

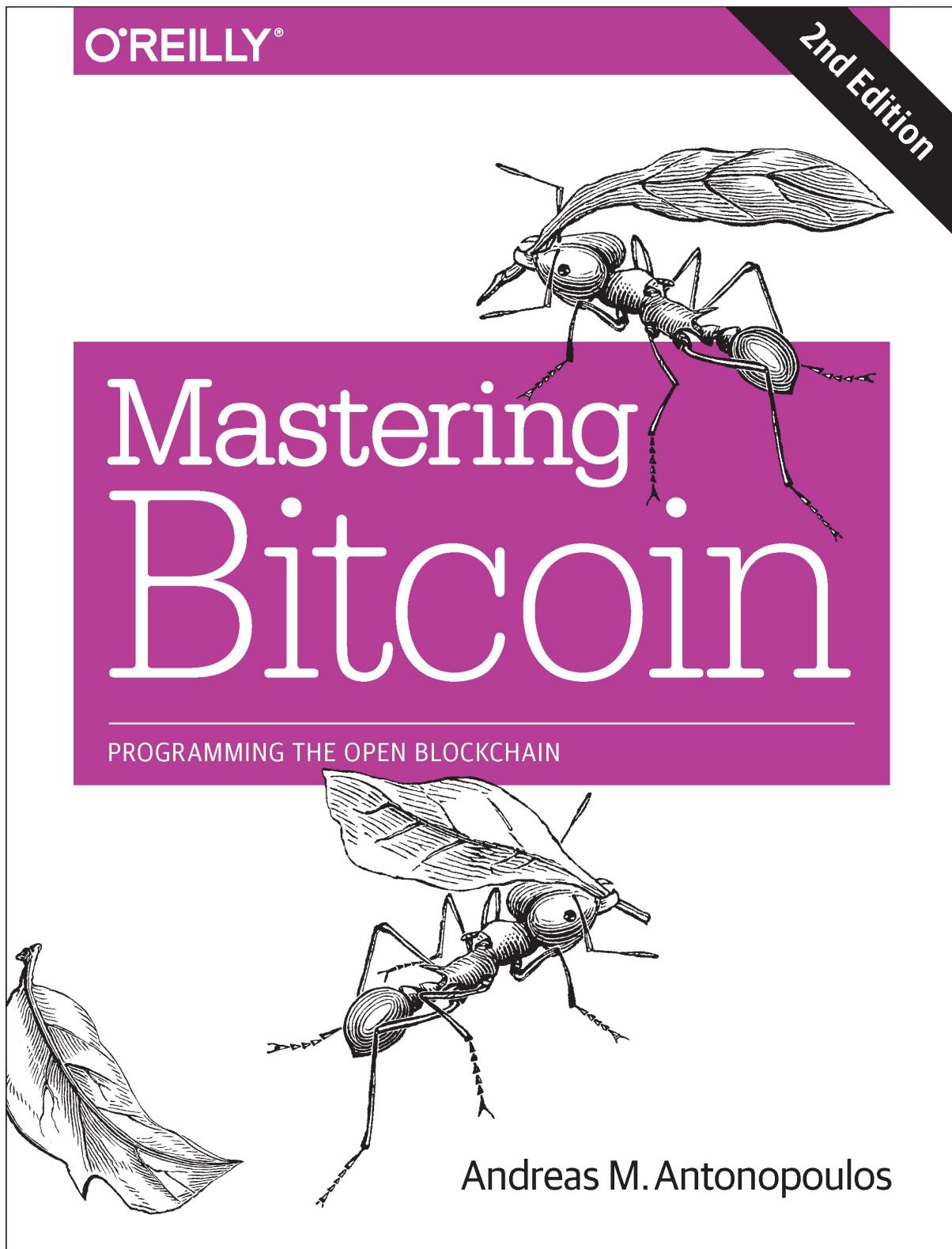
---

# Table of Contents

Introduction	1.1
前言	1.2
術語	1.3
第一章 概述	1.4
第二章 比特幣如何運作	1.5
第三章 Bitcoin Core參考實現	1.6
第四章 密鑰和地址	1.7
第五章 錢包	1.8
第六章 交易	1.9
第七章 高級交易和腳本	1.10
第八章 比特幣網路	1.11
第九章 區塊鏈	1.12
第十章 挖礦和共識	1.13
第十一章 比特幣安全	1.14
第十二章 區塊鏈應用	1.15

## Mastering Bitcoin 2nd Edition - 繁中

Language Traditional Chinese Author aantonop Translator inoutcode Translator Chen Po Wei



本書主要面向開發人員，前兩章對比特幣的介紹也適用於非開發人員。任何對技術有基本瞭解的人都可以閱讀前兩章，以深入瞭解比特幣。

## Mastering Bitcoin - Second Edition

The tags, [second\\_edition\\_print\\_1](#) and [second\\_edition\\_print2](#), correspond to the first (June 8th 2017) and second (July 20th 2017) print of Mastering Bitcoin (Second Edition), as published by O'Reilly Media.



Mastering Bitcoin - Second Edition by [Andreas M. Antonopoulos LLC](#) is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](#).

# 前言

## 寫作比特幣的書

我第一次接觸比特幣是在2011年年中，我的第一反應或多或少是“哈哈，愚蠢的貨幣”，然後我又忽視了它六個月，沒有意識到它的重要性。這是我在許多最聰明的人身上看到的一種反應，這給了我一些安慰。第二次遇到比特幣時，是在一次郵件列表討論中，我決定閱讀中本聰(Satoshi Nakamoto)撰寫的白皮書，研究權威來源，看看到底是怎麼回事。我仍然記得讀完那九頁的那一刻，我意識到比特幣不僅僅是一種數字貨幣，而是一種信任網路，它不僅可以為貨幣提供基礎，還可以為更多的東西提供基礎。意識到“這不是錢，這是一個分散的信任網路”，我開始了為期四個月的旅程，吞噬我所能找到的關於比特幣的所有訊息。我變得痴迷和著迷，每天花12個或更多的時間盯著屏幕，儘可能多地閱讀、寫作、編程和學習。當我從神遊的狀態走出來，由於飲食不穩定，我的體重減輕了20多磅，我決心致力於比特幣的研究。

兩年後，我創建了一些小型初創公司，探索各種比特幣相關的服務和產品，我決定是時候寫我的第一本書了。比特幣這個話題讓我陷入了瘋狂的創造中，吞噬了我的思想；這是自互聯網以來我遇到的最令人興奮的技術。現在是時候向更廣泛的觀眾分享我對這項神奇技術的激情了。

## 目標讀者

這本書主要是寫給開發者的。如果你能使用一種程式語言，這本書將教你密碼貨幣如何工作，如何使用它們，以及如何開發與它們一起工作的軟體。前幾章也適合於非開發者，那些試圖瞭解比特幣和密碼貨幣內部工作原理的人，用於對比特幣的深入介紹。

## 為什麼封面上有隻蟲子

切葉蟻是一種在群體超有機體中表現出高度複雜行為的物種，但每一隻螞蟻都是在一套簡單的規則上運作的，這些規則是由社交互動和化學氣味(訊息素)的交換驅動的。根據維基百科：“除人類外，切葉蟻是地球上最大、最複雜的動物群落”。切葉蟻實際上並不吃樹葉，而是用它們來種植真菌，這是蟻群的主要食物來源。由此我們能得出什麼？這些螞蟻是在耕作！

雖然螞蟻形成了一個基於群落的社會，並有一個繁殖後代的蟻后，但在蟻群中沒有中央權威或領導。一個由數百萬人組成的群體所表現出的高度智能和複雜的行為，是社交網路中個體相互作用的一種新興屬性。

自然表明，分散的系統可以是有彈性的，並且可以產生突發性的複雜性和難以置信的複雜性，而不需要一箇中央權威、層次或複雜的部分。

比特幣是一個高度複雜的分散的信託網路，可以支持各種金融流程。然而，比特幣網路中的每個節點都遵循一些簡單的數學規則。許多節點之間的交互是導致複雜行為出現的原因，而不是任何單個節點的固有複雜性或信任。就像螞蟻群落一樣，比特幣網路是一個由簡單的節點組成的彈性網路，遵循簡單的規則，這些節點在沒有任何集中協調的情況下可以一起完成令人驚歎的事情。

## 快速術語表

這個術語表包含了許多與比特幣相關的術語。這些術語在整本書中都有使用，所以把它作為一個快速參考。

### 地址 (*address*)

比特幣的地址看起來是這樣的：1DSrfJdB2AnWaFNgSbv3MZC2m74996JafV。它由一串字母和數字組成。它實際上是一個公共密鑰160位雜湊的base58check編碼版本。就像你讓別人給你發電子郵件一樣，你也會讓別人把比特幣發給你的一個比特幣地址。

### 比特幣改進建議 (*bip*)

比特幣改進建議，比特幣社區的成員們提交了一套改進比特幣的提議。例如，BIP-21提議改進比特幣統一資源標識(URI)方案。

### 比特幣 (*bitcoin*)

貨幣單位(coin)的名稱、網路和軟體。

### 區塊 (*block*)

以時間戳為標記的一組交易，以及上一個區塊的指紋。區塊的頭被進行雜湊運算以產生工作證明，從而驗證交易。有效的區塊在網路上達成共識後被到主區塊鏈中。

### 區塊鏈 (*blockchain*)

一系列經過驗證的區塊組成的列表，每個區塊都鏈接到它的前一個區塊，一直到創世區塊。

### 拜占庭將軍問題 (*Byzantine Generals Problem*)

一個可靠的電腦系統必須能夠處理它的一個或多個組件的故障情況。失敗的組件可能表現出一種經常被忽略的行為，即向系統的不同部分發送衝突訊息。處理這類失敗的問題抽象地表述為拜占庭將軍問題。

### 幣基 (*coinbase*)

一種特殊的欄位，用作幣基交易的唯一輸入。coinbase允許聲明區塊獎勵，並可提供最多100字節的任意數據。不要與幣基交易 (coinbase transaction) 混淆。

### 幣基交易 (*coinbase transaction*)

區塊中的第一筆交易。通常由礦工創建，包含一個幣基。不要與幣基 (coinbase) 混淆。

### 冷儲存 (*cold storage*)

指的是保持比特幣的離線儲備。當比特幣私鑰被創建並儲存在安全的脫機環境中時，就可以實現冷儲存。對持有比特幣的人來說，冷儲存很重要。網路電腦容易受到黑客攻擊，不應該用來儲存大量比特幣。

### 彩色幣 (*colored coins*)

一個開源的比特幣2.0協議，允許開發者利用比特幣區塊鏈的功能創建數字資產。

### 確認 (*confirmations*)

一旦交易包含在一個區塊中，它就有一個確認。只要有另一個區塊在同一個區塊鏈上開採，交易就會有兩個確認訊息，依此類推。六個或更多的確認被認為足以證明交易無法撤銷。

### 共識 (*consensus*)

當幾個節點（通常是網路上的大多數節點）在其本地驗證的最佳區塊鏈中都具有相同的區塊時，即達成共識。不要與共識規則 (consensus rules) 混淆。

### 共識規則 (*consensus rules*)

完整節點遵循的區塊驗證規則，為了與其他節點達成共識。不要與共識 (consensus) 混淆。

### 挖礦難度 (*difficulty*)

一個全網範圍的配置，控制需要多少計算來產生工作證明。

### 難度重新計算 (*difficulty retargeting*)

每產生2,016個區塊，全網重新計算一次挖礦難度，並考慮先前的2016個區塊的雜湊計算能力。

#### 難度目標 (*difficulty target*)

使網路中平均10分鐘生產一個區塊的難度。

#### 雙重支付 (*double-spending*)

雙重支付是將某筆前成功花費一次以上的結果。通過驗證每個添加到區塊鏈的交易來確保交易的輸入先前沒有花費，比特幣可以防止雙重支出。

#### 橢圓曲線數字簽名演算法 (*ECDSA*)

ECDSA (Elliptic Curve Digital Signature Algorithm) 是比特幣使用的一種加密演算法，以確保資金只能由其合法所有者使用。

#### 額外的隨機數 (*extra nonce*)

隨著困難的增加，礦工經常循環遍歷所有40億個臨時值，仍沒有產生區塊。因為幣基 (coinbase) 腳本可以儲存2到100個字節的數據，所以礦工開始使用這個空間作為額外的隨機數空間，允許他們探索更大範圍的區塊頭數據以找到有效的區塊。

#### 交易費 (*fees*)

交易的發起人通常會向網路提供交易處理的費用。大多數交易需要0.5mBTC的最低費用。

#### 分叉 (*fork*)

分叉，或意外分叉，當兩個或兩個以上的區塊具有相同的區塊高度時發生，導致區塊鏈分叉。通常發生在兩個或多礦工幾乎同時發現礦區塊時，也可能作為一種攻擊方式發生。

#### 創世區塊 (*genesis block*)

區塊鏈中的第一個區塊，用來初始化密碼貨幣。

#### 硬分叉 (*hard fork*)

硬分叉 (Hard fork)，也叫做硬分叉更改 (Hard-Forking Change)，是區塊鏈中的一種永久分歧，通常發生於未升級的節點無法驗證升級節點創建的區塊時。不要與分叉 (fork) ，軟分叉 (soft fork) ，軟體分叉 (software fork) 或 Git分叉 (git fork) 混淆。

#### 硬體錢包 (*hardware wallet*)

硬體錢包是一種特殊類型的比特幣錢包，它將用戶的私鑰儲存在安全的硬體設備中。

#### 雜湊 (*hash*)

一些二進制輸入的數字指紋。

#### 雜湊鎖 (*hashlocks*)

雜湊鎖是一種限制一筆輸出在指定的數據公開前不能被消費的財產留置權。雜湊鎖非常有用，一旦一把雜湊鎖被打開，任何其他使用相同密鑰保護的雜湊鎖也會被打開。這使得我們可以創建多個輸出，這些輸出都被同一個雜湊鎖留置，並且可以在同一時間變成可消費的。

#### 分層確定性協議 (*HD protocol*)

分層確定性 (HD) 密鑰創建和傳輸協議 (BIP32) ，允許從層次結構中的父密鑰創建子密鑰。

#### 分層確定性錢包 (*HD wallet*)

使用分層確定性 (HD Protocol) 密鑰創建和傳輸協議 (BIP32) 的錢包。

#### 分層確定性錢包種子 (*HD wallet seed*)

HD錢包種子或根種子是一種可能很短的值，用於生成HD錢包的主私鑰和主鏈程式碼的種子。

#### 雜湊時間鎖定合約 (*HTLC*)

雜湊時間合約 (Hashed TimeLock Contract) 或HTLC是一種支付類型，它使用雜湊鎖和時間鎖來要求一筆支付的收款方要麼在指定日期之前通過生成加密收款證明，要麼放棄接受支付的權力，將其返還給支付方。

#### 瞭解你的客戶 (*KYC*)

瞭解你的客戶 (Know your customer, KYC) 是一項企業活動，表示識別並驗證它的客戶。該術語也用於指代管理這些活動的銀行法規。

#### **LevelDB**

LevelDB是一個開源的基於硬碟的鍵值儲存引擎。LevelDB是一個用於持久化儲存的，輕量級的，單用途庫，與許多平臺綁定。

#### **閃電網路 (Lightning Networks)**

閃電網路是帶有雙向支付渠道的雜湊時間鎖合約 (HTLC) 的建議實現，其允許多筆支付在多個點對點支付渠道上安全路由。這樣就可以形成一個網路，網路中的任何一點都可以向任何其他點發起支付，即使他們之間沒有直接通道。

#### **鎖定時間 (Locktime)**

Locktime, 或者更專業地叫做nLockTime, 是交易的一部分，它表明該交易可能被添加到區塊鏈時最早的時候或最早的區塊。

#### **Memory pool (mempool)**

比特幣Memory pool (memory pool) 是經過比特幣節點驗證但尚未確認的所有交易數據的集合。

#### **默克爾根 (merkle root)**

Merkle樹的根節點，區塊頭必須包含一個有效的merkle根，根據該區塊中的所有交易生成。

#### **默克爾樹 (merkle tree)**

通過計算每對兒數據（樹葉）的雜湊值構建的樹，然後再對結果進行配對和雜湊，直到只剩一個雜湊值，即merkle根。在比特幣中，葉子幾乎總是來自單個區塊的交易。

#### **礦工 (miner)**

是指一個網路節點，通過重複雜湊計算，來尋找新區塊的有效工作證明。

#### **多重簽名 (multisignature)**

多重簽名 (multisigature) 是指要求多個密鑰授權比特幣交易。

#### **網路 (network)**

一個點對點網路，用於將交易和數據區塊傳播到網路上的每個比特幣節點。

#### **隨機數 (nonce)**

比特幣區塊中的“nonce”是一個32位（4字節）的欄位，通過設置它的值可以使得區塊的雜湊值包含若干個前導零。其餘的欄位可能不會改變，因為它們具有定義的含義。

#### **脫鏈交易 (off-chain transactions)**

脫鏈交易是區塊鏈之外的價值轉移，鏈上交易（通常簡稱為交易）修改區塊鏈並依靠區塊鏈來確定其有效性，脫鏈交易依賴於其他方法來記錄和驗證交易。

#### **opcode**

比特幣腳本語言的操作程式碼，用於在公鑰腳本或簽名腳本中推送數據或執行功能。

#### **開放資產協議 (Open Assets protocol)**

開放資產協議 (Open Assets Protocol) 是一個建立在比特幣區塊鏈之上的簡單而強大的協議。它允許發佈和傳輸用戶創建的資產。開放資產協議是彩色幣概念的演變。

#### **OP\_RETURN**

OP\_RETURN交易中的一個輸出中使用的操作碼。不要與OP\_RETURN交易混淆。

#### **OP\_RETURN 交易**

一種交易類型，它將任意數據添加到可證明不可消費的pubkey腳本中，完整節點不需要儲存在其UTXO資料庫中。不要與OP\_RETURN操作碼混淆。

#### **孤兒區塊 (orphan block)**

其父區塊還未被本地節點驗證的區塊，所以它們也不能被完全驗證。不要和陳腐區塊 (stale block) 混淆

### 孤兒交易 (*orphan transactions*)

由於缺少一個或多個輸入交易，而無法進入交易池的交易。

### P2PKH

向比特幣地址支付的交易包含P2PKH（Pay To PubKey Hash）腳本，由P2PKH鎖定的輸出可以通過公鑰和由對應的私鑰創建的數字簽名來解鎖（消費）。

### P2SH

P2SH（Pay-to-Script-Hash）是一種功能強大的新型交易，大大簡化了複雜交易腳本的使用。通過P2SH，詳細說明消費輸出（贖回腳本）的複雜腳本不會顯示在鎖定腳本中，只有它的雜湊值在鎖定腳本中。

### P2SH地址

P2SH地址是一個腳本的20比特雜湊值的Base58Check編碼，P2SH地址使用版本前綴“5”，導致Base58Check編碼後的地址以“3”開頭。P2SH地址隱藏了所有的複雜性，因此付款人看不到腳本。

### P2WPKH

P2WPKH（Pay-to-Witness-Public-Key-Hash）的簽名包含與P2PKH支出相同的訊息，但位於witness欄位而不是scriptSig欄位。scriptPubKey也被修改了。

### P2WSH

P2SH和P2WSH（Pay-to-Witness-Script-Hash）之間的差異是加密證明訊息的位置從scriptSig欄位變為witness欄位，scriptPubKey欄位也被修改了。

### 紙錢包 (*paper wallet*)

具體來講，紙質錢包是一個檔案，其中包含生成任意數量的比特幣私鑰所需的所有數據，形成了密鑰的錢包。但是，人們經常使用這個術語來表示任何將比特幣作為物理文件離線儲存的方式。第二個定義還包括紙密鑰和可兌換程式碼。

### 支付通道 (*payment channels*)

小額支付通道或支付通道是一類技術，旨在允許用戶進行多個比特幣交易，而無需將所有交易交給比特幣區塊鏈。在典型的支付通道中，只有兩筆交易被添加到區塊鏈中，但參與者之間可以進行無限次或幾乎無限次數的付款。

### 礦池挖礦 (*pooled mining*)

礦池採礦是一種採礦方式，其中多個客戶端合力產生一個區塊，然後根據它們提供的處理能力分割區塊獎勵。

### 權益證明 (*Proof-of-Stake*)

權益證明（Proof-of-Stake，PoS）是一種密碼貨幣區塊鏈網路實現分佈式共識的方法。權益證明要求用戶證明一定數量的貨幣（它們在貨幣中的“股份”）的所有權。

### 工作量證明 (*Proof-of-Work*)

需要大量計算才能找到特定數據。在比特幣中，礦工必須找到SHA256演算法的數字解決方案，以滿足整個網路的目標，即難度目標。

### 獎勵 (*reward*)

包含在每個新區塊中的一定數量的比特幣，作為對網路中發現工作證明解決方案的礦工的獎勵。目前它是每個區塊12.5比特幣。

### RIPEMD-160

RIPEMD-160是一個160位的加密雜湊方法。RIPEMD-160是RIPEMD的一個加強版本，具有160位雜湊結果，預計在未來十年或更長時間內可以保證安全。

### 中本聰 (*satoshi*)

聰（satoshi）是可以記錄在區塊鏈上的最小比特幣單位。它相當於0.00000001比特幣，以比特幣的創造者中本聰（Satoshi Nakamoto）命名

### 中本聰（Satoshi Nakamoto）

中本聰（Satoshi Nakamoto）是設計比特幣並開發原始參考實現程式碼的一個人或幾個人的用名，作為實施的一部分，他們還設計了第一個區塊鏈資料庫。在這個過程中，他們率先解決了數字貨幣的雙重支付問題，但他們的真實身份仍然未知。

### 腳本（Script）

比特幣使用腳本系統進行交易。腳本很簡單，基於堆疊，並且從左到右進行處理。它故意設計成不是圖靈完備的，不支持迴圈。

### 公鑰腳本（ScriptPubKey，pubkey script）

ScriptPubKey或pubkey script, 是一個包含在輸出中的腳本，它為消費那些satoshis設定了必須滿足的條件。滿足條件的數據可以在簽名腳本中提供。

### 簽名腳本（ScriptSig，signature script）

ScriptSig或signature script, 是由付款人生成的，作為滿足公鑰腳本（PubKey Script）的變數

### 私鑰（secret key，private key）

解鎖特定地址上的比特幣的私密數字，看起來如下：

```
5J76sF8L5jTtzE96r66Sf8cka9y44wdpJjMwCxR3tzLh3ibVPxh
```

### 隔離見證（Segregated Witness）

隔離見證是對比特幣協議的升級建議，技術上創新地將簽名數據與比特幣交易分開。隔離見證是一種建議的軟分叉，技術上的變化使得比特幣的協議規則更具限制性。

### SHA

安全雜湊演算法或SHA是美國國家標準與技術研究院（NIST）發佈的一系列加密雜湊函數。

### 簡單支付驗證（Simplified Payment Verification，SPV）

簡單支付驗證（SPV）或是一種驗證特定交易是否包含在一個區塊中的方法，不需要下載整個區塊。該方法被一些輕量級比特幣客戶使用。

### 軟分叉（soft fork）

軟分叉是區塊鏈中的臨時分叉，通常當礦工使用不遵從新共識方法的未升級的節點時發生。不要和分叉、硬分叉、軟體分叉或Git分叉混淆。

### 陳腐區塊（stale block）

已成功開採但未包括在當前最佳區塊鏈中的區塊，可能是因為同一高度的其他區塊首先擴展了其鏈條。不要與孤兒區塊混淆。

### 時間鎖（timelocks）

時間鎖是一種限制某些比特幣直到指定的未來時間或區塊高度才能支出的留置權。時間鎖在許多比特幣合約中有重要作用，包括支付通道，和雜湊時間鎖定合約。

### 交易（transaction）

簡單來說，是指從一個地址向另一個地址傳輸比特幣。具體而言，交易是表示價值轉移的簽名資料結構。交易通過比特幣網路進行傳輸，由礦工收集并包含在區塊鏈中，永久保存在區塊鏈中。

### 交易池（transaction pool）

一個無序的交易集合，它不在主鏈中的區塊中，但是我們可以拿到輸入交易。

### 圖靈完備（Turing completeness）

如果程序語言能夠運行圖靈機可以運行的任何程序，並給予足夠的時間和內存，那麼它就稱為“圖靈完備”的。

### 未花費交易輸出（unspent transaction output，UTXO）

UTXO是一項未花費的交易輸出，可以作為新交易的輸入使用。

### 錢包 (*wallet*)

擁有你的所有比特幣的地址和密鑰的軟體，用它來發送，接收和儲存你的比特幣。

### 錢包匯入格式 (*Wallet Import Format, WIF*)

WIF或電子錢包匯入格式是一種數據交換格式，允許匯出和匯入帶有標誌的單個私鑰，該標誌表示它是否使用壓縮的公鑰。

## 概述

### 什麼是比特幣

比特幣是組成數字貨幣生態的一系列概念和技術的集合。比特幣也用作在比特幣網路的參與者之間儲存和傳遞價值的貨幣單位。比特幣用戶主要通過互聯網使用比特幣協議進行通信，當然，也可以使用其他傳輸網路。比特幣協議堆疊是開源的，易於使用，可運行在各種計算設備上，包括筆記本電腦和智能手機。

用戶可以通過比特幣完成傳統貨幣可以完成的任何事情，包括購買和出售商品，向人們或組織匯款，或延長信貸。用戶可以在專門的交易所購買、出售比特幣，與其他貨幣進行兌換。某種意義上，比特幣是互聯網金錢的完美形式，因為它是快速，安全和無邊界的。

與傳統貨幣不同，比特幣是完全虛擬的。它沒有實體硬幣甚至數字硬幣。比特幣是隱含在發起者向接收者傳遞價值的交易中的。通過擁有密鑰，用戶可以證明比特幣的所有權，可以簽署交易來解鎖價值，並將其轉移給新的所有者。密鑰通常儲存在每個用戶電腦或手機中的數字錢包裡。擁有密鑰是消費比特幣的唯一先決條件，比特幣的控制權完全掌握在每個用戶的手中。

比特幣是一個點對點的分佈式系統，系統中不存在中央伺服器或控制點。比特幣是通過一個叫做挖礦的過程創造的，礦工在處理比特幣交易時通過大量計算競猜一個數學題的答案。比特幣網路的任何參與者（使用一個設備運營比特幣完整協議堆疊的用戶）都可以成為礦工，利用他們電腦的處理能力來驗證和記錄交易。平均每隔10分鐘，會有一個礦工計算出數學題的答案，可以驗證過去10分鐘的交易，並且獲得新發行的比特幣獎勵。實質上，比特幣將中央銀行的貨幣發行和清算功能進行了去中心化，不再需要任何中央銀行了。

比特幣協議包含了內置的用於調節網路中挖礦方法的演算法。挖礦過程的難度是動態調整的，所以，無論網路中有多少礦工，平均每10分鐘都會有一個人成功。比特幣的總量限制在2100萬個，每隔4年，比特幣的發行速率都會減半，到2140年，所有比特幣發行完畢。由於比特幣的發行速度遞減，長期來看，比特幣是貨幣通縮的。

比特幣是協議的名稱，是一個點對點網路，也是分佈式計算的創新。比特幣貨幣本身只是這項創新的第一個應用。比特幣代表了數十年密碼學和分佈式系統研究的一個高峰，以獨特而強大的方式將四項關鍵創新融合在一起：

- 去中心化的點對點網路（比特幣協議）
- 公開的賬本（區塊鏈）
- 一套獨立交易驗證和貨幣發行的規則（共識協議）
- 在有效的區塊鏈上達成全球分散共識的機制（PoW工作證明機制）

作為一名開發人員，我認為比特幣就像貨幣互聯網一樣，是一個通過分佈式計算傳播價值和保護數字資產所有權的網路。比特幣包含的內容比第一眼見到的還要多。

本章，我們將從解釋一些主要概念和名詞開始，安裝必要的軟體，並使用比特幣進行簡單的交易。在後面的章節，我們將展開底層技術，研究比特幣網路和協議的內部機制。

#### 比特幣之前的數字貨幣

可行數字貨幣的出現與加密技術的發展密切相關。使用數據位代表交換物品和服務的價值的想法並不奇怪，但想讓人接受數字貨幣，要面對3個基本問題：

1. 可以相信這筆錢是真實的，而不是偽造的嗎？
2. 可以確定數字貨幣只能使用一次嗎？（俗稱“雙重支付”，“雙花”問題）
3. 可以確保除了我以外，沒有人可以聲稱我的一筆錢屬於他們嗎？

紙幣發行者使用日益複雜的紙張和印刷技術來對抗偽造問題。物理貨幣可以輕易解決雙重支付問題，因為同一張紙幣不可能在兩個地方出現。當然，傳統貨幣也經常以數字方式儲存和傳輸。在這種情況下，偽造和雙重支付問題的方法是通過中央當局來清算所有電子交易，他們對流通中的貨幣具有全局視野。對於不能利用深奧的墨水或全息條帶技術的

數字貨幣，密碼學為其提供了可信的所有權的基礎。具體而言，加密數字簽名使用戶能夠簽署證明該資產所有權的數字資產或交易。採用適當的架構，數字簽名也可以解決雙重支付問題。

隨著密碼學在20世紀80年代後期開始廣泛應用，許多研究人員開始嘗試使用密碼學構建數字貨幣。這些早期發行的數字貨幣，通常由國家法幣或稀有金屬（黃金）做背書。

儘管這些早期數字貨幣可以運行，但它們是中心化的，結果是，很容易受到政府或黑客的攻擊。它們使用中央票據交換所來定期處理所有交易，就像傳統的銀行系統一樣。不幸的是，多數情況下，這些早期數字貨幣成為了過度擔心的政府的目標，不復存在。為了防止被幹預（無論政府還是犯罪分子），需要可以避免單點攻擊的去中心化數字貨幣。比特幣就是這樣一個系統，去中心化設計，不需要可能崩潰或者被攻擊的任何中心機構。

## 比特幣的歷史

比特幣是在2008年隨著一篇署名中本聰的論文 "Bitcoin: A Peer-to-Peer Electronic Cash System,"<sup>[1]</sup> 的發表誕生的 (see [satoshi\_whitepaper](<https://bitcoin.org/bitcoin.pdf>)). 中本聰結合了b-money和HashCash等之前幾個發明，創造了一個完全去中心化的電子現金系統，它不依賴於中央機構進行貨幣發行、交易結算和確認。關鍵性的創新是使用分佈式計算系統（Proof-of-Work（工作證明）演算法）每10分鐘進行一次全局“選舉”，從而使去中心化網路就交易狀態達成共識。這個方法優雅地解決了雙重支付（一筆現金可以支付兩次）問題。在之前，雙重支付問題是數字貨幣的弱點，通常通過中央機構清算所有交易解決。

比特幣網路於2009年上線，基於中本聰發佈的一個參考實現並由許多其他開發者修訂。為比特幣提供安全性和可靠的Proof-of-Work演算法（挖礦演算法）的算力呈指數型增長，現在已經超過了世界上頂尖的超級電腦的算力。比特幣的市值曾超過1350億美元。目前最大的一筆交易是4億美元，即時到賬，手續費1美元。

中本聰於2011年4月淡出了公眾視野，將維護比特幣程式碼和網路的任務交給了一組志願者。這個比特幣背後的人或組織的身份仍然是未知的。然而，中本聰和其他人都沒有對比特幣系統進行控制，該系統基於完全透明的數學原理，開源程式碼以及參與者之間的共識。這項發明本身就具有開創性，已經在分佈式計算，經濟學和計量經濟學領域產生了新的科學。

### 分佈式計算問題的解決方案

中本聰的發明也是一個分佈式計算問題的實踐和創新：“拜占庭將軍問題”。簡單來說，這個問題涉及試圖在不可靠且可能受損的網路上交換訊息來達成行動方案或系統狀態的統一。中本聰的Proof-of-Work的方案，不需要中心化的可信機構，即可達成共識，代表了分佈式計算領域的突破，有著超越數字貨幣的廣泛適用性。它可以用於在去中心化網路中達成共識，證明選舉，抽獎，資產登記，數字公證等的公平性。

## 比特幣的用法，用戶和故事

比特幣是古老的金錢技術的創新。金錢的核心是簡單地促進人們之間交換價值。所以，為了全面理解比特幣和它的用法，我們將從人們使用它的視角進行研究。下面列出的每個故事，都涉及了一種或多種用法：

### 北美的低價零售

Alice住在北美加利福尼亞灣區。她從技術人員朋友那裡聽說了比特幣之後想嘗試一下。我們將跟隨她的故事，瞭解比特幣，獲取比特幣，花費一些買一杯咖啡。這個故事將從零售消費者的角度介紹軟體，交易所，和基本交易。

### 北美高價零售

Carol是舊金山的藝術畫廊老闆。她使用比特幣出售昂貴的作品。這個故事將向高價值商品的零售商介紹“51%”共識攻擊的風險。

### 離岸合同服務

帕洛阿爾託的咖啡館老闆Bob正在建設一個新網站。他與居住在印度班加羅爾的一位網路開發人員Gopesh簽約。Gopesh已同意接受比特幣支付。這個故事將研究比特幣在外包，合同服務和國際電匯方面的用途。

### 網路商店

Gabriel是里約熱內盧的一名進取的年輕少年，他經營一家小型網上商店，銷售比特幣品牌的T恤，咖啡杯和貼紙。加百列太年輕了，沒有銀行賬戶，但他的父母鼓勵他的企業家精神。

## 慈善捐款

Eugenia是菲律賓兒童慈善組織的負責人。最近，她發現了比特幣，並希望利用它來接觸一群全新的外國和國內捐助者，為她的慈善事業籌款。她還在研究如何使用比特幣將資金快速分配到需要的地方。這個故事將展示比特幣在跨越貨幣和邊界的全球籌款活動中的應用，以及透明的公開賬本在慈善組織中的使用。

## 進出口

穆罕默德是迪拜的一家電子產品進口商。他試圖用比特幣從美國和中國購買電子產品進口到阿聯酋，以加快進口支付流程。這個故事將展示如何將比特幣用於與實體商品相關的大型企業對企業國際支付。

## 比特幣挖礦

Jing是上海的電腦工程專業的學生。他已經建立了一個礦機，利用他的工程技能來挖掘比特幣，以獲取額外收入。這個故事將研究比特幣的“工業”基礎：用於保護比特幣網路和發行新貨幣的專用設備。

每一個故事都基於真實的人和真正的行業，目前正在使用比特幣來創建新的市場，新的行業以及針對全球經濟問題的創新解決方案。

## 開始

比特幣是一種協議，可以通過使用遵守協議的客戶端訪問。“比特幣錢包”是比特幣系統最常用的用戶界面，就像網路瀏覽器是HTTP協議最常用的用戶界面一樣。比特幣錢包有很多實現和品牌，就像許多品牌的網路瀏覽器（例如，Chrome，Safari，Firefox和Internet Explorer）一樣。就像我們都有我們最喜歡的瀏覽器（Mozilla Firefox）和最討厭的瀏覽器（Internet Explorer）一樣，比特幣錢包在質量，性能，安全性，隱私和可靠性方面各不相同。比特幣協議還有一個源自中本聰編寫的包含錢包的參考實現，名為“Satoshi Client”或“Bitcoin Core”。

## 選擇比特幣錢包

比特幣錢包是比特幣生態系統中最積極開發的應用之一。競爭很激烈，可能現在有人正在開發一個新的錢包，但去年的一些錢包已不再被維護。許多錢包專注於特定平臺或特定用途，有些更適合初學者，而其他則提供更多高級功能。如何選擇錢包依賴於用途和用戶體驗，所以無法推薦一個特定的品牌或錢包。但是，我們可以根據它們的平臺和功能進行分類，並對這些不同的錢包進行介紹。有一點好處是，在比特幣錢包之間移動鑰匙或種子相對容易，所以可以多嘗試幾個錢包直到找到符合你需求的。

比特幣錢包根據平臺分類如下：

### 桌面錢包

桌面錢包是作為參考實現創建的第一種比特幣錢包，許多用戶因為它們提供的功能、自治和控制而使用桌面錢包。運行在Windows或MacOS操作系統上有安全缺陷，因為這些系統通常是不安全和配置不善的。

### 移動錢包

移動錢包是最常用的。這類錢包運行在iOS或Android操作系統上，是新用戶的不錯選擇。多數設計簡單易用，但也有提供給高級用戶使用的功能全面的移動錢包。

### 網路錢包

網路錢包是通過瀏覽器訪問的，並且將用戶的錢包儲存在第三方的伺服器上。一些這樣的服務通過在用戶的瀏覽器中使用客戶端程式碼進行操作，該程式碼將比特幣密鑰控制在用戶手中。然而，多數情況下，第三方會控制用戶的比特幣密鑰以便用戶方便使用。將大量比特幣儲存在第三方系統上是不可取的。

### 硬體錢包

硬體錢包是在專用硬體上運行安全的自包含比特幣錢包的設備。它們通過USB鏈接桌面Web瀏覽器，或通過移動設備上的近場通信（NFC）功能進行操作。在專用硬體上處理所有與比特幣相關的操作被認為非常安全，適合儲存大量的比特幣。

### 紙錢包

控制比特幣的密鑰也可以打印到紙上，也可以使用其他材料（木材，金屬等），這些被稱為紙錢包。紙錢包提供了一種低技術含量但高度安全的長期儲存比特幣的手段。脫機儲存通常也被稱為冷儲存。

另一種給比特幣錢包分類的方法是根據他們的自治程度以及與如何比特幣網路交互：

#### **完整節點客戶端 (*Full-node client*)**

一個完整的客戶端或“完整節點”儲存比特幣交易歷史（每個用戶的每次交易），管理用戶的錢包，並且可以直接在比特幣網路上啟動交易。完整節點處理協議的所有方面，並可獨立驗證整個區塊鏈和任何事務。完整節點需要消耗大量電腦資源（例如，超過125 GB的硬碟，2GB的RAM），但可提供完整的自主權和獨立的事務驗證。

#### **輕量級客戶端**

輕量級客戶端也稱為簡單支付驗證（SPV，Simple-payment-verification）客戶端，連接到比特幣完整節點以訪問比特幣交易訊息，但將用戶錢包本地儲存並獨立創建，驗證和傳輸交易。輕量級客戶端與比特幣網路直接交互，無需中間人。

#### **第三方API客戶端**

第三方API客戶端是通過第三方系統的API與比特幣交互的客戶端，而不是直接連接到比特幣網路。錢包可以由用戶或第三方伺服器儲存，但所有交易都通過第三方。

結合這些分類，許多比特幣錢包會被分入多個組內，其中最常見的三種是桌面完整客戶端，移動輕量級錢包和第三方網路錢包。不同類別之間的界限通常很模糊，因為許多錢包在多個平臺上運行，並且可能以不同的方式與網路進行交互。

為了本書的目的，我們將演示使用各種可下載的比特幣客戶端，從參考實現（比特幣核心）到移動錢包和網路錢包。一些例子將需要使用比特幣核心，除了作為一個完整的客戶端之外，它還將API暴露給錢包，網路和交易服務。如果你計劃探索比特幣系統的編程接口，則需要運行比特幣核心或其他客戶端之一。

## **快速開始**

我們之前介紹的Alice不是技術人員，而且最近才從朋友Joe那聽說比特幣。在一次派對上，Joe又一次熱情地向周圍的人講解和演示比特幣。出於好奇，Alice想知道她如何開始使用比特幣。Joe說移動錢包最適合新用戶，並推薦了一些他最喜愛的錢包。Alice便將“Mycelium”安裝到了她的Android手機上。

當愛麗絲第一次運行Mycelium時，程序會自動為她創建一個新錢包。Alice看到的錢包界面，如[The Mycelium Mobile Wallet](#)所示（注意：不要將比特幣發送到此示例地址，它將永遠丟失）。

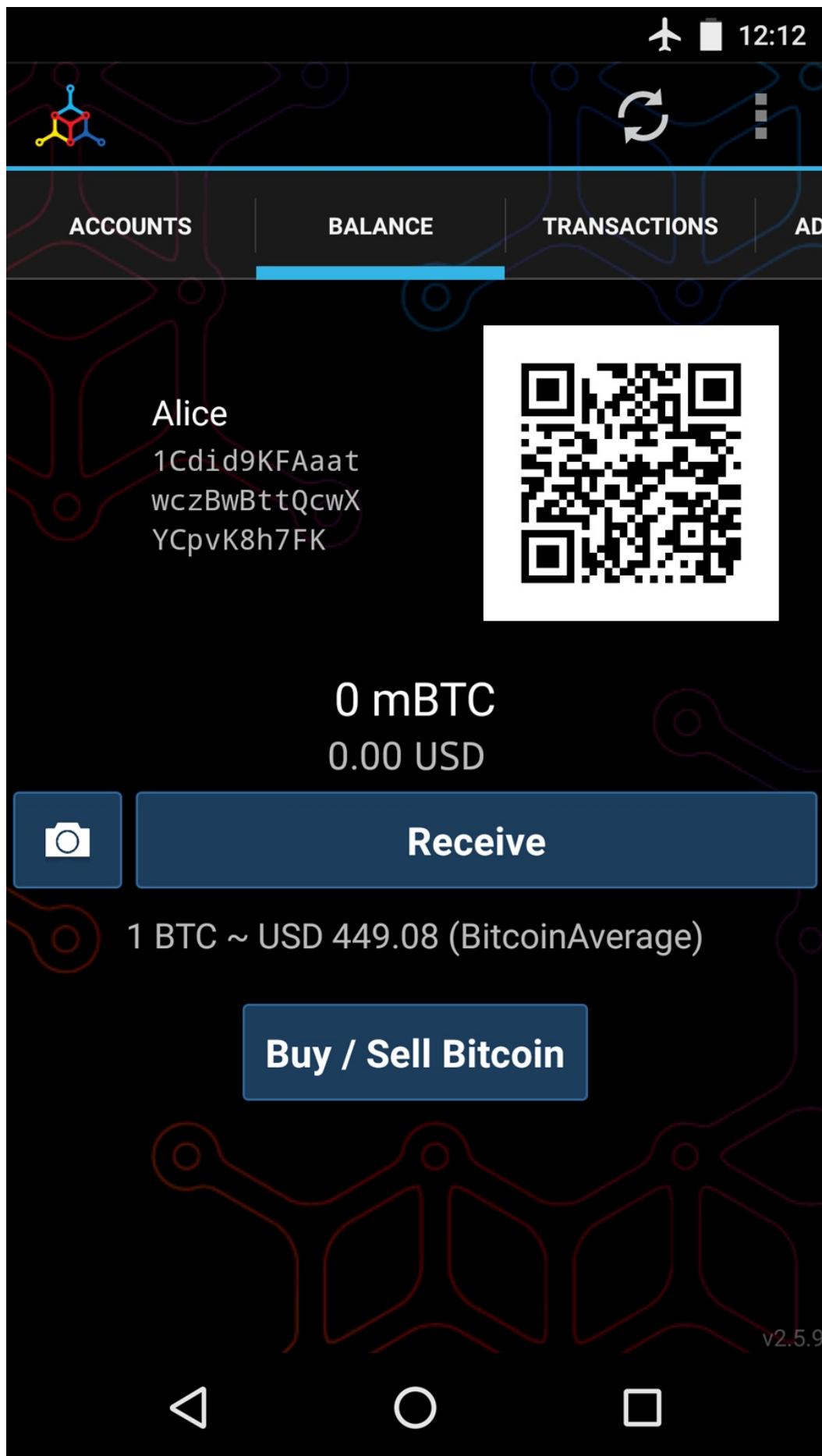


Figure 1. The Mycelium Mobile Wallet

界面上最重要的部分是Alice的比特幣地址 (*bitcoin address*)，是數字和字母的組合：

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK。比特幣地址旁邊是存有相同訊息的二維碼，條形碼，可以通過手機掃描。Alice可以通過點擊二維碼或Receive按鈕保存比特幣地址，或將二維碼保存到手機中。在大多數錢包中，二維碼可以點擊放大，更方便掃描。

Tip

比特幣地址以"1"或者"3"開頭。就像email地址一樣，它們可以分享給其他比特幣用戶以允許它們向你的錢包發送比特幣。從安全角度來說，比特幣地址不存在任何敏感訊息，他可以被髮送到任何地方。與email地址不同，你可以經常創建新的比特幣地址，所有的地址都關聯到你的錢包。許多現代錢包會自動為每筆交易創建一個新地址，以最大限度地保護隱私。錢包只是地址和解鎖資金的密鑰集合。

Alice現在已經準備好接收資金了。她的錢包應用會隨機生成一個私鑰（在[[private\\_keys](#)]中更詳細地描述）以及相應的比特幣地址。這時，她的比特幣地址不為比特幣網路所知，或者在比特幣系統的任何部分“註冊”。她的比特幣地址只是一個數字，對應於一個可以用來控制資金訪問權限的密鑰。它是由她的錢包獨立生成的，沒有參考或註冊任何服務。事實上，在大多數錢包中，比特幣地址與包括用戶身份在內的任何外部可識別訊息之間不存在關聯。在比特幣地址被比特幣賬本上發佈的交易引用，作為接收地址之前，它僅僅是比特幣中有效的大量可能的地址的一部分。只有與交易關聯後，它才會成為網路中已知地址的一部分。

Alice現在準備開始使用她的新比特幣錢包了。

## 獲得你的第一個比特幣

新用戶的第一個也是最困難的任務是購買一些比特幣。與其他外幣不同，你還不能在銀行或外匯交易市場購買比特幣。

比特幣交易是不可逆轉的。大多數電子支付網路如信用卡，借記卡，PayPal和銀行賬戶轉賬都是可逆的。對於銷售比特幣的人來說，這種差異帶來了非常高的風險，即買家在收到比特幣後會逆轉電子支付，實際上欺騙了賣家。為了緩解這種風險，接受傳統電子支付以換取比特幣的公司通常要求買家進行身份驗證和信用評估檢查，這可能需要幾天或幾周的時間。作為新用戶，這意味著你無法使用信用卡立即購買比特幣。然而，用一點耐心和創造性思維，你就不需要這樣。

以下是新用戶獲取比特幣的一些方法：

- 找一個有比特幣的朋友，直接向他買一些。許多比特幣用戶以這種方式開始。這種方法最簡單。與擁有比特幣的人見面的一種方式是參加在 [Meetup.com](#)列出的本地比特幣聚會。
- 使用分類服務，例如 [localbitcoins.com](#) 找到你所在地區的賣家以現金購買比特幣。
- 通過銷售產品或服務賺取比特幣。如果你是開發者，就賣你的編程技能。如果你是理髮師，就剪頭髮收比特幣。
- 使用比特幣ATM。比特幣ATM是一種接受現金並將比特幣發送到智能手機比特幣錢包的機器。使用 [Coin ATM Radar](#) 的在線地圖查找附近的比特幣ATM。
- 使用比特幣交易所。許多國家現在有交易所，為買賣雙方提供以當地貨幣交換比特幣的市場。Exchange-rate服務（例如 [BitcoinAverage](#)）可以顯示每種貨幣的比特幣交易所列表。

Tip

比特幣優於其他支付系統的一個優點是，如果使用得當，它可以為用戶提供更多的隱私。獲取，持有和支出比特幣並不要求你向第三方洩露敏感和個人身份訊息。但是，比特幣涉及諸如貨幣兌換等傳統系統時，國家和國際法規通常適用。為了以你的國家貨幣兌換比特幣，你通常需要提供身份證明和銀行訊息。用戶應該知道，一旦比特幣地址附加到身份，所有相關的比特幣交易也很容易識別和跟蹤。這是許多用戶選擇維護與他們的錢包不相關的專用交換賬戶的原因之一。

Alice是被通過朋友介紹知道比特幣的，因此她可以輕鬆獲得她的第一個比特幣。接下來，我們將看看她如何從她的朋友Joe那購買比特幣，以及Joe如何將比特幣發送到她的錢包。

## 查看比特幣的當前價格

在Alice可以從Joe那購買比特幣之前，他們必須同意比特幣和美元之間的匯率。這給那些比特幣新手帶來了一個共同的問題：“誰設定的比特幣價格？”簡而言之，價格是由市場決定的。

像大多數其他貨幣一樣，比特幣具有浮動匯率，這意味著比特幣相對於任何其他貨幣的價值根據其交易市場的供求情況而變化。例如，比特幣的美元價格是根據最近比特幣和美元的交易計算出來的。因此，價格每秒鐘會出現幾次波動。定價服務將彙總來自多個市場的價格並計算代表貨幣對的廣泛市場匯率（例如BTC / USD）的成交量加權平均值。

有數百個應用程式和網站可以提供當前的市場價格。這裡是一些最流行的：

#### ***Bitcoin Average***

一個提供每種貨幣的成交量加權平均值簡單視圖的網站。

#### ***CoinCap***

這項服務列出了數百種密碼貨幣（包括比特幣）的市值和匯率

#### ***Chicago Mercantile Exchange Bitcoin Reference Rate***

可用於機構和合同參考的參考利率，作為CME的一部分投資數據源。

除了這些網站和應用程式之外，大多數比特幣錢包會自動將比特幣和其他貨幣進行轉換。在將比特幣發送給Alice之前，Joe會使用他的錢包自動轉換價格。

## 發送和接收比特幣

愛麗絲決定兌換10美元的比特幣，以免在這項新技術上冒太多風險。她給了Joe 10美元現金，打開她的Mycelium錢包應用程式，並選擇Receive。這顯示了Alice的第一個比特幣地址的QR碼。

Joe在他的智能手機錢包上選擇“Send”，然後看到包含兩個輸入的界面：

- 目標比特幣地址
- 要發送的數量，以BTC或者他的本地貨幣（USD）為單位。

在比特幣地址的輸入欄位中，有一個看起來像二維碼的小圖標。這使得Joe可以用他的手機攝像頭掃描條碼，這樣他就不必輸入Alice的比特幣地址，這個地址很長很難敲。Joe點擊二維碼圖標激活智能手機攝像頭，掃描Alice手機上顯示的二維碼。

Joe現在已經將Alice的比特幣地址設置為收件人了。Joe輸入金額為10美元，他的錢包通過訪問在線服務的最新匯率來轉換它。當時的匯率是每比特幣100美元，所以10美元價值0.10比特幣（BTC）或100毫比特幣（mBTC），如Joe的錢包截圖所示 (see [Airbitz mobile bitcoin wallet send screen](#)).

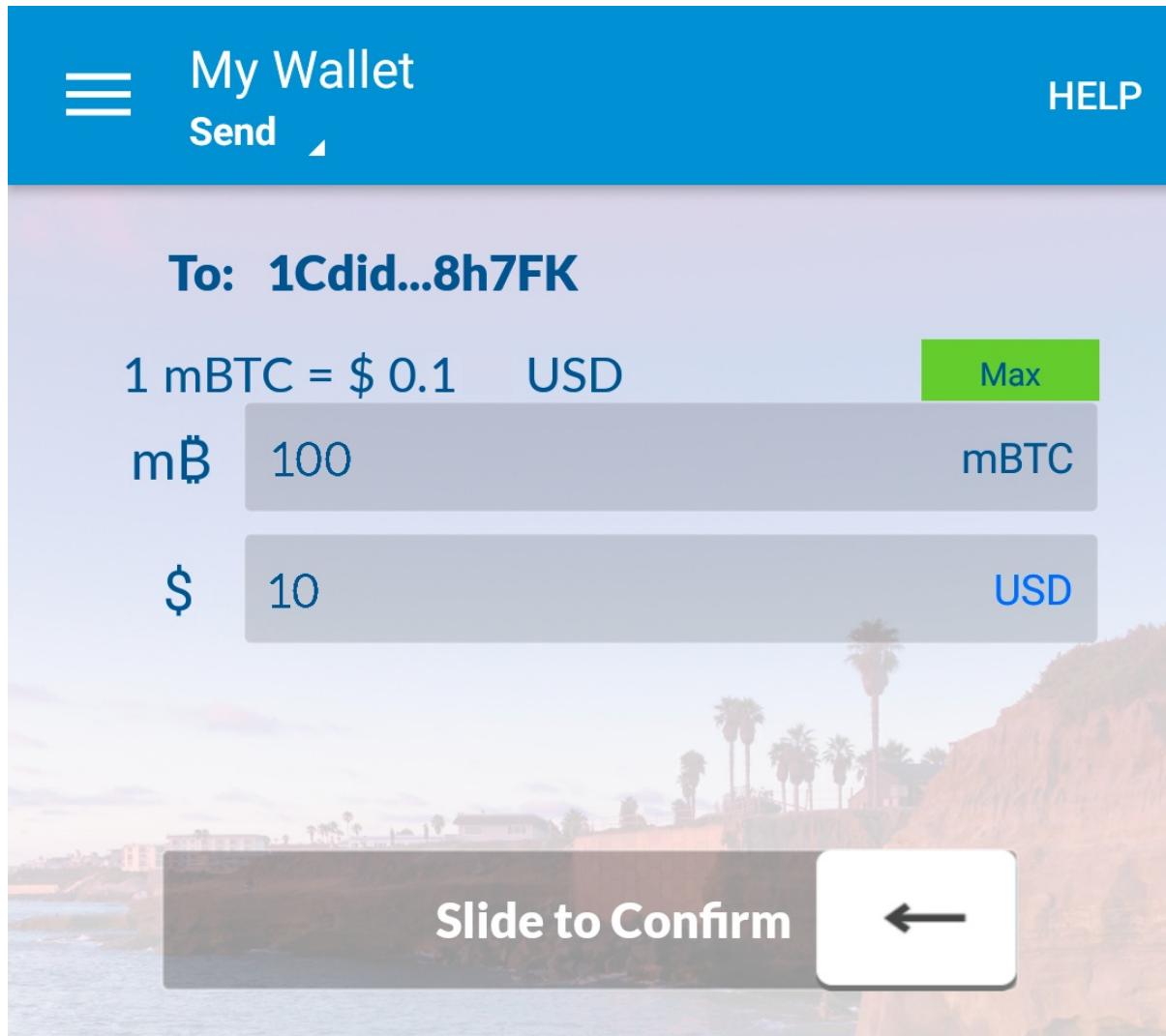


Figure 2. Airbitz mobile bitcoin wallet send screen

然後Joe仔細檢查以確保他輸入了正確的金額，因為他即將轉賬，錯誤不可逆轉。在仔細檢查地址和金額後，他按下Send來傳輸交易。Joe的比特幣錢包構建了一筆交易，將0.10BTC發送到Alice的地址，從Joe的錢包中獲取資金並使用Joe的私鑰簽署交易。這告訴比特幣網路，喬已經授權將價值轉移給Alice的新地址。由於交易是通過點對點協議傳輸的，因此它可以快速傳播到比特幣網路。在不到一秒的時間內，網路中大多數連接良好的節點都會收到交易並首次查看Alice的地址。

與此同時，Alice的錢包不斷“監聽”比特幣網路上的已發佈交易，尋找與她的錢包中的地址相匹配的任何交易。在Joe的錢包傳輸交易後幾秒鐘，Alice的錢包就會顯示它正在接收0.10BTC。

#### 確認

起初，Alice的地址將顯示Joe的交易為“未確認”。這意味著交易已經傳播到網路，但尚未記錄在比特幣交易賬本（即區塊鏈）中。要確認，交易必須包含在一個區塊中，並添加到區塊鏈中，平均每10分鐘發生一次。在傳統的財務術語中，這被稱為清算。有關比特幣交易的傳播，驗證和清算（確認）的更多詳細訊息，請參閱“採礦”。

Alice現在是那0.10BTC的所有者了。在下一章中，我們將看到她第一次使用比特幣購買東西，並更詳細地研究背後的交易和傳播技術。

---

1. "Bitcoin: A Peer-to-Peer Electronic Cash System," Satoshi Nakamoto (<https://bitcoin.org/bitcoin.pdf>).



# 比特幣如何運轉

## 交易，區塊，挖礦和區塊鏈

與傳統的銀行和支付系統不同，比特幣系統基於去中心化的信任。在比特幣中，信任是比特幣系統中不同參與者的交互的湧現特性達成的。在本章中，我們將站在比較高的視角研究比特幣，通過在比特幣系統中跟蹤一筆交易，看到它被比特幣分佈式共識機制所信任和接受，並最終記錄在區塊鏈中（所有交易的分佈式賬本）。後續章節將深入探討交易，網路和挖礦背後的技術。

## 比特幣概覽

在[Bitcoin overview](#)的概覽圖中，我們看到比特幣系統由包含密鑰的錢包，通過網路傳播的交易，以及產生（通過競爭性計算）共識區塊鏈的礦工組成，區塊鏈是所有交易的權威帳本。

本章中的每個示例均基於在比特幣網路上進行的實際交易，通過從一個錢包向另一個錢包發送資金來模擬用戶（Joe，Alice，Bob和Gopesh）之間的交互。在通過比特幣網路跟蹤交易到區塊鏈的同時，我們將使用[blockchain explorer](#)網站可視化每個步驟。區塊鏈瀏覽器（blockchain explorer）是一個作為比特幣搜索引擎運行的Web應用，它允許你搜索地址，交易和區塊，並查看它們之間的關係和流程。

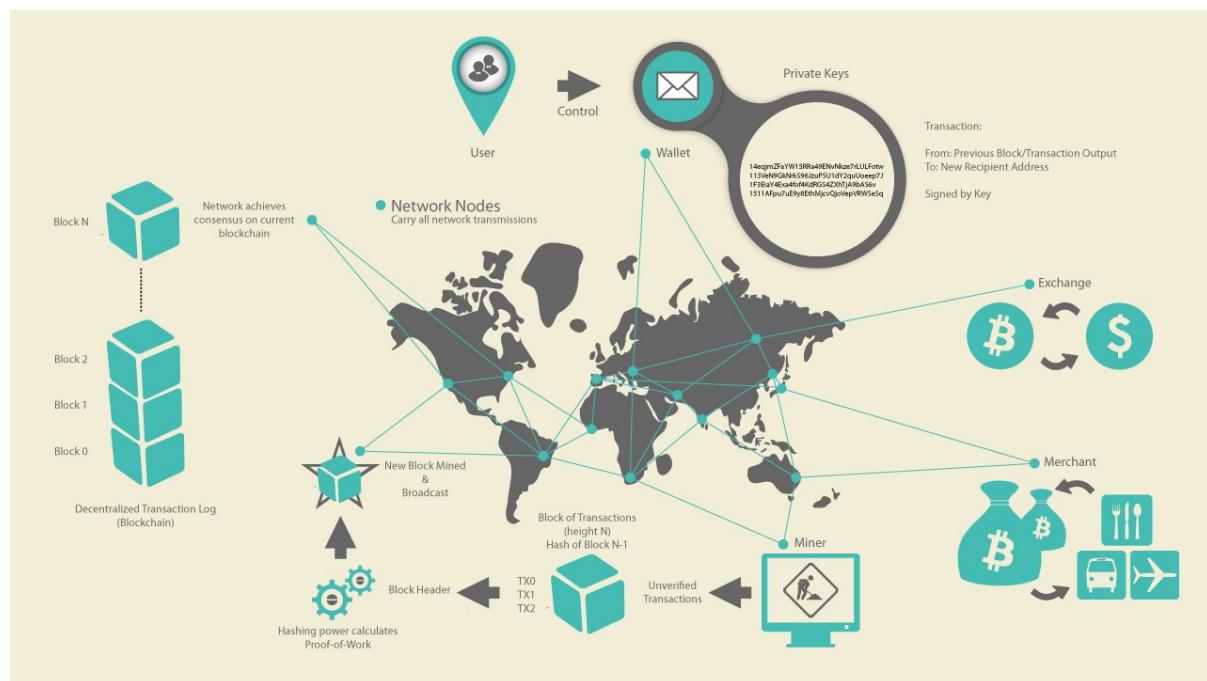


Figure 1. Bitcoin overview

流行的區塊鏈瀏覽器包括：

- [BlockCypher Explorer](#)
- [blockchain.info](#)
- [BitPay Insight](#)

其中每一個都有搜索功能，可以採用比特幣地址，交易雜湊值，區塊號或區塊雜湊值，從比特幣網路中檢索相應的訊息。對於每個交易或區塊示例，我們將提供一個URL，以便你可以自己查看並詳細研究它。

## 購買一杯咖啡

上一章介紹的Alice是剛剛獲得了第一個比特幣的新用戶。在[getting\_first\_bitcoin]中，Alice會見了她的朋友Joe，用現金交換了一些比特幣。Joe創造的交易把0.10BTC發送到了Alice的錢包。現在，Alice將進行她的第一筆零售交易，在加利福尼亞州帕洛阿爾託的Bob咖啡店購買一杯咖啡。

Bob咖啡最近開始接受比特幣支付，在其銷售系統中增加了一個比特幣選項。Bob咖啡的價格以當地貨幣（美元）列出，但在支付時，顧客可以選擇美元或比特幣。Alice下了一杯咖啡的訂單，Bob將它輸入到系統中，就像處理所有交易一樣。銷售系統自動將總價從美元轉換為比特幣，並以當前市場匯率以兩種貨幣進行顯示：

```
Total:  
$1.50 USD  
0.015 BTC
```

Bob說，“1.5美元，或者15毫比特幣”

Bob的銷售系統也會自動創建一個包含支付請求 (*payment request*) 的特殊二維碼（參見Payment request QR code）。

與僅包含目標比特幣地址的二維碼不同，支付請求是URL的二維碼，其包含目標地址，付款金額以及諸如“Bob's Cafe”的描述。這允許比特幣錢包應用預先填充用於發送付款的訊息，同時向用戶顯示可讀的訊息。你可以使用比特幣錢包應用掃描二維碼，以查看Alice會看到的內容。



Figure 2. Payment request QR code

Tip	嘗試用錢包掃描以查看地址和金額，但不要發送金錢。
-----	--------------------------

支付請求對BIP-21中定義的如下的URL進行編碼：

```
bitcoin:1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA?  
amount=0.015&  
label=Bob%27s%20Cafe&  
message=Purchase%20at%20Bob%27s%20Cafe
```

URL的組成部分

```
接收比特幣的地址："1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"  
支付金額："0.015"  
收件人地址的標籤："Bob's Cafe"  
支付詳情："Purchase at Bob's Cafe"
```

Alice用她的手機掃描屏幕上的二維碼。她的手機顯示支付 0.0150 BTC 至 Bob's Cafe，她選擇“發送”以授權付款。在幾秒鐘內（大約與信用卡授權相同的時間），Bob在他的系統中看到交易，交易就完成了。

## Note

比特幣網路可以以小數值進行交易，例如從millibitcoin（比特幣的1/1000）到聰（比特幣的1 / 100,000,000）。在本書中，我們將使用術語“比特幣”來表示從最小單位（1 satoshi）到將要開採的所有比特幣總數（21,000,000）的任何數量的比特幣。

你可以使用區塊鏈瀏覽器網站（[查看Alice的交易 blockchain.info](#)）在區塊鏈中檢查Alice對Bob's Cafe的交易：

Example 1. 查看Alice的交易 [blockchain.info](#)

```
https://blockchain.info/tx/0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286  
c345c2f2
```

## 比特幣交易

簡而言之，一筆交易告訴網路，一些比特幣價值的所有者已經授權將該價值轉移給另一個所有者。新的所有者現在可以通過創建另一個授權轉讓給另一個所有者的交易來支付比特幣，等等。

### 交易的輸入和輸出

交易就像複式簿記帳中的行一樣。每筆交易包含一個或多個“輸入”，就像比特幣賬戶的“借方”，在交易的另一端，有一個或多個“輸出”，就像比特幣賬戶中的“貸方”一樣。輸入和輸出（借方和貸方）加起來不一定數額相同。相反，輸出加起來略少於輸入，差額代表隱含的交易費用，這是由礦工收取的一筆小額付款，該礦工將交易加入到賬本中。比特幣交易在[Transaction as double-entry bookkeeping](#)中顯示為賬本中的記錄條目。

該交易還包含每個正在被花費的比特幣（輸入）的所有權證明，以所有者的數字簽名的形式出現，任何人都可以獨立驗證。用比特幣的術語來說，“花費”正在簽署一項交易，它將來自前一筆交易的價值轉移給由比特幣地址標識的新所有者。

Transaction as Double-Entry Bookkeeping			
Inputs	Value	Outputs	Value
Input 1	0.10 BTC	Output 1	0.10 BTC
Input 2	0.20 BTC	Output 2	0.20 BTC
Input 3	0.10 BTC	Output 3	0.20 BTC
Input 4	0.15 BTC		
Total Inputs:	0.55 BTC	Total Outputs:	0.50 BTC
<i>Inputs</i>	<i>0.55 BTC</i>		
<i>Outputs</i>	<i>0.50 BTC</i>		
<i>Difference</i>	<i>0.05 BTC (implied transaction fee)</i>		

Figure 3. Transaction as double-entry bookkeeping

## 交易鏈

Alice向Bob's Cafe的付款使用先前交易的輸出作為其輸入。在上一章中，Alice從她的朋友Joe那裡用現金換取了比特幣。該交易創建了一個由Alice的密鑰鎖定的比特幣值。她向Bob's Cafe的新交易引用之前的交易作為輸入，並創造新的輸出來支付咖啡錢並接收找零。交易形成一個鏈，最近一次交易的輸入與之前交易的輸出相對應。Alice的密鑰提供瞭解鎖先前那些交易輸出的簽名，從而向比特幣網路證明她擁有資金。她將咖啡的錢支付給Bob的地址，從而“留置”輸出，要求Bob必須簽名才能花費這筆金額。這代表了Alice和Bob之間的價值轉移。這個從Joe到Alice到Bob的交易鏈在 [A chain of transactions, where the output of one transaction is the input of the next transaction](#) 中進行了說明。

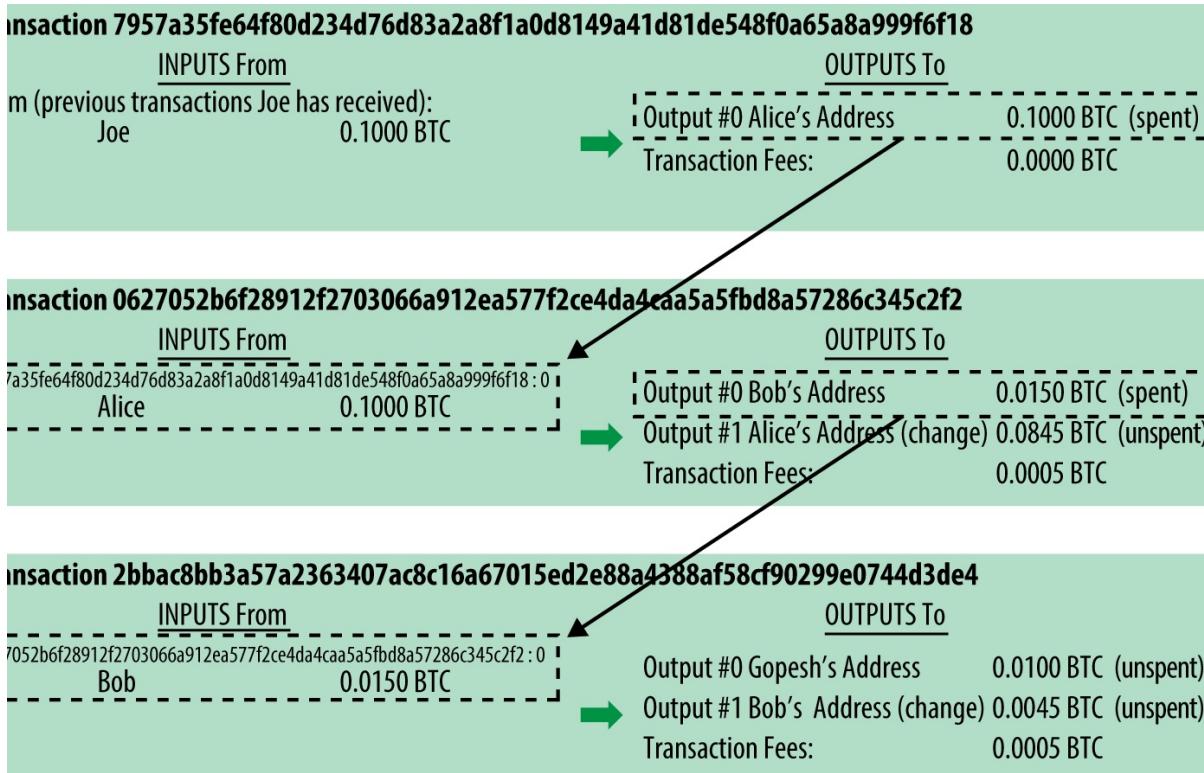


Figure 4. A chain of transactions, where the output of one transaction is the input of the next transaction

## 找零

許多比特幣交易的輸出既引用新所有者的地址，又引用當前所有者的地址（這稱為找零地址）。這是因為交易輸入（像鈔票一樣）不能分開。如果你在商店購買價值5美元的物品，但使用20美元的美元賬單來支付該物品，你將獲得15美元的找零。相同的概念適用於比特幣交易的輸入。如果你購買的產品需要5比特幣，但只有20比特幣的輸入能使用，你可以將一個5比特幣的輸出發送給店主，並將一個15比特幣輸出作為找零（減去涉及的交易費用）。重要的是，找零地址不必與輸入地址相同，並且出於隱私方面考慮，通常是來自所有者錢包的新地址。

