是一種透過將自利引導向公平結果來利用自利的公平協議。

比特幣和閃電網路都是公平協議的實作。那麼為什麼我們需要閃電網路？比特幣還不夠嗎？

1.4. 閃電網路的動機

比特幣是一個將交易記錄在全球複製的公共帳本上的系統。每筆交易都被每台參與的電腦看到、驗證和儲存。正如您可以想像的，這會產生大量資料，並且難以擴展。

隨著比特幣和交易需求的增長，每個區塊中的交易數量增加，直到最終達到區塊大小限制。 一旦區塊「滿了」，多餘的交易就會留在佇列中等待。許多使用者會增加他們願意支付的費用，以在下一個區塊中為他們的交易購買空間。

如果需求持續超過網路容量，越來越多使用者的交易將等待未確認。對費用的競爭也增加了每筆交易的成本，使許多小額交易（例如微支付）在需求特別高的時期完全不經濟。

為了解決這個問題，我們可以增加區塊大小限制，為更多交易創造空間。區塊空間「供應」的增加將導致交易費用的較低價格均衡。

然而，增加區塊大小會將成本轉移給節點運營商，並要求他們花費更多資源來驗證和儲存區塊鏈。因為區塊鏈是八卦協議，每個節點都需要知道和驗證網路上發生的每一筆交易。此外，一旦驗證，每筆交易和區塊必須傳播給節點的「鄰居」，成倍增加頻寬需求。因此，區塊大小越大，每個單獨節點的頻寬、處理和儲存需求就越大。以這種方式增加交易容量會透過減少節點數量和節點運營商而產生使系統中心化的不良效果。由於節點運營商不會因運行節點而獲得補償，如果運行節點非常昂貴，只有少數資金充足的節點運營商會繼續運行節點。

1.4.1. 擴展區塊鏈

增加區塊大小或減少出塊時間對網路中心化的副作用是嚴重的，從數字計算可以看出。

讓我們假設比特幣的使用增長到網路必須處理每秒 40,000 筆交易，這大約是 Visa 網路在高峰使用期間的交易處理水平。

假設每筆交易平均 250 位元組，這將導致每秒 10 兆位元組（MBps）或每秒 80 兆位元（Mbps）的資料流，僅僅是為了能夠接收所有交易。 這還不包括將交易資訊轉發給其他節點的流量開銷。 雖然在高速光纖和 5G 行動速度的背景下 10 MBps 似乎不算極端，但它實際上會排除任何無法滿足此要求的人運行節點，特別是在高效能網際網路不可負擔或不普及的國家。

使用者還對其頻寬有許多其他需求，不能指望僅為接收交易就花費這麼多。

此外，在本地儲存這些資訊將導致每天 864 GB。這大約是 1 TB 的資料，或一個硬碟的大小。

每秒驗證 40,000 個橢圓曲線數位簽章演算法（ECDSA）簽名也幾乎不可行（參見 StackExchange 上的這篇文章 (https://bitcoin.stackexchange.com/questions/95339/how-many-bitcoin-transactions-can-be-verified-per-second)），使比特幣區塊鏈的「初始區塊下載（IBD）」（從創世區塊開始同步和驗證所有內容）在沒有非常昂貴的硬體的情況下幾乎不可能。

雖然每秒 40,000 筆交易看起來很多，但它只是在高峰時段達到傳統金融支付網路的同等水平。機器對機器支付、微交易和其他應用的創新可能會將需求推高到比這高出許多個數量級。

簡單地說：您無法以去中心化的方式擴展區塊鏈來驗證全世界的交易。

但是，如果每個節點不需要知道和驗證每一筆交易呢？如果有一種方法可以實現可擴展的鏈下交易，而不會失去比特幣網路的安全性呢？

2015 年 2 月，Joseph Poon 和 Thaddeus Dryja 發表了《The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments》[2]，提出了比特幣可擴展性問題的可能解決方案。

在（現已過時的）白皮書中，Poon 和 Dryja 估計，為了讓比特幣達到 Visa 在高峰期處理的每秒 47,000 筆交易，需要 8 GB 的區塊。 這將使運行節點對於除了大型企業和工業級運營之外的任何人都完全不可行。 結果將是一個只有少數使用者可以實際驗證帳本狀態的網路。 比特幣依賴使用者自己驗證帳本，而不是明確信任第三方，以保持去中心化。 使使用者無法負擔運行節點的費用將迫使普通使用者信任第三方來發現帳本狀態，最終破壞比特幣的信任模型。

閃電網路提出了一個新網路，一個第二層，使用者可以在這裡進行點對點支付，而不需要為每筆支付在比特幣區塊鏈上發布交易。 使用者可以在閃電網路上相互支付任意多次，而不會產生額外的比特幣交易或鏈上費用。 他們只使用比特幣區塊鏈最初將比特幣載入閃電網路，以及「結算」，即從閃電網路中移除比特幣。 結果是更多的比特幣支付可以在鏈下進行，只有初始載入和最終結算交易需要由比特幣節點驗證和儲存。 除了減輕節點負擔之外，閃電網路上的支付對使用者來說更便宜，因為他們不需要支付區塊鏈費用，對使用者來說更私密，因為它們不會發布給網路的所有參與者，而且不會永久儲存。

雖然閃電網路最初是為比特幣設計的，但它可以在任何滿足一些基本技術要求的區塊鏈上實現。其他區塊鏈，如萊特幣（Litecoin），已經支援閃電網路。此外，還有其他幾個區塊鏈正在開發類似的第二層或「Layer 2」解決方案來幫助它們擴展。

1.5. 閃電網路的定義特性

閃電網路是一個作為第二層協議運行在比特幣和其他區塊鏈之上的網路。閃電網路實現了快速、安全、私密、無需信任和無需許可的支付。以下是閃電網路的一些特性：

- 閃電網路的使用者可以以低成本即時向彼此路由支付。
- 在閃電網路上交換價值的使用者不需要等待區塊確認。
- 一旦閃電網路上的支付完成，通常在幾秒鐘內，它就是最終的且不可逆轉。與比特幣交易一樣，閃電網路上的支付只能由接收者退款。
- 鏈上比特幣交易是廣播並由網路中所有節點驗證的，而在閃電網路上路由的支付是在成對節點之間傳輸的，對所有人不可見，從而實現了更大的隱私性。
- 與比特幣網路上的交易不同，閃電網路上路由的支付不需要永久儲存。因此閃電網路使用更少的資源，因此更便宜。這個特性也有隱私方面的好處。
- 閃電網路使用洋蔥路由，類似於洋蔥路由器（Tor）隱私網路使用的協議，因此即使參與路由支付的節點也只直接知道其在支付路線中的前一個和後一個節點。
- 當在比特幣之上使用時，閃電網路使用真正的比特幣，始終由使用者擁有（保管）並完全控制。閃電網路不是一個單獨的代幣或幣種，它「就是」比特幣。

1.6. 閃電網路使用案例、使用者及其故事

為了更好地理解閃電網路的實際運作方式，以及人們為什麼使用它，我們將跟隨一些使用者及其故事。

在我們的例子中，有些人已經使用過比特幣，有些人則是比特幣網路的新手。每個人及其故事，如下所列，說明了一個或多個具體的使用案例。我們將在整本書中重新訪問他們：

消費者

Alice 是一位比特幣使用者，她想為小額零售購買進行快速、安全、便宜且私密的支付。她使用閃電網路用比特幣購買咖啡。

商家

Bob 擁有一家咖啡店，「Bob’s Cafe」。鏈上比特幣支付無法擴展到像一杯咖啡這樣的小額支付，所以他使用閃電網路幾乎即時接受比特幣支付，並且費用很低。

軟體服務業務

Chan 是一位中國企業家，銷售與閃電網路以及比特幣和其他加密貨幣相關的資訊服務。Chan 透過在閃電網路上實現微支付來銷售這些資訊服務。此外，Chan 還實施了一項流動性提供者服務，在閃電網路上出租入站通道容量，對每個租期收取少量比特幣費用。

遊戲玩家

Dina 是一位來自俄羅斯的青少年玩家。她玩許多不同的電腦遊戲，但她最喜歡的是那些具有基於真實貨幣的「遊戲內經濟」的遊戲。在玩遊戲時，她還透過獲取和銷售虛擬遊戲內物品來賺錢。閃電網路允許她為遊戲內物品進行小額交易，以及透過完成任務賺取小額收入。

1.7. 結論

在本章中，我們討論了比特幣和閃電網路的基礎概念：公平協議。

我們回顧了閃電網路的歷史以及比特幣和其他基於區塊鏈網路的第二層擴展解決方案背後的動機。

我們學習了基本術語，包括節點、支付通道、鏈上交易和鏈下支付。

最後，我們認識了 Alice、Bob、Chan 和 Dina，我們將在本書的其餘部分跟隨他們。在下一章中，我們將認識 Alice，並跟隨她的思考過程，當她選擇閃電網路錢包並準備進行她的第一筆閃電網路支付——從 Bob’s Cafe 購買一杯咖啡。

2. 入門指南

在本章中，我們將從大多數人第一次接觸閃電網路時開始——選擇軟體參與閃電網路經濟。我們將透過範例，跟隨兩位代表閃電網路常見使用案例的使用者來學習。Alice 是一位咖啡店顧客，她將使用行動裝置上的閃電網路錢包向 Bob’s Cafe 購買咖啡。Bob 是一位商家，他將使用閃電網路節點和錢包在他的咖啡店運行銷售點系統，以便接受閃電網路付款。

2.1. Alice 的第一個閃電網路錢包

Alice 是一位長期的比特幣使用者。我們在《Mastering Bitcoin》第一章首次認識了 Alice，[3] 當時她使用比特幣交易在 Bob’s Cafe 購買了一杯咖啡。如果您還不熟悉比特幣交易的運作方式或需要複習，請閱讀《Mastering Bitcoin》或 比特幣基礎回顧 中的摘要。

Alice 最近得知 Bob’s Cafe 開始接受閃電網路支付了！Alice 很想學習和嘗試閃電網路；她想成為 Bob 的第一批閃電網路客戶之一。為此，首先，Alice 必須選擇一個滿足她需求的閃電網路錢包。

Alice 不想將她的比特幣託管給第三方。她對加密貨幣已經有足夠的了解，知道如何使用錢包。她還想要一個行動錢包，這樣她就可以在外出時用它進行小額支付，所以她選擇了 *Eclair* 錢包，一個流行的非託管行動閃電網路錢包。讓我們了解更多關於她如何以及為什麼做出這些選擇。

2.2. 閃電網路節點

閃電網路透過能夠使用閃電網路協議的軟體應用程式來存取。*閃電網路節點*（LN 節點或簡稱節點）是一個具有三個重要特性的軟體應用程式。首先，閃電網路節點是錢包，因此它們可以透過閃電網路以及比特幣網路發送和接收付款。其次，節點必須與其他閃電網路節點進行點對點通訊，從而創建網路。最後，閃電網路節點還需要存取比特幣區塊鏈（或其他加密貨幣的其他區塊鏈）來保護用於支付的資金。

使用者透過運行自己的比特幣節點和閃電網路節點來獲得最高程度的控制。然而，閃電網路節點也可以使用輕量級比特幣客戶端，通常稱為簡化支付驗證（Simplified Payment Verification，SPV），來與比特幣區塊鏈互動。

2.3. 閃電網路瀏覽器

閃電網路瀏覽器是顯示節點、通道和網路容量統計資訊的有用工具。

以下是一個不完整的列表：

- 1ML 閃電網路瀏覽器 (https://1ml.com)
- ACINQ 的閃電網路瀏覽器 (https://explorer.acinq.co)，具有精美的視覺化效果
- Amboss Space 閃電網路瀏覽器 (https://amboss.space)，具有社群指標和直觀的視覺化

- Fiatjaf 的閃電網路瀏覽器 (https://ln.bigsun.xyz)，有許多圖表
- hashXP 閃電網路瀏覽器 (https://hashxp.org/lightning/node)

> 請注意，使用閃電網路瀏覽器時，就像使用其他區塊瀏覽器一樣，隱私可能是一個問題。 如果使用者不小心，網站可能會追蹤他們的 IP 地址並收集他們的行為記錄（例如，使用者感興趣的節點）。
>
> 此外，應該注意的是，由於目前的閃電網路圖或任何現有通道策略的當前狀態沒有全球共識，使用者永遠不應依賴閃電網路瀏覽器來檢索最新資訊。 此外，隨著使用者開啟、關閉和更新通道，圖會發生變化，個別閃電網路瀏覽器可能不是最新的。 使用閃電網路瀏覽器來視覺化網路或收集資訊，但不要將其作為閃電網路上正在發生什麼的權威來源。 要獲得閃電網路的權威視圖，請運行您自己的閃電網路節點，它將建立通道圖並收集各種統計資料，您可以透過網頁介面查看。

2.4. 閃電網路錢包

「閃電網路錢包」一詞有些模糊，因為它可以描述與某些使用者介面結合的各種元件。閃電網路錢包軟體最常見的元件包括：

- 一個金鑰儲存庫，用於保存秘密，如私鑰
- 一個閃電網路節點，如前所述，在點對點網路上通訊
- 一個比特幣節點，儲存區塊鏈資料並與其他比特幣節點通訊
- 一個閃電網路上公告的節點和通道的資料庫「地圖」
- 一個可以開啟和關閉閃電網路通道的通道管理器
- 一個可以找到從支付來源到支付目的地的連接通道路徑的路徑搜尋系統

閃電網路錢包可能包含所有這些功能，作為一個「完整」錢包，不依賴任何第三方服務。或者，一個或多個這些元件可能（部分或完全）依賴第三方服務來調解這些功能。

一個「關鍵」的區別（雙關語）是金鑰儲存功能是內部的還是外包的。在區塊鏈中，對金鑰的控制決定了對資金的託管，正如「你的金鑰，你的幣；不是你的金鑰，不是你的幣」這句話所銘記的。任何將金鑰管理外包的錢包都被稱為「託管」錢包，因為作為託管人的第三方控制著使用者的資金，而不是使用者。相比之下，「非託管」或「自託管」錢包是金鑰儲存庫是錢包的一部分，金鑰由使用者直接控制的錢包。非託管錢包這個術語僅意味著金鑰儲存庫是本地的並在使用者控制之下。然而，其他錢包元件中的一個或多個可能會或可能不會外包並依賴受信任的第三方。

區塊鏈，特別是像比特幣這樣的開放區塊鏈，試圖最小化或消除對第三方的信任並賦予使用者權力。這通常被稱為「無需信任」模型，儘管「信任最小化」是一個更好的術語。在這樣的系統中，使用者信任軟體規則，而不是第三方。因此，對金鑰的控制問題是選擇閃電網路錢包時的主要考慮因素。

閃電網路錢包的每個其他元件都帶來類似的信任考慮。如果所有元件都在使用者的控制之下，那麼對第三方的信任就會最小化，為使用者帶來最大的權力。當然，這帶來了直接的權衡，因為有了這種權力就有相應的責任來管理複雜的軟體。

每個使用者在決定使用什麼類型的閃電網路錢包之前必須考慮自己的技術技能。那些具有強大技術技能的人應該使用將所有元件置於使用者直接控制之下的閃電網路錢包。那些技術技能較少，但希望控制自己資金的人應該選擇非託管閃電網路錢包。 這些情況下的信任通常與隱私有關。 如果使用者決定將某些功能外包給第三方，他們通常會放棄一些隱私，因為第三方會了解他們的一些資訊。

最後，那些尋求簡單和便利的人，即使以犧牲控制和安全為代價，也可以選擇託管閃電網路錢包。這是技術上最不具挑戰性的選項，但它「破壞了加密貨幣的信任模型」，因此只應被視為邁向更多控制和自力更生的過渡步驟。

有許多方法可以對錢包進行特徵化或分類。 關於特定錢包要問的最重要問題是：

1. 這個閃電網路錢包是否有完整的閃電網路節點，還是使用第三方閃電網路節點？
2. 這個閃電網路錢包是否有完整的比特幣節點，還是使用第三方比特幣節點？
3. 這個閃電網路錢包是否在使用者控制下儲存自己的金鑰（自託管），還是金鑰由第三方託管人持有？

> 如果閃電網路錢包使用第三方閃電網路節點，則由該第三方閃電網路節點決定如何與比特幣通訊。因此，使用第三方閃電網路節點意味著您也在使用第三方比特幣節點。只有當閃電網路錢包使用自己的閃電網路節點時，才存在完整比特幣節點和第三方比特幣節點之間的選擇。

在最高抽象層次上，問題 1 和 3 是最基本的。 從這兩個問題，我們可以得出四個可能的類別。我們可以將這四個類別放入一個象限，如 閃電網路錢包象限 所示。 但請記住，這只是對閃電網路錢包進行分類的一種方式。

Table 1. 閃電網路錢包象限

	完整閃電網路節點	**第三方閃電網路節點**
自託管	Q1：高技術技能，對第三方信任最少，最無需許可	Q2：中等以下技術技能，對第三方信任中等以下，需要一些許可
託管	Q3：中等以上技術技能，對第三方信任中等以上，需要一些許可	Q4：低技術技能，對第三方信任高，最需要許可

象限 3（Q3），使用完整閃電網路節點，但金鑰由託管人持有，目前並不常見。 該象限的未來錢包可能會讓使用者擔心其節點的操作方面，但然後將金鑰的存取權委託給主要使用冷儲存的第三方。

閃電網路錢包可以安裝在各種裝置上，包括筆記型電腦、伺服器和行動裝置。要運行完整的閃電網路節點，您需要使用伺服器或桌上型電腦，因為行動裝置和筆記型電腦在容量、處理能力、電池壽命和連接性方面通常不夠強大。

第三方閃電網路節點類別可以再細分：

輕量級

這意味著錢包不運行閃電網路節點，因此需要透過網際網路從其他人的閃電網路節點獲取有關閃電網路的資訊。

無

這意味著不僅閃電網路節點由第三方運行，而且大部分錢包都由第三方在雲端運行。這是一個託管錢包，其他人控制資金的託管。

這些子類別在 流行閃電網路錢包範例 中使用。

流行閃電網路錢包範例 中「比特幣節點」列中需要解釋的其他術語是：

Neutrino

這個錢包不運行比特幣節點。相反，由其他人（第三方）運行的比特幣節點透過 Neutrino 協議存取。

Electrum

這個錢包不運行比特幣節點。相反，由其他人（第三方）運行的比特幣節點透過 Electrum 協議存取。

Bitcoin Core

這是比特幣節點的一個實作。

btcd

這是比特幣節點的另一個實作。

在 流行閃電網路錢包範例 中，我們看到一些目前流行的閃電網路節點和錢包應用程式適用於不同類型裝置的範例。該列表首先按裝置類型排序，然後按字母順序排序。

Table 2. 流行閃電網路錢包範例

應用程式	裝置	閃電網路節點	比特幣節點	金鑰儲存
Blue Wallet	行動	無	無	託管
Breez Wallet	行動	完整節點	Neutrino	自託管
Eclair Mobile	行動	輕量級	Electrum	自託管
lntxbot	行動	無	無	託管
Muun	行動	輕量級	Neutrino	自託管
Phoenix Wallet	行動	輕量級	Electrum	自託管
Zeus	行動	完整節點	Bitcoin Core/btcd	自託管
Electrum	桌面	完整節點	Bitcoin Core/Electrum	自託管
Zap Desktop	桌面	完整節點	Neutrino	自託管
c-lightning	伺服器	完整節點	Bitcoin Core	自託管
Eclair Server	伺服器	完整節點	Bitcoin Core/Electrum	自託管
lnd	伺服器	完整節點	Bitcoin Core/btcd	自託管

2.4.1. 測試網比特幣

比特幣系統提供了一個用於測試目的的替代鏈，稱為「測試網」（testnet），與被稱為「主網」（mainnet）的「正常」比特幣鏈形成對比。在測試網上，貨幣是「測試網比特幣」（tBTC），它是專門用於測試的無價值的比特幣副本。比特幣的每個功能都被精確複製，但這些錢沒有價值，所以您真的沒有什麼可失去的！

一些閃電網路錢包也可以在測試網上運行，允許您使用測試網比特幣進行閃電網路支付，而不會冒真實資金的風險。這是一種安全地試驗閃電網路的好方法。Alice 在本章中使用的 Eclair Mobile 就是支援測試網操作的閃電網路錢包的一個例子。

您可以從「測試網比特幣水龍頭」獲得一些 tBTC 來玩，它會按需免費發放 tBTC。以下是一些測試網水龍頭：

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

本書中的所有範例都可以用 tBTC 在測試網上精確複製，所以如果您想跟著做而不冒真實金錢的風險，可以這樣做。

2.5. 平衡複雜性和控制

閃電網路錢包必須在複雜性和使用者控制之間謹慎平衡。那些給予使用者對其資金最多控制、最高程度隱私以及對第三方服務最大獨立性的錢包必然更複雜且更難操作。隨著技術的進步，其中一些權衡將變得不那麼明顯，使用者可能能夠獲得更多控制而不需要更多複雜性。然而，目前，不同的公司和專案正在沿著這個控制-複雜性光譜探索不同的位置，希望為他們目標使用者找到「甜蜜點」。

選擇錢包時，請記住即使您看不到這些權衡，它們仍然存在。例如，許多錢包會嘗試從使用者那裡移除通道管理的負擔。為此，它們引入了所有錢包自動連接的中央「樞紐節點」。雖然這種權衡簡化了使用者介面和使用者體驗，但它引入了單點故障（SPoF），因為這些樞紐節點對錢包的運行變得不可或缺。此外，像這樣依賴「樞紐」可能會降低使用者隱私，因為樞紐知道發送者，並且可能（如果代表使用者構建支付路徑）也知道使用者錢包進行的每筆支付的接收者。

在下一節中，我們將回到我們的第一位使用者，並逐步介紹她的第一個閃電網路錢包設定。她選擇了一個比更容易的託管錢包更複雜的錢包。這讓我們可以展示一些底層複雜性並介紹進階錢包的一些內部運作。您可能會發現您的第一個理想錢包是面向易用性的，接受一些控制和隱私的權衡。或者也許您是更高級的使用者，想要運行自己的閃電網路和比特幣節點作為錢包解決方案的一部分。

2.6. 下載和安裝閃電網路錢包

在尋找新的加密貨幣錢包時，您必須非常小心地選擇軟體的安全來源。

不幸的是，許多假錢包應用程式會竊取您的錢，其中一些甚至會進入像 Apple 和 Google 應用程式商店這樣有信譽且據稱經過審查的軟體網站。無論您是安裝第一個還是第十個錢包，都要始終格外小心。流氓應用程式可能不僅會竊取您託付給它的任何資金，還可能透過入侵您的行動裝置作業系統來竊取其他應用程式的金鑰和密碼。

Alice 使用 Android 裝置，將使用 Google Play 商店下載並安裝 Eclair 錢包。在 Google Play 上搜尋，她找到了「Eclair Mobile」的條目，如 Google Play 商店中的 Eclair Mobile 所示。

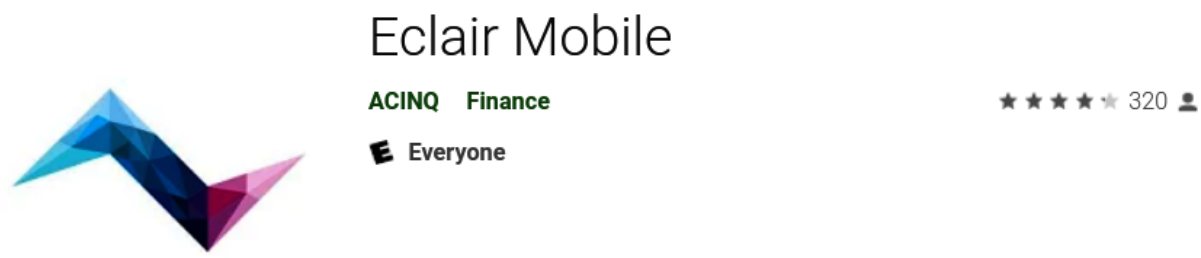

Figure 1. Google Play 商店中的 Eclair Mobile

可以使用測試網比特幣以零風險（除了您自己的時間）來試驗和測試所有比特幣類型的軟體。您也可以透過前往 Google Play 商店下載 Eclair 測試網錢包來嘗試閃電網路（在測試網上）。

Alice 注意到此頁面上有幾個不同的元素，幫助她確定這很可能是她正在尋找的正確「Eclair Mobile」錢包。首先，組織 ACINQ[4] 被列為此行動錢包的開發者，Alice 從她的研究中知道這是正確的開發者。其次，該錢包已被安裝「10,000+」次，並有超過 320 條正面評論。這不太可能是一個潛入 Google Play 商店的流氓應用程式。作為第三步，她前往 ACINQ 網站 (https://acinq.co)。她透過檢查地址是否以 https 開頭或在某些瀏覽器中以掛鎖為前綴來驗證網頁是否安全。在網站上，她前往下載部分或尋找到 Google App Store 的連結。她找到連結並點擊它。她比較這個連結是否帶她到 Google App Store 中的同一個應用程式。對這些發現感到滿意後，Alice 在她的行動裝置上安裝了 Eclair 應用程式。

在任何裝置上安裝軟體時都要格外小心。有許多假的加密貨幣錢包不僅會竊取您的錢，還可能危及您裝置上的所有其他應用程式。

2.7. 建立新錢包

當 Alice 第一次打開 Eclair Mobile 應用程式時，她會看到「建立新錢包」或「匯入現有錢包」的選項。Alice 將建立一個新錢包，但讓我們先討論為什麼這裡會出現這些選項以及匯入現有錢包意味著什麼。

2.7.1. 金鑰託管的責任

正如我們在本節開頭提到的，Eclair 是一個「非託管」錢包，這意味著 Alice 獨自託管用於控制她比特幣的金鑰。這也意味著 Alice 有責任保護和備份這些金鑰。如果 Alice 失去了金鑰，沒有人可以幫助她恢復比特幣，它們將永遠失去。

使用 Eclair Mobile 錢包，Alice 擁有金鑰的託管和控制權，因此有完全的責任保持金鑰的安全和備份。如果她失去了金鑰，她就失去了比特幣，沒有人可以幫助她從那種損失中恢復！

2.7.2. 助記詞

與大多數比特幣錢包類似，Eclair Mobile 提供一個「助記詞」（有時也稱為「種子」或「種子短語」）供 Alice 備份。助記詞由 24 個英文單詞組成，由軟體隨機選擇，用作錢包生成金鑰的基礎。在行動裝置遺失、軟體錯誤或記憶體損壞的情況下，Alice 可以使用助記詞恢復 Eclair Mobile 錢包中的所有交易和資金。

這些備份詞的正確術語是「助記詞」。我們避免使用「種子」一詞來指稱助記詞，因為儘管它的使用很常見，但它是不正確的。

當 Alice 選擇建立新錢包時，她將看到一個顯示她助記詞的螢幕，如 新錢包助記詞 中的螢幕截圖所示。

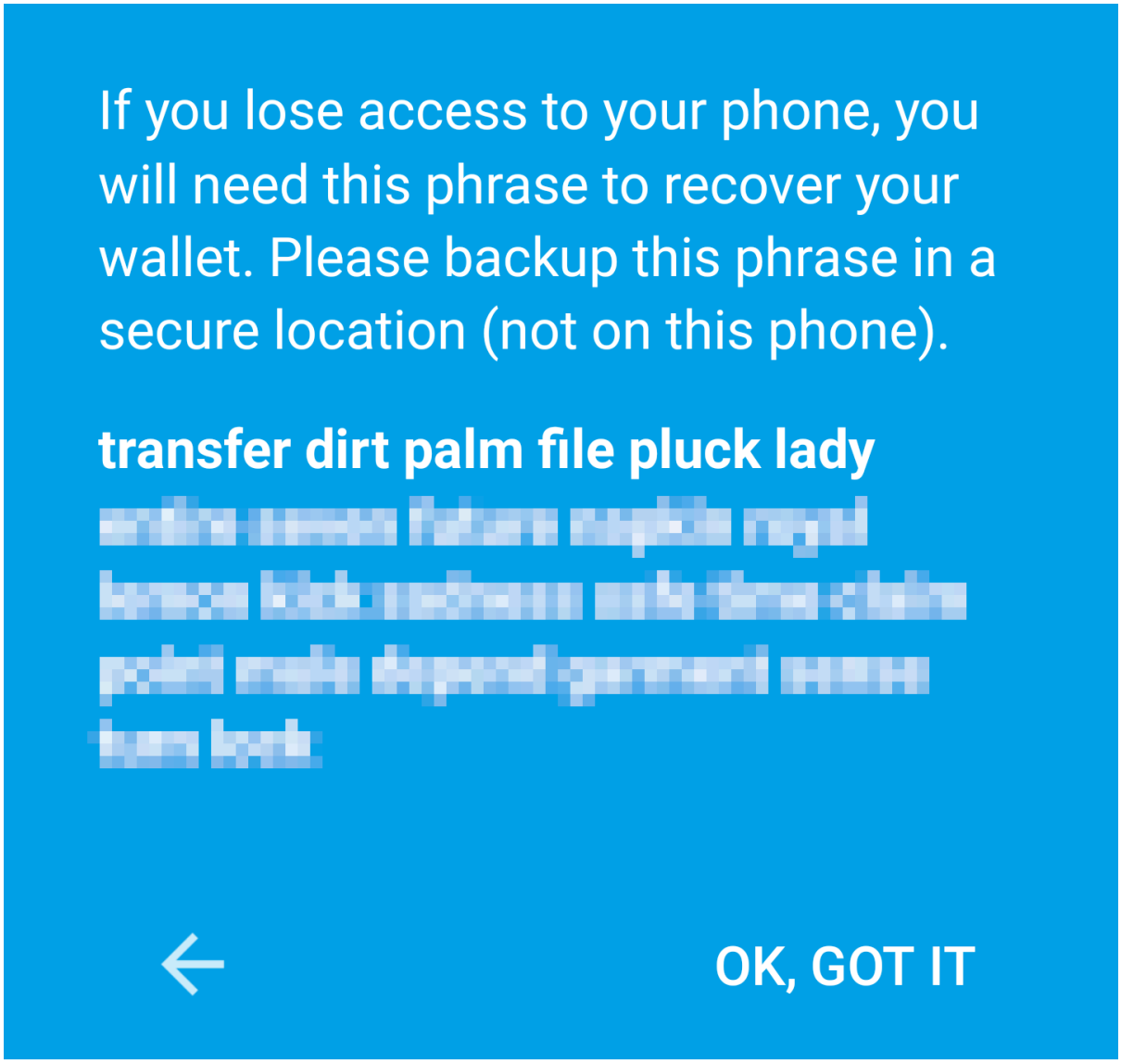

Figure 2. 新錢包助記詞

在 新錢包助記詞 中，我們故意模糊了部分助記詞，以防止本書讀者重複使用該助記詞。

2.7.3. 安全儲存助記詞

Alice 需要小心地以一種防止盜竊但也避免意外遺失的方式儲存助記詞。正確平衡這些風險的推薦方法是將助記詞的兩份副本寫在紙上，每個單詞都編號——順序很重要。

一旦 Alice 記錄了助記詞，在她的螢幕上點擊「OK GOT IT」後，她將被出示一個測驗，以確保她正確記錄了助記詞。測驗將隨機詢問三到四個單詞。Alice 沒有預料到會有測驗，但由於她正確記錄了助記詞，她毫無困難地通過了。

一旦 Alice 記錄了助記詞並通過了測驗，她應該將每份副本儲存在單獨的安全位置，如上鎖的抽屜或防火保險箱。

> 永遠不要嘗試任何與 安全儲存助記詞 中最佳實踐建議有任何偏差的「DIY」安全方案。不要把您的助記詞切成兩半、製作螢幕截圖、儲存在 USB 驅動器或雲端驅動器上、加密它，或嘗試任何其他非標準方法。您會以這樣一種方式破壞平衡，以至於冒著永久損失的風險。許多人失去了資金，不是因為盜竊，而是因為他們嘗試了一種非標準解決方案，卻沒有專業知識來平衡所涉及的風險。最佳實踐建議是由專家仔細考慮的，適合絕大多數使用者。

Alice 初始化她的 Eclair Mobile 錢包後，她將看到一個簡短的教學，突出顯示使用者介面的各種元素。我們不會在這裡複製教學，但我們將在跟隨 Alice 嘗試購買一杯咖啡時探索所有這些元素！

2.8. 將比特幣載入錢包

Alice 現在有了一個閃電網路錢包。但它是空的！她現在面臨這個實驗中更具挑戰性的方面之一：她必須找到一種方法來獲取一些比特幣並將其載入她的 Eclair 錢包。

> 如果 Alice 已經在另一個錢包中擁有比特幣，她可以選擇將該比特幣發送到她的 Eclair 錢包，而不是獲取新的比特幣來載入她的新錢包。

2.8.1. 獲取比特幣

Alice 可以透過幾種方式獲取比特幣：

- 她可以在加密貨幣交易所用她的法定貨幣（例如美元）兌換一些。
- 她可以從朋友或比特幣聚會上的熟人那裡用現金購買一些。
- 她可以在她所在地區找到一台「比特幣 ATM」，它就像一台自動販賣機，用現金出售比特幣。
- 她可以提供她的技能或她銷售的產品，並接受比特幣付款。
- 她可以要求她的雇主或客戶用比特幣支付她。

所有這些方法都有不同程度的難度，許多會涉及支付費用。有些還需要 Alice 提供身份證明文件以符合當地銀行法規。然而，透過所有這些方法，Alice 將能夠收到比特幣。

2.8.2. 接收比特幣

假設 Alice 找到了當地的比特幣 ATM 並決定用現金購買一些比特幣。比特幣 ATM 的一個例子，由 Lamassu 公司製造的，如 Lamassu 比特幣 ATM 所示。這種比特幣 ATM 透過現金插槽接受法定貨幣（現金），並使用內建攝影機將比特幣發送到從使用者錢包掃描的比特幣地址。

Figure 3. Lamassu 比特幣 ATM

要在她的 Eclair 閃電網路錢包中接收比特幣，Alice 需要向 ATM 出示 Eclair 閃電網路錢包中的比特幣地址。然後 ATM 可以將 Alice 新獲得的比特幣發送到這個比特幣地址。

要在 Eclair 錢包上看到比特幣地址，Alice 必須向左滑動到標題為 YOUR BITCOIN ADDRESS 的欄位（參見 Eclair 中顯示的 Alice 的比特幣地址），在那裡她會看到一個方形條碼（稱為 *QR 碼*）和下面的一串字母和數字。

QR 碼包含與下面顯示的相同的字母和數字字串，格式易於掃描。這樣，Alice 就不必輸入比特幣地址。在螢幕截圖（Eclair 中顯示的 Alice 的比特幣地址）中，我們故意模糊了兩者，以防止讀者無意中向這個地址發送比特幣。

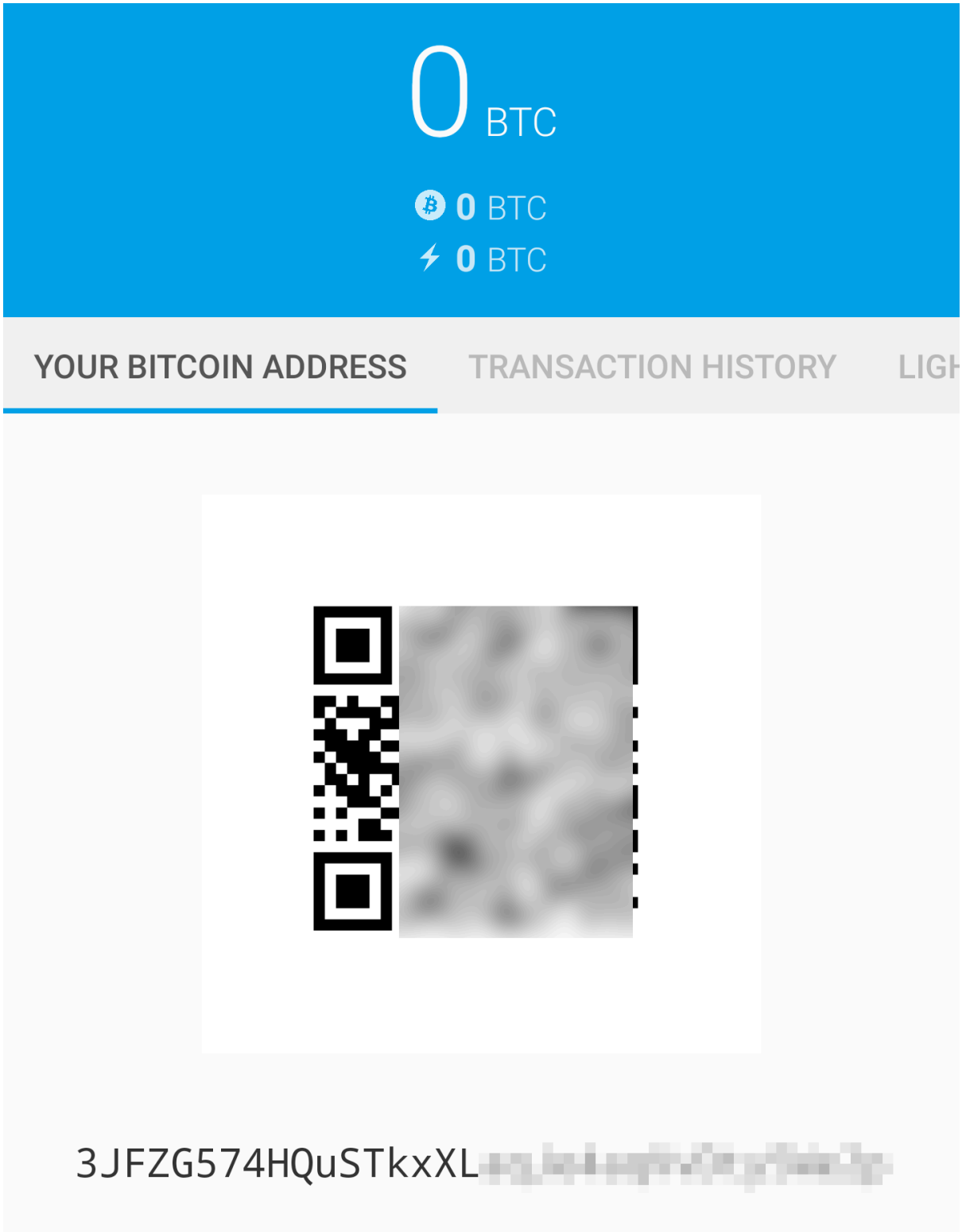

Figure 4. Eclair 中顯示的 Alice 的比特幣地址

比特幣地址和 QR 碼都包含錯誤檢測資訊，可以防止任何打字或掃描錯誤產生「錯誤的」比特幣地址。如果地址有錯誤，任何比特幣錢包都會注意到錯誤並拒絕接受比特幣地址為有效。

Alice 可以將她的行動裝置拿到 ATM 並向內建攝影機展示它，如 比特幣 ATM 掃描 QR 碼 所示。將一些現金插入插槽後，她將在 Eclair 中收到比特幣！

Figure 5. 比特幣 ATM 掃描 QR 碼

Alice 將在 Eclair 錢包的 TRANSACTION HISTORY 標籤中看到來自 ATM 的交易。雖然 Eclair 會在幾秒鐘內檢測到比特幣交易，但比特幣交易需要大約一個小時才能在比特幣區塊鏈上「確認」。如您在 Alice 收到比特幣 中所見，Alice 的 Eclair 錢包在交易下方顯示「6+ conf」，表示該交易已收到所需的最少六次確認，她的資金現在可以使用了。

交易上的確認數是自包含該交易的區塊（含）以來開採的區塊數。六次確認是最佳實踐，但不同的閃電網路錢包可以在任意數量的確認後認為通道已開啟。一些錢包甚至根據通道的貨幣價值來擴展預期確認數。

雖然在這個例子中 Alice 使用 ATM 獲得了她的第一筆比特幣，但即使她使用 獲取比特幣 中的其他方法之一，同樣的基本概念也適用。例如，如果 Alice 想銷售產品或提供專業服務以換取比特幣，她的客戶可以用他們的錢包掃描比特幣地址並用比特幣支付給她。

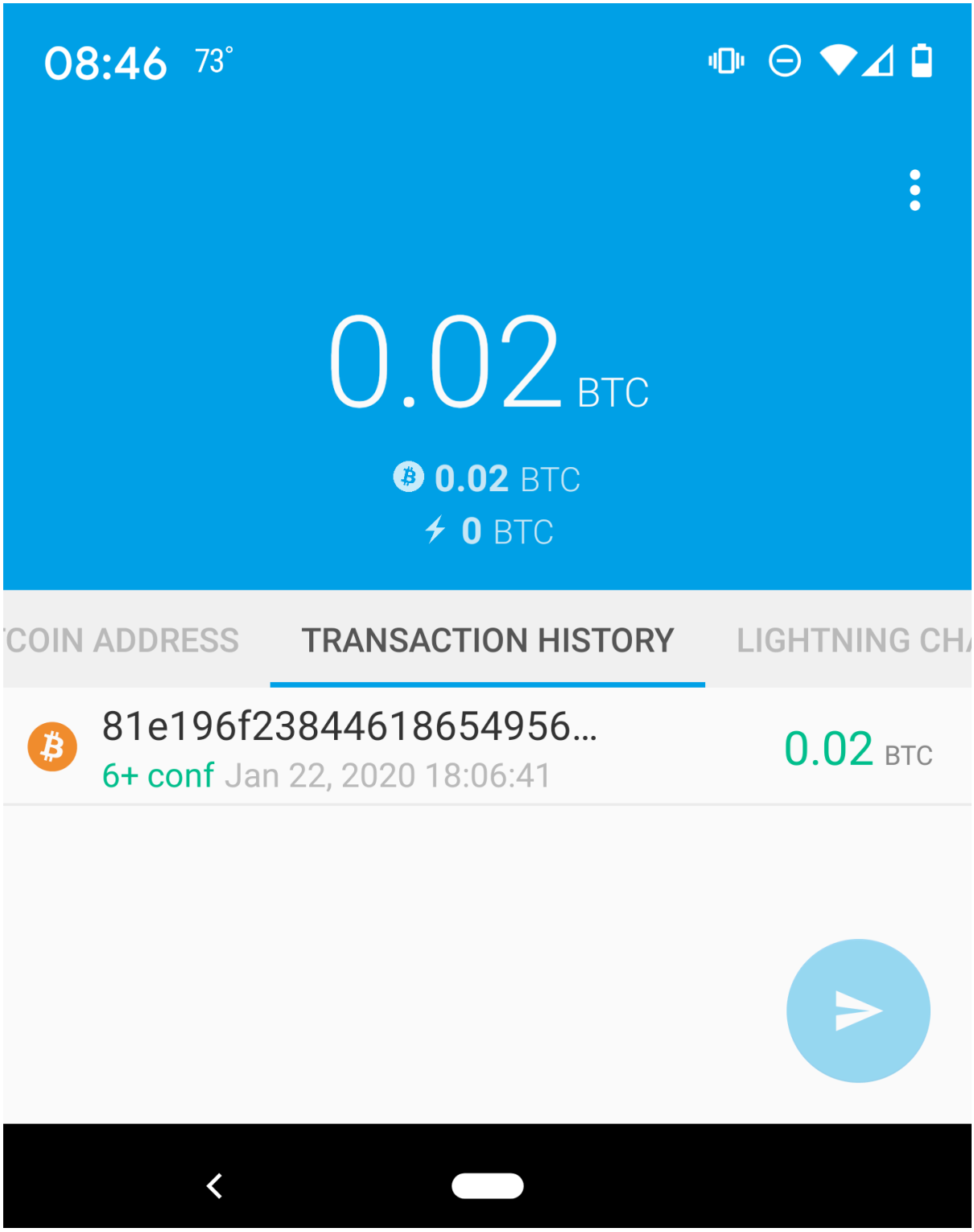

Figure 6. Alice 收到比特幣

同樣，如果她為透過網際網路提供的服務向客戶開帳單，Alice 可以向她的客戶發送包含比特幣地址或 QR 碼的電子郵件或即時訊息，他們可以將資訊貼上或掃描到比特幣錢包中來支付給她。

Alice 甚至可以列印 QR 碼並將其貼在標牌上並公開展示以接收小費。例如，她可以在她的吉他上貼一個 QR 碼，在街頭表演時接收小費！[5]

最後，如果 Alice 從加密貨幣交易所購買了比特幣，她可以（而且應該）透過將她的比特幣地址貼到交易所網站上來「提取」比特幣。然後交易所將直接把比特幣發送到她的地址。

2.9. 從比特幣到閃電網路

Alice 的比特幣現在由她的 Eclair 錢包控制，並已記錄在比特幣區塊鏈上。此時，Alice 的比特幣是「鏈上」的，這意味著該交易已廣播到整個比特幣網路，由所有比特幣節點驗證，並「開採」（記錄）到比特幣區塊鏈上。

到目前為止，Eclair Mobile 錢包僅作為比特幣錢包運行，Alice 還沒有使用 Eclair 的閃電網路功能。與許多閃電網路錢包的情況一樣，Eclair 透過同時作為比特幣錢包和閃電網路錢包來橋接比特幣和閃電網路。

現在，Alice 準備開始使用閃電網路，透過將她的比特幣移到鏈下，以利用閃電網路提供的快速、便宜和私密的支付。

2.9.1. 閃電網路通道

向右滑動，Alice 進入 Eclair 的 LIGHTNING CHANNELS 部分。在這裡，她可以管理將她的錢包連接到閃電網路的通道。

讓我們在這裡回顧一下閃電網路通道的定義，以使事情更清楚一些。首先，「通道」這個詞是 Alice 的閃電網路錢包和另一個閃電網路錢包之間「財務關係」的隱喻。我們稱之為通道是因為它是 Alice 的錢包和這另一個錢包在閃電網路上（鏈下）相互交換許多支付的方式，而無需將交易提交到比特幣區塊鏈（鏈上）。

Alice 開啟通道連接的錢包或「節點」稱為她的「通道對等節點」。一旦「開啟」，通道可用於在 Alice 的錢包和她的通道對等節點之間來回發送許多支付。

此外，Alice 的通道對等節點可以透過其他通道將支付「轉發」到閃電網路的更遠處。這樣，Alice 可以將支付「路由」到任何錢包（例如 Bob 的閃電網路錢包），只要 Alice 的錢包可以找到一條由從一個通道跳到另一個通道組成的可行「路徑」，一直到 Bob 的錢包。

> 並非所有通道對等節點都是路由支付的「好」對等節點。連接良好的對等節點將能夠透過更短的路徑將支付路由到目的地，增加成功的機會。擁有充足資金的通道對等節點將能夠路由更大的支付。

換句話說，Alice 需要一個或多個通道將她連接到閃電網路上的一個或多個其他節點。她不需要一個直接將她的錢包連接到 Bob’s Cafe 的通道來向 Bob 發送支付，儘管她也可以選擇開啟直接通道。閃電網路中的任何節點都可以用於 Alice 的第一個通道。節點連接越好，Alice 可以接觸到的人就越多。在這個例子中，由於我們還想示範支付路由，我們不會讓 Alice 直接向 Bob 的錢包開啟通道。相反，我們將讓 Alice 向一個連接良好的節點開啟通道，然後稍後使用該節點轉發她的支付，根據需要透過任何其他節點路由到達 Bob。

起初，沒有開啟的通道，所以如我們在 LIGHTNING CHANNELS 標籤 中看到的，LIGHTNING CHANNELS 標籤顯示一個空列表。如果您注意到，在右下角有一個加號（+），這是開啟新通道的按鈕。

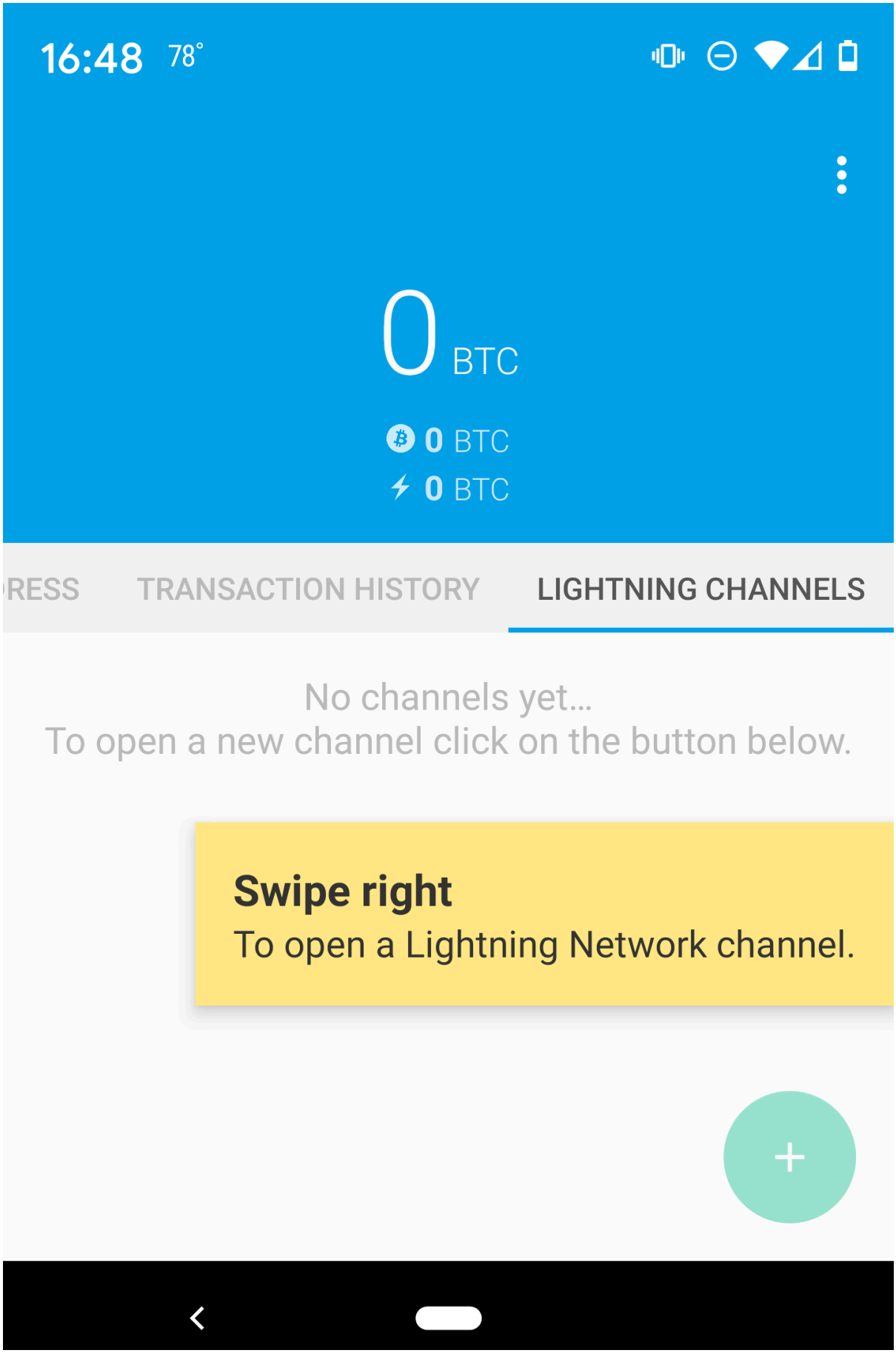

Figure 7. LIGHTNING CHANNELS 標籤

Alice 按下加號，並被提供四種可能的方式來開啟通道：

- 貼上節點 URI
- 掃描節點 URI
- 隨機節點
- ACINQ 節點

「節點 URI」是識別特定閃電網路節點的統一資源識別碼（URI）。Alice 可以從她的剪貼簿貼上這樣的 URI，或掃描包含相同資訊的 QR 碼。節點 URI 的範例顯示為 節點 URI 作為 QR 碼 中的 QR 碼，然後顯示為文字字串。

Figure 8. 節點 URI 作為 QR 碼

節點 URI

```
0237fefbe8626bf888de0cad8c73630e32746a22a2c4faa91c1d9877a3826e1174@1.ln.aantonop
.com:9735
```

雖然 Alice 可以選擇特定的閃電網路節點，或使用「隨機節點」選項讓 Eclair 錢包隨機選擇一個節點，但她將選擇 ACINQ 節點選項連接到 ACINQ 連接良好的閃電網路節點之一。

選擇 ACINQ 節點會稍微降低 Alice 的隱私，因為它會讓 ACINQ 能夠看到 Alice 的所有交易。它還會創建一個單點故障，因為 Alice 將只有一個通道，如果 ACINQ 節點不可用，Alice 將無法進行支付。為了一開始保持簡單，我們將接受這些權衡。在後續章節中，我們將逐步學習如何獲得更多獨立性並做出更少的權衡！

Alice 選擇 ACINQ 節點，準備在閃電網路上開啟她的第一個通道。

2.9.2. 開啟閃電網路通道

當 Alice 選擇一個節點來開啟新通道時，系統會詢問她想要為這個通道分配多少比特幣。在後續章節中，我們將討論這些選擇的影響，但現在 Alice 將把幾乎所有資金分配給通道。由於她必須支付交易費用來開啟通道，她將選擇一個略低於她總餘額的金額。[6]

Alice 將她 0.020 BTC 總額中的 0.018 BTC 分配給她的通道，並接受預設費率，如 開啟閃電網路通道 所示。

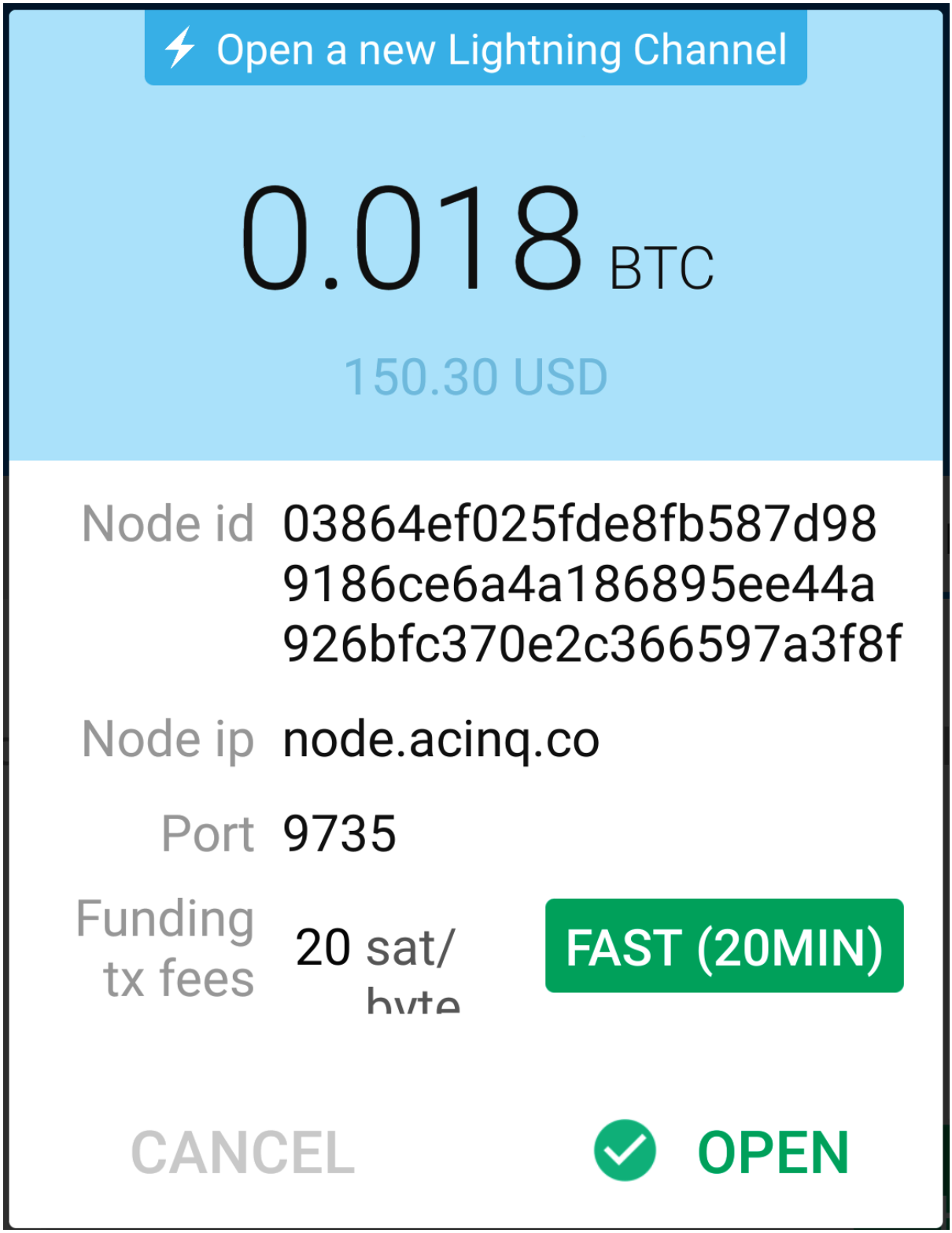

Figure 9. 開啟閃電網路通道

一旦她點擊 OPEN，她的錢包就會構建開啟閃電網路通道的特殊比特幣交易，稱為「資金交易」。鏈上資金交易被發送到比特幣網路進行確認。

Alice 現在必須再次等待（參見 等待資金交易開啟通道），等待交易記錄在比特幣區塊鏈上。與她用來獲取比特幣的初始比特幣交易一樣，她必須等待六次或更多確認（大約一小時）。

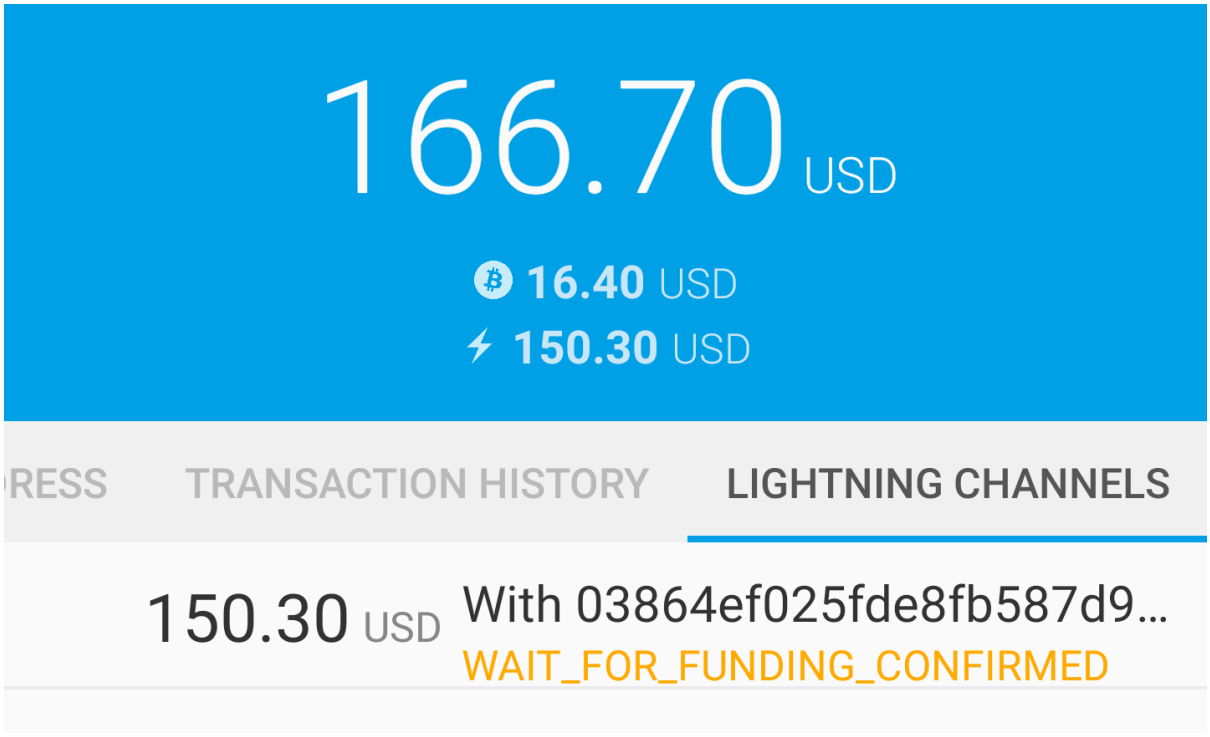

Figure 10. 等待資金交易開啟通道

一旦資金交易被確認，Alice 到 ACINQ 節點的通道就開啟、注資並準備就緒，如 通道已開啟 所示。

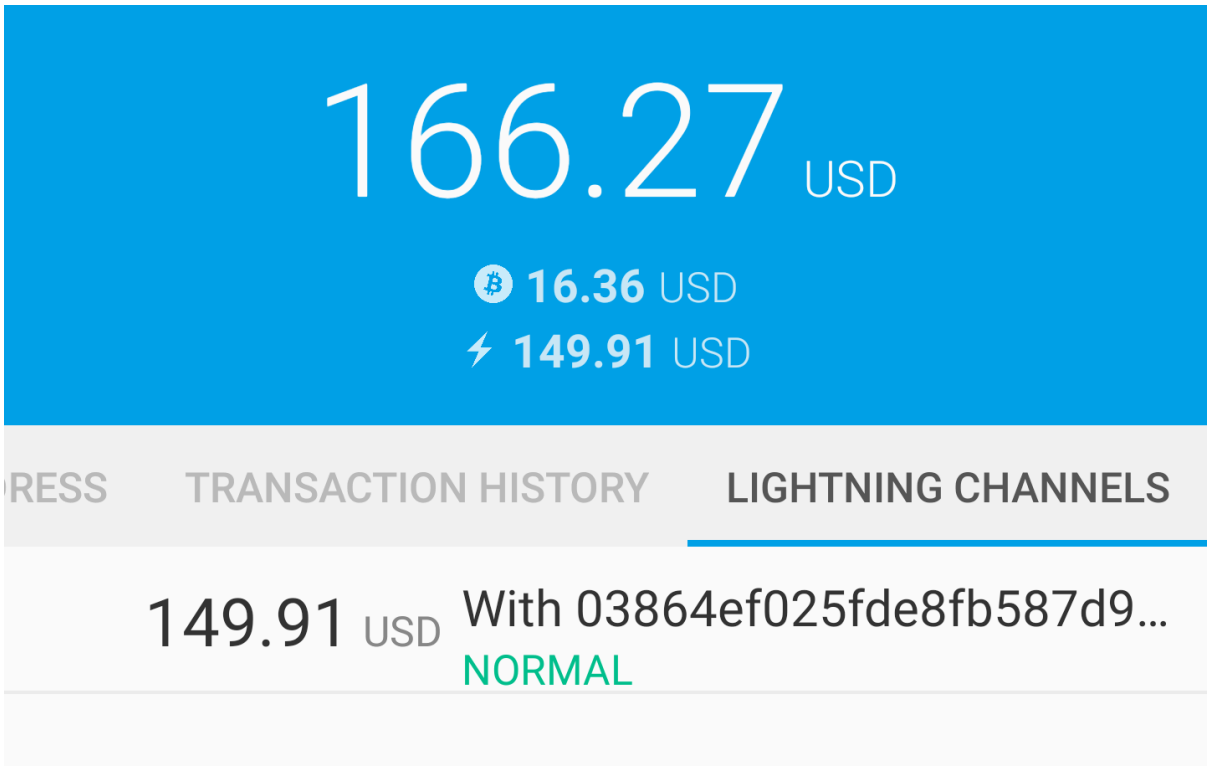

Figure 11. 通道已開啟

您是否注意到通道金額似乎已經改變了？其實沒有：通道包含 0.018 BTC，但在截圖之間，BTC 匯率發生了變化，所以美元價值不同。您可以選擇以 BTC 或 USD 顯示餘額，但請記住，USD 值是實時計算的，會發生變化！

2.10. 使用閃電網路購買一杯咖啡

Alice 現在已經準備好開始使用閃電網路了。如您所見，這需要一些工作和一些等待確認的時間。然而，現在後續的操作快速而簡單。閃電網路實現了無需等待確認的支付，資金在幾秒鐘內結算。

Alice 拿起她的行動裝置，跑到她附近的 Bob’ s Cafe。她很興奮嘗試她的新閃電網路錢包，用它來購買東西！

2.10.1. Bob’ s Cafe

Bob 有一個簡單的銷售點（PoS）應用程式，供任何想要透過閃電網路用比特幣付款的客戶使用。正如我們將在下一章看到的，Bob 使用流行的開源平台 *BTCPay Server*，它包含電子商務或零售解決方案所需的所有組件，例如：

- 使用 Bitcoin Core 軟體的比特幣節點
- 使用 c-lightning 軟體的閃電網路節點
- 用於平板電腦的簡單 PoS 應用程式

BTCPay Server 使安裝所有必要軟體、上傳圖片和產品價格以及快速啟動商店變得簡單。

在 Bob’ s Cafe 的櫃檯上，有一台平板裝置顯示您在 Bob 的銷售點應用程式 中看到的內容。

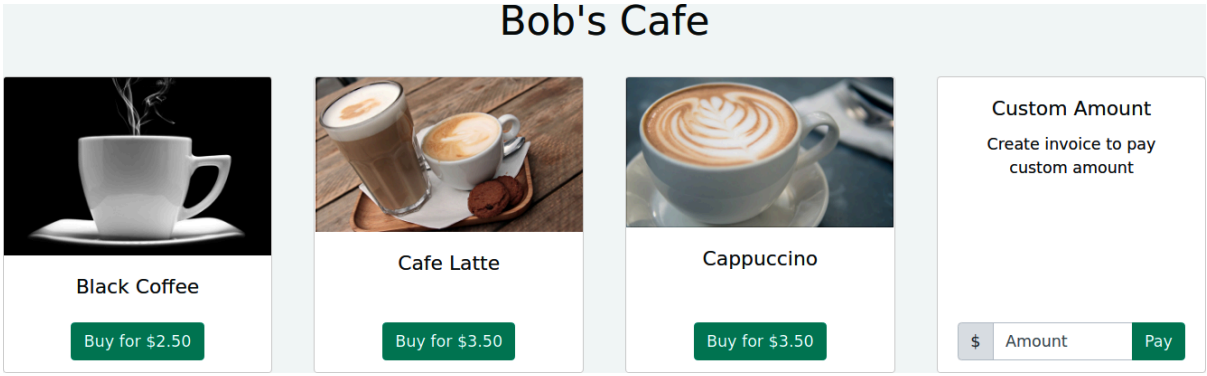

Figure 12. Bob 的銷售點應用程式

2.10.2. 閃電網路發票

Alice 從螢幕上選擇 Cafe Latte 選項，並被提供一張「閃電網路發票」（也稱為「付款請求」），如 Alice 拿鐵的閃電網路發票 所示。

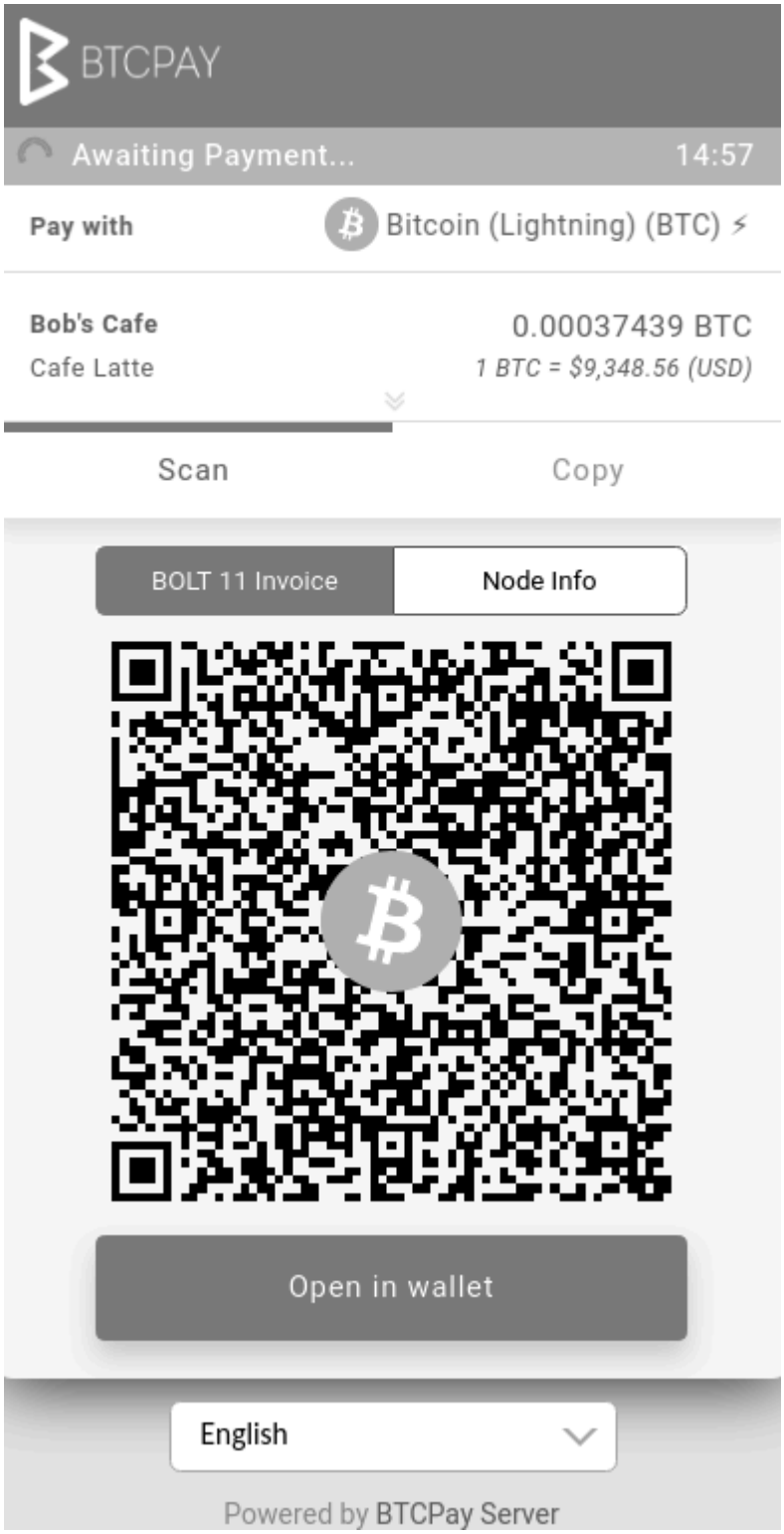

Figure 13. Alice 拿鐵的閃電網路發票

要支付發票，Alice 打開她的 Eclair 錢包，並選擇 TRANSACTION HISTORY 標籤下的 Send 按鈕（看起來像一個向上的箭頭），如 Alice 選擇發送 所示。

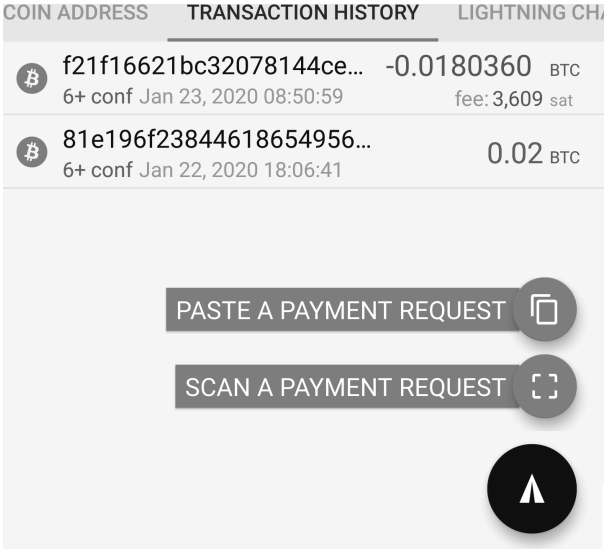

Figure 14. Alice 選擇發送

「付款請求」一詞可以指比特幣付款請求或閃電網路發票，「發票」和「付款請求」這兩個術語經常互換使用。正確的技術術語是「閃電網路發票」，無論錢包中如何命名。

Alice 選擇「掃描付款請求」選項，掃描平板螢幕上顯示的 QR 碼（參見 Alice 拿鐵的閃電網路發票），並被提示確認她的付款，如 Alice 的發送確認 所示。

Alice 按下 PAY，一秒鐘後，Bob 的平板顯示付款成功。Alice 完成了她的第一筆閃電網路支付！它快速、便宜且簡單。現在她可以享受她用比特幣透過快速、便宜且去中心化的支付系統購買的拿鐵了。從現在起，Alice 可以簡單地在 Bob 的平板螢幕上選擇一個項目，用她的手機掃描 QR 碼，點擊 PAY，然後在幾秒鐘內被端上咖啡，所有這些都不需要鏈上交易。

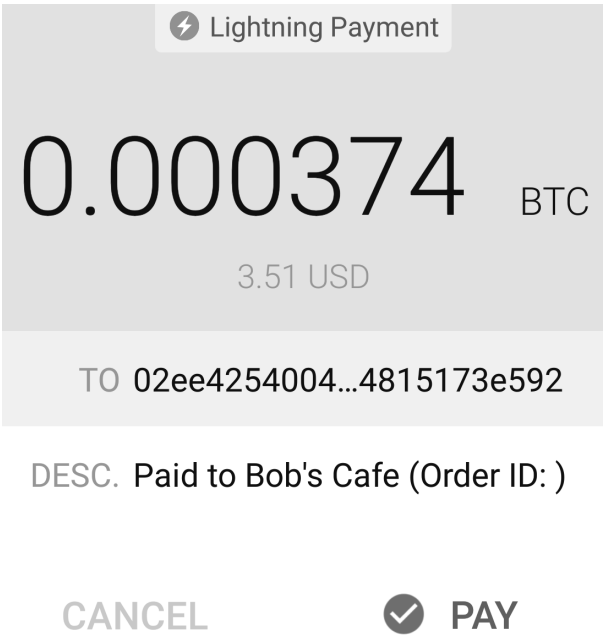

Figure 15. Alice 的發送確認

閃電網路支付對 Bob 來說也更好。他確信他會收到 Alice 拿鐵的錢，而不需要等待鏈上確認。將來，每當 Alice 想在 Bob’s Cafe 喝咖啡時，她可以選擇在比特幣網路或閃電網路上用比特幣付款。您認為她會選擇哪一個？

2.11. 結論

在本章中，我們跟隨 Alice 下載並安裝她的第一個閃電網路錢包，獲取並轉移一些比特幣，開啟她的第一個閃電網路通道，並透過在閃電網路上進行第一筆支付購買了一杯咖啡。在接下來的章節中，我們將深入了解閃電網路中每個元件的運作方式，以及 Alice 的支付如何到達 Bob’s Cafe。

3. 閃電網路運作原理

現在我們已經跟隨 Alice 設定了閃電網路錢包並從 Bob 那裡購買了一杯咖啡，讓我們深入了解並拆解該過程中涉及的閃電網路不同組件。 本章將提供高層次的概述，不會深入所有技術細節。 目標是幫助您了解閃電網路最重要的概念和構建模組。

如果您有計算機科學、密碼學、比特幣和協議開發的經驗，那麼本章應該足以讓您能夠自己填補連接的細節。 如果您經驗較少，本章將為您提供足夠好的概述，讓您更容易理解正式的協議規範，即 BOLT（Basis of Lightning Technology，閃電技術基礎）。 如果您是初學者，本章將幫助您更好地理解本書的技術章節。

如果您需要複習比特幣的基礎知識，可以在 比特幣基礎回顧 中找到以下主題的摘要回顧：

- 金鑰和地址
- 雜湊函數
- 數位簽章
- 交易結構
- 交易輸入和輸出
- 交易鏈接
- 比特幣腳本
- 多重簽名地址和腳本
- 時間鎖
- 複雜腳本

我們將從閃電網路的一句話定義開始，並在本章其餘部分對其進行分解。

閃電網路是一個「支付通道」的點對點網路，作為智慧合約在「比特幣區塊鏈」上實現，同時也是一個通訊協議，定義參與者如何設定和執行這些智慧合約。

3.1. 什麼是支付通道？

根據上下文，有幾種方式可以描述支付通道。讓我們從高層次開始，然後添加更多細節。

支付通道是閃電網路上兩個節點之間的「財務關係」，稱為「通道夥伴」。財務關係在兩個通道夥伴之間分配「資金餘額」（以毫聰計價）。

支付通道由「密碼學協議」管理，這意味著通道夥伴使用基於密碼學的預定義流程來重新分配通道餘額，使一方或另一方通道夥伴受益。密碼學協議確保一個通道夥伴無法欺騙另一個，因此夥伴不需要相互信任。

密碼學協議是透過對 2-of-2「多重簽名地址」注資來建立的，該地址要求兩個通道夥伴合作，並防止任何一方單方面花費資金。

總結：支付通道是節點之間的財務關係，透過嚴格定義的密碼學協議從多重簽名地址分配資金。

3.2. 支付通道基礎

支付通道的底層只是比特幣區塊鏈上的一個 2-of-2 多重簽名地址，您持有一把金鑰，您的通道夥伴持有另一把金鑰。

您和您的通道夥伴協商一系列從這個多重簽名地址花費的交易。您們不是將這些交易傳輸並記錄在比特幣區塊鏈上，而是都保留它們，未花費。

該序列中的最新交易編碼了通道的餘額，並定義了該餘額如何在您和您的通道夥伴之間分配。

因此，向這個序列添加新交易相當於將部分通道餘額從一個通道夥伴轉移到另一個，而比特幣網路並不知道。當您協商每筆新交易、改變通道中的資金分配時，您還會撤銷之前的交易，這樣任何一方都無法回退到之前的狀態。

序列中的每筆交易都使用比特幣的腳本語言，因此您和您的通道夥伴之間的資金協商由比特幣智慧合約管理。 智慧合約設定為懲罰試圖提交先前已撤銷的通道狀態的通道成員。

> 如果您有一筆來自 2-of-2 多重簽名地址的未發布交易，該交易向您支付部分餘額，那麼來自另一方的簽名確保您可以隨時透過添加自己的簽名獨立發布此交易。
>
> 持有部分簽名的交易、離線且未發布、並可選擇隨時發布和擁有該餘額的能力，是閃電網路的基礎。

3.3. 跨通道路由支付

一旦多個參與者擁有從一方到另一方的通道，支付也可以透過設定跨越網路的「路徑」，從支付通道「轉發」到支付通道，將多個支付通道連接在一起。

例如，如果 Alice 與 Bob 有通道，而 Bob 與 Charlie 有通道，Alice 可以向 Charlie 發送資金。

透過閃電網路的設計，可以擴展操作通道的智慧合約，使 Bob 無法竊取正在透過其通道轉發的資金。

就像智慧合約保護通道夥伴使他們不需要相互信任一樣，整個網路保護參與者，使他們可以轉發支付而不需要信任任何其他參與者。

因為通道是由多重簽名地址構建的，而餘額更新交易是預簽名的比特幣交易，操作閃電網路所需的所有信任都來自對去中心化比特幣網路的信任！

上述創新無疑是允許創建閃電網路的重大突破。 然而，閃電網路遠不止是比特幣腳本語言之上的密碼學協議。 它是一個全面的通訊協議，允許對等節點交換閃電網路訊息以實現比特幣的轉移。 通訊協議定義了閃電網路訊息如何加密和交換。

閃電網路還使用 gossip 協議向所有參與者分發有關通道（網路拓撲）的公開資訊。

例如，Alice 需要網路拓撲資訊才能知道 Bob 和 Charlie 之間的通道，這樣她才能構建一條到 Charlie 的路徑。

最後但同樣重要的是，理解閃電網路只不過是比特幣之上的應用程式，使用比特幣交易和比特幣腳本。沒有「閃電幣」或「閃電區塊鏈」。 除了所有技術原語之外，閃電網路協議是一種創造性的方式，透過允許任意數量的即時支付和即時結算，從比特幣中獲得更多好處，而不需要信任比特幣網路以外的任何人。

3.4. 支付通道

正如我們在上一章中看到的，Alice 使用她的錢包軟體在她自己和另一個閃電網路參與者之間創建了一個支付通道。

通道只受三件事限制：

- 首先，網際網路傳輸協議所需的幾百位元組資料，將資金從通道一端移動到另一端所需的時間
- 其次，通道的容量，即開啟通道時承諾到通道的比特幣數量
- 第三，比特幣交易的最大大小限制也限制了可以同時透過通道傳輸的未完成（進行中）路由支付數量

支付通道有一些非常有趣和有用的特性：

- 因為更新通道的時間主要受網際網路通訊速度的限制，在支付通道上進行支付幾乎可以是即時的。
- 如果通道是開啟的，進行支付不需要比特幣區塊的確認。事實上——只要您和您的通道夥伴遵循協議——它不需要與比特幣網路或您的通道夥伴以外的任何人互動。
- 密碼學協議的構建使得您和您的通道夥伴之間幾乎不需要信任。如果您的夥伴變得無回應或試圖欺騙您，您可以要求比特幣系統充當「法院」，解決您和您的夥伴之前同意的智慧合約。
- 在支付通道中進行的支付只有您和您的夥伴知道。在這個意義上，與比特幣相比，您獲得了隱私，因為比特幣中每筆交易都是公開的。只有最終餘額，即該通道中所有支付的總和，才會在比特幣區塊鏈上可見。

比特幣大約五歲時，才華橫溢的開發者首次弄清楚如何構建雙向、無限期、可路由的支付通道，到目前為止，至少有三種已知的方法。

本章將重點介紹 2015 年 Joseph Poon 和 Thaddeus Dryja 在 閃電網路白皮書 (https://lightning.network/lightning-network-paper.pdf) 中首次描述的通道構建方法。這些被稱為 *Poon-Dryja* 通道，是目前閃電網路中使用的通道構建方法。 另外兩種提出的方法是 *Duplex Micropayment* 通道（由 Christian Decker 在與 Poon-Dryja 通道大約同一時間引入）和 *eltoo* 通道（在 2018 年由 Christian Decker、Rusty Russel 和本書合著者 Olaoluwa Osuntokun 在 "eltoo: A Simple Layer2 Protocol for Bitcoin" (https://blockstream.com/eltoo.pdf) 中引入）。

eltoo 通道有一些有趣的特性，並簡化了支付通道的實現。然而，eltoo 通道需要比特幣腳本語言的更改，因此截至 2020 年無法在比特幣主網上實現。

3.4.1. 多重簽名地址

支付通道建立在 2-of-2 多重簽名地址之上。

總結來說，多重簽名地址是比特幣被鎖定的地方，需要多個簽名才能解鎖和花費。在閃電網路中使用的 2-of-2 多重簽名地址中，有兩個參與簽名者，「兩者」都需要簽名才能花費資金。

多重簽名腳本和地址在 多重簽章腳本 中有更詳細的解釋。

3.4.2. 資金交易

支付通道的基本構建模組是 2-of-2 多重簽名地址。兩個通道夥伴中的一個將透過向多重簽名地址發送比特幣來為支付通道注資。這筆交易稱為「資金交易」，並記錄在比特幣區塊鏈上。[7]

即使資金交易是公開的，除非通道被公開宣布，否則在關閉之前，它不會明顯是閃電網路支付通道。通道通常由希望轉發支付的路由節點公開宣布。然而，未宣布的通道也存在，通常由不積極參與路由的行動節點創建。此外，通道支付仍然對通道夥伴以外的任何人不可見，通道餘額在他們之間的分配也是如此。

存入多重簽名地址的金額稱為「通道容量」，它設定了可以透過支付通道發送的最大金額。然而，由於資金可以來回發送，通道容量不是可以流經通道的價值上限。這是因為如果通道容量在一個方向的支付中耗盡，它可以再次用於在相反方向發送支付。

> 在資金交易中發送到多重簽名地址的資金有時被稱為「鎖定在閃電網路通道中」。然而，實際上，閃電網路通道中的資金不是「鎖定」而是「釋放」的。閃電網路通道資金比比特幣區塊鏈上的資金更具流動性，因為它們可以更快、更便宜、更私密地花費。將資金轉移到閃電網路有一些缺點（例如需要將它們保存在「熱」錢包中），但將資金「鎖定」在閃電網路中的想法是誤導性的。

糟糕的通道開啟程序範例

如果您仔細考慮 2-of-2 多重簽名地址，您會意識到將您的資金放入這樣的地址似乎有一些風險。如果您的通道夥伴拒絕簽署交易來釋放資金怎麼辦？它們會永遠卡住嗎？現在讓我們看看那個場景以及閃電網路協議如何避免它。

Alice 和 Bob 想要創建一個支付通道。他們各自創建一個私鑰/公鑰對，然後交換公鑰。現在，他們可以用這兩個公鑰構建一個多重簽名 2-of-2，形成他們支付通道的基礎。

接下來，Alice 構建一筆比特幣交易，從她的錢包發送幾個 mBTC 到由 Alice 和 Bob 的公鑰創建的多重簽名地址。如果 Alice 不採取任何額外步驟就簡單地廣播這筆交易，她必須相信 Bob 會提供他的簽名來從多重簽名地址花費。另一方面，Bob 有機會透過扣留他的簽名並拒絕 Alice 存取她的資金來勒索 Alice。

為了防止這種情況，Alice 需要創建一筆額外的交易，從多重簽名地址花費，將她的 mBTC 退還給她。然後 Alice 讓 Bob 在將她的資金交易廣播到比特幣網路「之前」簽署退款交易。這樣，即使 Bob 消失或未能合作，Alice 也可以獲得退款。

保護 Alice 的「退款」交易是一類稱為「承諾交易」的交易中的第一個，我們將在下一節中更詳細地檢查。

3.4.3. 承諾交易

「承諾交易」是向每個通道夥伴支付其通道餘額並確保通道夥伴不必相互信任的交易。透過簽署承諾交易，每個通道夥伴「承諾」當前餘額，並讓另一個通道夥伴能夠隨時取回他們的資金。

透過持有已簽名的承諾交易，每個通道夥伴即使沒有另一個通道夥伴的合作也可以獲得他們的資金。這保護他們免受另一個通道夥伴的消失、拒絕合作或透過違反支付通道協議試圖欺騙的影響。

Alice 在前面範例中準備的承諾交易是她對多重簽名地址初始支付的退款。然而，更一般地說，承諾交易分割支付通道的資金，根據兩個通道夥伴各自持有的分配（餘額）向他們支付。起初，Alice 持有所有餘額，所以這是一個簡單的退款。但隨著資金從 Alice 流向 Bob，他們會交換代表新餘額分配的新承諾交易簽名，部分資金支付給 Alice，部分支付給 Bob。

假設 Alice 與 Bob 開啟一個容量為 100,000 聰的通道。 最初，Alice 擁有 100,000 聰，即通道中的全部資金。以下是支付通道協議的運作方式：

1. Alice 創建一個新的私鑰/公鑰對，並透過 `open_channel` 訊息（閃電網路協議中的訊息）通知 Bob 她希望開啟一個通道。
2. Bob 也創建一個新的私鑰/公鑰對，並同意接受來自 Alice 的通道，透過 `accept_channel` 訊息將他的公鑰發送給 Alice。
3. Alice 現在從她的錢包創建一筆資金交易，將 100k 聰發送到具有鎖定腳本的多重簽名地址：2 <PubKey Alice> <PubKey Bob> 2 CHECKMULTISIG。
4. Alice 尚未廣播這筆資金交易，而是在 `funding_created` 訊息中將交易 ID 發送給 Bob，以及她對 Bob 承諾交易的簽名。
5. Alice 和 Bob 都創建他們版本的承諾交易。這筆交易將從資金交易花費，並將所有比特幣發送回 Alice 控制的地址。

6. Alice 和 Bob 不需要交換這些承諾交易，因為他們各自知道它們是如何構建的，可以獨立構建兩者（因為他們已經就輸入和輸出的規範順序達成一致）。他們只需要交換簽名。
7. Bob 為 Alice 的承諾交易提供簽名，並透過 `funding_signed` 訊息將其發送回 Alice。
8. 現在簽名已經交換，Alice 將把資金交易廣播到比特幣網路。

透過遵循這個協議，即使資金被發送到 Alice 只控制一把金鑰的 2-of-2 多重簽名地址，Alice 也不會放棄她 100k 聰的所有權。 如果 Bob 停止回應 Alice，她將能夠廣播她的承諾交易並取回她的資金。 她唯一的成本是鏈上交易的費用。 只要她遵循協議，這就是她開啟通道時的唯一風險。

在這次初始交換之後，每當通道餘額變化時都會創建承諾交易。換句話說，每次在 Alice 和 Bob 之間發送支付時，都會創建新的承諾交易並交換簽名。每筆新的承諾交易都編碼了 Alice 和 Bob 之間的最新餘額。

如果 Alice 想向 Bob 發送 30k 聰，兩人都會創建他們承諾交易的新版本，現在向 Alice 支付 70k 聰，向 Bob 支付 30k 聰。透過編碼 Alice 和 Bob 的新餘額，新的承諾交易是透過通道「發送」支付的方式。

現在我們理解了承諾交易，讓我們看看一些更微妙的細節。您可能注意到這個協議留下了一種讓 Alice 或 Bob 作弊的方式。

3.4.4. 用先前狀態作弊

Alice 向 Bob 支付 30k 聰後持有多少筆承諾交易？她持有兩筆：原始的那筆向她支付 100k 聰，以及較新的那筆向她支付 70k 聰、向 Bob 支付 30k 聰。

在我們目前看到的通道協議中，沒有任何東西阻止 Alice 發布先前的承諾交易。作弊的 Alice 可以發布向她授予 100k 聰的承諾交易。 由於該承諾交易是由 Bob 簽署的，他無法阻止 Alice 傳輸它。

需要某種機制來防止 Alice 發布舊的承諾交易。現在讓我們找出如何實現這一點，以及它如何使閃電網路能夠在 Alice 和 Bob 之間無需任何信任的情況下運行。

因為比特幣是抗審查的，沒有人可以阻止某人發布舊的承諾交易。為了防止這種形式的作弊，承諾交易被構建為如果舊的被傳輸，作弊者可以受到懲罰。透過使懲罰足夠大，我們創造了強大的反作弊激勵，這使系統安全。

懲罰的運作方式是給被欺騙方一個機會來索取作弊者的餘額。因此，如果有人試圖透過廣播舊的承諾交易來作弊，在該交易中他們獲得的餘額比他們應得的更高，另一方可以透過同時拿走他們自己的餘額「和」作弊者的餘額來懲罰他們。作弊者失去一切。

> 您可能注意到，如果 Alice 幾乎完全耗盡她的通道餘額，她就可以嘗試以很小的風險作弊。如果她的通道餘額很低，Bob 的懲罰就不會那麼痛苦。為了防止這種情況，閃電網路協議要求每個通道夥伴在通道中保持最低餘額（稱為「準備金」），這樣他們總是有「利害關係」。

讓我們再次經歷通道構建場景，添加懲罰機制來防止作弊：

1. Alice 與 Bob 創建一個通道，並投入 100k 聰。
2. Alice 向 Bob 發送 30k 聰。
3. Alice 試圖透過發布舊的承諾交易來欺騙 Bob 賺得的 30k 聰，為自己索取全部 100k 聰。
4. Bob 檢測到欺詐並透過為自己拿走全部 100k 聰來懲罰 Alice。
5. Bob 最終擁有 100k 聰，因抓住 Alice 作弊而獲得 70k 聰。
6. Alice 最終擁有 0 聰。
7. 試圖欺騙 Bob 30k 聰，她失去了她擁有的 70k 聰。

有了強大的懲罰機制，Alice 不會想透過發布舊的承諾交易來作弊，因為她冒著失去全部餘額的風險。

> 在《Mastering Bitcoin》第 12 章中，Andreas Antonopoulos（本書合著者）如此陳述：「比特幣的一個關鍵特徵是，一旦交易有效，它就保持有效且不會過期。取消交易的唯一方法是在其被開採之前用另一筆交易雙花其輸入。」

現在我們理解了「為什麼」需要懲罰機制以及它將如何防止作弊，讓我們詳細看看它是「如何」運作的。

通常，承諾交易至少有兩個輸出，向每個通道夥伴支付。我們改變這一點，向其中一個支付添加「時間鎖延遲」和「撤銷秘密」。時間鎖防止輸出的所有者在承諾交易被包含在區塊中後立即花費它。撤銷秘密允許任何一方立即花費該支付，繞過時間鎖。

因此，在我們的範例中，Bob 持有一筆「立即」向 Alice 支付的承諾交易，但他自己的支付是延遲和可撤銷的。Alice 也持有一筆承諾交易，但她的是相反的：它立即向 Bob 支付，但她自己的支付是延遲和可撤銷的。

兩個通道夥伴各持有撤銷秘密的一半，所以他們都不知道整個秘密。如果他們分享自己的一半，那麼另一個通道夥伴就擁有完整的秘密，可以用它來行使撤銷條件。當簽署新的承諾交易時，每個通道夥伴透過將他們的撤銷秘密一半給另一方來撤銷先前的承諾。

我們將在 撤銷和重新承諾 中更詳細地檢查撤銷機制，在那裡我們將學習撤銷秘密如何構建和使用的細節。

簡單來說，只有當 Bob 提供他上一個承諾的撤銷秘密一半時，Alice 才簽署 Bob 的新承諾交易。只有當 Alice 給他上一個承諾的撤銷秘密一半時，Bob 才簽署 Alice 的新承諾交易。

每次新的承諾，他們都交換必要的「懲罰」秘密，使他們能夠透過使傳輸先前承諾變得無利可圖來有效地「撤銷」先前的承諾交易。本質上，他們在簽署新承諾時摧毀了使用舊承諾的能力。我們的意思是，雖然技術上仍然可以使用舊的承諾，但懲罰機制使這樣做在經濟上是不理性的。

時間鎖設定為最多 2,016 個區塊（大約兩週）。如果任何一個通道夥伴在未與另一個夥伴合作的情況下發布承諾交易，他們將不得不等待該區塊數（例如兩週）才能索取他們的餘額。另一個通道夥伴可以隨時索取他們自己的餘額。此外，如果發布的承諾之前已被撤銷，通道夥伴「也」可以立即索取作弊方的餘額，繞過時間鎖並懲罰作弊者。

時間鎖是可調整的，可以在通道夥伴之間協商。通常，對於較大容量的通道，時間鎖較長，對於較小的通道，時間鎖較短，以使激勵與資金價值保持一致。

對於通道餘額的每次新更新，都必須創建和保存新的承諾交易和新的撤銷秘密。只要通道保持開啟，為通道創建的所有撤銷秘密都需要保留，因為它們將來可能需要。幸運的是，秘密相當小，只有通道夥伴需要保留它們，而不是整個網路。此外，由於用於導出撤銷秘密的智慧導出機制，我們只需要儲存最新的秘密，因為先前的秘密可以從中導出（參見 作弊和懲罰的實踐）。

儘管如此，管理和儲存撤銷秘密是閃電網路節點更精細的部分之一，需要節點運營商維護備份。

諸如瞭望塔服務或將通道構建協議更改為 eltoo 協議等技術可能是未來緩解這些問題並減少對撤銷秘密、懲罰交易和通道備份需求的策略。

如果 Bob 不回應，Alice 可以隨時關閉通道，索取她公平份額的餘額。 在鏈上發布「最後」的承諾交易後，Alice 必須等待時間鎖過期才能從承諾交易中花費她的資金。正如我們稍後將看到的，只要 Alice 和 Bob 都在線並合作以正確的餘額分配關閉通道，就有一種更簡單的方法來關閉通道而無需等待。但每個通道夥伴儲存的承諾交易充當故障保護，確保如果他們的通道夥伴有問題，他們不會失去資金。

3.4.5. 宣布通道

通道夥伴可以同意向整個閃電網路宣布他們的通道，使其成為「公開通道」。為了宣布通道，他們使用閃電網路的 gossip 協議告訴其他節點通道的存在、容量和費用。

公開宣布通道允許其他節點使用它們進行支付路由，從而也為通道夥伴產生路由費用。

相比之下，通道夥伴可能決定不宣布通道，使其成為「未宣布」通道。

> 您可能聽到「私有通道」這個術語用來描述未宣布的通道。我們避免使用這個術語，因為它具有誤導性並創造了一種虛假的隱私感。雖然未宣布的通道在使用時不會被他人知道，但當通道關閉時，其存在和容量將被揭示，因為這些細節將在最終結算交易中在鏈上可見。其存在也可能透過各種其他方式洩露，所以我們避免稱其為「私有」。

未宣布的通道仍然用於路由支付，但只有知道其存在的節點，或給予「路由提示」關於包含未宣布通道的路徑。

當使用 gossip 協議公開宣布通道及其容量時，宣布還可以包含有關通道的資訊（元資料），例如其路由費用和時間鎖持續時間。

當新節點加入閃電網路時，它們從對等節點收集透過 gossip 協議傳播的通道宣布，建立閃電網路的內部地圖。然後可以使用此地圖來查找支付路徑，將通道端到端連接在一起。

3.4.6. 關閉通道

關閉通道的最佳方式是…不要關閉它！ 開啟和關閉通道需要鏈上交易，這將產生交易費用。 因此，最好盡可能長時間保持通道開啟。 只要您在通道一端有足夠的容量，您就可以繼續使用通道進行和轉發支付。 但即使您將所有餘額發送到通道的另一端，您也可以使用通道從您的通道夥伴那裡接收支付。 這種在一個方向使用通道然後在相反方向使用它的概念稱為「再平衡」，我們將在另一章中更詳細地檢查它。 透過再平衡通道，它可以幾乎無限期地保持開啟，並用於基本上無限數量的支付。

然而，有時關閉通道是可取的或必要的。例如：

- 您出於安全原因想要減少閃電網路通道上持有的餘額，並希望將資金發送到「冷儲存」。
- 您的通道夥伴長時間無回應，您無法再使用該通道。
- 通道不經常使用，因為您的通道夥伴不是一個連接良好的節點，所以您想將資金用於與更好連接節點的另一個通道。
- 您的通道夥伴違反了協議，無論是由於軟體錯誤還是故意的，迫使您關閉通道以保護您的資金。

有三種方式可以關閉支付通道：

- 互相關閉（好的方式）
- 強制關閉（壞的方式）
- 協議違規（醜陋的方式）

每種方法在不同情況下都是有用的，我們將在本章的下一節中探討。 例如，如果您的通道夥伴離線，您將無法遵循「好的方式」，因為沒有合作夥伴無法進行互相關閉。 通常，您的閃電網路軟體會根據情況自動選擇最佳的關閉機制。

互相關閉（好的方式）

互相關閉是當兩個通道夥伴同意關閉通道時，是首選的通道關閉方法。

當您決定要關閉通道時，您的閃電網路節點會通知您的通道夥伴您的意圖。 現在您的節點和通道夥伴的節點一起工作來關閉通道。 任何一方都不會接受新的路由嘗試，任何正在進行的路由嘗試都將在超時後結算或移除。 最終確定路由嘗試需要時間，所以互相關閉也可能需要一些時間才能完成。

一旦沒有待處理的路由嘗試，節點就會合作準備一筆「關閉交易」。 這筆交易類似於承諾交易：它編碼通道的最後餘額，但輸出不受時間鎖的限制。

關閉交易的鏈上交易費用由開啟通道的通道夥伴支付，而不是由發起關閉程序的人支付。 使用鏈上費用估算器，通道夥伴就適當的費用達成一致，雙方都簽署關閉交易。

一旦關閉交易被廣播並被比特幣網路確認，通道就有效地關閉了，每個通道夥伴都收到了他們在通道餘額中的份額。 儘管有等待時間，互相關閉通常比強制關閉更快。

強制關閉（壞的方式）

強制關閉是當一個通道夥伴試圖在未經另一個通道夥伴同意的情況下關閉通道。

這通常發生在其中一個通道夥伴無法聯繫時，所以互相關閉是不可能的。 在這種情況下，您會發起強制關閉，單方面關閉通道並「釋放」資金。

要發起強制關閉，您可以簡單地發布您節點持有的最後一筆承諾交易。 畢竟，這就是承諾交易的用途——它們保證您不需要信任您的通道夥伴就可以取回您通道的餘額。

一旦您將最後一筆承諾交易廣播到比特幣網路並被確認，它將創建兩個可花費的輸出，一個給您，一個給您的夥伴。 正如我們之前討論的，比特幣網路無法知道這是最近的承諾交易還是發布來從您夥伴那裡偷竊的舊交易。 因此，這筆承諾交易會給您的夥伴一點優勢。 發起強制關閉的夥伴的輸出會受到時間鎖的限制，而另一個夥伴的輸出會立即可花費。 如果您廣播了較早的承諾交易，時間鎖延遲給您的夥伴機會使用撤銷秘密對交易提出異議並懲罰您的作弊行為。

在強制關閉期間發布承諾交易時，鏈上費用會比互相關閉更高，原因有幾個：

1. 當承諾交易被協商時，通道夥伴不知道交易廣播時的鏈上費用會是多少。由於沒有雙方簽名就無法更改費用而不更改承諾交易的輸出，而且強制關閉發生在通道夥伴無法簽名時，協議開發者決定對承諾交易中包含的費率非常慷慨。它可以比承諾交易協商時費用估算器建議的高出五倍。
2. 承諾交易包括任何待處理路由嘗試雜湊時間鎖定合約（HTLC）的額外輸出，這使得承諾交易比互相關閉交易更大（以位元組計）。更大的交易產生更多費用。
3. 任何待處理的路由嘗試都必須在鏈上解決，導致額外的鏈上交易。

> 雜湊時間鎖定合約（HTLC）將在 雜湊時間鎖定合約 中詳細介紹。 現在，假設這些是透過閃電網路路由的支付，而不是兩個通道夥伴之間直接進行的支付。 這些 HTLC 作為承諾交易中的額外輸出攜帶，從而增加交易大小和鏈上費用。

一般來說，除非絕對必要，否則不建議強制關閉。 您的資金會被鎖定更長時間，開啟通道的人將不得不支付更高的費用。 此外，即使您沒有開啟通道，您也可能需要支付鏈上費用來中止或結算路由嘗試。

如果通道夥伴是您認識的，您可能會考慮聯繫該個人或公司，詢問他們的閃電網路節點為什麼關閉，並請求他們重新啟動它，以便您可以實現通道的互相關閉。

您應該只將強制關閉視為最後的手段。

協議違規（醜陋的方式）

協議違規是當您的通道夥伴試圖欺騙您時，無論是故意還是無意的，透過將過時的承諾交易發布到比特幣區塊鏈，本質上是從他們那邊發起（不誠實的）強制關閉。

您的節點必須在線並監視比特幣區塊鏈上的新區塊和交易才能檢測到這一點。

因為您的通道夥伴的支付會受到時間鎖的限制，您的節點有一些時間在時間鎖過期之前檢測協議違規並發布「懲罰交易」。

如果您成功檢測到協議違規並執行懲罰，您將收到通道中的所有資金，包括您通道夥伴的資金。

在這種情況下，通道關閉會相當快。 您將不得不支付鏈上費用來發布懲罰交易，但您的節點可以根據費用估算設定這些費用，而不會多付。 您通常會想支付更高的費用以保證盡快確認。 然而，因為您最終會收到作弊者的所有資金，本質上是作弊者在支付這筆交易。

如果您未能檢測到協議違規且時間鎖過期，您將只收到您夥伴發布的承諾交易分配給您的資金。 在此之後您收到的任何資金都會被您的夥伴竊取。 如果有任何餘額分配給您，您將不得不支付鏈上費用來收取該餘額。

與強制關閉一樣，所有待處理的路由嘗試也必須在承諾交易中解決。

協議違規可以比互相關閉執行得更快，因為您不需要等待與您的夥伴協商關閉，也比強制關閉更快，因為您不需要等待您的時間鎖過期。

賽局理論預測作弊不是一個有吸引力的策略，因為很容易檢測到作弊者，而且作弊者冒著失去「所有」資金的風險，而只能獲得他們在較早狀態中的收益。 此外，隨著閃電網路的成熟和瞭望塔變得廣泛可用，即使被欺騙的通道夥伴離線，作弊者也會被第三方檢測到。

因此，我們不建議作弊。 然而，我們確實建議任何抓住作弊者的人透過拿走他們的資金來懲罰他們。

那麼，您如何在日常活動中抓住作弊或協議違規？ 您透過運行監控公開比特幣區塊鏈的軟體來做到這一點，該軟體監控與您任何通道的任何承諾交易相對應的鏈上交易。 此軟體是三種類型之一：

- 正確維護的閃電網路節點，全天候運行
- 您運行來監視通道的單一用途瞭望塔節點
- 您付費來監視通道的第三方瞭望塔節點

請記住，承諾交易有一個以給定區塊數指定的超時期限，最多 2,016 個區塊。 只要您在超時期限到達之前運行閃電網路節點一次，它就會抓住所有作弊嘗試。 不建議冒這種風險；保持一個維護良好的節點持續運行很重要（參見 為什麼可靠性對運行閃電網路節點很重要？）。

3.5. 發票

閃電網路上的大多數支付都是從接收者創建的發票開始的。在我們之前的範例中，Bob 創建了一張發票來請求 Alice 付款。

有一種方法可以在沒有發票的情況下發送未經請求的支付，使用協議中稱為 keysend 的解決方法。我們將在 Keysend 自發付款 中檢查這一點。

發票是一個簡單的支付指令，包含諸如唯一支付識別碼（稱為支付雜湊）、接收者、金額和可選文字描述等資訊。

發票最重要的部分是支付雜湊，它允許支付以「原子」方式跨多個通道傳輸。在計算機科學中，原子意味著任何完全成功完成或根本不完成的動作或狀態變化——沒有中間狀態或部分動作的可能性。在閃電網路中，這意味著支付要麼走完整條路徑，要麼完全失敗。它不能部分完成，使得路徑上的中間節點可以收到支付並保留它。 沒有「部分支付」或「部分成功支付」這回事。

發票不透過閃電網路傳輸。相反，它們「帶外」傳輸，使用任何其他通訊機制。這類似於比特幣地址如何在比特幣網路之外傳輸給發送者：作為 QR 碼、透過電子郵件或簡訊。例如，Bob 可以向 Alice 展示閃電網路發票作為 QR 碼、透過電子郵件或透過任何其他訊息通道。

發票通常編碼為長的 *bech32* 編碼字串或 QR 碼，由智慧手機閃電網路錢包掃描。發票包含請求的比特幣金額和接收者的簽名。發送者使用簽名來提取接收者的公鑰（也稱為節點 ID），這樣發送者就知道將支付發送到哪裡。

您注意到這與比特幣的對比以及使用了不同的術語嗎？在比特幣中，接收者將地址傳遞給發送者。在閃電網路中，接收者創建發票並將發票發送給發送者。在比特幣中，發送者將資金發送到地址。在閃電網路中，發送者支付發票，支付被路由到接收者。比特幣基於「地址」的概

念，而閃電網路是基於「發票」概念的支付網路。在比特幣中，我們創建「交易」，而在閃電網路中，我們發送「支付」。

3.5.1. 支付雜湊和原像

發票最重要的部分是「支付雜湊」。當構建發票時，Bob 會這樣做支付雜湊：

1. Bob 選擇一個隨機數 *r*。這個隨機數稱為「原像」或「支付秘密」。
2. Bob 使用 SHA-256 計算 *r* 的雜湊 *H*，稱為「支付雜湊」：*H* = SHA-256(*r*)。

> 術語「原像」來自數學。在任何函數 *y* = *f*(*x*) 中，產生某個值 *y* 的輸入集合稱為 *y* 的原像。在這種情況下，函數是 SHA-256 雜湊演算法，任何產生雜湊 *H* 的值 *r* 都稱為原像。

沒有已知的方法可以找到 SHA-256 的逆（即從雜湊計算原像）。只有 Bob 知道值 *r*，所以它是 Bob 的秘密。但一旦 Bob 透露 *r*，任何擁有雜湊 *H* 的人都可以透過計算 SHA-256(*r*) 並看到它與 *H* 匹配來檢查 *r* 是否是正確的秘密。

只有當 *r* 完全隨機選擇且不可預測時，閃電網路的支付過程才是安全的。這種安全性依賴於雜湊函數不能被反轉或可行地暴力破解的事實，因此沒有人可以從 *H* 找到 *r*。

3.5.2. 額外元資料

發票可以選擇性地包含其他有用的元資料，例如簡短的文字描述。如果使用者有多張發票要支付，使用者可以閱讀描述並被提醒發票是關於什麼的。

發票還可以包含一些「路由提示」，允許發送者使用未宣布的通道來構建到接收者的路徑。路由提示也可以用於建議公開通道，例如接收者知道有足夠入站容量來路由支付的通道。

如果發送者的閃電網路節點無法透過閃電網路發送支付，發票可以選擇性地包含一個鏈上比特幣地址作為備用。

> 雖然總是可以「回退」到鏈上比特幣交易，但實際上更好的做法是向接收者開啟一個新通道。如果您必須產生鏈上費用來進行支付，您不如產生這些費用來開啟一個通道並透過閃電網路進行支付。支付完成後，您會留下一個在接收者端有流動性的開啟通道，可以用於將來將支付路由回您的閃電網路節點。一筆鏈上交易給您一筆支付和一個未來使用的通道。

閃電網路發票包含一個過期日期。由於接收者必須為每張發出的發票保留原像 *r*，讓發票過期是有用的，這樣這些原像就不需要永遠保留。一旦發票過期或被支付，接收者可以丟棄原像。

3.6. 傳遞支付

我們已經看到接收者如何創建包含支付雜湊的發票。這個支付雜湊將用於透過一系列支付通道從發送者移動支付到接收者，即使他們之間沒有直接的支付通道。

在接下來的幾節中，我們將深入了解用於透過閃電網路傳遞支付的想法和方法，並使用我們迄今為止呈現的所有概念。

首先，讓我們看看閃電網路的通訊協議。

3.6.1. 點對點 Gossip 協議

正如我們之前提到的，當構建支付通道時，通道夥伴可以選擇公開它，向整個閃電網路宣布其存在和細節。

通道宣布透過點對點「gossip 協議」傳播。gossip 協議是一種通訊協議，每個節點連接到網路中隨機選擇的其他節點，通常透過 TCP/IP。直接連接到您節點（透過 TCP/IP）的每個節點稱為您的「對等節點」。您的節點反過來是他們的對等節點之一。請記住，當我們說您的節點連接到其他對等節點時，我們不是指您有支付通道，而只是您透過 gossip 協議連接。

開啟通道後，節點可以選擇透過 `channel_announcement` 訊息向其對等節點發送通道的宣布。 每個對等節點驗證來自 `channel_announcement` 訊息的資訊，並驗證資金交易是否在比特幣區塊鏈上確認。 驗證後，節點會將 gossip 訊息轉發給其自己的對等節點，他們會將其轉發給他們的對等節點，依此類推，將宣布傳播到整個網路。 為了避免過度通訊，只有當節點之前尚未轉發該宣布時，才會轉發通道宣布。

gossip 協議還用於透過 `node_announcement` 訊息宣布有關已知節點的資訊。 為了轉發此訊息，節點必須至少有一個公開通道在 gossip 協議上宣布，再次是為了避免過度通訊流量。

支付通道有各種元資料，對網路的其他參與者有用。 這些元資料主要用於做出路由決策。 因為節點可能偶爾會更改其通道的元資料，這些資訊透過 `channel_update` 訊息共享。 這些訊息大約每天只會轉發四次（每個通道），以防止過度通訊。 gossip 協議還有各種查詢和訊息，用於最初同步節點與網路視圖，或在離線一段時間後更新節點的視圖。

閃電網路參與者的一個主要挑戰是，由 gossip 協議共享的拓撲資訊只是部分的。 例如，支付通道的容量透過 `channel_announcement` 訊息在 gossip 協議上共享。 然而，這些資訊不如兩個通道夥伴之間本地餘額方面的容量實際分配那麼有用。 一個節點只能轉發它在該通道內實際擁有（本地餘額）的比特幣數量。

雖然閃電網路本可以設計為共享通道的餘額資訊和精確拓撲，但由於以下幾個原因而沒有這樣做：

- 為了保護使用者的隱私，它不會大聲廣播每一筆金融交易和支付。通道餘額更新會透露支付已通過通道移動。這些資訊可能會被關聯以揭示所有支付來源和目的地。

- 為了擴展可以透過閃電網路進行的支付數量。請記住，閃電網路首先被創建是因為通知每個參與者每一筆支付無法很好地擴展。因此，閃電網路不能以在參與者之間共享通道餘額更新的方式設計。
- 閃電網路是一個動態系統。它不斷且頻繁地變化。節點被添加，其他節點被關閉，餘額變化等。即使一切都始終被傳達，資訊也只會在短時間內有效。事實上，資訊在收到時通常已經過時。

我們將在後面的章節中檢查 gossip 協議的細節。

現在，只需要知道 gossip 協議存在，並且它用於共享閃電網路的拓撲資訊。 這種拓撲資訊對於透過支付通道網路傳遞支付至關重要。

3.6.2. 路徑搜尋和路由

閃電網路上的支付沿著由連接一個參與者到另一個參與者的通道組成的「路徑」轉發，從支付來源到支付目的地。找到從來源到目的地路徑的過程稱為「路徑搜尋」。使用該路徑進行支付的過程稱為「路由」。

> 對閃電網路的一個常見批評是路由沒有解決，甚至是「無法解決」的問題。事實上，路由是微不足道的。另一方面，路徑搜尋是一個困難的問題。這兩個術語經常被混淆，需要明確定義以確定我們試圖解決哪個問題。

正如我們接下來將看到的，閃電網路目前使用「基於來源」的協議進行路徑搜尋，使用「洋蔥路由」協議進行支付路由。基於來源意味著支付的發送者必須找到一條穿過網路到達預期目的地的路徑。洋蔥路由意味著路徑的元素是分層的（像洋蔥一樣），每一層都加密，所以一次只能被一個節點看到。我們將在下一節討論洋蔥路由。

3.7. 基於來源的路徑搜尋

如果我們知道每個通道的確切通道餘額，我們可以使用任何計算機科學課上教授的標準路徑搜尋演算法輕鬆計算支付路徑。這甚至可以以優化支付給節點轉發支付的費用的方式來解決。

然而，所有通道的餘額資訊不是也不能被網路的所有參與者知道。我們需要更創新的路徑搜尋策略。

由於只有網路拓撲的部分資訊，路徑搜尋是一個真正的挑戰，對閃電網路這部分的積極研究仍在進行中。路徑搜尋問題在閃電網路中沒有「完全解決」的事實是對該技術的一個主要批評點。

> 對閃電網路中路徑搜尋的一個常見批評是它無法解決，因為它等同於 NP 完全的「旅行推銷員問題」（TSP），這是計算複雜性理論中的一個基本問題。事實上，閃電網路中的路徑搜尋不等同於 TSP，屬於不同類別的問題。每次我們要求 Google 給我們避開交通的駕駛方向時，我們都成功地解決了這些類型的問題（在資訊不完整的圖中進行路徑搜尋）。每次我們在閃電網路上路由支付時，我們也成功地解決了這個問題。

路徑搜尋和路由可以以多種不同的方式實現，多種路徑搜尋和路由演算法可以在閃電網路上共存，就像多種路徑搜尋和路由演算法存在於網際網路上一樣。基於來源的路徑搜尋是許多可能解決方案之一，在閃電網路目前的規模下是成功的。

閃電網路節點目前實現的路徑搜尋策略是迭代嘗試路徑，直到找到一條有足夠流動性來轉發支付的路徑。這是一個試錯的迭代過程，直到成功或找不到路徑。該演算法目前不一定會產生費用最低的路徑。雖然這不是最優的，肯定可以改進，但即使這種簡單的策略也運作得相當好。