在彙集輸入以執行用戶的支付請求時，不同的錢包可以使用不同的策略。他們可能會彙集很多小的輸入，或者使用等於或大於期望付款的輸入。除非錢包能夠按照付款和交易費用的總額精確彙集輸入，否則錢包將需要產生一些零錢。這與人們處理現金非常相似。如果你總是使用口袋裡最大的鈔票，那麼最終你會得到一個充滿零錢的口袋。如果你只使用零錢，你將永遠只有大額賬單。人們潛意識地在這兩個極端之間尋找平衡點，比特幣錢包開發者努力編程實現這種平衡。

總之，交易將交易的輸入的值移至交易的輸出。輸入是對前一個事務輸出的引用，表示值來自哪裡。交易輸出將特定值指向新所有者的比特幣地址，並且可以將零錢輸出給原始所有者。來自一個交易的輸出可以用作新交易的輸入，因此當價值從一個所有者轉移到另一個所有者時會產生一個所有權鏈（參見 [A chain of transactions, where the output of one transaction is the input of the next transaction](#)）。

## 常見交易形式

最常見的交易形式是從一個地址到另一個地址的簡單支付，通常包括一些“零錢”返回到原始所有者。這類交易有一個輸入和兩個輸出，參見 [Most common transaction](#)：

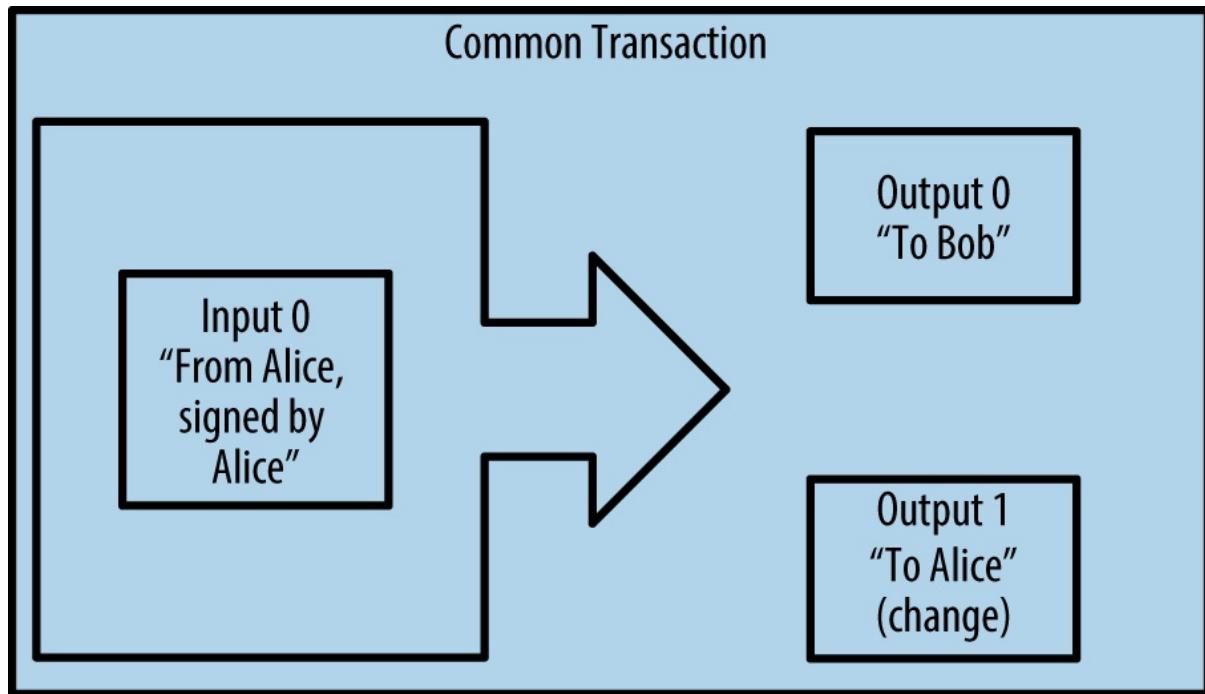


Figure 5. Most common transaction

另一種常見形式是彙集多個輸入到一個輸出的交易 (參見 [Transaction aggregating funds](#))。這類似於現實世界中將一堆硬幣和紙幣換成單一較大面值的紙幣的情況。此類交易有時由錢包應用生成，以清理收到的大量小額零錢。

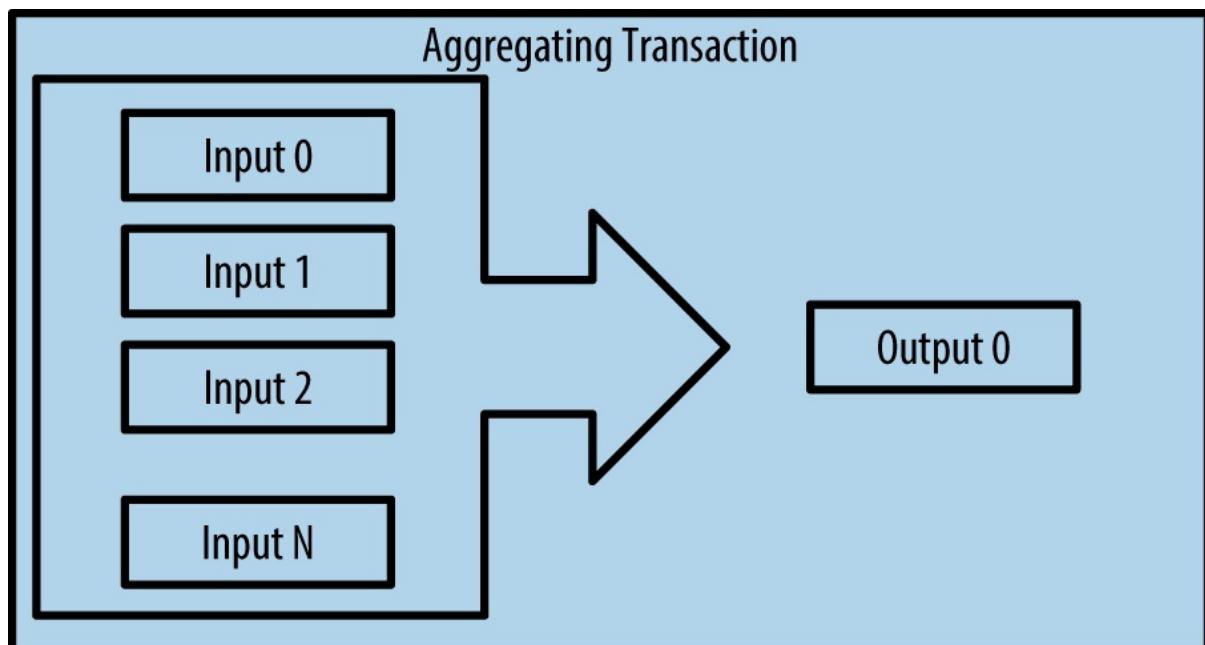


Figure 6. Transaction aggregating funds

最後，比特幣賬本中經常出現的另一種交易形式是將一個輸入分配給代表多個收款人的多個輸出的交易 (參見 [Transaction distributing funds](#))。這類交易有時被企業用來分配資金，例如在向多個僱員支付工資時。

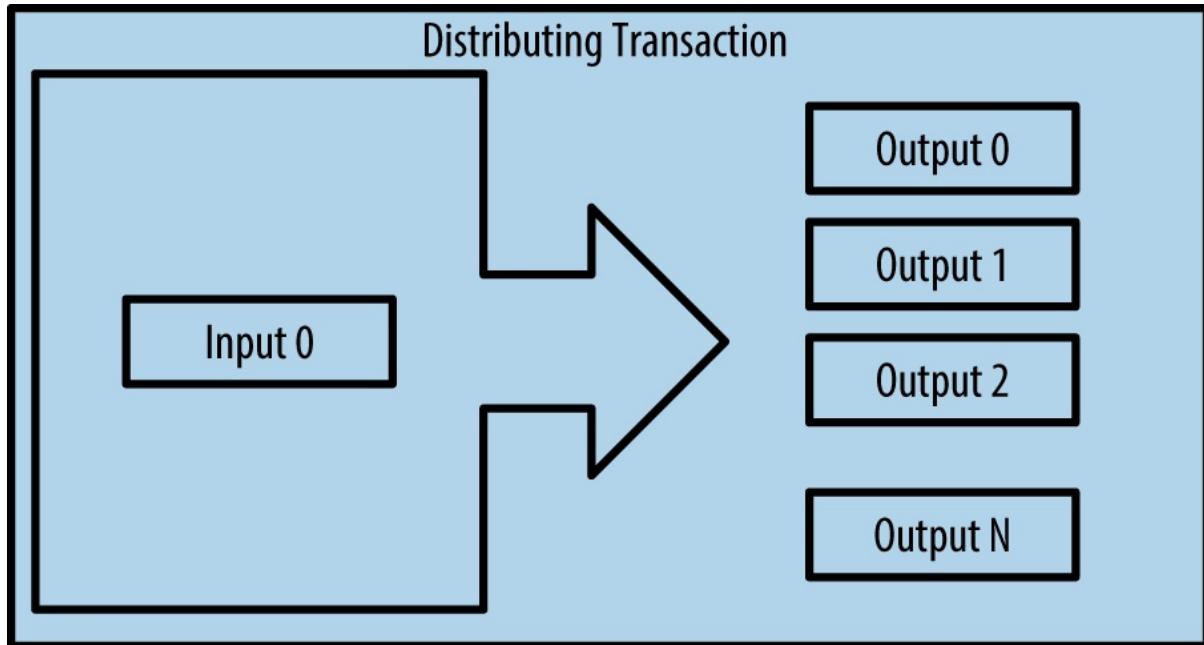


Figure 7. Transaction distributing funds

## 創建一筆交易

Alice的錢包應用包含了選擇合適的輸入和輸出的所有邏輯，根據Alice的具體設定創建交易。Alice只需要指定目的地和金額，剩下的事情交給錢包應用，Alice不用關心細節。重要的是，即使錢包應用完全脫機，錢包應用也可以創建交易。就像在家裡寫一張支票，然後通過信封發送給銀行一樣，交易不要求在連接到比特幣網路時進行創建和簽署。

### 獲得正確的輸入

Alice的錢包應用首先必須找到可以支付她想要發送給Bob的金額的輸入。大多數錢包跟蹤屬於錢包中地址的所有可用輸出。因此，Alice的錢包將包含Joe的交易輸出的副本，該交易是由現金交換創建的（參見[\[getting\\_first\\_bitcoin\]](#)）。作為完整節點客戶端運行的比特幣錢包應用實際上包含區塊鏈中每筆交易的未使用輸出的副本。這允許錢包創建交易輸入，以及快速驗證傳入的交易具有正確的輸入。但是，由於全節點客戶端佔用大量硬碟空間，所以大多數用戶錢包運行“輕量級”客戶端，僅跟蹤用戶自己未使用的輸出。

如果錢包應用未保存未花費的交易的輸出的副本，它可以用不同提供商提供的各種API，查詢比特幣網路，詢問完整節點來檢索該訊息。[Look up all the unspent outputs for Alice's bitcoin address](#)展示了API請求，向特定的URL發起HTTP GET請求。該URL將返回這個地址上所有未使用的交易的輸出，為應用提供構建交易輸入的訊息。我們使用簡單的命令行HTTP客戶端curl來請求。

Example 2. Look up all the unspent outputs for Alice's bitcoin address

```
$ curl https://blockchain.info/unspent?active=1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK
```

```
{
  "unspent_outputs": [
    {
      "tx_hash": "186f9f998a5...2836dd734d2804fe65fa35779",
      "tx_index": 104810202,
      "tx_output_n": 0,
      "script": "76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",
      "value": 10000000,
      "value_hex": "00989680",
      "confirmations": 0
    }
  ]
}
```

```

    }
}
```

Look up all the unspent outputs for Alice's bitcoin address 中的響應展示了在Alice的地址

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK 下有一筆未花費的輸出。響應內容包括包含這筆輸出的交易的引用，以及它的價值，1000萬（單位是聰），相當於0.10比特幣，利用這些訊息，Alice的錢包應用可以構建一個交易，將該值轉移到新的所有者地址。

Tip

查看 transaction from Joe to Alice.

如你所見，愛麗絲的錢包包含支付一杯咖啡的足夠的比特幣。否則，Alice的錢包應用可能需要“翻遍”一堆較小的未使用的輸出，就像從錢包中找硬幣一樣，直到它能夠找到足夠的錢來支付咖啡。在這兩種情況下，可能都需要進行一些找零，我們將在下一部分中看到，錢包應用創建交易輸出（付款）。

## 創建輸出

交易的輸出是以腳本形式創建的，該腳本在比特幣價值上創建了一個“留置”，只能通過提供腳本解決方案來進行提取。簡而言之，Alice的交易輸出將包含一個腳本，其內容如下：“這筆支出屬於能使用Bob的公共地址對應的私鑰進行簽名的人。”因為只有Bob擁有與該地址對應的私鑰，所以只有Bob的錢包可以提供這樣的簽名來提取該輸出。因此，Alice可以通過要求Bob的簽名，來“限制”這筆輸出的使用。

這筆交易還包括第二筆輸出，因為愛麗絲的資金為0.10BTC，對於0.015BTC的咖啡來說太多了，需要找零0.085BTC。Alice的找零付款由Alice的錢包創建，作為Bob的付款的同一筆交易中的輸出。愛麗絲的錢包將其資金分成兩筆付款：一筆給Bob，一份給自己。然後，她可以在後續交易中使用（花費）這次找零的輸出。

最後，為了讓網路及時處理這筆交易，Alice的錢包應用將增加一筆小額費用。這在交易中並不明確；這是由輸入和輸出的差值隱形包含的。如果Alice不創建0.085的找零，而是0.0845，就會剩下0.0005BTC（半毫比特幣）。輸入的0.10BTC沒有完全用於兩個輸出，因為它們的總和小於0.10。由此產生的差值就是礦工收取的交易費用，用於驗證交易並將交易包括到區塊鏈中。

生成的交易可以使用區塊鏈瀏覽器查看，如Alice's transaction to Bob's Cafe所示。

## Transaction View information about a bitcoin transaction

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286c345c2f2

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK (0.1 BTC - Output)



1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA

- (Unspent) 0.015 BTC

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK -

(Unspent) 0.0845 BTC

97 Confirmations

0.0995 BTC

### Summary

Size 258 (bytes)

Received Time 2013-12-27 23:03:05

Included In 277316 (2013-12-27 23:11:54 +9  
Blocks minutes)

### Inputs and Outputs

Total Input 0.1 BTC

Total Output 0.0995 BTC

Fees 0.0005 BTC

Estimated BTC Transacted 0.015 BTC

Figure 8. Alice's transaction to Bob's Cafe

Tip

[查看 transaction from Alice to Bob's Cafe.](#)

## 將交易加入帳本

Alice的錢包應用創建的交易長度為258個字節，包含確認資金所有權和分配新的所有者所需的所有內容。現在，交易必須傳輸到比特幣網路，併成為區塊鏈的一部分。在下一節中，我們將看到交易如何成為新區塊的一部分，以及區塊如何被“挖掘”。最後，我們將看到當區塊加入區塊鏈後，會隨著區塊的增加越來越被網路信任。

### 傳輸交易

交易包含了處理所需的所有訊息，因此傳送到比特幣網路的方式或位置無關緊要。比特幣網路是一個點對點網路，每個比特幣客戶端通過連接到其他幾個比特幣客戶端來參與。比特幣網路的目的是向所有參與者傳播交易和區塊。

### 如何傳播

任何遵守比特幣協議，加入到比特幣網路的系統，如同服務器，桌面應用程式或錢包，都稱為比特幣節點 (*bitcoin node*)。Alice的錢包應用可以通過任何類型的連接（有線，WiFi，移動等）將相關交易發送到任何比特幣節點。她的比特幣錢包不必直接連接到Bob的比特幣錢包，她不必使用咖啡館提供的互聯網連接，但這兩種選擇都是可能的。任何比特幣節點接收到一個它沒見過的有效交易之後，會立即轉發到它連接到的所有其他節點，這被稱為泛洪 (*flooding*) 傳播技術。因此，事務在點對點網路中迅速傳播，可在幾秒鐘內達到大部分節點。

### Bob的視角

如果Bob的比特幣錢包應用直接連接到Alice的錢包應用，則Bob的錢包應用可能是第一個接收到該交易的節點。即使Alice的錢包通過其他節點發送交易，它也會在幾秒鐘內到達Bob的錢包。Bob的錢包會立即將Alice的交易識別為收款，因為它包含可由Bob的私鑰提取的輸出。Bob的錢包應用還可以獨立驗證交易數據是格式正確的，使用的是之前未花費的輸入，並且包含足夠的交易費用以包含在下一個區塊中。此時，鮑勃可以認為風險很小，即交易將很快包含在一個區塊中並得到確認。

Tip

關於比特幣交易的一個常見誤解是，它們必須等待10分鐘新區塊的產生才能被“確認”，或者最多60分鐘才能完成6個確認。雖然確認確保交易已被整個網路所接受，但對於諸如一杯咖啡等小值物品，這種延遲是不必要的。商家可以接受沒有確認的有效小額交易。沒有比沒有身份或簽名的信用卡支付風險更大的了，商家現在也經常接受。

## 比特幣挖礦

Alice的交易現在已經傳播到比特幣網路上了。但在它被驗證並經歷一個名為挖礦 (*mining*) 的過程包含在區塊中之前，不會成為區塊鏈的一部分。有關詳細說明，請參閱[\[mining\]](#)。

比特幣的信任系統基於計算。交易被捆綁到區塊中，這需要大量的計算來提供工作證明，但只需少量的計算進行驗證。挖礦過程在比特幣中有兩個作用：

- 挖礦節點通過遵從比特幣的共識規則來驗證所有交易。因此，挖礦通過拒絕無效或格式錯誤的交易來為比特幣交易提供安全保障。
- 每個區塊被挖出時會創造新的比特幣，就像中央銀行印錢一樣。按照固定的發行時間表，每個區塊創建的比特幣數量是有限的，隨著時間的推移會逐漸減少。

挖礦在成本和回報之間達到了良好的平衡。挖礦用電解決數學問題。一位成功的礦工將通過新的比特幣和交易費的形式獲得一份獎勵。只有礦工正確地驗證了所有交易，並且符合共識的規則，才會獲得獎勵。這種微妙的平衡為沒有中央管理機構的比特幣提供了安全性。

描述挖礦的一種好的類比是數獨遊戲，這種大量競爭的遊戲，每次有人找到解決方案時都會重置，其難度會自動調整，因此需要大約10分鐘才能找到解決方案。想象一下，數以千計的行和列的巨大數獨謎題。如果我告訴你一個完整的謎題，你可以很快驗證它。但是，如果拼圖有幾個方格填充，其餘的都是空的，則需要花費大量工作來解決！數獨的難度

可以通過改變它的大小（更多或更少的行和列）來調整，但即使它非常大，它仍然可以很容易地被驗證。比特幣中使用的“謎題”基於密碼雜湊，具有相似的特徵：它不對稱，難以解決，但易於驗證，並且可以調整難度。

在 [user-stories] 中，我們介紹了 Jing，一個上海的企業家。Jing 經營著一個礦池，包含數千臺專業採礦電腦，爭奪獎勵。每 10 分鐘左右，Jing 的採礦電腦就會在全球競賽中與成千上萬的類似的系統競爭，尋找解決方案。為了找到解決方案，所謂的 \_ 工作量證明 (Proof-of-Work, PoW) ，比特幣網路需要每秒進行數千萬億 (quadrillions) 次雜湊運算。工作量證明的演算法涉及使用 SHA256 密碼演算法重複地對區塊的頭部數據和隨機數進行雜湊，直到出現與預定模式匹配的結果為止。找到這種解決方案的第一位礦工贏得一輪競爭，並將該區塊發佈到區塊鏈中。

Jing 於 2010 年開始使用一臺速度非常快的臺式電腦進行挖礦，以找到適用於新區塊的工作量證明 Proof-of Work。隨著越來越多的礦工加入比特幣網路，解題的難度迅速增加。很快，Jing 和其他礦工升級到更專用的硬體，如高端顯卡 (GPU)。

在撰寫本書時，難度已經大到需要採用專用集成電路 (ASIC) ，將數百種挖礦演算法印刷到硬體上，在單個硅片上並行運行。Jing 的公司也參與了一個礦池，這就像一個彩票池，允許參與者共享他們的算力和獎勵。Jing 的公司現在運營著一個倉庫，其中包含數千名 ASIC 礦工，每天 24 小時進行比特幣挖礦。該公司通過出售開採出來的比特幣來支付其電力成本，從利潤中獲取收入。

## 挖掘區塊中的交易

新的交易不斷從用戶錢包和其他應用流入網路。當被比特幣網路節點看到時，會被添加到由每個節點維護的未經驗證的臨時交易池中。隨著礦工構建一個新的區塊，他們將未驗證的交易從該池中取出添加到新的區塊，然後嘗試用挖礦演算法 (PoW) 來證明新區塊的有效性。挖礦的詳細過程請參見 [mining]。

交易添加到新的區塊後，根據交易費高低和其他一些條件按優先級排列。每個礦工通過網路收到前一個區塊時，便知道它已經輸掉了上一輪競爭，會開始挖掘新的區塊。他立即創建一個新區塊，填入交易數據和前一個區塊的指紋，並開始計算新區塊的 PoW。每個礦工在他的區塊中都包含一筆特殊交易，一筆支付給它自己的比特幣地址的獎勵（目前為 12.5 個新比特幣）加上該區塊中包含的所有交易的交易費用總和。如果他發現一個可以使這個區塊有效的解決方案，就會“獲得”這些獎勵，因為他成功挖掘的區塊被添加到全局區塊鏈中。他創建的這筆獎勵交易也變得可花費。加入採礦池的 Jing 建立了自己的軟體來創建新的區塊，將獎勵分配到礦池的地址，一部分獎勵將按照上一輪貢獻的工作量比例分配給 Jing 和其他礦工。

Alice 的交易首先被網路接收，並被包括在未經驗證的交易中。一旦被挖礦軟體驗證，它就被包含在一個叫做候選區塊的新區塊中（由 Jing 的礦池生成的）。參與該採礦池的所有礦工立即開始計算候選區塊的 PoW。在 Alice 的錢包傳輸交易後約五分鐘，Jing 的一位 ASIC 礦工找到了候選區塊的解決方案並將其發佈給網路。一旦其他礦工驗證了這個獲勝的區塊，他們將開始競爭挖掘下一個區塊。

Jing 挖到的區塊作為 #277316 區塊成為了區塊鏈的一部分，包含 419 筆交易，其中包括 Alice 的交易。Alice 的交易被包含到一個區塊中，視為該交易的一個“確認”。

Tip

查看包含 [Alice's transaction](#) 的區塊。

大約 19 分鐘後，另一個礦工開採出 #277317 區塊。由於這個新區塊建立在包含 Alice 交易的 #277316 區塊的頂部，因此它為區塊鏈增加了更多計算量，從而加強了對這些交易的信任。在包含交易的區塊的頂部開採的每個區塊都為 Alice 交易增加確認數。隨著區塊堆疊在一起，修改歷史交易變得極其困難，從而使其越來越受到網路的信任。

在圖 [Alice's transaction included in block #277316](#) 中，我們可以看到包含 Alice 的交易的 #277316 區塊。在它下面有 277,316 個區塊（包括區塊 #0），在區塊鏈 (blockchain) 中彼此鏈接，一直到區塊 #0，稱為創世區塊 (genesis block)。隨著時間的推移，隨著區塊的“高度”增加，每個區塊和整個鏈的計算難度也會增加。在包含 Alice 的交易的區塊之後開採的區塊作為進一步的保證，因為它們在更長的鏈中堆積更多的計算。按照慣例，任何具有多於六個確認的區塊都被認為是不可撤銷的，因為需要巨大的計算量來重新計算六個區塊。我們將在 [minig] 中更詳細地探討採礦過程及其建立信任的方式。

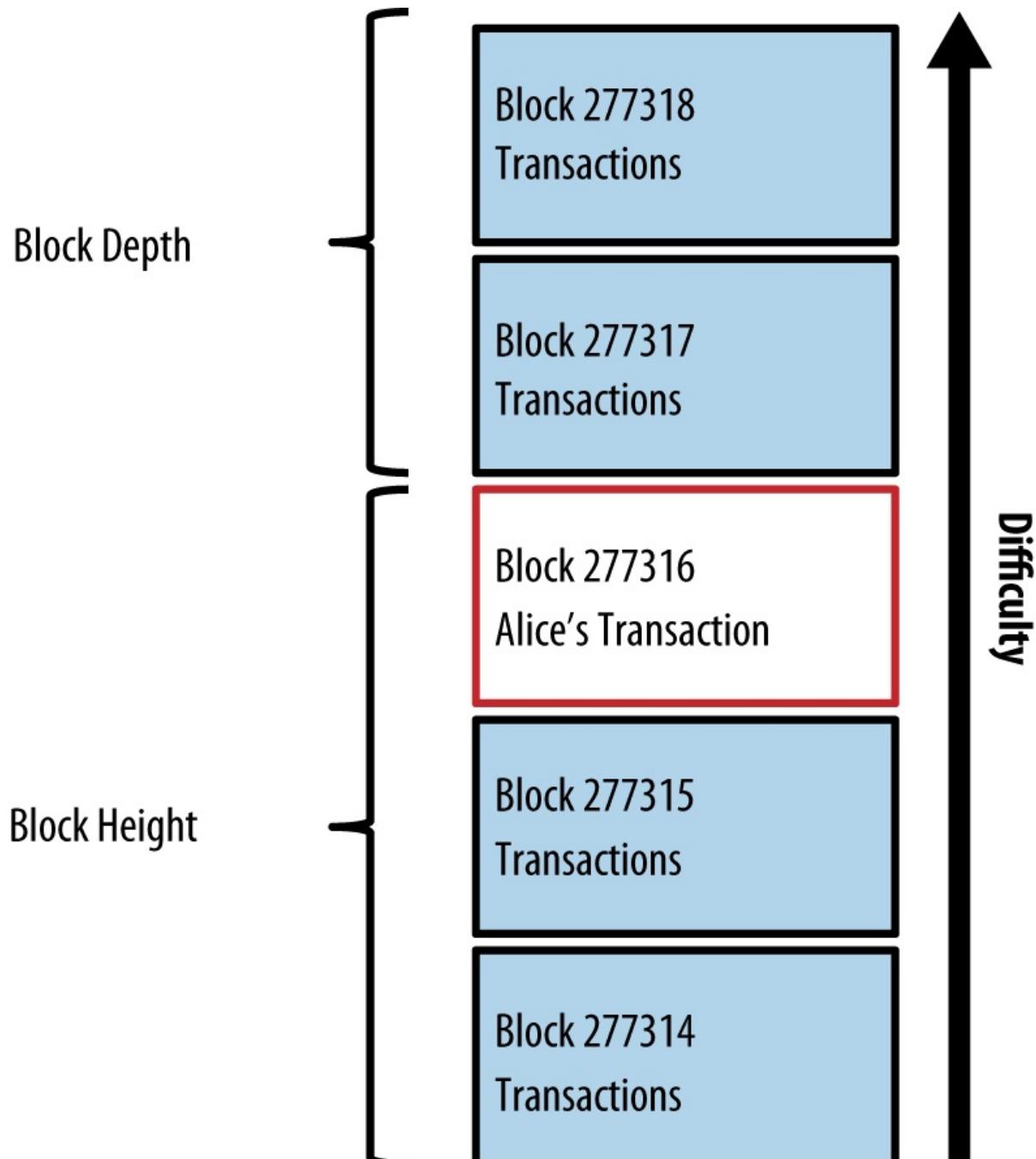


Figure 9. Alice's transaction included in block #277316

## 花費比特幣

既然愛麗絲的交易作為一個區塊的一部分嵌入在區塊鏈中，它就是比特幣分佈式賬本的一部分，並且對於所有的比特幣應用程式都是可見的。每個比特幣客戶端都可以獨立驗證該交易的有效性和可用性。完整節點客戶可以從比特幣首次在一個區塊中生成的那一刻開始追蹤資金來源，從一筆交易到另一筆交易，直到到達Bob的地址。輕量級客戶可以通過確認交易在區塊鏈中，計算其後又開採了多少個區塊，來做所謂的簡單支付驗證（參見[\[spv\\_nodes\]](#)），從而保證礦工接受它為有效的。

Bob現在可以花費這筆交易和其他交易的輸出了。例如，Bob可以通過將價值從Alice的咖啡支付轉移給新的所有者，支付費用給承包商或供應商。最有可能的是，Bob的比特幣軟體將許多小額付款合併為一筆更大的款項，例如將全天的比特幣彙集到一筆交易中。有關彙集交易，請參閱[Transaction aggregating funds](#)。

當Bob花費從Alice和其他客戶收到的款項時，他擴展了交易鏈。假設Bob向在班加羅爾的網頁設計師Gopesh支付了一個新頁面的設計費用。現在，交易鏈看起來像Alice's transaction as part of a transaction chain from Joe to Gopesh。

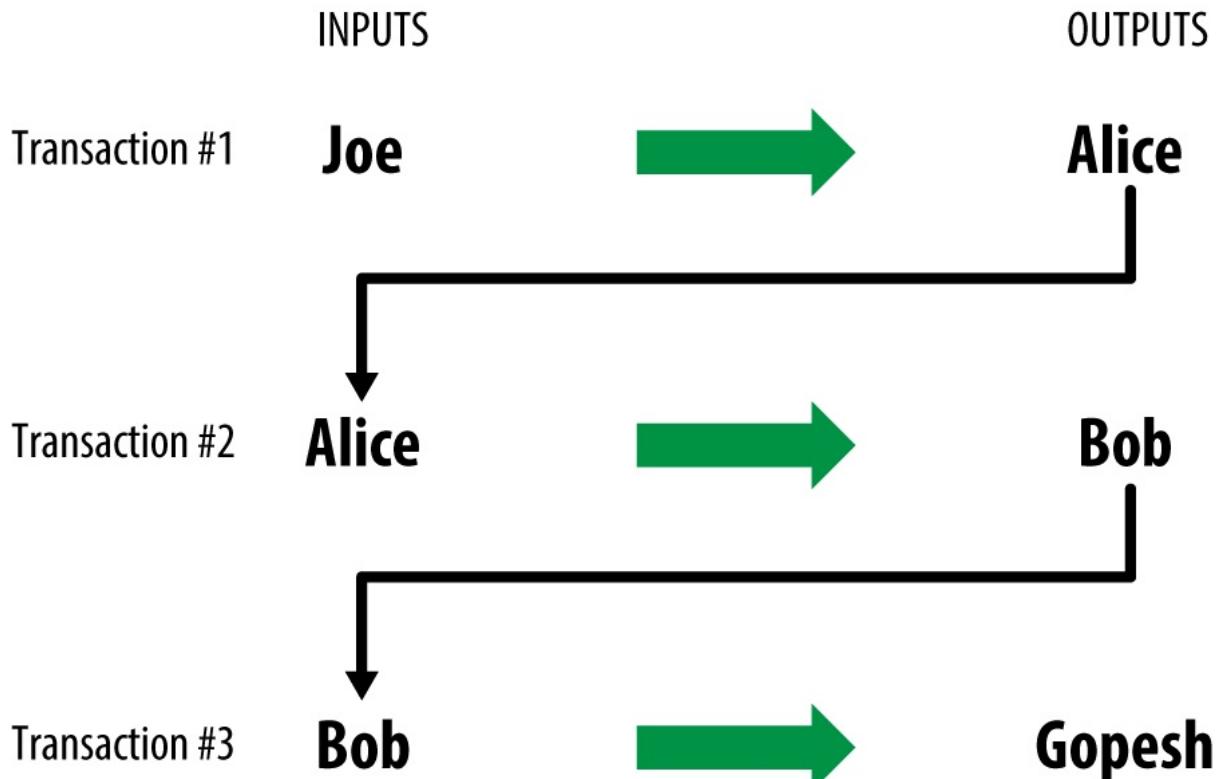


Figure 10. Alice's transaction as part of a transaction chain from Joe to Gopesh

在本章中，我們看到交易如何建立一個鏈條，將價值從一個所有者轉移到另一個所有者。我們還追蹤了Alice的交易，在她的錢包中創建，傳輸到比特幣網路，礦工將其記錄在區塊鏈上。在本書的其餘部分，我們將研究錢包，地址，簽名，交易，網路以及挖礦背後的具體技術。

## Bitcoin Core：參考實現

比特幣是開源的項目，源程式碼使用MIT授權方式，可以免費下載和使用。開源不僅意味著免費使用，也意味著比特幣是由開源志願者社區開發的。起初，這個社區只包括中本聰（Satoshi Nakamoto）。到2016年，比特幣的源程式碼擁有超過400個貢獻者，大約有十幾位開發人員幾乎全職工作，另外幾十人兼職工作。任何人都可以貢獻程式碼，包括你！

當中本聰創造比特幣時，該軟體實際上已經在白皮書出現之前完成。在寫這篇文章之前，中本聰想確保它可以工作。第一版實現已經進行了大量修改和改進，已經發展成為所謂的*Bitcoin Core (Bitcoin Core)*，以區別於其他兼容的實現。Bitcoin Core是比特幣系統的參考實現，這意味著它是關於如何實施每一部分技術的權威參考。Bitcoin Core實現了比特幣的各個方面，包括錢包，交易和區塊驗證引擎，以及點對點比特幣網路中的完整網路節點。

## Warning

儘管Bitcoin Core包含錢包的參考實現，但並不建議將其用作用戶或應用程式生產環境中的錢包。建議應用程式開發人員使用現代標準（如BIP-39和BIP-32）開發錢包（請參閱[\[mnemonic\\_code\\_words\]](#)和[\[hd\\_wallets\]](#)）。BIP代表Bitcoin改進建議。

Bitcoin Core architecture (Source: Eric Lombrozo) 展示了Bitcoin Core的架構。

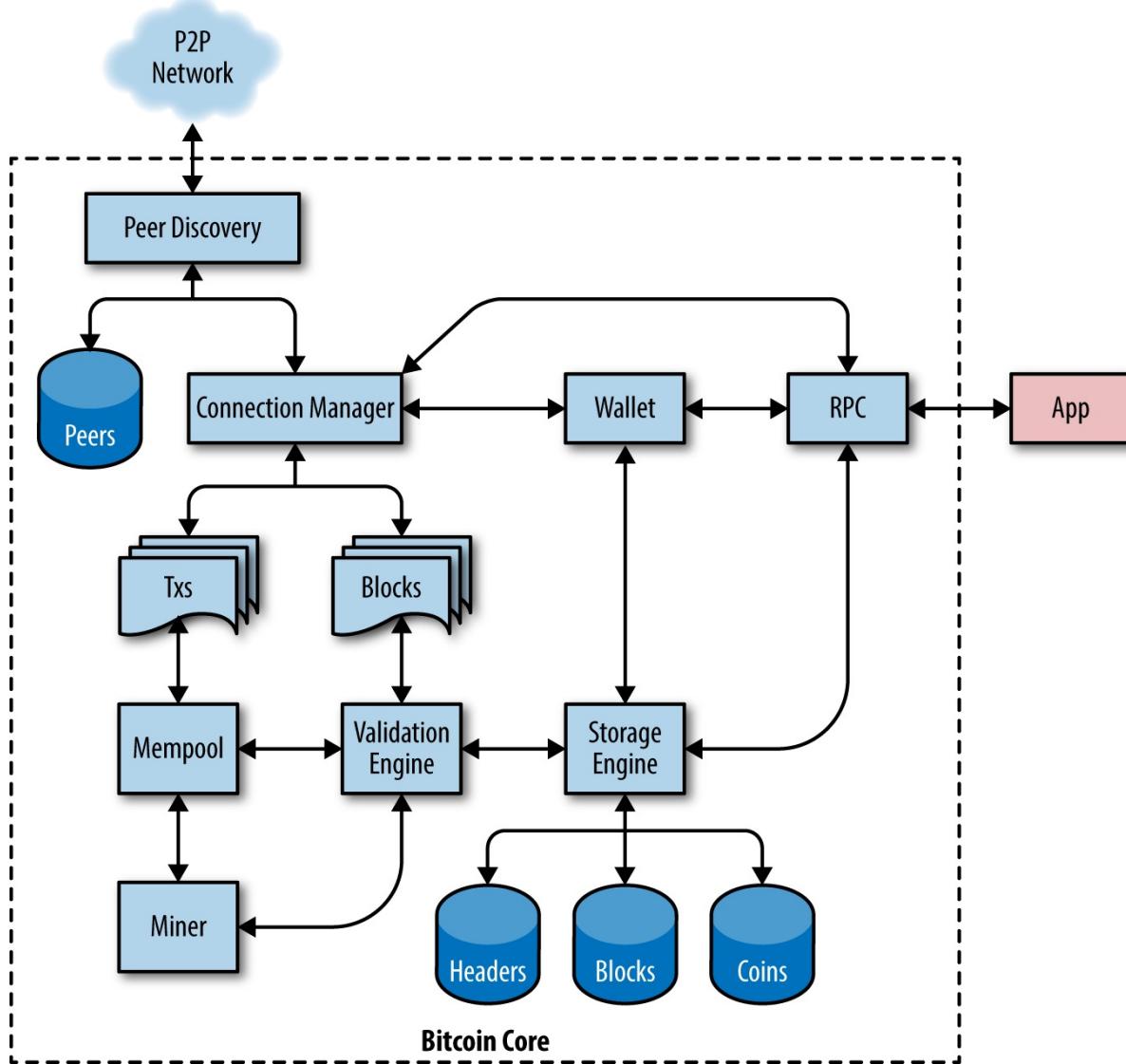


Figure 1. Bitcoin Core architecture (Source: Eric Lombrozo)

## 比特幣開發環境

如果你是一名開發者，你需要建立一個開發環境，其中包含用於編寫比特幣應用程式的所有工具，庫和支持軟體。在這個高度技術性的章節中，我們將一步一步地介紹該過程。如果覺得過於複雜（並且你實際上沒有設置開發環境），請隨意跳到下一章，技術性較弱的章節。

### 通過源程式碼編譯Bitcoin Core

可以從Github下載Bitcoin Core的源程式碼壓縮包或克隆項目。例如，在 [Bitcoin Core download page](#) 上，選擇最新版本的源碼壓縮包，`bitcoin-0.15.0.2.tar.gz`。或者，使用`git clone`命令在本地創建一個備份。[GitHub bitcoin page](#).

Tip

在本章的許多示例中，我們將使用操作系統的命令行界面（也稱為“shell”），通過“終端”應用程式訪問它，shell將顯示一個提示符；你輸入一個命令；shell會為你的下一個命令返回一些文本和一個新的提示符。提示可能在你的系統上看起來不同，但在以下示例中，它由 \$ 號表示。在示例中，當你在 \$ 符號後面看到文本時，請勿鍵入 \$ 符號，而是在其後面緊接著輸入命令，然後按Enter執行該命令。在示例中，每條命令下面的行是操作系統對該命令的響應。當你看到下一個 \$ 前綴時，你應該知道這是一個新的命令行，你可以重複這個過程。

在這個例子中，我們使用 `git` 命令創建源程式碼的本地副本。

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

Tip

Git是使用最廣泛的分佈式版本控制系統，它是軟體開發人員工具箱的重要組成部分。如果你尚未安裝，請在操作系統上安裝 `git` 命令或git的圖形用戶界面。

當git克隆操作完成後，你將在`bitcoin`目錄中擁有完整的源程式碼庫本地副本。在提示符處輸入 `cd bitcoin` 切換到此目錄：

```
$ cd bitcoin
```

### 選擇Bitcoin Core的發行版

預設情況下，本地副本將同步最新的程式碼，這可能是比特幣的不穩定版或beta版。編譯程式碼之前，通過檢查`tag`來選擇特定版本。這將使本地副本與由關鍵字標記標識的程式碼儲存庫的特定快照同步。開發人員使用標籤通過版本號標記程式碼的特定版本。首先，為了找到可用的標籤，我們使用 `git tag` 命令：

```
$ git tag
v0.1.5
v0.1.6test1
v0.10.0
...
v0.11.2
v0.11.2rc1
```

```
v0.12.0rc1
v0.12.0rc2
...
```

標籤列表顯示比特幣的所有發行版本。按照慣例，用於測試的發行預覽版（*release candidates*）具有後綴“rc”。可以在生產系統上運行的穩定版本沒有後綴。從上面的列表中選擇最高版本的發行版本，在撰寫本文時是v0.15.0。要使本地程式碼與此版本同步，請使用 git checkout 命令：

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

你可以通過命令 git status 來確認你已經“檢出”了所需的版本：

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

## Bitcoin Core的構建配置

源程式碼包括文件，可以在許多檔案中找到。輸入 \*\* more README.md\*\*，查看bitcoin目錄中的README.md主文件，可使用空格鍵進行翻頁。在本章中，我們將構建命令行比特幣客戶端（command-line bitcoin client），在Linux上也稱為 bitcoind。輸入 \*\*more doc/build-unix.md\*\* 來查看在你的平臺上編譯 bitcoind 的說明。macOS和Windows的說明可以在doc目錄中找到，分別為build-osx.md或build-windows.md。

仔細查看構建文件第一部分中的依賴庫，如boost-devel, libevent-devel, openssl-devel, gcc-c++, libdb4-cxx-devel, autoconf, automake, libtool等。在你開始編譯比特幣之前，這些庫必須存在於你的系統中，否則構建過程將失敗。如果因為漏掉了某些依賴庫而導致失敗，可以安裝它，然後從之前停止的地方恢復構建過程。你可以通過使用autogen.sh腳本生成一組構建腳本來啟動構建過程。

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

autogen.sh腳本創建一組自動配置腳本，它們將詢問你的系統以發現正確的設置，並確保你擁有編譯程式碼所需的全部庫。其中最重要的是 configure 腳本，它提供了許多不同的選項來定製構建過程。鍵入 \*\*./configure --help\*\* 查看各種選項。

```
$ ./configure --help
```

```
'configure' configures Bitcoin Core 0.15.0 to adapt to many kinds of systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

...
Optional Features:
--disable-option-checking ignore unrecognized --enable/--with options
--disable-FEATURE      do not include FEATURE (same as --enable-FEATURE=no)
--enable-FEATURE[=ARG]  include FEATURE [ARG=yes]

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

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

configure 腳本允許你通過使用 `-enable-FEATURE` 和 `-disable-FEATURE` 標誌啟用或禁用 `bitcoind` 的某些功能，其中 `FEATURE` 被替換為幫助輸出中列出的特徵名稱。在本章中，我們將構建帶有所有預設功能的 `bitcoind` 客戶端。我們不會使用配置標誌，但你應該查看它們以瞭解哪些可選功能是客戶端的一部分。如果你處於學術環境中，實驗室可能會要求你將應用程式安裝到你的主目錄中（例如，使用`-prefix=$HOME`）。

以下是覆蓋配置腳本預設行為的一些有用選項：

- prefix=\$HOME**  
覆蓋生成的可執行檔案的預設安裝位置 (`/usr/local/`)。使用 `$HOME` 以將所有內容放在你的主目錄，也可以使用其他路徑。
- disable-wallet**  
禁用錢包的參考實現。
- with-incompatible-bdb**  
如果你正在構建錢包，可以允許使用不兼容Berkeley DB庫的版本。
- with-gui=no**  
不構建需要Qt庫的圖形用戶界面。這隻會構建伺服器和命令行。

接下來，運行 `configure` 腳本，它會自動發現所有必要的庫，併為你的系統創建一個特定的構建腳本：

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

如果一切順利，`configure` 命令將創建允許我們編譯 `bitcoind` 的定製構建腳本並結束。如果有任何缺失的庫或錯誤，`configure` 命令將終止並顯示錯誤。如果發生錯誤，很可能是因為缺少或不兼容的庫。再次查看構建文件，並確保安裝缺少的先決條件。然後再次運行 `configure` 並查看是否修復了錯誤。

## 構建Bitcoin Core可執行檔案

接下來，你將編譯源程式碼，這個過程可能需要一個小時才能完成，具體取決於CPU的速度和可用內存。在編譯過程中，你應該每隔幾秒或幾分鐘看一次輸出。如果發生錯誤，或者編譯過程中斷，可以通過再次輸入 make 恢復。鍵入 \*\*make\*\* 開始編譯可執行應用程式：

```
$ make
Making all in src
  CXX      crypto/libbitcoinconsensus_la-hmac_sha512.lo
  CXX      crypto/libbitcoinconsensus_la-ripemd160.lo
  CXX      crypto/libbitcoinconsensus_la-sha1.lo
  CXX      crypto/libbitcoinconsensus_la-sha256.lo
  CXX      crypto/libbitcoinconsensus_la-sha512.lo
  CXX      libbitcoinconsensus_la-hash.lo
  CXX      primitives/libbitcoinconsensus_la-transaction.lo
  CXX      libbitcoinconsensus_la-pubkey.lo
  CXX      script/libbitcoinconsensus_la-bitcoinconsensus.lo
  CXX      script/libbitcoinconsensus_la-interpreter.lo

[... many more compilation messages follow ...]

$
```

在具有多個CPU的系統上，你可以設置並行編譯作業的核數。例如，make -j 2 將使用兩個CPU核。如果一切順利，Bitcoin Core已經編譯完成，你應該使用 make check 運行單元測試套件，以確保鏈接庫不會中斷。最後一步是使用 make install 命令在你的系統上安裝可執行檔案。系統可能會提示你輸入用戶密碼，因為此步驟需要管理權限：

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

bitcoind預設安裝在 */usr/local/bin* 中，你可以查看：

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

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

## 運行Bitcoin Core節點

比特幣的點對點網路由網路“節點”組成，主要由志願者和一些構建比特幣應用的企業運行。那些運行比特幣節點的人對比特幣區塊鏈擁有直接和權威的視野，並擁有所有交易的本地副本，由他們自己的系統進行獨立驗證。通過運行節點，你可以不依靠任何第三方來驗證交易。此外，你還可以通過增強比特幣網路的能力，為比特幣網路做出貢獻。

然而，運行一個節點，需要有足夠資源來處理所有比特幣交易的永久的可連接的系統。取決於你是否選擇索引所有交易並保存區塊鏈的完整副本，你可能還需要大量的硬碟空間和內存。在2018年初，一個全部索引的節點需要至少2GB內存和160GB硬盤空間（參見 <https://blockchain.info/charts/blocks-size>）。比特幣節點也需要網路帶寬來傳輸和接收比特幣交易和區塊。如果你的網路帶寬有限，則你可能不應該在其上運行比特幣節點，或者以限制其帶寬的方式運行它（參見 [Sample configuration of a resource-constrained system](#)）。

**Tip**

Bitcoin Core預設保存完整的區塊鏈副本，包括2009年以來所有比特幣網路上發生的交易。該數據集的大小為幾十GB，可以在幾天或幾周內遞增下載，具體取決於你的CPU和網路速度。在完整的區塊鏈數據集下載完之前，Bitcoin Core將無法處理交易或更新賬戶餘額。確保你有足夠的硬碟空間，帶寬和時間來完成初始同步。你可以配置Bitcoin Core，通過丟棄舊區塊來減少區塊鏈的大小。（參見 [Sample configuration of a resource-constrained system](#)），但在丟棄數據之前它仍會下載整個數據集。

儘管存在資源限制，仍有數千志願者運行比特幣節點。有些系統像Raspberry Pi一樣簡單（35美元的紙盒大小的電腦）。許多志願者還在租用伺服器上運行比特幣節點，通常是Linux的一些變體。*Virtual Private Server (VPS)* 或*Cloud Computing Server*實例可用於運行比特幣節點。這些伺服器可以每月25美元或50美元的價格從各種提供商處獲得。

為什麼要運行比特幣節點呢？以下是一些理由：

- 你正在開發比特幣軟體，需要依靠比特幣節點進行網路和區塊鏈的API訪問。
- 你正在構建必須根據比特幣的共識規則驗證交易的應用程式。例如，比特幣軟體公司通常運行多個節點。
- 你想支持比特幣。運行一個節點可以使網路更強大，能夠服務更多的錢包，更多的用戶和更多的交易。
- 你不想依賴任何第三方來處理或驗證你的交易。

如果你正在閱讀本書並且對開發比特幣軟體感興趣，那麼你應該運行自己的節點。

## 配置Bitcoin Core節點

Bitcoin Core在每次啟動時會在其數據目錄中查找配置檔案。在本節中，我們將研究各種配置選項並進行配置。要找到配置檔案，請在終端中運行 `bitcoind -printtoconsole` 並查看前幾行。

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

確定了配置檔案的位置之後，你可以按Ctrl-C關閉該節點。通常配置檔案位於用戶主目錄下的`.bitcoin`數據目錄中。接下來在編輯器中打開配置檔案。

Bitcoin Core提供了超過100種配置選項，可以修改網路節點的行為，區塊鏈的儲存以及許多其他方面。要查看這些選項的列表，請運行 `bitcoind --help`：

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

Usage:
  bitcoind [options]                                Start Bitcoin Core Daemon
```

```

Options:

-?
    Print this help message and exit

-version
    Print version and exit

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

...
[many more options]
...

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

```

以下是你可以在配置檔案中設置的一些最重要的選項，也可以作為 bitcoind 的命令行參數：

#### **alertnotify**

運行指定的命令或腳本，向該節點的所有者發送緊急警報，通常通過電子郵件的形式。

#### **conf**

配置檔案的替代位置。這僅適用於 bitcoind 的命令行參數。

#### **datadir**

放置所有區塊鏈數據的目錄。預設情況下，這是你的主目錄的 *.bitcoin* 目錄的子目錄。確保這個目錄所在的檔案系統有數GB的可用空間。

#### **prune**

通過刪除舊的區塊，將硬碟空間需求減少到多少MB。在不適合存放完整區塊鏈的資源受限節點上使用它。

#### **txindex**

維護所有交易的索引。這意味著，允許你以編程方式通過ID檢索一個完整的區塊鏈副本的任何交易。

#### **dbcache**

UTXO緩存的大小。預設值是300MB。在高端硬體上增加該值，在低端硬體上減少該值的大小以節省內存，但會導致更多硬碟開銷。

#### **maxconnections**

設置可以從最多多少個節點接受連接。將預設值減小將減少你的帶寬消耗。如果你有帶寬限制或按帶寬支付，請使用此選項。

#### **maxmempool**

將交易Memory pool的大小設置為多少MB。在內存受限的節點上使用。

#### **maxreceivebuffer/maxsendbuffer**

將每個連接的內存緩衝區限制為多少KB。在內存受限的節點上使用。

#### **minrelaytxfee**

設置你願意傳播的交易的最低費率。在此值以下，交易處理為非標準交易，從交易池中拒絕並且不轉發。

交易資料庫索引和txindex選項

預設情況下，Bitcoin Core只創建包含用戶的錢包相關交易的資料庫。如果你想使用 getrawtransaction 之類的命令訪問任何交易，(參見 [檢查並解碼交易](#))，你需要在配置檔案中設置 txindex=1 以使 Bitcoin Core 創建完整的交易索引。如果你一開始未設置此選項，在之後設置完成後需要使用 bitcoind -reindex 重啟並等待其重新構建索引。

[Sample configuration of a full-index node](#) 展示瞭如何將前面的選項與完全索引的節點相結合，作為比特幣應用程式的 API 後端運行。

Example 1. Sample configuration of a full-index node

```
alertnotify=myemailscript.sh "Alert: %s"
datadir=/lotsofspace/bitcoin
txindex=1
```

[Sample configuration of a resource-constrained system](#) 展示了資源受限節點的配置。

Example 2. Sample configuration of a resource-constrained system

```
alertnotify=myemailscript.sh "Alert: %s"
maxconnections=15
prune=5000
dbcache=150
maxmempool=150
maxreceivebuffer=2500
maxsendbuffer=500
```

在按照你需求編輯了配置檔案之後，可以運行 bitcoind -printtoconsole 來測試

```
$ bitcoind -printtoconsole

Bitcoin version v0.15.0
InitParameterInteraction: parameter interaction: -whitelistforcerelay=1 -> setting
-whitelistrelay=1
Assuming ancestors of block
0000000000000000003b9ce759c2a087d52abc4266f8f4ebd6d768b89defa50a have valid
signatures.
Using the 'standard' SHA256 implementation
Default data directory /home/ubuntu/.bitcoin
Using data directory /lotsofspace/.bitcoin
Using config file /home/ubuntu/.bitcoin/bitcoin.conf
Using at most 125 automatic connections (1048576 file descriptors available)
Using 16 MiB out of 32/2 requested for signature cache, able to store 524288
elements
Using 16 MiB out of 32/2 requested for script execution cache, able to store 524288
elements
Using 2 threads for script verification
HTTP: creating work queue of depth 16
No rpcpassword set - using random cookie authentication
Generated RPC authentication cookie /lotsofspace/.bitcoin/.cookie
HTTP: starting 4 worker threads
init message: Verifying wallet(s)...
Using BerkeleyDB version Berkeley DB 4.8.30: (April 9, 2010)
Using wallet wallet.dat
```

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

[... more startup messages ...]
```

確認配置正確被加載後可以按Ctrl-C結束進程。

要將Bitcoin Core作為後臺進程運行，可以使用 `bitcoind -daemon`。

要觀察節點的進程和運行時狀態，可使用 `bitcoin-cli getblockchaininfo`:

```
$ bitcoin-cli getblockchaininfo
```

這展示了區塊鏈高度為0個區塊，有83999個區塊頭的節點。節點先獲取最佳鏈的區塊頭，然後繼續下載完整區塊。

在你完成選項配置之後，應該將比特幣添加到操作系統中的啟動腳本中，以便它可以持續運行並在操作系統重新啟動時重啟。你可以在 `contrib/init` 下找到比特幣源目錄中各種操作系統的啟動腳本示例以及顯示哪個系統使用哪個腳本的 `README.md` 檔案。

## Bitcoin Core API

Bitcoin Core 客戶端實現了JSON-RPC接口，也可以使用命令行工具 `bitcoin-cli` 來訪問。命令行允許我們以交互的方式試驗通過API方式提供的功能。首先，調用 `help` 命令查看可用比特幣RPC命令的列表：

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

```
...
verifymessage "bitcoinaddress" "signature" "message"
walletlock
walletpassphrase "passphrase" timeout
walletpassphrasechange "oldpassphrase" "newpassphrase"
```

每個命令都可能需要許多參數。要獲得更詳細的參數訊息，請在 help 後添加命令名。例如，要查看 getblockhash RPC 命令的幫助：

```
$ bitcoin-cli help getblockhash
getblockhash height

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

Arguments:
1. height          (numeric, required) The height index

Result:
"hash"            (string) The block hash

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

在幫助訊息的末尾，你將看到兩個RPC命令的例子，分別使用 bitcoin-cli 和HTTP客戶端 curl。這些示例演示瞭如何調用該命令。複製第一個示例並查看結果：

```
$ bitcoin-cli getblockhash 1000
00000000c937983704a73af28acdec37b049d214adbda81d7e2a3dd146f6ed09
```

結果是一個區塊的雜湊值，在下面的章節中有更詳細的描述。該命令應該在你的系統上返回相同的結果，表明你的 Bitcoin Core節點正在運行，接受命令，並且將有關區塊1000的訊息返回給你。

在接下來的部分中，我們將演示一些非常有用的RPC命令及其預期的輸出。

## 獲得Bitcoin Core客戶端的狀態訊息

Bitcoin Core 通過 JSON-RPC 接口提供不同模塊的狀態報告。最重要的命令包括 getblockchaininfo, getmempoolinfo, getnetworkinfo 和 getwalletinfo。

比特幣的 getblockchaininfo RPC 命令之前已經介紹了。getnetworkinfo 命令用於展示比特幣網路節點的基本狀態訊息。使用 bitcoin-cli 調用：

```
$ bitcoin-cli getnetworkinfo
```

```
"version": 150000,
"subversion": "/Satoshi:0.15.0/",
"protocolversion": 70015,
"localservices": "000000000000000d",
```

```
"localrelay": true,
"timeoffset": 0,
"networkactive": true,
"connections": 8,
"networks": [
    ...
    detailed information about all networks (ipv4, ipv6 or onion)
    ...
],
"relayfee": 0.00001000,
"incrementalfee": 0.00001000,
"localaddresses": [
],
"warnings": ""
}
```

數據通過JSON格式返回,可以被所有程式語言處理,而且是可讀的。在這些數據中,我們可以看到比特幣軟體客戶端版本號(150000)和比特幣協議版本號(70015),當前的連接數(8),以及有關比特幣網路的各種訊息與此客戶端相關的設置。

**Tip** 对于 bitcoind 客户端来说，它需要一段时间（或许超过一天的时间）“赶上”当前区块链高度，因为它从其他比特币客户端下载区块。你可以使用 `getblockchaininfo` 来查看其进度，以查看已知区块的数量。

## 檢查並解碼交易

命令: getrawtransaction, decoderawtransaction

在 [\[cup\\_of\\_coffee\]](#) 的案例中，Alice 從 Bob's Cafe 購買了一杯咖啡。她的交易 ID (txid) 為 0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286c345c2f2。讓我們使用 API，通過交易 ID 來檢索和查看這筆交易：

```
$ bitcoin-cli getrawtransaction 0627052b6f28912f2703066a912ea577f2ce4da4caa5a←  
5fbd8a57286c345c2f2  
  
01000000001186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd734d2804fe65fa35779000←  
000008b483045022100884d142d86652a3f47ba4746ec719bbfb040a570b1deccbb6498c75c4←  
ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813014←  
10484ecc0d46f1918b30928fa0e4ed99f16a0fb4fd0735e7ade8416ab9fe423cc54123363767←  
89d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adfffffffff0260e3160000000←  
0001976a914ab68025513c3dbd2f7b92a94e0581f5d50f654e788acd0ef80000000000001976a9←  
147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac00000000
```

**Tip** 在交易確認之前，交易ID不是可信的。區塊鏈中沒有交易的雜湊值並不意味著交易未處理。這被稱為“交易可鑄性 (transaction malleability) ”，因為在區塊中確認之前，交易的雜湊可以被修改。確認後，txid是不可變的和可信的。

`getrawtransaction` 命令以十六進制返回一個序列化的交易。將它作為 `decoderawtransaction` 命令的參數可以解碼：

```
$ bitcoin-cli decoderawtransaction 01000000001186f9f998a5aa6f048e51dd8419a14d8-a0f1a8a2836dd734d2804fe65fa35779000000008b483045022100884d142d86652a3f47ba474-6ec719bbfb0d40a570b1deccbb6498c75c4ae24cb02204b9f039ff08df09cbe9f6addac960298-cad530a863ea8f53982c09db8f6e381301410484ecc0d46f1918b30928fa0e4ed99f16a0fb4fd-e0735e7ade8416ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc17b4a10fa-336a8d752adffffffffff0260e31600000000001976a914ab68025513c3dbd2f7b92a94e0581f5-
```

```
d50f654e788acd0ef80000000000001976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8+  
88ac00000000
```

```
{
  "txid": "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286c345c2f2",
  "size": 258,
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid": "7957a35fe64f80d234d76d83a2...8149a41d81de548f0a65a8a999f6f18",
      "vout": 0,
      "scriptSig": {
        "asm": "3045022100884d142d86652a3f47ba4746ec719bbfb040a570b1decc...",
        "hex": "483045022100884d142d86652a3f47ba4746ec719bbfb040a570b1de..."
      },
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.01500000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 ab68...5f654e7 OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a914ab68025513c3dbd2f7b92a94e0581f5d50f654e788ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"
        ]
      }
    },
    {
      "value": 0.08450000,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 7f9b1a...025a8 OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK"
        ]
      }
    }
  ]
}
```

解碼後的交易展示了此交易的所有組成部分，包括交易的輸入和輸出。我們看到將15mBitcoin轉到新地址的交易使用了一個輸入併產生了兩個輸出。此交易的輸入是以前確認的交易的輸出（vin中的txid，以7957a35fe開頭）。兩個輸出對應於15mBitcoin的款項，和返回給發送者的零錢。

我們可以通過使用getrawtransaction檢查此交易中引用的前一個交易的txid來進一步探索區塊鏈。我們可以追蹤一筆比特幣在不同所有者地址之間傳遞的交易鏈。

## 檢查區塊

命令: getblock, getblockhash

檢查區塊與檢查交易類似。區塊可以通過區塊高度（height）或區塊雜湊（hash）來引用。首先，我們根據高度找到一個區塊。在[\[cup\\_of\\_coffee\]](#)中，我們看到Alice的交易包含在區塊#277316中。

將區塊高度作為getblockhash命令的參數，，將返回區塊的雜湊值：

```
$ bitcoin-cli getblockhash 277316
00000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

現在我們知道Alice的交易被包含在哪個區塊中了，我們可以使用getblock命令，傳遞區塊雜湊值來查詢該區塊。

```
$ bitcoin-cli getblock 00000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

```
{
  "hash": "00000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4",
  "confirmations": 37371,
  "size": 218629,
  "height": 277316,
  "version": 2,
  "merkleroot": "c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e",
  "tx": [
    "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f",
    "b268b45c59b39d759614757718b9918caf0ba9d97c56f3b91956ff877c503fbe",
    "04905ff987ddd4cfe603b03cfb7ca50ee81d89d1f8f5f265c38f763eea4a21fd",
    "32467aab5d04f51940075055c2f20bbd1195727c961431bf0aff8443f9710f81",
    "561c5216944e21fa29dd12aaa1a45e3397f9c0d888359cb05e1f79fe73da37bd",
    [... hundreds of transactions ...]
    "78b300b2a1d2d9449b58db7bc71c3884d6e0579617e0da4991b9734cef7ab23a",
    "6c87130ec283ab4c2c493b190c20de4b28ff3caf72d16ffa1ce3e96f2069aca9",
    "6f423dbc3636ef193fd8898dfdf7621dcade1bbe509e963ffbff91f696d81a62",
    "802ba8b2adabc5796a9471f25b02ae6aeee2439c679a5c33c4bbcee97e081196",
    "eaaf6a048588d9ad4d1c092539bd571dd8af30635c152a3b0e8b611e67d1a1af",
    "e67abc6bd5e2cac169821afc51b207127f42b92a841e976f9b752157879ba8bd",
    "d38985a6a1bfd35037cb7776b2dc86797abb7a06630f5d03df2785d50d5a2ac",
    "45ea0a3f6016d2bb90ab92c34a7aac9767671a8a84b9bcc6c019e60197c134b",
    "c098445d748ced5f178ef2ff96f2758cbe9eb32cb0fc65db313bcac1d3bc98f"
  ],
  "time": 1388185914,
  "mediantime": 1388183675,
```

這個區塊包含 419 筆交易，第64個交易 (0627052b...) 是Alice的交易。height 欄位告訴我們它是區塊鏈中的第277316 個區塊。

## 使用Bitcoin Core的編程接口

bitcoin-cli 助手對於探索比特幣核心API和測試功能非常有用。但API的重要功能是以編程方式訪問。在本節中，我們將演示如何通過另一個程序訪問比特幣核心。

Bitcoin Core的API是JSON-RPC接口。JSON代表JavaScript Object Notation，是一種非常方便人類和程序閱讀的數據格式。RPC代表遠程過程調用，這意味著我們通過網路協議調用遠程（在比特幣核心節點上的）函數。在這種情況下，網路協議是HTTP或HTTPS（用於加密連接）。

當我們使用 `bitcoin-cli` 命令獲取命令幫助時，它向我們展示了使用 curl（常用的命令行HTTP客戶端）構造JSON-RPC 調用的示例：

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

此命令表示 curl 向本地主機（127.0.0.1）提交HTTP請求，連接到預設的比特幣端口（8332），並使用 text/plain 編碼為 getblockchaininfo 方法提交 jsonrpc 請求。

你可能會注意到curl會要求憑證隨請求一起發送。Bitcoin Core在每次啟動時創建一個隨機密碼，並將其放置在名稱為.cookie的數據目錄中。bitcoin-cli助手可以根據數據目錄讀取此密碼檔案。同樣，你可以複製密碼並將其傳遞給curl（或任何更高級別的Bitcoin Core RPC包裝器）。或者，你可以使用Bitcoin Core源碼目錄中的./share/rpcuser/rpcuser.py中提供的助手程序腳本創建一個靜態密碼。

你可以正在自己的程序中使用HTTP庫來實現JSON-RPC調用，類似於前面的curl示例。

然而，大多數程式語言都有一些“包裝”了比特幣核心API的庫，簡便很多。我們將使用 `python-bitcoinlib` 庫來簡化API訪問。你需要有一個正在運行的Bitcoin Core實例，用於進行JSON-RPC調用。

Running `getblockchaininfo` via Bitcoin Core's JSON-RPC API 中的Python腳本調用 `getblockchaininfo` 並打印返回數據中的區塊個數。

Example 3. Running `getblockchaininfo` via Bitcoin Core's JSON-RPC API

link:code/rpc\_example.py[1]

運行它可以得到下面的結果：

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

它標明本地的Bitcoin Core節點在其區塊鏈中有394075個區塊。

接下來，讓我們使用 `getrawtransaction` 和 `decoderawtransaction` 調用來檢索Alice的咖啡交易細節。在[Retrieving a transaction and iterating its outputs](#)中，我們檢索Alice的交易並列出交易的輸出。對於每個輸出，我們顯示收件人地址和值。提醒一下，Alice的交易有一個支付給Bob's Cafe的輸出和一個返回給Alice的找零輸出。

#### Example 4. Retrieving a transaction and iterating its outputs

```
link:code/rpc_transaction.py[]
```

運行這段程式碼：

```
$ python rpc_transaction.py
([u'1GdK9UzpHBzqZX2A9JFP3Di4weBwqqmoQA'], Decimal('0.01500000'))
([u'1Cdid9KFAaatwcBwBttQcwXYCpvK8h7FK'], Decimal('0.08450000'))
```

前面的兩個例子都很簡單。你並不需要寫程序來運行它們；你可以輕鬆使用 `bitcoin-cli` 助手。然而，下一個例子需要幾百次RPC調用，更清楚地說明了編程接口的作用。

在[Retrieving a block and adding all the transaction outputs](#)中，我們首先檢索第277316個區塊，然後使用交易ID檢索區塊內419個交易中的每一個。接下來，我們遍歷每筆交易的輸出並累加起來。

#### Example 5. Retrieving a block and adding all the transaction outputs

```
link:code/rpc_block.py[]
```

運行這段程式碼：

```
$ python rpc_block.py
('Total value in block: ', Decimal('10322.07722534'))
```

我們的示例程式碼計算出此區塊的總價值為10322.07722534 BTC（包括25BTC獎勵和0.0909BTC費用）。通過在區塊瀏覽器中搜索雜湊或高度，進行數據比較。一些區塊瀏覽器報告不包括獎勵和排除費用的總價值，看看你是否可以發現差異。

## 可選的客戶端、程序庫和工具包

比特幣生態系統中有許多可選的客戶端，程序庫和工具包，甚至是完整節點的實現。它們以各種程式語言實現，為開發者提供其首選程式語言的原生接口。

下面列出了一些：

### C/C++

#### [Bitcoin Core](#)

比特幣的參考實現

#### [libbitcoin](#)

跨平臺的C++開發工具，節點和共識函數庫

#### [bitcoin explorer](#)

Libbitcoin的命令行工具

#### [picocoin](#)

Jeff Garzik寫的C語言的輕量級比特幣客戶端庫

## JavaScript

### *bcoin*

帶有API的模塊化，可擴展的完整節點實現

### *Bitcore*

Bitpay提供的完整節點，API和程序庫

### *BitcoinJS*

用於node.js和瀏覽器的純JavaScript比特幣庫

## Java

### *bitcoinj*

Java版完整節點客戶端庫

### *Bits of Proof (BOP)*

比特幣的Java企業級實現

## PHP

### *bitwasp/bitcoin*

PHP比特幣庫, 和相關的項目

## Python

### *python-bitcoinlib*

Python比特幣庫，共識庫，和Peter Todd寫的節點

### *pycoin*

Richard Kiss寫的Python比特幣庫

### *pybitcointools*

Vitalik Buterin寫的Python比特幣庫

## Ruby

### *bitcoin-client*

Ruby封裝的JSON-RPC API

## Go

### *btcd*

Go語言的完整節點客戶端

## Rust

### *rust-bitcoin*

用於序列化，解析和API調用的Rust比特幣庫

## C#

**NBitcoin**

.NET框架的綜合比特幣庫

**Objective-C**

**CoreBitcoin**

為ObjC和Swift提供的比特幣工具包

還有各種程式語言的庫存在，還有更多的庫在開發。

## 密鑰和地址

你可能聽說過比特幣是基於 密碼學 的，它是電腦安全領域廣泛使用的數學分支。密碼學在希臘文中的意思是“祕密寫作”，但密碼學的科學不僅僅包含祕密寫作，它被稱為加密。密碼學也可以用來在不洩露保密內容的情況下，證明一個人知道保密內容（數字簽名），或證明數據的真實性（數字指紋）。這些密碼學基礎是比特幣的關鍵數學工具，並廣泛用於比特幣應用。諷刺的是，加密並不是比特幣的重要組成部分，因為它的通信和交易數據沒有加密，也不需要通過加密保護資金。在本章中，我們將介紹比特幣中使用的密碼學，以密鑰，地址和錢包的形式控制資金的所有權。

### 簡介

比特幣的所有權通過 數字密鑰 (*digital keys*) ， 比特幣地址 (*bitcoin addresses*) 和 數字指紋 (*digital signatures*) 建立。數字密鑰實際上並不儲存在網路中，而是由用戶創建並儲存在檔案或稱為 錢包 (*wallet*) 的簡單資料庫中。用戶錢包中的數字密鑰完全獨立於比特幣協議，可以由用戶的錢包軟體生成和管理，無需參考區塊鏈或訪問互聯網。密鑰支撐了比特幣的許多有趣特性，包括去中心化的信任和控制，所有權證明以及有密碼學保障的安全模型。

為了包含在區塊鏈中，大多數比特幣交易需要有效的數字簽名，這些交易只能使用密鑰生成；因此，任何擁有該密鑰副本的人都可以控制這些比特幣。用於花費資金的數字簽名也被稱為 證據 (*witness*)，是密碼學中的術語。比特幣交易中的證據證明瞭所花費資金的真實所有權。

密鑰由一對公鑰和私鑰組成。將公鑰視為類似於銀行帳號，將私鑰視為PIN或支票上的簽名，用於控制帳戶。比特幣用戶很少看到這些數字密鑰。大多數情況下，它們儲存在錢包檔案中並由比特幣錢包軟體管理。

在比特幣交易的付款部分，收款人的公鑰通過其數字指紋表示，稱為 比特幣地址 (*bitcoin address*)，與支票上的收款人姓名一樣使用（即“付款到誰的帳戶”）。大多數情況下，比特幣地址是從公鑰生成的並且對應於公鑰。但是，並非所有的比特幣地址都代表公鑰；他們也可以代表其他受益者，如腳本，我們將在本章後面看到。通過這種方式，比特幣地址可以抽象為資金接收者，這使交易目的地變得靈活，類似於紙質支票：可用於支付個人賬戶，支付公司賬戶，支付賬單或兌換現金。比特幣地址是密鑰的唯一展現形式，用戶常會看到，因為他們需要向世界公開。

首先，我們將介紹密碼學並解釋比特幣中使用的數學。接下來，我們將看看密鑰是如何生成，儲存和管理的。我們將看一下用於表示私鑰公鑰，地址和腳本地址的各種編碼格式。最後，我們將看看密鑰和地址的高級用法：虛榮 (Vanity)，多重簽名，腳本地址和紙錢包。

### 公鑰加密和密碼貨幣

公鑰密碼技術發明於20世紀70年代，是電腦和訊息安全的數學基礎。

公鑰密碼技術發明後，發現了一些合適的數學函數，例如質數指數運算和橢圓曲線乘法。這些數學函數實際上是不可逆的，這意味著它們很容易在一個方向上計算，但在相反方向上計算是不可行的。基於這些數學函數，密碼學可以創建數字密鑰和不可偽造的數字簽名。比特幣使用橢圓曲線乘法作為其密碼學的基礎。

在比特幣中，我們使用公鑰密碼技術來創建一個控制比特幣訪問的密鑰對。密鑰對由一個私鑰和從它派生的一個唯一的公鑰組成。公鑰用於接收資金，私鑰用於簽署交易以支付資金。

公鑰和私鑰之間存在數學關係，可以用私鑰生成一個訊息的簽名，然後使用公鑰在不公開私鑰的情況下驗證簽名。

在花費比特幣時，當前比特幣的所有者需要在交易中提供他的公鑰和簽名（每次都不同，但由相同的私鑰創建）。通過公鑰和簽名，比特幣網路中的每個人都可以驗證該交易的有效性並接受，從而確認轉讓這筆比特幣的人擁有它們。

Tip

在大多數錢包實現中，為了方便起見，私鑰和公鑰一起儲存為 密鑰對兒 (*key pair*)。由於可以從私鑰計算公鑰，因此只儲存私鑰也是可能的。

### 私鑰和公鑰

比特幣錢包包含密鑰對兒的集合，每個密鑰對兒包含一個私鑰和一個公鑰。私鑰 ( $k$ ) 是一個數字，通常隨機選取。我們使用橢圓曲線乘法（單向加密函數）通過私鑰生成公鑰 ( $K$ )。從公鑰 ( $K$ ) 中，我們使用單向加密雜湊函數來生成比特幣地址 ( $A$ )。在本節中，我們將開始生成私鑰，查看用於將其轉換為公鑰的橢圓曲線數學運算，最後從公鑰生成一個比特幣地址。私鑰，公鑰和比特幣地址之間的關係如[Private key, public key, and bitcoin address](#)所示。

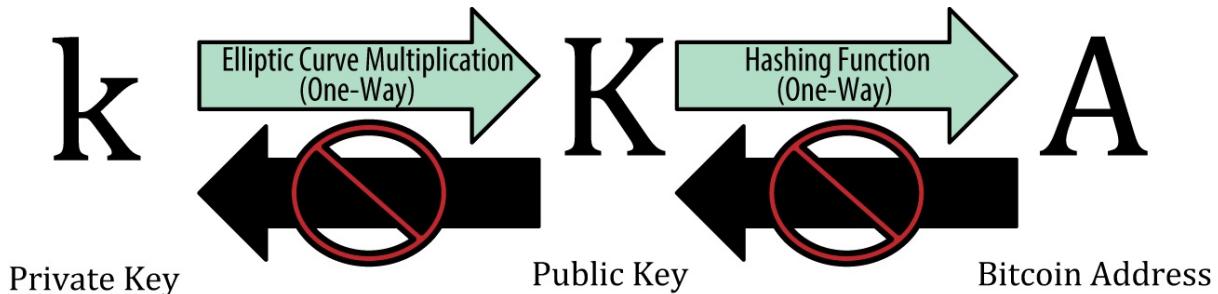


Figure 1. Private key, public key, and bitcoin address

為什麼使用非對稱加密 (公鑰私鑰)？

為什麼在比特幣中使用非對稱加密技術？因為它不是用來“加密”（保密）交易的。相反，非對稱加密的有用特性是產生數字簽名。私鑰可用於為交易生成指紋（數字簽名）。這個簽名只能由知道私鑰的人制作。但是，任何有權訪問公鑰和交易指紋的人都可以使用它們來驗證簽名確實是私鑰的擁有者生成的。非對稱加密的這一有用特性使任何人都可以驗證每筆交易的每個簽名，同時確保只有私鑰所有者才能生成有效的簽名。

## 私鑰

私鑰只是一個隨機選取的數字。對私鑰的所有權和控制權是用戶控制相應比特幣地址所關聯的所有資金的根本。私鑰用於通過證明交易中使用的資金的所有權來創建花費比特幣所需的簽名。私鑰在任何時候都必須保密，因為向第三方透露它相當於讓它們控制由該密鑰保護的比特幣。私鑰還必須備份和防止意外丟失，因為如果丟失了私鑰，它就無法恢復，並且它所保護的資金也會永遠丟失。

**Tip**

比特幣私鑰只是一個數字。你可以使用硬幣，鉛筆和紙隨機挑選你的私鑰：投擲硬幣256次，就可以獲得隨機的一串二進制（0和1）數字，在錢包中使用。然後可以從私鑰生成公鑰。

### 通過隨機數生成私鑰

生成密鑰的第一步也是最重要的一步是找到一個安全的熵源或隨機數。創建比特幣私鑰本質上與“選擇一個1到 $2^{256}$ 之間的數字”相同。只要保證不可預測性和不可重複性，用於選擇該數字的確切方法並不重要。比特幣軟體使用底層操作系統的隨機數生成器生成256位的私鑰（隨機數）。通常，操作系統隨機數生成器是由一個人為的隨機源進行初始化的，這就是為什麼你可能會被要求將鼠標擺動幾秒鐘。

更具體地來說，私鑰可以是 0 到  $n-1$  的任何數字，這裡 $n$ 是一個常數 ( $n = 1.1578 \times 10^{77}$ , 略小於  $2^{256}$ ) 定義為比特幣中使用的橢圓曲線的階數 (see [橢圓曲線密碼學解釋](#))。為了創建這樣的密鑰，我們隨機選擇一個256位的數字並檢查它是否小於 $n$ 。從編程的角度說，通常是通過從密碼學安全的隨機源收集的大量隨機數輸入SHA256雜湊演算法中，該演算法將產生256位的數字。如果結果小於 $n$ ，我們就找到了一個合適的私鑰。否則，我們只需使用另一個隨機數再次嘗試。

**Warning**

不要自己編寫程式碼或使用你的程式語言提供的“簡單”隨機數生成器來創建一個隨機數。使用密碼學安全的偽隨機數生成器（CSPRNG）和來自足夠熵源的種子。研究你選擇的隨機數生成器庫的文件，以確保其是密碼學安全的。正確實施CSPRNG對於密鑰的安全至關重要。

以下是以十六進制格式顯示的隨機生成的私鑰 ( $k$ )（256位，顯示為64個十六進制數字，每個4位）：

1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD

**Tip**

比特幣私鑰的數值空間的大小 ( $2^{256}$ ) 是非常大的數目。十進制大約是 $10^{77}$ 。可見的宇宙估計含有 $10^{80}$ 原子。

要使用 Bitcoin Core 客戶端生成新的密鑰 (see [ch03\_bitcoin\_client]), 可以用 `getnewaddress` 命令。出於安全考慮，它只顯示公鑰，而不顯示私鑰。可以使用 `dumpprivatekey` 命令要求 `bitcoind` 公開私鑰。`dumpprivatekey` 命令以Base58 checksum編碼顯示私鑰，稱為 錢包匯入格式 (WIF) ，我們將在[私鑰格式](#)中更詳細地介紹。以下是使用這兩個命令生成和顯示私鑰的示例：

```
$ bitcoin-cli getnewaddress
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
$ bitcoin-cli dumpprivatekey 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
KxFc1jmwwCoACiCAwZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

`dumpprivatekey` 命令打開錢包並提取由 `getnewaddress` 命令生成的私鑰。除非它們都儲存在錢包中，否則`bitcoind`不可能通過公鑰知道私鑰。

Tip

`dumpprivatekey`命令不會通過公鑰生成私鑰，因為這是不可能的。該命令只是顯示錢包已知的由+  
`getnewaddress`+命令生成的私鑰。

你還可以使用比特幣資源管理器命令行工具（參見[appdx\\_bx](#)）使用命令 `seed`，`ec-new` 和 `ec-to-wif` 來生成和顯示私鑰：

```
$ bx seed | bx ec-new | bx ec-to-wif
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

## 公鑰

公鑰使用私鑰通過橢圓曲線乘法計算，這是不可逆的： $K = k * G$ ，其中  $k$  是私鑰， $G$  是一個稱為 生成點 (*generator point*) 的固定的點， $K$  是公鑰。如果你知道  $K$ ，那麼稱為“尋找離散對數”的逆運算與嘗試所有可能的  $k$  值（即蠻力搜索）一樣困難。在我們演示如何從私鑰生成公鑰之前，我們先來看一下橢圓曲線加密。

Tip

橢圓曲線乘法是密碼學家稱為“陷阱門”的一種函數：在一個方向（乘法）很容易做到，而在相反方向（除法）不可能做到。私鑰的所有者可以很容易地創建公鑰，然後與世界共享，因為知道沒有人能夠反轉該函數並從公鑰計算私鑰。這種數學技巧成為證明比特幣資金所有權的不可偽造且安全的數字簽名的基礎。

## 橢圓曲線密碼學解釋

橢圓曲線密碼術是一種基於離散對數問題的非對稱或公鑰密碼技術，用橢圓曲線上的加法和乘法表示。

An [elliptic curve](#) 是一個橢圓曲線的示例，與比特幣使用的類似。

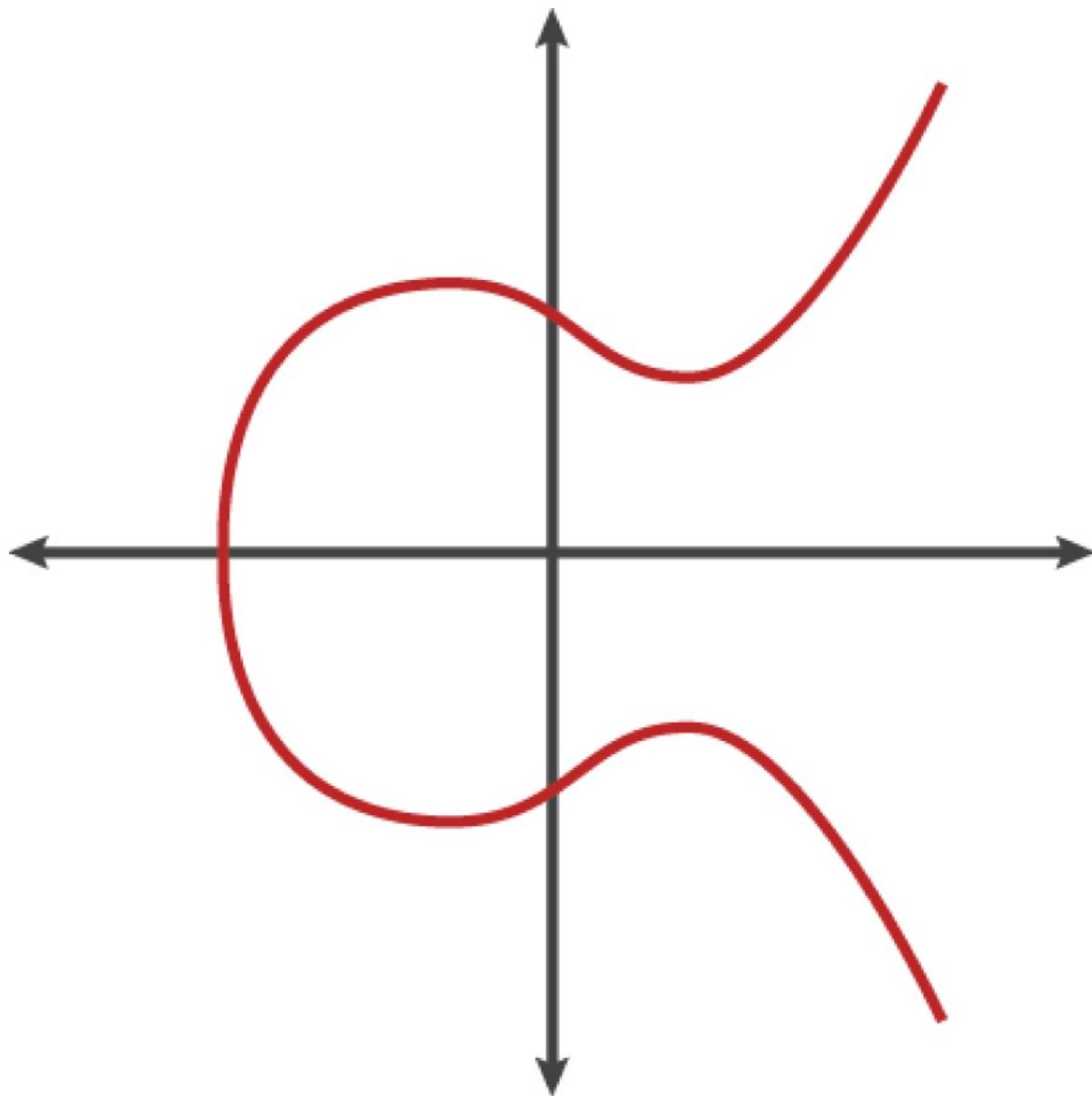


Figure 2. An elliptic curve

比特幣使用由美國國家標準與技術研究院（NIST）建立的稱為 secp256k1 的標準中定義的特定橢圓曲線和一組數學常數。secp256k1 曲線由以下函數定義，產生一個橢圓曲線：

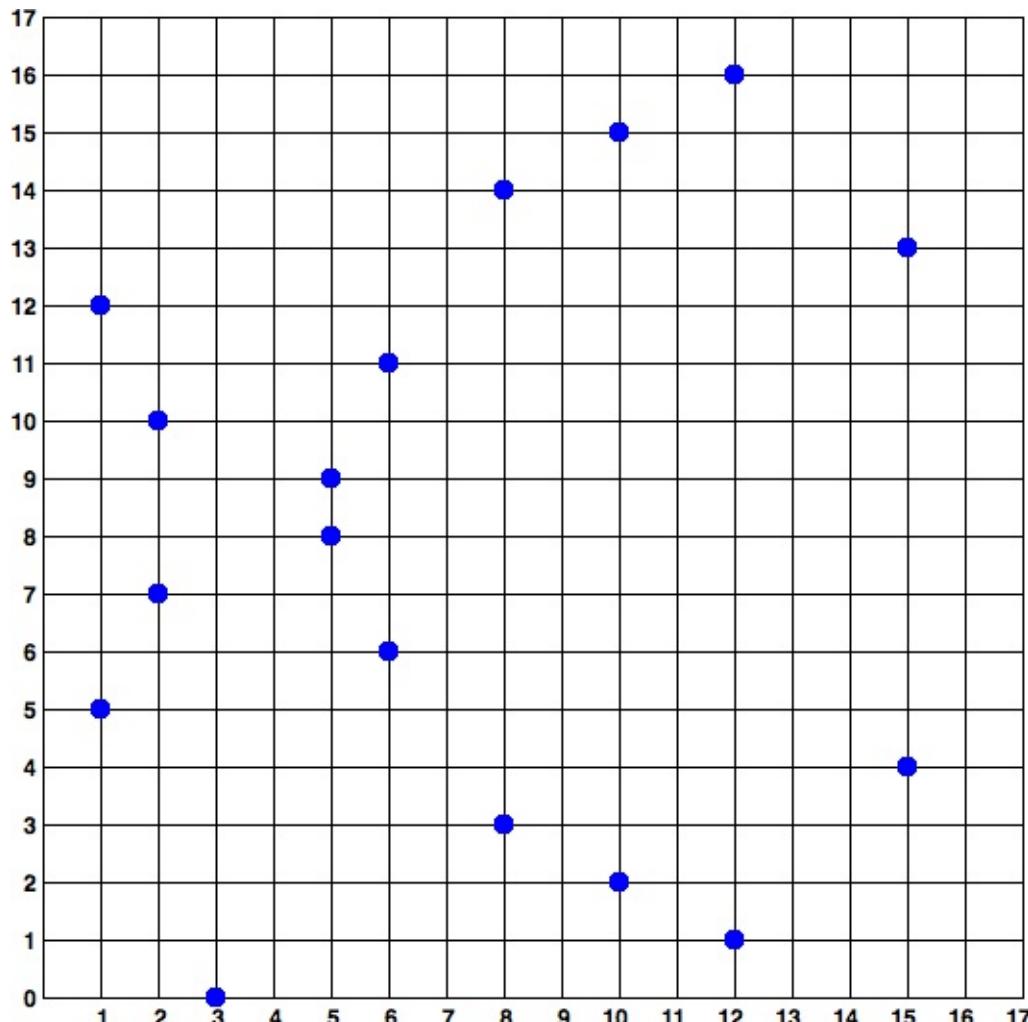
$$\text{\begin{equation} } y^2 = (x^3 + 7) \text{\text{over}} (\mathbb{F}_p) \text{\end{equation}}$$

or

$$\text{\begin{equation} } y^2 \text{\text{mod}} p = (x^3 + 7) \text{\text{mod}} p \text{\end{equation}}$$

$\text{mod } p$  (模質數  $p$ ) 表明該曲線位於質數階的有限域上。 $p$ , 也寫作  $(\mathbb{F}_p)$ ,  $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$ , 是一個非常大的質數。

因為這條曲線是在有限的質數階上而不是在實數上定義的，所以它看起來像是一個散佈在二維中的點的模式，難以可視化。然而，運算與實數上的橢圓曲線的是相同的。作為示例，[Elliptic curve cryptography: visualizing an elliptic curve over  \$\mathbb{F}\(p\)\$ , with  \$p=17\$](#)  在一個更小的質數階 17 的有限域上顯示了相同的橢圓曲線，顯示了一個網格上的點的圖案。可以認為 secp256k1 比特幣橢圓曲線是一個不可思議的大網格上的非常複雜的點陣。

Figure 3. Elliptic curve cryptography: visualizing an elliptic curve over  $F(p)$ , with  $p=17$ 

例如，以下是座標為  $(x, y)$  的點  $P$ ，它是 secp256k1 曲線上的一個點：

```
P = (55066263022277343669578718895168534326250603453777594175500187360389116729240,
32670510020758816978083085130507043184471273380659243275938904335757337482424)
```

[Using Python to confirm that this point is on the elliptic curve](#) 展示瞭如何使用Python檢驗：

Example 1. Using Python to confirm that this point is on the elliptic curve

```
Python 3.4.0 (default, Mar 30 2014, 19:23:13)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> p = 115792089237316195423570985008687907853269984665640564039457584007908834671663
>>> x = 55066263022277343669578718895168534326250603453777594175500187360389116729240
>>> y = 32670510020758816978083085130507043184471273380659243275938904335757337482424
>>> (x ** 3 + 7 - y**2) % p
0
```

在橢圓曲線中，有一個叫做“無限點”的點，它大致相當於零點的作用。在電腦上，它有時用  $x = y = 0$  表示（它不滿足橢圓曲線方程，但它是一個容易區分的情況）。

還有一個  $+$  運算符，稱為 “加法”，與傳統的實數加法有類似的屬性。給定橢圓曲線上的點  $P_1$  和  $P_2$ ，則  $P_3 = P_1 + P_2$ ，也在橢圓曲線上。

幾何上來說， $P_3$ 是通過在 $P_1$ 和 $P_2$ 之間畫一條直線來計算的。這條線將在另外一點與橢圓曲線相交，稱此點為  $P_3' = (x, y)$ 。然後在x軸上反射得到  $P_3 = (x, -y)$ 。有幾個特殊情況解釋了“無限點”的需要。

如果  $P_1$  和  $P_2$  是同一點，則  $P_1$  和  $P_1$  之間的直線應該延伸到曲線上  $P_1$  的切線。該切線恰好與曲線相交於一個新的點。你可以使用微積分技術來確定切線的斜率。儘管我們侷限在具有兩個整數座標的曲線上，但這些機制仍然可以神奇的運轉。

在某些情況下（如  $P_1$  和  $P_2$  具有相同的x值但不同的y值），切線將是垂直的，在這種情況下  $P_3$  = “無限點”。

如果  $P_1$  是“無窮遠點”，則  $P_1 + P_2 = P_2$ 。同樣，如果  $P_2$  是無窮遠點，則  $P_1 + P_2 = P_1$ 。這展示了無窮遠點如何扮演零的角色。

$+$  是可結合的，這意味著  $(A + B) + C = A + (B + C)$ 。這意味著我們可以書寫  $A + B + C$ ，沒有括號也沒有歧義。

現在我們已經定義了加法，我們可以用擴展加法的標準方式來定義乘法。對於橢圓曲線上的點  $P$ ，如果  $k$  是整數，則  $kP = P + P + P + \dots + P$  ( $k$  次)。在這種情況下， $k$  有時會被混淆地稱為“指數”。

## 生成公鑰

從一個隨機生成的私鑰  $k$  開始，我們將它乘以曲線上的一個預定點，稱為 生成點 (generator point)  $G$ ，以在曲線上的其他位置生成另一個點，這是相應的公鑰  $K$ 。生成點被指定為 secp256k1 標準的一部分，並且對於比特幣中的所有密鑰都是相同的：

$\begin{aligned} K &= k * G \end{aligned}$

其中  $k$  是私鑰， $G$  是生成點， $K$  是生成的公鑰，即曲線上的一個點。由於所有比特幣用戶的生成點始終相同，因此  $G$  乘以  $G$  的私鑰始終會生成相同的公鑰  $K$ 。 $k$  和  $K$  之間的關係是固定的，但只能從  $k$  到  $K$  的一個方向進行計算。這就是為什麼比特幣地址（從  $K$  派生）可以與任何人共享，並且不會洩露用戶的私鑰（ $k$ ）。

Tip

私鑰可以轉換為公鑰，但公鑰不能轉換回私鑰，因為計算是單向的。

實現橢圓曲線乘法，我們將先前生成的私鑰  $k$  與乘法生成點  $G$  相乘得到公鑰  $K$ ：

```
K = 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD * G
```

公鑰  $K$  被定義為一個點  $K = (x, y)$

$K = (x, y)$

其中，

$x = F028892BAD7ED57D2FB57BF33081D5CF6F9ED3D3D7F159C2E2FFF579DC341A$

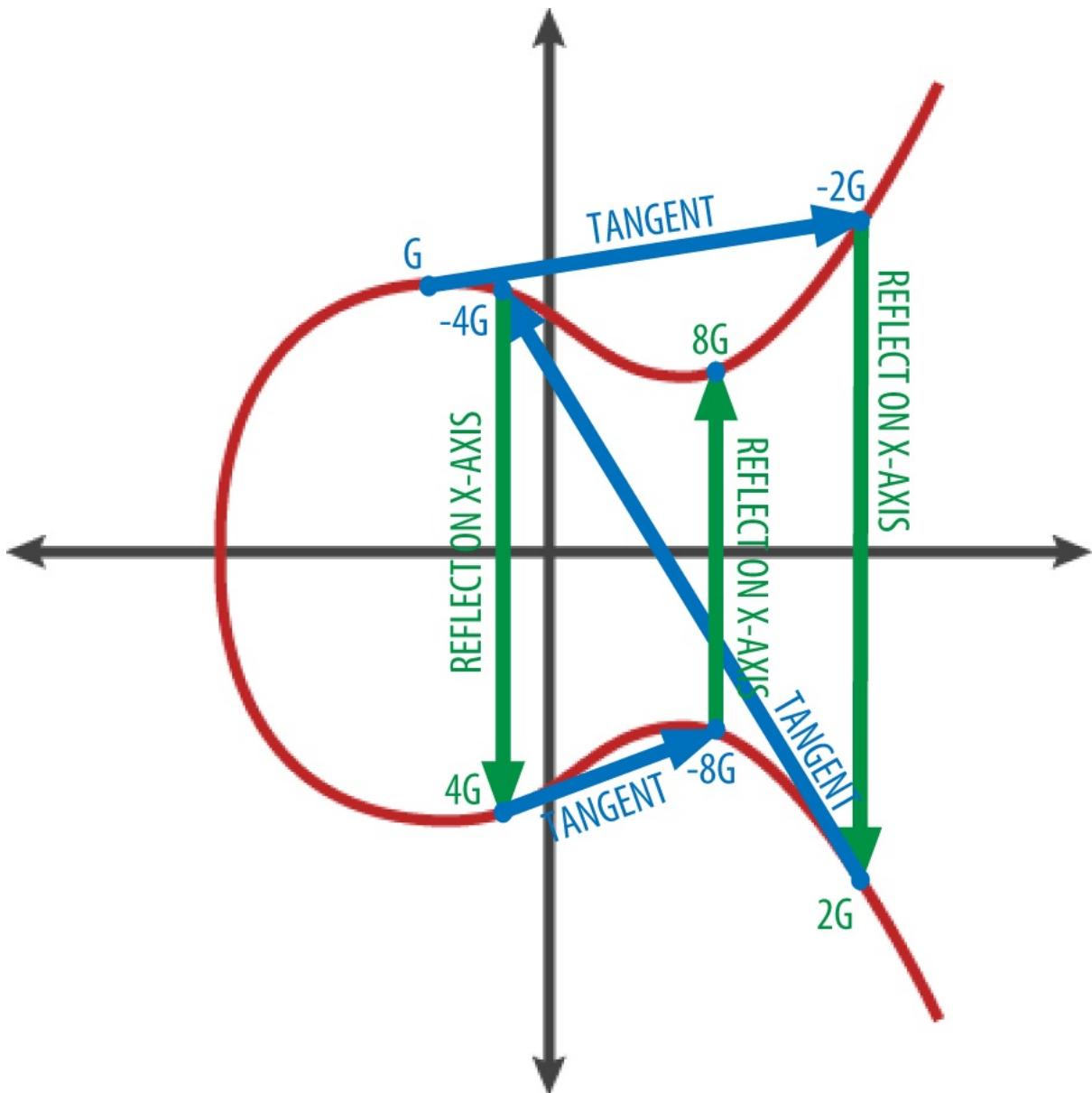
$y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB$

為了可視化一個點與整數的乘積，我們將使用簡單的橢圓曲線來代替實數——記住，演算法是相同的。我們的目標是找到生成點  $G$  的多個  $kG$ ，這與將  $G$  自身相加  $k$  次相同。在橢圓曲線中，一個點自身相加相當於在該點上繪製切線並找到它再次與曲線相交的位置，然後在x軸上反射該點。

[Elliptic curve cryptography: visualizing the multiplication of a point G by an integer k on an elliptic curve](#) 展示了  $G$ ,  $2G$ ,  $4G$  在曲線上的幾何操作。

Tip

大多數比特幣實現使用 [OpenSSL cryptographic library](#) 進行橢圓曲線運算。例如，可以使用 `EC_POINT_mul()` 函數生成公鑰。

Figure 4. Elliptic curve cryptography: visualizing the multiplication of a point  $G$  by an integer  $k$  on an elliptic curve

## 比特幣地址

比特幣地址是一串數字和字符，可以與任何想要向你匯款的人分享。從公鑰生成的地址由一串數字和字母組成，從數字“1”開始。以下是一個比特幣地址的例子：

```
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
```

比特幣地址是交易中最常見的資金“接收者”地址。如果我們將比特幣交易與紙質支票進行比較，那麼比特幣地址就是受益人，這是“支付給誰”後面要填寫的。對紙質支票來說，受益人有時可以是銀行賬戶的持有人，但也可以包括公司，機構甚至現金。由於紙質支票不需要指定賬戶，而是使用抽象名稱作為資金的接收者，所以它們是非常靈活的支付工具。比特幣的交易使用類似的抽象：比特幣地址，從而使它們非常靈活。比特幣地址可以表示私鑰/公鑰對兒的所有者，也可以表示其他內容，比如付款腳本，我們將在[p2sh]中看到。現在，讓我們來看一個簡單的例子，一個代表公鑰的比特幣地址。

比特幣地址是由公鑰單向加密雜湊而來的。“雜湊演算法”是一種單向函數，可以為任意大小的輸入產生指紋或“雜湊”。加密雜湊函數廣泛用於比特幣：比特幣地址，腳本地址和挖礦PoW驗證演算法。用於從公鑰生成比特幣地址的演算法是“安全雜湊演算法”（SHA）和“RACE完整性基元評估訊息摘要演算法”（RIPEMD），具體來說是SHA256和RIPEMD160。

從公鑰  $K$  開始，我們計算它的SHA256雜湊值，然後再計算結果的RIPEMD160雜湊值，產生一個160位（20字節）的數字：

$$\begin{aligned} & \text{\begin{equation} A = \text{RIPEMD160(SHA256}(K)\text{)} \end{equation}} \\ & \end{aligned}$$

其中  $K$  是公鑰， $A$  是生成的比特幣地址。

Tip

比特幣地址與公鑰不一樣。比特幣地址是使用單向函數從公鑰匯出的。

比特幣地址幾乎總是被編碼為“Base58Check”（參見[Base58 和 Base58Check 編碼](#)），該地址使用58個字符（Base58數字系統）和校驗和來提供可讀性，避免模糊不清，防止地址轉錄和輸入中的錯誤。Base58Check也可用在其他需要用戶閱讀並正確轉錄數字（比如比特幣地址，私鑰，加密密鑰或腳本雜湊）的地方。在下一節中，我們將研究[Base58Check編碼和解碼的機制以及由此產生的表示](#)。[Public key to bitcoin address: conversion of a public key into a bitcoin address](#) 說明瞭公鑰轉換為比特幣地址的過程。

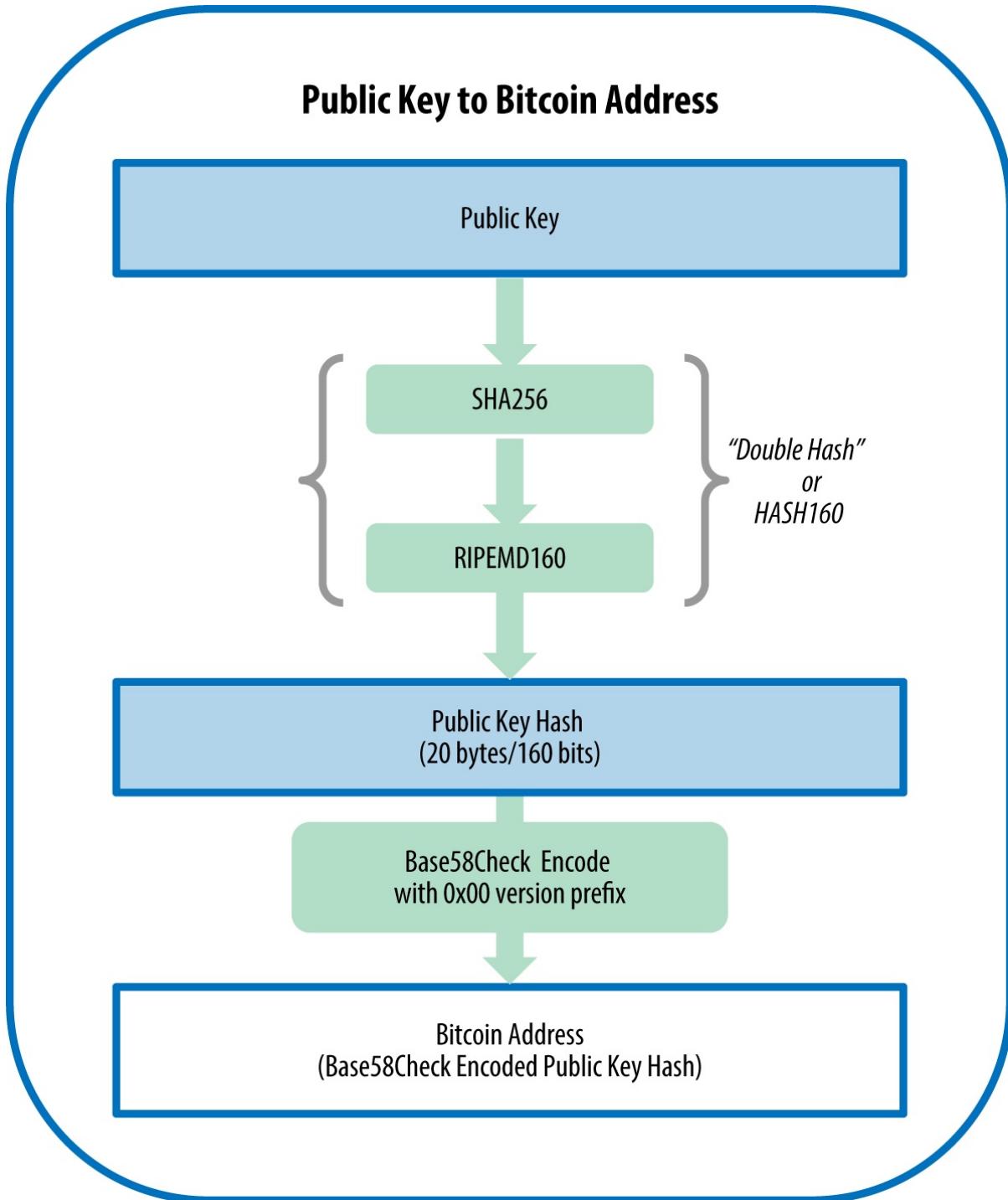


Figure 5. Public key to bitcoin address: conversion of a public key into a bitcoin address

## Base58 和 Base58Check 編碼

為了使用少量的符號，以緊湊的形式展示很長的數字，許多電腦系統使用基數（進制）高於10的混合字母數字表示。例如，傳統的十進制系統使用0到9的10個數字，十六進制使用16個（字母A到F作為六個附加符號）。以十六進制格式表示的數字比等效的十進製表示更短。更加緊湊的Base64表示使用26個小寫字母，26個大寫字母，10個數字和另外2個字符（如“+”和“/”）在基於文本的媒體（如電子郵件）上傳輸二進制數據。Base64最常用於向電子郵件添加二進制附件。Base58是一種基於文本的二進制編碼格式，用於比特幣和許多其他密碼貨幣。它在緊湊表示，可讀性和錯誤檢測與預防之間提供了平衡。Base58是Base64的一個子集，使用大小寫字母和數字，省略了一些經常被混淆的，或在使用某些字體顯示時看起來相同的。具體來說，相比Base64，Base58沒有0（數字0），O（大寫o），l（小寫L），I（大寫i）和符號“+”和“/”。[Bitcoin's Base58 alphabet](#)是完整的Base58字母表。

## Example 2. Bitcoin's Base58 alphabet

```
123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
```

為了增加防範輸入錯誤或轉錄錯誤的額外安全性，Base58Check是有內置錯誤校驗程式碼的Base58編碼格式，經常在比特幣中使用。校驗和是添加到編碼數據末尾的四個字節。校驗和來自編碼數據的雜湊雜湊值，可用於檢測和防止轉錄和輸入錯誤。當使用Base58Check程式碼時，解碼軟體將計算數據的校驗和並將其與程式碼中包含的校驗和進行比較。如果兩者不匹配，則會引入錯誤並且Base58Check數據無效。這可以防止錯誤的比特幣地址被錢包軟體接收，導致資金損失。

要將數據（數字）轉換為Base58Check格式，我們首先為數據添加一個名為“版本字節”的前綴，以便輕鬆識別編碼數據的類型。例如，比特幣地址的前綴為零（十六進制中的0x00），而編碼私鑰時使用的前綴為128（十六進制中的0x80）。常用的版本前綴參見 [Base58Check version prefix and encoded result examples](#)。

接下來，我們計算“double-SHA”校驗和，在前面的結果（前綴和數據）上應用兩次SHA256雜湊演算法：

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

在產生的32字節雜湊（hash-of-a-hash）中，我們只取前四個字節，作為錯誤檢查程式碼或校驗和。將校驗和追加到最後。

結果由三項組成：前綴，數據和校驗和。該結果使用前面描述的Base58字母表進行編碼。[Base58Check encoding: a Base58, versioned, and checksummed format for unambiguously encoding bitcoin data](#) 展示了Base58Check編碼過程。

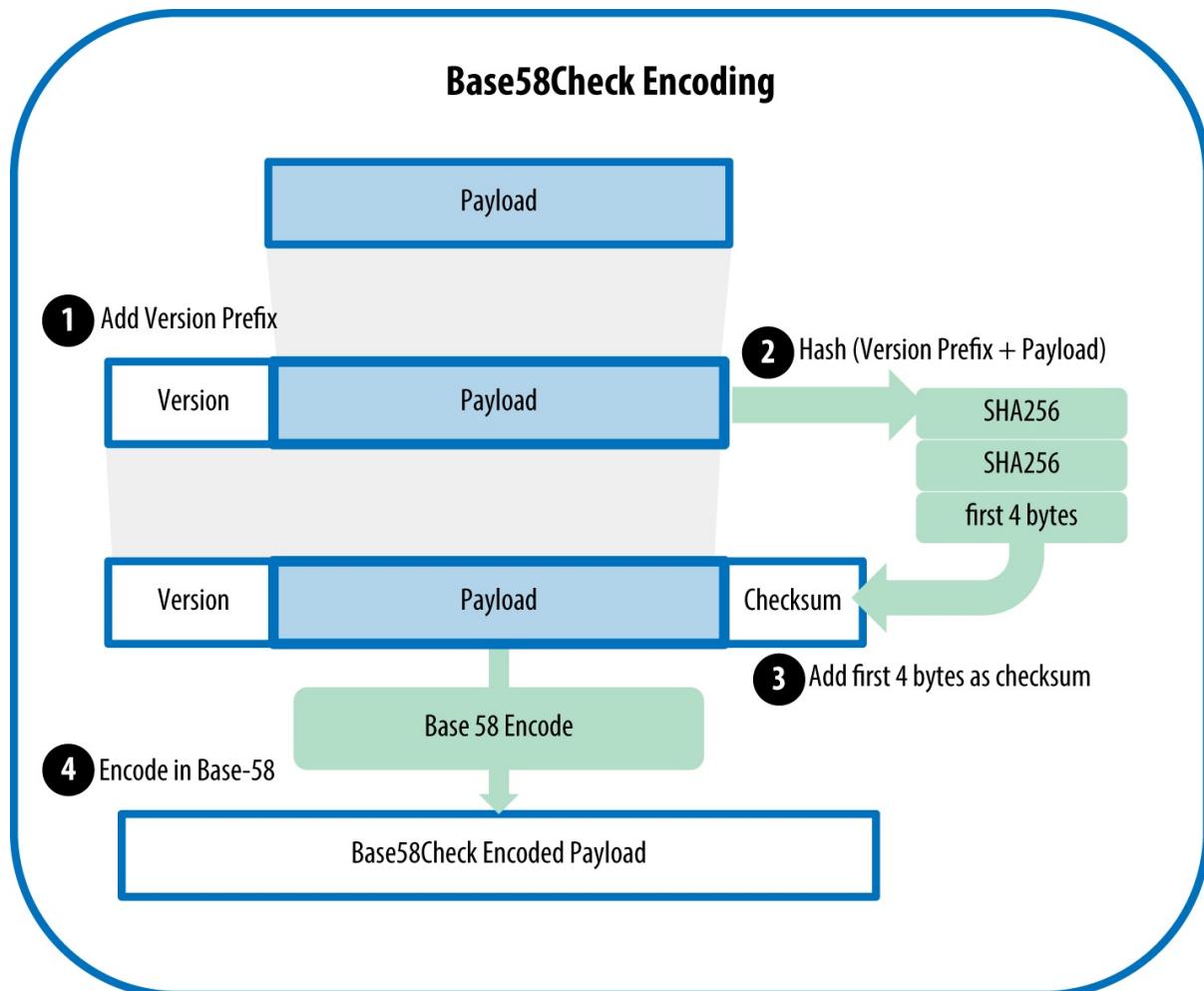


Figure 6. Base58Check encoding: a Base58, versioned, and checksummed format for unambiguously encoding bitcoin data

在比特幣中，大部分呈現給用戶的數據都是Base58Check編碼的，以使其緊湊，易於閱讀並易於檢測錯誤。

Base58Check編碼中的版本前綴用於創建容易區分的格式。這些字符使人們很容易得知編碼數據的類型以及如何使用它。例如，Base58Check編碼的比特幣地址以1開頭，Base58Check編碼的密鑰錢包匯入格式（WIF）以5開頭。一些示例版本前綴和Base58字符顯示在 [Base58Check version prefix and encoded result examples](#)中。

Table 1. Base58Check version prefix and encoded result examples

Type	Version prefix (hex)	Base58 result prefix
Bitcoin Address	0x00	1
Pay-to-Script-Hash Address	0x05	3
Bitcoin Testnet Address	0x6F	m or n
Private Key WIF	0x80	5, K, or L
BIP-38 Encrypted Private Key	0x0142	6P
BIP-32 Extended Public Key	0x0488B21E	xpub

## 密鑰格式

私鑰和公鑰都可以用不同的格式表示。即使這些格式看起來不同，但它們的編碼相同。這些格式主要用於使人們輕鬆閱讀和轉錄密鑰而不會引入錯誤。

### 私鑰格式

私鑰可以用不同的格式表示，所有這些格式都對應於相同的256位數字。[Private key representations \(encoding formats\)](#) 顯示了用於表示私鑰的三種常用格式。不同的格式用在不同的情況。十六進制和原始二進制格式在軟體內部使用，很少向用戶顯示。WIF用於在錢包之間匯入/匯出，並經常用於私鑰的QR碼（條形碼）表示。

Table 2. Private key representations (encoding formats)

Type	Prefix	Description
Raw	None	32 bytes
Hex	None	64 hexadecimal digits
WIF	5	Base58Check encoding: Base58 with version prefix of 128- and 32-bit checksum
WIF-compressed	K or L	As above, with added suffix 0x01 before encoding

[Example: Same key, different formats](#) 展示了三種編碼形式的私鑰。

Table 3. Example: Same key, different formats

Format	Private key
Hex	1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn

WIF-compressed	KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawrtJ
----------------	---

所有這些表示都是顯示相同數字和相同私鑰的不同方式。它們看起來不同，但任何一種格式都可以輕鬆轉換為任何其他格式。請注意，“原始二進制”沒有顯示在[Example: Same key, different formats](#)中。

我們使用 Bitcoin Explorer 的 wif-to-ec 命令（參見[appdx\\_bx](#)）來演示兩個WIF密鑰代表相同的私鑰：

```
$ bx wif-to-ec 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd

$ bx wif-to-ec KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawrtJ
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
```

### Base58Check解碼

Bitcoin Explorer 命令（參見[appdx\\_bx](#)）讓我們很容易通過編寫shell腳本和命令行“管道”，操作比特幣密鑰，地址和交易。你可以使用Bitcoin Explorer在命令行上解碼Base58Check格式。

我們使用 base58check-decode 命令解碼未壓縮的密鑰：

```
$ bx base58check-decode 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
wrapper
{
  checksum 4286807748
  payload 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
  version 128
}
```

結果包含密鑰的數據內容，WIF版本前綴128，以及校驗和。

請注意，壓縮密鑰的“數據內容”附加了後綴01，表示派生的公鑰將被壓縮：

```
$ bx base58check-decode KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawrtJ
wrapper
{
  checksum 2339607926
  payload 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01
  version 128
}
```

### 將十六進制編碼為Base58Check

要編碼到Base58Check，與上一個命令相對，我們使用Bitcoin Explorer的 base58check-encode 命令（請參見[appdx\\_bx](#)）並提供十六進制私鑰，以及WIF版本的前綴128：

```
bx base58check-encode
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd --version 128
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

### 從十六進制（壓縮的密鑰）編碼為Base58Check

要將“壓縮”的私鑰（請參見[壓縮的私鑰](#)）編碼為Base58Check，要將後綴01附加到十六進制密鑰後面，然後按照之前的方式進行編碼：

```
$ bx base58check-encode
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01 --version 128
KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

生成的WIF-compressed格式以“K”開頭，表示內部的私鑰具有後綴“01”，將僅用於生成壓縮的公鑰（請參閱[壓縮的公鑰](#)）。

## 公鑰格式

公鑰也能以不同的方式呈現，通常是*compressed*或*uncompressed*公鑰。

如之前所見，公鑰是由一對座標（x，y）組成的橢圓曲線上的一個點。它通常帶有前綴04，後跟兩個256位數字：一個是該點的x座標，另一個是y座標。前綴04表示未壓縮的公鑰，02或03開頭表示壓縮的公鑰。

這是我們先前創建的私鑰生成的公鑰，顯示為座標 x 和 y：

```
x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

這是以520位數字（130十六進制數字）表示的公鑰，結構為 04 x y：

```
K = 04F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A+
07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

## 壓縮的公鑰

壓縮公鑰被引入到比特幣中，以減少交易處理的大小並節省儲存空間。大多數交易包括公鑰，這是驗證所有者憑證並花費比特幣所需的。每個公鑰需要520位（前綴 + x + y），每個區塊有幾百個交易，每天產生千上萬的交易時，會將大量數據添加到區塊鏈中。

正如我們在[公鑰](#)看到的那樣，公鑰是橢圓曲線上的一個點（x，y）。因為曲線表達了一個數學函數，所以曲線上的一個點代表該方程的一個解，因此，如果我們知道x座標，我們可以通過求解方程來計算y座標  $y^2 \bmod p = (x^3 + 7) \bmod p$ 。這允許我們只儲存公鑰的x座標，省略y座標並減少密鑰的大小和所需的256位空間。在每次交易中，幾乎減少了50%的尺寸，加起來可以節省大量的數據！

未壓縮的公鑰的前綴為04，壓縮的公鑰以02或03前綴開頭。讓我們看看為什麼有兩個可能的前綴：因為方程的左邊是 $y^2$ ，所以y的解是一個平方根，它可以具有正值或負值。從視覺上來說，這意味著生成的y座標可以在x軸的上方或下方。從[An elliptic curve](#)中的橢圓曲線圖可以看出，曲線是對稱的，這意味著它在x軸上像鏡子一樣反射。因此，雖然我們可以省略y座標，但我們必須儲存y的sign（正數或負數）；換句話說，我們必須記住它高於或低於x軸，因為每個選項代表不同的點和不同的公鑰。當在質數階p的有限域上以二進制演算法計算橢圓曲線時，y座標是偶數或奇數，如前所述，它對應於正/負號。因此，為了區分y的兩個可能值，我們儲存一個壓縮公鑰，如果y是偶數，則前綴為02；如果是奇數，則儲存前綴為03，從而允許軟體從x座標正確推匯出y座標，並將公鑰解壓為該點的完整座標。[Public key compression](#)中說明瞭公鑰的壓縮。

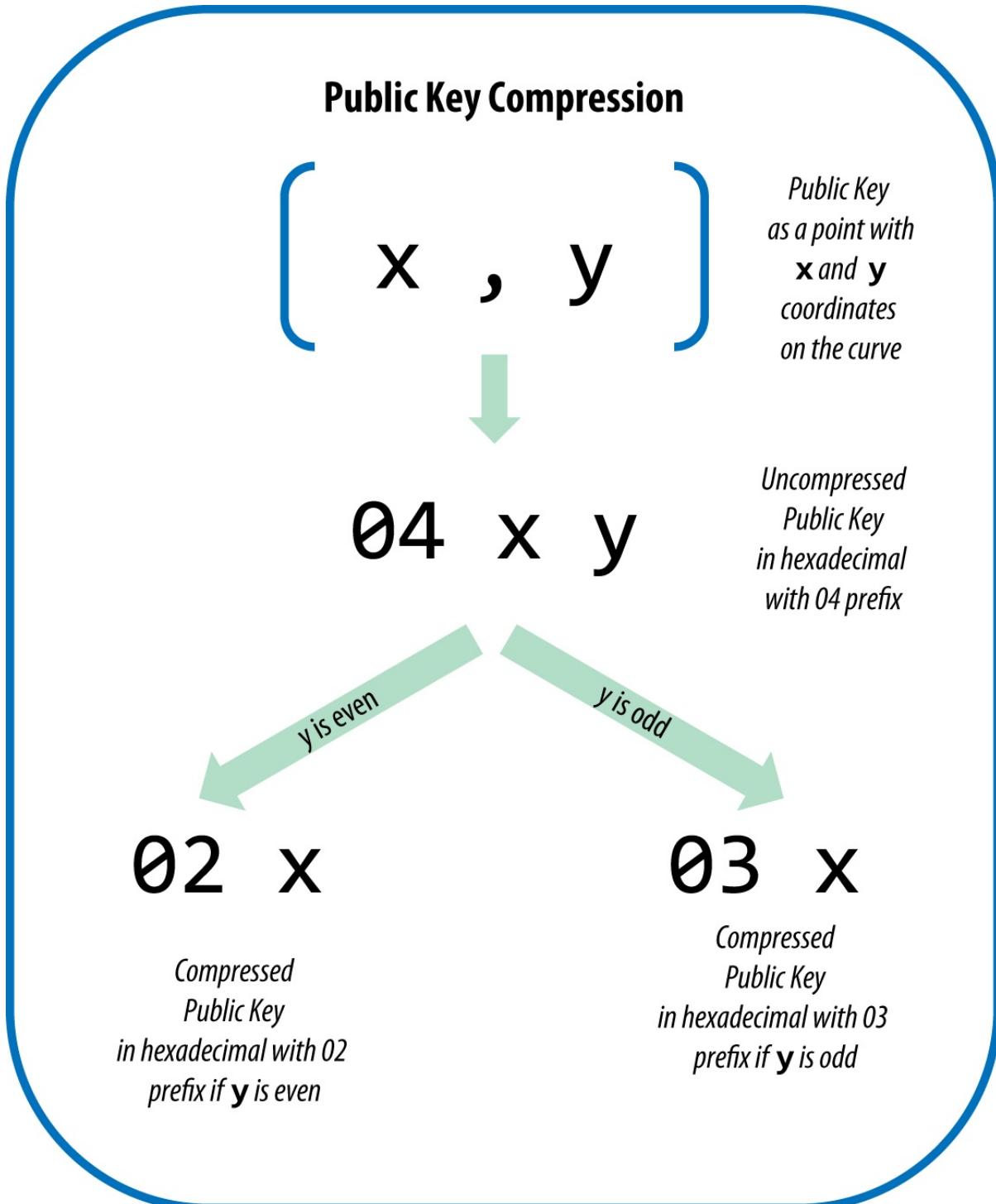


Figure 7. Public key compression

以下先前生成的公鑰，顯示為以264位（66位十六進制數字）儲存的壓縮公鑰，前綴03表示y座標為奇數：

```
K = 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
```

這個壓縮的公鑰對應於相同的私鑰，表示它是從相同的私鑰生成的。但是，它看起來與未壓縮的公鑰不同。更重要的是，如果我們使用雙雜湊函數（RIPEMD160（SHA256（K））將此壓縮公鑰轉換為比特幣地址，它將生成一個不同的比特幣地址。這可能會造成混淆，因為這意味著單個私鑰可以產生以兩種不同格式（壓縮和未壓縮）表示的公鑰，這兩種格式產生兩個不同的比特幣地址。但是，兩個比特幣地址的私鑰是相同的。

壓縮公鑰正在逐漸成為比特幣客戶端的預設設置，這對減少交易和區塊鏈的規模具有重大影響。但是，並非所有客戶端都支持壓縮的公鑰。支持壓縮公鑰的較新客戶端必須考慮來自不支持壓縮公鑰的較舊客戶端的交易。當錢包應用從另一個比特幣錢包應用匯入私鑰時，這一點尤其重要，因為新錢包需要掃描區塊鏈以查找與這些匯入的密鑰相對應的交易。比特幣錢包應該掃描哪些比特幣地址？由未壓縮的公鑰生成的比特幣地址，還是由壓縮公鑰生成的比特幣地址？兩者都是有效的比特幣地址，並且可以用私鑰簽名，但它們是不同的地址！

要解決此問題，從錢包中匯出私鑰時，用WIF表示它們在新比特幣錢包中以不同方式實現，表明這些私鑰已用於生成*compressed*公鑰和*compressed*比特幣地址。這允許匯入的錢包區分源自舊的或較新的錢包的私鑰，並分別在區塊鏈中搜索與未壓縮的或壓縮的公共密鑰對應的比特幣地址的交易。下一節我們來看看更詳細的工作原理。

### 壓縮的私鑰

諷刺的是，術語“壓縮私鑰”是一種用詞不當，因為當私鑰以“WIF-compressed”的形式匯出時，它實際上比“未壓縮”的私鑰多一個字節。這是因為私鑰增加了一個字節的後綴（在[Example: Same key, different formats](#)中以十六進制顯示為01），表示私鑰來自較新的錢包並且應該僅用於產生壓縮的公鑰。私鑰本身並不壓縮，也不能被壓縮。術語“壓縮私鑰”實際上是指“只能從私鑰匯出壓縮的公鑰”，而“未壓縮的私鑰”實際上是指“只能從私鑰匯出未壓縮的公鑰才”。你只應將匯出格式稱為“WIF-compressed”或“WIF”，不要將私鑰本身稱為“壓縮”的以避免進一步混淆。

[Example: Same key, different formats](#) 展示了相同的密鑰以 WIF 和 WIF-compressed 格式編碼。

Table 4. Example: Same key, different formats

Format	Private key
Hex	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Hex-compressed	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD01
WIF-compressed	KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawrtJ

十六進制壓縮的私鑰格式在結尾處有一個額外的字節（十六進制中的01）。雖然Base58編碼的版本前綴對於WIF和WIF-compressed格式都是相同的（0x80），但在數字末尾添加一個字節會導致Base58編碼的第一個字符從5更改为K或L。可以把它看作Base58相當於數字100和數字99之間的十進制編碼差異。雖然100是比99更長的一位數字，但它前綴是1而不是9。長度的變化會影響前綴。在Base58中，隨著數字長度增加一個字節，前綴5變為K或L。

請記住，這些格式不可互換使用。在實現壓縮公鑰的新錢包中，私鑰只能以WIF-compressed方式匯出（帶有K或L前綴）。如果錢包是較舊的實現並且不使用壓縮的公鑰，則私鑰只能以WIF形式匯出（帶有前綴5）。這裡的目標是嚮匯入這些私鑰的錢包發出信號，告知它是否必須在區塊鏈中搜索壓縮或未壓縮的公鑰和地址。

如果一個比特幣錢包能夠實現壓縮公鑰，它將在所有交易中使用這些公鑰。錢包中的私鑰將用於派生曲線上的公鑰，並壓縮。壓縮後的公鑰將用於生成比特幣地址用於交易。從實現壓縮公鑰的新錢包中匯出私鑰時，將修改WIF，並在私鑰上添加一個字節的後綴01。由此產生的Base58Check編碼的私鑰稱為“WIF-compressed”，並以字母K或L開頭，而不是像來自舊錢包的WIF編碼一樣以“5”開頭。

Tip

“壓縮私鑰”是一個誤用！它們沒有被壓縮；相反，WIF-compressed意味著密鑰只能用於派生壓縮的公鑰及其相應的比特幣地址。諷刺的是，一個“WIF-compressed”編碼私鑰多了1個字節，因為它具有附加的01後綴，可以將其與“未壓縮”的區別開來。

## 用 C++ 實現密鑰和地址

讓我們看一下創建比特幣地址的完整過程，從私鑰到公鑰（橢圓曲線上的一個點），再到雙重雜湊地址，最後是Base58Check編碼。[Creating a Base58Check-encoded bitcoin address from a private key](#) 中的C++程式碼顯示了完整的過程。程式碼示例使用了[\[alt\\_libraries\]](#) 中介紹的libbitcoin庫來提供一些幫助函數。

#### Example 3. Creating a Base58Check-encoded bitcoin address from a private key

```
link:code/addr.cpp[]
```

該程式碼使用預定義的私鑰在每次運行時產生相同的比特幣地址，如[Compiling and running the addr code](#) 所示。

#### Example 4. Compiling and running the addr code

```
# Compile the addr.cpp code
$ g++ -o addr addr.cpp -std=c++11 $(pkg-config --cflags --libs libbitcoin)
# Run the addr executable
$ ./addr
Public key: 0202a406624211f2abbd68da3df929f938c3399dd79fac1b51b0e4ad1d26a47aa
Address: 1PRTTaJesdNovgne6Ehcd1fpEdX7913CK
```

Tip

[Compiling and running the addr code](#) 中的程式碼從一個壓縮的公鑰（參見[壓縮的公鑰](#)）生成了一個比特幣地址 (1PRTT...)。如果你使用未壓縮的公鑰，它會產生不同的比特幣地址 (14K1y...).

## 用 Python 實現密鑰和地址

Python中最全面的比特幣庫是Vitalik Buterin寫的[pybitcointools](#)。在[Key and address generation and formatting with the pybitcointools library](#)中，我們使用 pybitcointools 函數庫 (imported as "bitcoin") 以各種格式生成和顯示密鑰與地址。

#### Example 5. Key and address generation and formatting with the pybitcointools library

```
link:code/key-to-address-ecc-example.py[]
```

[Running key-to-address-ecc-example.py](#) 展示了運行結果。

#### Example 6. Running key-to-address-ecc-example.py

```
$ python key-to-address-ecc-example.py
Private Key (hex) is:
3aba4162c7251c891207b747840551a71939b0de081f85c4e44cf7c13e41daa6
Private Key (decimal) is:
26563230048437957592232553826663696440606756685920117476832299673293013768870
Private Key (WIF) is:
5JG9hT3beGTJuUAmCQEmNaxAuMacCTfxuw1R3FCXig23RQHMr4K
Private Key Compressed (hex) is:
3aba4162c7251c891207b747840551a71939b0de081f85c4e44cf7c13e41daa601
Private Key (WIF-Compressed) is:
KyBsPXXtUVd82av65KZkrGrWi5qLMah5SdNq6uftawDbgKa2wv6S
Public Key (x,y) coordinates is:
(41637322786646325214887832269588396900663353932545912953362782457239403430124L,
16388935128781238405526710466724741593761085120864331449066658622400339362166L)
Public Key (hex) is:
045c0de3b9c8ab18dd04e3511243ec2952002dbfadec864b9628910169d9b9b00ec↔
243bcefdd4347074d44bd7356d6a53c495737dd96295e2a9374bf5f02ebfc176
Compressed Public Key (hex) is:
025c0de3b9c8ab18dd04e3511243ec2952002dbfadec864b9628910169d9b9b00ec
Bitcoin Address (b58check) is:
```

```
1thMirt546nngXqyPEz532S8fLwbozud8
Compressed Bitcoin Address (b58check) is:
14cxpo3MBCYYWCgF74SWTdcmxipnGUspw3
```

[A script demonstrating elliptic curve math used for bitcoin keys](#) 是另外一個例子，使用橢圓曲線計算的 Python ECDSA 庫。

Example 7. A script demonstrating elliptic curve math used for bitcoin keys

```
link:code/ec-math.py[]
```

[Installing the Python ECDSA library and running the ec\\_math.py script](#) shows the output produced by running this script.