這種「探測」由閃電網路節點或錢包完成，使用者不會直接看到。 使用者可能只有在支付沒有立即完成時才意識到正在進行探測。

> 在網際網路上，我們使用網際網路協議和 IP 轉發演算法將網際網路封包從發送者轉發到目的地。雖然這些協議具有允許網際網路主機協作找到資訊流經網際網路路徑的良好特性，但我們不能重用和採用這個協議來在閃電網路上轉發支付。與網際網路不同，閃電網路支付必須是「原子的」，通道餘額必須保持「私密」。此外，閃電網路中的通道容量變化頻繁，不像網際網路中連接容量相對靜態。這些約束需要新穎的策略。

當然，如果我們想向我們的直接通道夥伴支付，並且我們在通道一側有足夠的餘額來這樣做，路徑搜尋是微不足道的。在所有其他情況下，我們的節點使用來自 gossip 協議的資訊進行路徑搜尋。這包括目前已知的公開支付通道、已知節點、已知拓撲（已知節點如何連接）、已知通道容量，以及節點所有者設定的已知費用政策。

3.7.1. 洋蔥路由

閃電網路使用類似於著名的 Tor（洋蔥路由器）網路的「洋蔥路由協議」。 閃電網路中使用的洋蔥路由協議稱為 *SPHINX Mix Format*，[8]將在後面的章節中詳細解釋。

> 閃電網路的洋蔥路由 SPHINX Mix Format 在概念上只與 Tor 網路路由相似，但協議和實現與 Tor 網路中使用的完全不同。

用於路由的支付封包稱為「洋蔥」。[9]

讓我們用洋蔥的類比來跟隨路由支付。在從支付發送者（付款人）到支付目的地（收款人）的路線上，洋蔥沿著路徑從一個節點傳遞到另一個節點。發送者從中心向外構建整個洋蔥。首先，發送者為支付的（最終）接收者創建支付資訊，並用只有接收者可以解密的加密層對其加密。然後，發送者用路徑中「緊接最終接收者之前」的節點的指令包裹該層，並用只有該節點可以解密的層加密。

各層是用指令構建的，向後工作直到整個路徑被編碼在各層中。然後發送者將完整的洋蔥交給路徑中的第一個節點，該節點只能讀取最外層。每個節點剝掉一層，發現裡面的指令揭示路徑中的下一個節點，並將洋蔥傳遞下去。當每個節點剝掉一層時，它無法讀取洋蔥的其餘部分。它所知道的只是洋蔥剛從哪裡來和它要去哪裡，沒有任何關於誰是原始發送者或最終接收者的指示。

這種情況繼續，直到洋蔥到達支付目的地（收款人）。然後，目的地節點打開洋蔥，發現沒有更多層可以解密，可以讀取裡面的支付資訊。

> 與真正的洋蔥不同，當剝掉每一層時，節點會添加一些加密填充以保持洋蔥的大小對下一個節點相同。正如我們將看到的，這使得任何中間節點都無法知道路徑的大小（長度）、涉及多少節點進行路由、前面有多少節點或後面有多少節點。這增加了隱私，防止了瑣碎的流量分析攻擊。

閃電網路中使用的洋蔥路由協議具有以下特性：

- 中介節點只能看到它在哪個通道上收到洋蔥以及在哪個通道上轉發洋蔥。這意味著沒有路由節點可以知道誰發起了支付以及支付的目的地是誰。這是最重要的特性，它帶來高度的隱私。
- 洋蔥足夠小，可以放入單個 TCP/IP 封包甚至鏈路層（例如乙太網）幀中。這使得支付的流量分析變得更加困難，進一步增加了隱私。
- 洋蔥的構建使得無論處理節點在路徑上的位置如何，它們都將具有相同的長度。當每一層被「剝掉」時，洋蔥會用加密的「垃圾」資料填充以保持洋蔥的大小相同。這防止中介節點知道他們在路徑中的位置。
- 洋蔥在每一層都有 HMAC（基於雜湊的訊息認證碼），因此洋蔥的操縱被防止且實際上是不可能的。
- 洋蔥最多可以有約 26 跳，如果您願意也可以稱為洋蔥層。這允許足夠長的路徑。精確的可用路徑長度取決於分配給每跳路由負載的位元組數。
- 每一跳的洋蔥加密使用不同的臨時加密金鑰。如果金鑰（特別是節點的私鑰）在某個時間點洩露，攻擊者無法解密它們。簡單來說，金鑰永遠不會重用以實現更高的安全性。
- 錯誤可以使用相同的洋蔥路由協議從出錯節點發送回原始發送者。對於外部觀察者和中介節點，錯誤洋蔥與路由洋蔥無法區分。錯誤路由使用於找到有足夠容量成功路由支付的路徑的試錯「探測」方法成為可能。

洋蔥路由將在 洋蔥路由 中詳細檢查。

3.7.2. 支付轉發演算法

一旦支付的發送者找到了一條可能的跨網路路徑並構建了洋蔥，支付就會由路徑中的每個節點轉發。每個節點處理洋蔥的一層並將其轉發給路徑中的下一個節點。

每個中介節點收到一條稱為 `update_add_htlc` 的閃電網路訊息，其中包含支付雜湊和洋蔥。中介節點執行一系列步驟，稱為「支付轉發演算法」：

1. 節點解密洋蔥的外層並檢查訊息的完整性。
2. 它確認它可以根據通道費用和出站通道上的可用容量來滿足路由提示。
3. 它與入站通道上的通道夥伴一起更新通道狀態。
4. 它在洋蔥末端添加一些填充以保持其恆定長度，因為它從開頭移除了一些資料。
5. 它遵循路由提示在其出站支付通道上轉發修改後的洋蔥封包，也發送一條包含相同支付雜湊和洋蔥的 `update_add_htlc` 訊息。
6. 它與出站通道上的通道夥伴一起更新通道狀態。

當然，如果檢測到錯誤，這些步驟會被中斷和中止，並向 `update_add_htlc` 訊息的發起者發送錯誤訊息。錯誤訊息也被格式化為洋蔥，並在入站通道上向後發送。

隨著錯誤在路徑上的每個通道向後傳播，通道夥伴移除待處理支付，以與開始相反的方式回滾支付。

雖然如果支付沒有快速結算，支付失敗的可能性很高，但節點永遠不應該在洋蔥帶著錯誤返回之前沿著不同的路徑發起另一次支付嘗試。如果兩次支付嘗試最終都成功，發送者將支付兩次。

3.8. 點對點通訊加密

閃電網路協議主要是其參與者之間的點對點協議。正如我們在前面章節中看到的，網路中有兩個重疊的功能，形成兩個邏輯網路，共同構成「閃電網路」：

1. 一個廣泛的點對點網路，使用 gossip 協議傳播拓撲資訊，對等節點隨機相互連接。對等節點之間不一定有支付通道，所以他們不一定是通道夥伴。
2. 通道夥伴之間的支付通道網路。通道夥伴也 gossip 拓撲資訊，這意味著他們是 gossip 協議中的對等節點。

對等節點之間的所有通訊都透過稱為「閃電網路訊息」的訊息發送。這些訊息都是加密的，使用稱為「Noise 協議框架」的密碼學通訊框架。Noise 協議框架允許構建提供身份驗證、加密、前向保密和身份隱私的密碼學通訊協議。Noise 協議框架還用於許多流行的端到端加密通

訊系統，如 WhatsApp、WireGuard 和 I2P。更多資訊可以在 Noise 協議框架網站 (https://noiseprotocol.org) 找到。

在閃電網路中使用 Noise 協議框架確保網路上的每條訊息都經過身份驗證和加密，增加了網路的隱私性及其對流量分析、深度封包檢測和竊聽的抵抗力。然而，作為副作用，這使得協議開發和測試有點棘手，因為人們不能簡單地用封包擷取或網路分析工具（如 Wireshark）觀察網路。相反，開發者必須使用專門的外掛程式，從一個節點的角度解密協議，例如 *lightning dissector* (https://github.com/nayutaco/lightning-dissector)，一個 Wireshark 外掛程式。

3.9. 關於信任的思考

只要一個人遵循協議並保護其節點安全，在參與閃電網路時就沒有重大的資金損失風險。 然而，開啟通道時需要支付鏈上費用。 任何成本都應該伴隨著相應的收益。 在我們的案例中，Alice 承擔開啟通道成本的回報是，Alice 可以隨時在閃電網路上發送和（在將一些幣移動到通道另一端後）接收比特幣支付，並且她可以透過為其他人轉發支付來賺取比特幣費用。 Alice 知道理論上 Bob 可以在開啟後立即關閉通道，導致 Alice 產生鏈上關閉費用。 Alice 需要對 Bob 有少量的信任。 Alice 去過 Bob’s Cafe，Bob 顯然有興趣賣給她咖啡，所以 Alice 在這個意義上可以信任 Bob。 Alice 和 Bob 都有共同的利益。 Alice 決定回報足以讓她承擔創建與 Bob 通道的鏈上費用成本。 相比之下，Alice 不會向一個不請自來發電子郵件要求她開啟新通道的未知人開啟通道。

3.10. 與比特幣的比較

雖然閃電網路建立在比特幣之上並繼承了它的許多特性和屬性，但有一些重要的差異需要兩個網路的使用者注意。

其中一些差異是術語差異。還有架構差異和使用者體驗差異。在接下來的幾節中，我們將檢查差異和相似之處，解釋術語，並調整我們的期望。

3.10.1. 地址與發票，交易與支付

在使用比特幣進行典型支付時，使用者收到一個比特幣地址（例如，掃描網頁上的 QR 碼，或在即時訊息或電子郵件中從朋友那裡收到）。然後他們使用比特幣錢包創建交易，將資金發送到這個地址。

在閃電網路上，支付的接收者創建發票。閃電網路發票可以被視為類似於比特幣地址。預期接收者將閃電網路發票作為 QR 碼或字元字串給發送者，就像比特幣地址一樣。

發送者使用他們的閃電網路錢包支付發票，複製發票文字或掃描發票 QR 碼。閃電網路支付類似於比特幣「交易」。

然而，使用者體驗有一些差異。比特幣地址是「可重用的」。比特幣地址永不過期，如果地址的所有者仍然持有金鑰，其中的資金始終可以存取。發送者可以向以前使用的地址發送任意數量的比特幣，接收者可以發布單一靜態地址來接收許多支付。雖然出於隱私原因這違背了最佳實踐，但它在技術上是可能的，實際上相當常見。

然而，在閃電網路中，每張發票只能用於特定支付金額的一次支付。您不能支付更多或更少，您不能重複使用發票，發票有內建的過期時間。在閃電網路中，接收者必須為每次支付生成一張新發票，預先指定支付金額。有一個例外，一種稱為 *keysend* 的機制，我們將在 Keysend 自發付款 中檢查。

3.10.2. 選擇輸出與尋找路徑

要在比特幣網路上進行支付，發送者需要消耗一個或多個未花費的交易輸出（UTXO）。 如果使用者有多個 UTXO，他們（或者更確切地說是他們的錢包）將需要選擇使用哪個 UTXO。 例如，進行 1 BTC 支付的使用者可以使用單一價值 1 BTC 的輸出、兩個價值 0.25 BTC 和 0.75 BTC 的輸出，或四個各價值 0.25 BTC 的輸出。

在閃電網路上，支付不需要消耗輸入。相反，每次支付都會導致通道餘額的更新，在兩個通道夥伴之間重新分配。發送者將此視為將通道餘額從他們的通道一端「移動」到另一端，到他們的通道夥伴。閃電網路支付使用一系列通道從發送者路由到接收者。這些通道中的每一個都必須有足夠的容量來路由支付。

因為許多可能的通道和路徑可以用於進行支付，閃電網路使用者對通道和路徑的選擇在某種程度上類似於比特幣使用者對 UTXO 的選擇。

透過原子多路徑支付（AMP）和多部分支付（MPP）等技術，我們將在後續章節中回顧，多條閃電網路路徑可以聚合成單一原子支付，就像多個比特幣 UTXO 可以聚合成單一原子比特幣交易一樣。

3.10.3. 比特幣上的找零輸出與閃電網路上無找零

要在比特幣網路上進行支付，發送者需要消耗一個或多個未花費的交易輸出（UTXO）。UTXO 只能完整花費；它們不能被分割和部分花費。因此，如果使用者希望花費 0.8 BTC，但只有 1 BTC 的 UTXO，他們需要花費整個 1 BTC UTXO，發送 0.8 BTC 給接收者，0.2 BTC 作為找零返回給自己。0.2 BTC 找零支付創建了一個新的 UTXO，稱為「找零輸出」。

在閃電網路上，資金交易花費一些比特幣 UTXO，創建一個多重簽名 UTXO 來開啟通道。一旦比特幣被鎖定在該通道內，它的一部分可以在通道內來回發送，而不需要創建任何找零。 這是因為通道夥伴只是更新通道餘額，只有在通道最終使用通道關閉交易關閉時才創建新的 UTXO。

3.10.4. 挖礦費用與路由費用

在比特幣網路上，使用者支付費用給礦工，讓他們的交易被包含在區塊中。 這些費用支付給開採該特定區塊的礦工。 費用金額基於交易在區塊中使用的「位元組」大小，以及使用者希望該交易被開採的速度。 因為礦工通常會首先開採最有利可圖的交易，想要立即開採交易的使用者將支付每位元組「更高」的費用，而不著急的使用者將支付每位元組「更低」的費用。

在閃電網路上，使用者支付費用給其他（中介節點）使用者，以透過他們的通道路由支付。 為了路由支付，中介節點必須在他們擁有的兩個或更多通道中移動資金，以及為發送者的支付傳輸資料。通常，路由使用者會根據支付的「價值」向發送者收費，已建立最低「基本費用」

（每次支付的固定費用）和「費率」（與支付價值成比例的按比例費用）。因此，價值較高的支付路由成本較高，形成流動性市場，不同使用者對透過其通道路由收取不同的費用。

3.10.5. 根據流量變化的費用與宣布的費用

在比特幣網路上，礦工是追求利潤的，通常會在區塊中包含盡可能多的交易，同時保持在稱為「區塊權重」的區塊容量內。

如果佇列中的交易（稱為 *mempool*）多於一個區塊可以容納的，他們會首先開採每單位（位元組）「交易權重」支付最高費用的交易。 因此，當佇列中有許多交易時，使用者必須支付更高的費用才能被包含在下一個區塊中，否則他們必須等到佇列中的交易較少。 這自然導致費用市場的出現，使用者根據他們需要將交易包含在下一個區塊中的緊迫程度來支付。

比特幣網路上的稀缺資源是區塊中的空間。比特幣使用者競爭區塊空間，比特幣費用市場基於可用區塊空間。閃電網路中的稀缺資源是「通道流動性」（通道中可用於路由的資金容量）和「通道連接性」（通道可以到達多少連接良好的節點）。閃電網路使用者競爭容量和連接性；因此，閃電網路費用市場由容量和連接性驅動。

在閃電網路上，使用者向路由其支付的使用者支付費用。在經濟學術語中，路由支付無非是向發送者提供和分配容量。自然，對相同容量收取較低費用的路由器將更有吸引力進行路由。因此，存在一個費用市場，路由器就透過其通道路由支付的費用相互競爭。

3.10.6. 公開的比特幣交易與私密的閃電網路支付

在比特幣網路上，每筆交易都在比特幣區塊鏈上公開可見。雖然涉及的地址是假名的，通常不與身份關聯，但它們仍然被網路上的每個其他使用者看到和驗證。 此外，區塊鏈監控公司大規模收集和分析這些資料，並將其出售給感興趣的各方，如私人公司、政府和情報機構。

另一方面，閃電網路支付幾乎完全私密。通常，只有發送者和接收者完全知道特定支付的來源、目的地和交易金額。此外，接收者甚至可能不知道支付的來源。因為支付是洋蔥路由的，路由支付的使用者只知道支付金額，他們無法確定來源或目的地。

總結來說，比特幣交易是公開廣播的，永久儲存。閃電網路支付在幾個選定的對等節點之間執行，有關它們的資訊只在通道關閉前私密儲存。創建與比特幣上使用的類似的大規模監控和分析工具在閃電網路上將困難得多。

3.10.7. 等待確認與即時結算

在比特幣網路上，交易只有在被包含在區塊中後才結算，在這種情況下，它們被稱為在該區塊中「確認」。隨著更多區塊被開採，交易獲得更多「確認」並被認為更安全。

在閃電網路上，確認只對開啟和關閉鏈上通道有意義。一旦資金交易達到適當的確認數（例如 3 次），通道夥伴就認為通道已開啟。因為通道中的比特幣由管理該通道的智慧合約保護，支付在最終接收者收到後「即時」結算。 實際上，即時結算意味著支付只需要幾秒鐘即可執行和結算。與比特幣一樣，閃電網路支付是不可逆的。

最後，當通道關閉時，會在比特幣網路上進行交易；一旦該交易被確認，通道就被認為已關閉。

3.10.8. 發送任意金額與容量限制

在比特幣網路上，使用者可以向另一個使用者發送他們擁有的任意數量的比特幣，沒有容量限制。理論上，單筆交易可以發送高達 2100 萬比特幣作為支付。

在閃電網路上，使用者只能發送其特定通道一側目前存在的比特幣數量給通道夥伴。例如，如果使用者擁有一個通道，其一側有 0.4 BTC，另一個通道一側有 0.2 BTC，那麼他們用一筆支付可以發送的最大金額是 0.4 BTC。無論使用者目前在其比特幣錢包中有多少比特幣，這都是真的。

多部分支付（MPP）是一項功能，在上述範例中，允許使用者組合其 0.4 BTC 和 0.2 BTC 通道，用一筆支付發送最多 0.6 BTC。MPP 目前正在閃電網路上測試，預計在本書完成時將被廣泛使用。有關 MPP 的更多詳情，請參見 多部分付款。

如果支付被路由，路由路徑上的每個路由節點都必須擁有容量至少與路由的支付金額相同的通道。對於支付路由通過的每個通道，這都必須成立。路徑中容量最低的通道設定了整個路徑容量的上限。

因此，容量和連接性是閃電網路中關鍵且稀缺的資源。

3.10.9. 大額支付與小額支付的激勵

比特幣的費用結構與交易價值無關。 一筆 100 萬美元的交易與 1 美元的交易在比特幣上費用相同，假設類似的交易大小（以位元組計，更具體地說是 SegWit [隔離見證協議] 後的「虛擬」位元組）。 在閃電網路中，費用是固定基本費用加上交易價值的百分比。 因此，在閃電網路中，支付費用隨支付價值增加。 這些相反的費用結構創造了不同的激勵，並導致在交易價值方面的不同使用。 較高價值的交易在比特幣上會更便宜；因此，使用者會更傾向於用比特幣進行大額交易。同樣，在另一端，使用者會更傾向於用閃電網路進行小額交易。

3.10.10. 使用區塊鏈作為帳本與作為法院系統

在比特幣網路上，每筆交易最終都記錄在區塊鏈上的一個區塊中。 區塊鏈因此形成了自比特幣創建以來每筆交易的完整歷史，以及完全審計每一個存在的比特幣的方式。 一旦交易被包含在區塊鏈中，它就是最終的。 因此，不會出現爭議，在區塊鏈的特定點上，特定地址控制多少比特幣是明確的。

在閃電網路上，特定時間通道中的餘額只有兩個通道夥伴知道，只有在通道關閉時才對網路的其餘部分可見。 當通道關閉時，通道的最終餘額提交到比特幣區塊鏈，每個夥伴收到他們在該通道中的比特幣份額。 例如，如果開啟餘額是 Alice 支付的 1 BTC，Alice 向 Bob 支付了 0.3 BTC，那麼通道的最終餘額是 Alice 0.7 BTC，Bob 0.3 BTC。 如果 Alice 試圖透過向比特幣區塊鏈提交通道的開啟狀態來作弊，Alice 1 BTC，Bob 0 BTC，那麼 Bob 可以透過提交通道的真實

最終狀態來報復，並創建一筆懲罰交易，給他通道中的所有比特幣。 對於閃電網路，比特幣區塊鏈充當法院系統。 像一個機器人法官，比特幣記錄每個通道的初始和最終餘額，如果其中一方試圖作弊，則批准懲罰。

3.10.11. 離線與線上，非同步與同步

當比特幣使用者向目標地址發送資金時，他們不需要知道接收者的任何資訊。接收者可能離線或線上，發送者和接收者之間不需要互動。互動是在發送者和比特幣區塊鏈之間進行的。在比特幣區塊鏈上接收比特幣是一種「被動」和「非同步」活動，不需要接收者的任何互動或接收者在任何時候都線上。比特幣地址甚至可以離線生成，永遠不會在比特幣網路上「註冊」。只有花費比特幣需要互動。

在閃電網路中，接收者必須線上才能在支付過期之前完成支付。 接收者必須運行一個節點或有人代表他們運行節點（第三方託管人）。準確地說，發送者和接收者的節點在支付時都必須線上，並且必須協調。接收閃電網路支付是發送者和接收者之間的「主動」和「同步」活動，沒有大部分閃電網路或比特幣網路的參與（除了中介路由節點，如果有的話）。

閃電網路的同步和始終線上的性質可能是使用者體驗中最大的差異，這經常讓習慣於比特幣的使用者感到困惑。

3.10.12. 聰與毫聰

在比特幣網路上，最小單位是「聰」，不能再細分。閃電網路更靈活一些，閃電網路節點使用「毫聰」（千分之一聰）。這允許透過閃電網路發送微小的支付。單一毫聰支付可以透過支付通道發送，這個金額如此之小，應該恰當地被描述為「納米支付」。

當然，毫聰單位不能以這種精度在比特幣區塊鏈上結算。在通道關閉時，餘額四捨五入到最接近的聰。但在通道的生命週期內，數以百萬計的納米支付可以以毫聰級別進行。閃電網路突破了微支付障礙。

3.11. 比特幣和閃電網路的共同點

雖然閃電網路在許多方面與比特幣不同，包括在架構和使用者體驗方面，但它建立在比特幣之上並保留了比特幣的許多核心特徵。

3.11.1. 貨幣單位

比特幣網路和閃電網路都使用相同的貨幣單位：比特幣。閃電網路支付使用與比特幣交易完全相同的比特幣。作為一個含義，因為貨幣單位相同，貨幣限制也相同：少於 2100 萬比特幣。在比特幣的 2100 萬總比特幣中，一些已經作為閃電網路支付通道的一部分分配給 2-of-2 多重簽名地址。

3.11.2. 支付的不可逆性和最終性

比特幣交易和閃電網路支付都是不可逆和不可變的。任何一個系統都沒有「撤消」操作或「退款」。作為任何一個的發送者，您必須負責任地行動，但同樣，作為接收者，您可以保證交易的最終性。

3.11.3. 信任和對手方風險

與比特幣一樣，閃電網路只要求使用者信任數學、加密，以及軟體沒有任何關鍵錯誤。無論是比特幣還是閃電網路都不需要使用者信任個人、公司、機構或政府。 因為閃電網路位於比特幣之上並依賴比特幣作為其底層基礎層，很明顯閃電網路的安全模型歸結為比特幣的安全性。這意味著閃電網路在大多數情況下提供與比特幣大致相同的安全性，只有在某些狹窄情況下安全性略有降低。

3.11.4. 無需許可的操作

比特幣和閃電網路都可以由任何有網際網路存取和適當軟體（例如節點和錢包）的人使用。 任何一個網路都不要求使用者獲得第三方、公司、機構或政府的許可、審查或授權。政府可以在其管轄範圍內禁止比特幣或閃電網路，但不能阻止其全球使用。

3.11.5. 開源和開放系統

比特幣和閃電網路都是由去中心化的全球志願者社群構建的開源軟體系統，在開放許可下提供。兩者都基於開放和可互操作的協議，作為開放系統和開放網路運行。全球、開放、自由。

3.12. 結論

在本章中，我們研究了閃電網路的實際運作方式以及所有組成部分。我們檢查了構建、操作和關閉通道的每個步驟。我們研究了支付如何路由，最後，我們將閃電網路與比特幣進行了比較，分析了它們的差異和共同點。

在接下來的幾章中，我們將更詳細地重新審視所有這些主題。

4. 閃電網路節點軟體

正如我們在前幾章所見，閃電網路節點是參與閃電網路的電腦系統。閃電網路不是一個產品或公司；它是一套定義互通性基準的開放標準。因此，閃電網路節點軟體已由多家公司和社群團體開發。絕大多數閃電網路軟體都是_開源_的，這意味著原始碼是開放的，並以允許協作、分享和社群參與開發過程的方式授權。同樣地，我們將在本章介紹的閃電網路節點實作都是開源的，並且是協作開發的。

與比特幣不同，比特幣的標準是由軟體中的_參考實作_（Bitcoin Core）定義的，閃電網路的標準是由一系列稱為_閃電網路技術基礎_（*Basis of Lightning Technology*，*BOLT*）的標準文件定義的，可在 *lightning-rfc* 儲存庫 (https://github.com/lightningnetwork/lightning-rfc)找到。

閃電網路沒有參考實作，但有幾個相互競爭、符合 BOLT 標準且可互通的實作，由不同的團隊和組織開發。開發閃電網路軟體的團隊也參與 BOLT 標準的開發和演進。

閃電網路節點軟體和比特幣節點軟體的另一個主要區別是，閃電網路節點不需要與共識規則同步運作，並且可以擁有超出 BOLT 基準的擴展功能。因此，不同的團隊可能會追求各種實驗性功能，如果這些功能成功並被廣泛部署，可能會在之後成為 BOLT 的一部分。

在本章中，你將學習如何設定最流行的閃電網路節點實作的每個軟體套件。我們按字母順序呈現它們，以強調我們通常不偏好或背書任何一個。每個都有其優點和缺點，選擇哪一個取決於多種因素。由於它們是用不同的程式語言開發的（例如 Go、C 等），你的選擇也可能取決於你對特定語言和開發工具集的熟悉程度和專業知識。

4.1. 閃電網路開發環境

如果你是開發者，你會想要設定一個開發環境，其中包含用於編寫和執行閃電網路軟體的所有工具、函式庫和支援軟體。在這個高度技術性的章節中，我們將逐步介紹這個過程。如果材料變得過於密集，或者你實際上並沒有設定開發環境，那麼可以跳到下一章，那裡的技術性較低。

4.1.1. 使用命令列

本章以及本書大部分內容中的範例都使用命令列終端機。這意味著你在終端機中輸入命令並接收文字回應。此外，這些範例是在基於 Linux 核心和 GNU 軟體系統的作業系統上演示的，特別是最新的 Ubuntu 長期穩定版本（Ubuntu 20.04 LTS）。大多數範例可以在其他作業系統（如 Windows 或 macOS）上複製執行，只需對命令進行少量修改。作業系統之間最大的區別是安裝各種軟體函式庫及其先決條件的_套件管理器_。在給出的範例中，我們將使用 apt，這是 Ubuntu 的套件管理器。在 macOS 上，一個常用於開源開發的套件管理器是 Homebrew (https://brew.sh)，透過命令 brew 存取。

在這裡的大多數範例中，我們將直接從原始碼建構軟體。雖然這可能相當具有挑戰性，但它給了我們最大的能力和控制權。如果遇到困難，你也可以選擇使用 Docker 容器、預編譯套件或其他安裝機制！

> 在本章的許多範例中，我們將使用作業系統的命令列介面（也稱為 *shell*），透過_終端機_應用程式存取。shell 將首先顯示提示符作為準備好接收你命令的指示。然後你輸入命令並按 Enter 鍵，shell 會以一些文字和一個新的提示符回應，等待你的下一個命令。在你的系統上，提示符可能看起來不同，但在以下範例中，它用 $ 符號表示。在範例中，當你看到 $ 符號後面的文字時，不要輸入 $ 符號，而是輸入緊隨其後的命令。然後按 Enter 鍵執行命令。在範例中，每個命令後面的行是作業系統對該命令的回應。當你看到下一個 $ 前綴時，你就知道這是一個新命令，你應該重複這個過程。

為了保持一致性，我們在所有命令列範例中使用 bash shell。雖然其他 shell 會以類似的方式運作，並且你可以在沒有它的情況下執行所有範例，但一些 shell 腳本是專門為 bash shell 編寫的，可能需要一些更改或自訂才能在其他 shell 中執行。為了一致性，你可以在 Windows 和 macOS 上安裝 bash shell，而它在大多數 Linux 系統上預設已安裝。

4.1.2. 下載本書儲存庫

所有程式碼範例都可在本書的線上儲存庫中找到。由於儲存庫會盡可能保持更新，你應該始終在線上儲存庫中查找最新版本，而不是從紙本書或電子書中複製。

你可以訪問 GitHub (https://github.com/lnbook/lnbook) 並選擇右側的綠色 Code 按鈕，以 ZIP 壓縮包的形式下載儲存庫。

或者，你可以使用 git 命令在本地電腦上建立儲存庫的版本控制複製品，從而讓你可以與後續更改保持同步，而不需要再次下載整個儲存庫。按照 Git 專案 (https://git-scm.com)的說明下載並安裝 git。

要在你的電腦上建立儲存庫的本地副本，請按以下方式執行 git 命令：

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

你現在在一個名為 lnbook 的資料夾中擁有了本書儲存庫的完整副本。你會想要透過執行以下命令切換到新下載的目錄：

```
$ cd lnbook
```

所有後續範例將假設你在這個資料夾內執行命令。

4.2. Docker 容器

許多開發者使用_容器_，這是一種虛擬機器，用於安裝預先配置的作業系統和應用程式及所有必要的依賴項。大部分閃電網路軟體也可以使用容器系統（如 *Docker*）來安裝，可在 Docker 主頁 (https://docker.com)找到。容器安裝要容易得多，特別是對於那些不習慣命令列環境的人。

本書的儲存庫包含一系列 Docker 容器，可用於設定一致的開發環境，以便在任何系統上練習和複製範例。因為容器是一個完整的作業系統，以一致的配置執行，你可以確保範例將在你的電腦上運作，而不需要擔心依賴項、函式庫版本或配置差異。

Docker 容器通常被優化為小型的，即佔用最小的磁碟空間。然而，在本書中，我們使用容器來_標準化_環境並使其對所有讀者保持一致。此外，這些容器並不是要用來在後台執行服務的。相反，它們是用來測試範例並透過與軟體互動來學習的。因此，這些容器相當大，並附帶了大量的開發工具和實用程式。通常，Alpine 發行版用於 Linux 容器，因為它們的體積較小。儘管如此，我們提供的容器是基於 Ubuntu 建構的，因為更多的開發者熟悉 Ubuntu，對我們來說，這種熟悉度比體積更重要。

Docker 的安裝和使用及其命令在 Docker 基本安裝和使用 中有詳細說明。如果你不熟悉 Docker，現在是快速瀏覽該部分的好時機。

你可以在本書儲存庫的 *code/docker* 資料夾下找到最新的容器定義和建構配置。每個容器都在一個單獨的資料夾中，如下所示：

```
$ 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*
```

正如我們將在接下來幾節中看到的，你可以在本地建構這些容器，或者你可以從 *Docker Hub* (https://hub.docker.com/orgs/lnbook) 上本書的儲存庫拉取它們。以下各節將假設你已安裝 Docker 並熟悉 docker 命令的基本用法。

4.3. Bitcoin Core 和 Regtest

大多數閃電網路節點實作需要存取完整的比特幣節點才能運作。

安裝完整的比特幣節點並同步比特幣區塊鏈超出了本書的範圍，這本身就是一個相對複雜的工作。如果你想嘗試，請參考 *精通比特幣* (https://github.com/bitcoinbook/bitcoinbook)，「第 3 章：Bitcoin Core：參考實作」，其中討論了比特幣節點的安裝和操作。

比特幣節點可以在 `regtest` 模式下運作，其中節點建立一個本地模擬的比特幣區塊鏈用於測試目的。在以下範例中，我們將使用 regtest 模式，讓我們能夠演示閃電網路，而不需要同步比特幣節點或冒任何資金風險。

Bitcoin Core 的容器是 bitcoind。它被配置為在 regtest 模式下執行 Bitcoin Core，並每 10 秒挖掘 6 個新區塊。它的遠端程序呼叫（RPC）埠暴露在埠 18443 上，可以使用使用者名稱 regtest 和密碼 regtest 進行 RPC 呼叫。你也可以使用互動式 shell 並在本地執行 bitcoin-cli 命令。

4.3.1. 建構 Bitcoin Core 容器

讓我們準備 bitcoind 容器。最簡單的方法是從 *Docker Hub* 拉取最新的容器：

```
$ 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
```

或者，你可以從 *code/docker/bitcoind/Dockerfile* 中的本地容器定義自行建構容器。

如果你之前使用 pull 命令從 Docker Hub 拉取了容器，則不需要建構容器。

在本地建構容器將使用較少的網路頻寬，但會使用更多的 CPU 時間來建構。我們使用 docker build 命令來建構它：

```
$ 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"
]
```

如你所見，bitcoind 啟動並挖掘 101 個模擬區塊以啟動區塊鏈。這是因為根據比特幣共識規則，新挖出的比特幣在經過 100 個區塊之前是不能花費的。透過挖掘 101 個區塊，我們使第一個區塊的 coinbase 可以花費。之後，每 10 秒挖掘 6 個新區塊，以保持區塊鏈向前推進。

目前沒有交易。但我們有一些測試比特幣已經被挖掘到錢包中，可以使用。當我們將一些閃電網路節點連接到這條鏈時，我們會發送一些比特幣到它們的錢包，這樣我們就可以在閃電網路節點之間開設一些閃電網路通道。

與 bitcoin core 容器互動

同時，我們也可以透過向 bitcoind 容器發送 shell 命令來與之互動。容器正在向終端機發送日誌檔案，顯示 bitcoind 進程的挖礦過程。要與 shell 互動，我們可以在另一個終端機中使用 docker exec 命令發出命令。由於我們之前使用 name 參數命名了正在執行的容器，當我們執行 docker exec 命令時，可以透過該名稱引用它。首先，讓我們執行一個互動式 bash shell：

```
$ 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#
```

執行互動式 shell 讓我們進入容器「內部」。它以使用者 root 登入，如新 shell 提示符 root@e027fd56e31a:/bitcoind# 中的 root@ 前綴所示。如果我們發出 ps x 命令查看正在執行的進程，我們會看到 bitcoind 和腳本 mine.sh 都在背景執行。要退出此 shell，請按 Ctrl-D 或輸入 **exit**，你將返回到你的作業系統提示符。

除了執行互動式 shell，我們還可以發出在容器內執行的單個命令。為了方便，bitcoin-cli 命令有一個別名「cli」，它傳遞正確的配置。因此，讓我們執行它來向 Bitcoin Code 詢問區塊鏈資訊。我們執行 cli getblockchaininfo：

BASH

```
$ docker exec bitcoind cli getblockchaininfo
{
  "chain": "regtest",
  "blocks": 131,
  "headers": 131,
  "bestblockhash":
"2cf57aac35365f52fa5c2e626491df634113b2f1e5197c478d57378e5a146110",

[...]

  "warnings": ""
}
```

bitcoind 容器中的 cli 命令允許我們向 Bitcoin Core 節點發出 RPC 命令並獲取 JavaScript 物件表示法（JSON）編碼的結果。

此外，我們所有的 Docker 容器都預裝了一個名為 jq 的命令列 JSON 編碼器/解碼器。jq 幫助我們透過命令列或從腳本內處理 JSON 格式的資料。你可以使用 | 字元將任何命令的 JSON 輸出發送到 jq。這個字元以及這個操作稱為「管道」。讓我們將 pipe 和 jq 應用到前面的命令如下：

BASH

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

jq .blocks 指示 jq JSON 解碼器從 getblockchaininfo 結果中提取欄位 blocks。在我們的例子中，它提取並列印值 197，我們可以在後續命令中使用。

正如你將在以下章節中看到的，我們可以同時執行多個容器，然後單獨與它們互動。我們可以發出命令提取資訊，例如閃電網路節點公鑰，或採取行動，例如向另一個節點開設閃電網路通道。docker run 和 docker exec 命令，加上用於 JSON 解碼的 jq，是我們建構混合許多不同節點實作的工作閃電網路所需的全部。這使我們能夠在自己的電腦上嘗試各種實驗。

4.4. c-lightning 閃電網路節點專案

`c-lightning` 是一個輕量級、高度可自訂且符合標準的閃電網路協定實作，由 Blockstream 作為 Elements 專案的一部分開發。該專案是開源的，在 GitHub (https://github.com/ElementsProject/lightning) 上協作開發。

在以下各節中，我們將建構一個 Docker 容器，執行連接到我們之前建構的 bitcoind 容器的 `c-lightning` 節點。我們還將向你展示如何直接從原始碼配置和建構 `c-lightning` 軟體。

4.4.1. 建構 c-lightning 作為 Docker 容器

`c-lightning` 軟體發行版有一個 Docker 容器，但它設計用於在生產系統中執行 `c-lightning` 並與 bitcoind 節點一起執行。我們將使用一個稍微簡單的容器，配置為在演示目的下執行 `c-lightning`。

讓我們從本書的 Docker Hub 儲存庫拉取 `c-lightning` 容器：

```
$ 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
```

BASH

或者，我們可以從你之前下載到名為 lnbook 目錄中的本書檔案建構 `c-lightning` Docker 容器。和之前一樣，我們將在 code/docker 子目錄中使用 docker build 命令。我們將使用標籤 lnbook/c-lightning 標記容器映像，如下所示：

```
$ 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
```
BASH

我們的容器現在已建構並準備好執行。然而，在執行 `c-lightning` 容器之前，我們需要在另一個終端機中啟動 bitcoind 容器，因為 `c-lightning` 依賴於 bitcoind。我們還需要設定一個 Docker 網路，允許容器相互連接，就像它們位於同一個區域網路上一樣。

> Docker 容器可以透過 Docker 系統管理的虛擬區域網路相互「對話」。每個容器可以有一個自訂名稱，其他容器可以使用該名稱解析其 IP 位址並輕鬆連接到它。

4.4.2. 設定 Docker 網路

一旦設定了 Docker 網路，每次 Docker 啟動時（例如重新開機後），Docker 都會在我們的本地電腦上啟動該網路。因此，我們只需要使用 docker network create 命令設定一次網路。網路名稱本身並不重要，但它必須在我們的電腦上是唯一的。預設情況下，Docker 有三個名為 host、bridge 和 none 的網路。我們將把新網路命名為 lnbook 並像這樣建立它：

```
$ 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
```
BASH

如你所見，執行 docker network ls 給我們一個 Docker 網路列表。我們的 lnbook 網路已建立。我們可以忽略網路 ID，因為它是自動管理的。

4.4.3. 執行 bitcoind 和 c-lightning 容器

下一步是啟動 bitcoind 和 `c-lightning` 容器並將它們連接到 lnbook 網路。要在特定網路中執行容器，我們必須將 network 參數傳遞給 docker run。為了讓容器容易找到彼此，我們還將使用 name 參數給每個容器一個名稱。我們像這樣啟動 bitcoind：

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

你應該看到 bitcoind 啟動並開始每 10 秒挖掘區塊。讓它繼續執行並開啟一個新的終端機視窗來啟動 `c-lightning`。我們使用帶有 network 和 name 參數的類似 docker run 命令來啟動 `c-lightning`，如下所示：

```
$ 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
```
BASH

`c-lightning` 容器啟動並透過 Docker 網路連接到 bitcoind 容器。首先，我們的 `c-lightning` 節點將等待 bitcoind 啟動，然後它將等待 bitcoind 挖掘一些比特幣到其錢包中。最後，作為容器啟動的一部分，一個腳本將向 bitcoind 節點發送一個 RPC 命令，建立一筆交易，用 10 個測試 BTC 為 `c-lightning` 錢包注資。現在我們的 `c-lightning` 節點不僅在執行，而且還有一些測試比特幣可以使用！

正如我們對 bitcoind 容器所演示的，我們可以在另一個終端機中向 `c-lightning` 容器發出命令，以提取資訊、開設通道等。允許我們向 `c-lightning` 節點發出命令列指令的命令稱為 lightning-cli。這個 lightning-cli 命令在此容器中也有別名 cli。要獲取 `c-lightning` 節點的資訊，請在另一個終端機視窗中使用以下 docker exec 命令：

```
$ 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"
}
```

BASH

我們現在有了第一個在虛擬網路上執行並與測試比特幣區塊鏈通訊的閃電網路節點。稍後在本章中，我們將啟動更多節點並將它們相互連接，以進行一些閃電網路支付。

在下一節中，我們還將看看如何直接從原始碼下載、配置和編譯 `c-lightning`。這是一個可選的進階步驟，將教你如何使用建構工具，並允許你對 `c-lightning` 原始碼進行修改。有了這些知識，你可以編寫一些程式碼、修復一些錯誤或為 `c-lightning` 建立外掛程式。

如果你不打算深入研究閃電網路節點的原始碼或程式設計，可以完全跳過下一節。我們剛剛建構的 Docker 容器足以完成本書中的大多數範例。

4.4.4. 從原始碼安裝 c-lightning

`c-lightning` 開發者提供了從原始碼建構 `c-lightning` 的詳細說明。我們將遵循 GitHub 上的說明 (https://github.com/ElementsProject/lightning/blob/master/doc/INSTALL.md)。

4.4.5. 安裝先決條件函式庫和套件

這些安裝說明假設你在 Linux 或類似系統上使用 GNU 建構工具建構 `c-lightning`。如果不是這種情況，請在 Elements 專案儲存庫中查找適用於你作業系統的說明。

常見的第一步是安裝先決條件函式庫。我們使用 apt 套件管理器來安裝這些：

```
$ 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) ...

[...]
$
```

幾分鐘後和大量的螢幕活動之後，你將安裝好所有必要的套件和函式庫。許多這些函式庫也被其他閃電網路套件使用，並且是軟體開發的一般需求。

4.4.6. 複製 c-lightning 原始碼

接下來，我們將從原始碼儲存庫複製最新版本的 `c-lightning`。為此，我們將使用 git clone 命令，它會在你的本地機器上複製一個版本控制的副本，從而讓你可以與後續更改保持同步，而不需要再次下載整個儲存庫：

```bash
$ 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
```

我們現在在 *lightning* 子資料夾中有了 `c-lightning` 的副本，並且我們使用 cd（更改目錄）命令進入該子資料夾。

4.4.7. 編譯 c-lightning 原始碼

接下來，我們使用一組在許多開源專案中常見的_建構腳本_。這些建構腳本使用 configure 和 make 命令，讓 我們可以：

- 選擇建構選項並檢查必要的依賴項（configure）
- 建構並安裝可執行檔和函式庫（make）

使用 help 選項執行 configure 將向我們顯示所有可用選項：

```
$ ./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
```

在本範例中，我們不需要更改任何預設值。因此，我們不帶任何選項再次執行 configure 以使用預設值：

```
$ ./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
$
```

接下來，我們使用 make 命令來建構 `c-lightning` 專案的函式庫、元件和可執行檔。這部分需要幾分鐘才能完成，並且會大量使用你電腦的 CPU 和磁碟。預期會聽到風扇的噪音！執行 make：

BASH
```
$ make

cc -DBINTOPKGLIBEXECDIR="\"../libexec/c-lightning\"" -Wall -Wundef -Wmis...

[...]

cc   -Og  ccan-asort.o ccan-autodata.o ccan-bitmap.o ccan-bitops.o ccan-...
```

如果一切順利，你不會看到任何 ERROR 訊息阻止前述命令的執行。`c-lightning` 軟體套件已從原始碼編譯完成，我們現在準備安裝在前一步驟中建立的可執行元件：

```
$ 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
[...]
```

要驗證 lightningd 和 lightning-cli 命令是否已正確安裝，我們將向每個可執行檔詢問其版本資訊：

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

版本由最新發布版本（v0.10.1）組成，後面是發布後的更改數量（34），最後是一個精確識別修訂版的雜湊值（fe86c11）。你可能會看到與前面顯示的不同版本，因為軟體在本書出版後會繼續發展。然而，無論你看到什麼版本，命令執行並以版本資訊回應的事實意味著你已成功建構 `c-lightning` 軟體。

4.5. Lightning Network Daemon 節點專案

Lightning Network Daemon（LND）是由 Lightning Labs 完整實作的閃電網路節點。LND 專案提供了許多可執行應用程式，包括 lnd（守護程式本身）和 lncli（命令列工具）。LND 有幾個可插拔的後端鏈服務，包括 btcd（完整節點）、bitcoind（Bitcoin Core）和 Neutrino（一個新的實驗性輕客戶端）。LND 是用 Go 程式語言編寫的。該專案是開源的，在 GitHub (https://github.com/LightningNetwork/lnd) 上協作開發。

在接下來的幾節中，我們將建構一個 Docker 容器來執行 LND，從原始碼建構 LND，並學習如何配置和執行 LND。

4.5.1. LND Docker 容器

我們可以從本書的 Docker Hub 儲存庫拉取 LND 範例 Docker 容器：

```
$ 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
```

或者，我們可以在本地建構 LND 容器。該容器位於 *code/docker/lnd*。我們將工作目錄更改為 *code/docker* 並執行 docker build 命令：

```
$ 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
```

BASH

我們的容器現在準備好執行了。與我們之前建構的 `c-lightning` 容器一樣，LND 容器也依賴於正在執行的 Bitcoin Core 實例。和之前一樣，我們需要在另一個終端機中啟動 bitcoind 容器，並透過 Docker 網路將 LND 連接到它。我們已經設定了一個名為 lnbook 的 Docker 網路，並將在這裡再次使用它。

> 通常，每個節點運營者在自己的伺服器上執行自己的閃電網路節點和比特幣節點。對我們來說，單個 bitcoind 容器可以服務多個閃電網路節點。在我們的模擬網路上，我們可以執行多個閃電網路節點，所有節點都連接到處於 regtest 模式的單個比特幣節點。

4.5.2. 執行 bitcoind 和 LND 容器

和之前一樣，我們在一個終端機中啟動 bitcoind 容器，在另一個終端機中啟動 LND。如果你已經在執行 bitcoind 容器，則不需要重新啟動它。只需讓它繼續執行並跳過下一步。要在 lnbook 網路中啟動 bitcoind，我們像這樣使用 docker run：

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

BASH

接下來，我們啟動剛剛建構的 LND 容器。和之前一樣，我們需要將它附加到 lnbook 網路並給它一個名稱：

```bash
$ 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...
```

LND 容器啟動並透過 Docker 網路連接到 bitcoind 容器。首先，我們的 LND 節點將等待 bitcoind 啟動，然後它將等待 bitcoind 挖掘一些比特幣到其錢包中。最後，作為容器啟動的一部分，一個腳本將向 bitcoind 節點發送一個 RPC 命令，從而建立一筆交易，用 10 個測試 BTC 為 LND 錢包注資。

正如我們之前演示的，我們可以在另一個終端機中向容器發出命令，以提取資訊、開設通道等。允許我們向 lnd 守護程式發出命令列指令的命令稱為 lncli。同樣，在此容器中，我們提供了別名 cli，它以所有適當的參數執行 lncli。讓我們在另一個終端機視窗中使用 docker exec 命令獲取節點資訊：

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

[...]

}
```

我們現在在 lnbook 網路上執行了另一個閃電網路節點，並與 bitcoind 通訊。如果你仍在執行 `c-lightning` 容器，那麼現在有兩個節點在執行。它們還沒有相互連接，但我們很快就會將它們連接起來。

如果需要，你可以在同一個閃電網路上執行 LND 和 `c-lightning` 節點的任意組合。例如，要執行第二個 LND 節點，你可以使用不同的容器名稱發出 docker run 命令，如下所示：

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

在前面的命令中，我們啟動了另一個 LND 容器，命名為 lnd2。名稱完全由你決定，只要它們是唯一的。如果你不提供名稱，Docker 將透過隨機組合兩個英文單詞來構造一個唯一的名稱，例如「naughty_einstein」。這是我們撰寫這段文字時 Docker 為我們選擇的實際名稱。多

有趣！

在下一節中，我們將看看如何直接從原始碼下載和編譯 LND。這是一個可選的進階步驟，將教你如何使用 Go 語言建構工具，並允許你對 LND 原始碼進行修改。有了這些知識，你可以編寫一些程式碼或修復一些錯誤。

如果你不打算深入研究閃電網路節點的原始碼或程式設計，可以完全跳過下一節。我們剛剛建構的 Docker 容器足以完成本書中的大多數範例。

4.5.3. 從原始碼安裝 LND

在本節中，我們將從頭開始建構 LND。LND 是用 Go 程式語言編寫的。如果你想了解更多關於 Go 的資訊，搜尋 golang 而不是 go 以避免不相關的結果。因為它是用 Go 而不是 C 或 C++ 編寫的，所以它使用的「建構」框架與我們之前在 `c-lightning` 中看到的 GNU autotools/make 框架不同。不過不用擔心，安裝和使用 golang 工具非常容易，我們將在這裡展示每個步驟。Go 是一種適合協作軟體開發的出色語言，因為無論作者數量如何，它都能產生非常一致、精確且易於閱讀的程式碼。Go 專注且「極簡」，以一種鼓勵語言版本間一致性的方式。作為編譯語言，它也相當高效。讓我們開始吧。

我們將遵循 LND 專案文件 (https://github.com/lightningnetwork/lnd/blob/master/docs/INSTALL.md) 中的安裝說明。

首先，我們將安裝 golang 套件和相關函式庫。我們嚴格要求 Go 版本 1.13 或更高版本。官方 Go 語言套件作為二進位檔案從 Go 專案 (https://golang.org/dl)分發。為方便起見，它們也被打包為 Debian 套件，可透過 apt 命令獲得。你可以按照 Go 專案 (https://golang.org/dl)的說明，或者如 GitHub 關於 Go 語言的 wiki 頁面 (https://github.com/golang/go/wiki/Ubuntu)所述，在 Debian/Ubuntu Linux 系統上使用以下 apt 命令：

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

BASH

透過執行以下命令檢查你是否安裝並準備好使用正確的版本：

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

BASH

我們有 1.13.4，所以我們準備好…Go！接下來我們需要告訴任何程式在哪裡找到 Go 程式碼。這是透過設定環境變數 GOPATH 來完成的。通常，Go 程式碼位於使用者主目錄下一個名為 *gocode* 的目錄中。透過以下兩個命令，我們一致地設定 GOPATH 並確保你的 shell 將其添加到可執行 PATH 中。請注意，使用者的主目錄在 shell 中稱為 ~。

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

BASH

為了避免每次開啟 shell 時都必須設定這些環境變數，你可以使用你選擇的編輯器將這兩行添加到主目錄中 bash shell 配置檔案 *.bashrc* 的末尾。

4.5.4. 複製 LND 原始碼

與現今許多開源專案一樣，LND 的原始碼在 GitHub（*www.github.com*）上。go get 命令可以使用 Git 協定直接獲取它：

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

一旦 go get 完成，你將在 GOPATH 下有一個包含 LND 原始碼的子目錄。

4.5.5. 編譯 LND 原始碼

LND 使用 make 建構系統。要建構專案，我們切換目錄到 LND 的原始碼，然後像這樣使用 make：

```
$ cd $GOPATH/src/github.com/lightningnetwork/lnd
$ make && make install
```
BASH

幾分鐘後，你將安裝好兩個新命令 lnd 和 lncli。嘗試它們並檢查它們的版本以確保它們已安裝：

```
$ 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
```
BASH

你可能會看到與前面顯示的不同版本，因為軟體在本書出版後會繼續發展。然而，無論你看到什麼版本，命令執行並顯示版本資訊的事實意味著你已成功建構 LND 軟體。

4.6. Eclair 閃電網路節點專案

Eclair（法語中的閃電）是由 ACINQ 製作的閃電網路 Scala 實作。Eclair 也是最受歡迎和開創性的行動閃電網路錢包之一，我們在 入門指南 中使用它來演示閃電網路支付。在本節中，我們將檢視執行閃電網路節點的 Eclair 伺服器專案。Eclair 是一個開源專案，可在 GitHub (https://github.com/ACINQ/eclair) 上找到。

在接下來的幾節中，我們將建構一個 Docker 容器來執行 Eclair，就像我們之前對 `c-lightning` 和 LND 所做的那樣。我們還將直接從原始碼建構 Eclair。

4.6.1. Eclair Docker 容器

讓我們從 Docker Hub 儲存庫拉取本書的 Eclair 容器：

```bash
$ 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
```

或者，我們可以在本地建構容器。到目前為止，你幾乎是 Docker 基本操作的專家了！在本節中，我們將重複許多之前看到的命令來建構 Eclair 容器。該容器位於 *code/docker/eclair*。我們在終端機中首先將工作目錄切換到 *code/docker* 並發出 docker build 命令：

```bash
$ 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
```

我們的映像現在準備好執行了。Eclair 容器也依賴於正在執行的 Bitcoin Core 實例。和之前一樣，我們需要在另一個終端機中啟動 bitcoind 容器，並透過 Docker 網路將 Eclair 連接到它。我們已經設定了一個名為 lnbook 的 Docker 網路，並將在這裡重複使用它。

Eclair 與 LND 或 `c-lightning` 的一個顯著區別是，Eclair 不包含單獨的比特幣錢包，而是直接依賴 Bitcoin Core 中的比特幣錢包。回想一下，使用 LND 時，我們透過執行交易從 Bitcoin Core 的錢包轉移比特幣到 LND 的比特幣錢包來為其注資。使用 Eclair 時，這個步驟是不必要的。執行 Eclair 時，Bitcoin Core 錢包直接用作開設通道的資金來源。因此，與 LND 或 `c-lightning` 容器不同，Eclair 容器不包含在啟動時將比特幣轉入其錢包的腳本。

4.6.2. 執行 bitcoind 和 Eclair 容器

和之前一樣，我們在一個終端機中啟動 bitcoind 容器，在另一個終端機中啟動 Eclair 容器。如果你已經在執行 bitcoind 容器，則不需要重新啟動它。只需讓它繼續執行並跳過下一步。要在 lnbook 網路中啟動 bitcoind，我們像這樣使用 docker run：

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

BASH

接下來，我們啟動剛剛建構的 Eclair 容器。我們需要將它附加到 lnbook 網路並給它一個名稱，就像我們對其他容器所做的那樣：

```
$ 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

[...]
```

BASH

Eclair 容器啟動並透過 Docker 網路連接到 bitcoind 容器。首先，我們的 Eclair 節點將等待 bitcoind 啟動，然後它將等待 bitcoind 挖掘一些比特幣到其錢包中。

正如我們之前演示的，我們可以在另一個終端機中向容器發出命令，以提取資訊、開設通道等。允許我們向 eclair 守護程式發出命令列指令的命令稱為 eclair-cli。和之前一樣，在此容器中，我們提供了 eclair-cli 的有用別名，簡稱為 cli，它提供必要的參數。在另一個終端機視窗中使用 docker exec 命令，我們從 Eclair 獲取節點資訊：

```
BASH
$ 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"
}
```

我們現在在 lnbook 網路上執行了另一個閃電網路節點，並與 bitcoind 通訊。你可以在同一個閃電網路上執行任意數量和任意組合的閃電網路節點。任意數量的 Eclair、LND 和 `c-lightning` 節點都可以共存。例如，要執行第二個 Eclair 節點，你可以使用不同的容器名稱發出 docker run 命令，如下所示：

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

在前面的命令中，我們啟動了另一個名為 eclair2 的 Eclair 容器。

在下一節中，我們還將看看如何直接從原始碼下載和編譯 Eclair。這是一個可選的進階步驟，將教你如何使用 Scala 和 Java 語言建構工具，並允許你對 Eclair 的原始碼進行修改。有了這些知識，你可以編寫一些程式碼或修復一些錯誤。

如果你不打算深入研究閃電網路節點的原始碼或程式設計，可以完全跳過下一節。我們剛剛建構的 Docker 容器足以完成本書中的大多數範例。

4.6.3. 從原始碼安裝 Eclair

在本節中，我們將從頭開始建構 Eclair。Eclair 是用 Scala 程式語言編寫的，使用 Java 編譯器編譯。要執行 Eclair，我們首先需要安裝 Java 及其建構工具。我們將遵循 Eclair 專案的 *BUILD.md* 文件 (https://github.com/ACINQ/eclair/blob/master/BUILD.md)中的說明。

所需的 Java 編譯器是 OpenJDK 11 的一部分。我們還需要一個名為 Maven 的建構框架，版本 3.6.0 或更高。

在 Debian/Ubuntu Linux 系統上，我們可以使用 apt 命令安裝 OpenJDK 11 和 Maven，如下所示：

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

BASH

透過執行以下命令驗證你是否安裝了正確的版本：

```
$ 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
```

BASH

我們有 OpenJDK 11.0.7 和 Maven 3.6.1，所以我們準備好了。

4.6.4. 複製 Eclair 原始碼

Eclair 的原始碼在 GitHub 上。git clone 命令可以為我們建立一個本地副本。讓我們切換到主目錄並在那裡執行它：

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

BASH

一旦 git clone 完成，你將有一個名為 eclair 的子目錄，其中包含 Eclair 伺服器的原始碼。

4.6.5. 編譯 Eclair 原始碼

Eclair 使用 Maven 建構系統。要建構專案，我們將工作目錄更改為 Eclair 的原始碼，然後像這樣使用 mvn package：

```bash
$ 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] ------------------------------------------------------------------------
```

幾分鐘後，Eclair 套件的建構應該完成。然而，「package」操作也會執行測試，其中一些連接到網路，可能會失敗。如果你想跳過測試，請在命令中添加 -DskipTests。

現在，按照 GitHub 上的 安裝 Eclair 說明 (https://github.com/ACINQ/eclair#installing-eclair)解壓縮並執行建構的套件。

恭喜！你已從原始碼建構了 Eclair，你準備好編寫程式碼、測試、修復錯誤並為這個專案做出貢獻！

4.7. 建構多樣化閃電網路節點的完整網路

本節介紹的最後一個範例，將把我們建構的所有各種容器組合在一起，形成一個由多樣化（LND、`c-lightning`、Eclair）節點實作組成的閃電網路。我們將透過將節點相互連接並從一個節點到另一個節點開設通道來組建網路。作為最後一步，我們將透過這些通道路由一筆支付！

在這個範例中，我們將建構一個由四個閃電網路節點組成的演示閃電網路，分別命名為 Alice、Bob、Chan 和 Dina。我們將把 Alice 連接到 Bob，Bob 連接到 Chan，Chan 連接到 Dina。這在 四個節點的小型演示網路 中顯示。

Figure 16. 四個節點的小型演示網路

最後，我們將讓 Dina 建立一張發票，並讓 Alice 支付該發票。由於 Alice 和 Dina 沒有直接連接，支付將作為 HTLC 在所有支付通道上路由。

4.7.1. 使用 docker-compose 編排 Docker 容器

為了使這個範例運作，我們將使用一個_容器編排_工具，作為名為 docker-compose 的命令提供。這個命令允許我們指定一個由多個容器組成的應用程式，並透過一起啟動所有協作容器來執行應用程式。

首先，讓我們安裝 docker-compose。https://docs.docker.com/compose/install[說明]取決於你的作業系統。

安裝完成後，你可以透過像這樣執行 `docker-compose` 來驗證安裝：

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

最常見的 docker-compose 命令是 up 和 down，例如 docker-compose up。

4.7.2. docker-compose 配置

docker-compose 的配置檔案位於 *code/docker* 目錄中，名為 *docker-compose.yml*。它包含網路和四個容器中每個容器的規格。頂部看起來像這樣：

```
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
```

前面的片段定義了一個名為 lnnet 的網路和一個名為 bitcoind 的容器，該容器將附加到 lnnet 網路。這個容器與我們在本章開頭建構的相同。我們暴露了容器的三個埠，允許我們向它發送命令並監控區塊和交易。接下來，配置指定了一個名為「Alice」的 LND 容器。往下你還會看到名為「Bob」（`c-lightning`）、「Chan」（Eclair）和「Dina」（又是 LND）的容器規格。

由於所有這些不同的實作都遵循 BOLT 規範，並且已經過廣泛的互通性測試，因此它們可以毫無困難地一起工作來建構閃電網路。

4.7.3. 啟動範例閃電網路

在開始之前，我們應該確保沒有正在執行任何容器。如果新容器與已經執行的容器共用相同的名稱，它將無法啟動。根據需要使用 docker ps、docker stop 和 docker rm 來停止和刪除任何當前正在執行的容器！

> 因為我們對這些編排的 Docker 容器使用相同的名稱，我們可能需要「清理」以避免任何名稱衝突。

要啟動範例，我們切換到包含 *docker-compose.yml* 配置檔案的目錄，然後發出 docker-compose up 命令：

```
$ 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...

[...]
```

BASH

啟動後，你會看到每個節點啟動並報告其進度時的整個日誌流。在你的螢幕上可能看起來相當混亂，但每個輸出行都以容器名稱為前綴，如前所示。如果你想只觀看一個容器的日誌，可以在另一個終端機視窗中使用帶有 f（*follow*）標誌和特定容器名稱的 docker-compose logs 命令：

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

BASH

4.7.4. 開設通道和路由支付

我們的閃電網路現在應該正在執行。正如我們在本章前面章節中看到的，我們可以使用 docker exec 命令向正在執行的 Docker 容器發出命令。無論我們是使用 docker run 啟動容器還是使用 docker-compose up 啟動一堆容器，我們仍然可以使用 Docker 命令單獨存取容器。

支付演示包含在名為 run-payment-demo.sh 的 Bash shell 腳本中。要執行此演示，你必須在電腦上安裝 Bash shell。大多數 Linux 和類 Unix 系統（例如 macOS）都預裝了 bash。Windows 使用者可以安裝 Windows Subsystem for Linux 並使用像 Ubuntu 這樣的 Linux 發行版來在其電腦上獲得原生的 bash 命令。

讓我們執行腳本看看效果，然後我們將看看它內部是如何運作的。我們使用 bash 作為命令執行它：

```
$ 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:
lnbcrt100u1psnuzzrpp5rz5dg4wy27973yr7ehwns5ldeusceqdaq0hguu8c29n4nsqkznjsdqqcqzp
gxqyz5vqsp5vdpehw33fljnmmexa6ljk55544f3syd8nfttqlm3ljewu4r0q20q9qyyssqxh5nhkpjgf
m47yxn4p9ecvndz7zddlsgpufnpyjl0kmnq227tdujlm0acdv39hcuqp2vhs40aav70c9yp0tee6tgzk
8ut79mr877q0cpkjcfvr
=====================================================

Attempting payment from Alice to Dina
Successful payment!
```

如你從輸出中看到的，腳本首先獲取四個節點中每個節點的節點 ID（公鑰）。然後，它連接節點並從網路中每個節點到下一個節點建立一個 1,000,000 聰的通道。最後，它從 Dina 的節點發出一張 10,000 聰的發票，並從 Alice 的節點支付該發票。

如果腳本失敗，你可以嘗試從頭開始再次執行它。或者你可以手動逐一發出腳本中的命令並查看結果。

該腳本中有很多內容需要回顧，但隨著你對底層技術的理解加深，越來越多的資訊將變得清晰。歡迎你稍後再回顧這個範例。

當然，你可以用這個測試網路做的事情遠不止三通道、四節點的支付。以下是一些實驗的想法：

- 透過啟動更多不同類型的節點來建立更複雜的網路。編輯 *docker-compose.yml* 檔案並複製章節，根據需要重新命名容器。
- 以更複雜的拓撲結構連接節點：循環路由、輪輻式或全網狀。
- 執行大量支付以耗盡通道容量。然後以相反方向執行支付以重新平衡通道。看看路由演算法如何適應。
- 更改通道費用，看看路由演算法如何協商多條路由以及它應用什麼優化。便宜的長路由是否比昂貴的短路由更好？
- 執行一個從節點返回到自身的循環支付以重新平衡自己的通道。看看這如何影響所有其他通道和節點。
- 在迴圈中生成數百或數千張小額發票，然後在另一個迴圈中盡可能快地支付它們。測量你可以從這個測試網路中擠出多少每秒交易量。

Lightning Polar (https://lightningpolar.com) 允許你視覺化你一直在使用 Docker 進行實驗的網路。

4.8. 結論

在本章中，我們研究了實作 BOLT 規範的各種專案。我們建構了容器來執行一個範例閃電網路，並學習了如何從原始碼建構每個專案。你現在已準備好進一步探索和深入挖掘。

5. 操作閃電網路節點

讀到這裡，你可能已經設定了一個閃電網路錢包。在本章中，我們將更進一步，設定一個完整的閃電網路節點。除了設定之外，我們還將學習如何操作和維護它。

你可能想要設定自己的閃電網路節點的原因有很多，包括：

- 成為閃電網路的完整、積極參與者，而不僅僅是終端使用者
- 經營電子商務商店或透過閃電網路支付接收收入
- 透過閃電網路路由費用或出租通道流動性來賺取收入
- 為閃電網路開發新服務、應用程式或外掛程式
- 在使用閃電網路時增加你的財務隱私
- 使用建立在閃電網路之上的一些應用程式，例如閃電網路驅動的即時通訊應用程式
- 實現財務自由、獨立和主權

運行閃電網路節點有相關成本。你需要一台電腦、永久的網路連線、大量的磁碟空間和大量的時間！ 營運成本還包括電費。

但你從這次經歷中學到的技能是有價值的，也可以應用於各種其他任務。

讓我們開始吧！

> 重要的是，你要根據準確的事實正確設定自己的期望。 如果你計劃_僅僅_為了透過賺取路由費用來獲得收入而運行閃電網路節點， 請先認真做好功課。 透過運行閃電網路節點來經營盈利業務 絕對_不_容易。在試算表中計算所有初始和持續成本。仔細研究閃電網路統計資料。 目前的支付量是多少？每個節點的交易量是多少？目前的平均路由費用是多少？諮詢論壇並詢問 已經獲得實際經驗的其他社群成員的建議或回饋。只有在 完成這項盡職調查之_後_，才形成你自己有依據的意見。大多數人會發現他們運行節點的動機不在於經濟收益， 而在於其他地方。

5.1. 選擇你的平台

你可以透過多種方式運行閃電網路節點，從托管在家中的小型迷你電腦或專用伺服器，到雲端的托管伺服器。你選擇的方法將取決於你擁有的資源以及你想花多少錢。

5.1.1. 為什麼可靠性對運行閃電網路節點很重要？

在比特幣中，除非專門運行挖礦節點，否則硬體並不特別重要。 Bitcoin Core 節點軟體可以在任何滿足其最低要求的機器上運行，而且不需要在線才能接收付款——只有發送付款時才需要。 如果比特幣節點長時間離線，使用者只需重新啟動節點，一旦它連接到網路的其餘部分，

它就會重新同步區塊鏈。

然而，在閃電網路中，使用者需要同時在線才能發送_和_接收付款。如果閃電網路節點離線，它就無法從任何人那裡接收任何付款，因此其開立的發票無法履行。 此外，離線節點的開放通道不能用於路由付款。你的通道合作夥伴會注意到你離線且無法聯繫你來路由付款。如果你太常離線，他們可能會認為與你通道中鎖定的比特幣是未充分利用的容量，並可能關閉這些通道。我們已經討論過協定攻擊的情況，即你的通道合作夥伴試圖透過提交較早的承諾交易來欺騙你。如果你離線且你的通道沒有被監控，那麼企圖的盜竊可能會成功，一旦時間鎖到期，你將沒有追索權。 因此，節點可靠性對閃電網路節點極為重要。

還有硬體故障和資料遺失的問題。在比特幣中，如果使用者備份了助記詞或私鑰，硬體故障可能是一個微不足道的問題。比特幣錢包和錢包中的比特幣可以很容易地從私鑰在新電腦上恢復。大多數資訊可以從區塊鏈重新下載。

相比之下，在閃電網路中，關於使用者通道的資訊，包括承諾交易和撤銷秘密，不是公開的，只儲存在個別使用者的硬體上。 因此，閃電網路中的軟體和硬體故障很容易導致資金損失。

5.1.2. 硬體閃電網路節點的類型

硬體閃電網路節點主要有三種類型：

通用電腦

閃電網路節點可以在運行 Windows、macOS 或 Linux 的家用電腦或筆記型電腦上運行。通常這是與比特幣節點一起運行的。

專用硬體

閃電網路節點也可以在專用硬體上運行，如 Raspberry Pi、Rock64 或迷你電腦。這種設定通常會運行軟體堆疊，包括比特幣節點和其他應用程式。這種設定很受歡迎，因為硬體專門用於運行和維護閃電網路節點，通常使用安裝「助手」進行設定。

預配置硬體

閃電網路節點也可以在專門為其選擇和配置的專用硬體上運行。這包括可以作為套件或交鑰匙系統購買的「開箱即用」閃電網路節點解決方案。

5.1.3. 在「雲端」運行

虛擬私人伺服器（VPS）和雲端運算服務，如 Microsoft Azure、Google Cloud、Amazon Web Services（AWS）或 DigitalOcean，價格相當實惠，可以非常快速地設定。閃電網路節點可以在這樣的服務上以每月 20 到 40 美元的價格托管。

然而，俗話說，「『雲端』只是別人的電腦。」使用這些服務意味著在別人的電腦上運行你的節點。這帶來了相應的優缺點。主要優點是便利性、效率、正常運行時間，甚至可能是成本。雲端運營商在很大程度上管理和運行節點，自動為你提供便利性和效率。他們提供出色的正常運行時間和可用性，通常比個人在家中可以達到的要好得多。如果你考慮到在許多西方國家僅運

行伺服器的電費就約為每月 10 美元，然後加上網路頻寬的成本和硬體本身，VPS 產品在經濟上具有競爭力。最後，使用 VPS，你不需要在家裡為電腦騰出空間，也沒有電腦噪音或熱量的問題。 另一方面，有幾個顯著的缺點。在「雲端」運行的閃電網路節點總是比在你自己的電腦上運行的節點安全性和隱私性更低。此外，這些雲端運算服務非常集中化。在這些服務上運行的絕大多數比特幣和閃電網路節點位於維吉尼亞州、桑尼維爾、西雅圖、倫敦和法蘭克福的少數資料中心。當這些提供商的網路或資料中心出現服務問題時，它會影響所謂「去中心化」網路上的數千個節點。

如果你有可能和能力在家裡或辦公室的自己電腦上運行節點，那麼這可能比在雲端運行更可取。儘管如此，如果運行自己的伺服器不是一個選項，請務必考慮在 VPS 上運行一個。

5.1.4. 在家運行節點

如果你在家裡或辦公室有合理容量的網路連線，你當然可以在那裡運行閃電網路節點。任何「寬頻」連線都足以運行輕量級節點，快速連線也可以讓你運行比特幣完整節點。

雖然你可以在筆記型電腦上運行閃電網路節點（甚至比特幣節點），但很快就會變得煩人。這些程式會消耗你電腦的資源，需要 24/7 全天候運行。你的使用者應用程式，如瀏覽器或試算表，將與閃電網路背景服務競爭你電腦的資源。換句話說，你的瀏覽器和其他桌面工作負載會變慢。 當你的文書處理應用程式使你的筆記型電腦當機時，你的閃電網路節點也會當機，使你無法接收交易，並可能容易受到攻擊。此外，你永遠不應該關閉你的筆記型電腦。 所有這些加在一起導致的設定不是很理想。同樣的情況也適用於你日常使用的個人桌上型電腦。

相反，大多數使用者會選擇在專用電腦上運行節點。 幸運的是，你不需要一台「伺服器」級的電腦來做到這一點。 你可以在單板電腦上運行閃電網路節點，例如 Raspberry Pi 或迷你電腦（通常作為家庭影院電腦銷售）。 這些是通常用作家庭自動化中心或媒體伺服器的簡單電腦。與電腦或筆記型電腦相比，它們相對便宜。 專用設備作為閃電網路和比特幣節點平台的優勢在於，它可以在你的家庭網路上持續、靜默且不引人注目地運行，藏在路由器或電視後面。 沒有人會知道這個小盒子實際上是全球銀行系統的一部分！

> 不建議在 32 位元作業系統和/或 32 位元 CPU 上操作節點，因為節點軟體可能會遇到資源問題，導致當機並可能造成資金損失。

5.1.5. 運行閃電網路節點需要什麼硬體？

運行閃電網路節點至少需要以下條件：

CPU

需要足夠的處理能力來運行比特幣節點，它會持續下載和驗證新區塊。使用者還需要在設定新比特幣節點時考慮初始區塊下載（IBD），這可能需要幾個小時到幾天的時間。建議使用 2 核心或 4 核心 CPU。

RAM

具有 2 GB RAM 的系統_勉強_可以運行比特幣和閃電網路節點。如果至少有 4 GB RAM，效能會好得多。如果 RAM 少於 4 GB，IBD 將特別具有挑戰性。超過 8 GB 的 RAM 是不必要的，因為對於這些類型的服務，CPU 是更大的瓶頸，這是由於簽名驗證等密碼學操作。

儲存驅動器

這可以是硬碟驅動器（HDD）或固態驅動器（SSD）。 SSD 運行節點會明顯更快（但更貴）。大部分儲存空間用於比特幣區塊鏈，大小為數百 GB。 一個合理的權衡（成本與複雜性）是購買一個小型 SSD 來啟動作業系統，一個較大的 HDD 來儲存大型資料物件（主要是資料庫）。

由於成本和零件可用性，Raspberry Pi 是運行節點軟體的常見選擇。 在設備上運行的作業系統通常從安全數位（SD）卡啟動。 對於大多數使用情況，這不是問題，但 Bitcoin Core 以 I/O 密集著稱。 你應該確保將比特幣區塊鏈和閃電網路資料目錄放在不同的驅動器上，因為長期密集的 I/O 可能會導致 SD 卡故障。

網路連線

需要可靠的網路連線來下載新的比特幣區塊，以及與其他閃電網路對等節點通訊。在運行期間，估計的資料使用量範圍從每月 10 到 100 GB，取決於配置。在啟動時，比特幣完整節點會下載完整的區塊鏈。

電源供應

需要可靠的電源供應，因為閃電網路節點需要始終保持在線。電源故障將導致進行中的付款失敗。對於繁忙的路由節點，在停電時備用或不間斷電源（UPS）很有用。 理想情況下，你也應該將網路路由器連接到這個 UPS。

備份

備份至關重要，因為故障可能導致資料遺失，從而導致資金損失。 你會想要考慮某種資料備份解決方案。這可以是基於雲端的自動備份到你控制的伺服器或網路服務。或者，它可以是自動本地硬體備份，例如第二個硬碟。為了獲得最佳結果，可以結合本地和遠端備份。

5.1.6. 在雲端切換伺服器配置

當租用雲端伺服器時，在兩個運行階段之間更改配置通常很划算。在 IBD 期間（例如第一天）需要更快的 CPU 和更快的儲存。區塊鏈同步後，CPU 和儲存速度要求就低得多，因此可以將效能降級到更具成本效益的水平。

例如，在 Amazon 的雲端上，我們會在 IBD 期間使用 8-16 GB RAM、8 核心 CPU（例如 t3-large 或 m3.large）和更快的 400 GB SSD（1000+ 配置的每秒輸入/輸出操作數 [IOPS]），將其時間縮短到僅 6-8 小時。完成後，我們會將伺服器實例切換為 2 GB RAM、2 核心 CPU（例如 t3.small）和儲存改為通用 1 TB HDD。這大約與你在較慢的伺服器上運行整個時間的成本相同，但它會讓你在不到一天內啟動並運行，而不是必須等待將近一週的 IBD。

永久資料儲存（驅動器）

如果你使用迷你電腦或租用伺服器，儲存可能是最昂貴的部分，成本可能與電腦和連線（資料）加在一起一樣多。

讓我們看看可用的不同選項。首先，有兩種主要類型的驅動器，HDD 和 SSD。HDD 更便宜，SSD 更快，但兩者都能完成工作。

目前最快的 SSD 使用非揮發性記憶體快速通道（NVMe）介面。NVMe SSD 在高端機器中更快，但也更貴。 傳統的基於 SATA 的 SSD 更便宜，但沒有那麼快。SATA SSD 對於你的節點設定來說效能已經足夠好了。 較小的電腦可能無法利用 NVMe SSD 的優勢。 例如，Raspberry Pi 4 由於其 USB 埠的頻寬有限而無法從中受益。

要選擇大小，讓我們看看比特幣區塊鏈。截至 2021 年 8 月，其大小為 360 GB，包括交易索引，每年大約增長 60 GB。如果你想為未來的增長留有一些餘地或在你的節點上安裝其他資料，請至少購買 512 GB 的驅動器，或者更好的是 1 TB 的驅動器。

5.2. 使用安裝程式或助手

如果你不熟悉命令列環境，安裝閃電網路節點或比特幣節點可能會令人望而生畏。幸運的是，有許多專案製作「助手」，即為你安裝和配置各種元件的軟體。你仍然需要學習一些命令列咒語來與你的節點互動，但大部分初始工作已經為你完成。

5.2.1. RaspiBlitz

最受歡迎和完整的「助手」之一是 *RaspiBlitz*（RaspiBlitz 節點），這是由 Christian Rotzoll 建立的專案。它旨在安裝在 Raspberry Pi 4 上。RaspiBlitz 附帶推薦的硬體套件，你可以在幾個小時或最多一個週末內組裝完成。如果你參加你所在城市的閃電網路「黑客松」，你很可能會看到很多人在他們的 RaspiBlitz 設定上工作，交換技巧並互相幫助。你可以在 GitHub (https://github.com/rootzoll/raspiblitz) 上找到 RaspiBlitz 專案。

除了比特幣和閃電網路節點之外，RaspiBlitz 還可以安裝許多額外的服務，例如：

- Tor（作為隱藏服務運行）
- ElectRS（Rust 中的 Electrum 伺服器）
- BTCPay Server（加密貨幣支付處理器）
- BTC RPC Explorer（比特幣區塊鏈瀏覽器）
- Ride The Lightning（閃電網路節點管理圖形介面）
- LNbits（閃電網路錢包/帳戶系統）
- Specter Desktop（多重簽名 Trezor、Ledger、Coldcard 錢包和 Specter-DIY）
- lndmanage（用於進階通道管理的命令列介面）
- Loop（潛水艇交換服務）

- JoinMarket（CoinJoin 服務）

Figure 17. RaspiBlitz 節點

5.2.2. Mynode

myNode (https://mynodebtc.com) 是另一個流行的開源「助手」專案，包含許多與比特幣相關的軟體。它易於安裝：你將安裝程式「刷入」SD 卡，然後從 SD 卡啟動你的迷你電腦。你不需要任何顯示器來使用 myNode，因為管理工具可以從瀏覽器遠端存取。如果你的迷你電腦沒有顯示器、滑鼠或鍵盤，你可以從另一台電腦甚至從你的智慧型手機管理它。安裝後，前往 http://mynode.local 並點擊兩下建立閃電網路錢包和節點。

除了比特幣和閃電網路節點之外，myNode 還可以選擇性地安裝各種額外的服務，例如：

- Ride The Lightning（閃電網路節點管理圖形介面）
- OpenVPN（用於遠端管理或錢包的虛擬私人網路 [VPN] 支援）
- lndmanage（用於進階通道管理的命令列介面）
- BTC RPC Explorer（比特幣區塊鏈瀏覽器）

5.2.3. Umbrel

以其 UX/UI 聞名（如 Umbrel 網頁介面 所示），*Umbrel* 提供了一種非常簡單且易於使用的方式來快速啟動和運行你的比特幣和閃電網路節點，特別是對於初學者。一個非常獨特的功能是 Umbrel 在 IBD 期間使用 Neutrino/SPV，這樣你可以立即開始使用你的節點。一旦 Bitcoin Core 在背景完全同步，它會自動切換並停用 SPV 模式。Umbrel OS 支援 Raspberry Pi 4，也可以安裝在任何基於 Linux 的作業系統上，或者在 macOS 或 Windows 上的虛擬機器中。你還可以連接任何支援 Bitcoin Core P2P、Bitcoin Core RPC、Electrum 協定或 lndconnect 的錢包。

不需要等待雨天—你可以直接前往 Umbrel (https://getumbrel.com) 了解更多資訊。

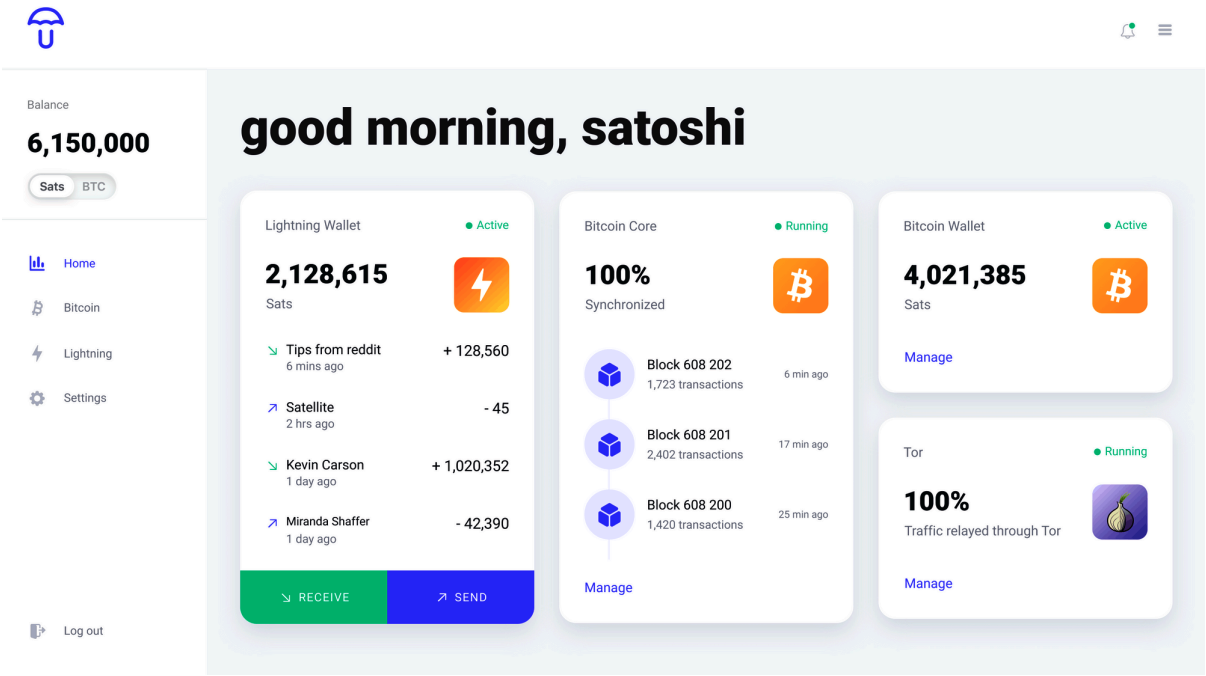

Figure 18. Umbrel 網頁介面

除了比特幣和閃電網路節點之外，Umbrel 推出了 Umbrel App Store，你可以在其中輕鬆安裝額外的服務，例如：

- Lightning Terminal（用於管理通道流動性、Loop In 和 Loop Out 的介面）
- Ride The Lightning（閃電網路節點管理圖形介面）
- Specter Desktop（用於多重簽名和單金鑰比特幣錢包的唯讀協調器）
- BTCPay Server（加密貨幣支付處理器）
- BTC RPC Explorer（比特幣區塊鏈瀏覽器）
- ThunderHub（監控和管理你的節點）
- Sphinx Relay（處理 Sphinx 聊天的連接和儲存）
- mempool.space（記憶體池視覺化器和區塊瀏覽器）
- LNbits（閃電網路錢包/帳戶系統）

Umbrel 目前仍處於測試階段，不被認為是安全的。

5.2.4. BTCPay Server

雖然最初不是作為安裝「助手」設計的，但電子商務和支付平台 *BTCPay Server* 有一個非常簡單的安裝系統，使用 Docker 容器和 docker-compose 來安裝比特幣節點、閃電網路節點和支付閘道，以及許多其他服務。它可以安裝在各種硬體平台上，從簡單的 Raspberry Pi 4（建議 4 GB）到迷你電腦或舊筆記型電腦、桌上型電腦或伺服器。

BTCPay Server (https://btcpayserver.org) 是一個功能齊全的自托管、自保管電子商務平台，可以與許多電子商務平台整合，例如 WordPress WooCommerce 等。完整節點的安裝只是電子商務平台安裝的一個步驟。 雖然最初是作為 *BitPay* 商業支付服務和 API 的功能對功能替代品開發的，但它已經超越了這一點，成為與電子商務相關的 BTC 和閃電網路服務的完整平台。對於許多賣家或商店來說，這是一個電子商務的一站式交鑰匙解決方案。

除了比特幣和閃電網路節點之外，BTCPay Server 還可以安裝各種服務，包括：

- `c-lightning` 或 LND 閃電網路節點
- Litecoin 支援
- Monero 支援
- Spark 伺服器（`c-lightning` 網頁錢包）
- Charge 伺服器（`c-lightning` 電子商務 API）
- Ride The Lightning（閃電網路節點管理網頁圖形介面）
- 許多 BTC 分叉
- BTCTransmuter（支援貨幣兌換的事件-動作自動化服務）

額外服務和功能的數量正在快速增長，因此前面的列表只是 BTCPay Server 平台上可用內容的一小部分。

5.2.5. 比特幣節點還是輕量級閃電網路

你設定的一個關鍵選擇將是比特幣節點及其配置的選擇。*Bitcoin Core*，參考實作，是最常見的選擇，但不是唯一可用的選擇。一個替代選擇是 *btcd*，它是比特幣節點的 Go 語言實作。btcd 支援一些對運行 LND 閃電網路節點有用的功能，這些功能在 Bitcoin Core 中不可用。

第二個考慮因素是你是否會運行一個帶有區塊鏈完整副本（2021 年中約 350 GB）的_存檔_比特幣節點，還是一個只保留最近區塊的_修剪_區塊鏈。修剪的區塊鏈可以為你節省一些磁碟空間，但你仍然需要至少下載一次完整的區塊鏈（在 IBD 期間）。因此，它不會為你節省任何網路流量。使用修剪節點運行閃電網路節點仍然是一個實驗性功能，可能不支援所有功能。然而，許多人正在成功運行這樣的節點。

最後，你也可以選擇根本不運行比特幣節點。相反，你可以使用 Neutrino 協定以「輕量級」模式操作 LND 閃電網路節點，從其他人操作的公共比特幣節點檢索區塊鏈資訊。這樣運行意味著你在沒有提供任何回報的情況下從比特幣網路獲取資源。相反，你是在提供你的資源並為閃電網路社群做出貢獻。對於較小的閃電網路節點，與運行完整比特幣節點相比，這通常會減少網路流量。

請記住，操作比特幣節點允許你支援其他服務，除了閃電網路節點之外。這些其他服務可能需要存檔（非修剪）比特幣節點，通常無法在沒有比特幣節點的情況下運行。預先考慮你現在或將來可能想要運行的其他服務，以便對你選擇的比特幣節點類型做出明智的決定。

這個決定的底線是：如果你能負擔得起大於 500 GB 的磁碟，請運行完整的存檔比特幣節點。你將為比特幣系統貢獻資源，並幫助那些負擔不起這樣做的人。如果你負擔不起這麼大的磁碟，運行修剪節點。如果你連修剪節點的磁碟或頻寬都負擔不起，運行一個使用 Neutrino 的輕量級 LND 節點。

5.2.6. 作業系統選擇

下一步是為你的節點選擇一個作業系統。絕大多數網路伺服器運行某種 Linux 變體。Linux 是網路的首選平台，因為它是一個強大的開源作業系統。然而，Linux 有一個陡峭的學習曲線，需要熟悉命令列環境。對於新使用者來說，它通常令人望而生畏。

最終，大多數服務可以在任何現代 POSIX 作業系統上運行，包括 macOS、Windows，當然還有 Linux。你的選擇應該更多地取決於你對作業系統的熟悉程度和舒適度以及你的學習目標。如果你想擴展你的知識並學習如何操作 Linux 系統，這是一個很好的機會，可以透過特定專案和明確目標來實現。如果你只想讓節點啟動並運行，就選擇你熟悉的。

如今，許多服務也以容器的形式提供，通常基於 Docker 系統。這些容器可以部署在各種作業系統上，抽象化底層作業系統。儘管如此，你可能仍然需要學習一些 Linux CLI 命令，因為大多數容器內部運行某種 Linux 變體。

5.3. 選擇你的閃電網路節點實作

與作業系統的選擇一樣，你對閃電網路節點實作的選擇應該主要取決於你對專案使用的程式語言和開發工具的熟悉程度。雖然各種節點實作之間在功能上有一些小差異，但這些差異相對較小，大多數實作都趨向於 BOLT 定義的共同標準。

另一方面，對程式語言和建構系統的熟悉是選擇節點的良好基礎。這是因為安裝、配置、持續維護和故障排除都將涉及與建構系統使用的各種工具互動。這包括：

- `c-lightning` 的 Make、Autotools 和 GNU 實用程式
- LND 的 Go 實用程式
- Eclair 的 Java/Maven

程式語言不僅影響建構系統的選擇，還影響程式的許多其他方面。每種程式語言都帶有完整的設計理念，並影響許多其他方面，例如：

- 配置檔案的格式和語法
- 檔案位置（在檔案系統中）
- 命令列參數及其語法
- 錯誤訊息格式
- 先決條件函式庫
- 遠端程序呼叫介面

當你選擇閃電網路節點時，你也在選擇所有上述特性。因此，你對這些工具和設計理念的熟悉程度將使運行節點變得更容易。或者更難，如果你落入一個不熟悉的領域。

另一方面，如果這是你第一次進入命令列和伺服器/服務環境，你會發現自己對任何實作都不熟悉，並有機會學習全新的東西。在這種情況下，你可能想根據一些其他因素來決定，例如：

- 支援論壇和聊天室的品質
- 文件的品質
- 與你想要運行的其他工具的整合程度

作為最後的考慮，你可能想要檢查不同節點實作的效能和可靠性。如果你將在生產環境中使用此節點並期望高流量和高可靠性要求，這一點尤其重要。如果你計劃在其上運行商店的支付系統，情況可能就是這樣。

5.4. 安裝比特幣或閃電網路節點

你決定不使用安裝「助手」，而是深入 Linux 作業系統的命令列？這是一個勇敢的決定，我們會盡力幫助你成功。如果你不想嘗試手動執行此操作，請考慮使用幫助你安裝節點軟體的應用程式或基於容器的解決方案，如 使用安裝程式或助手 中所述。

> 本節將深入探討從命令列進行系統管理的進階主題。Linux 管理是一項超出本書範圍的獨立技能集。這是一個複雜的主題，有許多陷阱。請謹慎進行！

在接下來的幾節中，我們將簡要描述如何在 Linux 作業系統上安裝和配置比特幣和閃電網路節點。你需要查看你決定使用的特定比特幣和閃電網路節點應用程式的安裝說明。你通常可以在名為 *INSTALL* 的檔案或每個專案的 *docs* 子目錄中找到這些說明。我們只會描述適用於所有此類服務的一些常見步驟，我們提供的說明必然是不完整的。

5.4.1. 背景服務

對於那些習慣在桌面或智慧型手機上運行應用程式的人來說，應用程式總是有圖形使用者介面，即使它有時可能在背景運行。然而，比特幣和閃電網路節點應用程式非常不同。這些應用程式沒有內建的圖形使用者介面。相反，它們作為_無頭_背景服務運行，這意味著它們始終在背景操作，不直接與使用者互動。

這可能會給不習慣運行背景服務的使用者帶來一些困惑。你如何知道這樣的服務目前是否正在運行？你如何啟動和停止它？你如何與它互動？這些問題的答案取決於你使用的作業系統。現在我們假設你使用某種 Linux 變體，並在該環境下回答這些問題。

5.4.2. 程序隔離

背景服務通常在特定的使用者帳戶下運行，以將它們與作業系統和彼此隔離。例如，Bitcoin Core 被配置為以使用者 bitcoin 運行。你需要使用命令列為你運行的每個服務建立一個使用者。

此外，如果你連接了外部驅動器，你需要告訴作業系統將使用者的主目錄重新定位到該驅動器。這是因為像 Bitcoin Core 這樣的服務會在使用者的主目錄下建立檔案。如果你將其設定為下載完整的比特幣區塊鏈，這些檔案將佔用數百 GB。在這裡，我們假設你已經連接了外部驅動器，並且它位於作業系統的 */external_drive/* 路徑上。

在大多數 Linux 系統上，你可以使用 useradd 命令建立新使用者，如下所示：

```
$ sudo useradd -m -d /external_drive/bitcoin -s /dev/null bitcoin
```

m 和 d 標誌建立使用者的主目錄，在本例中指定為 */external_drive/bitcoin*。s 標誌分配使用者的互動式 shell。在本例中，我們將其設定為 */dev/null* 以停用互動式 shell 使用。最後一個參數是新使用者的使用者名 bitcoin。

5.4.3. 節點啟動

對於比特幣和閃電網路節點服務，「安裝」還包括建立所謂的_啟動腳本_，以確保節點在電腦啟動時啟動。背景服務的啟動和關閉由作業系統程序處理，在 Linux 中稱為 init 或 systemd。你通常可以在每個專案的 contrib 子目錄中找到系統啟動腳本。例如，如果你在使用 systemd 的現代 Linux 作業系統上，你會找到一個名為 *bitcoind.service* 的腳本，它可以啟動和停止 Bitcoin Core 節點服務。

以下是比特幣節點啟動腳本的範例，取自 Bitcoin Core 程式碼儲存庫：

來自 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
```

作為 root 使用者，透過將腳本複製到 systemd 服務資料夾 */lib/systemd/system/* 來安裝腳本，然後重新載入 systemd：

```
$ sudo systemctl daemon-reload
```

接下來，啟用服務：

```
$ sudo systemctl enable bitcoind
```

你現在可以啟動和停止服務。先不要啟動它，因為我們還沒有配置比特幣節點。

```
$ sudo systemctl start bitcoind
$ sudo systemctl stop bitcoind
```

5.4.4. 節點配置

要配置你的節點，你需要建立並引用一個配置檔案。按照慣例，這個檔案通常在 */etc* 中建立，在一個以程式名稱命名的目錄下。例如，Bitcoin Core 和 LND 配置通常分別儲存在 */etc/bitcoin/bitcoin.conf* 和 */etc/lnd/lnd.conf* 中。

這些配置檔案是文字檔案，每行表達一個配置選項及其值。配置檔案中未定義的任何內容都假定為預設值。你可以透過兩種方式查看配置中可以設定的選項。首先，使用 help 參數運行節點應用程式將顯示可以在命令列上定義的選項。這些相同的選項可以在配置檔案中定義。其次，你通常可以在軟體的程式碼儲存庫中找到一個範例配置檔案，其中包含所有預設選項。

你可以在我們在 閃電網路節點軟體 中使用的每個 Docker 映像中找到一個配置檔案範例。例如，檔案 *code/docker/bitcoind/bitcoind/bitcoin.conf*：

docker bitcoind 的配置檔案 (code/docker/bitcoind/bitcoind/bitcoin.conf)

```
Unresolved directive in 05_node_operations.adoc -
include::code/docker/bitcoind/bitcoind/bitcoin.conf[]
```

該特定配置檔案將 Bitcoin Core 配置為作為 regtest 節點運行，並提供弱使用者名和密碼用於遠端存取，因此你不應該將其用於你的節點配置。然而，它用於說明配置檔案的語法，你可以在 Docker 容器中對其進行調整以試驗不同的選項。看看你是否可以使用 bitcoind -help 命令來理解每個選項在我們在 閃電網路節點軟體 中建構的 Docker 網路的環境下做什麼。

通常，預設值就足夠了，只需進行少量修改即可快速配置你的節點軟體。要使 Bitcoin Core 節點以最少的自訂運行，你只需要四行配置：

```
server=1
daemon=1
txindex=1
rpcuser=USERNAME
rpcpassword=PASSWORD
```

甚至 txindex 選項也不是嚴格必需的，儘管它將確保你的比特幣節點建立所有交易的索引，這是某些應用程式所必需的。運行閃電網路節點不需要 txindex 選項。

在同一伺服器上運行的 `c-lightning` 閃電網路節點也只需要在配置中添加幾行：

```
network=mainnet
bitcoin-rpcuser=USERNAME
bitcoin-rpcpassword=PASSWORD
```

一般來說，最好盡量減少對這些系統的自訂。預設配置經過精心設計，以支援最常見的部署。如果你修改預設值，以後可能會導致問題或降低節點的效能。簡而言之，只在必要時修改！

5.4.5. 網路配置

在配置新應用程式時，網路配置通常不是問題。然而，像比特幣和閃電網路這樣的點對點網路對網路配置提出了一些獨特的挑戰。

在集中式服務中，你的電腦連接到某些公司的「大型伺服器」，而不是相反。你的家庭網路連線實際上是假設你只是他人提供的服務的消費者來配置的。但在點對點系統中，每個對等節點既消費也提供服務給其他節點。如果你在家裡運行比特幣或閃電網路節點，你就是在為網路上的其他電腦提供服務。你的網路服務預設未配置為允許你運行伺服器，可能需要一些額外配置才能讓其他人連接到你的節點。

如果你想運行比特幣或閃電網路節點，你需要讓網路上的其他節點能夠連接到你。這意味著需要啟用到比特幣埠（預設埠 8333）或閃電網路埠（預設埠 9735）的傳入 TCP 連線。雖然你可以在沒有傳入連線的情況下運行比特幣節點，但閃電網路節點不能這樣做。閃電網路節點必須從你的網路外部可被其他人存取。

預設情況下，你的家庭網路路由器不期望來自外部的傳入連線，事實上傳入連線是被阻擋的。你的網路路由器 IP 位址是唯一外部可存取的 IP 位址，你家庭網路內運行的所有電腦都共享該單一 IP 位址。這是透過一種稱為_網路位址轉換_（*NAT*）的機制實現的，它允許你的網路路由器充當所有外出連線的中介。如果你想允許傳入連線，你必須設定_埠轉發_，它告訴你的網路路由器特定埠上的傳入連線應該轉發到網路內的特定電腦。你可以透過變更網路路由器配置手動完成此操作，或者如果你的路由器支援，透過稱為_通用隨插即用_（*UPnP*）的自動埠轉發機制完成。

另一種替代埠轉發的機制是啟用洋蔥路由器（Tor），它提供一種虛擬私人網路覆蓋，允許傳入連線到_洋蔥位址_。如果你運行 Tor，你不需要進行埠轉發或啟用到比特幣或閃電網路埠的傳入連線。如果你使用 Tor 運行節點，所有流量都經過 Tor，不使用其他埠。

讓我們看看可以讓其他人連接到你的節點的不同方式。我們將按從最簡單到最困難的順序來看這些替代方案。

開箱即用！

有可能你的網路服務提供商或路由器預設配置為支援 UPnP，一切都會自動運作。讓我們先嘗試這種方法，以防我們幸運。

假設你已經有比特幣或閃電網路節點正在運行，我們將嘗試看看它們是否可以從外部存取。

要使此測試有效，你必須在家庭網路上運行比特幣或閃電網路節點（或兩者都有）。如果你的路由器支援 UPnP，傳入流量將自動轉發到運行節點的電腦上的相應埠。

你可以使用一些非常流行和有用的網站來找出你的外部 IP 位址，以及它是否允許並轉發到已知埠的傳入連線。這裡有兩個可靠的網站：

- https://canyouseeme.org
- https://www.whatismyip.com/port-scanner

預設情況下，這些服務只允許你檢查從你連接的 IP 位址的傳入連線。這樣做是為了防止你使用該服務掃描其他人的網路和電腦。你將看到你的路由器的外部 IP 位址和一個輸入埠號的欄位。如果你沒有在節點配置中變更預設埠，請嘗試埠 8333（比特幣）和/或 9735（閃電網路）。

在 檢查傳入埠 9735 中，你可以使用 *whatismyip.com* 埠掃描工具看到在運行閃電網路的伺服器上檢查埠 9735 的結果。它顯示伺服器正在接受到閃電網路埠的傳入連線。如果你看到這樣的結果，你就設定好了！

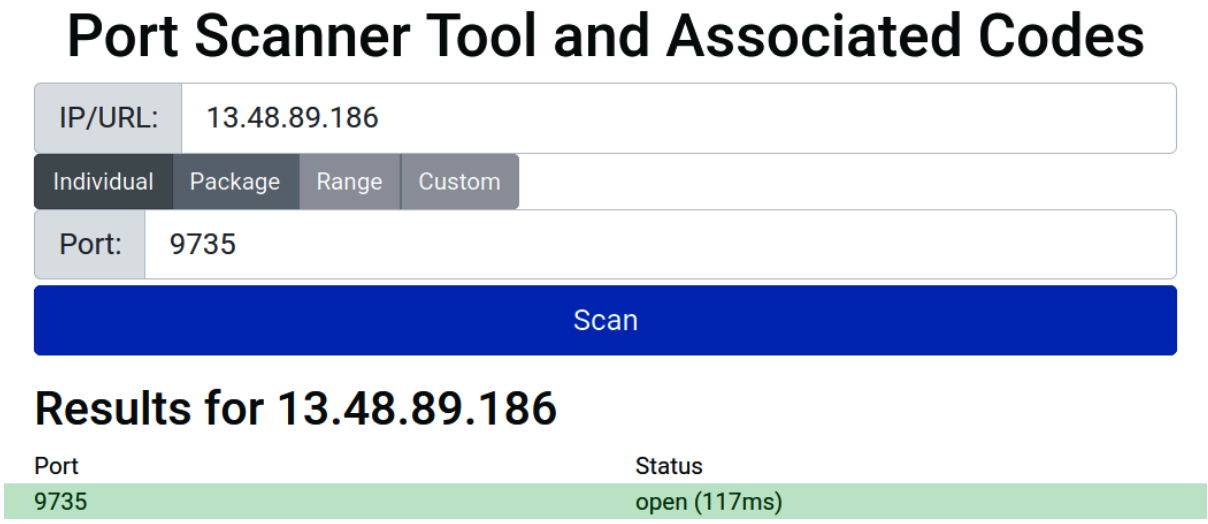

Figure 19. 檢查傳入埠 9735

使用 UPnP 自動埠轉發

有時候，即使你的網路路由器支援 UPnP，它可能預設是關閉的。在這種情況下，你需要從其網頁管理介面變更網路路由器配置：

1. 連接到你的網路路由器的配置網站。通常可以透過使用網頁瀏覽器連接到家庭網路的_閘道位址_來完成。你可以透過查看家庭網路中任何電腦的 IP 配置來找到閘道位址。它通常是非路由網路中的第一個位址，如 192.168.0.1 或 10.0.0.1。也請檢查路由器上的所有貼紙以找到_閘道位址_。找到後，打開瀏覽器並在瀏覽器的網址/搜尋框中輸入 IP 位址，例如「192.168.0.1」或「http://192.168.0.1」。

2. 找到路由器網頁配置面板的管理員使用者名和密碼。這通常寫在路由器本身的貼紙上，可能簡單如「admin」和「password」。快速搜尋你的 ISP 和路由器型號也可以幫助你找到這些資訊。
3. 找到 UPnP 設定並打開它。

重新啟動你的比特幣和/或閃電網路節點，並使用我們在上一節使用的網站之一重複開放埠測試。

使用 Tor 進行傳入連線

洋蔥路由器（*Tor*）是一種 VPN，具有特殊屬性，它會加密躍點之間的通訊，使任何中間節點都無法確定封包的來源或目的地。比特幣和閃電網路節點都支援透過 Tor 運行，這使你能夠在不透露 IP 位址或位置的情況下操作節點。因此，它為你的網路流量提供了高度的隱私保護。運行 Tor 的額外好處是，因為它作為 VPN 運行，它解決了從網路路由器進行埠轉發的問題。傳入連線透過 Tor 隧道接收，你的節點可以透過臨時生成的_洋蔥位址_而不是 IP 位址被找到。

啟用 Tor 需要兩個步驟。首先，你必須在電腦上安裝 Tor 路由器和代理。其次，你必須在比特幣或閃電網路配置中啟用 Tor 代理的使用。

要在使用 apt 套件管理器的 Ubuntu Linux 系統上安裝 Tor，請執行：

```
sudo apt install tor
```

接下來，我們配置閃電網路節點使用 Tor 進行其外部連線。這是 LND 的範例配置：

```
[Tor]
tor.active=true
tor.v3=true
tor.streamisolation=true
listen=localhost
```

這將啟用 Tor（tor.active）、建立 v3 洋蔥服務（tor.v3=true）、為每個連線使用不同的洋蔥串流（tor.streamisolation），並將監聽連線限制為僅本機，以避免洩露你的 IP 位址（`listen=localhost`）。

你可以透過運行一個簡單的單行命令來檢查 Tor 是否正確安裝並運行。此命令應該在大多數 Linux 版本上有效：

```
curl --socks5 localhost:9050 --socks5-hostname localhost:9050 -s
https://check.torproject.org/ | cat | grep -m 1 Congratulations | xargs
```

如果一切正常運作，此命令的回應應該是 "Congratulations. This browser is configured to use Tor."