#### Note

[A script demonstrating elliptic curve math used for bitcoin keys](#) 使用 os.urandom, 體現了底層操作系統提供的密碼學安全的隨機數生成器 (CSRNG)。警告：根據操作系統的不同，os.urandom 可能無法以足夠的安全性，並且可能不適合生成高質量的比特幣密鑰。

Example 8. Installing the Python ECDSA library and running the ec\_math.py script

```
$ # Install Python PIP package manager
$ sudo apt-get install python-pip
$ # Install the Python ECDSA library
$ sudo pip install ecdsa
$ # Run the script
$ python ec-math.py
Secret:
38090835015954358862481132628887443905906204995912378278060168703580660294000
EC point:
(70048853531867179489857750497606966272382583471322935454624595540007269312627,
105262206478686743191060800263479589329920209527285803935736021686045542353380)
BTC public key: 029ade3effb0a67d5c8609850d797366af428f4a0d5194cb221d807770a1522873
```

## 高級的密鑰和地址

在下面的章節中，我們將看看高級形式的密鑰和地址，例如加密私鑰，腳本和多重簽名地址，虛榮地址和紙錢包。

### 加密私鑰 (BIP-38)

私鑰必須保密，但對私鑰的“保密性”的需求是在實踐中很難實現，因為它與同樣重要的 可用性 安全目標相衝突。當你需要備份私鑰以避免丟失私鑰時，保持私鑰私密性更加困難。儲存在通過密碼加密的錢包中可能是安全的，但該錢包需要備份。有時，用戶需要將密鑰從一個錢包移動到另一個錢包 - 例如，升級或替換錢包軟體。私鑰的備份也可能儲存在紙張上（請參見[紙錢包](#)）或外部儲存介質（如USB閃存驅動器）中。但是如果備份本身被盜或丟失怎麼辦？這些相互衝突的安全目標促成了一種便攜式和便捷的加密私鑰的標準，這種加密方式可以被許多不同的錢包和比特幣客戶端理解，通過BIP-38標準化(參見 [\[apdxbtcoinimpproposals\]](#))。

BIP-38 提出了一個通用標準，用密碼對私鑰進行加密，並使用Base58Check對其進行編碼，以便它們可以安全地儲存在備份介質上，在錢包之間安全地傳輸，或保存在密鑰可能暴露的任何其他情況下。加密標準使用高級加密標準 (AES)，這是NIST建立的標準，廣泛用於商業和軍事應用的數據加密實現。

BIP-38加密方案將通常使用WIF編碼（前綴為“5”的Base58Check字串）的比特幣私鑰作為輸入。此外，BIP-38加密方案需要一個密碼短語，通常由幾個詞或一串複雜的字母數字字符組成。BIP-38加密的結果是以前綴6P開始的Base58Check加密私鑰。如果你看到一個以6P開頭的密鑰，則該密鑰是加密的，需要密碼才能將其轉換（解密）為可

用於任何錢包的WIF格式的私鑰（前綴5）。許多錢包應用程式現在可識別BIP-38加密的私鑰，並提示用戶輸入密碼以解密並匯入密鑰。第三方應用程式，例如非常實用的基於瀏覽器的應用程式 [Bit Address](#) (Wallet Details tab)，可以用來解密BIP-38密鑰。

BIP-38加密密鑰最常見的用例是可用於備份私鑰的紙錢包。只要用戶選擇強密碼，帶有BIP-38加密私鑰的紙錢包就非常安全，並且是創建離線比特幣儲存（也稱為“冷儲存”）的好方法。

使用[bitaddress.org](#)測試[Example of BIP-38 encrypted private key](#)中的加密密鑰，瞭解如何通過輸入密碼來獲取解密的密鑰。

Table 5. Example of BIP-38 encrypted private key

<b>Private Key (WIF)</b>	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
<b>Passphrase</b>	MyTestPassphrase
<b>Encrypted Key (BIP-38)</b>	6PRTHL6mWa48xSopbU1cKrVjpKbBZxcLRRCdctLJ3z5yxE87MobKoXdTsJ

## 支付給腳本的雜湊（Pay-to-Script Hash，P2SH）和多重簽名地址

如我們所知，傳統的以數字“1”開頭的比特幣地址，源自公鑰，而公鑰又是通過私鑰生成的。儘管任何人都可以將比特幣發送到“1”地址，但只能通過提供相應的私鑰簽名和公鑰雜湊來花費比特幣。

以數字“3”開頭的比特幣地址是支付給腳本的雜湊（P2SH）地址，有時被錯誤地稱為多重簽名或多重地址。它們將比特幣交易的受益人指定為腳本的雜湊，而不是公鑰的所有者。該功能在2012年1月由BIP-16提出（參見[\[appdx\\_bitcoinimproposals\]](#)），正在被廣泛採用，因為它提供了向地址本身添加功能的機會。與將資金“發送”到傳統的“1”比特幣地址（也稱為付費至公鑰的雜湊（P2PKH））的交易不同，發送至“3”地址的資金需要的不僅僅是一個公鑰雜湊和一個私鑰簽名作為所有權證明。這些要求是創建地址時在腳本中指定的，並且對該地址的所有輸入也要按照相同的要求進行設置。

P2SH地址由交易腳本創建，該腳本定義誰可以使用交易的輸出（有關更多詳細訊息，請參見[\[p2sh\]](#)）。對P2SH地址進行編碼使用與創建比特幣地址時用到的相同的雙重雜湊函數，只是應用於腳本而不是公鑰：

```
script hash = RIPEMD160(SHA256(script))
```

生成的“腳本雜湊”使用前綴5進行Base58Check編碼，導致編碼地址以3開頭。P2SH地址的示例：

3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM，可以使用Bitcoin Explorer的 script-encode, sha256, ripemd160, 和 base58check-encode 命令（參見[\[appdx\\_bx\]](#)）生成：

```
$ echo \
'DUP HASH160 [89abcdefabbaabbaabbaabbaabbaabbaabba] EQUALVERIFY CHECKSIG' >
script
$ bx script-encode < script | bx sha256 | bx ripemd160 \
| bx base58check-encode --version 5
3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM
```

Tip

P2SH不一定與多重簽名交易相同。P2SH地址常用來表示多重簽名腳本，但它也可能表示其他類型交易的腳本。

### 多重簽名地址與P2SH

目前，P2SH功能最常見的實現是多重簽名地址腳本。顧名思義，底層腳本需要多個簽名才能證明所有權，才能花費資金。比特幣多重簽名特徵被設計為需要來自總共N個密鑰的M個簽名（也稱為“閾值”），稱為M-N多重簽名，其中M等於或小於N。例如，[\[ch01\\_intro\\_what\\_is\\_bitcoin\]](#) 中的咖啡店老闆Bob可以使用一個多重簽名地址，要求屬於他的一把鑰匙和屬於他的妻子的一把鑰匙中的一個簽名，以確保他們中的任何一個簽字可以簽署鎖定到這個地址的一筆交易輸出。這與在傳統銀行中實施的“聯名賬戶”類似，夫妻的任一方都可以花費一個簽名。再例如，網頁設計師Gopesh，可能會為其業務提供2/3的多重簽名地址，確保除非至少有兩個業務合作伙伴簽署交易，否則不會花費任何資金。

我們將在 [\[transactions\]](#) 中探討如何創建花費 P2SH（和多重簽名）地址的資金的交易。

## 虛榮地址 (Vanity Addresses)

虛榮地址是包含人類可讀訊息的有效比特幣地址。例如，1LoveBPzzD72PUXLzCkYAtGFYmK5vYNR33 是一個有效的地址，其中包含形成單詞“Love”的字母作為前四個Base-58字母。虛擬地址需要生成並測試數十億個候選私鑰，直到找到具有所需模式的比特幣地址。雖然在虛榮生成演算法中有一些優化，但這個過程主要包括隨機選擇一個私鑰，匯出公鑰，匯出比特幣地址，並檢查它是否符合所需的虛擬模式，重複數十億次，直到匹配被發現。

一旦找到與所需模式相匹配的虛擬地址，所有者就可以使用從中得到的私鑰來並以與其他地址完全相同的方式使用比特幣。虛擬地址的安全性不低於其他地址。它們依賴於與其他地址相同的橢圓曲線加密（ECC）和SHA。

在 [\[ch01\\_intro\\_what\\_is\\_bitcoin\]](#) 中，我們介紹了在菲律賓開展業務的兒童慈善機構Eugenia。假設Eugenia正在組織比特幣籌款活動，並希望使用虛擬比特幣地址來宣傳籌款活動。Eugenia將創建一個以“1Kids”開頭的虛榮地址來宣傳兒童慈善籌款活動。讓我們看看這個虛榮的地址如何創建，以及它對於Eugenia慈善機構的安全意味著什麼。

### 生成虛榮地址

認識到比特幣地址僅僅是Base58字母表中的符號代表的數字很重要。搜索諸如“1Kids”之類的模式可以被視為搜索範圍從 1Kids1111111111111111111111111111 到 +1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz + 的地址。該範圍內的地址約有  $58^{29}$  (約 $1.4 * 10 ^ {51}$ ) 個，全部以“1Kids”開頭。[The range of vanity addresses starting with "1Kids"](#) 顯示了具有前綴1Kids的地址範圍。

Table 6. The range of vanity addresses starting with "1Kids"

<b>From</b>	1Kids1111111111111111111111111111
	1Kids1111111111111111111111111112
	1Kids1111111111111111111111111113
	...
<b>To</b>	1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzzz

讓我們把“1Kids”模式當作數字，看看我們在比特幣地址中可能發現這種模式的概率（參見[The frequency of a vanity pattern \(1KidsCharity\) and average search time on a desktop PC](#)）。一臺普通的臺式電腦個人電腦，沒有任何專門的硬體，可以每秒搜索約100000個密鑰。

Table 7. The frequency of a vanity pattern (1KidsCharity) and average search time on a desktop PC

Length	Pattern	Frequency	Average search time
1	1K	1 in 58 keys	< 1 milliseconds
2	1Ki	1 in 3,364	50 milliseconds
3	1Kid	1 in 195,000	< 2 seconds
4	1Kids	1 in 11 million	1 minute

5	1KidsC	1 in 656 million	1 hour
6	1KidsCh	1 in 38 billion	2 days
7	1KidsCha	1 in 2.2 trillion	3–4 months
8	1KidsChar	1 in 128 trillion	13–18 years
9	1KidsChari	1 in 7 quadrillion	800 years
10	1KidsCharit	1 in 400 quadrillion	46,000 years
11	1KidsCharity	1 in 23 quintillion	2.5 million years

如你所見，即使Eugenia能夠訪問幾千臺電腦，也不能很快創建“1KidsCharity”虛擬地址。每增加一個字符都會將難度增加58倍。超過7個字符的模式通常由專用硬體尋找，例如具有多個GPU的定製桌面電腦。這些往往是用於比特幣挖礦的“鑽井平臺”，為比特幣不再適合盈利，但可用於找到虛榮地址。GPU系統上的虛度搜索速度比通用CPU上的快很多個數量級。

找到虛榮地址的另一種方法是將工作外包給一個虛榮礦工池，例如 [Vanity Pool](#)。這是一項服務，允許那些使用GPU硬體的人為其他人搜索比特幣虛擬地址。僅需小額付款（本文寫作時為0.01比特幣或大約5美元），Eugenia可以將7個字符的模式虛擬地址搜索外包，並在幾個小時內獲得結果，而不必進行幾個月的CPU搜索。

生成虛擬地址是一個暴力搜索：嘗試一個隨機密鑰，檢查結果地址以查看它是否與所需模式匹配，重複直到成功。

[Vanity address miner](#) 顯示了一個“虛榮礦工”的例子，這是一個用C++編寫的用於查找虛榮地址的程序。這個例子使用了我們在[\[alt\\_libraries\]](#)中介紹的libbitcoin庫。

#### Example 9. Vanity address miner

```
link:code/vanity-miner.cpp[]
```

Note

[Compiling and running the vanity-miner example](#) 使用 `std::random_device`. 根據具體實現不同，它可能反映了底層操作系統提供的CSRNG。在類Unix等操作系統的情況下，它從`/dev/urandom`中提取。這裡使用的隨機數生成器用於演示目的，它不適合生成生產環境質量要求的比特幣密鑰，因為它沒有足夠的安全性來實現。

示例程式碼必須使用 C++ 編譯器編譯並鏈接libbitcoin庫（必須先安裝在該系統上）。要運行該示例，請運行不帶參數的 `vanity-miner` 可執行檔案（參見[Compiling and running the vanity-miner example](#)），它將嘗試查找以“1kid”開頭的虛擬地址。

#### Example 10. Compiling and running the vanity-miner example

```
$ # Compile the code with g++
$ g++ -o vanity-miner vanity-miner.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Run the example
$ ./vanity-miner
Found vanity address! 1KiDzkG4MxmovZryZRj8tK81oQRhbZ46YT
Secret: 57cc268a05f83a23ac9d930bc8565bac4e277055f4794cbd1a39e5e71c038f3f
$ # Run it again for a different result
$ ./vanity-miner
Found vanity address! 1Kidxr3wsmMzzouwXibKfwTYs5Pau8TUFn
Secret: 7f65bbbbe6d8caae74a0c6a0d2d7b5c6663d71b60337299a1a2cf34c04b2a623
# Use "time" to see how long it takes to find a result
$ time ./vanity-miner
Found vanity address! 1KidPWhKgGRQWD5PP5TAnGfDyfWp5yceXM
Secret: 2a802e7a53d8aa237cd059377b616d2bfcfa4b0140bc85fa008f2d3d4b225349
```

```

real    0m8.868s
user    0m8.828s
sys     0m0.035s

```

我們可以看到，我們使用Unix命令 time 來測量執行時間，示例程式碼需要幾秒鐘找到三字符模式“kid”的匹配項。更改源程式碼中的 search 模式並查看四或五個字符模式需要多長時間！

### 虛榮地址的安全性

虛榮地址可以用來增強和破壞安全性，它們確實是一把雙刃劍。作為提高安全性時，獨特的地址使得攻擊者難以用自己的地址替代你的地址，並欺騙客戶付錢給他們，而不是你。不幸的是，虛榮地址也使得任何人都可以創建一個地址，以便將任何隨機地址或甚至另一個虛榮地址重新排列，從而欺騙客戶。

Eugenia 可以發佈一個隨機生成的地址（例如 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy），人們可以向這個址發送給他們的捐款。或者，她可以生成一個以1Kids開頭的虛榮地址，以使其更具特色。

在這兩種情況下，使用單個固定地址（而不是為每個捐助者單獨生成動態地址）的風險之一是小偷可能滲透你的網站並用自己的地址替換它，從而將捐贈轉移給自己。如果你在多個不同的地方刊登了捐款地址，用戶可能會在進行付款之前直觀地檢查地址，以確保它與你的網站，電子郵件和傳單上看到的地址相同。像

1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy 這樣的隨機地址，普通用戶可能會檢查前幾個字符“1J7mdg”並確認地址匹配。使用虛名地址生成器，想竊取資金的人可以快速生成與前幾個字符匹配的地址，如[Generating vanity addresses to match a random address](#)所示。

Table 8. Generating vanity addresses to match a random address

<b>Original Random Address</b>	1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
<b>Vanity (4-character match)</b>	1J7md1QqU4LpctBetHS2ZoyLV5d6dShhEy
<b>Vanity (5-character match)</b>	1J7mdgYqyNd4ya3UEcq31Q7sqRMXw2XZ6n
<b>Vanity (6-character match)</b>	1J7mdg5WxGENmwyJP9xuGhG5KRzu99BBCX

那麼虛榮的地址是否能增加安全性呢？如果Eugenia生成虛擬地址 1Kids33q44erFfpeXrmDSz7zEqG2FesZEN，用戶可能會查看虛空模式單詞 和後續的 幾個字符，例如注意到地址的“1Kids33”部分。這會迫使攻擊者產生一個至少匹配六個角色（兩個以上）的虛榮地址，花費的努力比Eugenia花費4個字符虛榮心的努力高出3364倍 ( $58^2$ )。從根本上說，Eugenia花費的努力（或支付虛榮礦工池）“推動”攻擊者必須產生更長的模式虛榮。如果Eugenia支付一個虛榮礦工池產生一個8個字符的虛榮地址，攻擊者將被推入10個角色的領域，這在個人電腦上是不可行的，即使使用定製的虛榮採礦裝備或虛榮池也很昂貴。對於Eugenia而言，負擔得起的東西對於攻擊者來說是不可承受的，特別是如果潛在的欺詐收益不足以支付虛榮地址生成的代價。

### 紙錢包

紙錢包是印在紙上的比特幣私鑰。通常為方便起見，紙錢包還包括相應的比特幣地址，但這不是必須的，因為它可以用私鑰生成。紙錢包是創建備份或離線比特幣儲存（也稱為“冷儲存”）的非常有效的方式。作為備份機制，紙錢包可以防止由於電腦故障（如硬盤驅動器故障，被盜或意外刪除）而導致密鑰丟失。作為一種“冷儲存”機制，如果紙錢包密鑰是離線生成的，永遠不會儲存在電腦系統中，可以很好的防範黑客，按鍵記錄器和其他在線電腦威脅。

紙錢包可以有許多形狀，大小和設計，最基本的只是紙上的密鑰和地址。[Simplest form of a paper wallet—a printout of the bitcoin address and private key](#) 展示了紙錢包最簡單的形式。

Table 9. Simplest form of a paper wallet—a printout of the bitcoin address and private key

Public address	Private key (WIF)
1424C2F4bC9JidNjjTUZCbUxv6Sa1Mt62x	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn

例如位於`bitaddress.org`的客戶端JavaScript生成器的工具可以輕鬆生成紙錢包。此頁面包含生成密鑰和紙錢包所需的全部程式碼，即使與互聯網完全斷開。要使用它，請將HTML頁面保存在本地硬盤或外部USB儲存上，斷開互聯網並在瀏覽器中打開它。更好的是，使用乾淨的操作系統啟動電腦，例如可以從CD-ROM啟動的Linux操作系統。離線時使用此工具生成的任何密鑰都可以通過USB（非無線）在本地打印機上打印，從而創建紙錢包，其密鑰僅存在於紙張上，從未儲存在任何在線系統上。將這些紙錢包放入防火保險櫃中，並將比特幣“發送”至其比特幣地址，以實施簡單而高效的“冷儲存”解決方案。[An example of a simple paper wallet from bitaddress.org](#) 展示了從`bitaddress.org`網站生成的紙錢包。



Figure 8. An example of a simple paper wallet from `bitaddress.org`

簡單紙錢包的缺點是印刷的鑰匙容易被盜。能夠訪問紙張的小偷可以竊取它或拍攝鑰匙，即可控制這些鑰匙鎖定的比特幣。更復雜的紙錢包儲存系統使用BIP-38加密的私人密鑰。紙錢包上印有的鑰匙受到由主人記在腦中的密碼保護。沒有密碼，加密的密鑰就沒用了。紙錢包仍然優於密碼保護的錢包，因為密鑰從未在線並且必須從安全或其他有物理保護的儲存裝置中獲取。[An example of an encrypted paper wallet from bitaddress.org. The passphrase is "test."](#) 顯示在`bitaddress.org`網站上創建的帶有加密私鑰（BIP-38）的紙幣。



Figure 9. An example of an encrypted paper wallet from `bitaddress.org`. The passphrase is "test."

Warning	<p>儘管你可以多次將資金存入紙錢包，但你應該一次性收回所有資金，一次性花費。這是因為在解鎖和花費資金的過程中，如果花費少於全部金額，某些錢包可能會生成零錢地址。此外，如果你用來簽署交易的電腦受到威脅，可能會洩露私鑰。一次性花費紙錢包的全部餘額，可以降低密鑰洩漏的風險。如果你只需要少量資金，請在一筆交易中將剩餘資金髮送到一個新的紙錢包中。</p>
---------	--

紙錢包可以有許多不同的設計和尺寸，和不同的特徵。一些用來當作禮物贈送，並具有季節性主題，如聖誕節和新年主題。其他設計用於存放在銀行保險庫或帶有隱藏的密碼保護的保險箱中，或者使用不透明的刮擦貼紙，或者使用防篡改粘貼箔摺疊和密封。圖 `#paper_wallet_bpw` 到 `#paper_wallet_spw` 顯示具有安全和備份功能的各種紙錢包示例。

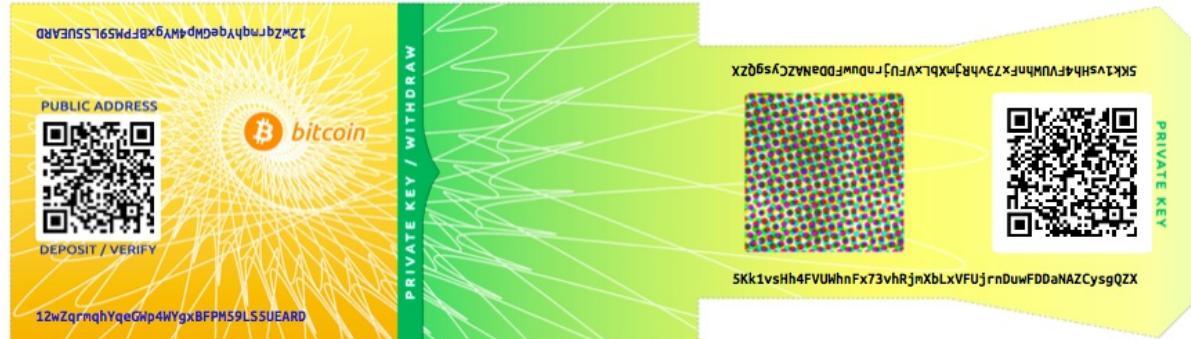


Figure 10. An example of a paper wallet from bitcoinpaperwallet.com with the private key on a folding flap



Figure 11. The bitcoinpaperwallet.com paper wallet with the private key concealed

其他設計還提供了鑰匙和地址的附加副本，形式為與票根類似的可拆卸存根，允許你儲存多個副本以防止火災，洪水或其他自然災害。



Figure 12. An example of a paper wallet with additional copies of the keys on a backup "stub"

## 錢包

在比特幣中，“錢包”一詞用於描述幾個不同的東西。

從較高的角度來說，錢包是用戶使用的應用程式，控制對用戶資金的訪問，管理密鑰和地址，追蹤餘額以及創建和簽署交易。

更狹義地，從開發者的角度來看，“錢包”一詞是指用於儲存和管理用戶密鑰的資料結構。

在本章中，我們將看看第二個含義，即錢包是私鑰的容器，通常以結構化檔案或簡單資料庫的形式實現。

### 錢包技術概述

在本節中，我們總結了用於構建用戶友好，安全和靈活的比特幣錢包的各種技術。

關於比特幣的一個常見誤解是比特幣錢包包含比特幣。實際上，錢包只包含密鑰。“比特幣”被記錄在比特幣網路的區塊鏈中。用戶通過使用錢包中的密鑰簽署交易來控制網路上的硬幣。從某種意義上說，比特幣錢包是一個密鑰串 *keychain*。

Tip

比特幣錢包包含鑰匙，而不是硬幣。每個用戶都有一個包含密鑰的錢包。錢包真的是包含私鑰/公鑰的鑰匙串（參見 [\[private\\_public\\_keys\]](#)）。用戶使用密鑰簽署交易，從而證明他們擁有交易的輸出（他們的比特幣）。比特幣以交易輸出的形式（通常記作vout或txout）儲存在區塊鏈中。

根據包含的密鑰是否彼此相關劃分，主要有兩種類型的錢包。

第一種是非確定性錢包 *nondeterministic wallet*，其中每個密鑰都是從隨機數中獨立生成的。密鑰不相互關聯。這種類型的錢包也被稱JBOK錢包（Just a Bunch Of Keys）。

第二種是確定性錢包 *deterministic wallet*，其中所有密鑰都來自單個主密鑰，稱為種子 *seed*。這種錢包中的所有密鑰都是相互關聯的，如果有原始種子，可以再次生成。確定性錢包中使用了許多密鑰派生 *key derivation* 方法。最常用的派生方法使用樹狀結構，並稱為分層確定性 *hierarchical deterministic* 錢包或HD錢包。

確定性錢包是從種子初始化的。為了使這些更容易使用，種子被編碼為英文單詞，也被稱為助記詞 *mnemonic code words*。

接下來的幾節將從較高的角度介紹這些技術。

### 非確定性（隨機）錢包

在第一個比特幣錢包（現在稱為Bitcoin Core）中，錢包是隨機生成的私鑰集合。例如，Bitcoin Core客戶端首次啟動時生成100個隨機私鑰，並根據需要生成更多的密鑰，每個密鑰只使用一次。這些錢包正在被確定性的錢包取代，因為它們的管理，備份和匯入很麻煩。隨機密鑰的缺點是，如果你生成了很多密鑰，你必須保留所有密鑰的副本，這意味著錢包必須經常備份。每個密鑰都必須備份，否則，如果錢包變得不可用，則其控制的資金將不可撤銷地丟失。這與避免地址重用的原則直接衝突，即每個比特幣地址僅用於一次交易。地址重用將多個交易和地址相互關聯來，會減少隱私。0型非確定性錢包是窮人的選擇，如果你想避免地址重用，就要管理許多密鑰，頻繁備份。儘管Bitcoin Core客戶端包含0型錢包，但Bitcoin Core開發人員不鼓勵使用此錢包。[Type-0 nondeterministic \(random\) wallet: a collection of randomly generated keys](#) 展示了一個非確定性錢包，它包含一個鬆散的隨機密鑰集合。

Tip

除了簡單的測試以外，不推薦使用非確定性錢包，備份和使用起來太麻煩了。請使用基於行業標準的有助記詞 *HD wallet* 進行備份。

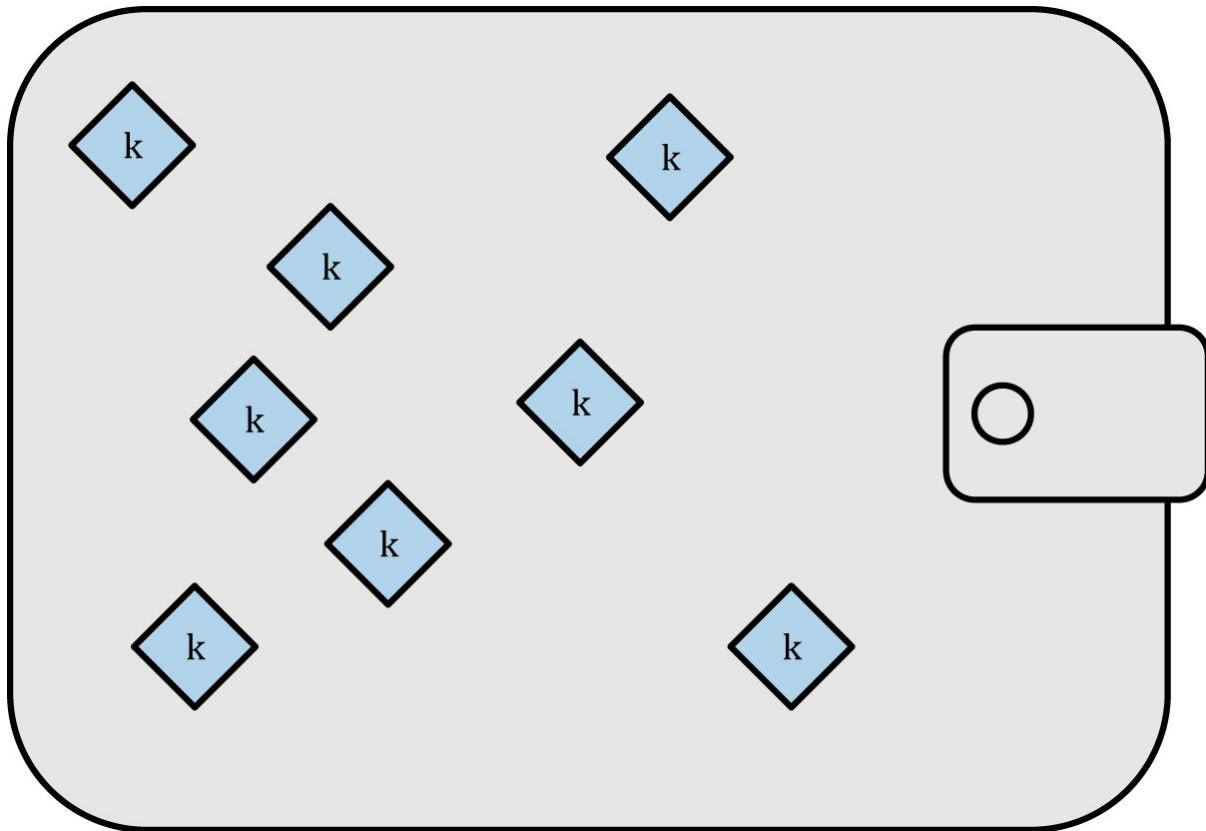


Figure 1. Type-0 nondeterministic (random) wallet: a collection of randomly generated keys

## Deterministic (Seeded) Wallets

確定性的，或“基於種子的”錢包是包含私鑰的錢包，這些私鑰都是通過使用單向雜湊函數從公共種子派生的。種子是隨機生成的數字，與其他數據（如索引編號或“chain code”（參見[分層確定性錢包（HD Wallets）\(BIP-32/BIP-44\)](#)））組合以匯出私鑰。在確定性錢包中，種子足以恢復所有的派生密鑰，因此在創建時一次備份就足夠了。種子對於錢包匯出或匯入也是足夠的，允許在不同的錢包實施之間輕鬆遷移所有用戶的密鑰。[Type-1 deterministic \(seeded\) wallet: a deterministic sequence of keys derived from a seed](#) 展示了確定性錢包的邏輯圖。

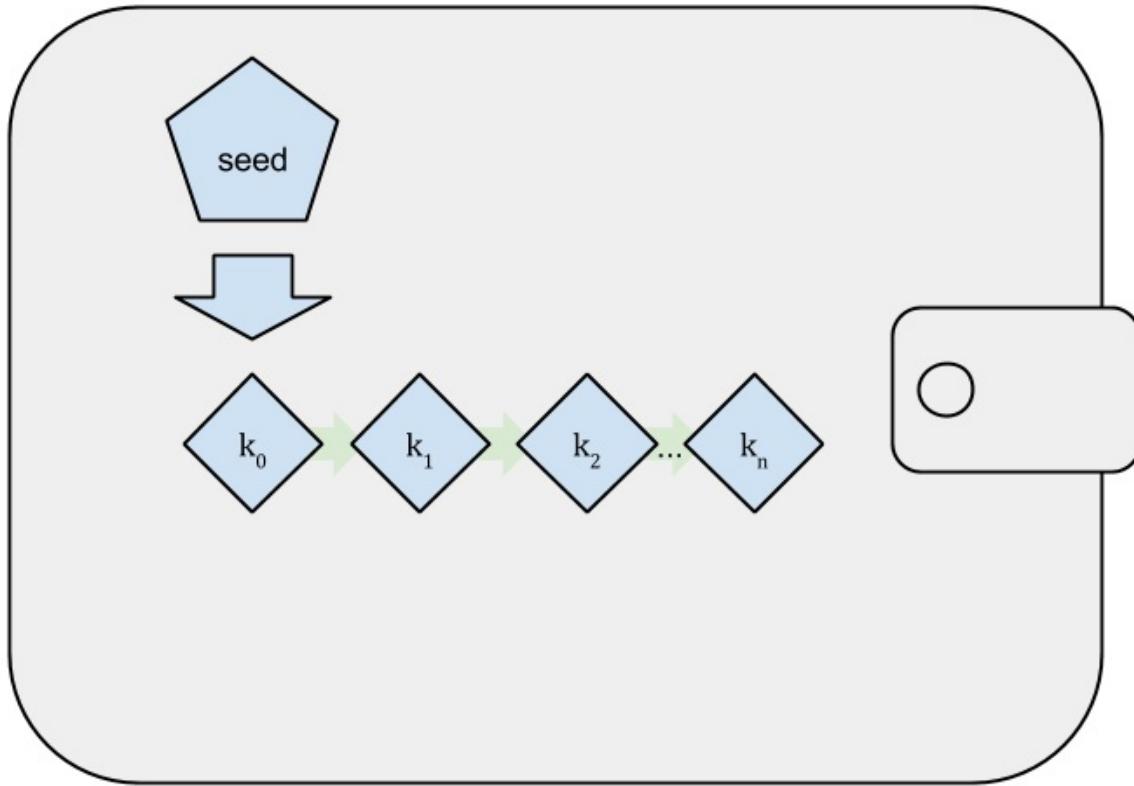


Figure 2. Type-1 deterministic (seeded) wallet: a deterministic sequence of keys derived from a seed

### 分層確定性錢包（HD Wallets）(BIP-32/BIP-44)

確定性錢包的開發使得從單個“種子”中獲得許多密鑰變得容易。確定性錢包的最高級形式是由BIP-32標準定義的HD錢包。HD錢包包含以樹結構匯出的密鑰，父密鑰可以匯出一系列的子密鑰，每個子密鑰可以匯出一系列孫子密鑰等等，可達到無限深度。這個樹結構在[Type-2 HD wallet: a tree of keys generated from a single seed](#)中進行了說明。

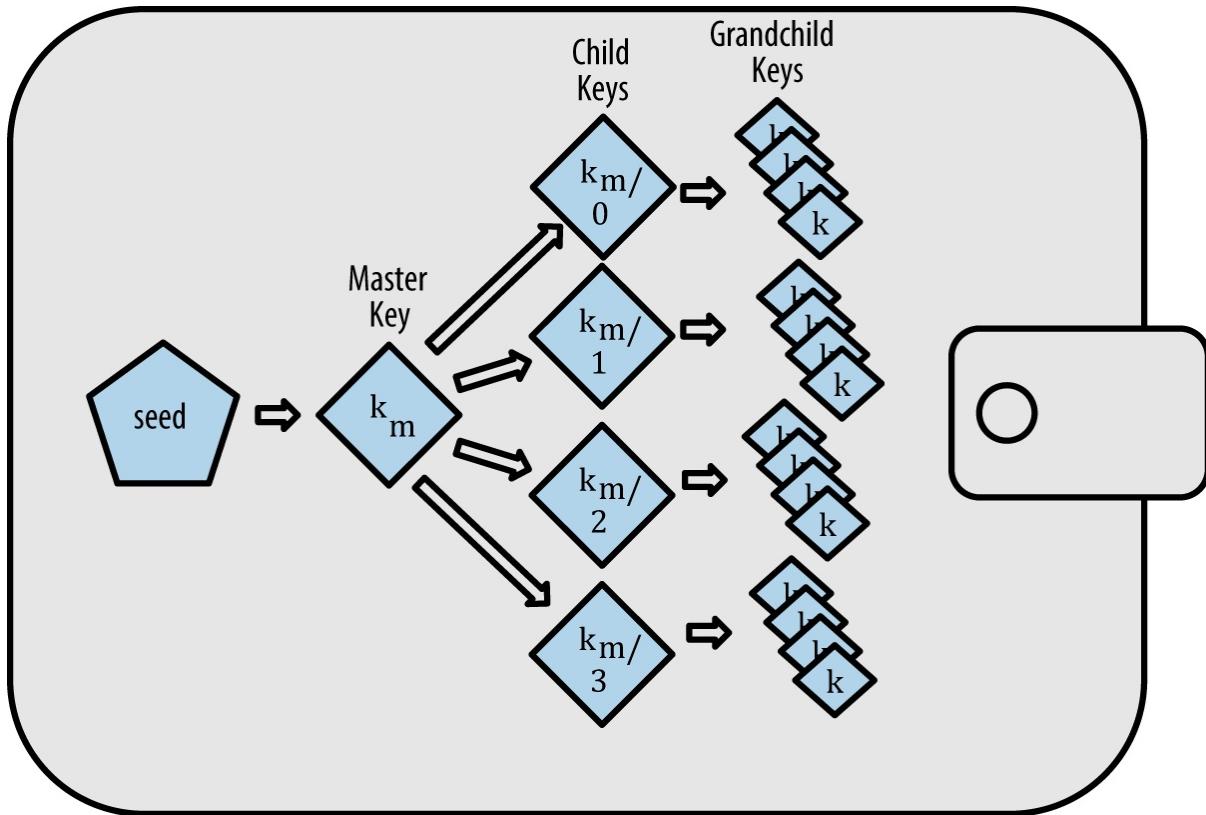


Figure 3. Type-2 HD wallet: a tree of keys generated from a single seed

與隨機（非確定性）密鑰相比，HD錢包具有兩大優勢。首先，樹結構可以用來表達額外的組織含義，例如，使用子密鑰的特定分支來接收傳入的支付，使用另一個分支來接收支付時的零錢。分支的密鑰也可用於組織機構設置，將不同分支分配給部門，子公司，特定功能或會計類別。

HD錢包的第二個優點是用戶可以創建一系列公鑰而無需訪問相應的私鑰。這允許HD錢包用於不安全的伺服器或僅作為接收用途，為每次交易發出不同的公鑰。公鑰不需要事先預加載或派生，伺服器也沒有可以花費資金的私鑰。

## 種子和助記詞 (BIP-39)

HD錢包是管理許多密鑰和地址的非常強大的機制。如果將它們與標準化方式相結合，從一系列易於轉錄，匯出和跨錢包匯入的英語單詞創建種子，就更加有用。這被稱為 助記，BIP-39定義了這個標準。今天，大多數比特幣錢包（以及用於其他密碼貨幣的錢包）都使用此標準，並且可以使用可互操作的助記詞匯入和匯出種子以進行備份和恢復。

我們實際來看一下。下列哪類種子更易於轉錄，在紙上記錄，無誤地讀取，匯出/匯入另一個錢包？

A seed for an deterministic wallet, in hex

```
0C1E24E5917779D297E14D45F14E1A1A
```

A seed for an deterministic wallet, from a 12-word mnemonic

```
army van defense carry jealous true  
garbage claim echo media make crunch
```

## 錢包最佳實踐

隨著比特幣錢包技術的成熟，出現了一些常見的行業標準，使比特幣錢包具有廣泛的互操作性，易用性，安全性和靈活性。這些通用標準是：

- 助記詞 (mnemonic code words)，基於BIP-39

- 分層確定性錢包 (HD wallets) , 基於BIP-32
- 多用途分層確定性結構 (Multipurpose HD wallet structure) , 基於BIP-43
- 多幣種和多帳戶錢包 (Multicurrency and multiaccount wallets) , 基於BIP-44

這些標準可能會改變或因未來的發展而過時，但現在它們形成了一系列連鎖技術，這些技術已成為比特幣事實上的錢包標準。

這些標準已被軟體和硬體比特幣錢包廣泛採用，使所有這些錢包可以互操作。用戶可以匯出其中一個錢包上生成的助記詞並將其匯入另一個錢包，恢復所有交易，密鑰和地址。

支持這些標準的軟體錢包的一些例子包括（按字母順序排列）Breadwallet，Copay，Multibit HD和Mycelium。支持這些標準的硬體錢包的例子包括（按字母順序列出）Keepkey，Ledger和Trezor。

以下各節詳細介紹這些技術。

Tip

如果你正在實施比特幣錢包，則應按照BIP-32，BIP-39，BIP-43和BIP-44標準，將其構建為HD錢包，並將種子編碼為助記詞用於備份，就像以下章節介紹的那樣。

## 使用比特幣錢包

在 [user-stories] 中我們介紹了Gabriel, 一位在里約熱內盧的富有進取精神的年輕人，他正在經營一家簡單的網上商店，銷售比特幣品牌的T恤，咖啡杯和貼紙。

Gabriel 使用 Trezor 比特幣硬體錢包 ([A Trezor device: a bitcoin HD wallet in hardware](#)) 安全地管理他的比特幣。Trezor是一個有兩個按鈕的簡單的USB設備，用於儲存密鑰（以HD錢包的形式），簽署交易。Trezor錢包實現了本章介紹的所有工業標準，因此Gabriel並不依賴任何專有技術或單一供應商解決方案。



Figure 4. A Trezor device: a bitcoin HD wallet in hardware

當Gabriel首次使用Trezor時，該設備通過內置硬體隨機數生成器生成助記符和種子。在這個初始化階段，錢包在屏幕上逐一顯示帶有編號的單詞序列（參見 [Trezor displaying one of the mnemonic words](#)）。

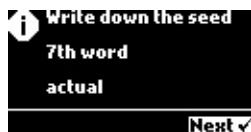


Figure 5. Trezor displaying one of the mnemonic words

記錄下助記詞，Gabriel可以在他的Trezor設備丟失或損壞時使用備份的助記詞進行恢復。這種助記符可以用於新的Trezor設備或任意一個兼容的軟體或硬體錢包。請注意，助記詞的順序很重要。

Table 1. Gabriel's paper backup of the mnemonic

1.	army	7.	garbage
----	------	----	---------

2.	<i>van</i>	8.	<i>claim</i>
3.	<i>defense</i>	9.	<i>echo</i>
4.	<i>carry</i>	10.	<i>media</i>
5.	<i>jealous</i>	11.	<i>make</i>
6.	<i>true</i>	12.	<i>crunch</i>

**Note**

為簡單起見，在 [Gabriel's paper backup of the mnemonic](#) 中展示了12個助記詞。實際上，大多數硬體錢包可以產生更安全的24個助記詞。不管長度如何，助記詞的使用方式完全相同。

對於第一次網店實踐，Gabriel使用Trezor設備上生成的單個比特幣地址。所有客戶都可以使用這個地址進行所有訂單。正如我們將看到的，這種方法有一些缺點，可以通過HD錢包進行改進。

## 錢包技術細節

現在我們來仔細研究比特幣錢包所使用的每個重要行業標準。

### 助記詞（Mnemonic Code Words）(BIP-39)

助記詞是表示（編碼）用作派生確定性錢包的種子的隨機數的一個單詞序列。單詞序列足以重新創建種子，並重新創建錢包和所有派生的密鑰。使用助記詞實現確定性錢包的錢包應用會在首次創建錢包時向用戶顯示12至24個單詞的序列。這個單詞序列是錢包的備份，可用於在相同或任何兼容的錢包應用中恢復和重新創建所有密鑰。與隨機數字序列相比，助記詞使得用戶更容易備份錢包，因為它們易於閱讀和正確轉錄。

**Tip**

助記詞通常與“大腦錢包（brainwallets）”混淆。他們不一樣。主要區別在於大腦錢包由用戶選擇的單詞組成，而助記詞由錢包隨機創建並呈現給用戶。這個重要的區別使助記詞更加安全，因為人類是非常貧乏的隨機性來源。

助記詞在BIP-39中定義（參見[\[appdxbitcoinimpproposals\]](#)）。注意，BIP-39是助記詞標準的一個實現。還有一個不同的標準，使用一組不同的詞，在BIP-39之前由Electrum錢包使用。BIP-39由生產Trezor硬體錢包的公司提出，與Electrum不兼容。但是，BIP-39現在已經獲得了廣泛的行業支持，數十種產品可以互操作，被視為事實上的行業標準。

BIP-39定義了助記詞和種子的創建方法，我們通過九個步驟來描述它。為了清楚起見，該過程分為兩部分：步驟1至6在[\[generate\\_mnemonic\\_words\]](#)中，步驟7至9在[從助記符到種子](#)中。

#### 生成助記詞

助記詞是由錢包使用BIP-39中定義的標準化過程自動生成的。錢包從一個熵源開始，添加校驗和，將熵映射到單詞列表：

1. 創建一個128到256位的隨機序列（熵）。
2. 通過取其SHA256雜湊的第一個（熵長度/ 32）位創建隨機序列的校驗和。
3. 將校驗和添加到隨機序列的末尾。
4. 將結果拆分為11位長的多個段。
5. 將每個11位值映射到有2048個單詞的預定義字典中的一個單詞。
6. 助記詞就是這些單詞的序列。

Generating entropy and encoding as mnemonic words 展示瞭如何使用熵來生成助記詞。

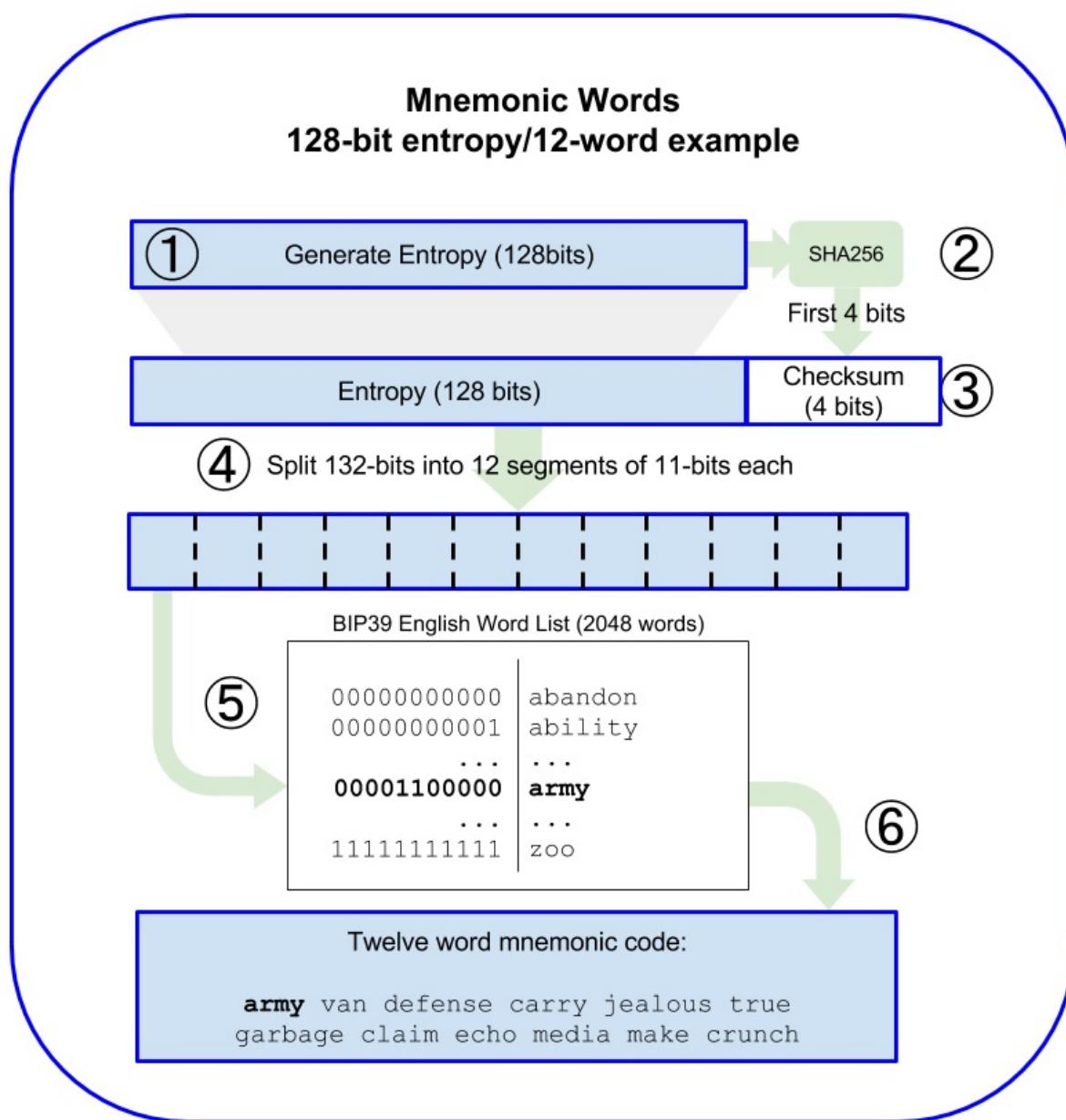


Figure 6. Generating entropy and encoding as mnemonic words

Mnemonic codes: entropy and word length 顯示了熵數據的大小與助記詞的長度之間的關係。

Table 2. Mnemonic codes: entropy and word length

Entropy (bits)	Checksum (bits)	Entropy + checksum (bits)	Mnemonic length (words)
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
256	8	264	24

### 從助記符到種子

助記詞表示長度為128到256位的熵。然後使用熵通過使用密鑰擴展函數PBKDF2來匯出更長的（512位）種子。之後使用生成的種子構建確定性錢包並匯出其密鑰。

密鑰擴展函數需要兩個參數：助記詞和 鹽 *salt*。在密鑰擴展函數中使用鹽的目的是使構建一個查找表並暴力破解難以實現。在BIP-39標準中，鹽有另一個目的 - 它允許引入密碼，作為保護種子的附加安全因素，我們將在 [BIP-39中可選的密碼](#) 中詳細描述。

步驟7到9中描述的過程從 [\[generated\\_mnemonic\\_words\]](#) 中的過程繼續：

7. PPBKDF2密鑰擴展函數的第一個參數是步驟6中產生的 助記詞
8. PPBKDF2密鑰擴展函數的第一個參數是 鹽 (*salt*) 。鹽由字串 " mnemonic " 加上可選的用戶提供的密碼組成。
9. PBKDF2使用HMAC-SHA512演算法執行2048輪雜湊來擴展助記詞和鹽，產生一個512位值，就是種子。

From mnemonic to seed 展示瞭如何使用助記詞來生成種子。

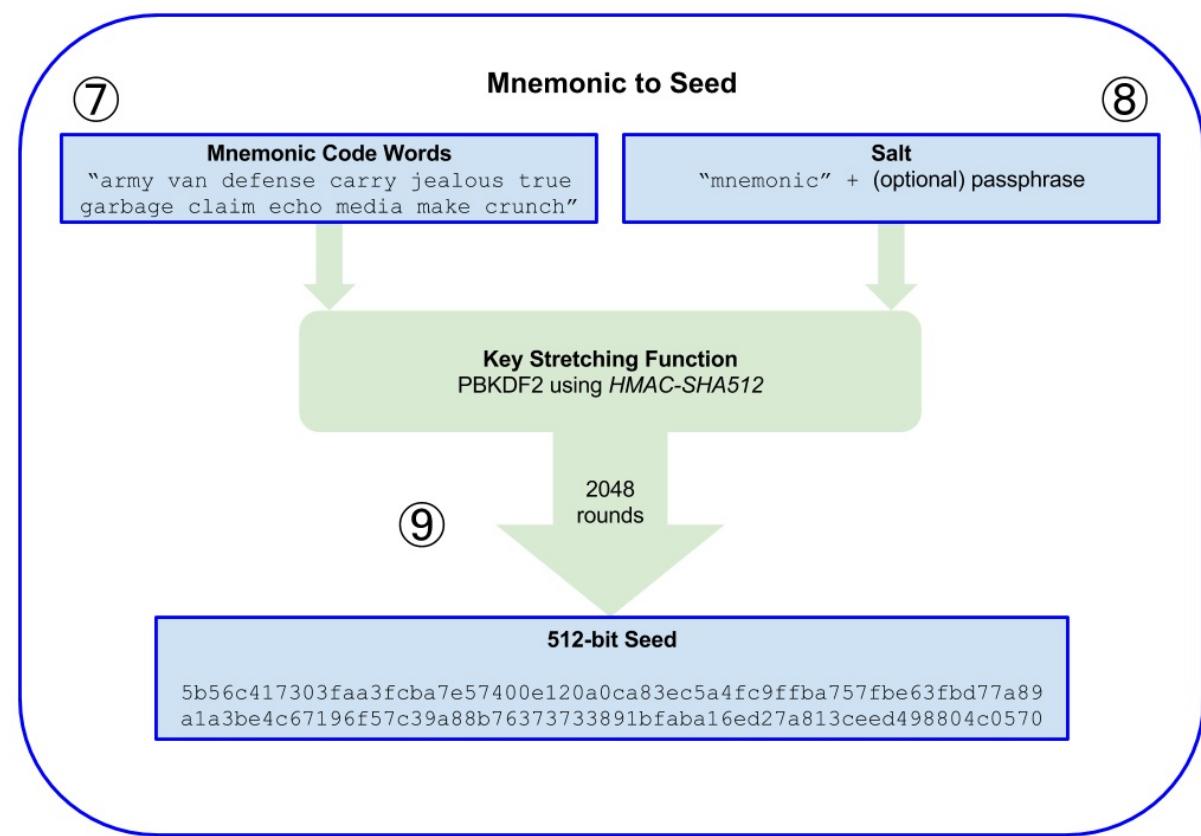


Figure 7. From mnemonic to seed

Tip

密鑰擴展方法及其2048輪雜湊是一種非常有效的防止對助記詞或密碼短語攻擊的保護。它使得嘗試超過幾千個密碼和助記符組合的成本非常高，而可能派生的種子數量很大 ( $2^{512}$ ) 。

表格 [#mnemonic\\_128\\_no\\_pass](#), [#mnemonic\\_128\\_w\\_pass](#), and [#mnemonic\\_256\\_no\\_pass](#) 顯示一些助記詞和他們產生的種子（沒有任何密碼）的例子。

Table 3. 128-bit entropy mnemonic code, no passphrase, resulting seed

<b>Entropy input (128 bits)</b>	0c1e24e5917779d297e14d45f14e1a1a
<b>Mnemonic (12 words)</b>	army van defense carry jealous true garbage claim echo media make crunch

<b>Passphrase</b>	(none)
<b>Seed (512 bits)</b>	5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fbd77a89a1a3be4c67196f57c39a88b76373733891bfaba16ed27a813ceed498804c0570

Table 4. 128-bit entropy mnemonic code, with passphrase, resulting seed

<b>Entropy input (128 bits)</b>	0c1e24e5917779d297e14d45f14e1a1a
<b>Mnemonic (12 words)</b>	army van defense carry jealous true garbage claim echo media make crunch
<b>Passphrase</b>	SuperDuperSecret
<b>Seed (512 bits)</b>	3b5df16df2157104cfdd22830162a5e170c0161653e3afe6c88defefb0818c793dbb28ab3ab09189715861dc8a18358f80b79d49acf64142ae57037d1d54

Table 5. 256-bit entropy mnemonic code, no passphrase, resulting seed

<b>Entropy input (256 bits)</b>	2041546864449caff939d32d574753fe684d3c947c3346713dd8423e74abcf8c
<b>Mnemonic (24 words)</b>	cake apple borrow silk endorse fitness top denial coil riot stay wolf luggage oxygen faint major edit measure invite love trap field dilemma oblige
<b>Passphrase</b>	(none)
<b>Seed (512 bits)</b>	3269bce2674acbd188d4f120072b13b088a0ecf87c6e4cae41657a0bb78f5315b33b3a04356e53d05f1e0deaa082df8d487381379df848a6ad7e98798404

### BIP-39中可選的密碼

BIP-39標準允許在派生種子中使用可選的密碼。如果沒有使用密碼，助記詞將被一個常量字串 mnemonic 的鹽擴展，產生一個特定的512位種子。如果使用密碼短語，則擴展函數會從同一助記符中生成一個不同的種子。對於一個助記詞，每一個可能的密碼都會導致不同的種子。本質上，沒有“錯誤的”密碼。所有密碼都是有效的，會生成不同的種子，形成一大批未初始化的錢包。可能的錢包的集合非常大 ( $2^{512}$ )，因此沒有可能暴力破解或意外猜測出正在使用的錢包。

Tip

BIP-39中沒有“錯誤的”口令。每個密碼都會導致一些錢包，除非以前使用過，錢包將是空的。

可選的密碼引入了兩個重要功能：

- 第二重保護，需要記憶的密碼使得只獲得助記詞沒有用，避免助記詞被盜時的損失。
- 一種似是而非的拒絕形式或“脅迫錢包”，一個選定的密碼會導致進入一個帶有少量資金的錢包，用於將攻擊者的注意力從有大部分資金的“真實”錢包引開。

但是，要注意使用密碼也會導致丟失的風險：

- 如果錢包所有者無行為能力或死亡，而且沒有其他人知道密碼，則種子無用，錢包中儲存的所有資金都將永久丟失。
- 相反，如果所有者在與種子相同的位置備份密碼，它將失去第二重保護的意義。

雖然密碼非常有用，但應該結合精心策劃的備份和恢復過程，需要考慮主人是否存活，要允許其家人恢復密碼貨幣資產。

## 使用助記詞

BIP-39有許多不同的程式語言庫實現：

### *python-mnemonic*

提出BIP-39標準的SatoshiLabs團隊用Python寫的參考實現

### *bitcoinjs/bip39*

BIP-39的JavaScript實現，是流行的bitcoinJS框架的一部分。

### *libbitcoin/mnemonic*

BIP-39的C++實現，是流行的Libbitcoin框架的一部分。

還有一個在網頁中實現的BIP-39生成器，這對於測試非常有用。[A BIP-39 generator as a standalone web page](#) 展示了生成助記符，種子和擴展私鑰的網頁。

## Mnemonic

You can enter an existing BIP39 mnemonic, or generate a new random one. Typing your own twelve words will probably not work how you expect, since the words require a particular structure (the last word is a checksum)

For more info see the [BIP39 spec](#)

Generate a random 12 word mnemonic, or enter your own below.

BIP39 Mnemonic	army van defense carry jealous true garbage claim echo media make crunch
BIP39 Passphrase (optional)	
BIP39 Seed	5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fb77a89a1a3be4c6719 6f57c39a88b76373733891bfaba16ed27a813ceed498804c0570
Coin	Bitcoin
BIP32 Root Key	xprv9s21ZrQH143K3t4UZrNgeA3w861fwjYLaGwmPtQyPMmzshV2owVpfBSd2Q7YsHZ9j6 i6ddYjb5PLtUdMZn8LhvuCVhGcQntq5rn7JVMqnje

Figure 8. A BIP-39 generator as a standalone web page

這個頁面 (<https://iancoleman.github.io/bip39/>) 可以離線或在線訪問

## 通過種子創建HD錢包

HD錢包是由一個根種子 *root seed* 創建的，是一個128位，256位或512位的隨機數。通常，這個種子是從助記詞 *mnemonic* 生成的，詳見前一節。

HD錢包中的每個密鑰都是從這個根種子確定性地派生出來的，這使得可以在任何兼容的HD錢包中從該種子重新創建整個HD錢包。這使得備份，恢復，匯出和匯入包含數千乃至數百萬個密鑰的HD錢包變得很容易，只需傳輸根種子的助記詞即可。

創建主密鑰 *master keys* 和主鏈碼 *master chain code* 的過程如 [Creating master keys and chain code from a root seed](#) 所示。

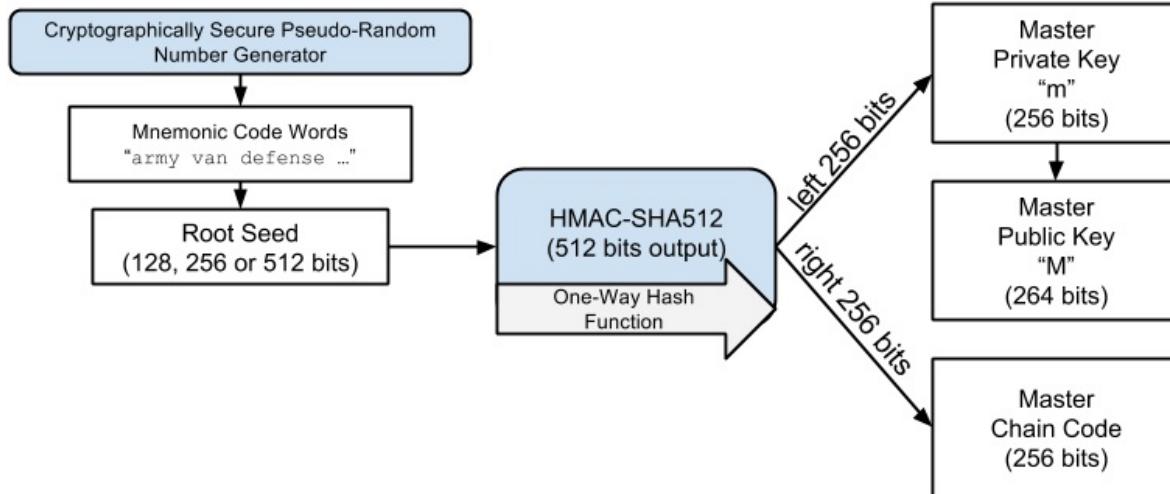


Figure 9. Creating master keys and chain code from a root seed

將根種子作為 HMAC-SHA512 演算法的輸入，生成的雜湊結果用來生成 主私鑰 *master private key (m)* 和 主鏈碼 *master chain code (c)*。

然後使用我們在 [\[pubkey\]](#) 中看到的橢圓曲線乘法  $m * G$  利用主密鑰 (m) 生成相應的主公鑰 (M)。

主鏈碼 (c) 用於在從父鍵創建子鍵的函數中引入熵，我們將在下一節看到。

### 子私鑰的派生

HD錢包使用 子密鑰派生 *child key derivation (CKD)* 方法從父密鑰派生子密鑰。

子密鑰派生方法基於單向雜湊函數，該函數結合：

- 一個父級私鑰或公鑰 (ECDSA未壓縮密鑰)
- 一個稱作鏈碼(chain code)的種子 (256 bits)
- 一個索引數字 (32 bits)

鏈碼用於向過程中引入確定性隨機數據，所以只知道索引和子密鑰不足以派生其他子密鑰。除非有鏈碼，否則知道一個子鑰匙不能找到它的兄弟姐妹。初始鏈碼種子（樹的根部）由種子製成，而後續子鏈碼則從每個父鏈碼中匯出。

這三項（父密鑰，鏈碼和索引）被組合並雜湊以生成子鍵，如下所示。

使用HMAC-SHA512演算法將父公鑰，鏈碼和索引組合並雜湊，以產生512位雜湊。這個512位雜湊平分為兩部分。右半部分256位作為後代的鏈碼，左半部分256位被添加到父私鑰以生成子私鑰。在 [Extending a parent private key to create a child private key](#) 中，我們看到這個例子中的索引設置為0，以產生父項的“零”級（第一個索引）孩子。

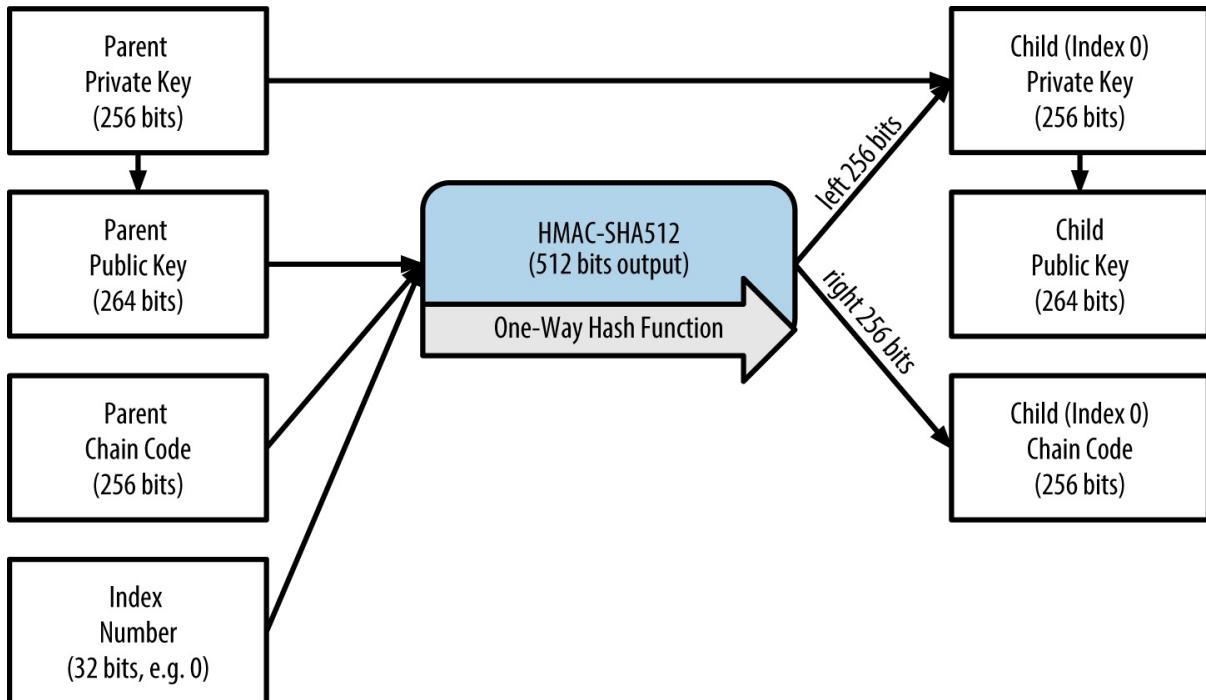


Figure 10. Extending a parent private key to create a child private key

更改索引允許我們擴展父項並創建序列中的其他子項，例如Child 0，Child 1，Child 2等。每個父項可以有 $2,147,483,647 (2^{31})$ 個子項 ( $2^{32}$ 範圍的一半  $2^{31}$ 是可用的，另一半保留用於特殊類型的推導，我們將在本章後面討論)。

在樹的下一層重複這個過程，每個孩子都可以成為父項並在無限的世代中創造自己的孩子。

### 使用派生的子密鑰

子私鑰與非確定性（隨機）密鑰沒有區別。因為派生函數是單向函數，不能使用子項來尋找父項和尋找任何兄弟姐妹。不能通過第n個子項找到它的兄弟姐妹，如第 n-1 個子項或者第 n+1 個子項，或者任何這個序列上的子項。只能通過父密鑰和鏈碼派生所有的孩子。如果沒有子鏈碼，子密鑰也不能派生任何孫項。你需要子私鑰和子鏈碼來啟動一個新分支並派生孫項。

那麼，子私鑰能用來幹什麼呢？它可以用來製作公鑰和比特幣地址。然後，它可以用來簽署交易，並花費任何支付給該地址的費用。

Tip

子私鑰，相應的公鑰和比特幣地址都與隨機創建的密鑰和地址沒有區別。在創建它們的HD錢包之外是不知道它們屬於一個序列的。一旦創建，就像“普通”鍵一樣工作。

### 擴展密鑰

如我們所見，基於三個輸入：密鑰，鏈碼和所需子項的索引，可以使用密鑰派生函數在樹的任何級別創建子項。這兩個基本要素是密鑰和鏈式程式碼，它們的組合稱為 **擴展密鑰 extended key**。也可以認為“擴展密鑰”是“可擴展的密鑰”，因為這樣的密鑰可以用來派生孩子。

擴展密鑰簡單地表示為由256位的密鑰和256位的鏈碼串聯成的512位序列。有兩種類型的擴展密鑰：擴展私鑰是私鑰和鏈碼的組合，可用於派生子私鑰（從它們產生子公鑰）；擴展公鑰是公鑰和鏈碼，可用於創建子公鑰（只有子公鑰），如 [\[public\\_key\\_derivation\]](#) 中所述。

將擴展密鑰視為HD錢包樹形結構中分支的根。可以通過分支的根，派生出其他分支。擴展私鑰可以創建一個完整的分支，而擴展公鑰只能創建一個公鑰分支。

Tip

擴展密鑰由私鑰或公鑰和鏈碼組成。擴展密鑰可以創建子項，在樹結構中生成自己的分支。共享一個擴展密鑰可以訪問整個分支。

擴展密鑰使用Base58Check編碼，可以輕鬆匯出匯入BIP-32兼容的錢包。擴展密鑰的Base58Check編碼使用特殊的版本號，當使用Base58字符進行編碼時，其前綴為“xprv”和“xpub”，以使其易於識別。因為擴展的密鑰是512或513位，所以它比我們以前見過的其他Base58Check編碼的字串要長得多。

這是一個Base58Check編碼的擴展私鑰：

```
xprv9tyUQV64JT5qs3RSTJKXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i  
6GoNMKUga5biW6Hx4tws2six3b9c
```

這是對應的Base58Check編碼的擴展公鑰：

```
xpub67xpozcx8pe95XVuZLHXZeG6WXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVKZBPLrtJu  
nSDMstweyLXhRgPxpd14sk9tJPW9
```

### 子公鑰派生

如前所述，HD錢包的一個非常有用的特性是能夠從父公鑰中獲得子公鑰，而沒有私鑰。這為我們提供了兩種派生子公鑰的方法：從子私鑰或直接從父公鑰獲取子公鑰。

因此，可以使用擴展公鑰，匯出HD錢包該分支中的所有公鑰（注意只有公鑰）。

此快捷方式可用於創建非常安全的公鑰 - 只有部署伺服器或應用程式具有擴展公鑰的副本，並且沒有任何私鑰。這種部署可以產生無限數量的公鑰和比特幣地址，但無法花費發送到這些地址的任何資金。與此同時，在另一個更安全的伺服器上，擴展私鑰可以匯出所有相應的私鑰來簽署交易並花費金錢。

這個解決方案的一個常見應用是在提供電子商務應用程式的Web伺服器上安裝擴展公鑰。網路伺服器可以使用公鑰匯出函數來為每個交易（例如，為顧客購物車）創建新的比特幣地址。Web伺服器上不會有任何易被盜的私鑰。沒有HD錢包，唯一的方法就是在單獨的安全伺服器上生成數千個比特幣地址，然後將其預先加載到電子商務伺服器上。這種方法很麻煩，需要不斷的維護以確保電子商務伺服器不會“用完”密鑰。

另一個常見應用是用於冷儲存或硬體錢包。在這種情況下，擴展私鑰可以儲存在紙錢包或硬體設備（如Trezor硬體錢包）上，而擴展公鑰可以保持在線。用戶可以隨意創建“接收”地址，而私鑰可以安全地在離線狀態下儲存。為了花費資金，用戶可以在離線簽名比特幣客戶端使用擴展私鑰簽名，或在硬體錢包設備上簽名交易（例如Trezor）。[Extending a parent public key to create a child public key](#) 演示了用擴展父公鑰派生子公鑰的機制。

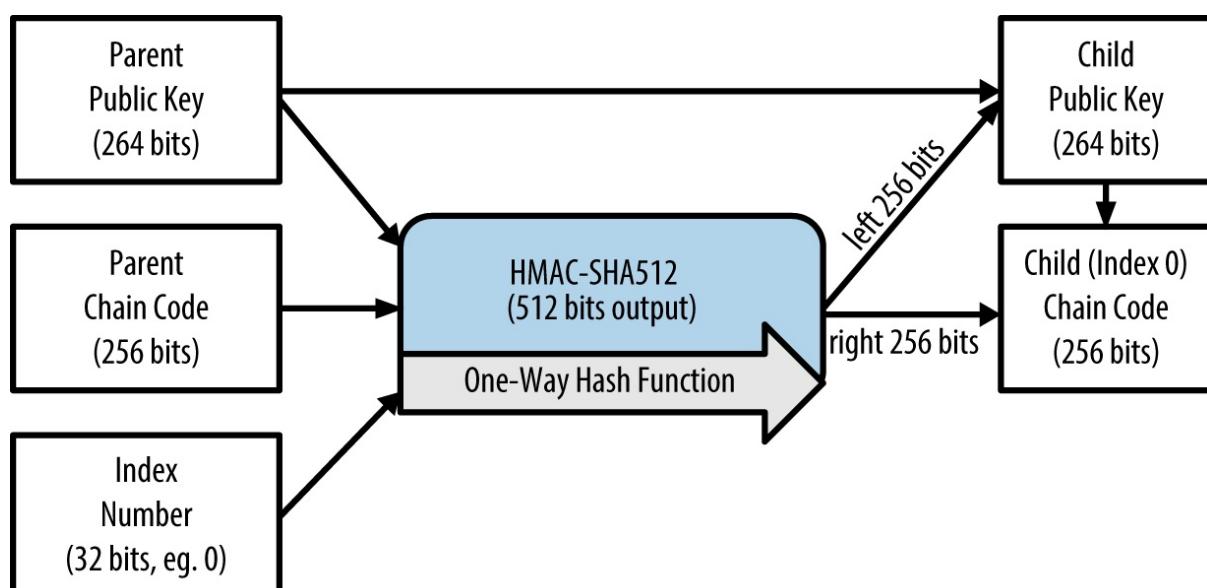


Figure 11. Extending a parent public key to create a child public key

## 在網店中使用擴展公鑰

讓我們看看如何使用HD錢包繼續Gabriel的網上商店故事。

Gabriel首先出於愛好建立了他的網上商店，基於簡單的Wordpress。他的商店非常簡單，只有幾個頁面和有一個比特幣地址的下單表單。

Gabriel使用他的Trezor設備生成的第一個比特幣地址作為他的商店的主要比特幣地址。這樣，所有收到的付款都將支付給他的Trezor硬體錢包所控制的地址。

客戶將使用表單提交訂單並將支付款項發送至Gabriel發佈的比特幣地址，觸發一封電子郵件，其中包含Gabriel要處理的訂單詳情。每週只有幾個訂單，這個系統運行得很好。

然而，這家小型網上商店變得非常成功，吸引了當地的許多訂單。不久，Gabriel便不知所措了。由於所有訂單都支付相同的地址，很難正確匹配訂單和交易，尤其是當同一數量的多個訂單緊密結合在一起時。

Gabriel的HD錢包通過在不知道私鑰的情況下派生子公鑰的能力提供了更好的解決方案。Gabriel可以在他的網站上加載一個擴展公鑰(xpub)，用來為每個客戶訂單派生一個唯一的地址。Gabriel可以從他的Trezor花費資金，但在網站上加載的xpub只能生成地址並獲得資金。HD錢包的這個特點是一個很好的安全功能。Gabriel的網站不包含任何私鑰，因此不需要高度的安全性。

Gabriel將Web軟體與Trezor硬體錢包一起使用匯出xpub。必須插入Trezor設備才能匯出公鑰。請注意，硬體錢包永遠不會匯出私鑰——這些密鑰始終保留在設備上。[Exporting an xpub from a Trezor hardware wallet](#)展示了Gabriel用於匯出xpub的Web界面。

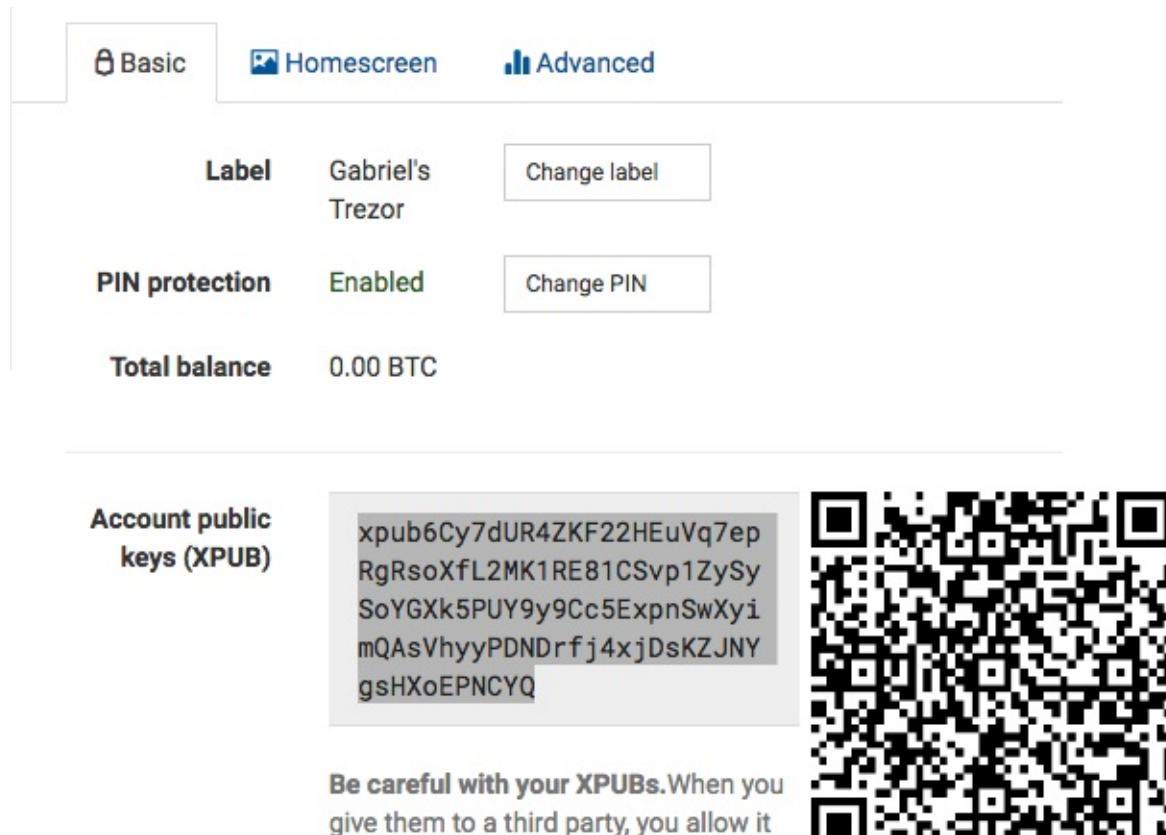


Figure 12. Exporting an xpub from a Trezor hardware wallet

Gabriel將xpub複製到他的網上商店的比特幣商店軟體中。並使用 *Mycelium Gear*，這是一個開源的網上商店插件，用於各種網站託管和內容平臺。*Mycelium Gear*使用xpub為每次購買生成一個唯一的地址。

### 強化的子密鑰派生

從 xpub 派生公鑰的分支是非常有用的，但有潛在的風險。訪問 xpub 不會訪問子私鑰。但是，因為 xpub 包含鏈碼，所以如果某個子私鑰已知，或者以某種方式洩漏，則可以與鏈式程式碼一起使用，派生所有其他子私鑰。一個洩露的子私鑰和一個父鏈碼可以生成所有其他的子私鑰。更糟的是，可以使用子私鑰和父鏈碼來推導父私鑰。

為了應對這種風險，HD 錢包使用一種稱為 *hardened derivation* 的替代派生函數，該函數“破壞”父公鑰和子鏈碼之間的關係。強化派生函數使用父私鑰來派生子鏈碼，而不是父公鑰。這會在父/子序列中創建一個“防火牆”，鏈碼不能危害父級或同級的私鑰。父私鑰替代父公鑰作為雜湊函數的輸入，強化後的派生函數看起來與正常的子私鑰派生幾乎相同，如 [Hardened derivation of a child key; omits the parent public key](#) 中的圖所示。

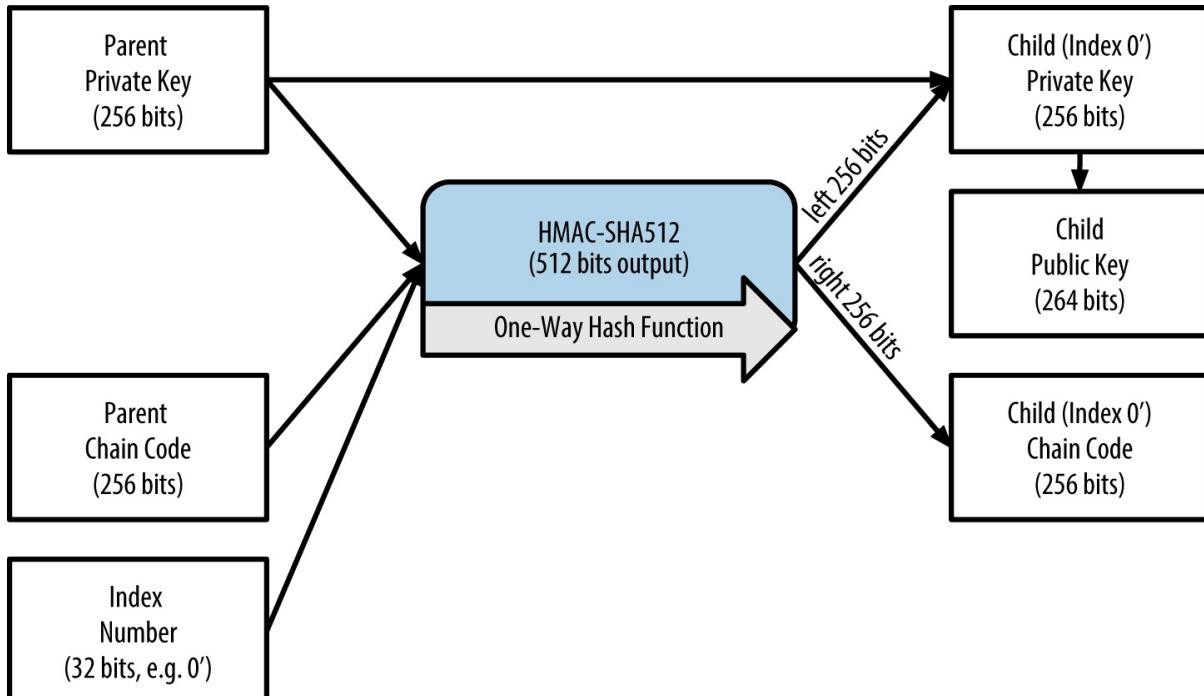


Figure 13. Hardened derivation of a child key; omits the parent public key

當使用強化的私有派生函數時，生成的子私鑰和鏈碼與正常派生函數所產生的完全不同。由此產生的“分支”密鑰可用於生成不易受攻擊的擴展公鑰，因為它們所包含的鏈碼不能用於揭示任何私鑰。因此，強化派生用於在繼承樹上使用擴展公鑰的級別之上創建“屏障”。

簡而言之，如果你想使用 xpub 的便利性來派生分支公鑰，而不想面臨洩漏鏈碼的風險，應該從強化的父項派生。作為最佳實踐，主密鑰的1級子密鑰始終使用強化派生，以防止主密鑰受到破壞。

### 常規派生與強化派生的索引號

在派生函數中使用的索引號是一個32位整數。為了便於區分通過常規推導函數派生的密鑰與通過強化派生派生的密鑰，該索引號分為兩個範圍。0到 $2^{31} - 1$  (0x0到0x7FFFFFFF) 之間的索引號僅用於常規推導。 $2^{31}$  和  $2^{32} - 1$  (0x80000000到0xFFFFFFFF) 之間的索引號僅用於硬化派生。因此，如果索引號小於 $2^{31}$ ，則子密鑰是常規的，而如果索引號等於或大於 $2^{31}$ ，則子密鑰是強化派生的。

為了使索引號碼更容易閱讀和顯示，強化子密鑰的索引號從零開始顯示，但帶有一個符號。第一個常規子密鑰表示成 0，第一個強化子祕鑰（索引號是 0x80000000）表示成 0'。以此類推，第二個強化子密鑰（0x80000001）表示成 1'。當你看到HD錢包索引'i'時，它表示 $2^{31} + i$ 。

### HD錢包密鑰標識符 (path)

HD錢包中的密鑰使用“路徑(path)”命名約定來標識，樹的每個級別都用斜槓 (/) 字符分隔（請參見 [HD wallet path examples](#)）。從主密鑰派生的私鑰以“m”開頭。從主公鑰派生的公鑰以“M”開始。因此，主私鑰的第一個子私鑰為 m/0。第一個子公鑰是 M/0。第一個子私鑰的第二個子私鑰是 m/0/1，依此類推。

從右向左讀取一個密鑰的“祖先”，直到到達派生出它的主密鑰。例如，標識符  $m/x/y/z$  描述了私鑰  $m/x/y$  的第 $z$ 個子私鑰， $m/x/y$  是私鑰  $m/x$  的第 $y$ 個子私鑰， $m/x$  是  $m$  的第 $x$ 個子私鑰。

Table 6. HD wallet path examples

HD path	Key described
$m/0$	The first (0) child private key from the master private key ( $m$ )
$m/0/0$	The first grandchild private key from the first child ( $m/0$ )
$m/0'/0$	The first normal grandchild from the first <i>hardened</i> child ( $m/0'$ )
$m/1/0$	The first grandchild private key from the second child ( $m/1$ )
$M/23/17/0/0$	The first great-great-grandchild public key from the first great-grandchild from the 18th grandchild from the 24th child

#### HD錢包的樹狀結構導航

HD錢包的樹狀結構提供了巨大的靈活性。每個父級擴展密鑰的可以有40億個子密鑰：20個常規子密鑰和20億強化子密鑰。這些子密鑰中的每一個又可以有另外40億子密鑰。這棵樹像你想要的一樣深，有無限的世代。然而，這些靈活性，導致在這個無限樹中導航變得非常困難。在不同實現之間轉移HD錢包尤其困難，因為內部分支和子分支的可能性是無窮無盡的。

有兩個BIP為HD錢包的樹狀結構提出了一些建議的標準，為這種複雜性提供解決方案。BIP-43建議使用第一個強化子索引作為表示樹狀結構“用途”的特殊標識符。基於BIP-43，HD錢包應該只使用樹的一個1級分支，索引號通過定義其用途來標識樹的其餘部分的結構和名稱空間。例如，僅使用分支  $m/i'$  的HD錢包表示特定用途，用途由索引號“ $i'$ ”標識。

BIP-44在BIP-43下提出了一個多帳戶結構作為“用途”號碼  $44'$ 。所有遵守BIP-44的HD錢包通過僅使用樹的一個分支來體現： $m/44'$ 。

BIP-44定義了包含五個預定義樹級的結構：

```
m / purpose' / coin_type' / account' / change / address_index
```

第一級“用途”始終設置為  $44'$ ，第二級“ $coin\_type'$ ”表示密碼貨幣的類型，以支持多貨幣HD錢包，其中每種貨幣在第二級下具有其自己的子樹。現在定義了三種貨幣：比特幣是  $m/44'/0'$ ，比特幣測試網是  $m/44'/1'$ ，萊特幣（Litecoin）是  $m/44'/2'$ 。

樹的第三層是“帳戶”，允許用戶將他們的錢包細分為單獨的邏輯子帳戶，以用於會計或組織目的。例如，一個HD錢包可能包含兩個比特幣“帳戶”： $m/44'/0'/0'$  和  $m/44'/0'/1'$ 。每個帳戶都是自己的子樹的根。

在第四層，“零錢”，HD錢包有兩個子樹，一個用於創建接收地址，另一個用於創建零錢地址。請注意，雖然以前的層級使用強化派生，但此層級使用常規派生。這是為了允許樹的這個級別匯出擴展的公鑰以供在不安全的環境中使用。“地址\_索引”由HD錢包的第四級派生，也就是第五級。例如，主帳戶中比特幣支付的第三個接收地址為  $M/44'/0'/0'/0/2$ 。

[BIP-44 HD wallet structure examples](#) 顯示了幾個例子。

Table 7. BIP-44 HD wallet structure examples

HD path	Key described
$M/44'/0'/0'/0/2$	主要比特幣帳戶的第三個接收地址公鑰

M/44'/0'3'/1/14	第四個比特幣帳戶的第十五個零錢地址公鑰
m/44'/2'0'/0/1	Litecoin主賬戶中的第二個私鑰，用於簽署交易

# 交易

## 簡介

交易是比特幣系統中最重要的部分。比特幣中其他的一切都旨在確保交易可以創建，傳播，驗證並最終添加到交易（區塊鏈）的全球總賬中。交易是對比特幣系統參與者之間的價值轉移進行編碼的資料結構。每筆交易都是比特幣區塊鏈中的公開條目，即全球複式簿記分類賬。

在本章中，我們將檢查各種形式的交易，它們包含的內容，如何創建它們，如何驗證以及它們如何成為所有交易永久記錄的一部分。當我們在本章中使用術語“錢包”時，我們指的是構建交易的軟體，而不僅僅是密鑰的資料庫。

## 交易詳情

在 [ch02\_bitcoin\_overview] 中，我們使用區塊瀏覽器查看了Alice在Bob的咖啡店購買咖啡的交易（Alice's transaction to Bob's Cafe）。

區塊瀏覽器顯示一個從Alice的“地址”到Bob的“地址”的交易。這是交易中包含的內容的簡化視圖。事實上，我們將在本章中看到，大部分訊息都是由區塊瀏覽器構建的，實際上並不在交易中。

### Transaction View information about a bitcoin transaction

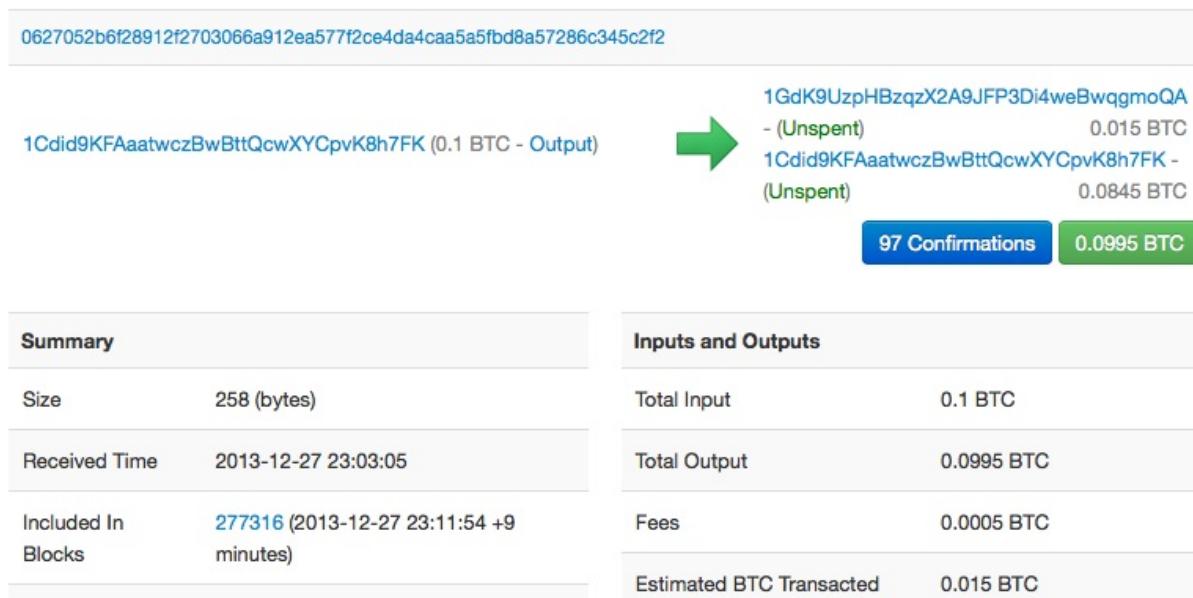


Figure 1. Alice's transaction to Bob's Cafe

## 交易背後

實際的交易看起來與典型的區塊瀏覽器提供的非常不同。實際上，我們在各種比特幣應用界面中看到的高層次結構並不實際存在於比特幣系統中。

我們可以使用Bitcoin Core的命令行界面（getrawtransaction 和 decoderawtransaction）來檢索Alice的“原始”交易，對其進行解碼並查看它包含的內容。結果如下所示：

Alice's transaction decoded

```
{
  "version": 1,
  "locktime": 0,
```

```

"vin": [
  {
    "txid": "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
    "vout": 0,
    "scriptSig": "3045022100884d142d86652a3f47ba4746ec719bbfb040a570b1deccbb6498c75c4ae24cb02204b9f039ff08d
f09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813[ALL] 0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade84
16ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
    "sequence": 4294967295
  }
],
"vout": [
  {
    "value": 0.01500000,
    "scriptPubKey": "OP_DUP OP_HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7 OP_EQUALVERIFY OP_CHECKSIG"
  },
  {
    "value": 0.08450000,
    "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG",
  }
]
}

```

你可能只注意到有關此次交易的幾個訊息，大多數訊息缺失了！Alice的地址在哪裡？Bob的地址在哪裡？Alice發送的0.1輸入在哪裡？在比特幣中，沒有硬幣，沒有發送者，沒有接收者，沒有餘額，沒有帳戶，也沒有地址。所有這些東西都是在更高層次上構建的，以使事情更易於理解。

你可能會注意到很多奇怪的，難以辨認的欄位和十六進制字串。別擔心，我們將在本章中詳細解釋每個欄位。

## 交易的輸出和輸入

比特幣交易的基本構建區塊是 交易的輸出 *transaction output*。交易輸出是不可分割的比特幣貨幣，記錄在區塊鏈中，被整個網路識別為有效的。比特幣完整節點跟蹤所有可用和可花費的輸出，稱為 未花費的交易輸出 *unspent transaction outputs* 或 *UTXO*。所有UTXO的集合被稱為 *UTXO set*，目前有數以百萬的UTXO。UTXO集的大小隨著新UTXO的增加而增長，並在UTXO被消耗時縮小。每個交易都表示UTXO集中的更改（狀態轉移）。

當我們說用戶的錢包“收到”比特幣時，意思是錢包檢測到一個可以使用該錢包控制的密鑰來花費的UTXO。因此，用戶的比特幣“餘額”是用戶錢包可以花費的所有UTXO的總和，可以分散在數百個交易和數百個區塊中。餘額的概念是由錢包應用創建的。錢包掃描區塊鏈並將錢包可以使用它的密鑰花費的任何UTXO彙總計算用戶的餘額。大多數錢包維護資料庫或使用資料庫服務來儲存它們可以花費的所有UTXO的快照。

一個交易輸出可以有一個任意的（整數）等於satoshis倍數的值作為。正如美元可以分為小數點後兩位數字一樣，比特幣可以分為小數點後八位，作為satoshis。儘管輸出可以具有任意值，但一旦創建就是不可分割的。這是需要強調的輸出的一個重要特徵：輸出是 不連續的 和 不可分割的 的價值，以整數satoshis為單位。未使用的輸出只能由交易全部花費。

如果UTXO大於交易的期望值，它仍然必須全部使用，並且必須在交易中生成零錢。換句話說，如果你有一個價值20比特幣的UTXO，並且只需要支付1比特幣，那麼你的交易必須消費整個20比特幣的UTXO，併產生兩個輸出：一個支付1比特幣給你想要的收款人，另一個支付19比特幣回到你的錢包。由於交易輸出的不可分割性，大多數比特幣交易將不得不產生零錢。

想象一下，一個購物者購買了1.50美元的飲料，並試圖從她的錢包找到硬幣和鈔票的組合，以支付1.50美元。如果可能，購物者將找到正好好的零錢，例如，一美元鈔票和兩個二十五分硬幣（0.25美元），或小面值（六個二十五分硬幣）的組合；或者，直接向店主支付5美元，她會得到3.50美元的找零，放回她的錢包並且可用於未來的交易。

同樣，比特幣交易必須從用戶的UTXO創建，無論用戶有什麼樣的面額。用戶無法將UTXO削減一半，就像不能將美元分成兩半使用一樣。用戶的錢包應用通常會從用戶的可用UTXO中進行選擇，使組合的金額大於或等於期望交易金額。

與現實一樣，比特幣應用可以使用多種策略來滿足支付需求：合併幾個較小的單位，找到正好好的零錢，或者使用比交易價值更大的單元並進行找零。所有這些花費UTXO的複雜操作都由用戶的錢包自動完成，對用戶不可見。只有在編寫程序構建來自UTXO的原始交易時才有意義。

交易消耗先前記錄的未使用的交易輸出，並創建可供未來交易使用的新交易輸出。這樣，大量的比特幣價值通過創建UTXO的交易鏈在所有者之間轉移。

輸出和輸入鏈的例外是稱為 *幣基 coinbase* 交易的特殊類型的交易，它是每個區塊中的第一個交易。這筆交易由“獲勝”的礦工設置，創建全新的比特幣並支付給該礦工作為挖礦獎勵。此特殊的coinbase交易不消費UTXO，相反，它有一種稱為“coinbase”的特殊輸入類型。這就是比特幣在挖礦過程中創造的貨幣數量，正如我們將在 [minig] 中看到的那樣。

Tip

先有的什麼？輸入還是輸出？雞還是雞蛋？嚴格地說，輸出是第一位的，因為產生新比特幣的幣基交易沒有輸入，是憑空產生的輸出。

## 交易輸出

每筆比特幣交易都產生輸出，這些輸出記錄在比特幣賬本上。除了一個例外（參見 [\[op\\_return\]](#)），幾乎所有這些輸出都創造了稱為UTXO的可支付的比特幣，由整個網路認可並可供所有者在未來的交易中花費。

每個完整節點比特幣客戶端都跟蹤UTXO。新交易消耗（花費）UTXO集合的一個或多個輸出。

交易輸出由兩部分組成：

- 一些比特幣，最小單位為 聰 *satoshis*
- 定義了花費這些輸出所需條件的加密謎題

這個謎題也被稱為 鎖定腳本 *locking script*，見證腳本 *witness script*，或者 *scriptPubKey*。

在 [交易腳本和腳本語言](#) 中詳細討論了前面提到的鎖定腳本中使用的交易腳本語言。

現在，我們來看看Alice的交易（[交易背後](#)），看看我們是否可以識別輸出。在JSON編碼中，輸出位於名為 *vout* 的陣列（列表）中：

```
"vout": [
  {
    "value": 0.01500000,
    "scriptPubKey": "OP_DUP OP_HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7 OP_EQUALVERIFY
OP_CHECKSIG"
  },
  {
    "value": 0.08450000,
    "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG",
  }
]
```

如你所見，該交易包含兩個輸出。每個輸出由一個值和一個加密謎題定義。在Bitcoin Core顯示的編碼中，該值以比特幣為單位，但在交易本身中，它被記錄為以satoshis為單位的整數。每個輸出的第二部分是設置消費條件的加密謎題。Bitcoin Core將其顯示為 *scriptPubKey* 並展示了該腳本的人類可讀的表示。

鎖定和解鎖UTXO的主題將在稍後的 [創建腳本（鎖定 + 解鎖）](#) 中討論。在 [交易腳本和腳本語言](#) 中討論了 *scriptPubKey* 中使用的腳本語言。但在深入研究這些話題之前，我們需要了解交易輸入和輸出的總體結構。

### 交易序列化 —— 輸出

當交易通過網路傳輸或在應用程式之間交換時，它們是 *序列化* 的。序列化是將資料結構的內部表示轉換為可以一次傳輸一個字節的格式（也稱為字節流）的過程。序列化最常用於對通過網路傳輸或儲存在檔案中的資料結構進行編碼。交易輸出的序列化格式展示在 [Transaction output serialization](#) 中。

Table 1. Transaction output serialization

Size	Field	Description
8 字節 (小端序)	數量 Amount	以聰 ( $\text{satoshis} = 10^{-8}$ bitcoin) 為單位的比特幣價值
1—9 字節 (VarInt)	鎖定腳本的大小 Locking-Script Size	後面的鎖定腳本的字節數
變數	鎖定腳本 Locking-Script	定義花費該輸出的條件的腳本

大多數比特幣庫和框架在內部不以字節流的形式儲存交易，因為每次需要訪問單個欄位時都需要進行復雜的解析。為了方便和易讀，比特幣庫在資料結構（通常是物件導向的結構）中儲存交易。

從交易的字節流表示轉換為庫的內部表示資料結構的過程稱為 反序列化 *deserialization* 或 交易解析 *transaction parsing*。轉換回字節流以通過網路進行傳輸，進行雜湊或儲存在硬碟上的過程稱為 序列化 *serialization*。大多數比特幣庫具有用於交易序列化和反序列化的內置函數。

看看你是否可以從序列化的十六進制形式手動解碼Alice的交易，找到我們以前看到的一些欄位。兩個輸出部分在 [Alice's transaction, serialized and presented in hexadecimal notation](#) 中突出顯示：

Example 1. Alice's transaction, serialized and presented in hexadecimal notation

```
0100000001186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd73
4d2804fe65fa35779000000008b483045022100884d142d86652a3f47
ba4746ec719bbfb0d040a570b1deccbb6498c75c4ae24cb02204b9f039
ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813
01410484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade84
16ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc1
7b4a10fa336a8d752adfffffff0260e316000000000001976a914ab6
8025513c3dbd2f7b92a94e0581f5d50f654e788acd0ef800000000000
1976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac 00000000
```

這裡有一些提示：

- 突出顯示的部分有兩個輸出，每個輸出按照 [Transaction output serialization](#) 所示進行了序列化。
- 0.015比特幣是1,500,000聰。十六進制表示為 16 e3 60。
- 在序列化的交易中，16 e3 60 以小端序（低位字節在前）編碼，所以看起來是： 60 e3 16。
- `scriptPubKey` 的長度是 25 字節，十六進制表示為 19。

## 交易輸入

交易輸入標識（通過引用）將使用哪個UTXO並通過解鎖腳本提供所有權證明。

為了建立交易，錢包從其控制的UTXO中選擇具有足夠價值的UTXO進行所請求的付款。有時候一個UTXO就足夠了，有時候需要多個UTXO。對於將用於進行此項付款的每個UTXO，錢包將創建一個指向UTXO的輸入，並使用解鎖腳本將其解鎖。

讓我們更詳細地看看輸入的組成部分。輸入的第一部分是指向UTXO的指針，引用交易的雜湊值和輸出索引，該索引標識該交易中特定的UTXO。第二部分是一個解鎖腳本，由錢包構建，為了滿足UTXO中設置的花費條件。大多數情況下，解鎖腳本是證明比特幣所有權的數字簽名和公鑰。但是，並非所有解鎖腳本都包含簽名。第三部分是序列號，稍後將進行討論。

考慮 [交易背後](#) 中的示例，交易的輸出是 vin 陣列：

The transaction inputs in Alice's transaction

```

"vin": [
  {
    "txid": "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
    "vout": 0,
    "scriptSig" : "3045022100884d142d86652a3f47ba4746ec719bbfb040a570b1deccbb6498c75c4ae24cb02204b9f039ff08df0
9cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813[ALL] 0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416
ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
    "sequence": 4294967295
  }
]

```

如你所見，列表中只有一個輸入（因為這個UTXO包含足夠的值來完成此次付款）。輸入包含四個元素：

- 交易ID，引用包含正在使用的UTXO的交易
- 輸出索引（`vout`），標識使用來自該交易的哪個UTXO（第一個從0開始）
- `scriptSig`，滿足UTXO上的條件的腳本，用於解鎖並花費
- 一個序列號（後面討論）

在Alice的交易中，輸入指向交易ID：

```
7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18
```

輸出索引0（即由該交易創建的第一個UTXO）。解鎖腳本由Alice的錢包構建，首先檢索引用的UTXO，檢查其鎖定腳本，然後使用它構建必要的解鎖腳本以滿足它。

只看輸入內容，你可能已經注意到我們對這個UTXO一無所知，只有對包含它的交易的引用。我們不知道它的價值（satoshi的數量），也不知道設置花費條件的鎖定腳本。要找到這些訊息，我們必須通過檢索底層交易來檢索引用的UTXO。請注意，因為輸入值沒有明確說明，我們還必須使用引用的UTXO來計算將在此次交易中支付的費用（請參見[交易費用](#)）。

不僅Alice的錢包需要檢索輸入中引用的UTXO。一旦這個交易被廣播到網路中，每個驗證節點也將需要檢索在交易輸入中引用的UTXO以驗證交易。

這些交易本身似乎不完整，因為它們缺乏上下文。他們在其輸入中引用UTXO，但不檢索該UTXO，我們不知道輸入值或鎖定條件。在編寫比特幣軟體時，只要你想要驗證交易，計算費用或檢查解鎖腳本，你的程式碼首先必須從區塊鏈中檢索引用的UTXO，以便構建輸入中引用的UTXO隱含但不包括的上下文。例如，要計算支付的費用金額，你必須知道輸入和輸出值的總和。如果不檢索輸入中引用的UTXO，則不知道它們的價值。因此，像單筆交易中計費的看似簡單的操作實際上涉及多個交易的多個步驟和數據。

我們可以使用在檢索Alice的交易時使用的相同的Bitcoin Core命令序列（`getrawtransaction`和`decoderawtransaction`）。得到前面輸入中引用的UTXO：

Alice's UTXO from the previous transaction, referenced in the input

```

"vout": [
  {
    "value": 0.10000000,
    "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG"
  }
]

```

我們看到這個UTXO的值為0.1 BTC，包含一個鎖定腳本（`scriptPubKey`）：“OP\_DUP OP\_HASH160...”。

Tip

為了完全理解Alice的交易，我們必須檢索輸入引用的交易。幾乎每個比特幣庫和API中都有一個函數，用於檢索以前的交易和未使用的交易輸出。

## 交易序列化 —— 輸入

當交易被序列化以便在網路上傳輸時，它們的輸入被編碼為字節流，如 [Transaction input serialization](#) 所示。

Table 2. Transaction input serialization

Size	Field	Description
32 字節	交易的雜湊值 Transaction Hash	指向包含要花費的UTXO的交易的指針
4 字節	輸出的索引 Output Index	要花費的UTXO的索引，從0開始
1—9 字節 (VarInt)	解鎖腳本的大小 Unlocking-Script Size	後面的解鎖腳本的字節長度
變數	解鎖腳本 Unlocking-Script	滿足UTXO鎖定腳本條件的腳本
4 字節	序列號 Sequence Number	用於鎖定時間 (locktime) 或禁用 (0xFFFFFFFF)

與輸出一樣，看看是否能夠在序列化格式中查找來自Alice的交易的輸入。首先，解碼的輸入如下：

```
"vin": [
  {
    "txid": "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
    "vout": 0,
    "scriptSig" : "3045022100884d142d86652a3f47ba4746ec719bbfb040a570b1deccbb6498c75c4ae24cb02204b9f039ff08df0
9fbe9f6addac960298cad530a863ea8f53982c09db8f6e3813[ALL] 0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416
ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
    "sequence": 4294967295
  }
],
```

現在，看看我們是否可以在 [Alice's transaction, serialized and presented in hexadecimal notation](#) 中的序列化的十六進制編碼中識別這些欄位：

Example 2. Alice's transaction, serialized and presented in hexadecimal notation  
010000001186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd73  
4d2804fe65fa3577900000008b483045022100884d142d86652a3f47  
ba4746ec719bbfb040a570b1deccbb6498c75c4ae24cb02204b9f039  
ff08df09fbe9f6addac960298cad530a863ea8f53982c09db8f6e3813  
01410484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade84  
16ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc1  
7b4a10fa336a8d752adffffffff0260e31600000000001976a914ab6  
8025513c3dbd2f7b92a94e0581f5d50f654e788acd0ef800000000000  
1976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac00000 000

提示:

- 交易ID是以反向字節順序序列化的，因此它以（十六進制）18 開頭並以 79 結尾
- 輸出索引是一個4字節的零，容易識別
- scriptSig 的長度為139個字節，十六進制的 8b
- 序列號設置為 FFFFFFFF，也易於識別

## 交易費用

大多數交易包括交易費用，以獎勵比特幣礦工，保證網路安全。費用本身也可以作為一種安全機制，因為攻擊者通過大量交易充斥網路在經濟上是不可行的。[\[mining\]](#) 更詳細地討論了礦工以及礦工收取的費用和獎勵。

本節探討交易費用如何包含在典型的交易中。大多數錢包會自動計算並包含交易費用。但是，如果你以編程方式構建交易或使用命令行界面，則必須手動進行計算並包含這些費用。

交易費用是將交易納入下一個區塊的激勵措施，也是對每次交易徵收小額費用以抵制系統濫用的防範機制。交易費由礦工收集，該礦工將開採在區塊鏈上記錄交易的區塊。

交易費用是以交易數據的大小（KB）計算的，而不是比特幣交易的價值。總體而言，交易費用是根據比特幣網路內的市場力量設定的。礦工根據許多不同的優先條件（包括費用）處理交易，也可能在某些情況下免費處理交易。交易費用會影響處理優先權，這意味著如果交易費用足夠，交易就可能包含在下一個開採區塊中，而費用不足或不收費的交易可能會延遲，在幾個區塊後以盡力而為的方式處理，或者根本不處理。交易費用不是強制性的，沒有費用的交易最終可以被處理；但是，包括交易費用鼓勵優先處理。

隨著時間的推移，交易費用的計算方式以及它們對交易優先級的影響已經發生了變化。起初，交易費用在整個網路中是固定不變的。逐漸地，收費結構放鬆，並可能受到基於網路容量和交易量的市場力量的影響。至少從2016年初開始，比特幣的容量限制已經造成了交易之間的競爭，導致了更高的費用，使免費的交易成為了歷史。免費或低費用的交易很少能被開採，有時甚至不會通過網路傳播。

在Bitcoin Core中，收費中繼策略由 minrelaytxfee 選項設置。當前的預設值是每KB數據0.00001比特幣或0.01毫比特幣。因此，預設情況下，低於0.00001比特幣的交易將被視為免費，並且只在Memory pool有空間時才會被中轉；否則，它們將被丟棄。比特幣節點可以通過調整 minrelaytxfee 的值來覆蓋預設的收費中繼策略。

任何創建交易的比特幣服務，包括錢包，交易所，零售應用等，都必須實施動態費用。動態費用可以通過第三方費用估算服務或內置費用估算演算法來實現。如果你不確定，請先從第三方服務開始，如果你希望移除第三方依賴關係，設計並實現自己的演算法。

費用估算演算法根據容量和“競爭”交易提供的費用計算適當的費用。這些演算法從簡單（最後一個區塊的平均費用或中值費用）到複雜（統計分析）。他們估計必要的費用（每字節多少satoshis），使交易被選中並包含在一定數量的區塊內的可能性很高。大多數服務為用戶提供選擇高、中、低優先級費用的選項。高優先級意味著用戶支付更高的費用，但交易很可能包含在下一個區塊中。中等和低優先級意味著用戶支付較低的交易費用，但交易可能需要更長時間才能確認。

許多錢包應用使用第三方服務計算費用。一種流行的服務是 <http://bitcoinfees.21.co>，它提供了一個API和一個可視圖表，顯示了不同優先級的 satoshi/字節 費用。

Tip

比特幣網路上的固定費用已不再可行。設置固定費用的錢包將產生糟糕的用戶體驗，因為交易通常會“卡住”，不被驗證。不瞭解比特幣交易和費用的用戶會因為“停滯”的交易感到沮喪，他們會認為錢已經丟失了。

Fee estimation service [bitcoinfees.21.co](http://bitcoinfees.21.co) 中的圖表以10 satoshi/字節的增量顯示實時的費用估算值，以及每個費用範圍內的預期確認時間（以分鐘和區塊數表示）。對於每個費用範圍（例如，61-70 satoshi/字節），兩個橫條顯示了未確認交易的數量（1405）和過去24小時內的交易總數（102,975）。根據圖表，此時建議的高優先級費用為 80 satoshi / 字節，可能使交易在下一個區塊中開採（0區塊延遲）。交易規模的中位數為226字節，所以此交易規模的建議費用為 18,080 satoshis (0.00018080 BTC)。

費用估算數據可以通過簡單的HTTP REST API檢索，<https://bitcoinfees.21.co/api/v1/fees/recommended>。例如，在命令行中使用 curl 命令：

Using the fee estimation API

```
$ curl https://bitcoinfees.21.co/api/v1/fees/recommended
{"fastestFee":80,"halfHourFee":80,"hourFee":60}
```

API返回一個帶有當前費用估計的JSON物件，包含最快速度確認（fasterFee），三個區塊內確認（halfHourFee）和六個區塊內確認（hourFee）的費用，單位是 satoshi/字節。

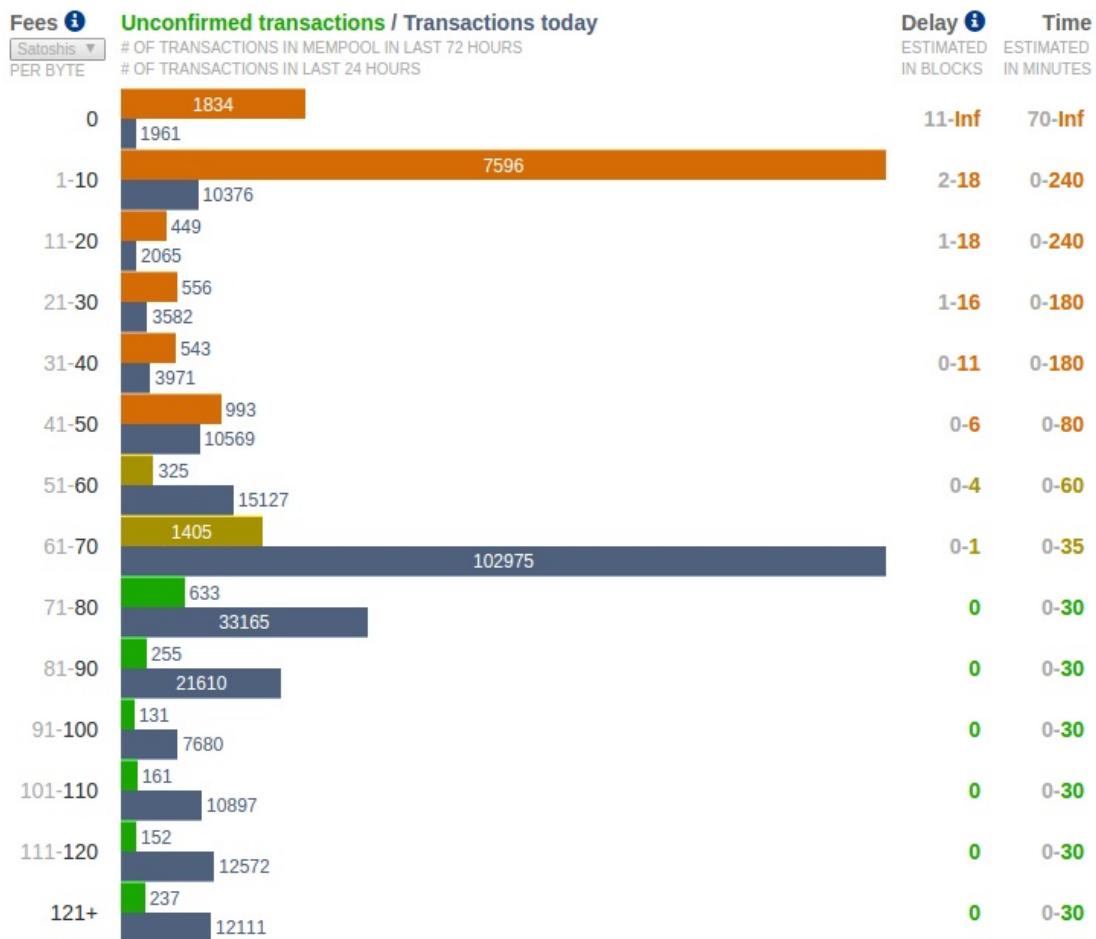


Figure 2. Fee estimation service bitcoinfees.21.co

## 將費用添加到交易

交易的資料結構沒有費用欄位。相反，費用隱含表示為輸入總和與輸出總和的差額。從所有輸入中扣除所有輸出後剩餘的金額都是礦工收取的費用：

Transaction fees are implied, as the excess of inputs minus outputs:

$$\text{Fees} = \text{Sum(Inputs)} - \text{Sum(Outputs)}$$

這是一個有點令人困惑的交易元素，也是需要理解的重要一點，因為如果你正在構建自己的交易，則必須確保你不會花費了很少的輸入卻無意中包含非常高的費用。這意味著你必須考慮所有輸入，必要時創建找零，否則最終會給礦工一個非常高的小費！

例如，如果你使用20比特幣UTXO進行1比特幣支付，則必須將19比特幣零錢輸出回你的錢包。否則，19比特幣將被算作交易費用，並將由礦工在一個區塊中進行交易。雖然你會得到優先處理並讓礦工很高興，但這可能不是你想要的。

### Warning

如果你忘記在手動構建的交易中添加找零輸出，則你將支付零錢作為交易費用。“不用找了！”可能不是你想要的。

我們再來看看Alice購買咖啡的情況，看看它在實踐中是如何運作的。愛麗絲想花0.015比特幣來買咖啡。為確保此交易得到及時處理，她希望包含交易費用，例如0.001。這意味著交易的總成本將是0.016。她的錢包因此必須提供一些UTXO，加起來0.016比特幣或更多，如有必要，可以創建找零。假設她的錢包有一個0.2比特幣的UTXO。因此，它需要消費這個UTXO，創建一個給Bob 0.015的輸出，和一個0.184比特幣的零錢輸出，返回她自己的錢包，剩下0.001比特幣未分配，作為隱含的交易費用。

現在讓我們看看不同的場景。菲律賓的兒童慈善總監Eugenia已經完成了為兒童購買教科書的籌款活動。她收到了來自世界各地的數千人的小額捐款，共計50比特幣，所以她的錢包充滿了非常多的小額未使用輸出(UTXO)。現在她想從本地出版商處購買數百本教科書，用比特幣支付。

Eugenia的錢包應用試圖構建一個較大的付款交易，因此它必須從可用的小金額UTXO集合中獲取資金。這意味著由此產生的交易將有超過一百個小型UTXO輸入，只有一個輸出支付給書籍出版商。具有許多輸入的交易將大於一千字節，也許幾千字節大小。因此，它需要比中等規模交易高得多的費用。

Eugenia的錢包應用程式將通過衡量交易規模並將其乘以每千字節的費用來計算適當的費用。許多錢包會為較大的交易多付費用，以確保交易得到及時處理。較高的費用並不是因為Eugenia花費更多的錢，而是因為她的交易規模更大更復雜 - 收費與交易的比特幣價值無關。

## 交易腳本和腳本語言

比特幣交易腳本語言，稱為 *Script*，是一種類似Forth的逆波蘭表示法的基於堆疊的執行語言。如果這聽起來像是胡言亂語，那麼你可能沒有研究過60年代的程式語言，但沒關係 - 我們將在本章中解釋它。放置在UTXO上的鎖定腳本和解鎖腳本都是用這種腳本語言編寫的。當一個交易被驗證時，每個輸入中的解鎖腳本將與相應的鎖定腳本一起執行，以查看它是否滿足花費條件。

腳本是一種非常簡單的語言，在有限的範圍內設計，可在一系列硬體上執行，可能與嵌入式設備一樣簡單。它只需要很少的處理，並且不能完成許多現代程式語言能夠做的事情。為了用於驗證可編程的金錢，這是一個深思熟慮的安全特性。

今天，大多數通過比特幣網路處理的交易具有“支付給Bob的比特幣地址”的形式，並且基於稱為 Pay-to-Public-Key-Hash (付費到公鑰雜湊) 的腳本。但是，比特幣交易不限於“支付給Bob的比特幣地址”類型的腳本。事實上，可以編寫鎖定腳本來表達各種複雜的條件。為了理解這些更復雜的腳本，我們必須首先了解交易腳本和腳本語言的基礎知識。

在本節中，我們將演示比特幣交易腳本語言的基本組件，並說明如何使用它來表達簡單的花費條件以及解鎖腳本如何滿足這些條件。

Tip

比特幣交易驗證不是基於靜態模式的，而是通過執行腳本語言來實現的。這種語言允許表示幾乎無限的各種條件。這就是比特幣如何獲得“可編程金錢”力量的。

## 圖靈不完備

比特幣交易腳本語言包含許多操作符，但是故意在一個重要方面進行了限制 - 除了條件控制外，沒有迴圈或複雜的流程控制功能。這確保語言不是 圖靈完備 *Turing Complete* 的，這意味著腳本具有有限的複雜性和可預測的執行時間。腳本不是通用語言。這些限制確保了該語言不能用於創建無限迴圈或其他形式的“邏輯炸彈”，這種“邏輯炸彈”可能嵌入交易中，導致對比特幣網路的拒絕服務攻擊。請記住，每筆交易都由比特幣網路上的每個完整節點驗證。有限制的語言會阻止交易驗證機制被當作漏洞。

## 無狀態驗證

比特幣交易腳本語言是無狀態的，在執行腳本之前沒有狀態，在執行腳本之後也不保存狀態。因此，執行腳本所需的所有訊息都包含在腳本中。腳本在任何系統上都能可預測地執行。如果你的系統驗證了腳本，你可以確定比特幣網路中的其他每個系統都會驗證該腳本，這意味著有效的交易對每個人都有效，每個人都知道這一點。結果的可預測性是比特幣系統的一個重要好處。

## 創建腳本 (鎖定 + 解鎖)

比特幣的交易驗證引擎依靠兩種類型的腳本來驗證交易：鎖定腳本和解鎖腳本。

鎖定腳本是放置在輸出上的花費條件：它指定將來要花費輸出必須滿足的條件。由於歷史原因，鎖定腳本被稱為 *scriptPubKey*，因為它通常包含公鑰或比特幣地址（公鑰的雜湊）。在本書中，我們將其稱為“鎖定腳本”，以表示此腳本技術更廣泛的可能性。在大多數比特幣應用中，我們所稱的鎖定腳本將作為 *scriptPubKey* 出現在源程式碼中。你還會看到被稱為 *witness script* 的鎖定腳本（參見 [\[segwit\]](#)）或更一般地稱為 *cryptographic puzzle*。這些術語在不同的抽象層次代表著相同的東西。

解鎖腳本是可以“解決”或滿足鎖定腳本放置到輸出上的條件，從而花費輸出的腳本。解鎖腳本是每個交易輸入的一部分。大多數情況下，它們包含用戶錢包利用私鑰生成的數字簽名。由於歷史原因，解鎖腳本被稱為 *scriptSig*，因為它通常包含數字簽名。在大多數比特幣應用中，源程式碼將解鎖腳本稱為 *scriptSig*。你還將看到稱為 *witness* 的解鎖腳本（參見[\[segwit\]](#)）。在本書中，我們將其稱為“解鎖腳本”來表示更廣泛的鎖定腳本，因為並非所有解鎖腳本都必須包含簽名。

每個比特幣驗證節點通過一起執行鎖定和解鎖腳本來驗證交易。每個輸入都包含一個解鎖腳本，並引用先前存在的 UTXO。驗證軟體將複製解鎖腳本，檢索輸入引用的UTXO，並從該UTXO複製鎖定腳本。然後按順序執行解鎖和鎖定腳本。如果解鎖腳本滿足鎖定腳本條件，則輸入有效（參見 [單獨執行解鎖和鎖定腳本](#)）。所有輸入都是作為交易整體驗證的一部分獨立驗證的。

請注意，UTXO永久記錄在區塊鏈中，因此不會改變，也不會因為在新交易中花費它的失敗嘗試而受到影響。只有正確滿足輸出條件的有效交易才會導致輸出被視為“已花費”並從未使用的交易輸出集和（UTXO集）中移除。

[Combining scriptSig and scriptPubKey to evaluate a transaction script](#) 是最常見類型的比特幣交易（支付到公鑰的雜湊）的解鎖和鎖定腳本示例，顯示了在腳本驗證之前將解鎖腳本和鎖定腳本連接在一起所產生的組合腳本。

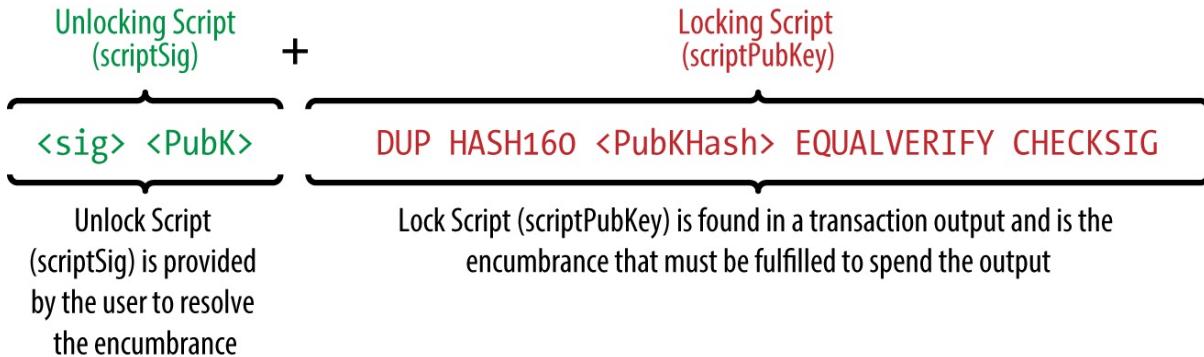


Figure 3. Combining scriptSig and scriptPubKey to evaluate a transaction script

### 腳本執行堆疊

比特幣的腳本語言稱為基於堆疊的語言，因為它使用稱為 堆疊 *stack* 的資料結構。堆疊是一個非常簡單的資料結構，可以將其視為一疊卡片。一個堆疊允許兩個操作：push和pop。Push會在堆疊頂部添加一個項目。Pop從堆疊中刪除頂部的項目。堆疊上的操作只能作用於堆疊中最頂端的項目。堆疊資料結構也稱為後進先出或“LIFO”隊列。

腳本語言通過從左向右處理每個項目來執行腳本。“數字”（數據常量）被push進入堆疊。“操作”從堆疊中pop一個或多個參數，執行操作，並可能將結果push到堆疊。例如，OP\_ADD 會從堆疊中彈出兩個項目，做加法，並將結果push到堆疊上。

條件運算符評估一個條件，產生TRUE或FALSE的布林結果。例如，OP\_EQUAL pop堆疊中的兩個項目，如果它們相等，則push TRUE (TRUE由數字1表示)，如果不相等，則push FALSE (由零表示)。比特幣交易腳本通常包含一個條件操作符，以便它們可以生成表示有效交易的TRUE結果。

### 一個簡單的腳本

現在讓我們將有關腳本和堆疊的知識應用於一些簡單的示例。

在 [Bitcoin's script validation doing simple math](#) 中，腳本 2 3 OP\_ADD 5 OP\_EQUAL 演示了算術加法運算符 OP\_ADD，將兩個數字相加並將結果放在堆疊上，後面跟著條件運算符 OP\_EQUAL，它檢查結果總和是否相等到 5。為簡潔起見，在示例中省略了 OP\_ 前綴。有關可用腳本運算符和函數的更多詳細訊息，請參見 [\[tx\\_script\\_ops\]](#)。

雖然大多數鎖定腳本都是指公鑰雜湊（本質上是比特幣地址），因此需要所有權證明來支付資金，腳本並不一定非常複雜。生成TRUE值的鎖定和解鎖腳本的任何組合都是有效的。我們用作腳本語言示例的簡單算術也是一個有效的鎖定腳本，可用於鎖定交易輸出。

使用算術示例腳本的一部分作為鎖定腳本：

```
3 OP_ADD 5 OP_EQUAL
```

可以被包含以下解鎖腳本的交易滿足：

```
2
```

驗證軟體將鎖定和解鎖腳本結合在一起：

```
2 3 OP_ADD 5 OP_EQUAL
```

正如我們在 [Bitcoin's script validation doing simple math](#) 中的示例中看到的，執行此腳本時，結果為 OP\_TRUE，交易有效。這不僅是一個有效的交易輸出鎖定腳本，而且由此產生的UTXO可以被具有任何知道數字2滿足腳本的人花費。

Tip

如果堆疊頂層結果為 TRUE（標記為 {0x01}），任何其他非零值，或者腳本執行後堆疊為空，則交易有效。如果堆疊頂部的值為 FALSE（一個零長度的空值，標記為 {}），或者腳本被運算符顯式終止了，例如 OP\_VERIFY，OP\_RETURN 或一個條件終止符，如 OP\_ENDIF，則交易無效。詳細訊息，請參見 [\[tx\\_script\\_ops\]](#)。

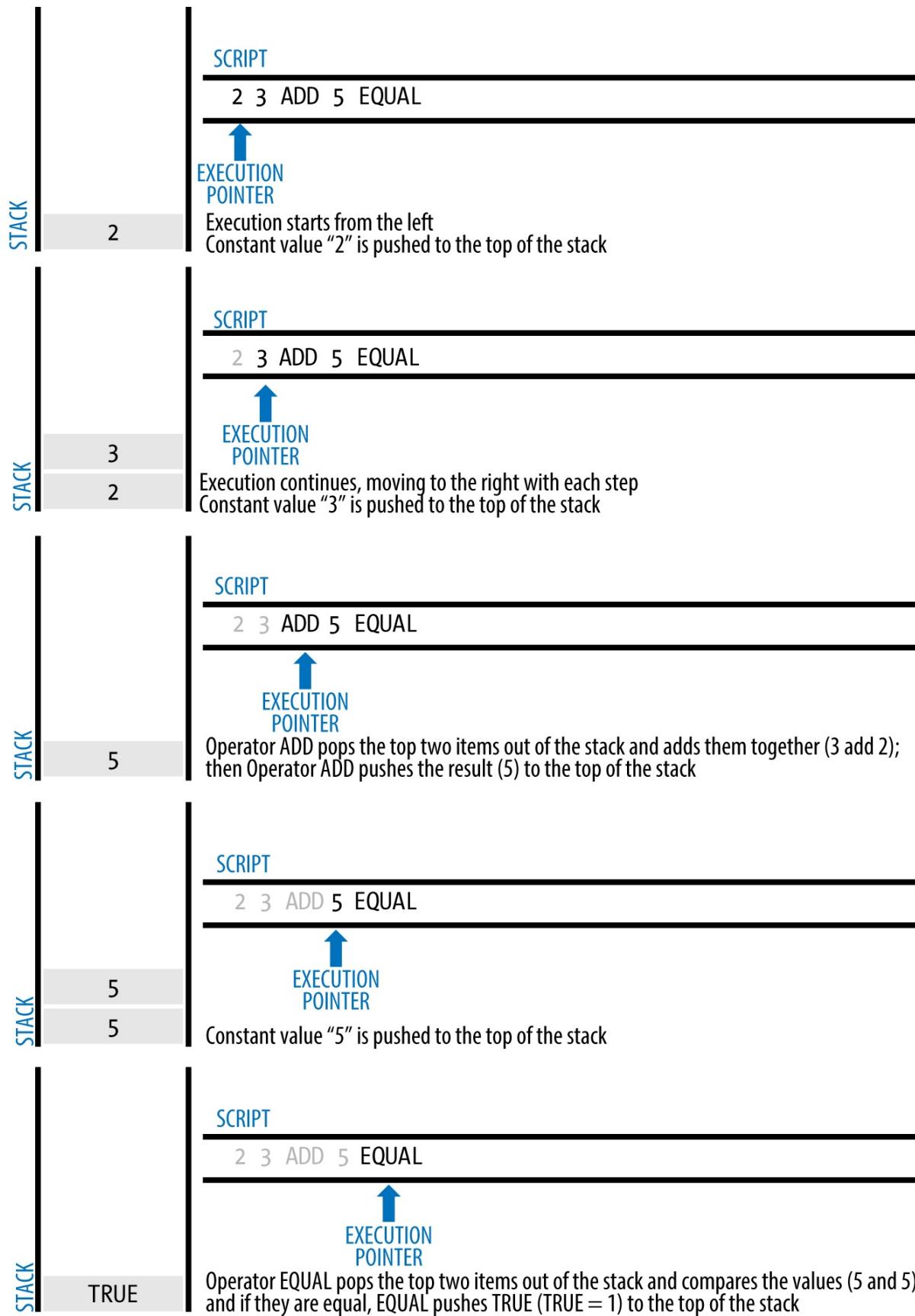


Figure 4. Bitcoin's script validation doing simple math

以下是一個稍微複雜的腳本，計算  $2 + 7 - 3 + 1$ 。請注意，當腳本在一行中包含多個運算符時，堆疊允許一個運算符的結果由下一個運算符執行：

```
2 7 OP_ADD 3 OP_SUB 1 OP_ADD 7 OP_EQUAL
```

嘗試使用筆和紙驗證前面的腳本。當腳本執行結束時，在堆疊中應該保留值 TRUE。

#### 單獨執行解鎖和鎖定腳本

在原始的比特幣客戶端中，解鎖和鎖定腳本按順序連接並執行。出於安全原因，2010年發生了變化，原因是存在一個漏洞，允許惡意解鎖腳本將數據推送到堆疊並破壞鎖定腳本。在當前的實現中，如下所述，腳本是在兩次執行之間傳輸堆疊的情況下單獨執行的。

首先，使用堆疊執行引擎執行解鎖腳本。如果解鎖腳本沒有錯誤地執行（例如，它沒有遺留的“懸掛（dangling）”操作符），則複製主堆疊並執行鎖定腳本。如果使用從解鎖腳本複製的堆疊數據執行鎖定腳本的結果為“TRUE”，則解鎖腳本已成功解決由鎖定腳本施加的條件，證明該輸入是用於花費UTXO的有效授權。如果在執行組合腳本後仍然存在除“TRUE”之外的結果，則輸入無效，因為它未能滿足放置在UTXO上的消費條件。

### 支付到公鑰雜湊 Pay-to-Public-Key-Hash (P2PKH)

在比特幣網路上處理的絕大多數交易花費由支付到公鑰雜湊（P2PKH）鎖定的輸出這些輸出包含一個鎖定腳本。這些輸出包含將它們鎖定到公鑰雜湊（比特幣地址）的腳本。由P2PKH腳本鎖定的輸出可以通過出示公鑰，和由相應私鑰創建的數字簽名來解鎖（花費）（參見 [數字簽名 \(ECDSA\)](#)）。

例如，讓我們再看看Alice對Bob's Cafe的付款。Alice向咖啡廳的比特幣地址支付了0.015比特幣。該交易輸出將具有以下形式的鎖定腳本：

```
OP_DUP OP_HASH160 <Cafe Public Key Hash> OP_EQUALVERIFY OP_CHECKSIG
```

Cafe Public Key Hash 等同於咖啡館的比特幣地址，沒有Base58Check編碼。大多數應用程式會以十六進制編碼顯示 *public key hash*，而不是以“1”開頭的大家熟悉的比特幣地址Base58Check格式。

上述鎖定腳本可以由以下形式的解鎖腳本滿足：

```
<Cafe Signature> <Cafe Public Key>
```

這兩個腳本組合在一起形成以下的驗證腳本：

```
<Cafe Signature> <Cafe Public Key> OP_DUP OP_HASH160
<Cafe Public Key Hash> OP_EQUALVERIFY OP_CHECKSIG
```

執行時，只有在解鎖腳本與鎖定腳本設置的條件匹配時，此組合腳本才會輸出TRUE。換句話說，如果解鎖腳本具有來自咖啡館的私鑰的有效簽名，該公鑰對應於公鑰雜湊集作為負擔，則結果為TRUE。

圖 #P2PubKHash1 和 #P2PubKHash2 顯示（分兩部分）了逐步執行的組合腳本，證明這是一個有效的交易。

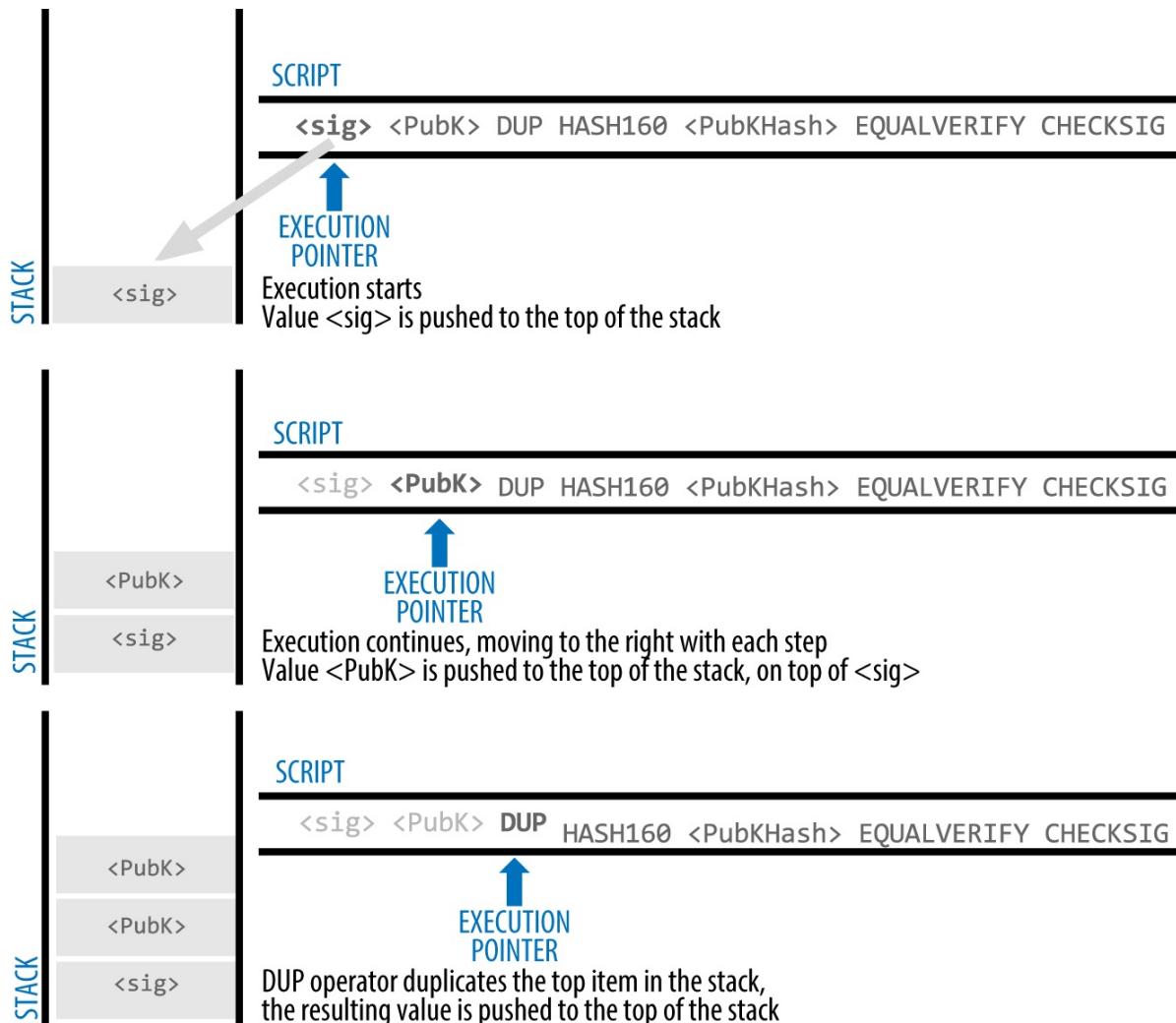


Figure 5. Evaluating a script for a P2PKH transaction (part 1 of 2)

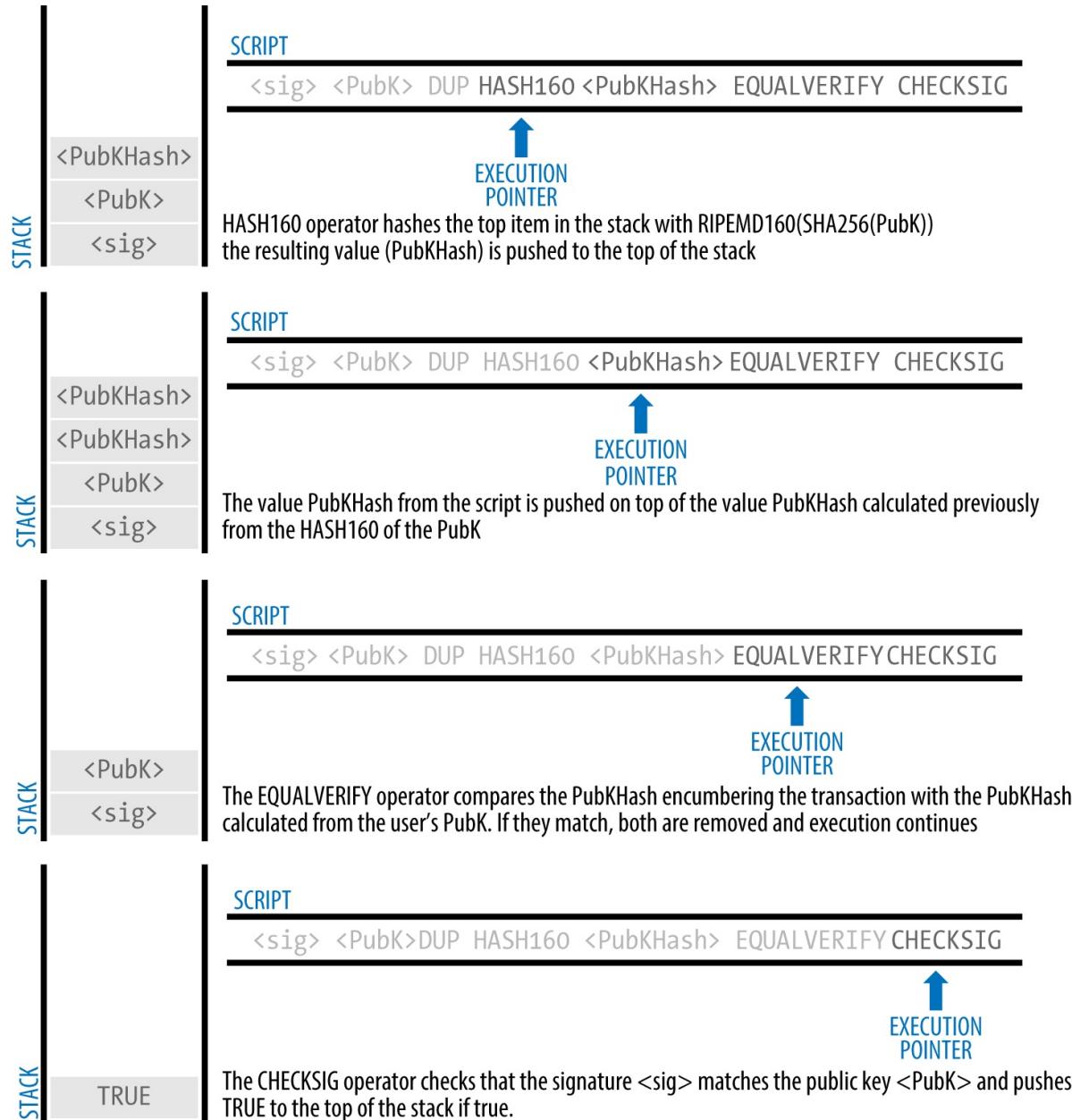


Figure 6. Evaluating a script for a P2PKH transaction (part 2 of 2)

## 數字簽名 (ECDSA)

到目前為止，我們還沒有深入探討“數字簽名”的細節。在本節中，我們將探討數字簽名如何工作，以及如何在不洩露私鑰的情況下提供私鑰的所有權證明。

比特幣中使用的數字簽名演算法是 *Elliptic Curve Digital Signature Algorithm* 或 *ECDSA*。ECDSA是用於基於橢圓曲線私鑰/公鑰對的數字簽名的演算法，如 [elliptic\_curve] 中所述。ECDSA由腳本函數 OP\_CHECKSIG, OP\_CHECKSIGVERIFY, OP\_CHECKMULTISIG 和 OP\_CHECKMULTISIGVERIFY 使用。無論何時，你在鎖定腳本中看到這些腳本的話，解鎖腳本都必須包含ECDSA簽名。

數字簽名在比特幣中有三個用途（參見下面的邊欄）。首先，簽名證明私鑰的所有者，暗示資金的所有者，已經 授權 支出這些資金。其次，授權證明是 不可否認的 *undeniable* (*nonrepudiation*)。第三，簽名證明交易（或交易的特定部分）在簽名後沒有也不能被任何人修改。

請注意，交易的每個輸入都是獨立簽署的。這是至關重要的，因為簽名和輸入都不必屬於同一個“所有者”或被其使用。事實上，一個名為“CoinJoin”的特定交易方案利用這一事實來創建隱私的多方交易。

**Note**

交易的每個輸入及其可能包含的任何簽名完全獨立於任何其他輸入或簽名。多方可以協作構建交易並各自簽署一個輸入。

Wikipedia's Definition of a "Digital Signature"

A digital signature is a mathematical scheme for demonstrating the authenticity of a digital message or documents. A valid digital signature gives a recipient reason to believe that the message was created by a known sender (authentication), that the sender cannot deny having sent the message (nonrepudiation), and that the message was not altered in transit (integrity).