由於 Tor 的性質，你無法輕鬆使用外部服務來檢查你的節點是否可以透過洋蔥位址存取。儘管如此，你應該在閃電網路節點的日誌中看到你的 Tor 洋蔥位址。它是一長串字母和數字，後面跟著 .onion 後綴。你的節點現在應該可以從網路存取，還有額外的隱私保護！

手動埠轉發

這是最複雜的過程，需要相當多的技術技能。具體細節取決於你擁有的網路路由器類型、你的服務提供商設定和政策，以及許多其他上下文。在嘗試這個更困難的機制之前，請先嘗試 UPnP 或 Tor。

基本步驟如下：

1. 找到你的節點所在電腦的 IP 位址。這通常由動態主機配置協定（DHCP）動態分配，通常在 192.168.x.x 或 10.x.x.x 範圍內的某處。
2. 找到你的節點網路介面的媒體存取控制（MAC）位址。這可以在該電腦的網路設定中找到。
3. 為你的節點分配一個靜態 IP 位址，這樣它總是相同的。你可以使用它目前擁有的 IP 位址。在你的網路路由器上，在 DHCP 配置下查找「靜態租約」。將 MAC 位址映射到你選擇的 IP 位址。現在你的節點將始終分配到該 IP 位址。或者，你可以查看路由器的 DHCP 配置並找出其 DHCP 位址範圍是什麼。選擇 DHCP 位址範圍_之外_的未使用位址。然後，在伺服器上，配置網路停止使用 DHCP，並將選定的非 DHCP IP 位址硬編碼到作業系統網路配置中。
4. 最後，在你的網路路由器上設定「埠轉發」，將特定埠上的傳入流量路由到你的伺服器的選定 IP 位址。

完成重新配置後，使用前面章節中的其中一個網站重複埠檢查。

5.5. 節點安全

按定義，閃電網路節點是一個_熱錢包_。這意味著由閃電網路節點控制的資金（包括鏈上和鏈下）直接由載入節點記憶體或儲存在節點硬碟上的金鑰控制。如果閃電網路節點被入侵，創建鏈上或鏈下交易來轉移其資金是輕而易舉的事。因此，保護它免受未經授權的存取至關重要。

安全是一項整體工作，意味著你必須保護系統的每一層。俗話說：鏈條的強度取決於最薄弱的環節。這是資訊安全的重要概念，我們將把它應用到我們的節點上。

儘管你將採取所有安全措施，請記住閃電網路是一項早期的實驗性技術，你使用的任何專案的程式碼中都可能存在可利用的漏洞。*不要在閃電網路上放置超過你願意承擔損失風險的資金。*

5.5.1. 作業系統安全

保護作業系統是一個超出本書範圍的廣泛話題。然而，我們可以建立一些基本原則。

為了保護你的作業系統，以下是一些需要考慮的首要事項：

來源可靠性

首先確保你下載的是正確的作業系統映像，並在安裝前驗證任何簽名或校驗和。將此延伸到你安裝的任何軟體。仔細檢查你下載的任何來源或網址。透過簽名和校驗和驗證來驗證下載軟體的完整性和正確性。

維護

確保你的作業系統保持最新。啟用自動每日或每週安裝安全更新。

最小權限

為特定程序設定使用者並授予他們運行服務所需的最少存取權限。不要以管理員權限（例如 root）運行程序。

程序隔離

使用作業系統功能將程序相互隔離。

檔案系統權限

根據最小權限原則仔細配置檔案系統。不要讓檔案可被所有人讀取或寫入。

強認證

使用強大的隨機生成密碼，或者在可能的情況下使用公鑰認證。例如，使用帶有加密金鑰對的安全外殼（SSH）比使用密碼更安全。

雙因素認證（2FA）

盡可能使用雙因素認證，包括使用硬體安全金鑰的通用第二因素（U2F）。這適用於你可能使用的所有外部服務，例如你的雲端服務提供商。你也可以將此應用於你自己的設定，例如你自己的 SSH 配置。對間接服務也使用 2FA。例如，假設你使用雲端服務。你給了雲端服務提供商一個電子郵件地址，所以你也應該用 2FA 保護你的電子郵件地址。

備份

對你的系統進行備份，並確保你也用加密保護備份。定期執行這些備份。至少測試一次，看看你是否可以恢復備份，以及你的備份是否完整和可存取。如果可能的話，將備份的一個副本保存在不同的磁碟上，以避免單一硬碟故障同時破壞你的活動節點_和_備份副本。

漏洞和暴露管理

使用遠端掃描來確保你已將系統的攻擊面最小化。關閉任何不必要的服務或埠。只安裝你真正需要和使用的軟體和套件。解除安裝你不再使用的套件。建議你_不要_將節點電腦用於可以在其他電腦上執行的非節點活動。特別是，如果可以的話，_不要_使用節點電腦進行瀏覽、上網或閱讀電子郵件。

這是最基本安全措施的列表。它絕不是詳盡無遺的。

5.5.2. 節點存取

你的閃電網路節點將公開一個遠端程序呼叫（RPC）API。這意味著你的節點可以透過發送到特定 TCP 埠的命令遠端控制。對該 RPC API 的存取控制是透過某種形式的使用者認證來實現的。根據你設定的閃電網路節點類型，這將透過 使用者名/密碼 認證或稱為認證 *macaroon* 的機制來完成。顧名思義，macaroon 是一種更複雜的 cookie 類型。與 cookie 不同，它經過加密簽名，可以表達一組存取 功能。

例如，LND 使用 macaroons 來授予對 RPC API 的存取權限。預設情況下，LND 軟體建立三個具有不同存取級別的 macaroons，稱為 admin、invoice 和 readonly。根據你複製並在 RPC 客戶端中使用的 macaroon，你可以擁有_唯讀_存取權、_發票_存取權（包括唯讀功能）或_管理員_存取權，這給你完全控制權。LND 中還有一個 macaroon bakery 功能，可以用非常精細的控制構建具有任意功能組合的 macaroons。

如果你使用使用者名/密碼認證模型，請確保選擇一個長而隨機的密碼。你不需要經常輸入這個密碼，因為它將儲存在配置檔案中。因此，你應該選擇一個無法猜測的密碼。你看到的許多範例包含選擇不當的密碼，人們經常將這些複製到他們自己的系統中，為任何人提供輕鬆的存取。不要這樣做！使用密碼管理器生成一個長的隨機字母數字密碼。由於某些特殊字元如 $?/!*\&%`"' 可能會干擾命令列，因此最好避免在 shell 環境中使用的密碼中使用這些字元。為避免問題，請堅持使用長的隨機字母數字密碼。

超過 12 個字元且隨機生成的純字母數字序列通常就足夠了。如果你計劃在閃電網路節點上存放大量資金，並且擔心遠端暴力攻擊，請選擇超過 20 個字元的密碼長度，使此類攻擊實際上不可行。

5.6. 節點和通道備份

運行閃電網路節點時一個非常重要的考慮因素是備份問題。與比特幣錢包不同，在比特幣中 BIP-39 助記詞可以恢復錢包的所有狀態，在閃電網路中情況_不是_這樣。

閃電網路錢包確實使用 BIP-39 助記詞備份，但只用於鏈上錢包。然而，由於通道的構建方式，助記詞_不足以_恢復閃電網路節點。需要額外的備份層，稱為_靜態通道備份_（*SCB*）。沒有 SCB，如果閃電網路節點操作員失去閃電網路節點資料儲存，他們可能會失去通道中的_所有_資金。

> 在你建立持續備份通道狀態的系統之前，_不要_為通道注入資金。你的備份應該移動到「異地」，即與節點不同的系統和位置，以便它們可以在各種系統故障（電源中斷、資料損壞等）或自然災害（洪水、火災等）中存活。

SCB 不是萬靈丹。首先，每個通道的狀態需要在每次有新的承諾交易時備份。其次，從通道備份恢復是危險的。如果你沒有_最新_的承諾交易，並且意外廣播了舊的（已撤銷的）承諾，你的通道對等方將假設你試圖作弊，並用懲罰交易索取整個通道餘額。為了確保你正在關閉通道，你需要進行_協作關閉_。但惡意對等方可能會誤導你的節點在協作關閉期間廣播舊的、已撤銷的承諾，從而透過讓你的節點無意中嘗試作弊來欺騙你。

此外，你的通道備份需要加密以維護你的隱私和通道安全。否則，任何找到備份的人不僅可以看到你所有的通道，還可以使用備份以將餘額交給你的通道對等方的方式關閉你所有的通道。換句話說，獲得你備份存取權的惡意人士可能導致你失去所有通道資金。

你可以看到 SCB 不是萬無一失的保障。它們是一個薄弱的妥協，因為它們用一種類型的風險（資料損壞或遺失）交換另一種類型的風險（惡意對等方）。要從 SCB 恢復，你必須與你的通道對等方互動，並希望他們不會透過給你舊的承諾或欺騙你的節點廣播已撤銷的承諾讓他們可以懲罰你來試圖欺騙你。儘管 SCB 有弱點，SCB 確實有意義，你應該執行它們。如果你不執行 SCB 並且失去節點資料，你將永遠失去你的通道資金。保證！然而，如果你_確實_執行 SCB 並且失去節點資料，那麼你有合理的機會有些對等方是誠實的，你可以恢復一些通道資金。如果你幸運的話，你可能可以恢復所有資金。總之，最好對主節點硬碟以外的磁碟執行持續的 SCB。

通道備份機制仍在開發中，是大多數閃電網路實作的弱點。

在撰寫本書時，只有 LND 提供了 SCB 的內建機制。Eclair 對伺服器端部署有類似的機制，儘管 Eclair Mobile 確實提供可選的 Google Drive 備份。`c-lightning` 最近合併了外掛程式實作通道備份所需的介面。不幸的是，不同節點 實作 之間沒有一致、公認的備份機制。

閃電網路節點資料庫的基於檔案的備份充其量是部分解決方案，因為你有備份不一致資料庫狀態的風險。此外，你可能無法可靠地捕獲最新的狀態承諾。最好有一個在每次通道狀態變更時觸發的備份機制，從而確保資料一致性。

要在 LND 中設定 SCB，請在命令列或配置檔案中設定 backupfilepath 參數。然後 LND 將在該目錄路徑中儲存 SCB 檔案。當然，這只是解決方案的第一步。現在，你必須設定一個監控此檔案變更的機制。每次檔案變更時，備份 機制必須將此檔案複製到另一個，最好是異地的磁碟。這種備份機制超出了本書的範圍。儘管如此，任何複雜的備份解決方案都應該能夠處理這種情況。請記住，備份檔案也應該加密。

5.6.1. 熱錢包風險

如 我們之前討論過的，閃電網路由一個_熱錢包_網路組成。你儲存在閃電網路錢包中的資金始終在線上。這使它們容易受到攻擊。因此，你不應該在閃電網路錢包中存放大量資金。大量資金應該保存在_不_在線上並且只能在鏈上交易的_冷_錢包中。

即使你一開始存放少量資金，隨著時間的推移，你可能仍然會發現閃電網路錢包中有大量資金。這是商店老闆的典型場景。如果你將閃電網路節點用於電子商務操作，你的錢包可能會經常收到資金，但很少發送資金。因此，你最終會同時遇到兩個問題。首先，你的通道會不平衡，本地餘額大於遠端餘額。其次，你的錢包中會有太多資金。幸運的是，你也可以同時解決這兩個問題。

讓我們看看一些可以用來減少熱錢包中暴露資金的解決方案。

5.6.2. 清掃資金

如果你的閃電網路錢包餘額對你的風險承受能力來說變得太大，你將需要從錢包中「清掃」資金。你可以透過三種方式做到這一點：鏈上、鏈下和 Loop Out。讓我們在接下來的幾節中看看這些選項。

鏈上清掃

鏈上清掃資金是透過將資金從閃電網路錢包轉移到比特幣錢包來完成的。你透過關閉通道來做到這一點。當你關閉通道時，你本地餘額中的所有資金都被「清掃」到一個比特幣地址。用於鏈上資金的比特幣地址通常由你的閃電網路錢包生成，所以它仍然是一個熱錢包。你可能需要進行額外的鏈上交易，將資金轉移到更安全的地址，例如在你的硬體錢包上生成的地址。

關閉通道會產生鏈上費用，並會降低你的閃電網路節點的容量和連接性。然而，如果你運行一個受歡迎的電子商務節點，你不會缺少入站容量，可以策略性地關閉本地餘額較大的通道，基本上「捆綁」你的資金以進行鏈上轉移。你可能需要在關閉通道之前使用一些通道再平衡技術（參見 重新平衡通道）來最大化此策略的效益。

鏈下清掃

你可以使用的另一種技術涉及運行第二個未在網路上公布的閃電網路節點。你可以從你的公共節點（例如運行你商店的那個）到你未公布的（隱藏）節點建立大容量通道。定期透過向你的隱藏節點進行閃電網路付款來「清掃」資金。

這種技術的優勢在於接收你商店付款的閃電網路節點將是公開知名的。這使它成為駭客的目標，因為任何與商店相關聯的閃電網路節點都會被假設有大量餘額。一個不與你的商店相關聯的第二個節點不會輕易被識別為有價值的目標。

作為額外的安全措施，你可以使你的第二個節點成為隱藏的 Tor 服務，這樣它的 IP 位址就不會被知道。這進一步減少了攻擊的機會並增加了你的隱私。

你需要設定一個定期運行的腳本。此腳本的目的是在你的隱藏節點上建立發票，並從你商店的節點支付該發票，從而將資金轉移到你的隱藏節點。

請記住，這種技術不會將資金轉移到冷儲存。兩個閃電網路節點都是熱錢包。此清掃的目的是將資金從非常知名的熱錢包轉移到不起眼的熱錢包。

潛水艇交換清掃

減少閃電網路熱錢包餘額的另一種方法是使用稱為_潛水艇交換_的技術。潛水艇交換由共同作者 Olaoluwa Osuntokun 和 Alex Bosworth 提出概念，允許將鏈上比特幣與閃電網路付款進行交換，反之亦然。本質上，潛水艇交換是閃電網路鏈下資金和比特幣鏈上資金之間的原子交換。

節點操作員可以發起潛水艇交換，將所有可用的通道餘額發送給對方，對方將發送鏈上比特幣作為交換。

在未來，這可能成為閃電網路上節點提供的付費服務，這些節點會公布匯率或收取固定費用進行轉換。

潛水艇交換清掃資金的優勢是不需要關閉通道。這意味著我們保留了我們的通道，只是透過此操作重新平衡我們的通道。當我們發送閃電網路付款時，我們在一個或多個通道上將一些餘額從本地轉移到遠端。這不僅減少了節點熱錢包中暴露的餘額，還增加了未來接收付款的可用餘額。

你可以透過信任中介充當閘道來做到這一點，但這會冒著你的資金被偷的風險。然而，在潛水艇交換的情況下，操作不需要信任。潛水艇交換是非託管的_原子_操作。這意味著你的潛水艇交換對手方無法竊取你的資金，因為鏈上付款取決於鏈下付款的完成，反之亦然。

使用 Loop 進行潛水艇交換

潛水艇交換服務的一個例子是 Lightning Labs 的 *Loop*，這是開發 LND 的同一家公司。Loop 有兩種變體：Loop In 和 Loop Out。*Loop In* 接受比特幣鏈上付款並將其轉換為閃電網路鏈下付款。*Loop Out* 將閃電網路付款轉換為比特幣付款。

要使用 Loop 服務，你必須運行 LND 閃電網路節點。

為了減少閃電網路熱錢包的餘額，你將使用 Loop Out 服務。要使用 Loop 服務，你需要在節點上安裝一些額外的軟體。Loop 軟體與你的 LND 節點一起運行，並提供一些命令列工具來執行潛水艇交換。你可以在 GitHub (https://github.com/lightninglabs/loop) 上找到 Loop 軟體和安裝說明。

安裝並運行軟體後，Loop Out 操作就像運行單個命令一樣簡單：

```
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.
```

請注意，你的最高費用（代表最壞情況）將取決於你選擇的確認目標。

5.7. 閃電網路節點正常運行時間和可用性

與比特幣不同，閃電網路節點需要幾乎持續在線。你的節點需要在線才能接收付款、開啟通道、關閉通道（協作方式）以及監控協定違規。節點可用性是閃電網路中如此重要的要求，以至於它被各種自動通道管理工具（例如 autopilot）用作決定與哪些節點開啟通道的指標。你也可以在流行的節點瀏覽器（參見 閃電網路瀏覽器）上看到「可用性」作為節點指標，例如 1ML (https://1ml.com)。

節點可用性對於減輕和解決潛在的協定違規（即已撤銷的承諾）尤其重要。雖然你可以承受從一小時到一兩天的短暫中斷，但你不能讓節點長時間離線而不冒資金損失的風險。

持續保持節點在線並不容易，因為各種錯誤和資源限制會不時導致停機。特別是如果你運行一個繁忙且受歡迎的節點，你會遇到記憶體、交換空間、開啟檔案數量、磁碟空間等方面的限制。各種不同的問題會導致你的節點或伺服器當機。

5.7.1. 容錯和自動化

如果你有時間和技能，你應該在閃電網路測試網上測試一些基本的故障場景。在測試網上，你將學習到寶貴的經驗，而不會冒任何資金風險。你採取的任何自動化系統的步驟都將提高你的可用性：

自動電腦伺服器重新啟動

當你的伺服器或作業系統當機時會發生什麼？電源中斷時會發生什麼？透過按下電腦上的「重設」按鈕或拔掉電源線來模擬此故障。在當機、重設或電源故障後，電腦應自動重新啟動。一些電腦在其 BIOS 中有一個設定來指定電腦在電源故障時應如何反應。測試它以確保電腦在電源故障時確實會自動重新啟動，而無需人工干預。

自動節點重新啟動

當你的節點或其中一個節點當機時會發生什麼？透過終止相應的節點程序來模擬此故障。如果節點當機，它應該自動重新啟動。測試它以確保節點在故障時確實會自動重新啟動，而無需人工干預。如果情況並非如此，很可能你的節點沒有正確設定為作業系統服務。

自動網路重新連線

你的網路斷線時會發生什麼？當你的 ISP 暫時中斷時會發生什麼？當你的 ISP 為你的路由器或電腦分配新的 IP 位址時會發生什麼？當網路恢復時，你運行的節點是否會自動重新連接到網路？透過拔掉然後重新插入托管你節點的設備的網路線來模擬此故障。節點應自動重新連接並繼續運行，而無需人工干預。

配置你的日誌檔案

所有上述故障都應在相應的日誌檔案中留下文字條目。如果需要，請調高日誌詳細程度。在日誌檔案中找到這些錯誤條目，並使用它們進行監控。

5.7.2. 監控節點可用性

監控你的節點是保持其運行的重要部分。你不僅需要監控電腦本身的可用性，還需要監控閃電網路節點軟體的可用性和正確運行。

有多種方法可以做到這一點，但大多數需要一些自訂。你可以使用通用的基礎設施監控或應用程式監控工具，但你必須專門自訂它們來查詢閃電網路節點 API，以確保節點正在運行、已與區塊鏈同步並連接到通道對等方。

Lightning.watch (https://lightning.watch) 提供一項專門的服務，提供閃電網路節點監控。它使用 Telegram 機器人來通知你任何服務中斷。這是一項免費服務，但你可以付費（當然是透過閃電網路）以獲得更快的警報。

隨著時間的推移，我們期望更多第三方服務提供專門的閃電網路節點監控，並透過微支付付費。也許這些服務及其 API 將標準化，並有一天被閃電網路節點軟體直接支援。

5.7.3. 瞭望塔

_瞭望塔_是一種將閃電網路協定違規的監控和懲罰解決外包的機制。

正如我們在前幾章中提到的，閃電網路協定透過懲罰機制來維護安全性。如果你的通道合作夥伴之一廣播舊的承諾交易，你的節點將需要行使撤銷條款並廣播懲罰交易以避免資金損失。但如果你的節點在協定違規期間處於離線狀態，你可能會損失資金。

為了解決這個問題，我們可以使用一個或多個瞭望塔來外包監控協定違規和發出懲罰交易的工作。瞭望塔設定有兩個部分：瞭望塔伺服器（或簡稱瞭望塔）監控區塊鏈，瞭望塔客戶端請求瞭望塔伺服器提供此監控服務。

瞭望塔技術仍處於早期開發階段，並未得到廣泛支援。然而，在以下段落中，我們列出了一些你可以嘗試的實驗性實作。

LND 軟體包含瞭望塔伺服器和瞭望塔客戶端。你可以透過添加以下配置選項來啟動瞭望塔伺服器：

```
[watchtower]
watchtower.active=1
watchtower.towerdir=/path_to_watchtower_data_directory
```

你可以透過在配置中啟動 LND 的瞭望塔客戶端，然後使用命令列將其連接到瞭望塔伺服器來使用它。配置如下：

```
[wtclient]
wtclient.active=1
```

LND 的命令列客戶端 lncli 顯示以下用於管理瞭望塔客戶端的選項：

```
$ 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
```

`c-lightning` 具有瞭望塔客戶端外掛程式所需的 API 掛鉤，儘管目前尚未實作此類外掛程式。

最後，一個流行的獨立瞭望塔伺服器是 *The Eye of Satoshi*（TEOS）。它可以在 GitHub (https://github.com/talaia-labs/python-teos) 上找到。

5.8. 通道管理

作為閃電網路節點操作員，你需要執行的經常性任務之一是管理你的通道。這意味著從你的節點向其他節點開啟出站通道，以及讓其他節點向你的節點開啟入站通道。在未來，可能會有協作通道建構，這樣你就可以開啟在建立時兩端都有資金承諾的對稱通道。然而，目前新通道只有一端有資金，在發起者那一側。因此，為了使你的節點_平衡_，同時具有入站和出站容量，你需要向其他人開啟通道，並吸引其他人向你的節點開啟通道。

5.8.1. 開啟出站通道

一旦你的閃電網路節點啟動並運行，你就可以為其比特幣錢包注資，然後開始用這些資金開啟通道。

你必須謹慎選擇通道合作夥伴，因為你的節點發送付款的能力取決於你的通道合作夥伴是誰，以及他們與閃電網路其餘部分的連接程度。你還希望擁有多個通道，以避免單點故障。由於閃電網路現在支援多部分付款，你可以將初始資金分成幾個通道，並透過組合它們的容量來路由更大的付款。同時，避免讓你的通道太小。由於你需要支付比特幣交易費用來開啟和關閉通道，通道餘額不應該小到鏈上費用消耗其中很大一部分。一切都是關於平衡！

總結：

- 連接到幾個連接良好的節點

- 開啟多個通道
- 不要開啟太多通道
- 不要讓通道太小

找到連接良好的節點的一種方法是向在閃電網路上銷售產品的熱門商家開啟通道。這些節點往往資金充足且連接良好。因此，當你準備好透過閃電網路在線上購買東西時，你可以直接向商家的節點開啟通道。商家的節點 ID 將在你嘗試購買東西時收到的發票中。這使得它很容易。

找到連接良好的節點的另一種方法是使用閃電網路瀏覽器（參見 閃電網路瀏覽器），如 1ML (https://1ml.com)，並瀏覽按通道容量和通道數量排序的節點列表。不要選擇最大的節點，因為那會鼓勵中心化。選擇列表中間的節點，這樣你可以幫助它們成長。另一個需要考慮的因素可能是節點運營的時間跨度。運營超過一年的節點可能比一週前開始運營的節點更有信譽且風險更低。

自動導航

開啟通道的任務可以透過使用_自動導航_來部分自動化，這是一種根據某些啟發式規則自動開啟通道的軟體。自動導航軟體仍然相對較新，它並不總是為你選擇最佳的通道合作夥伴。特別是在開始時，手動開啟通道可能更好。 自動導航目前以三種形式存在：

- lnd 包含一個與 lnd 完全整合的自動導航，並在開啟時在背景持續運行。
- lib_autopilot.py 可以根據八卦和通道資料為任何節點實作提供自動導航計算。
- 基於 lib_autopilot.py 的 c-lightning 外掛程式存在，為 c-lightning 使用者提供易於使用的介面。

請注意，一旦透過配置檔案開啟，lnd 自動導航將開始在背景運行。因此，如果你的 lnd 錢包中有鏈上輸出，它將立即開始開啟通道。 如果你想完全控制你進行的比特幣交易和你開啟的通道，請確保在用比特幣資金載入你的 lnd 錢包_之前_關閉自動導航。 如果自動導航之前已經開啟，你可能需要在透過鏈上交易為錢包充值或關閉通道（這實際上會再次給你鏈上資金）之前重新啟動你的 lnd。 如果你想運行自動導航，設定關鍵配置值至關重要。 看看這個範例配置：

```
[lnd-autopilot]
autopilot.active=1
autopilot.maxchannels=40
autopilot.allocation=0.70
autopilot.minchansize=500000
autopilot.maxchansize=5000000
autopilot.heuristic=top_centrality:1.0
```

此配置檔案將啟動自動導航。 只要滿足以下兩個條件，它就會開啟通道：

1. 你的節點目前開啟的通道少於 40 個。
2. 你的總資金中少於 70% 在支付通道中處於鏈下狀態。

這裡選擇的數字 40 和 0.7 完全是任意的，因為我們無法對每個人都有效地建議你應該開啟多少通道以及你的資金應該有多少百分比在鏈下。 lnd 中的自動導航不會考慮鏈上費用。換句話說，它不會延遲開啟通道到費用較低的時間段。 為了降低費用，你可以在費用較低的時間段（例如週末）手動開啟通道。 每當條件滿足時，自動導航就會提出通道建議，並立即嘗試使用適當的當前費用開啟通道。 根據前面的配置檔案，通道大小將在 5 mBTC（`minchansize` = 500,000 satoshi）和 50 mBTC（`maxchansize` = 5,000,000 satoshi）之間。 按照慣例，配置檔案中的金額以 satoshi 為單位。 目前，低於 1 mBTC 的通道不是很有用，我們不建議你開啟低於此金額的過小通道。 隨著多部分付款的更廣泛採用，較小的通道負擔會減少。但目前，這是我們的建議。

由 René Pickhardt（本書的共同作者之一）最初撰寫的 c-lightning 外掛程式與 lnd 自動導航相比工作方式非常不同。 首先，它在用於提出建議的演算法上有所不同。我們這裡不會涵蓋這些。其次，它在使用者介面上有所不同。 你需要從 c-lightning 外掛程式 儲存庫 (https://github.com/lightningd/plugins/tree/master/autopilot) 下載 *autopilot plug-in* 並啟動它。

要在 c-lightning 中啟動外掛程式，請將其放入 *~/.lightning/plugins* 目錄，確保它是可執行的（例如 `chmod x ~/.lightning/plugins/autopilot.py`），然後重新啟動 +lightningd。

或者，如果你不希望外掛程式在啟動 lightningd 時自動啟動，你可以將其放在不同的目錄中，並使用 lightningd 的 plugin 參數手動啟動它：

```
lightningd --plugin=~/lightning-plugins/autopilot.py
```

c-lightning 中的自動導航透過三個配置值控制，這些值可以在配置檔案中設定，或者在啟動 lightningd 時作為命令列參數：

```
[c-lightning-autopilot]
autopilot-percent=75
autopilot-num-channels=10
autopilot-min-channel-size-msat=100000000msat
```

這些值是實際的預設配置，你根本不需要設定它們。

自動導航不會像 lnd 那樣在背景自動運行。 相反，如果你希望自動導航開啟建議的通道，你必須使用 `lightning-cli autopilot-run-once` 專門啟動一次運行。 但如果你只想讓它為你提供建議，從中你可以手動挑選節點，你可以附加可選的 `dryrun` 參數。

lnd 和 c-lightning 自動導航之間的一個關鍵區別是 c-lightning 自動導航還會為通道大小提出建議。 例如，如果自動導航建議與只有小通道的小節點開啟通道，它不會建議開啟大通道。 然而，如果它與一個連接良好且也有許多大通道的節點開啟通道，它可能會建議更大的通道大

小。

如你所見，c-lightning 自動導航不像 lnd 的那樣自動，但它給你多一點控制。 這些差異反映了個人偏好，實際上可能是你選擇一個實作而不是另一個的決定性因素。

請記住，目前的自動導航主要會使用來自八卦協定的關於閃電網路當前拓撲的公開資訊。 很明顯，你對通道的個人需求只能在一定程度上反映出來。 更進階的自動導航會使用你的節點過去運行時收集的歷史和使用資訊，包括路由成功的資訊、你過去支付給誰以及誰支付給你。 在未來，這種改進的自動導航可能還會使用這些收集的資料來提出關閉通道和重新分配資金的建議。

總體而言，在撰寫本書時，請謹慎不要過度依賴自動導航。

5.8.2. 獲取入站流動性

在閃電網路的當前設計中，使用者更典型的是_先_獲取出站流動性，_然後_獲取入站流動性。他們會透過與另一個節點開啟通道來做到這一點，通常他們會在能夠接收比特幣之前先能夠花費比特幣。 有三種典型的方式獲取入站流動性：

- 開啟一個有出站流動性的通道，然後花掉其中一些資金。現在餘額在通道的另一端，這意味著你可以接收付款。
- 請求某人向你的節點開啟通道。提供互惠，這樣你們雙方的節點都會變得更好連接和平衡。
- 使用潛水艇交換（例如 Loop In）將鏈上 BTC 兌換為通往你節點的入站通道。
- 付費給第三方服務讓他們與你開啟通道。存在幾個這樣的服務。有些收取費用來提供流動性，有些是免費的。

以下是目前可用的流動性提供者列表，它們會收費向你的節點開啟通道：

- Bitrefill 的 Thor 服務 (https://www.bitrefill.com/thor-lightning-network-channels)
- Lightning To Me (https://lightningto.me)
- LNBig (https://lnbig.com)
- Lightning Conductor (https://lightningconductor.net/channels)

從實際和使用者體驗的角度來看，創建入站流動性是具有挑戰性的。入站流動性不會自動發生，所以你必須找到方法為你的節點建立它。支付通道的這種不對稱性也不直觀。在大多數其他支付系統中，你先收到付款（入站），然後才付款給其他人（出站）。

如果你是商家或為閃電網路付款出售服務，創建入站流動性的挑戰最為明顯。在這種情況下，你需要保持警惕，確保你有足夠的入站流動性來繼續接收付款。如果你的商店有買家激增，但他們實際上因為沒有更多入站容量而無法付款給你怎麼辦？

在未來，這些挑戰可以透過實作雙向資助通道來部分緩解，這種通道從雙方資助，提供平衡的入站和出站容量。這種負擔也可以透過更複雜的自動導航軟體來減輕，它可以根據需要請求並支付入站容量。

最終，閃電網路使用者需要對通道管理有策略且主動，以確保有足夠的入站容量來滿足他們的需求。

5.8.3. 關閉通道

如本書前面所討論的，*協作關閉_是關閉通道的首選方式。然而，有些情況下需要_強制關閉*。

一些例子：

- 你的通道合作夥伴離線，無法聯繫以發起協作關閉。
- 你的通道合作夥伴在線，但不回應發起協作關閉的請求。
- 你的通道合作夥伴在線，你們的節點正在協商協作關閉，但它們卡住了，無法達成決議。

5.8.4. 重新平衡通道

在閃電網路上進行交易和路由付款的過程中，入站和出站容量的組合可能會變得不平衡。

例如，如果你的通道合作夥伴之一頻繁透過你的節點路由付款，你將耗盡該通道的入站容量，同時也耗盡出站通道的出站容量。一旦發生這種情況，你就無法再透過該路由路由付款。

有很多方法可以重新平衡通道，每種方法都有不同的優缺點。一種方法是使用潛水艇交換（例如 Loop Out），如本章前面所述。另一種重新平衡的方法是簡單地等待以相反方向流動的路由付款。如果你的節點連接良好，當特定路由在一個方向上耗盡時，相同的路由在相反方向上就變得可用。其他節點可能會「發現」相反方向的該路由並開始將其用作其付款路徑的一部分，從而再次重新平衡資金。

重新平衡通道的第三種方法是故意創建一個_循環路由_，將付款從你的節點透過閃電網路發送回你的節點。透過在具有大量本地容量的通道上發送付款，並安排路徑使其在具有大量遠端容量的通道上返回你的節點，這兩個通道都將變得更加平衡。循環路由重新平衡策略的範例可以在 循環路由重新平衡 中看到。

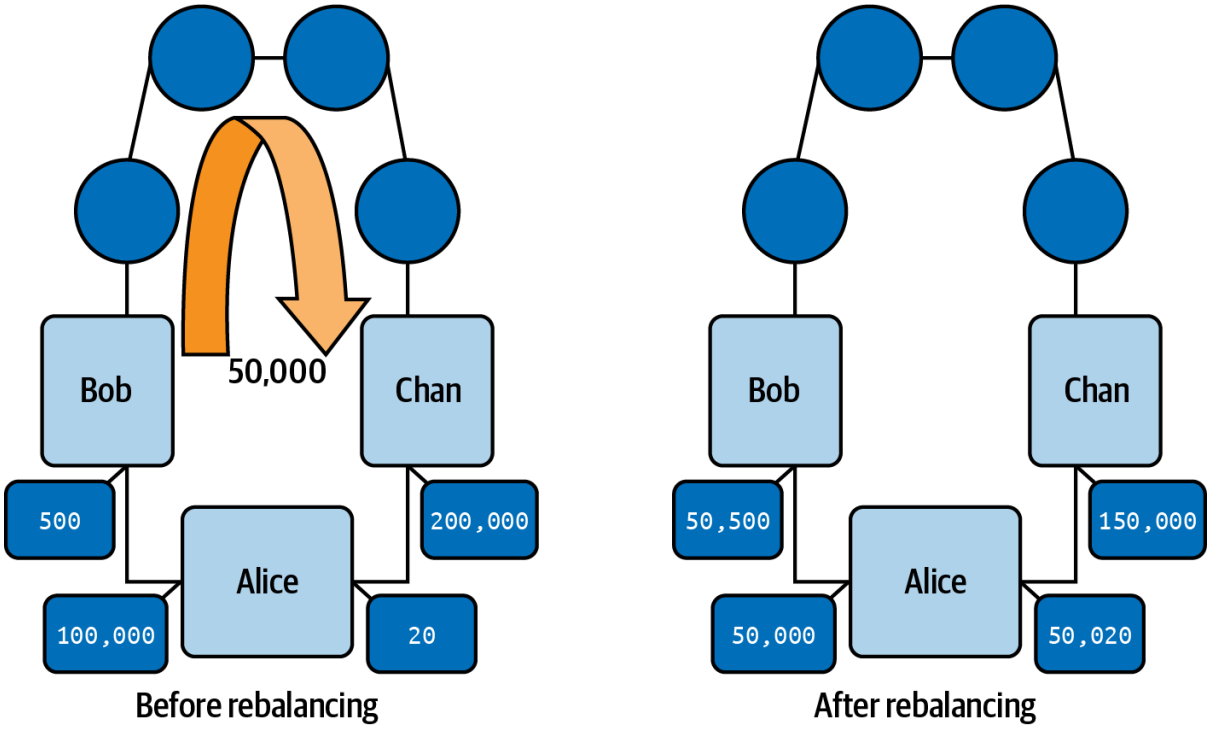

Figure 20. 循環路由重新平衡

大多數閃電網路節點實作都支援循環重新平衡，可以在命令列上或透過 Ride The Lightning（參見 Ride The Lightning）等網頁管理介面完成。

通道重新平衡是一個複雜的問題，是活躍研究的主題，在 重新平衡通道 中有更詳細的介紹。

5.9. 路由費用

運行閃電網路節點允許你透過跨通道路由付款來賺取費用。路由費用通常不是重要的收入來源，與運行節點的成本相比微不足道。例如，在一個相對繁忙的每天路由十幾筆付款的節點上，費用不超過 2,000 satoshi。

節點透過在每個通道上設定其期望的費率來競爭路由費用。路由費用由每個通道上的兩個參數設定：對任何付款收取的固定_基本費用_和與付款金額成比例的額外可變_費率_。

發送閃電網路付款時，節點將選擇一條路徑以最小化費用、最小化跳數，或兩者兼顧。因此，路由費用市場從這些互動中產生。目前有許多節點收取非常低或不收取路由費用，對路由費用市場造成下行壓力。

如果你不做任何選擇，你的閃電網路節點將為每個新通道設定預設的基本費用和費率。預設值取決於你使用的節點實作。 基本費用以_毫聰_（千分之一聰）為單位設定。比例費率以_百萬分之一_為單位設定，並應用於付款金額。 百萬分之一的單位通常縮寫為 *ppm*（百萬分之幾）。 例如，基本費用為 1,000（毫聰）和費率為 1,000 ppm（百萬分之一）將導致對 100,000 satoshi 付款收取以下費用：

$$
\begin{aligned}
P &= 100,000 \text{ satoshi} \\
F_{base} &= 1,000 \text{ millisatoshi} = 1 \text{ satoshi} \\
F_{rate} &= 1,000 \text{ ppm} = 1,000/1,000,000 = 1/1,000 = 0.001 = 0.1\% \\
F_{total} &= F_{base} + (P * F_{rate}) \\
\Rightarrow F_{total} &= 1 \text{ satoshi} + (100,000/1,000) \text{ satoshi} \\
\Rightarrow F_{total} &= 1 \text{ satoshi} + 100 \text{ satoshi} = 101 \text{ satoshi}
\end{aligned}
$$

廣義來說，你可以對路由費用採取兩種方法之一。你可以以低費用路由大量付款，透過高量彌補低費用。或者，你可以選擇收取更高的費用。如果你選擇設定更高的費用，只有當其他更便宜的路由不存在時，你的節點才會被選中。因此，你路由的頻率會降低，但每次成功路由賺取更多。

對於大多數節點，通常最好使用預設的路由費用值。這樣，你的節點就會與其他使用預設值的節點在大致相同的競技場上競爭。

你也可以使用路由費用設定來重新平衡通道。如果你的大多數通道都有預設費用，但你想重新平衡特定通道，只需將該特定通道的費用降低到零或非常低的費率。然後坐下來等待某人透過你的「便宜」路由路由付款，並作為副作用為你重新平衡通道。

5.10. 節點管理

在命令列上管理你的閃電網路節點顯然不容易。它為你提供了節點 API 的完整靈活性和編寫自己的自訂腳本來滿足你個人需求的能力。但如果你不想處理命令列的複雜性，只需要一些基本的節點管理功能，你應該考慮安裝基於網頁的使用者介面，使節點管理更容易。

有許多競爭專案提供基於網頁的閃電網路節點管理。以下部分描述了一些最受歡迎的專案。

5.10.1. Ride The Lightning

Ride The Lightning（RTL）是一個圖形化網頁使用者介面，幫助使用者管理三個主要閃電網路節點實作（LND、`c-lightning` 和 Eclair）的閃電網路節點操作。RTL 是由 Shahana Farooqui 和許多其他貢獻者開發的開源專案。你可以在 GitHub (https://github.com/Ride-The-Lightning/RTL) 上找到 RTL 軟體。

RTL 網頁介面範例 顯示了 RTL 網頁介面的範例螢幕截圖，如專案儲存庫所提供。

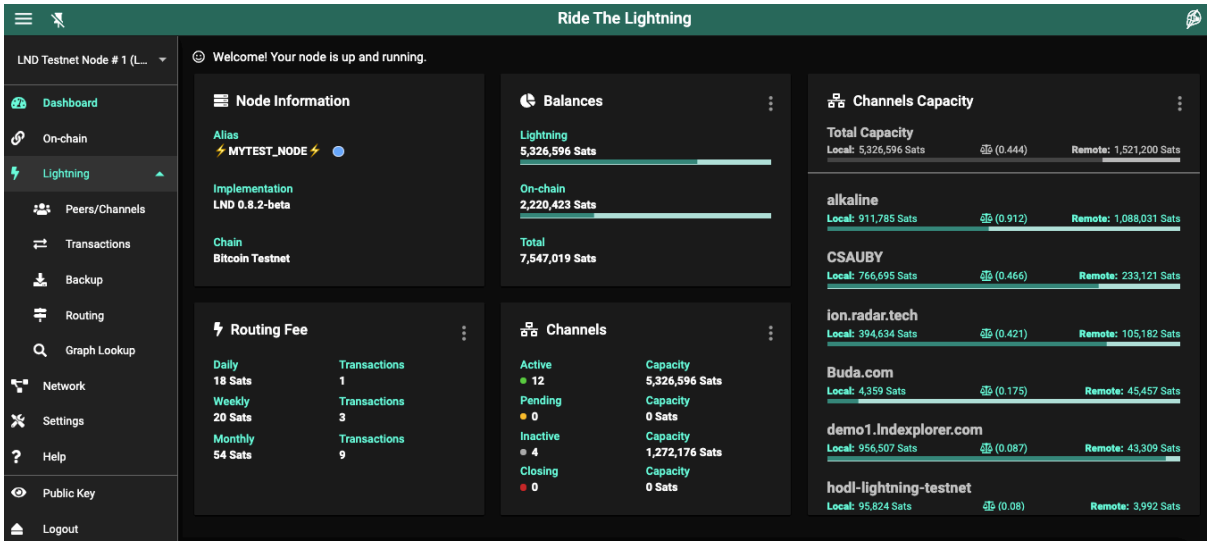

Figure 21. RTL 網頁介面範例

5.10.2. lndmon

Lightning Labs，LND 的製造商，提供了一個名為 lndmon 的基於網頁的圖形使用者介面來監控 LND 閃電網路節點的各種指標。lndmon 僅適用於 LND 節點。它是用於監控的唯讀介面，因此不允許你主動管理節點。它無法開啟通道或進行付款。在 GitHub (https://github.com/lightninglabs/lndmon) 上找到 lndmon。

5.10.3. ThunderHub

ThunderHub (https://thunderhub.io) 是一個非常美觀的基於網頁的圖形使用者介面，類似於 RTL，但專屬於 LND。它可用於進行付款、重新平衡通道，並透過各種功能管理節點。

5.11. 結論

隨著你維護節點並獲得經驗，你將學到很多關於閃電網路的知識。成為節點操作員是一項具有挑戰性但有回報的任務。掌握這些技能將使你能夠為這項技術和閃電網路本身的成長和發展做出貢獻。此外，你將獲得以最大程度的控制和便利發送和接收閃電網路付款的能力。你將在網路基礎設施中扮演核心角色，而不僅僅是邊緣的參與者。

第二部分：技術深入篇

6. 閃電網路架構

在本書的第一部分，我們介紹了閃電網路的主要概念，並透過一個完整的範例來路由付款以及設定我們可以用來進一步探索的工具。在本書的第二部分，我們將更詳細地從技術層面探索閃電網路，剖析每個構建模組。

在本節中，我們將更詳細地概述閃電網路的組成部分，並提供一個「整體視角」來引導你閱讀接下來的章節。

6.1. 閃電網路協定套件

閃電網路由一組運行在網際網路之上的複雜協定集合組成。我們可以將這些協定大致分為五個不同的層次，形成一個_協定堆疊_，其中每一層都建構在下一層之上並使用其協定。同時，每個協定層都抽象化了底層，並「隱藏」了一些複雜性。

閃電網路協定套件 所示的架構圖提供了這些層次及其組成協定的概覽。

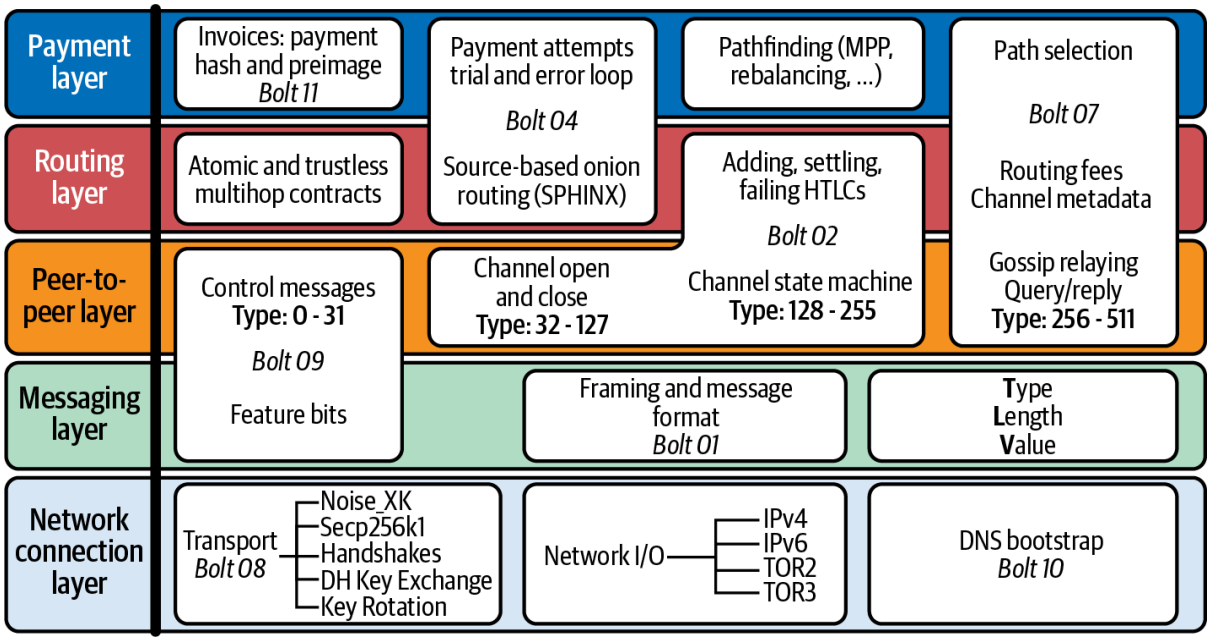

Figure 22. 閃電網路協定套件

閃電網路的五個層次，從下到上分別是：

網路連線層

包含直接與網際網路核心協定（TCP/IP）、覆蓋協定（Tor v2/v3）和網際網路服務（DNS）互動的協定。此層還包含保護閃電網路 訊息 的加密傳輸協定。

訊息層

此層包含節點用於協商功能、格式化訊息和編碼訊息欄位的協定。

點對點（P2P）層

此層是閃電網路節點之間通訊的主要協定層，包含節點之間交換的所有不同訊息。

路由層

此層包含用於在節點之間端對端且原子性地路由付款的協定。此層包含閃電網路的核心功能：路由付款。

支付層

網路的最高層，向應用程式呈現可靠的支付介面。

6.2. 閃電網路詳解

在接下來的 10 章中，我們將剖析協定套件，並詳細檢視閃電網路的每個組成部分。

我們花了相當多的時間試圖決定呈現這些細節的最佳順序。這不是一個容易的選擇，因為不同組件之間有太多相互依賴性：當你開始解釋一個組件時，你會發現它牽涉到相當多其他組件。我們最終選擇了一條更曲折的路徑，而不是自上而下或自下而上的方法，從閃電網路獨有的最基本構建模組——支付通道開始，然後從那裡向外擴展。但由於這條路徑並不明顯，我們將使用 閃電網路協定套件 所示的閃電網路協定套件作為地圖。在每一章中，我們將聚焦於一個或多個相關組件，你會看到它們在協定套件中被突顯出來。就像地圖上的標記說「你在這裡！」

以下是我們將涵蓋的內容：

#payment_channels

在本章中，我們將深入研究支付通道的運作方式，比本書前面部分看到的更加深入。我們將查看資金交易和承諾交易的結構和比特幣腳本，以及節點用於協商協定每個步驟的過程。

#routing

接下來，我們將把多個支付通道組成一個網路，並將付款從一端路由到另一端。在這個過程中，我們將深入研究雜湊時間鎖定合約（HTLC）智慧合約以及我們用來構建它的比特幣腳本。

#channel_operation

將簡單支付通道和使用 HTLC 的路由付款的概念結合起來，我們現在將看看 HTLC 如何成為每個通道承諾交易的一部分。我們還將研究從承諾中添加、結算、失敗和移除 HTLC 的協定。

#onion_routing

接下來，我們將研究 HTLC 資訊如何在洋蔥路由協定內跨網路傳播。我們將研究分層加密和解密的機制，這賦予了閃電網路一些隱私特性。

#gossip

在本章中，我們將研究閃電網路節點如何找到彼此，以及如何了解已發布的通道，以構建可用於在網路中尋找路徑的通道圖。

>#path_finding

接下來，我們將看到每個節點如何使用來自八卦協定的資訊來構建整個網路的「地圖」，用於尋找從一點到另一點的路徑來路由付款。我們還將研究路徑尋找的令人興奮的創新，例如多部分付款。

#wire_protocol

閃電網路的基礎是節點用來交換關於網路和通道訊息的點對點協定。在本章中，我們將研究這些訊息是如何構建的，以及使用功能位元和類型-長度-值（TLV）編碼構建到訊息中的擴展能力。

#encrypted_message_transport

移動到網路的較低層部分，我們將研究確保節點之間所有通訊的保密性和完整性的底層加密傳輸系統。

#invoices

閃電網路的一個關鍵部分是付款請求，也稱為閃電網路發票。在本章中，我們將剖析發票的結構和編碼。

讓我們開始深入探索吧！

7. 支付通道

在本章中，我們將深入探討支付通道，了解它們是如何構建的。我們將從 Alice 的節點向 Bob 的節點開啟通道開始，建構在本書開頭介紹的範例之上。

Alice 和 Bob 的節點交換的訊息在 「BOLT #2：通道管理的對等協定」(https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md) 中定義。Alice 和 Bob 的節點創建的交易在 「BOLT #3：比特幣交易和腳本格式」(https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md) 中定義。在本章中，我們聚焦於閃電網路協定架構的「通道開啟和關閉」以及「通道狀態機」部分，在 閃電網路協定套件中的支付通道 中心（點對點層）用輪廓線突顯。

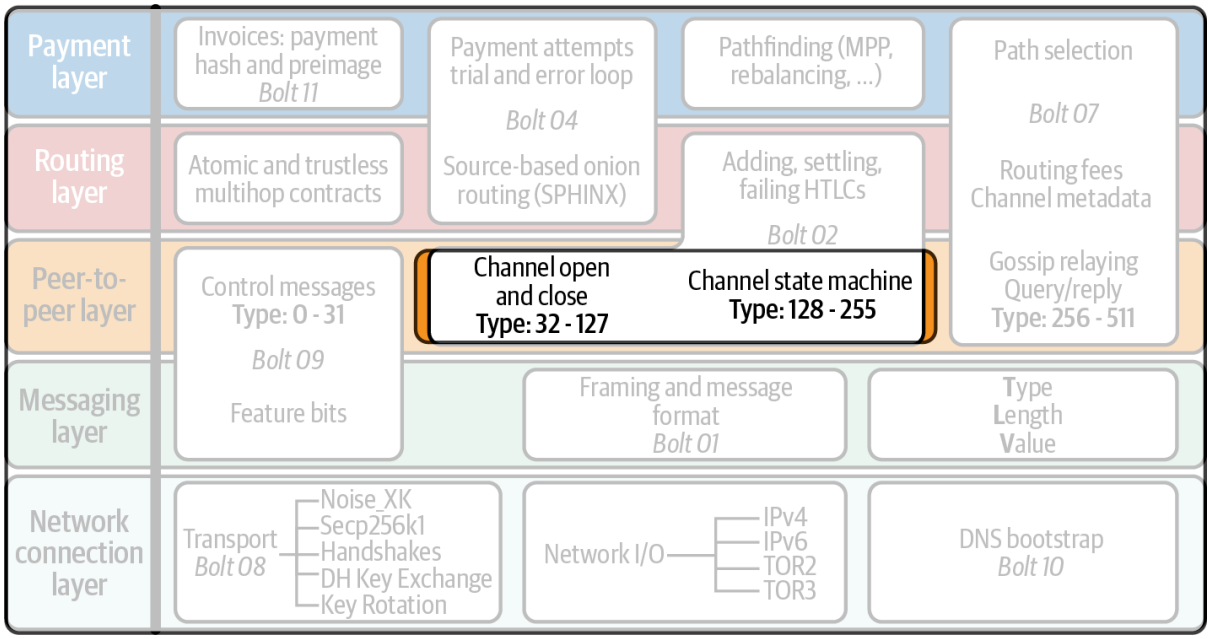

Figure 23. 閃電網路協定套件中的支付通道

7.1. 使用比特幣系統的不同方式

閃電網路通常被描述為「第二層比特幣協定」，這使它聽起來與比特幣截然不同。另一種描述閃電網路的方式是「更智慧的比特幣使用方式」，或者只是「建構在比特幣之上的應用程式」。讓我們來探索一下。

從歷史上看，比特幣交易被廣播給所有人，並記錄在比特幣區塊鏈上才被視為有效。然而，正如我們將看到的，如果某人持有一個預先簽名的比特幣交易，該交易花費一個 2-of-2 多重簽名輸出，給予他們獨家花費該比特幣的能力，那麼他們實際上擁有該比特幣，即使他們沒有廣播該交易。

你可以把預先簽名的比特幣交易想像成一張遠期支票，可以隨時兌現。然而，與傳統銀行系統不同，這筆交易不是付款的「承諾」（也稱為欠條 IOU），而是一種可驗證的持票人票據，相當於現金。只要交易中引用的比特幣在贖回時（或你試圖「兌現」支票時）尚未被花費，比特幣

系統就保證這筆預先簽名的交易可以隨時廣播和記錄。當然，這只有在這是唯一的預先簽名交易時才成立。在閃電網路中，同時存在兩個或多個這樣的預先簽名交易；因此，我們需要一個更複雜的機制來仍然具有這種可驗證持票人票據的功能，正如你將在本章中學到的。

閃電網路只是使用比特幣的一種不同且創造性的方式。在閃電網路中，記錄在區塊鏈上（鏈上）的交易和預先簽名但保留（鏈下）的交易的組合形成了一個「層」的支付，這是一種更快、更便宜、更私密的比特幣使用方式。你可以在 由鏈上和鏈下交易組成的閃電網路支付通道 中看到鏈上和鏈下比特幣交易之間的關係。

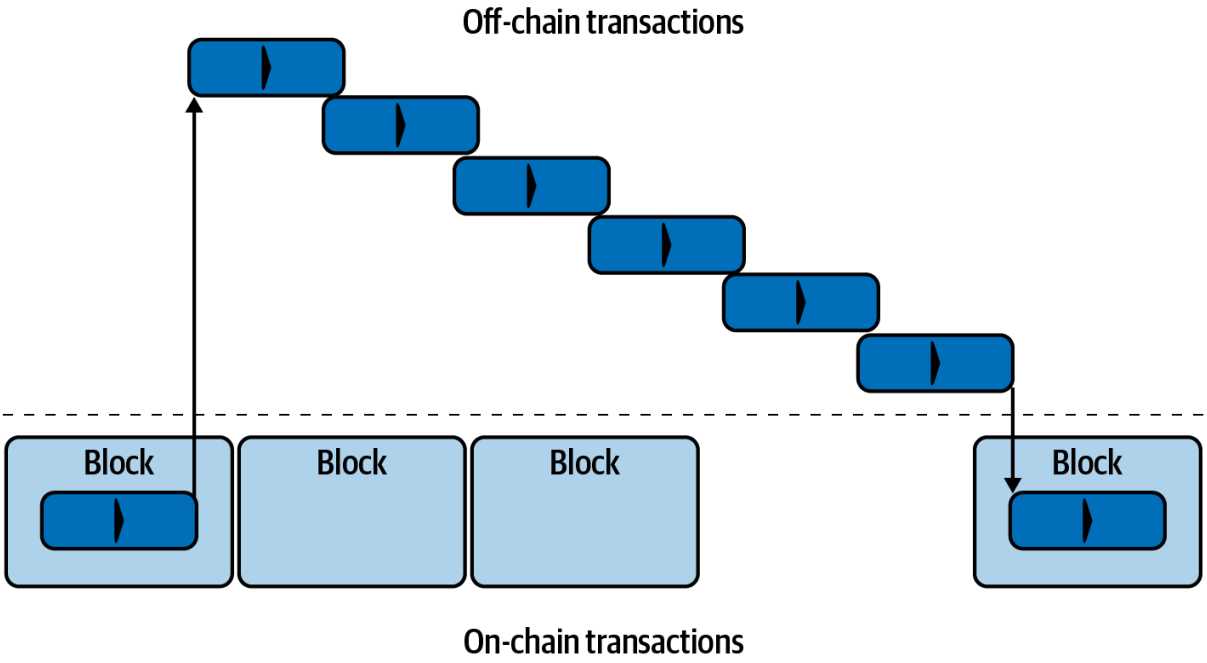

Figure 24. 由鏈上和鏈下交易組成的閃電網路支付通道

閃電網路就是比特幣。它只是使用比特幣系統的不同方式。

7.2. 比特幣的所有權和控制權

在我們理解支付通道之前，我們必須退後一小步，了解比特幣中的所有權和控制權是如何運作的。

當某人說他們「擁有」比特幣時，他們通常是指他們知道某個比特幣地址的私鑰，該地址有一些未花費的交易輸出（參見 比特幣基礎回顧）。私鑰允許他們透過將比特幣轉移到不同的地址來簽署交易以花費該比特幣。在比特幣中，比特幣的「所有權」可以定義為_花費_該比特幣的能力。

請記住，比特幣中使用的「所有權」一詞與法律意義上使用的「所有權」一詞不同。擁有私鑰並能花費比特幣的小偷是該比特幣的_事實上的所有者_，即使他們不是合法所有者。

比特幣的所有權只與金鑰的控制以及使用這些金鑰花費比特幣的能力有關。正如流行的比特幣諺語所說：「你的金鑰，你的代幣——不是你的金鑰，不是你的代幣。」

7.2.1. 獨立所有權的多樣性和多重簽名

私鑰的所有權和控制權並不總是掌握在一個人手中。這就是事情變得有趣和複雜的地方。我們知道，不止一個人可以知道相同的私鑰，無論是透過盜竊還是因為原始金鑰持有者複製並交給其他人。這些人都是所有者嗎？從實際意義上說，他們是的，因為知道私鑰的任何人都可以在未經任何其他人批准的情況下花費比特幣。

比特幣還有多重簽名地址，在花費之前需要多個私鑰簽名（參見 多重簽章腳本）。從實際角度來看，多重簽名地址中的所有權取決於 *K*-of-*N* 方案中定義的法定人數（*K*）和總數（*N*）。1-of-10 多重簽名方案將允許 10 個（*N*）簽名者中的任何 1 個（*K*）花費鎖定在該地址中的比特幣金額。這類似於 10 個人擁有相同私鑰副本且任何人都可以獨立花費的場景。

7.2.2. 沒有獨立控制權的共同所有權

還有一種_無人_擁有法定人數的情況。在閃電網路使用的 2-of-2 方案中，任何一方都不能在未獲得對方簽名的情況下花費比特幣。在這種情況下，誰擁有比特幣？實際上沒有人擁有所有權，因為沒有人擁有控制權。他們每個人擁有相當於決策中的投票權，但需要雙方投票。在比特幣和法律中，2-of-2 方案的一個關鍵問題（雙關語）是，如果其中一方無法聯繫，或者如果投票陷入僵局且任何一方拒絕合作，會發生什麼。

7.2.3. 防止「鎖定」和無法花費的比特幣

如果 2-of-2 多重簽名的兩個簽名者之一不能或不願意簽名，資金將變得無法花費。這種情況不僅可能意外發生（金鑰丟失），而且可以被任何一方用作一種勒索形式：「除非你把部分資金給我，否則我不簽名。」

閃電網路中的支付通道基於 2-of-2 多重簽名地址，兩個通道合作夥伴作為多重簽名中的簽名者。目前，通道只由兩個通道合作夥伴中的一個出資：當你選擇「開啟」一個通道時，你透過交易將資金存入 2-of-2 多重簽名地址。一旦該交易被挖掘並且資金在多重簽名中，如果沒有你的通道合作夥伴的合作，你就無法取回它們，因為你也需要他們的簽名來花費比特幣。

在下一節中，當我們研究如何開啟（創建）閃電網路通道時，我們將看到如何透過實施通道建構的公平協定來防止資金損失或兩個合作夥伴之間的任何勒索情況，借助預先簽名的交易來花費多重簽名輸出，以一種給予通道中對等方獨家能力花費其中一個輸出的方式，該輸出編碼了他們在通道中擁有的比特幣數量。

7.3. 構建支付通道

在 什麼是支付通道？ 中，我們將支付通道描述為兩個閃電網路節點之間的_金融關係_，透過向兩個通道合作夥伴的 2-of-2 多重簽名地址注資而建立。

假設 Alice 想要構建一個支付通道，允許她直接連接到 Bob 的商店。首先，兩個節點（Alice 的和 Bob 的）必須建立彼此之間的網路連線，以便他們可以協商支付通道。

7.3.1. 節點私鑰和公鑰

閃電網路上的每個節點都由一個_節點公鑰_標識。公鑰唯一標識特定節點，通常以十六進制編碼呈現。例如，René Pickhardt 目前運行一個閃電網路節點（ln.rene-pickhardt.de），由以下節點公鑰標識：

```
02a1cebfacb2674143b5ad0df3c22c609e935f7bc0ebe801f37b8e9023d45ea7b8
```

每個節點在首次初始化時生成一個根私鑰。私鑰始終保持私密（從不分享）並安全地儲存在節點的錢包中。從該私鑰，節點衍生出作為節點標識符並與網路分享的公鑰。由於金鑰空間非常龐大，只要每個節點隨機生成私鑰，它就會擁有唯一的公鑰，因此可以在網路上唯一標識它。

7.3.2. 節點網路地址

此外，每個節點還會公告可以聯繫到它的網路地址，以下幾種可能的格式之一：

TCP/IP

IPv4 或 IPv6 地址和 TCP 埠號

TCP/Tor

Tor「洋蔥」地址和 TCP 埠號

網路地址標識符寫作 Address:Port，這與國際標準的網路標識符一致，例如在網頁上使用的。

例如，René 的節點公鑰為 02a1ceb...45ea7b8，目前公告其網路地址為 TCP/IP 地址：

```
172.16.235.20:9735
```

閃電網路的預設 TCP 埠是 9735，但節點可以選擇監聽任何 TCP 埠。

7.3.3. 節點標識符

節點公鑰和網路地址一起，以以下格式書寫，用 @ 符號分隔，如 *NodeID@Address:Port*。

所以 René 的節點的完整標識符將是：

```
02a1cebfacb2674143b5ad0df3c22c609e935f7bc0ebe801f37b8e9023d45ea7b8
@172.16.235.20:9735
```

René 的節點別名是 ln.rene-pickhardt.de；然而，這個名稱只是為了更好的可讀性而存在。每個節點操作員都可以公告他們想要的任何別名，沒有機制阻止節點操作員選擇已經被使用的別名。因此，要引用一個節點，必須使用 *NodeID@Address:Port* 格式。

前面的標識符通常編碼在 QR 碼中，如果使用者想要將他們自己的節點連接到由該地址標識的特定節點，可以更容易地掃描。

就像比特幣節點一樣，閃電網路節點透過「八卦」其節點公鑰和網路地址來公告它們在閃電網路上的存在。這樣，其他節點可以找到它們，並保持一個清單（資料庫），記錄所有它們可以連接並交換閃電網路 P2P 訊息協定中定義的訊息的已知節點。

7.3.4. 將節點連接為直接對等方

為了讓 Alice 的節點連接到 Bob 的節點，她需要 Bob 的節點公鑰，或者包含公鑰、IP 或 Tor 地址和埠的完整地址。因為 Bob 經營一家商店，Bob 的節點地址可以從發票或網路上的商店支付頁面獲取。Alice 可以掃描包含地址的 QR 碼，並指示她的節點連接到 Bob 的節點。

一旦 Alice 連接到 Bob 的節點，他們的節點現在就是直接連接的對等方。

要開啟支付通道，兩個節點必須首先透過網路（或 Tor）建立連線，作為直接對等方連接。

7.4. 構建通道

現在 Alice 和 Bob 的閃電網路節點已經連接，他們可以開始構建支付通道的過程。在本節中，我們將回顧他們節點之間的通訊，稱為_通道管理的閃電網路對等協定_，以及他們用於構建比特幣交易的加密協定。

我們在這個場景中描述兩個不同的協定。首先，有一個_訊息協定_，它建立閃電網路節點如何透過網路通訊以及它們彼此交換什麼訊息。其次，有_加密協定_，它建立兩個節點如何構建和簽署比特幣 交易。

7.4.1. 通道管理的對等協定

通道管理的閃電網路對等協定在 BOLT #2：通道管理的對等協定 (https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md) 中定義。在本章中，我們將更詳細地回顧 BOLT #2 的「通道建立」和「通道關閉」部分。

7.4.2. 通道建立訊息流程

通道建立是透過 Alice 和 Bob 節點之間交換六個訊息實現的（每個對等方三個）：open_channel、accept_channel、funding_created、funding_signed、funding_locked 和 funding_locked。這六個訊息在 通道建立訊息流程 中以時序圖顯示。

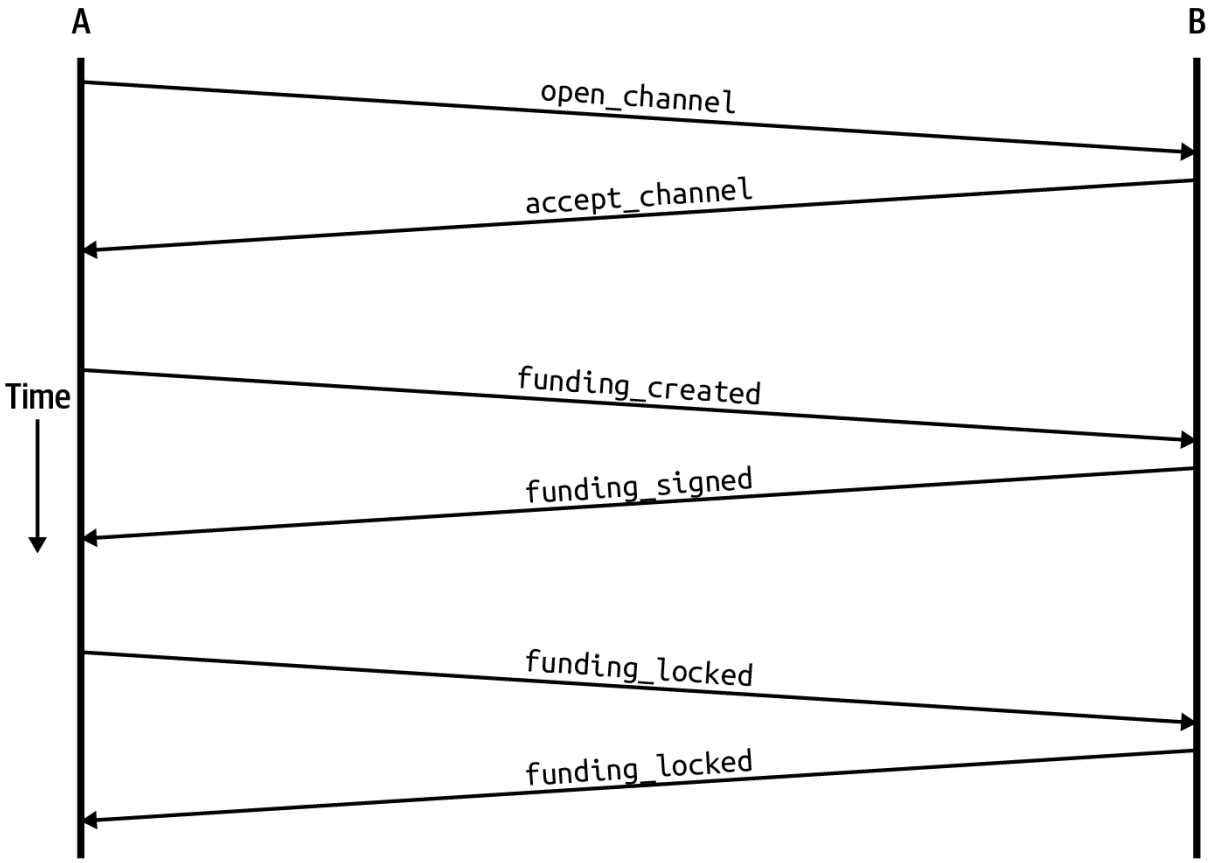

Figure 25. 通道建立訊息流程

在 通道建立訊息流程 中，Alice 和 Bob 的節點由圖表兩側的垂直線「A」和「B」表示。像這樣的時序圖顯示時間向下流動，訊息在兩個通訊對等方之間從一側流向另一側。線條向下傾斜以表示傳輸每個訊息所需的時間，訊息的方向由每條線末端的箭頭顯示。

通道建立涉及三個部分。首先，兩個對等方透過 Alice 發起 open_channel 請求和 Bob 透過 accept_channel 接受通道請求來溝通他們的能力和期望。

其次，Alice 構建資金和退款交易（我們將在本節稍後看到），並向 Bob 發送 funding_created。「退款」交易的另一個名稱是「承諾」交易，因為它承諾了通道中餘額的當前分配。Bob 透過 funding_signed 回覆必要的簽名。這種互動是保護通道和防止盜竊的_加密協定_的基礎。Alice 現在將廣播資金交易（鏈上）以建立和錨定支付通道。該交易需要在比特幣區塊鏈上得到確認。

> funding_signed 訊息的名稱可能有點令人困惑。此訊息不包含資金交易的簽名，而是包含 Bob 對退款交易的簽名，允許 Alice 從多重簽名中取回她的比特幣。

一旦交易有足夠的確認（由 `accept_channel` 訊息中的 `minimum_depth` 欄位定義），Alice 和 Bob 交換 funding_locked 訊息，通道進入正常運作模式。

open_channel 訊息

Alice 的節點透過發送 open_channel 訊息向 Bob 的節點請求支付通道。該訊息包含關於 Alice 對通道設定的_期望_的資訊，Bob 可以接受或拒絕。

open_channel 訊息的結構（取自 BOLT #2）如 `open_channel` 訊息 所示。

Example 1. `open_channel` *訊息*

```
[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]
```

此訊息中包含的欄位指定了 Alice 想要的通道參數，以及來自 Alice 節點的各種配置設定，反映了通道運作的安全期望。

以下列出了一些通道建構參數：

chain_hash

這標識將用於此通道的區塊鏈（例如比特幣主網）。它通常是該區塊鏈創世區塊的雜湊。

funding_satoshis

Alice 用於為通道注資的金額，即總通道容量。

channel_reserve_satoshis

在通道每一側保留的最低餘額，以 satoshi 為單位。我們在談論懲罰時會回到這個問題。

push_msat

一個可選金額，Alice 將在通道注資後立即「推送」給 Bob 作為付款。*將此值設定為 0 以外的任何值實際上意味著將資金贈送給你的通道合作夥伴，應謹慎使用。*

to_self_delay

協定的一個非常重要的安全參數。`open_channel` 訊息中的值用於回應者的承諾交易，`accept_channel` 中的值用於發起者的承諾交易。這種不對稱性的存在是為了允許每一方表達對方在單方面索取承諾交易中的資金時需要等待多長時間。如果 Bob 在任何時候違背 Alice 的意願單方面關閉通道，他承諾在這裡定義的延遲期內不存取他自己的資金。這個值越高，Alice 的安全性越高，但 Bob 的資金可能被鎖定的時間越長。

funding_pubkey

Alice 將貢獻給錨定此通道的 2-of-2 多重簽名的公鑰。

X_basepoint

主金鑰，用於為承諾、撤銷、路由付款（HTLCs）和關閉交易的各個部分衍生子金鑰。這些將在後續章節中使用和解釋。

如果你想了解我們在本書中未討論的其他欄位和閃電網路對等協定訊息，我們建議你在 BOLT 規範中查閱它們。這些訊息和欄位很重要，但無法在本書的範圍內足夠詳細地涵蓋。我們希望你能夠充分理解基本原則，以便你可以透過閱讀實際的協定規範（BOLTs）來填補細節。

accept_channel 訊息

回應 Alice 的 open_channel 訊息，Bob 發回 `accept_channel` 訊息 中顯示的 accept_channel 訊息。

Example 2. `accept_channel` *訊息*

```
[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]
```

如你所見，這與 open_channel 訊息類似，包含 Bob 的節點期望和配置值。

Alice 將用於構建支付通道的 accept_channel 中最重要的兩個欄位是：

funding_pubkey

Bob 的節點為錨定通道的 2-of-2 多重簽名地址貢獻的公鑰。

minimum_depth

Bob 的節點期望資金交易在區塊鏈上有多少確認數才認為通道「開啟」並準備使用。

7.4.3. 資金交易

一旦 Alice 的節點收到 Bob 的 accept_channel 訊息，它就有了構建將通道錨定到比特幣區塊鏈的_資金交易_所需的資訊。正如我們在前幾章中討論的，閃電網路支付通道由 2-of-2 多重簽名地址錨定。首先，我們需要生成該多重簽名地址，以便我們可以構建資金交易（以及隨後描述的退款交易）。

7.4.4. 生成多重簽名地址

資金交易將一定數量的比特幣（來自 open_channel 訊息的 funding_satoshis）發送到由 Alice 和 Bob 的 funding_pubkey 公鑰構建的 2-of-2 多重簽名輸出。

Alice 的節點構建如下所示的多重簽名腳本：

```
2 <Alice_funding_pubkey> <Bob_funding_pubkey> 2 CHECKMULTISIG
```

請注意，在實踐中，資金金鑰在放入見證腳本之前會確定性地_排序_（使用序列化壓縮格式公鑰的字典順序）。透過事先同意這種排序順序，我們確保雙方將構建相同的資金交易輸出，該輸出由交換的承諾交易簽名簽署。

此腳本編碼為 Pay-to-Witness-Script-Hash（P2WSH）比特幣地址，看起來像這樣：

```
bc1q89ju02heg32yrqdrnqghe6132wek25p6sv6e564znvrvez7tq5zqt4dn02
```

7.4.5. 構建資金交易

Alice 的節點現在可以構建資金交易，將與 Bob 商定的金額（`funding_satoshis`）發送到 2-of-2 多重簽名地址。假設 funding_satoshis 是 140,000，Alice 正在花費一個 200,000 satoshi 的輸出並創建 60,000 satoshi 的找零。交易看起來會像 Alice 構建資金交易。

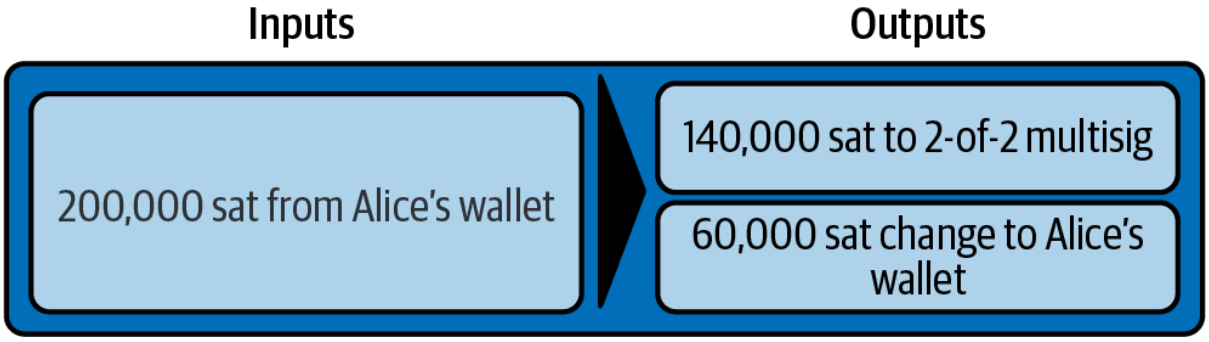

Figure 26. Alice 構建資金交易

Alice _不廣播_這筆交易，因為這樣做會讓她的 140,000 satoshi 面臨風險。一旦花費到 2-of-2 多重簽名，如果沒有 Bob 的簽名，Alice 就無法取回她的錢。

雙向注資支付通道

在閃電網路的當前實作中，通道只由發起通道的節點（在我們的例子中是 Alice）注資。雙向注資通道已被提出，但尚未實作。在雙向注資通道中，Alice 和 Bob 都將向資金交易貢獻輸入。雙向注資通道需要稍微更複雜的訊息流程和加密協定，因此它們尚未實作，但計劃在未來的閃電網路 BOLT 更新中實作。`c-lightning` 實作包含雙向注資通道變體的實驗版本。

7.4.6. 持有已簽名的交易而不廣播

使閃電網路成為可能的一個重要比特幣功能是構建和簽署交易但不廣播它們的能力。該交易在各方面都是_有效的_，但在它被廣播並在比特幣區塊鏈上確認之前，它不被承認，其輸出不能花費，因為它們尚未在區塊鏈上創建。我們將在閃電網路中多次使用這種能力，Alice 的節點在構建資金交易時使用這種能力：持有它並且尚未廣播它。

7.4.7. 先有退款後有注資

為了防止資金損失，Alice 在有辦法在出問題時獲得退款之前，不能將她的比特幣放入 2-of-2。本質上，她必須在進入這種安排之前計劃好「退出」通道。

考慮婚前協議的法律構造，也稱為「prenup」。當兩個人結婚時，他們的錢依法（取決於司法管轄區）綁定在一起。在結婚之前，他們可以簽署一份協議，指定如果他們透過離婚解除婚姻，如何分割他們的資產。

我們可以在比特幣中創建類似的協議。例如，我們可以創建一個退款交易，它的功能類似於婚前協議，允許雙方在他們的資金實際鎖定到多重簽名資金地址之前決定如何分配他們通道中的資金。

7.4.8. 構建預先簽名的退款交易

Alice 將在構建（但不廣播）資金交易後立即構建退款交易。退款交易將 2-of-2 多重簽名 花費回 Alice 的錢包。我們將這個退款交易稱為_承諾交易_，因為它承諾雙方通道合作夥伴公平分配通道餘額。由於 Alice 獨自為通道注資，她獲得全部餘額，Alice 和 Bob 都承諾用這筆交易退款給 Alice。

在實踐中，它比較複雜，我們將在後續章節中看到，但現在讓我們保持簡單，假設它看起來像 Alice 也構建退款交易。

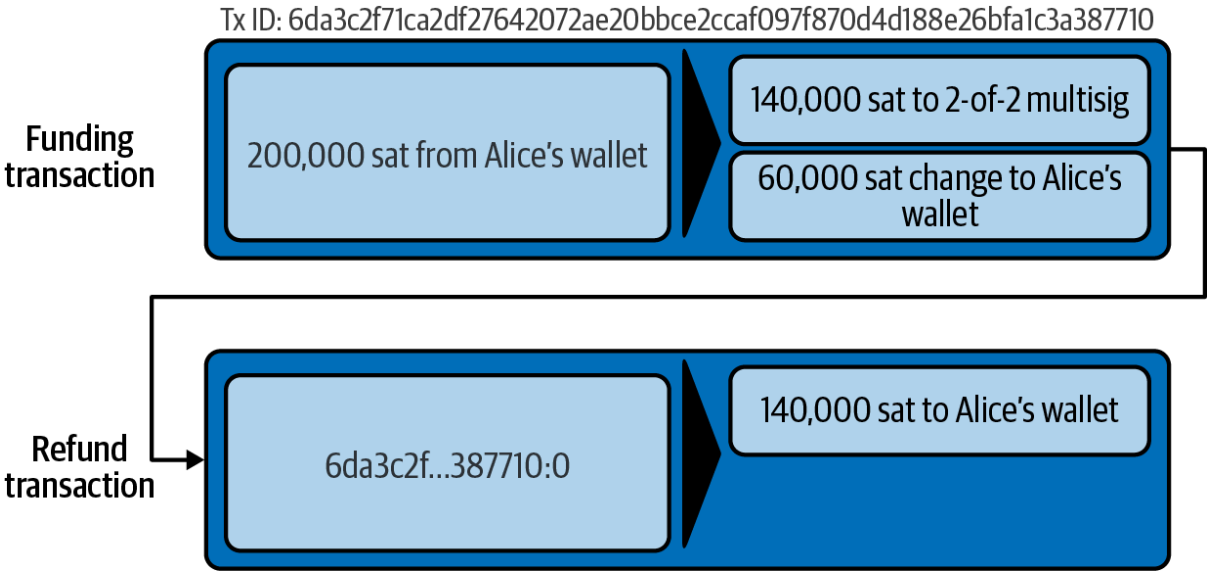

Figure 27. Alice 也構建退款交易

在本章後面，我們將看到如何創建更多承諾交易以不同金額分配通道餘額。

7.4.9. 在不廣播的情況下鏈接交易

現在，Alice 已經構建了 Alice 也構建退款交易 中顯示的兩個交易。但你可能想知道這是怎麼可能的。Alice 還沒有將資金交易廣播到比特幣區塊鏈。就網路上的每個人而言，該交易不存在。退款交易被構建為_花費_資金交易的其中一個輸出，即使該輸出也還不存在。如何花費尚未在比特幣區塊鏈上確認的輸出？

退款交易尚不是有效交易。要使其成為有效交易，必須發生兩件事：

- 資金交易必須廣播到比特幣網路。（為了確保閃電網路的安全，我們還要求它被比特幣區塊鏈確認，儘管這對於鏈接 交易 並不是嚴格必要的。）
- 退款交易的輸入需要 Alice 和 Bob 的簽名。

但即使這兩件事尚未發生，即使 Alice 的節點尚未廣播資金交易，她仍然可以構建退款交易。她可以這樣做，因為她可以計算資金交易的雜湊並在退款交易中將其引用為輸入。

請注意 Alice 是如何計算出 6da3c2...387710 作為資金交易雜湊的？如果資金交易被廣播，該雜湊將被記錄為資金交易的交易 ID。因此，資金交易的 `0` 輸出（2-of-2 地址輸出）將被引用為輸出 ID 6da3c2...387710:0。即使退款交易尚不存在，也可以構建為花費該資金交易輸出，因為 Alice 知道一旦確認其標識符將是什麼。

這意味著 Alice 可以透過引用尚不存在的輸出來創建鏈接交易，知道如果資金交易被確認，該引用將是有效的，使退款交易也有效。正如我們將在下一節中看到的，這種在廣播之前鏈接交易的「技巧」需要比特幣在 2017 年 8 月引入的一個非常重要的功能：*隔離見證*。

7.4.10. 解決可塑性問題（隔離見證）

Alice 必須依賴在確認之前知道資金交易的交易 ID。但在 2017 年 8 月引入隔離見證（SegWit）之前，這不足以保護 Alice。由於交易的構建方式，簽名（見證）包含在交易 ID 中，第三方（例如 Bob）有可能廣播一個具有_可塑_（修改的）交易 ID 的替代版本交易。這被稱為_交易可塑性_，在 SegWit 之前，這個問題使得安全實作無限期生命週期的支付通道變得困難。

如果 Bob 可以在確認之前修改 Alice 的資金交易，並產生具有不同交易 ID 的副本，Bob 可以使 Alice 的退款交易無效並劫持她的比特幣。Alice 將任由 Bob 擺佈，需要他的簽名來釋放她的資金，並且很容易被勒索。Bob 無法竊取資金，但他可以阻止 Alice 取回資金。

SegWit 的引入使未確認的交易 ID 從第三方的角度變得不可變，這意味著 Alice 可以確定資金交易的交易 ID 不會改變。因此，Alice 可以確信，如果她獲得 Bob 對退款交易的簽名，她就有辦法收回她的錢。她現在有辦法在將資金鎖定到多重簽名之前實施比特幣版的「婚前協議」。

> 你可能想知道 Bob 如何能夠更改（可塑化）Alice 創建並簽署的交易。Bob 當然沒有 Alice 的私鑰。然而，訊息的 ECDSA 簽名不是唯一的。知道簽名（包含在有效交易中）允許人們產生許多看起來不同但仍然有效的簽名。在 SegWit 從交易摘要演算法中移除簽名之前，Bob 可以用等效的有效簽名替換簽名，產生不同的交易 ID，打破資金交易和退款交易之間的鏈接。

funding_created 訊息

現在 Alice 已經構建了必要的交易，通道建構訊息流程繼續。Alice 向 Bob 傳輸 funding_created 訊息。你可以在這裡看到此訊息的內容：

funding_created 訊息

```
[32*byte:temporary_channel_id]
[sha256:funding_txid]
[u16:funding_output_index]
[signature:signature]
```

透過此訊息，Alice 向 Bob 提供了錨定支付通道的資金交易的重要資訊：

funding_txid

這是資金交易的交易 ID（TxID），用於在通道建立後創建通道 ID。

funding_output_index

這是輸出索引，讓 Bob 知道交易的哪個輸出（例如輸出 `0`）是 Alice 注資的 2-of-2 多重簽名輸出。這也用於形成通道 ID。

最後，Alice 還發送對應於 Alice 的 `funding_pubkey` 的 signature，用於從 2-of-2 多重簽名花費。Bob 需要這個是因為他也需要創建他自己版本的承諾交易。該承諾交易需要 Alice 的簽名，她提供給他。請注意，Alice 和 Bob 的承諾交易看起來略有不同，因此簽名也會不同。知

道對方的承諾交易是什麼樣子是至關重要的，也是提供有效簽名的協定的一部分。

在閃電網路協定中，我們經常看到節點發送簽名而不是整個已簽名的交易。這是因為任何一方都可以重建相同的交易，因此只需要簽名就可以使其有效。只發送簽名而不是整個交易可以節省大量網路頻寬。

funding_signed 訊息

從 Alice 收到 funding_created 訊息後，Bob 現在知道了資金交易 ID 和輸出索引。通道 ID 是透過資金交易 ID 和輸出索引的逐位元「互斥或」（XOR）生成的：

```
channel_id = funding_txid XOR funding_output_index
```

更準確地說，`channel_id` 是資金 UTXO 的 32 位元組表示，透過將資金 TxID 的低 2 位元組與資金輸出的索引進行 XOR 運算生成。

Bob 還需要向 Alice 發送他對退款交易的簽名，基於形成 2-of-2 多重簽名的 Bob 的 `funding_pubkey`。雖然 Bob 已經有他的本地退款交易，但這將允許 Alice 用所有必要的簽名完成退款交易，並確保她的錢在出問題時可以退還。

Bob 構建一個 funding_signed 訊息並發送給 Alice。在這裡我們看到此訊息的內容：

funding_signed 訊息

```
[channel_id:channel_id]
[signature:signature]
```

7.4.11. 廣播資金交易

從 Bob 收到 funding_signed 訊息後，Alice 現在有了簽署退款交易所需的兩個簽名。她的「退出計劃」現在是安全的，因此她可以廣播資金交易而不用擔心資金被鎖定。如果出了什麼問題，Alice 可以簡單地廣播退款交易並取回她的錢，而無需 Bob 的任何進一步幫助。

Alice 現在將資金交易發送到比特幣網路，以便它可以被挖掘到區塊鏈中。Alice 和 Bob 都將監控這筆交易，並等待比特幣區塊鏈上的 minimum_depth 確認（例如六次確認）。

當然，Alice 將使用比特幣協定來驗證 Bob 發送給她的簽名是否確實有效。這一步非常關鍵。如果出於某種原因 Bob 向 Alice 發送了錯誤的資料，她的「退出計劃」將被破壞。

funding_locked 訊息

一旦資金交易達到所需的確認數，Alice 和 Bob 都會向對方發送 funding_locked 訊息，通道就準備好使用了。

7.5. 透過通道發送付款

通道已經建立，但在初始狀態下，所有容量（140,000 satoshi）都在 Alice 這一側。這意味著 Alice 可以透過通道向 Bob 發送付款，但 Bob 還沒有資金可以發送給 Alice。

在接下來的幾節中，我們將展示付款是如何透過支付通道進行的，以及_通道狀態_是如何更新的。

假設 Alice 想要向 Bob 發送 70,000 satoshi 來支付她在 Bob 咖啡店的帳單。

7.5.1. 分割餘額

原則上，從 Alice 向 Bob 發送付款只是重新分配通道餘額的問題。在付款發送之前，Alice 有 140,000 satoshi，Bob 沒有。在 70,000 satoshi 付款發送後，Alice 有 70,000 satoshi，Bob 也有 70,000 satoshi。

因此，Alice 和 Bob 所要做的就是創建並簽署一個交易，將 2-of-2 多重簽名花費到兩個輸出，分別支付 Alice 和 Bob 他們對應的餘額。我們將這個更新的交易稱為_承諾交易_。

Alice 和 Bob 透過_推進通道狀態_來操作支付通道，透過一系列承諾。每個承諾更新餘額以反映流經通道的付款。Alice 和 Bob 都可以發起新的承諾來更新通道。

在 多個承諾交易 中我們看到幾個承諾交易。

多個承諾交易 中顯示的第一個承諾交易是 Alice 在為通道注資之前構建的退款交易。在圖中，這是承諾 #0。在 Alice 向 Bob 支付 70,000 satoshi 後，新的承諾交易（承諾 #1）有兩個輸出，分別支付 Alice 和 Bob 各自的餘額。我們包含了兩個後續的承諾交易（承諾 #2 和承諾 #3），分別代表 Alice 向 Bob 額外支付 10,000 satoshi 和然後 20,000 satoshi。

每個簽署且有效的承諾交易都可以被任一通道合作夥伴在任何時候用於關閉通道，透過將其廣播到比特幣網路。由於他們都有最新的承諾交易並且可以隨時使用它，他們也可以只是持有它而不廣播。這是他們公平退出通道的保證。

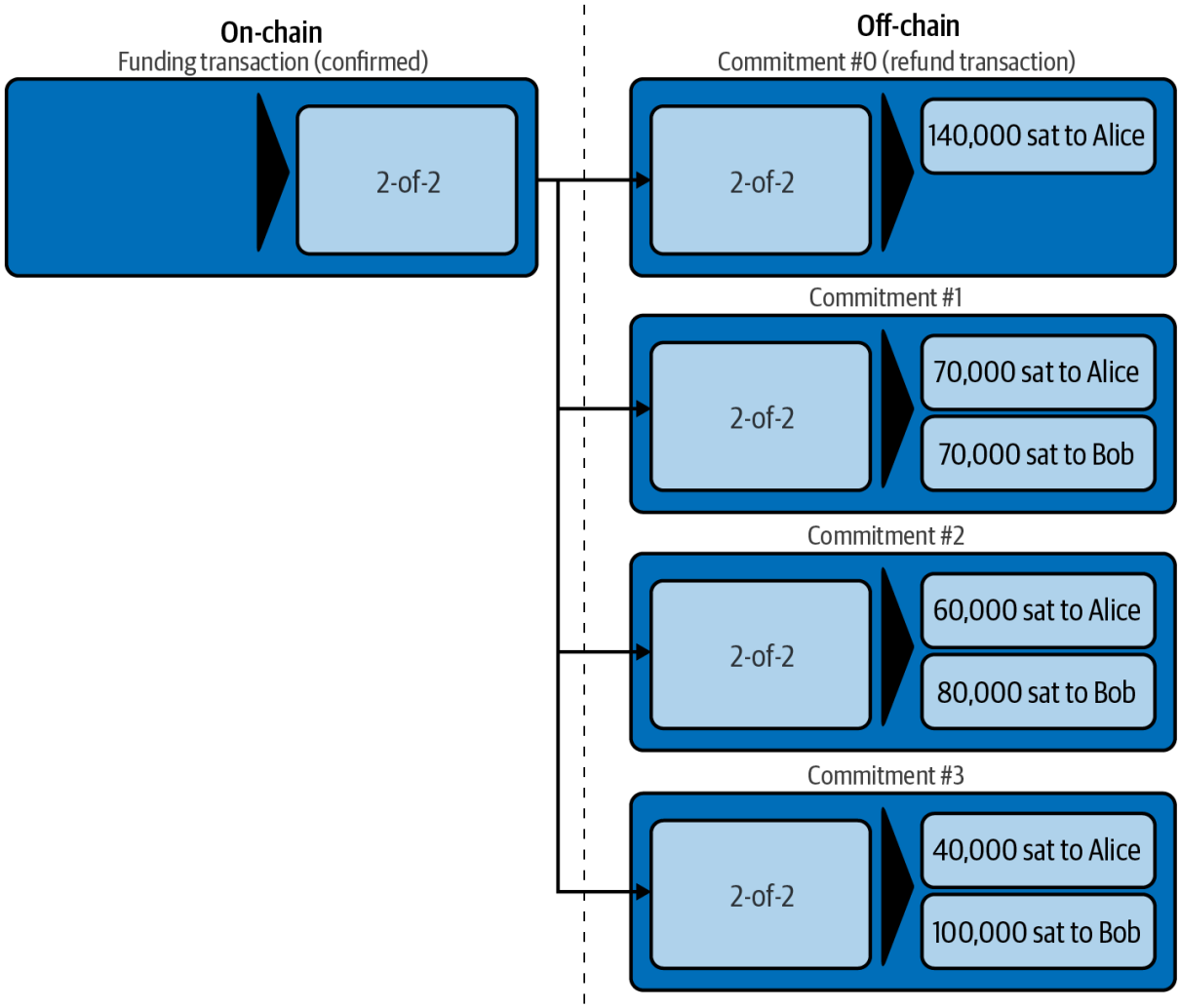

Figure 28. 多個承諾交易

7.5.2. 競爭的承諾

你可能想知道 Alice 和 Bob 怎麼可能有多個承諾交易，所有這些交易都試圖花費資金交易的同一個 2-of-2 輸出。這些承諾交易不是衝突的嗎？這不是比特幣系統旨在防止的「雙重花費」嗎？

確實是！事實上，我們依賴比特幣_防止_雙重花費的能力來使閃電網路運作。無論 Alice 和 Bob 構建和簽署多少承諾交易，只有其中一個可以真正得到確認。

只要 Alice 和 Bob 持有這些交易而不廣播它們，資金輸出就是未花費的。但如果一個承諾交易被廣播並確認，它將花費資金輸出。如果 Alice 或 Bob 試圖廣播多個承諾交易，只有其中一個會被確認，其他的將被拒絕為嘗試（且失敗的）雙重花費。

如果多個承諾交易被廣播，有許多因素會決定哪一個首先被確認：包含的費用金額、這些競爭交易的傳播速度、網路拓撲等。本質上它變成了一場結果不可預測的競賽。這聽起來不太安全。聽起來像是有人可以作弊。

7.5.3. 用舊承諾交易作弊

讓我們更仔細地看看 多個承諾交易 中的承諾交易。所有四個承諾交易都已簽署且有效。但只有最後一個準確反映了最新的通道餘額。在這個特定場景中，Alice 有機會透過廣播一個較舊的承諾並讓它在比特幣區塊鏈上確認來作弊。假設 Alice 傳輸承諾 #0 並讓它確認：她將有效地關閉通道並獨自拿走所有 140,000 satoshi。事實上，在這個特定例子中，除了承諾 #3 以外的任何承諾都會改善 Alice 的狀況，並允許她「取消」至少部分反映在通道中的付款。

在下一節中，我們將看到閃電網路如何透過撤銷和懲罰機制解決這個問題——防止較舊的承諾交易被通道合作夥伴使用。還有其他方法可以防止傳輸較舊的承諾交易，例如 eltoo 通道，但它們需要對比特幣進行升級，稱為輸入重新綁定（參見 比特幣協定和比特幣腳本創新）。

7.5.4. 撤銷舊承諾交易

比特幣交易不會過期，也不能被「取消」。一旦廣播，它們也不能被停止或審查。那麼我們如何「撤銷」另一個人持有的已經簽署的交易呢？

閃電網路中使用的解決方案是公平協定的另一個例子。不是試圖控制廣播交易的能力，而是有一個內建的_懲罰機制_，確保潛在作弊者傳輸舊承諾交易不符合其最佳利益。他們總是可以廣播它，但如果他們這樣做，很可能會損失金錢。

> 「撤銷」這個詞是用詞不當，因為它暗示舊承諾在某種程度上變得無效，不能被廣播和確認。但事實並非如此，因為有效的比特幣交易不能被撤銷。相反，閃電網路協定使用懲罰機制來懲罰廣播舊承諾的通道合作夥伴。

閃電網路協定的撤銷和懲罰機制有三個組成要素：

不對稱承諾交易

Alice 的承諾交易與 Bob 持有的略有不同。

延遲花費

向持有承諾交易的一方的付款被延遲（時間鎖定），而向另一方的付款可以立即領取。

撤銷金鑰

用於解鎖舊承諾的懲罰選項。

讓我們依次看看這三個要素。

7.5.5. 不對稱承諾交易

Alice 和 Bob 持有略有不同的承諾交易。讓我們更詳細地看看 多個承諾交易 中的承諾 #2，如 承諾交易 #2 所示。

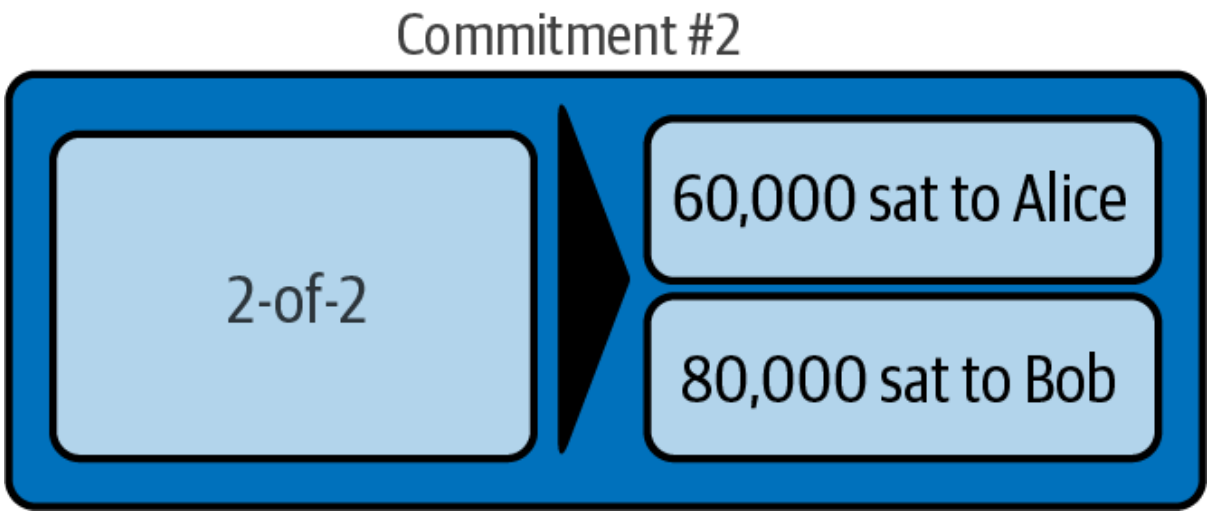

Figure 29. 承諾交易 #2

Alice 和 Bob 持有此交易的兩個不同變體，如 不對稱承諾交易 所示。

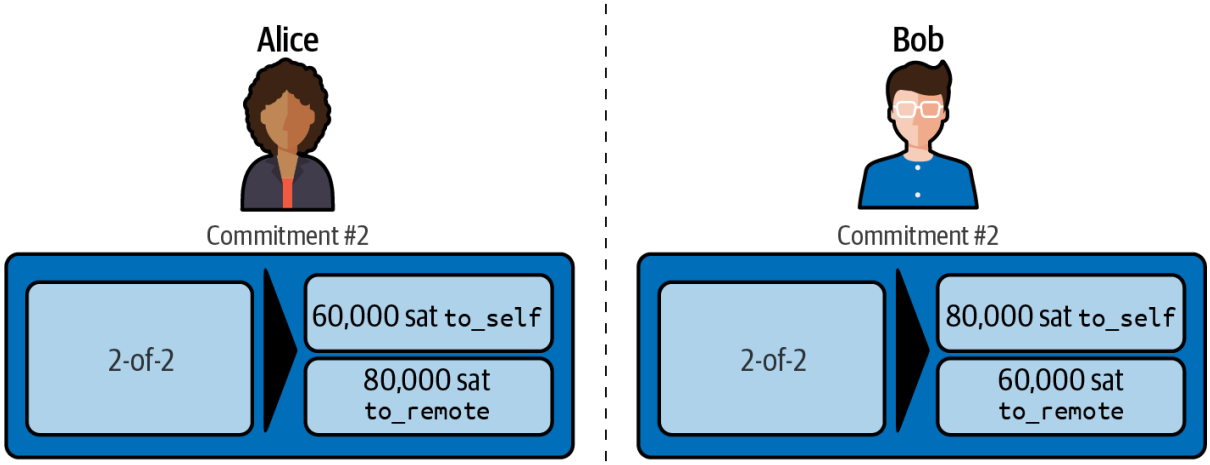

Figure 30. 不對稱承諾交易

按照慣例，在閃電網路協定中，我們將兩個通道合作夥伴稱為 `self`（也稱為 `local`）和 `remote`，取決於我們從哪一側看。支付給每個通道合作夥伴的輸出分別稱為 `to_local` 和 `to_remote`。

在 不對稱承諾交易 中，我們看到 Alice 持有一個交易，支付 60,000 satoshi `to_self`（可以由 Alice 的金鑰花費），和 80,000 satoshi `to_remote`（可以由 Bob 的金鑰花費）。

Bob 持有該交易的鏡像，其中第一個輸出是 80,000 satoshi `to_self`（可以由 Bob 的金鑰花費），和 60,000 satoshi `to_remote`（可以由 Alice 的金鑰花費）。

7.5.6. 延遲（時間鎖定）花費 to_self

使用不對稱交易允許協定輕鬆將_責任_歸咎於作弊方。確保_廣播_方必須始終等待的不變性確保「誠實」方有時間反駁索賠並撤銷他們的資金。這種不對稱性體現在每一方的不同輸出形式中：`to_local` 輸出始終被時間鎖定，不能立即花費，而 `to_remote` 輸出沒有時間鎖定，可以立即花費。

例如，在 Alice 持有的承諾交易中，支付給她的 `to_local` 輸出被時間鎖定 432 個區塊，而支付給 Bob 的 `to_remote` 輸出可以立即花費（參見 不對稱和延遲的承諾交易）。Bob 對承諾 #2 的承諾交易是鏡像的：他自己的（`to_local`）輸出被時間鎖定，Alice 的 `to_remote` 輸出可以立即花費。

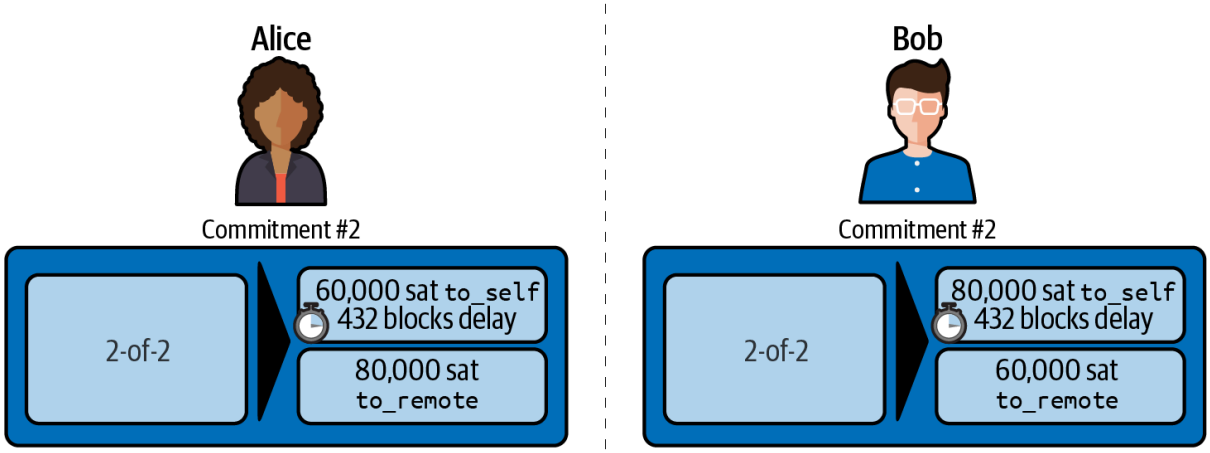

Figure 31. 不對稱和延遲的承諾交易

這意味著如果 Alice 透過廣播並確認她持有的承諾交易來關閉通道，她在 432 個區塊內不能花費她的餘額，但 Bob 可以立即領取他的餘額。如果 Bob 使用他持有的承諾交易關閉通道，他在 432 個區塊內不能花費他的輸出，而 Alice 可以立即花費她的。

延遲存在的一個原因是：允許_遠端_方在舊的（已撤銷的）承諾被另一個通道合作夥伴廣播時行使懲罰選項。接下來讓我們看看撤銷金鑰和懲罰選項。

延遲是在初始通道建構訊息流程中由 Alice 和 Bob 協商的，作為一個名為 to_self_delay 的欄位。為了確保通道的安全，延遲會根據通道容量進行調整——這意味著擁有更多資金的通道在承諾的 to_self 輸出中有更長的延遲。Alice 的節點在 open_channel 訊息中包含所需的 to_self_delay。如果 Bob 認為可以接受，他的節點在 accept_channel 訊息中包含相同的 to_self_delay 值。如果他們不同意，通道就會被拒絕（參見 shutdown 訊息）。

7.5.7. 撤銷金鑰

正如我們之前討論的，「撤銷」這個詞有點誤導，因為它暗示「已撤銷」的交易不能被使用。

事實上，已撤銷的交易可以被使用，但如果它被使用，而且它已經被撤銷，那麼通道合作夥伴之一可以透過創建懲罰交易來獲取所有通道資金。

運作方式是 `to_local` 輸出不僅被時間鎖定，而且在腳本中還有兩個花費條件：它可以在時間鎖定延遲後由 *self* 花費，*或者_它可以由 _remote* 使用此承諾的撤銷金鑰立即花費。

所以，在我們的例子中，每一方持有的承諾交易在 `to_local` 輸出中都包含一個撤銷選項，如 不對稱、延遲和可撤銷的承諾 所示。

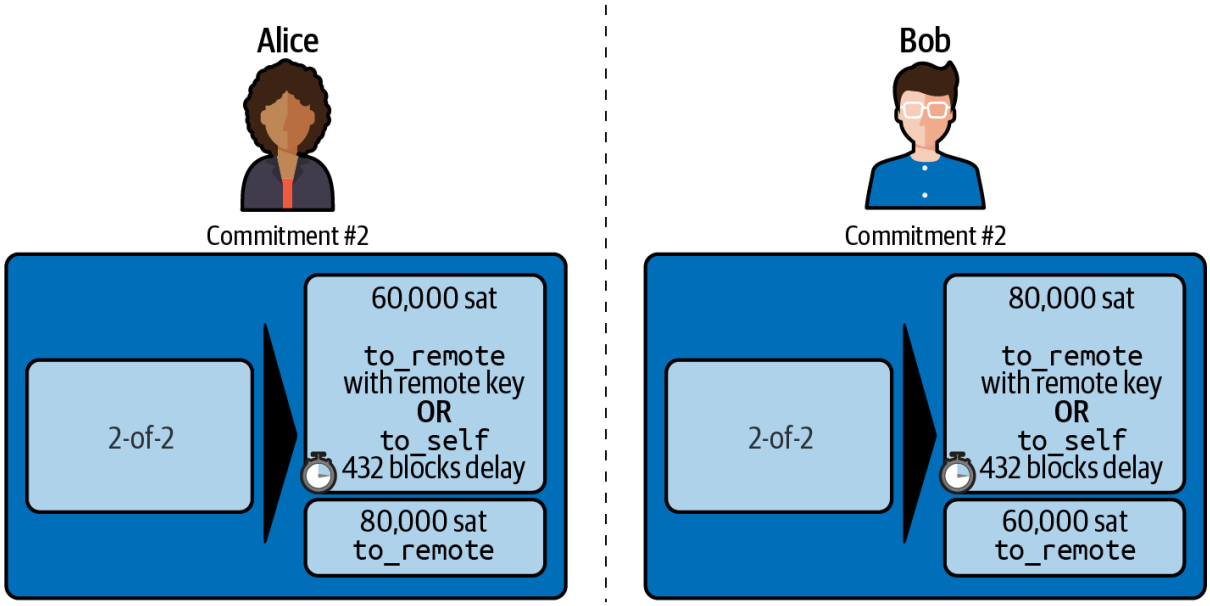

Figure 32. 不對稱、延遲和可撤銷的承諾

7.6. 承諾交易

現在我們理解了承諾交易的結構以及為什麼我們需要不對稱、延遲、可撤銷的承諾，讓我們看看實作這些的比特幣腳本。

承諾交易的第一個（`to_local`）輸出在 BOLT #3：承諾交易，`to_local` 輸出 (https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md#to_local-output) 中定義，如下：

```
OP_IF
    # 懲罰交易
    <revocationpubkey>
OP_ELSE
    <to_self_delay>
    OP_CHECKSEQUENCEVERIFY
    OP_DROP
    <local_delayedpubkey>
OP_ENDIF
OP_CHECKSIG
```

這是一個條件腳本（參見 具有多個條件的腳本），這意味著如果滿足兩個條件中的_任一個_，輸出就可以被花費。第一個子句允許任何能為 <revocationpubkey> 簽名的人花費輸出。第二個子句被 <to_self_delay> 個區塊時間鎖定，只有在那麼多區塊之後才能被任何能為 <local_delayedpubkey> 簽名的人花費。在我們的例子中，我們將 <to_self_delay> 時間鎖設定為 432 個區塊，但這是一個可配置的延遲，由兩個通道合作夥伴協商。to_self_delay 時間鎖持續時間通常根據通道容量成比例選擇，這意味著更大容量的通道（更多資金）有更長的 to_self_delay 時間鎖來保護各方。

第一個子句允許任何能為 <revocationpubkey> 簽名的人花費輸出。此腳本安全性的一個關鍵要求是遠端方_不能_單方面用 `revocationpubkey` 簽名。要了解為什麼這很重要，考慮遠端方違反先前撤銷的承諾的場景。如果他們可以用這個金鑰簽名，那麼他們可以簡單地_自己_獲取撤銷子句並竊取通道中的所有資金。相反，我們根據_自身_（本地）和遠端方的資訊為_每個_狀態衍生 `revocationpubkey`。巧妙地使用對稱和非對稱加密，允許雙方計算 `revocationpubkey` 公鑰，但在給定其秘密資訊的情況下，只允許誠實的自身方計算私鑰，如 撤銷和承諾秘密衍生 中詳述。

撤銷和承諾秘密衍生

每一方在初始通道協商訊息期間發送一個 `revocation_basepoint` 以及一個 `first_per_commitment_point`。`revocation_basepoint` 在通道的生命週期內是靜態的，而每個新的通道狀態將基於新的 `first_per_commitment_point`。

給定這些資訊，每個通道狀態的 `revocationpubkey` 透過以下一系列橢圓曲線和雜湊運算衍生：

```
revocationpubkey = revocation_basepoint * sha256(revocation_basepoint ||
per_commitment_point) + per_commitment_point * sha256(per_commitment_point
|| revocation_basepoint)
```

由於橢圓曲線定義在阿貝爾群上的交換性質，一旦遠端方透露 `per_commitment_secret`（`per_commitment_point` 的私鑰），自身可以用以下操作衍生 `revocationpubkey` 的私鑰：

```
revocation_priv = (revocationbase_priv * sha256(revocation_basepoint ||
per_commitment_point)) + (per_commitment_secret *
sha256(per_commitment_point || revocation_basepoint)) mod N
```

要了解為什麼這在實踐中有效，請注意我們可以_重新排序_（交換）並展開原始 `revocationpubkey` 公式的公鑰計算：

```
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))
```

換句話說，`revocationbase_priv` 只能由知道 `revocationbase_priv` *和* `per_commitment_secret` 的一方衍生（並用於為 `revocationpubkey` 簽名）。這個小技巧是使閃電網路中使用的基於公鑰的撤銷系統安全的關鍵。

> 承諾交易中使用 CHECKSEQUENCEVERIFY 的時間鎖是_相對時間鎖_。它計算從此輸出確認開始經過的區塊數。這意味著在此承諾交易廣播並確認後的 to_self_delay 個區塊_之後_它才能被花費。

承諾交易的第二個輸出（to_remote）在 BOLT #3：承諾交易，`to_remote` 輸出 (https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md#to_remote-output) 中定義，最簡單的形式是 <remote_pubkey> 的 Pay-to-Witness-Public-Key-Hash（P2WPKH），這意味著它只是支付給能為 <remote_pubkey> 簽名的所有者。

現在我們已經詳細定義了承諾交易，讓我們看看 Alice 和 Bob 如何推進通道狀態，創建和簽署新的承諾交易，以及撤銷舊的承諾交易。

7.7. 推進通道狀態

為了推進通道狀態，Alice 和 Bob 交換兩個訊息：commitment_signed 和 revoke_and_ack 訊息。當任一通道合作夥伴有通道狀態更新時，可以發送 commitment_signed 訊息。另一個通道合作夥伴然後可以用 revoke_and_ack 回應以_撤銷_舊承諾並_確認_新承諾。

在 承諾和撤銷訊息流程 中，我們看到 Alice 和 Bob 交換兩對 commitment_signed 和 revoke_and_ack。第一個流程顯示由 Alice 發起的狀態更新（從左到右 commitment_signed），Bob 回應（從右到左 revoke_and_ack）。第二個流程顯示由 Bob 發起並由 Alice 回應的狀態更新。

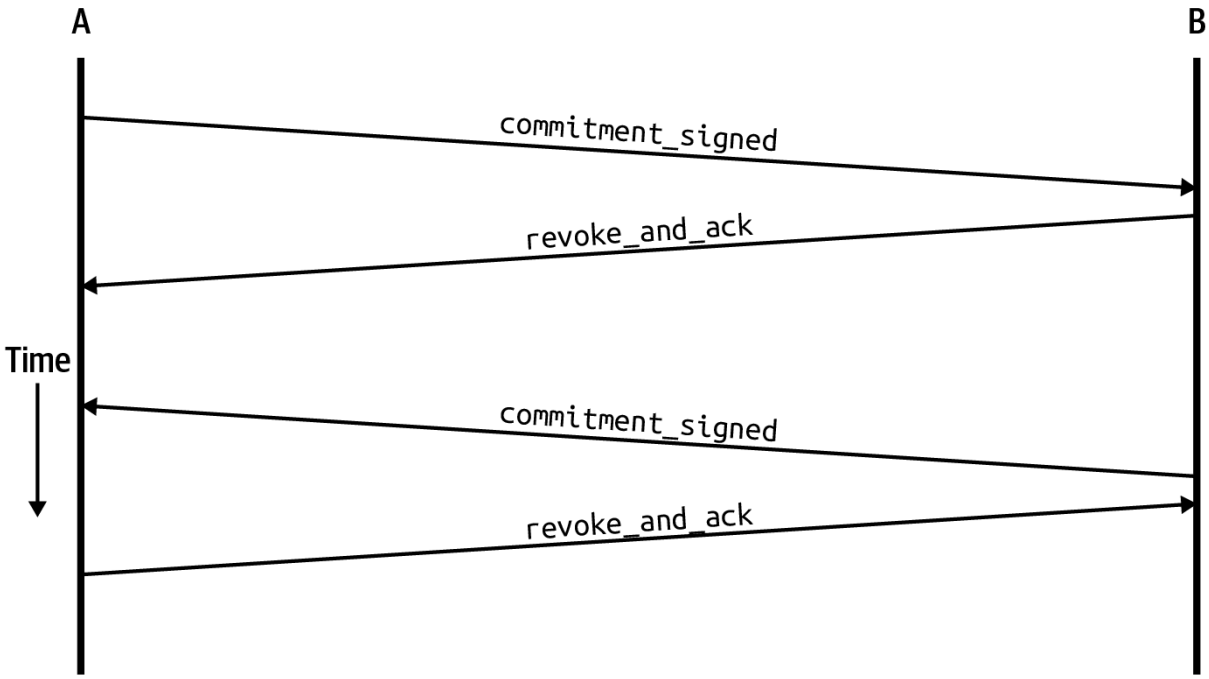

Figure 33. 承諾和撤銷訊息流程

7.7.1. commitment_signed 訊息

commitment_signed 訊息的結構在 BOLT #2：對等協定，`commitment_signed` (https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md#committing-updates-so-far-commitment_signed) 中定義，如下所示：

commitment_signed 訊息

```
[channel_id:channel_id]
[signature:signature]
[u16:num_htlcs]
[num_htlcs*signature:htlc_signature]
```

channel_id

通道的標識符

signature

新遠端承諾的簽名

num_htlcs

此承諾中更新的 HTLC 數量

htlc_signature

更新的簽名

使用 HTLC 提交更新將在 雜湊時間鎖定合約 和 通道操作與支付轉發 中詳細解釋。

Alice 的 commitment_signed 訊息給 Bob 提供了新承諾交易所需的簽名（Alice 在 2-of-2 中的部分）。

7.7.2. revoke_and_ack 訊息

現在 Bob 有了新的承諾交易，他可以透過給 Alice 一個撤銷金鑰來撤銷先前的承諾，並用 Alice 的簽名構建新的承諾。

revoke_and_ack 訊息在 BOLT #2：對等協定，`revoke_and_ack` (https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md#completing-the-transition-to-the-updated-state-revoke_and_ack) 中定義，如下所示：

revoke_and_ack 訊息

```
[channel_id:channel_id]
[32*byte:per_commitment_secret]
[point:next_per_commitment_point]
```

channel_id

這是通道的標識符。

per_commitment_secret

用於為先前（舊的）承諾生成撤銷金鑰，有效地撤銷它。

next_per_commitment_point

用於為新承諾構建 `revocation_pubkey`，以便以後可以撤銷。

7.7.3. 撤銷和重新承諾

讓我們更仔細地看看 Alice 和 Bob 之間的這種互動。

Alice 給 Bob 提供了創建新承諾的方法。作為回報，Bob 撤銷舊承諾以向 Alice 保證他不會使用它。只有當 Alice 有撤銷金鑰來懲罰 Bob 發布舊承諾時，她才能信任新的承諾。從 Bob 的角度來看，他可以安全地撤銷舊承諾，透過給 Alice 懲罰他的金鑰，因為他有新承諾的簽名。

當 Bob 用 revoke_and_ack 回應時，他給 Alice 一個 per_commitment_secret。這個秘密可以用來構建舊承諾的撤銷簽名金鑰，允許 Alice 透過行使懲罰來奪取所有通道資金。

一旦 Bob 將這個秘密給了 Alice，他_絕不能_再廣播那個舊承諾。如果他這樣做，他將給 Alice 機會透過獲取資金來懲罰他。本質上，Bob 給 Alice 讓他對廣播舊承諾負責的能力，實際上他已經撤銷了使用那個舊承諾的能力。

一旦 Alice 從 Bob 收到 revoke_and_ack，她可以確信 Bob 不能在不受懲罰的情況下廣播舊承諾。她現在有必要的金鑰來創建懲罰交易，如果 Bob 廣播舊承諾的話。

7.7.4. 作弊和懲罰的實踐

在實踐中，Alice 和 Bob 都必須監控作弊行為。他們監控比特幣區塊鏈，尋找與他們正在操作的任何通道相關的任何承諾交易。如果他們看到承諾交易在鏈上確認，他們將檢查它是否是最新的承諾。如果它是「舊的」承諾，他們必須立即構建並廣播懲罰交易。懲罰交易花費 to_local 和 to_remote 輸出_兩者_，關閉通道並將兩個餘額都發送給被欺騙的通道合作夥伴。

為了更容易讓雙方跟蹤過去撤銷承諾的承諾號碼，每個承諾實際上在鎖定時間和序列欄位中_編碼_承諾的號碼。在 協定中，這種特殊編碼被稱為_狀態提示_。假設一方知道當前的承諾號碼，他們能夠使用狀態提示輕鬆識別廣播的承諾是否是已撤銷的，如果是，哪個承諾號碼被違反，因為該號碼用於輕鬆查找撤銷秘密樹（shachain）中應該使用哪個撤銷秘密。

不是將狀態提示以明文編碼，而是使用_混淆的_狀態提示。這種混淆是透過首先將當前承諾號碼與使用雙方資金公鑰確定性生成的一組隨機位元組進行 XOR 來實現的。鎖定時間和序列中共有 6 個位元組（鎖定時間的 24 位元和序列的 24 位元）用於在承諾交易中編碼狀態提示，因此需要 6 個隨機位元組用於 XOR。為了獲得這 6 個位元組，雙方獲取發起者資金金鑰與回應者資金金鑰連接的 SHA-256 雜湊。在編碼當前承諾高度之前，整數與此狀態提示混淆器進行 XOR，然後編碼在鎖定時間的低 24 位元和序列的高 64 位元中。

讓我們回顧 Alice 和 Bob 之間的通道，並展示懲罰交易的具體例子。在 已撤銷和當前承諾 中，我們看到 Alice 和 Bob 通道上的四個承諾。Alice 向 Bob 進行了三筆付款：

- 用承諾 #1 支付並承諾給 Bob 70,000 satoshi
- 用承諾 #2 支付並承諾給 Bob 10,000 satoshi
- 用承諾 #3 支付並承諾給 Bob 20,000 satoshi

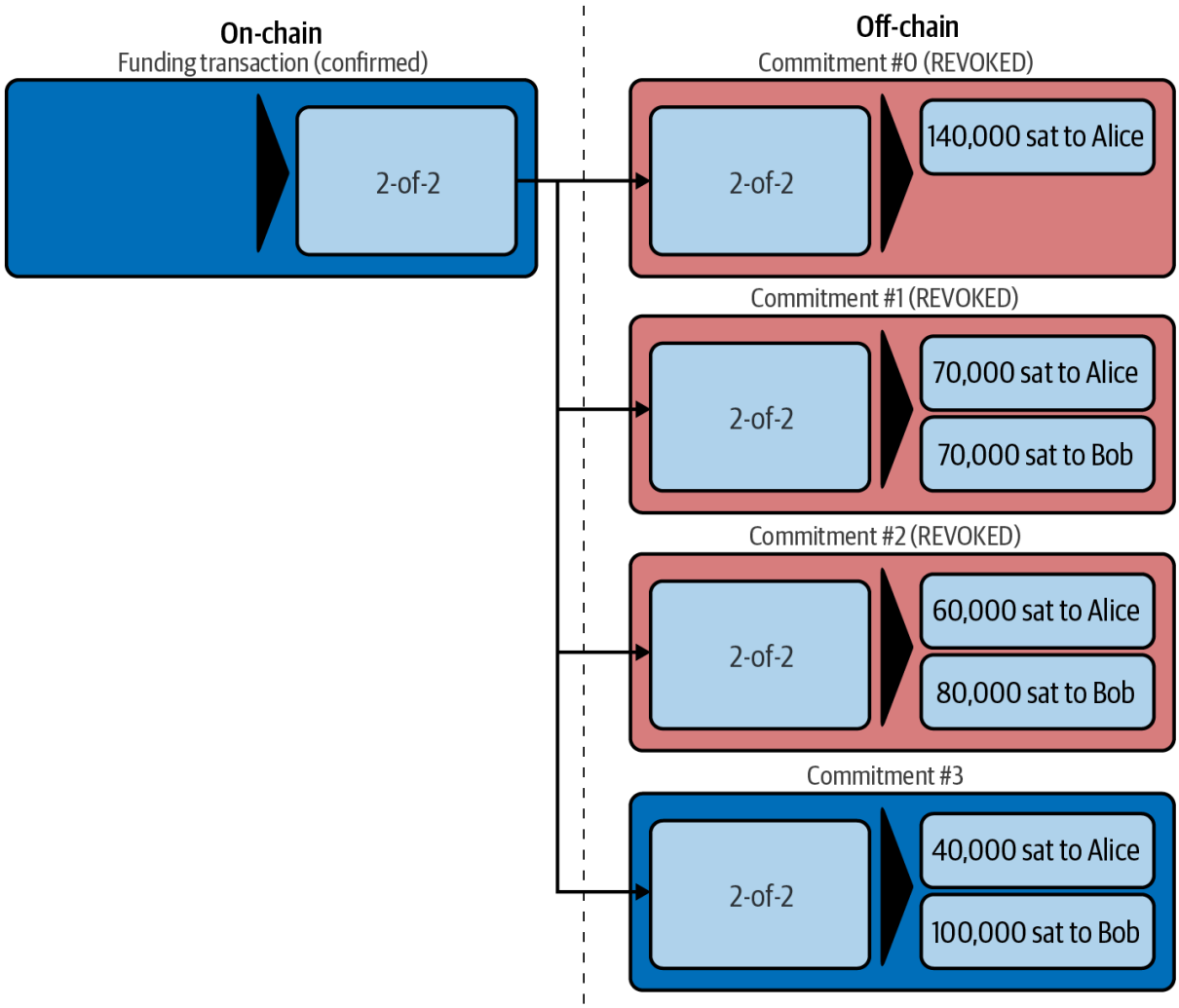

Figure 34. 已撤銷和當前承諾

每次承諾，Alice 都撤銷了先前（較舊的）承諾。通道的當前狀態和正確餘額由承諾 #3 表示。所有先前的承諾都已被撤銷，Bob 有必要的金鑰對它們發出懲罰交易，以防 Alice 試圖廣播其中之一。

Alice 可能有動機作弊，因為所有先前的承諾交易都會給她比她應得的更高比例的通道餘額。例如，假設 Alice 試圖廣播承諾 #1。該承諾交易將支付 Alice 70,000 satoshi 和 Bob 70,000 satoshi。如果 Alice 能夠廣播並花費她的 to_local 輸出，她實際上是透過回滾她對 Bob 的最後兩筆付款來從 Bob 那裡偷走 30,000 satoshi。

Alice 決定冒巨大風險，廣播已撤銷的承諾 #1，從 Bob 那裡偷走 30,000 satoshi。在 Alice 作弊中，我們看到 Alice 的舊承諾，她將其廣播到比特幣區塊鏈。