Source: [https://en.wikipedia.org/wiki/Digital\\_signature](https://en.wikipedia.org/wiki/Digital_signature)

## 數字簽名如何工作

數字簽名是由兩部分組成的數學模式 *mathematical scheme*。第一部分是使用私鑰（簽名密鑰）從訊息（交易）創建簽名的演算法。第二部分是，允許任何人使用訊息和公鑰驗證簽名的演算法

### 創建數字簽名

在比特幣的ECDSA演算法實現中，被簽名的“訊息”是交易，或者更準確地說是交易中特定數據子集的雜湊（參見 [簽名雜湊的類型 \(SIGHASH\)](#)）。簽名密鑰是用戶的私鑰。結果是如下簽名：

$\text{Sig} = F_{\text{sig}}(F_{\text{hash}}(m), dA)$

其中：

- $dA$  是簽名私鑰
- $m$  是交易（或交易的一部分）
- $F_{\text{hash}}$  是雜湊函數
- $F_{\text{sig}}$  是簽名演算法
- $Sig$  是簽名結果

更多關於ECDSA的細節可以在 [ECDSA 數學](#) 中找到。

$F_{\text{sig}}$  方法生成簽名  $Sig$ ，由兩部分組成：R 和 S:

$Sig = (R, S)$

現在已經計算了兩個值+ R 和 S +，它們使用稱為 *Distinguished Encoding Rules* 或 *DER* 的國際標準編碼方案序列化為字節流。

### 簽名的序列化 (DER)

讓我們再看一下Alice創建的交易。在交易輸入中有一個解鎖腳本，其中包含來自Alice錢包的DER編碼簽名：

```
3045022100884d142d86652a3f47ba4746ec719bbfb040a570b1deccbb6498c75c4ae24cb02204b9f0
39ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e381301
```

該簽名是Alice的錢包生成的 R 和 S 的序列化字節流，用於證明她擁有授權使用該輸出的私鑰。序列化格式由以下九個元素組成：

- 0x30 —— 標識 DER 序列的開始
- 0x45 —— 序列長度 (69 bytes)

- 0x02 —— 接下來是一個整數
- 0x21 —— 整數的長度 (33 bytes)
- R —— 00884d142d86652a3f47ba4746ec719bbfb0d040a570b1deccbb6498c75c4ae24cb
- 0x02 —— 接下來是另一個整數
- 0x20 —— 另一個整數的長度 (32 bytes)
- S —— 4b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813
- 一個後綴 (0x01) 標識使用的雜湊類型 (SIGHASH\_ALL)

看看你是否可以使用這個列表解碼Alice的序列化（DER編碼）簽名。重要的數字是 R 和 S；其餘的數據是DER編碼方案的一部分。

## 驗證簽名

要驗證簽名，必須拿到簽名（R 和 S），序列化交易和公鑰（對應的用於創建簽名的私鑰）。實質上，對簽名的驗證意味著“只有生成此公鑰的私鑰的所有者才能在此交易上產生此簽名”。

簽名驗證演算法採用訊息（交易或其部分數據的雜湊），簽名者的公鑰和簽名（R 和 S 值），如果簽名對此訊息和公鑰有效，則返回TRUE。

## 簽名雜湊的類型 (SIGHASH)

數字簽名是應用於訊息的，對比特幣來說，訊息就是交易。簽名意味著簽名者對具體交易數據的保證 *commitment*。最簡單的形式是，簽名應用於整個交易，從而保證所有輸入，輸出和其他交易欄位。但是，簽名也可以只保證交易中的一部分數據，在許多場景下很有用，我們將在本節中看到。

比特幣的簽名可以使用 SIGHASH 指示交易數據的哪部分包含在由私鑰簽名的雜湊中。SIGHASH 標誌是附加到簽名後面的單個字節。每個簽名都有一個 SIGHASH 標誌，並且該標誌對於不同輸入是不同的。具有三個簽名輸入的交易可以具有三個不同的帶有 SIGHASH 標誌的簽名，每個簽名簽署（保證）交易的不同部分。

請記住，每個輸入都能在其解鎖腳本中包含一個簽名。因此，包含多個輸入的交易可能具有不同的帶有 SIGHASH 標誌的簽名，這些標誌會在每個輸入中保證交易的不同部分。還要注意的是，比特幣交易可能包含來自不同“所有者”的輸入，他們可能在部分構建的（無效的）交易中僅簽署一個輸入，需要其他人合作收集所有必要的簽名才能進行有效交易。許多 SIGHASH 標誌類型只有在你認為多位參與者在比特幣網路之外協作並各自更新部分簽名的交易時才有意義。

有三種 SIGHASH 標誌: ALL, NONE, 和 SINGLE, 如 [SIGHASH types and their meanings](#) 所示。

Table 3. SIGHASH types and their meanings

SIGHASH flag	Value	Description
ALL	0x01	簽名應用於所有輸入和輸出。
NONE	0x02	簽名應用於所有輸入，不包括任何輸出
SINGLE	0x03	簽名應用於所有輸入，但僅應用於與簽名輸入具有相同索引編號的一個輸出

另外，還有一個修飾符標誌 SIGHASH\_ANYONECANPAY，它可以與前面的每個標誌結合使用。當設置了 ANYONECANPAY 時，只有一個輸入被簽名，剩下的（及其序列號）保持開放可以修改。ANYONECANPAY 的值為 0x80，並按位OR應用，生成組合的標誌，如 [SIGHASH types with modifiers and their meanings](#) 所示。

Table 4. SIGHASH types with modifiers and their meanings

SIGHASH flag	Value	Description
ALL ANYONECANPAY	0x81	簽名應用於一個輸入和所有輸出
NONE ANYONECANPAY	0x82	簽名應用於一個輸入，不應用於輸出
SINGLE ANYONECANPAY	0x83	簽名應用於一個輸入和有相同索引號的輸出

在簽名和驗證過程中應用 SIGHASH 標誌的方式是創建交易的副本，將內部的某些欄位截斷（設置長度為零並清空）。將產生的交易序列化。將 SIGHASH 標誌添加到序列化交易的末尾，並對結果進行雜湊雜湊。雜湊本身就是被簽名的“訊息”。根據使用哪個 SIGHASH 標誌，交易的不同部分被截斷。結果雜湊取決於交易中數據的不同子集。在雜湊之前最後一步包含了 SIGHASH，簽名也保證了 SIGHASH 類型，不能被（礦工）改變。

#### Note

所有 SIGHASH 類型都簽署了交易的 nLocktime 欄位（請參見 [\[transaction\\_locktime\\_nlocktime\]](#)）。另外，SIGHASH 類型本身在簽名之前附加到交易中，在簽名後不能修改。

在Alice的交易示例中（請參見 [簽名的序列化 \(DER\)](#) 中的列表），我們看到DER編碼簽名的最後一部分是 01，它是 SIGHASH\_ALL 標誌。這會鎖定交易數據，所以Alice的簽名會保證所有輸入和輸出的狀態。這是最常見的簽名形式。

讓我們看看其他類型的 SIGHASH 以及它們如何在實踐中使用：

#### ALL|ANYONECANPAY

這種結構可以用來進行“眾籌”式的交易。試圖籌集資金的人可以創建一個單一輸出的交易。單一輸出向資金籌集人支付“目標”金額。這樣的交易顯然是無效的，因為它沒有輸入。現在，其他人可以通過添加自己的輸入來進行修改這筆交易，作為捐贈。他們用 ALL|ANYONECANPAY 來簽名自己的輸入。除非收集到足夠的投入，達到輸出的價值，否則交易無效。每一筆捐款都是一種“承諾/抵押”，在籌集到目標金額之前，籌款不能收回。

#### NONE

這種結構可用於創建特定數量的“不記名支票”或“空白支票”。它交付輸入，但允許更改輸出鎖定腳本。任何人都可以將自己的比特幣地址寫入輸出鎖定腳本並贖回資金。但是，輸出值本身被簽名鎖定。

#### NONE|ANYONECANPAY

這種結構可以用來建立一個“集塵器”。錢包裡有微型UTXO的用戶，如果不支付超過灰塵價值的費用，就無法消費這些東西。有了這種簽名，微型UTXO可以捐贈給任何人，聚集並在任何時候花費它們。

有一些關於修改或擴展 SIGHASH 系統的建議。其中一個是 Blockstream 的 Glenn Willen 提出的 *BitTek Sighash Modes*，是 Elements 項目的一部分。它旨在創建一個靈活的 SIGHASH 類型替代方案，允許“輸入和輸出的任意的，礦工可重寫的位掩碼”，可以表達“更復雜的合同預先承諾方案，例如在分佈式資產交換中簽署帶有更改的報價”。

#### Note

你不會在用戶的錢包應用程式中看到+ SIGHASH 標誌選項。除了少數例外，錢包構建P2PKH腳本並使用 +SIGHASH\_ALL 標誌進行簽名。要使用不同的 SIGHASH 標誌，你必須編寫軟體來創建和簽署交易。更重要的是，SIGHASH 標誌可以被特殊用途的比特幣應用程式使用，實現新用途。

## ECDSA 數學

如前所述，簽名是由一個數學函數  $F_{sig}$  創建的，產生由兩個值  $R$  和  $S$  組成的簽名。在本節中，我們將更詳細地討論函數  $F_{sig}$ 。

簽名演算法首先生成 *ephemeral*（臨時）私鑰公鑰對。在涉及簽名私鑰和交易雜湊的轉換之後，此臨時密鑰對用於計算  $R$  和  $S$  值。

臨時密鑰對基於隨機數  $k$ ，也就是臨時私鑰。從  $k$  開始，我們生成相應的臨時公鑰  $P$ （按照  $P = k * G$  計算，與比特幣公鑰的生成方式相同；參見 [\[pubkey\]](#)）。數字簽名的  $R$  值就是臨時公鑰  $P$  的  $x$  座標。

演算法計算簽名的S值，如下：

$$S = k^{-1} (Hash(m) + dA * R) \bmod p$$

其中：

- $k$  是臨時私鑰
- $R$  是臨時公鑰的 x 座標
- $dA$  是簽名私鑰
- $m$  是交易數據
- $p$  是橢圓曲線的主要階數

“驗證”是簽名生成函數的反函數，使用  $R$ ， $S$  值和公鑰來計算一個值  $P$ ，它是橢圓曲線上的一個點（簽名創建中使用的臨時公鑰）：

$$P = S^{-1} * Hash(m) * G + S^{-1} * R * Qa$$

where:

- $R$  和  $S$  是簽名的值
- $Qa$  是Alice的公鑰
- $m$  是被簽名的交易數據
- $G$  是橢圓曲線的生成點

如果計算點  $P$  的 x 座標等於  $R$ ，那麼驗證者可以推斷簽名是有效的。

請注意，在驗證簽名時，沒有用到私鑰，也不會被洩露。

Tip

ECDSA是一門相當複雜的數學；完整的解釋超出了本書的範圍。許多優秀的在線指南會一步一步地講解它：搜索“ECDSA解釋”或嘗試這一個：<http://bit.ly/2r0HhGB>。

## 隨機性在簽名中的重要性

正如我們在 [ECDSA 數學](#) 中看到的，簽名生成演算法使用隨機密鑰  $k$  作為臨時私鑰/公鑰對的基礎。 $k$  的值並不重要，只要它是隨機的。如果使用相同的值  $k$  在不同的訊息（交易）上生成兩個簽名，那麼則任何人都可以計算簽名私鑰。在簽名演算法中重複使用  $k$  的相同值會導致私鑰的暴露！

Warning

如果在兩個不同交易的簽名演算法中使用相同的  $k$ ，則可以計算私鑰並將其公開給全世界！

這不僅僅是一種理論上的可能性。我們已經看到這個問題導致私鑰暴露在比特幣的幾種不同的交易簽名演算法中。由於無意中重複使用  $k$  值，有人資金被盜。重用  $k$  值的最常見原因是沒有初始化正確的隨機數生成器。

為避免此漏洞，最佳做法是不生成帶有熵的隨機數生成器的  $k$ ，而是使用通過交易數據本身作為種子的確定性隨機過程。這確保每筆交易產生不同的  $k$ 。 $k$  的確定性初始化的行業標準演算法在 Internet Engineering Task Force 發佈的 [RFC 6979](#) 中定義。

如果你正在實施一種演算法來簽署比特幣交易，你必須使用RFC 6979或類似的確定性隨機演算法來確保你為每筆交易生成不同的  $k$ 。

## 比特幣地址，餘額和其他抽象

我們發現交易在“幕後”看起來與它們在“錢包”，區塊鏈瀏覽器，和其他面向用戶的應用程式中的呈現方式非常不同。交易的結構中似乎沒有來自前幾章的許多簡單和熟悉的概念，比如比特幣地址和餘額。我們看到交易本身不包含比特幣地址，而是通過鎖定和解鎖比特幣的離散值的腳本進行操作。餘額不存在於此系統的任何位置，但每個錢包應用程式會突出顯示用戶錢包的餘額。

現在我們已經研究了實際包含在比特幣交易中的內容，我們可以研究更高層次的抽象是如何從交易的看似原始的組成部分中獲得的。

讓我們再看看Alice的交易是如何在區塊鏈瀏覽器（Alice's transaction to Bob's Cafe）上展示的。

## Transaction View information about a bitcoin transaction

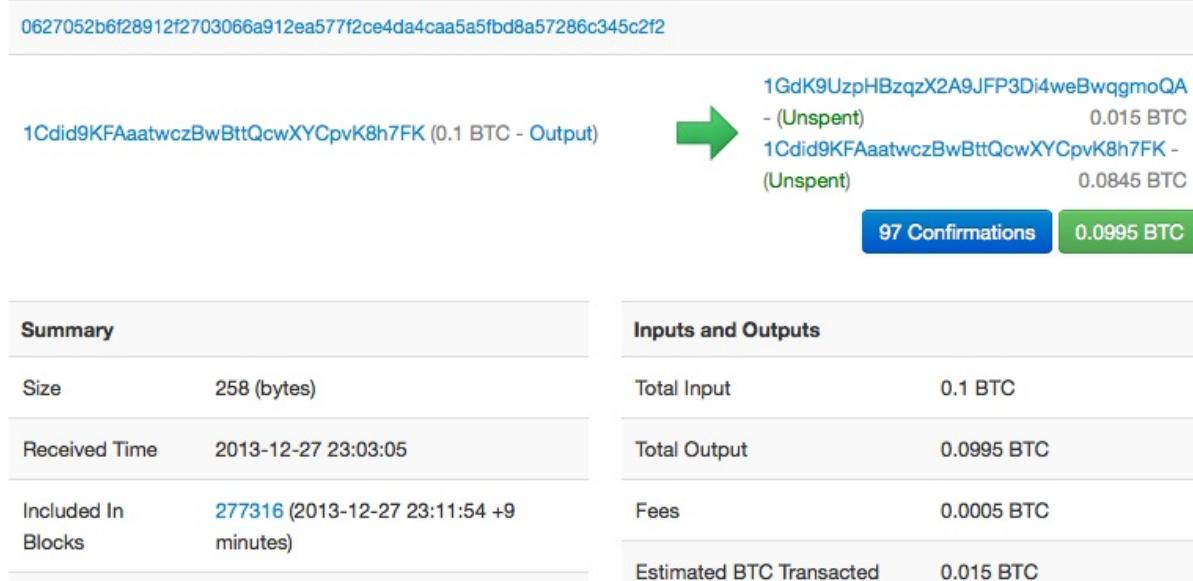


Figure 7. Alice's transaction to Bob's Cafe

在交易左側，區塊鏈瀏覽器顯示Alice的比特幣地址為“發件人”。事實上，這些訊息並不在交易本身中。當區塊鏈瀏覽器檢索到該交易時，它還檢索到輸入中引用的前一個交易，並從這個之前的交易中提取第一個輸出。該輸出中是一個鎖定腳本，將UTXO鎖定到Alice的公鑰雜湊（一個P2PKH腳本）。區塊鏈瀏覽器提取公鑰雜湊並使用Base58Check編碼對其進行編碼，以生成並顯示代表該公鑰的比特幣地址。

同樣，在右側，區塊鏈瀏覽器顯示了兩個輸出；第一個是Bob的比特幣地址，第二個是Alice的比特幣地址（找零）。再次，為了創建這些比特幣地址，區塊鏈瀏覽器從每個輸出中提取鎖定腳本，將其識別為P2PKH腳本，並從內部提取公鑰雜湊。最後，區塊鏈瀏覽器使用Base58Check重新編碼該公鑰，以生成並顯示比特幣地址。

如果你點擊了Bob的比特幣地址，區塊鏈瀏覽器會顯示 The balance of Bob's bitcoin address 中的視圖。

## Bitcoin Address Addresses are identifiers which you use to send bitcoins to another person.



Figure 8. The balance of Bob's bitcoin address

區塊鏈瀏覽器顯示Bob的比特幣地址的餘額。但比特幣系統中沒有任何地方存在“餘額”的概念。這裡顯示的值是由區塊鏈瀏覽器構建的，如下所示。

為了構建“總共收到的”金額，區塊鏈瀏覽器首先解碼比特幣地址的Base58Check編碼，以檢索編碼在地址中的Bob的公鑰的160位雜湊。然後，區塊鏈瀏覽器將搜索交易資料庫，尋找包含Bob公鑰雜湊P2PKH鎖定腳本的輸出。通過彙總所有輸出的值，區塊鏈瀏覽器可以產生收到的總價值。

構建當前餘額（顯示為“最終餘額 Final Balance”）需要更多的工作。區塊鏈瀏覽器維護了目前未使用的輸出的單獨的資料庫，即UTXO集。為了維護此資料庫，區塊鏈瀏覽器必須實時監控比特幣網路，添加新創建的UTXO，並實時刪除已花費的UTXO，當它們出現在未經確認的交易中時。這是一個複雜的過程，它依賴於跟蹤交易的傳播過程，以及與比特幣網路保持一致，以確保遵循正確的鏈條。有時，區塊鏈瀏覽器不同步，並且其UTXO集的視角不完整或不正確。

從UTXO集合中，區塊鏈瀏覽器彙總所有引用Bob的公鑰雜湊值的未使用輸出的值，併產生顯示給用戶的“最終餘額”數字。

為了製作這一張帶兩個“餘額”圖片，區塊鏈瀏覽器必須對幾十，幾百甚至幾十萬的交易進行索引和搜索。

總之，錢包應用程式，區塊鏈瀏覽器和其他比特幣用戶界面呈現給用戶的訊息通常由更高級別的抽象組成，這些抽象通過搜索許多不同的交易，檢查其內容並操縱其中包含的數據而派生。為了呈現這種簡單的比特幣交易視圖，類似於從一個發件人到一個收件人的銀行支票，這些應用程式必須抽象許多底層細節。他們主要關注常見類型的交易：P2PKH 和 SIGHASH\_ALL 在每個輸入上簽名。因此，雖然比特幣應用程式可以以易於閱讀的方式呈現超過80%的交易，但它們有時會被偏離規範的交易所難倒。包含更復雜的鎖定腳本，或不同的 SIGHASH 標誌，或許多輸入和輸出的交易，表明了這些抽象的簡單性和缺陷。

每天，在區塊鏈中確認數百個不包含P2PKH輸出的交易。區塊鏈瀏覽器通常會用紅色警告訊息顯示他們無法解碼地址。以下鏈接包含未完全解碼的最新的“奇怪交易”：[\[https://www.blockchain.com/btc/strange-transactions\]](https://www.blockchain.com/btc/strange-transactions) (<https://www.blockchain.com/btc/strange-transactions>)。

我們將在下一章中看到的，這些並不一定是奇怪的交易。它們是包含比普通 P2PKH 更復雜的鎖定腳本的交易。我們將學習如何解碼和理解更復雜的腳本及其支持的應用程式。

# 高級交易和腳本

## 概述

上一章我們介紹了比特幣交易的基本元素，並研究了最常見的交易腳本類型，即P2PKH腳本。在本章中，我們將介紹更高級的腳本以及如何使用它來構建複雜條件下的交易。

首先，我們將介紹 *multisignature* 腳本。接下來，我們將看一下第二個常見的交易腳本 *Pay-to-Script-Hash*，它打開了複雜腳本的世界。然後，我們將研究通過 *timelocks* 為比特幣添加時間維度的新的腳本運算符。最後，我們將看看 *Segregated Witness*，這是對交易結構的架構更改。

## 多重簽名

多重簽名腳本設置了一個條件，N 個公鑰記錄在腳本中，並且需要其中至少 M 個提供簽名才能解鎖資金。這也被稱為 M-of-N 方案，其中 N 是密鑰的總數，M 是驗證所需簽名個數的閾值。例如，一個 2-of-3 的多重簽名是三個公鑰被列為潛在簽名者並且其中至少兩個必須被用來創建簽名，從而創建有效的交易花費資金。

目前，標準的 多重簽名腳本最多隻能列出3個公鑰，這意味著你可以執行從 1-of-1 到 1-of-3 之間的任意組合的多重簽名。本書出版時，列出3個公鑰的限制可能已經解除，因此請檢查 `IsStandard()` 函數以查看網路當前接受的操作。請注意，3鍵的限制僅適用於標準（也稱為“裸”）多重簽名腳本，而不適用於包含在支付到腳本雜湊（P2SH）中的多重簽名腳本。P2SH多重簽名腳本限於15個鍵，最多允許15-of-15的多重簽名。我們將在 [支付到腳本雜湊 Pay-to-Script-Hash \(P2SH\)](#) 中學習P2SH。

M-of-N 多重簽名條件的鎖定腳本設置通常形式如下：

```
M <Public Key 1> <Public Key 2> ... <Public Key N> N CHECKMULTISIG
```

其中 N 是列出的公鑰數量，M 是花費這筆支出所需的簽名個數。

一個 2-of-3 多重簽名條件的鎖定腳本設置如下：

```
2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG
```

上面的鎖定腳本可以被包含簽名和公鑰對兒的解鎖腳本滿足：

```
<Signature B> <Signature C>
```

或者3個公鑰中的任意兩個對應的私鑰生成的簽名的組合

兩個腳本組合起來形成下面的驗證腳本

```
<Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3  
CHECKMULTISIG
```

執行時，只有在解鎖腳本與鎖定腳本設置的條件匹配時，此組合腳本才會評估為TRUE。在這種情況下，條件是解鎖腳本是否具有來自3個公鑰中的兩個對應私鑰的有效簽名。

### CHECKMULTISIG執行中的一個錯誤

在 CHECKMULTISIG 的執行過程中有一個錯誤，需要稍微解決一下。當 CHECKMULTISIG 執行時，它應該消耗堆疊中的 M + N + 2 個項目作為參數。但是，由於該錯誤，CHECKMULTISIG 會彈出額外的值或超出預期的值。

讓我們用前面的驗證示例更詳細地看一下：

```
<Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3
CHECKMULTISIG
```

首先，CHECKMULTISIG彈出頂部元素，它是 N（在本例中為“3”）。然後它彈出 N 個元素，這是可簽名的公鑰。在這個例子中，是公鑰 A, B 和 C。然後，它彈出一個項目，即 M，仲裁數（需要多少個簽名）。這裡 M = 2。此時，CHECKMULTISIG 應該彈出最後的 M 個元素，這是簽名，並查看它們是否有效。然而，不幸的是，實現中的一個錯誤會導致 CHECKMULTISIG 彈出另一個元素（總數為M + 1）。額外的項目在檢查簽名時被忽略，因此它對 CHECKMULTISIG 本身沒有直接影響。但是，必須存在額外的值，因為如果它不存在，當 CHECKMULTISIG 試圖彈出空堆疊時，它將導致堆疊錯誤和腳本失敗（將交易標記為無效）。由於額外的項目被忽略，它可以是任何東西，但通常使用 0。

由於這個bug成為了共識規則的一部分，現在必須永久複製。因此，正確的腳本驗證將如下所示：

```
0 <Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3
CHECKMULTISIG
```

所以，正確的解鎖腳本不是

```
<Signature B> <Signature C>
```

而是：

```
0 <Signature B> <Signature C>
```

從現在起，如果你看到一個 multisig 解鎖腳本，你應該在開始時看到一個額外的 0，其唯一目的是修正意外成為共識規則的錯誤。

## 支付到腳本雜湊 Pay-to-Script-Hash (P2SH)

支付到腳本雜湊 (P2SH) 是2012年推出的一種強大的新型交易，大大簡化了複雜交易腳本的使用。為了解釋對P2SH 的需求，我們來看一個實際的例子。

在 [ch01\_intro\_what\_is\_bitcoin] 中，我們介紹了位於迪拜的電子產品進口商Mohammed。Mohammed公司的公司帳戶廣泛使用比特幣的多重簽名功能。多重簽名腳本是比特幣高級腳本功能的最常見用途之一，並且是一個非常強大的功能。Mohammed的公司為所有客戶付款使用多重簽名腳本，在會計術語中稱為“應收賬款”或AR。使用多重簽名方案時，客戶進行的任何付款都會被鎖定，以至於他們需要至少兩個簽名才能從Mohammed及其合作伙伴或擁有備份密鑰的律師處獲得釋放。像這樣的多重簽名方案提供公司治理控制並防止盜竊，盜用或損失。

最終的腳本很長，看起來是這樣的：

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key> <Partner3
Public Key> <Attorney Public Key> 5 CHECKMULTISIG
```

儘管多重簽名腳本是一個強大的功能，但它們使用起來很麻煩。對於前面的腳本，Mohammed必須在付款之前將此腳本傳達給每位客戶。每個客戶都必須使用特殊的比特幣錢包軟體來創建自定義交易腳本，並且每個客戶都必須瞭解如何使用自定義腳本創建交易。此外，由此產生的交易將比簡單的支付交易大五倍，因為該腳本包含非常長的公鑰。該特大交易的負擔將由客戶以費用的形式承擔。最後，像這樣的大型交易腳本將在每個完整節點的內存中的UTXO集中儲存，直到耗盡內存為止。所有這些問題使得在實踐中使用複雜的鎖定腳本變得困難。

P2SH是為了解決這些實際困難而開發的，使複雜腳本的使用像支付比特幣地址一樣簡單。通過P2SH支付，複雜的鎖定腳本將被其數字指紋（一種加密雜湊）所取代。當試圖花費UTXO的交易在之後出現時，除了解鎖腳本外，它還必須包含與鎖定腳本的指紋相同的腳本。簡而言之，P2SH的意思是“支付給與該雜湊值相匹配的腳本，這個腳本將在稍後花費輸出時使用”。

在P2SH交易中，由雜湊值代替的鎖定腳本稱為 贖回腳本 *redeem script*，因為它在贖回時提供給系統，而不是作為鎖定腳本。[Complex script without P2SH](#) 顯示沒有P2SH的腳本，[Complex script as P2SH](#) 顯示與P2SH編碼的腳本相同。

Table 1. Complex script without P2SH

Locking Script	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 CHECKMULTISIG
Unlocking Script	Sig1 Sig2

Table 2. Complex script as P2SH

Redeem Script	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 CHECKMULTISIG
Locking Script	HASH160 <20-byte hash of redeem script> EQUAL
Unlocking Script	Sig1 Sig2 <redeem script>

如你所見，使用P2SH時，複雜的腳本詳細說明了花費輸出（贖回腳本）的條件，但不是在鎖定腳本中顯示。只有它的雜湊在鎖定腳本中，而且贖回腳本本身稍後會作為解鎖腳本的一部分在花費輸出時呈現。這將費用和複雜性的負擔從交易的發送者轉移到了接收者（消費者）。

讓我們看看Mohammed的公司，複雜的多重簽名腳本，以及由此產生的P2SH腳本。

首先，Mohammed公司為所有客戶的付款使用多重簽名腳本：

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key> <Partner3  
Public Key> <Attorney Public Key> 5 CHECKMULTISIG
```

如果佔位符被實際的公鑰取代（這裡顯示為以04開頭的520位數字），你可以看到該腳本變得非常長：

```
2  
04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395D7E  
EC6984D83F1F50C900A24DD47F569FD4193AF5DE762C58704A2192968D8655D6A935BEAF2CA23E3FB87  
A3495E7AF308EDF08DAC3C1FCBFC2C75B4B0F4D0B1B70CD2423657738C0C2B1D5CE65C97D78D0E34224  
858008E8B49047E63248B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D5737812F393DA  
7D4420D7E1A9162F0279CFC10F1E8E8F3020DECDBC3C0DD389D99779650421D65CBD7149B255382ED7F  
78E946580657EE6FDA162A187543A9D85BAAA93A4AB3A8F044DADA618D087227440645ABE8A35DA8C5B  
73997AD343BE5C2AFD94A5043752580AFA1ECED3C68D446BCAB69AC0BA7DF50D56231BE0AABF1FDEEC7  
8A6A45E394BA29A1EDF518C022DD618DA774D207D137AAB59E0B000EB7ED238F4D800 5  
CHECKMULTISIG
```

整個腳本可以使用20字節的加密雜湊取代，首先應用SHA256雜湊演算法，然後對結果應用RIPEMD160演算法。

我們在命令行上使用 libbitcoin-explorer (bx) 來生成腳本雜湊，如下所示：

```
echo \  
2 \  
[04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395D7
```

```

EEC6984D83F1F50C900A24DD47F569FD4193AF5DE762C587] \
[04A2192968D8655D6A935BEAF2CA23E3FB87A3495E7AF308EDF08DAC3C1FCBFC2C75B4B0F4D0B1B70C
D2423657738C0C2B1D5CE65C97D78D0E34224858008E8B49] \
[047E63248B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D5737812F393DA7D4420D7E1
A9162F0279CFC10F1E8E8F3020DECDBC3C0DD389D9977965] \
[0421D65CBD7149B255382ED7F78E946580657EE6FDA162A187543A9D85BAAA93A4AB3A8F044DADA618
D087227440645ABE8A35DA8C5B73997AD343BE5C2AFD94A5] \
[043752580AFA1ECED3C68D446BCAB69AC0BA7DF50D56231BE0AABF1FDEEC78A6A45E394BA29A1EDF51
8C022DD618DA774D207D137AAB59E0B000EB7ED238F4D800] \
5 CHECKMULTISIG \
| bx script-encode | bx sha256 | bx ripemd160
54c557e07dde5bb6cb791c7a540e0a4796f5e97e

```

上述一系列命令首先將Mohammed的multisig redeem腳本編碼為十六進制的序列化的比特幣腳本。下一個 bx 命令計算其SHA256雜湊值。下一個 bx 命令再次使用RIPEMD160進行雜湊運算，產生最終的腳本雜湊：

Mohammed的贖回腳本的20字節雜湊值是：

```
54c557e07dde5bb6cb791c7a540e0a4796f5e97e
```

P2SH交易使用以下鎖定腳本將輸出鎖定到此雜湊值，而不是之前更長的贖回腳本：

```
HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e EQUAL
```

如你所見，它要短得多。與“支付到5個密鑰的多重簽名腳本”不同，P2SH等價交易是“支付到這個雜湊值的腳本”。向Mohammed公司付款的客戶只需在付款中包含更短的鎖定腳本。當Mohammed和他的合作伙伴想要使用這個UTXO時，他們必須出示原始贖回腳本（用雜湊值鎖定UTXO的那個腳本）和解鎖它的必要簽名，如下所示：

```
<Sig1> <Sig2> <2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG>
```

這兩個腳本組合為兩個階段。首先，根據鎖定腳本檢查贖回腳本以確保雜湊值匹配：

```
<2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG> HASH160 <redeem scriptHash> EQUAL
```

如果贖回腳本雜湊值匹配，解鎖腳本將自行執行，以解鎖贖回腳本：

```
<Sig1> <Sig2> 2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG
```

本章介紹的幾乎所有腳本都只能作為P2SH腳本實現。它們不能直接用在UTXO的鎖定腳本中。

## P2SH 地址

P2SH功能的另一個重要部分是將腳本雜湊編碼為地址的能力，如BIP-13中所定義的那樣。P2SH地址是腳本的20字節雜湊的Base58Check編碼，就像比特幣地址是公鑰的20字節雜湊的Base58Check編碼一樣。P2SH地址使用版本前綴“5”，這導致以“3”開頭的Base58Check編碼地址。

例如，Mohammed的複雜腳本，通過雜湊和Base58Check編碼，生成P2SH地址39RF6JqABiHdYHkfChV6USGMe6Nsr66Gzw。我們可以用 bx 命令來確認

```
echo \
```

```
'54c557e07dde5bb6cb791c7a540e0a4796f5e97e' \
| bx address-encode -v 5
39RF6JqABiHdYHkfChV6USGMe6Nsr66Gzw
```

現在，Mohammed可以給他的客戶提供這個“地址”，他們幾乎可以使用任何比特幣錢包進行簡單付款，就像它是一個比特幣地址一樣。前綴3給他們一個暗示，這是一種特殊的地址類型，對應於腳本而不是公鑰，但是它的作用方式與支付比特幣地址的方式完全相同。

P2SH地址隱藏了所有的複雜性，因此付款人看不到腳本。

## P2SH 的好處

與在鎖定輸出時直接使用複雜腳本相比，P2SH具有以下優點：

- 複雜的腳本在交易輸出中被更短的指紋代替，從而使交易數據更小。
- 腳本可以編碼為地址，發件人和發件人的錢包不需要複雜的工程來實現P2SH。
- P2SH將構建腳本的負擔轉移給收件人，而不是發件人。
- P2SH將長腳本的數據儲存負擔從輸出中（儲存在區塊鏈中的UTXO集中）轉移到輸入中（僅儲存在區塊鏈中）。
- P2SH將長檔案的數據儲存負擔從當前時間（支付）轉移到未來時間（花費時間）。
- P2SH將長腳本的交易費用從發件人轉移到收件人，收件人必須包含很長的兌換腳本才能使用。

## 贖回腳本和驗證

在Bitcoin Core客戶端版本0.9.2之前，Pay-to-Script-Hash通過 `IsStandard()` 函數僅限於標準類型的比特幣交易腳本。這意味著消費交易中提供的贖回腳本只能是標準類型之一：P2PK，P2PKH或multisig。

從Bitcoin Cor客戶端版本0.9.2開始，P2SH交易可以包含任何有效的腳本，使P2SH標準更加靈活，並允許對許多新型和複雜類型的交易進行實驗。

你無法將P2SH放入P2SH贖回腳本中，因為P2SH規範不是遞歸的。另外，技術上可以在贖回腳本中包含 RETURN（參見 [數據記錄輸出 \(RETURN\)](#)），規則中的任何內容都無法阻止你這樣做，但沒有實際意義，因為在驗證期間執行 RETURN 將導致交易被標記為無效。

請注意，因為贖回腳本在你嘗試使用P2SH的輸出之前未呈現給網路，所以如果你使用無效的贖回腳本的雜湊鎖定該輸出，它將被忽略。UTXO將被成功鎖定，但你將無法花費這筆費用，包含贖回腳本的花費交易不會被接受，因為它是無效的腳本。這會產生風險，因為你可以將比特幣鎖定在以後不能使用的P2SH中。網路會接收對應於無效的贖回腳本的鎖定腳本。

Warning

P2SH鎖定腳本包含贖回腳本的雜湊，但不會提供有關贖回腳本本身內容的線索。即使贖回腳本無效，P2SH交易也將被視為有效的並被接受。你可能會意外鎖定比特幣，之後無法花費。

## 數據記錄輸出 (RETURN)

比特幣的分佈式時間戳帳本，區塊鏈（blockchain），具有遠遠超出支付範圍的潛在用途。許多開發人員嘗試使用交易腳本語言，利用系統的安全性和靈活性，應用於數字公證服務，股票證書和智能合約等。將比特幣的腳本語言用於這些目的的早期嘗試包括創建交易輸出，在區塊鏈上記錄數據；例如，記錄檔案的數字指紋，使得任何人都可以通過引用該交易作為該檔案在特定日期存在的證明。

使用比特幣區塊鏈來儲存與比特幣付款無關的數據是一個有爭議的話題。許多開發人員認為這種使用是濫用，並希望阻止它。其他人則認為這是區塊鏈技術強大功能的一個示例，並且希望鼓勵這種實驗。那些反對納入未付款數據的人爭辯說，它會導致“區塊鏈膨脹”，使那些運行完整比特幣節點的人承擔儲存區塊鏈無意承載的數據帶來的成本。此外，此類

交易創建了不能用於支付的，使用20字節的目標比特幣地址的UTXO。由於該地址用於數據，因此它不對應於私鑰，生成的UTXO不會被花費；這是虛假的付款。因此，永遠不會花費的這些交易永遠不會從UTXO集中移除，並導致UTXO資料庫的大小永遠增加或“膨脹”。

在Bitcoin Core客戶端的0.9版本中，通過引入 RETURN 運算符達成了一個折衷方案。RETURN 允許開發人員將80個字節的非付款數據添加到交易輸出中。但是，與使用“假”UTXO不同，RETURN 運算符會創建一個顯式的 可驗證不可消費 的輸出，該輸出不需要儲存在UTXO集合中。RETURN 輸出記錄在區塊鏈中，因此它們消耗硬碟空間並會導致區塊鏈大小的增加，但它們不儲存在UTXO集中，因此不會使UTXOMemory pool膨脹，完整節點頁不用承擔昂貴的內存負擔。

RETURN 腳本看起來如下

```
RETURN <data>
```

數據部分被限制為80字節，並且通常表示雜湊，例如SHA256演算法的輸出（32字節）。許多應用程式在數據前加上前綴以幫助識別應用。例如，<http://proofofexistence.com>[Proof of Existence] 數字公證服務使用8字節前綴 DOCPROOF，十六進制ASCII編碼為 44 4f 43 50 52 4f 4f 46。

請記住，沒有對應於 RETURN 的“解鎖腳本”，用於“花費” RETURN 輸出。RETURN 的全部意義在於你不能把錢鎖定在那個輸出中，因此它不需要被保存在UTXO集合中（潛在可花費的）—— RETURN 是可驗證不可花費的。RETURN 通常是比特幣金額為零的輸出，因為分配給這種輸出的任何比特幣都會永久丟失。如果在交易中引用 RETURN 作為輸入，腳本驗證引擎將暫停驗證腳本的執行並將交易標記為無效。RETURN 的執行本質上導致腳本以 FALSE “返回”並暫停。因此，如果你意外地將 RETURN 輸出引用為交易中的輸入，則該交易無效。

標準交易（符合 IsStandard() 檢查的交易）只能有一個 RETURN 輸出。但是，一個 RETURN 輸出可以與任何其他類型的輸出組合在一個交易中。

Bitcoin Core 0.10中增加了兩個新的命令行選項。選項 `datacarrier` 控制是否中轉和開採 RETURN 交易，預設設置為“1”以允許。選項 `datacarriersize` 接受一個數字參數，指定 RETURN 腳本的最大字節數，缺省為83字節，表示 RETURN 數據最多80個字節，加上 RETURN 操作碼的一個字節，和 PUSHDATA 操作碼的兩個字節。

#### Note

RETURN 最初提出來的最大限制為80個字節，但在發佈功能時限制已減少到40個字節。2015年2月，在比特幣核心版本0.10中，限制提高到80字節。節點可以選擇不中轉或使用 RETURN，或者只中轉和開採包含少於80字節數據的 RETURN 交易。

## 時間鎖 Timelocks

時間鎖是對交易或輸出的限制，只允許在某個時間點之後花費。比特幣從一開始就具有交易級別的時間鎖定功能。它由交易中的 `nLocktime` 欄位實現。2015年末和2016年中推出了兩個新的時間鎖功能，可提供UTXO級別的時間鎖定。這些是 `CHECKLOCKTIMEVERIFY` 和 `CHECKSEQUENCEVERIFY`。

時間鎖定對於推遲日期的交易非常有用，將資金鎖定在未來的日期。更重要的是，時間鎖將比特幣腳本延伸到時間維度，為複雜的多步智能合約打開了大門。

## 交易時間鎖 (nLocktime)

從一開始，比特幣就具有交易級別的時間鎖定功能。交易鎖定時間是交易級別的設置（交易資料結構中的一個欄位），用於定義交易有效的最早時間，並且可以在網路上中轉或添加到區塊鏈。`Locktime`也被稱為 `nLocktime`，來自Bitcoin Core程式碼庫中使用的變數名稱。在大多數交易中它被設置為0以表示立即傳播和執行。如果 `nLocktime` 非零且低於5億，會被解釋為區塊高度，表示交易無效並且不會在指定區塊高度之前中轉或包含在區塊鏈中。如果它超過5億，它會被解釋為Unix紀元時間戳（自1970年1月1日以來的秒數），表示交易在指定時間之前無效。使用 `nLocktime` 指定未來區塊或時間的交易必須由發起的系統持有，只有在它們生效後才傳輸到比特幣網路。如果交易在指定的 `nLocktime` 之前傳輸到網路，交易將被第一個節點認為無效並拒絕，不會被中轉到其他節點。`nLocktime` 的使用等同於推遲日期的紙質支票。

### 交易鎖定時間限制

`nLocktime` 具有侷限性，雖然它允許一些輸出在將來被花費，但不會使這些輸出在那個時間之前不能被花費。我們用下面的例子來解釋一下。

Alice簽署了一筆交易，將其的一個輸出指定到Bob的地址，並將 `nLocktime` 設置為3個月之後。Alice將該交易發送給了Bob。通過這次交易，Alice和Bob知道：

- 在3個月過去之前，Bob不能發起贖回資金的交易。
- Bob可能會在3個月後發起交易。

但是：

- Alice可以創建另一個交易，在沒有鎖定時間的情況下重複使用相同的輸入。因此，Alice可以在3個月過去之前花費相同的UTXO。
- Bob無法保證Alice不這麼做。

瞭解交易 `nLocktime` 的侷限性非常重要。唯一的保證是鮑勃在3個月之前不能贖回，而無法保證鮑勃將獲得資金。要達到這樣的保證，時間限制必須放在UTXO上，併成為鎖定腳本的一部分，而不是交易的一部分。這是通過稱為 檢查鎖定時間驗證 Check Lock Time Verify (CLTV) 的下一種時間形式實現的。

### Check Lock Time Verify (CLTV)

2015年12月，一種新的時間鎖形式作為軟分叉升級引入了比特幣。根據BIP-65中的規範，一種名為 `CHECKLOCKTIMEVERIFY (CLTV)` 的腳本操作符添加到腳本語言中。CLTV 是每個輸出的時間鎖，而不是 使用 `nLocktime` 情況下的每個交易的時間鎖。允許時間鎖的應用更加靈活。

簡而言之，通過在輸出的贖回腳本中添加 CLTV 操作碼，可以限制輸出只能在指定的時間過後才能使用。

Tip	<code>nLocktime</code> 是交易級別的時間鎖，CLTV 是基於輸出的時間鎖。
-----	--

CLTV 並沒有取代 `nLocktime`，而是限制特定UTXO，以使它們只能在 `nLocktime` 設置為更大或相等的值的未來交易中使用。

CLTV 操作碼將一個參數作為輸入，該參數以與 `nLocktime` (區塊高度或Unix紀元時間) 相同的格式表示。如 `VERIFY` 後綴所示，CLTV 是在結果為 `FALSE` 時停止執行腳本的操作碼。如果結果為 `TRUE`，則繼續執行。

為了用 CLTV 鎖定輸出，可以在創建這筆輸出的交易中，將其插入到輸出的贖回腳本中。例如，如果Alice正在向Bob的地址支付，輸出通常會包含如下所示的P2PKH腳本：

```
DUP HASH160 <Bob's Public Key Hash> EQUALVERIFY CHECKSIG
```

為了將其鎖定一段時間，比如從現在開始3個月，這筆交易將帶有如下的贖回腳本：

```
<now + 3 months> CHECKLOCKTIMEVERIFY DROP DUP HASH160 <Bob's Public Key Hash>
EQUALVERIFY CHECKSIG
```

其中 `<now {plus} 3 months>` 是從這筆交易被開採後3個月的區塊高度或者時間戳估計，當前區塊高度 + 12,960 (區塊) 或者 當前Unix時間戳 + 7,760,000 (秒)。現在，不要在意 `CHECKLOCKTIMEVERIFY` 之後的 `DROP` 操作符，我們之後會解釋。

當Bob嘗試花費這個UTXO時，構建一個以UTXO作為輸入的交易，在輸入的解鎖腳本中使用他的簽名和公鑰，並將交易的 `nLocktime` 設置為等於或大於 `CHECKLOCKTIMEVERIFY` 中Alice設置的 timelock，然後在比特幣網路上廣播交易。

Bob的交易被進行如下的評估，如果Alice設置的 CHECKLOCKTIMEVERIFY 的參數小於或等於消費交易的 nLocktime，則腳本執行繼續（如同執行 "no operation" 或 NOP操作碼一樣）。否則，腳本執行會停止，並且交易被視為無效。

更準確地說，CHECKLOCKTIMEVERIFY 失敗並暫停執行，標記交易無效，如果達成以下條件（來源：BIP-65）：

1. 堆疊為空；或者
2. 堆疊頂元素小於0；或者
3. 堆疊頂元素鎖定時間的類型（區塊高度或時間戳）與 nLocktime 欄位不同；或者
4. 堆疊頂元素大於交易的 nLocktime 欄位；或者
5. 輸入的 nSequence 欄位為 0xffffffff。

#### Note

CLTV 和 nLocktime 使用相同的格式來描述時間鎖，可以是區塊高度，也可以是自 Unix 紀元以來的秒數。重要的是，當一起使用時，nLocktime 的格式必須與輸出中的 CLTV 的格式匹配——它們都必須表示區塊高度，或以秒為單位的時間。

執行後，如果滿足 CLTV，則其前面的時間參數將保留為堆疊頂元素，需要使用 DROP 將其刪除，以正確執行後續腳本操作碼。出於這個原因，你經常會在腳本中看到 CHECKLOCKTIMEVERIFY 和 DROP。

通過將 nLocktime 與 CLTV 結合使用，[交易鎖定時間限制](#) 中描述的場景會發生變化。Alice 不能再花費這筆資金了（因為它被 Bob 的密鑰鎖定），Bob 在 3 個月的鎖定時間到期之前也不能花費。

通過將時間鎖功能直接引入腳本語言，CLTV 允許我們開發一些非常有趣的複雜腳本。

標準的定義參見 [BIP-65 \(CHECKLOCKTIMEVERIFY\)](#)。

## 相對時間鎖

nLocktime 和 CLTV 都是 純絕對時間鎖 *absolute timelocks*，表示一個絕對的時間點。接下來我們要研究的兩個時間鎖功能是 相對時間鎖 *relative timelocks*，它們指定從輸出在區塊鏈中被確認時開始的一段時間，作為花費輸出的條件。

相對時間鎖是有用的，它們允許兩個或多個相互依賴的交易組成的交易鏈進行脫鏈處理，對一個依賴於前一個交易確認後一段時間的交易施加時間限制。換句話說，直到 UTXO 被記錄在區塊鏈上時，時鐘才會開始計數。這個功能在雙向狀態通道（bidirectional state channels）和閃電網路（Lightning Networks）中特別有用，我們將在 [\[state\\_channels\]](#) 中看到。

相對時間鎖與絕對時間鎖一樣，都是通過交易級功能和腳本級操作碼實現的。交易級別的相對時間鎖實現為 nSequence（每個交易輸入中設置的欄位）值的共識規則。腳本級別的相對時間鎖使用 CHECKSEQUENCEVERIFY (CSV) 操作碼實現。

相對時間鎖是根據 [BIP-68, Relative lock-time using consensus-enforced sequence numbers](#) 和 [BIP-112, CHECKSEQUENCEVERIFY](#) 中的規範實現的。

BIP-68 和 BIP-112 於 2016 年 5 月作為共識規則的軟分叉升級啟用。

## nSequence相對時間鎖

通過設置 nSequence 欄位，可以在交易的每個輸入上設置相對時間鎖。

### nSequence的原始含義

nSequence 欄位的本意在於（但從未正確實施）允許修改 mempool 中的交易。在該用途中，包含 nSequence 值低於  $2^{32} - 1$  (0xFFFFFFFF) 的輸入的交易指示尚未“完成”的交易。這樣的交易將保留在 mempool 中，直到它被另一個花費相同的輸入但具有更高的 nSequence 值的交易替代。一旦接收到輸入的 nSequence 值為 0xFFFFFFFF 的交易，它將被視為“完成的”並被開採。

nSequence 的原始含義從未正確實現，nSequence 的值通常在不使用時間鎖定的交易中被設置為0xFFFFFFFF。對於具有 nLocktime 或 CHECKLOCKTIMEVERIFY 的交易，必須將 nSequence 值設置為小於  $2^{31}$ ，才能使時間保護具有效果，如下所述。

#### nSequence 作為共識執行的相對時間鎖

自BIP-68啟用以來，新的共識規則適用於包含 nSequence 值小於  $2^{31}$  的輸入的任何交易。從編程的角度來說，這意味著如果最高有效位（第1<<31位）未設置為1，則表示“相對鎖定時間”。否則（1<<31設置為1），nSequence 的值被保留用於其他用途，例如啟用 CHECKLOCKTIMEVERIFY，nLocktime，Opt-In-Replace-By-Fee以及其他未來的開發。

具有小於  $2^{31}$  的 nSequence 值的交易輸入被解釋為具有相對時間鎖。這種交易只有在輸入已經過相對時間鎖表示的時間後才有效。例如，具有 nSequence 為30個區塊的相對時間鎖的一個輸入的交易，僅在從輸入中引用的UTXO被開採的時間起，至少經過30個區塊時才有效。由於 nSequence 是每個輸入的欄位，交易可能包含任意數量的時間鎖定輸入，所有這些輸入都必須滿足時間要求交易才有效。一個交易可以同時包括時間鎖定的輸入（nSequence  $< 2^{31}$ ）和沒有時間鎖定的輸入（nSequence  $\geq 2^{31}$ ）。

nSequence 值以區塊或秒為單位，但與我們在 nLocktime 中使用的格式略有不同。類型標誌（type-flag）用於區分表示區塊數還是表示時間（以秒為單位）。類型標誌被設置在第23個最低有效位（即值 $1 \ll 22$ ）中。如果類型標誌為1，則 nSequence 值被解釋為512秒的倍數。如果類型標誌為0，則 nSequence 值將被解釋為區塊數。

當將 nSequence 解釋為相對時間鎖時，僅考慮16個最低有效位。一旦標誌位（比特32和23）檢測完成，通常使用 nSequence 的16位掩碼（例如，nSequence & 0x0000FFFF）。

[BIP-68 definition of nSequence encoding \(Source: BIP-68\)](#) 展示了 nSequence 的二進制結構，由 BIP-68 定義。

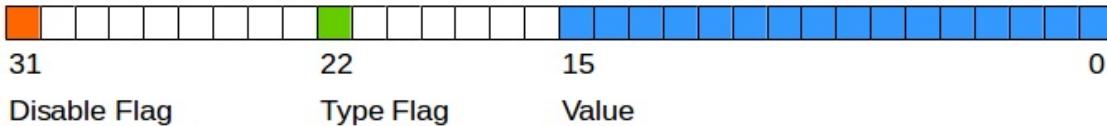


Figure 1. BIP-68 definition of nSequence encoding (Source: BIP-68)

基於 nSequence 值共識執行的相對時間鎖在BIP-68中定義。

這個標準定義在 [BIP-68, Relative lock-time using consensus-enforced sequence numbers](#) 中。

#### 使用CSV的相對時間鎖

與CLTV和 nLocktime 一樣，有一個在腳本中使用 nSequence 值作為相對時間鎖的腳本操作碼。該操作碼是 CHECKSEQUENCEVERIFY，通常簡稱為 CSV。

在UTXO的贖回腳本中執行時，CSV 操作碼僅允許輸入的 nSequence 值大於或等於 CSV 參數的交易。從本質上講，這限制了UTXO直到相對於UTXO開採的時間已經過去了一定數量的區塊或秒之後才能被花費。

與CLTV一樣，CSV 中的值必須與相應的 nSequence 值中的格式匹配。如果 CSV 指定的是區塊數量，nSequence 也必須是區塊數量。如果 CSV 指定的是秒，那麼 nSequence 也必須是秒。

當幾個（鏈接的）交易被創建和簽名，但保持“脫鏈”狀態不會傳播時，使用 CSV 的相對時間鎖特別有用。直到父交易被傳播，開採和沉澱到相對時間鎖定中指定的時間後，才能使用子交易。可以在 [\[state\\_channels\]](#) 和 [\[lightning\\_network\]](#) 中看到這個用例的應用。

CSV 在 [BIP-112, CHECKSEQUENCEVERIFY](#) 中詳細定義。

#### 過去中位時間 Median-Time-Past

作為啟用相對時間鎖的一部分，時間鎖（絕對和相對）的“時間”計算方式有所變化。在比特幣中，實際時間（wall time）和共識時間（consensus time）之間存在著微妙但重要的差異。比特幣是一個去中心化的網路，這意味著每個參與者都有自己的時間視角。網路上的事件並不是每時每刻都在發生。必須在每個節點的角度考慮網路延遲。最終，所有內容都會同步，創建一個公共的賬本。與過去一樣，比特幣每隔10分鐘就會對賬本的狀態達成共識。

區塊頭中的時間戳是由礦工設置的。共識規則留有一定的餘地來解決分散的節點之間的時鐘精度差異。然而，這帶來了一種不幸的激勵，促使礦工在一個區塊內對時間撒謊，以便通過納入尚未成熟的時間鎖定交易來收取額外的費用。有關更多訊息，請參見下面的部分。

為了消除對撒謊的激勵，並加強時間鎖的安全性，BIP-113與關於相對時間鎖的BIP同時提出並被啟用，它定義了一個稱為過去中位時間 *Median-Time-Past* 的新的時間測量方法。

*Median-Time-Past* 過去11個區塊的時間戳並的中位數。中位時間成為共識時間，並用於所有時間鎖的計算。通過取過去大約兩個小時的中點，任何一個區塊的時間戳影響都會減小。通過結合11個區塊，沒有一個礦工可以為了獲得尚未成熟的時間鎖定交易費用而影響時間戳。

*Median-Time-Past* 改變了 nLocktime，CLTV，nSequence 和 CSV 的時間計算實現。由*Median-Time-Past* 計算的共識時間總是比實際時間晚大約一個小時。如果你創建時間鎖定交易，應該在估計 nLocktime，nSequence，CLTV 和 CSV 中編碼的時間時考慮這一點。

*Median-Time-Past* 在 [BIP-113](#) 中被定義。

## 對抗費用狙擊的時間鎖防禦 Timelock Defense Against Fee Sniping

費用狙擊（Fee-sniping）是一種理論上的攻擊場景，表示試圖改寫過去的區塊的礦工“狙擊”未來區塊中更高費用的交易，來最大化盈利的方法。

例如，假設現在的最高區塊是區塊 # 100,000。一些礦工嘗試重新開採區塊 #100,000，而不是嘗試開採區塊 # 100,001 來增長區塊鏈。這些礦工可以選擇在他們的候選區塊 # 100,000 中包含任何有效的交易（尚未開採）。他們不必通過相同的交易來重新開採區塊。事實上，他們有動力選擇最有利可圖的（每KB最高費用）交易並包含在他們的區塊中。它們可以包括任何在“舊”區塊 # 100,000 中的交易，也可以包含來自當前 mempool 的任何交易。本質上，當他們重新創建區塊 # 100,000 時，他們可以選擇將交易從“現在”轉移到重寫的“過去”。

今天，這種攻擊不是很有利可圖，因為區塊獎勵遠高於每個區塊的總費用。但是在將來，交易費用將成為獎勵的一大部分（甚至獎勵的全部）。那時候，這種情況就不可避免了。

在Bitcoin Core創建交易時，為了防止“費用狙擊”，會預設使用 nLocktime 將其限制為“下一個區塊”。在我們的場景中，Bitcoin Core 會在其創建的任何交易中將 nLocktime 設置為 100,001。正常情況下，這個 nLocktime 不起作用——無論如何，交易只能包含在 # 100001 區塊中；這是下一個區塊。

但是，在區塊鏈分叉攻擊下，礦工們將無法從 mempool 中獲取高額交易，因為所有這些交易都會被鎖定到 # 100,001 區塊。他們只能使用當時有效的交易重新計算 # 100,000，實質上不會獲得新的費用。

為此，Bitcoin Core 將所有新交易的 nLocktime 設置為 <當前區塊 # + 1>，並將所有輸入的 nSequence 設置為 0xFFFFFFFF 以啟用 nLocktime。

## 流程控制腳本（條件語句）

比特幣腳本的一個更強大的功能是流程控制，也稱為條件語句。你可能熟悉多種語言中的 IF...THEN...ELSE 流程控制。比特幣的條件語句看起來有點不同，但基本構造是相同的。

比特幣條件操作碼允許我們構建一種有兩種解鎖方式的贖回腳本，取決於條件語句的結果是 TRUE 還是 FALSE。例如，如果 x 為 TRUE，則贖回腳本為 A，否則（ELSE），贖回腳本為 B。

此外，比特幣條件表達式可以無限“嵌套”，條件語句可以包含另一個條件語句。比特幣流程控制腳本可用於構建具有數百或甚至數千個可能的執行路徑的複雜腳本。嵌套沒有限制，但共識規則會對腳本的最大大小（以字節為單位）施加限制。

比特幣使用 IF, ELSE, ENDIF 和 NOTIF 操作碼實現流程控制。此外，條件表達式可以包含布林運算符，例如 BOOLAND, BOOLOR，和 NOT。

乍一看，你可能會對比特幣的流程控制腳本感到困惑。這是因為比特幣腳本是一種堆疊語言。1 AND 1 逆序表示為 1 1 ADD。

在大多數傳統（過程式）程式語言中，流程控制看起來是這樣的：

Pseudocode of flow control in most programming languages

```
if (condition):
    code to run when condition is true
else:
    code to run when condition is false
code to run in either case
```

在類似比特幣腳本的基於堆疊的語音中，邏輯條件放在 IF 之前，使其看起來是逆序的：

Bitcoin Script flow control

```
condition
IF
    code to run when condition is true
ELSE
    code to run when condition is false
ENDIF
code to run in either case
```

在閱讀比特幣腳本時，記住條件判斷是在 IF 操作碼之前的。

## 條件語句的 VERIFY 操作碼

比特幣腳本中另外一種條件形式是以 VERIFY 結尾的任何操作碼。VERIFY 後綴表示如果所評估的條件不是 TRUE，腳本將立即終止執行，並且交易被視為無效。

與 IF 語句提供不同的執行路徑不同，VERIFY 後綴用作 守護語句 *guard clause*, 只有滿足前面的條件時繼續執行。

例如，以下腳本需要Bob的簽名和產生特定雜湊的原象（pre-image）（密鑰）。必須滿足這兩個條件才能解鎖：

A redeem script with an EQUALVERIFY guard clause.

```
HASH160 <expected hash> EQUALVERIFY <Bob's Pubkey> CHECKSIG
```

為了贖回這筆資金, Bob 必須創建提供原象（pre-image）和簽名的解鎖腳本：

An unlocking script to satisfy the above redeem script

```
<Bob's Sig> <hash pre-image>
```

在不提供原象的情況下，Bob無法執行到檢查其簽名的腳本部分。

這個腳本可以用 IF 語句寫成:

A redeem script with an IF guard clause

```
HASH160 <expected hash> EQUAL
IF
```

```
<Bob's Pubkey> CHECKSIG
ENDIF
```

Bob的解鎖腳本是相同的：

An unlocking script to satisfy the above redeem script

```
<Bob's Sig> <hash pre-image>
```

帶 IF 的腳本與使用 VERIFY 後綴的操作碼的功能相同；他們都作為守護語句運行。但是，VERIFY 構造更高效，使用兩個較少的操作碼。

那什麼時候使用 VERIFY，什麼時候使用 IF 呢？如果我們只是附加一個先決條件（guard clause），那麼 VERIFY 更好。但是如果有多個執行路徑（流程控制），那麼需要使用 IF...ELSE 流程控制語句。

Tip

諸如 EQUAL 之類的操作碼會將結果（TRUE 或 FALSE）推到堆疊上，用於後續操作碼的判斷。相反，操作碼 EQUALVERIFY 不會在堆疊中留下任何內容。以 VERIFY 結尾的操作碼都不會將結果留在堆疊上。

## 在腳本中使用流程控制

比特幣腳本中，流程控制的一個常見用途是構建贖回腳本，提供多個執行路徑，每種贖回方式都可以贖回UTXO。

看一個簡單的例子，有兩個簽名者，Alice和Bob，任何一個都可以兌換。使用multisig時，這將表示為 1-of-2 的多重簽名腳本。為了演示，我們使用 IF 語句做同樣的事情：

```
IF
<Alice's Pubkey> CHECKSIG
ELSE
<Bob's Pubkey> CHECKSIG
ENDIF
```

看到這個贖回腳本，你可能會想：“條件在哪裡？在 IF 語句之前沒有任何東西啊！”

條件不是贖回腳本的一部分。而是在解鎖腳本中提供，從而允許 Alice 和 Bob “選擇” 他們想要的執行路徑。

Alice使用以下解鎖腳本進行贖回：

```
<Alice's Sig> 1
```

最後的 1 作為條件（TRUE），使 IF 語句執行Alice簽名的第一個贖回路徑。

如果Bob要贖回，他必須通過給 IF 語句提供一個 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置了 0，導致 IF 語句執行第二個（ELSE）腳本，從而需要Bob的簽名。

由於 IF 語句可以嵌套，我們可以創建執行路徑的“迷宮”。解鎖腳本可以提供一個選擇執行路徑實際執行的“映射”：

```
IF
script A
ELSE
```

```

IF
  script B
ELSE
  script C
ENDIF
ENDIF

```

在這種情況下，有三個執行路徑（腳本A，腳本B 和 腳本C）。解鎖腳本以 TRUE 或 FALSE 值的順序提供路徑。例如，要選擇路徑 腳本B，解鎖腳本必須以 10 ( TRUE , FALSE ) 結尾。這些值將被壓入堆疊，以便第二個值 ( FALSE ) 作為堆疊頂部。外層的 IF 語句彈出 FALSE 並執行第一個 ELSE 語句。然後 TRUE 移動到堆疊頂，並由內部的（嵌套的）IF 判斷，從而選擇 B 執行路徑。

使用這種構造，我們可以用數十或數百個執行路徑構建回腳本，每個腳本都提供了一種不同的方式來贖回UTXO。為了花費UTXO，我們構建一個解鎖腳本，通過在每個流程控制點的堆疊上放置相應的 TRUE 和 FALSE 來選擇執行路徑。

## 複雜腳本示例

在本節中，我們將本章中的許多概念結合到一個示例中。

我們的例子使用了迪拜公司所有者Mohammed的故事，該公司經營進出口業務。

在這個例子中，Mohammed希望建立一個規則靈活的公司資本賬戶。他創建的方案需要根據時間鎖進行不同級別的授權。多重簽名方案的參與者是Mohammed，他的兩個合夥人Saeed和Zaira，以及他們公司的律師Abdul。三位合夥人根據多數規則作出決定，即三位合夥人中的兩位必須同意。但是，如果他們的密鑰出現問題，他們希望他們的律師能夠用三個合夥人中一個的簽名來恢復資金。最後，如果所有合作伙伴都暫時沒空或無法工作，他們希望律師能夠直接管理帳戶。

以下是Mohammed設計的實現此目標的贖回腳本：

Variable Multi-Signature with Timelock

```

01 IF
02   IF
03     2
04   ELSE
05     <30 days> CHECKSEQUENCEVERIFY DROP
06     <Abdul the Lawyer's Pubkey> CHECKSIGVERIFY
07     1
08   ENDIF
09   <Mohammed's Pubkey> <Saeed's Pubkey> <Zaira's Pubkey> 3 CHECKMULTISIG
10 ELSE
11   <90 days> CHECKSEQUENCEVERIFY DROP
12   <Abdul the Lawyer's Pubkey> CHECKSIG
13 ENDIF

```

Mohammed的腳本使用嵌套的 IF...ELSE 流程控制語句實現了三個執行路徑。

在第一個執行路徑中，這個腳本作為一個簡單的 2-of-3 多重簽名。此執行路徑由第3行和第9行組成。第3行將multisig的法定數設置為 2 (2/3)。這個執行路徑可以通過在解鎖腳本的末尾加上 TRUE TRUE 來選擇：

Unlocking script for the first execution path (2-of-3 multisig)

```
0 <Mohammed's Sig> <Zaira's Sig> TRUE TRUE
```

Tip

這個解鎖腳本開頭的 0 是因為 CHECKMULTISIG 中的一個錯誤，它會從堆疊中彈出一個額外的值。額外的值被 CHECKMULTISIG 忽略，但它必須存在。正如 [CHECKMULTISIG執行中的一個錯誤](#) 中所述，推入 0 (通常) 是該bug的解決方法。

第二個執行路徑只能在創建 UTXO 30天后才能使用。到時，它需要律師Abdul和三個合夥人之一的前面（1-of-3 的多重簽名）。這通過第7行來實現，該行將multisig的法定數設置為 1。要選擇此執行路徑，解鎖腳本將以 FALSE TRUE 結束：

Unlocking script for the second execution path (Lawyer + 1-of-3)

```
0 <Saeed's Sig> <Abdul's Sig> FALSE TRUE
```

Tip

為什麼 FALSE TRUE？因為這兩個值被推送到堆疊上，所以首先推入 FALSE，然後再推入 TRUE。因此 TRUE 被第一個 IF 操作碼彈出。

最後，第三個執行路徑允許律師Abdul單獨花費資金，但只能在90天后。要選擇此執行路徑，解鎖腳本必須以 FALSE 結尾：

Unlocking script for the third execution path (Lawyer only)

```
<Abdul's Sig> FALSE
```

嘗試在紙上運行腳本以查看它在堆疊上的行為。

閱讀本示例時需要考慮幾件事情。看看你能否找到答案：

- 為什麼律師無法通過在解鎖腳本上選擇 FALSE 執行第三條路徑來隨時贖回？
- 在UTXO開採之後的5天，35天和105天，分別可以使用的執行路徑數量？
- 如果律師失去了密鑰，資金是否會流失？如果91天過去了，你的答案會改變嗎？
- 合夥人如何每隔29或89天“重置”時鐘以防止律師獲得資金？
- 為什麼這個腳本中的一些 CHECKSIG 操作碼有 VERIFY 後綴，而其他的則沒有？

## 隔離見證 Segregated Witness

隔離見證 Segregated Witness (segwit) 是比特幣共識規則和網路協議的升級，由BIP-9提出並作為軟分叉實施，於2017年8月1日在比特幣主網啟用。

在密碼學中，術語“見證”用於描述密碼謎題的解決方案。對比特幣來說，“見證”能夠滿足放在未支付交易輸出（UTXO）上的加密條件。

在比特幣的情況下，數字簽名是“見證”的一種類型，但更寬泛地來說，“見證”是能夠滿足UTXO所設置的條件，解鎖並花費UTXO的任何解決方案。術語“見證”是“解鎖腳本”或“scriptSig”的更一般的術語。

在segwit引入之前，交易中的每個輸入之後都是解鎖它的見證數據。見證數據作為每個輸入的一部分嵌入在交易中。術語 *segregated\_witness* 或簡稱 *segwit* 僅僅意味著將特定輸出的簽名或解鎖腳本分離。考慮最簡單的形式，“單獨的 scriptSig”或“單獨簽名”。

因此，隔離見證是比特幣的體系結構變化，旨在將見證數據從交易的 scriptSig（解鎖腳本）欄位移動到伴隨交易的單獨的 *witness* 資料結構中。客戶端可以選擇是否附帶見證數據請求交易。

在本節中，我們將看看隔離見證的好處，描述部署和實施此架構的機制，並演示如何在交易和地址中使用隔離見證。

隔離見證由以下BIP定義：

### BIP-141

Segregated Witness 的主要定義。

### BIP-143

版本0見證程序的交易簽名驗證

**BIP-144**

對等服務 - 新的網路訊息和序列化格式

**BIP-145**

隔離見證的getblocktemplate（用於挖礦）更新

**BIP-173**

原生 v0-16 見證輸出的 Base32 地址格式

## 為什麼要隔離見證？

隔離見證是一種體系結構變化，它對比特幣的可擴展性，安全性，經濟效益和性能有以下影響：

### 交易可鍛性 *Transaction Malleability*

通過將見證數據移動到交易外部，用作標識符的交易雜湊將不再包含見證數據。由於見證數據是交易中唯一可以由第三方修改的部分（請參閱 [交易標識符 Transaction identifiers](#)），因此去除它也消除了交易可鍛性攻擊的機會。使用隔離見證，交易雜湊變得不可能由交易的創建者以外的任何人改變，這極大地改進了許多其他協議的實施，這些協議依賴於先進的比特幣交易建設，例如支付通道，鏈式交易和閃電網路。

### 腳本版本控制 *Script Versioning*

通過隔離見證腳本的進入，每個鎖定腳本前面都有一個 *script\_version* 數字，類似於交易和區塊的版本號。腳本版本號的添加允許腳本語言以向後兼容的方式升級（即使用軟叉升級）來引入新的腳本操作符，語法或語義。以無中斷方式升級腳本語言的能力將大大加速比特幣的創新速度。

### 網路和儲存的可擴展性 *Network and Storage Scaling*

見證數據通常是交易總規模的主要貢獻者。更復雜的腳本，比如用於multisig或支付通道的腳本非常龐大。在某些情況下，這些腳本佔交易數據的大部分（超過75%）。將見證數據移到交易之外，提高了比特幣的可擴展性。節點可以在驗證簽名後裁剪見證數據，或者在進行簡單付款驗證時完全忽略見證數據。見證數據不需要傳輸到所有節點，也不需要被所有節點儲存在硬碟上。

### 簽名驗證優化 *Signature Verification Optimization*

隔離見證升級了簽名方法（*CHECKSIG*，*CHECKMULTISIG* 等）以降低演算法的計算複雜度。在隔離之前，用於生成簽名的演算法需要一些與交易大小成正比的雜湊操作。數據雜湊的計算複雜度相對於簽名操作是 $O(n^2)$ ，在驗證簽名的所有節點上引入了大量的計算負擔。使用segwit時，演算法將複雜度降低到 $O(n)$ 。

### 離線簽名改進 *Offline Signing Improvement*

隔離見證簽名包含了簽名的雜湊中每個輸入引用的值（金額）。以前，離線簽名設備（如硬體錢包）必須在簽署交易之前驗證每個輸入的數量。這通常是通過流式傳輸大量關於以引用為輸入的交易的數據來完成的。由於金額現在是已簽名的雜湊的一部分，因此離線設備不需要先前的事務。如果金額不匹配（由被入侵的系統篡改），簽名將無效。

## 隔離見證如何工作

隔離見證看起來改變了交易如何構建，是交易層面的特性，但事實並非如此。相反，隔離見證是對如何花費單個UTXO的改變，因此是每個輸出層面的特性。

交易可以花費隔離見證的輸出或傳統（內聯見證）的輸出，或同時花費兩者。因此，將交易稱為“隔離見證交易”沒有什麼意義。我們應該將具體的交易輸出稱為“隔離見證輸出”。

當交易花費UTXO時，它必須提供見證。在傳統的UTXO中，鎖定腳本要求在花費UTXO的交易的輸入部分提供在線的見證數據。然而，隔離見證UTXO指定了一個鎖定腳本，它可以被輸入之外的（隔離的）見證數據滿足。

## 軟分叉 (向後兼容)

隔離見證是輸出和交易架構方式的重大變化。這種改變通常需要每個比特幣節點和錢包同時改變以升級共識規則——所謂的硬分叉。然而，隔離見證的引入具有較少的破壞性變化，是向後兼容的，被稱為軟分叉。這種類型的升級允許非升級軟體忽略更改並繼續運行而不會中斷。

隔離見證構建的輸出，使不能識別“見證”的舊系統仍然可以驗證它們。對於舊的錢包或節點，隔離見證輸出看起來像是任何人都可以花費的輸出。這樣的輸出可以用一個空的簽名來花費，交易內部沒有簽名（隔離的）並不會使交易失效。然而，較新的錢包和挖礦節點會看到隔離見證輸出，並期望在交易的見證數據中找到有效的見證。

## 隔離見證輸出和交易示例

讓我們來看一些示例交易，看它們將如何隨著隔離見證改變。首先看一下如何使用隔離見證程序來轉換Pay-to-Public-Key-Hash (P2PKH)。然後，看一下Pay-to-Script-Hash (P2SH) 腳本的隔離見證等價物。最後，我們將看看如何將之前的隔離見證程序嵌入到P2SH腳本中。

### Pay-to-Witness-Public-Key-Hash (P2WPKH)

在 [cup\_of\_coffee] 中，Alice創建了一筆交易，向Bob購買一杯咖啡。該交易創建了一個值為0.015 BTC的P2PKH輸出，該輸出可由Bob使用。輸出的腳本如下所示：

Example P2PKH output script

```
DUP HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7 EQUALVERIFY CHECKSIG
```

使用隔離見證，Alice將創建一個 Pay-to-Witness-Public-Key-Hash (P2WPKH) 腳本，看起來如下：

Example P2WPKH output script

```
0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
```

如你所見，隔離見證輸出的鎖定腳本比傳統輸出簡單得多。它由推送到腳本計算堆疊的兩個值組成。對於老的（不支持隔離見證的 *nonsegwit-aware*）比特幣客戶端來說，這看起來像是任何人都可以花費的輸出，並且不需要簽名（或者更確切地說，可以使用空簽名）。對於一個更新的，支持segwit的客戶端，第一個數字（0）被解釋為版本號（*witness version*），第二部分（20字節）相當於被稱為 *witness program* 的鎖定腳本。20字節的見證程序就是公鑰的雜湊，就像在P2PKH腳本中一樣。

現在，我們來看看Bob用來花費這個輸出的相應的交易。對於原始腳本（nonsegwit），Bob的交易必須在交易輸入中包含簽名

Decoded transaction showing a P2PKH output being spent with a signature

```
[...]
"Vin" : [
  "txid": "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286c345c2f2",
  "vout": 0,
  "scriptSig": "<Bob's scriptSig>",
]
[...]
```

但是，要花費隔離見證的輸出，交易在那個輸入上沒有簽名。相反，Bob的交易包含一個空的 scriptSig 和一個在交易之外的隔離見證。

Decoded transaction showing a P2WPKH output being spent with separate witness data

```
[...]
"Vin" : [...]
```

```

"txid": "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286c345c2f2",
"vout": 0,
    "scriptSig": "",
]
[...]
"witness": "<Bob's witness data>"
[...]