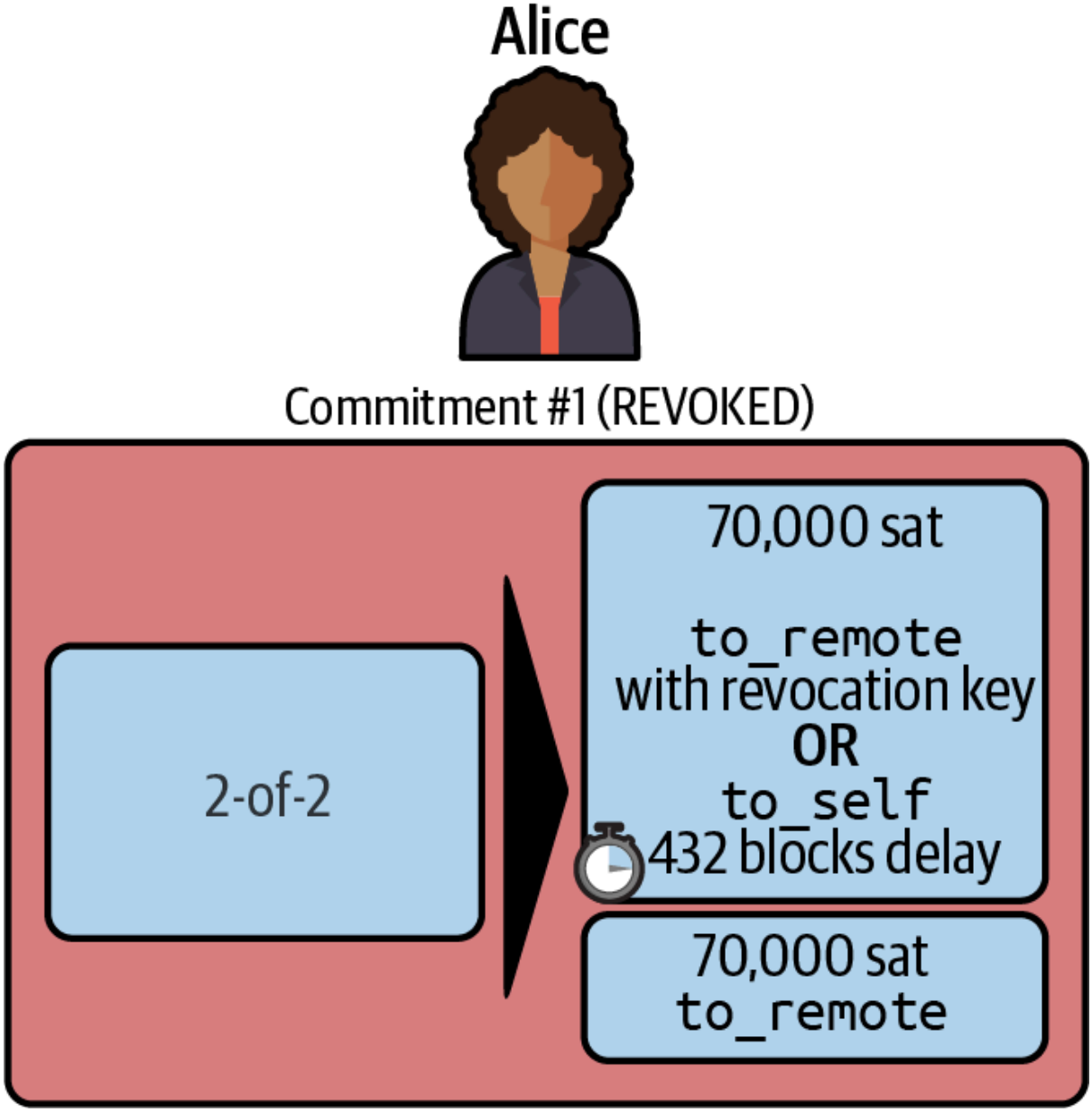

Figure 35. Alice 作弊

如你所見，Alice 的舊承諾有兩個輸出，一個支付給她自己 70,000 satoshi（to_local 輸出），一個支付給 Bob 70,000 satoshi。Alice 還不能花費她的 70,000 to_local 輸出，因為它有 432 個區塊（3 天）的時間鎖。她現在希望 Bob 三天內不會注意到。

不幸的是對 Alice 來說，Bob 的節點正在勤奮地監控比特幣區塊鏈，看到一個舊承諾交易被廣播並（最終）在鏈上確認。

Bob 的節點將立即廣播懲罰交易。由於這個舊承諾被 Alice 撤銷，Bob 有 Alice 發送給他的 per_commitment_secret。他使用那個秘密為 revocation_pubkey 構建簽名。雖然 Alice 必須等待 432 個區塊，Bob 可以立即花費_兩個_輸出。他可以用他的私鑰花費 to_remote 輸出，因為它本來就是要支付給他的。他也可以用撤銷金鑰的簽名花費本來給 Alice 的輸出。他的節點廣播 作弊和懲罰 中顯示的懲罰交易。

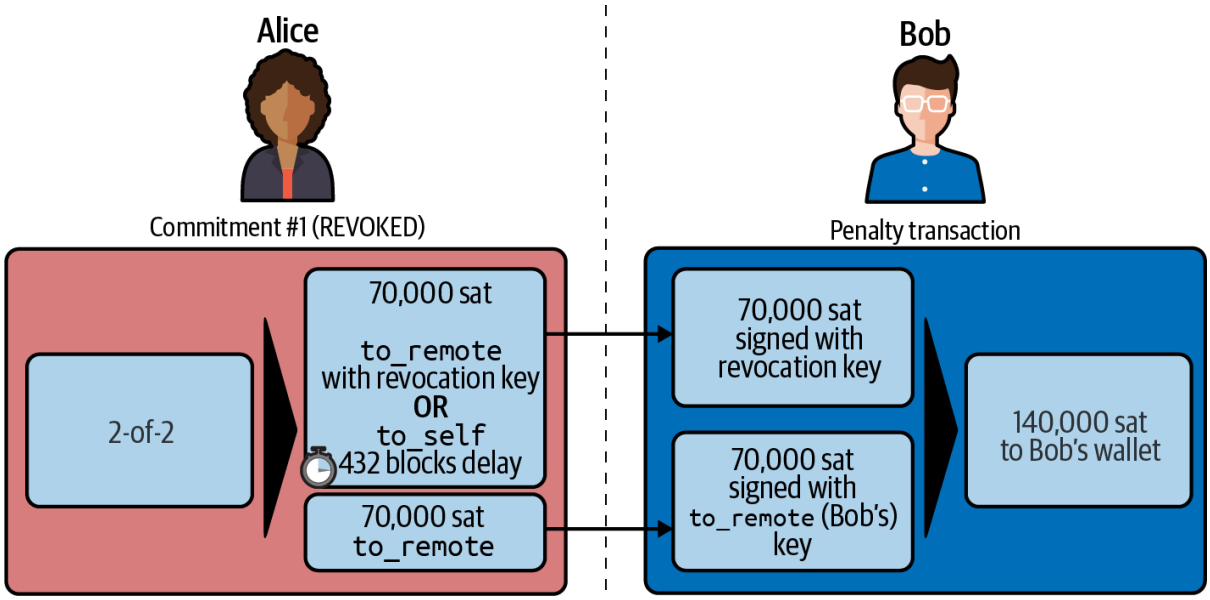

Figure 36. 作弊和懲罰

Bob 的懲罰交易將 140,000 satoshi 支付到他自己的錢包，獲取了整個通道容量。Alice 不僅沒能作弊成功，她還在嘗試中失去了一切！

7.7.5. 通道儲備金：確保利益攸關

你可能注意到有一種特殊情況需要處理。如果 Alice 可以繼續花費她的餘額直到為零，她將處於可以透過廣播舊承諾交易關閉通道而不冒懲罰風險的位置：要麼已撤銷的承諾交易在延遲後成功，要麼作弊者被抓住但沒有後果因為懲罰是零。從博弈論的角度來看，在這種情況下嘗試作弊是免費的錢。這就是為什麼通道儲備金在起作用，所以潛在的作弊者總是面臨懲罰的風險。

7.8. 關閉通道（協作關閉）

到目前為止，我們已經將承諾交易視為關閉通道的一種可能方式，即單方面關閉。這種類型的通道關閉不是理想的，因為它對使用它的通道合作夥伴施加時間鎖。

關閉通道的更好方式是協作關閉。在協作關閉中，兩個 通道合作夥伴協商一個最終的承諾交易，稱為_關閉交易_，立即支付每一方的餘額到他們選擇的目標錢包。然後，發起通道關閉流程的合作夥伴將廣播關閉交易。

關閉訊息流程在 BOLT #2：對等協定，通道關閉 (https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md#channel-close) 中定義，並顯示在 通道關閉訊息流程 中。

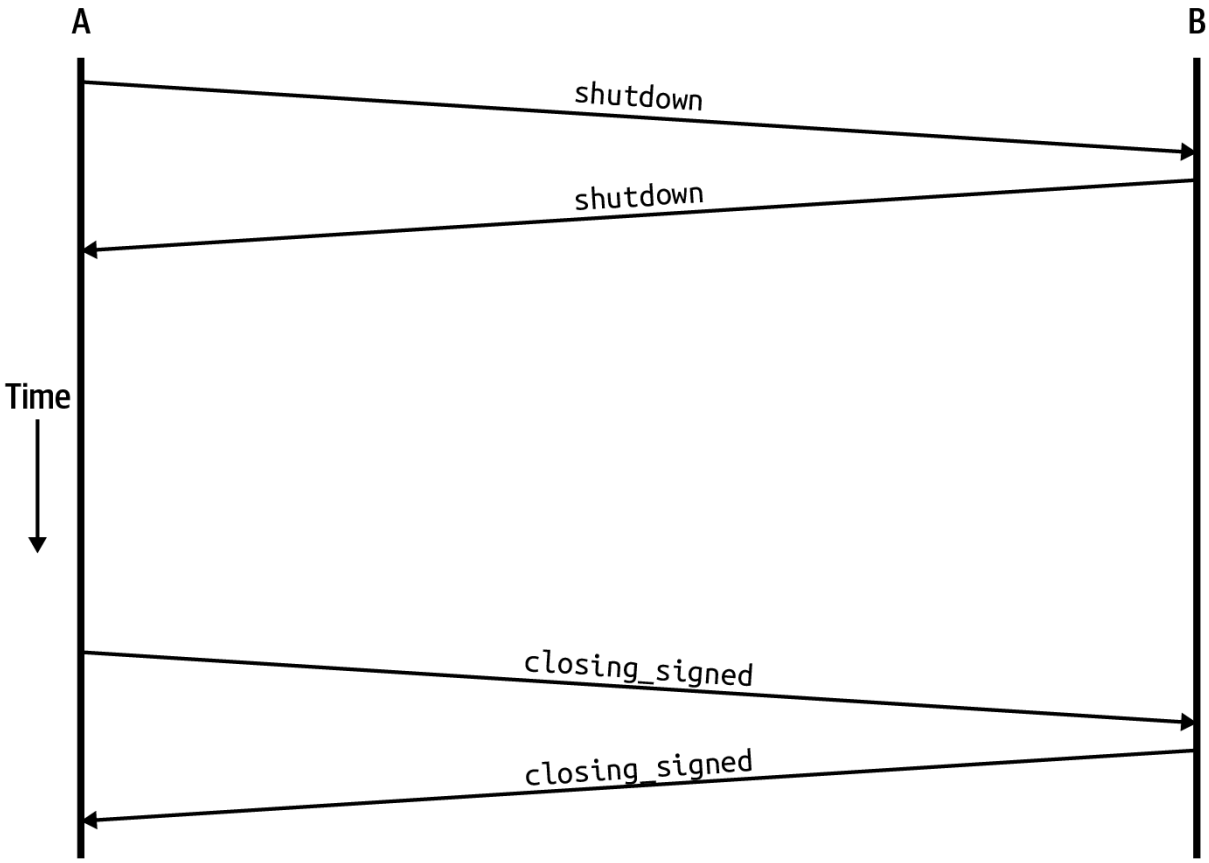

Figure 37. 通道關閉訊息流程

7.8.1. shutdown 訊息

通道關閉從兩個通道合作夥伴之一發送 shutdown 訊息開始。此訊息的內容如下所示：

shutdown 訊息

```
[channel_id:channel_id]
[u16:len]
[len*byte:scriptpubkey]
```

channel_id

我們想要關閉的通道的通道標識符

len

此通道合作夥伴希望接收其餘額的目標錢包腳本長度

scriptpubkey

目標錢包的比特幣腳本，使用「標準」比特幣地址格式之一（P2PKH、P2SH、P2WPKH、P2WSH 等；參見 術語表）

假設 Alice 向 Bob 發送 shutdown 訊息以關閉他們的通道。Alice 將指定一個對應於她錢包比特幣地址的比特幣腳本。她告訴 Bob：讓我們創建一個關閉交易，將我的餘額支付到這個錢包。

Bob 將用他自己的 shutdown 訊息回應，表示他同意協作關閉通道。他的 shutdown 訊息包含他錢包地址的腳本。

現在 Alice 和 Bob 都有對方的首選錢包地址，他們可以構建相同的關閉交易來結算通道餘額。

7.8.2. closing_signed 訊息

假設通道沒有未完成的承諾或更新，並且通道合作夥伴已經交換了前一節中顯示的 shutdown 訊息，他們現在可以完成這個協作關閉。

通道的_資助者_（在我們的例子中是 Alice）首先向 Bob 發送 closing_signed 訊息。此訊息為鏈上交易提議交易費用，以及 Alice 對關閉交易的簽名（2-of-2 多重簽名）。closing_signed 訊息如下所示：

closing_signed 訊息

```
[channel_id:channel_id]
[u64:fee_satoshis]
[signature:signature]
```

channel_id

通道標識符

fee_satoshis

提議的鏈上交易費用，以 satoshi 為單位

signature

發送者對關閉交易的簽名

當 Bob 收到這個訊息時，他可以用自己的 closing_signed 訊息回覆。如果他同意費用，他只需返回相同的提議費用和他自己的簽名。如果他不同意，他必須提議不同的 fee_satoshis 費用。

這種協商可能會繼續進行來回的 closing_signed 訊息，直到兩個通道合作夥伴同意費用。

一旦 Alice 收到一個 closing_signed 訊息，其費用與她在上一條訊息中提議的相同，協商就完成了。Alice 簽署並廣播關閉交易，通道就關閉了。

7.8.3. 協作關閉交易

協作關閉交易看起來類似於 Alice 和 Bob 商定的最後一個承諾交易。然而，與最後一個承諾交易不同，它的輸出中沒有時間鎖或懲罰撤銷金鑰。由於雙方合作產生此交易並且他們不會進行任何進一步的承諾，因此此交易中不需要不對稱、延遲和可撤銷的元素。

通常，此協作關閉交易中使用的地址是為每個被關閉的通道新生成的。然而，雙方也可以_鎖定_一個「交付」地址，用於發送他們協作結算的資金。在 `open_channel` 和 `accept_channel` 訊息的 TLV 命名空間內，雙方可以自由指定「預先關閉腳本」。通常，此地址從駐留在冷儲存中的金鑰衍生。這種做法有助於增加通道的安全性：如果通道合作夥伴以某種方式被入侵，那麼駭客就無法使用他們控制的地址協作關閉通道。相反，如果未使用指定的預先關閉地址，未受損的誠實通道合作夥伴將拒絕在通道關閉上合作。此功能有效地創建了一個「閉環」，限制資金從給定通道流出。

Alice 廣播 協作關閉交易 中顯示的交易來關閉通道。

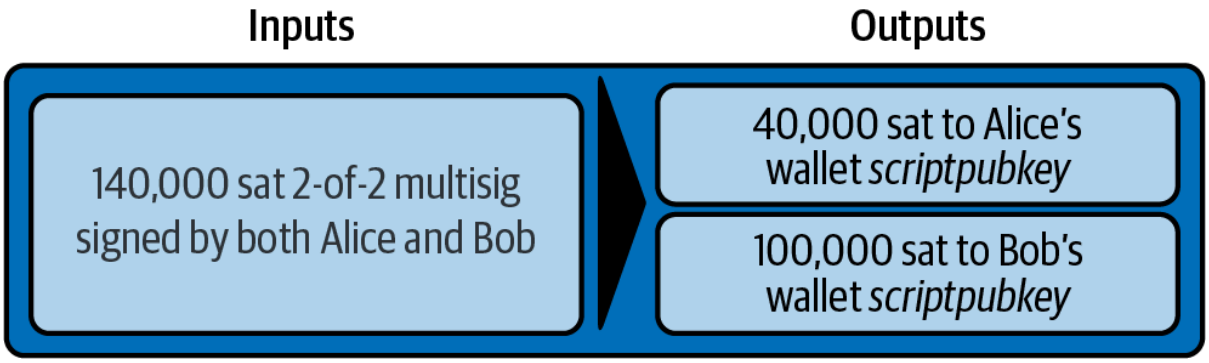

Figure 38. 協作關閉交易

一旦這個關閉交易在比特幣區塊鏈上確認，通道就關閉了。現在，Alice 和 Bob 可以隨意花費他們的輸出。

7.9. 結論

在本節中，我們更詳細地研究了支付通道。我們檢視了 Alice 和 Bob 用於協商注資、承諾和關閉通道的三個訊息流程。我們還展示了資金、承諾和關閉交易的結構，並研究了撤銷和懲罰機制。

正如我們將在接下來的幾章中看到的，HTLC 甚至用於通道合作夥伴之間的本地付款。它們不是必需的，但如果本地（一個通道）和路由（多個通道）付款以相同的方式進行，協定會簡單得多。

在單個支付通道中，每秒付款的數量僅受 Alice 和 Bob 之間網路容量的限制。只要通道合作夥伴能夠來回發送幾個位元組的資料以同意新的通道餘額，他們就有效地進行了一筆付款。這就是為什麼我們可以在閃電網路（鏈下）上實現比比特幣區塊鏈（鏈上）可以處理的交易吞吐量大得多的付款吞吐量。

在接下來的幾章中，我們將討論路由、HTLC 及其在通道操作中的使用。

8. 在支付通道網路上路由

在本章中，我們將最終解析如何透過稱為_路由_的過程將支付通道連接起來形成支付通道網路。具體來說，我們將研究路由層的第一部分，即「原子性和無需信任的多跳合約」協定。這在協定套件中以輪廓突顯，如 閃電網路協定套件中的原子支付路由 所示。

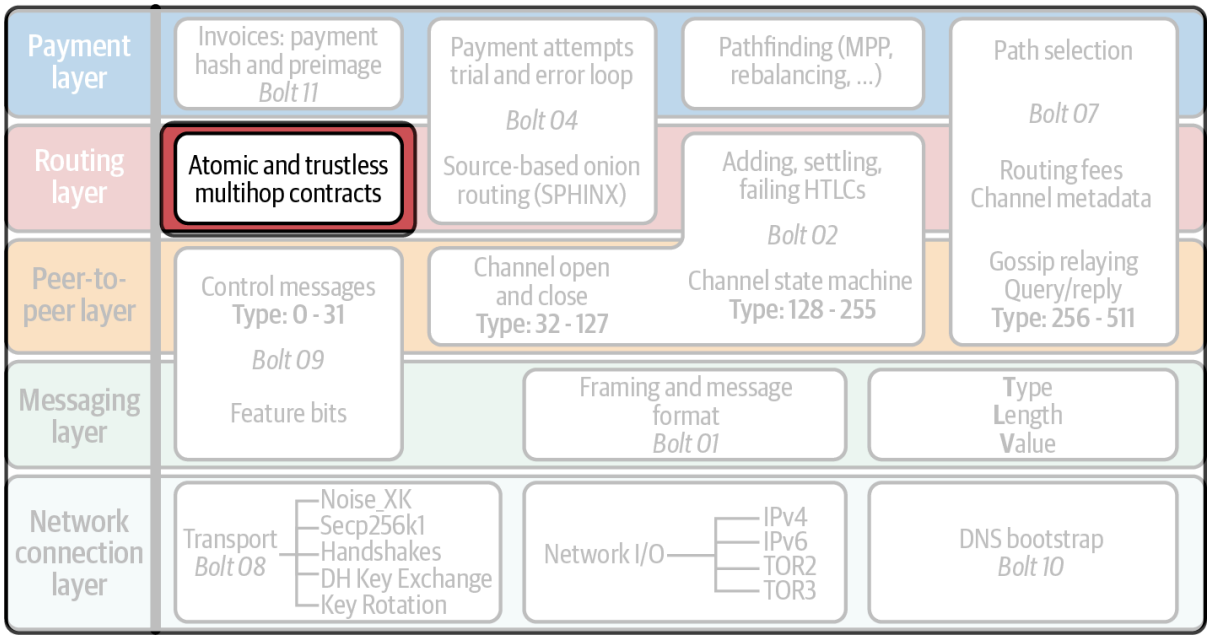

Figure 39. 閃電網路協定套件中的原子支付路由

8.1. 路由付款

在本節中，我們將從 Dina 的角度來研究路由，她是一位在直播遊戲時接收粉絲打賞的玩家。

路由支付通道的創新使 Dina 能夠接收打賞，而無需與每位想要打賞她的粉絲維護一個單獨的通道。 只要從該觀眾到 Dina 存在一條資金充足的通道路徑，她就能夠收到該粉絲的付款。

在 粉絲在閃電網路上（間接）連接到 Dina 中，我們看到由各個閃電網路節點之間的支付通道建立的可能網路佈局。圖中的每個人都可以透過建構路徑向 Dina 發送付款。想像一下粉絲 4 想要向 Dina 發送付款。你能看到可以實現這一點的路徑嗎？粉絲 4 可以透過粉絲 3、Bob 和 Chan 向 Dina 路由付款。同樣地，Alice 可以透過 Bob 和 Chan 向 Dina 路由付款。

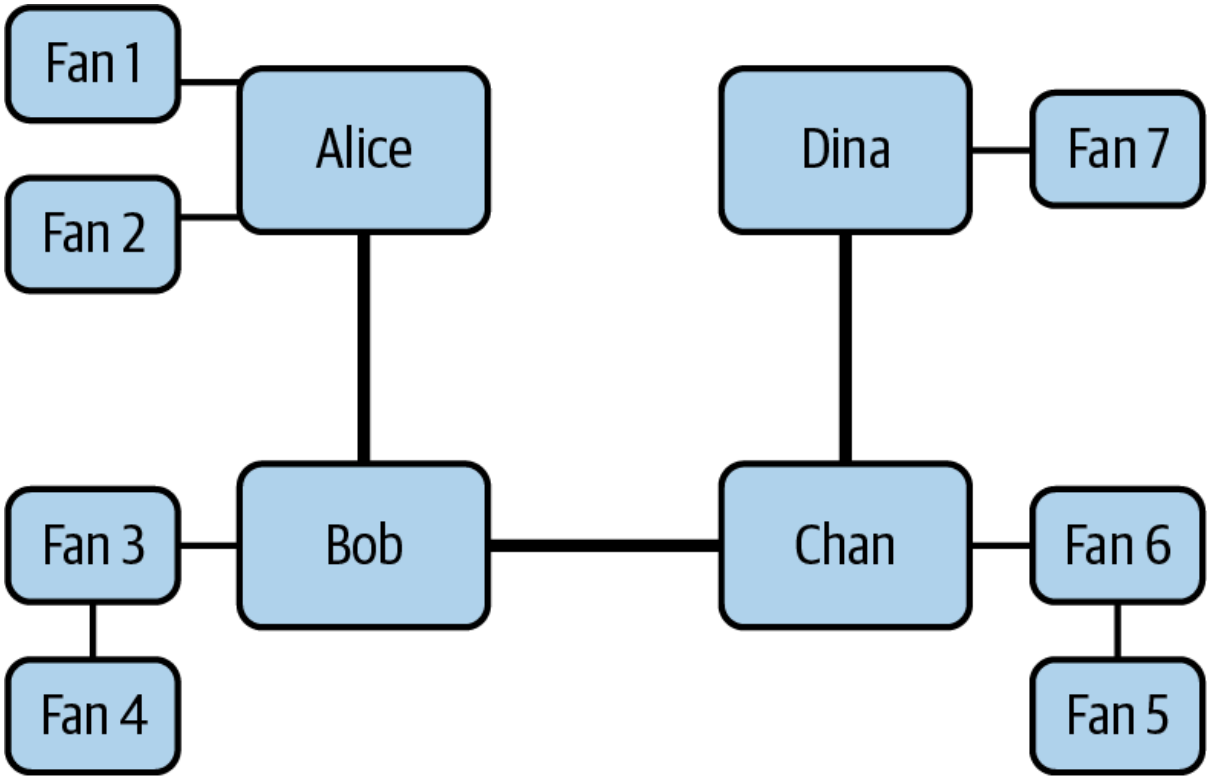

Figure 40. 粉絲在閃電網路上（間接）連接到 Dina

從粉絲到 Dina 路徑上的節點是中間節點，在路由付款的背景下稱為_路由節點_。路由節點與 Dina 粉絲操作的節點之間沒有功能差異。任何閃電網路節點都能夠透過其支付通道路由付款。

重要的是，路由節點在路由從粉絲到 Dina 的付款時無法竊取資金。 此外，路由節點在參與路由過程時不會損失金錢。 路由節點可以因作為中間人而收取路由費用，儘管他們不必這樣做，可以選擇免費路由付款。

另一個重要的細節是，由於使用洋蔥路由，中間節點只明確知道路由中前一個節點和後一個節點。 它們不一定知道誰是付款的發送者和接收者。 這使粉絲能夠使用中間節點向 Dina 付款，而不會洩露私人資訊，也不會有被盜的風險。

這種連接一系列支付通道並具有端對端安全性的過程，以及節點_轉發_付款的激勵結構，是閃電網路的關鍵創新之一。

在本章中，我們將深入探討閃電網路中路由的機制，詳細說明付款在網路中流動的精確方式。首先，我們將釐清路由的概念，並將其與路徑尋找進行比較，因為這些經常被混淆並互換使用。接下來，我們將建構公平協定：一種用於路由付款的原子性、無需信任的多跳協定。為了展示這個公平協定如何運作，我們將使用在四個人之間轉移金幣的物理等效例子。最後，我們將研究目前在閃電網路中使用的原子性、無需信任的多跳協定實作，稱為雜湊時間鎖定合約（HTLC）。

8.2. 路由與路徑尋找

重要的是要注意，我們將_路由_的概念與_路徑尋找_的概念分開。這兩個概念經常被混淆，_路由_這個術語經常被用來描述這兩個概念。讓我們在繼續之前消除歧義。

路徑尋找將在 路徑尋找與付款傳遞 中討論，它是尋找和選擇由支付通道組成的連續路徑的過程，該路徑將發送者 A 連接到接收者 B。付款的發送者透過檢查從其他節點八卦獲得的通道公告組裝的_通道圖_來進行路徑尋找。

路由指的是嘗試將付款從某點 A 轉發到另一點 B 的網路互動系列，沿著路徑尋找先前選擇的路徑。路由是在路徑上發送付款的主動過程，需要沿該路徑的所有中間節點的合作。

一個重要的經驗法則是，Alice 和 Bob 之間可能存在一條_路徑_（甚至可能不止一條），但可能不存在可以發送付款的活躍_路由_。一個例子是所有連接 Alice 和 Bob 的節點目前都離線的情況。在這個例子中，可以檢查通道圖並從 Alice 到 Bob 連接一系列支付通道，因此存在一條_路徑_。然而，由於中間節點離線，付款無法發送，因此不存在_路由_。

8.3. 建立支付通道網路

在我們深入研究原子無需信任多跳付款的概念之前，讓我們先透過一個例子來說明。 讓我們回到 Alice，在前幾章中，她從 Bob 那裡購買了一杯咖啡，她與 Bob 有一個開放的通道。 現在 Alice 正在觀看玩家 Dina 的直播，想透過閃電網路向 Dina 發送 50,000 聰的打賞。但 Alice 與 Dina 沒有直接通道。Alice 該怎麼辦？

Alice 可以與 Dina 開設一個直接通道；然而，這需要流動性和鏈上費用，可能比打賞本身的價值還要高。相反，Alice 可以使用她現有的開放通道向 Dina 發送打賞，_而不需要_直接與 Dina 開設通道。只要從 Alice 到 Dina 存在某些具有足夠容量的通道路徑，這就是可能的。

如你在 Alice 和 Dina 之間的支付通道網路 中所見，Alice 與咖啡店老闆 Bob 有一個開放的通道。Bob 又與軟體開發者 Chan 有一個開放的通道，Chan 幫助他處理咖啡店使用的銷售點系統。Chan 也是一家大型軟體公司的老闆，該公司開發了 Dina 玩的遊戲，他們已經有一個開放的通道，Dina 用它來支付遊戲授權和遊戲內物品。

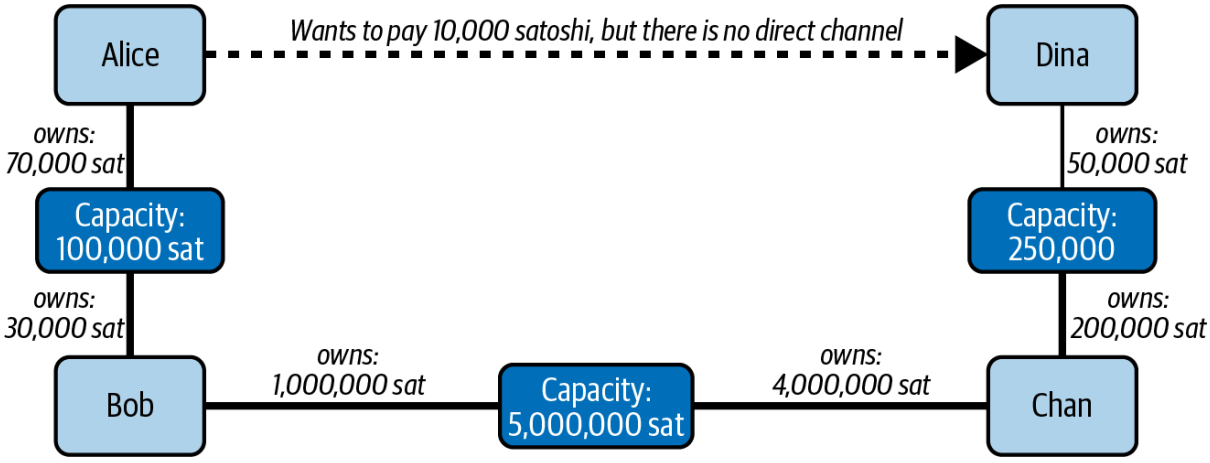

Figure 41. Alice 和 Dina 之間的支付通道網路

可以追蹤一條從 Alice 到 Dina 的_路徑_，使用 Bob 和 Chan 作為中間路由節點。 然後 Alice 可以從這條規劃的路徑製作一條_路由_，並用它向 Dina 發送幾千聰的打賞，付款由 Bob 和 Chan *轉發*。 本質上，Alice 將付款給 Bob，Bob 將付款給 Chan，Chan 將付款給 Dina。不需要從 Alice 到 Dina 的直接通道。

主要挑戰是以一種防止 Bob 和 Chan 竊取 Alice 想要交付給 Dina 的錢的方式來做到這一點。

8.4. 「路由」的物理範例

為了理解閃電網路如何在路由時保護付款，我們可以將其與現實世界中使用金幣路由物理付款的範例進行比較。

假設 Alice 想給 Dina 10 枚金幣，但沒有直接接觸 Dina 的途徑。然而，Alice 認識 Bob，Bob 認識 Chan，Chan 認識 Dina，所以她決定請 Bob 和 Chan 幫忙。這在 Alice 想付給 Dina 10 枚金幣 中顯示。

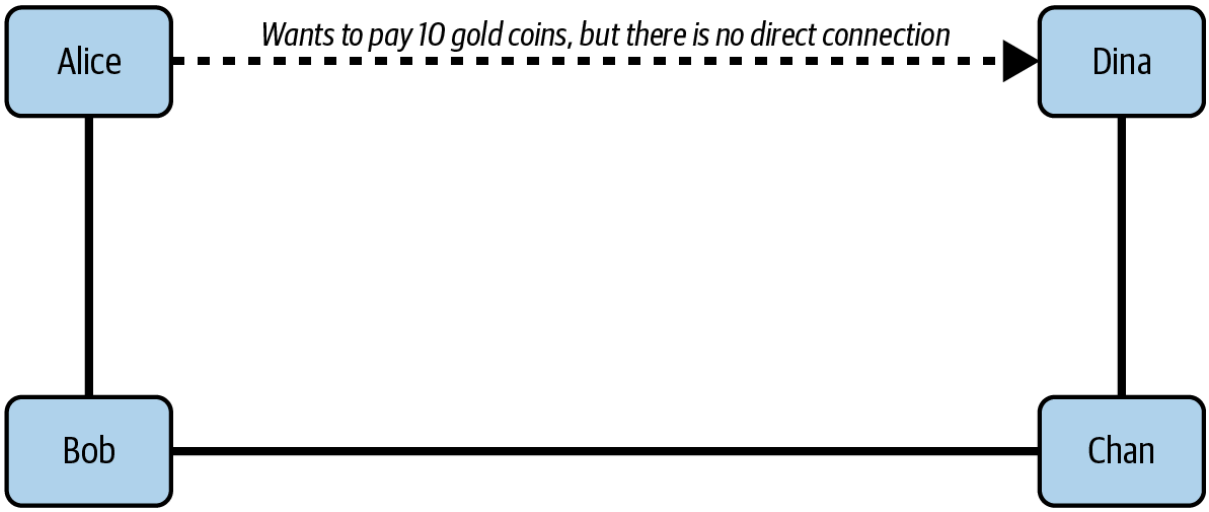

Figure 42. Alice 想付給 Dina 10 枚金幣

Alice 可以付錢給 Bob 讓他付給 Chan 再付給 Dina，但她如何確保 Bob 或 Chan 在收到金幣後不會帶著金幣跑掉？ 在物理世界中，可以使用合約來安全地進行一系列付款。

Alice 可以與 Bob 協商一份合約，內容如下：

> *“我，Alice，如果你把它們傳給 Chan，我將給你 Bob 10 枚金幣。*

雖然這份合約在抽象概念上很好，但在現實世界中，Alice 面臨 Bob 可能違反合約並希望不被發現的風險。 即使 Bob 被抓到並被起訴，Alice 也面臨他可能破產無法歸還她 10 枚金幣的風險。 假設這些問題被神奇地解決了，仍然不清楚如何利用這樣的合約來實現我們想要的結果：將金幣送到 Dina 手中。

讓我們改進我們的合約以納入這些考量：

> *“我，Alice，如果你能向我證明（例如，透過收據）你已經將 10 枚金幣交給 Chan，我將償還你 Bob 10 枚金幣。*

你可能會問自己為什麼 Bob 要簽署這樣的合約。 他必須付錢給 Chan，但最終從交換中什麼也得不到，而且他面臨 Alice 可能不償還他的風險。Bob 可以向 Chan 提供類似的合約來付款給 Dina，但同樣 Chan 也沒有理由接受。

即使撇開風險不談，Bob 和 Chan _必須已經_有 10 枚金幣可以發送；否則，他們將無法參與合約。

因此，Bob 和 Chan 在同意這份合約時面臨風險和機會成本，他們需要得到補償才會接受。

然後 Alice 可以透過提供每人一枚金幣的費用來使這對 Bob 和 Chan 都有吸引力，如果他們將她的付款傳送給 Dina。

合約將變成：

> *“我，Alice，如果你能向我證明（例如，透過收據）你已經將 11 枚金幣交給 Chan，我將償還你 Bob 12 枚金幣。*

Alice 現在承諾給 Bob 12 枚金幣。有 10 枚要交給 Dina，2 枚用於費用。如果他能證明他已轉發 11 枚給 Chan，她承諾給他 12 枚。 一枚金幣的差額是 Bob 幫助這筆特定付款所賺取的費用。在 Alice 付給 Bob，Bob 付給 Chan，Chan 付給 Dina 中，我們看到這種安排如何透過 Bob 和 Chan 將 10 枚金幣送到 Dina 手中。

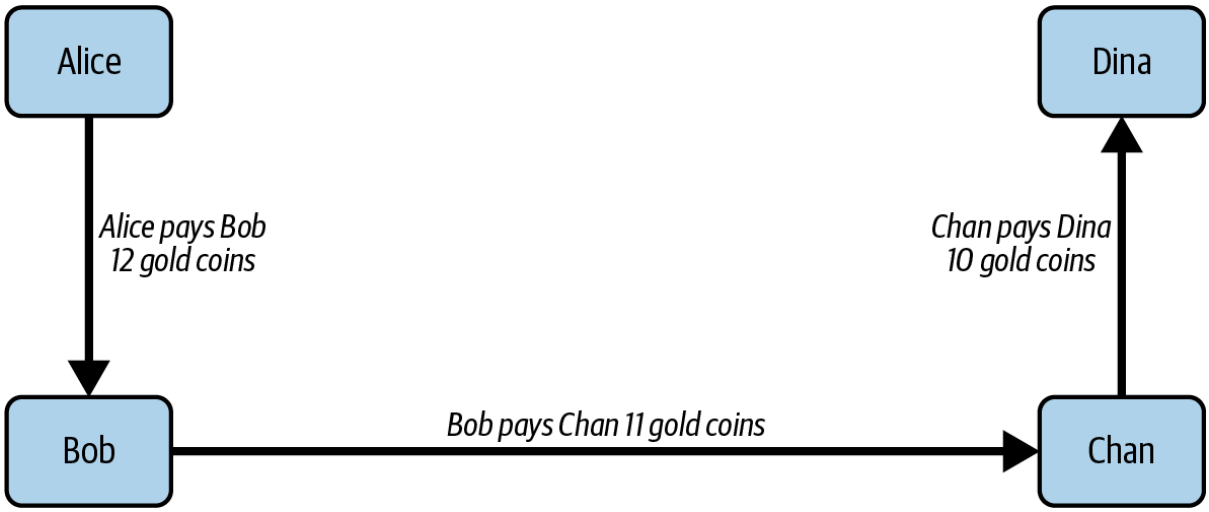

Figure 43. Alice 付給 Bob，Bob 付給 Chan，Chan 付給 Dina

因為仍然存在信任問題和 Alice 或 Bob 不履行合約的風險，所有各方決定使用第三方託管服務。 在交換開始時，Alice 可以將這 12 枚金幣「鎖定」在託管中，只有在 Bob 證明他已付給 Chan 11 枚金幣後才會支付給 Bob。

這個託管服務是理想化的，不會引入其他風險（例如，交易對手風險）。稍後我們將看到如何用比特幣智慧合約取代託管。現在假設每個人都信任這個託管服務。

在閃電網路中，收據（付款證明）可以採用只有 Dina 知道的秘密的形式。 實際上，這個秘密將是一個大到足以防止他人猜測的隨機數（通常是一個_非常非常_大的數字，使用 256 位元編碼！）。

Dina 從隨機數產生器生成這個秘密值 R。

然後可以透過在合約本身中包含秘密的 SHA-256 雜湊來將秘密承諾到合約中，如下所示：

- *H* = SHA-256(*R*)

我們將付款秘密的雜湊稱為_付款雜湊_。 「解鎖」付款的秘密稱為_付款秘密_。

現在，我們保持簡單，假設 Dina 的秘密只是文字行：`Dinas secret`。這個秘密訊息稱為_付款秘密_或_付款原像_。

為了「承諾」這個秘密，Dina 計算 SHA-256 雜湊，以十六進位編碼時，可以顯示如下：

```
0575965b3b44be51e8057d551c4016d83cb1fba9ea8d6e986447ba33fe69f6b3
```

為了促進 Alice 的付款，Dina 將建立付款秘密和付款雜湊，並將付款雜湊發送給 Alice。在 Dina 向 Alice 發送雜湊後的秘密 中，我們看到 Dina 透過某些外部通道（虛線）向 Alice 發送付款雜湊，例如電子郵件或簡訊。

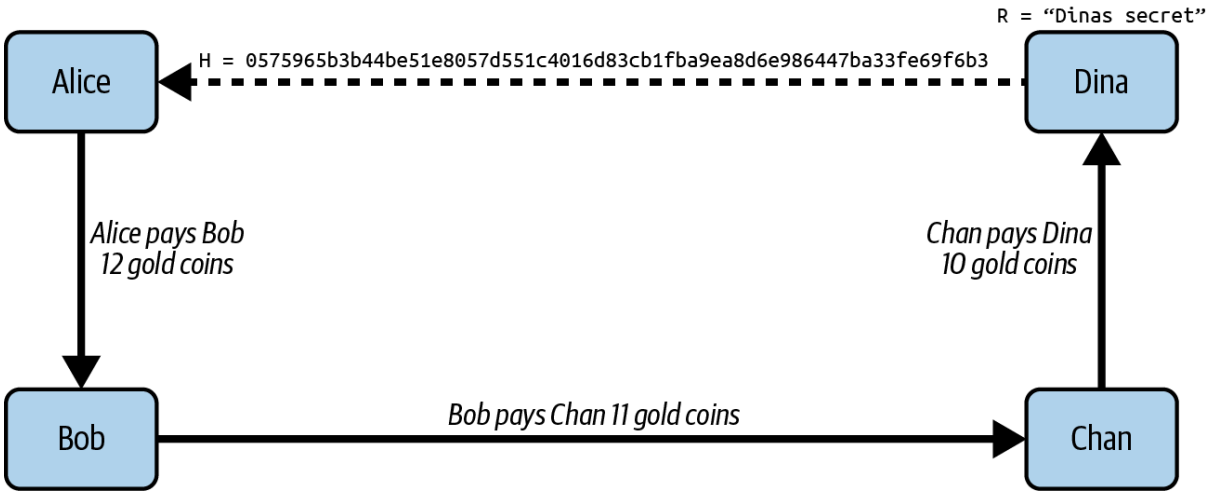

Figure 44. Dina 向 Alice 發送雜湊後的秘密

Alice 不知道秘密，但她可以重寫她的合約，使用秘密的雜湊作為付款證明：

> *“我，Alice，如果你能向我出示一個有效訊息，其雜湊為：`057596`…，我將償還你 Bob 12 枚金幣。 你可以透過與 Chan 建立類似的合約來獲取此訊息，Chan 必須與 Dina 建立類似的合約。 為了向你保證你會得到償還，在你建立下一份合約之前，我會將 12 枚金幣提供給可信的託管方。*

這份新合約現在保護 Alice 免受 Bob 不轉發給 Chan 的影響，保護 Bob 免受 Alice 不償還的影響，並確保透過 Dina 秘密的雜湊有證據證明 Dina 最終收到了付款。

在 Bob 和 Alice 同意合約後，Bob 收到託管方的訊息說 Alice 已存入 12 枚金幣，Bob 現在可以與 Chan 協商類似的合約。

注意，由於 Bob 收取 1 枚金幣的服務費，一旦 Chan 出示他已付款給 Dina 的證明，他只會轉發 11 枚金幣給 Chan。 同樣，Chan 也會要求收費，一旦他證明他已付給 Dina 承諾的 10 枚金幣，他將期望收到 11 枚金幣。

Bob 與 Chan 的合約將是：

> *我，Bob，如果你能向我出示一個有效訊息，其雜湊為：* `057596` *….，我將償還你 Chan 11 枚金幣。 你可以透過與 Dina 建立類似的合約來獲取此訊息。 為了向你保證你會得到償還，在你建立下一份合約之前，我會將 11 枚金幣提供給可信的託管方。*

一旦 Chan 收到託管方的訊息說 Bob 已存入 11 枚金幣，Chan 與 Dina 建立類似的合約：

> *我，Chan，如果你能向我出示一個有效訊息，其雜湊為：* `057596` *….，我將償還你 Dina 10 枚金幣。 為了向你保證在揭示秘密後你會得到償還，我會將 10 枚金幣提供給可信的託管方。*

一切現在就緒。 Alice 與 Bob 有合約，並已將 12 枚金幣放入託管。 Bob 與 Chan 有合約，並已將 11 枚金幣放入託管。 Chan 與 Dina 有合約，並已將 10 枚金幣放入託管。 現在由 Dina 來揭示秘密，這是她建立作為付款證明的雜湊的原像。

Dina 現在向 Chan 發送 Dinas secret。

Chan 檢查 Dinas secret 的雜湊是否為 057596….。Chan 現在有了付款證明，因此指示託管服務將 10 枚金幣釋放給 Dina。

Chan 現在將秘密提供給 Bob。Bob 檢查它並指示託管服務將 11 枚金幣釋放給 Chan。

Bob 現在將秘密提供給 Alice。 Alice 檢查它並指示託管方將 12 枚金幣釋放給 Bob。

所有合約現在都已結算。 Alice 總共支付了 12 枚金幣，其中 1 枚由 Bob 收到，1 枚由 Chan 收到，10 枚由 Dina 收到。 有了這樣的合約鏈，Bob 和 Chan 不能帶著錢跑掉，因為他們先把錢存入了託管。

然而，還有一個問題。 如果 Dina 拒絕釋放她的秘密原像，那麼 Chan、Bob 和 Alice 都會將他們的金幣困在託管中而無法得到償還。 同樣，如果鏈中的其他任何人未能傳遞秘密，同樣的情況會發生。 所以雖然沒有人能從 Alice 那裡偷錢，但每個人的錢仍然會永久困在託管中。

幸運的是，這可以透過在合約中添加截止日期來解決。

我們可以修改合約，使得如果在某個截止日期之前未履行，合約就會過期，託管服務將錢退還給最初存款的人。 我們稱這個截止日期為_時間鎖_。

存款在託管服務中鎖定一定時間，即使沒有提供付款證明，最終也會被釋放。

為了考慮這一點，Alice 和 Bob 之間的合約再次修改，增加了新條款：

> *Bob 在合約簽署後有 24 小時出示秘密。 如果 Bob 在此時間前沒有提供秘密，Alice 的存款將由託管服務退還，合約變為無效。*

當然，Bob 現在必須確保在 24 小時內收到付款證明。 即使他成功付款給 Chan，如果他在 24 小時後才收到付款證明，他將不會得到償還。為了消除這種風險，Bob 必須給 Chan 一個更短的截止日期。

反過來，Bob 將修改他與 Chan 的合約如下：

> *Chan 在合約簽署後有 22 小時出示秘密。 如果他在此時間前沒有提供秘密，Bob 的存款將由託管服務退還，合約變為無效。*

你可能已經猜到，Chan 也會修改他與 Dina 的合約：

> *Dina 在合約簽署後有 20 小時出示秘密。 如果她在此時間前沒有提供秘密，Chan 的存款將由託管服務退還，合約變為無效。*

有了這樣的合約鏈，我們可以確保在 24 小時後，付款將成功地從 Alice 到 Bob 到 Chan 到 Dina，或者它將失敗，每個人都將得到退款。 合約要麼失敗要麼成功，沒有中間地帶。

在閃電網路的背景下，我們稱這種「全有或全無」的屬性為_原子性_。

只要託管是值得信賴的並忠實地履行其職責，任何一方都不會在過程中被盜走金幣。

這條_路由_能夠工作的前提是路徑中的所有各方都有足夠的錢來滿足所需的一系列存款。

雖然這似乎是一個小細節，但我們稍後將在本章中看到，這個要求實際上是閃電網路節點面臨的更困難的問題之一。 隨著付款金額的增加，這變得越來越困難。 此外，當資金被鎖定在託管中時，各方無法使用他們的錢。

因此，轉發付款的使用者面臨鎖定資金的機會成本，這最終透過路由費用得到補償，正如我們在前面的例子中看到的。

現在我們已經看到了一個物理付款路由範例，我們將看看如何在比特幣區塊鏈上實現這一點，而不需要任何第三方託管。為此，我們將使用比特幣腳本在參與者之間建立合約。我們用實作公平協定的_智慧合約_取代第三方託管。讓我們分解這個概念並實現它！

8.5. 公平協定

正如我們在本書第一章中看到的，比特幣的創新是能夠使用密碼學原語來實作公平協定，用可信的協定替代對第三方（中間人）的信任。

在我們的金幣範例中，我們需要一個託管服務來防止任何一方違背其義務。密碼學公平協定的創新使我們能夠用協定取代託管服務。

我們想要建立的公平協定的屬性是：

無需信任的操作

路由付款的參與者不需要相互信任，也不需要信任任何中間人或第三方。相反，他們信任協定來保護他們免受欺騙。

原子性

付款要麼完全執行，要麼失敗並且每個人都得到退款。中間人不可能收取路由付款而不將其轉發到下一跳。因此，中間人不能欺騙或偷竊。

多跳

系統的安全性端對端地擴展到透過多個支付通道路由的付款，就像單個支付通道兩端之間的付款一樣。

一個可選的附加屬性是能夠將付款分成多個部分，同時保持整個付款的原子性。這些被稱為_多部分付款_（*MPP*），將在 多部分付款 中進一步探討。

8.5.1. 實作原子無需信任多跳付款

比特幣腳本足夠靈活，有數十種方法可以實作具有原子性、無需信任操作和多跳安全性屬性的公平協定。選擇特定的實作取決於隱私、效率和複雜性之間的某些權衡。

目前在閃電網路中用於路由的公平協定稱為雜湊時間鎖定合約（HTLC）。HTLC 使用雜湊原像作為解鎖付款的秘密，正如我們在本章的金幣範例中看到的。付款的接收者生成一個隨機秘密數字並計算其雜湊。雜湊成為付款的條件，一旦秘密被揭示，所有參與者都可以兌換他們收到的付款。HTLC 提供原子性、無需信任操作和多跳安全性。

另一種提議的路由實作機制是_點時間鎖定合約_（*PTLC*）。PTLC 也實現原子性、無需信任操作和多跳安全性，但具有更高的效率和更好的隱私。PTLC 的高效實作依賴於一種名為_Schnorr 簽名_的新數位簽章演算法，預計將於 2021 年在比特幣中啟用。

8.6. 重新審視打賞範例

讓我們重新審視本章第一部分的範例。Alice 想用閃電網路付款向 Dina 打賞。假設 Alice 想向 Dina 發送 50,000 聰作為打賞。

為了讓 Alice 付款給 Dina，Alice 需要 Dina 的節點生成一張閃電網路發票。我們將在 閃電網路付款請求 中更詳細地討論這一點。現在，讓我們假設 Dina 有一個網站，可以為打賞生成閃電網路發票。

閃電網路付款可以在沒有發票的情況下使用一種名為 *keysend* 的功能發送，我們將在 Keysend 自發付款 中更詳細地討論。現在，我們將解釋使用發票的更簡單付款流程。

Alice 訪問 Dina 的網站，在表單中輸入 50,000 聰的金額，作為回應，Dina 的閃電網路節點以閃電網路發票的形式生成 50,000 聰的付款請求。這種互動透過網路發生，在閃電網路之外，如 Alice 從 Dina 的網站請求發票 所示。

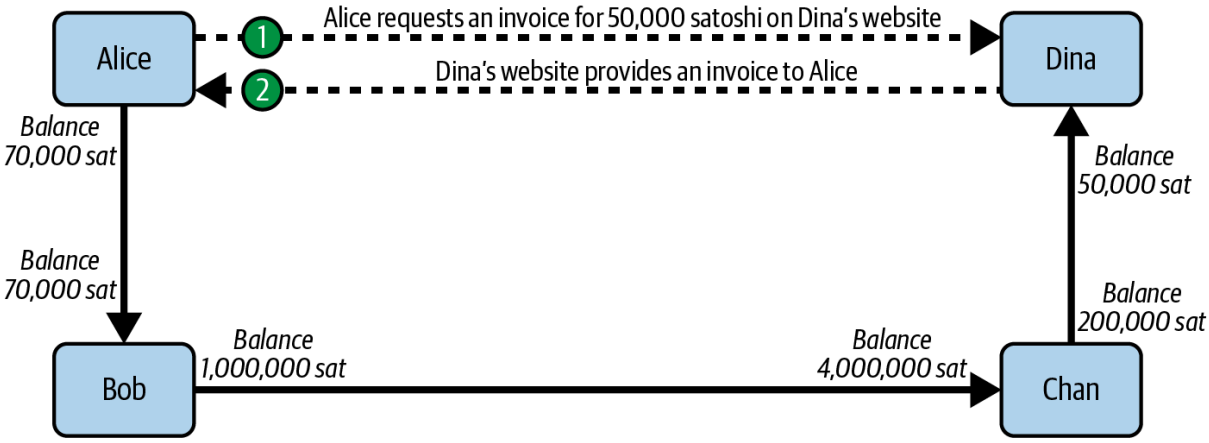

Figure 45. Alice 從 Dina 的網站請求發票

正如我們在前面的範例中看到的，我們假設 Alice 與 Dina 沒有直接的支付通道。相反，Alice 與 Bob 有通道，Bob 與 Chan 有通道，Chan 與 Dina 有通道。為了付款給 Dina，Alice 必須找到一條連接她到 Dina 的路徑。我們將在 路徑尋找與付款傳遞 中更詳細地討論這一步。現在，讓我們假設 Alice 能夠收集有關可用通道的資訊，並看到從她到 Dina 有一條路徑，途經 Bob 和 Chan。

還記得 Bob 和 Chan 可能會期望獲得一小筆補償來透過他們的節點路由付款嗎？Alice 想付給 Dina 50,000 聰，但正如你將在以下章節中看到的，她將向 Bob 發送 50,200 聰。額外的 200 聰將作為路由費支付給 Bob 和 Chan 各 100 聰。

現在，Alice 的節點可以構建一筆閃電網路付款。在接下來的幾節中，我們將看到 Alice 的節點如何構建 HTLC 來付款給 Dina，以及該 HTLC 如何沿路徑從 Alice 轉發到 Dina。

8.6.1. 鏈上與鏈下 HTLC 結算

閃電網路的目的是實現_鏈下_交易，這些交易與鏈上交易一樣被信任，因為沒有人可以欺騙。沒有人可以欺騙的原因是，在任何時候，任何參與者都可以將他們的鏈下交易放到鏈上。每筆鏈下交易都準備好隨時提交到比特幣區塊鏈。因此，如果有必要，比特幣區塊鏈充當爭議解決

和最終結算機制。

任何交易都可以隨時上鏈的事實正是所有這些交易可以保持在鏈下的原因。如果你知道你有追索權，你可以繼續與其他參與者合作，避免鏈上結算和額外費用的需要。

在以下所有範例中，我們將假設這些交易中的任何一筆都可以隨時上鏈。參與者將選擇將它們保持在鏈下，但除了鏈上挖礦交易產生的較高費用和延遲外，系統的功能沒有區別。無論所有交易都在鏈上還是鏈下，範例的工作方式都相同。

8.7. 雜湊時間鎖定合約

在本節中，我們解釋 HTLC 如何運作。

HTLC 的第一部分是_雜湊_。這指的是使用密碼學雜湊演算法來承諾一個隨機生成的秘密。知道這個秘密就可以兌換付款。密碼學雜湊函數保證雖然任何人都不可能猜到秘密原像，但任何人都可以輕鬆驗證雜湊，而且只有一個可能的原像可以解決付款條件。

在 Alice 從 Dina 獲得付款雜湊 中，我們看到 Alice 從 Dina 那裡獲得一張閃電網路發票。在該發票內部，Dina 編碼了一個_付款雜湊_，這是 Dina 節點生成的秘密的密碼學雜湊。Dina 的秘密稱為_付款原像_。付款雜湊作為識別碼，可用於將付款路由到 Dina。付款原像在付款完成後作為收據和付款證明。

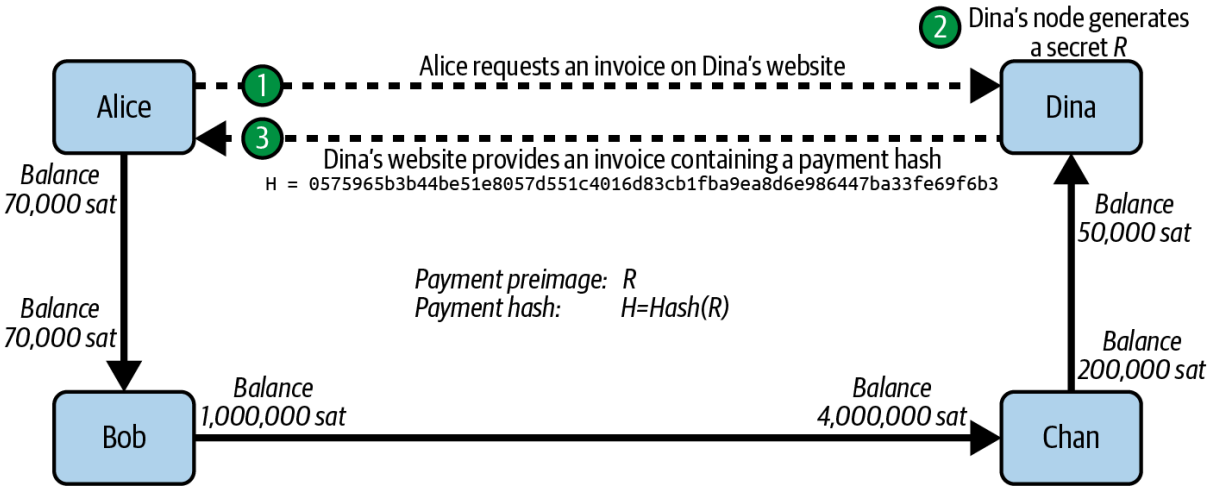

Figure 46. Alice 從 Dina 獲得付款雜湊

在閃電網路中，Dina 的付款原像不會是像 Dinas secret 這樣的短語，而是由 Dina 節點生成的隨機數。讓我們稱那個隨機數為 *R*。

Dina 的節點將計算 *R* 的密碼學雜湊，使得：

- *H* = SHA-256(*R*)

在這個等式中，*H* 是雜湊或_付款雜湊_，*R* 是秘密或_付款原像_。

使用密碼學雜湊函數是保證_無需信任操作_的一個元素。付款中間人不需要相互信任，因為他們知道沒有人可以猜測秘密或偽造它。

8.7.1. 比特幣腳本中的 HTLC

在我們的金幣範例中，Alice 有一份由託管執行的合約，如下所示：

> *Alice 將償還 Bob 12 枚金幣，如果你能出示一個有效訊息，其雜湊為：0575...f6b3。Bob 在合約簽署後有 24 小時出示秘密。如果 Bob 在此時間前沒有提供秘密，Alice 的存款將由託管服務退還，合約變為無效。*

讓我們看看如何在比特幣腳本中將其實作為 HTLC。在 在比特幣腳本中實作的 HTLC（BOLT #3） 中，我們看到目前在閃電網路中使用的 HTLC 比特幣腳本。你可以在 BOLT #3, Transactions (https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md#offered-htlc-outputs) 中找到這個定義。

Example 3. 在比特幣腳本中實作的 HTLC（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
```

哇，這看起來很複雜！不過別擔心，我們會一步一步地分析並簡化它。

目前在閃電網路中使用的比特幣腳本相當複雜，因為它針對鏈上空間效率進行了優化，這使它非常緊湊但難以閱讀。

在以下章節中，我們將專注於腳本的主要元素，並呈現與閃電網路中實際使用的略有不同的簡化腳本。

HTLC 的主要部分在 在比特幣腳本中實作的 HTLC（BOLT #3） 的第 10 行。讓我們從頭開始建構它！

8.7.2. 付款原像和雜湊驗證

HTLC 的核心是雜湊，如果接收者知道付款原像，就可以進行付款。Alice 將付款鎖定到特定的付款雜湊，Bob 必須出示付款原像才能領取資金。比特幣系統可以透過對 Bob 的付款原像進行雜湊並將結果與 Alice 用於鎖定資金的付款雜湊進行比較來驗證 Bob 的付款原像是否正確。

HTLC 的這部分可以在比特幣腳本中實作如下：

```
OP_SHA256 <H> OP_EQUAL
```

Alice 可以建立一筆交易輸出，支付 50,200 聰，使用上面的鎖定腳本，將 `<H>` 替換為 Dina 提供的雜湊值 0575...f6b3。然後，Alice 可以簽署這筆交易並提供給 Bob：

Alice 向 Bob 提供 50,200 聰的 HTLC

```
OP_SHA256 0575...f6b3 OP_EQUAL
```

Bob 在知道 Dina 的秘密之前無法花費這個 HTLC，因此花費 HTLC 取決於 Bob 完成一路到 Dina 的付款。

一旦 Bob 有了 Dina 的秘密，Bob 可以用包含秘密原像值 *R* 的解鎖腳本來花費這個輸出。

解鎖腳本與鎖定腳本結合將產生：

```
<R> OP_SHA256 <H> OP_EQUAL
```

比特幣腳本引擎將如下評估此腳本：

1. R 被推入堆疊。
2. `OP_SHA256` 運算子從堆疊中取出值 R 並對其進行雜湊，將結果 H~R~ 推入堆疊。
3. H 被推入堆疊。
4. `OP_EQUAL` 運算子比較 H 和 H~R~。如果它們相等，結果為 TRUE，腳本完成，付款被驗證。

8.7.3. 將 HTLC 從 Alice 擴展到 Dina

Alice 現在將把 HTLC 擴展到整個網路，使其到達 Dina。

在 在網路中傳播 HTLC 中，我們看到 HTLC 從 Alice 傳播到 Dina。Alice 給了 Bob 一個 50,200 聰的 HTLC。Bob 現在可以建立一個 50,100 聰的 HTLC 並給 Chan。

Bob 知道 Chan 在不廣播秘密的情況下無法兌換 Bob 的 HTLC，此時 Bob 也可以使用秘密來兌換 Alice 的 HTLC。這是一個非常重要的點，因為它確保了 HTLC 的端對端_原子性_。要花費 HTLC，需要揭示秘密，然後這使得其他人也可以花費他們的 HTLC。要麼所有的 HTLC 都可以

花費，要麼所有的 HTLC 都不能花費：原子性！

因為 Alice 的 HTLC 比 Bob 給 Chan 的 HTLC 多 100 聰，如果這筆付款完成，Bob 將賺取 100 聰作為路由費。

Bob 沒有承擔風險，也不信任 Alice 或 Chan。相反，Bob 信任簽名的交易加上秘密將可以在比特幣區塊鏈上兌換。

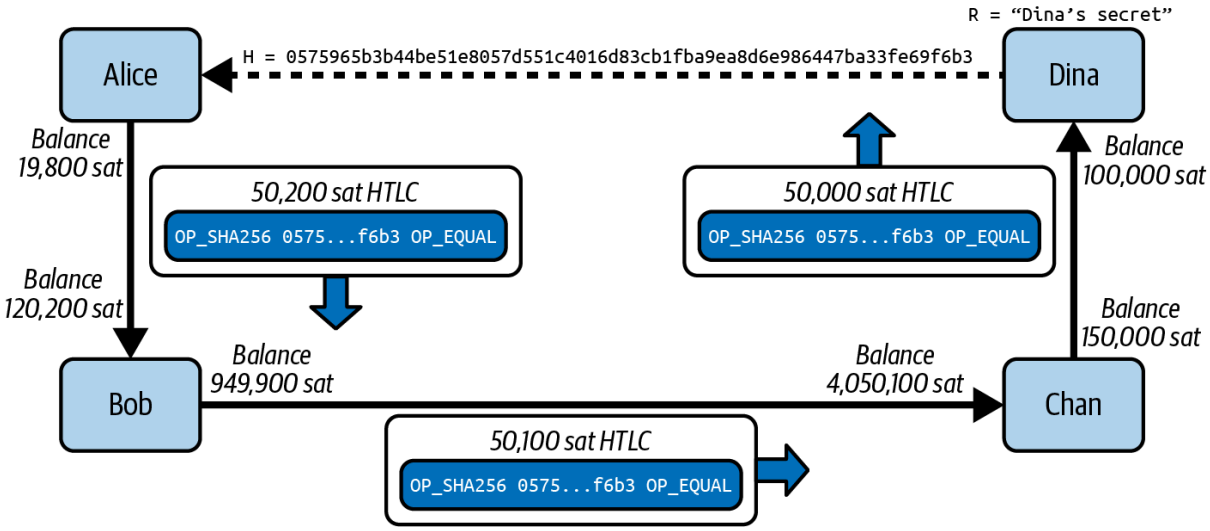

Figure 47. 在網路中傳播 HTLC

同樣，Chan 可以向 Dina 擴展 50,000 聰的 HTLC。他沒有冒任何風險，也不信任 Bob 或 Dina。要兌換 HTLC，Dina 必須廣播秘密，Chan 可以用它來兌換 Bob 的 HTLC。Chan 也將賺取 100 聰作為路由費。

8.7.4. 反向傳播秘密

一旦 Dina 從 Chan 收到 50,000 聰的 HTLC，她現在可以獲得付款了。Dina 可以簡單地將這個 HTLC 提交到鏈上，並透過在花費交易中揭示秘密來花費它。或者，Dina 可以透過將秘密給 Chan 來更新與 Chan 的通道餘額。沒有理由產生交易費用並上鏈。因此，Dina 將秘密發送給 Chan，他們同意更新他們的通道餘額以反映 50,000 聰的閃電網路付款給 Dina。在 Dina 在鏈下結算 Chan 的 HTLC 中，我們看到 Dina 將秘密給 Chan，從而履行 HTLC。

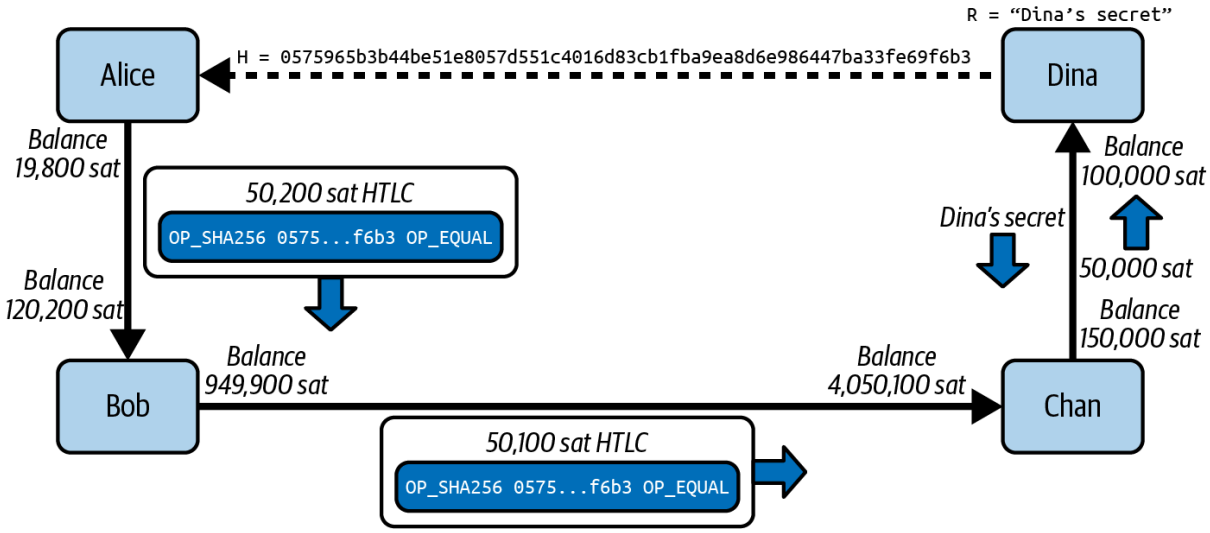

Figure 48. Dina 在鏈下結算 Chan 的 HTLC

注意 Dina 的通道餘額從 50,000 聰變為 100,000 聰。Chan 的通道餘額從 200,000 聰減少到 150,000 聰。通道容量沒有改變，但 50,000 聰從 Chan 這邊移動到 Dina 這邊。

Chan 現在有了秘密並且已經付給 Dina 50,000 聰。他可以這樣做而沒有任何風險，因為秘密允許 Chan 兌換 Bob 的 50,100 聰 HTLC。Chan 可以選擇將該 HTLC 提交到鏈上並透過在比特幣區塊鏈上揭示秘密來花費它。但是，像 Dina 一樣，他寧願避免交易費用。所以，他將秘密發送給 Bob，這樣他們就可以更新他們的通道餘額以反映從 Bob 到 Chan 的 50,100 聰閃電網路付款。在 Chan 在鏈下結算 Bob 的 HTLC 中，我們看到 Chan 將秘密發送給 Bob 並收到付款作為回報。

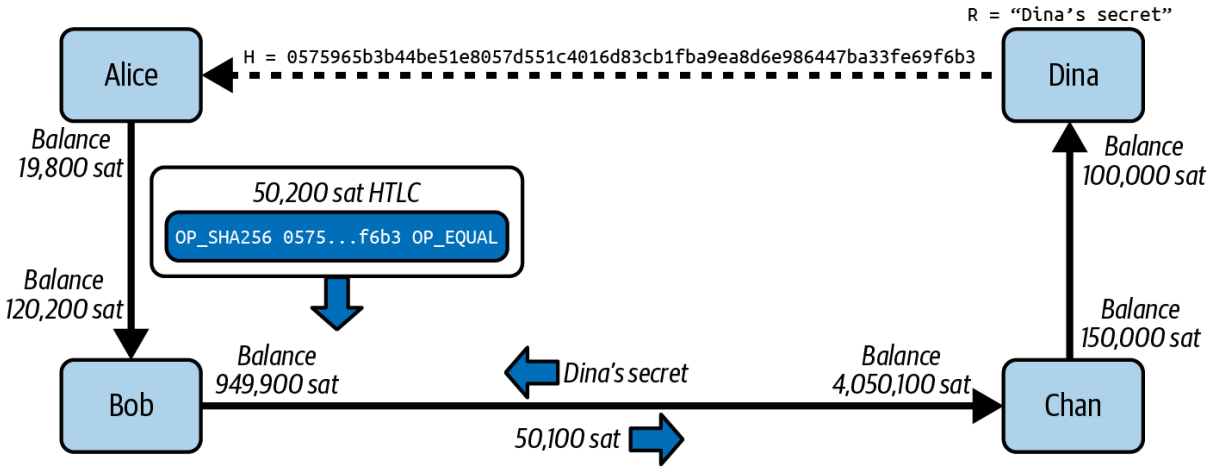

Figure 49. Chan 在鏈下結算 Bob 的 HTLC

Chan 已經付給 Dina 50,000 聰，並從 Bob 那裡收到 50,100 聰。所以 Chan 的通道餘額多了 100 聰，這是他作為路由費賺取的。

Bob 現在也有了秘密。他可以用它來花費 Alice 的鏈上 HTLC。或者，他可以透過在與 Alice 的通道中結算 HTLC 來避免交易費用。在 Bob 在鏈下結算 Alice 的 HTLC 中，我們看到 Bob 將秘密發送給 Alice，他們更新通道餘額以反映從 Alice 到 Bob 的 50,200 聰閃電網路付款。

Figure 50. Bob 在鏈下結算 Alice 的 HTLC

Bob 從 Alice 收到 50,200 聰，並付給 Chan 50,100 聰，所以他從路由費中多了 100 聰的通道餘額。

Alice 收到秘密並已結算 50,200 聰的 HTLC。秘密可以用作_收據_來證明 Dina 為該特定付款雜湊收到了付款。

最終的通道餘額反映了 Alice 付款給 Dina 以及在每一跳支付的路由費，如 付款後的通道餘額 所示。

Figure 51. 付款後的通道餘額

8.7.5. 簽名綁定：防止 HTLC 被盜

有一個陷阱。你注意到了嗎？

如果 Alice、Bob 和 Chan 如 付款後的通道餘額 所示建立 HTLC，他們面臨著雖小但不可忽視的損失風險。任何知道秘密的人都可以兌換（花費）那些 HTLC。起初只有 Dina 知道秘密。Dina 應該只花費 Chan 的 HTLC。但 Dina 可以同時花費所有三個 HTLC，甚至在一筆花費交易中！畢竟，Dina 比任何人都先知道秘密。同樣，一旦 Chan 知道秘密，他應該只花費 Bob 提供的 HTLC。但如果 Chan 也花費 Alice 提供的 HTLC 呢？

這不是_無需信任的_！它未能通過最重要的安全特性。我們需要修復這個問題。

HTLC 腳本必須有一個額外的條件，將每個 HTLC 綁定到特定的接收者。我們透過要求與每個接收者的公鑰匹配的數位簽章來做到這一點，從而防止任何其他人花費該 HTLC。由於只有指定的接收者能夠產生與該公鑰匹配的數位簽章，只有指定的接收者才能花費該 HTLC。

讓我們再次查看帶有此修改的腳本。Alice 給 Bob 的 HTLC 被修改為包含 Bob 的公鑰和 OP_CHECKSIG 運算子。

這是修改後的 HTLC 腳本：

```
OP_SHA256 <H> OP_EQUALVERIFY <Bob's Pub> OP_CHECKSIG
```

> 注意我們還將 OP_EQUAL 改為 OP_EQUALVERIFY。當運算子有 VERIFY 後綴時，它不會在堆疊上返回 TRUE 或 FALSE。相反，如果結果為假，它會_停止_執行並使腳本失敗，如果為真則繼續而不產生任何堆疊輸出。

要兌換這個 HTLC，Bob 必須出示包含 Bob 私鑰簽名以及秘密付款原像的解鎖腳本，如下所示：

```
<Bob's Signature> <R>
```

解鎖和鎖定腳本結合起來由腳本引擎評估，如下所示：

```
<Bob's Sig> <R> OP_SHA256 <H> OP_EQUALVERIFY <Bob's Pub> OP_CHECKSIG
```

1. <Bob's Sig> 被推入堆疊。
2. R 被推入堆疊。
3. OP_SHA256 從堆疊頂部彈出 R 並對其進行雜湊，將 H~R~ 推入堆疊。
4. H 被推入堆疊。
5. OP_EQUALVERIFY 彈出 H 和 H~R~ 並比較它們。如果它們不相同，執行停止。否則，我們繼續而不輸出到堆疊。
6. <Bob's Pub> 公鑰被推入堆疊。
7. OP_CHECKSIG 彈出 <Bob's Sig> 和 <Bob's Pub> 並驗證簽名。結果（`TRUE/FALSE`）被推入堆疊。

如你所見，這稍微複雜了一些，但現在我們已經修復了 HTLC 並確保只有預期的接收者才能花費它。

8.7.6. 雜湊優化

讓我們看看到目前為止 HTLC 腳本的第一部分：

```
OP_SHA256 <H> OP_EQUALVERIFY
```

如果我們在前面的符號表示中看這個，看起來 OP_ 運算子佔用了最多的空間。但事實並非如此。比特幣腳本以二進位編碼，每個運算子代表一個位元組。同時，我們用作付款雜湊占位符的 <H> 值是一個 32 位元組（256 位元）的值。你可以在 Bitcoin Wiki: Script (https://en.bitcoin.it/wiki/Script) 或 《精通比特幣》附錄 D「交易腳本語言運算子、常數和符號」 (https://github.com/bitcoinbook/bitcoinbook/blob/develop/appdx-scriptops.asciidoc) 中找到所有比特幣腳本運算子及其二進位和十六進位編碼的列表。

用十六進位表示，我們的 HTLC 腳本看起來像這樣：

```
a8 0575965b3b44be51e8057d551c4016d83cb1fba9ea8d6e986447ba33fe69f6b3 88
```

在十六進位編碼中，OP_SHA256 是 a8，OP_EQUALVERIFY 是 88。這個腳本的總長度是 34 位元組，其中 32 位元組是雜湊。

正如我們之前提到的，閃電網路中的任何參與者都應該能夠將他們持有的鏈下交易放到鏈上，如果他們需要強制執行他們對資金的索取權。要將交易上鏈，他們必須向礦工支付交易費用，這些費用與交易的位元組大小成正比。

因此，我們希望找到方法來最小化交易的鏈上「權重」，盡可能優化腳本。一種方法是在 SHA-256 演算法之上添加另一個雜湊函數，一個產生更小雜湊的函數。比特幣腳本語言提供了 OP_HASH160 運算子，它對原像進行「雙重雜湊」：首先用 SHA-256 對原像進行雜湊，然後用 RIPEMD160 雜湊演算法對結果雜湊再次進行雜湊。RIPEMD160 產生的雜湊是 160 位元或 20 位元組——更加緊湊。在比特幣腳本中，這是一種非常常見的優化，用於許多常見的地址格式。

所以，讓我們改用這種優化。我們的 SHA-256 雜湊是 057596...69f6b3。用 RIPEMD160 再次進行雜湊給我們結果：

```
R = "Dinas secret"
H256 = SHA256(R)
H256 = 0575965b3b44be51e8057d551c4016d83cb1fba9ea8d6e986447ba33fe69f6b3
H160 = RIPEMD160(H256)
H160 = 9e017f6767971ed7cea17f98528d5f5c0ccb2c71
```

Alice 可以計算 Dina 提供的付款雜湊的 RIPEMD160 雜湊，並在她的 HTLC 中使用較短的雜湊，Bob 和 Chan 也可以這樣做！

「優化後的」HTLC 腳本看起來像這樣：

```
OP_HASH160 <H160> OP_EQUALVERIFY
```

用十六進位編碼，這是：

```
a9 9e017f6767971ed7cea17f98528d5f5c0ccb2c71 88
```

其中 OP_HASH160 是 a9，OP_EQUALVERIFY 是 88。這個腳本只有 22 位元組長！我們從每筆在鏈上兌換 HTLC 的交易中節省了 12 位元組。

有了這個優化，你現在可以看到我們如何得出 在比特幣腳本中實作的 HTLC（BOLT #3） 第 10 行所示的 HTLC 腳本：

```
...
    # To local node via HTLC-success transaction.
    OP_HASH160 <RIPEMD160(payment_hash)> OP_EQUALVERIFY...
```

8.7.7. HTLC 合作失敗和超時失敗

到目前為止，我們研究了 HTLC 的「雜湊」部分以及如果每個人都合作並在付款時在線，它會如何運作。

如果有人離線或不合作會發生什麼？如果付款無法成功會發生什麼？

我們需要確保一種「優雅失敗」的方式，因為偶爾的路由失敗是不可避免的。有兩種失敗方式：合作式和帶時間鎖退款。

合作失敗相對簡單：HTLC 由路由中的每個參與者解除，從他們的承諾交易中移除 HTLC 輸出而不改變餘額。我們將在 通道操作與支付轉發 中詳細研究這是如何運作的。

讓我們看看如何在沒有一個或多個參與者合作的情況下逆轉 HTLC。我們需要確保如果其中一個參與者不合作，資金不會簡單地_永遠_鎖定在 HTLC 中。這會給某人勒索另一個參與者資金的機會：「如果你不付我贖金，我就讓你的資金永遠被鎖住。」

為了防止這種情況，每個 HTLC 腳本都包含一個與時間鎖連接的退款條款。還記得我們最初的託管合約嗎？「Bob 在合約簽署後有 24 小時出示秘密。如果 Bob 在此時間前沒有提供秘密，Alice 的存款將由託管服務退還。」

帶時間鎖的退款是確保_原子性_的腳本的重要部分，使整個端對端付款要麼成功，要麼優雅失敗。沒有需要擔心的「半支付」狀態。如果失敗，每個參與者都可以與他們的通道夥伴合作解除 HTLC，或者單方面將帶時間鎖的退款交易上鏈以取回他們的錢。

要在比特幣腳本中實作這個退款，我們使用一個特殊的運算子 `OP_CHECKLOCKTIMEVERIFY`，簡稱 OP_CLTV。這是腳本，如之前在 在比特幣腳本中實作的 HTLC（BOLT #3） 第 13 行所見：

```
...
    OP_DROP <cltv_expiry> OP_CHECKLOCKTIMEVERIFY OP_DROP
    OP_CHECKSIG
...
```

OP_CLTV 運算子採用一個到期時間，定義為此交易有效的區塊高度。如果交易時間鎖與 <cltv_expiry> 設定不同，腳本評估失敗，交易無效。否則，腳本繼續而不輸出到堆疊。記住，VERIFY 後綴意味著此運算子不輸出 TRUE 或 FALSE，而是停止/失敗或繼續而不產生堆疊輸出。

本質上，OP_CLTV 作為「守門人」，如果比特幣區塊鏈上還沒有達到 <cltv_expiry> 區塊高度，就阻止腳本繼續進行。

OP_DROP 運算子簡單地丟棄腳本堆疊上最頂端的項目。這在開始時是必要的，因為有一個來自前面腳本行的「剩餘」項目。在 OP_CLTV _之後_這是必要的，以從堆疊頂部移除 <cltv_expiry> 項目，因為它不再需要。

最後，一旦堆疊被清理，應該還剩下一個公鑰和簽名，OP_CHECKSIG 可以驗證它們。正如我們在 簽名綁定：防止 HTLC 被盜 中看到的，這是必要的，以確保只有資金的合法所有者才能領取它們，透過將此輸出綁定到他們的公鑰並要求簽名。

8.7.8. 遞減時間鎖

當 HTLC 從 Alice 擴展到 Dina 時，每個 HTLC 中的帶時間鎖退款條款有_不同的_ cltv_expiry 值。我們將在 洋蔥路由 中更詳細地看到這一點。但足以說明的是，為了確保有序地解除失敗的付款，每一跳需要等待的退款時間略有不同。每跳之間時間鎖的差異稱為 cltv_expiry_delta，由每個節點設定並向網路公告，我們將在 八卦協定與通道圖 中看到。

例如，Alice 將第一個 HTLC 的退款時間鎖設定為當前 + 500 個區塊的區塊高度（「當前」是當前區塊高度）。Bob 然後將 HTLC 給 Chan 的時間鎖 cltv_expiry 設定為當前 + 450 個區塊。Chan 將時間鎖設定為從當前區塊高度的當前 + 400 個區塊。這樣，Chan 可以在 Bob 獲得他提供給 Chan 的 HTLC 退款_之前_獲得他提供給 Dina 的 HTLC 的退款。Bob 可以在 Alice 可以獲得她提供給 Bob 的 HTLC 退款之前獲得他提供給 Chan 的 HTLC 的退款。遞減的時間鎖防止競爭條件並確保 HTLC 鏈從目的地向源頭反向解除。

8.8. 結論

在本章中，我們看到即使 Alice 與 Dina 沒有直接的支付通道，她也可以付款給 Dina。Alice 可以找到一條連接她到 Dina 的路徑，並透過多個支付通道路由付款，使其到達 Dina。

為了確保付款在多跳之間是原子性和無需信任的，Alice 必須與路徑中的所有中間節點合作實作公平協定。公平協定目前實作為 HTLC，它將資金承諾到從秘密付款原像派生的付款雜湊。

付款路由中的每個參與者都可以向下一個參與者擴展 HTLC，而不用擔心被盜或資金卡住。HTLC 可以透過揭示秘密付款原像來兌換。一旦 HTLC 到達 Dina，她就揭示原像，原像向後流動，解決所有提供的 HTLC。

最後，我們看到帶時間鎖的退款條款如何完成 HTLC，確保每個參與者在付款失敗但由於某種原因其中一個參與者不合作解除 HTLC 時都能獲得退款。透過始終有上鏈退款的選項，HTLC 實現了原子性和無需信任操作的公平目標。

9. 通道操作與支付轉發

在本章中，我們將把支付通道和雜湊時間鎖定合約（HTLC）結合在一起。在 支付通道 中，我們解釋了 Alice 和 Bob 如何在他們兩個節點之間建構支付通道。我們還研究了保護支付通道的承諾和懲罰機制。在 在支付通道網路上路由 中，我們研究了 HTLC 以及如何使用它們透過由多個支付通道組成的路徑路由付款。在本章中，我們透過研究 HTLC 如何在每個支付通道上管理、HTLC 如何承諾到通道狀態，以及如何結算它們來更新通道餘額，將這兩個概念結合在一起。

具體來說，我們將討論「添加、結算、失敗 HTLC」和「通道狀態機」，它們構成了點對點層和路由層之間的重疊，如 閃電網路協定套件中的通道操作和支付轉發 中的輪廓所突顯。

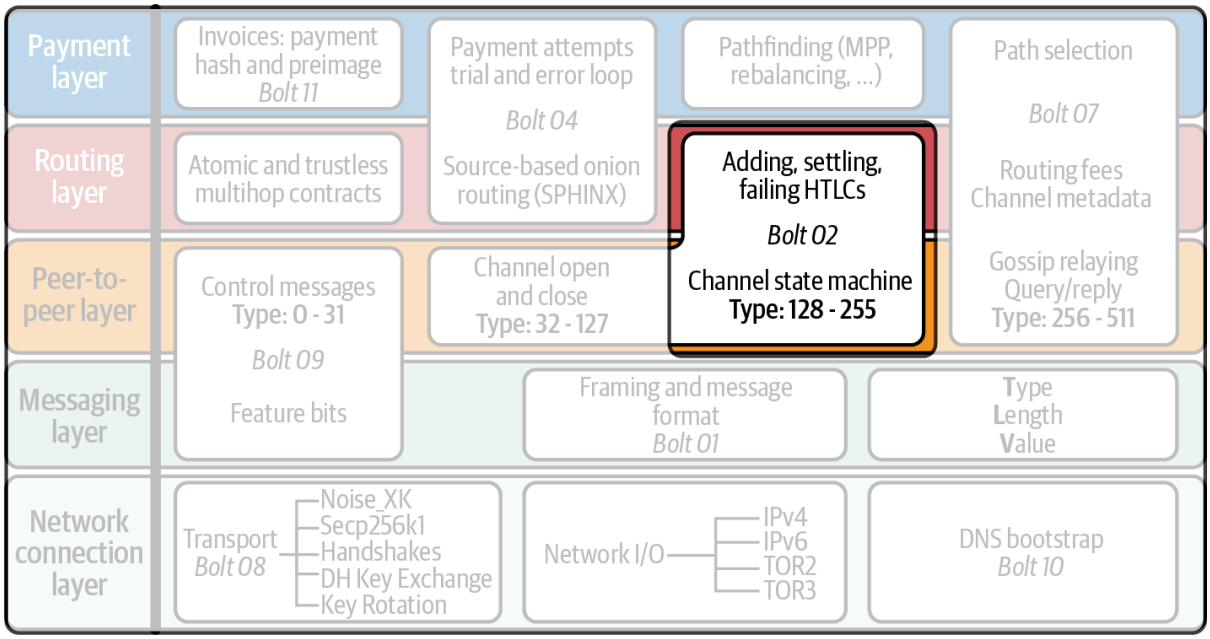

Figure 52. 閃電網路協定套件中的通道操作和支付轉發

9.1. 本地（單通道）與路由（多通道）

儘管可以透過簡單地更新通道餘額和建立新的承諾交易來跨支付通道發送付款，但閃電網路協定即使對於跨支付通道的「本地」付款也使用 HTLC。這樣做的原因是為了維持相同的協定設計，無論付款是只有一跳（跨單個支付通道）還是多跳（跨多個支付通道路由）。

透過為本地和遠端維持相同的抽象，我們不僅簡化了協定設計，還提高了隱私性。對於付款的接收者來說，直接由其通道夥伴發起的付款與其通道夥伴代表其他人轉發的付款之間沒有可辨別的區別。

9.2. 使用 HTLC 轉發付款和更新承諾

我們將重新審視 在支付通道網路上路由 中的範例，以展示從 Alice 到 Dina 的 HTLC 如何承諾到每個支付通道。如你所記得的，在我們的範例中，Alice 透過 Bob 和 Chan 路由 HTLC 向 Dina 支付 50,000 聰。網路如 Alice 透過 Bob 和 Chan 路由 HTLC 向 Dina 付款 所示。

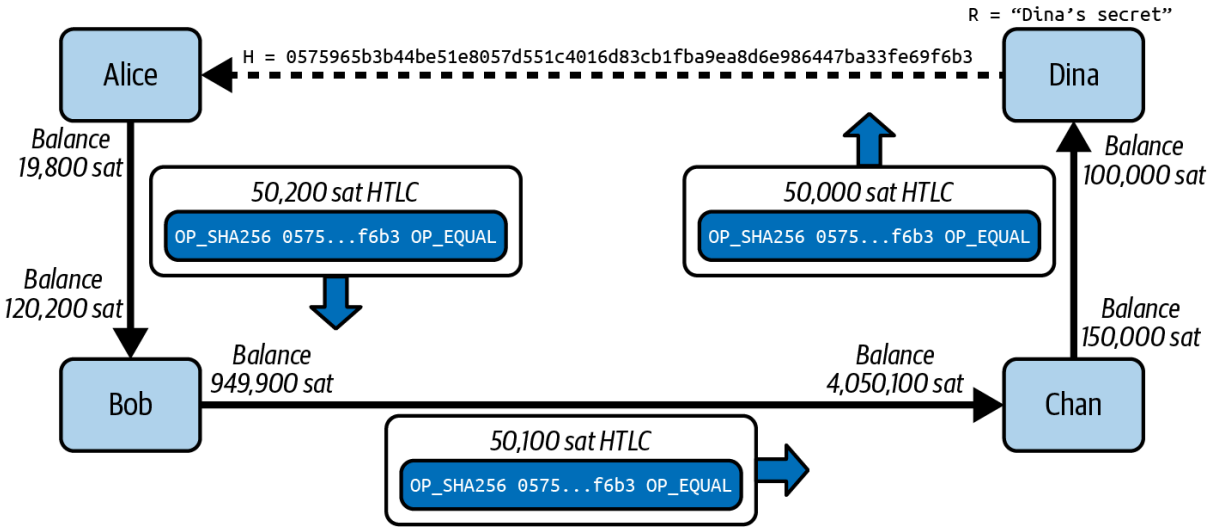

Figure 53. Alice 透過 Bob 和 Chan 路由 HTLC 向 Dina 付款

我們將專注於 Alice 和 Bob 之間的支付通道，並審查他們用於處理此 HTLC 的訊息和交易。

9.2.1. HTLC 和承諾訊息流程

Alice 和 Bob 之間（以及任何一對通道夥伴之間）的訊息流程如 通道夥伴之間 HTLC 承諾的訊息流程 所示。

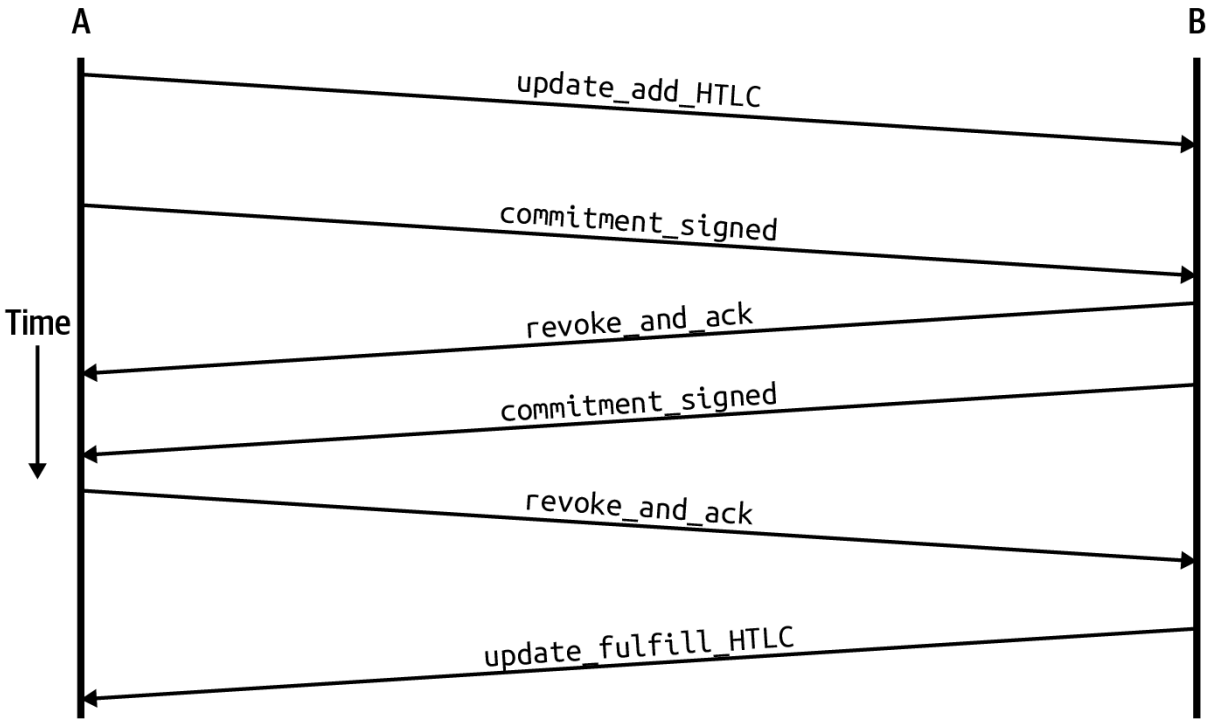

Figure 54. 通道夥伴之間 HTLC 承諾的訊息流程

我們已經在 支付通道 中看過 commitment_signed 和 revoke_and_ack。現在我們將看到 HTLC 如何融入承諾方案。兩個新訊息是 update_add_htlc，Alice 用它來要求 Bob 添加 HTLC，以及 update_fulfill_htlc，Bob 在收到付款秘密（Dina 的秘密）後用它來兌換 HTLC。

9.3. 使用 HTLC 轉發付款

Alice 和 Bob 開始時的支付通道每邊有 70,000 聰的餘額。

正如我們在 支付通道 中看到的，這意味著 Alice 和 Bob 已經協商並各自持有承諾交易。這些承諾交易是非對稱的、延遲的和可撤銷的，看起來像 Alice 和 Bob 的初始承諾交易 中的範例。

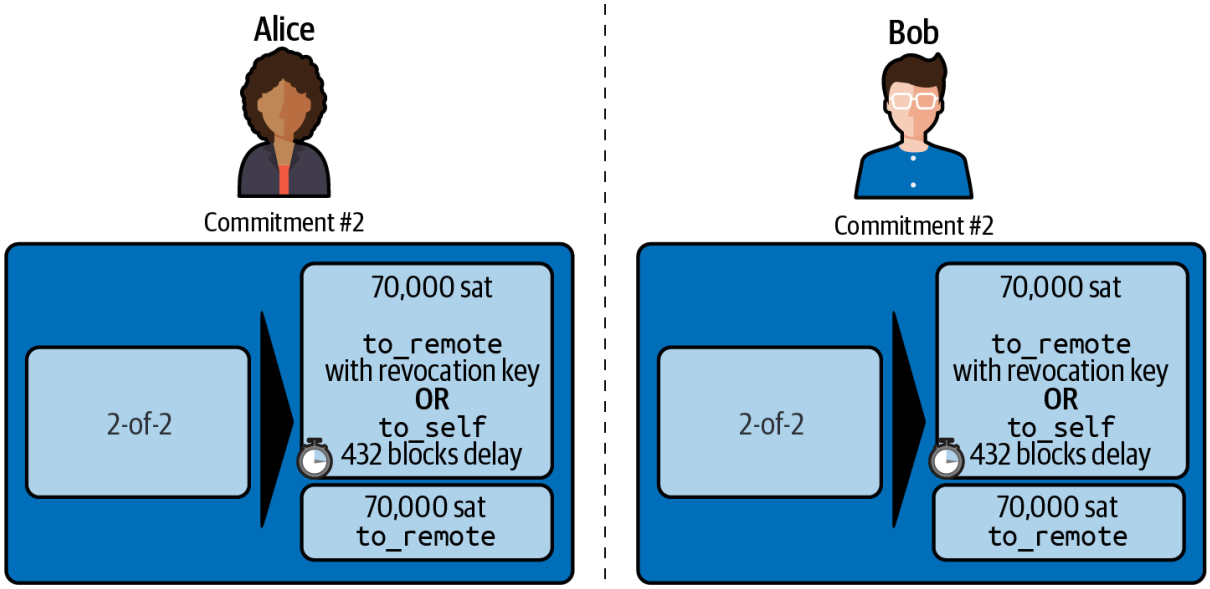

Figure 55. Alice 和 Bob 的初始承諾交易

9.3.1. 添加 HTLC

Alice 希望 Bob 接受一個價值 50,200 聰的 HTLC 以轉發給 Dina。為此，Alice 必須向 Bob 發送此 HTLC 的詳細資訊，包括付款雜湊和金額。Bob 還需要知道將其轉發到哪裡，這是我們在 洋蔥路由 中詳細討論的內容。

要添加 HTLC，Alice 透過向 Bob 發送 update_add_htlc 訊息來開始我們在 通道夥伴之間 HTLC 承諾的訊息流程 中看到的流程。

9.3.2. update_add_HTLC 訊息

Alice 向 Bob 發送 `update_add_HTLC` 閃電網路訊息。此訊息定義在 BOLT #2: Peer Protocol, `update_add_HTLC` (https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md#adding-an-htlc-update_add_htlc)，如範例 9-1 所示。

Example 4. `update_add_HTLC` 訊息

```
[channel_id:channel_id]
[u64:id]
[u64:amount_msat]
[sha256:payment_hash]
[u32:cltv_expiry]
[1366*byte:onion_routing_packet]
```

channel_id

這是 Alice 與 Bob 要添加 HTLC 的通道。記住，Alice 和 Bob 之間可能有多個通道。

id

這是 HTLC 計數器，從 Alice 向 Bob 提供的第一個 HTLC 的 0 開始，每個後續提供的 HTLC 遞增。

amount_msat

這是 HTLC 的金額（價值），以毫聰為單位。在我們的範例中，這是 50,200,000 毫聰（即 50,200 聰）。

payment_hash

這是從 Dina 的發票計算出的付款雜湊。它是 H = RIPEMD160(SHA-256(R))，其中 R 是 Dina 的秘密，只有 Dina 知道，如果 Dina 收到付款就會被揭示。

cltv_expiry

這是此 HTLC 的到期時間，如果 HTLC 在此時間內未能到達 Dina，將編碼為帶時間鎖的退款。

onion_routing_packet

這是一個洋蔥加密的路由，告訴 Bob 下一步將此 HTLC 轉發到哪裡（給 Chan）。洋蔥路由將在 洋蔥路由 中詳細介紹。

> 提醒一下，閃電網路內的記帳單位是毫聰（千分之一聰），而比特幣記帳單位是聰。HTLC 中的任何金額都是毫聰，然後在比特幣承諾交易中四捨五入到最接近的聰。

9.3.3. 承諾交易中的 HTLC

收到的資訊足以讓 Bob 建立新的承諾交易。新的承諾交易有相同的兩個輸出 to_self 和 to_remote 用於 Alice 和 Bob 的餘額，以及一個代表 Alice 提供的 HTLC 的_新_輸出。

我們已經在 在支付通道網路上路由 中看到了 HTLC 的基本結構。提供的 HTLC 的完整腳本定義在 BOLT #3: Transactions, Offered HTLC Output (https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md#offered-htlc-outputs)，如 提供的 HTLC 輸出腳本 所示。

Example 5. 提供的 HTLC 輸出腳本

```
# 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) `OP_IF` 條件的第一個子句可由 Alice 使用撤銷金鑰兌換。如果此承諾稍後被撤銷，Alice 將擁有撤銷金鑰來在懲罰交易中領取此輸出，取走整個通道餘額。

(2) 第二個子句可由原像（付款秘密，或在我們的範例中，Dina 的秘密）兌換，如果它被揭示的話。這允許 Bob 在他有 Dina 的秘密時領取此輸出，意味著他已成功將付款交付給 Dina。

(3) 第三個也是最後一個子句是如果 HTLC 過期而未到達 Dina，則將 HTLC 退還給 Alice。它有帶到期時間 cltv_expiry 的時間鎖。這確保了 Alice 的餘額不會「卡」在無法路由到 Dina 的 HTLC 中。

有三種方法可以領取此輸出。嘗試閱讀腳本，看看你能否弄清楚（記住，它是一種基於堆疊的語言，所以事情看起來是「倒過來的」）。

9.3.4. 帶 HTLC 輸出的新承諾

Bob 現在有了必要的資訊來將此 HTLC 腳本作為額外輸出添加並建立新的承諾交易。Bob 的新承諾將在 HTLC 輸出中有 50,200 聰。該金額將來自 Alice 的通道餘額，因此 Alice 的新餘額將是 19,800 聰（70,000 – 50,200 = 19,800）。Bob 將此承諾建構為暫定的「承諾 #3」，如 Bob 帶 HTLC 輸出的新承諾 所示。

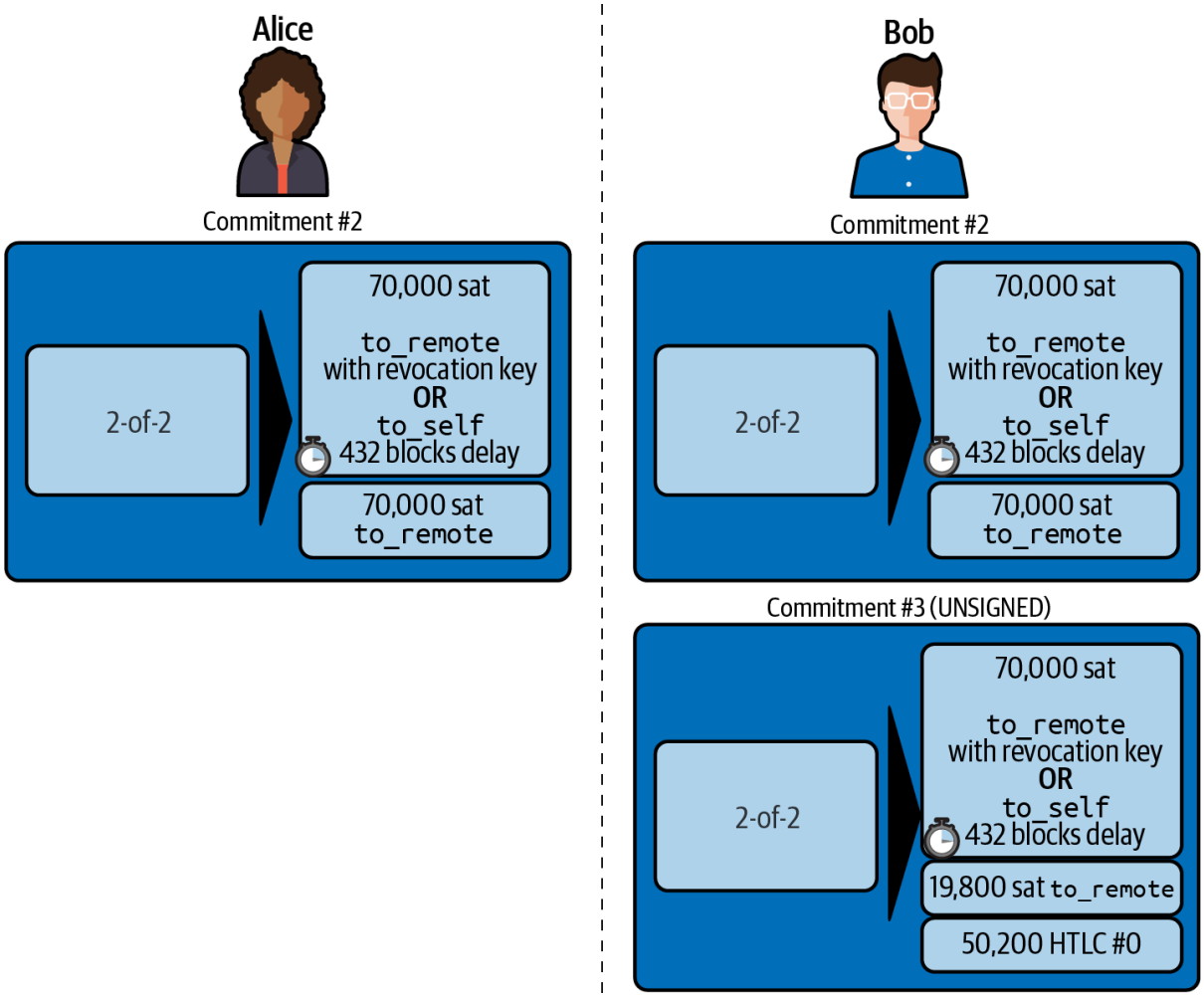

Figure 56. Bob 帶 HTLC 輸出的新承諾

9.3.5. Alice 提交

在發送 update_add_htlc 訊息後不久，她將承諾到通道的新狀態，以便 Bob 可以安全地添加 HTLC。Bob 有 HTLC 資訊並已建構新的承諾，但還沒有由 Alice 簽署的新承諾。

Alice 向 Bob 發送 commitment_signed，包含新承諾的簽名和其中 HTLC 的簽名。我們在 支付通道 中看過 commitment_signed 訊息，但現在我們可以理解其餘欄位了。作為提醒，它顯示在 `commitment_signed` 訊息 中。

Example 6. `commitment_signed` *訊息*

```
[channel_id:channel_id]
[signature:signature]
[u16:num_htlcs]
[num_htlcs*signature:htlc_signature]
```

num_htlcs 和 htlc_signature 欄位現在更有意義了：

num_htlcs

這是承諾交易中未完成的 HTLC 數量。在我們的範例中，只有一個 HTLC，即 Alice 提供的那個。

htlc_signature

這是一個簽名陣列（長度為 num_htlcs），包含 HTLC 輸出的簽名。

Alice 可以毫不猶豫地發送這些簽名：如果 HTLC 過期而未路由到 Dina，她總是可以獲得退款。

現在，Bob 有了新的簽名承諾交易，如 Bob 有了新的簽名承諾 所示。

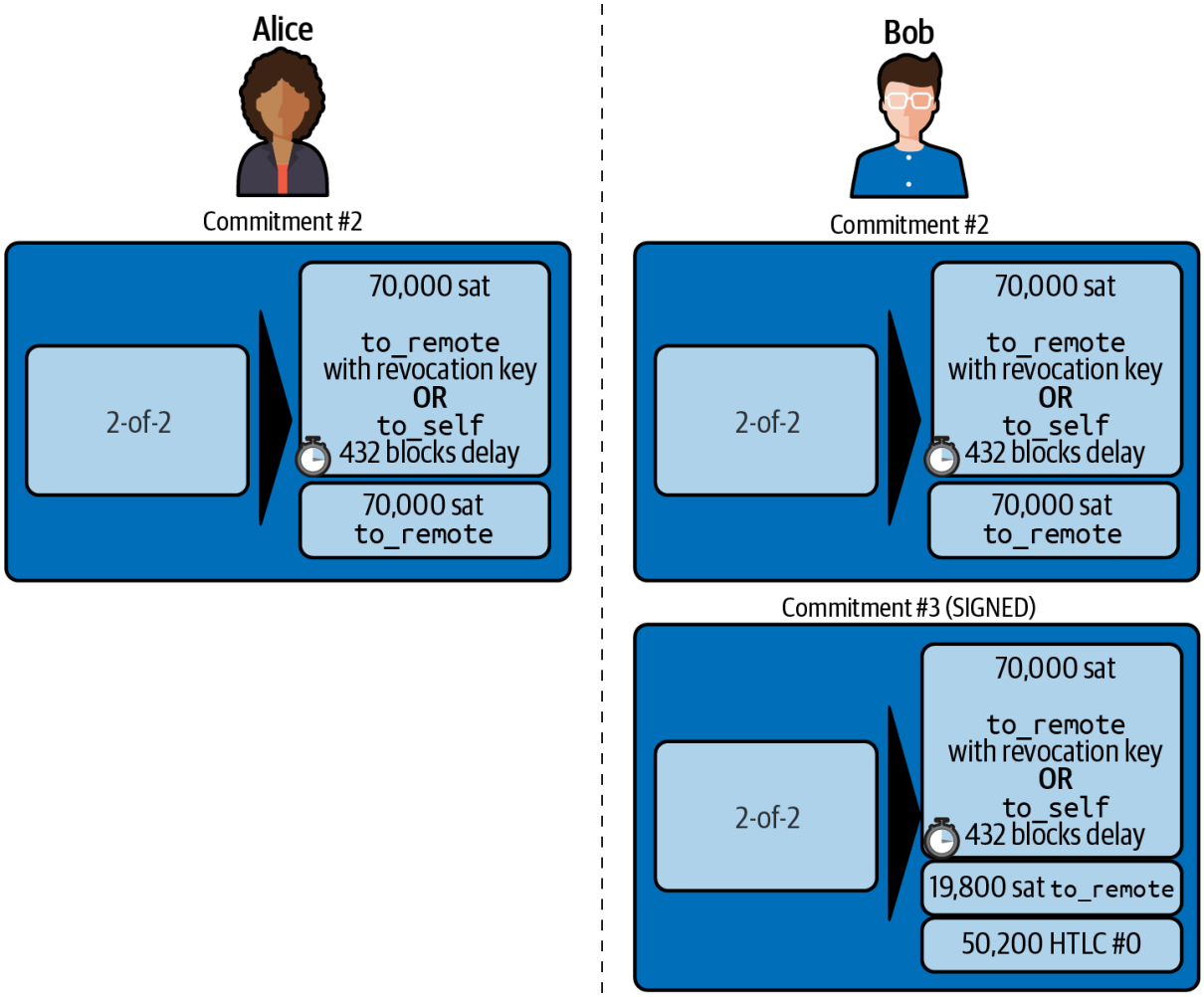

Figure 57. Bob 有了新的簽名承諾

9.3.6. Bob 確認新承諾並撤銷舊承諾

現在 Bob 有了新的簽名承諾，他需要確認它並撤銷舊承諾。他透過發送 revoke_and_ack 訊息來做到這一點，正如我們之前在 支付通道 中看到的。作為提醒，該訊息顯示在 revoke_and_ack 訊息 中。

Example 7. revoke_and_ack 訊息

```
[channel_id:channel_id]
[32*byte:per_commitment_secret]
[point:next_per_commitment_point]
```

Bob 發送 per_commitment_secret，允許 Alice 建構撤銷金鑰來建立花費 Bob 舊承諾的懲罰交易。一旦 Bob 發送了這個，如果他發布「承諾 #2」，他就冒著懲罰交易和失去所有資金的風險。因此，舊承諾實際上被撤銷了。

Bob 實際上已經將通道狀態向前推進，如 Bob 已撤銷舊承諾 所示。

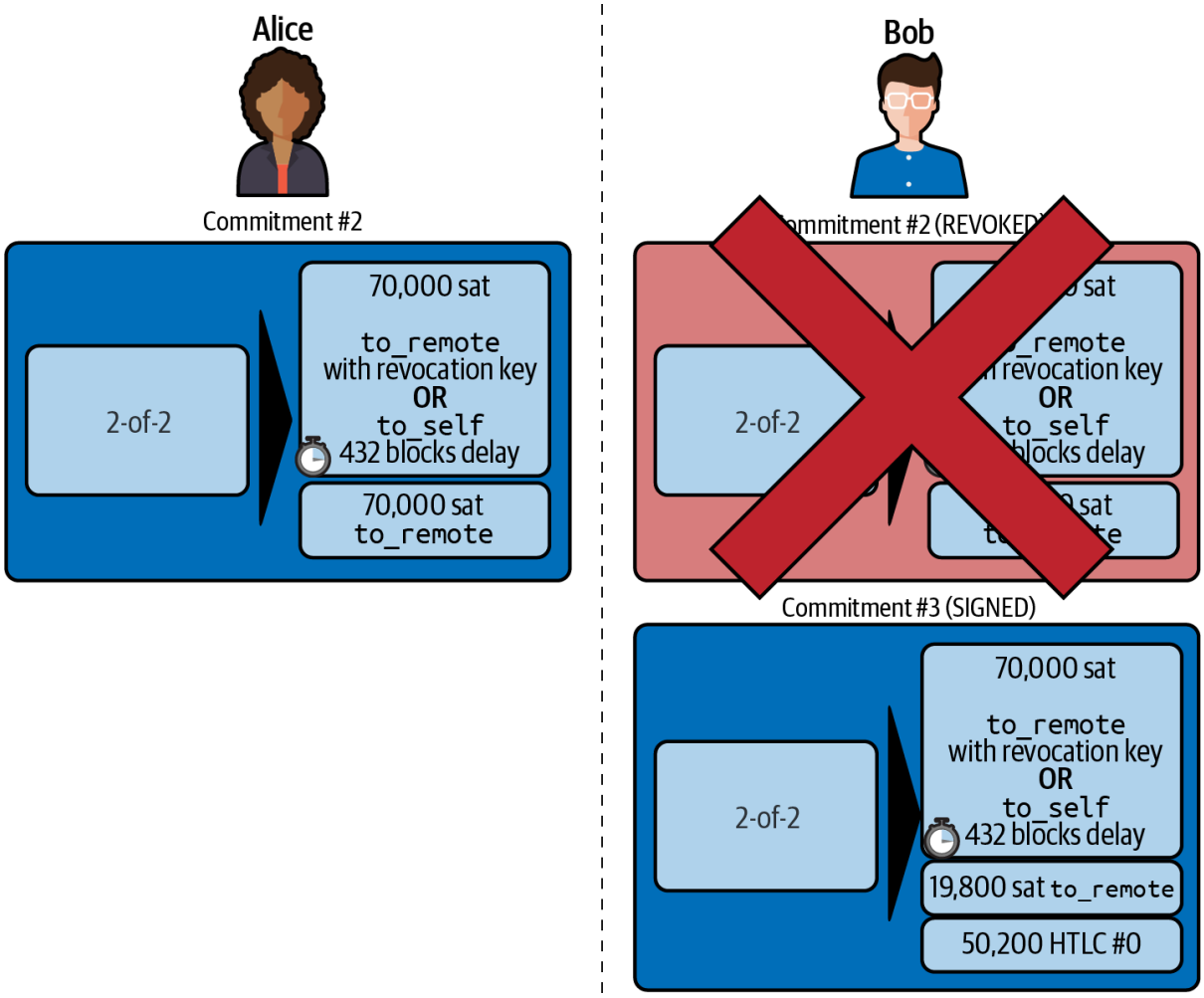

Figure 58. Bob 已撤銷舊承諾

儘管 Bob 有一個新的（簽名的）承諾交易和裡面的 HTLC 輸出，他還不能認為他的 HTLC 已成功設定。

他首先需要讓 Alice 撤銷她的舊承諾，因為否則，Alice 可以將她的餘額回滾到 70,000 聰。Bob 需要確保 Alice 也有一個包含 HTLC 的承諾交易，並且已經撤銷了舊承諾。

這就是為什麼，如果 Bob 不是 HTLC 資金的最終接收者，他還不應該透過在與 Chan 的下一個通道上提供 HTLC 來轉發 HTLC。

Alice 已經建構了一個包含新 HTLC 的鏡像新承諾交易，但它還沒有被 Bob 簽署。我們可以在 Alice 帶 HTLC 輸出的新承諾 中看到它。

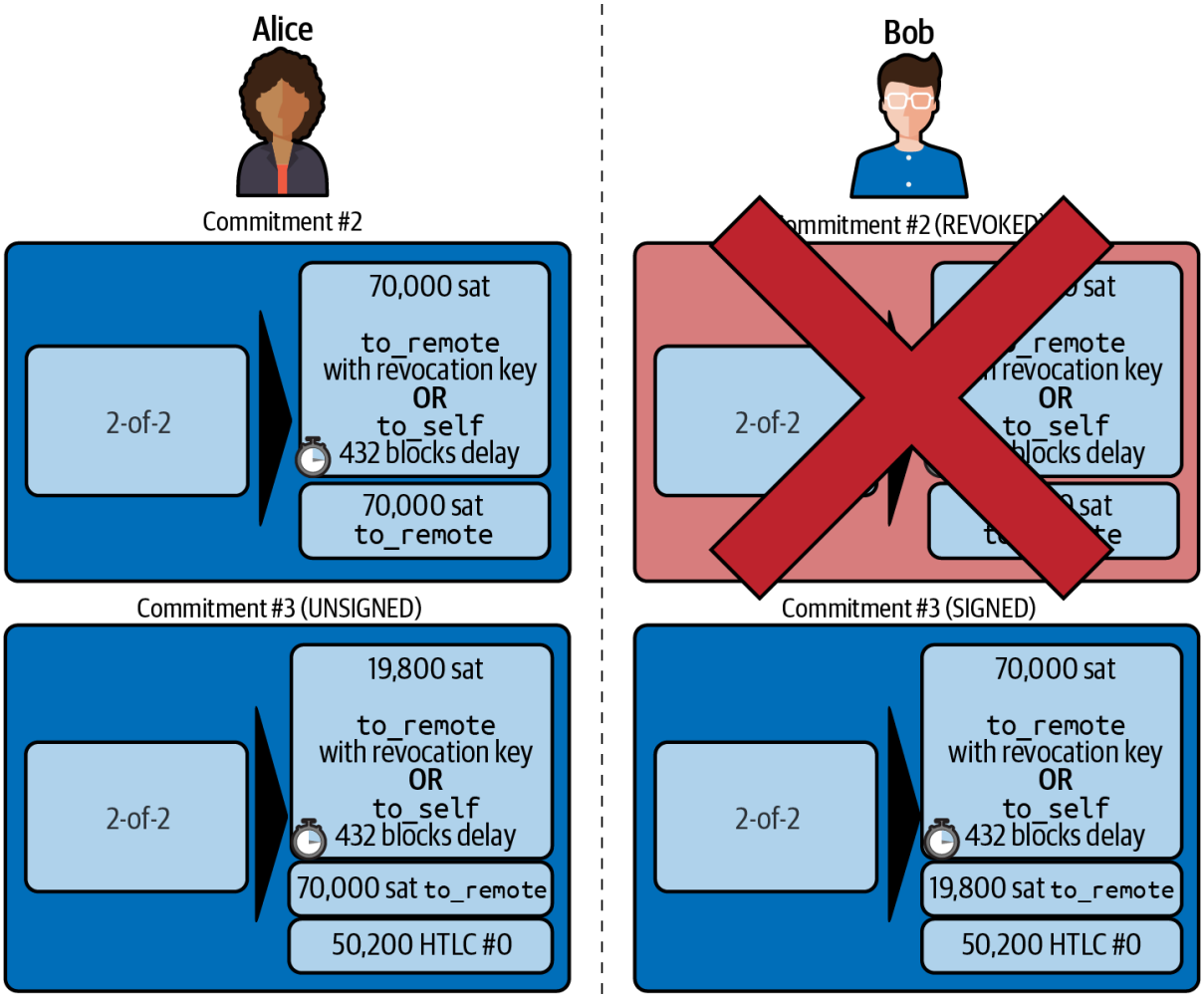

Figure 59. Alice 帶 HTLC 輸出的新承諾

正如我們在 支付通道 中描述的，Alice 的承諾是 Bob 的鏡像，因為它包含用於撤銷和懲罰執行舊承諾的非對稱、延遲、可撤銷結構。Alice 的 19,800 聰餘額（扣除 HTLC 值後）是延遲的和可撤銷的。Bob 的 70,000 聰餘額可以立即兌換。

接下來，commitment_signed 和 revoke_and_ack 的訊息流程現在重複，但方向相反。Bob 發送 commitment_signed 來簽署 Alice 的新承諾，Alice 透過撤銷她的舊承諾來回應。

為了完整起見，讓我們快速審查這輪承諾/撤銷發生時的承諾交易。

9.3.7. Bob 提交

Bob 現在向 Alice 發送 commitment_signed，包含他對 Alice 新承諾交易的簽名，包括她添加的 HTLC 輸出。

現在 Alice 有了新承諾交易的簽名。通道的狀態如 Alice 有了新的簽名承諾 所示。

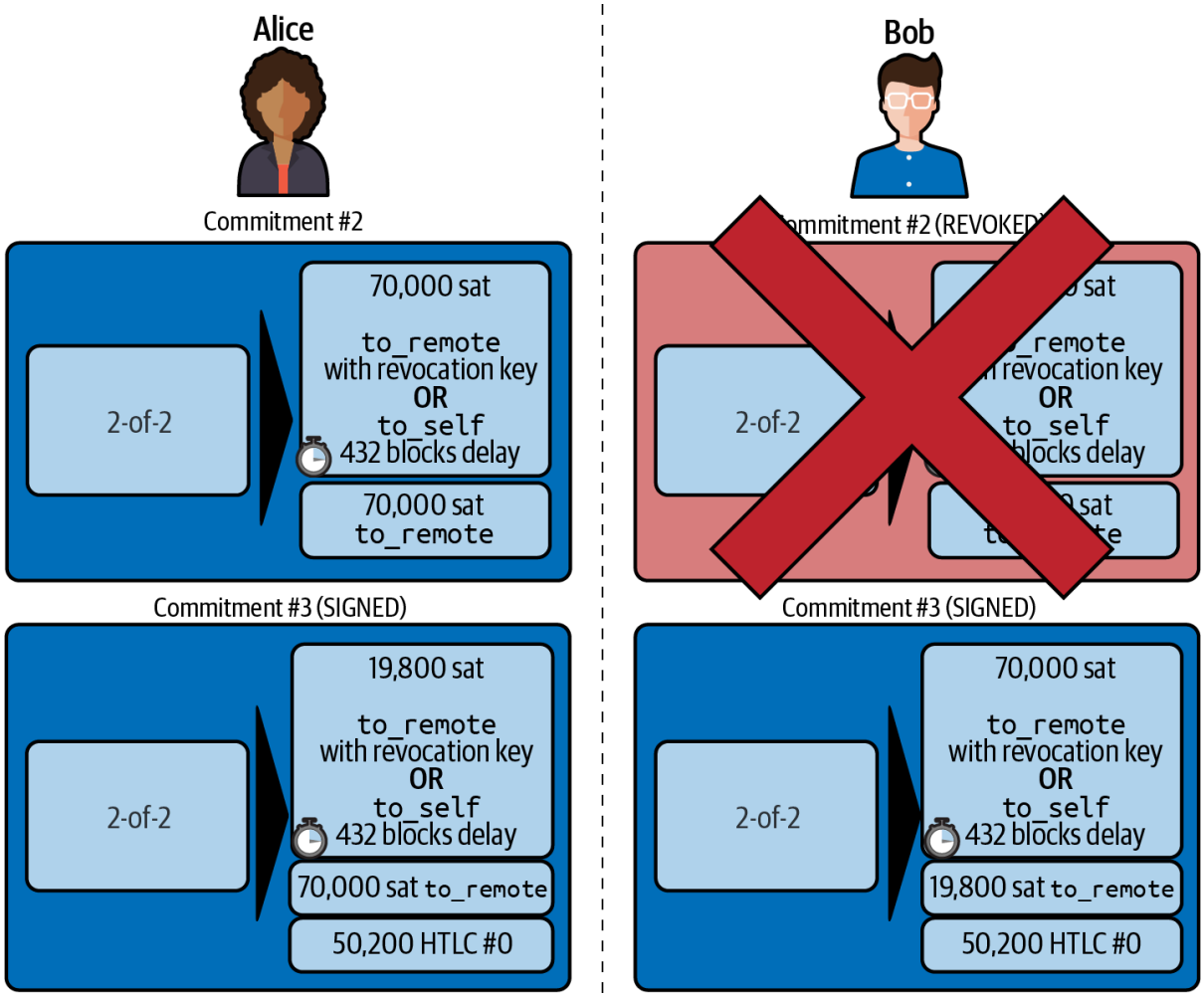

Figure 60. Alice 有了新的簽名承諾

Alice 現在可以透過撤銷舊承諾來確認新承諾。Alice 發送包含必要 per_commitment_point 的 revoke_and_ack 訊息，這將允許 Bob 建構撤銷金鑰和懲罰交易。因此，Alice 撤銷了她的舊承諾。

通道狀態如 Alice 已撤銷舊承諾 所示。

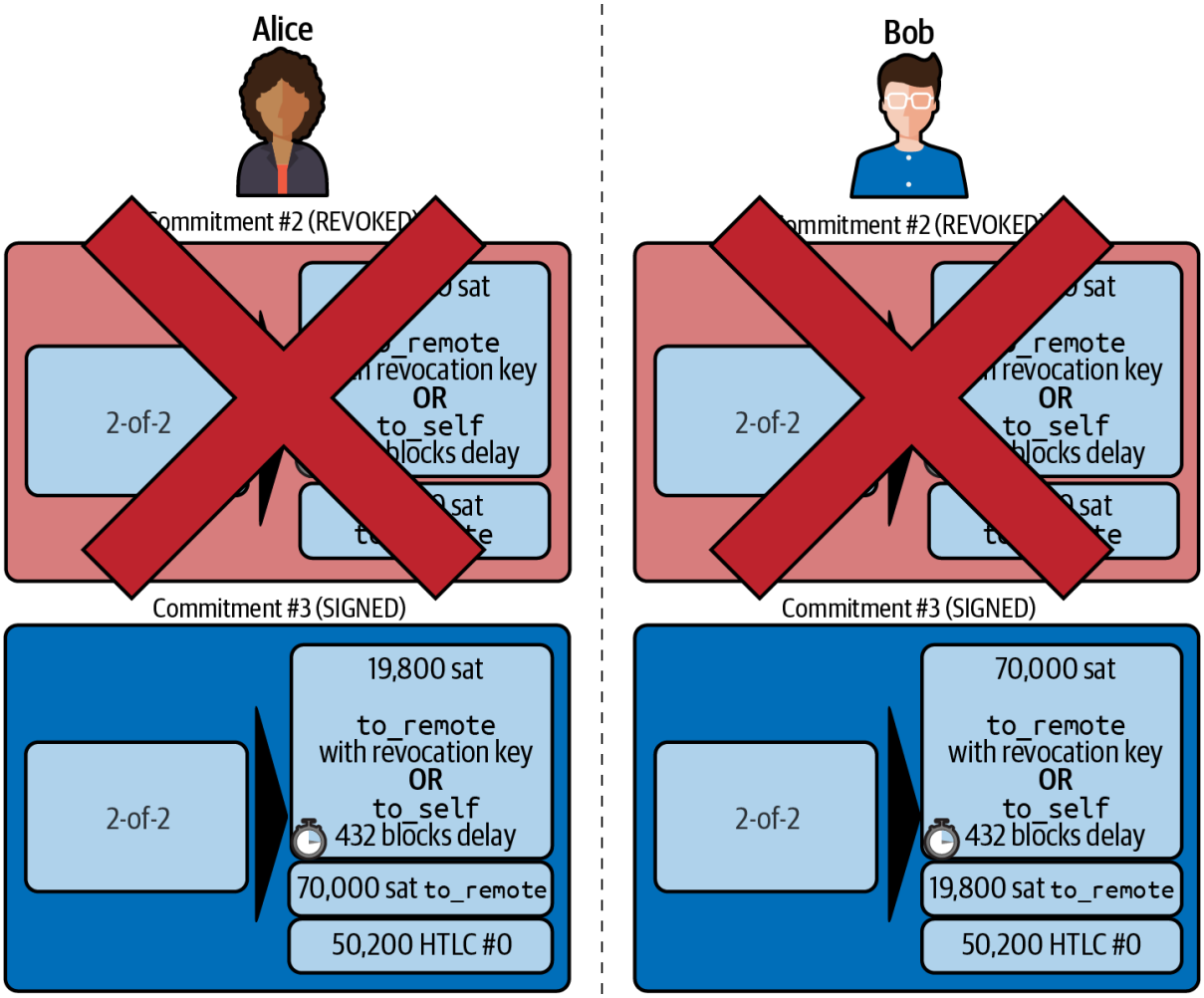

Figure 61. Alice 已撤銷舊承諾

9.4. 多個 HTLC

在任何時間點，Alice 和 Bob 在單個通道上可能有數十個甚至數百個 HTLC。每個 HTLC 作為額外輸出被提供並添加到承諾交易中。因此，承諾交易始終有兩個輸出用於通道夥伴餘額，以及任意數量的 HTLC 輸出，每個 HTLC 一個。

正如我們在 commitment_signed 訊息中看到的，有一個 HTLC 簽名陣列，以便可以同時傳輸多個 HTLC 承諾。

目前通道上允許的 HTLC 最大數量是 483 個，以考慮比特幣交易的最大大小並確保承諾交易繼續是有效的比特幣交易。

正如我們將在下一節中看到的，最大值僅適用於_待處理_的 HTLC，因為一旦 HTLC 被履行（或由於超時/錯誤而失敗），它就會從承諾交易中移除。

9.5. HTLC 履行

現在 Bob 和 Alice 都有一個帶有額外 HTLC 輸出的新承諾交易，我們已經朝著更新支付通道邁出了重要一步。

Alice 和 Bob 的新餘額尚未反映 Alice 已成功向 Bob 發送 50,200 聰。

然而，HTLC 現在的設定方式確保了交換付款證明後可以安全結算。

9.5.1. HTLC 傳播

讓我們假設 Bob 繼續這個鏈條，並與 Chan 設定一個 50,100 聰的 HTLC。過程將與我們剛才在 Alice 和 Bob 之間看到的完全相同。Bob 將向 Chan 發送 update_add_htlc，然後他們將在兩輪中交換 commitment_signed 和 revoke_and_ack 訊息，將他們的通道推進到下一個狀態。

接下來，Chan 將與 Dina 做同樣的事情：提供 50,000 聰的 HTLC，承諾和撤銷，等等。然而，Dina 是 HTLC 的最終接收者。Dina 是唯一知道付款秘密（付款雜湊的原像）的人。因此，Dina 可以立即與 Chan 履行 HTLC！

9.5.2. Dina 與 Chan 履行 HTLC

Dina 可以透過向 Chan 發送 update_fulfill_htlc 訊息來結算 HTLC。update_fulfill_htlc 訊息定義在 BOLT #2: Peer Protocol, `update_fulfill_htlc` (https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md#removing-an-htlc-update_fulfill_htlc-update_fail_htlc-and-update_fail_malformed_htlc)，如下所示：

update_fulfill_htlc 訊息

```
[channel_id:channel_id]
[u64:id]
[32*byte:payment_preimage]
```

這是一個非常簡單的訊息：

channel_id

HTLC 承諾所在的通道 ID。

id

HTLC 的 ID（我們從 0 開始，並為通道上的每個 HTLC 遞增）。

payment_preimage

證明付款已完成並兌換 HTLC 的秘密。這是 Dina 雜湊後產生 Alice 發票中付款雜湊的 R 值。

當 Chan 收到此訊息時，他會立即檢查 `payment_preimage`（讓我們稱之為 *R*）是否產生他提供給 Dina 的 HTLC 中的付款雜湊（讓我們稱之為 *H*）。他這樣雜湊它：

- *H* = RIPEMD160(SHA-256 (*R*))

如果結果 *H* 與 HTLC 中的付款雜湊匹配，Chan 可以跳一支小舞來慶祝。這個期待已久的秘密可以用來兌換 HTLC，並將沿著支付通道鏈一路傳回給 Alice，解決作為這筆付款給 Dina 一部分的每個 HTLC。

讓我們回到 Alice 和 Bob 的通道，看看他們如何解開 HTLC。為了達到那裡，讓我們假設 Dina 向 Chan 發送了 update_fulfill_htlc，Chan 向 Bob 發送了 update_fulfill_htlc，Bob 向 Alice 發送了 update_fulfill_htlc。付款原像已經一路傳回給 Alice。

9.5.3. Bob 與 Alice 結算 HTLC

當 Bob 向 Alice 發送 update_fulfill_htlc 時，它將包含 Dina 為她的發票選擇的相同 payment_preimage。該 payment_preimage 已經沿付款路徑向後傳播。在每一步，channel_id 將不同，id（HTLC ID）可能不同。但原像是相同的！

Alice 也會驗證從 Bob 收到的 payment_preimage。她會將其雜湊與她提供給 Bob 的 HTLC 中的 HTLC 付款雜湊進行比較。她還會發現這個原像與 Dina 發票中的雜湊匹配。這是 Dina 已被付款的證明。

Alice 和 Bob 之間的訊息流程如 HTLC 履行訊息流程 所示。

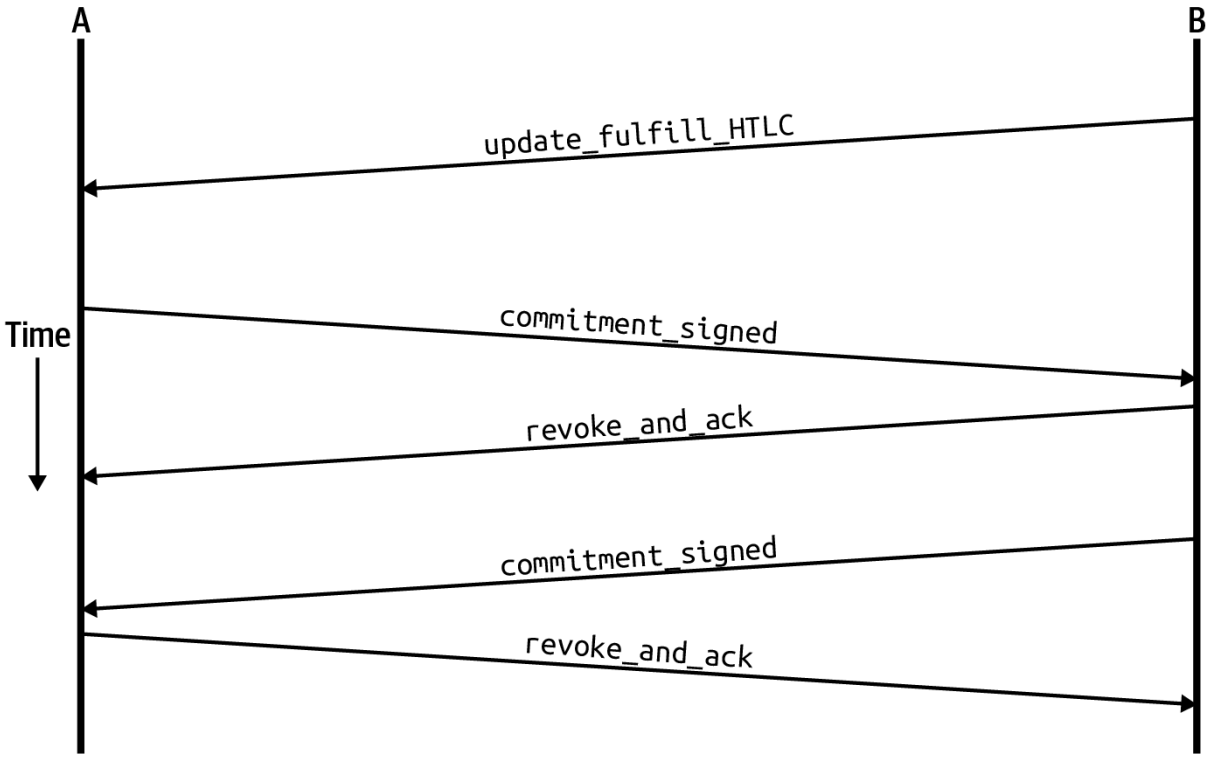

Figure 62. HTLC 履行訊息流程

Alice 和 Bob 現在都可以從承諾交易中移除 HTLC 並更新他們的通道餘額。

他們建立新的承諾（承諾 #4），如 HTLC 被移除，餘額在新承諾中更新 所示。

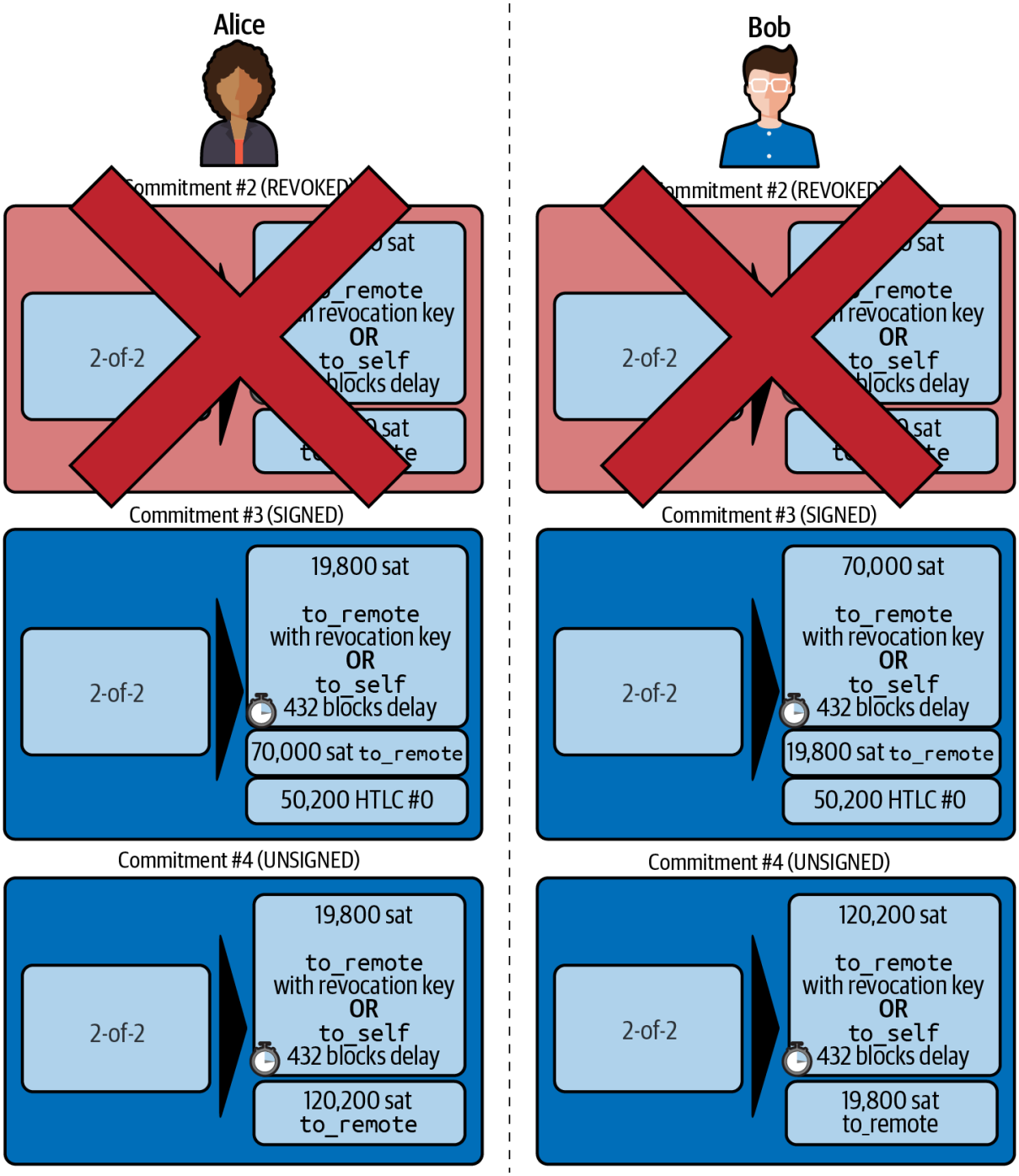

Figure 63. HTLC 被移除，餘額在新承諾中更新

接下來，他們完成兩輪承諾和撤銷。首先，Alice 發送 commitment_signed 來簽署 Bob 的新承諾交易。Bob 用 revoke_and_ack 回應以撤銷他的舊承諾。一旦 Bob 將通道狀態向前推進，承諾看起來像我們在 Alice 簽署 Bob 的新承諾，Bob 撤銷舊承諾 中看到的。

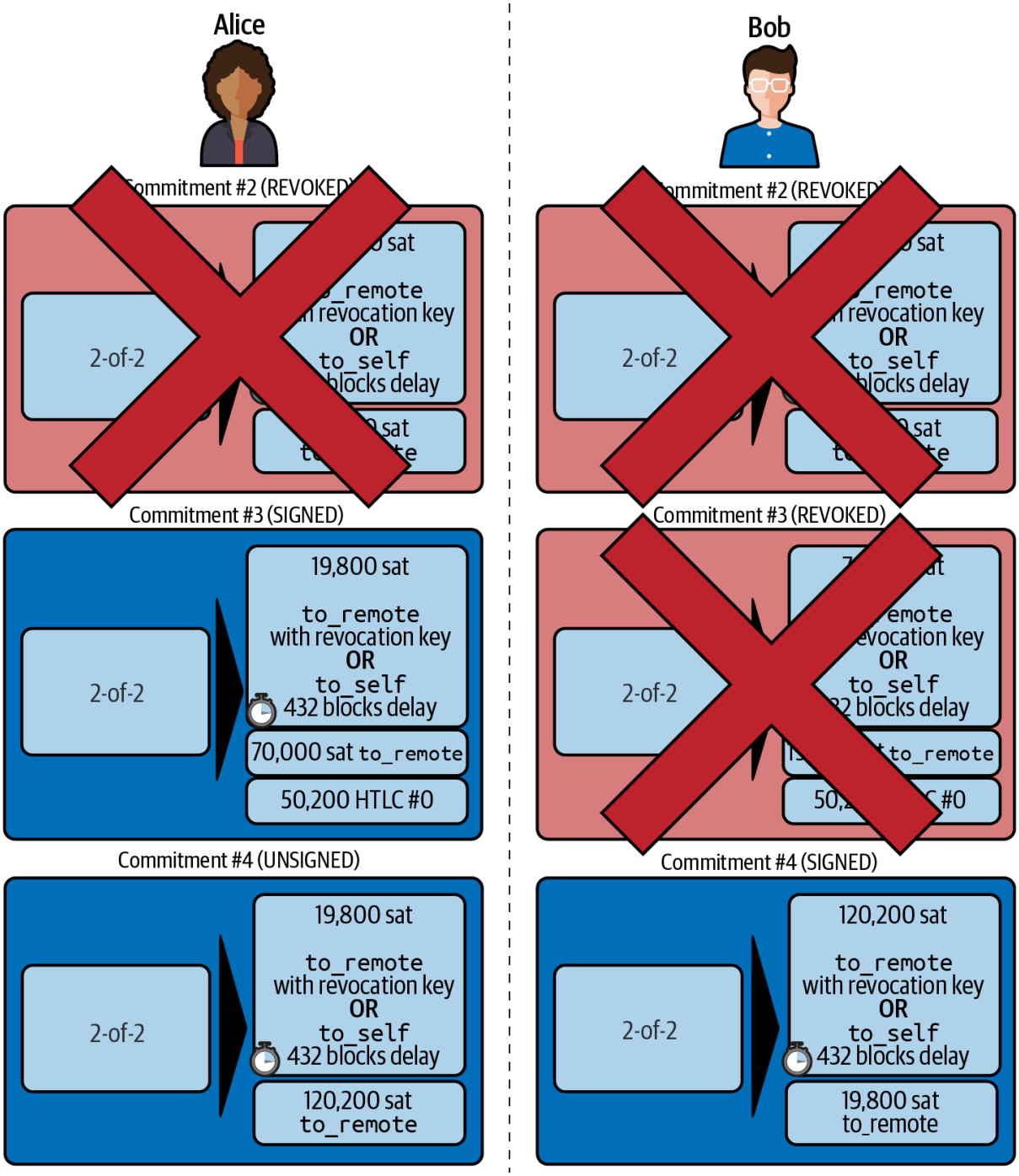

Figure 64. Alice 簽署 Bob 的新承諾，Bob 撤銷舊承諾

最後，Bob 透過向 Alice 發送 commitment_signed 訊息來簽署 Alice 的承諾。然後 Alice 透過向 Bob 發送 revoke_and_ack 來確認並撤銷她的舊承諾。最終結果是 Alice 和 Bob 都已將他們的通道狀態移動到承諾 #4，已移除 HTLC，並已更新他們的餘額。他們目前的通道狀態由 Alice 和 Bob 結算 HTLC 並更新餘額 中顯示的承諾交易表示。

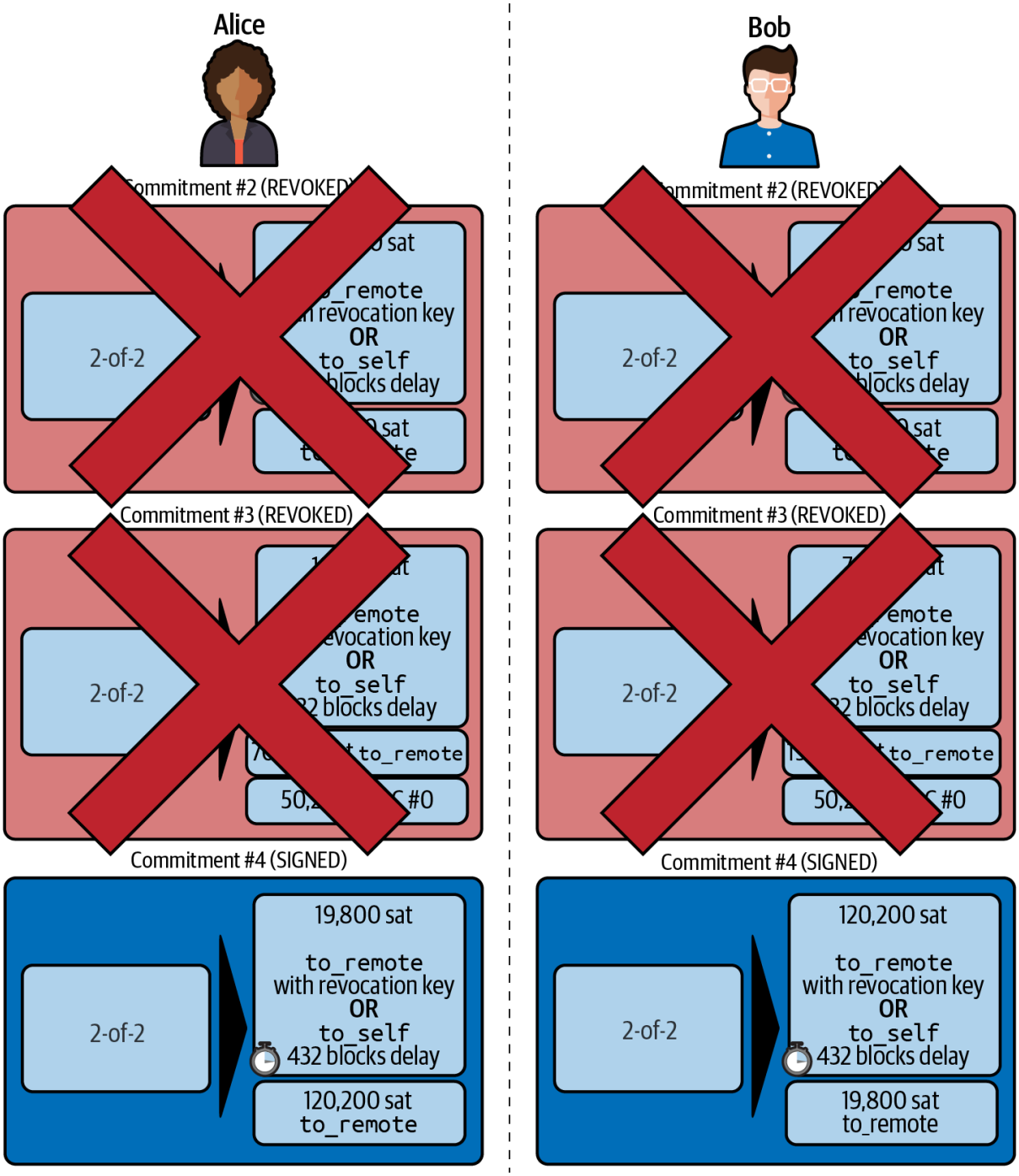

Figure 65. Alice 和 Bob 結算 HTLC 並更新餘額

9.6. 由於錯誤或過期而移除 HTLC

如果 HTLC 無法履行，可以使用相同的承諾和撤銷過程從通道承諾中移除它。

Bob 會發送 update_fail_htlc 或 update_fail_malformed_htlc，而不是 update_fulfill_htlc。這兩個訊息定義在 BOLT #2: Peer Protocol, Removing an HTLC (https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md#removing-an-htlc-update_fulfill_htlc-update_fail_htlc-and-update_fail_malformed_htlc)。

update_fail_htlc 訊息如下所示：

update_fail_htlc 訊息

```
[channel_id:channel_id]
[u64:id]
[u16:len]
[len*byte:reason]
```

這是不言自明的。多位元組的 reason 欄位定義在 BOLT #4: Onion Routing (https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#failure-messages)，我們將在 洋蔥路由 中描述。

如果 Alice 從 Bob 收到 update_fail_htlc，過程將以大致相同的方式展開：兩個通道夥伴將移除 HTLC，建立更新的承諾交易，並經過兩輪承諾/撤銷將通道狀態向前推進到下一個承諾。唯一的區別：最終餘額將恢復到沒有 HTLC 時的狀態，本質上是為 Alice 退還 HTLC 的價值。

9.7. 進行本地付款

在這一點上，你會很容易理解為什麼 HTLC 同時用於遠端和本地付款。當 Alice 為咖啡付款給 Bob 時，她不只是更新通道餘額並承諾到新狀態。相反，付款是用 HTLC 進行的，與 Alice 付款給 Dina 的方式相同。只有一個通道跳的事實沒有區別。它的工作方式如下：

1. Alice 從 Bob 的商店頁面訂購咖啡。
2. Bob 的商店發送帶有付款雜湊的發票。
3. Alice 從該付款雜湊建構一個 HTLC。
4. Alice 使用 update_add_htlc 向 Bob 提供 HTLC。
5. Alice 和 Bob 交換承諾和撤銷，將 HTLC 添加到他們的承諾交易中。
6. Bob 使用付款原像向 Alice 發送 update_fulfill_htlc。
7. Alice 和 Bob 交換承諾和撤銷，移除 HTLC 並更新通道餘額。

無論 HTLC 是跨多個通道轉發還是只在單個通道「跳」中履行，過程完全相同。

9.8. 結論

在本章中，我們看到了承諾交易（來自 支付通道）和 HTLC（來自 在支付通道網路上路由）如何協同工作。我們看到了 HTLC 如何添加到承諾交易，以及它如何被履行。我們看到了用於強制執行通道狀態的非對稱、延遲、可撤銷系統如何擴展到 HTLC。

我們還看到了本地付款和多跳路由付款如何被相同處理：使用 HTLC。

在下一章中，我們將研究稱為_洋蔥路由_的加密訊息路由系統。

10. 洋蔥路由

在本章中，我們將描述閃電網路的洋蔥路由機制。_洋蔥路由_的發明比閃電網路早了 25 年！洋蔥路由是由美國海軍研究人員作為通訊安全協定發明的。洋蔥路由最著名的用途是 Tor，這是洋蔥路由的網際網路覆蓋網路，允許研究人員、活動人士、情報人員和其他所有人私密且匿名地使用網際網路。

在本章中，我們將重點介紹閃電網路協定架構中的「基於源的洋蔥路由（SPHINX）」部分，如閃電網路協定套件中的洋蔥路由 中心（路由層）的輪廓所突顯。

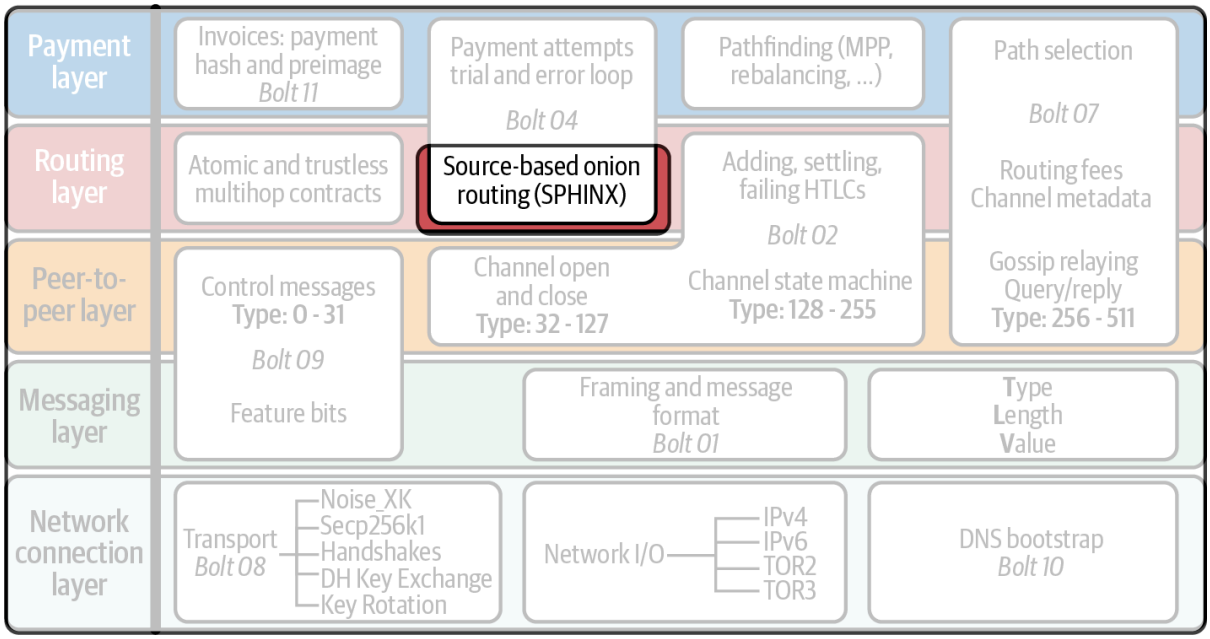

Figure 66. 閃電網路協定套件中的洋蔥路由

洋蔥路由描述了一種加密通訊方法，訊息發送者建構連續的_嵌套加密層_，由每個中間節點「剝離」，直到最內層被傳遞給預期的接收者。「洋蔥路由」這個名稱描述了這種分層加密的使用，它像洋蔥皮一樣一次剝離一層。

每個中間節點只能「剝離」一層並查看誰是通訊路徑中的下一個。洋蔥路由確保除發送者外沒有人知道目的地或通訊路徑的長度。每個中間節點只知道前一跳和下一跳。

閃電網路使用基於 *Sphinx* 的洋蔥路由協定實作，[10] 由 George Danezis 和 Ian Goldberg 於 2009 年開發。

閃電網路中洋蔥路由的實作定義在 BOLT #4: Onion Routing Protocol (https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md)。

10.1. 說明洋蔥路由的物理範例

有很多方法可以描述洋蔥路由，但最簡單的方法之一是使用密封信封的物理等效物。信封代表一層加密，只允許指定的收件人打開它並閱讀內容。

假設 Alice 想透過一些中間人間接向 Dina 發送一封秘密信件。

10.1.1. 選擇路徑

閃電網路使用_源路由_，這意味著付款路徑由發送者選擇和指定，而且只有發送者。在這個例子中，Alice 給 Dina 的秘密信件將等同於一筆付款。為了確保信件到達 Dina，Alice 將建立一條從她到 Dina 的路徑，使用 Bob 和 Chan 作為中間人。

> 可能有許多路徑使 Alice 能夠到達 Dina。我們將在 路徑尋找與付款傳遞 中解釋選擇_最佳_路徑的過程。現在，我們假設 Alice 選擇的路徑使用 Bob 和 Chan 作為到達 Dina 的中間人。

作為提醒，Alice 選擇的路徑如 路徑：Alice 到 Bob 到 Chan 到 Dina 所示。

Figure 67. 路徑：Alice 到 Bob 到 Chan 到 Dina

讓我們看看 Alice 如何使用這條路徑而不向中間人 Bob 和 Chan 透露資訊。

基於源的路由

基於源的路由不是當今網際網路上通常路由封包的方式，儘管在早期源路由是可能的。網際網路路由基於每個中間路由節點的_封包交換_。例如，IPv4 封包包括發送者和接收者的 IP 位址，每個其他 IP 路由節點決定如何將每個封包轉發到目的地。 然而，這種路由機制缺乏隱私，每個中間節點都能看到發送者和接收者，這使得它不適合用於支付網路。

10.1.2. 建構層次

Alice 首先給 Dina 寫一封秘密信件。然後她將信件密封在信封內並在外面寫上「給 Dina」（見 Dina 的秘密信件，密封在信封中）。信封代表用 Dina 的公鑰加密，只有 Dina 可以打開信封並閱讀信件。

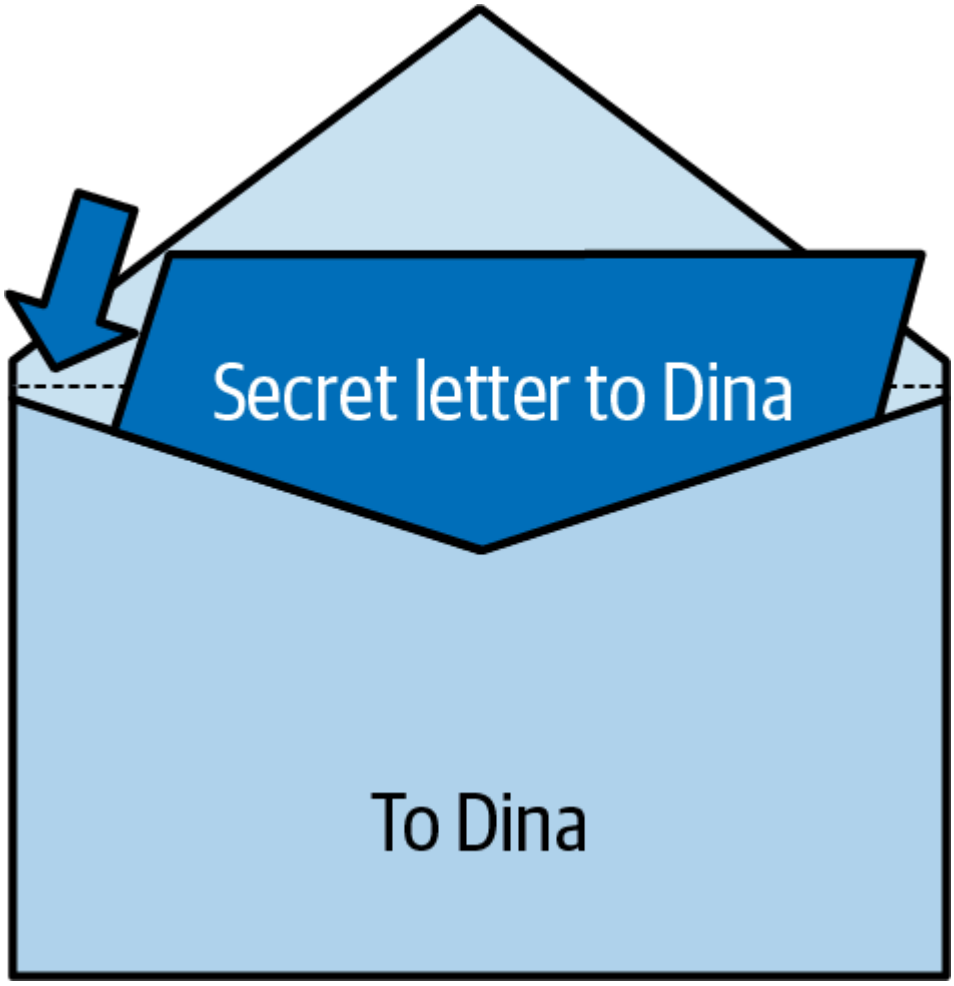

Figure 68. Dina 的秘密信件，密封在信封中

Dina 的信件將由 Chan 交給 Dina，Chan 在「路徑」中緊鄰 Dina 之前。所以，Alice 把 Dina 的信封放進一個寫給 Chan 的信封裡（見 Chan 的信封，包含 Dina 密封的信封）。Chan 唯一能讀到的是目的地（路由指示）：「給 Dina」。將此密封在寫給 Chan 的信封中代表用 Chan 的公鑰加密，只有 Chan 可以讀取信封地址。Chan 仍然無法打開 Dina 的信封。他只能看到外面的指示（地址）。

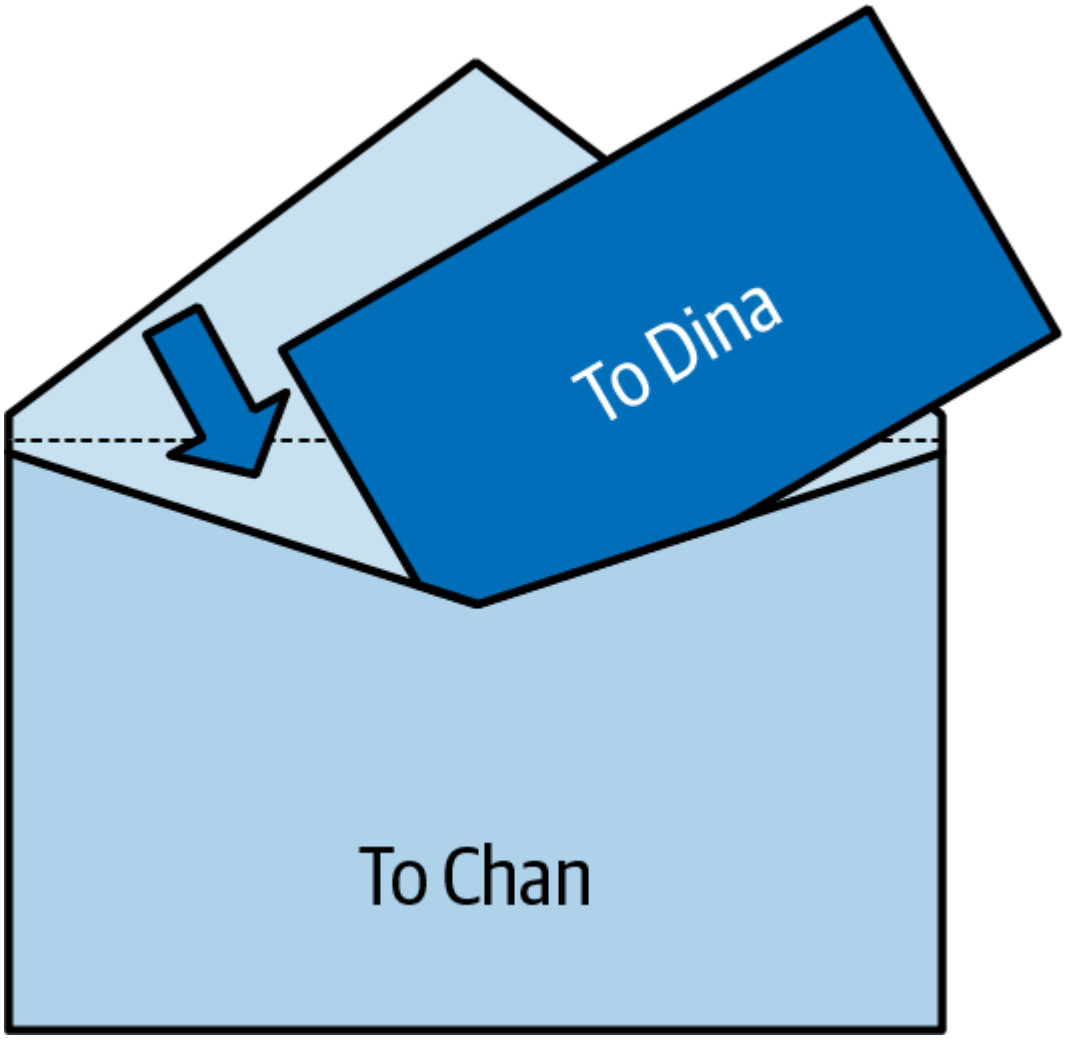

Figure 69. Chan 的信封，包含 Dina 密封的信封

現在，這封信將由 Bob 交給 Chan。所以 Alice 把它放進一個寫給 Bob 的信封裡（見 Bob 的信封，包含 Chan 密封的信封）。和以前一樣，信封代表加密給 Bob 的訊息，只有 Bob 可以讀取。Bob 只能讀取 Chan 信封的外面（地址），所以他知道要把它送給 Chan。

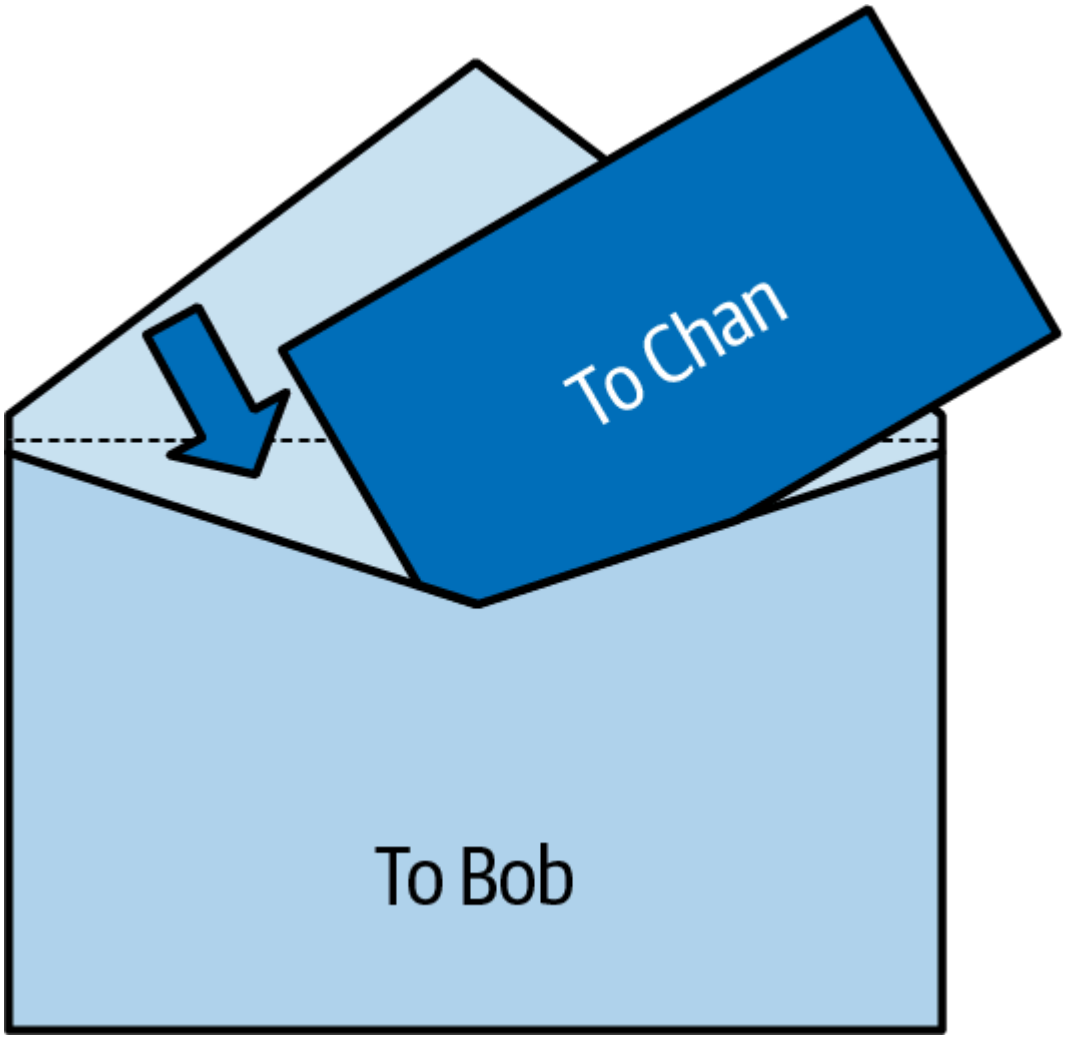

Figure 70. Bob 的信封，包含 Chan 密封的信封

現在，如果我們可以透過信封（用 X 光！）我們會看到信封一個套一個嵌套，如 嵌套的信封 所示。

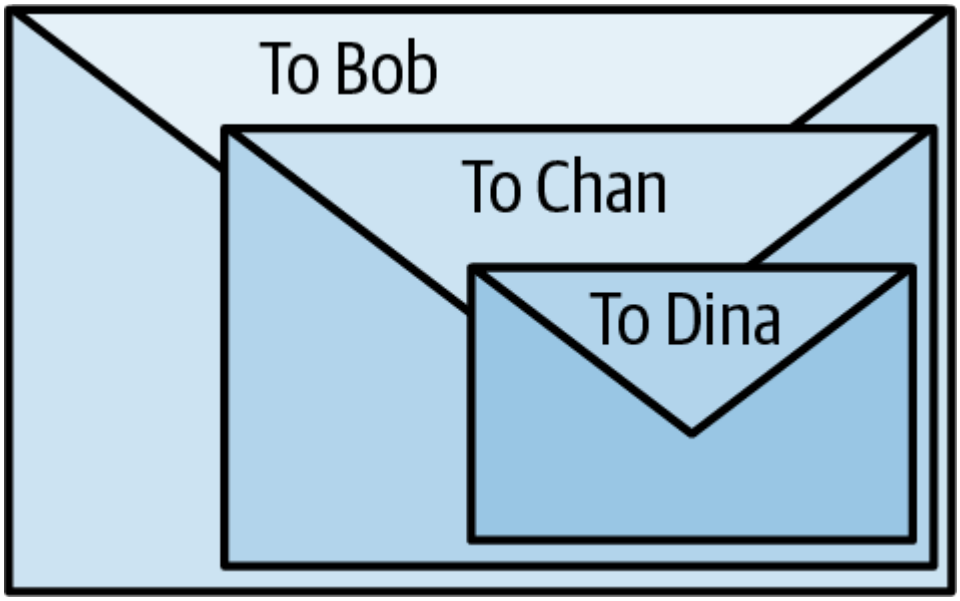

Figure 71. 嵌套的信封

10.1.3. 剝離層次

Alice 現在有一個信封，外面寫著「給 Bob」。它代表一個只有 Bob 可以打開（解密）的加密訊息。Alice 現在透過把這個發送給 Bob 來開始這個過程。整個過程如 發送信封 所示。

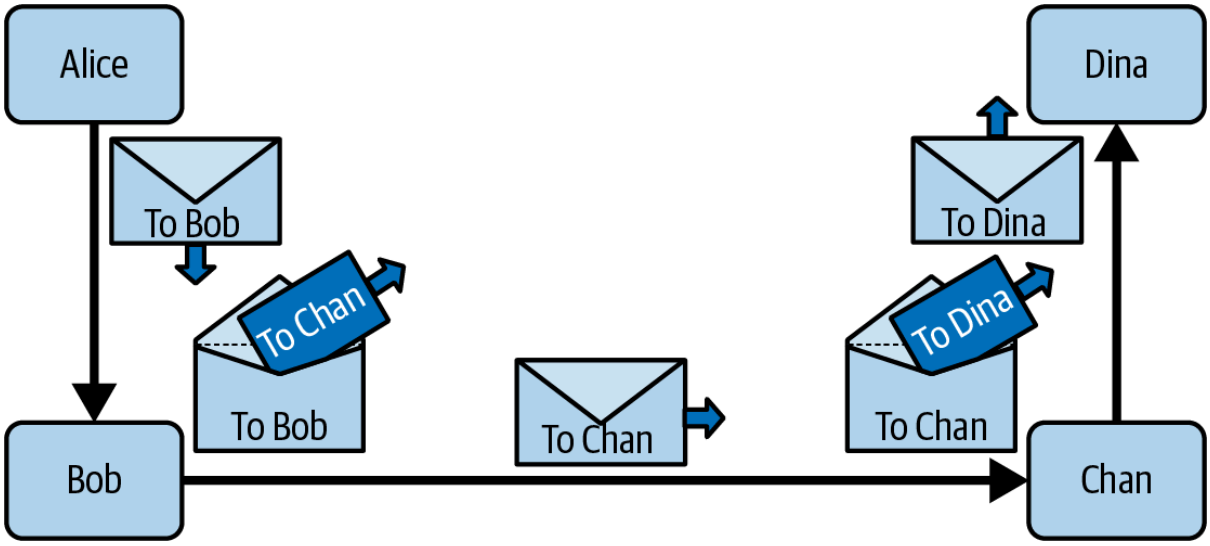

Figure 72. 發送信封

如你所見，Bob 從 Alice 那裡收到信封。他知道它來自 Alice，但不知道 Alice 是原始發送者還是只是轉發信封的人。他打開它發現裡面有一個信封寫著「給 Chan」。由於這是寫給 Chan 的，Bob 無法打開它。他不知道裡面是什麼，不知道 Chan 是收到一封信還是另一個要轉發的信封。Bob 不知道 Chan 是否是最終收件人。Bob 將信封轉發給 Chan。

Chan 從 Bob 那裡收到信封。他不知道它來自 Alice。他不知道 Bob 是中間人還是信件的發送者。Chan 打開信封發現裡面有另一個信封寫著「給 Dina」，他無法打開。Chan 將它轉發給 Dina，不知道 Dina 是否是最終收件人。

Dina 從 Chan 那裡收到一個信封。打開它她發現裡面有一封信，所以現在她知道她是這條訊息的預期收件人。她閱讀這封信，知道沒有中間人知道它從哪裡來，也沒有其他人讀過她的秘密信件！

這就是洋蔥路由的本質。發送者將訊息包裹在層次中，精確指定它將如何路由，並防止任何中間人獲得有關路徑或有效載荷的任何資訊。每個中間人剝離一層，只看到轉發地址，除了路徑中的前一跳和下一跳外什麼都不知道。

現在，讓我們看看閃電網路中洋蔥路由實作的細節。

10.2. HTLC 洋蔥路由介紹

閃電網路中的洋蔥路由乍看之下似乎很複雜，但一旦你理解了基本概念，它實際上非常簡單。

從實際角度來看，Alice 告訴每個中間節點要與路徑中的下一個節點設定什麼 HTLC。

第一個節點，即付款發送者或在我們例子中的 Alice，稱為_起源節點_。最後一個節點，即付款接收者或在我們例子中的 Dina，稱為_最終節點_。

每個中間節點，或在我們例子中的 Bob 和 Chan，稱為_跳_。每一跳必須為下一跳設定一個_出站 HTLC_。Alice 傳達給每一跳的資訊稱為_跳有效載荷_或_跳資料_。從 Alice 路由到 Dina 的訊息稱為_洋蔥_，由加密的_跳有效載荷_或_跳資料_訊息組成，這些訊息為每一跳加密。

現在我們知道了閃電網路洋蔥路由中使用的術語，讓我們重新陳述 Alice 的任務：Alice 必須用跳資料建構一個洋蔥，告訴每一跳如何建構出站 HTLC 以將付款發送到最終節點（Dina）。

10.2.1. Alice 選擇路徑

從 在支付通道網路上路由 我們知道 Alice 將透過 Bob 和 Chan 向 Dina 發送 50,000 聰的付款。此付款透過一系列 HTLC 傳輸，如 從 Alice 到 Dina 的 HTLC 付款路徑 所示。

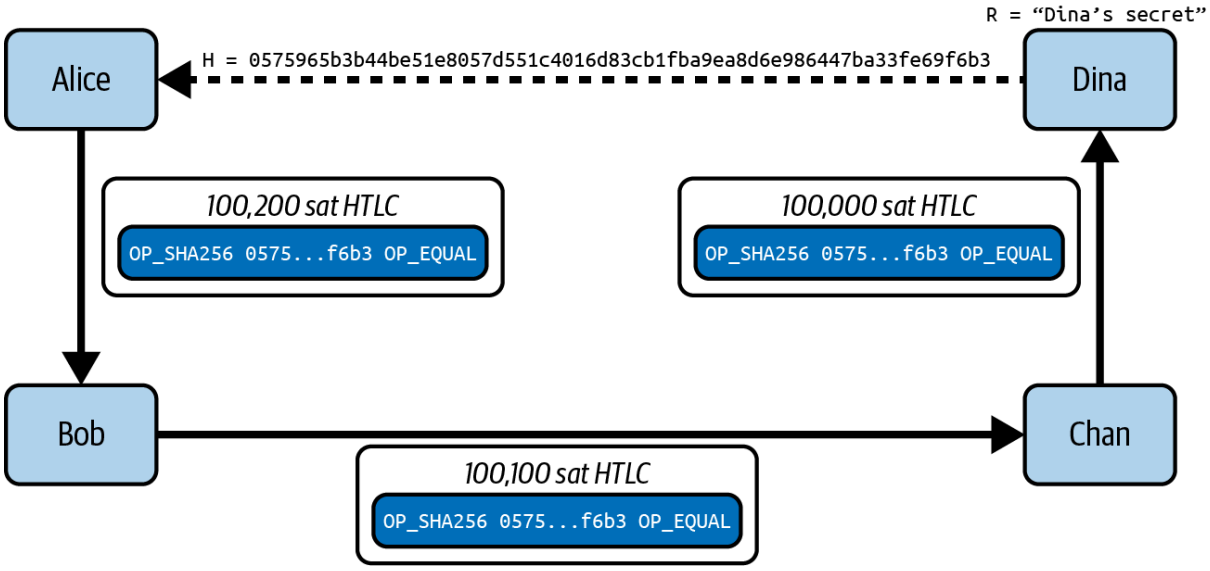

Figure 73. 從 Alice 到 Dina 的 HTLC 付款路徑

正如我們將在 八卦協定與通道圖 中看到的，Alice 能夠建構這條到 Dina 的路徑，因為閃電網路節點使用閃電網路八卦協定向整個閃電網路宣布它們的通道。在初始通道公告後，Bob 和 Chan 各自發送了額外的 `channel_update` 訊息，包含他們對付款路由的路由費和時間鎖預期。

從公告和更新中，Alice 知道關於 Bob、Chan 和 Dina 之間通道的以下資訊：

- 每個通道的 short_channel_id（短通道 ID），Alice 可以在建構路徑時使用它來參考通道
- 一個 cltv_expiry_delta（時間鎖差值），Alice 可以將其添加到每個 HTLC 的到期時間
- 一個 fee_base_msat 和 fee_proportional_millionths，Alice 可以用來計算該節點在該通道上中繼預期的總路由費。

在實踐中，還會交換其他資訊，例如通道將承載的最大（`htlc_maximum_msat`）和最小（`htlc_minimum_msat`）HTLC，但這些在洋蔥路由建構期間不像前面的欄位那樣直接使用。

Alice 使用此資訊來識別以下詳細路徑的節點、通道、費用和時間鎖，如 從八卦的通道和節點資訊建構的詳細路徑 所示。

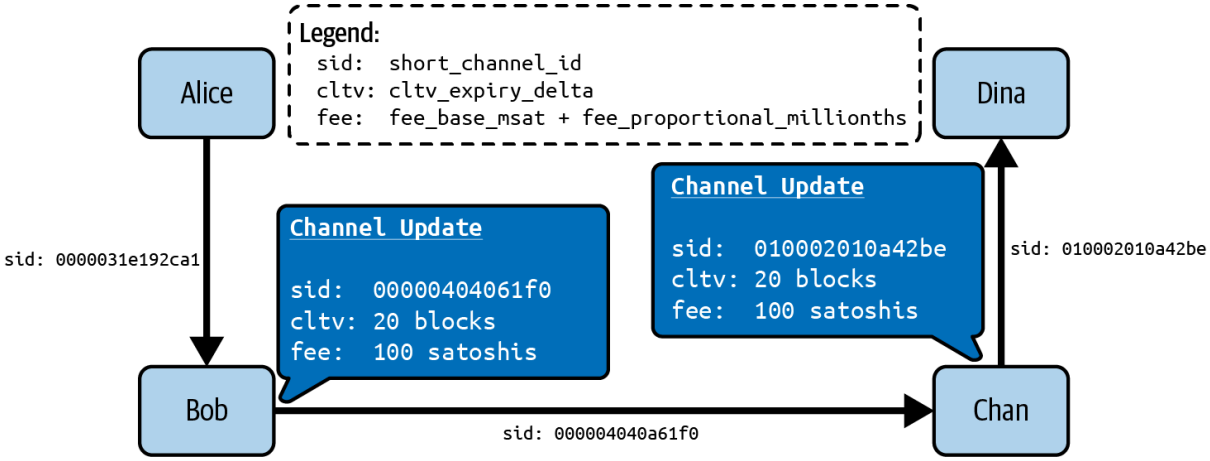

Figure 74. 從八卦的通道和節點資訊建構的詳細路徑

Alice 已經知道她自己到 Bob 的通道，因此不需要此資訊來建構路徑。另請注意，Alice 不需要 Dina 的通道更新，因為她有 Chan 對該路徑中最後一個通道的更新。

10.2.2. Alice 建構有效載荷

Alice 可以使用兩種可能的格式來傳達給每一跳的資訊：一種稱為_跳資料_的遺留固定長度格式和一種更靈活的基於類型-長度-值（TLV）的格式稱為_跳有效載荷_。TLV 訊息格式在 類型-長度-值格式 中有更詳細的解釋。它透過允許隨意向協定添加欄位來提供靈活性。

這兩種格式都在 BOLT #4: Onion Routing Protocol, Packet Structure (https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#packet-structure) 中指定。

Alice 將從路徑的末端向後建構跳資料：Dina、Chan，然後是 Bob。

Dina 的最終節點有效載荷

Alice 首先建構將傳遞給 Dina 的有效載荷。Dina 不會建構「出站 HTLC」，因為 Dina 是最終節點和付款接收者。因此，Dina 的有效載荷與所有其他的不同（short_channel_id 使用全零），但只有 Dina 會知道這一點，因為它將在洋蔥的最內層被加密。本質上，這就是我們在物理信封範例中看到的「給 Dina 的秘密信件」。

Dina 的跳有效載荷必須匹配 Dina 為 Alice 生成的發票中的資訊，並將包含（至少）以下 TLV 格式的欄位：

amt_to_forward

此付款的金額，以毫聰為單位。如果這只是多部分付款的一部分，金額小於總額。否則，這是單筆完整付款，它等於發票金額和 total_msat 值。

outgoing_cltv_value

付款到期時間鎖設定為發票中的值 min_final_cltv_expiry。

payment_secret

發票中的特殊 256 位元秘密值，允許 Dina 識別此傳入付款。這也防止了一類探測，這些探測以前使零值發票不安全。中間節點的探測被減輕，因為此值_只_對接收者加密，意味著他們無法重建一個看起來「合法」的最終封包。

total_msat

與發票匹配的總金額。如果只有一部分，這可以省略，在這種情況下假設它匹配 amt_to_forward 並且必須等於發票金額。

Alice 從 Dina 收到的發票指定金額為 50,000 聰，即 50,000,000 毫聰。Dina 指定付款的最低到期時間 min_final_cltv_expiry 為 18 個區塊（3 小時，假設比特幣區塊平均 10 分鐘）。在 Alice 嘗試付款時，假設比特幣區塊鏈已記錄 700,000 個區塊。所以 Alice 必須將 outgoing_cltv_value 設定為_最低_ 700,018 個區塊的區塊高度。

Alice 如下建構 Dina 的跳有效載荷：

```
amt_to_forward : 50,000,000
outgoing_cltv_value: 700,018
payment_secret: fb53d94b7b65580f75b98f10...03521bdab6d519143cd521d1b3826
total_msat: 50,000,000
```

Alice 將其序列化為 TLV 格式，如（簡化的）Dina 的有效載荷由 Alice 建構 所示。

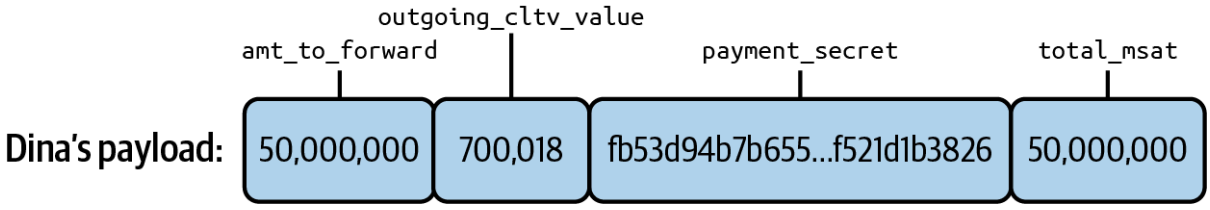

Figure 75. Dina 的有效載荷由 Alice 建構

Chan 的跳有效載荷

接下來，Alice 為 Chan 建構跳有效載荷。這將告訴 Chan 如何為 Dina 設定出站 HTLC。

Chan 的跳有效載荷包含三個欄位：short_channel_id、amt_to_forward 和 outgoing_cltv_value：

```
short_channel_id: 010002010a42be
amt_to_forward: 50,000,000
outgoing_cltv_value: 700,018
```

Alice 將此有效載荷序列化為 TLV 格式，如（簡化的）Chan 的有效載荷由 Alice 建構 所示。

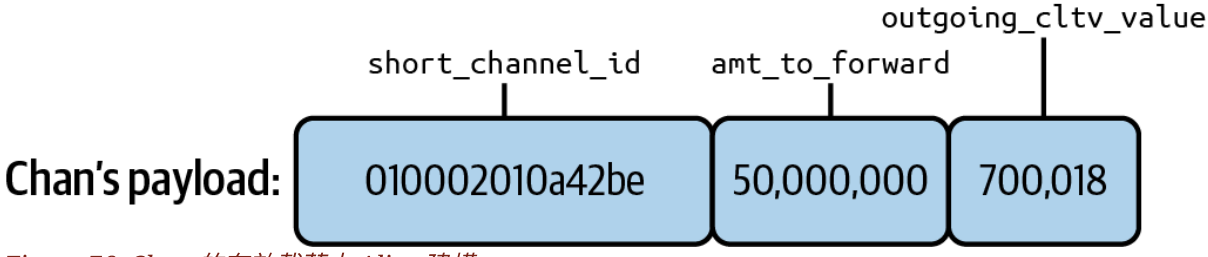

Figure 76. Chan 的有效載荷由 Alice 建構

Bob 的跳有效載荷

最後，Alice 為 Bob 建構跳有效載荷，它也包含與 Chan 的跳有效載荷相同的三個欄位，但值不同：

```
short_channel_id: 000004040a61f0
amt_to_forward: 50,100,000
outgoing_cltv_value: 700,038
```

如你所見，amt_to_forward 欄位是 50,100,000 毫聰，或 50,100 聰。這是因為 Chan 期望收取 100 聰的費用來將付款路由到 Dina。為了讓 Chan「賺取」該路由費，Chan 的入站 HTLC 必須比 Chan 的出站 HTLC 多 100 聰。由於 Chan 的入站 HTLC 是 Bob 的出站 HTLC，給 Bob 的指示反映了 Chan 賺取的費用。簡單來說，Bob 需要被告知向 Chan 發送 50,100 聰，這樣 Chan 可以發送 50,000 聰並保留 100 聰。

同樣，Chan 期望 20 個區塊的時間鎖差值。所以 Chan 的入站 HTLC 必須比 Chan 的出站 HTLC 晚 20 個區塊到期。為了實現這一點，Alice 告訴 Bob 使他的出站 HTLC 給 Chan 在區塊高度 700,038 到期——比 Chan 給 Dina 的 HTLC 晚 20 個區塊。

> 通道的費用和時間鎖差值預期由入站和出站 HTLC 之間的差異設定。由於入站 HTLC 由_前一個節點_建立，費用和時間鎖差值設定在給該前一個節點的洋蔥有效載荷中。Bob 被告知如何製作一個滿足 Chan 費用和時間鎖預期的 HTLC。

Alice 將此有效載荷序列化為 TLV 格式，如（簡化的）Bob 的有效載荷由 Alice 建構 所示。

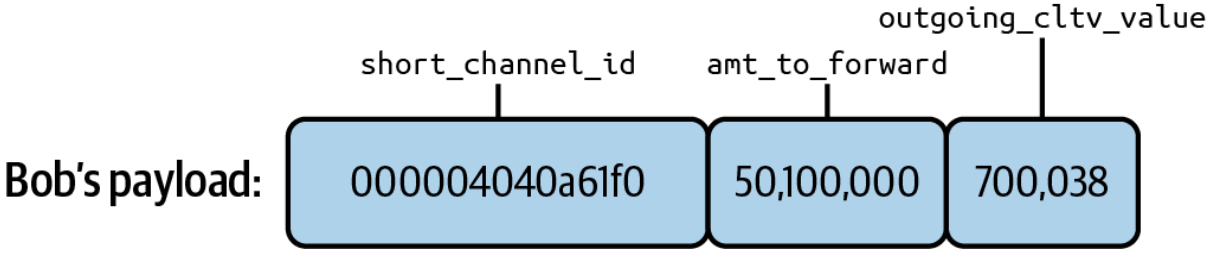

Figure 77. Bob 的有效載荷由 Alice 建構

完成的跳有效載荷

Alice 現在已經建構了將包裹在洋蔥中的三個跳有效載荷。有效載荷的簡化視圖如 所有跳的跳有效載荷 所示。

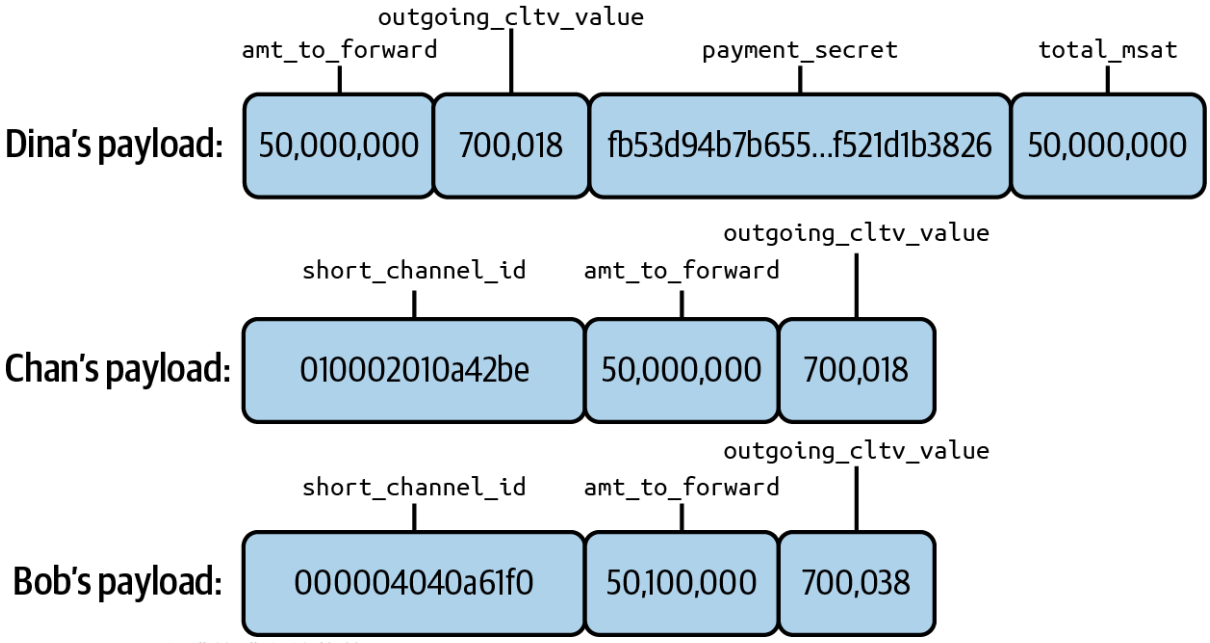

Figure 78. 所有跳的跳有效載荷

10.2.3. 金鑰生成

Alice 現在必須生成幾個金鑰，用於加密洋蔥中的各個層次。

使用這些金鑰，Alice 可以實現高度的隱私和完整性：

- Alice 可以加密洋蔥的每一層，使只有預期的接收者可以讀取它。
- 每個中間人都可以檢查訊息是否被修改。
- 路徑中沒有人會知道誰發送了這個洋蔥或它要去哪裡。Alice 不會透露她作為發送者的身份或 Dina 作為付款接收者的身份。
- 每一跳只了解前一跳和下一跳。
- 沒有人能知道路徑有多長，或者他們在路徑中的位置。

> 像切好的洋蔥一樣，以下技術細節可能會讓你流淚。如果你感到困惑，可以跳到下一節。如果你想了解更多，回來閱讀這部分和 [BOLT #4: Onion Routing, Packet Construction (https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#packet-construction)]。

洋蔥中使用的所有金鑰的基礎是 Alice 和 Bob 都可以使用橢圓曲線 Diffie–Hellman（ECDH）演算法獨立生成的_共享秘密_。從共享秘密（ss）中，他們可以獨立生成四個名為 rho、mu、um 和 pad 的附加金鑰：

rho

用於從串流加密器生成隨機位元組流（用作 CSPRNG）。這些位元組用於在 Sphinx 封包處理期間加密/解密訊息正文以及填充零位元組。

mu

用於基於雜湊的訊息認證碼（HMAC）進行完整性/真實性驗證。

um

用於錯誤報告。

pad

用於生成填充洋蔥到固定長度的填充位元組。

各種金鑰之間的關係以及它們如何生成如 洋蔥金鑰生成 所示。

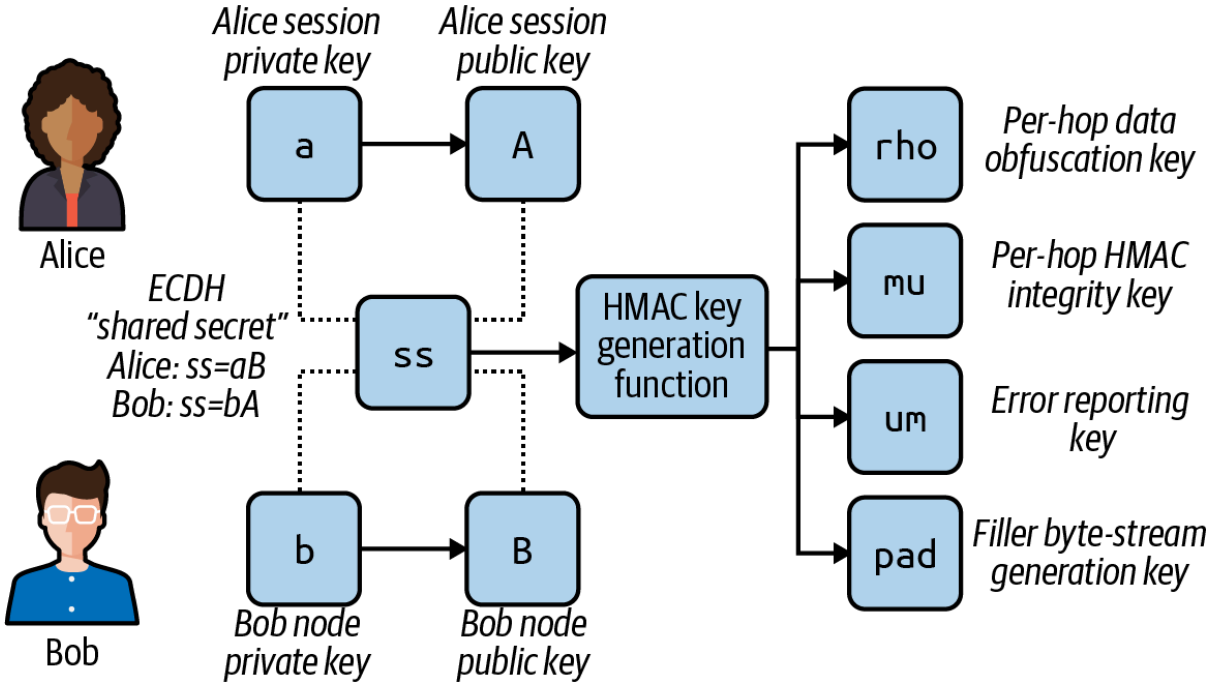

Figure 79. 洋蔥金鑰生成

Alice 的會話金鑰

為了避免透露她的身份，Alice 不使用她自己節點的公鑰來建構洋蔥。相反，Alice 建立一個臨時的 32 位元組（256 位元）金鑰，稱為_會話私鑰_和對應的_會話公鑰_。這用作_僅用於此洋蔥_的臨時「身份」和金鑰。從這個會話金鑰，Alice 將建構此洋蔥中將使用的所有其他金鑰。

金鑰生成細節

金鑰生成、隨機位元組生成、臨時金鑰以及它們在封包建構中的使用方式在 BOLT #4 的三個部分中指定：

- Key Generation (https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#key-generation)

- Random Byte Stream (https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#pseudo-random-byte-stream)
- Packet Construction (https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#packet-construction)

為了簡單起見，避免過於技術化，我們沒有在書中包含這些細節。如果你想了解內部運作，請參見前面的連結。

共享秘密生成

一個看起來幾乎神奇的重要細節是 Alice 只需知道另一個節點的公鑰就能與其建立_共享秘密_的能力。這基於 1970 年代發明的 Diffie–Hellman 金鑰交換（DH），它徹底改變了密碼學。閃電網路洋蔥路由在比特幣的 secp256k1 曲線上使用橢圓曲線 Diffie–Hellman（ECDH）。這是一個很酷的技巧，我們嘗試在 橢圓曲線 Diffie–Hellman 解釋 中用簡單的術語解釋它。

橢圓曲線 Diffie–Hellman 解釋

假設 Alice 的私鑰是 a，Bob 的私鑰是 b。使用橢圓曲線，Alice 和 Bob 各自將他們的私鑰乘以生成點 G 以產生他們各自的公鑰 A 和 B：

- $A = aG$
- $B = bG$

現在 Alice 和 Bob 可以使用_橢圓曲線 Diffie–Hellman 金鑰交換_來建立共享秘密 ss，這是一個他們都可以獨立計算而無需交換任何資訊的值。

共享秘密 ss 由每個人透過將他們自己的私鑰乘以_對方的_公鑰來計算，使得：

- $ss = aB = bA$

但為什麼這兩個乘法會產生相同的值 ss？ 跟著我們，我們將證明這是可能的數學：

- ss
- $= aB$

由知道 a（她的私鑰）和 B（Bob 的公鑰）的 Alice 計算

- $= a(bG)$

因為我們知道 $B = bG$，我們代入

- $= (ab)G$

因為結合律，我們可以移動括號

- = (*ba*)*G*

因為 *xy* = *yx*（曲線是阿貝爾群）

- = *b*(*aG*)

因為結合律，我們可以移動括號

- = *bA*

我們可以用 *A* 代替 *aG*。

結果 *bA* 可以由知道 *b*（他的私鑰）和 *A*（Alice 的公鑰）的 Bob 獨立計算。

因此我們已經證明：

- *ss* = *aB*（Alice 可以計算這個）
- *ss* = *bA*（Bob 可以計算這個）

因此，他們可以各自獨立計算 *ss*，然後將其用作共享金鑰來對稱加密他們之間的秘密，而無需通訊共享秘密。

Sphinx 作為混合網路封包格式的一個獨特特性是，不是為路由中的每一跳包含一個獨特的會話金鑰，這會大大增加混合網路封包的大小，而是使用一種巧妙的_盲化_方案在每一跳確定性地隨機化會話金鑰。

在實踐中，這個小技巧允許我們保持洋蔥封包盡可能緊湊，同時仍然保留所需的安全屬性。

跳 `i` 的會話金鑰使用節點公鑰和跳 `i - 1` 的派生共享秘密來派生：

```
session_key_i = session_key_{i-1} * SHA-256(node_pubkey_{i-1} ||
shared_secret_{i-1})
```

換句話說，我們取前一跳的會話金鑰，並將其乘以從該跳的公鑰和派生共享秘密派生的值。

由於橢圓曲線乘法可以在不知道私鑰的情況下對公鑰執行，每一跳都能夠以確定性的方式為下一跳重新隨機化會話金鑰。

洋蔥封包的建立者知道所有的共享秘密（因為他們為每一跳唯一地加密了封包），因此能夠預先派生所有的盲化因子。

這些知識允許他們在封包生成期間預先派生使用的所有會話金鑰。

請注意，第一跳使用最初生成的原始會話金鑰，因為此金鑰用於啟動每個後續跳的會話金鑰盲化。

10.3. 包裹洋蔥層

包裹洋蔥的過程詳見 BOLT #4: Onion Routing, Packet Construction (https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#packet-construction)
。

在本節中，我們將在較高且有些簡化的層面描述此過程，省略某些細節。

10.3.1. 固定長度洋蔥

我們已經提到，「跳」節點都不知道路徑有多長，或者他們在路徑中的位置。這怎麼可能？

如果你有一組方向，即使是加密的，難道不能透過查看你在方向列表中的_位置_來判斷你離開始或結束有多遠嗎？

洋蔥路由中使用的技巧是始終為每個節點使路徑（方向列表）長度相同。這透過在每一步保持洋蔥封包相同的長度來實現。

在每一跳，跳有效載荷出現在洋蔥有效載荷的開頭，後面跟著_看起來像是_ 19 個更多的跳有效載荷。每一跳都將自己視為 20 跳中的第一跳。

> 洋蔥有效載荷是 1,300 位元組。每個跳有效載荷是 65 位元組或更少（如果更少則填充到 65 位元組）。所以總洋蔥有效載荷可以容納 20 個跳有效載荷（1300 = 20 × 65）。因此，最大洋蔥路由路徑是 20 跳。

當每一層被「剝離」時，會在洋蔥有效載荷的末尾添加更多填充資料（本質上是垃圾），這樣下一跳得到相同大小的洋蔥，並且再次成為洋蔥中的「第一跳」。

洋蔥大小是 1,366 位元組，結構如 洋蔥封包 所示：

1 位元組

版本位元組

33 位元組

壓縮的公開會話金鑰（Alice 的會話金鑰），每一跳可以從中生成共享秘密（共享秘密生成）而不會透露 Alice 的身份

1,300 位元組

實際的_洋蔥有效載荷_，包含每一跳的指示

32 位元組

HMAC 完整性校驗和

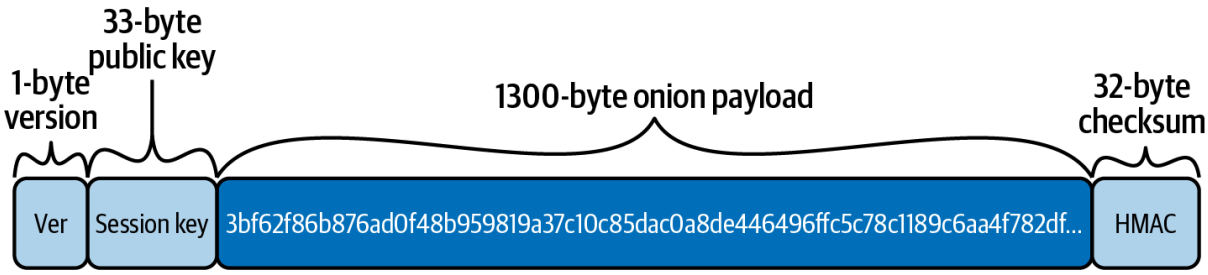

Figure 80. 洋蔥封包

Sphinx 作為混合網路封包格式的一個獨特特性是，不是為路由中的每一跳包含一個獨特的會話金鑰，這會大大增加混合網路封包的大小，而是使用一種巧妙的_盲化_方案在每一跳確定性地隨機化會話金鑰。

在實踐中，這個小技巧允許我們保持洋蔥封包盡可能緊湊，同時仍然保留所需的安全屬性。

10.3.2. 包裹洋蔥（概述）

這是包裹洋蔥的過程，接下來概述。當我們用真實世界的例子探索每一步時，回到這個列表。

對於每一跳，發送者（Alice）重複相同的過程：

1. Alice 生成每跳共享秘密和 rho、mu 和 pad 金鑰。
2. Alice 生成 1,300 位元組的填充並用此填充填滿 1,300 位元組的洋蔥有效載荷欄位。
3. Alice 計算跳有效載荷的 HMAC（最終跳為零）。
4. Alice 計算跳有效載荷 + HMAC + 儲存長度本身所需空間的長度。
5. Alice 將洋蔥有效載荷_右移_計算出的容納跳有效載荷所需的空間。最右邊的「填充」資料被丟棄，在左邊為有效載荷騰出足夠的空間。
6. Alice 在從移動填充騰出的空間的有效載荷欄位前面插入長度 + 跳有效載荷 + HMAC。
7. Alice 使用 rho 金鑰生成 1,300 位元組的一次性填充。
8. Alice 透過與從 rho 生成的位元組進行 XOR 來混淆整個洋蔥有效載荷。
9. Alice 使用 mu 金鑰計算洋蔥有效載荷的 HMAC。
10. Alice 添加會話公鑰（以便跳可以計算共享秘密）。
11. Alice 添加版本號。
12. Alice 使用透過雜湊共享秘密和前一跳公鑰派生的值確定性地重新盲化會話金鑰。

接下來，Alice 重複這個過程。計算新金鑰，洋蔥有效載荷被移動（丟棄更多垃圾），新的跳有效載荷被添加到前面，整個洋蔥有效載荷用下一跳的 rho 位元組流加密。

對於最終跳，步驟 #3 中包含的明文指示上的 HMAC 實際上_全為零_。 最終跳使用此信號來確定它確實是路由中的最終跳。 或者，有效載荷中包含的表示「下一跳」的 `short_chan_id` 全為零也可以使用。

請注意，在每個階段，mu 金鑰用於在_加密的_（從處理有效載荷的節點的角度）洋蔥封包上生成 HMAC，以及在移除單層加密的封包內容上生成 HMAC。 這個外部 HMAC 允許處理封包的節點驗證洋蔥封包的完整性（沒有位元組被修改）。 內部 HMAC 然後在前面描述的「移動和加密」例程的逆過程中被揭示，這作為下一跳的_外部_ HMAC。

10.3.3. 包裹 Dina 的跳有效載荷

作為提醒，洋蔥是從路徑的末端開始包裹的，即 Dina，最終節點或接收者。然後路徑反向建構一直回到發送者 Alice。

Alice 從一個空的 1,300 位元組欄位開始，即固定長度的_洋蔥有效載荷_。然後，她用從 pad 金鑰生成的偽隨機位元組流「填充」填充洋蔥有效載荷。

這如 用隨機位元組流填充洋蔥有效載荷 所示。

隨機位元組流生成使用 ChaCha20 演算法，作為密碼學安全的偽隨機數產生器（CSPRNG）。這種演算法將從初始種子生成確定性的、長的、不重複的看似隨機的位元組流。細節在 BOLT #4: Onion Routing, Pseudo Random Byte Stream (https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#pseudo-random-byte-stream) 中指定。

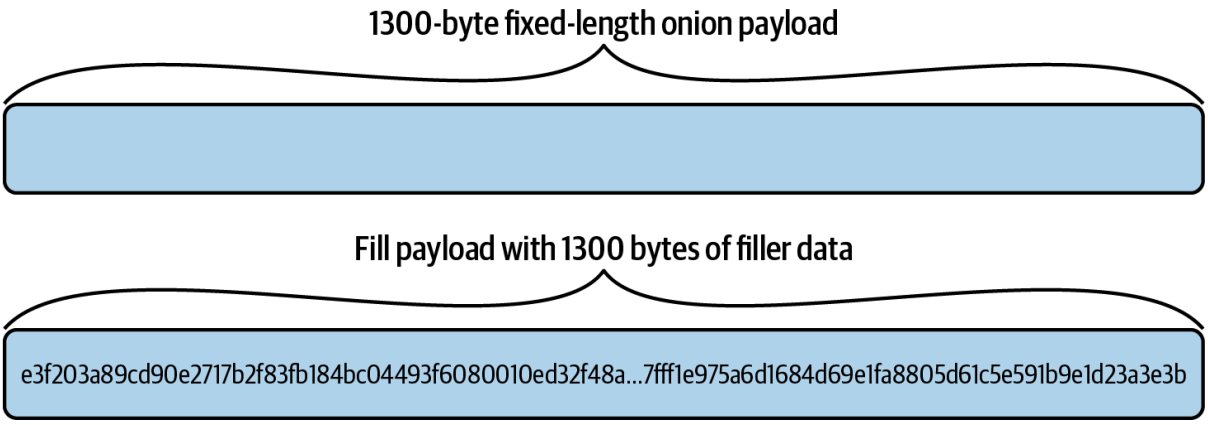

Figure 81. 用隨機位元組流填充洋蔥有效載荷

Alice 現在將 Dina 的跳有效載荷插入 1,300 位元組陣列的左側，將填充向右移動並丟棄任何溢出的內容。這在 添加 Dina 的跳有效載荷 中視覺化。

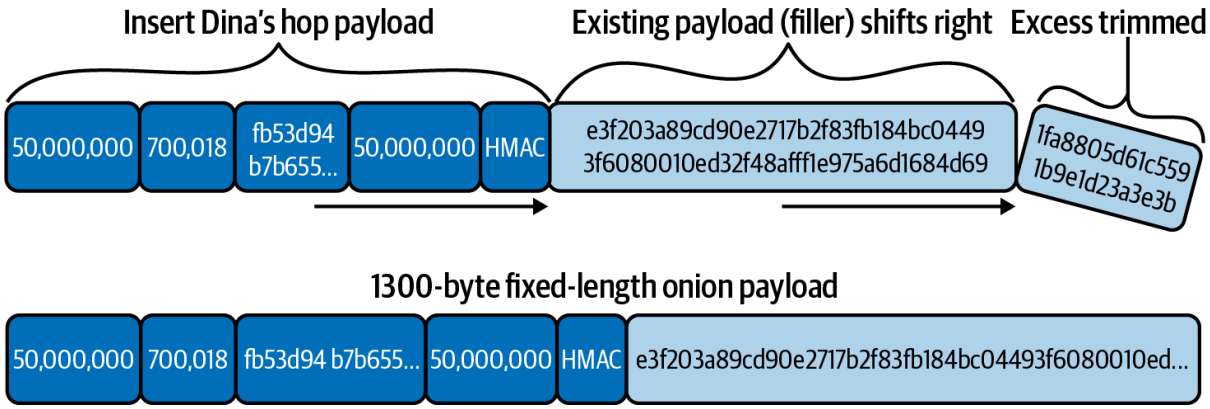

Figure 82. 添加 Dina 的跳有效載荷

另一種看法是 Alice 測量 Dina 跳有效載荷的長度，將填充向右移動以在洋蔥有效載荷的左側建立相等的空間，並將 Dina 的有效載荷插入該空間。

向下一行我們看到結果：1,300 位元組的洋蔥有效載荷包含 Dina 的跳有效載荷，然後是填充位元組流填充其餘空間。

接下來，Alice 混淆整個洋蔥有效載荷，使得_只有 Dina_可以讀取它。

為此，Alice 使用 rho 金鑰生成一個位元組流（Dina 也知道這個）。Alice 在洋蔥有效載荷的位元和從 rho 建立的位元組流之間使用按位互斥或（XOR）。結果看起來像一個 1,300 位元組長度的隨機（或加密的）位元組流。這一步如 混淆洋蔥有效載荷 所示。

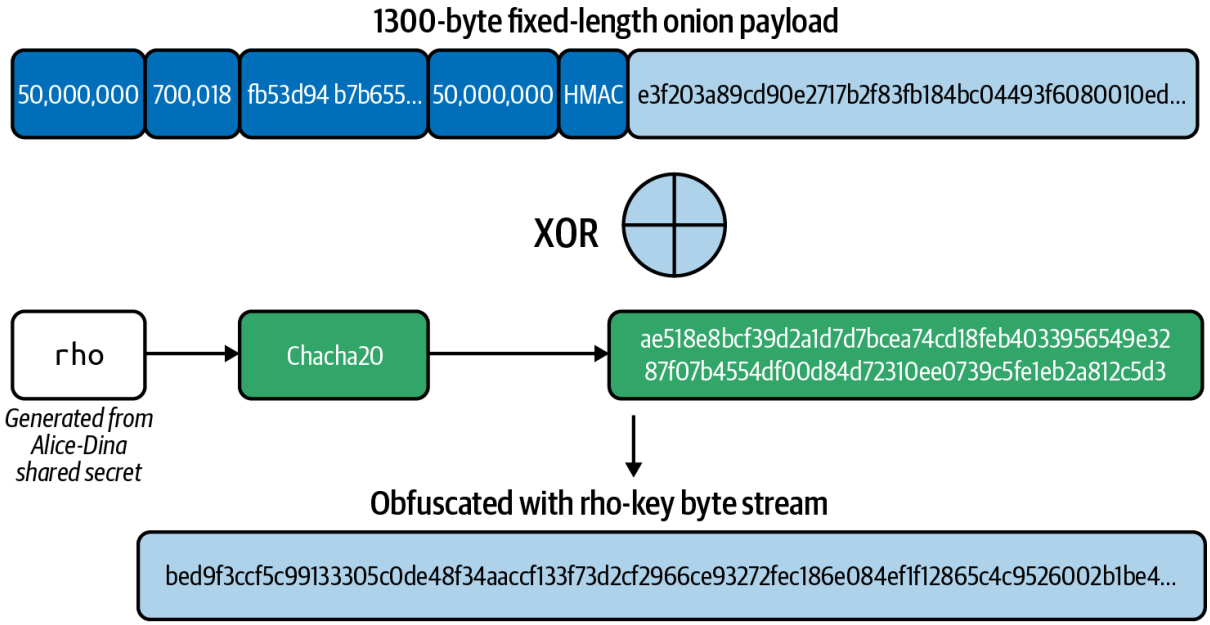

Figure 83. 混淆洋蔥有效載荷

XOR 的一個屬性是，如果你做兩次，你會得到原始資料。正如我們將在 Bob 去混淆他的跳有效載荷 中看到的，如果 Dina 使用從 rho 生成的位元組流應用相同的 XOR 操作，它將揭示原始洋蔥有效載荷。

XOR 是一個_對合_函數，意味著如果應用兩次，它會撤銷自身。具體來說 XOR(XOR(*a*, *b*), *b*) = *a*。此屬性在對稱金鑰密碼學中被廣泛使用。

因為只有 Alice 和 Dina 有 rho 金鑰（從 Alice 和 Dina 的共享秘密派生），只有他們可以做到這一點。實際上，這為 Dina 的眼睛加密了洋蔥有效載荷。

最後，Alice 為 Dina 的有效載荷計算基於雜湊的訊息認證碼（HMAC），使用 mu 金鑰作為其初始化金鑰。這如 為 Dina 的跳有效載荷添加 HMAC 完整性校驗和 所示。

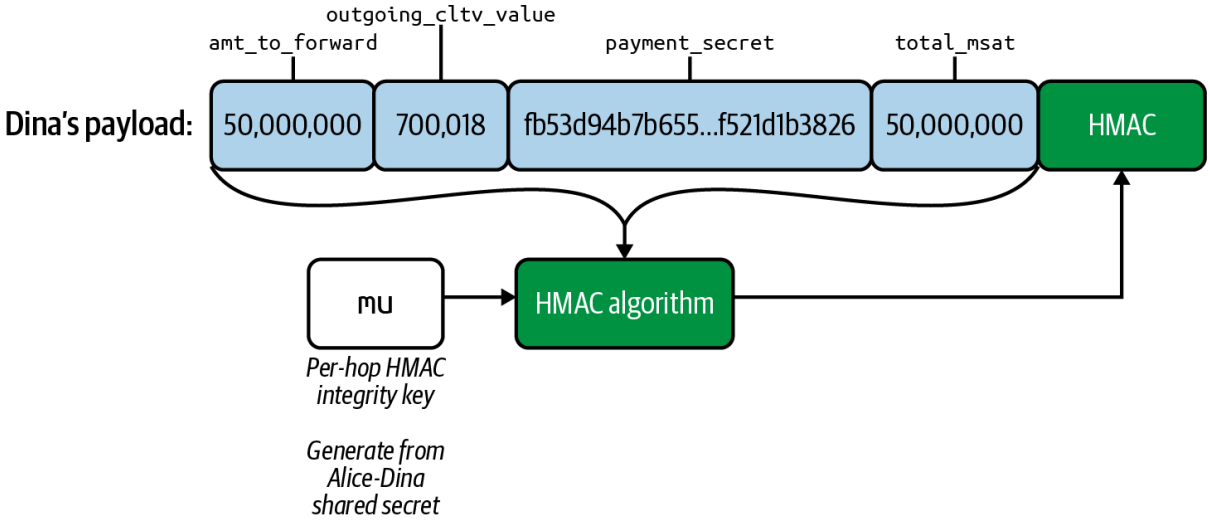

Figure 84. 為 Dina 的跳有效載荷添加 HMAC 完整性校驗和

洋蔥路由重播保護和檢測

HMAC 作為安全校驗和，幫助 Dina 驗證跳有效載荷的完整性。32 位元組的 HMAC 附加到 Dina 的跳有效載荷。 請注意，我們是在_加密_資料上計算 HMAC，而不是在明文資料上。 這稱為_先加密後 MAC_，是使用 MAC 的推薦方式，因為它提供明文_和_密文完整性。

現代認證加密還允許使用一組可選的明文位元組進行認證，稱為_關聯資料_。 在實踐中，這通常是像明文封包標頭或其他輔助資訊。 透過在要認證（MAC）的有效載荷中包含此關聯資料，MAC 的驗證者確保此關聯資料沒有被篡改（例如，交換加密封包上的明文標頭）。

在閃電網路的背景下，此關聯資料用於_加強_此方案的重播保護。 正如我們將在下面了解的，重播保護確保攻擊者不能_重新傳輸_（重播）封包到網路並觀察其產生的路徑。 相反，中間節點能夠使用定義的重播保護措施來檢測和拒絕重播的封包。 基本 Sphinx 封包格式使用所有用過的臨時秘密金鑰的日誌來檢測重播。 如果秘密金鑰再次使用，節點可以檢測到它並拒絕封包。

閃電網路中 HTLC 的性質允許我們透過添加額外的_經濟_激勵來進一步加強重播保護。 記住，HTLC 的付款雜湊只能安全地使用（用於完整付款）一次。 如果付款雜湊再次使用並穿過已經看過該雜湊的付款秘密的節點，那麼他們可以簡單地提取資金並收取整個付款金額而不轉發！

我們可以利用這一事實來加強重播保護，要求_付款雜湊_作為關聯資料包含在我們的 HMAC 計算中。 有了這個添加的步驟，嘗試重播洋蔥封包還要求發送者承諾使用_相同的_付款雜湊。 因此，除了正常的重播保護之外，攻擊者還可能失去重播的 HTLC 的全部金額。

隨著為重播保護儲存的會話金鑰集不斷增加的一個考慮是：節點能夠回收這個空間嗎？ 在閃電網路的背景下，答案是：是的！ 再次，由於 HTLC 構造的獨特屬性，我們可以對基本 Sphinx 協定進行進一步改進。 鑑於 HTLC 是基於絕對區塊高度的_時間鎖定_合約，一旦 HTLC 過期，合約就永久關閉。 因此，節點可以使用此 CLTV（CHECKLOCKTIMEVERIFY 運算子）到期高度作為指標，以知道何時可以安全地垃圾收集反重播日誌中的條目。

10.3.4. 包裹 Chan 的跳有效載荷

在 為 Chan 包裹洋蔥 中，我們看到用於將 Chan 的跳有效載荷包裹在洋蔥中的步驟。這些是 Alice 用來包裹 Dina 跳有效載荷的相同步驟。

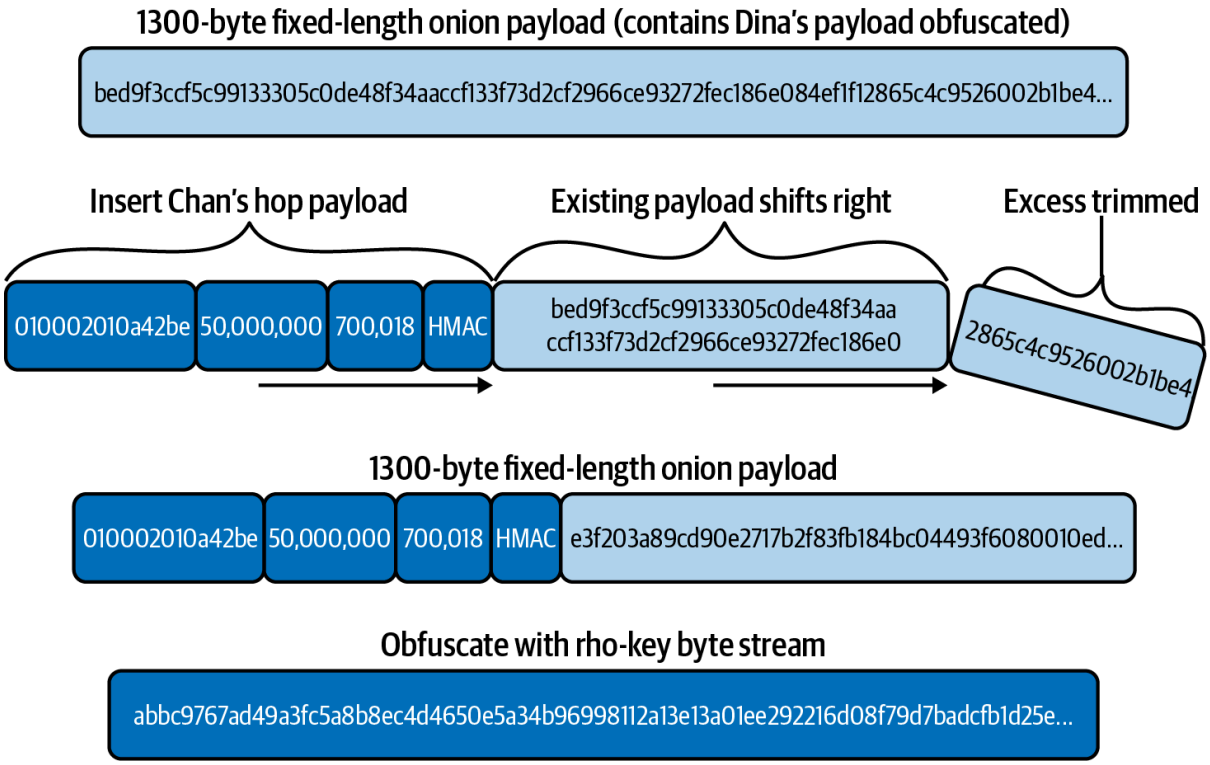

Figure 85. 為 Chan 包裹洋蔥

Alice 從為 Dina 建立的 1,300 洋蔥有效載荷開始。前 65（或更少）位元組是 Dina 的混淆有效載荷，其餘是填充。Alice 必須小心不要覆蓋 Dina 的有效載荷。

接下來，Alice 需要找到將在此跳前置到路由封包的臨時公鑰（在最開始為每一跳生成）。

記住，Sphinx 不是為每一跳包含一個唯一的臨時公鑰（發送者和中間節點在 ECDH 操作中使用它來生成共享秘密），而是使用單個臨時公鑰，在每一跳確定性地隨機化。

在處理封包時，Dina 將使用她的共享秘密和公鑰來派生盲化因子（`b_dina`），並使用它來重新隨機化臨時公鑰，與 Alice 在初始封包建構期間執行的操作相同。

Alice 為 Chan 的有效載荷添加內部 HMAC 校驗和，並將其插入洋蔥有效載荷的「前面」（左側），將現有有效載荷向右移動相同的量。 記住，該方案中實際上使用了_兩個_ HMAC：外部 HMAC 和內部 HMAC。 在這種情況下，Chan 的_內部_ HMAC 實際上是 Dina 的_外部_ HMAC。

現在 Chan 的有效載荷在洋蔥的前面。當 Chan 看到這個時，他不知道之前或之後有多少有效載荷。它看起來總是像 20 跳中的第一跳！

接下來，Alice 用從 Alice-Chan rho 金鑰生成的位元組流 XOR 混淆整個有效載荷。只有 Alice 和 Chan 有這個 rho 金鑰，只有他們可以產生位元組流來混淆和去混淆洋蔥。 最後，正如我們在前面的步驟中所做的，我們計算 Chan 的外部 HMAC，這是她用來驗證加密洋蔥封包完整性的。

10.3.5. 包裹 Bob 的跳有效載荷

在 為 Bob 包裹洋蔥 中，我們看到用於將 Bob 的跳有效載荷包裹在洋蔥中的步驟。

好了，現在這很容易了！

從包含 Chan 和 Dina 跳有效載荷的洋蔥有效載荷（混淆的）開始。

獲取從前一跳生成的盲化因子派生的此跳的會話金鑰。 將前一跳的外部 HMAC 作為此跳的內部 HMAC 包含。 在開頭插入 Bob 的跳有效載荷，並將其他所有內容向右移動，從末尾丟棄一個 Bob 跳有效載荷大小的塊（反正是填充）。

用來自 Alice-Bob 共享秘密的 rho 金鑰 XOR 混淆整個內容，使得只有 Bob 可以解開這個。

計算外部 HMAC 並將其貼在 Bob 跳有效載荷的末尾。

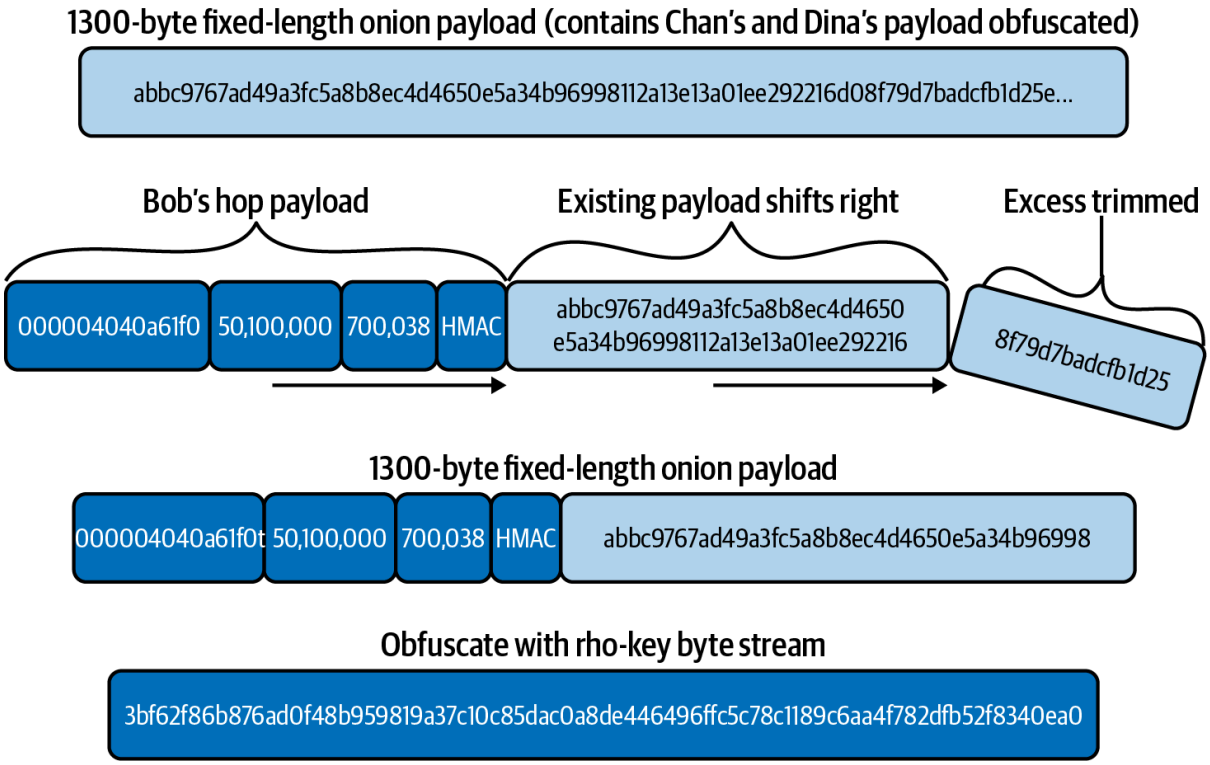

Figure 86. 為 Bob 包裹洋蔥

10.3.6. 最終洋蔥封包

最終洋蔥有效載荷準備好發送給 Bob 了。Alice 不需要再添加任何跳有效載荷。

Alice 為洋蔥有效載荷計算 HMAC，用校驗和以密碼學方式保護它，Bob 可以驗證。

Alice 添加一個 33 位元組的公開會話金鑰，每一跳將使用它來生成共享秘密和 rho、mu 和 pad 金鑰。

最後 Alice 在前面放置洋蔥版本號（目前為 0）。這允許洋蔥封包格式的未來升級。

結果可以在 洋蔥封包 中看到。

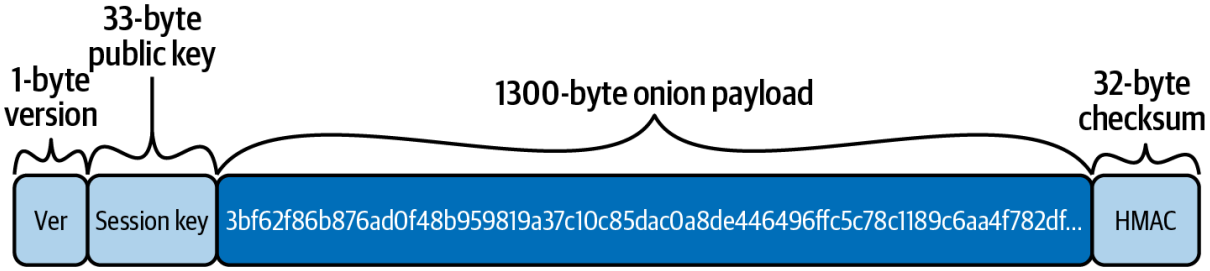

Figure 87. 洋蔥封包

10.4. 發送洋蔥

在本節中，我們將研究洋蔥封包如何轉發以及 HTLC 如何沿路徑部署。

10.4.1. update_add_htlc 訊息

洋蔥封包作為 update_add_htlc 訊息的一部分發送。如果你回憶一下 update_add_HTLC 訊息，在 通道操作與支付轉發 中，我們看到 update_add_htlc 訊息的內容如下：

```
[channel_id:channel_id]
[u64:id]
[u64:amount_msat]
[sha256:payment_hash]
[u32:cltv_expiry]
[1366*byte:onion_routing_packet]
```

你會記得這個訊息是由一個通道夥伴發送給另一個通道夥伴，要求他們添加 HTLC。這就是 Alice 將如何要求 Bob 添加 HTLC 來付款給 Dina。現在你理解了最後一個欄位 onion_routing_packet 的目的，它是 1,366 位元組長。它是我們剛剛建構的完全包裹的洋蔥封包！

10.4.2. Alice 向 Bob 發送洋蔥

Alice 將向 Bob 發送 update_add_htlc 訊息。讓我們看看這個訊息將包含什麼：

channel_id

此欄位包含 Alice-Bob 通道 ID，在我們的例子中是 0000031e192ca1（見 從八卦的通道和節點資訊建構的詳細路徑）。

id

此通道中此 HTLC 的 ID，從 0 開始。

amount_msat

HTLC 的金額：50,200,000 毫聰。

payment_hash

RIPEMD160(SHA-256) 付款雜湊：

9e017f6767971ed7cea17f98528d5f5c0ccb2c71。

cltv_expiry

HTLC 的到期時間鎖將是 700,058。Alice 根據 Bob 協商的 cltv_expiry_delta 將 20 個區塊添加到 Bob 有效載荷中設定的到期時間。

onion_routing_packet

Alice 建構的包含所有跳有效載荷的最終洋蔥封包！

10.4.3. Bob 檢查洋蔥

正如我們在 通道操作與支付轉發 中看到的，Bob 將把 HTLC 添加到承諾交易中，並與 Alice 更新通道的狀態。

Bob 將如下解開他從 Alice 收到的洋蔥：

1. Bob 從洋蔥封包中取出會話金鑰並派生 Alice-Bob 共享秘密。
2. Bob 從共享秘密生成 mu 金鑰，並使用它來驗證洋蔥封包 HMAC 校驗和。

現在 Bob 已經生成了共享金鑰並驗證了 HMAC，他可以開始解開洋蔥封包內的 1,300 位元組洋蔥有效載荷。目標是讓 Bob 檢索自己的跳有效載荷，然後將剩餘的洋蔥轉發到下一跳。

如果 Bob 提取並移除他的跳有效載荷，剩餘的洋蔥將不是 1,300 位元組，它會更短！所以下一跳會知道他們不是第一跳，並能夠檢測路徑有多長。為了防止這種情況，Bob 需要添加更多填充來重新填充洋蔥。

10.4.4. Bob 生成填充

Bob 以與 Alice 略有不同但遵循相同一般原則的方式生成填充。

首先，Bob _擴展_洋蔥有效載荷 1,300 位元組，並用 0 值填充它們。現在洋蔥封包是 2,600 位元組長，前半部分包含 Alice 發送的資料，後半部分包含零。這個操作如 Bob 將洋蔥有效載荷擴展 1,300（零填充）位元組 所示。

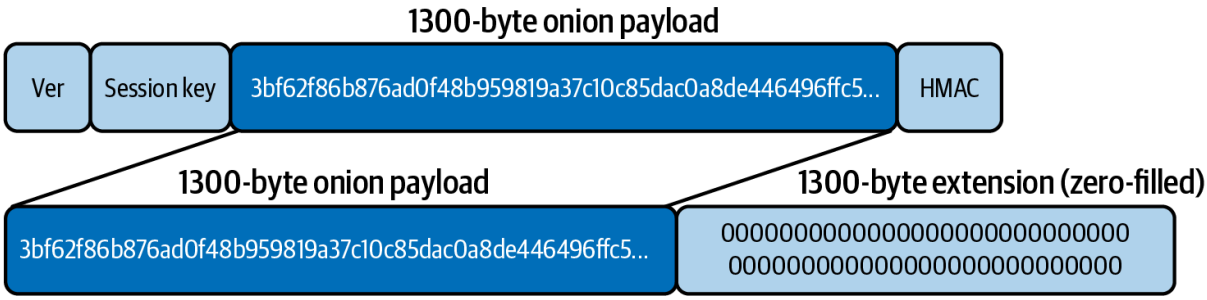

Figure 88. Bob 將洋蔥有效載荷擴展 1,300（零填充）位元組

這個空白空間將被混淆並透過 Bob 用來去混淆他自己跳有效載荷的相同過程變成「填充」。讓我們看看這是如何運作的。

10.4.5. Bob 去混淆他的跳有效載荷

接下來，Bob 將從 Alice-Bob 共享金鑰生成 rho 金鑰。他將使用 ChaCha20 演算法用這個金鑰生成 2,600 位元組流。

Bob 使用 rho 金鑰生成的位元組流的前 1,300 位元組與 Alice 使用 rho 金鑰生成的完全相同。

接下來，Bob 用按位 XOR 操作將 2,600 位元組的 rho 位元組流應用到 2,600 位元組的洋蔥有效載荷。

前 1,300 位元組將透過此 XOR 操作被去混淆，因為它是 Alice 應用的相同操作，而 XOR 是對合的。所以 Bob 將_揭示_他的跳有效載荷，後面跟著一些看起來被打亂的資料。