```

### 錢包的P2WPKH構建

注意到P2WPKH只能由收款人創建，而不能由付款人從已知公鑰，P2PKH腳本或地址轉換，這一點非常重要。付款人無法知道收款人的錢包是否有能力構建隔離見證交易並花費P2WPKH輸出。

此外，P2WPKH輸出必須由 壓縮公鑰的雜湊構造。未壓縮的公鑰在segwit中是非標準的，並且可能會被未來的軟分支明確禁用。如果P2WPKH中使用未壓縮的公鑰雜湊，則它可能是不可靠的，你可能會失去資金。P2WPKH輸出應該由收款人的錢包通過從其私鑰匯出的壓縮公鑰來創建。

Warning	P2WPKH應由收款人通過將壓縮公鑰轉換為P2WPKH雜湊來構造。你不應將P2PKH腳本，比特幣地址或未壓縮的公鑰轉換為P2WPKH見證腳本。
---------	---

### Pay-to-Witness-Script-Hash (P2WSH)

第二種見證程序對應於支付到腳本雜湊（P2SH）的腳本。我們在 [支付到腳本雜湊 Pay-to-Script-Hash \(P2SH\)](#) 中看到過這種類型的腳本。在這個例子中，Mohammed的公司使用P2SH來表示多重簽名腳本。對Mohammed的公司的付款用這種鎖定腳本編碼：

Example P2SH output script

```
HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e EQUAL
```

該P2SH腳本引用了 贖回腳本 *redeem\_script* 的雜湊，該腳本定義了花費資金的 2-of-3 多重簽名要求。為了使用這種輸出，Mohammed的公司將在交易輸入中提供贖回腳本（其雜湊與P2SH輸出中的腳本雜湊匹配）以及滿足贖回腳本所需的簽名：

Decoded transaction showing a P2SH output being spent

```

[...]
"Vin" : [
"txid": "abcdef12345...",
"vout": 0,
    "scriptSig": "<SigA> <SigB> <2 PubA PubB PubC PubD PubE 5 CHECKMULTISIG>",
]

```

現在，讓我們看看整個示例如何升級到segwit。如果Mohammed的客戶使用兼容segwit的錢包，他們將創建一個付款，包含一個Pay-to-Witness-Script-Hash (P2WSH) 輸出，看起來像這樣：

Example P2WSH output script

```
0 a9b7b38d972cabcc7961dbfbcb841ad4508d133c47ba87457b4a0e8aae86dbb89
```

同樣，與P2WPKH的例子一樣，你可以看到隔離見證等效腳本更簡單，並且省略了你在P2SH腳本中看到的各種腳本操作數。相反，隔離見證程序由推送到堆疊的兩個值組成：見證版本（0）和贖回腳本的32字節SHA256雜湊。

**Tip** 雖然P2SH使用20字節 RIPEMD160( SHA256(script) ) 雜湊，P2WSH見證程序使用32字節 SHA256(script)雜湊。這種雜湊演算法選擇上的差異是故意的，用於區分兩種類型的見證程序（P2WPKH 和P2WSH）之間的雜湊長度，併為P2WSH提供更強的安全性（P2WSH中的128位安全性，對比P2SH中的80位安全性）。

Mohammed的公司可以通過提供正確的贖回腳本和足夠的簽名來滿足它，用於花費P2WSH的輸出。作為見證數據的一部分，贖回腳本和簽名都將作為消費交易的一部分進行隔離。在交易輸入中，Mohammed的錢包會放置一個空的 scriptSig：

Decoded transaction showing a P2WSH output being spent with separate witness data

```
[...]
"Vin": [
  "txid": "abcdef12345...",
  "vout": 0,
  "scriptSig": "",
]
[...]
"witness": "<SigA> <SigB> <2 PubA PubB PubC PubD PubE 5 CHECKMULTISIG>"
[...]
```

### P2WPKH 和 P2WSH 的區別

在前兩節中，我們演示了兩種類型的見證程序：[Pay-to-Witness-Public-Key-Hash \(P2WPKH\)](#) 和 [Pay-to-Witness-Script-Hash \(P2WSH\)](#)。兩種見證程序都由一個單字節版本號和一個較長的雜湊組成。它們看起來非常相似，但是卻有著不同的解釋：一個被解釋為一個公鑰的雜湊，它被簽名滿足，另一個被解釋為腳本的雜湊，被一個贖回腳本滿足。它們之間的關鍵區別在於雜湊的長度：

- P2WPKH 中公鑰的雜湊是 20 字節
- P2WSH 中腳本的雜湊是 32 字節

這是允許錢包區分兩種見證程序的一個區別。通過查看雜湊的長度，錢包可以確定它是什麼類型的見證程序，P2WPKH 或P2WSH。

### 升級到隔離見證

從前面的例子可以看出，升級為隔離見證是一個兩步的過程。首先，錢包必須創建特殊的隔離型輸出。然後，這些輸出可以被知道如何構建隔離見證交易的錢包花費。在這些例子中，Alice的錢包支持segwit，能夠使用Segregated Witness 腳本創建特殊輸出。鮑勃的錢包也是支持segwit的，能夠花費那些輸出。從這個例子中可能不明顯的是，在實踐中，Alice的錢包需要知道Bob使用了一個支持segwit的錢包並可以使用這些輸出。否則，如果Bob的錢包沒有升級，當Alice試圖向Bob進行segwit支付，那麼Bob的錢包將無法檢測到這些支付。

**Tip** 對於P2WPKH和P2WSH付款類型，付款人和收款人的錢包都需要升級才能使用segwit。此外，付款人的錢包需要知道收款人的錢包是否具有隔離見證功能。

隔離見證不會在整個網路中同時實施。而是向後兼容的升級，新老客戶可以共存。錢包開發人員將獨立升級錢包軟體以添加隔離見證功能。當付款人和收款人都支持隔離見證時，可以使用P2WPKH和P2WSH付款類型。傳統的P2PKH和P2SH將繼續為沒有升級的錢包工作。這留下了兩個重要的場景，下一節將討論這些：

- 付款人的錢包不支持隔離見證的，向支持隔離見證的收款人錢包付款
- 付款人的支持隔離見證的錢包通過地址識別和區分收款方是否支持隔離見證的能力

### P2SH中嵌入的隔離見證

舉個例子，假設Alice的錢包沒有升級到segwit，但是Bob的錢包已升級並可以處理segwit交易。Alice和Bob可以使“舊”的非segwit交易。但是Bob可能想使用segwit，利用適用於隔離見證的折扣，降低交易費用。

在這種情況下，Bob的錢包可以構建一個內部包含segwit腳本的P2SH地址。Alice的錢包將其視為“正常”的P2SH地址，並且可以在不知道segwit的情況下付款。然後Bob的錢包可以通過segwit交易來花費這筆款項，充分利用segwit並降低交易費用。

兩種形式的見證腳本，P2WPKH 和 P2WSH，都可以嵌入到P2SH地址中。第一個被記作P2SH (P2WPKH)，第二個被記作P2SH (P2WSH)。

### Pay-to-Script-Hash 中的 Pay-to-Witness-Public-Key-Hash

我們將研究的第一種見證腳本是P2SH (P2WPKH)。這是一個Pay-to-Witness-Public-Key-Hash見證程序，嵌入在Pay-to-Script-Hash腳本中，以便它可以被不知道segwit的錢包使用。

Bob的錢包用Bob的公鑰構造了一個P2WPKH見證程序。這個見證程序之後被雜湊，並將結果編碼為P2SH腳本。這個P2SH腳本轉換為比特幣地址，其中一個以“3”開頭，正如我們在[支付到腳本雜湊 Pay-to-Script-Hash \(P2SH\)](#)部分看到的那樣。

Bob的錢包從我們之前看到的P2WPKH見證程序開始：

Bob's P2WPKH witness program

```
0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
```

P2WPKH見證程序由見證版本和Bob的20字節公鑰雜湊組成。

Bob的錢包然後對之前的見證程序進行雜湊，首先是SHA256，然後是RIPEMD160，產生另一個20字節的雜湊值。

讓我們使用命令行中的 bx 命令來重現：

HASH160 of the P2WPKH witness program

```
echo \
'0 [ab68025513c3dbd2f7b92a94e0581f5d50f654e7]' \
| bx script-encode | bx sha256 | bx ripemd160
3e0547268b3b19288b3adef9719ec8659f4b2b0b
```

接著，將贖回腳本的雜湊值轉換為比特幣地址。再次使用 bx：

P2SH address

```
echo \
'3e0547268b3b19288b3adef9719ec8659f4b2b0b' \
| bx address-encode -v 5
37Lx99uaGn5avKBxiW26HqedQE3LrDCZru
```

現在，Bob可以對客戶展示這個地址，讓他們為咖啡付費。Alice的錢包可以支付給37Lx99uaGn5avKBxiW26HqedQE3LrDCZru，就像支付給任何其他比特幣地址一樣。

為了向Bob付款，Alice的錢包會使用如下的P2HS脚本鎖定輸出：

```
HASH160 3e0547268b3b19288b3adef9719ec8659f4b2b0b EQUAL
```

即使Alice的錢包不支持隔離見證，這筆付款也可以被Bob使用隔離見證交易消費：

### Pay-to-Script-Hash 中的 Pay-to-Witness-Script-Hash

類似地，多重簽名腳本或其他複雜腳本的P2WSH見證程序也可以嵌入到P2SH腳本和地址中，使任何錢包都可以進行segwit兼容的支付。

正如我們在 [Pay-to-Witness-Script-Hash \(P2WSH\)](#) 中看到的，Mohammed的公司正在對多重簽名腳本使用隔離見證付款。為了使任何客戶都能向他的公司付款（無論他們的錢包是否升級到了支持segwit的版本），Mohammed的錢包可以在一個P2SH腳本中嵌入P2WSH見證程序。

首先，Mohammed的錢包用SHA256(僅此一次)將贖回腳本進行了雜湊。讓我們在命令行上使用 bx 來完成：

Mohammed's wallet creates a P2WSH witness program

```
echo \
2 \
[04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395D7
EEC6984D83F1F50C900A24DD47F569FD4193AF5DE762C587] \
[04A2192968D8655D6A935BEAF2CA23E3FB87A3495E7AF308EDF08DAC3C1FCBFC2C75B4B0F4D0B1B70C
D2423657738C0C2B1D5CE65C97D78D0E34224858008E8B49] \
[047E63248B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D5737812F393DA7D4420D7E1
A9162F0279FC10F1E8E8F3020DECDBC3C0DD389D9977965] \
[0421D65CBD7149B255382ED7F78E946580657EE6FDA162A187543A9D85BAAA93A4AB3A8F044DADA618
D087227440645ABE8A35DA8C5B73997AD343BE5C2AFD94A5] \
[043752580AFA1ECED3C68D446BCAB69AC0BA7DF50D56231BE0AABF1FDEEC78A6A45E394BA29A1EDF51
8C022DD618DA774D207D137AAB59E0B000EB7ED238F4D800] \
5 CHECKMULTISIG \
| bx script-encode | bx sha256
9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73
```

接下來，雜湊後的贖回腳本轉換為P2WSH見證程序：

```
0 9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73
```

然後，使用SHA256和RIPEMD160對見證程序本身進行雜湊處理，生成一個新的20字節的雜湊，就像傳統的P2SH那樣，我們使用 bx 實驗：

The HASH160 of the P2WSH witness program

```
echo \
'0 [9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73]' \
| bx script-encode | bx sha256 | bx ripemd160
86762607e8fe87c0c37740cddee880988b9455b2
```

再然後，錢包從這個雜湊值構建一個P2SH比特幣地址，使用 bx 實驗：

P2SH bitcoin address

```
echo \
'86762607e8fe87c0c37740cddee880988b9455b2' \
| bx address-encode -v 5
3Dwz1MXhM6EfFoJChHCxh1jWWh8GQqRenG
```

現在，Mohammed的客戶不需要必須支持segwit就可以支付到這個地址。要向Mohammed付款，錢包將用以下P2SH腳本鎖定輸出：

P2SH script used to lock payments to Mohammed's multisig

```
HASH160 86762607e8fe87c0c37740cddee880988b9455b2 EQUAL
```

然後，Mohammed的公司可以利用segwit的好處(包括較低的交易費用)，構建segwit交易來花費這些款項。

### 隔離見證地址

即使是在segwit啟用後，大部分的錢包升級也需要一些時間。一開始，segwit將被嵌入P2SH，如我們在前一節中看到的那樣，來方便地兼容支持segwit和不支持segwit的錢包。

然而，一旦錢包廣泛支持segwit，就有必要將目擊者腳本直接編碼成為segwit的原生地址格式，而不是嵌入到P2SH中。

原生segwit地址格式定義在 BIP-173 中：

#### **BIP-173**

Base32 address format for native v0-16 witness outputs

BIP-173僅對見證腳本（P2WPKH和P2WSH）進行編碼。它與非segwit P2PKH或P2SH腳本不兼容。與傳統的比特幣地址的Base58編碼相比，BIP-173是Base32校驗和編碼。BIP-173地址也稱為 *bech32* 地址，發音為 "beh-ch thirty two"，暗指使用"BCH"錯誤檢測演算法和32字符編碼集。

BIP-173地址使用32個小寫字母的字母數字字符集，經過仔細選擇以減少誤讀或錯誤輸入。通過只選擇小寫字母集，bech32更容易閱讀，朗讀，並且在QR碼中的編碼效率提高了45%。

BCH錯誤檢測演算法比以前的校驗和演算法(Base58Check)有了很大的改進，它不僅檢測，還能糾正錯誤。地址輸入接口(如表單中的文本框)可以檢測並突出顯示在檢測錯誤時最可能出現錯誤的字符。

根據BIP-173規範，這裡是一些 bech32 地址的示例：

#### **Mainnet P2WPKH**

bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4

#### **Testnet P2WPKH**

tb1qw508d6qejxtdg4y5r3zarvary0c5xw7kxpjzsx

#### **Mainnet P2WSH**

bc1qrp33g0q5c5txsp9arysr4k6zdkfs4nce4xj0gdcccefvpysxf3qccfrmv3

#### **Testnet P2WSH**

tb1qrp33g0q5c5txsp9arysr4k6zdkfs4nce4xj0gdcccefvpysxf3q0sl5k7

如你所見，segwit bech32字串長達90個字符，由三個部分組成：

#### 人類可讀的部分

"bc" 或 "tb" 標明主網（mainnet）還是測試網（testnet）。

#### 分隔符

數字 "1"，不是32字符編碼集的一部分，只當做分隔符出現。

#### 數據部分

至少6個字母數字字符，校驗和編碼的見證腳本

此時，只有少數錢包接受或生成原生segwit bech32地址，但隨著segwit的廣泛使用，你將越來越多地看到這些地址。

#### 交易標識符 Transaction identifiers

隔離見證的最大好處之一是它消除了第三方交易可鑽性。

在segwit之前，交易的簽名可以被第三方微妙地修改，改變它們的交易ID(雜湊)，而不改變任何基本屬性(輸入、輸出、數量)。這為拒絕服務攻擊和攻擊編寫糟糕的錢包軟體創造了機會，這些錢包假定未經確認的交易的雜湊是不可變的。

通過引入隔離見證，交易有了兩個標識符 txid 和 wtxid。傳統的交易ID txid 是序列化交易的雙SHA256雜湊，沒有見證數據。交易的 wtxid 是具有見證數據的交易的新的序列化格式的雙SHA256雜湊。

傳統的 txid 的計算方式與nonsegwit交易完全相同。但是，由於segwit交易在每個輸入中都有空的 scriptSig，因此不存在可由第三方修改的交易部分。因此，在segwit交易中，即使交易未經確認，txid 也是第三方不可變的。

wtxid 就像一個“擴展”的ID，因為這個雜湊還包含了見證數據。如果交易不帶有見證數據傳輸，那麼 wtxid 和 txid 是相同的。注意，由於 wtxid 包含見證數據（簽名），並且由於見證數據可能具有可鍛性，所以應認為 wtxid 在交易確認之前具有可鍛性。只有當交易的輸入都是segwit輸入時，segwit 交易的 txid 才能被認為是不可變的。

Tip

隔離見證交易有兩個ID：txid 和 wtxid。txid 是不包含見證數據的交易的雜湊，wtxid 是包含見證數據的雜湊。所有輸入都為 segwit 輸入的交易的 txid 不易受第三方交易可鍛性影響。

## 隔離見證的新簽名演算法

隔離見證修改了四種簽名驗證函數（CHECKSIG，CHECKSIGVERIFY，CHECKMULTISIG 和 CHECKMULTISIGVERIFY）的語義，改變了交易保證雜湊的計算方式。

比特幣交易中的簽名應用於*commitment hash*，這是根據交易數據計算的，鎖定數據的特定部分，表明簽名者對這些值的保證。例如，在簡單的 SIGHASH\_ALL 類型簽名中，保證雜湊包括所有輸入和輸出。

不幸的是，計算保證雜湊的方式引入了驗證簽名的節點可能被迫執行大量雜湊計算的可能性。具體而言，雜湊操作相對於交易中的簽名操作的數量以  $O(n^2)$  的複雜度增長。因此，攻擊者可以創建帶有大量簽名操作的交易，導致整個比特幣網路必須執行數百或數千次雜湊操作才能驗證交易。

Segwit提供了通過改變保證雜湊計算方式來解決這個問題的機會。對於segwit版本0見證程序，使用BIP-143中規定的改進的保證雜湊演算法進行簽名驗證。

新演算法實現了兩個重要目標。首先，雜湊操作的數量隨著簽名操作的數量逐漸以  $O(n)$  增長，減少了用過於複雜的交易創建拒絕服務攻擊的機會。其次，保證雜湊現在還將每個輸入的值（金額）作為雜湊的一部分，這意味著簽名者無需“獲取”並檢查輸入引用的前一個交易就可以保證特定的輸入值。對於離線設備（例如硬體錢包），這大大簡化了主機與硬體錢包之間的通信，消除了對以前的交易進行驗證的需要。硬體錢包可以接受不受信任的主機“所聲明的”輸入值，因為如果輸入值不正確則簽名無效，硬體錢包在簽名輸入前不需要驗證該值。

## 隔離見證的經濟效益

比特幣挖掘節點和完整節點會產生用於支持比特幣網路和區塊鏈的資源的成本。隨著比特幣交易量的增加，資源成本（CPU，網路帶寬，硬碟空間，內存）也不斷增加。礦工通過收取與每次交易的大小（字節）成比例的費用來補償這些成本。非挖礦（Nonmining）完整節點沒有得到補償，蒙受了損失，因為他們需要運行一個權威的完全驗證的全索引節點，可能是因為他們使用節點來經營比特幣業務。

如果沒有交易費用，比特幣數據的增長可能會大幅增加。費用旨在通過基於市場的價格發現機制，將比特幣用戶的需求與交易對網路帶來的負擔相匹配。

基於交易規模的費用計算將交易中的所有數據視為成本相同的。但是從完整節點和礦工的角度來看，交易的某些部分承擔了更高的成本。添加到比特幣網路的每筆交易都會影響節點上四種資源的消耗：

### 硬盤空間

每筆交易都儲存在區塊鏈中，添加到區塊鏈的總大小上。區塊鏈儲存在硬碟上，但是可以通過“刪除”舊的交易來優化儲存。

### CPU

每筆交易都必須被驗證，這需要CPU時間。

### 帶寬

每筆交易都在網路上至少傳輸一次（通過泛洪傳播），如果在區塊傳播協議中沒有進行任何優化，交易將作為區塊的一部分再次傳輸，從而對網路容量的影響加倍。

## 內存

驗證交易的節點將UTXO索引或整個UTXO集保存在內存中，以加快驗證。因為內存至少比硬碟貴一個數量級，所以UTXO集的增長不成比例地增加了運行節點的成本。

從列表中可以看出，並非交易的每個部分都對運行節點的成本或比特幣支持更多交易的能力產生同等影響。交易中最昂貴的部分是新創建的輸出，因為它們被添加到內存中的UTXO集合中。相比之下，簽名（又名見證數據）為增加了最小的網路負擔和節點運行成本，因為見證數據只被驗證一次，然後再也不會使用。此外，在收到新的交易並驗證見證數據之後，節點立即丟棄該見證數據。如果費用是根據交易規模計算的，而不區分這兩種數據，那麼市場化的費用激勵就不符合交易實際施加的成本。實際上，目前的費用結構實際上鼓勵了相反的行為，因為見證數據是交易的最大部分。

交易在其輸入中花費UTXO，並在輸出中創建新的UTXO。因此，一個輸入數量大於輸出數量的交易將導致UTXO集的減少，而一個輸出數量大於輸入數量的交易將導致UTXO集的增加。讓我們考慮輸入和輸出之間的差異，並稱之為“淨增UTXO”（"Net-new-UTXO"）。這是一個重要的指標，因為它告訴我們一個交易將對網路中最昂貴的資源（即內存裡的UTXO集）產生什麼影響。Net-new-UTXO為正的交易增加負擔，Net-new-UTXO為負的交易減少負擔。因此，我們希望鼓勵Net-new-UTXO為負或為0的交易。

讓我們看一個例子，說明在有無隔離見證的情況下，交易費用計算產生了哪些激勵。我們將看兩個不同的交易。交易A是有3個輸入2個輸出的交易，Net-new-UTXO為-1。交易B是2個輸入3個輸出的交易，Net-new-UTXO為1，意味著它增加了一個UTXO到UTXO集，給整個比特幣網路帶來了額外的成本。這兩筆交易都使用多重簽名（2-of-3）腳本來說明覆雜腳本如何增加隔離見證對費用的影響。假設交易費為每字節30 satoshi，見證數據擁有75%的費用折扣：

### ***Without Segregated Witness***

Transaction A fee: 25,710 satoshi

Transaction B fee: 18,990 satoshi

### ***With Segregated Witness***

Transaction A fee: 8,130 satoshi

Transaction B fee: 12,045 satoshi

這兩種交易在實施隔離見證時都較為便宜。但是比較這兩筆交易的成本，我們發現在隔離見證之前，Net-new-UTXO為負的交易費用較高。在隔離見證後，交易費用與鼓勵減少新的UTXO產生的激勵相一致，不會無意地懲罰有許多輸入的交易。

因此，隔離見證對比特幣用戶支付的費用有兩個主要影響。首先，segwit通過見證數據折扣，和增加比特幣區塊鏈的能力，來降低交易的總體成本。其次，segwit對見證數據的折扣糾正了可能無意中導致UTXO集合中更加膨脹的激勵錯配。

# 比特幣網路

## 點對點網路架構

比特幣是構建在互聯網之上的點對點網路體系結構。術語點對點（P2P）意味著參與網路的電腦是彼此對等的，它們都是平等的，沒有“特殊”節點，並且所有節點都分攤提供網路服務的負擔。網路節點以“扁平”拓撲互連在網狀網路中。網路中沒有中央伺服器，沒有集中化服務，也沒有層次結構。P2P網路中的節點同時提供和消費服務，彼此互惠。P2P網路具有天然的彈性，去中心性和開放性。P2P網路架構的一個卓越例子就是早期的互聯網本身，IP網路上的節點是平等的。如今，互聯網結構更有層次，但互聯網協議仍然保留了其扁平拓撲的本質。除比特幣之外，P2P技術最大最成功的應用是檔案共享，Napster為先鋒，BitTorrent是該架構的最新演變。

比特幣的P2P網路架構不只是一種拓撲選擇。比特幣是一種P2P設計的數字現金系統，網路架構既是該核心特徵的反映，也是其基礎。控制權的去中心化是一個核心設計原則，只能通過一個扁平的，去中心化的P2P共識網路來實現和維護。

術語“比特幣網路”是指運行比特幣P2P協議的節點的集合。除比特幣P2P協議外，還有其他一些協議，如Stratum，用於採礦和輕量級或移動錢包。這些附加協議由網關路由伺服器提供，網關路由伺服器使用比特幣P2P協議訪問比特幣網路，然後將該網路擴展到運行其他協議的節點。例如，Stratum伺服器通過Stratum協議將Stratum挖礦節點連接到比特幣主網，並將Stratum協議連接到比特幣P2P協議。我們使用術語“擴展比特幣網路”來指包括比特幣P2P協議，礦池協議，Stratum協議以及連接比特幣系統組件的任何其他相關協議的整體網路。

## 節點類型和角色

儘管比特幣P2P網路中的節點是對等的，但根據其支持的功能不同，它們承擔的角色可能不同。比特幣節點是一組功能的集合：路由，區塊鏈資料庫，挖礦和錢包服務。具有全部四個功能的完整節點顯示在 [A bitcoin network node with all four functions: wallet, miner, full blockchain database, and network routing](#) 中。

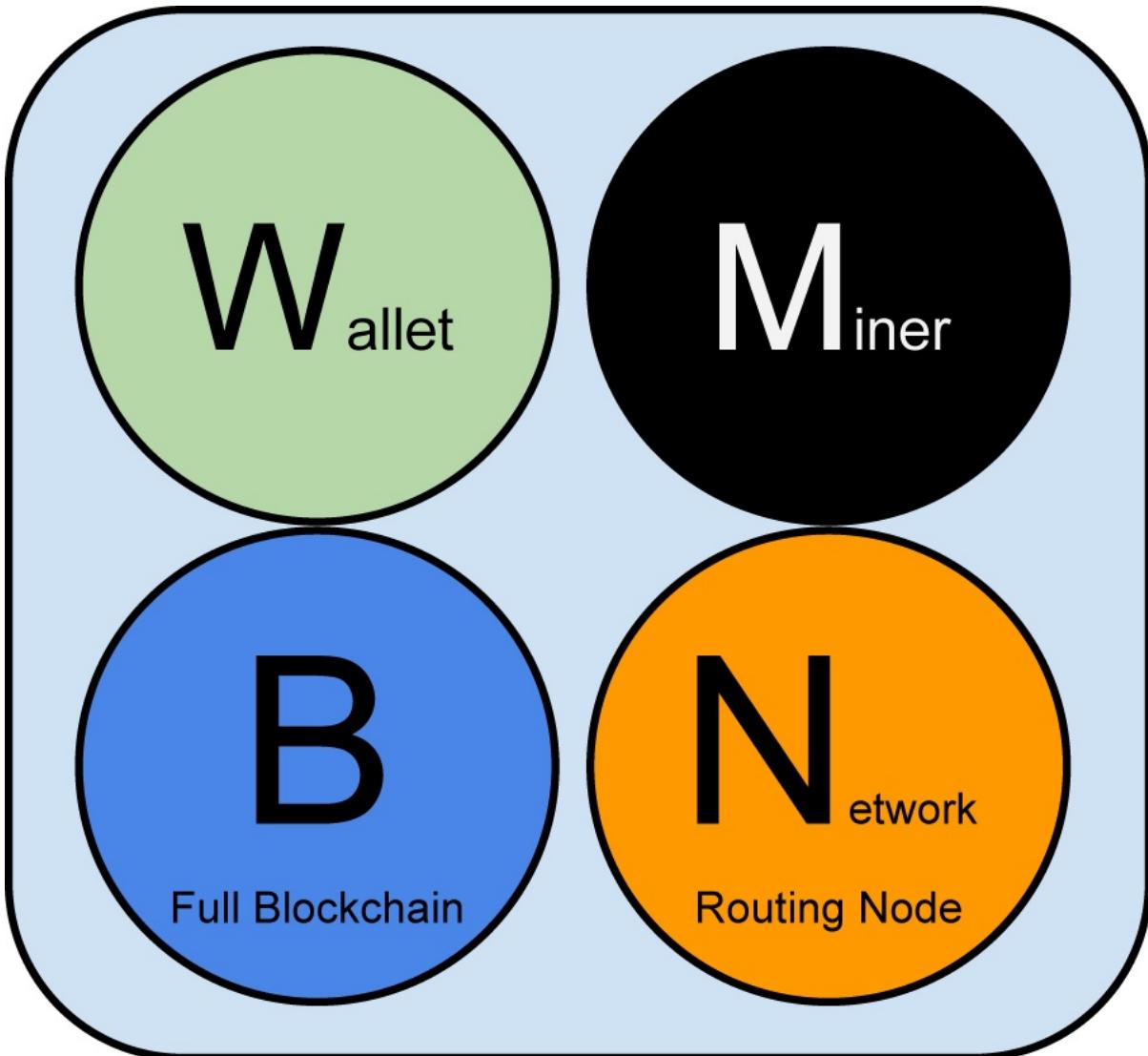


Figure 1. A bitcoin network node with all four functions: wallet, miner, full blockchain database, and network routing  
所有節點都包含用於參與網路的路由功能，可能包含其他功能。所有節點都會驗證並傳播交易和區塊，並發現和維護與其他節點的連接。在 [A bitcoin network node with all four functions: wallet, miner, full blockchain database, and network routing](#) 的完整節點示例中，路由功能由名為“Network Routing Node”的橙色圓圈或字母“N”表示。

一些稱為完整節點的節點也保留區塊鏈的完整和最新的副本。完整的節點可以自主和權威地驗證任何交易，無需外部參考。一些節點只維護區塊鏈的一個子集，並使用稱為 *simplified payment verification* 或SPV的方法驗證交易，這些節點被稱為SPV節點或輕量級節點。在圖中的完整節點示例中，完整節點區塊鏈資料庫功能由稱為“Full Blockchain”或字母“B”的圓圈表示。在 [The extended bitcoin network showing various node types, gateways, and protocols](#) 中，SPV節點被繪製時沒有“B”，表明它們沒有完整的區塊鏈副本。

挖礦節點通過運行專門的硬體解決Proof-of-Work演算法來競爭創建新區塊。一些挖礦節點也是完整節點，維護區塊鏈的完整副本，而另一些節點是加入礦池的輕量級節點，並且依賴於池伺服器維護完整節點。挖礦功能在完整節點中顯示為一個稱為“Miner”或字母“M”的黑色圓圈。

用戶錢包可能是完整節點的一部分，通常與桌面比特幣客戶端情況相同。越來越多的用戶錢包，尤其是那些運行在資源受限設備（如智能手機）上的用戶錢包是SPV節點。錢包功能在 [A bitcoin network node with all four functions: wallet, miner, full blockchain database, and network routing](#) 中顯示為稱為“Wallet”或字母“W”的綠色圓圈。

除了比特幣P2P協議的主要節點類型外，還有運行其他協議的伺服器和節點，例如專門的礦池協議和輕量級客戶端訪問協議。

[Different types of nodes on the extended bitcoin network](#) 展示了擴展比特幣網路中的多數普通節點類型

## 擴展比特幣網路

運行比特幣P2P協議的主要比特幣網路由5000到8000個運行各種版本比特幣參考客戶端（Bitcoin Core）的監聽節點，和幾百個運行比特幣P2P協議的各種其他實現的節點組成，例如Bitcoin Classic，Bitcoin Unlimited，BitcoinJ，Libbitcoin，btcd和bcoin。比特幣P2P網路中的一小部分節點也是挖礦節點，在挖礦過程中競爭，驗證交易並創建新區塊。各種大公司通過運行基於比特幣核心客戶端的全節點客戶端與比特幣網路進行接口，具有完整的區塊鏈副本和網路節點，但沒有挖掘或錢包功能。這些節點充當網路邊緣路由器，允許將各種其他服務（交易所，錢包，區塊瀏覽器，商家支付處理）其上構建。

擴展比特幣網路包括運行比特幣P2P協議的網路，以及運行特殊協議的節點。連接到主比特幣P2P網路的是許多礦池伺服器和連接運行其他協議的節點的協議網關。這些其他協議節點主要是礦池節點（請參閱[\[mining\]](#)）和輕量級錢包客戶端，它們不包含區塊鏈的完整副本。

[The extended bitcoin network showing various node types, gateways, and protocols](#) 顯示了擴展比特幣網路，其中包括各種類型的節點，網關伺服器，邊緣路由器和錢包客戶端以及它們用於彼此連接的各種協議。

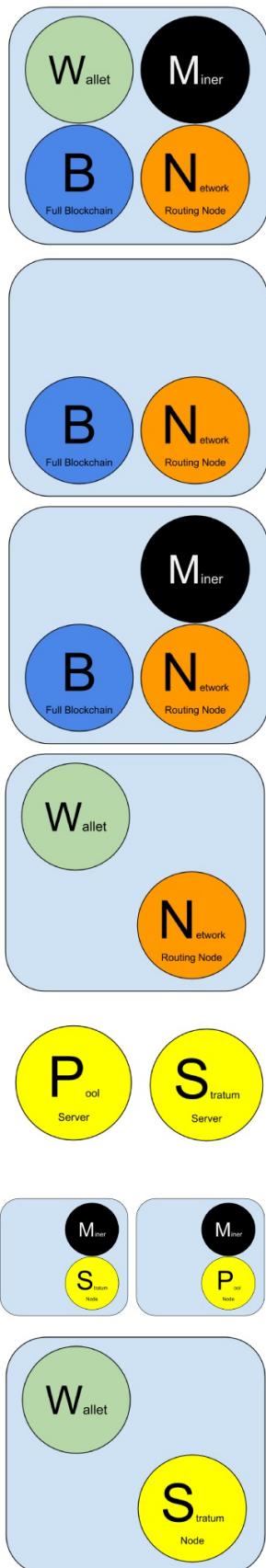


Figure 2. Different types of nodes on the extended bitcoin network

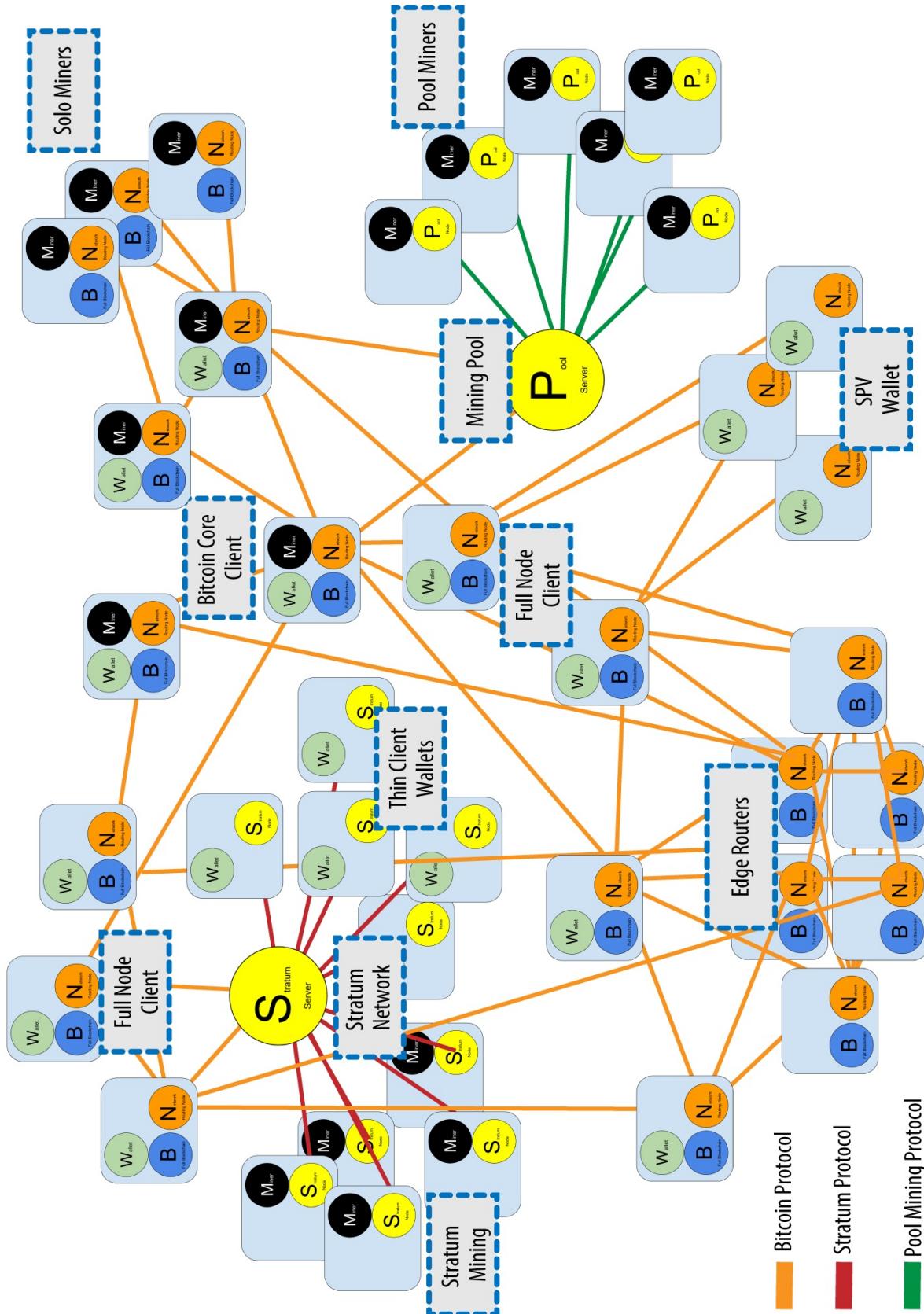


Figure 3. The extended bitcoin network showing various node types, gateways, and protocols

## 比特幣中繼網路

雖然比特幣P2P網路服務於各種節點類型的普遍需求，但它為比特幣挖礦節點的特殊需求呈現出過高的網路延遲。

比特幣礦工們進行時間敏感的競爭，解決工作量證明問題來擴展區塊鏈（參見[mining]）。參加這場比賽時，比特幣礦工必須儘量縮短傳播獲勝區塊和下一輪比賽開始之間的時間。在採礦中，網路延遲與利潤空間直接相關。

比特幣中繼網路 *Bitcoin Relay Network* 是旨在儘量減少礦工之間區塊傳輸延遲的網路。最初的 *Bitcoin Relay Network* 由核心開發者 Matt Corallo 在 2015 年創建，使礦工之間以極小的延遲快速同步區塊。該網路由幾個專門的節點組成，這些節點位於世界各地的亞馬遜網路服務基礎設施上，用於連接大多數礦工和礦池。

最初的比特幣中繼網路在 2016 年被 *Fast Internet Bitcoin Relay Engine* (<http://bitcoinfibre.org> [FIBRE]) 替代，這也是由核心開發人員 Matt Corallo 創建的。FIBER 是一個基於 UDP 的中繼網路，用於中繼節點網路中的區塊。FIBER 實現了緊湊的區塊 *compact block* 優化以進一步減少傳輸的數據量和網路延遲。

另一箇中繼網路（仍處於提案階段）是 <http://www.falcon-net.org/about> [Falcon]，是基於康奈爾大學的研究的。Falcon 使用“直通式路由”（cut-through-routing）而不是“儲存轉發”（store-and-forward）來減少等待時間，方法是傳輸部分數據區塊，而不是等待接收完整數據區塊。

中繼網路並不是比特幣 P2P 網路的替代品。相反，它們是覆蓋網路，為具有特殊需求的節點之間提供附加連接。像高速公路不是農村公路的替代品，而是交通繁忙的兩個點之間的捷徑，你仍然需要小路連接到高速公路。

## 網路發現

當新節點啟動時，它必須發現網路上的其他比特幣節點才能加入。要啟動此過程，新節點必須發現網路上至少一個現有節點並連接到該節點。地理位置不重要，比特幣網路拓撲結構沒有地理上的定義。因此，任何現有的比特幣節點都可以被隨機選擇。

要連接到一個已知的節點，節點建立一個 TCP 連接，通常連接到端口 8333（比特幣通常使用的端口），或者提供一個替代端口。在建立連接後，節點將通過發送包含基本標識訊息的版本（version）訊息開始“握手”（請參見 [The initial handshake between peers](#)），其中包括：

### **nVersion**

客戶端使用的比特幣 P2P 協議版本（例如，70002）

### **nLocalServices**

一個本節點支持的本地服務列表，現在只是 NODE\_NETWORK

### **nTime**

當前時間

### **addrYou**

遠程節點的 IP 地址

### **addrMe**

本地節點的 IP 地址

### **subver**

體現在此節點上運行的軟體類型的子版本（例如，/Satoshi:0.9.2.1/）

### **BestHeight**

本節點的區塊鏈的區塊高度

（查看 [GitHub](#) 上的 version 網路訊息示例。）

version 訊息通常是節點發送給另一個對等節點的第一條訊息。接收到 version 訊息的本地節點將檢查遠程節點報告的 nVersion 然後決定是否兼容遠程節點。如果是兼容的，本地節點將認可 version 訊息並通過 verack 訊息建立鏈接。

新節點如何查找對等節點？第一種方法是使用許多“DNS種子”來查詢 DNS，這些 DNS 伺服器提供比特幣節點的 IP 地址列表。其中一些 DNS 種子提供穩定的比特幣偵聽節點的 IP 地址的靜態列表。一些 DNS 種子是 BIND (Berkeley Internet Name 守護進程) 的自定義實現，它從一個爬蟲或一個長時間運行的比特幣節點收集的比特幣節點地址列表中返回一個隨

機子集。比特幣核心客戶端包含五個不同DNS種子的名稱。不同DNS種子的所有權和實現的多樣性為初始引導過程提供了高度的可靠性。在Bitcoin Core客戶端中，使用DNS種子的選項由選項開關 `-dnsseed`（預設設置為1，以使用DNS種子）控制。

或者，一個對網路一無所知的啟動節點必須被給予至少一個比特幣節點的IP地址，之後它可以通過進一步的介紹建立連接。命令行參數 `-seednode` 可以用於連接到一個節點，只是為了將其作為種子使用。在使用初始種子節點進行介紹之後，客戶端將與其斷開並使用新發現的對等節點。

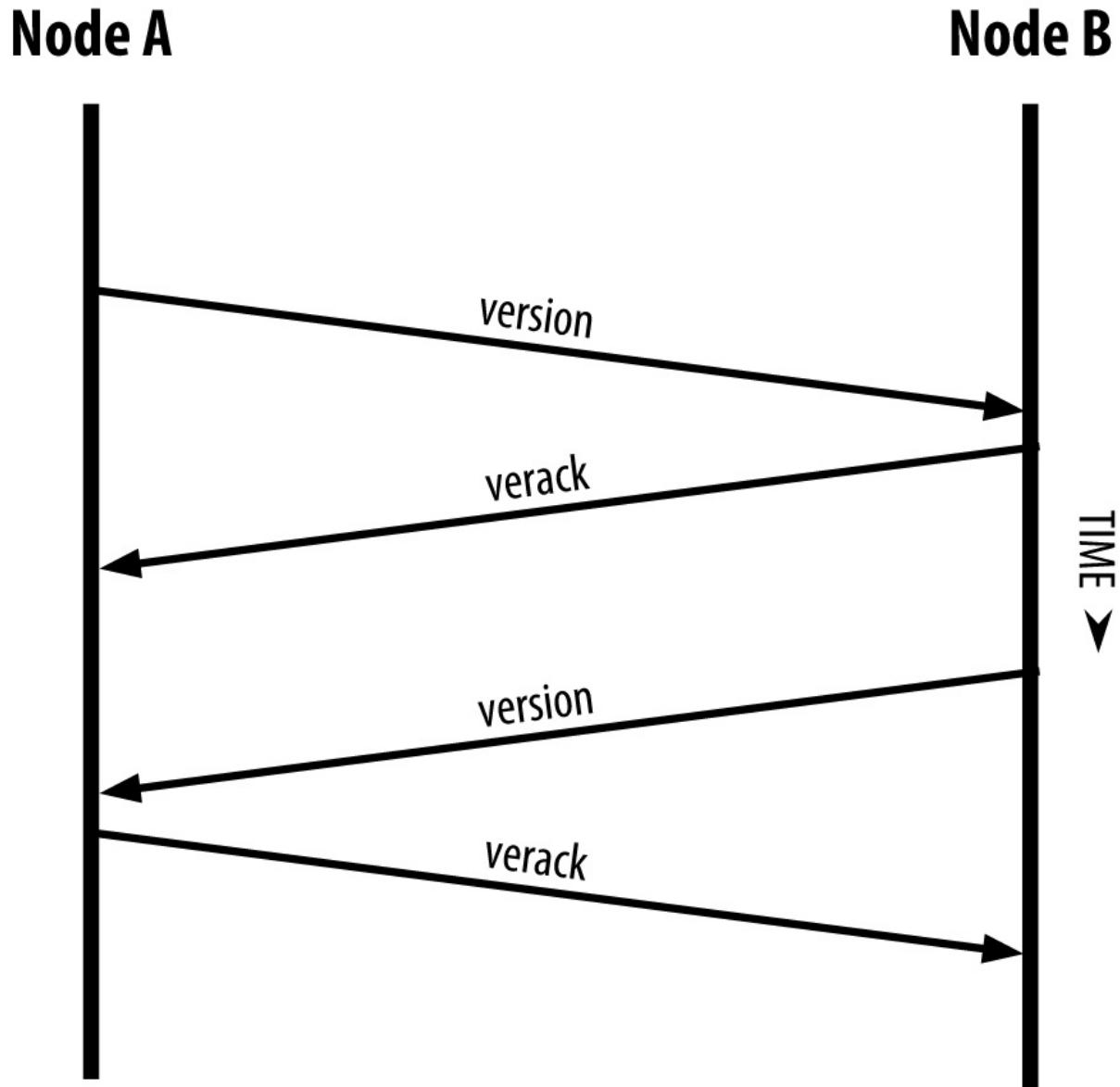


Figure 4. The initial handshake between peers

一旦建立了一個或多個連接，新節點將向其鄰居發送一個包含自己IP地址的 `addr` 訊息。鄰居將依次將 `addr` 訊息轉發給它們的鄰居，以確保新連接的節點變得眾所周知並且更好地連接。另外，新連接的節點可以向鄰居發送 `getaddr`，要求他們返回其他對等節點的IP地址列表。這樣，一個節點能找到可以連接的對等節點，並在網路上通告其存在以供其他節點找到它。[Address propagation and discovery](#) 展示了地址發現協議。

# Node A

# Node B

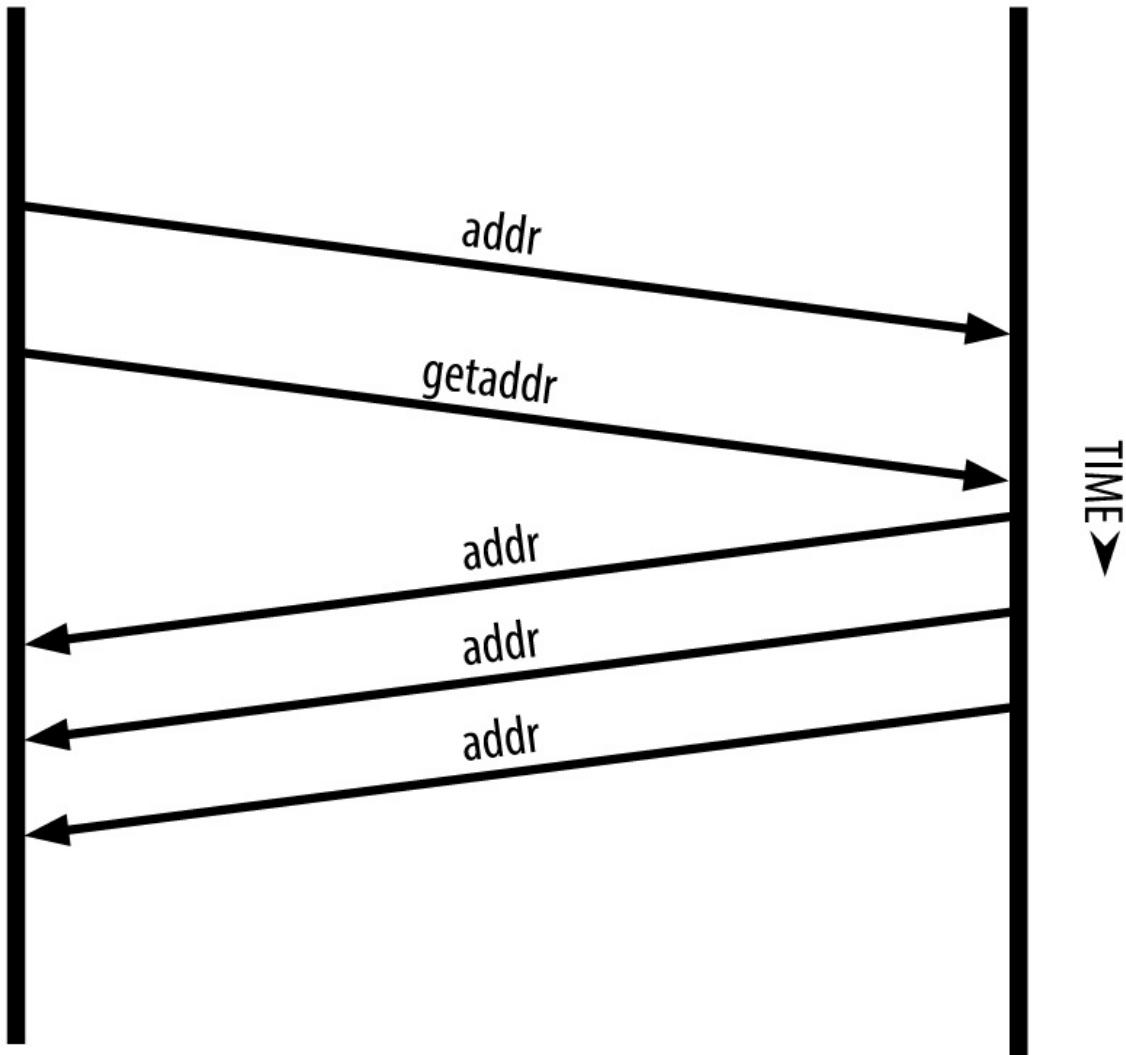


Figure 5. Address propagation and discovery

一個節點必須連接到幾個不同的對等節點，以便建立到比特幣網路的不同路徑。路徑不是可靠的 - 節點隨時可以加入或離開 - 所以節點必須在丟失舊鏈接時持續發現新節點，並在啟動時幫助（通知）其他節點。啟動時只需要一個連接，因為第一個節點可以向他的對等節點介紹本節點，這些節點又可以提供進一步的介紹。連接到過多的節點也是不必要和浪費網路資源的。啟動之後，節點將記住其最近成功的對等連接，如果重新啟動，它可以快速重新建立與其以前的對等網路的連接。如果以前的對等節點都沒有響應其連接請求，則該節點可以使用種子節點重新引導。

在運行Bitcoin Core客戶端的節點上，你可以使用命令 `getpeerinfo` 列出對等連接：

```
$ bitcoin-cli getpeerinfo
```

```
[
{
    "addr" : "85.213.199.39:8333",
    "services" : "00000001",
    "lastsend" : 1405634126,
    "lastrecv" : 1405634127,
    "bytessent" : 23487651,
    "bytesrecv" : 138679099,
    "conntime" : 1405021768,
    "pingtime" : 0.00000000,
```

```

    "version" : 70002,
    "subver" : "/Satoshi:0.9.2.1/",
    "inbound" : false,
    "startingheight" : 310131,
    "banscore" : 0,
    "syncnode" : true
},
{
    "addr" : "58.23.244.20:8333",
    "services" : "00000001",
    "lastsend" : 1405634127,
    "lastrecv" : 1405634124,
    "bytessent" : 4460918,
    "bytesrecv" : 8903575,
    "conntime" : 1405559628,
    "pingtime" : 0.00000000,
    "version" : 70001,
    "subver" : "/Satoshi:0.8.6/",
    "inbound" : false,
    "startingheight" : 311074,
    "banscore" : 0,
    "syncnode" : false
}
]

```

要覆蓋對等節點的自動管理並指定IP地址列表，用戶可以提供選項 `-connect = <IPAddress>` 指定一個或多個IP地址。如果使用此選項，節點將只連接到選定的IP地址，而不是自動發現和維護對等連接。

如果連接上沒有流量，節點將定期發送訊息來維護連接。如果一個節點在連接上超過90分鐘沒有進行通信，則認為它斷開連接並尋找新的對等節點。因此，網路可以動態適應瞬態節點和網路問題，並且可以根據需要進行有機增長和收縮，而無需任何中央控制。

## 完整節點

完整的節點是維護所有交易完整區塊鏈的節點。更準確地說，應該是“完整區塊鏈節點”。在比特幣早期，所有節點都是完整節點，目前Bitcoin Core客戶端是完整區塊鏈節點。然而，在過去的兩年裡，產生了不能維護完整區塊鏈的新的比特幣客戶端，以輕量級客戶端運行。我們將在下一節詳細介紹這些內容。

完整區塊鏈節點保存完整和最新的，包含所有交易的比特幣區塊鏈副本，它們獨立構建和驗證，從第一個區塊（創世區塊）開始，構建到網路中最新的已知區塊。完整區塊鏈節點可獨立並權威地驗證任何交易，無需依賴任何其他節點或訊息來源。完整區塊鏈節點依靠網路接收有關交易的新區塊的更新，然後驗證並將其合併到本地區塊鏈副本中。

運行完整區塊鏈節點為你提供純粹的比特幣體驗：獨立驗證所有交易，無需依賴或信任任何其他系統。很容易判斷你是否運行完整節點，因為它需要超過100 GB的硬碟空間來儲存完整的區塊鏈。如果你需要大量硬碟並且需要兩到三天才能與網路同步，則你正在運行完整節點。這是完全獨立和不依賴中央權威機構的代價。

完整區塊鏈比特幣客戶端有幾種可選的實現，它們使用不同的程式語言和軟體體系結構構建。然而，最常見的實現方式是Bitcoin Core參考實現，也稱為Satoshi客戶端。比特幣網路上超過75%的節點運行各種版本的比特幣核心。它在 `version` 訊息中發送的子版本字串中被標識為“Satoshi”，如我們前面看到的那樣，由命令 `getpeerinfo` 顯示，例如，`/Satoshi:0.8.6/`。

## 交換“庫存”

完整節點連接到對等節點之後的第一件事就是嘗試構建一個完整的區塊鏈。如果它是一個全新的節點，並且根本沒有區塊鏈，它只會知道一個區塊，創世區塊，這個區塊是靜態嵌入到客戶端軟體中的。從區塊 # 0（創世區塊）開始，新節點將下載數十萬個區塊來與網路同步並重新建立完整的區塊鏈。

同步區塊鏈的過程從 version 訊息開始，因為它包含 BestHeight，節點當前的區塊鏈高度（區塊數）。一個節點會看到來自對等節點的 version 訊息，知道它們各自擁有多少區塊，與它自己的區塊鏈中的區塊數進行比較。對等節點將交換 getblocks 訊息，其中包含本地區塊鏈上頂部區塊的雜湊（指紋）。另一個對等節點會識別出接收到的雜湊不是頂部的區塊，而是較舊的區塊，由此推斷其自身的本地區塊鏈比其對等節點更長。

具有較長區塊鏈的對等體比另一個節點具有更多的區塊，並且可以識別出另一個節點需要“趕上”哪些區塊。它將識別前 500 個區塊，使用 inv（庫存）訊息來共享和傳輸雜湊。缺少這些區塊的節點將通過發出一系列 getdata 訊息來請求完整區塊數據並使用 inv 訊息中的雜湊標識請求的區塊。

例如，假設一個節點只有創世區塊。然後它會收到來自對等節點的包含鏈中未來 500 個區塊的雜湊的 inv 訊息。它將開始從所有連接的對等節點請求數據區塊，分散負載，確保它不會用請求淹沒任何對等節點。該節點記錄每個對等連接“正在傳輸”的區塊數，即它已請求但未收到的區塊，並檢查它未超過限制（`MAX_BLOCKS_IN_TRANSIT_PER_PEER`）。這樣，如果需要很多區塊，它只會在先前的請求得到滿足後才請求新區塊，從而使對等節點能夠控制更新的速度並且不會壓倒網路。每個區塊被接收後，將被添加到區塊鏈中，我們將在 [blockchain] 中看到。隨著本地區塊鏈逐漸建立，更多的區塊被請求和接收，並且該過程繼續，直到節點趕上網路的其餘部分。

節點只要離線任意時間，就會將本地區塊鏈與對等節點進行比較，並獲取任何缺失的區塊。無論節點離線幾分鐘，缺少幾個區塊，或離線一個月，缺少幾千個區塊，它都會首先發送 getblocks，獲取 inv 韻應，並開始下載缺失的區塊。

[Node synchronizing the blockchain by retrieving blocks from a peer](#) 展示了庫存和區塊傳播協議。

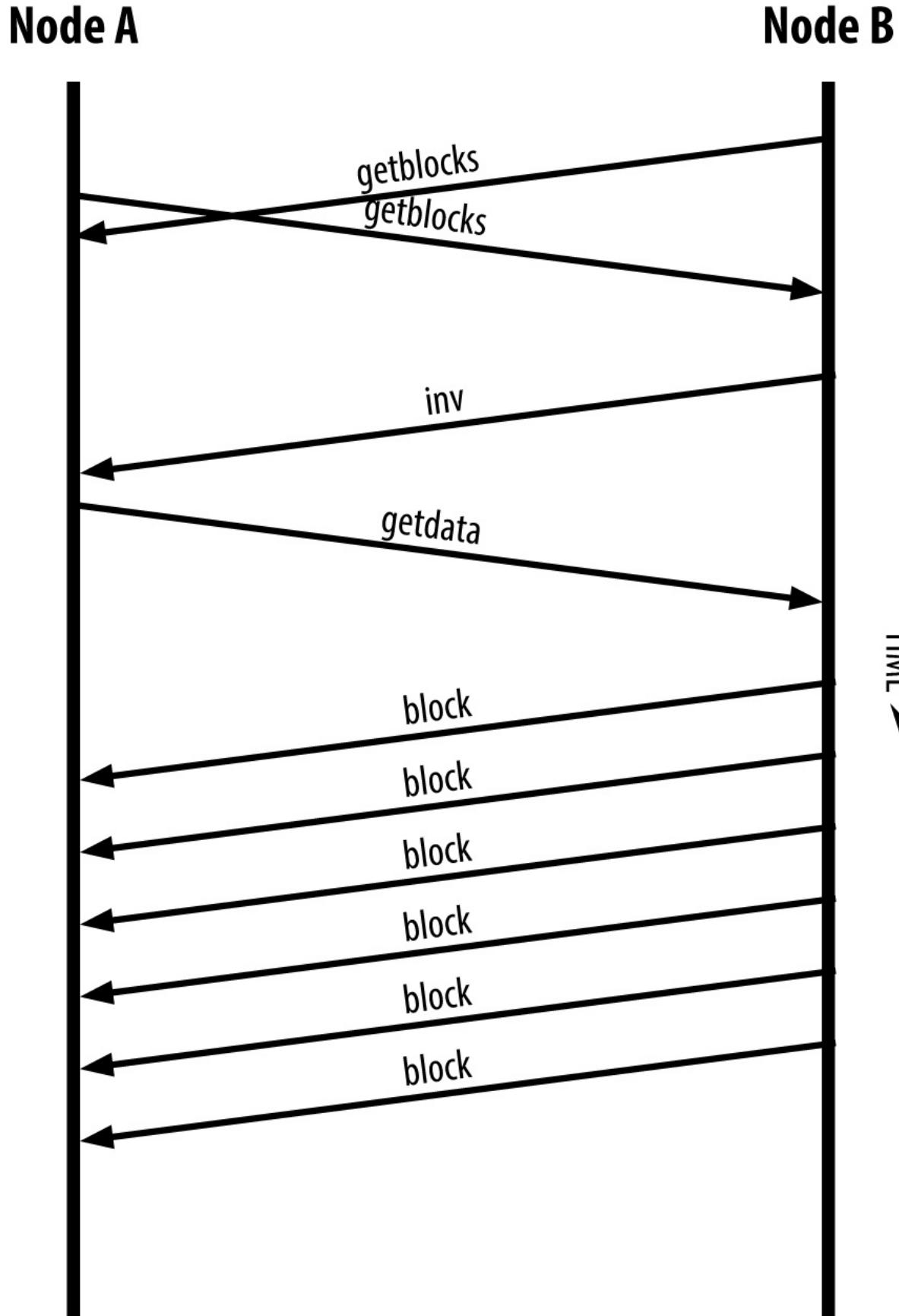


Figure 6. Node synchronizing the blockchain by retrieving blocks from a peer

### 簡單支付驗證 (SPV)

並非所有節點都有能力儲存完整的區塊鏈。許多比特幣客戶端被設計用於在空間和功耗受限的設備上運行，如智能手機，平板電腦或嵌入式系統。對於此類設備，使用 *simplified payment\_verification* (SPV) 方法可以在不儲存完整區塊鏈的情況下進行操作。這些類型的客戶端稱為SPV客戶端或輕量級客戶端。隨著比特幣的普及，SPV節點正成為比特幣節點的最常見形式，特別是比特幣錢包。

SPV節點僅下載區塊頭，而不下載每個區塊中包含的交易。由此產生的區塊鏈，比完整區塊鏈小1000倍。SPV節點無法構建可用於支出的所有UTXO的完整畫面，因為他們不知道網路上的所有交易。SPV節點使用一種不同的方法驗證交易，這種方法依賴對等節點按需提供區塊鏈相關部分的部分視圖。

作為一個比喻，一個完整節點就像一個配備了每條街道和每個地址的詳細地圖的陌生城市遊客。相比之下，一個SPV節點就像是一個只知道一條主幹道，隨機向陌生人打聽路線的陌生城市遊客。儘管兩位遊客都可以通過訪問來驗證街道的存在，但沒有地圖的遊客並不知道任何一條小街道的位置，也不知道其他街道是否存在。位於教堂街23號的前面，沒有地圖的旅遊者無法知道該市是否有其他“教堂街23號”地址，以及這是否是正確的。沒有地圖的遊客最好的機會是問足夠多的人，並期望他們中的一些人不會毆打他。

SPV通過交易在區塊鏈中的深度而不是高度來驗證。而一個完整的區塊鏈節點將構建一個完全驗證的鏈，有成千上萬的區塊和交易，一直鏈接到創世區塊。一個SPV節點將驗證所有區塊鏈（但不是所有交易）並將該鏈鏈接到感興趣的交易。

例如，當檢查第300,000區塊中的交易時，一個將所有300,000個區塊連接起來，並建立了一個完整UTXO資料庫的完整節點，通過確認UTXO的未花費狀態來確定交易的有效性。SPV節點無法驗證UTXO是否已花費。相反，SPV節點將使用 *merkle path*（參見 [\[merkle\\_trees\]](#)）在交易和包含它的區塊之間建立鏈接。然後，SPV節點等待，直到它看到在包含該交易的區塊的頂部的六個區塊300,001至300,006，並通過在區塊300,006至300,001之下建立的深度來驗證它。事實上，網路上的其他節點接受了300,000區塊，做了必要的工作，並在其上生成了六區塊以上的區塊，這代理地（間接地）證明交易不是雙重花費的事實。

當交易實際上不存在時，不能說服SPV節點在區塊中存在交易。SPV節點通過請求merkle路徑證明，並驗證區塊鏈中的工作量證明，來建立交易存在於區塊中的證明。但是，交易的存在可以從SPV節點“隱藏”。SPV節點可以明確證明交易存在，但無法驗證交易（例如同一個UTXO的雙重花費）不存在，因為它沒有所有交易的記錄。此漏洞可用於拒絕服務攻擊或針對SPV節點的雙重支出攻擊。為了防止這種情況發生，SPV節點需要隨機地連接到多個節點，以增加與至少一個誠實節點接觸的概率。這種隨機連接的需要意味著SPV節點也容易遭受網路分區攻擊或Sybil攻擊，即它們連接到了假節點或假網路，並且無法訪問誠實節點或真正的比特幣網路。

對於大多數實際的目的，連接良好的SPV節點足夠安全，在資源需求、實用性和安全性之間取得平衡。然而，對於絕對可靠的安全性，沒有什麼比運行一個完整的區塊鏈節點更好。

Tip

一個完整的區塊鏈節點通過檢查其下數千個區塊來驗證交易，以確保UTXO沒有被消耗，而SPV節點則檢查區塊在其上方的幾個區塊中埋藏的深度。

要獲取區塊頭，SPV節點使用 `getheaders` 訊息而不是 `getblocks`。響應端會使用一個 `header` 訊息發送至多2000個區塊頭。該過程與完整節點用於檢索完整區塊的過程相同。SPV節點還在與對等節點的連接上設置過濾器，以過濾由對等節點發送的未來的區塊和交易。任何感興趣的交易都使用 `getdata` 請求來檢索。對等節點生成一個包含交易的 `tx` 訊息，作為響應。[SPV node synchronizing the block headers](#) 展示了區塊頭的同步。

由於SPV節點需要檢索特定交易以選擇性地驗證它們，因此它們也會產生隱私風險。與收集每個區塊內所有交易的完整區塊鏈節點不同，SPV節點對特定數據的請求可能會無意中洩露其錢包中的地址。例如，監控網路的第三方可以跟蹤SPV節點上的錢包所請求的所有交易，並使用它們將比特幣地址與該錢包的用戶相關聯，從而破壞用戶的隱私。