同時，將 rho 位元組流應用到添加到洋蔥有效載荷的 1,300 個零將把它們變成看似隨機的填充資料。這個操作如 Bob 去混淆洋蔥，混淆填充 所示。

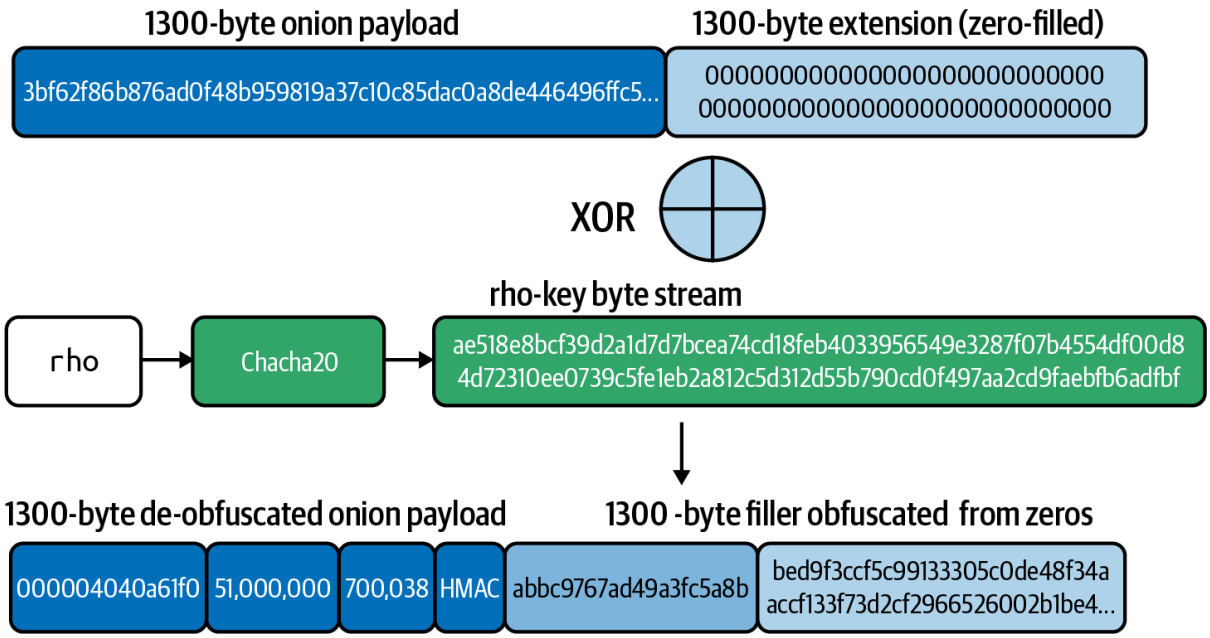

Figure 89. Bob 去混淆洋蔥，混淆填充

10.4.6. Bob 提取下一跳的外部 HMAC

記住，每一跳都包含一個內部 HMAC，然後它將成為_下一跳_的外部 HMAC。 在這種情況下，Bob 提取內部 HMAC（他已經用外部 HMAC 驗證了加密封包的完整性），並將其放在一邊，因為他將把它附加到去混淆的封包上，以允許 Chan 驗證他加密封包的 HMAC。

10.4.7. Bob 移除他的有效載荷並左移洋蔥

現在 Bob 可以從洋蔥前面移除他的跳有效載荷，並左移剩餘的資料。來自後半 1,300 位元組填充的等於 Bob 跳有效載荷的資料量現在將移入洋蔥有效載荷空間。這如 Bob 移除跳有效載荷並左移其餘部分，用新填充填補空白 所示。

現在 Bob 可以保留前半 1,300 位元組，並丟棄擴展的（填充）1,300 位元組。

Bob 現在有一個 1,300 位元組的洋蔥封包要發送到下一跳。它幾乎與 Alice 為 Chan 建立的洋蔥有效載荷相同，除了最後 65 位元組左右的填充是 Bob 放的，會有所不同。

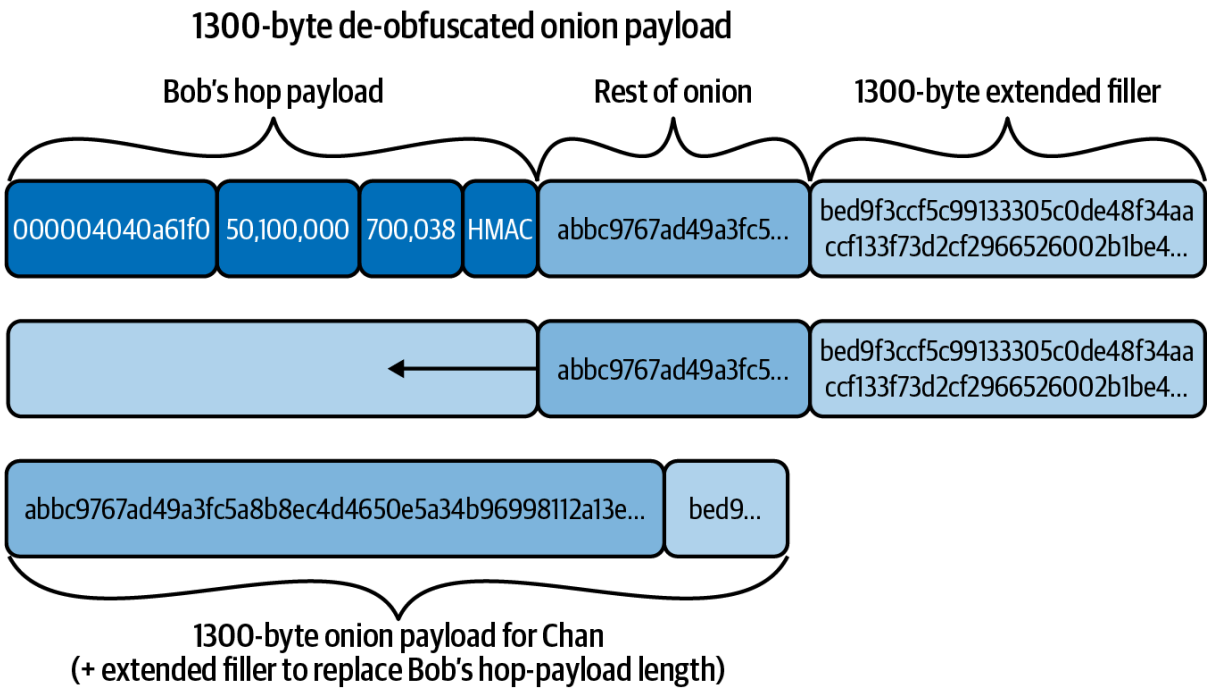

Figure 90. Bob 移除跳有效載荷並左移其餘部分，用新填充填補空白

沒有人能分辨 Alice 放的填充和 Bob 放的填充之間的區別。填充就是填充！反正都是隨機位元組。請注意，如果 Bob（或 Bob 的其他別名之一）在路由中出現在兩個不同的位置，那麼他們可以分辨區別，因為基本協定總是在整個路由中使用相同的付款雜湊。原子多路徑付款（AMP）和點時間鎖定合約（PTLC）透過在每個路由/跳上隨機化付款識別碼來消除關聯向量。

10.4.8. Bob 建構新的洋蔥封包

Bob 現在將洋蔥有效載荷複製到洋蔥封包中，附加 Chan 的外部 HMAC，使用橢圓曲線乘法操作重新隨機化會話金鑰（與發送者 Alice 做的方式相同），並附加一個新的版本位元組。

要重新隨機化會話金鑰，Bob 首先使用他的節點公鑰和他派生的共享秘密計算他這一跳的盲化因子：

```
b_bob = SHA-256(P_bob || shared_secret_bob)
```

有了這個生成的，Bob 現在透過使用他的會話金鑰和盲化因子執行 EC 乘法來重新隨機化會話金鑰：

```
session_key_chan = session_key_bob * b_bob
```

`session_key_chan` 公鑰然後將被附加到洋蔥封包的前面，供 Chan 處理。

10.4.9. Bob 驗證 HTLC 細節

Bob 的跳有效載荷包含為 Chan 建立 HTLC 所需的指示。

在跳有效載荷中，Bob 找到 short_channel_id、amt_to_forward 和 cltv_expiry。

首先，Bob 檢查他是否有具有該短 ID 的通道。他發現他與 Chan 有這樣的通道。

接下來，Bob 確認出站金額（50,100 聰）小於入站金額（50,200 聰），因此 Bob 的費用預期得到滿足。

同樣，Bob 檢查出站 cltv_expiry 小於入站 cltv_expiry，給 Bob 足夠的時間在有違約時領取入站 HTLC。

10.4.10. Bob 向 Chan 發送 update_add_htlc

Bob 現在建構並發送給 Chan 的 HTLC，如下：

channel_id

此欄位包含 Bob-Chan 通道 ID，在我們的例子中是 000004040a61f0（見 從八卦的通道和節點資訊建構的詳細路徑）。

id

此通道中此 HTLC 的 ID，從 0 開始。

amount_msat

HTLC 的金額：50,100,000 毫聰。

payment_hash

RIPEMD160(SHA-256) 付款雜湊：

9e017f6767971ed7cea17f98528d5f5c0ccb2c71。

這與 Alice 的 HTLC 的付款雜湊相同。

cltv_expiry

HTLC 的到期時間鎖將是 700,038。

onion_routing_packet

Bob 在移除他的跳有效載荷後重建的洋蔥封包。

10.4.11. Chan 轉發洋蔥

Chan 重複與 Bob 完全相同的過程：

1. Chan 收到 update_add_htlc 並處理 HTLC 請求，將其添加到承諾交易中。
2. Chan 生成 Alice-Chan 共享金鑰和 mu 子金鑰。
3. Chan 驗證洋蔥封包 HMAC，然後提取 1,300 位元組的洋蔥 有效載荷。

4. Chan 將洋蔥有效載荷擴展 1,300 額外位元組，用零填充。
5. Chan 使用 rho 金鑰產生 2,600 位元組。
6. Chan 使用生成的位元組流 XOR 去混淆洋蔥有效載荷。同時，XOR 操作混淆額外的 1,300 個零，將它們變成填充。
7. Chan 提取有效載荷中的內部 HMAC，這將成為 Dina 的外部 HMAC。
8. Chan 移除他的跳有效載荷並將洋蔥有效載荷左移相同的量。在 1,300 擴展位元組中生成的一些填充移入前半 1,300 位元組，成為洋蔥有效載荷的一部分。
9. Chan 用這個洋蔥有效載荷為 Dina 建構洋蔥封包。
10. Chan 為 Dina 建構 update_add_htlc 訊息並將洋蔥封包插入其中。
11. Chan 向 Dina 發送 update_add_htlc。
12. Chan 像 Bob 在前一跳為 Dina 做的那樣重新隨機化會話金鑰。

10.4.12. Dina 收到最終有效載荷

當 Dina 從 Chan 收到 update_add_htlc 訊息時，她從 payment_hash 知道這是給她的付款。她知道她是洋蔥中的最後一跳。

Dina 按照與 Bob 和 Chan 完全相同的過程來驗證和解開洋蔥，除了她不建構新填充也不轉發任何東西。相反，Dina 用 update_fulfill_htlc 回應 Chan 以兌換 HTLC。update_fulfill_htlc 將沿路徑向後流動直到到達 Alice。所有 HTLC 都被兌換，通道餘額被更新。付款完成！

10.5. 返回錯誤

到目前為止，我們已經研究了洋蔥的前向傳播建立 HTLC，以及付款成功後付款秘密的反向傳播解開 HTLC。

還有另一個非常重要的洋蔥路由功能：*錯誤返回*。如果付款、洋蔥或跳出現問題，我們必須向後傳播錯誤以通知所有節點故障並解開任何 HTLC。

錯誤通常分為三類：洋蔥故障、節點故障和通道故障。此外，這些可以細分為永久性和暫時性錯誤。最後，一些錯誤包含通道更新以幫助未來的付款傳遞嘗試。

> 與點對點（P2P）協定中的訊息（定義在 BOLT #2: Peer Protocol for Channel Management (https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md)）不同，錯誤不是作為 P2P 訊息發送的，而是包裹在洋蔥返回封包內，並遵循洋蔥路徑的反向（反向傳播）。

錯誤返回定義在 BOLT #4: Onion Routing, Returning Errors (https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#returning-errors)
。

錯誤由返回節點（發現錯誤的節點）編碼在_返回封包_中，如下：

```
[32*byte:hmac]
[u16:failure_len]
[failure_len*byte:failuremsg]
[u16:pad_len]
[pad_len*byte:pad]
```

返回封包 HMAC 驗證校驗和使用 um 金鑰計算，該金鑰從洋蔥建立的共享秘密生成。

um 金鑰名稱是 mu 名稱的反向，表示相同的用途但方向相反（反向傳播）。

接下來，返回節點生成一個 ammag（單詞「gamma」的反向）金鑰，並使用與從 ammag 生成的位元組流的 XOR 操作混淆返回封包。

最後，返回節點將返回封包發送到它收到原始洋蔥的跳。

每個收到錯誤的跳將生成一個 ammag 金鑰，並再次使用與來自 ammag 的位元組流的 XOR 操作混淆返回封包。

最終，發送者（起源節點）收到返回封包。它將為每一跳生成 ammag 和 um 金鑰，並 XOR 去混淆返回錯誤，迭代直到揭示返回封包。

10.5.1. 失敗訊息

failuremsg 定義在 BOLT #4: Onion Routing, Failure Messages (https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#failure-messages)
。

失敗訊息由兩位元組 failure code 後跟適用於該失敗類型的資料組成。

failure_code 的頂部位元組是一組可以組合（用二進位 OR）的二進位標誌：

0x8000 (`BADONION`)

發送對等方加密的洋蔥無法解析

0x4000 (`PERM`)

永久故障（否則為暫時性）

0x2000 (`NODE`)

節點故障（否則為通道）

0x1000 (`UPDATE`)

包含新的通道更新

[failure_types_table] 中顯示的失敗類型目前已定義。

Unresolved directive in 10_onion_routing.adoc - include::failure_types_table.asciidoc[]

卡住的付款

在閃電網路的目前實作中，付款嘗試有可能變得_卡住_：既不被履行也不被錯誤取消。這可能由於中間節點上的錯誤、節點在處理 HTLC 時離線，或惡意節點持有 HTLC 而不報告錯誤而發生。在所有這些情況下，HTLC 無法解決直到它過期。在每個 HTLC 上設定的時間鎖（CLTV）有助於解決這種情況（以及其他可能的 HTLC 路由和通道故障）。

然而，這意味著 HTLC 的發送者必須等到過期，並且承諾給該 HTLC 的資金在 HTLC 過期之前仍然不可用。此外，發送者_不能重試_相同的付款，因為如果他們這樣做，他們會冒著原始付款和重試付款_都_成功的風險——接收者收到兩次付款。這是因為，一旦發送，HTLC 不能被發送者「取消」——它要麼必須失敗要麼過期。卡住的付款雖然罕見，但會造成不良的使用者體驗，使用者的錢包無法支付或取消付款。

針對此問題的一個提議解決方案稱為_無卡住付款_，它依賴於點時間鎖定合約（PTLC），這是使用與 HTLC 不同密碼學原語的付款合約（即橢圓曲線上的點加法而不是雜湊和秘密原像）。使用 ECDSA 的 PTLC 很麻煩，但使用比特幣的 Taproot 和 Schnorr 簽名功能則容易得多，這些功能最近已鎖定於 2021 年 11 月啟用。預計在這些比特幣功能啟用後，PTLC 將在閃電網路中實作。

10.6. Keysend 自發付款

在本章前面描述的付款流程中，我們假設 Dina 「帶外」或透過與協定無關的某些機制（通常是複製/貼上或 QR 碼掃描）從 Alice 那裡收到發票。這個特性意味著付款過程總是需要兩步：首先，發送者獲得發票，其次，使用付款雜湊（編碼在發票中）成功路由 HTLC。在透過閃電網路進行微付款串流的應用中，在付款之前獲取發票所需的額外往返可能是瓶頸。如果我們可以自發地「推送」付款，而不必首先從接收者那裡獲得發票呢？`keysend` 協定是閃電網路協定的端對端擴展（只有發送者和接收者知道），它允許自發推送付款。

10.6.1. 自訂洋蔥 TLV 記錄

現代閃電網路協定使用洋蔥中的 TLV（類型-長度-值）編碼來編碼告訴每個節點_在哪裡_和_如何_轉發付款的資訊。利用 TLV 格式，每條路由資訊（如將 HTLC 傳遞到的下一個節點）被分配一個特定類型（或鍵），編碼為 `BigSize` 可變長度整數（最大為 64 位元整數）。這些「基本」（低於 `65536` 的保留值）類型與洋蔥路由的其餘細節一起定義在 BOLT #4 中。值大於 `65536` 的洋蔥類型旨在被錢包和應用程式用作「自訂記錄」。

自訂記錄允許付款應用程式將額外的中繼資料或上下文作為洋蔥中的鍵/值對附加到付款。由於自訂記錄包含在洋蔥有效載荷本身中，與所有其他跳內容一樣，記錄是端對端加密的。由於自訂記錄實際上消耗了固定大小 1300 位元組洋蔥封包的一部分，編碼每個自訂記錄的每個鍵和值會減少可用於編碼路由其餘部分的空間。在實踐中，這意味著用於自訂記錄的洋蔥空間越多，路由可以越短。鑑於每個 HTLC 封包是固定大小的，自訂記錄不會_添加_任何額外資料到 HTLC；相反，它們重新分配否則會填充隨機資料的位元組。

10.6.2. 發送和接收 Keysend 付款

`keysend` 付款反轉了接收者向發送者揭示秘密原像的典型 HTLC 流程。相反，發送者在洋蔥_內_包含原像給接收者，並將 HTLC 路由到接收者。接收者然後解密洋蔥有效載荷，並使用包含的原像（_必須_匹配 HTLC 的付款雜湊）來結算付款。因此，`keysend` 付款可以在首先從接收者獲取發票之前進行，因為原像被「推送」到接收者。`keysend` 付款使用 TLV 自訂記錄類型 `5482373484` 來編碼 32 位元組原像值。

10.6.3. 閃電網路應用中的 Keysend 和自訂記錄

許多串流閃電網路應用程式使用 `keysend` 協定來持續向網路中由其公鑰識別的目的地串流聰。通常，應用程式還會在 `keysend` 記錄之外包含中繼資料，例如打賞/捐贈備註或其他應用程式級資訊。

10.7. 結論

閃電網路的洋蔥路由協定改編自 Sphinx 協定，以更好地服務於支付網路的需求。因此，與公開透明的比特幣區塊鏈相比，它在隱私和反監視方面提供了巨大的改進。

在 路徑尋找與付款傳遞 中，我們將看到源路由和洋蔥路由的組合如何被 Alice 用來找到一條好路徑並將付款路由到 Dina。要找到路徑，Alice 首先需要了解網路拓撲，這是 八卦協定與通道圖 的主題。

11. 八卦協定與通道圖

在本章中，我們將描述閃電網路的八卦協定（gossip protocol）以及節點如何使用它來建構和維護通道圖。我們還將回顧用於尋找「八卦」對等節點的 DNS 引導機制。

「路由費用和八卦中繼」部分在 閃電網路協定套件中的八卦協定 中以跨越路由層和點對點層的輪廓突顯。

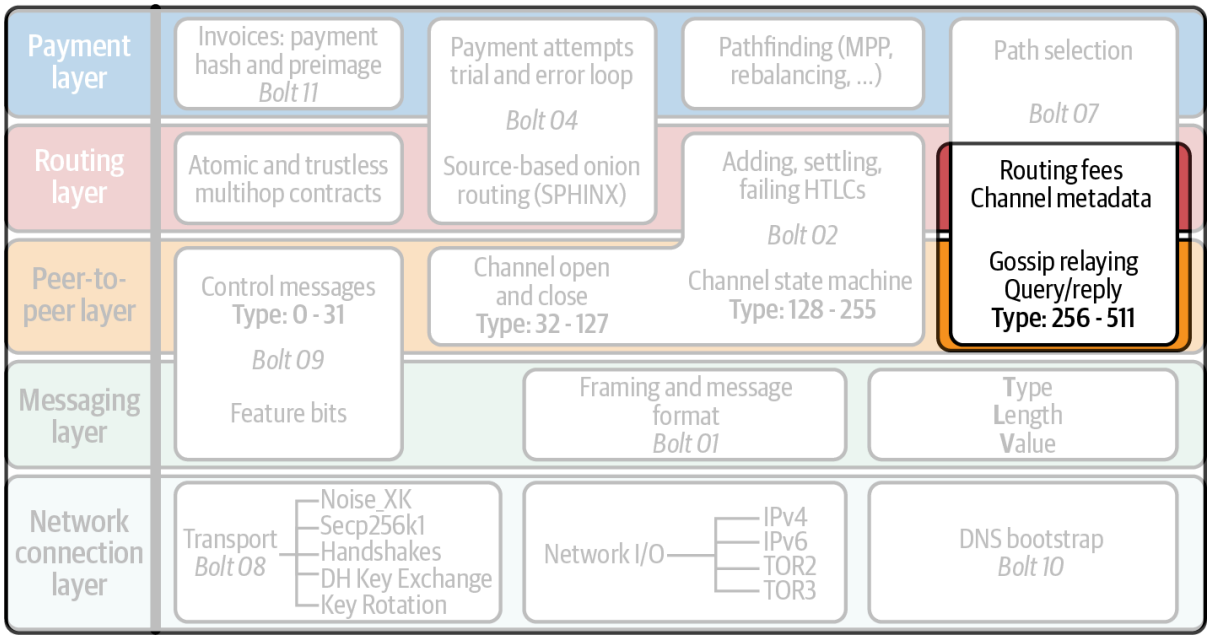

Figure 91. 閃電網路協定套件中的八卦協定

正如我們已經了解的，閃電網路使用基於源的洋蔥路由協定將付款從發送者傳遞給接收者。為此，發送節點必須能夠構建一條連接它與接收者的支付通道路徑，正如我們將在 路徑尋找與付款傳遞 中看到的。因此，發送者必須能夠通過構建通道圖來繪製閃電網路的地圖。*通道圖* 是公開宣告的通道以及這些通道連接的節點的互連集合。

由於通道是由鏈上發生的資金交易支持的，人們可能會錯誤地認為閃電網路節點可以直接從比特幣區塊鏈中提取現有通道。然而，這只能在一定程度上實現。資金交易是見證腳本雜湊支付（P2WSH）地址，只有當資金交易輸出被花費時才會揭示腳本的性質（2-of-2 多重簽名）。即使腳本的性質是已知的，重要的是要記住並非所有 2-of-2 多重簽名腳本都對應於支付通道。

還有更多原因說明為什麼查看比特幣區塊鏈可能沒有幫助。例如，在閃電網路上，用於簽名的比特幣密鑰由節點為每個通道和更新進行輪換。因此，即使我們可以可靠地檢測到比特幣區塊鏈上的資金交易，我們也不知道閃電網路上的哪兩個節點擁有該特定通道。

閃電網路通過實現 *八卦協定* 來解決這個問題。八卦協定是點對點（P2P）網路的典型協定，允許節點僅通過幾個與對等節點的直接連接就能與整個網路共享資訊。閃電節點彼此之間建立加密的點對點連接，並共享（八卦）它們從其他對等節點收到的資訊。一旦一個節點想要共享一

些資訊，例如關於新建立的通道，它就會向所有對等節點發送一條訊息。收到訊息後，節點會決定收到的訊息是否是新的，如果是，則將資訊轉發給其對等節點。這樣，如果點對點網路連接良好，所有對網路運作必要的新資訊最終都會傳播到所有其他對等節點。

顯然，如果一個新的對等節點第一次加入網路，它需要知道網路上的一些其他對等節點，這樣它才能連接到其他節點並參與網路。

在本章中，我們將探討閃電節點如何發現彼此、發現和更新其節點狀態，以及如何相互通訊。

當大多數人提到閃電網路的 *網路* 部分時，他們指的是 *通道圖*，它本身是一種獨特的認證資料結構，*錨定* 在基礎比特幣區塊鏈中。

然而，閃電網路也是一個節點的點對點網路，八卦傳播支付通道和節點的資訊。通常，為了讓兩個對等節點維護一個支付通道，它們需要直接相互通訊，這意味著它們之間會有一個對等連接。這暗示通道圖是點對點網路的子網路。然而這並不正確，因為即使一個或兩個對等節點暫時離線，支付通道也可以保持開放。

讓我們重新回顧一下我們在整本書中使用的一些術語，特別是看看它們在通道圖和點對點網路方面的含義（見 不同網路的術語）。

Table 3. 不同網路的術語

通道圖	**點對點網路**
通道	連接
開啟	連線
關閉	斷開連線
資金交易	加密的 TCP/IP 連接
發送	傳輸
付款	訊息

因為閃電網路是一個點對點網路，需要一些初始引導才能讓對等節點發現彼此。在本章中，我們將跟隨一個新對等節點第一次連接到網路的故事，並檢查引導過程中的每個步驟，從初始對等節點發現到通道圖同步和驗證。

作為初始步驟，我們的新節點需要以某種方式 *發現* 至少 *一個* 已經連接到網路並具有完整通道圖的對等節點（正如我們稍後將看到的，沒有規範版本的通道圖）。使用多種初始引導協定之一找到第一個對等節點後，在建立連接後，我們的新對等節點現在需要 *下載* 和 *驗證* 通道圖。一旦通道圖完全驗證，我們的新對等節點就可以開始開啟通道並在網路上發送付款了。

在初始引導之後，網路上的節點需要繼續維護其通道圖視圖，方法是處理新的通道路由策略更新、發現和驗證新通道、移除已在鏈上關閉的通道，最後修剪那些未能每兩週左右發送適當「心跳」的通道。

完成本章後，你將理解點對點閃電網路的一個關鍵組件：即對等節點如何發現彼此並維護通道圖的本地副本（視角）。我們將從探索一個剛剛啟動並需要找到網路上其他對等節點連接的新節點的故事開始。

11.1. 對等節點發現

在本節中，我們將開始跟隨一個希望加入網路的新閃電節點經歷三個步驟：

1. 發現一組引導對等節點
2. 下載並驗證通道圖
3. 開始通道圖本身的持續維護過程

11.1.1. P2P 引導

在做任何其他事情之前，我們的新節點首先需要發現一組已經是網路一部分的對等節點。我們稱這個過程為初始對等節點引導，這是每個點對點網路都需要正確實現的事情，以確保一個健壯、健康的網路。

將新對等節點引導到現有點對點網路是一個研究得非常透徹的問題，有幾種已知的解決方案，每種都有其獨特的權衡。這個問題最簡單的解決方案是在打包的 P2P 節點軟體中打包一組 *硬編碼* 的引導對等節點。這很簡單，因為每個新節點在其運行的軟體中都有一個引導對等節點列表，但相當脆弱，因為如果引導對等節點集離線，則沒有新節點能夠加入網路。由於這種脆弱性，這個選項通常用作備用方案，以防其他 P2P 引導機制無法正常工作。

與其在軟體/二進位檔本身中硬編碼引導對等節點集，我們可以讓對等節點動態獲取一組新鮮的引導對等節點，用於加入網路。我們將稱這個過程為 *初始對等節點發現*。通常我們會利用現有的網際網路協定來維護和分發一組引導對等節點。過去用於完成初始對等節點發現的協定的非詳盡列表包括：

- 域名系統（DNS）
- 網際網路中繼聊天（IRC）
- 超文本傳輸協定（HTTP）

與比特幣協定類似，閃電網路中使用的主要初始對等節點發現機制是通過 DNS 進行的。因為初始對等節點發現是網路的關鍵和通用任務，這個過程已在 BOLT #10：DNS 引導 (https://github.com/lightningnetwork/lightning-rfc/blob/master/10-dns-bootstrap.md) 中被 *標準化*。

11.1.2. DNS 引導

BOLT #10 (https://github.com/lightningnetwork/lightning-rfc/blob/master/10-dns-bootstrap.md) 文件描述了使用 DNS 實現對等節點發現的標準化方式。閃電網路的 DNS 引導風格使用多達三種不同的記錄類型：

- SRV 記錄用於發現一組 *節點公鑰*。
- A 記錄用於將節點的公鑰映射到其當前的 IPv4 地址。
- AAA 記錄用於將節點的公鑰映射到其當前的 IPv6 地址。

那些對 DNS 協定有些熟悉的人可能已經熟悉 A（名稱到 IPv4 地址）和 AAA（名稱到 IPv6 地址）記錄類型，但不熟悉 SRV 類型。SRV 記錄類型被構建在 DNS 之上的協定用於確定指定服務的 *位置*。在我們的上下文中，所討論的服務是給定的閃電節點，位置是其 IP 地址。我們需要使用這個額外的記錄類型，因為與比特幣協定中的節點不同，我們需要公鑰 *和* IP 地址才能連接到節點。正如我們在 線路協定：框架和可擴展性 中看到的，閃電網路中使用的傳輸加密協定需要在連接之前知道節點的公鑰，以便為網路中的節點實現身份隱藏。

新對等節點的引導工作流程

在深入了解 BOLT #10 (https://github.com/lightningnetwork/lightning-rfc/blob/master/10-dns-bootstrap.md) 的細節之前，我們將首先概述一個希望使用 BOLT #10 加入網路的新節點的高層流程。

首先，節點需要識別一個或一組理解 BOLT #10 的 DNS 伺服器，以便用於 P2P 引導。

雖然 BOLT #10 使用 *lseed.bitcoinstats.com* 作為種子伺服器，但並不存在用於此目的的「官方」DNS 種子集，但每個主要實現都維護自己的 DNS 種子，並且它們會互相查詢對方的種子以實現冗餘。在 已知閃電網路 DNS 種子伺服器表 中，你將看到一些流行 DNS 種子伺服器的非詳盡列表。

Table 4. 已知閃電網路 DNS 種子伺服器表

DNS 伺服器	維護者
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

DNS 種子存在於比特幣的主網和測試網。為了我們的範例，我們將假設在 *nodes.lightning.directory* 存在一個有效的 BOLT #10 DNS 種子。

接下來，我們的新節點將發出 SRV 查詢以獲取一組 *候選引導對等節點*。對我們查詢的回應將是一系列 bech32 編碼的公鑰。因為 DNS 是基於文本的協定，我們無法發送原始二進位資料，所以需要一個編碼方案。由於 bech32 在更廣泛的比特幣生態系統中的使用，BOLT #10 指定了 bech32 編碼。返回的編碼公鑰數量取決於返回查詢的伺服器，以及位於客戶端和權威伺服器之間的所有解析器。

使用廣泛可用的 dig 命令列工具，我們可以使用以下命令查詢前面提到的 DNS 種子的 *測試網* 版本：

```
$ dig @8.8.8.8 test.nodes.lightning.directory SRV
```

我們使用 @ 參數強制通過 Google 的名稱伺服器（IP 地址為 8.8.8.8）解析，因為它不會過濾大型 SRV 查詢回應。在命令末尾，我們指定只需要返回 SRV 記錄。範例回應如 查詢 DNS 種子以獲取可達節點 所示。

Example 8. 查詢 DNS 種子以獲取可達節點

```
$ 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.
lightning.directory. (2)
test.nodes.lightning.directory. 59 IN   SRV 10 10 15735
ln1qtgsl3efj8verd4z27k44xu0a59kncvsarxatahm334exgnuvwhnz8dkhx8.test.nodes.
lightning.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) 可以連接到閃電節點的 TCP 連接埠號。

(2) 作為虛擬域名編碼的節點公鑰（ID）。

為簡潔起見，我們截斷了回應，只顯示兩個返回的回應。回應包含目標節點的「虛擬」域名，然後在左邊我們有可以連接到該節點的 *TCP 連接埠*。第一個回應使用閃電網路的標準 TCP 連接埠：9735。第二個回應使用自訂連接埠，這是協定允許的。

接下來，我們將嘗試獲取連接到節點所需的另一條資訊：其 IP 地址。然而，在我們可以查詢之前，我們將首先 *解碼* 虛擬域名中公鑰的 bech32 編碼：

```
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7
```

解碼這個 bech32 字串，我們得到以下有效的 secp256k1 公鑰：

```
026c64f5a7f24c6f7f0e1d6ec877f23b2f672fb48967c2545f227d70636395eaf3
```

現在我們有了原始公鑰，我們將要求 DNS 伺服器 *解析* 給定的虛擬主機，以便我們可以獲取節點的 IP 資訊（A 記錄），如 [ex1102] 所示。

獲取節點的最新 IP 地址

```
$ dig
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test.nodes.lightning
.directory A

; <<>> DiG 9.10.6 <<>>
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test.nodes.lightning
.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.lightnin
g.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
;; MSG SIZE  rcvd: 138
```

1

DNS 伺服器返回 IP 地址 `X.X.X.X`。我們在此處將其替換為 X 以避免呈現真實的 IP 地址。

在前面的命令中，我們查詢了伺服器以便獲取目標節點的 IPv4 （`A` 記錄） 地址（在前面的範例中替換為 __X.X.X.X__）。現在我們有了原始公鑰、IP 地址和 TCP 連接埠，我們可以通過以下方式連接到節點傳輸協定：

```
026c64f5a7f24c6f7f0e1d6ec877f23b2f672fb48967c2545f227d70636395eaf3@X.X.X.X:9735
```

查詢給定節點的當前 DNS A 記錄也可用於查找 *最新* 的地址集。與等待八卦網路上的地址更新相比，此類查詢可用於更快地同步節點的最新地址資訊（見 node_announcement 訊息）。

在我們旅程的這個時刻，我們的新閃電節點已經找到了它的第一個對等節點並建立了它的第一個連接！現在我們可以開始新對等節點引導的第二階段：通道圖同步和驗證。

首先，我們將更深入地探索 BOLT #10 本身的複雜性，以更深入地了解底層的工作原理。

11.1.3. SRV 查詢選項

BOLT #10 (https://github.com/lightningnetwork/lightning-rfc/blob/master/10-dns-bootstrap.md) 標準由於使用巢狀子域作為附加查詢選項的通訊層而具有高度可擴展性。引導協定允許客戶端進一步指定它們嘗試查詢的節點 *類型*，而不是預設接收查詢回應中的隨機節點子集。

查詢選項子域方案使用一系列鍵值對，其中鍵本身是 *單個字母*，其餘文本集是值本身。以下查詢類型存在於當前版本的 BOLT #10 (https://github.com/lightningnetwork/lightning-rfc/blob/master/10-dns-bootstrap.md) 標準文件中：

r

領域 位元組，用於確定應該為哪個鏈或領域返回查詢。目前，這個鍵的唯一值是 0，表示「比特幣」。

a

允許客戶端根據它們宣告的地址 *類型* 過濾返回的節點。例如，這可用於僅獲取宣告有效 IPv6 地址的節點。此類型後面的值基於一個位元欄位，該欄位 *索引* 到 BOLT #7 (https://github.com/lightningnetwork/lightning-rfc/blob/master/07-routing-gossip.md) 中定義的指定地址 *類型* 集合中。此欄位的預設值為 6，表示 IPv4 和 IPv6（設置了位元 1 和 2）。

l

以壓縮格式序列化的有效節點公鑰。這允許客戶端查詢指定的節點而不是接收一組隨機節點。

n

要返回的記錄數。此欄位的預設值為 25。

帶有附加查詢選項的範例查詢看起來像這樣：

```
r0.a2.n10.nodes.lightning.directory
```

逐個鍵值對分解查詢，我們獲得以下見解：

r0

查詢針對比特幣領域

a2

查詢只想要返回 IPv4 地址

n10

查詢請求 10 條記錄

使用 dig DNS 命令列工具自己嘗試各種標誌的組合：

```
dig @8.8.8.8 r0.a6.nodes.lightning.directory SRV
```

11.2. 通道圖

現在我們的新節點能夠使用 DNS 引導協定連接到它的第一個對等節點，它可以開始同步通道圖了！然而，在我們同步通道圖之前，我們需要確切地了解我們所說的通道圖是什麼意思。在本節中，我們將探索通道圖的精確 *結構*，並檢查通道圖與電腦科學領域中眾所周知/使用的典型抽象「圖」資料結構相比的獨特方面。

11.2.1. 有向圖

電腦科學中的 *圖* 是由頂點（通常稱為節點）和邊（也稱為連結）組成的特殊資料結構。兩個節點可以由一條或多條邊連接。通道圖也是 *有向的*，因為付款能夠在給定邊（通道）的任一方向流動。有向圖的範例如 有向圖 所示。

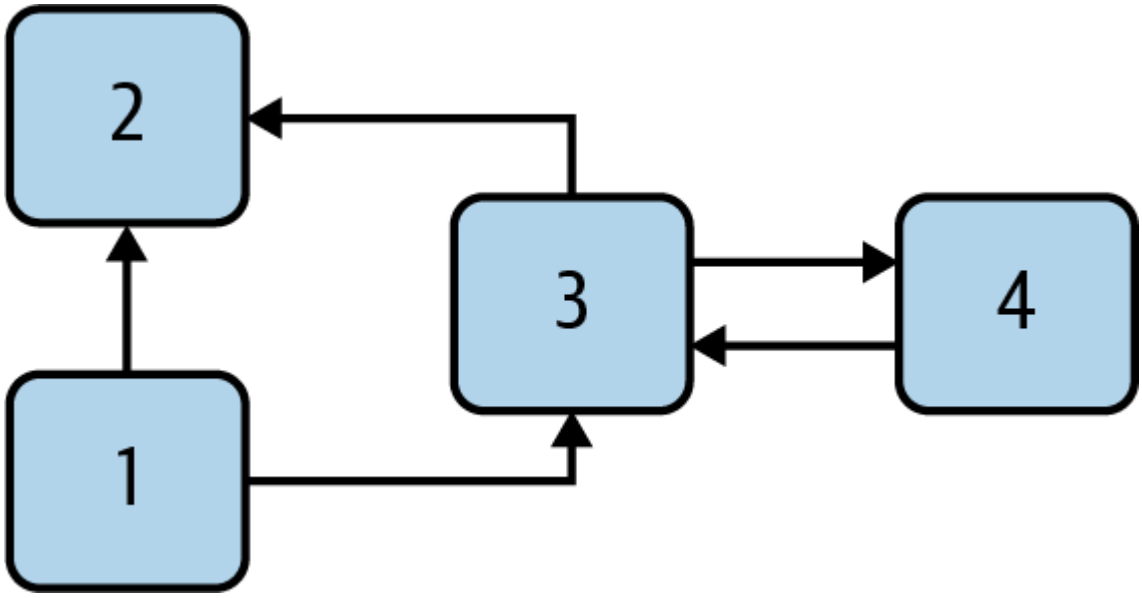

Figure 92. 有向圖

在閃電網路的上下文中，我們的頂點是閃電節點本身，我們的邊是連接這些節點的支付通道。因為我們關心的是 *路由付款*，在我們的模型中，沒有邊（沒有支付通道）的節點不被認為是圖的一部分，因為它沒有用處。

因為通道本身是 UTXO（資金充足的 2-of-2 多重簽名地址），我們可以將通道圖視為比特幣 UTXO 集的特殊子集，在其上我們可以添加一些額外資訊（節點等）以得到最終的覆蓋結構，即通道圖。通道圖的基本組件錨定在基礎比特幣區塊鏈中，這意味著不可能 *偽造* 一個有效的通道圖，這在防止垃圾郵件方面具有有用的屬性，正如我們稍後將看到的。

11.3. 八卦協定訊息

通道圖資訊通過三種訊息在閃電 P2P 網路上傳播，這些訊息在 BOLT #7 (https://github.com/lightningnetwork/lightning-rfc/blob/master/07-routing-gossip.md) 中描述：

node_announcement

我們圖中的頂點，傳達節點的公鑰，以及如何通過網際網路連接到該節點和一些額外的元資料，描述該節點支援的 *功能* 集。

channel_announcement

區塊鏈錨定的兩個獨立節點之間通道存在的證明。任何第三方都可以驗證此證明以確保正在宣告 *真實* 的通道。與 node_announcement 類似，此訊息還包含描述通道 *能力* 的資訊，這在嘗試路由付款時很有用。

channel_update

一 *對* 結構，描述給定通道的路由策略集。channel_update 訊息成對出現，因為通道是有向邊，所以通道的每一側都可以指定自己的自訂路由策略。

重要的是要注意，通道圖的每個組件都是 *經過認證的*，允許第三方確保通道/更新/節點的所有者實際上是發送更新的人。這有效地使通道圖成為一種獨特類型的 *認證資料結構*，不能被偽造。對於認證，我們使用對訊息本身序列化摘要的 secp256k1 ECDSA 數位簽名（或一系列簽名）。在本章中我們不會深入討論閃電網路中使用的訊息框架/序列化的具體細節，因為我們將在 線路協定：框架和可擴展性 中涵蓋這些資訊。

在佈局了通道圖的高層結構之後，我們現在將深入探討用於八卦傳播通道圖的三種訊息中每一種的精確結構。我們還將解釋如何驗證通道圖的每條訊息和組件。

11.3.1. node_announcement 訊息

首先，我們有 node_announcement 訊息，它有兩個主要目的：

1. 宣告連接資訊，以便其他節點可以連接到該節點，無論是引導到網路還是嘗試與該節點建立新的支付通道。
2. 傳達節點理解/支援的協定級功能集。節點之間的功能協商允許開發人員獨立添加新功能，並在選擇加入的基礎上與網路上的任何其他節點支援它們。

與通道公告不同，節點公告不錨定在基礎區塊鏈中。因此，只有當節點公告與相應的通道公告一起傳播時，才被認為是有效的。換句話說，我們總是拒絕沒有支付通道的節點，以確保惡意對等節點不能用不屬於通道圖的虛假節點淹沒網路。

node_announcement 訊息結構

node_announcement 由以下欄位組成：

signature

一個有效的 ECDSA 簽名，覆蓋下面列出的所有欄位的序列化摘要。此簽名必須對應於宣告節點的公鑰。

features

一個位元向量，描述此節點理解的協定功能集。我們將在 功能位元和協定可擴展性 關於閃電協定可擴展性的部分更詳細地介紹此欄位。在高層次上，此欄位攜帶一組表示節點理解的功能的位元。例如，節點可以發出信號表示它理解最新的通道類型。

timestamp

Unix 紀元編碼的時間戳。這允許客戶端對節點公告的更新強制執行部分排序。

node_id

此節點公告所屬的 secp256k1 公鑰。在任何給定時間，通道圖中給定節點只能有一個 node_announcement。因此，如果 node_announcement 攜帶更高（較晚）的時間戳，則可以取代同一節點的先前 node_announcement。

rgb_color

允許節點指定與其關聯的 RGB 顏色的欄位，通常用於通道圖視覺化和節點目錄。

alias

一個 UTF-8 字串，作為給定節點的暱稱。請注意，這些別名不需要全域唯一，也不會以任何方式進行驗證。因此，它們不應被依賴為一種身份形式——它們可以很容易地被偽造。

addresses

一組與給定節點關聯的公共網際網路可達地址。在協定的當前版本中，支援四種地址類型：IPv4（類型：1）、IPv6（類型：2）、Tor v2（類型：3）和 Tor v3（類型：4）。在 node_announcement 訊息中，這些地址類型中的每一種都由一個整數類型表示，該類型包含在地址類型後面的括號中。

驗證節點公告

驗證傳入的 node_announcement 很簡單。在檢查節點公告時，應該堅持以下斷言：

- 如果該節點已經存在一個 node_announcement，則新傳入的 node_announcement 的 timestamp 欄位必須大於之前的。
- 通過這個約束，我們強制執行一定程度的「新鮮度」。
- 如果給定節點不存在 node_announcement，則引用給定節點的現有 channel_announcement（稍後會詳細介紹）必須已經存在於本地通道圖中。
- 包含的 signature 必須是一個有效的 ECDSA 簽名，使用包含的 node_id 公鑰和原始訊息編碼（減去簽名和幀頭）的雙 SHA-256 摘要作為訊息進行驗證。
- 所有包含的 addresses 必須根據其地址標識符按升序排序。
- 包含的 alias 位元組必須是有效的 UTF-8 字串。

11.3.2. channel_announcement 訊息

接下來，我們有 channel_announcement 訊息，用於向更廣泛的網路 *宣告* 新的 *公開* 通道。請注意，宣告通道是 *可選的*。只有當通道打算被閃電網路用於路由時，才需要宣告通道。活躍的路由節點可能希望宣告他們所有的通道。然而，某些節點如行動節點可能沒有正常運行時間或成為活躍路由節點的意願。因此，這些行動節點（通常使用輕客戶端連接到比特幣 P2P 網路）可能只有純粹 *未宣告*（私有）的通道。

未宣告（私有）通道

未宣告的通道不是已知公開通道圖的一部分，但仍然可以用於發送/接收付款。精明的讀者現在可能想知道不屬於公開通道圖的通道如何能夠接收付款。這個問題的解決方案是一組我們稱為路由提示的「路徑尋找輔助工具」。正如我們將在 閃電網路付款請求 中看到的，由具有未宣告通道的節點創建的發票將包含資訊以幫助發送者路由到它們，假設該節點至少有一個與現有公開路由節點的通道。

由於未宣告通道的存在，通道圖的 *真實* 大小（包括公開和私有組件）是未知的。

在比特幣區塊鏈上定位通道

如前所述，由於使用公鑰密碼學以及比特幣區塊鏈作為垃圾郵件防範系統，通道圖是經過認證的。為了讓節點接受新的 channel_announcement，廣告必須 *證明* 通道確實存在於比特幣區塊鏈中。這個證明系統為向通道圖添加新條目增加了前期成本（創建通道 UTXO 必須支付的鏈上費用）。因此，我們減輕了垃圾郵件，並確保網路上的不誠實節點無法以零成本用虛假通道填滿誠實節點的記憶體。

鑑於我們需要構建一個通道存在的證明，一個自然產生的問題是：我們如何「指向」或引用驗證者的給定通道？鑑於支付通道錨定在未花費的交易輸出中（見 輸入和輸出），最初的想法可能是首先嘗試宣告通道的完整出點（txid:index）。鑑於出點是全域唯一的並在鏈上確認，這聽起來像是個好主意；然而，它有一個缺點：驗證者必須維護 UTXO 集的完整副本才能驗證通道。這對比特幣全節點來說工作正常，但依賴輕量級驗證的客戶端通常不維護完整的 UTXO 集。因為我們想確保我們可以在閃電網路中支援行動節點，我們被迫尋找另一個解決方案。

如果我們不是通過其 UTXO 引用通道，而是根據其在鏈中的「位置」引用它呢？為此，我們需要一個方案，允許我們引用給定的區塊，然後是該區塊內的交易，最後是該交易創建的特定輸出。這樣的標識符在 BOLT #7 (https://github.com/lightningnetwork/lightning-rfc/blob/master/07-routing-gossip.md) 中描述，被稱為 *短通道 ID*，或 scid。scid 用於 channel_announcement（和 channel_update）以及我們在 洋蔥路由 中了解到的包含在 HTLC 中的洋蔥加密路由封包中。

短通道 ID

基於前面的資訊，我們有三條需要編碼的資訊來唯一引用給定的通道。因為我們想要一個緊湊的表示，我們將嘗試將資訊編碼為 *單個* 整數。我們選擇的整數格式是無符號 64 位整數，由 8 個位元組組成。

首先，區塊高度。使用 3 個位元組（24 位），我們可以編碼 16,777,216 個區塊。這留下 5 個位元組供我們分別編碼交易索引和輸出索引。我們將使用接下來的 3 個位元組來編碼區塊 *內* 的交易索引。鑑於在當前區塊大小下一個區塊中只能容納數萬筆交易，這已經足夠了。這留下 2 個位元組供我們編碼交易內通道的輸出索引。

我們最終的 scid 格式類似於：

```
block_height (3 bytes) || transaction_index (3 bytes) || output_index (2 bytes)
```

使用位元打包技術，我們首先將最高有效的 3 個位元組編碼為區塊高度，接下來的 3 個位元組編碼為交易索引，最低有效的 2 個位元組編碼為創建通道 UTXO 的輸出索引。

短通道 ID 可以表示為單個整數（695313561322258433）或更人性化的字串：632384x1568x1。在這裡我們看到通道是在區塊 632384 中開採的，是區塊中的第 1568 筆交易，通道輸出是交易產生的第二個（UTXO 是零索引的）輸出。

現在我們能夠簡潔地指向鏈中給定的通道資金輸出，我們可以檢查 channel_announcement 訊息的完整結構，以及了解如何驗證訊息中包含的存在證明。

channel_announcement 訊息結構

channel_announcement 主要傳達兩件事：

1. 節點 A 和節點 B 之間存在通道的證明，兩個節點都控制該通道輸出中的多重簽名密鑰。
2. 通道的能力集（它可以路由什麼類型的 HTLC 等）。

在描述證明時，我們通常會提到節點 1 和節點 2。在通道連接的兩個節點中，當我們以壓縮格式十六進位編碼按字典順序比較兩個節點的公鑰時，具有「較低」公鑰編碼的節點是「第一個」節點。相應地，除了網路上的節點公鑰外，每個節點還應該在比特幣區塊鏈中控制一個公鑰。

與 node_announcement 訊息類似，channel_announcement 訊息的所有包含簽名應該針對最終簽名 *之後* 的訊息原始編碼（減去頭部）進行簽名/驗證（因為數位簽名不可能簽署自己）。

話雖如此，channel_announcement 訊息具有以下欄位：

node_signature_1

第一個節點對訊息摘要的簽名。

node_signature_2

第二個節點對訊息摘要的簽名。

bitcoin_signature_1

第一個節點的多重簽名密鑰（在資金輸出中）對訊息摘要的簽名。

bitcoin_signature_2

第二個節點的多重簽名密鑰（在資金輸出中）對訊息摘要的簽名。

features

描述此通道支援的協定級功能集的功能位元向量。

chain_hash

32 位元組雜湊，通常是開啟通道的區塊鏈（例如比特幣主網）的創世區塊雜湊。

short_channel_id

在區塊鏈中唯一定位給定通道資金輸出的 scid。

node_id_1

網路中第一個節點的公鑰。

node_id_2

網路中第二個節點的公鑰。

bitcoin_key_1

網路中第一個節點的通道資金輸出的原始多重簽名密鑰。

bitcoin_key_2

網路中第二個節點的通道資金輸出的原始多重簽名密鑰。

通道公告驗證

現在我們知道 channel_announcement 包含什麼，我們可以看看如何驗證通道在鏈上的存在。

有了 channel_announcement 中的資訊，任何閃電節點（即使沒有比特幣區塊鏈的完整副本）都可以驗證支付通道的存在和真實性。

首先，驗證者將使用短通道 ID 來找出哪個比特幣區塊包含通道資金輸出。有了區塊高度資訊，驗證者可以只從比特幣節點請求那個特定的區塊。然後可以通過向後跟隨區塊頭鏈（驗證工作量證明）將該區塊連結回創世區塊，確認這確實是屬於比特幣區塊鏈的區塊。

接下來，驗證者使用交易索引號來識別包含支付通道的交易的交易 ID。大多數現代比特幣庫將允許根據交易在較大區塊中的索引來索引區塊的交易。

接下來，驗證者使用比特幣庫（以驗證者的語言）根據其在區塊內的索引提取相關交易。驗證者將驗證交易（檢查它是否正確簽名並在雜湊時產生相同的交易 ID）。

接下來，驗證者將提取由短通道 ID 的輸出索引號引用的見證腳本雜湊支付（P2WSH）輸出。這是通道資金輸出的地址。此外，驗證者將確保所宣稱通道的大小與指定輸出索引處產生的輸出值相匹配。

最後，驗證者將從 bitcoin_key_1 和 bitcoin_key_2 重建多重簽名腳本，並確認它產生與輸出中相同的地址。

驗證者現在已經獨立驗證了公告中的支付通道已在比特幣區塊鏈上獲得資金並確認！

11.3.3. channel_update 訊息

八卦協定中使用的第三個也是最後一個訊息是 channel_update 訊息。每個支付通道會生成兩個這樣的訊息（每個通道合作夥伴一個），宣告它們的路由費用、時間鎖期望和能力。

channel_update 訊息還包含一個時間戳，允許節點通過發送具有更高（較晚）時間戳的新 channel_update 訊息來更新其路由費用和其他期望和能力，該訊息取代任何舊的更新。

channel_update 訊息包含以下欄位：

signature

匹配節點公鑰的數位簽名，用於認證通道更新的來源和完整性

chain_hash

包含通道的鏈的創世區塊雜湊

short_channel_id

用於識別通道的短通道 ID

timestamp

此更新的時間戳，允許接收者對更新進行排序並替換舊的更新

message_flags

指示 channel_update 訊息中額外欄位存在的位元欄位

channel_flags

顯示通道方向和其他通道選項的位元欄位

cltv_expiry_delta

此節點用於路由的時間鎖增量期望（見 洋蔥路由）

htlc_minimum_msat

將被路由的最小 HTLC 金額

fee_base_msat

將為路由收取的基本費用

fee_proportional_millionths

將為路由收取的比例費率

htlc_maximum_msat (option_channel_htlc_max)

將被路由的最大金額

接收 channel_update 訊息的節點可以將此元資料附加到通道圖邊以啟用路徑尋找，正如我們將在 路徑尋找與付款傳遞 中看到的。

11.4. 持續的通道圖維護

通道圖的構建不是一次性事件，而是一項持續的活動。當節點引導進入網路時，它將開始以三種更新訊息的形式接收「八卦」。它將使用這些訊息立即開始構建經過驗證的通道圖。

節點接收的資訊越多，它對閃電網路的「地圖」就越好，它在路徑尋找和付款傳遞方面就越有效。

節點不僅會向通道圖添加資訊。它還會追蹤通道上次更新的時間，並刪除超過兩週未更新的「陳舊」通道。最後，如果它看到某個節點不再有任何通道，它也會移除該節點。

從八卦協定收集的資訊不是唯一可以存儲在通道圖中的資訊。不同的閃電節點實現可能會將其他元資料附加到節點和通道上。例如，一些節點實現會計算一個「分數」來評估節點作為路由對等節點的「品質」。這個分數用作路徑尋找的一部分，以優先或降低路徑的優先級。

11.5. 結論

在本章中，我們了解了閃電節點如何發現彼此、發現和更新其節點狀態，以及如何相互通訊。我們了解了通道圖是如何創建和維護的，並探索了閃電網路阻止惡意行為者或不誠實節點向網路發送垃圾郵件的幾種方式。

12. 路徑尋找與付款傳遞

閃電網路上的付款傳遞依賴於尋找一條從發送者到接收者的路徑，這個過程稱為 *路徑尋找*。由於路由是由發送者完成的，發送者必須找到一條合適的路徑來到達目的地。然後這條路徑被編碼在洋蔥中，正如我們在 洋蔥路由 中看到的。

在本章中，我們將檢視路徑尋找的問題，理解通道餘額的不確定性如何使這個問題複雜化，並了解典型的路徑尋找實現如何嘗試解決它。

12.1. 閃電協定套件中的路徑尋找

路徑尋找、路徑選擇、多部分付款（MPP）和付款嘗試試錯迴圈佔據了協定套件頂部付款層的大部分。

這些組件在協定套件中以輪廓突顯，如 閃電協定套件中的付款傳遞 所示。

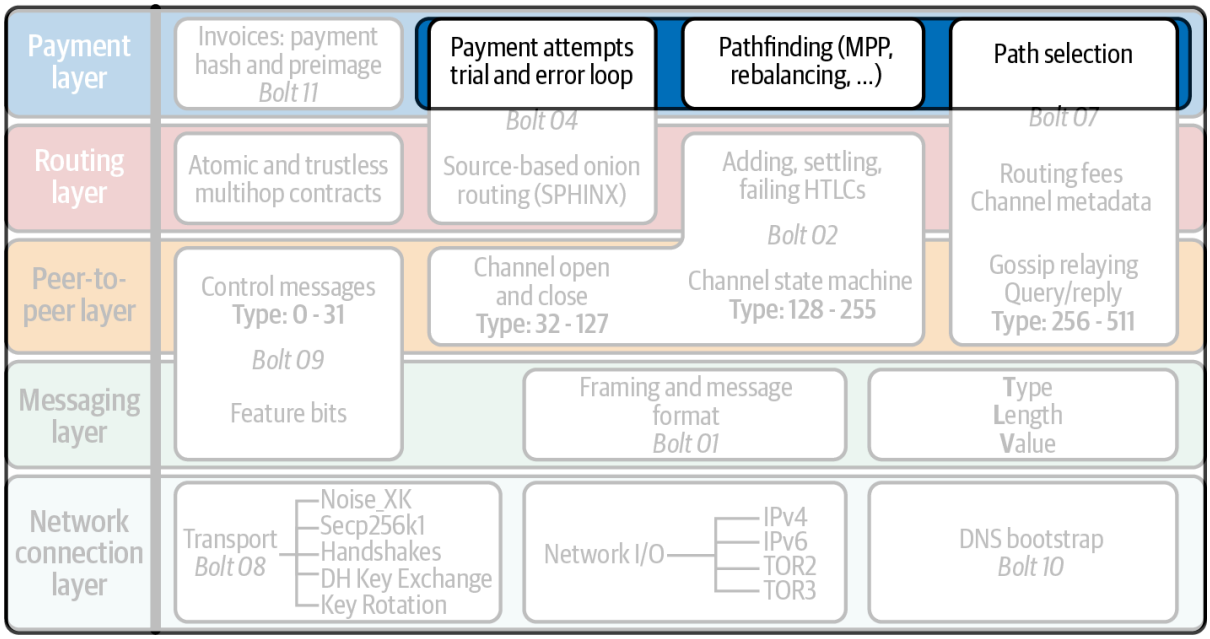

Figure 93. 閃電協定套件中的付款傳遞

12.1.1. BOLT 在哪裡？

到目前為止，我們已經研究了幾種作為閃電網路一部分的技術，並且看到了它們作為 BOLT 標準一部分的確切規範。你可能會驚訝地發現路徑尋找不是 BOLT 的一部分！

這是因為路徑尋找不是需要不同實現之間任何形式的協調或互操作性的活動。正如我們所見，路徑由發送者選擇。即使路由細節在 BOLT 中有詳細規定，路徑發現和選擇完全留給發送者。因此，每個節點實現可以選擇不同的策略/演算法來尋找路徑。事實上，不同的節點/客戶端和錢包實現甚至可以競爭，並將其路徑尋找演算法作為差異化的亮點。

12.2. 路徑尋找：我們在解決什麼問題？

路徑尋找這個術語可能有些誤導，因為它暗示搜尋連接兩個節點的 *單一路徑*。在開始時，當閃電網路很小且連接不夠充分時，問題確實是關於找到一種方式來連接支付通道以到達接收者。

但是，隨著閃電網路的爆炸性增長，路徑尋找問題的性質已經改變。在 2021 年中期，當我們完成這本書時，閃電網路由 20,000 個節點組成，由至少 55,000 個公開通道連接，總容量接近 2,000 BTC。一個節點平均有 8.8 個通道，而前 10 個連接最多的節點 *每個* 有 400 到 2,000 個通道。閃電網路通道圖的一小部分子集的視覺化如 截至 2021 年 7 月閃電網路部分的視覺化 所示。

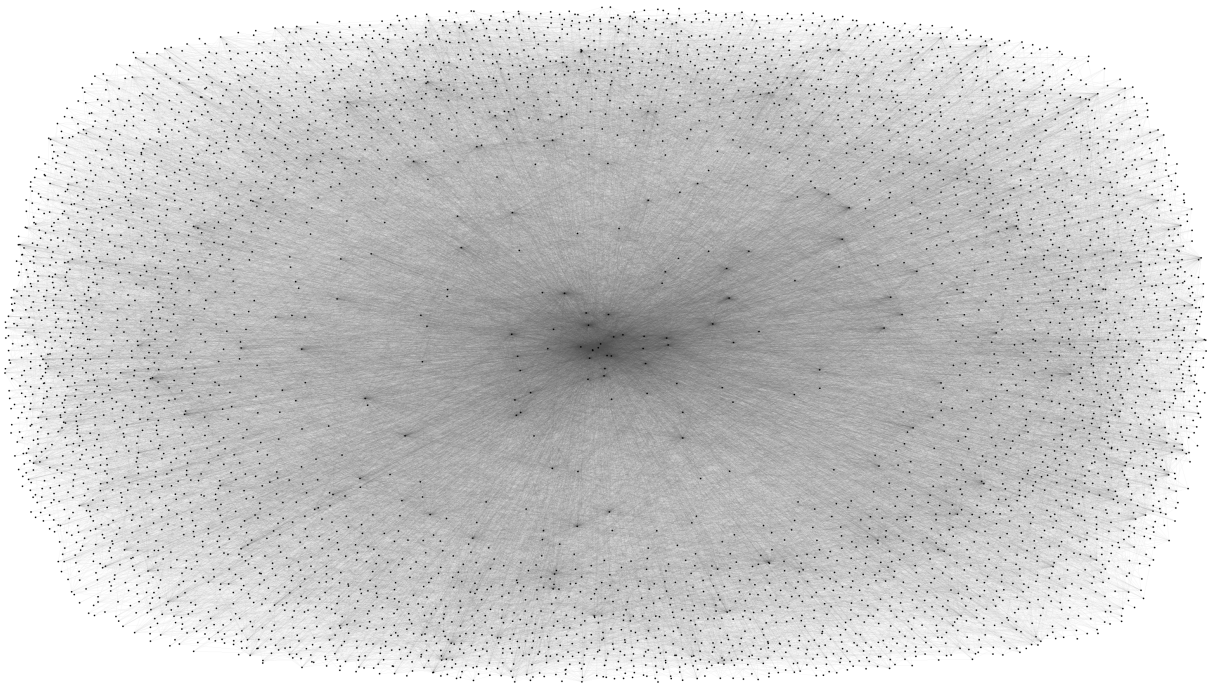

Figure 94. 截至 2021 年 7 月閃電網路部分的視覺化

截至 2021 年 7 月閃電網路部分的視覺化 中的網路視覺化是用一個簡單的 Python 腳本生成的，你可以在本書儲存庫的 code/lngraph 中找到它。

如果發送者和接收者都連接到其他連接良好的節點，並且至少有一個具有足夠容量的通道，那麼將有數千條路徑。問題變成從數千條可能的路徑列表中選擇將成功傳遞付款的 *最佳* 路徑。

12.2.1. 選擇最佳路徑

要選擇最佳路徑，我們必須首先定義「最佳」是什麼意思。可能有許多不同的標準，例如：

- 具有足夠流動性的路徑。顯然，如果一條路徑沒有足夠的流動性來路由我們的付款，那麼它就不是一條合適的路徑。
- 費用低的路徑。如果我們有幾個候選者，我們可能想要選擇費用較低的那些。
- 時間鎖短的路徑。我們可能想要避免鎖定我們的資金太長時間，因此選擇時間鎖較短的路徑。

所有這些標準在某種程度上都是可取的，選擇在多個維度上都有利的路徑不是一件容易的事。像這樣的最佳化問題可能太複雜而無法解決「最佳」解決方案，但通常可以解決最優解的某種近似值，這是好消息，因為否則路徑尋找將是一個棘手的問題。

12.2.2. 數學和電腦科學中的路徑尋找

閃電網路中的路徑尋找屬於數學中 *圖論* 的一般類別，以及電腦科學中 *圖遍歷* 的更具體類別。

像閃電網路這樣的網路可以表示為一種稱為 *圖* 的數學結構，其中 *節點* 通過 *邊*（相當於支付通道）相互連接。閃電網路形成一個 *有向圖*，因為節點是 *不對稱* 連接的，因為通道餘額在兩個通道合作夥伴之間分配，付款流動性在每個方向上都不同。在其邊上具有數值容量約束的有向圖稱為 *流網路*，這是一種用於優化運輸和其他類似網路的數學結構。當解決方案需要在最小化成本的同時實現特定流量時，可以使用流網路作為框架，這被稱為最小成本流問題（MCFP）。

12.2.3. 容量、餘額、流動性

為了更好地理解將聰從 A 點傳輸到 B 點的問題，我們需要更好地定義三個重要術語：容量、餘額和流動性。我們使用這些術語來描述支付通道路由付款的能力。

在連接 A←→B 的支付通道中：

容量

這是通過資金交易注入 2-of-2 多重簽名的聰的總量。它代表通道中持有的最大價值。通道容量由八卦協定宣告，節點可以知道。

餘額

這是每個通道合作夥伴持有的可以發送給另一個通道合作夥伴的聰的數量。A 的餘額的一部分可以朝著節點 B 的方向（A→B）發送。B 的餘額的一部分可以朝相反方向（A←B）發送。

流動性

可以實際在通道上朝一個方向發送的可用（子集）餘額。A 的流動性等於 A 的餘額減去通道儲備金和 A 承諾的任何待處理 HTLC。

網路（通過八卦公告）唯一知道的值是通道的總容量。該容量的某些未知部分作為每個合作夥伴的餘額分配。該餘額的某些子集可用於朝一個方向通過通道發送：

- capacity = balance(A) + balance(B)
- liquidity(A) = balance(A) – channel_reserve(A) – pending_HTLCs(A)

12.2.4. 餘額的不確定性

如果我們知道每個通道的確切通道餘額，我們可以使用在良好的電腦科學課程中教授的任何標準路徑尋找演算法來計算一條或多條付款路徑。但我們不知道通道餘額；我們只知道總通道容量，這是由節點在通道公告中宣傳的。為了使付款成功，通道的發送側必須有足夠的餘額。如

果我們不知道容量如何在通道合作夥伴之間分配，我們就不知道我們嘗試發送付款的方向上是否有足夠的餘額。

餘額不在通道更新中宣告有兩個原因：隱私和可擴展性。首先，宣告餘額會降低閃電網路的隱私性，因為它將允許通過對餘額變化的統計分析來監視付款。其次，如果節點（全域地）在每次付款時宣告餘額，閃電網路的擴展性將與向所有參與者廣播的鏈上比特幣交易一樣糟糕。因此，餘額不被宣告。為了在餘額不確定的情況下解決路徑尋找問題，我們需要創新的路徑尋找策略。這些策略必須與使用的路由演算法密切相關，即基於源的洋蔥路由，其中發送者有責任通過網路找到一條路徑。

不確定性問題可以用數學方式描述為 *流動性範圍*，根據已知資訊指示流動性的下限和上限。由於我們知道通道的容量，並且我們知道通道儲備餘額（每端允許的最小餘額），流動性可以定義為：

- min(liquidity) = channel_reserve
- max(liquidity) = capacity – channel_reserve

或作為範圍：

- channel_reserve <= liquidity <= (capacity – channel_reserve)

我們的通道流動性不確定性範圍是最小和最大可能流動性之間的範圍。這對網路來說是未知的，除了兩個通道合作夥伴。然而，正如我們將看到的，我們可以使用從付款嘗試返回的失敗 HTLC 來更新我們的流動性估計並減少不確定性。例如，如果我們收到一個 HTLC 失敗代碼，告訴我們一個通道無法完成一個小於我們對最大流動性估計的 HTLC，這意味著最大流動性可以更新為失敗 HTLC 的金額。簡單來說，如果我們認為流動性可以處理 *N* 聰的 HTLC，而我們發現它無法傳遞 *M* 聰（其中 *M* 更小），那麼我們可以將我們的估計更新為 *M*–1 作為上限。我們試圖找到天花板並撞上了它，所以它比我們想像的要低！

12.2.5. 路徑尋找的複雜性

通過圖找到路徑是現代電腦可以相當高效地解決的問題。如果邊的權重都相等，開發人員主要選擇廣度優先搜尋。在邊的權重不相等的情況下，使用基於 Dijkstra 演算法的演算法，例如 A*（讀作「A-star」） (https://en.wikipedia.org/wiki/A*_search_algorithm)。在我們的情況下，邊的權重可以代表路由費用。只有容量大於要發送金額的邊才會被包含在搜尋中。在這種基本形式中，閃電網路中的路徑尋找非常簡單直接。

然而，通道流動性對發送者來說是未知的。這將我們簡單的理論電腦科學問題變成了一個相當複雜的現實問題。我們現在必須在只有部分知識的情況下解決路徑尋找問題。例如，我們懷疑哪些邊可能能夠轉發付款，因為它們的容量看起來足夠大。但我們無法確定，除非我們嘗試一下或直接詢問通道所有者。即使我們能夠直接詢問通道所有者，在我們詢問其他人、計算路徑、構建洋蔥並發送它的時候，他們的餘額可能已經改變。我們不僅資訊有限，而且我們擁有的資訊是高度動態的，可能在任何時候在我們不知情的情況下改變。

12.2.6. 保持簡單

閃電節點中實現的路徑尋找機制是首先創建一個候選路徑列表，通過某種函數進行過濾和排序。然後，節點或錢包將在試錯迴圈中探測路徑（通過嘗試傳遞付款），直到找到成功傳遞付款的路徑。

這種探測是由閃電節點或錢包完成的，用戶不會直接觀察到軟體的這個過程。然而，如果付款沒有立即完成，用戶可能會懷疑正在進行探測。

雖然盲目探測不是最優的，留有很大的改進空間，但應該注意到，即使是這種簡單的策略對於較小的付款和連接良好的節點也出奇地有效。

大多數閃電節點和錢包實現通過對候選路徑列表進行排序/加權來改進這種方法。一些實現按成本（費用）或成本和容量的某種組合對候選路徑進行排序。

12.3. 路徑尋找和付款傳遞流程

路徑尋找和付款傳遞涉及幾個步驟，我們在此列出。不同的實現可能使用不同的演算法和策略，但基本步驟可能非常相似：

1. 從公告和更新創建包含每個通道容量的 *通道圖*，並過濾該圖，忽略任何容量不足以發送我們想要金額的通道。
2. 找到連接發送者和接收者的路徑。
3. 按某種權重對路徑進行排序（這可能是上一步 演算法 的一部分）。
4. 按順序嘗試每條路徑直到付款成功（試錯迴圈）。
5. 可選地使用 HTLC 失敗返回來更新我們的圖，減少 不確定性。

我們可以將這些步驟分為三個主要活動：

- 通道圖構建
- 路徑尋找（通過一些啟發式方法過濾和排序）
- 付款嘗試

如果我們使用失敗返回來更新圖，或者如果我們正在進行多部分付款（見 多部分付款），這三個活動可以在 *付款回合* 中重複。

在接下來的章節中，我們將更詳細地研究這些步驟中的每一個，以及更高級的付款策略。

12.4. 通道圖構建

在 八卦協定與通道圖 中，我們介紹了節點在其八卦中使用的三種主要訊息：node_announcement、channel_announcement 和 channel_update。這三種訊息允許任何節點以 *通道圖* 的形式逐漸構建閃電網路的「地圖」。這些訊息中的每一個都為通道圖提供了關鍵的資訊：

node_announcement

這包含閃電網路上節點的資訊，例如其節點 ID（公鑰）、網路地址（例如 IPv4/6 或 Tor）、能力/功能等。

channel_announcement

這包含兩個節點之間公開（已宣告）通道的容量和通道 ID，以及通道存在和所有權的證明。

channel_update

這包含節點的費用和時間鎖（CLTV）期望，用於在指定通道上路由出站（從該節點的角度）付款。

從數學圖的角度來看，node_announcement 是創建圖的節點或 *頂點* 所需的資訊。channel_announcement 允許我們創建代表支付通道的圖的 *邊*。由於支付通道的每個方向都有自己的餘額，我們創建一個有向圖。channel_update 允許我們納入費用和時間鎖來設置圖邊的 *成本* 或 *權重*。

根據我們將用於路徑尋找的演算法，我們可能為圖的邊建立多個不同的成本函數。

現在，讓我們忽略成本函數，只使用 node_announcement 和 channel_announcement 訊息建立一個顯示節點和通道的通道圖。

在本章中，我們將看到 Selena 如何嘗試找到一條路徑向 Rashid 支付一百萬聰。首先，Selena 正在使用來自閃電網路八卦的資訊來構建通道圖，以發現節點和通道。然後 Selena 將探索她的通道圖以找到一條向 Rashid 發送付款的路徑。

這是 *Selena 的* 通道圖。沒有所謂的 *那個* 通道圖，永遠只有 *一個通道圖*，而且它總是從構建它的節點的角度來看（見 地圖-領土關係）。

> Selena 並不只在發送付款時構建通道圖。相反，Selena 的節點 *持續* 構建和更新通道圖。從 Selena 的節點啟動並連接到網路上的任何對等節點的那一刻起，它就會參與八卦並使用每條訊息來盡可能多地了解網路。

地圖-領土關係

來自維基百科關於 地圖-領土關係 (https://en.wikipedia.org/wiki/Map%E2%80%93territory_relation) 的頁面，「地圖-領土關係描述了一個物體與其表示之間的關係，就像地理領土與其地圖之間的關係一樣。」

地圖-領土關係最好用 Lewis Carroll 的短篇故事《Sylvie and Bruno Concluded》來說明，該故事描述了一張虛構的地圖，其比例為 1:1，因此具有完美的準確性，但變得完全無用，因為如果展開它將覆蓋整個領土。

這對閃電網路意味著什麼？閃電網路是領土，通道圖是該領土的地圖。

雖然我們可以想像一個代表閃電網路完整、最新地圖的理論（柏拉圖式理想）通道圖，但這樣的地圖只是閃電網路本身。每個節點都有自己從公告構建的通道圖，而且必然是不完整的、不正確的和過時的！

地圖永遠無法完全準確地描述領土。

Selena 監聽 node_announcement 訊息並發現另外四個節點（除了預期接收者 Rashid）。結果圖代表六個節點的網路：Selena 和 Rashid 分別是發送者和接收者；Alice、Bob、Xavier 和 Yan 是中間節點。Selena 的初始圖只是一個節點列表，如 節點公告 所示。

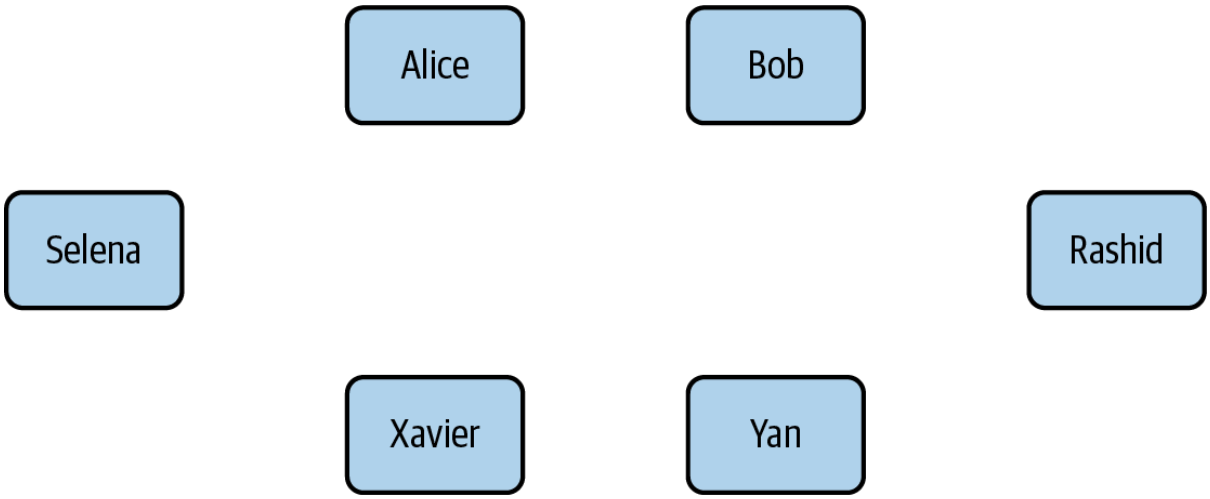

Figure 95. 節點公告

Selena 還收到七條帶有相應通道容量的 channel_announcement 訊息，允許她構建網路的基本「地圖」，如 通道圖 所示。（名稱 Alice、Bob、Selena、Xavier、Yan 和 Rashid 已被它們的首字母取代：分別是 A、B、S、X 和 R。）

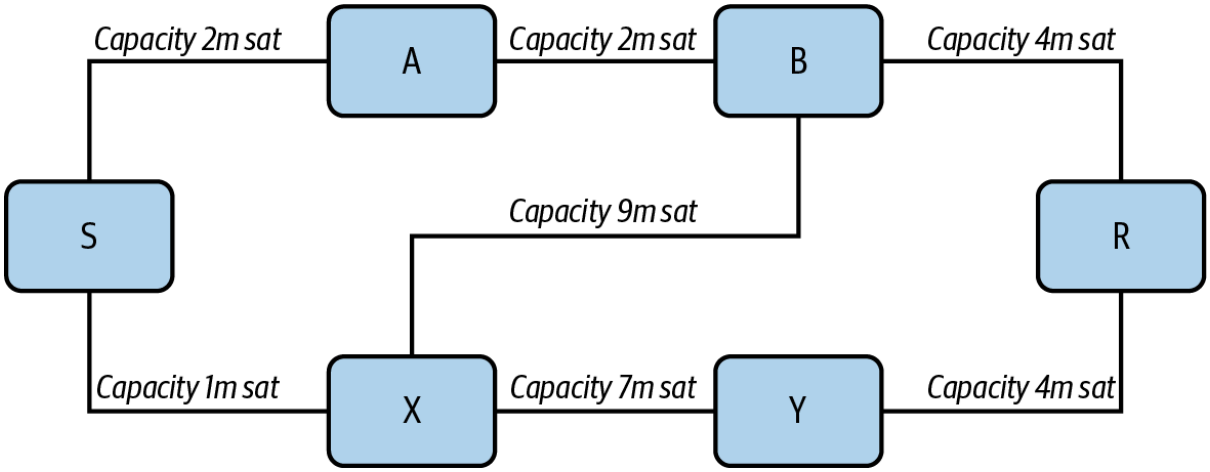

Figure 96. 通道圖

通道圖中的不確定性

正如你從 通道圖 中看到的，Selena 不知道任何通道的餘額。她的初始通道圖包含最高程度的不確定性。

但是等等：Selena 確實知道 *一些* 通道餘額！她知道她自己的節點與其他節點連接的通道的餘額。雖然這看起來不多，但它實際上是構建路徑的非常重要的資訊——Selena 知道她自己通道的實際流動性。讓我們更新通道圖以顯示這些資訊。我們將使用「?」符號來表示未知餘額，如 帶有已知和未知餘額的通道圖 所示。

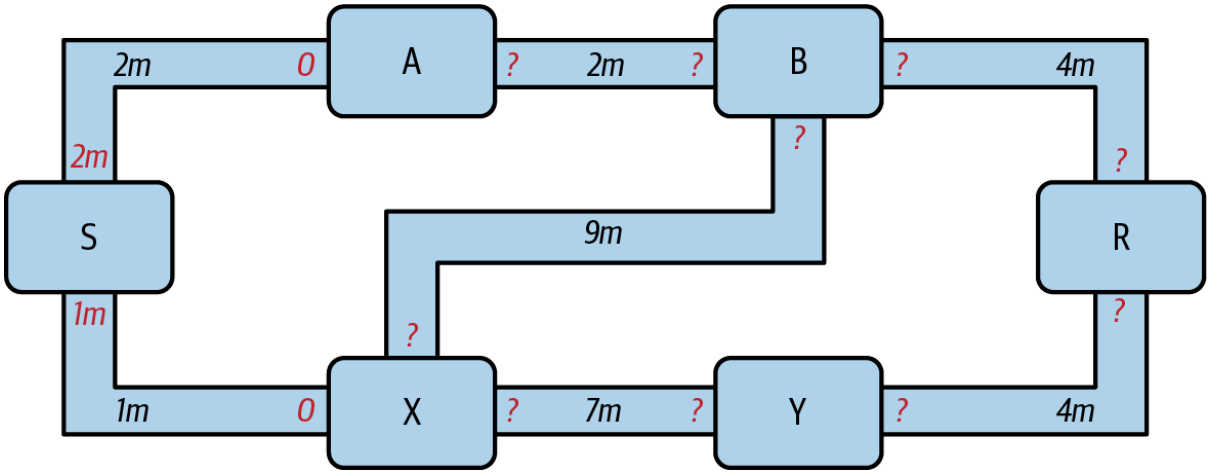

Figure 97. 帶有已知和未知餘額的通道圖

雖然「?」符號看起來很不祥，但缺乏確定性與完全無知不同。我們可以 *量化* 不確定性，並通過用我們嘗試的成功/失敗 HTLC 更新圖來 *減少* 它。

不確定性可以被量化，因為我們知道最大和最小可能的流動性，並且可以計算更小（更精確）範圍的機率。

一旦我們嘗試發送 HTLC，我們就可以了解更多關於通道餘額的資訊：如果我們成功，那麼餘額 *至少* 足以傳輸特定金額。同時，如果我們收到「臨時通道失敗」錯誤，最可能的原因是特定金額的流動性不足。

> 你可能在想，「從成功的 HTLC 學習有什麼意義？」畢竟，如果成功了我們就「完成了」。但請考慮到我們可能正在發送多部分付款的一部分。我們也可能在短時間內發送其他單部分付款。我們了解到的關於流動性的任何資訊對下一次嘗試都是有用的！

12.4.2. 流動性不確定性和機率

為了量化通道流動性的不確定性，我們可以應用機率論。付款傳遞機率的基本模型將導致一些相當明顯但重要的結論：

- 較小的付款通過路徑成功傳遞的機會更大。
- 較大容量的通道將為我們提供特定金額付款傳遞的更好機會。
- 通道（跳數）越多，成功的機會越低。

雖然這些可能很明顯，但它們有重要的影響，特別是對於多部分付款的使用（見 多部分付款）。數學並不難理解。

讓我們使用機率論來看看我們如何得出這些結論。

首先，讓我們假設容量為 c 的通道在一側有一個未知值在 (0, c) 範圍內的流動性，即「0 到 c 之間的範圍」。例如，如果容量是 5，那麼流動性將在 (0, 5) 範圍內。現在，從這裡我們看到如果我們想發送 5 聰，我們成功的機會只有 1/6（16.66%），因為我們只有在流動性恰好是 5 時才會成功。

更簡單地說，如果流動性的可能值是 0、1、2、3、4 和 5，這六個可能值中只有一個足以發送我們的付款。繼續這個例子，如果我們的付款金額是 3，那麼如果流動性是 3、4 或 5，我們就會成功。所以我們成功的機會是 3/6（50%）。用數學表示，單個通道的成功機率函數是：

$$P_c(a) = (c + 1 - a)/(c + 1)$$

其中 a 是金額，c 是容量。

從方程式中我們看到，如果金額接近 0，機率接近 1，而如果金額超過容量，機率為零。

換句話說：「較小的付款成功傳遞的機會更大」或「較大容量的通道為我們提供特定金額的更好傳遞機會」以及「你無法在容量不足的通道上發送付款」。

現在讓我們考慮通過由多個通道組成的路徑成功的機率。假設我們的第一個通道有 50% 的成功機會（P = 0.5）。然後如果我們的第二個通道有 50% 的成功機會（P = 0.5），直覺上我們的整體機會是 25%（P = 0.25）。

我們可以將此表示為一個方程式，將付款成功的機率計算為路徑中每個通道機率的乘積：

$$P_{payment} = \prod_{i=1}^{n} P_i$$

其中 P_i 是通過一條路徑或通道成功的機率，$P_{payment}$ 是通過所有路徑/通道成功付款的整體機率。

從方程式中我們看到，由於通過單個通道成功的機率總是小於或等於 1，通過多個通道的機率將 *呈指數下降*。

換句話說，「你使用的通道（跳數）越多，成功的機會越低。」

關於通道流動性不確定性的數學理論和建模有很多。關於通道流動性不確定性區間建模的基礎工作可以在 Pickhardt 等人（本書合著者）的論文《Security and Privacy of Lightning Network Payments with Uncertain Channel Balances》 (https://arxiv.org/abs/2103.08576) 中找到。

12.4.3. 費用和其他通道指標

接下來，我們的發送者將根據從中間節點收到的 channel_update 訊息向圖添加資訊。提醒一下，channel_update 包含關於通道和通道合作夥伴之一期望的豐富資訊。

在 通道圖費用和其他通道指標 中，我們看到 Selena 如何根據來自 A、B、X 和 Y 的 channel_update 訊息更新通道圖。請注意，通道 ID 和通道方向（包含在 channel_flags 中）告訴 Selena 這個更新指的是哪個通道和哪個方向。每個通道合作夥伴八卦一條或多條 channel_update 訊息來宣告他們的費用期望和關於通道的其他資訊。例如，在左上角我們看到 Alice 為通道 A—B 和方向 A 到 B 發送的 channel_update。通過這個更新，Alice 告訴網路她將收取多少費用來通過該特定通道將 HTLC 路由到 Bob。Bob 可能會為相反方向宣告一個通道更新（未在此圖中顯示），具有完全不同的費用期望。任何節點都可以隨時發送新的 channel_update 來更改費用或時間鎖期望。

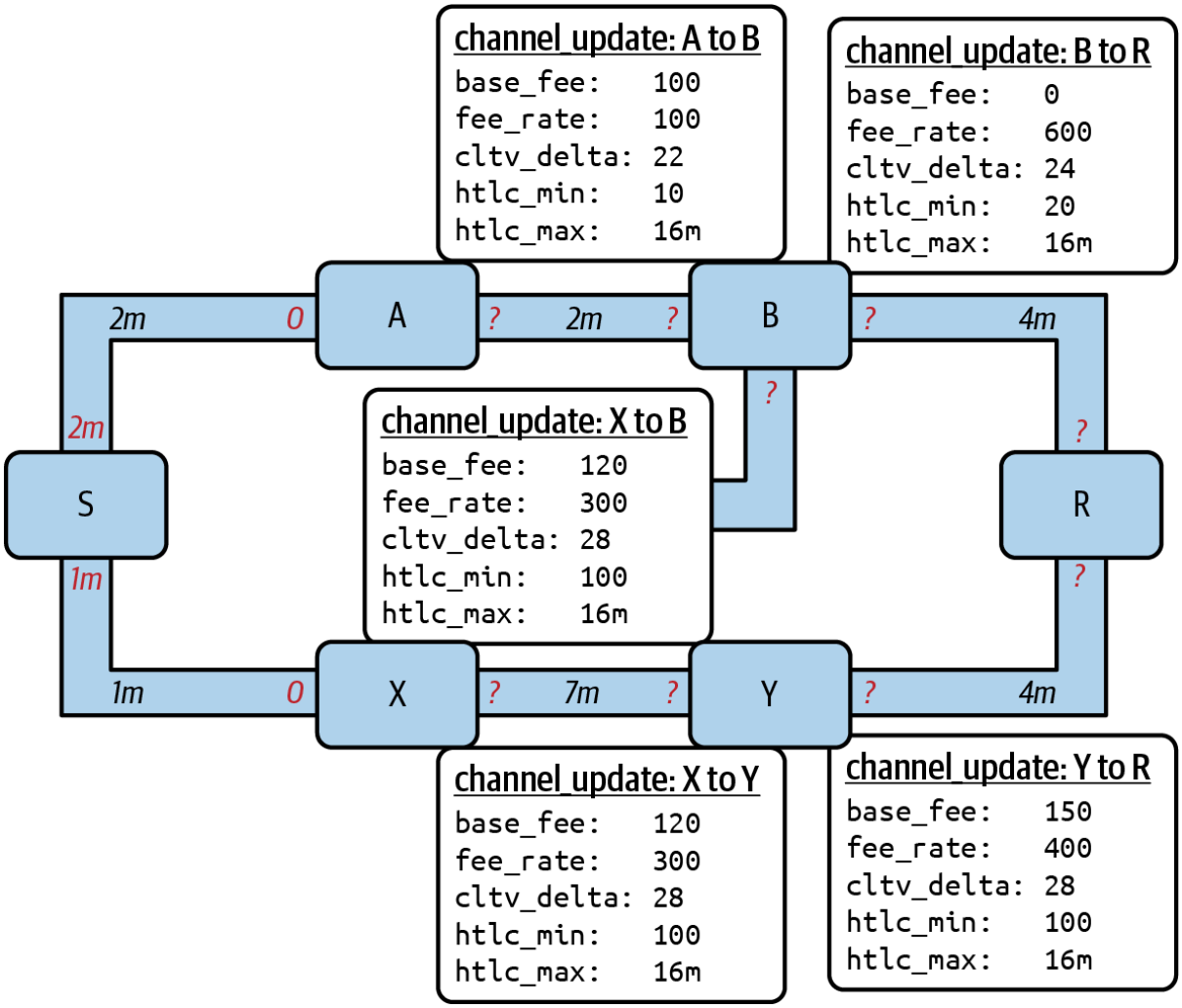

Figure 98. 通道圖費用和其他通道指標

費用和時間鎖資訊非常重要，不僅作為路徑選擇指標。正如我們在 洋蔥路由 中看到的，發送者需要在每一跳添加費用和時間鎖（cltv_expiry_delta）來製作洋蔥。計算費用的過程是從接收者到發送者 *向後* 沿著路徑進行的，因為每個中間跳期望傳入的 HTLC 具有比他們將發送給下一跳的傳出 HTLC 更高的金額和到期時間鎖。因此，例如，如果 Bob 想要 1,000 聰的費用和 30 個區塊的到期時間鎖增量來向 Rashid 發送付款，那麼該金額和到期增量必須添加到 *來自 Alice* 的 HTLC 中。

同樣重要的是要注意，通道必須具有不僅足夠付款金額而且足夠所有後續跳的累計費用的流動性。即使 Selena 到 Xavier 的通道（S→X）有足夠的流動性用於 1M 聰的付款，一旦我們考慮費用，它 *就沒有* 足夠的流動性了。我們需要知道費用，因為只有具有 *付款和所有費用* 足夠流動性的路徑才會被考慮。

12.5. 尋找候選路徑

通過像這樣的有向圖找到合適的路徑是一個研究得很充分的電腦科學問題（廣泛稱為 *最短路徑問題*），可以通過各種演算法來解決，具體取決於所需的優化和資源約束。

解決這個問題最著名的演算法是由荷蘭數學家 E. W. Dijkstra 於 1956 年發明的，簡稱為 *Dijkstra 演算法* (https://en.wikipedia.org/wiki/Dijkstra's_algorithm)。除了原始的 Dijkstra 演算法外，還有許多變體和優化，例如 A*（「A-star」） (https://en.wikipedia.org/wiki/A*_search_algorithm)，這是一種基於啟發式的演算法。

如前所述，「搜尋」*必須 向後* 應用以計算從接收者到發送者累積的費用。因此，Dijkstra、A* 或其他演算法會從接收者搜尋到發送者的路徑，使用費用、估計流動性和時間鎖增量（或某種組合）作為每一跳的成本函數。

使用這樣的演算法，Selena 計算出幾條可能到達 Rashid 的路徑，按最短路徑排序：

1. S→A→B→R
2. S→X→Y→R
3. S→X→B→R
4. S→A→B→X→Y→R

但是，正如我們之前看到的，一旦考慮費用，通道 S→X 沒有足夠的流動性用於 1M 聰的付款。所以路徑 2 和 3 不可行。這留下路徑 1 和 4 作為付款的可能路徑。

有了兩條可能的路徑，Selena 準備嘗試傳遞！

12.6. 付款傳遞（試錯迴圈）

Selena 的節點通過構建 HTLC、建立洋蔥並嘗試傳遞付款來開始試錯迴圈。對於每次嘗試，有三種可能的結果：

- 成功結果（update_fulfill_htlc）
- 錯誤（update_fail_htlc）
- 「卡住」的付款，沒有回應（既沒有成功也沒有失敗）

如果付款失敗，可以通過更新圖（更改通道的指標）並重新計算替代路徑來通過不同的路徑重試。

我們在 卡住的付款 中研究了付款「卡住」時會發生什麼。重要的細節是，卡住的付款是最糟糕的結果，因為我們無法用另一個 HTLC 重試，因為兩者（卡住的那個和重試的那個）最終都可能通過並導致雙重付款。

12.6.1. 第一次嘗試（路徑 #1）

Selena 嘗試第一條路徑（S→A→B→R）。她構建洋蔥並發送它，但收到 Bob 節點的失敗代碼。Bob 報告了 temporary channel failure，並帶有 channel_update 識別通道 B→R 為無法傳遞的通道。這次嘗試如 路徑 #1 嘗試失敗 所示。

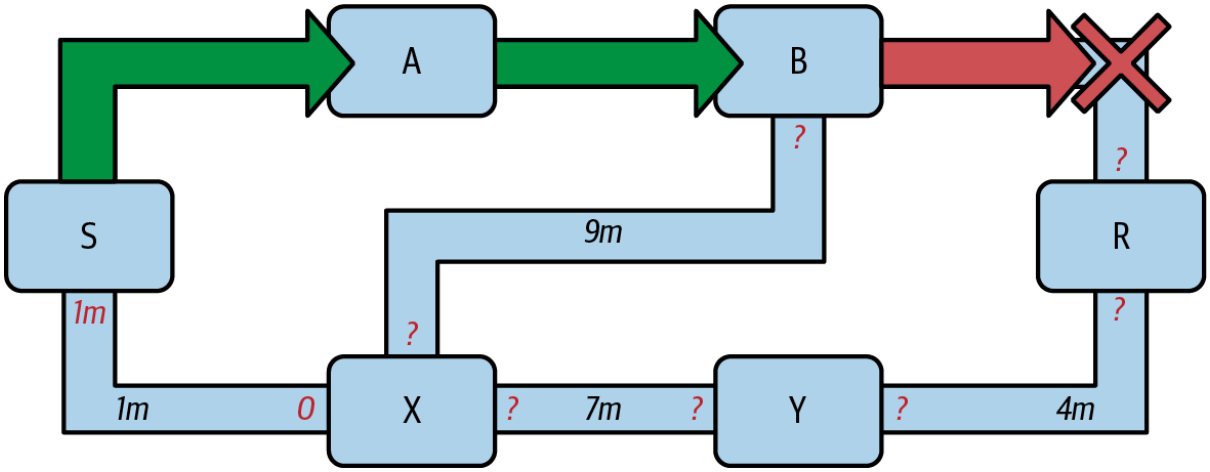

Figure 99. 路徑 #1 嘗試失敗

從失敗中學習

從這個失敗代碼，Selena 將推斷 Bob 在該通道上沒有足夠的流動性來向 Rashid 傳遞付款。重要的是，這個失敗縮小了該通道流動性的不確定性！之前，Selena 的節點假設 Bob 那邊通道的流動性在 (0, 4M) 範圍內。現在，她可以假設流動性在 (0, 999999) 範圍內。同樣，Selena 現在可以假設 Rashid 那邊該通道的流動性在 (1M, 4M) 範圍內，而不是 (0, 4M)。Selena 從這次失敗中學到了很多。

12.6.2. 第二次嘗試（路徑 #4）

現在 Selena 嘗試第四條候選路徑（S→A→B→X→Y→R）。這是一條更長的路徑，將產生更多費用，但它現在是傳遞付款的最佳選擇。

幸運的是，Selena 從 Alice 收到 update_fulfill_htlc 訊息，表示付款成功，如 路徑 #4 嘗試成功 所示。

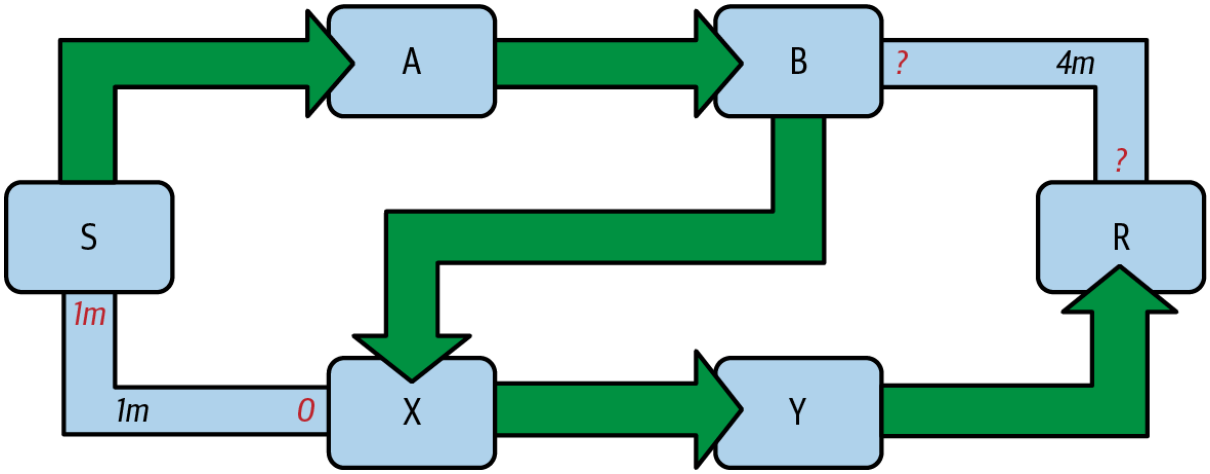

Figure 100. 路徑 #4 嘗試成功

從成功中學習

Selena 也從這次成功的付款中學到了很多。她現在知道路徑 S→A→B→X→Y→R 上的所有通道都有足夠的流動性來傳遞付款。此外，她現在知道這些通道中的每一個都已將 HTLC 金額（1M + 費用）移動到通道的另一端。這允許 Selena 重新計算該路徑中所有通道接收端的流動性範

圍，用 1M + 費用替換最小流動性。

過時的知識？

Selena 現在對閃電網路有了更好的「地圖」（至少就這七個通道而言）。這些知識對 Selena 嘗試進行的任何後續付款都很有用。

然而，當其他節點發送或路由付款時，這些知識會變得有些「過時」。Selena 永遠不會看到這些付款（除非她是發送者）。即使她參與路由付款，洋蔥路由機制意味著她只能看到一跳（她自己的通道）的變化。

因此，Selena 的節點必須考慮在假設這些知識已過時且不再有用之前保留多長時間。

12.7. 多部分付款

多部分付款（MPP） 是 2020 年在閃電網路中引入的功能，現在已經非常廣泛可用。多部分付款允許將付款拆分為多個 *部分*，這些部分作為 HTLC 通過幾條不同的路徑發送給預期接收者，同時保持整體付款的 *原子性*。在這個上下文中，原子性意味著付款的所有 HTLC 部分最終要麼全部完成，要麼整個付款失敗並且所有 HTLC 部分都失敗。不存在部分成功付款的可能性。

多部分付款是閃電網路的重大改進，因為它使發送不「適合」任何單個通道的金額成為可能，方法是將它們拆分為具有足夠流動性的較小金額。此外，與單路徑付款相比，多部分付款已被證明可以增加成功付款的機率。

> 現在 MPP 可用了，最好將單路徑付款視為 MPP 的子類別。本質上，單路徑只是大小為一的多部分。所有付款都可以被視為多部分付款，除非付款的大小和可用的流動性使得可以用單個部分傳遞。

12.7.1. 使用 MPP

MPP 不是用戶會選擇的東西，而是一種節點路徑尋找和付款傳遞策略。相同的基本步驟被實現：創建圖、選擇路徑和試錯迴圈。區別在於，在路徑選擇期間，我們還必須考慮如何拆分付款以優化傳遞。

在我們的例子中，我們可以看到使用 MPP 可以對我們的路徑尋找問題進行一些直接的改進。首先，我們可以利用 S→X 通道，它已知流動性不足以傳輸 1M 聰加上費用。通過在該通道上發送較小的部分，我們可以使用以前不可用的路徑。其次，我們有 B→R 通道的未知流動性，它不足以傳輸 1M 金額，但可能足以傳輸較小的金額。

拆分付款

根本問題是如何拆分付款。更具體地說，拆分的最佳數量和每次拆分的最佳金額是多少？

這是一個正在進行研究的領域，新的策略正在出現。多部分付款導致與單路徑付款不同的演算法方法，即使單路徑解決方案可以從多部分優化中出現（即，多部分路徑尋找演算法可能建議單路徑作為最優解決方案）。

如果你還記得，我們發現流動性/餘額的不確定性導致了一些（有些明顯的）結論，我們可以在 MPP 路徑尋找中應用，即：

- 較小的付款成功的機會更高。
- 你使用的通道越多，成功的機會（呈指數）越低。

從第一個見解，我們可能得出結論，將大額付款（例如 1 百萬聰）拆分為小額付款會增加這些較小付款中每一個成功的機會。如果我們發送較小的金額，具有足夠流動性的可能路徑數量將更大。

將這個想法推向極端，為什麼不將 1M 聰的付款拆分為一百萬個獨立的一聰部分？答案在於我們的第二個見解：由於我們將使用更多的通道/路徑來發送我們的一百萬個單聰 HTLC，我們成功的機會將呈指數下降。

如果不明顯，前面兩個見解創造了一個「最佳點」，我們可以在其中最大化我們成功的機會：拆分成較小的付款但不要拆分太多！

量化給定通道圖的大小/拆分數量的這種最優平衡超出了本書的範圍，但這是一個活躍的研究領域。一些當前的實現使用非常簡單的策略，將付款拆分為兩半、四分之一等。

> 要閱讀更多關於將付款拆分為不同大小並將它們分配到路徑時涉及的最小成本流優化問題，請參閱 René Pickhardt（本書合著者）和 Stefan Richter 的論文《Optimally Reliable & Cheap Payment Flows on the Lightning Network》 (https://arxiv.org/abs/2107.05322)。

在我們的例子中，Selena 的節點將嘗試將 1M 聰的付款拆分為 2 個部分，分別為 600k 和 400k 聰，並在 2 條不同的路徑上發送它們。這如 發送多部分付款的兩個部分 所示。

因為 S→X 通道現在可以使用，並且（對 Selena 來說幸運的是），B→R 通道有足夠的流動性用於 600k 聰，這 2 個部分在以前不可能的路徑上成功了。

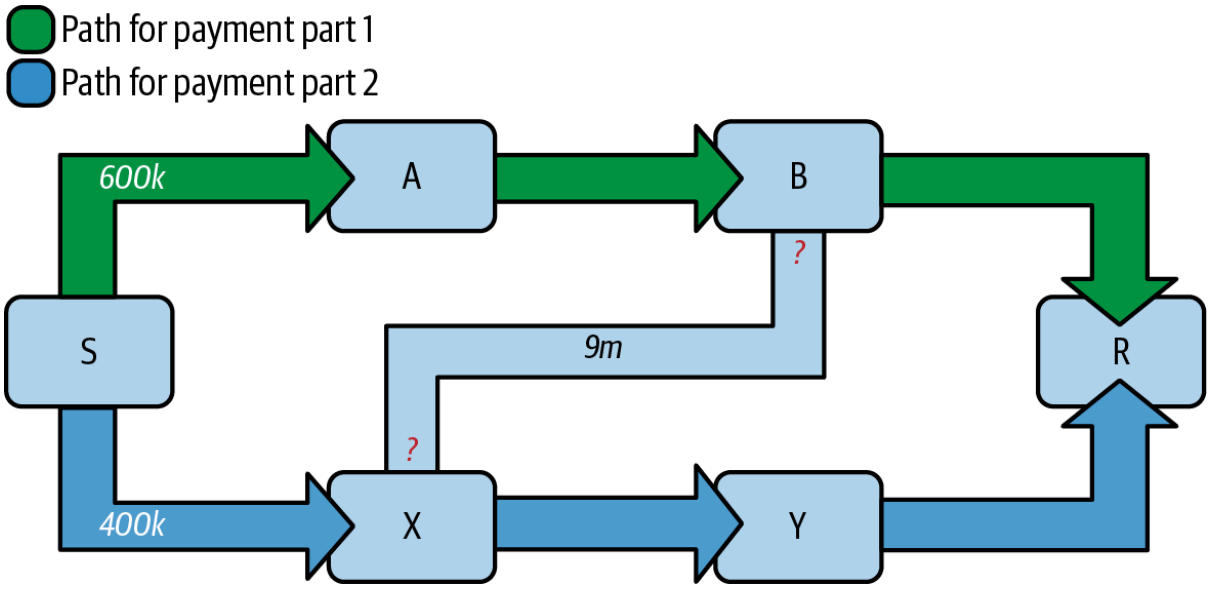

Figure 101. 發送多部分付款的兩個部分

12.7.2. 多「回合」的試錯

多部分付款導致付款傳遞的試錯迴圈有所修改。因為我們在每次嘗試中嘗試多條路徑，我們有四種可能的結果：

- 所有部分都成功，付款成功
- 一些部分成功，一些失敗並返回錯誤
- 所有部分都失敗並返回錯誤
- 一些部分「卡住」，沒有返回錯誤

在第二種情況下，一些部分失敗並返回錯誤，一些部分成功，我們現在可以 *重複* 試錯迴圈，但 *只針對剩餘金額*。

例如，假設 Selena 有一個更大的通道圖，有數百條可能的路徑到達 Rashid。她的路徑尋找演算法可能找到一個由 26 個不同大小的部分組成的最佳付款拆分。在第一輪嘗試發送所有 26 個部分後，其中 3 個部分失敗並返回錯誤。

如果這 3 個部分包含，比如說 155k 聰，那麼 Selena 將重新開始路徑尋找工作，只針對 155k 聰。下一輪可以找到完全不同的路徑（針對 155k 的剩餘金額進行優化），並將 155k 金額拆分成完全不同的拆分！

雖然 26 個拆分部分看起來很多，但在閃電網路上的測試已經成功地通過將 0.3679 BTC 拆分成 345 個部分來傳遞付款。

此外，Selena 的節點將使用從第一輪的成功和錯誤中收集的資訊來更新通道圖，以找到第二輪的最優路徑和拆分。

假設 Selena 的節點計算出發送 155k 剩餘金額的最佳方式是 6 個部分，拆分為 80k、42k、15k、11k、6.5k 和 500 聰。在下一輪中，Selena 只收到一個錯誤，表示 11k 聰部分失敗了。Selena 再次根據收集的資訊更新通道圖，並再次運行路徑尋找以發送 11k 剩餘金額。這一次，她成功了，分別用 6k 和 5k 聰的 2 個部分。

這個使用 MPP 發送付款的多回合範例如 使用 MPP 在多回合中發送付款 所示。

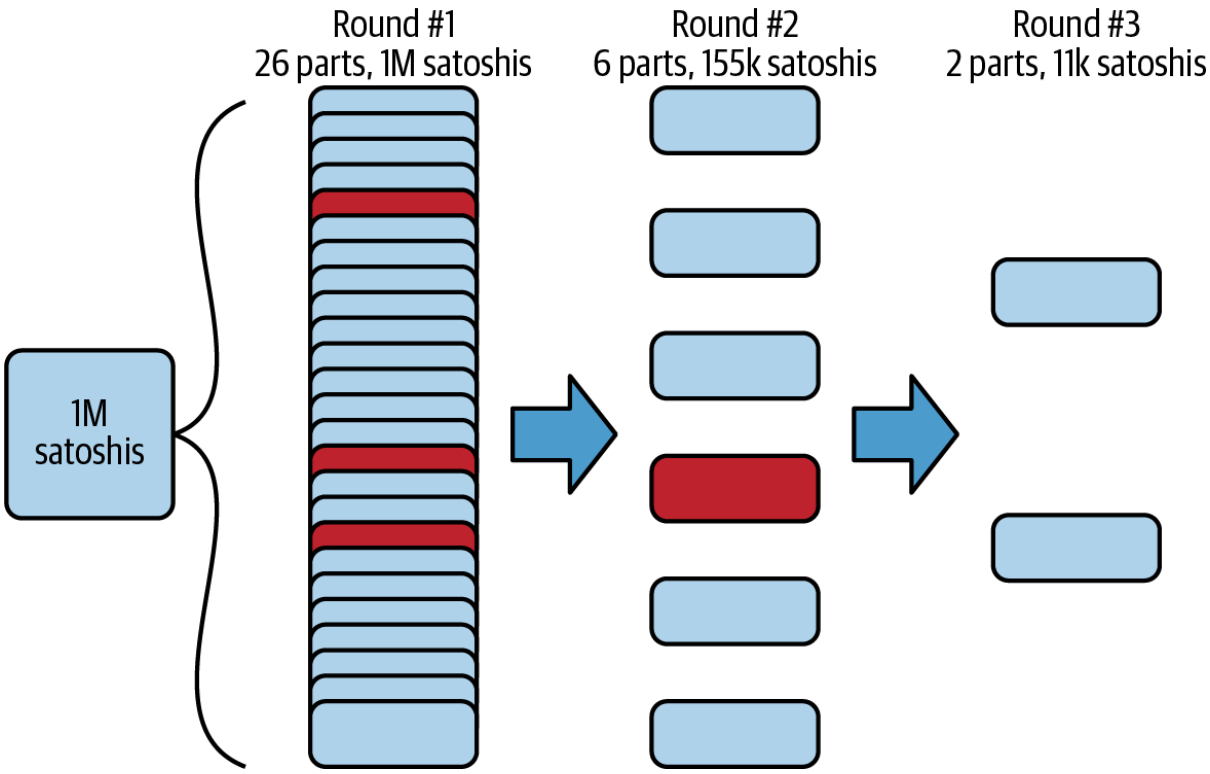

Figure 102. 使用 MPP 在多回合中發送付款

最後，Selena 的節點使用 3 回合的路徑尋找將 1M 聰分成 30 個部分發送。

12.8. 結論

在本章中，我們研究了路徑尋找和付款傳遞。我們看到了如何使用通道圖來找到從發送者到接收者的路徑。我們還看到了發送者如何嘗試在候選路徑上傳遞付款並在試錯迴圈中重複。

我們還檢視了通道流動性的不確定性（從發送者的角度）以及這對路徑尋找的影響。我們看到了如何量化不確定性並使用機率論得出一些有用的結論。我們還看到了如何通過從成功和失敗的付款中學習來減少不確定性。

最後，我們看到了新部署的多部分付款功能如何允許我們將付款拆分成部分，即使對於較大的付款也能增加成功的機率。

13. 線路協定：框架和可擴展性

在本章中，我們深入探討閃電網路的線路協定，並涵蓋協定中內建的各種可擴展性槓桿。在本章結束時，有雄心的讀者應該能夠為閃電網路編寫自己的線路協定解析器。除了能夠編寫自訂線路協定解析器外，本章的讀者還將深入理解協定中內建的各種升級機制。

13.1. 閃電協定套件中的訊息層

訊息層在本章中詳細介紹，包括「框架和訊息格式」、「類型-長度-值」編碼和「功能位元」。這些組件在協定套件中以輪廓突顯，如 閃電協定套件中的訊息層 所示。

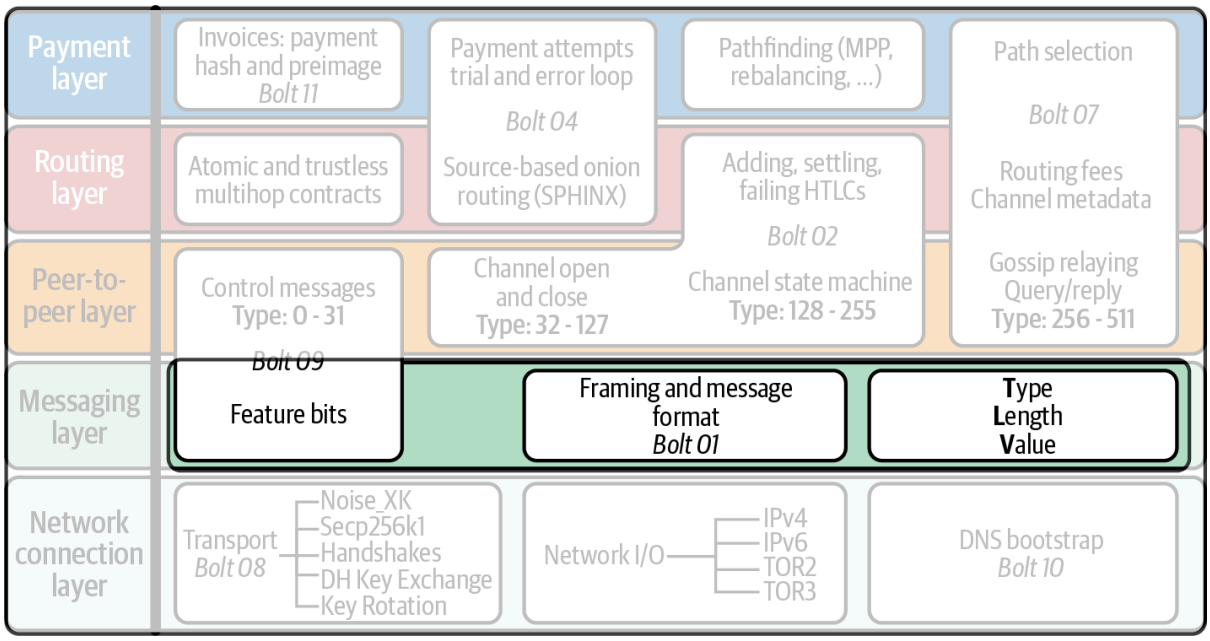

Figure 103. 閃電協定套件中的訊息層

13.2. 線路框架

我們首先描述協定中線路 *框架* 的高層結構。當我們說框架時，我們指的是位元組在線路上打包以 *編碼* 特定協定訊息的方式。如果不了解協定中使用的框架系統，線路上的位元組字串會像是一系列隨機位元組，因為沒有強加任何結構。通過應用適當的框架來解碼線路上的這些位元組，我們將能夠提取結構，並最終在我們的更高級語言中將此結構解析為協定訊息。

重要的是要注意，閃電網路是一個 *端對端加密* 的協定，線路框架本身封裝在 *加密* 的訊息傳輸層中。正如我們在 閃電網路的加密訊息傳輸 中看到的，閃電網路使用 Noise 協定的自訂變體來處理傳輸加密。在本章中，每當我們給出線路框架的範例時，我們假設加密層已經被剝離（解碼時），或者我們尚未加密我們在線路上發送的位元組集（編碼時）。

13.2.1. 高層線路框架

話雖如此，我們準備描述用於在線路上編碼訊息的高層架構：

- 線路上的訊息以 *2 位元組* 類型欄位開始，後面跟著訊息有效載荷。

- 訊息有效載荷本身最大可達 65 KB。
- 所有整數都以大端序（網路順序）編碼。
- 在定義的訊息之後跟隨的任何位元組都可以安全地忽略。

是的，就是這樣。由於協定依賴於 *封裝* 的傳輸協定加密層，我們不需要為每種訊息類型提供明確的長度。這是因為傳輸加密在 *訊息* 級別工作，所以當我們準備解碼下一條訊息時，我們已經知道訊息本身的總位元組數。使用 2 個位元組作為訊息類型（以大端序編碼）意味著協定可以有多達 2^16 – 1 或 65,535 種不同的訊息。繼續，因為我們知道所有訊息必須小於 65 KB，這簡化了我們的解析，因為我們可以使用 *固定大小* 的緩衝區，並對解析傳入線路訊息所需的總記憶體量保持強邊界。

最後一點允許一定程度的 *向後* 相容性，因為新節點能夠在舊節點（可能不理解它們）可以安全忽略的線路訊息中提供資訊。正如我們隨後看到的，這個功能與非常靈活的線路訊息可擴展性格式相結合，也允許協定實現 *向前* 相容性。

13.2.2. 類型編碼

有了這個高層背景，我們現在從最原始的層開始：解析原始類型。除了編碼整數外，閃電協定還允許編碼大量類型，包括可變長度位元組切片、橢圓曲線公鑰、比特幣地址和簽名。當我們在本章後面描述線路訊息的 *結構* 時，我們引用高層類型（抽象類型）而不是該類型的低層表示。在本節中，我們剝離這個抽象層，以確保我們未來的線路解析器能夠正確編碼/解碼任何更高層的類型。

在 高層訊息類型 中，我們將給定訊息類型的名稱映射到用於編碼/解碼該類型的高層例程。

Table 5. 高層訊息類型

高層類型	框架	註解
node_alias	32 位元組固定長度位元組切片	解碼時，如果內容不是有效的 UTF-8 字串則拒絕
channel_id	32 位元組固定長度位元組切片，將出點映射到 32 位元組值	給定一個出點，可以通過取出點的 TxID 並與索引（解釋為低 2 位元組）進行 XOR 來將其轉換為 channel_id
short_chan_id	無符號 64 位整數（uint64）	由區塊高度（24 位元）、交易索引（24 位元）和輸出索引（16 位元）打包成 8 位元組組成
milli_satoshi	無符號 64 位整數（uint64）	代表千分之一聰
satoshi	無符號 64 位整數（uint64）	比特幣的基本單位
pubkey	以 *壓縮* 格式編碼的 secp256k1 公鑰，佔用 33 位元組	在線路上佔用固定的 33 位元組長度
sig	secp256k1 橢圓曲線的 ECDSA 簽名	編碼為 *固定* 64 位元組的位元組切片，打包為 `R || S`
uint8	8 位整數	
uint16	16 位整數	
uint64	64 位整數	
[]byte	可變長度位元組切片	以表示位元組長度的 16 位整數為前綴
color_rgb	RGB 顏色編碼	編碼為一系列 8 位整數
net_addr	網路地址的編碼	以表示地址類型的 1 位元組前綴編碼，後面跟著地址主體

在下一節中，我們描述每個線路訊息的結構，包括訊息的前綴類型以及其訊息主體的內容。

13.3. 類型-長度-值訊息擴展

在本章前面，我們提到訊息最大可達 65 KB，如果在解析訊息時，剩餘額外的位元組，那麼這些位元組將被忽略。乍一看，這個要求可能看起來有些武斷；然而，這個要求允許閃電協定本身的解耦非同步演進。我們在本章末尾更多地討論這一點。但首先，我們把注意力轉向訊息末尾的那些「額外位元組」到底可以用來做什麼。

13.3.1. Protocol Buffers 訊息格式

Protocol Buffers（Protobuf）訊息序列化格式最初是 Google 內部使用的格式，現已發展成為全球開發人員使用的最流行的訊息序列化格式之一。Protobuf 格式描述了訊息（通常是與 API 相關的某種資料結構）如何在線路上編碼並在另一端解碼。數十種語言中存在幾種「Protobuf 編譯器」，它們充當橋樑，允許任何語言編碼的 Protobuf 能夠被另一種語言中的相容解碼器解碼。這種跨語言的資料結構相容性允許廣泛的創新，因為可以跨語言和抽象邊界傳輸結構甚至類型化的資料結構。

Protobuf 還以其在處理底層訊息結構變化方面的靈活性而聞名。只要遵守欄位編號架構，較新的 Protobuf 編寫器就可以在 Protobuf 中包含任何較舊讀取器可能不知道的資訊。當舊讀取器遇到新的序列化格式時，如果有它不理解的類型/欄位，那麼它只是 *忽略* 它們。這允許舊客戶端和新客戶端共存，因為所有客戶端都可以解析較新訊息格式的某些部分。

13.3.2. 向前和向後相容性

Protobuf 在開發人員中非常流行，因為它們內建支援向前和向後相容性。大多數開發人員可能熟悉向後相容性的概念。簡單來說，該原則指出，對訊息格式或 API 的任何更改都應以不破壞對舊客戶端支援的方式進行。在我們前面的 Protobuf 可擴展性範例中，通過確保對 Protobuf 格式的新添加不會破壞舊讀取器的已知部分來實現向後相容性。另一方面，向前相容性對於非同步更新同樣重要；然而，它不太為人所知。為了使更改向前相容，客戶端只需忽略他們不理解的任何資訊。升級比特幣共識系統的軟分叉機制可以說是向前和向後相容的：任何不更新的客戶端仍然可以使用比特幣，如果他們遇到任何不理解的交易，那麼他們只是忽略它們，因為他們的資金沒有使用那些新功能。

13.4. 類型-長度-值格式

為了能夠以向前和向後相容的方式升級訊息，除了功能位元（稍後會詳細介紹）外，閃電網路還使用一種自訂訊息序列化格式，簡稱為類型-長度-值，或簡稱 TLV。該格式受到廣泛使用的 Protobuf 格式的啟發，借用了許多概念，同時大大簡化了實現以及與訊息解析互動的軟體。好奇的讀者可能會問，「為什麼不直接使用 Protobuf？」作為回應，閃電開發人員會說，我們能夠擁有 Protobuf 可擴展性的最佳特性，同時也有更小的實現優勢，因此攻擊面更小。截至 3.15.6 版本，Protobuf 編譯器有超過 656,671 行程式碼。相比之下，LND 的 TLV 訊息格式實現只有 2.3k 行程式碼（包括測試）。

有了必要的背景介紹，我們現在準備詳細描述 TLV 格式。TLV 訊息擴展被稱為單個 TLV 記錄 的串流。單個 TLV 記錄有三個組件：記錄的類型、記錄的長度，最後是記錄的不透明值：

`type`
: 表示正在編碼的記錄名稱的整數

`length`
: 記錄的長度

`value`
: 記錄的不透明值

`type` 和 `length` 都使用受比特幣 P2P 協定中使用的可變大小整數（varint）啟發的可變大小整數編碼，簡稱為 `BigSize`。

13.4.1. BigSize 整數編碼

在其最完整的形式中，`BigSize` 整數可以表示最多 64 位元的值。與比特幣的 varint 格式相比，`BigSize` 格式使用大端序位元組排序來編碼整數。

`BigSize` varint 有兩個組件：判別式和主體。在 `BigSize` 整數的上下文中，判別式向解碼器傳達後面跟隨的可變大小整數的大小。請記住，可變大小整數的獨特之處在於它們允許解析器使用比較大整數更少的位元組來編碼較小的整數，從而節省空間。`BigSize` 整數的編碼遵循以下四個選項之一：

1. 如果值小於 `0xfd`（`253`）：則判別式實際上不使用，編碼只是整數本身。這允許我們編碼非常小的整數而沒有額外開銷。
2. 如果值小於或等於 `0xffff`（`65535`）：判別式編碼為 `0xfd`，這表示後面的值大於 `0xfd`，但小於 `0xffff`。然後數字編碼為 16 位整數。包括判別式，我們可以使用 3 個位元組編碼大於 253 但小於 65,535 的值。
3. 如果值小於 `0xffffffff`（`4294967295`）：判別式編碼為 `0xfe`。主體使用 32 位整數編碼，包括判別式，我們可以使用 5 個位元組編碼小於 `4,294,967,295` 的值。
4. 否則，我們只是將值編碼為完整大小的 64 位整數。

13.4.2. TLV 編碼約束

在 TLV 訊息的上下文中，低於 `2^16` 的記錄類型被稱為 *保留* 供將來使用。超出此範圍的類型用於更高級應用協定使用的「自訂」訊息擴展。

記錄的 `value` 取決於 `type`。換句話說，它可以採取任何形式，因為解析器將嘗試根據類型本身的上下文來解釋它。

13.4.3. TLV 規範編碼

Protobuf 格式的一個問題是，當由兩個不同版本的編譯器編碼時，相同訊息的編碼可能輸出完全不同的位元組集。在閃電的上下文中，這種非規範編碼的情況是不可接受的，因為許多訊息包含訊息摘要的簽名。如果一條訊息可能以兩種不同的方式編碼，那麼通過使用線路上稍微不同的位元組集重新編碼訊息，可能會無意中破壞簽名的認證。

為了確保所有編碼的訊息都是規範的，在編碼時定義了以下約束：

- TLV 串流中的所有記錄必須按嚴格遞增類型的順序編碼。
- 所有記錄必須最小化編碼 `type` 和 `length` 欄位。換句話說，必須始終使用整數的最小 `BigSize` 表示。
- 每個 `type` 在給定的 TLV 串流中只能出現一次。

除了這些編碼約束外，還根據給定 `type` 整數的 *元數* 定義了一系列更高級的解釋要求。我們在本章末尾更深入地探討這些細節，一旦我們描述了閃電協定在實踐和理論上是如何升級的。

13.5. 功能位元和協定可擴展性

因為閃電網路是一個去中心化系統，沒有單一實體可以對系統的所有用戶強制執行協定更改或修改。這一特性也出現在其他去中心化網路中，如比特幣。然而，與比特幣不同的是，壓倒性的共識 *不是* 改變閃電網路子集所必需的。閃電網路能夠隨意演進，而無需強烈的協調要求，因為與比特幣不同，閃電網路中不需要全域共識。由於這個事實以及閃電網路中嵌入的幾種升級機制，只有希望使用這些新閃電網路功能的參與者需要升級，然後他們就能夠彼此互動。

在本節中，我們探索開發人員和用戶能夠設計和部署閃電網路新功能的各種方式。原始閃電網路的設計者知道網路和底層協定有許多可能的未來方向。因此，他們確保在系統中實現了幾種可擴展性機制，可用於以解耦、非同步和去中心化的方式部分或完全升級它。

13.5.1. 功能位元作為升級發現機制

精明的讀者可能已經注意到閃電協定中包含功能位元的各個位置。*功能位元* 是一個位元欄位，可用於宣傳對可能的網路協定更新的理解或遵守。功能位元通常成對分配，這意味著每個潛在的新功能/升級總是在位元欄位中定義兩個位元。一個位元表示宣傳的功能是 *可選的*，意味著該節點知道該功能並可以使用它，但不認為它對正常操作是必需的。另一個位元表示該功能是 *必需的*，意味著如果潛在的對等節點不理解該功能，該節點將不會繼續操作。

使用這兩個位元（可選和必需），我們可以構建一個簡單的相容性矩陣，節點/用戶可以參考它來確定對等節點是否與所需功能相容，如 功能位元相容性矩陣 所示。

Table 6. 功能位元相容性矩陣

位元類型	遠端可選	遠端必需	遠端未知
本地可選	✅	✅	✅
本地必需	✅	✅	❌
本地未知	✅	❌	❌

從這個簡化的相容性矩陣中，我們可以看到，只要對方知道我們的功能位元，那麼我們就可以使用該協定與他們互動。如果對方甚至不知道我們指的是什麼位元 *並且* 他們需要該功能，那麼我們與他們不相容。在網路中，可選功能使用 *奇數位元號* 發信號，而必需功能使用 *偶數位元號* 發信號。例如，如果一個對等節點發信號說他們知道一個使用位元 15 的功能，那麼我們知道這是一個可選功能，即使我們不知道該功能，我們也可以與他們互動或回應他們的訊息。如果他們使用位元 16 發信號該功能，那麼我們知道這是一個必需功能，除非我們的節點也理解該功能，否則我們無法與他們互動。

閃電開發人員想出了一個易於記憶的短語來編碼這個矩陣：「奇數是可以的」（it’ s OK to be odd）。這個簡單的規則允許協定內的豐富互動集，因為兩個功能位元向量之間的簡單位元遮罩操作允許對等節點確定某些互動是否彼此相容。換句話說，功能位元用作升級發現機制：它們很容易讓對等節點根據可選、必需和未知功能位元的概念來理解它們是否相容。

功能位元在協定中的 `node_announcement` 、 `channel_announcement` 和 `init` 訊息中找到。因此，這三種訊息可用於在網路內發信號傳輸中協定更新的知識和/或理解。 `node_announcement` 訊息中的功能位元可以讓對等節點確定其 *連接* 是否相容。 `channel_announcement` 訊息中的功能位元允許對等節點確定給定的付款類型或 HTLC 是否可以通過給定的對等節點傳輸。 `init` 訊息中的功能位元允許對等節點理解他們是否可以維持連接，以及為給定連接的生命週期協商了哪些功能。

13.5.2. TLV 用於向前和向後相容性

正如我們在本章前面學到的，TLV 記錄可用於以向前和向後相容的方式擴展訊息。隨著時間的推移，這些記錄已被用來擴展現有訊息，而不會通過利用訊息中超出已知位元組集的「未定義」區域來破壞協定。

例如，原始的閃電協定沒有「最大金額 HTLC」的概念，這是由路由策略規定的可以通過通道傳輸的最大金額。後來， `max_htlc` 欄位被添加到 `channel_update` 訊息中，以隨著時間的推移逐步引入這個概念。接收設置了這樣欄位的 `channel_update` 但甚至不知道該升級存在的對等節點不受變化影響，但如果他們的 HTLC 超過限制，他們的 HTLC 會被拒絕。另一方面，較新的對等節點能夠解析、驗證和利用新欄位。

熟悉比特幣中軟分叉概念的人現在可能會看到兩種機制之間的一些相似之處。與比特幣共識級軟分叉不同，閃電網路的升級不需要壓倒性的共識才能被採用。相反，至少只有網路中的兩個對等節點需要理解新的升級才能開始使用它。通常這兩個對等節點可能是付款的接收者和發送

者，或者可能是新支付通道的通道合作夥伴。

13.5.3. 升級機制的分類

與其在網路中有一個廣泛使用的單一升級機制（如比特幣的軟分叉），閃電網路中存在幾種可能的升級機制。在本節中，我們列舉這些升級機制並提供過去使用它們的真實範例。

內部網路升級

我們從需要最多協定級協調的升級類型開始：內部網路升級。內部網路升級的特點是需要預期付款路徑中的 *每個節點* 都理解新功能。這種升級類似於網際網路中需要在升級的核心中繼部分進行硬體級升級的任何升級。然而，在閃電網路的上下文中，我們處理的是純軟體，所以這種升級更容易部署，但它們仍然比網路中的任何其他升級機制需要更多的協調。

網路中這種升級的一個例子是引入 TLV 編碼用於洋蔥封包中編碼的路由資訊。之前的格式使用硬編碼的固定長度訊息格式來傳達諸如下一跳之類的資訊。因為這種格式是固定的，這意味著新的協定級升級是不可能的。轉向更靈活的 TLV 格式意味著在此升級之後，任何修改每一跳傳達的資訊類型的功能都可以隨意推出。

值得一提的是，TLV 洋蔥升級是一種「軟」內部網路升級，因為如果付款沒有使用超出新路由資訊編碼的任何新功能，那麼付款可以使用混合節點集傳輸。

端對端升級

為了與內部網路升級形成對比，在本節中我們描述 *端對端* 網路升級。這種升級機制與內部網路升級的不同之處在於，它只需要付款的「兩端」，即發送者和接收者，進行升級。

這種類型的升級允許在網路內進行廣泛的不受限制的創新。由於網路內付款的洋蔥加密性質，在網路中心轉發 HTLC 的那些節點甚至可能不知道正在使用新功能。

網路中端對端升級的一個例子是多部分付款（MPP）的推出。MPP 是一種協定級功能，使單個付款能夠分成多個部分或路徑，在接收者處組裝進行結算。MPP 的推出與一個新的 `node_announcement` 級功能位元相結合，該功能位元表示接收者知道如何處理部分付款。假設發送者和接收者彼此了解（可能通過 BOLT #11 發票），那麼他們就能夠使用新功能而無需任何進一步協商。

端對端升級的另一個例子是網路中部署的各種類型的 *自發* 付款。一種早期類型的自發付款稱為 *keysend*，它通過簡單地將付款的原像放在加密洋蔥中來工作。收到後，目的地將解密原像，然後使用它來結算付款。因為整個封包是端對端加密的，所以這種付款類型是安全的，因為沒有任何中間節點能夠完全解開洋蔥以揭示付款原像。

13.5.4. 通道構建級更新

最後一類廣泛的更新是那些發生在通道構建級別的更新，但不會修改網路中廣泛使用的 HTLC 結構。當我們說通道構建時，我們指的是通道是如何資金充足或創建的。例如，eltoo 通道類型可以使用新的 `node_announcement` 級功能位元以及 `channel_announcement` 級功能位

元在網路中推出。只有通道兩側的兩個對等節點需要理解和宣傳這些新功能。然後這個通道對可以用來轉發任何付款類型，前提是通道支援它。

另一個是 *錨定輸出* 通道格式，它允許通過比特幣的子付父（CPFP）費用管理機制來提高承諾費用。

13.6. 結論

閃電網路的線路協定極其靈活，允許快速創新和互操作性，而無需嚴格的共識。這是閃電網路經歷更快發展並吸引許多開發人員的原因之一，否則這些開發人員可能會發現比特幣的開發風格過於保守和緩慢。

14. 閃電網路的加密訊息傳輸

在本章中，我們將回顧閃電網路的_加密訊息傳輸_，有時也稱為*Brontide 協定*，它允許節點建立端對端加密通訊、身份驗證和完整性檢查。

本章的部分內容包含一些關於加密協定和閃電網路加密傳輸中使用的加密演算法的高度技術細節。如果你對這些細節不感興趣，可以跳過該部分。

14.1. 閃電協定套件中的加密傳輸

傳輸元件及其若干組件顯示在 閃電協定套件中的加密訊息傳輸 中網路連線層的最左側部分。

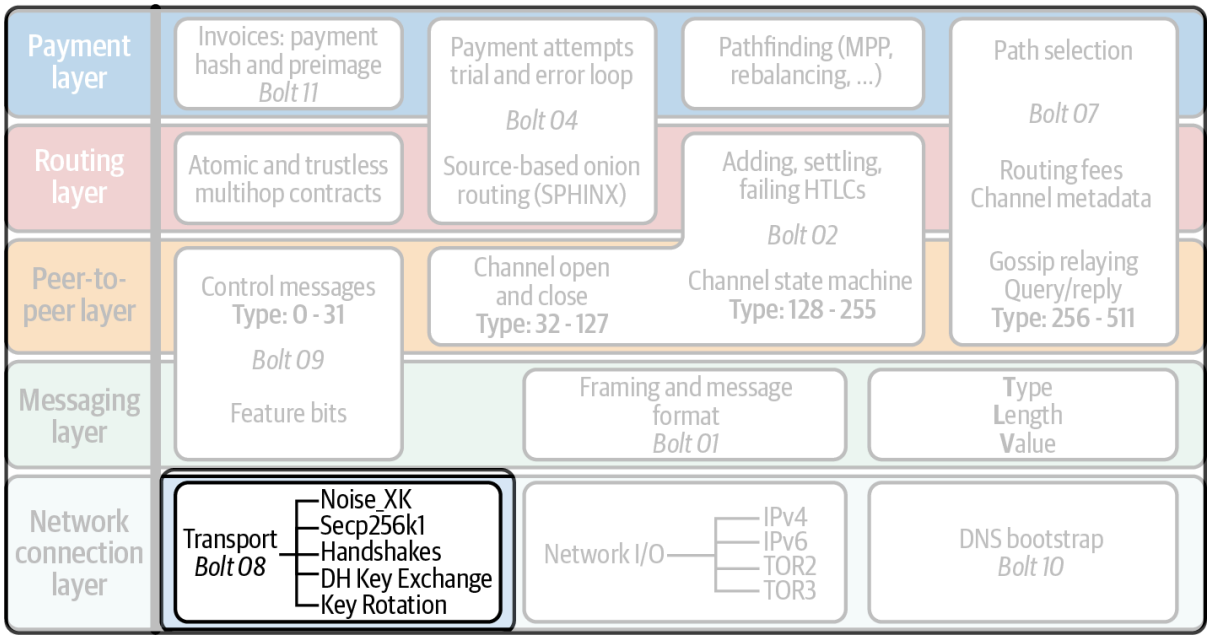

Figure 104. 閃電協定套件中的加密訊息傳輸

14.2. 簡介

與原始的比特幣 P2P 網路不同，閃電網路中的每個節點都由唯一的公鑰標識，該公鑰作為其身份。預設情況下，此公鑰用於端對端加密網路內的_所有_通訊。在協定最低層預設啟用加密確保所有訊息都經過身份驗證、免受中間人（MITM）攻擊和第三方監聽，並確保在基本傳輸層面的隱私。在本章中，我們將詳細了解閃電網路使用的加密協定。完成本章後，讀者將熟悉加密訊息協定的最新技術，以及此類協定為網路提供的各種特性。值得一提的是，加密訊息傳輸的核心對於在閃電網路環境中的使用是_不可知的_。因此，閃電網路使用的自訂加密訊息傳輸可以應用到任何需要雙方之間加密通訊的環境中。

14.3. 通道圖作為去中心化公鑰基礎設施

正如我們在 在支付通道網路上路由 中學到的，每個節點都有一個長期身份，在路徑尋找過程中用作頂點的識別碼，也用於與建立洋蔥加密路由封包相關的非對稱加密操作。這個作為節點長期身份的公鑰包含在 DNS 引導回應中，也嵌入在通道圖中。因此，在節點嘗試連接到 P2P

網路上的另一個節點之前，它已經知道要連接的節點的公鑰。

此外，如果被連接的節點已經在圖中擁有一系列公開通道，那麼連接節點能夠進一步驗證該節點的身份。因為整個通道圖是完全經過身份驗證的，所以可以將其視為一種去中心化公鑰基礎設施（PKI）：要註冊一個金鑰，必須在比特幣區塊鏈上開設一個公開通道，一旦節點不再擁有任何公開通道，它們就相當於被從 PKI 中移除了。

因為閃電網路是一個去中心化網路，所以不能指定任何一個中央機構來負責在網路中提供公鑰身份。取代中央機構的是，閃電網路使用比特幣區塊鏈作為 Sybil 攻擊緩解機制，因為在網路上獲得身份有實際成本：在區塊鏈中建立通道所需的費用，以及分配給通道的資本的機會成本。在基本上建立特定領域 PKI 的過程中，閃電網路能夠顯著簡化其加密傳輸協定，因為它不需要處理 TLS（傳輸層安全協定）帶來的所有複雜性。

14.4. 為什麼不用 TLS？

熟悉 TLS 系統的讀者可能會想：儘管現有 PKI 系統有缺點，為什麼不使用 TLS？確實，「自簽名證書」可以通過簡單地在一組節點之間斷言給定公鑰的身份來有效地繞過現有的全球 PKI 系統。然而，即使排除現有的 PKI 系統，TLS 也有幾個缺點促使閃電網路的創建者選擇了更緊湊的自訂加密協定。

首先，TLS 是一個已經存在了幾十年的協定，因此隨著傳輸加密領域的新進展而不斷演進。然而，隨著時間的推移，這種演進導致協定的規模和複雜性急劇膨脹。在過去幾十年中，TLS 中發現並修補了多個漏洞，每次演進都進一步增加了協定的複雜性。由於協定的歷史悠久，存在多個版本和迭代，這意味著客戶端需要理解許多先前的協定迭代才能與公共網際網路的大部分進行通訊，進一步增加了實作複雜性。

過去，在廣泛使用的 SSL/TLS 實作中發現了多個記憶體安全漏洞。在每個閃電節點中打包這樣的協定將增加暴露在公共點對點網路中的節點的攻擊面。為了增加整個網路的安全性並最小化可利用的攻擊面，閃電網路的創建者選擇採用 Noise 協定框架。Noise 作為一個協定，內化了幾十年來對 TLS 協定持續審查所學到的安全和隱私經驗教訓。在某種程度上，Noise 的存在使社群能夠有效地「重新開始」，使用一個更緊湊、更簡化的協定，同時保留 TLS 的所有額外優點。

14.5. Noise 協定框架

Noise 協定框架是一個現代、可擴展且靈活的訊息加密協定，由 Signal 協定的創建者設計。Signal 協定是世界上使用最廣泛的訊息加密協定之一。它被 Signal 和 WhatsApp 使用，這兩者加起來被全球超過十億人使用。Noise 框架是學術界和訊息加密協定行業幾十年演進的結果。閃電網路使用 Noise 協定框架來實作所有節點相互通訊時使用的_面向訊息_的加密協定。

使用 Noise 的通訊會話有兩個不同的階段：握手階段和訊息傳遞階段。在雙方可以相互通訊之前，他們首先需要達成一個只有他們知道的共享秘密，該秘密將用於加密和驗證相互發送的訊息。一種經過身份驗證的金鑰協議用於在雙方之間達成最終的共享金鑰。在 Noise 協定的背景

下，這種經過身份驗證的金鑰協議被稱為_握手_。一旦握手完成，兩個節點就可以開始相互發送加密訊息。每次節點需要連接或重新連接時，都會執行握手協定的新迭代，確保實現前向保密性（洩露先前副本的金鑰不會危及任何未來的副本）。

因為 Noise 協定允許協定設計者從多種加密原語中選擇，例如對稱加密和公鑰加密，所以習慣上每種 Noise 協定的變體都有一個唯一的名稱。本著「Noise」的精神，協定的每種變體都選擇了一個源自某種「噪音」的名稱。在閃電網路的背景下，使用的 Noise 協定變體有時被稱為 Brontide。*brontide* 是一種低沉的隆隆聲，類似於在非常遠的地方雷暴時聽到的聲音。

14.6. 閃電網路加密傳輸詳解

在本節中，我們將分解閃電網路加密傳輸協定，並深入研究用於在節點之間建立加密、經過身份驗證和完整性保證通訊的加密演算法和協定的細節。如果你覺得這種程度的細節令人生畏，可以跳過本節。

14.6.1. Noise_XK：閃電網路的 Noise 握手

Noise 協定非常靈活，它提供了多種握手方式，每種都有不同的安全和隱私特性，供潛在的協定實作者選擇。深入探討每種握手及其各種權衡超出了本章的範圍。話雖如此，閃電網路使用一種稱為 `Noise_XK` 的特定握手。這種握手提供的獨特特性是*身份隱藏*：為了讓節點發起與另一個節點的連接，它必須首先知道其公鑰。從機制上講，這意味著回應者的公鑰實際上從未在握手過程中傳輸。相反，使用一系列巧妙的橢圓曲線 Diffie-Hellman（ECDH）和訊息驗證碼（MAC）檢查來驗證回應者的身份。

14.6.2. 握手符號和協定流程

每個握手通常由幾個步驟組成。在每個步驟中，一些（可能）加密的材料被發送給對方，執行一個 ECDH（或多個），握手的結果被「混合」到協定_副本_中。這個副本用於驗證協定的每個步驟，並有助於阻止一種中間人攻擊。在握手結束時，產生兩個金鑰 `ck` 和 `k`，用於在會話的整個生命週期中加密訊息（`k`）和輪換金鑰（`ck`）。

在握手的背景下，`s` 通常是一個長期靜態公鑰。在我們的情況下，使用的公鑰加密系統是橢圓曲線系統，使用 `secp256k1` 曲線實例化，這也在比特幣的其他地方使用。在整個握手過程中會生成多個臨時金鑰。我們使用 `e` 來表示新的臨時金鑰。兩個金鑰之間的 ECDH 操作表示為兩個金鑰的連接。例如，`ee` 表示兩個臨時金鑰之間的 ECDH 操作。

14.6.3. 高層概述

使用前面列出的符號，我們可以簡潔地描述 `Noise_XK` 如下：

```
    Noise_XK(s, rs):
       <- rs
       ...
       -> e, e(rs)
       <- e, ee
       -> s, se
```

協定從回應者的靜態金鑰（`rs`）「預傳輸」給發起者開始。在執行握手之前，發起者要生成自己的靜態金鑰（`s`）。在握手的每個步驟中，所有透過線路發送的材料和發送/使用的金鑰都會逐漸雜湊到_握手摘要_ `h` 中。這個摘要在握手期間從不透過線路發送，而是在 AEAD（關聯資料認證加密）透過線路發送時用作「關聯資料」。*關聯資料*（AD）允許加密協定在密文封包旁邊驗證額外資訊。在其他領域，AD 可能是域名或封包的明文部分。

`h` 的存在確保如果傳輸的握手訊息的一部分被替換，對方會注意到。在每個步驟中，都會檢查 MAC 摘要。如果 MAC 檢查成功，那麼接收方知道到目前為止握手已經成功。否則，如果 MAC 檢查失敗，那麼握手過程就失敗了，應該終止連接。

協定還為每個握手訊息添加了一個新的資料：協定版本。初始協定版本是 `0`。在撰寫本文時，尚未建立新的協定版本。因此，如果節點收到 `0` 以外的版本，則應拒絕握手發起嘗試。

至於加密原語，SHA-256 用作選擇的雜湊函數，`secp256k1` 作為橢圓曲線，`ChaChaPoly-130` 作為 AEAD（對稱加密）構造。

Noise 協定的每個變體都有一個唯一的 ASCII 字串用於引用它。為了確保雙方使用相同的協定變體，ASCII 字串被雜湊成摘要，用於初始化起始握手狀態。在閃電網路的背景下，描述協定的 ASCII 字串是 `Noise_XK_secp256k1_ChaChaPoly_SHA256`。

14.6.4. 三幕握手

握手部分可以分為三個不同的「幕」。整個握手在發起者和回應者之間需要 1.5 個往返。在每一幕中，雙方之間發送一條訊息。握手訊息是一個固定大小的有效載荷，前綴是協定版本。

Noise 協定使用物件導向的符號來描述每個步驟的協定。在設定握手狀態期間，每一方都會初始化以下變數：

`ck`

鏈結金鑰。這個值是所有先前 ECDH 輸出的累積雜湊。在握手結束時，`ck` 用於派生閃電訊息的加密金鑰。

`h`

握手雜湊。這個值是握手過程中到目前為止發送和接收的_所有_握手資料的累積雜湊。

`temp_k1`、`temp_k2`、`temp_k3`

中間金鑰。這些用於在每個握手訊息結束時加密和解密零長度 AEAD 有效載荷。

`e`

一方的_臨時金鑰對_。對於每個會話，節點必須使用強加密隨機性生成新的臨時金鑰。

`s`

一方的_靜態金鑰對_（`ls` 表示本地，`rs` 表示遠端）。

給定這個握手加訊息會話狀態，我們將定義一系列函數來操作握手和訊息狀態。在描述握手協定時，我們將以類似於虛擬碼的方式使用這些變數，以減少協定中每個步驟解釋的冗長程度。我們將握手的_功能_原語定義為：

`ECDH(k, rk)`

使用 `k`（一個有效的 `secp256k1` 私鑰）和 `rk`（一個有效的公鑰）執行橢圓曲線 Diffie-Hellman 操作。

返回值是生成點的壓縮格式的 SHA-256。

`HKDF(salt,ikm)`

`RFC 5869` 中定義的函數，使用零長度 `info` 欄位進行評估。

所有 `HKDF` 的呼叫都隱式地使用提取和擴展元件返回 64 位元組的加密隨機性。

`encryptWithAD(k, n, ad, plaintext)`

輸出 `encrypt(k, n, ad, plaintext)`。

其中 `encrypt` 是使用傳遞的參數對 `ChaCha20-Poly1305`（網際網路工程任務組變體）的評估，nonce `n` 編碼為 32 個零位元，後跟一個_小端序_ 64 位元值。注意：這遵循 Noise 協定慣例，而不是我們通常的端序。

`decryptWithAD(k, n, ad, ciphertext)`

輸出 `decrypt(k, n, ad, ciphertext)`。

其中 `decrypt` 是使用傳遞的參數對 `ChaCha20-Poly1305`（IETF 變體）的評估，nonce `n` 編碼為 32 個零位元，後跟一個_小端序_ 64 位元值。

`generateKey()`

生成並返回一個新的 `secp256k1` 金鑰對。

其中 `generateKey` 返回的物件有兩個屬性：`.pub`，返回表示公鑰的抽象物件；和 `.priv`，表示用於生成公鑰的私鑰

其中物件還有一個方法：`.serializeCompressed()`

`a || b`

這表示兩個位元組字串 `a` 和 `b` 的連接。

握手會話狀態初始化

在開始握手過程之前，雙方都需要初始化用於推進握手過程的起始狀態。首先，雙方需要構建初始握手摘要 `h`。

1. h = SHA-256(__protocolName__)

   其中 __protocolName__ = "Noise_XK_secp256k1_ChaChaPoly_SHA256" 編碼為 ASCII 字串。

2. `ck = h`

3. h = SHA-256(h || __prologue__)

   其中 __prologue__ 是 ASCII 字串：`lightning`。

除了協定名稱之外，我們還添加了一個額外的「序言」，用於進一步將協定上下文綁定到閃電網路。

為了完成初始化步驟，雙方將回應者的公鑰混合到握手摘要中。因為這個摘要在帶有零長度密文（僅 MAC）的關聯資料發送時使用，這確保了發起者確實知道回應者的公鑰。

- 發起節點混合回應節點的靜態公鑰（以比特幣的壓縮格式序列化）：`h = SHA-256(h || rs.pub.serializeCompressed())`
- 回應節點混合其本地靜態公鑰（以比特幣的壓縮格式序列化）：`h = SHA-256(h || ls.pub.serializeCompressed())`

握手幕

在初始握手初始化之後，我們可以開始握手過程的實際執行。握手由發起者和回應者之間發送的一系列三條訊息組成，以下稱為「幕」。因為每一幕是雙方之間發送的單一訊息，所以握手總共需要 1.5 個往返（每幕 0.5 個）。

第一幕完成增量三重 Diffie-Hellman（DH）金鑰交換的初始部分（使用發起者生成的新臨時金鑰），並確保發起者確實知道回應者的長期公鑰。在第二幕中，回應者將他們希望用於會話的臨時金鑰傳輸給發起者，並再次增量地將這個新金鑰混合到三重 DH 握手中。在第三幕也是最後一幕中，發起者將其長期靜態公鑰傳輸給回應者，並執行最終的 DH 操作將其混合到最終的共享秘密中。

第一幕

```
    -> e, es
```

第一幕從發起者發送到回應者。在第一幕中，發起者嘗試滿足回應者的隱式挑戰。為了完成這個挑戰，發起者必須知道回應者的靜態公鑰。

握手訊息_恰好_ 50 位元組：1 位元組用於握手版本，33 位元組用於發起者的壓縮臨時公鑰，16 位元組用於 `poly1305` 標籤。

發送者動作：

1. `e = generateKey()`

2. `h = SHA-256(h || e.pub.serializeCompressed())`

   新生成的臨時金鑰被累積到正在運行的握手摘要中。

3. `es = ECDH(e.priv, rs)`

   發起者在其新生成的臨時金鑰和遠端節點的靜態公鑰之間執行 ECDH。

4. `ck, temp_k1 = HKDF(ck, es)`

   生成一個新的臨時加密金鑰，用於生成認證 MAC。

5. `c = encryptWithAD(temp_k1, 0, h, zero)`

   其中 `zero` 是零長度明文。

6. `h = SHA-256(h || c)`

   最後，生成的密文被累積到認證握手摘要中。

7. 通過網路緩衝區向回應者發送 `m = 0 || e.pub.serializeCompressed() || c`。

接收者動作：

1. 從網路緩衝區_精確_讀取 50 位元組。
2. 將讀取的訊息（`m`）解析為 `v`、`re` 和 `c`：
   - 其中 `v` 是 `m` 的_第一_位元組，`re` 是 `m` 的接下來 33 位元組，`c` 是 `m` 的最後 16 位元組。
   - 遠端方臨時公鑰（`re`）的原始位元組將使用金鑰的序列化壓縮格式編碼的仿射座標反序列化為曲線上的點。
3. 如果 `v` 是無法識別的握手版本，則回應者必須中止連接嘗試。
4. `h = SHA-256(h || re.serializeCompressed())`

   回應者將發起者的臨時金鑰累積到認證握手摘要中。

5. `es = ECDH(s.priv, re)`

   回應者在其靜態私鑰和發起者的臨時公鑰之間執行 ECDH。

6. `ck, temp_k1 = HKDF(ck, es)`

   生成一個新的臨時加密金鑰，稍後將用於檢查認證 MAC。

7. `p = decryptWithAD(temp_k1, 0, h, c)`

   如果此操作中的 MAC 檢查失敗，則發起者_不_知道回應者的靜態公鑰。如果是這種情況，則回應者必須在沒有任何進一步訊息的情況下終止連接。

8. `h = SHA-256(h || c)`

   接收到的密文被混合到握手摘要中。這一步確保有效載荷沒有被中間人修改。

第二幕

```
<- e, ee
```

第二幕從回應者發送到發起者。只有在第一幕成功的情況下，第二幕才會發生。如果回應者能夠正確解密並檢查第一幕結束時發送的標籤的 MAC，則第一幕成功。

握手_恰好_ 50 位元組：1 位元組用於握手版本，33 位元組用於回應者的壓縮臨時公鑰，16 位元組用於 `poly1305` 標籤。

發送者動作：

1. `e = generateKey()`
2. `h = SHA-256(h || e.pub.serializeCompressed())`

   新生成的臨時金鑰被累積到正在運行的握手摘要中。
3. `ee = ECDH(e.priv, re)`

   其中 `re` 是發起者的臨時金鑰，在第一幕中接收。
4. `ck, temp_k2 = HKDF(ck, ee)`

   生成一個新的臨時加密金鑰，用於生成認證 MAC。
5. `c = encryptWithAD(temp_k2, 0, h, zero)`

   其中 `zero` 是零長度明文。
6. `h = SHA-256(h || c)`

   最後，生成的密文被累積到認證握手摘要中。
7. 通過網路緩衝區向發起者發送 `m = 0 || e.pub.serializeCompressed() || c`。

接收者動作：

1. 從網路緩衝區_精確_讀取 50 位元組。
2. 將讀取的訊息（`m`）解析為 `v`、`re` 和 `c`：

   其中 `v` 是 `m` 的_第一_位元組，`re` 是 `m` 的接下來 33 位元組，`c` 是 `m` 的最後 16 位元組。
3. 如果 `v` 是無法識別的握手版本，則回應者必須中止連接嘗試。
4. `h = SHA-256(h || re.serializeCompressed())`
5. `ee = ECDH(e.priv, re)`

   其中 `re` 是回應者的臨時公鑰。

遠端方臨時公鑰（`re`）的原始位元組將使用金鑰的序列化壓縮格式編碼的仿射座標反序列化為曲線上的點。

6. `ck, temp_k2 = HKDF(ck, ee)`

   生成一個新的臨時加密金鑰，用於生成認證 MAC。

7. `p = decryptWithAD(temp_k2, 0, h, c)`

   如果此操作中的 MAC 檢查失敗，則發起者必須在沒有任何進一步訊息的情況下終止連接。

8. `h = SHA-256(h || c)`

   接收到的密文被混合到握手摘要中。這一步確保有效載荷沒有被中間人修改。

第三幕

```
    -> s, se
```

第三幕是本節描述的經過身份驗證的金鑰協議的最後階段。這一幕作為結束步驟從發起者發送到回應者。第三幕_當且僅當_第二幕成功時執行。在第三幕中，發起者使用_強_前向保密性將其靜態公鑰傳輸給回應者加密，使用握手這一點累積的 `HKDF` 派生秘密金鑰。

握手_恰好_ 66 位元組：1 位元組用於握手版本，33 位元組用於使用 `ChaCha20` 流密碼加密的靜態公鑰，16 位元組用於通過 AEAD 構造生成的加密公鑰標籤，16 位元組用於最終認證標籤。

發送者動作：

1. `c = encryptWithAD(temp_k2, 1, h, s.pub.serializeCompressed())`

   其中 `s` 是發起者的靜態公鑰。

2. `h = SHA-256(h || c)`

3. `se = ECDH(s.priv, re)`

   其中 `re` 是回應者的臨時公鑰。

4. `ck, temp_k3 = HKDF(ck, se)`

   最終的中間共享秘密被混合到正在運行的鏈結金鑰中。

5. `t = encryptWithAD(temp_k3, 0, h, zero)`

   其中 `zero` 是零長度明文。

6. `sk, rk = HKDF(ck, zero)`

   其中 `zero` 是零長度明文，`sk` 是發起者用於加密發送給回應者的訊息的金鑰，`rk` 是發起者用於解密回應者發送的訊息的金鑰。

   生成最終的加密金鑰，用於在會話期間發送和接收訊息。

7. `rn = 0, sn = 0`

   發送和接收 nonce 初始化為 0。

8. 通過網路緩衝區發送 `m = 0 || c || t`。

接收者動作：

1. 從網路緩衝區_精確_讀取 66 位元組。
2. 將讀取的訊息（`m`）解析為 `v`、`c` 和 `t`：

   其中 `v` 是 `m` 的_第一_位元組，`c` 是 `m` 的接下來 49 位元組，`t` 是 `m` 的最後 16 位元組。

3. 如果 `v` 是無法識別的握手版本，則回應者必須中止連接嘗試。
4. `rs = decryptWithAD(temp_k2, 1, h, c)`

   此時，回應者已經恢復了發起者的靜態公鑰。

5. `h = SHA-256(h || c)`
6. `se = ECDH(e.priv, rs)`

   其中 `e` 是回應者的原始臨時金鑰。

7. `ck, temp_k3 = HKDF(ck, se)`
8. `p = decryptWithAD(temp_k3, 0, h, t)`

   如果此操作中的 MAC 檢查失敗，則回應者必須在沒有任何進一步訊息的情況下終止連接。

9. `rk, sk = HKDF(ck, zero)`

   其中 `zero` 是零長度明文，`rk` 是回應者用於解密發起者發送的訊息的金鑰，`sk` 是回應者用於加密發送給發起者的訊息的金鑰。

   生成最終的加密金鑰，用於在會話期間發送和接收訊息。

10. `rn = 0, sn = 0`

    發送和接收 nonce 初始化為 0。

傳輸訊息加密

在第三幕結束時，雙方都派生了加密金鑰，這些金鑰將用於在會話的剩餘時間內加密和解密訊息。

實際的閃電協定訊息封裝在 AEAD 密文中。每條訊息都以另一個 AEAD 密文作為前綴，該密文編碼了後續閃電訊息的總長度（不包括其 MAC）。

任何閃電訊息的_最大_大小不得超過 65,535 位元組。65,535 的最大大小簡化了測試，使記憶體管理更容易，並有助於緩解記憶體耗盡攻擊。

為了使流量分析更加困難，所有加密閃電訊息的長度前綴也被加密。此外，16 位元組的 `Poly-1305` 標籤被添加到加密的長度前綴中，以確保封包長度在傳輸中未被修改，並避免建立解密預言機。

線路上封包的結構類似於 加密封包結構 中的圖示。

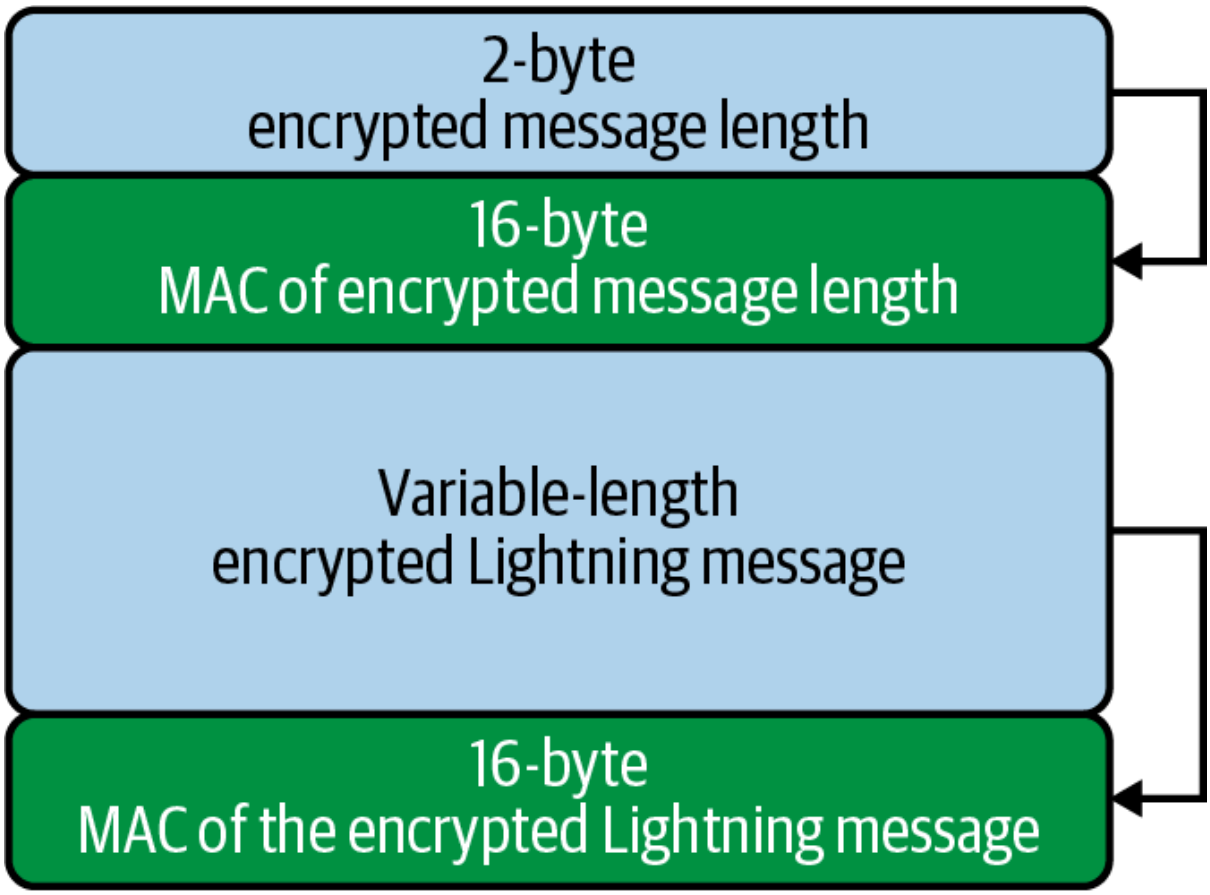

Figure 105. 加密封包結構

前綴訊息長度編碼為 2 位元組大端序整數，總最大封包長度為 2 + 16 + 65,535 + 16 = 65,569 位元組。

加密和發送訊息

要加密閃電訊息（`m`）並發送到網路串流，給定發送金鑰（`sk`）和 nonce（`sn`），完成以下步驟：

1. 設 `l = len(m)`。

   其中 `len` 獲取閃電訊息的位元組長度。

2. 將 `l` 序列化為 2 位元組，編碼為大端序整數。

3. 加密 `l`（使用 `ChaChaPoly-1305`、`sn` 和 `sk`），得到 `lc`（18 位元組）。

   - nonce `sn` 編碼為 96 位元小端序數字。由於解碼的 nonce 是 64 位元，96 位元 nonce 編碼為 32 位元前導零後跟 64 位元值。
   - 此步驟後 nonce `sn` 必須遞增。

- 零長度位元組切片作為 AD（關聯資料）傳遞。

4. 最後，使用與加密長度前綴相同的程序加密訊息本身（`m`）。讓這個加密的密文稱為 `c`。

   此步驟後 nonce `sn` 必須遞增。

5. 通過網路緩衝區發送 `lc || c`。

接收和解密訊息

要解密網路串流中的_下一條_訊息，完成以下步驟：

1. 從網路緩衝區_精確_讀取 18 位元組。
2. 設加密的長度前綴為 `lc`。
3. 解密 `lc`（使用 `ChaCha20-Poly1305`、`rn` 和 `rk`）得到加密封包的大小 `l`。
   - 零長度位元組切片作為 AD（關聯資料）傳遞。
   - 此步驟後 nonce `rn` 必須遞增。
4. 從網路緩衝區_精確_讀取 `l + 16` 位元組，設這些位元組為 `c`。
5. 解密 `c`（使用 `ChaCha20-Poly1305`、`rn` 和 `rk`）得到解密的明文封包 `p`。

   此步驟後 nonce `rn` 必須遞增。

閃電訊息金鑰輪換

定期更換金鑰並忘記先前的金鑰對於防止舊訊息在後來金鑰洩露的情況下被解密（即後向保密性）是有用的。

金鑰輪換對每個金鑰（`sk` 和 `rk`）_單獨_執行。當一方使用金鑰加密或解密 1,000 次後（即每 500 條訊息），該金鑰將被輪換。這可以通過在專用於該金鑰的 nonce 超過 1,000 時輪換金鑰來正確計算。

金鑰 `k` 的金鑰輪換按照以下步驟執行：

1. 設 `ck` 為第三幕結束時獲得的鏈結金鑰。
2. `ck', k' = HKDF(ck, k)`
3. 將金鑰的 nonce 重置為 `n = 0`。
4. `k = k'`
5. `ck = ck'`

14.7. 結論

閃電網路的底層傳輸加密基於 Noise 協定，為閃電節點之間的所有通訊提供了強大的隱私、真實性和完整性安全保證。

與比特幣節點經常「明文」通訊（不加密）不同，所有閃電通訊都是點對點加密的。除了傳輸加密（點對點）之外，在閃電網路中，付款_也_被加密成洋蔥封包（跳對跳），並且付款詳情在發送者和接收者之間帶外發送（端對端）。所有這些安全機制的組合是累積的，並為去匿名化、中間人攻擊和網路監控提供了分層防禦。

當然，沒有安全是完美的，我們將在 閃電網路的 安全性與隱私 中看到這些特性如何被降級和攻擊。然而，閃電網路顯著改善了比特幣的隱私性。

15. 閃電網路付款請求

在本章中，我們將了解_閃電網路付款請求_，或者它們更常見的名稱——*閃電網路發票*。

15.1. 閃電協定套件中的發票

付款請求，又稱_發票_，是支付層的一部分，顯示在 閃電協定套件中的付款請求 的左上方。

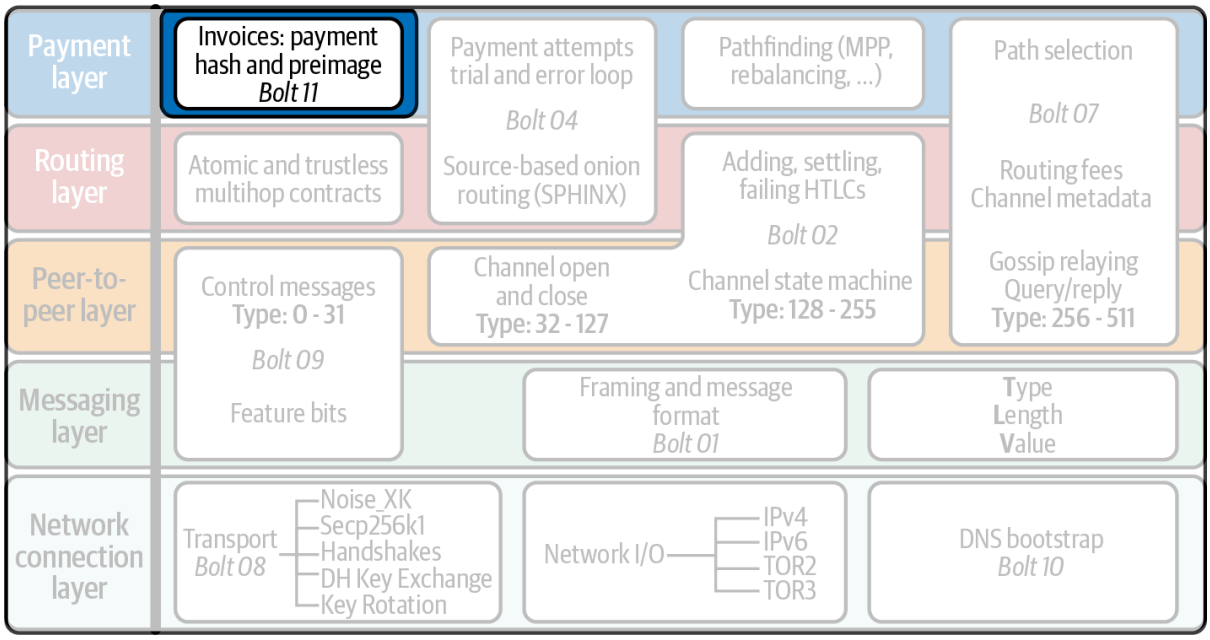

Figure 106. 閃電協定套件中的付款請求

15.2. 簡介

正如我們在整本書中所學到的，完成閃電網路付款至少需要兩個資料：付款雜湊和目的地。由於閃電網路中使用 SHA-256 來實作 HTLC，這些資訊需要 32 位元組來傳輸。另一方面，目的地只是希望接收付款的節點的 `secp256k1` 公鑰。在閃電網路的背景下，付款請求的目的是將這兩個資訊從發送者傳輸到接收者。用於傳輸完成付款所需資訊的 QR 碼友好格式從接收者到發送者的方式在 BOLT #11：閃電網路付款發票協定 (https://github.com/lightningnetwork/lightning-rfc/blob/master/11-payment-encoding.md) 中描述。實際上，付款請求中傳輸的不僅僅是付款雜湊和目的地，還包含更多功能。

15.3. 閃電網路付款請求與比特幣地址的比較

人們第一次接觸閃電網路付款請求時經常問的一個問題是：為什麼不能使用普通的靜態地址格式？

要回答這個問題，你必須首先理解閃電網路作為支付方式與基礎層比特幣的不同之處。與可能用於進行潛在無限次數付款的比特幣地址（儘管重複使用比特幣地址可能會降低隱私性）相比，閃電網路付款請求應該只使用_一次_。這是因為向比特幣地址發送付款本質上使用公鑰密碼系統以一種只有該比特幣地址的真正「擁有者」才能贖回的方式「編碼」付款。

相比之下，要完成閃電網路付款，接收者必須向整個支付路徑（包括發送者）揭示一個「秘密」。這可以被解釋為使用一種特定領域的對稱密碼學，因為付款原像實際上是一個 nonce（僅使用一次的數字）。如果發送者嘗試使用相同的付款雜湊進行另一次付款，那麼他們將面臨資金損失的風險，因為付款可能實際上不會傳遞到目的地。可以安全地假設，在原像被揭示後，路徑中的所有節點都會永久保留它，然後與其轉發 HTLC 以在付款完成時收取路由費用，不如在那個時刻直接結算付款並獲得全部付款金額作為回報。因此，多次使用付款請求是不安全的。

原始閃電網路付款請求存在新的變體，允許發送者根據需要多次重複使用它們。這些變體翻轉了正常的付款流程，因為發送者在加密的洋蔥有效載荷中向接收者傳輸原像，而接收者是唯一能夠解密它並結算付款的人。或者，假設有一種機制允許發送者通常從接收者那裡請求新的付款請求，那麼可以使用互動式協定來允許一定程度的付款請求重複使用。

15.4. BOLT #11：閃電網路付款請求序列化 和解釋

在本節中，我們將描述用於編碼在閃電網路上完成付款所需資訊集的機制。如前所述，付款雜湊和目的地是完成付款所需的最少資訊。然而，在實踐中，還會傳輸更多資訊，如時間鎖資訊、付款請求過期時間，以及可能的鏈上備用地址。完整的規範文件是 BOLT #11：閃電網路付款發票協定 (https://github.com/lightningnetwork/lightning-rfc/blob/master/11-payment-encoding.md)。

15.4.1. 實踐中的付款請求編碼

首先，讓我們看看實際的付款請求是什麼樣子。以下是一個有效的付款請求，在建立時可用於完成主網閃電網路上的付款：

```
lnbc2500u1pvjluezpp5qqqsyqcyq5rqwzqfqqqsyqcyq5rqwzqfqqqsyqcyq5rqwzqfqypqdq5xysx
xatsyp3k7enxv4jsxqzpuaztrnwngzn3kdzw5hydlzf03qdgm2hdq27cqv3agm2awhz5se903vruatf
hq77w3ls4evs3ch9zw97j25emudupq63nyw24cg27h2rspfj9srp
```

15.4.2. 人類可讀前綴

看看這個字串，我們可以挑出一部分可以用眼睛解析的內容，而其餘部分看起來只是一組隨機字串。人類可以在某種程度上解析的部分被稱為_人類可讀前綴_。它允許人類快速從付款請求中一眼提取一些相關資訊。在這種情況下，我們可以看到這是閃電網路主網實例的付款（`lnbc`），並且請求 2,500 uBTC（微比特幣），即 25,000,000 聰。後半部分被稱為資料部分，使用可擴展格式來編碼完成付款所需的資訊。

每個閃電網路實例版本（主網、測試網等）都有自己的人類可讀前綴（參見 BOLT #11 網路前綴）。這允許客戶端軟體和人類快速確定他們的節點是否可以滿足付款請求。

Table 7. BOLT #11 網路前綴

網路	BOLT #11 前綴
主網	`lnbc`
測試網	`lntb`
simnet/regtest	`lnbcrt`

人類可讀前綴的第一部分是付款請求金額的_緊湊_表達式。緊湊金額以兩部分編碼。首先，使用整數作為_基本_金額。然後跟著一個乘數，允許我們指定相對於基本金額的不同數量級增加。如果我們回到最初的例子，那麼我們可以取 `2500u` 部分並將其減少 1,000 倍，改為使用 `2500m` 或（2,500 mBTC）。作為經驗法則，要一眼確定發票的金額，取基本因子並乘以乘數。

目前定義的乘數完整列表見 BOLT #11 金額乘數。

Table 8. BOLT #11 金額乘數

乘數	比特幣單位	乘法因子
m	毫	0.001
u	微	0.000001
n	奈	0.000000001
p	皮	0.000000000001

15.4.3. bech32 和資料段

如果「不可讀」的部分看起來很熟悉，那是因為它使用了與今天 SegWit 相容的比特幣地址相同的編碼方案，即 bech32。描述 bech32 編碼方案超出了本章的範圍。簡而言之，這是一種編碼短字串的複雜方法，具有非常好的錯誤糾正和檢測特性。

資料部分可以分為三個部分：

- 時間戳記
- 零個或多個標記的鍵值對
- 整個發票的簽名

時間戳記以自 1970 年即 Unix 紀元以來的秒數表示。這個時間戳記允許發送者判斷發票有多舊，正如我們稍後將看到的，如果接收者願意，可以強制發票只在一段時間內有效。

類似於我們在 類型-長度-值格式 中學到的 TLV 格式，BOLT #11 發票格式使用一系列可擴展的鍵值對來編碼完成付款所需的資訊。因為使用了鍵值對，所以如果將來引入新的付款類型或額外的要求/功能，很容易添加新值。

最後，包含一個簽名，覆蓋由付款目的地簽署的整個發票。這個簽名允許發送者驗證付款請求確實是由付款目的地建立的。與未簽名的比特幣付款請求不同，這使我們能夠確保特定實體簽署了付款請求。簽名本身使用恢復 ID 編碼，這允許使用更緊湊的簽名來進行公鑰提取。驗證簽名時，恢復 ID 提取公鑰，然後對照發票中包含的公鑰進行驗證。

標記的發票欄位

標記的發票欄位編碼在發票的主體中。這些欄位表示不同的鍵值對，表達可能有助於完成付款的額外資訊或完成付款_所需_的資訊。因為使用了 bech32 的輕微變體，這些欄位中的每一個實際上都在「基數 5」域中。

給定的標記欄位由三個組成部分組成：

- 欄位的 `type`（5 位元）
- 欄位資料的 `length`（10 位元）
- `data` 本身，大小為 `length * 5` 位元組

所有目前定義的標記欄位的完整列表見 BOLT #11 標記的發票欄位。

Table 9. BOLT #11 標記的發票欄位

欄位標記	資料長度	用途
p	52	SHA-256 付款雜湊。
s	52	256 位元秘密，通過減輕中間節點的探測來增加付款的端對端隱私。
d	可變	描述，付款目的的簡短 UTF-8 字串。
n	53	目的地節點的公鑰。
h	52	表示付款本身描述的雜湊。這可用於承諾長度超過 639 位元組的描述。
x	可變	付款的過期時間，以秒為單位。如果未指定，預設為 1 小時（3,600）。
c	可變	用於路由中最後一跳的 `min_cltv_expiry`。如果未指定，預設為 9。
f	可變	備用鏈上地址，如果無法通過閃電網路完成付款，則用於完成付款。
r	可變	一個或多個條目，允許接收者為發送者提供額外的臨時邊以完成付款。
9	可變	一組 5 位元值，包含完成付款所需的功能位元。

欄位 `r` 中包含的元素通常被稱為_路由提示_。它們允許接收者傳達一組額外的邊，可能有助於發送者完成付款。當接收者有一些/全部私有通道，並希望引導發送者進入通道圖的這個「未映射」部分時，通常會使用提示。路由提示有效地編碼與正常 `channel_update` 訊息相同的資訊。更新本身被打包成一個具有以下欄位的單一值：

- 邊中出站節點的 `pubkey`（264 位元）

- 「虛擬」邊的 `short_channel_id`（64 位元）
- 邊的基本費用（`fee_base_msat`）（32 位元）
- 比例費用（`fee_proportional_millionths`）（32 位元）
- CLTV 過期增量（`cltv_expiry_delta`）（16 位元）

資料段的最後部分是一組功能位元，向發送者傳達完成付款所需的功能。例如，如果將來添加了與原始付款類型不向後相容的新付款類型，那麼接收者可以設定一個_必需_功能位元來傳達付款者需要理解該功能才能完成付款。

15.5. 結論

正如我們所看到的，發票遠不止是金額請求。它們包含有關_如何_進行付款的關鍵資訊，例如路由提示、目的地節點的公鑰、增加安全性的臨時金鑰等等。

16. 閃電網路的 安全性與隱私

在本章中，我們將探討與閃電網路安全性和隱私相關的一些最重要議題。首先，我們將考慮隱私，它意味著什麼，如何評估它，以及在使用閃電網路時可以採取哪些措施來保護自己的隱私。然後我們將探討一些常見的攻擊和緩解技術。

16.1. 為什麼隱私很重要？

加密貨幣的關鍵價值主張是抗審查的貨幣。比特幣為參與者提供了儲存和轉移財富的可能性，而不受政府、銀行或企業的干預。閃電網路繼續這一使命。

與託管式比特幣銀行等簡單的擴展解決方案不同，閃電網路旨在擴展比特幣而不損害自我託管，這應該會導致比特幣生態系統中更大的抗審查性。然而，閃電網路在不同的安全模型下運行，這引入了新的安全和隱私挑戰。

16.2. 隱私的定義

「閃電網路是私密的嗎？」這個問題沒有直接的答案。隱私是一個複雜的話題；通常很難精確定義我們所說的隱私是什麼意思，特別是如果你不是隱私研究人員的話。幸運的是，隱私研究人員使用流程來分析和評估系統的隱私特性，我們也可以使用它們！讓我們看看安全研究人員如何通過兩個一般步驟來回答「閃電網路是私密的嗎？」這個問題。

首先，隱私研究人員會定義一個_安全模型_，指定對手能夠做什麼以及其目標是什麼。 然後，他們會描述系統的相關屬性並檢查它是否符合要求。

16.3. 評估隱私的過程

安全模型基於一組基礎的_安全假設_。 在加密系統中，這些假設通常圍繞加密原語的數學屬性，例如密碼、簽名和雜湊函數。 閃電網路的安全假設是協定中使用的 ECDSA 簽名、SHA-256 雜湊函數和其他加密函數在其安全定義範圍內運行。 例如，我們假設實際上不可能找到雜湊函數的原像（和第二原像）。 這允許閃電網路依賴 HTLC 機制（使用雜湊函數的原像）來實現多跳付款的原子性：除了最終接收者之外，沒有人可以揭示付款秘密並解決 HTLC。 我們還假設網路中有一定程度的連通性，即閃電通道形成一個連通圖。因此，可以找到從任何發送者到任何接收者的路徑。最後，我們假設網路訊息在特定超時時間內傳播。

現在我們已經確定了一些基礎假設，讓我們考慮一些可能的對手。

以下是閃電網路中一些可能的對手模型。 「誠實但好奇」的轉發節點可以觀察付款金額、緊鄰的前一個和後一個節點，以及已公告通道及其容量的圖。 連接非常好的節點可以做同樣的事情，但程度更大。 例如，考慮一個流行錢包的開發者維護一個節點，他們的使用者預設連接到該節點。 這個節點將負責路由來自和去往該錢包使用者的大部分付款。 如果多個節點在對手控制下呢？ 如果兩個串通的節點恰好在同一支付路徑上，它們會理解他們正在轉發屬於同一付款的 HTLC，因為 HTLC 具有相同的付款雜湊。

多部分付款（參見 多部分付款）使使用者能夠混淆他們的付款金額，因為其非均勻的分割大小。

閃電網路攻擊者的目標可能是什麼？ 資訊安全通常用三個主要屬性來描述：機密性、完整性和可用性。

機密性

資訊只傳達給預期的接收者。

完整性

資訊在傳輸過程中不會被篡改。

可用性

系統大部分時間都在運行。

閃電網路的重要屬性主要圍繞機密性和可用性。一些最重要的保護屬性包括：

- 只有發送者和接收者知道付款金額。
- 沒有人可以將發送者和接收者聯繫起來。
- 誠實的使用者不能被阻止發送和接收付款。

對於每個隱私目標和安全模型，攻擊者成功的概率各不相同。 這個概率取決於各種因素，例如網路的大小和結構。 在其他條件相同的情況下，攻擊小型網路通常比攻擊大型網路更容易成功。 同樣，網路越集中，如果「中心」節點在攻擊者控制下，攻擊者的能力就越強。 當然，必須精確定義集中化一詞才能圍繞它建立安全模型，並且有許多可能的定義來描述網路的集中程度。 最後，作為支付網路，閃電網路依賴於經濟激勵。 費用的大小和結構影響路由演算法，因此可以通過將大多數付款轉發通過攻擊者的節點來幫助攻擊者，或者阻止這種情況發生。

16.4. 匿名集

去匿名化某人意味著什麼？ 簡單來說，去匿名化意味著將某些行為與一個人的真實世界身份聯繫起來，例如他們的姓名或實際地址。 在隱私研究中，去匿名化的概念更加細緻。 首先，我們不一定談論的是姓名和地址。 發現某人的 IP 地址或電話號碼也可能被視為去匿名化。 允許將使用者的行為與其先前行為聯繫起來的資訊被稱為_身份_。 其次，去匿名化不是二元的；使用者既不是完全匿名的，也不是完全去匿名的。 相反，隱私研究會將匿名性與匿名集進行比較。

_匿名集_是隱私研究中的核心概念。 它指的是一組身份，從攻擊者的角度來看，給定的行為可能對應於該集合中的任何人。 考慮一個現實生活中的例子。 想像你在城市街道上遇到一個人。 從你的角度來看，他們的匿名集是什麼？ 如果你不認識他們，並且沒有任何額外資訊，他們的匿名集大致等於城市人口，包括旅行者。 如果你額外考慮他們的外表，你可能能夠大致

估計他們的年齡，並從匿名集中排除明顯比該人年長或年輕的城市居民。 此外，如果你注意到該人使用電子徽章走進 X 公司的辦公室，匿名集就縮小到 X 公司的員工和訪客數量。 最後，你可能會注意到他們用來到達該地點的汽車的車牌號碼。 如果你是一個普通觀察者，這對你沒有太大幫助。 然而，如果你是城市官員並且可以訪問將車牌號碼與姓名匹配的資料庫，你可以將匿名集縮小到只有幾個人：車主和任何可能借用汽車的親密朋友和親戚。

這個例子說明了幾個重要的觀點。 首先，每一條資訊都可能使對手更接近其目標。 可能不需要將匿名集縮小到一個人的大小。 例如，如果對手計劃進行有針對性的阻斷服務（DoS）攻擊並可以攻陷 100 台伺服器，那麼 100 人的匿名集就足夠了。 其次，對手可以交叉關聯來自不同來源的資訊。 即使隱私洩露看起來相對良性，我們永遠不知道它與其他資料來源結合可以達到什麼效果。 最後，特別是在加密設定中，攻擊者總是有暴力搜索的「最後手段」。 加密原語的設計使得實際上不可能猜測諸如私鑰之類的秘密。 然而，每一條資訊都使對手更接近這個目標，在某個時候，它變得可以實現。

就閃電網路而言，去匿名化通常意味著得出付款與由節點 ID 識別的使用者之間的對應關係。 每筆付款可能被分配一個發送者匿名集和一個接收者匿名集。 理想情況下，匿名集由網路的所有使用者組成。 這確保攻擊者沒有任何資訊。 然而，真實網路會洩露資訊，使攻擊者能夠縮小搜索範圍。 匿名集越小，成功去匿名化的機會就越高。

16.5. 閃電網路與比特幣在隱私方面的差異

雖然比特幣網路上的交易不會將真實世界身份與比特幣地址相關聯是事實，但所有交易都以明文廣播並可以被分析。 多家公司已經成立，專門對比特幣和其他加密貨幣的使用者進行去匿名化。

乍看之下，閃電網路比比特幣提供了更好的隱私，因為閃電網路付款不會廣播到整個網路。 雖然這改善了隱私基線，但閃電協定的其他屬性可能使匿名付款更具挑戰性。 例如，較大的付款可能有較少的路由選項。 這可能允許控制資本充足節點的對手路由大多數大額付款並發現付款金額以及可能的其他細節。隨著時間的推移，隨著閃電網路的增長，這可能會變得不那麼嚴重。

閃電網路和比特幣之間的另一個相關區別是閃電節點維護永久身份，而比特幣節點則不是。 老練的比特幣使用者可以輕鬆切換用於接收區塊鏈資料和廣播交易的節點。 相反，閃電網路使用者通過他們用來開設支付通道的節點發送和接收付款。 此外，閃電協定假設路由節點除了節點 ID 之外還公告其 IP 地址。 這在節點 ID 和 IP 地址之間建立了永久連結，考慮到 IP 地址通常是與使用者實際位置相關的匿名攻擊中的中間步驟，並且在大多數情況下與真實世界身份相關，這可能是危險的。 可以通過 Tor 使用閃電網路，但許多節點不使用此功能，這可以從 從節點公告收集的統計資料 (https://1ml.com/statistics)中看出。

閃電網路使用者在發送付款時，其匿名集中有其鄰居。 具體來說，路由節點只知道緊鄰的前一個和後一個節點。 路由節點不知道其在支付路徑中的直接鄰居是否是最終發送者或接收者。 因此，閃電網路中節點的匿名集大致等於其鄰居（參見 Alice 和 Bob 的匿名集由他們的鄰居組成）。

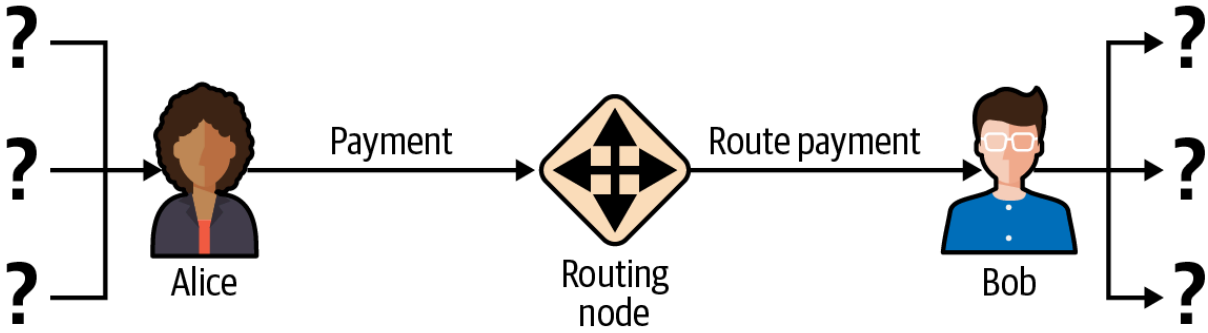

Figure 107. Alice 和 Bob 的匿名集由他們的鄰居組成

類似的邏輯適用於付款接收者。 許多使用者只開設少數幾個支付通道，因此限制了他們的匿名集。 此外，在閃電網路中，匿名集是靜態的或至少是緩慢變化的。 相比之下，可以在鏈上 CoinJoin 交易中實現顯著更大的匿名集。 匿名集大於 50 的 CoinJoin 交易相當頻繁。 通常，CoinJoin 交易中的匿名集對應於一組動態變化的使用者。

最後，閃電網路使用者也可能被拒絕服務，其通道被攻擊者阻塞或耗盡。 轉發付款需要資本——一種稀缺資源！——暫時鎖定在路徑上的 HTLC 中。 攻擊者可能發送許多付款但未能完成它們，長時間佔用誠實使用者的資本。 這種攻擊向量在比特幣中不存在（或至少不那麼明顯）。

總之，雖然閃電網路架構的某些方面表明它在隱私方面比比特幣更進一步，但協定的其他屬性可能使隱私攻擊更容易。需要進行徹底的研究來評估閃電網路提供什麼樣的隱私保證並改善現狀。

本章這部分討論的問題總結了 2021 年中期可用的研究。然而，這個研究和開發領域正在快速增長。我們很高興地報告，作者知道目前有多個研究團隊正在研究閃電網路隱私。

現在讓我們回顧一些在學術文獻中描述的對閃電網路隱私的攻擊。

16.6. 對閃電網路的攻擊

最近的研究描述了閃電網路安全性和隱私可能受到損害的各種方式。

16.6.1. 觀察付款金額

隱私保護支付系統的目標之一是對無關各方隱藏付款金額。 閃電網路在這方面是對第一層的改進。 雖然比特幣交易以明文廣播並且任何人都可以觀察到，但閃電網路付款只通過支付路徑上的幾個節點傳輸。 然而，中間節點確實看到付款金額，儘管這個付款金額可能不對應於實際的總付款金額（參見 多部分付款）。 這是在每一跳建立新的 HTLC 所必需的。 中間節點可以獲得付款金額不會構成直接威脅。 然而，_誠實但好奇_的中間節點可能將其用作更大攻擊的一部分。

16.6.2. 連結發送者和接收者

攻擊者可能有興趣了解付款的發送者和/或接收者以揭示某些經濟關係。 這種隱私洩露可能損害抗審查性，因為中間節點可能審查來自或去往某些接收者或發送者的付款。 理想情況下，除了發送者和接收者之外，任何人都不應該能夠將發送者與接收者聯繫起來。

在以下章節中，我們將考慮兩種類型的對手：路徑外對手和路徑上對手。 路徑外對手試圖在不參與付款路由過程的情況下評估付款的發送者和接收者。 路徑上對手可以利用他們通過路由相關付款可能獲得的任何資訊。

首先，考慮_路徑外對手_。 在這種攻擊場景的第一步中，強大的路徑外對手通過探測（在後續章節中描述）推斷每個支付通道中的個別餘額，並在時間 t_1 形成網路快照。為簡單起見，讓我們設 t_1 等於 12:05。 然後它在稍後的某個時間 t_2 再次探測網路，我們將其設為 12:10。然後攻擊者會比較 12:10 和 12:05 的快照，並使用兩個快照之間的差異通過查看已更改的路徑來推斷有關付款的資訊。 在最簡單的情況下，如果在 12:10 和 12:05 之間只發生了一筆付款，對手會觀察到餘額變化相同金額的單一路徑。 因此，對手幾乎了解了這筆付款的所有資訊：發送者、接收者和金額。 如果多條支付路徑重疊，對手需要應用啟發式方法來識別這種重疊並分離付款。

現在，我們將注意力轉向_路徑上對手_。 這樣的對手可能看起來很複雜。 然而，在 2020 年 6 月，研究人員注意到最中心的單一節點 觀察到接近 50% 的所有閃電網路付款 (https://arxiv.org/pdf/2006.12143.pdf)，而四個最中心的節點 平均觀察到 72% 的付款 (https://arxiv.org/pdf/1909.06890.pdf)。 這些發現強調了路徑上攻擊者模型的相關性。 儘管支付路徑中的中介只知道其後繼者和前驅者，但惡意或誠實但好奇的中介可能會利用幾種洩漏來推斷發送者和接收者。

路徑上對手可以觀察任何路由付款的金額以及時間鎖增量（參見 洋蔥路由）。 因此，對手可以從發送者或接收者的匿名集中排除任何容量低於路由金額的節點。 因此，我們觀察到隱私和付款金額之間的權衡。 通常，付款金額越大，匿名集越小。 我們注意到，這種洩漏可以通過多部分付款或大容量支付通道來最小化。 同樣，具有小時間鎖增量的支付通道可以從支付路徑中排除。 更準確地說，如果付款可能被鎖定的剩餘時間大於轉發節點願意接受的時間，則支付通道不能屬於付款。 這種洩漏可以通過遵循所謂的影子路由來消除。

路徑上對手可以利用的最微妙但最強大的洩漏之一是時序分析。 路徑上對手可以為每筆路由的付款保留日誌，以及節點回應 HTLC 請求所需的時間。 在開始攻擊之前，攻擊者通過向閃電網路中的每個節點發送請求來了解其延遲特性。 自然地，這可以幫助確定對手在支付路徑中的精確位置。 更重要的是，正如最近所展示的，攻擊者可以使用基於時間的估計器從一組可能的發送者和接收者中成功確定付款的發送者和接收者。

最後，重要的是要認識到可能存在未知或未研究的洩漏，這些洩漏可能有助於去匿名化嘗試。例如，因為不同的閃電錢包應用不同的路由演算法，即使知道應用的路由演算法也可以幫助排除某些節點成為付款的發送者和/或接收者。

16.6.3. 揭示通道餘額（探測）

閃電通道的餘額應該出於隱私和效率原因而被隱藏。 閃電節點只知道其相鄰通道的餘額。 協定沒有提供查詢遠端通道餘額的標準方式。

然而，攻擊者可以在_探測攻擊_中揭示遠端通道的餘額。 在資訊安全中，探測是指向目標系統發送請求並根據收到的回應對其私有狀態得出結論的技術。

閃電通道容易受到探測。 回想一下，標準的閃電付款從接收者建立隨機付款秘密並將其雜湊發送給發送者開始。 請注意，對於中間節點，所有雜湊看起來都是隨機的。 沒有辦法判斷雜湊是否對應於真正的秘密還是隨機生成的。

探測攻擊按以下方式進行。 假設攻擊者 Mallory 想要揭示 Alice 在 Alice 和 Bob 之間公開通道中的餘額。 假設該通道的總容量為 100 萬聰。 Alice 的餘額可能是從零到 100 萬聰之間的任何數值（準確地說，由於通道儲備金，估計會更緊一些，但為了簡單起見，我們在這裡不考慮它）。 Mallory 與 Alice 開設一個 100 萬聰的通道，並使用_隨機數_作為付款雜湊通過 Alice 向 Bob 發送 50 萬聰。 當然，這個數字不對應於任何已知的付款秘密。 因此，付款將失敗。 問題是：它究竟會如何失敗？

有兩種情況。 如果 Alice 在她與 Bob 的通道上有超過 50 萬聰，她會轉發付款。 Bob 解密付款洋蔥並意識到付款是給他的。 他查找他本地的付款秘密儲存並搜索與付款雜湊對應的原像，但沒有找到。 按照協定，Bob 向 Alice 返回「未知付款雜湊」錯誤，Alice 將其轉發給 Mallory。 因此，Mallory 知道如果付款雜湊是真實的，付款_本可以成功_。 因此，Mallory 可以將她對 Alice 餘額的估計從「零到 100 萬之間」更新為「50 萬到 100 萬之間」。 另一種情況發生在 Alice 的餘額低於 50 萬聰時。 在這種情況下，Alice 無法轉發付款並向 Mallory 返回「餘額不足」錯誤。 Mallory 將她的估計從「零到 100 萬之間」更新為「零到 50 萬之間」。

請注意，在任何情況下，Mallory 的估計在僅僅一次探測後就變得精確了兩倍！ 她可以繼續探測，選擇下一個探測金額使其將當前估計區間分成兩半。 這種眾所周知的搜索技術稱為_二分搜索_。 使用二分搜索，探測次數與所需精度成_對數_關係。 例如，要在 100 萬聰的通道中精確到單個聰獲得 Alice 的餘額，Mallory 只需執行 $\log_2(1{,}000{,}000) \approx 20$ 次探測。 如果一次探測需要 3 秒，一個通道可以在大約一分鐘內被精確探測！

通道探測可以變得更加高效。 在最簡單的變體中，Mallory 直接連接到她想要探測的通道。 是否可以在不與其端點之一開設通道的情況下探測通道？ 想像一下，Mallory 現在想要探測 Bob 和 Charlie 之間的通道，但不想開設另一個通道，這需要支付鏈上費用並等待資金交易的確認。 相反，Mallory 重複使用她現有的到 Alice 的通道，並沿著路線 Mallory → Alice → Bob → Charlie 發送探測。 Mallory 可以像以前一樣解釋「未知付款雜湊」錯誤：探測已到達目的地；因此，路線上的所有通道都有足夠的餘額來轉發它。 但是，如果 Mallory 收到「餘額不足」錯誤怎麼辦？ 這是否意味著 Alice 和 Bob 之間或 Bob 和 Charlie 之間的餘額不足？

在當前的閃電協定中，錯誤訊息不僅報告發生了_哪個_錯誤，還報告它發生在_哪裡_。 因此，通過更仔細的錯誤處理，Mallory 現在知道哪個通道失敗了。 如果這是目標通道，她更新她的估計；如果不是，她選擇另一條到目標通道的路線。 除了目標通道之外，她甚至獲得了有

關中間通道餘額的_額外_資訊。

探測攻擊可以進一步用於連結發送者和接收者，如前一節所述。

此時，你可能會問：為什麼閃電網路在保護其使用者的私有資料方面做得如此糟糕？ 不向發送者揭示付款為什麼以及在哪裡失敗不是更好嗎？ 確實，這可能是一種潛在的對策，但它有顯著的缺點。 閃電網路必須在隱私和效率之間取得仔細的平衡。 請記住，常規節點不知道遠端通道中的餘額分佈。 因此，付款可能（而且經常）因為中間跳的餘額不足而失敗。 錯誤訊息允許發送者在構建另一條路線時排除失敗的通道。 一個流行的閃電錢包甚至在內部執行探測以檢查構建的路線是否真的可以處理付款。

還有其他潛在的對策來防止通道探測。 首先，攻擊者很難針對未公告的通道。 其次，實作即時（JIT）路由的節點可能不太容易受到攻擊。 最後，由於多部分付款使容量不足的問題不那麼嚴重，協定開發者可能會考慮隱藏一些錯誤細節而不損害效率。

16.6.4. 阻斷服務

當資源公開可用時，攻擊者可能會嘗試通過執行阻斷服務（DoS）攻擊使該資源不可用。 通常，這是通過攻擊者用請求轟炸資源來實現的，這些請求與合法查詢無法區分。 這些攻擊很少導致目標遭受經濟損失，除了其服務中斷的機會成本，僅僅是為了使目標感到困擾。

典型的 DoS 攻擊緩解措施需要對請求進行身份驗證，以將合法使用者與惡意使用者分開。這些緩解措施對常規使用者產生的成本微不足道，但對於大規模發起請求的攻擊者來說將充當足夠的威懾。 反阻斷服務措施在網際網路上隨處可見——網站應用速率限制以確保沒有單一使用者可以消耗其伺服器的所有注意力，電影評論網站需要登入身份驗證以阻止憤怒的 r/prequelmemes（Reddit 群組）成員，資料服務銷售 API 金鑰以限制查詢數量。

比特幣中的 DoS

在比特幣中，節點用於中繼交易的頻寬和它們以記憶體池形式提供給網路的空間是公開可用的資源。 網路上的任何節點都可以通過發送有效交易來消耗頻寬和記憶體池空間。 如果這筆交易在有效區塊中被挖掘，他們將支付交易費用，這增加了使用這些共享網路資源的成本。

過去，比特幣網路曾面臨 DoS 攻擊嘗試，攻擊者用低費用交易向網路發送垃圾郵件。 由於交易費用低，許多這些交易沒有被礦工選中，因此攻擊者可以在不支付費用的情況下消耗網路資源。 為了解決這個問題，設定了最低交易中繼費用，該費用設定了節點傳播交易所需的閾值費用。 這一措施在很大程度上確保了消耗網路資源的交易最終將支付其鏈上費用。 最低中繼費用對常規使用者來說是可以接受的，但如果攻擊者試圖向網路發送垃圾郵件，將在經濟上受到傷害。 雖然一些交易可能無法在高費用環境中進入有效區塊，但這些措施在很大程度上有效地阻止了這類垃圾郵件。

閃電網路中的 DoS

與比特幣類似，閃電網路對其公共資源的使用收取費用，但在這種情況下，資源是公共通道，費用以路由費用的形式出現。 通過節點路由付款以換取費用的能力為網路提供了巨大的可擴展性優勢——不直接連接的節點仍然可以進行交易——但這是以暴露必須保護免受 DoS 攻擊的公

共資源為代價的。

當閃電節點代表你轉發付款時，它使用資料和付款頻寬來更新其承諾交易，付款金額被保留在其通道餘額中，直到它被結算或失敗。 在成功的付款中，這是可以接受的，因為節點最終會獲得其費用。 失敗的付款在當前協定中不收取費用。 這允許節點無成本地通過任何通道路由失敗的付款。 這對合法使用者來說很好，他們不想為失敗的嘗試付費，但也允許攻擊者無成本地消耗節點的資源——就像比特幣上那些永遠不會支付礦工費用的低費用交易一樣。

在撰寫本文時，lightning-dev 郵件列表上正在 討論 (https://lists.linuxfoundation.org/pipermail/lightning-dev/2020-June/002734.html)如何最好地解決這個問題。

已知的 DoS 攻擊

公共閃電網路通道上有兩種已知的 DoS 攻擊，它們使目標通道或一組目標通道無法使用。 這兩種攻擊都涉及通過公共通道路由付款，然後保持它們直到超時，從而最大化攻擊的持續時間。 不支付費用而使付款失敗的要求相當容易滿足，因為惡意節點可以簡單地將付款重新路由給自己。 在沒有失敗付款費用的情況下，攻擊者唯一的成本是開設通道以發送這些付款的鏈上成本，在低費用環境中這可以是微不足道的。

16.6.5. 承諾阻塞

閃電節點使用非對稱承諾交易更新其共享狀態，在這些交易上添加和移除 HTLC 以促進付款。 每一方在承諾交易中一次最多限制為 483 (https://github.com/lightningnetwork/lightning-rfc/blob/c053ce7afb4cbf88615877a0d5fc7b8dbe2b9ba0/02-peer-protocol.md#the-open_channel-message)
個 HTLC。 通道阻塞攻擊允許攻擊者通過目標通道路由 483 筆付款並保持它們直到超時來使通道無法使用。

應該注意的是，規範中選擇這個限制是為了確保所有 HTLC 都可以在 單一懲罰交易 (https://github.com/lightningnetwork/lightning-rfc/blob/master/05-onchain.md#penalty-transaction-weight-calculation)
中被清掃。 雖然這個限制_可能_會增加，但交易仍然受到區塊大小的限制，因此可用插槽的數量可能仍然有限。

16.6.6. 通道流動性鎖定

通道流動性鎖定攻擊與通道阻塞攻擊類似，因為它通過通道路由付款並保持它們以使通道無法使用。 這種攻擊不是鎖定通道承諾上的插槽，而是通過目標通道路由大額 HTLC，消耗通道的所有可用頻寬。 這種攻擊的資本承諾高於承諾阻塞攻擊，因為攻擊節點需要更多資金來通過目標路由失敗的付款。

16.7. 跨層去匿名化

電腦網路通常是分層的。 分層允許關注點分離並使整個系統易於管理。 如果設計網站需要理解所有 TCP/IP 堆疊直到光纖電纜中位元的物理編碼，那就沒人能設計網站了。 每一層都應該以乾淨的方式為上一層提供功能。 理想情況下，上層應該將下層視為黑盒。 然而，在現實中，實作並不完美，細節會_洩漏_到上層。 這就是洩漏抽象的問題。

在閃電網路的背景下，閃電協定依賴於比特幣協定和閃電 P2P 網路。 到目前為止，我們只考慮了閃電網路單獨提供的隱私保證。 然而，建立和關閉支付通道本質上是在比特幣區塊鏈上執行的。 因此，為了完整分析閃電網路的隱私規定，需要考慮使用者可能與之互動的技術堆疊的每一層。 具體來說，試圖去匿名化閃電網路使用者的對手可以並且將使用鏈下和鏈上資料來聚類或連結閃電節點與相應的比特幣地址。

在跨層背景下，試圖去匿名化閃電網路使用者的攻擊者可能有各種目標：

- 聚類同一使用者擁有的比特幣地址（第一層）。我們稱這些為比特幣實體。
- 聚類同一使用者擁有的閃電節點（第二層）。
- 明確地將閃電節點集合與控制它們的比特幣實體集合聯繫起來。

有幾種啟發式方法和使用模式允許對手聚類同一閃電網路使用者擁有的比特幣地址和閃電節點。 此外，這些聚類可以使用其他強大的跨層連結啟發式方法跨層連結。 最後一種類型的啟發式方法，跨層連結技術，強調了需要對隱私有整體視角。具體來說，我們必須在兩層一起的背景下考慮隱私。

16.7.1. 鏈上比特幣實體聚類

閃電網路區塊鏈互動永久反映在比特幣實體圖中。 即使通道關閉，攻擊者也可以觀察哪個地址為通道提供了資金，以及關閉後代幣花費到哪裡。 對於這個分析，讓我們考慮四個獨立的實體。 開設通道導致從_來源實體_到_資金實體_的貨幣流動；關閉通道導致從_結算實體_到_目標實體_的流動。

在 2021 年初，https://arxiv.org/pdf/2007.00764.pdf[Romiti 等人]確定了四種允許聚類這些實體的啟發式方法。 其中兩種捕獲某些洩漏的資金行為，兩種描述洩漏的結算行為。

星形啟發式（資金）

如果一個組件包含一個將資金轉發給一個或多個資金實體的來源實體，這些資金實體很可能由同一使用者控制。

蛇形啟發式（資金）

如果一個組件包含一個將資金轉發給一個或多個實體的來源實體，這些實體本身被用作來源和資金實體，那麼所有這些實體很可能由同一使用者控制。

收集器啟發式（結算）

如果一個組件包含一個從一個或多個結算實體接收資金的目標實體，這些結算實體很可能由同一使用者控制。

代理啟發式（結算）

如果一個組件包含一個從一個或多個實體接收資金的目標實體，這些實體本身被用作結算和目標實體，那麼這些實體很可能由同一使用者控制。

值得指出的是，這些啟發式方法可能產生誤報。 例如，如果多個不相關使用者的交易在 CoinJoin 交易中合併，那麼星形或代理啟發式可能產生誤報。 如果使用者從 CoinJoin 交易為支付通道提供資金，這可能發生。 另一個潛在的誤報來源可能是，如果聚類的地址由服務（例如交易所）控制或代表其使用者（託管錢包）控制，則實體可能代表多個使用者。 然而，這些誤報可以有效地被過濾掉。

對策

如果資金交易的輸出不被重複使用來開設其他通道，蛇形啟發式就不起作用。 如果使用者避免從單一外部來源為通道提供資金並避免在單一外部目標實體中收集資金，其他啟發式方法將不會產生任何顯著結果。

16.7.2. 鏈下閃電節點聚類

閃電節點公告別名，例如 *LNBig.com*。 別名可以提高系統的可用性。 然而，使用者傾向於為自己的不同節點使用相似的別名。 例如，*LNBig.com Billing* 很可能與別名為 *LNBig.com* 的節點由同一使用者擁有。 基於這一觀察，可以通過應用節點別名來聚類閃電節點。 具體來說，如果閃電節點的別名相對於某些字串相似度指標相似，則將它們聚類到單一地址。

另一種聚類閃電節點的方法是應用其 IP 或 Tor 地址。 如果相同的 IP 或 Tor 地址對應於不同的閃電節點，這些節點很可能由同一使用者控制。

對策

為了更多隱私，別名應該彼此充分不同。 雖然對於希望在閃電網路中擁有入站通道的節點來說，公開公告 IP 地址可能是不可避免的，但如果每個節點的客戶端託管在不同的服務提供商上，因此使用不同的 IP 地址，則可以減輕同一使用者跨節點的可連結性。

16.7.3. 跨層連結：閃電節點和比特幣實體

將閃電節點與比特幣實體關聯是一種嚴重的隱私洩露，由於大多數閃電節點公開暴露其 IP 地址而加劇。 通常，IP 地址可以被視為使用者的唯一識別碼。 兩種廣泛觀察到的行為模式揭示了閃電節點和比特幣實體之間的連結：

代幣重複使用

每當使用者關閉支付通道時，他們會取回相應的代幣。然而，許多使用者在開設新通道時重複使用這些代幣。 這些代幣可以有效地連結到一個共同的閃電節點。

實體重複使用

通常，使用者從屬於同一比特幣實體的比特幣地址為其支付通道提供資金。

如果使用者擁有多個未聚類的地址或使用多個錢包與閃電網路互動，這些跨層連結演算法可能會被挫敗。

比特幣實體可能被去匿名化的事實說明了同時考慮兩層隱私而不是一次一層是多麼重要。

16.8. 閃電圖

閃電網路，顧名思義，是一個支付通道的點對點網路。 因此，它的許多屬性（隱私、健壯性、連通性、路由效率）受其網路性質的影響和特徵化。

在本節中，我們從網路科學的角度討論和分析閃電網路。 我們特別感興趣的是理解閃電通道圖、其健壯性、連通性和其他重要特性。

16.8.1. 閃電圖在現實中是什麼樣子？

人們可能預期閃電網路是一個隨機圖，其中邊在節點之間隨機形成。 如果是這種情況，那麼閃電網路的度分佈將遵循高斯正態分佈。 特別是，大多數節點將具有大致相同的度，我們不會期望具有異常大度的節點。 這是因為正態分佈對於分佈平均值周圍區間之外的值呈指數級下降。 隨機圖的描繪（正如我們在 截至 2021 年 7 月閃電網路部分的視覺化 中看到的）看起來像網狀網路拓撲。 它看起來是去中心化和非層級的：每個節點似乎具有同等的重要性。 此外，隨機圖具有較大的直徑。 特別是，在這樣的圖中路由是具有挑戰性的，因為任意兩個節點之間的最短路徑相當長。

然而，形成鮮明對比的是，閃電圖完全不同。

今天的閃電圖

閃電網路是一個金融網路。 因此，網路的增長和形成也受到經濟激勵的影響。 每當節點加入閃電網路時，它可能希望最大化與其他節點的連通性以增加其路由效率。這種現象稱為優先連接。 這些經濟激勵導致了一個與隨機圖根本不同的網路。

根據公開宣布通道的快照，閃電網路的度分佈遵循冪律函數。 在這樣的圖中，絕大多數節點與其他節點的連接非常少，而只有少數節點有許多連接。 在高層次上，這種圖拓撲類似於星形：網路有一個連接良好的核心和一個鬆散連接的外圍。 具有冪律度分佈的網路也稱為無標度網路。 這種拓撲有利於高效路由付款，但容易受到某些基於拓撲的攻擊。

基於拓撲的攻擊

對手可能想要破壞閃電網路，並可能決定其目標是將整個網路拆分成許多較小的組件，使得在整個網路中進行付款路由實際上不可能。 一個不那麼雄心勃勃但仍然惡意和嚴重的目標可能只是攻陷某些網路節點。 這種破壞可能發生在節點層面或邊層面。

讓我們假設對手可以攻陷閃電網路中的任何節點。 例如，它可以通過分散式阻斷服務（DDoS）攻擊來攻擊它們或通過任何方式使它們無法運行。 事實證明，如果對手隨機選擇節點，那麼像閃電網路這樣的無標度網路對節點移除攻擊是健壯的。 這是因為隨機節點位於外圍，連接數量

較少，因此在網路連通性中扮演的角色可以忽略不計。 然而，如果對手更加謹慎，它可以針對連接最好的節點。 不出所料，閃電網路和其他無標度網路對有針對性的節點移除攻擊_不_健壯。

另一方面，對手可能更加隱蔽。 幾種基於拓撲的攻擊針對單一節點或單一支付通道。 例如，對手可能有興趣故意耗盡某個支付通道的容量。 更一般地說，對手可以耗盡節點的所有出站容量以將其從路由市場中淘汰。 這可以通過路由通過受害節點的付款輕鬆獲得，金額等於每個支付通道的出站容量。 完成這種所謂的節點隔離攻擊後，受害者除非收到付款或重新平衡其通道，否則無法再發送或路由付款。

總之，即使是設計上，也可以從可路由的閃電網路中移除邊和節點。 然而，根據所使用的攻擊向量，對手可能必須提供更多或更少的資源來執行攻擊。

閃電網路的時間性

閃電網路是一個動態變化的、無需許可的網路。 節點可以自由加入或離開網路，它們可以隨時開設和建立支付通道。 因此，閃電圖的單一靜態快照是具有誤導性的。我們需要考慮網路的時間性和不斷變化的性質。目前，閃電圖在節點和支付通道數量方面正在增長。 其有效直徑也在縮小；也就是說，節點彼此越來越近，正如我們在 閃電網路在節點、通道和鎖定容量方面的穩定增長（截至 2021 年 9 月） 中看到的。

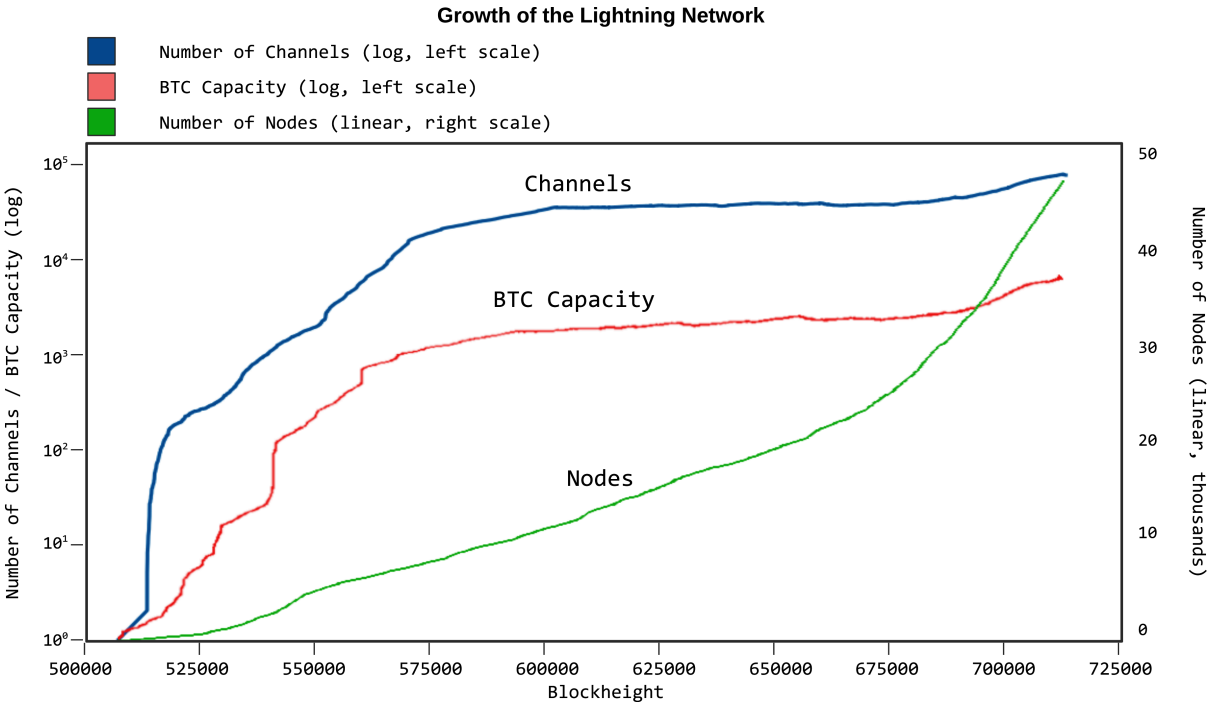

Figure 108. 閃電網路在節點、通道和鎖定容量方面的穩定增長（截至 2021 年 9 月）

在社交網路中，三角形閉合行為很常見。 具體來說，在節點代表人、友誼代表邊的圖中，預期圖中會出現三角形。 在這種情況下，三角形代表三個人之間的成對友誼。 例如，如果 Alice 認識 Bob，Bob 認識 Charlie，那麼在某個時候 Bob 很可能會將 Alice 介紹給 Charlie。 然而，這種行為在閃電網路中會很奇怪。 節點根本沒有動力去閉合三角形，因為它們本可以直接路由付款而不是開設新的支付通道。 令人驚訝的是，三角形閉合在閃電網路中是一種常見做法。 在

實作多部分付款之前，三角形的數量一直在穩定增長。 考慮到節點本可以通過三角形的兩邊路由付款而不是開設第三個通道，這是違反直覺和令人驚訝的。 這可能意味著路由效率低下激勵使用者閉合三角形而不是依賴路由。 希望多部分付款將有助於提高付款路由的效率。

16.9. 閃電網路中的中心化

評估圖中節點中心性的常用指標是其_介數中心性_。中心點支配度是從介數中心性派生的指標，用於評估網路的中心性。 有關中心點支配度的精確定義，請參閱 Freeman 的工作 (https://doi.org/10.2307/3033543)。

網路的中心點支配度越大，網路就越中心化。 我們可以觀察到閃電網路具有比同等大小的隨機圖（Erdős-Rényi 圖）或無標度圖（Barabási-Albert 圖）更大的中心點支配度（即更中心化）。

總的來說，我們對閃電通道圖動態性質的理解相當有限。 分析多部分付款等協定變更如何影響閃電網路的動態是有意義的。 更深入地探索閃電圖的時間性質將是有益的。

16.10. 經濟激勵和圖結構

閃電圖是自發形成的，節點基於共同利益相互連接。 因此，激勵驅動圖的發展。 讓我們看看一些相關的激勵：

- 理性激勵：
  - 節點建立通道來發送、接收和路由付款（賺取費用）。
  - 是什麼使通道更可能在兩個理性行事的節點之間建立？
- 利他激勵：
  - 節點「為了網路的利益」建立通道。
  - 雖然我們不應該將安全假設建立在利他主義之上，但在某種程度上，利他行為驅動比特幣（接受入站連接、提供區塊）。
  - 它在閃電網路中扮演什麼角色？

在閃電網路的早期階段，許多節點運營者聲稱賺取的路由費用無法補償來自流動性鎖定的機會成本。這表明運營節點可能主要是由「為了網路的利益」的利他激勵驅動的。 如果閃電網路有顯著更大的流量或者路由費用市場出現，這可能會在未來改變。 另一方面，如果節點希望優化其路由費用，它會最小化到每個其他節點的平均最短路徑長度。 換句話說，追求利潤的節點會嘗試將自己定位在通道圖的_中心_或接近 中心。

16.11. 使用者保護隱私的實用建議

我們仍處於閃電網路的早期階段。 本章列出的許多問題可能會隨著它的成熟和增長而得到解決。 與此同時，你可以採取一些措施來保護你的節點免受惡意使用者的攻擊；像更新節點運行的預設參數這樣簡單的事情就可以大大加強你的節點。

16.12. 未公告通道

如果你打算使用閃電網路在你控制的節點和錢包之間發送和接收資金，並且對路由其他使用者的付款不感興趣，那麼幾乎不需要向網路的其餘部分公告你的通道。 你可以在你運行完整節點的桌面電腦和你運行閃電錢包的手機之間開設一個通道，並簡單地放棄 閃電網路運作原理 中討論的通道公告。 這些有時被稱為「私有」通道；然而，更正確的說法是稱它們為「未公告」通道，因為它們並不是嚴格私有的。

未公告通道不會為網路的其餘部分所知，通常不會用於路由其他使用者的付款。 如果其他節點知道它們，它們仍然可以用於路由付款；例如，發票可能包含建議具有未公告通道的路徑的路由提示。 然而，假設你只與自己開設了未公告通道，你確實獲得了一些隱私。 由於你沒有將通道暴露給網路，你降低了節點遭受阻斷服務攻擊的風險。 你還可以更容易地管理這個通道的容量，因為它只會用於直接接收或發送到你的節點。

與已知的、經常交易的一方開設未公告通道也有優勢。 例如，如果 Alice 和 Bob 經常用比特幣玩撲克，他們可以開設一個通道來來回發送獎金。 在正常情況下，這個通道不會用於路由其他使用者的付款或收取費用。 由於通道不會為網路的其餘部分所知，Alice 和 Bob 之間的任何付款都無法通過追蹤通道路由容量的變化來推斷。 這為 Alice 和 Bob 提供了一些隱私；然而，如果其中一人決定讓其他使用者知道該通道，例如將其包含在發票的路由提示中，那麼這種隱私就會喪失。

還應該注意的是，要開設未公告通道，必須在比特幣區塊鏈上進行公開交易。 因此，如果惡意方監控區塊鏈的通道開設交易並嘗試將它們與網路上的通道匹配，則可以推斷通道的存在和大小。 此外，當通道關閉時，一旦它被提交到比特幣區塊鏈，通道的最終餘額將被公開。 然而，由於開設和承諾交易是假名的，將其連接回 Alice 或 Bob 不會是一件簡單的事。 此外，2021 年的 Taproot 升級使得難以區分通道開設和關閉交易與其他特定類型的比特幣交易。 因此，雖然未公告通道並不完全私有，但如果謹慎使用，它們確實提供了一些隱私優勢。

16.13. 路由考慮

如 阻斷服務 所述，開設公共通道的節點會面臨其通道受到一系列攻擊的風險。 雖然協定層面的緩解措施正在開發中，但節點可以採取許多步驟來保護其公共通道免受阻斷服務攻擊：

最小 HTLC 大小

在通道開設時，你的節點可以設定它將接受的最小 HTLC 大小。 設定更高的值確保你的每個可用通道插槽不會被非常小的付款佔用。

速率限制

許多節點實作允許節點動態接受或拒絕通過你的節點轉發的 HTLC。 自訂速率限制器的一些有用準則如下：

- 限制單一對等節點可能消耗的承諾插槽數量
- 監控單一對等節點的失敗率，如果其失敗突然激增則進行速率限制

影子通道

希望向單一目標開設大型通道的節點可以改為向目標開設單一公共通道，並通過進一步的私有通道（稱為 影子通道 (https://anchor.fm/tales-from-the-crypt/episodes/197-Joost-Jager-ekghn6)）支援它。這些通道仍然可以用於路由，但不會向潛在攻擊者公告。

16.13.1. 接受通道

目前，閃電節點難以引導入站流動性。雖然有一些付費解決方案來獲取入站流動性，如交換服務、通道市場和來自已知中心的付費通道開設服務，但許多節點將樂於接受任何看起來合法的通道開設請求以增加其入站流動性。

回到比特幣的背景，這可以與 Bitcoin Core 對待其入站和出站連接的方式進行比較，出於擔心節點可能被日蝕攻擊而區別對待。 如果一個節點向你的比特幣節點開設入站連接，你無法知道發起者是隨機選擇你還是有惡意意圖地專門針對你的節點。 你的出站連接不需要如此懷疑，因為節點要麼是從許多潛在對等節點的池中隨機選擇的，要麼你是故意手動連接到該對等節點的。

閃電網路也可以這樣說。 當你開設通道時，這是有意為之的，但當遠端方向你的節點開設通道時，你無法知道這個通道是否會被用來攻擊你的節點。 正如幾篇論文所指出的，啟動節點並向目標開設通道的相對較低成本是使攻擊容易的重要因素之一。 如果你接受入站通道，明智的做法是對你接受入站通道的節點設定一些限制。 許多實作暴露了通道接受掛鉤，允許你根據自己的偏好定制通道接受策略。

接受和拒絕通道的問題是一個哲學問題。 如果我們最終得到一個新節點無法參與因為無法開設任何通道的閃電網路會怎樣？ 我們的建議不是設定一個你將接受通道的「超級中心」的排他列表，而是以適合你風險偏好的方式接受通道。

一些潛在的策略是：

無風險

不接受任何入站通道。

低風險

僅接受來自你以前成功開設通道的已知節點集的通道。

中等風險

僅接受來自在圖中存在較長時間並擁有一些長期通道的節點的通道。

較高風險

接受任何入站通道，並實施 路由考慮 中描述的緩解措施。

16.14. 結論

總之，隱私和安全是細緻複雜的話題，雖然許多研究人員和開發人員正在尋找全網路範圍的改進，但參與網路的每個人都了解他們可以做什麼來保護自己的隱私並在個別節點層面提高安全性是很重要的。

16.15. 參考文獻和延伸閱讀

在本章中，我們使用了許多來自正在進行的閃電網路安全研究的參考文獻。你可以在以下按主題列出的列表中找到這些有用的文章和論文。

隱私和探測攻擊

- Jordi Herrera-Joancomartí 等人. "On the Difficulty of Hiding the Balance of Lightning Network Channels" (https://eprint.iacr.org/2019/328). *Asia CCS '19: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*（2019 年 7 月）：602–612。
- Utz Nisslmueller 等人. "Toward Active and Passive Confidentiality Attacks on Cryptocurrency Off-Chain Networks." arXiv 預印本，https://arxiv.org/abs/2003.00003[]（2020）。
- Sergei Tikhomirov 等人. "Probing Channel Balances in the Lightning Network." arXiv 預印本，https://arxiv.org/abs/2004.00333[]（2020）。
- George Kappos 等人. "An Empirical Analysis of Privacy in the Lightning Network." arXiv 預印本，https://arxiv.org/abs/2003.12470[]（2021）。
- Zap 原始碼中的探測函數 (https://github.com/LN-Zap/zap-desktop/blob/v0.7.2-beta/services/grpc/router.methods.js)。

擁塞攻擊

- Ayelet Mizrahi 和 Aviv Zohar. "Congestion Attacks in Payment Channel Networks." arXiv 預印本，https://arxiv.org/abs/2002.06564[]（2020）。

路由考慮

- Marty Bent，Joost Jager 訪談，*Tales from the Crypt*，播客音訊，2020 年 10 月 2 日，https://anchor.fm/tales-from-the-crypt/episodes/197-Joost-Jager-ekghn6[]。

17. 結論

在短短幾年內，閃電網路已從一篇白皮書發展成為一個快速成長的全球網路。作為比特幣的第二層，它實現了快速、低成本和私密支付的承諾。此外，它還引發了一場創新海嘯，將開發人員從比特幣開發中存在的同步共識限制中解放出來。

閃電網路的創新正在多個不同層面發生：

- 在比特幣核心協定層面，為新的比特幣腳本操作碼、簽名演算法和最佳化提供使用需求
- 在閃電協定層面，新功能在全網快速部署
- 在支付通道層面，有新的通道結構和增強功能
- 作為獨立的可選功能，由獨立實作端對端部署，發送者和接收者可以按需使用
- 在客戶端和協定之上構建新穎且令人興奮的閃電應用程式（LApps）

讓我們來看看這些創新如何在現在和不久的將來改變閃電網路。

17.1. 去中心化和非同步創新

閃電網路不受同步共識的約束，這與比特幣不同。這意味著不同的閃電客戶端可以實作不同的功能並協商它們的互動（參見 功能位元和協定可擴展性）。因此，閃電網路的創新速度比比特幣快得多。

閃電網路不僅在快速發展，而且正在為比特幣系統創造對新功能的需求。比特幣中許多近期和計劃中的創新都是因為它們在閃電網路中的使用而產生動機和理由。事實上，閃電網路經常被提及作為許多新功能的範例使用案例。

17.1.1. 比特幣協定和比特幣腳本創新

比特幣系統在本質上是一個保守的系統，必須保持與共識規則的相容性，以避免區塊鏈的意外分叉或 P2P 網路的分裂。因此，新功能在實作到主網（實際運行的生產系統）之前需要大量的協調和測試。

以下是一些當前或提議中由閃電網路使用案例推動的比特幣創新：

Neutrino

一種輕量級客戶端協定，與傳統的 SPV 協定相比具有改進的隱私功能。Neutrino 主要被閃電客戶端用來存取比特幣區塊鏈。

Schnorr 簽名

作為 *Taproot* 軟分叉的一部分引入，Schnorr 簽名將使閃電網路中的靈活點時間鎖定合約（PTLCs）成為可能用於通道建構。這可能特別會使用可撤銷簽名而不是可撤銷交易。

Taproot

也是 2021 年 11 月引入 Schnorr 簽名的軟分叉的一部分，Taproot 允許複雜的腳本看起來像單一付款人、單一收款人的支付，與比特幣上最常見的支付類型無法區分。這將使閃電通道的協作（相互）關閉交易看起來與簡單支付無法區分，並增強閃電網路使用者的隱私。

輸入重新綁定

也被稱為 SIGHASH_NOINPUT 或 SIGHASH_ANYPREVOUT，這個計劃中的比特幣腳本語言升級主要是由高級智慧合約（如 eltoo 通道協定）所推動。

契約（Covenants）

目前處於研究的早期階段，契約允許交易建立輸出，以限制未來花費這些輸出的交易。這種機制可以通過在承諾交易中強制執行地址白名單來提高閃電通道的安全性。

17.1.2. 閃電協定創新

閃電 P2P 協定具有高度可擴展性，自其誕生以來已經歷了大量變化。功能位元中使用的「奇數也可以」規則（參見 功能位元和協定可擴展性）確保節點可以協商它們支援的功能，從而實現協定的多個獨立升級。

17.1.3. TLV 可擴展性

類型-長度-值（參見 類型-長度-值格式）擴展訊息協定的機制非常強大，已經在保持向前和向後相容性的同時，實現了閃電網路中多項新功能的引入。 一個突出的範例是正在開發中的路徑盲化和蹦床支付，它允許接收者對發送者隱藏自己，同時也允許行動客戶端在不需要在設備上儲存完整通道圖的情況下發送支付，透過使用第三方來實現，而且不需要向該第三方透露最終接收者。

17.1.4. 支付通道建構

支付通道是由兩個通道夥伴運營的抽象概念。只要這兩方願意運行新程式碼，他們就可以同時實作各種通道機制。事實上，最近的研究表明，通道甚至可以動態升級到新機制，而無需關閉舊通道並開啟新類型的通道。

eltoo

一種提議的通道機制，使用輸入重新綁定來顯著簡化支付通道的操作，並消除對懲罰機制的需求。它需要一種新的比特幣簽名類型才能實作。

17.1.5. 可選的端對端功能

點時間鎖定合約（PTLCs）

一種不同於 HTLC 的方法，PTLC 可以增強隱私、減少洩露給中間節點的資訊，並且比基於 HTLC 的通道運作更有效率。

大型通道

大型或 *Wumbo* 通道以動態方式引入網路，無需協調。支援大額支付的通道作為通道公告訊息的一部分進行廣播，可以以可選方式使用。

多部分支付（MPP）

MPP 也是以可選方式引入的，更棒的是它只需要支付的發送者和接收者能夠使用 MPP。網路的其餘部分只需像路由單一部分支付一樣路由 HTLC。

JIT 路由

一種可選方法，可被路由節點用來提高其可靠性並保護自己免受垃圾訊息攻擊。

Keysend

由閃電客戶端實作獨立引入的升級，它允許發送者以「主動」和非同步的方式發送資金，而無需事先取得發票。

HODL 發票[11]

最終 HTLC 不被收取的支付，使發送者承諾該支付，但允許接收者延遲收取直到滿足某些其他條件，或在不收取的情況下取消發票。這也是由不同的閃電客戶端獨立實作的，可以以可選方式使用。

洋蔥路由訊息服務

洋蔥路由機制和節點的底層公鑰資料庫可用於發送與支付無關的資料，例如文字訊息或論壇貼文。使用閃電網路實現付費訊息作為解決垃圾貼文和女巫攻擊（垃圾訊息）的方案，是另一項獨立於核心協定實作的創新。

Offers

目前作為 BOLT #12 提議但已被某些節點實作，這是一種通訊協定，用於透過洋蔥訊息從遠端節點請求（週期性）發票。

17.2. 閃電應用程式（LApps）

雖然仍處於起步階段，我們已經看到有趣的閃電應用程式開始出現。廣義上定義為使用閃電協定或閃電客戶端作為組件的應用程式，LApps 是閃電網路的應用層。 一個突出的範例是 LNURL，它提供與 BOLT #12 offers 類似的功能，但是透過 HTTP 和閃電地址實現。它在 offers 之上運作，為使用者提供類似電子郵件風格的地址，其他人可以向該地址發送資金，而軟體在背景向節點的 LNURL 端點請求發票。 更多的 LApps 正在為簡單遊戲、訊息應用程式、微服務、可付費 API、付費自動販賣機（例如加油機）、衍生品交易系統等建構。

17.3. 準備就緒，出發！

未來看起來一片光明。閃電網路正在將比特幣帶入新的未探索市場和應用領域。憑藉本書中的知識，你可以探索這個新疆界，甚至可能作為先驅者加入，開闢一條新路徑。

附錄

術語表

本快速術語表包含許多與比特幣和閃電網路相關的術語。這些術語在整本書中都有使用，請將此頁面加入書籤以便快速參考。

地址（address）

比特幣地址緊湊地編碼了向接收者付款所需的資訊。現代地址由以 bc1 開頭的字母和數字組成，看起來像 bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4。地址是接收者鎖定腳本的簡寫，發送者可以用它來簽署資金給接收者。大多數地址要麼代表接收者的公鑰，要麼代表某種定義更複雜消費條件的腳本。前面的範例是一個 bech32 地址，將資金鎖定到公鑰雜湊的見證程式（參見 *Pay-to-Witness-Public-Key-Hash*）。還有以 1 或 3 開頭的較舊地址格式，使用 Base58Check 地址編碼來表示公鑰雜湊或腳本雜湊。

非對稱加密系統（asymmetric cryptographic system）

非對稱加密或公鑰加密是一種使用金鑰對的加密系統：公鑰可以廣泛傳播，而私鑰只有擁有者知道。 這類金鑰的生成依賴於基於數學問題的加密演算法，以產生單向容易計算但反向非常難以解決的函數。 有效的安全性只需要保持私鑰的私密性；公鑰可以公開分發而不會影響安全性。

自動導航（autopilot）

自動導航是閃電網路節點的推薦引擎，使用閃電網路拓撲的統計資料來建議節點應該與哪些節點開啟通道。 根據自動導航的實作，還可能推薦通道容量。 自動導航不是閃電網路協定的一部分。

餘額（balance）

支付通道的餘額是屬於每個通道夥伴的比特幣數量。 例如，Alice 可以與 Bob 開啟一個價值 1 BTC 的通道。 那麼通道餘額就是 Alice 1 BTC，Bob 0 BTC。 當使用者進行交易時，通道餘額會更新。 例如，如果 Alice 向 Bob 發送 0.2 BTC，那麼餘額現在是 Alice 0.8 BTC，Bob 0.2 BTC。 當通道關閉時，通道中的比特幣將根據承諾交易中編碼的最新餘額在兩個通道夥伴之間分配。 在閃電網路中，發送和接收支付的能力受到通道餘額的限制。 參見 *容量*。

bech32

bech32 指的是一種通用的帶校驗和的 base32 編碼格式，具有強大的錯誤檢測保證。雖然 bech32 最初是為原生 SegWit 輸出的地址格式而開發的（BIP-173），但它也用於編碼閃電發票（BOLT #11）。原生 SegWit 版本 0 輸出（P2WPKH 和 P2WSH）使用 bech32，而更高版本的原生 SegWit 輸出（例如 Pay-to-Taproot 或 P2TR）使用改進的變體 bech32m（BIP-350）。bech32m 地址有時被稱為「bc1」地址，反映了這類地址的前綴。原生 SegWit 輸出比舊地址更節省區塊空間，因此可能為這類地址的擁有者減少交易費用。

比特幣改進提案（Bitcoin Improvement Proposal，BIP）

比特幣社群成員提交的改進比特幣的提案。例如，BIP-21 是一個改進比特幣統一資源識別符（URI）方案的提案。BIP 可以在 GitHub (https://github.com/bitcoin/bips) 找到。

bitcoin（比特幣）、Bitcoin（比特幣）

根據上下文，可能指貨幣單位（硬幣）、網路或底層支援協定的名稱。以小寫「b」書寫的 bitcoin 通常指貨幣單位。以大寫「B」書寫的 Bitcoin 通常指協定或系統。

比特幣挖礦（Bitcoin mining）

比特幣挖礦是從最近的比特幣交易構建區塊，然後解決作為工作量證明所需的計算問題的過程。 這是更新共享比特幣帳本（即比特幣區塊鏈）以及將新交易納入帳本的過程。 這也是發行新比特幣的過程。 每當創建新區塊時，挖礦節點將收到在該區塊的 coinbase 交易中創建的新比特幣。

區塊（block）

區塊是比特幣區塊鏈中的資料結構，由標頭和比特幣交易主體組成。 區塊帶有時間戳，並承諾特定的前驅（父）區塊。 當對區塊標頭進行雜湊時，提供使區塊鏈在機率上不可變的工作量證明。 區塊必須遵守網路共識強制執行的規則才能擴展區塊鏈。 當區塊附加到區塊鏈時，包含的交易被認為有了第一次確認。

區塊鏈（blockchain）

區塊鏈是所有比特幣交易的分散式日誌或資料庫。 交易被分組到稱為區塊的離散更新中，限制為最多 400 萬權重單位。 區塊大約每 10 分鐘透過稱為挖礦的隨機過程產生。 每個區塊都包含計算密集型的「工作量證明」。 工作量證明要求用於調節區塊間隔並保護區塊鏈免受重寫歷史的攻擊： 攻擊者需要超越現有的工作量證明才能替換已發布的區塊，使每個區塊隨著被後續區塊埋藏而在機率上變得不可變。

BOLT

BOLT，即閃電技術基礎（Basis of Lightning Technology），是閃電網路的正式規範。與比特幣不同（比特幣有一個參考實作同時也作為協定規範），各種閃電網路實作遵循 BOLT 以便它們可以相互協作形成同一個網路。可在 GitHub (https://github.com/lightningnetwork/lightning-rfc) 獲取。

容量（capacity）

支付通道的容量等於資金交易提供的比特幣數量。 因為資金交易在區塊鏈上公開可見，且通道透過八卦協定宣告，所以容量是公開資訊。 它不會透露通道中每個通道夥伴擁有多少比特幣的任何資訊，即餘額。 高容量並不保證通道可以在兩個 方向 上都用於路由。

c-lightning

由位於維多利亞的公司 Blockstream (https://blockstream.com) 實作的閃電網路協定。以 C 語言編寫。原始碼在 GitHub (https://github.com/ElementsProject/lightning)。

關閉交易（closing transaction）

如果兩個通道夥伴同意關閉通道，他們將創建一個反映最新承諾交易的結算交易。 在交換關閉交易的簽名後，不應再進行進一步的通道更新。 透過關閉交易相互關閉通道的優點是，與透過發布承諾交易單方面強制關閉通道相比，需要較少的區塊鏈交易來索取所有資金。此外，雙方的資金可以從關閉交易中立即使用。

CLTV

CLTV 是比特幣腳本運算子 OP_CHECKLOCKTIMEVERIFY 的縮寫。這定義了在輸出可以被花費之前的絕對區塊高度。路由過程的原子性在很大程度上取決於 HTLC 中的 CLTV 值。路由節點透過八卦協定宣告它們希望任何傳入和傳出 HTLC 的預期 CLTV 到期延遲。

coinbase

coinbase 是一個特殊欄位，只允許在 coinbase 交易的唯一輸入中使用。 coinbase 允許最多 100 位元組的任意資料，但自 BIP-34 以來，它必須首先包含當前區塊高度以確保 coinbase 交易是唯一的。 不要與 coinbase 交易混淆。

coinbase 交易（coinbase transaction）

區塊中的第一筆交易，總是由礦工創建，包含一個 coinbase。 coinbase 交易可以索取區塊獎勵並將其分配給一個或多個輸出。 區塊獎勵由區塊補貼（新創建的比特幣）和區塊中包含的所有交易費用的總和組成。 coinbase 輸出只能在成熟 100 個區塊後才能使用。 如果區塊包含任何 SegWit 交易，coinbase 交易必須在額外的輸出中包含對見證交易識別符的承諾。

冷儲存（cold storage）

指將一定數量的比特幣保持離線。當比特幣私鑰在安全的離線環境中創建和儲存時，就實現了冷儲存。冷儲存對於保護比特幣持有量很重要。線上電腦容易受到駭客攻擊，不應用於儲存大量比特幣。

承諾交易（commitment transaction）

承諾交易是由兩個通道夥伴簽署的比特幣交易，編碼了通道的最新餘額。 每當使用通道進行新的支付或轉發時，通道餘額會更新，雙方會簽署新的承諾交易。 重要的是，在 Alice 和 Bob 之間的通道中，Alice 和 Bob 都保留自己版本的承諾交易，該交易也由對方簽署。 在任何時候，通道 可以由 Alice 或 Bob 關閉，如果他們將自己的承諾交易提交到比特幣區塊鏈。在閃電網路中，提交較舊（過時）的承諾交易被視為 *作弊*（即協定違規），可能會被對方懲罰，透過懲罰交易索取通道中的所有資金。

確認（confirmations）

一旦交易被包含在區塊中，它就有一次確認。一旦在區塊鏈上挖出 *另一個* 區塊，該交易就有兩次確認，依此類推。六次或更多確認被認為是交易無法被逆轉的充分證明。

合約（contract）

合約是一組比特幣交易，它們一起產生某種期望的行為。 範例包括用於創建無需信任的雙向支付通道的 RSMC，或用於創建允許透過第三方無需信任地轉發支付的機制的 HTLC。

迪菲-赫爾曼金鑰交換（Diffie–Hellman Key Exchange，DHKE）

在閃電網路中，使用橢圓曲線迪菲-赫爾曼（ECDH）方法。 這是一種匿名金鑰協議協定，允許兩方各自擁有橢圓曲線公私鑰對，在不安全的通訊通道上建立共享秘密。 這個共享秘密可以直接用作金鑰，或用於派生另一個金鑰。 金鑰或派生金鑰然後可用於使用對稱金鑰密碼加密後續通訊。 派生金鑰的一個範例是洋蔥發送者的臨時會話金鑰與洋蔥某一跳的節點公鑰之間的共享秘密，如 SPHINX Mix Format 所描述和使用的。

數位簽章（digital signature）

數位簽章是一種用於驗證數位訊息或文件的真實性和完整性的數學方案。 它可以被視為一種加密承諾，其中訊息不被隱藏。

雙重支付（double-spending）

雙重支付是成功地多次花費某些金錢的結果。 比特幣透過驗證添加到區塊鏈的每筆交易都遵守共識規則來防止雙重支付；這意味著檢查交易的輸入之前沒有被花費過。

橢圓曲線數位簽章演算法（Elliptic Curve Digital Signature Algorithm，ECDSA）

橢圓曲線數位簽章演算法或 ECDSA 是比特幣使用的一種加密演算法，以確保資金只能由正確私鑰的持有者使用。

Eclair

由位於巴黎的公司 ACINQ (https://acinq.co) 實作的閃電網路協定。以 Scala 語言編寫。原始碼在 GitHub (https://github.com/ACINQ/eclair)。

編碼（encoding）

編碼是將訊息轉換為不同形式的過程。例如，將數字從十進位轉換為十六進位。

Electrum 伺服器（Electrum server）

Electrum 伺服器是帶有額外介面（API）的比特幣節點。不運行完整節點的比特幣錢包通常需要它。例如，這些錢包使用 Electrum 伺服器 API 檢查特定交易的狀態或將交易廣播到記憶池。一些閃電錢包也使用 Electrum 伺服器。

臨時金鑰（ephemeral key）

臨時金鑰是僅短時間使用且使用後不保留的金鑰。它們通常從另一個長期持有的金鑰派生，用於一個會話。臨時金鑰主要用於閃電網路上的 SPHINX Mix Format 和洋蔥路由。 這增加了傳輸訊息或支付的安全性。 即使臨時金鑰洩露，也只有關於單個會話的資訊會公開。

功能位元（feature bits）

閃電節點用來相互通訊它們支援哪些功能的二進位字串。 功能位元包含在許多閃電訊息以及 BOLT #11 中。 它們可以使用 BOLT #9 解碼，並會告訴節點該節點啟用了哪些功能，以及這些功能是否向後相容。 也稱為功能旗標。

費用（fees）

在閃電網路的上下文中，節點會為轉發其他使用者的支付收取路由費用。 各個節點可以設定自己的費用政策，計算方式為固定 base_fee 和取決於支付金額的 fee_rate 的總和。 在比特幣的上下文中，交易的發送者向礦工支付交易費用以將交易包含在區塊中。 比特幣交易費用不包括基本費用，並且線性地取決於交易的權重，而不是金額。

資金交易（funding transaction）

資金交易用於開啟支付通道。資金交易的價值（以比特幣計）恰好是支付通道的容量。 資金交易的輸出是 2-of-2 多重簽名腳本（多重簽名），其中每個通道夥伴控制一個金鑰。由於其多重簽名性質，它只能透過通道夥伴之間的相互同意來花費。 它最終將被承諾交易之一或關閉交易花費。

全域功能（global features，globalfeatures 欄位）

閃電節點的全域功能是所有其他節點感興趣的功能。 它們最常與支援的路由格式相關。 它們在對等協定的 `init` 訊息以及八卦協定的 `channel_announcement` 和 `node_announcement` 訊息中宣告。

八卦協定（gossip protocol）

閃電網路節點透過與對等節點交換的八卦訊息發送和接收有關閃電網路拓撲的資訊。 八卦協定主要在 BOLT #7 中定義，並定義了 `node_announcement`、`channel_announcement` 和 `channel_update` 訊息的格式。 為防止垃圾訊息，節點公告訊息只有在節點已有通道時才會被轉發，通道公告訊息只有在通道的資金交易被比特幣網路確認時才會被轉發。 通常，閃電節點與其通道夥伴連接，但也可以與任何其他閃電節點連接以處理八卦訊息。

硬體錢包（hardware wallet）

硬體錢包是一種特殊類型的比特幣錢包，將使用者的私鑰儲存在安全的硬體設備中。 截至本書撰寫時，閃電網路節點沒有可用的硬體錢包，因為閃電網路使用的金鑰需要保持線上才能參與協定。

雜湊（hash）

某些任意長度二進位輸入的固定大小數位指紋。也稱為 *摘要*。

基於雜湊的訊息認證碼（hash-based message authentication code，HMAC）

HMAC 是一種基於雜湊函數和加密金鑰來驗證訊息完整性和真實性的演算法。 它用於洋蔥路由以確保每一跳封包的完整性，以及用於訊息加密的 Noise 協定變體中。

雜湊函數（hash function）

加密雜湊函數是一種數學演算法，將任意大小的資料映射到固定大小的位元字串（雜湊），並被設計為單向函數，即反向計算是不可行的。 從理想的加密雜湊函數的輸出重建輸入資料的唯一方法是嘗試對可能的輸入進行暴力搜尋以查看它們是否產生匹配。

雜湊鎖（hashlock）

雜湊鎖是一種比特幣腳本消費條件，在指定的資料被揭示之前限制輸出的消費。雜湊鎖有一個有用的特性，即一旦任何雜湊鎖透過消費被揭示，任何使用相同金鑰保護的其他雜湊鎖也可以被消費。這使得可以創建多個都受相同雜湊鎖約束的輸出，並且它們都在同一時間變得可消費。

雜湊時間鎖定合約（hash time-locked contract，HTLC）

雜湊時間鎖定合約（HTLC）是一種比特幣腳本，由雜湊鎖和時間鎖組成，要求支付的接收者要麼在截止日期前透過呈現雜湊原像來消費支付，要麼發送者可以在時間鎖到期後索取退款。 在閃電網路中，HTLC 是支付通道承諾交易中的輸出，用於實現無需信任的支付路由。

發票（invoice）

閃電網路上的支付流程由接收者（收款人）發起，發起時會開具發票，也稱為 *支付請求*。發票包括支付雜湊、金額、描述和到期時間。閃電發票在 BOLT #11 中定義。 發票還可以包含一個備用比特幣地址，在找不到路由的情況下可以向該地址付款，以及透過私有通道路由支付的提示。

即時路由（just-in-time routing，JIT routing）

即時（JIT）路由是源路由的替代方案，最初由共同作者 René Pickhardt 提出。 使用 JIT 路由，路徑上的中間節點可以暫停進行中的支付以在繼續支付之前重新平衡其通道。 這可能允許它們成功轉發因缺乏傳出容量而可能失敗的支付。

閃電訊息（Lightning message）

閃電訊息是可以在閃電網路上兩個對等節點之間發送的加密資料字串。與其他通訊協定類似，閃電訊息由標頭和主體組成。標頭和主體都有自己的 HMAC。閃電訊息是訊息層的主要構建區塊。

閃電網路（Lightning Network）、閃電網路協定（Lightning Network Protocol）、閃電協定（Lightning Protocol）

閃電網路是建立在比特幣（或其他加密貨幣）之上的協定。 它創建了一個支付通道網路，透過 HTLC 和洋蔥路由實現無需信任的支付轉發。 閃電網路的其他組成部分是八卦協定、傳輸層和支付請求。

閃電網路協定套件（Lightning Network protocol suite）

閃電網路協定套件由五層組成，負責協定的各個部分。 從底層（第一層）到頂層（第五層），這些層分別稱為網路通訊層、訊息層、對等層、路由層和支付層。 各種 BOLT 定義了一層或多層的部分。

閃電網路節點（Lightning Network node）、閃電節點（Lightning node）

透過閃電對等協定參與閃電網路的電腦。 閃電節點能夠與其他節點開啟通道、發送和接收支付、以及路由其他使用者的支付。 通常，閃電節點使用者也會運行比特幣節點。

lnd

由位於舊金山的公司 Lightning Labs (https://lightning.engineering) 實作的閃電網路協定。 以 Go 語言編寫。原始碼在 GitHub (https://github.com/lightningnetwork/lnd)。

本地功能（local features，欄位：localfeatures）

閃電網路節點的本地功能是其對等節點直接感興趣的可配置功能。 它們在對等協定的 `init` 訊息以及八卦協定的 `channel_announcement` 和 `node_announcement` 訊息中宣告。

鎖定時間（locktime）

鎖定時間，或更技術性地說 nLockTime，是比特幣交易的一部分，指示該交易可以添加到區塊鏈的最早時間或最早區塊。

訊息層（messaging layer）

訊息層建立在閃電網路協定套件的網路連接層之上。 它負責確保透過所選網路連接層協定進行加密和安全的通訊和資訊交換。 訊息層定義了 BOLT #1 中定義的閃電訊息的框架和格式。 BOLT #9 中定義的功能位元也是這一層的一部分。

毫聰（millisatoshi）

閃電網路上最小的記帳單位。一毫聰是單個比特幣的千億分之一。一毫聰是一聰的千分之一。毫聰不存在於比特幣網路上，也不能在比特幣網路上結算。

多部分支付（multipart payments，MPP）

多部分支付（MPP），通常也稱為多路徑支付，是一種將支付金額分成多個較小部分並透過一條或多條路徑傳送的方法。由於 MPP 可以透過單一路徑發送許多或所有部分，因此多部分支付這個術語比多路徑支付更準確。在電腦科學中，多部分支付被建模為網路流。

多重簽名（multisignature）

多重簽名（多簽）指的是需要多個簽名才能授權消費的腳本。 支付通道總是編碼為需要支付通道每個夥伴各一個簽名的多重簽名地址。 在兩方支付通道的標準情況下，使用 2-of-2 多重簽名地址。

節點（node）

參見 *閃電網路節點*。

網路容量（network capacity）

閃電網路容量是鎖定在閃電網路內部並在其中流通的比特幣總量。 它是每個公共通道容量的總和。 它在某種程度上反映了閃電網路的使用情況，因為我們預期人們將比特幣放入閃電通道以消費或轉發其他使用者的支付。 因此，閃電通道中的比特幣數量越多，閃電網路的預期使用量就越高。 請注意，由於只能觀察到公共通道容量，真正的網路容量是未知的。此外，由於通道的容量可以支援無限次的來回支付，網路容量並不意味著閃電網路上傳輸價值的限制。

網路連接層（network connection layer）

閃電網路協定套件的最底層。 其職責是支援網際網路協定如 IPv4、IPv6、TOR2 和 TOR3，並使用它們建立如 BOLT #8 中定義的安全加密通訊通道，或使用 DNS 進行如 BOLT #10 中定義的網路引導。

Noise_XK

Noise 協定框架的模板，用於在閃電網路的兩個對等節點之間建立經過認證和加密的通訊通道。 X 表示不需要預先知道連接發起者的公鑰。 K 表示需要預先知道接收者的公鑰。

洋蔥路由（onion routing）

洋蔥路由是一種在電腦網路上進行匿名通訊的技術。 在洋蔥網路中，訊息被封裝在多層加密中，類似於洋蔥的層次。 加密的資料透過一系列稱為洋蔥路由器的網路節點傳輸，每個節點剝去一層，揭示資料的下一個目的地。 當最後一層被解密時，訊息到達其目的地。 發送者保持匿名，因為每個中間節點只知道緊鄰的前一個和後一個節點的位置。

輸出（output）

比特幣交易的輸出，也稱為未花費交易輸出（UTXO）。 輸出是可以被花費的不可分割的比特幣數量，以及定義需要滿足什麼條件才能花費該比特幣的腳本。 每筆比特幣交易都消費一些先前記錄的交易的輸出，並創建可由後續交易花費的新輸出。 典型的比特幣輸出需要簽名才能花費，但輸出可以要求滿足更複雜的腳本。 例如，多重簽名腳本要求兩個或更多金鑰持有者簽名才能花費輸出，這是閃電網路的基本構建區塊。

付款至公鑰雜湊（Pay-to-Public-Key-Hash，P2PKH）

P2PKH 是一種將比特幣鎖定到公鑰雜湊的輸出類型。被 P2PKH 腳本鎖定的輸出可以透過呈現與雜湊匹配的公鑰和由相應私鑰創建的數位簽章來解鎖（花費）。

付款至腳本雜湊（Pay-to-Script-Hash，P2SH）

P2SH 是一種多功能的輸出類型，允許使用複雜的比特幣腳本。使用 P2SH，詳細說明花費輸出條件的複雜腳本（贖回腳本）不會出現在鎖定腳本中。相反，價值被鎖定到腳本的雜湊，必須呈現並滿足該腳本才能花費輸出。

P2SH 地址（P2SH address）

P2SH 地址是腳本 20 位元組雜湊的 Base58Check 編碼。P2SH 地址以「3」開頭。P2SH 地址隱藏了所有複雜性，因此支付的發送者看不到腳本。

付款至見證公鑰雜湊（Pay-to-Witness-Public-Key-Hash，P2WPKH）

P2WPKH 是 P2PKH 的 SegWit 等價物，使用隔離見證。花費 P2WPKH 輸出的簽名放在見證樹中而不是 ScriptSig 欄位中。參見 *SegWit*。

P2WPKH 地址（P2WPKH address）

「原生 SegWit v0」地址格式，P2WPKH 地址是 bech32 編碼的，以「bc1q」開頭。

付款至見證腳本雜湊（Pay-to-Witness-Script-Hash，P2WSH）

P2WSH 是 P2SH 的 SegWit 等價物，使用隔離見證。花費 P2WSH 輸出的簽名和腳本放在見證樹中而不是 ScriptSig 欄位中。參見 *SegWit*。

P2WSH 地址（P2WSH address）

「原生 SegWit v0」腳本地址格式，P2WSH 地址是 bech32 編碼的，以「bc1q」開頭。

付款至 Taproot（Pay-to-Taproot，P2TR）

於 2021 年 11 月啟用，Taproot 是一種新的輸出類型，將比特幣鎖定到消費條件樹或單一根條件。

P2TR 地址（P2TR address）

Taproot 地址格式，代表 SegWit v1，是 bech32m 編碼的地址，以「bc1p」開頭。

支付（payment）

當比特幣在閃電網路內轉移時，就會發生閃電支付。支付通常不會出現在比特幣區塊鏈上。

支付通道（payment channel）

支付通道是閃電網路上兩個節點之間的金融關係，使用支付給多重簽名地址的比特幣交易創建。 通道夥伴可以使用通道在彼此之間來回發送比特幣，而無需將所有交易提交到比特幣區塊鏈。 在典型的支付通道中，只有兩筆交易——資金交易和承諾交易——被添加到區塊鏈。 透過通道發送的支付不會記錄在區塊鏈上，據說是「鏈下」發生的。

支付層（payment layer）

閃電網路協定套件的頂層和第五層，運作在路由層之上。 其職責是透過 BOLT #11 發票實現支付流程。 雖然它大量使用 BOLT #7 中定義的八卦協定的通道圖，但實際的支付傳遞策略不是協定規範的一部分，留給各實作自行決定。 由於這個主題對確保支付傳遞過程的可靠性非常重要，我們在本書中包含了它。

對等節點（peer）

對等網路的參與者。在閃電網路中，對等節點透過 TCP 套接字、IP 或 Tor 上的加密認證通訊相互連接。

對等層（peer-to-peer layer）

對等層是閃電網路協定套件的第三層，運作在訊息層之上。 它負責定義透過閃電訊息在對等節點之間交換的資訊的語法和語義。 這包括 BOLT #9 中定義的控制訊息；BOLT #2 中定義的通道建立、操作和關閉訊息；以及 BOLT #7 中定義的八卦和路由訊息。

私有通道（private channel）

不向網路其餘部分宣告的通道。 技術上，「私有」是一個誤稱，因為這些通道仍然可以透過路由提示和承諾交易識別。 更好的描述是「未宣告」通道。 使用未宣告的通道，通道夥伴可以正常地在彼此之間發送和接收支付。 然而，網路的其餘部分不會知道該通道，因此通常無法使用它來路由支付。 由於未宣告通道的數量和容量是未知的，公共通道計數和容量總數只佔閃電網路總量的一部分。

原像（preimage）

在密碼學的上下文中，特別是在閃電網路中，原像指的是產生特定雜湊的雜湊函數的輸入。從雜湊計算原像是不可行的（雜湊函數只能單向運算）。透過選擇一個秘密隨機值作為原像並計算其雜湊，我們可以承諾該原像，之後再揭示它。任何人都可以確認揭示的原像正確產生該雜湊。

工作量證明（Proof of Work，PoW）

需要大量計算才能找到的資料，任何人都可以輕鬆驗證以證明產生它所需的工作量。 在比特幣中，礦工必須找到 SHA-256 演算法的數字解，以達到全網目標，稱為難度目標。 更多資訊參見 *比特幣挖礦*。

點時間鎖定合約（Point Time-Locked Contract，PTLC）

點時間鎖定合約（PTLC）是一種比特幣腳本，允許在呈現秘密或經過某個區塊高度後有條件地消費，類似於 HTLC。與 HTLC 不同，PTLC 不依賴雜湊函數的原像，而是依賴橢圓曲線點的私鑰。因此安全假設基於離散對數。PTLC 尚未在閃電網路上實作。

相對時間鎖（relative timelock）

相對時間鎖是一種時間鎖類型，允許輸入指定可以將輸入添加到區塊的最早時間。時間是相對的，基於該輸入引用的輸出何時被記錄在區塊中。相對時間鎖由 nSequence 交易欄位和 CHECKSEQUENCEVERIFY（CSV）比特幣腳本操作碼設定，由 BIP-68/112/113 引入。

可撤銷序列成熟度合約（Revocable Sequence Maturity Contract，RSMC）

該合約用於在兩個不需要相互信任的比特幣或閃電網路使用者之間構建支付通道。 名稱來自一系列編碼為承諾交易的狀態，如果被錯誤地發布並被比特幣網路挖礦，可以被撤銷。

撤銷金鑰（revocation key）

每個 RSMC 包含兩個撤銷金鑰。每個通道夥伴知道一個撤銷金鑰。知道兩個撤銷金鑰，RSMC 的輸出可以在預定義的時間鎖內被花費。在協商新的通道狀態時，舊的撤銷金鑰被分享，從而「撤銷」舊狀態。撤銷金鑰用於阻止通道夥伴廣播舊的通道狀態。

RIPEMD-160

RIPEMD-160 是一種產生 160 位元（20 位元組）雜湊的加密雜湊函數。

路由層（routing layer）

閃電網路協定套件的第四層，運作在對等層之上。 其職責是定義加密原語和必要的通訊協定，以允許比特幣從發送節點到接收節點的安全原子傳輸。 BOLT #4 定義了用於向沒有直接連接的遠端對等節點傳達傳輸資訊的洋蔥格式，而洋蔥的實際傳輸和加密原語在 BOLT #2 中定義。

拓撲（topology）

閃電網路的拓撲描述了閃電網路作為數學圖形的形狀。圖的節點是閃電節點（網路參與者/對等節點）。圖的邊是支付通道。 閃電網路的拓撲透過八卦協定公開廣播，但未宣告的通道除外。 這意味著閃電網路可能比宣告的通道和節點數量大得多。 了解拓撲對於支付的源路由過程特別重要，在該過程中發送者發現路由。

聰（satoshi）

聰是可以在區塊鏈上記錄的比特幣最小單位（面額）。一聰是一個比特幣的億分之一（0.00000001），以比特幣的創造者中本聰命名。

中本聰（Satoshi Nakamoto）

中本聰是設計比特幣並創建其原始參考實作的人或團體所使用的名字。作為實作的一部分，他們還設計了第一個區塊鏈資料庫。在這個過程中，他們首次解決了數位貨幣的雙重支付問題。他們的真實身份仍然未知。

Schnorr 簽章（Schnorr signature）

一種新的數位簽章方案，於 2021 年 11 月在比特幣中啟用。它在閃電網路上實現創新，例如高效的 PTLC（HTLC 的改進）。

腳本（script）、比特幣腳本（Bitcoin Script）

比特幣使用稱為比特幣腳本的腳本系統進行交易。類似於 Forth 程式語言，它是簡單的、基於堆疊的，從左到右處理。它故意不是圖靈完備的，沒有循環或遞歸。

ScriptPubKey（又名 pubkey script）

ScriptPubKey 或 pubkey script 是包含在輸出中的腳本，設定必須滿足才能花費這些輸出的條件。滿足條件的資料可以在簽名腳本中提供。另參見 *ScriptSig*。

ScriptSig（又名 signature script）

ScriptSig 或簽名腳本是由消費者生成的資料，幾乎總是用作變數來滿足 pubkey script。

秘密金鑰（secret key，又名私鑰）

解鎖發送到相應地址的比特幣的秘密數字。秘密 金鑰看起來像這樣：5J76sF8L5jTtzE96r66Sf8cka9y44wdpJjMwCxR3tzLh3ibVPxh。

隔離見證（Segregated Witness，SegWit）

隔離見證（SegWit）是 2017 年引入的比特幣協定升級，為簽名和其他交易授權證明添加了新的見證。這個新的見證欄位免於交易 ID 的計算，解決了大多數第三方交易可塑性問題。隔離見證作為軟分叉部署，是一個技術上使比特幣協定規則更加嚴格的變更。

安全雜湊演算法（Secure Hash Algorithm，SHA）

安全雜湊演算法或 SHA 是由美國國家標準與技術研究院（NIST）發布的一系列加密雜湊函數。比特幣協定目前使用 SHA-256，產生 256 位元雜湊。

短通道 ID（short channel ID，scid）

一旦通道建立，區塊鏈上資金交易的索引被用作短通道 ID 來唯一識別通道。 短通道 ID 由八個位元組組成，指向三個數字。 在其序列化形式中，它將這三個數字描繪為由字母「x」分隔的十進位值（例如 600123x01x00）。 第一個數字（4 位元組）是區塊高度。 第二個數字（2 位元組）是區塊內資金交易的索引。 最後一個數字（2 位元組）是交易輸出。

簡化支付驗證（simplified payment verification，SPV）

SPV 或簡化支付驗證是一種驗證特定交易是否包含在區塊中而無需下載整個區塊的方法。該方法被一些輕量級比特幣和閃電錢包使用。

源路由（source-based routing）

在閃電網路上，支付的發送者決定支付的路由。 雖然這降低了路由過程的成功率，但增加了支付的隱私性。 由於洋蔥路由使用的 SPHINX Mix Format，所有路由節點都不知道支付的發起者或最終接收者。 源路由與網際網路協定上的路由工作方式根本不同。

軟分叉（soft fork）

軟分叉或軟分叉變更是一種向前和向後相容的協定升級，因此它允許舊節點和新節點繼續使用同一條鏈。

SPHINX Mix Format

閃電網路中使用的一種特定洋蔥路由技術，由 George Danezis 和 Ian Goldberg 於 2009 年 (https://cypherpunks.ca/~iang/pubs/Sphinx_Oakland09.pdf) 發明。 使用 SPHINX Mix Format，洋蔥封包的每條訊息都用一些隨機資料填充，這樣沒有任何一跳可以估計它沿路由行進了多遠。 雖然發送者和接收者的隱私得到保護，每個節點仍然能夠沿路徑向訊息的發起者返回錯誤訊息。

潛艇交換（submarine swap）

潛艇交換是鏈上比特幣地址和鏈下閃電網路支付之間的無需信任的原子交換。就像閃電網路支付使用 HTLC 使資金的最終索取取決於接收者揭示秘密（雜湊原像），潛艇交換使用相同的機制以最小的信任跨越鏈上/鏈下障礙轉移資金。反向潛艇交換允許反方向的交換，從鏈下閃電網路支付到鏈上比特幣地址。

時間鎖（timelock）

時間鎖是一種限制某些比特幣在指定的未來時間或區塊高度之前不能花費的約束。時間鎖在許多比特幣合約中都很突出，包括支付通道和 HTLC。

交易（transaction）

交易是比特幣用來將比特幣從一個地址轉移到另一個地址的資料結構。 數千筆交易被聚合在一個區塊中，然後記錄（挖礦）在區塊鏈上。 每個區塊中的第一筆交易，稱為 coinbase 交易，生成新的比特幣。

交易可塑性（transaction malleability）

交易可塑性是指交易的雜湊可以在不改變交易語義的情況下改變的特性。 例如，更改簽名可以改變交易的雜湊。 承諾交易需要資金交易的雜湊，如果資金交易的雜湊改變，依賴它的交易將變得無效。這將使使用者無法索取退款（如果有的話）。 隔離見證軟分叉解決了這個問題，因此是支援閃電網路的重要升級。

傳輸層（transport layer）

在電腦網路中，傳輸層是電腦（最終是應用程式）相互通訊所使用方法的概念劃分。 傳輸層提供電腦之間的通訊服務，如流量控制、驗證和多工（允許多個應用程式同時在一台電腦上工作）。

未花費交易輸出（unspent transaction output，UTXO）

參見 *輸出*。

錢包（wallet）

錢包是一種持有比特幣私鑰的軟體。它用於創建和簽署比特幣交易。在閃電網路的上下文中，它還持有舊通道狀態的撤銷秘密和最新的預簽署承諾交易。

瞭望塔（watchtower）

瞭望塔是閃電網路上的安全服務，監控支付通道是否有潛在的協定違規。 如果其中一個通道夥伴離線或丟失備份，瞭望塔會保留備份並可以恢復其通道資訊。

瞭望塔還監控比特幣區塊鏈，如果其中一個夥伴試圖透過廣播過時狀態來「作弊」，可以提交懲罰交易。瞭望塔可以由通道夥伴自己運行，或作為第三方提供的付費服務。瞭望塔對通道中的資金本身沒有控制權。

一些貢獻的定義來自 Bitcoin Wiki (https://en.bitcoin.it/wiki/Main_Page)、https://en.wikipedia.org[Wikipedia]、https://github.com/bitcoinbook/bitcoinbook[*Mastering Bitcoin*] 或其他開源出版物，並在 CC-BY 授權下使用。

Appendix A: 比特幣基礎回顧

閃電網路能夠運行在多個區塊鏈之上，但主要錨定在比特幣上。要理解閃電網路，你需要對比特幣及其構建模組有基本的了解。

有許多好的資源可以用來學習更多關於比特幣的知識，包括 Andreas M. Antonopoulos 所著的配套書籍《精通比特幣》（第二版），你可以在 GitHub 上以開源授權找到它：https://github.com/bitcoinbook/bitcoinbook[]。然而，你不需要閱讀另一整本書來準備這本書！

在本章中，我們收集了你需要了解的關於比特幣最重要的概念，並在閃電網路的背景下進行解釋。這樣你可以準確地學習理解閃電網路所需的知識，而不會分心。

本章涵蓋了比特幣的幾個重要概念，包括：

- 金鑰和數位簽章
- 雜湊函數
- 比特幣交易及其結構
- 比特幣交易鏈接
- 交易輸出點
- 比特幣腳本：鎖定和解鎖腳本
- 基本鎖定腳本
- 複雜和條件式鎖定腳本
- 時間鎖

A.1. 金鑰和數位簽章

你可能聽說過比特幣是基於_密碼學_的，這是一個在電腦安全中廣泛使用的數學分支。密碼學還可以用來證明對秘密的知識而不揭露該秘密（數位簽章），或證明資料的真實性（數位指紋）。這些類型的密碼學證明是比特幣的關鍵數學工具，在比特幣應用中被廣泛使用。

比特幣的所有權是通過_數位金鑰_、_比特幣地址_和_數位簽章_建立的。數位金鑰實際上並不儲存在網路中，而是由使用者創建並儲存在一個稱為_錢包_的檔案或簡單資料庫中。使用者錢包中的數位金鑰完全獨立於比特幣協定，可以由使用者的錢包軟體生成和管理，無需參考區塊鏈或訪問網際網路。

大多數比特幣交易需要有效的數位簽章才能被包含在區塊鏈中，而這只能用秘密金鑰生成；因此，擁有該金鑰副本的任何人都可以控制比特幣。用於花費資金的數位簽章也被稱為_見證_，這是密碼學中使用的術語。比特幣交易中的見證資料證明了被花費資金的真正所有權。金鑰成

對出現，由私（秘密）鑰和公鑰組成。把公鑰想像成類似於銀行帳號，把私鑰想像成類似於秘密 PIN 碼。

A.1.1. 私鑰和公鑰

私鑰只是一個隨機選取的數字。在實踐中，為了便於管理許多金鑰，大多數比特幣錢包使用確定性衍生演算法從單一隨機_種子_生成一系列私鑰。簡單來說，單一隨機數字被用來產生一系列看似隨機的數字，這些數字被用作私鑰。這允許使用者只備份種子，就能從該種子_衍生_所有他們需要的金鑰。

比特幣和許多其他加密貨幣及區塊鏈一樣，使用_橢圓曲線_來提供安全性。在比特幣中，*secp256k1* 橢圓曲線上的橢圓曲線乘法被用作 *單向函數*。簡單來說，橢圓曲線數學的性質使得計算點的純量乘法變得簡單，但不可能計算反函數（除法或離散對數）。

每個私鑰都有一個對應的_公鑰_，通過在橢圓曲線上使用純量乘法從私鑰計算得出。簡單來說，有了私鑰 k，我們可以將它乘以常數 G 來產生公鑰 K：

- $K = k * G$

這個計算是不可逆的。給定公鑰 K，無法計算出私鑰 k。在橢圓曲線數學中無法進行除以 G 的運算。相反，必須在一個稱為_暴力攻擊_的窮舉過程中嘗試所有可能的 k 值。因為 k 是一個 256 位元的數字，用任何傳統電腦窮舉所有可能的值所需的時間和能量比這個宇宙中可用的還要多。

A.1.2. 雜湊

在比特幣和閃電網路中廣泛使用的另一個重要工具是_密碼學雜湊函數_，特別是 SHA-256 雜湊函數。

雜湊函數，也稱為_摘要函數_，是一種將任意長度的資料轉換為固定長度結果的函數，這個結果稱為_雜湊_、*摘要_或_指紋*（參見 SHA-256 密碼學雜湊演算法）。重要的是，雜湊函數是_單向_函數，這意味著你無法反轉它們並從指紋計算輸入資料。

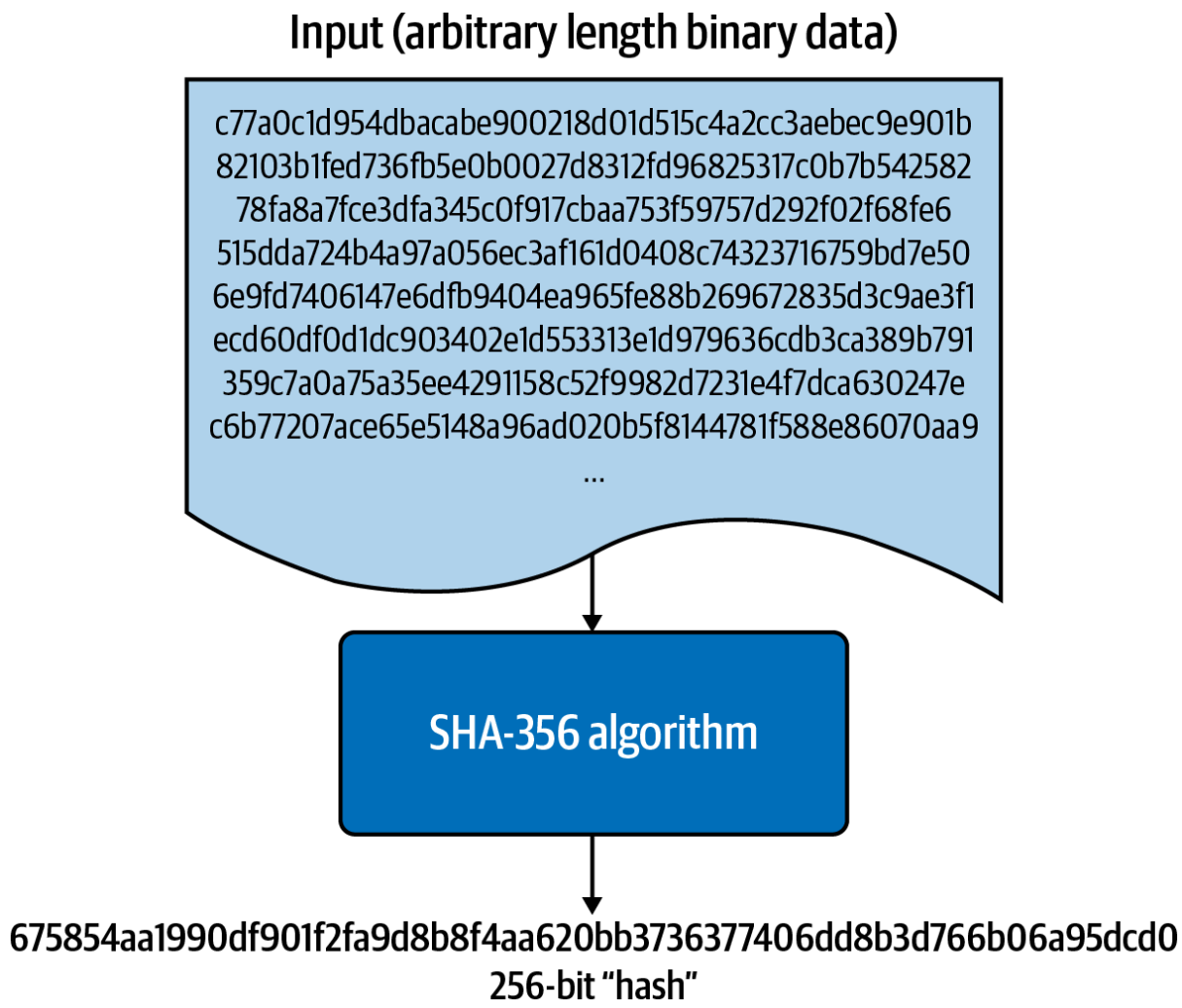

Figure 109. SHA-256 密碼學雜湊演算法

例如，如果我們使用命令列終端機將文字「Mastering the Lightning Network」輸入到 SHA-256 函數中，它將產生如下指紋：

```
$ echo -n "Mastering the Lightning Network" | shasum -a 256

ce86e4cd423d80d054b387aca23c02f5fc53b14be4f8d3ef14c089422b2235de  -
```

用於計算雜湊的輸入也稱為_原像_。

當然，輸入的長度可以更大得多。讓我們用中本聰的 比特幣白皮書 PDF 檔案 (https://bitcoin.org/bitcoin.pdf)嘗試同樣的事情：

```
$ wget http://bitcoin.org/bitcoin.pdf
$ cat bitcoin.pdf | shasum -a 256
b1674191a88ec5cdd733e4240a81803105dc412d6c6708d53ab94fc248f4f553  -
```

雖然比單一句子需要更長的時間，但 SHA-256 函數處理這個 9 頁的 PDF，將其「消化」成一個 256 位元的指紋。

現在你可能會想，一個可以消化無限大小資料的函數怎麼可能產生一個固定大小數字的唯一指紋？

理論上，由於原像（輸入）的數量是無限的，而指紋的數量是有限的，必然存在許多原像產生相同的 256 位元指紋。當兩個原像產生相同的雜湊時，這被稱為_碰撞_。

在實踐中，256 位元的數字是如此巨大，以至於你永遠無法故意找到碰撞。密碼學雜湊函數的工作基礎是，尋找碰撞是一種暴力努力，需要如此多的能量和時間，以至於實際上是不可能的。

密碼學雜湊函數因為具有一些有用的特性而被廣泛應用於各種應用中。它們是：

確定性

相同的輸入總是產生相同的雜湊。

不可逆性

不可能從雜湊計算原像。

防碰撞

在計算上不可行找到兩個具有相同雜湊的訊息。

不相關性

輸入的微小變化會在輸出中產生如此大的變化，以至於輸出看起來與輸入不相關。

均勻/隨機

密碼學雜湊函數產生的雜湊均勻分佈在整個 256 位元可能輸出空間中。雜湊的輸出看起來是隨機的，儘管它並非真正隨機。

使用密碼學雜湊的這些特性，我們可以構建一些有趣的 應用：

指紋

雜湊可用於對檔案或訊息進行指紋識別，以便唯一識別它。雜湊可用作任何資料集的通用識別碼。

完整性證明

檔案或訊息的指紋證明其完整性，因為檔案或訊息無法以任何方式被篡改或修改而不改變指紋。這通常用於確保軟體在安裝到你的電腦之前未被篡改。

承諾/不可否認性

你可以通過發布其雜湊來承諾特定的原像（例如，一個數字或訊息）而不揭露它。之後，你可以揭露秘密，每個人都可以驗證它與你之前承諾的是同一個東西，因為它產生了發布的雜湊。

工作量證明/雜湊研磨

你可以使用雜湊來證明你已經完成了計算工作，方法是在雜湊中顯示一個非隨機模式，這只能通過重複猜測原像來產生。例如，比特幣區塊頭的雜湊以許多零位元開始。產生它的唯一方法是更改標頭的一部分並將其雜湊數萬億次，直到它偶然產生該模式。

原子性

你可以將秘密原像作為在幾個連結交易中花費資金的先決條件。如果任何一方為了花費其中一個交易而揭露原像，則所有其他各方現在也可以花費他們的交易。所有或沒有一個變得可花費，在幾個交易中實現原子性。

A.1.3. 數位簽章

私鑰用於創建簽章，這些簽章是通過證明交易中使用的資金的所有權來花費比特幣所必需的。

_數位簽章_是通過將私鑰應用於特定訊息而計算出的數字。

給定訊息 m 和私鑰 k，簽名函數 F_{sign} 可以產生簽章 S：

$$\$S = F_{sign}(m, k)\$$$

這個簽章 S 可以由任何擁有公鑰 K（對應於私鑰 k）和訊息的人獨立驗證：

$$\$F_{verify}(m, K, S)\$$$

如果 F_{verify} 返回 true 結果，則驗證者可以確認訊息 m 是由擁有私鑰 k 的人簽署的。重要的是，數位簽章證明了在簽署時擁有私鑰 k，而不揭露 k。

數位簽章使用密碼學雜湊演算法。簽章應用於訊息的雜湊，因此訊息 m 被「摘要」成一個固定長度的雜湊 $H(m)$，作為指紋。

通過將數位簽章應用於交易的雜湊，簽章不僅證明了授權，還「鎖定」了交易資料，確保其完整性。已簽署的交易無法被修改，因為任何更改都會導致不同的雜湊並使簽章無效。

A.1.4. 簽章類型

簽章並不總是應用於整個交易。為了提供簽名彈性，比特幣數位簽章包含一個稱為簽章雜湊類型的前綴，它指定交易資料的哪一部分包含在雜湊中。這允許簽章承諾或「鎖定」交易中的全部或僅部分資料。最常見的簽章雜湊類型是 SIGHASH_ALL，它通過將所有交易資料包含在簽署的雜湊中來鎖定交易中的所有內容。相比之下，SIGHASH_SINGLE 鎖定所有交易輸入，但只鎖定一個輸出（更多關於輸入和輸出的內容在下一節）。不同的簽章雜湊類型可以組合產生六種不同的「模式」，用於被簽章鎖定的交易資料。

更多關於簽章雜湊類型的資訊可以在 《精通比特幣》第二版第 6 章的「簽章雜湊類型」一節 (https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch06.asciidoc#sighash_types)中找到。

A.2. 比特幣交易

_交易_是編碼比特幣系統中參與者之間價值轉移的資料結構。

A.2.1. 輸入和輸出

比特幣交易的基本構建模組是交易輸出。_交易輸出_是不可分割的比特幣貨幣塊，記錄在區塊鏈上，並被整個網路認可為有效。交易花費輸入並創建輸出。交易_輸入_只是對先前記錄交易的輸出的引用。這樣，每個交易都花費先前交易的輸出並創建新的輸出（參見 交易將價值從輸入轉移到輸出）。

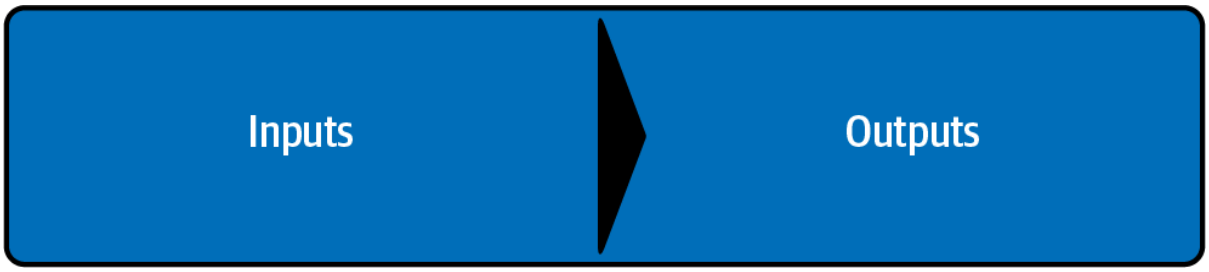

Figure 110. 交易將價值從輸入轉移到輸出

比特幣全節點追蹤所有可用和可花費的輸出，稱為_未花費交易輸出_（UTXOs）。所有 UTXOs 的集合稱為 UTXO 集，目前數量達數百萬個 UTXOs。當新的 UTXOs 被創建時，UTXO 集會增長；當 UTXOs 被消費時，它會縮小。每個交易都代表 UTXO 集的一個變化（狀態轉換），通過消費一個或多個 UTXOs 作為_交易輸入_，並創建一個或多個 UTXOs 作為其_交易輸出_。

例如，假設使用者 Alice 有一個她可以花費的 100,000 聰的 UTXO。Alice 可以通過構建一個交易來向 Bob 支付 100,000 聰，該交易有一個輸入（消費她現有的 100,000 聰輸入）和一個「支付」給 Bob 100,000 聰的輸出。現在 Bob 有了一個他可以花費的 100,000 聰 UTXO，創建一個消費這個新 UTXO 的新交易，並將其花費到另一個 UTXO 作為對另一個使用者的支付，依此類推（參見 Alice 向 Bob 支付 100,000 聰）。

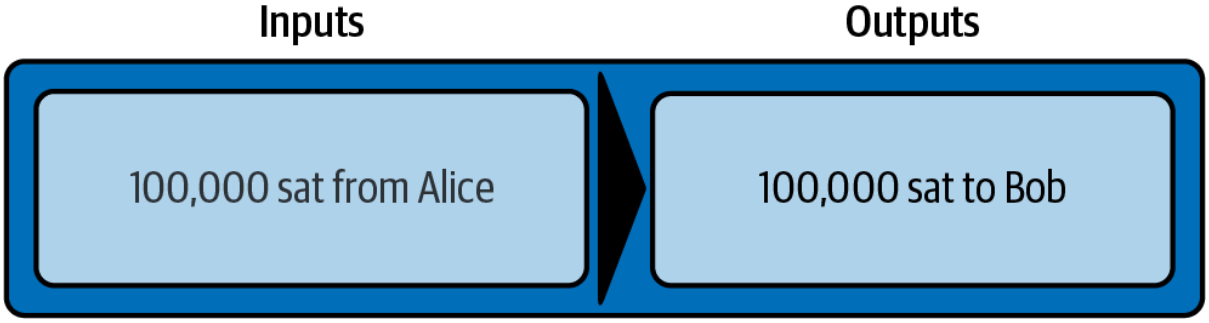

Figure 111. Alice 向 Bob 支付 100,000 聰

交易輸出可以有任意（整數）值，以聰計價。就像美元可以向下劃分到小數點後兩位為美分一樣，比特幣可以向下劃分到小數點後八位為聰。雖然輸出可以有任何任意值，但一旦創建，它就是不可分割的。這是輸出的一個重要特性，需要強調：輸出是離散的、不可分割的價值單

位，以整數聰計價。未花費的輸出只能在交易中完整消費。

那麼，如果 Alice 想向 Bob 支付 50,000 聰，但只有一個不可分割的 100,000 聰 UTXO 怎麼辦？Alice 需要創建一個交易，消費（作為其輸入）100,000 聰 UTXO，並有兩個輸出：一個支付 50,000 聰給 Bob，另一個支付 50,000 聰_回_給 Alice 作為「找零」（參見 Alice 向 Bob 支付 50,000 聰，向自己支付 50,000 聰作為找零）。

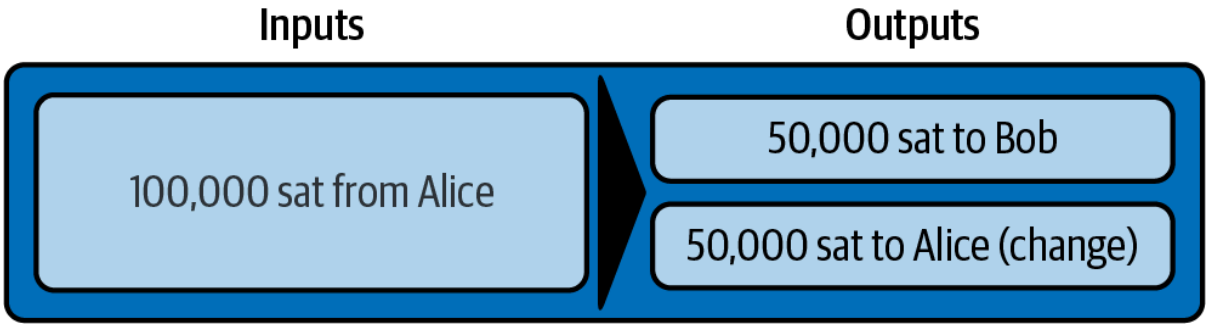

Figure 112. Alice 向 Bob 支付 50,000 聰，向自己支付 50,000 聰作為找零

找零輸出沒有什麼特別之處，也沒有辦法將其與任何其他輸出區分開來。它不必是最後一個輸出。可能有多個找零輸出，或沒有找零輸出。只有交易的創建者知道哪些輸出是給其他人的，哪些輸出是給他們自己擁有的地址的，因此是「找零」。

同樣，如果 Alice 想向 Bob 支付 85,000 聰，但有兩個可用的 50,000 聰 UTXOs，她必須創建一個具有兩個輸入（消費她的兩個 50,000 聰 UTXOs）和兩個輸出的交易，支付 Bob 85,000 聰並將 15,000 聰送回自己作為找零（參見 Alice 使用兩個 50,000 聰輸入向 Bob 支付 85,000 聰，向自己支付 15,000 聰作為找零）。

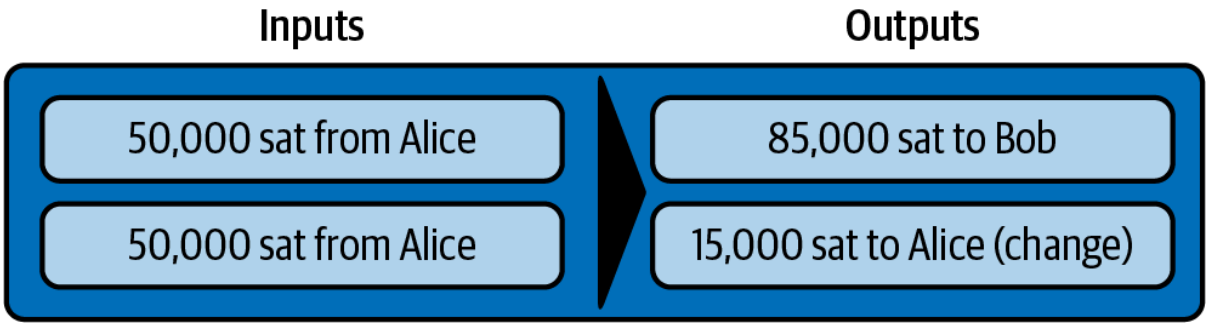

Figure 113. Alice 使用兩個 50,000 聰輸入向 Bob 支付 85,000 聰，向自己支付 15,000 聰作為找零

前面的圖解和範例展示了比特幣交易如何組合（花費）一個或多個輸入並創建一個或多個輸出。一個交易可以有數百甚至數千個輸入和輸出。

雖然閃電網路創建的交易有多個輸出，但它們本身沒有「找零」，因為通道的整個可用餘額在兩個通道夥伴之間分割。

A.2.2. 交易鏈

每個輸出都可以在後續交易中作為輸入花費。所以，例如，如果 Bob 決定花費 10,000 聰在一個向 Chan 支付的交易中，而 Chan 花費 4,000 聰向 Dina 支付，它會如 Alice 支付給 Bob，Bob 支付給 Chan，Chan 支付給 Dina 所示展開。

如果一個輸出在區塊鏈上記錄的另一個交易中被引用為輸入，則該輸出被視為_已花費_。如果沒有記錄的交易引用該輸出，則該輸出被視為_未花費_（並且可用於花費）。

唯一沒有輸入的交易類型是由比特幣礦工創建的特殊交易，稱為_幣基交易_。幣基交易只有輸出而沒有輸入，因為它通過挖礦創造新的比特幣。每個其他交易都花費一個或多個先前記錄的輸出作為其輸入。

由於交易是鏈接的，如果你隨機選擇一個交易，你可以沿著其任何一個輸入向後追蹤到創建它的先前交易。如果你繼續這樣做，你最終會到達比特幣最初被挖出的幣基交易。

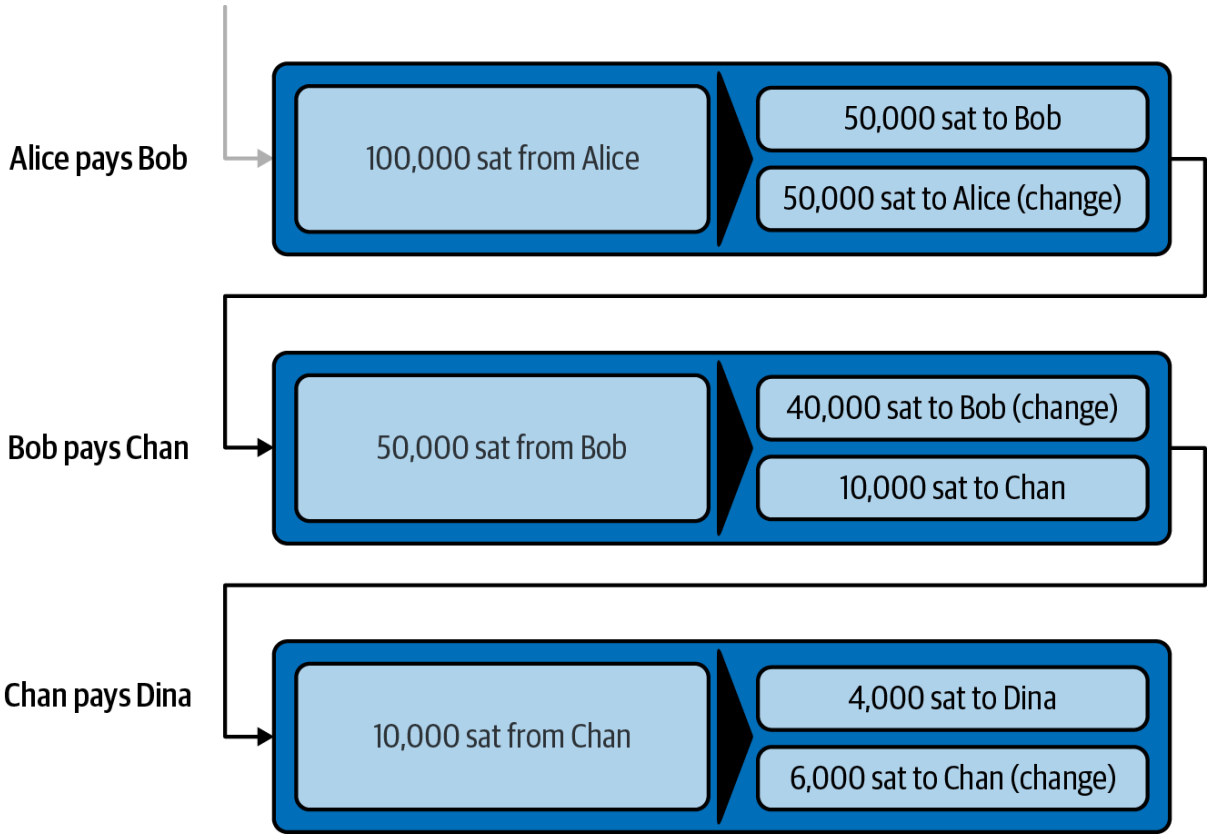

Figure 114. Alice 支付給 Bob，Bob 支付給 Chan，Chan 支付給 Dina

A.2.3. TxID：交易識別碼

比特幣系統中的每個交易都由唯一識別碼（假設存在 BIP-0030）識別，稱為_交易 ID_或簡稱 *TxID*。為了產生唯一識別碼，我們使用 SHA-256 密碼學雜湊函數產生交易資料的雜湊。這個「指紋」作為通用識別碼。交易可以通過其交易 ID 引用，一旦交易被記錄在比特幣區塊鏈上，比特幣網路中的每個節點都知道這個交易是有效的。

例如，交易 ID 可能看起來像這樣：

通過雜湊交易資料產生的交易 ID

```
e31e4e214c3f436937c74b8663b3ca58f7ad5b3fce7783eb84fd9a5ee5b9a54c
```

這是一個真實的交易（作為《精通比特幣》書籍的範例創建），可以在比特幣區塊鏈上找到。嘗試通過在區塊瀏覽器中輸入此 TxID 來找到它：

- *https://blockstream.info/tx/e31e4e214c3f436937c74b8663b3ca58f7ad5b3fce7783eb84fd9a5ee5b9a54c* (https://blockstream.info/tx/e31e4e214c3f436937c74b8663b3ca58f7ad5b3fce7783eb84fd9a5ee5b9a54c)

或使用短連結（區分大小寫）：

- *http://bit.ly/AliceTx* (http://bit.ly/AliceTx)

A.2.4. 輸出點：輸出識別碼

因為每個交易都有唯一 ID，我們還可以通過引用 TxID 和輸出索引號來唯一識別該交易中的交易輸出。交易中的第一個輸出是輸出索引 0，第二個輸出是輸出索引 1，依此類推。輸出識別碼通常稱為_輸出點_。

按照慣例，我們將輸出點寫為 TxID、冒號和輸出索引號：

輸出點：通過 TxID 和索引號識別輸出

```
7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18:0
```

輸出識別碼（輸出點）是將交易鏈接在一起的機制。每個交易輸入都是對先前交易的特定輸出的引用。該引用是一個輸出點：TxID 和輸出索引號。所以交易「花費」來自特定交易（通過 TxID）的特定輸出（通過索引號），以創建新的輸出，這些輸出本身可以通過引用輸出點來花費。

交易輸入引用輸出點形成鏈 展示了從 Alice 到 Bob 到 Chan 到 Dina 的交易鏈，這次在每個輸入中都有輸出點。

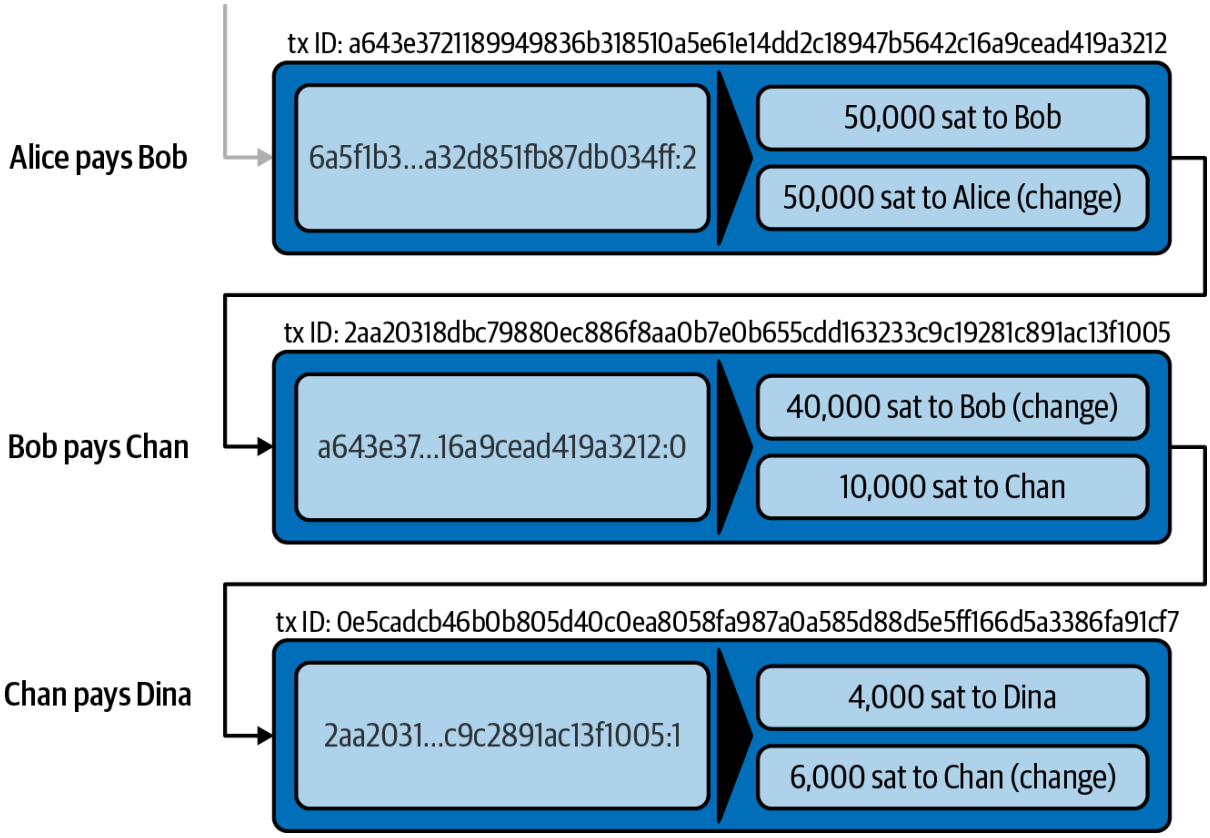

Figure 115. 交易輸入引用輸出點形成鏈

Bob 交易中的輸入引用 Alice 的交易（通過 TxID）和索引為 0 的輸出。

Chan 交易中的輸入引用 Bob 交易的 TxID 和第一個索引輸出，因為向 Chan 的支付是輸出 #1。在 Bob 向 Chan 的支付中，Bob 的找零是輸出 #0。[12]

現在，如果我們看 Alice 向 Bob 的支付，我們可以看到 Alice 正在花費一個輸出點，該輸出點是交易 ID 為 6a5f1b3[…] 的交易中的第三個（輸出索引 #2）輸出。我們在圖中沒有看到那個被引用的交易，但我們可以從輸出點推斷這些細節。

A.3. 比特幣腳本

完成我們理解所需的比特幣最後一個元素是控制輸出點訪問的腳本語言。到目前為止，我們通過說「Alice 簽署交易以向 Bob 支付」來簡化描述。然而，在幕後，有一些隱藏的複雜性使得實現更複雜的花費條件成為可能。最簡單和最常見的花費條件是「出示與以下公鑰匹配的簽章」。這樣的花費條件作為_鎖定腳本_記錄在每個輸出中，使用稱為_比特幣腳本_的腳本語言編寫。

比特幣腳本是一種極其簡單的基於堆疊的腳本語言。它不包含迴圈或遞迴，因此是_圖靈不完備_的（意味著它無法表達任意複雜性並且具有可預測的執行）。熟悉（現在已經古老的）FORTH 程式語言的人會認識其語法和風格。

A.3.1. 運行比特幣腳本

簡單來說，比特幣系統通過在堆疊上運行腳本來評估比特幣腳本；如果最終結果是 TRUE，它認為花費條件滿足，交易有效。

讓我們看一個非常簡單的比特幣腳本範例，它將數字 2 和 3 相加，然後將結果與數字 5 進行比較：

```
2 3 ADD 5 EQUAL
```

在 比特幣腳本執行範例 中，我們看到這個腳本是如何執行的（從左到右）。

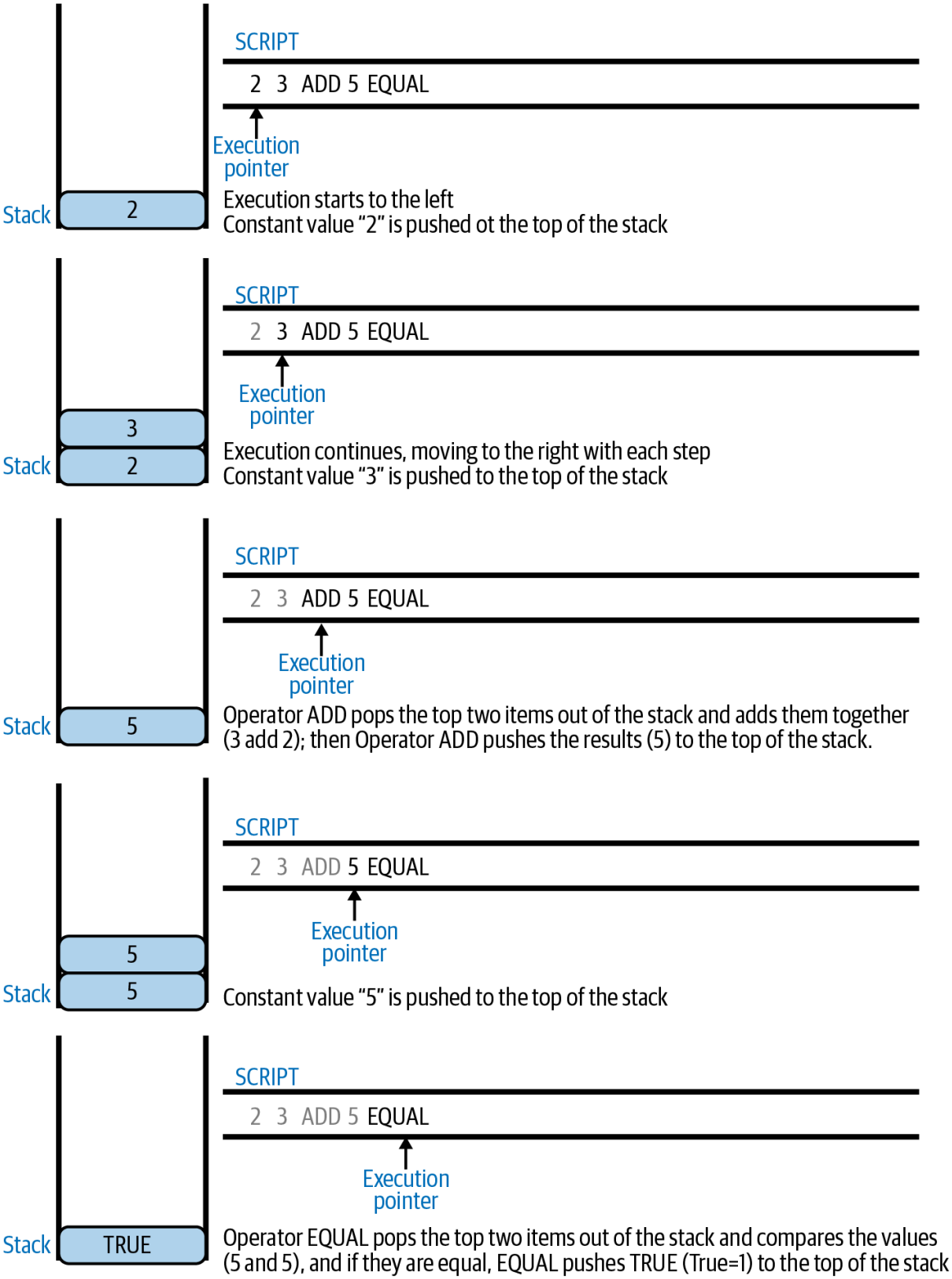

Figure 116. 比特幣腳本執行範例

A.3.2. 鎖定和解鎖腳本

比特幣腳本由兩部分組成：

鎖定腳本

這些嵌入在交易輸出中，設定花費該輸出必須滿足的條件。例如，Alice 的錢包在支付給 Bob 的輸出中添加一個鎖定腳本，設定需要 Bob 的簽章才能花費它的條件。

解鎖腳本

這些嵌入在交易輸入中，滿足被引用輸出的鎖定腳本設定的條件。例如，Bob 可以通過提供包含數位簽章的解鎖腳本來解鎖前面的輸出。

使用簡化模型，為了驗證，解鎖腳本和鎖定腳本被連接並執行（P2SH 和 SegWit 是例外）。例如，如果某人用鎖定腳本 "3 ADD 5 EQUAL" 鎖定了一個交易輸出，我們可以在交易輸入中用解鎖腳本 "2" 來花費它。任何驗證該交易的人都會連接我們的解鎖腳本（2）和鎖定腳本（3 ADD 5 EQUAL）並通過比特幣腳本執行引擎運行結果。他們會得到 TRUE，我們就能夠花費該輸出。

顯然，這個簡化的範例對於鎖定實際的比特幣輸出來說是一個非常糟糕的選擇，因為沒有秘密，只有基本算術。任何人都可以通過提供答案「2」來花費該輸出。因此，大多數鎖定腳本需要證明對秘密的知識。

A.3.3. 鎖定到公鑰（簽章）

最簡單的鎖定腳本形式是需要簽章的腳本。讓我們考慮 Alice 向 Bob 支付 50,000 聰的交易。Alice 創建的向 Bob 支付的輸出將有一個需要 Bob 簽章的鎖定腳本，看起來像這樣：

需要來自 Bob 私鑰的數位簽章的鎖定腳本

```
<Bob Public Key> CHECKSIG
```

運算子 `CHECKSIG` 從堆疊中取出兩個項目：簽章和公鑰。如你所見，Bob 的公鑰在鎖定腳本中，所以缺少的是與該公鑰對應的簽章。這個鎖定腳本只能由 Bob 花費，因為只有 Bob 有相應的私鑰來產生與公鑰匹配的數位簽章。

為了解鎖這個鎖定腳本，Bob 會提供一個只包含他的數位簽章的解鎖腳本：

包含（僅）來自 Bob 私鑰的數位簽章的解鎖腳本

```
<Bob Signature>
```

在 顯示鎖定腳本（輸出）和解鎖腳本（輸入）的交易鏈 中，你可以看到 Alice 交易中的鎖定腳本（在支付給 Bob 的輸出中）和 Bob 交易中的解鎖腳本（在花費該輸出的輸入中）。

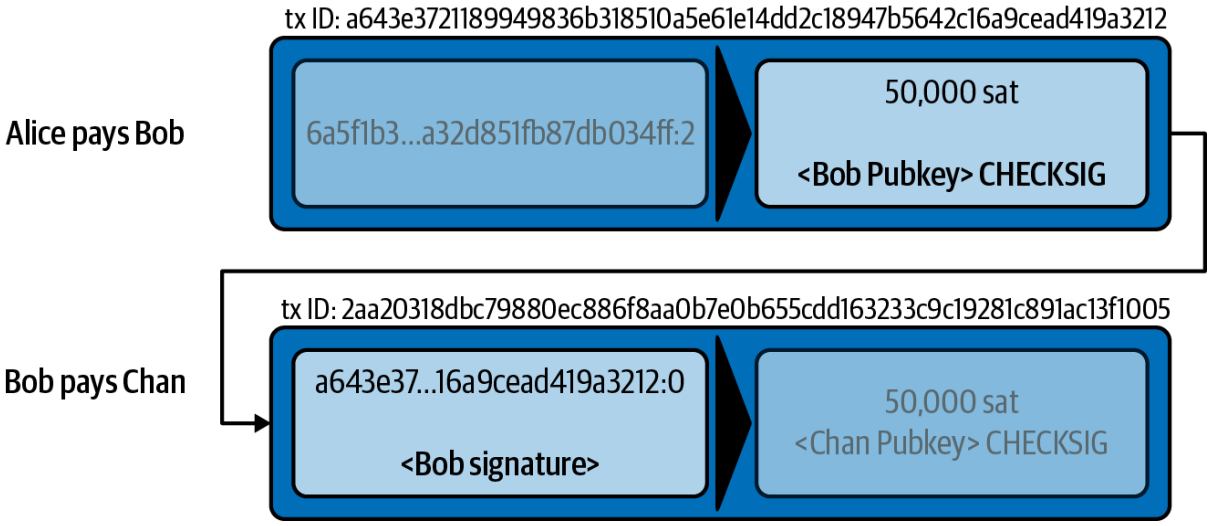

Figure 117. 顯示鎖定腳本（輸出）和解鎖腳本（輸入）的交易鏈

為了驗證 Bob 的交易，比特幣節點會執行以下操作：

1. 從輸入中提取解鎖腳本（<Bob Signature>）。
2. 查找它試圖花費的輸出點（a643e37...3213:0）。這是 Alice 的交易，會在區塊鏈上找到。
3. 從該輸出點提取鎖定腳本（<Bob PubKey> CHECKSIG）。
4. 連接成一個腳本，將解鎖腳本放在鎖定腳本前面（<Bob Signature> <Bob PubKey> CHECKSIG）。
5. 在比特幣腳本執行引擎上執行此腳本，看產生什麼結果。
6. 如果結果是 TRUE，則推斷 Bob 的交易是有效的，因為它能夠滿足花費該輸出點的花費條件。

A.3.4. 鎖定到雜湊（秘密）

另一種類型的鎖定腳本，在閃電網路中使用，是_雜湊鎖_。要解鎖它，你必須知道雜湊的秘密_原像_。

為了演示這一點，讓 Bob 生成一個隨機數 R 並保密：

```
R = 1833462189
```

現在，Bob 計算這個數字的 SHA-256 雜湊：

```
H = SHA256(R) =>
H = SHA256(1833462189) =>
H = 0ffd8bea4abdb0deafd6f2a8ad7941c13256a19248a7b0612407379e1460036a
```

現在，Bob 把我們之前計算的雜湊 H 給 Alice，但保留數字 R 秘密。回想一下，由於密碼學雜湊的特性，Alice 無法「反轉」雜湊計算並猜測數字 R。

Alice 創建一個支付 50,000 聰的輸出，鎖定腳本為：

```
HASH256 H EQUAL
```

其中 H 是 Bob 給 Alice 的實際雜湊值（0ffd8...036a）。

讓我們解釋這個腳本：

HASH256 運算子從堆疊中彈出一個值並計算該值的 SHA-256 雜湊。然後它將結果推回堆疊。

H 值被推到堆疊上，然後 EQUAL 運算子檢查兩個值是否相同，並相應地將 TRUE 或 FALSE 推到堆疊上。

因此，這個鎖定腳本只有在與包含 R 的解鎖腳本結合時才會起作用，這樣當連接時，我們有：

```
R HASH256 H EQUAL
```

只有 Bob 知道 R，所以只有 Bob 能夠產生一個揭示秘密值 R 的解鎖腳本的交易。

有趣的是，Bob 可以把 R 值給其他任何人，然後他們就可以花費那個比特幣。這使得秘密值 R 幾乎像比特幣「代金券」，因為任何擁有它的人都可以花費 Alice 創建的輸出。我們將看到這是閃電網路中一個有用的特性！

A.3.5. 多重簽章腳本

比特幣腳本語言提供了多重簽章構建模組（原語），可用於構建託管服務和多個利益相關者之間的複雜所有權配置。需要多個簽章才能花費比特幣的安排稱為_多重簽章方案_，進一步指定為 *K-of-N* 方案，其中：

- *N* 是多重簽章方案中識別的簽名者總數，以及
- *K* 是_法定人數_或_閾值_：授權花費的最小簽章數量。

K-of-*N* 多重簽章的腳本是：

```
K <PubKey1> <PubKey2> ... <PubKeyN> N CHECKMULTISIG
```

其中 *N* 是列出的公鑰總數（公鑰 1 到公鑰 *N*），*K* 是花費輸出所需簽章的閾值。

閃電網路使用 2-of-2 多重簽章方案來構建支付通道。例如，Alice 和 Bob 之間的支付通道將建立在這樣的 2-of-2 多重簽章上：

```
2 <PubKey Alice> <PubKey Bob> 2 CHECKMULTISIG
```

前面的鎖定腳本可以用包含一對簽章的解鎖腳本來滿足：[13]中有描述。]

```
0 <Sig Alice> <Sig Bob>
```

兩個腳本組合在一起將形成組合驗證腳本：

```
0 <Sig Alice> <Sig Bob> 2 <PubKey Alice> <PubKey Bob> 2 CHECKMULTISIG
```

多重簽章鎖定腳本可以由比特幣地址表示，該地址編碼鎖定腳本的雜湊。例如，閃電支付通道的初始資金交易是向一個地址支付的交易，該地址編碼了兩個通道夥伴的 2-of-2 多重簽章的鎖定腳本。

A.3.6. 時間鎖腳本

比特幣中存在並在閃電網路中廣泛使用的另一個重要構建模組是_時間鎖_。時間鎖是對花費的限制，要求在允許花費之前必須經過一定的時間或區塊高度。它有點像一張從銀行帳戶開出的遠期支票，在支票上的日期之前無法兌現。

比特幣有兩個級別的時間鎖：交易級時間鎖和輸出級時間鎖。

_交易級時間鎖_記錄在交易的 `nLockTime` 欄位中，防止整個交易在時間鎖過去之前被接受。交易級時間鎖是當今比特幣中最常用的時間鎖機制。

_輸出級時間鎖_由腳本運算子創建。有兩種類型的輸出時間鎖：絕對時間鎖和相對時間鎖。

輸出級_絕對時間鎖_由運算子 CHECKLOCKTIMEVERIFY 實作，在對話中通常縮寫為 *CLTV*。絕對時間鎖使用絕對時間戳或區塊高度實現時間約束，表達相當於「在區塊 800,000 之前不可花費」。

輸出級_相對時間鎖_由運算子 CHECKSEQUENCEVERIFY 實作，在對話中通常縮寫為 *CSV*。相對時間鎖實現相對於交易確認的花費約束，表達相當於「在確認後 1,024 個區塊之前無法花費」。

A.3.7. 具有多個條件的腳本

比特幣腳本更強大的功能之一是流程控制，也稱為條件子句。你可能熟悉各種程式語言中使用 IF...THEN...ELSE 結構的流程控制。比特幣條件子句看起來有點不同，但本質上是相同的結構。

在基本層面上，比特幣條件操作碼允許我們構建一個鎖定腳本，該腳本有兩種解鎖方式，取決於評估邏輯條件的 TRUE/FALSE 結果。例如，如果 x 是 TRUE，鎖定腳本是 A ELSE 鎖定腳本是 B。

此外，比特幣條件表達式可以無限_嵌套_，意味著一個條件子句可以包含另一個，另一個又包含另一個，等等。比特幣腳本流程控制可用於構建非常複雜的腳本，具有數百甚至數千個可能的執行路徑。嵌套沒有限制，但共識規則對腳本的最大位元組大小施加了限制。

比特幣使用 IF、ELSE、ENDIF 和 NOTIF 操作碼實作流程控制。此外，條件表達式可以包含布林運算子，如 BOOLAND、`BOOLOR` 和 NOT。

乍一看，你可能會發現比特幣的流程控制腳本令人困惑。那是因為比特幣腳本是一種堆疊語言。與算術運算 $1 + 1$ 在比特幣腳本中表示為 1 1 ADD 看起來「向後」一樣，比特幣中的流程控制子句也看起來「向後」。

在大多數傳統（程序式）程式語言中，流程控制看起來像這樣：

大多數程式語言中流程控制的虛擬碼

```
if (condition):
  code to run when condition is true
else:
  code to run when condition is false
code to run in either case
```

在像比特幣腳本這樣的基於堆疊的語言中，邏輯條件在 IF *之前*，這使它看起來「向後」，像這樣：

比特幣腳本流程控制

```
condition
IF
  code to run when condition is true
ELSE
  code to run when condition is false
ENDIF
code to run in either case
```

閱讀比特幣腳本時，請記住被評估的條件在 IF 操作碼_之前_。

A.3.8. 在腳本中使用流程控制

比特幣腳本中流程控制的一個非常常見的用途是構建一個鎖定腳本，該腳本提供多個執行路徑，每個路徑都是贖回 UTXO 的不同方式。

讓我們看一個簡單的範例，我們有兩個簽名者 Alice 和 Bob，其中任何一個都能夠贖回。使用多重簽章，這將表達為 1-of-2 多重簽章腳本。為了演示，我們將使用 IF 子句做同樣的事情：

```
IF
 <Alice's Pubkey> CHECKSIG
ELSE
 <Bob's Pubkey> CHECKSIG
ENDIF
```

看看這個鎖定腳本，你可能會想：「條件在哪裡？IF 子句前面什麼都沒有！」

條件不是鎖定腳本的一部分。相反，條件將_在解鎖腳本中提供_，允許 Alice 和 Bob「選擇」他們想要的執行路徑。

Alice 用這個解鎖腳本贖回：

```
<Alice's Sig> 1
```

末尾的 1 作為條件（TRUE），將使 IF 子句執行 Alice 有簽章的第一個贖回路徑。

對於 Bob 來說，要贖回這個，他必須通過給 IF 子句一個 FALSE 值來選擇第二個執行路徑：

```
<Bob's Sig> 0
```

Bob 的解鎖腳本在堆疊上放一個 0，導致 IF 子句執行第二個（ELSE）腳本，這需要 Bob 的簽章。

因為兩個條件中的每一個也需要簽章，Alice 不能使用第二個子句，Bob 不能使用第一個子句；他們沒有所需的簽章！

由於條件流可以嵌套，解鎖腳本中的 TRUE / FALSE 值也可以嵌套，以導航複雜的條件路徑。

在 閃電網路中使用的複雜腳本 中，你可以看到閃電網路中使用的那種複雜腳本的範例，具有多個條件。[14]閃電網路中使用的腳本經過高度優化和緊湊，以最小化鏈上足跡，所以它們不容易閱讀和理解。儘管如此，看看你是否能識別我們在本章中學到的一些比特幣腳本概念。

Example 9. 閃電網路中使用的複雜腳本

```
# 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
```

Appendix B: Docker 基本安裝和使用

本書包含許多在 Docker 容器中運行的範例，以便在不同作業系統之間實現標準化。

本節將幫助你安裝 Docker 並熟悉一些最常用的 Docker 命令，以便你可以運行本書的範例容器。

B.1. 安裝 Docker

在我們開始之前，你應該在電腦上安裝 Docker 容器系統。Docker 是一個開放系統，作為_社群版_免費分發給許多不同的作業系統，包括 Windows、macOS 和 Linux。Windows 和 Macintosh 版本稱為 *Docker Desktop*，由 GUI 桌面應用程式和命令列工具組成。Linux 版本稱為 *Docker Engine*，由伺服器守護程式和命令列工具組成。我們將使用命令列工具，這些工具在所有平台上都是相同的。

請按照 Docker 網站 (https://docs.docker.com/get-docker)的「Get Docker」說明為你的作業系統安裝 Docker。

從列表中選擇你的作業系統並按照安裝說明進行操作。

> 如果你在 Linux 上安裝，請遵循安裝後的說明，以確保你可以以普通使用者而非 root 使用者身分運行 Docker。否則，你需要在所有 docker 命令前加上 sudo，以 root 身分運行它們，例如：sudo docker。

安裝 Docker 後，你可以通過運行演示容器 hello-world 來測試你的安裝，如下所示：

```
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

[...]
```

B.2. 基本 Docker 命令

在本附錄中，我們會經常使用 Docker。我們將使用以下 Docker 命令和參數。

B.2.1. 構建容器

`docker build [-t tag] [directory]`
__tag__ 是我們如何識別我們正在構建的容器，__directory__ 是容器的上下文（資料夾和檔案）和定義檔案（Dockerfile）所在的位置。

B.2.2. 運行容器

`docker run -it [--network netname] [--name cname] tag`

__netname__ 是 Docker 網路的名稱，__cname__ 是我們為這個容器實例選擇的名稱，__tag__ 是我們構建容器時給它的名稱標籤。

B.2.3. 在容器中執行命令

`docker exec` *`cname command`*
__cname__ 是我們在 run 命令中給容器的名稱，__command__ 是我們想要在容器內運行的可執行檔案或腳本。

B.2.4. 停止和啟動容器

在大多數情況下，如果我們在_互動式_和_終端機_模式下運行容器，即設定了 i 和 t 標誌（組合為 -it），可以通過按 Ctrl-C 或使用 exit 或 Ctrl-D 退出 shell 來停止容器。如果容器沒有終止，你可以從另一個終端機這樣停止它：

`docker stop` *`cname`*
要恢復一個已經存在的容器，使用 `start` 命令如下：

`docker start` *`cname`*

B.2.5. 按名稱刪除容器

如果你給容器命名而不是讓 Docker 隨機命名，你不能重複使用該名稱，直到容器被刪除。Docker 會返回這樣的錯誤：

```
docker: Error response from daemon: Conflict. The container name "/bitcoind" is
already in use...
```

要解決這個問題，刪除容器的現有實例：

`docker rm` *`cname`*
__cname__ 是分配給容器的名稱（錯誤訊息範例中的 bitcoind）。

B.2.6. 列出運行中的容器

```
docker ps
```

此命令顯示當前運行的容器及其名稱。

B.2.7. 列出 Docker 映像

```
docker image ls
```

此命令顯示已在你的電腦上構建或下載的 Docker 映像。

B.3. 結論

這些基本的 Docker 命令將足以讓你入門，並允許你運行本書中的所有範例。

Appendix C: 線路協定訊息

本附錄列出了閃電 P2P 協定中所有目前定義的訊息類型。此外，我們展示了每個訊息的結構，根據協定流程將訊息分組為邏輯分類。

閃電協定訊息是可擴展的，其結構可能在全網升級期間發生變化。有關權威資訊，請參閱 GitHub Lightning-RFC 儲存庫 (https://github.com/lightningnetwork/lightning-rfc)中的最新版 BOLT。

C.1. 訊息類型

目前定義的訊息類型列在 訊息類型 中。

Table 10. 訊息類型

類型整數	訊息名稱	類別
16	`init`	連線建立
17	`error`	錯誤通訊
18	`ping`	連線活躍性
19	`pong`	連線活躍性
32	`open_channel`	通道資金
33	`accept_channel`	通道資金
34	`funding_created`	通道資金
35	`funding_signed`	通道資金
36	`funding_locked`	通道資金 + 通道操作
38	`shutdown`	通道關閉
39	`closing_signed`	通道關閉
128	`update_add_htlc`	通道操作
130	`update_fulfill_hltc`	通道操作
131	`update_fail_htlc`	通道操作
132	`commit_sig`	通道操作
133	`revoke_and_ack`	通道操作
134	`update_fee`	通道操作
135	`update_fail_malformed_htlc`	通道操作
136	`channel_reestablish`	通道操作
256	`channel_announcement`	通道公告
257	`node_announcement`	通道公告
258	`channel_update`	通道公告

類型整數	訊息名稱	類別
259	`announce_signatures`	通道公告
261	`query_short_chan_ids`	通道圖同步
262	`reply_short_chan_ids_end`	通道圖同步
263	`query_channel_range`	通道圖同步
264	`reply_channel_range`	通道圖同步
265	`gossip_timestamp_range`	通道圖同步

在 高層訊息類型 中，`Category`（類別）欄位允許我們根據訊息在協定本身中的功能快速對訊息進行分類。在高層次上，我們將訊息放入八個（非窮盡的）類別之一，包括：

連線建立

在點對點連線首次建立時發送。也用於協商新連線支援的功能集。

錯誤通訊

由節點用於相互通訊協定層級錯誤的發生。

連線活躍性

由節點用於檢查給定的傳輸連線是否仍然活躍。

通道資金

由節點用於創建新的支付通道。此過程也稱為通道資金過程。

通道操作

鏈下更新給定通道的行為。這包括發送和接收付款，以及在網路內轉發付款。

通道公告

向更廣泛的網路公告新的公開通道的過程，以便它可以用於路由目的。

通道圖同步

下載和驗證通道圖的過程。

請注意，屬於同一類別的訊息通常也共享相鄰的_訊息類型_。這是故意這樣做的，以便在規範本身中將語義相似的訊息分組在一起。

C.2. 訊息結構

我們現在詳細說明每個訊息類別，以定義閃電網路協定中所有已定義訊息的精確結構和語義。

C.2.1. 連線建立訊息

此類別中的訊息是節點在建立傳輸連線後發送的第一個訊息。在撰寫本章時，此類別中只存在一個訊息，即 `init` 訊息。`init` 訊息由連線的_兩端_在首次建立後發送。在兩方都發送 `init` 訊息之前，不應發送其他訊息。

init 訊息

`init` 訊息的結構定義如下：

- 類型: 16
- 欄位:
  - `uint16` : `global_features_len`
  - `global_features_len*byte` : `global_features`
  - `uint16` : `features_len`
  - `features_len*byte` : `features`
  - `tlv_stream_tlvs`

從結構上看，`init` 訊息由兩個可變大小的位元組切片組成，每個切片儲存一組_功能位元_。正如我們在 功能位元和協定可擴展性 中看到的，功能位元是協定中用於廣告節點理解（可選功能）或要求（必需功能）的協定功能集的原語。

請注意，現代節點實作只會使用 `features` 欄位，`global_features` 向量中的項目主要是出於_歷史_目的（向後相容）。

核心訊息之後是一系列類型-長度-值（TLV）記錄，可用於以向前和向後相容的方式在未來擴展訊息。我們將在本附錄稍後介紹什麼是 TLV 記錄以及它們是如何使用的。

`init` 訊息然後由節點檢查，以確定連線是否根據雙方廣告的可選和必需功能位元集定義良好。

可選功能意味著節點知道某個功能，但他們不認為它對新連線的操作至關重要。一個例子是理解現有訊息新添加欄位語義的能力。

另一方面，必需功能表示如果對方不知道該功能，則連線定義不明確。這種功能的一個例子是協定中理論上的新通道類型：如果你的對等節點不知道這個功能，那麼你不想保持連線，因為他們無法開啟你新的首選通道類型。

C.2.2. 錯誤通訊訊息

此類別中的訊息用於在兩個節點之間發送連線層級的錯誤。協定中存在另一種類型的錯誤：HTLC 轉發層級錯誤。連線層級錯誤可能表示功能位元不相容或意圖_強制關閉_（單方面廣播最新簽署的承諾）等情況。

error 訊息

此類別中唯一的訊息是 `error` 訊息。

- 類型: 17
- 欄位:
  - `channel_id` : `chan_id`
  - `uint16` : `data_len`
  - `data_len*byte` : `data`

`error` 訊息可以在特定通道的範圍內發送，方法是將 `channel_id` 設定為正在經歷此新錯誤狀態的通道的 `channel_id`。或者，如果錯誤適用於整個連線，則 `channel_id` 欄位應設定為全零。這個全零的 `channel_id` 也稱為錯誤的連線層級識別碼。

根據錯誤的性質，向你有通道的節點發送 `error` 訊息可能表示通道無法在沒有人工干預的情況下繼續，因此此時唯一的選項是通過廣播通道的最新承諾狀態來強制關閉通道。

C.2.3. 連線活躍性

本節中的訊息用於探測以確定連線是否仍然活躍。因為閃電網路協定在某種程度上抽象了用於傳輸訊息的底層傳輸，所以定義了一組協定層級的 `ping` 和 `pong` 訊息。

ping 訊息

`ping` 訊息用於檢查連線中的另一方是否「活著」。它包含以下欄位：

- 類型: 18
- 欄位:
  - `uint16` : `num_pong_bytes`
  - `uint16` : `ping_body_len`
  - `ping_body_len*bytes` : `ping_body`

接下來是它的伴侶，`pong` 訊息。

pong 訊息

pong 訊息作為對 ping 訊息的回應發送，包含以下欄位：

- 類型: 19

- 欄位:
  - `uint16`：`pong_body_len`
  - `ping_body_len*bytes`：`pong_body`

`ping` 訊息可以由任何一方在任何時候發送。

`ping` 訊息包含一個 `num_pong_bytes` 欄位，用於指示接收節點在其 `pong` 訊息中發送的有效載荷應該有多大。`ping` 訊息還包含一個 `ping_body` 不透明位元組集，可以安全地忽略。它只是允許發送者填充他們發送的 `ping` 訊息，這在嘗試基於線上封包大小阻止某些去匿名化技術時可能很有用。

`pong` 訊息應該作為對收到的 `ping` 訊息的回應發送。接收者應該讀取一組 `num_pong_bytes` 隨機位元組作為 `pong_body` 欄位發送回去。巧妙使用這些欄位/訊息可能允許注重隱私的路由節點嘗試阻止某些類別的網路去匿名化嘗試，因為他們可以創建一個基於跨線發送的封包大小類似於其他訊息的「假」轉錄。請記住，閃電網路預設使用_加密_傳輸，因此被動網路監視器無法讀取明文位元組，因此只能依賴計時和封包大小。

C.2.4. 通道資金

隨著我們繼續，我們進入了管理閃電協定功能和語義的核心訊息領域。在本節中，我們探討在創建新通道過程中發送的訊息。我們只會描述使用的欄位，因為我們將對資金過程的深入分析留給 支付通道。

在通道資金流程中發送的訊息屬於以下五個訊息的集合：`open_channel`、`accept_channel`、`funding_created`、`funding_signed` 和 `funding_locked`。

使用這些訊息的詳細協定流程在 支付通道 中描述。

open_channel 訊息

open_channel 訊息開始通道資金過程，包含以下欄位：

- 類型: 32
- 欄位:
  - `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`

這是節點希望與另一個節點執行新資金流程時發送的第一個訊息。此訊息包含雙方構建資金交易和承諾交易所需的所有必要資訊。

在撰寫本章時，在可以附加到已定義訊息末尾的可選 TLV 記錄集中定義了一個 TLV 記錄：

- 類型: 0
- 資料: `upfront_shutdown_script`

`upfront_shutdown_script` 是一個可變大小的位元組切片，必須是比特幣網路共識演算法接受的有效公鑰腳本。通過提供這樣的地址，發送方能夠有效地為他們的通道創建一個「封閉迴路」，因為雙方都不會簽署支付到任何其他地址的協作關閉交易。在實踐中，這個地址通常是從冷儲存錢包衍生的。

`channel_flags` 欄位是一個位元欄位，在撰寫本文時，只有_第一_位具有任何意義。如果設定了此位元，則此通道將作為可路由通道向公共網路廣告。否則，該通道被認為是未廣告的，通常也稱為私有通道。

accept_channel 訊息

`accept_channel` 訊息是對 `open_channel` 訊息的回應。

- 類型: 33
- 欄位:
  - `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`

`accept_channel` 訊息是資金流程中發送的第二個訊息。它用於確認與新遠端節點開啟通道的意圖。該訊息主要回應回應者希望應用於其版本承諾交易的參數集。在 支付通道 中，當我們詳細介紹資金過程時，我們會探討在開啟新通道時可以設定的各種參數的含義。

funding_created 訊息

作為回應，發起者將發送 `funding_created` 訊息。

- 類型: 34
- 欄位:
  - `32*byte`：`temp_chan_id`
  - `32*byte`：`funding_txid`
  - `uint16`：`funding_output_index`
  - `sig`：`commit_sig`

一旦通道的發起者從回應者那裡收到 `accept_channel` 訊息，他們就擁有構建承諾交易和資金交易所需的所有材料。由於通道預設是單一出資者（只有一方提交資金），只有發起者需要構建資金交易。因此，為了讓回應者為發起者簽署承諾交易版本，發起者只需要發送通道的資金輸出點。

funding_signed 訊息

最後，回應者發送 `funding_signed` 訊息。

- 類型: 34
- 欄位:
  - `channel_id`：`channel_id`
  - `sig`：`signature`

回應者收到 `funding_created` 訊息後，他們現在擁有發起者對承諾交易的有效簽章。有了這個簽章，他們可以隨時通過簽署多重簽章資金輸出的他們那一半並廣播交易來退出通道。這被稱為強制關閉。相反，為了讓發起者能夠關閉通道，回應者也簽署發起者的承諾交易。

一旦發起者收到此訊息，他們可以安全地廣播資金交易，因為他們現在能夠單方面退出通道協議。

funding_locked 訊息

一旦資金交易收到足夠的確認，就會發送 `funding_locked` 訊息。

- 類型: 36
- 欄位:
  - `channel_id`：`channel_id`
  - `pubkey`：`next_per_commitment_point`

一旦資金交易獲得 `minimum_depth` 數量的確認，雙方都應該發送 `funding_locked` 訊息。只有在發送和收到此訊息後，通道才能開始使用。

C.2.5. 通道關閉

通道關閉是一個多步驟過程。一個節點通過發送 `shutdown` 訊息來發起。然後兩個通道夥伴交換一系列 `closing_signed` 訊息來協商關閉交易的雙方都可接受的費用。通道出資者發送第一個 `closing_signed` 訊息，另一方可以通過發送具有相同費用值的 `closing_signed` 訊息來接受。

shutdown 訊息

shutdown 訊息發起關閉通道的過程，包含以下欄位：

- 類型: 38
- 欄位:
  - `channel_id`：`channel_id`
  - `u16`：`len`
  - `len*byte`：`scriptpubkey`

closing_signed 訊息

closing_signed 訊息由每個通道夥伴發送，直到他們就費用達成一致。它包含以下欄位：

- 類型: 39
- 欄位:
  - `channel_id` : `channel_id`
  - `u64` : `fee_satoshis`
  - `signature` : `signature`

C.2.6. 通道操作

在本節中，我們簡要描述用於允許節點操作通道的訊息集。通過操作，我們指的是能夠為給定通道發送、接收和轉發付款。

要通過通道發送、接收或轉發付款，必須首先將 HTLC 添加到構成通道連結的兩個承諾交易中。

update_add_htlc 訊息

`update_add_htlc` 訊息允許任何一方向對方的承諾交易添加新的 HTLC。

- 類型: 128
- 欄位:
  - `channel_id` : `channel_id`
  - `uint64` : `id`
  - `uint64` : `amount_msat`
  - `sha256` : `payment_hash`
  - `uint32` : `cltv_expiry`
  - `1366*byte` : `onion_routing_packet`

發送此訊息允許一方發起發送新付款或轉發通過傳入通道到達的現有付款。該訊息指定金額（`amount_msat`）以及解鎖付款本身的付款雜湊。下一跳的轉發指令集在 `onion_routing_packet` 欄位中進行洋蔥加密。在 洋蔥路由 中，關於多跳 HTLC 轉發，我們詳細介紹了閃電網路中使用的洋蔥路由協定。

請注意，發送的每個 HTLC 使用自動遞增的 ID，任何修改 HTLC（結算或取消）的訊息都使用該 ID 以在通道範圍內唯一引用 HTLC。

update_fulfill_hltc 訊息

`update_fulfill_hltc` 訊息允許贖回（接收）活動的 HTLC。

- 類型: 130
- 欄位:
  - `channel_id` : `channel_id`
  - `uint64` : `id`
  - `32*byte` : `payment_preimage`

此訊息由 HTLC 接收者發送給提議者以贖回活動的 HTLC。該訊息引用所討論 HTLC 的 `id`，並提供解鎖 HTLC 的原像。

update_fail_htlc 訊息

`update_fail_htlc` 訊息用於從承諾交易中移除 HTLC。

- 類型: 131
- 欄位:
  - `channel_id` : `channel_id`
  - `uint64` : `id`
  - `uint16` : `len`
  - `len*byte` : `reason`

`update_fail_htlc` 訊息是 `update_fulfill_hltc` 訊息的相反，因為它允許 HTLC 的接收者移除同一個 HTLC。當 HTLC 無法正確向上游路由並需要發送回發送者以解開 HTLC 鏈時，通常會發送此訊息。正如我們在 失敗訊息 中探討的，該訊息包含一個_加密_的失敗原因（`reason`），可能允許發送者調整其付款路由或在失敗本身是終端性的情況下終止。

commitment_signed 訊息

`commitment_signed` 訊息用於標記新承諾交易的創建。

- 類型: 132
- 欄位:
  - `channel_id` : `channel_id`
  - `sig` : `signature`
  - `uint16` : `num_htlcs`
  - `num_htlcs*sig` : `htlc_signature`

除了發送下一個承諾交易的簽章外，此訊息的發送者還需要為承諾交易上存在的每個 HTLC 發送簽章。

revoke_and_ack 訊息

`revoke_and_ack` 訊息用於撤銷過時的承諾。

- 類型: 133
- 欄位:
  - `channel_id`：`channel_id`
  - `32*byte`：`per_commitment_secret`
  - `pubkey`：`next_per_commitment_point`

因為閃電網路使用替換-撤銷承諾交易，在通過 `commit_sig` 訊息收到新的承諾交易後，一方必須撤銷其過去的承諾，然後才能收到另一個。在撤銷承諾交易時，撤銷者還提供下一個承諾點，這是允許另一方向他們發送新承諾狀態所必需的。

update_fee 訊息

`update_fee` 訊息用於更新當前承諾交易的費用。

- 類型: 134
- 欄位:
  - `channel_id`：`channel_id`
  - `uint32`：`feerate_per_kw`

此訊息只能由通道的發起者發送；只要通道開啟，他們就是支付通道承諾費用的人。

update_fail_malformed_htlc 訊息

`update_fail_malformed_htlc` 訊息用於移除損壞的 HTLC。

- 類型: 135
- 欄位:
  - `channel_id`：`channel_id`
  - `uint64`：`id`
  - `sha256`：`sha256_of_onion`
  - `uint16`：`failure_code`

此訊息類似於 `update_fail_htlc` 訊息，但在實踐中很少使用。如前所述，每個 HTLC 攜帶一個洋蔥加密路由封包，該封包還涵蓋 HTLC 本身部分的完整性。如果一方收到沿途以某種方式損壞的洋蔥封包，那麼它將無法解密該封包。因此，它也無法正確轉發 HTLC；因此，它將發送此訊息以表示 HTLC 在沿路由返回發送者的某處已損壞。

C.2.7. 通道公告

此類別中的訊息用於向更廣泛的網路公告通道圖經過身份驗證的資料結構的組件。通道圖有一系列獨特的屬性，因為添加到通道圖的所有資料也必須錨定在比特幣基礎區塊鏈中。因此，要向通道圖添加新條目，代理必須支付鏈上交易費用。這作為閃電網路的自然垃圾訊息威懾。

channel_announcement 訊息

`channel_announcement` 訊息用於向更廣泛的網路公告新通道。

- 類型: 256
- 欄位:
  - `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`

訊息中的一系列簽章和公鑰用於創建一個_證明_，證明通道確實存在於比特幣基礎區塊鏈中。正如我們在 短通道 ID 中詳細說明的，每個通道都由一個編碼其在區塊鏈中_位置_的定位器唯一識別。這個定位器稱為 `short_channel_id` ，可以放入一個 64 位元整數中。

node_announcement 訊息

`node_announcement` 訊息允許節點在更大的通道圖中公告/更新其頂點。

- 類型: 257
- 欄位:
  - `sig` : `signature`
  - `uint64` : `flen`
  - `flen*byte` : `features`

- `uint32`：`timestamp`
- `pubkey`：`node_id`
- `3*byte`：`rgb_color`
- `32*byte`：`alias`
- `uint16`：`addrlen`
- `addrlen*byte`：`addresses`

請注意，如果節點在通道圖中沒有任何已廣告的通道，則此訊息將被忽略，以確保向通道圖添加項目會產生鏈上成本。在這種情況下，鏈上成本將是創建此節點連接的通道的成本。

除了廣告其功能集外，此訊息還允許節點公告/更新可以到達它的網路 `addresses` 集。

channel_update 訊息

`channel_update` 訊息用於更新通道圖中活動通道邊緣的屬性和策略。

- 類型: 258
- 欄位:
  - `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`

除了能夠啟用/停用通道外，此訊息還允許節點更新其路由費用以及其他塑造允許通過此通道流動的付款類型的欄位。

announce_signatures 訊息

`announce_signatures` 訊息由通道對等方交換，以組裝產生 `channel_announcement` 訊息所需的簽章集。

- 類型: 259
- 欄位:
  - `channel_id` : `channel_id`
  - `short_channel_id` : `short_channel_id`
  - `sig` : `node_signature`
  - `sig` : `bitcoin_signature`

在發送 `funding_locked` 訊息後，如果雙方都希望向網路廣告其通道，那麼他們將各自發送 `announce_signatures` 訊息，這允許雙方放置生成 `announce_signatures` 訊息所需的四個簽章。

C.2.8. 通道圖同步

節點使用五個訊息來創建通道圖的本地視角：query_short_chan_ids、reply_short_chan_ids_end、query_channel_range、reply_channel_range 和 gossip_timestamp_range。

query_short_chan_ids 訊息

`query_short_chan_ids` 訊息允許節點獲取與一系列短通道 ID 相關的通道資訊。

- 類型: 261
- 欄位:
  - `chain_hash` : `chain_hash`
  - `u16` : `len`
  - `len*byte` : `encoded_short_ids`
  - `query_short_channel_ids_tlvs` : `tlvs`

正如我們在 八卦協定與通道圖 中學到的，這些通道 ID 可能是對發送者來說是新的或過時的一系列通道，這允許發送者獲取一組通道的最新資訊集。

reply_short_chan_ids_end 訊息

`reply_short_chan_ids_end` 訊息在節點完成回應先前的 `query_short_chan_ids` 訊息後發送。

- 類型: 262
- 欄位:
  - `chain_hash` : `chain_hash`
  - `byte` : `full_information`

此訊息向接收方發出信號，表示如果他們希望發送另一個查詢訊息，現在可以這樣做。

query_channel_range 訊息

`query_channel_range` 訊息允許節點查詢在區塊範圍內開啟的通道集。

- 類型: 263
- 欄位:
  - `chain_hash` : `chain_hash`
  - `u32` : `first_blocknum`
  - `u32` : `number_of_blocks`
  - `query_channel_range_tlvs` : `tlvs`

由於通道使用編碼通道在鏈中位置的短通道 ID 表示，網路上的節點可以使用區塊高度作為一種 _游標_來遍歷鏈，以發現一組新開啟的通道。

reply_channel_range 訊息

`reply_channel_range` 訊息是對 `query_channel_range` 訊息的回應，包括該範圍內已知通道的短通道 ID 集。

- 類型: 264
- 欄位:
  - `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`

作為對 `query_channel_range` 的回應，此訊息發送回在該範圍內開啟的通道集。此過程可以重複，請求者將其游標進一步向下移動以繼續同步通道圖。

gossip_timestamp_range 訊息

`gossip_timestamp_range` 訊息允許節點開始接收網路上新的傳入八卦訊息。

- 類型: 265
- 欄位:

- `chain_hash`：`chain_hash`
- `u32`：`first_timestamp`
- `u32`：`timestamp_range`

一旦節點同步了通道圖，如果他們希望接收通道圖變化的即時更新，可以發送此訊息。如果他們希望接收在離線期間可能錯過的更新積壓，他們還可以設定 `first_timestamp` 和 `timestamp_range` 欄位。

Appendix D: 來源與授權聲明

本附錄包含通過開放授權許可使用的材料的署名和授權聲明。

D.1. 來源

材料來自各種公共和開放授權來源：

- ION Lightning Network Wiki (https://wiki.ion.radar.tech)
- "Lightning 101: What Is a Lightning Invoice?" by Suredbits (https://medium.com/suredbits/lightning-101-what-is-a-lightning-invoice-d527db1a77e6)
- Lightning Network In-Progress Specifications GitHub (https://github.com/lightningnetwork/lightning-rfc); Creative Commons Attribution (CC-BY 4.0)
- Wikipedia page, "Elliptic-curve Diffie–Hellman" (https://w.wiki/4QCL)
- Wikipedia page, "Digital signature" (https://w.wiki/4QCX)
- Wikipedia page, "Cryptographic hash function" (https://w.wiki/4QCb)
- Wikipedia page, "Onion routing" (https://w.wiki/4QCc)
- Wikimedia Commons, "Lightning Network Protocol Suite" (https://w.wiki/4QCd)
- Wikimedia Commons, "Introduction to the Lightning Network Protocol and the Basics of Lightning Technology" (https://w.wiki/4QCf)

D.2. BTCPay Server

BTCPay Server logo, screenshots, and other images (https://github.com/btcpayserver/btcpayserver-media) 經 MIT 授權 (https://github.com/btcpayserver/btcpayserver-media/blob/master/LICENSE)許可使用：

MIT License

Copyright (c) 2018 BTCPay Server

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

D.3. Lamassu Industries AG

Lamassu 比特幣 ATM 中看到的 *Gaia* 比特幣 ATM (https://lamassu.is/product/gaia) 圖片經 Lamassu Industries AG 許可使用。使用這些圖片並非對該產品或公司的背書，而是作為比特幣 ATM 的視覺範例提供。

---

1. 維基百科 賽局理論條目 (https://en.wikipedia.org/wiki/Game_theory) 提供了更多資訊。
2. Joseph Poon and Thaddeus Dryja. "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments." DRAFT Version 0.5.9.2. January 14, 2016. https://lightning.network/lightning-network-paper.pdf.
3. Andreas M. Antonopoulos, *Mastering Bitcoin*, 2nd Edition, Chapter 1 (https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch01.asciidoc) (O'Reilly)
4. ACINQ：Eclair Mobile 閃電網路錢包的開發者。
5. 通常不建議為多筆付款重複使用同一個比特幣地址，因為所有比特幣交易都是公開的。 經過的好奇者可能會掃描 Alice 的 QR 碼，並在比特幣區塊鏈上看到 Alice 已經收到了多少小費到這個地址。 幸運的是，閃電網路為此提供了更私密的解決方案，將在本書後面討論！
6. Eclair 錢包不提供自動計算必要費用並將最大資金分配給通道的選項，所以 Alice 必須自己計算。
7. 雖然原始閃電網路白皮書描述了由兩個通道夥伴注資的通道，但截至 2020 年的當前規範假設只有一個夥伴向通道承諾資金。截至 2021 年 5 月，雙重注資閃電網路通道在 c-lightning 閃電網路實現中處於實驗階段。
8. George Danezis and Ian Goldberg, "Sphinx: A Compact and Provably Secure Mix Format," in *IEEE Symposium on Security and Privacy* (New York: IEEE, 2009), 269–282.

9. 「洋蔥」一詞最初由 Tor 專案使用。此外，Tor 網路也被稱為洋蔥網路，該專案使用洋蔥作為其標誌。Tor 服務在網際網路上使用的頂級域名是 *onion*。
10. George Danezis and Ian Goldberg, "Sphinx: A Compact and Provably Secure Mix Format," in *IEEE Symposium on Security and Privacy* (New York: IEEE, 2009), 269–282.
11. *HODL* 這個詞來自論壇中一個興奮的拼寫錯誤，將 "HOLD"（持有）誤拼，用來鼓勵人們不要在恐慌中出售比特幣。
12. 回想一下，找零不必是交易中的最後一個輸出，實際上與其他輸出無法區分。
13. 第一個參數（0）沒有任何意義，但由於比特幣多重簽章實作中的一個錯誤而需要。這個問題在《精通比特幣》https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch07.asciidoc[第 7 章
14. 來自 BOLT #3 (https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md)。

最後更新 2026-04-07 09:03:05 UTC