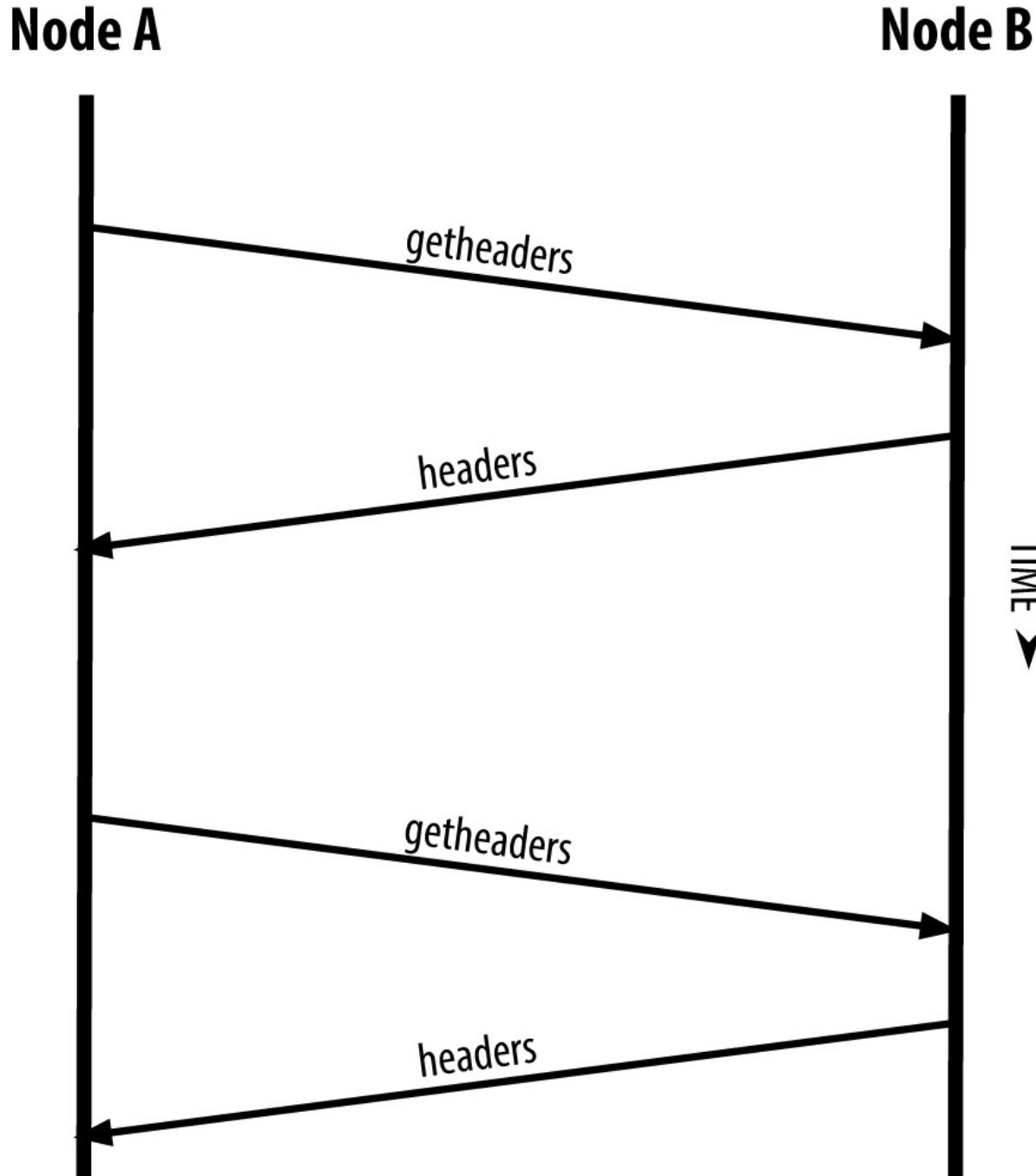


Figure 7. SPV node synchronizing the block headers

在引入SPV/輕量級節點後不久，比特幣開發人員添加了一項名為 布林過濾器 布林\_filters 的功能，以解決SPV節點的隱私風險。布林過濾器允許SPV節點通過使用概率而不是固定模式的過濾機制來接收交易子集，從而無需精確地揭示他們感興趣的地址。

[[布林\_filters]] === 布林過濾器 布林 Filters

布林過濾器是一種概率搜索過濾器，它是一種不必精確地描述所需模式的方法。布林過濾器提供了一種有效的方式來表達搜索模式，同時保護隱私。它們被SPV節點用來向他們的對等節點詢問符合特定模式的交易，而不會準確揭示他們正在搜索的地址，密鑰或交易。

在我們以前的比喻中，一個沒有地圖的遊客正在詢問指向特定地址的路線，“23 Church St.”如果她向陌生人詢問這條街的路線，她會無意中透露她的目的地。布林過濾器就像是問：“這個街區有什麼街道名稱以R-C-H結尾？”像這樣的問題揭露的目的地訊息要少一些。使用這種技術，遊客可以更詳細地指定希望的地址，例如“以U-R-C-H結尾”或更少的細

節，如“以H結尾”。通過改變搜索的精確度，遊客可以顯示或多或少的訊息，代價是獲得或多或少的具體結果。如果她提出一個不太具體的模式，她會得到更多可能的地址和更好的隱私，但是許多結果都是無關緊要的。如果她要求一個非常具體的模式，她會得到較少的結果，但會失去隱私。

布林過濾器通過允許SPV節點指定精度或隱私程度可調整的交易搜索模式來支持此功能。更具體的布林過濾器將產生準確的結果，但是以暴露SPV節點感興趣的模式為代價，從而揭示用戶錢包擁有的地址。一個不太具體的布林過濾器將產生更多關於更多交易的數據，許多數據與節點無關，但將使節點保持更好的隱私。

## 布林過濾器如何工作

布林過濾器被實現為具有N個二進制數字（比特位）的可變大小陣列，和可變數量的M個雜湊函數。雜湊函數被設計為始終產生1到N之間的輸出，對應於二進制數字的陣列。雜湊函數是確定性地生成的，以便任何實現布林過濾器的節點將總是使用相同的雜湊函數，並且針對特定輸入獲得相同的結果。通過選擇不同長度（N）布林過濾器和不同數量（M）的雜湊函數，可以調整布林過濾器，從而改變準確性水平和隱私。

在 <<布林1>> 中，我們使用非常小的16位陣列和三個雜湊函數來演示布林過濾器如何工作。

[[布林1]] .An example of a simplistic 布林 filter, with a 16-bit field and three hash functions  
image::images/mpc2\_0808.png["布林1"]

布林過濾器將位陣列全部初始化為零。要將模式添加到布林過濾器，依次由每個雜湊函數雜湊。將第一個雜湊函數應用於輸入會產生一個介於1和N之間的數字。找到陣列中的相應位（從1到N編號）並設置為1，從而記錄雜湊函數的輸出。然後，下一個雜湊函數被用來設置另一個位等等。應用了所有M個雜湊函數之後，搜索模式將在布林過濾器中被“記錄”為從0 變為1 的M個位。

<<布林2>> 是向 <<布林1>> 中所示的簡單布林過濾器添加模式“A”的示例。

添加第二個模式與重複此過程一樣簡單。該模式依次由每個雜湊函數進行雜湊，並通過對應的位設置為1 來記錄結果。請注意，由於布林過濾器填充了更多模式，因此雜湊函數結果可能與已設置為1 的位重合，在這種情況下該位不會更改。本質上，隨著更多模式記錄重疊位，布林過濾器開始變得飽和，更多位設置為1，濾波器的準確性降低。這就是為什麼過濾器是一個概率資料結構——隨著更多模式的添加，它變得不太準確。精確度取決於所添加的模式的數量與位陣列（N）的大小和雜湊函數（M）的數量。更大的位陣列和更多的雜湊函數可以以更高的準確度記錄更多的模式。較小的位陣列或更少的雜湊函數將記錄較少的模式併產生較低的準確性。

[[布林2]] .Adding a pattern "A" to our simple 布林 filter image::images/mpc2\_0809.png["布林2"]

<<布林3>> 是向簡單布林過濾器添加第二個模式“B”的示例。

[[布林3]]

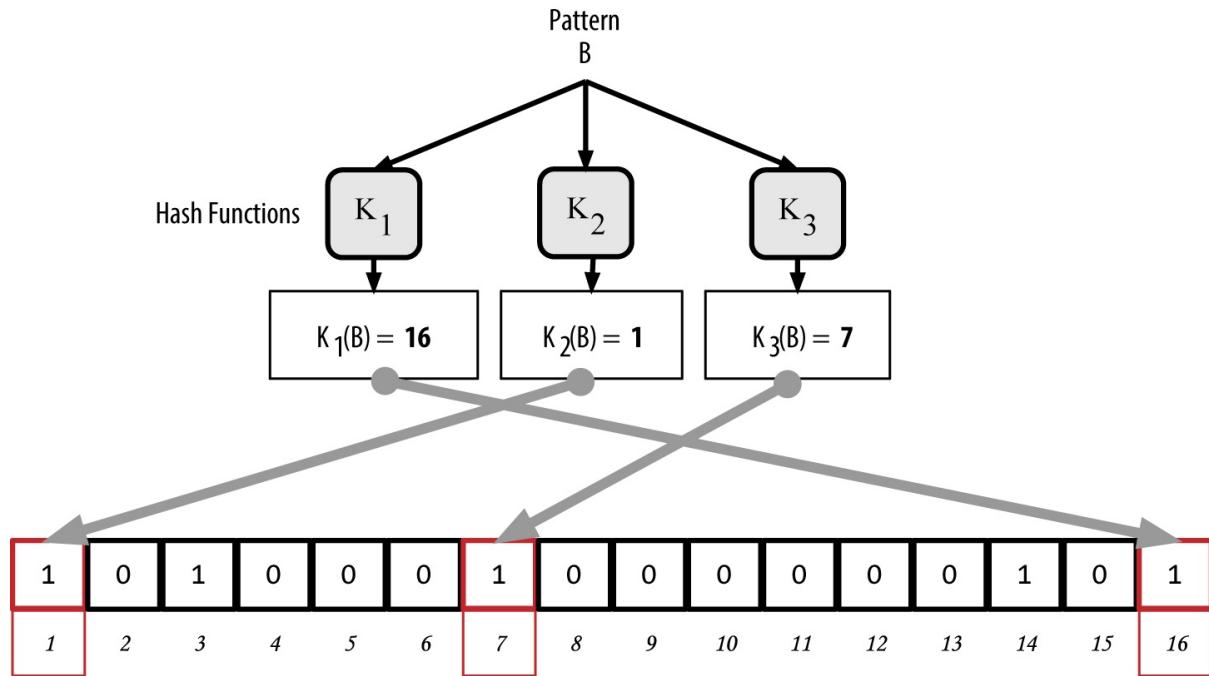


Figure 8. Adding a second pattern "B" to our simple 布林 filter

為了測試一個模式是否是布林過濾器的一部分，使用每個雜湊函數對模式進行雜湊處理，並根據比特陣列測試最終的位模式。如果由雜湊函數索引的所有位被設置為 1，則該模式可能在布林過濾器中記錄。因為這些比特可能因為多重模式的重疊而被設置，所以答案不確定，而是相當可能的。簡而言之，布林 Filter正面匹配是“可能是”。

<<布林4>> 是在簡單布林過濾器中測試模式“X”的存在的示例。相應的位被設置為 1，所以模式可能是匹配的。

[[布林4]]

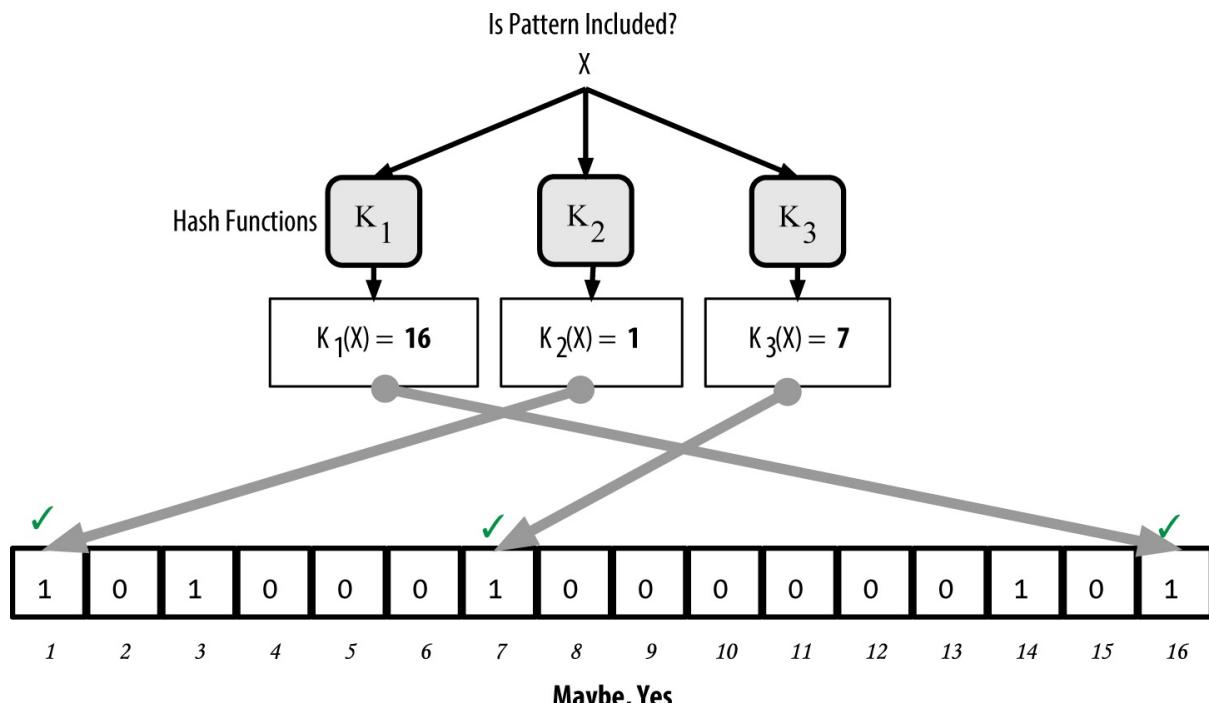


Figure 9. Testing the existence of pattern "X" in the 布林 filter. The result is a probabilistic positive match, meaning "Maybe."

相反，如果模式針對布林過濾器進行測試，並且任意一個比特設置為 0，則這證明該模式沒有記錄在布林過濾器中。否定的結果不是概率，而是肯定的。簡而言之，布林過濾器上的負面匹配是“絕對不是！”

<<布林5>> 是在簡單布林過濾器中測試模式“Y”的存在的一個例子。其中一個相應的位設置為 0，因此該模式絕對不匹配。

[[布林5]] .Testing the existence of pattern "Y" in the 布林 filter. The result is a definitive negative match, meaning "Definitely Not!" image::images/mpc2\_0812.png[]

## SPV節點如何使用布林過濾器

布林過濾器用於過濾SPV節點從其對等節點接收的交易（以及包含它們的區塊），僅選擇SPV節點感興趣的交易而不透露其感興趣的地址或密鑰。

SPV節點會將布林過濾器初始化為“空”；在該狀態下，布林過濾器將不匹配任何模式。然後，SPV節點將列出它感興趣的所有地址，密鑰和雜湊。它將通過從其錢包控制的任何UTXO中提取公共密鑰雜湊和腳本雜湊和交易ID來完成此操作。然後，SPV節點將這些模式中的每一個添加到布林過濾器，如果這些模式存在於交易中，布林過濾器將“匹配”，而不顯示模式本身。

SPV節點將向對等節點發送 filterload 訊息，其中包含要在連接上使用的布林過濾器。在對等節點中，布林過濾器將針對每個傳入交易進行檢查。完整節點根據布林過濾器檢查交易的多個部分，查找包含以下內容的匹配項：

- 交易ID
- 交易的每個輸出（腳本中的每個密鑰和雜湊）的鎖定腳本數據部分
- 每個交易輸入
- 每個輸入簽名數據部分（或見證腳本）

通過檢查所有這些組件，布林過濾器可用於匹配公鑰雜湊，腳本，OP\_RETURN 值，簽名中的公鑰或智能合約或複雜腳本的任何未來組件。

在建立過濾器後，對等節點將用布林過濾器測試每個交易的輸出。只有匹配過濾器的交易才會發送到節點。

為響應來自節點的 getdata 訊息，對等節點將發送 merkleblock 訊息，其中每個匹配交易僅包含與過濾器和merkle路徑匹配的區塊的頭部（請參見 [\[merkle\\_trees\]](#)）。對等節點隨後還會發送包含由過濾器匹配的交易的 tx 訊息。

當完整節點向SPV節點發送交易時，SPV節點丟棄所有誤報，並使用正確匹配的交易更新其UTXO集和錢包餘額。當它更新自己的UTXO集合時，它也修改布林過濾器以匹配任何引用它剛剛找到的UTXO的未來交易。完整的節點然後使用新的布林過濾器來匹配新的交易並重復整個過程。

通過發送 filteradd 訊息，設置布林過濾器的節點可以交互式地向過濾器添加模式。要清空布林過濾器，節點可以發送 filterclear 訊息。由於無法從布林過濾器中刪除模式，因此如果不再需要模式，節點必須清空並重新發送新的布林過濾器。

SPV節點的網路協議和布林過濾器機制在 [BIP-37 \(Peer Services\)](#) 中定義。

## SPV節點和隱私

實現SPV的節點比完整節點的隱私性更弱。一個完整節點接收所有交易，因此不會顯示它是否在錢包中使用某個地址。SPV節點接收與其錢包中的地址相關的過濾列表。因此，它降低了所有者的隱私。

布林過濾器是一種減少隱私損失的方法。沒有它們，SPV節點將不得不明確列出它感興趣的地址，從而嚴重暴露隱私。然而，即使使用布林過濾器，監控SPV客戶端的流量或直接作為P2P網路中的節點連接到它的對等節點，也可以收集足夠的訊息來學習SPV客戶端的錢包中的地址。

## 加密和認證的連接

大多數比特幣的新用戶都假定比特幣節點的網路通信是加密的。事實上，比特幣的原始實施完全是不加密的。雖然這不是完整節點的主要隱私問題，但對於SPV節點來說是一個大問題。

作為增加比特幣P2P網路隱私和安全性的一種方法，有兩種解決方案可以提供通信加密：*Tor Transport* (BIP-150) 和 *P2P認證與加密* (BIP-151)。

## Tor傳輸

Tor 代表 洋蔥路由網路 *The Onion Routing network*，是一個軟體項目，也是一種網路，通過具有匿名性，不可追蹤性和隱私性的隨機網路路徑，來提供數據加密和封裝。

比特幣核心提供了幾個配置選項，允許你運行比特幣節點，通過Tor網路傳輸流量。此外，Bitcoin Core還可以提供Tor隱藏服務，允許其他Tor節點直接通過Tor連接到你的節點。

從Bitcoin Core 0.12開始，如果節點能夠連接到本地的Tor服務，它將自動提供Tor隱藏服務。如果你安裝了Tor並且Bitcoin Core進程作為具有訪問Tor認證cookie權限的用戶運行，則它應該自動運行。使用 debug 標誌打開比特幣核心的Tor服務調試，如下所示：

```
$ bitcoind --daemon --debug=tor
```

你應該在日誌中看到 "tor: ADD\_ONION successful"，表明Bitcoin Core已經為Tor網路添加了隱藏服務。

你可以在Bitcoin Core文件（*docs/tor.md*）和各種在線教程中找到關於將Bitcoin Core作為Tor隱藏服務運行的更多說明。

## 點對點認證和加密 Peer-to-Peer Authentication and Encryption

兩項比特幣改進建議，BIP-150和BIP-151，增加了對比特幣P2P網路中P2P認證和支持。這兩個BIP定義了可能由兼容的比特幣節點提供的可選服務。BIP-151為兩個支持BIP-151的節點之間的所有通信啟用協商加密。BIP-150提供可選的對等身份驗證，允許節點使用ECDSA和私鑰對彼此的身份進行身份驗證。BIP-150要求在驗證之前，兩個節點按照BIP-151建立了加密通信。

截至2017年1月，BIP-150和BIP-151未在Bitcoin Core中實施。這兩個提案已經至少由一個名為bcoin的替代比特幣客戶端實施。

BIP-150和BIP-151允許用戶使用加密和身份驗證來運行連接到可信完整節點的SPV客戶端，以保護SPV客戶端的隱私。

此外，身份驗證可用於創建可信的比特幣節點網路並防止中間人攻擊（Man-in-the-Middle attacks）。最後，如果廣泛部署P2P加密，將會加強比特幣對流量分析和隱私侵蝕監控的阻力，特別是在互聯網使用受到嚴格控制和監控的極權主義國家。

標準定義在 [BIP-150 \(Peer Authentication\)](#) 和 [BIP-151 \(Peer-to-Peer Communication Encryption\)](#) 中。

## 交易池

幾乎比特幣網路上的每個節點都維護一個名為 *memory pool*，*mempool*或*transaction pool* 的未確認交易的臨時列表。節點使用該池來跟蹤網路已知但尚未包含在區塊鏈中的交易。例如，錢包節點將使用交易池來追蹤已經在網路上接收但尚未確認的到用戶錢包的傳入支付。

交易被接收和驗證後，會被添加到交易池並被中繼到相鄰節點以在網路上傳播。

一些節點實現還維護一個單獨的孤兒交易池。如果交易的投入引用尚未知曉的交易，好像遺失了父母，那麼孤兒交易將臨時儲存在孤兒池中，直至父交易到達。

將交易添加到交易池時，將檢查孤兒交易池是否有任何引用此交易輸出的孤兒（後續交易）。然後驗證任何匹配的孤兒。如果有效，它們將從孤兒交易池中刪除並添加到交易池中，從而完成從父交易開始的鏈。鑑於不再是孤兒的新增交易，該過程重複遞歸地尋找更多後代，直到找不到更多的後代。通過這個過程，父交易的到來觸發了整個鏈條相互依賴的交易的級聯重建，將孤兒與他們的父母重新整合在一起。

交易池和孤兒交易池都儲存在本地內存中，不會保存在持久性儲存上；而且，它們是從傳入的網路訊息動態填充的。當一個節點啟動時，這兩個池都是空的，並且會逐漸使用網路上收到的新交易填充。

比特幣客戶端的一些實現還維護UTXO資料庫或池，這是區塊鏈上所有未使用輸出的集合。儘管名稱“UTXO池”聽起來與交易池相似，但它代表了一組不同的數據。與交易和孤兒交易池不同，UTXO池並未初始化為空，而是包含了追溯到創世區塊的，數百萬未使用的交易輸出條目。UTXO池可以放置在本地內存中，也可以作為持久儲存上的索引資料庫表。

交易池和孤兒交易池代表單個節點的本地視角，根據節點啟動或重新啟動的時間不同，節點之間可能會有很大差異；UTXO池表示網路的自發共識，因此節點之間的差異很小。此外，交易池和孤兒交易池只包含未確認的交易，而UTXO池只包含確認的輸出。

# 區塊鏈

## 簡介

區塊鏈的資料結構是有序的，向前鏈接的區塊和交易的列表。區塊鏈可以儲存為一個扁平的檔案，或者簡單的資料庫。Bitcoin Core客戶端使用Google的LevelDB資料庫儲存區塊鏈的元數據。區塊向前鏈接，每個區塊都指向它的前一個區塊。區塊鏈經常被可視化為垂直的堆疊，第一個區塊作為堆疊底，其他的依次向上堆疊。彼此堆疊的區塊堆疊形式引出了 "height" 的術語，來代表區塊與第一個區塊的距離，"top" 和 "tip" 指代最新添加的區塊。

區塊鏈中的每個區塊都由一個雜湊值標識，在該區塊頭上使用SHA256加密雜湊演算法生成。每個區塊還通過區塊頭中的 "previous block hash"（上一個區塊的雜湊值）欄位引用先前的區塊，稱為 *parent* 區塊。換句話說，每個區塊在其自己的頭部中包含其父區塊的雜湊值。雜湊值序列創建了一個鏈，將每個區塊連接到其父項，一直鏈接到有第一個區塊，稱為 創世區塊 *genesis block*。

儘管一個區塊只有一個父區塊，但它可以暫時擁有多個子區塊。每個子區塊都指向相同的區塊作為它們的父區塊，並在 "previous block hash" 欄位中包含相同的（父區塊的）雜湊。在區塊鏈“分叉”期間會出現多個子區塊，這是一種臨時情況，當不同的礦區幾乎同時由不同的礦工發現時（參見 [\[forks\]](#)）。最終，只有一個子區塊成為區塊鏈的一部份，“fork”就解決了。一個區塊可能有多個子區塊，但每個區塊只有一個父區塊。這是因為一個區塊只有一個引用其單親的 "previous block hash" 欄位。

"previous block hash" 欄位在區塊的頭部，並且影響當前區塊的雜湊值。如果父區塊的標識改變，子區塊的標識也會改變。當父區塊以某種方式改變，父區塊的雜湊值就會改變。父區塊變化的雜湊值要求子區塊的 "previous block hash" 也必須改變，這又會引起孫子區塊的改變，以此類推。這種級聯效應可以確保一旦一個區塊在其之後有許多後代，它就不能在不強制重新計算所有後續區塊的情況下進行更改。由於重新計算需要大量的計算（耗費大量的能源），長鏈區塊的存在使得區塊鏈的歷史不可更改，這是比特幣安全性的一個關鍵特徵。

看待區塊鏈的一種方式就像地質構造中的層或冰川巖芯樣本。表層可能隨著季節變化，甚至在沉澱之前被吹走。但是一旦深入幾英寸，地質層就變得越來越穩定。當你向下看幾百英尺時，你會看到幾百萬年來一直沒有受到干擾的過去的快照。在區塊鏈中，如果由於分叉導致鏈重新計算，可能會修改最近的幾個區塊。前六個區塊就像是幾英寸的表土。一旦你深入區塊鏈超過六個區塊，區塊變化的可能性就越來越小。在100個區塊之前，穩定性非常高，以至於可以花費 coinbase 交易 - 包含新開採的比特幣的交易。幾千個區塊（一個月），對於所有實際目的來說，區塊鏈都已成確定的歷史。雖然協議總是允許一條鏈被一條較長的鏈消除，任何區塊被反轉的可能性總是存在的，但是這種事件的可能性會隨著時間流逝而減少，直到它變得無限小。

## 區塊的結構

區塊是一個容器資料結構，用於彙總包含在公共賬本（區塊鏈）中的交易。區塊有一個包含元數據的頭部，後面跟著一個長長的交易列表。區塊頭為80字節，平均交易至少為400字節，平均每區塊包含超過1900個交易。一個包含所有交易的完整區塊因此比區塊頭大10,000倍。[The structure of a block](#) 描述了一個區塊的結構。

Table 1. The structure of a block

Size	Field	Description
4 bytes	Block Size	The size of the block, in bytes, following this field
80 bytes	Block Header	Several fields form the block header
1—9 bytes (VarInt)	Transaction Counter	How many transactions follow
Variable	Transactions	The transactions recorded in this block

## 區塊頭

區塊頭由三組區塊元數據組成。首先，有一個對前區塊雜湊值的引用，它將這個區塊連接到區塊鏈中的前一個區塊。第二組元數據，分別為 難度 *difficulty*，時間戳 *timestamp* 和 隨機數 *nonce*，與挖礦競賽有關，參見 [\[mining\]](#)。第三個元數據是merkle樹根，這是一種資料結構，用於有效地彙總區塊中的所有交易。[The structure of the block header](#) 描述了區塊頭的結構。

Table 2. The structure of the block header

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 bytes	Difficulty Target	The Proof-of-Work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the Proof-of-Work algorithm

隨機數，難度目標，和時間戳用於挖礦過程，在 [\[mining\]](#) 中有詳細介紹。

## 區塊標識符：區塊頭的雜湊值和區塊高度

區塊的主要標識符是它的加密雜湊值，這是一種數字指紋，通過SHA256演算法將區塊頭兩次雜湊獲得。得到的32字節雜湊被稱為 *block hash*，更準確地說是 *block header hash*，因為只有區塊頭用於計算。例如 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f 是第一個區塊的雜湊值，區塊的雜湊值唯一而明確地標識一個區塊，任何節點都可以通過簡單地對區塊頭進行雜湊來獨立地派生它。

注意，區塊的雜湊值實際上並不包含在區塊的資料結構中，無論是在網路上傳輸區塊時，還是作為區塊鏈的一部分儲存在節點的持久性儲存時。相反，當從網路接收區塊時，每個節點計算區塊的雜湊值。區塊雜湊值可以作為區塊元數據的一部分儲存在單獨的資料庫表中，以方便索引並從硬碟快速檢索。

另一種識別區塊的方法是它在區塊鏈中的位置，稱為 區塊高度 *block height*。第一個區塊位於區塊高度0處，與被雜湊值 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f 引用的區塊相同。區塊可以通過兩種方式標識：通過引用區塊的雜湊值或通過引用區塊高度。在第一個區塊的“頂部”添加的每個後續區塊在區塊鏈中都在一個“更高”的位置，就像一個盒子疊在另一個上面一樣。2017年1月1日的區塊鏈高度約為44.6萬，這意味著在2009年1月創建的第一個區塊的頂部有446000個區塊。

與區塊雜湊值不同，區塊高度不是唯一標識符。儘管單個區塊總是具有特定且不變的區塊高度，但反過來並不正確——區塊高度並不總是標識一個區塊。兩個或更多區塊可能具有相同的區塊高度，爭奪區塊鏈中的相同位置。這種情況將在 [\[forks\]](#) 一節中詳細討論。區塊高度也不是區塊的資料結構的一部分；它不儲存在區塊中。當從比特幣網路收到區塊時，每個節點都會動態識別區塊在區塊鏈中的位置（高度）。區塊高度也可以作為元數據儲存在索引資料庫表中以加快檢索速度。

Tip 區塊的 *block hash* 總是唯一標識一個區塊。區塊也總是有一個特定的 *block height*。但是，特定的區塊高度並不總是能夠標識單個區塊。相反，兩區塊或更多區塊可能會在區塊鏈中爭奪一個位置。

## 創世區塊

區塊鏈中的第一個區塊被稱為創世區塊，於2009年創建。它是區塊鏈中所有區塊的共同祖先，這意味著如果你從任何區塊開始，並隨時間上向前追溯，最終將到達創世區塊。

節點總是以至少一個區塊的區塊鏈開始，因為這個區塊是在比特幣客戶端軟體中靜態編碼的，因此它不能被改變。每個節點總是“知道”起始區塊的雜湊和結構，它創建的固定時間，以及其中的單一交易。因此，每個節點都有區塊鏈的起點，這是一個安全的“根”，從中可以構建受信任的區塊鏈。

請參閱 [chainparams.cpp](#) 中 Bitcoin Core 客戶端內的靜態編碼的genesis區塊。

以下雜湊值標識符屬於創世區塊：

```
00000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

你可以在任何區塊瀏覽器網站（例如 [blockchain.info](#)）中搜索該區塊雜湊值，你將找到一個描述此區塊內容的頁面，包含該雜湊的URL：

<https://blockchain.info/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f>

在命令行中使用 Bitcoin Core 客戶端：

```
$ bitcoin-cli getblock
00000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

```
{
  "hash" : "000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
  "confirmations" : 308321,
  "size" : 285,
  "height" : 0,
  "version" : 1,
  "merkleroot" : "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",
  "tx" : [
    "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
  ],
  "time" : 1231006505,
  "nonce" : 2083236893,
  "bits" : "1d00ffff",
  "difficulty" : 1.00000000,
  "nextblockhash" : "00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048"
}
```

創世區塊包含一個隱藏的訊息。幣基交易的輸入包含的文字是 "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks."。此訊息旨在通過參考英國報紙 *The Times* 的標題提供此區塊創建的最早日期的證據。它還半開玩笑地提醒人們關注獨立貨幣體系的重要性，比特幣發行時正值前所未有的全球貨幣危機。比特幣的創造者Satoshi Nakamoto將這個訊息嵌入了第一區塊。

## 在區塊鏈中鏈接區塊

比特幣完整節點保留了區塊鏈從創世區塊開始的本地副本。區塊鏈的本地副本會隨著新區塊被發現並用於擴展鏈而不斷更新。當一個節點通過網路接收到區塊時，它驗證這些區塊，然後將它們鏈接到現有的區塊鏈。為了建立鏈接，節點將檢查傳入的區塊頭並查找“previous block hash”。

例如，假設一個節點在區塊鏈的本地副本中有277,314個區塊。節點知道的最後一個區塊是區塊277,314，區塊頭的雜湊值為：

```
000000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249
```

然後，該節點從網路接收一個新的區塊，解析如下：

```
{
  "size" : 43560,
  "version" : 2,
  "previousblockhash" :
    "000000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249",
  "merkleroot" :
    "5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1155e29d",
  "time" : 1388185038,
  "difficulty" : 1180923195.25802612,
  "nonce" : 4215469401,
  "tx" : [
    "257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7cf3f6c77",
    #[... many more transactions omitted ...]
    "05cf38f6ae6aa83674cc99e4d75a1458c165b7ab84725eda41d018a09176634"
  ]
}
```

查看這個新區塊，找到 `previousblockhash` 欄位，其中包含其父區塊的雜湊值。它是節點所知道的雜湊值，是位於鏈的277,314高度的最後一個區塊。因此，這個新區塊是鏈上最後一個區塊的孩子，並擴展了現有的區塊鏈。該節點將此新區塊添加到鏈的末端，使區塊鏈更長，新的高度為277,315。[Blocks linked in a chain by reference to the previous block header hash](#) 顯示三個區塊的鏈，通過 `previousblockhash` 欄位中的引用鏈接。

## 默克爾樹 Merkle Trees

比特幣區塊鏈中的每個區塊都包含一個 *merkle tree*，作為所有交易的彙總。

默克爾樹 *merkle tree*, 也叫做 二元雜湊樹 *binary hash tree*, 是一種用於有效彙總和驗證大型數據集的完整性的資料結構。Merkle樹是包含加密雜湊的二元樹。術語“樹”在電腦科學中被用來描述分支的資料結構，但是這些樹通常是顛倒顯示的，“根”在頂部，“葉子”在底部，你將在下面的例子中看到。

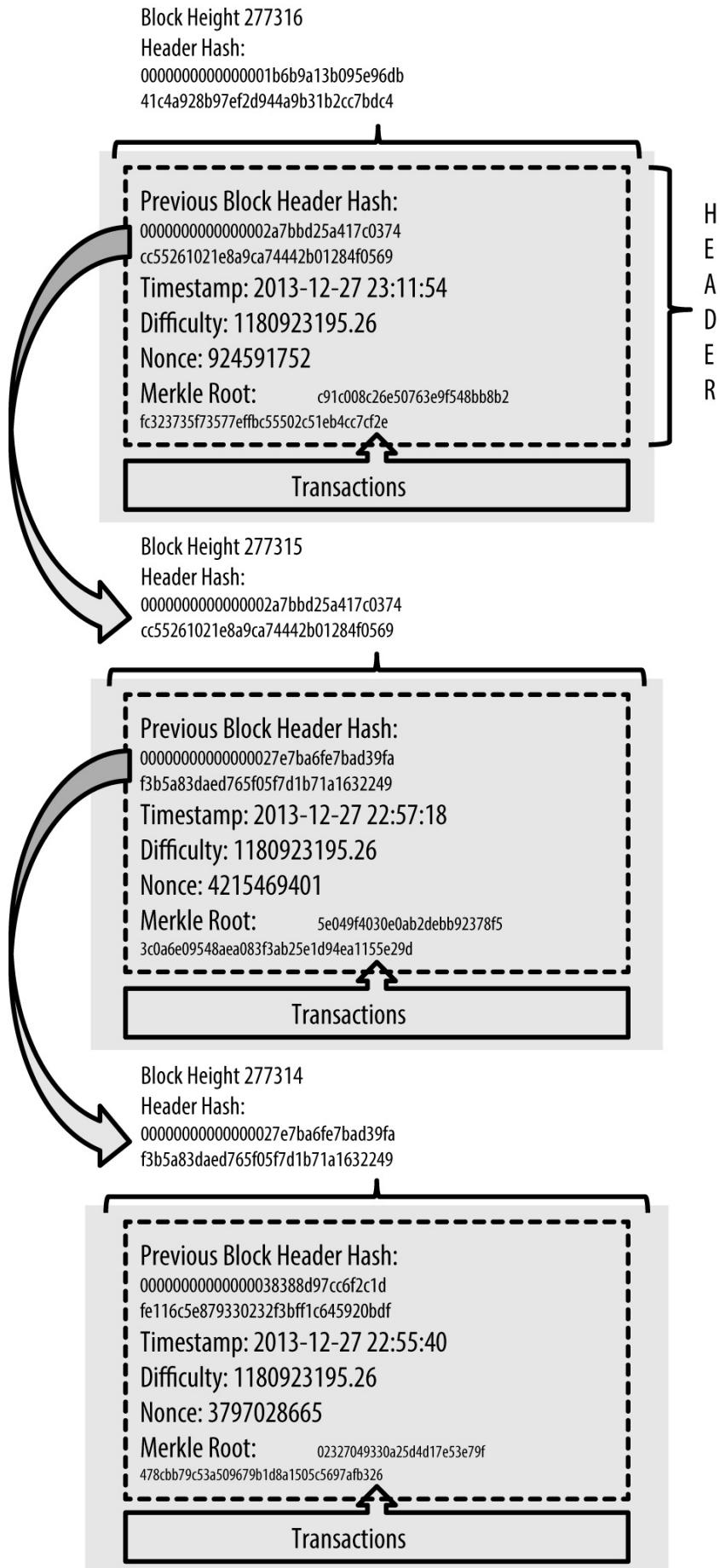


Figure 1. Blocks linked in a chain by reference to the previous block header hash

比特幣中使用Merkle樹來彙總區塊中的所有交易，為整個交易集提供全面的數字指紋，提供了一個非常有效的流程來驗證交易是否包含在區塊中。Merkle樹是通過對節點對兒（pairs of nodes）進行遞歸雜湊構造的，直到只有一個雜湊，稱為 *root* 或 *merkle root*。比特幣的merkle樹中使用的加密雜湊演算法是將SHA256應用兩次，也稱為double-SHA256。

當N個數據元素被雜湊並彙總到一個Merkle樹中時，你可以檢查樹中是否包含任何一個數據元素，並且最多隻需 $2^{\log_2 N}$ 次計算，因此這是一個非常有效的資料結構。

Merkle樹是自下而上構建的。在下面的例子中，我們從四個交易開始，A，B，C和D，它們構成了merkle樹的葉子 *leaves*，如 [Calculating the nodes in a merkle tree](#) 所示。交易不儲存在merkle樹中；相反，它們的數據被雜湊並且所得的雜湊值被儲存在每個葉節點中，如  $H_A$ ,  $H_B$ ,  $H_C$  和  $H_D$ ：

$$H_A = \text{SHA256}(\text{SHA256}(\text{Transaction A}))$$

然後將連續的葉節點對彙總到父節點中，方法是連接兩個雜湊值並對它們進行雜湊。例如，要構造父節點  $H_{AB}$ ，將子節點的兩個32字節雜湊值連接起來，以創建一個64字節的字串。然後對該字串進行雙重雜湊來產生父節點的雜湊值：

$$H_{AB} = \text{SHA256}(\text{SHA256}(H_A + H_B))$$

繼續該過程，直到頂部只有一個節點，該節點被稱為merkle根。該32字節雜湊值儲存在區塊頭中，彙總了四個交易中的所有數據。[Calculating the nodes in a merkle tree](#) 展示瞭如何通過節點的成對雜湊來計算根。

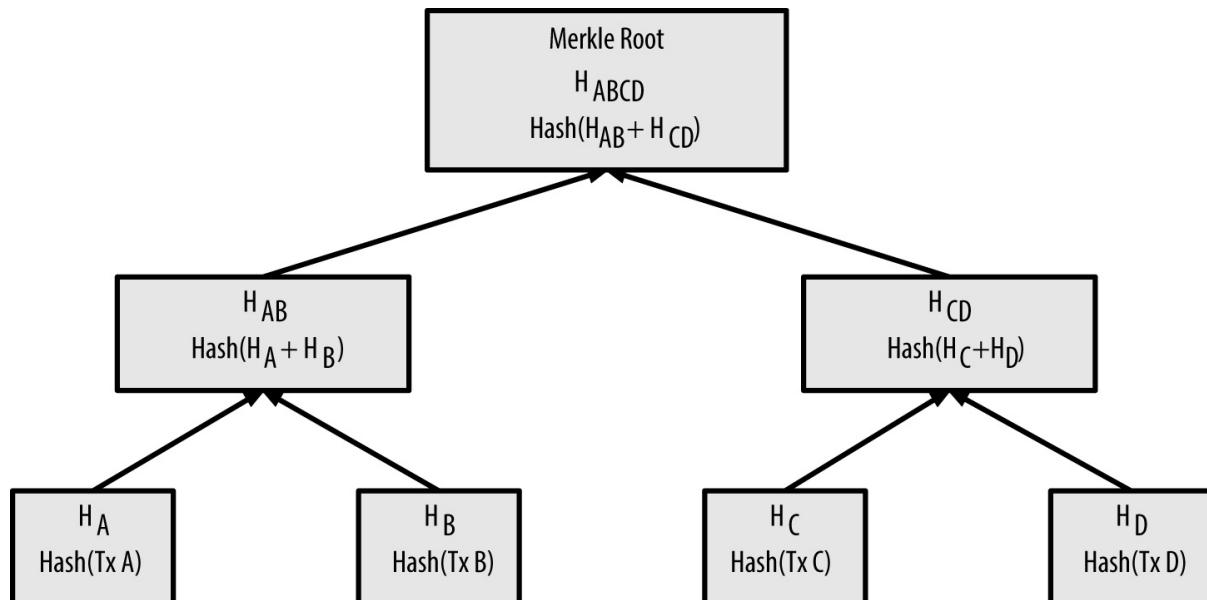


Figure 2. Calculating the nodes in a merkle tree

由於merkle樹是二元樹，它需要偶數個葉節點。如果要彙總的交易數量為奇數，則最後一個交易的雜湊值將被複制以創建偶數個葉節點，這稱為 平衡的樹 *balanced tree*。在 << merkle\_tree\_odd>> 中，交易C被複制。

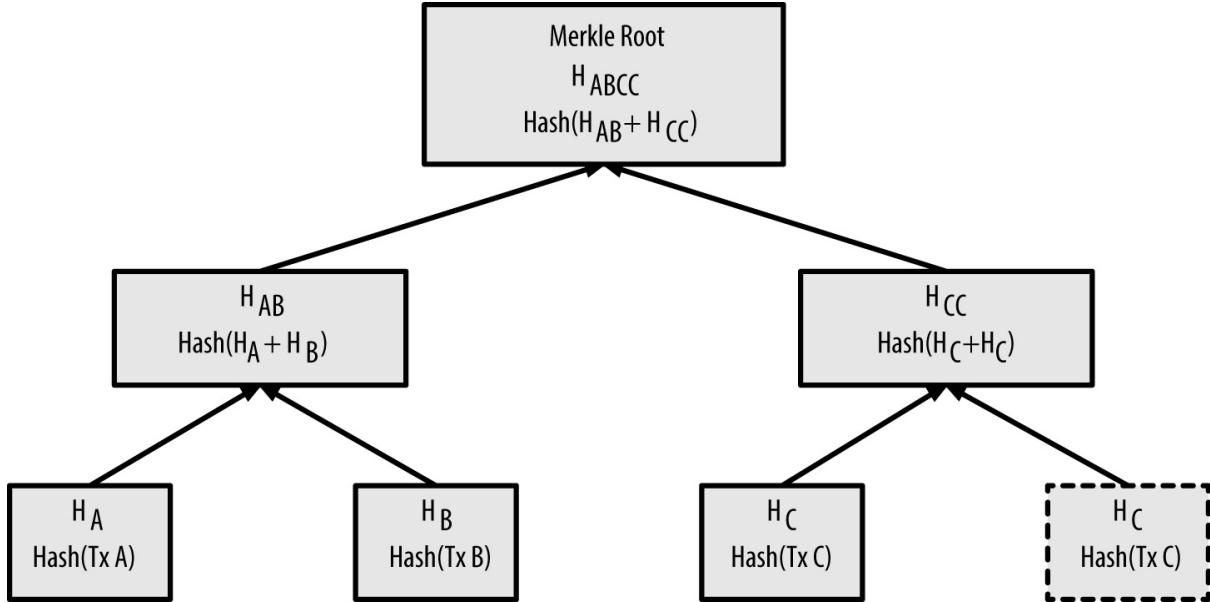


Figure 3. Duplicating one data element achieves an even number of data elements

使用四個交易構造樹的方法可以推廣到構造任意大小的樹。在比特幣中，通常在一個區塊中有幾百到幾千個交易，這些交易的彙總方式完全相同，僅產生單個Merkle根的32個字節的數據。在 [A merkle tree summarizing many data elements](#) 中，你將看到一棵由16個交易構成的樹。請注意，儘管根看起來比圖中的葉節點大，但它的大小完全相同，只有32個字節。無論區塊中是否有一個交易或十萬個交易，merkle根總是將它們總結為32個字節。

為了證明一個區塊中包含一個特定的交易，一個節點只需要產生  $\log_2(N)$  個32個字節的雜湊值，構成一個認證 *path* 或 *merkle\_path*，將特定的交易連接到樹的根。隨著交易數量的增加，這一點尤為重要，因為交易數量的基數為2的對數增長速度要慢得多。這使得比特幣節點能夠高效地生成10或12個雜湊值（320-384字節）的路徑，這可以提供兆字節大小的區塊中超過一千個交易中的單個交易的驗證。

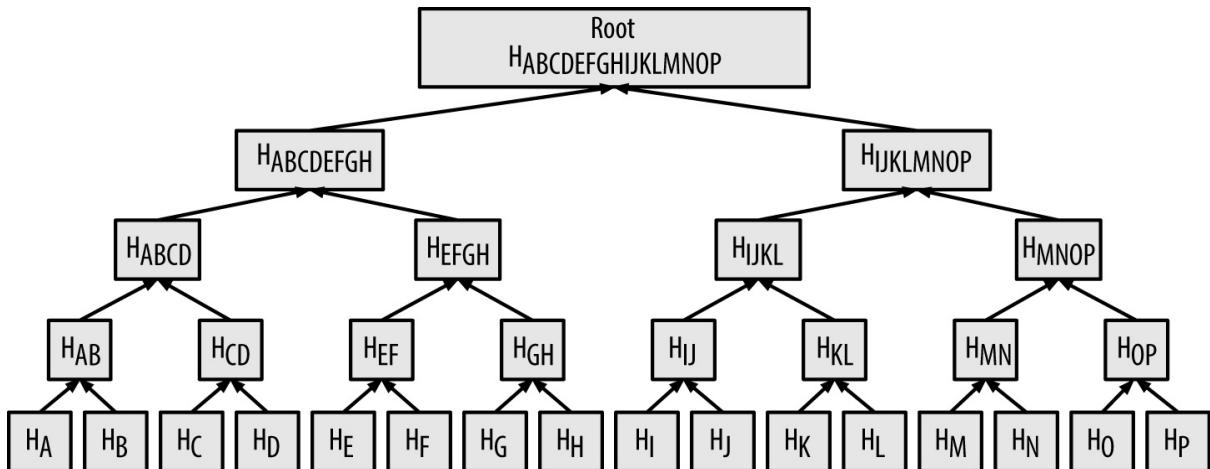


Figure 4. A merkle tree summarizing many data elements

在 [A merkle path used to prove inclusion of a data element](#) 中，節點可以通過產生只有四個32字節雜湊長（總共128字節）的merkle路徑來證明交易K包含在該區塊中。該路徑由四個雜湊值組成（在 <> merkle\_tree\_path >> 帶藍色背景的）， $H_L$ ,  $H_{IJ}$ ,  $H_{MNOP}$  和  $H_{ABCDEFGHIJKLMNP}$ 。通過提供這四個雜湊值作為驗證路徑，任何節點都可以通過計算四個額外的雜湊值來證明  $H_K$ （底部黑色背景的）包含在Merkle根中： $H_{KL}$ ,  $H_{IJKL}$ ,  $H_{IJKLMNOP}$  和merkle樹根（在圖中用虛線表示）。

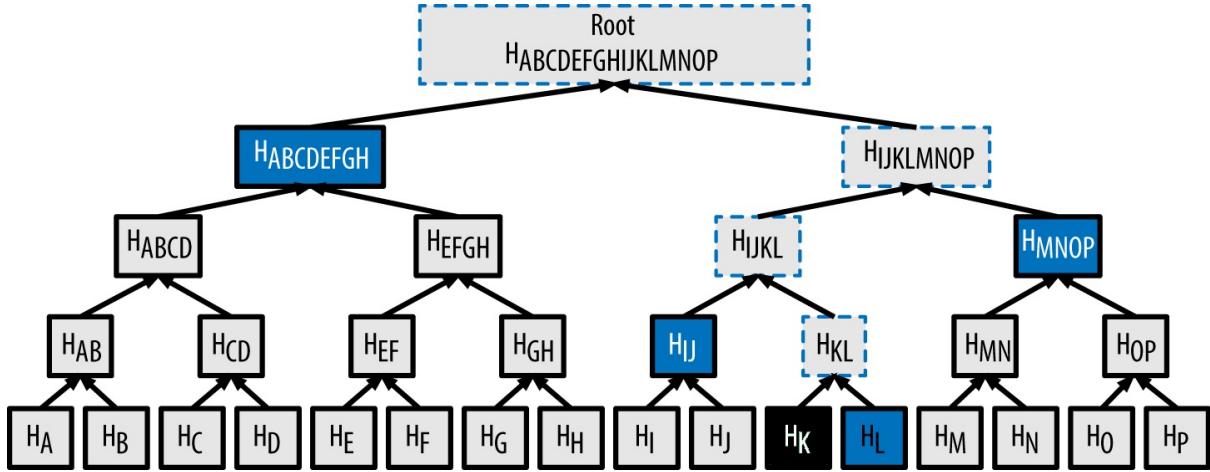


Figure 5. A merkle path used to prove inclusion of a data element

[Building a merkle tree](#) 中的程式碼演示瞭如何使用libbitcoin的一些輔助函數，創建從葉節點雜湊值一直到根的Merkle樹。

#### Example 1. Building a merkle tree

```
link:code/merkle.cpp[]
```

[Compiling and running the merkle example code](#) 展示了編譯和運行結果

#### Example 2. Compiling and running the merkle example code

```
$ # Compile the merkle.cpp code
$ g++ -o merkle merkle.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Run the merkle executable
$ ./merkle
Current merkle hash list:
32650049a0418e4380db0af81788635d8b65424d397170b8499cdc28c4d27006
30861db96905c8dc8b99398ca1cd5bd5b84ac3264a4e1b3e65afa1bcee7540c4

Current merkle hash list:
d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3

Result: d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3
```

隨著規模的增加，梅克爾樹的效率變得越來越明顯。[Merkle tree efficiency](#) 展示了證明交易是區塊的一部分鎖需要的作為merkle路徑交換的數據量。

Table 3. Merkle tree efficiency

Number of transactions	Approx. size of block	Path size (hashes)	Path size (bytes)
16 transactions	4 kilobytes	4 hashes	128 bytes
512 transactions	128 kilobytes	9 hashes	288 bytes
2048 transactions	512 kilobytes	11 hashes	352 bytes
65,535 transactions	16 megabytes	16 hashes	512 bytes

從表中可以看出，區塊大小從16個交易的4KB快速增加到65,535個交易的16MB，證明交易存在所需的Merkle路徑則增加得很慢，從128字節到只有512字節。使用merkle樹，節點可以只下載區塊頭（每區塊80個字節），並且仍然能夠通過從完整節點檢索小型merkle路徑來證實交易包含在區塊中，而不儲存或傳輸絕大量的（可能幾個GB）區塊鏈數據。不維護完整區塊鏈的節點稱為簡單支付驗證（SPV）節點，它使用merkle路徑驗證交易而無需下載完整區塊。

## Merkle 樹和簡單支付驗證節點

Merkle樹被SPV節點廣泛使用。 SPV節點沒有全部交易，並且不下載完整的區塊，只有區塊頭。為了驗證區塊中包含交易，而不必下載區塊中的所有交易，它們使用驗證路徑（merkle路徑）。

例如，考慮一個SPV節點，它對付款到它的錢包中地址的交易感興趣。SPV節點將在其與對等節點的連接上建立一個布林過濾器（參見 [\[bloom\\_filters\]](#)），將接收到的交易限制為那些只包含其感興趣地址的交易。當對等節點看到與bloom過濾器匹配的交易時，它將使用 merkleblock 訊息發送該區塊。merkleblock 訊息包含區塊的頭，以及將感興趣的交易鏈接到區塊中的merkle根的merkle路徑。SPV節點可以使用此Merkle路徑將交易連接到區塊並驗證交易是否包含在區塊中。 SPV節點還使用區塊頭將區塊鏈接到區塊鏈的其餘部分。交易和區塊之間以及區塊和區塊鏈之間的這兩個鏈接的組合證明交易記錄在區塊鏈中。總而言之，SPV節點將接收到少於一千字節的數據區塊頭和merkle路徑，其數據量比完整區塊（當前大約1兆字節）少一千倍以上。

## 比特幣的測試區塊鏈

你可能會驚訝地發現有多個比特幣區塊鏈。 2009年1月3日由中本聰創建的“主”比特幣區塊鏈，帶有我們本章研究的創世區塊，被稱為 主網 *mainnet*。還有其他用於測試的比特幣區塊鏈，現在有：*testnet*，*segnet* 和 *regtest*。讓我們依次看下。

### 測試網路 —— 比特幣的測試場

*Testnet*是用於測試目的的測試區塊鏈，網路和貨幣的名稱。測試網是一個全功能的活躍P2P網路，包括錢包，測試比特幣（*testnet*硬幣），挖礦以及 *mainnet* 的所有其他功能。實際上只有兩個區別：測試網點的硬幣價值很低，挖礦難度應該足夠低，以便任何人都可以相對容易地開採測試網硬幣。

任何擬用於比特幣主網生產的軟體開發應首先使用測試幣在測試網上進行測試。這可以保護開發人員免受由於錯誤導致的資金損失，並保護網路免受由於錯誤導致的意外行為。

然而，保持硬幣毫無價值並且容易挖掘，並不容易。儘管開發者提出了要求，但有些人使用高級採礦設備（GPU和ASIC）在測試網上進行挖掘，增加了難度，使得不可能用CPU進行挖掘，最終使其難以獲得，人們開始評估它們的價值，因此它們也不是毫無價值的。因此，現在或者之後，測試網必須被廢棄並從新的創世區塊重新啟動，重新設置難度。

當前的測試網成為 *testnet3*，第三代 *testnet*，2011年2月重啟，重置了前一代測試網的難度。

請記住，*testnet3*是一個大型的區塊鏈，2017年初超過了20GB。耗盡電腦資源完全同步需要一天左右的時間。不如 *mainnet*大，但也不是“輕量級”的。運行測試網路節點的一個好的方法是作為專用於此目的的虛擬機映像（例如，VirtualBox，Docker，雲伺服器等）。

#### 使用測試網

像幾乎所有其他比特幣軟體一樣，Bitcoin Core完全支持在*testnet*而不是*mainnet*上運行。比特幣核心的所有功能都在測試網路上運行，包括錢包，開採測試網的幣，以及同步完整的測試網節點。

要在測試網上啟動 Bitcoin Core，使用 *testnet* 選項：

```
$ bitcoind -testnet
```

在日誌中，你應該看到bitcoind正在預設bitcoind目錄的 *testnet3* 子目錄中構建新的區塊鏈：

```
bitcoind: Using data directory /home/username/.bitcoin/testnet3
```

你可以使用 *bitcoin-cli* 命令行工具連接到bitcoind，但也必須將其切換到*testnet*模式：

```
$ bitcoin-cli -testnet getblockchaininfo
{
  "chain": "test",
  "blocks": 1088,
  "headers": 139999,
  "bestblockhash":
"0000000063d29909d475a1c4ba26da64b368e56cce5d925097bf3a2084370128",
  "difficulty": 1,
  "mediantime": 1337966158,
  "verificationprogress": 0.001644065914099759,
  "chainwork": "000000000000000000000000000000000000000000000000000000000000000044104410441",
  "pruned": false,
  "softforks": [
    [...]
```

你還可以使用其他完整節點實現（如 btcd（用Go編寫）和 bcoin（用JavaScript編寫））在testnet3上運行，以便在其他程式語言和框架中進行實驗和學習。

2017年初，testnet3支持了mainnet的所有功能，包括隔離見證（參見[\[segwit\]](#)）。因此，testnet3也可以用來測試隔離見證功能。

## Segnet —— 隔離見證測試網

2016年啟動了一個特殊用途的測試網，幫助開發和測試隔離見證（又名segwit；見 [\[segwit\]](#)）。該測試區塊鏈被稱為 segnet，可以通過運行比特幣核心的特殊版本（分支）加入。

由於segwit已添加到testnet3，因此不再需要使用segnet來測試segwit功能。

未來，我們可能會看到其他像segnet一樣，專門設計用於測試單個功能或主要架構更改的testnet區塊鏈。

Regtest — 本地區塊鏈

Regtest代表“迴歸測試”，它是一種比特幣核心功能，允許你為測試目的創建本地區塊鏈。與公共測試區塊鏈testnet3不同，regtest區塊鏈旨在作為封閉系統運行以進行本地測試。你從零開始啟動一個regtest區塊鏈，創建一個本地創世區塊。你可以將其他節點添加到網路，或者僅使用單個節點運行它來測試比特幣核心軟體。

要以 `reqtest` 模式啟動 Bitcoin Core，使用 `reqtest` 選項：

```
$ bitcoind -regtest
```

與testnet一樣，Bitcoin Core會在你的bitcoind預設目錄的 `regtest` 子目錄下初始化一個新的區塊鏈：

bitcoind: Using data directory /home/username/.bitcoin/regtest

要使用命令行工具，你需要指定 `regtest` 標誌：

```
$ bitcoin-cli -regtest getblockchaininfo
{
  "chain": "regtest",
  "blocks": 0,
```

如你所見，現在還沒有區塊。讓我們挖掘一些（500區塊）並獲得獎勵：

```
$ bitcoin-cli -regtest generate 500
[
    "7afed70259f22c2bf11e406cb12ed5c0657b6e16a6477a9f8b28e2046b5ba1ca",
    "1aca2f154a80a9863a9aac4c72047a6d3f385c4eec5441a4aafa6acaa1dada14",
    "4334ecf6fb022f30fbd764c3ee778fabbd53b4a4d1950eae8a91f1f5158ed2d1",
    "5f951d34065efefaf64e54e91d00b260294fcdfc7f05dbb5599aec84b957a7766",
    "43744b5e77c1dfece9d05ab5f0e6796ebe627303163547e69e27f55d0f2b9353",
    [...]
    "6c31585a48d4fc2b3fd25521f4515b18aefb59d0def82bd9c2185c4ecb754327"
]
```

只需要幾秒鐘的時間來挖掘所有這些區塊，這使得測試很容易。如果你檢查你的錢包餘額，你會看到你獲得了前400個區塊的獎勵（coinbase獎勵必須達到到100個區塊後才可以消費）：

```
$ bitcoin-cli -regtest getbalance  
12462.50000000
```

## 使用測試區塊鏈進行開發

比特幣的各種區塊鏈（regtest，segnet，testnet3，mainnet）為比特幣開發提供了一系列測試環境。無論你是為Bitcoin Core開發還是另一個完整節點共識客戶端，都可以使用測試區塊鏈。應用程式，如錢包，交換，電子商務網站；甚至開發新穎的智能合約和複雜的腳本。

你可以使用測試區塊鏈建立開發管道。開發時，在 `regtest` 上本地測試你的程式碼。一旦準備好在公共網路上嘗試它，切換到 `testnet` 以將程式碼暴露於有多種的程式碼和應用的，更具動態性的環境中。最後，一旦你確信自己的程式碼能夠按預期工作，請切換到 `mainnet` 以在生產環境中進行部署。當你進行更改，改進，錯誤修復等時，請再次啟動管道，首先是 `regtest`，然後是 `testnet`，最後部署到生產環境中。

# 挖礦和共識

## 概述

"挖礦"一詞是有一些誤導性的。通過類比貴金屬的提取，它將我們的注意力集中在挖礦的獎勵上，這是每個區塊創建的新比特幣。儘管這種獎勵激勵了挖礦，但挖礦的主要目的不是獎勵或生成新的硬幣。如果你僅將挖礦看作是創建比特幣的過程，那麼你就錯誤地將手段（激勵）當成了過程的目標。挖礦是支撐去中心化清算所的機制，使交易得到驗證和清算。挖礦是使比特幣特別的發明，是一種去中心化的安全機制，是P2P數字現金的基礎。

新鑄造的硬幣和交易費用獎勵是一種激勵計劃，它將礦工的行為與網路的安全保持一致，同時實施貨幣供應。

Tip

挖礦的目的不是創造新的比特幣。這是激勵機制。挖礦是使比特幣的安全性 *security* 去中心化 *decentralized* 的機制。

礦工確認新的交易並將其記錄在全球總賬中。包含自上一個區塊以來發生的交易的新區塊，平均每10分鐘被"挖掘"，從而將這些交易添加到區塊鏈中。成為區塊的一部分並添加到區塊鏈中的交易被認為是"確認"的，這允許比特幣的新所有者花費他們在這些交易中收到的比特幣。

為礦工獲得兩種類型的獎勵以換取挖礦提供的安全性：每個新區塊創建的新幣以及該區塊中包含的所有交易的交易費用。為了獲得這種獎勵，礦工們競相解決基於密碼雜湊演算法的數學難題。這個難題的解決方案被稱為工作證明（Proof-of-Work），它被包含在新的區塊中，作為礦工大量計算工作的證據。解決PoW演算法贏得獎勵以及在區塊鏈上記錄交易的權利的競爭是比特幣安全模型的基礎。

這個過程稱為採礦，因為獎勵（新硬幣的產生）旨在模擬像開採貴金屬一樣的收益遞減。比特幣的貨幣供應是通過採礦創造的，類似於央行通過打印鈔票發行新貨幣的方式。大約每四年（或正好每210,000區塊），一個礦工可以添加到區塊的最大新增比特幣數量減少一半。2009年1月開始每個區塊50比特幣，2012年11月每個區塊減半到25比特幣，2016年7月再次減少到12.5比特幣。基於這個公式，比特幣挖礦獎勵指數級下降，到2140年左右，所有的比特幣（21千萬）將發行完畢。2140年以後，不會有新的比特幣發行。

比特幣礦工也從交易中賺取費用。每筆交易都可能包含一筆交易費用，費用以交易的輸入與輸出之間的盈餘形式體現。獲勝的比特幣礦工可以對包含在獲獎區塊中的交易"零錢"。今天，這筆費用佔比特幣礦工收入的0.5%或更少，絕大多數來自新鑄造的比特幣。然而，獎勵隨著時間推移而減少，每個區塊的交易數量逐漸增加，比特幣開採收入的更大比例將來自費用。逐漸地，採礦獎勵將由交易費取代，成為礦工的主要動機。2140年以後，每個區塊的新比特幣數量將降至零，比特幣開採將僅通過交易費用獲得激勵。

在本章中，我們首先將挖礦視為貨幣供應機制進行研究，然後研究挖礦最重要的功能：支持比特幣安全性的分散式共識機制。

要理解挖礦和共識，我們會跟蹤Alice的交易，它被Jing的挖礦設備接收並添加到一個區塊。然後我們將跟蹤這個區塊，它被挖掘並添加到區塊鏈，然後通過自發共識（emergent consensus）的過程被比特幣網路接受。

## 比特幣經濟學和貨幣創造

比特幣在創建每個區塊時以固定和遞減的速度被"鑄造"。平均每10分鐘產生一個包含全新的比特幣的區塊，憑空產生。每隔21萬個區塊，或大約每四年，貨幣發行速率就會下降50%。在網路運轉的前四年，每個區塊包含50個新的比特幣。

2012年11月，比特幣發行速率降至每區塊25比特幣。2016年7月，再次下降到每區塊12.5比特幣。它將在630,000區塊區塊再次減半至6.25比特幣，這將是2020年的某個時間。新硬幣的比率將按照指數規律進行32次"減半"，直到6,720,000區塊（大約在2137年開採），達到最低貨幣單位，1 satoshi。大約2140年之後，將有690萬個區塊，發行近2,099,999,997,690,000個satoshis，即將近2100萬比特幣。此後，區塊將不包含新的比特幣，礦工將僅通過交易費獲得獎勵。[Supply of bitcoin currency over time based on a geometrically decreasing issuance rate](#) 展示了隨時間推移流通的比特幣總量，貨幣發行量下降。

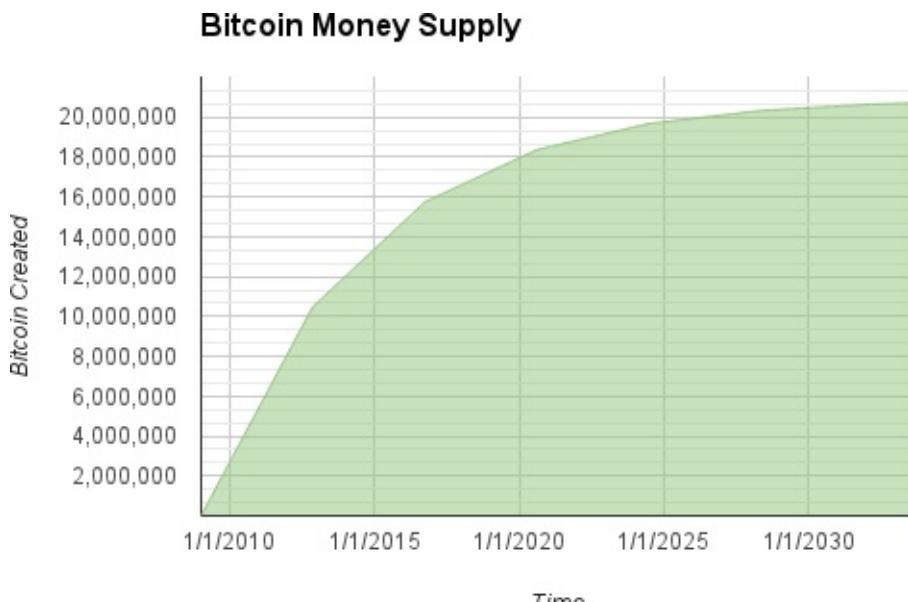


Figure 1. Supply of bitcoin currency over time based on a geometrically decreasing issuance rate

Note	開採的最大硬幣數量是比特幣可能的挖礦獎勵的上限。實際上，礦工可能會故意挖掘一個獲得的獎勵少於全部獎勵的區塊。已經有這樣的區塊，之後可能更多，這將導致貨幣供應量的減少。
------	---

在 [A script for calculating how much total bitcoin will be issued](#) 的示例程式碼中，我們計算比特幣的發行總量。

Example 1. A script for calculating how much total bitcoin will be issued

```
link:code/max_money.py[]
```

[Running the max\\_money.py script](#) 展示了運行腳本的結果

Example 2. Running the max\_money.py script

```
$ python max_money.py
Total BTC to ever be created: 2099999997690000 Satoshi
```

有限和遞減的發行，保證了固定的貨幣供應量，可以抵制通貨膨脹。不同於法定貨幣中央銀行的無限印錢，比特幣永遠不會因印錢而膨脹。

#### 通貨緊縮的貨幣

固定和遞減的貨幣發行的最重要的和有爭議的後果是，貨幣傾向於內在地 通貨緊縮。通貨緊縮是由於供求失衡導致貨幣價值（和匯率）升值的現象。與通貨膨脹相反，價格通縮意味著這些資金隨著時間的推移具有更多的購買力。

許多經濟學家認為，通貨緊縮的經濟是一場應該不惜一切代價避免的災難。這是因為在通貨緊縮時期，人們傾向於囤積錢而不是花錢，希望價格會下跌。日本“失去的十年”期間，這種現象就顯現出來了，當時需求的徹底崩潰將貨幣推向通縮螺旋。

比特幣專家認為，通貨緊縮本身並不糟糕。當然，通貨緊縮與需求崩潰有關，因為這是我們必須研究的唯一的通縮的例子。在可以無限印刷的法定貨幣下，進入通貨緊縮螺旋是非常困難的，除非需求完全崩潰並且政府不願印鈔。比特幣的通貨緊縮不是由需求崩潰引起的，而是由可預見的供應緊張造成的。

通貨緊縮的積極方面當然與通貨膨脹相反。通貨膨脹導致貨幣緩慢但不可避免的貶值，導致一種隱藏的稅收形式，為拯救債務人（包括最大的債務人，政府本身）而對儲戶進行懲罰。政府控制下的貨幣受到輕易發行債券的道德風險的影響，後者可以通過貶值而以犧牲儲蓄者為代價來消除。

當通貨緊縮不是經濟快速收縮帶來的問題時，通貨緊縮是有問題的還是有優勢的，還有待觀察，因為防止通貨膨脹和貶值的遠比通貨緊縮的風險重要。

## 去中心化共識（Decentralized Consensus）

在上一章中，我們考察了區塊鏈，即所有交易的全球公共賬本（列表），比特幣網路中的每個人都認可它作為所有權的權威記錄。

但是，網路中的每個人怎麼能夠就一個普遍的“真相”達成一致：誰擁有什麼，而不必相信任何人呢？所有傳統的支付系統都依賴於信託模式，該模式具有提供清算所服務的中央機構，驗證和清算所有交易。比特幣沒有中央權威機構，但每個完整的節點都有一個可以信任的權威記錄的公共賬本的完整副本。區塊鏈不是由中央機構創建的，而是由網路中的每個節點獨立組裝。而且，網路中的每個節點都會根據通過不安全的網路連接傳輸的訊息，得出相同的結論，並與其他人一樣收集相同的公共賬本。本章將探討比特幣網路在沒有中央權威機構的情況下達成全球共識的過程。

中本聰的主要發明是自發共識的去中心化機制。自發，是因為共識不是明確地達成的，達成共識時沒有選舉或固定的時刻。相反，共識是數千個遵循簡單規則的獨立節點，異步交互的自發性產物。比特幣的所有屬性，包括貨幣，交易，支付，以及不依賴中央機構或信任的安全模型，都源自於這項發明。

比特幣的去中心化共識來自四個獨立於網路節點的過程的相互作用：

- 每筆交易由完整節點獨立驗證，基於一份全面的標準清單
- 通過挖礦節點將交易獨立地聚合到新的區塊中，並通過PoW演算法證明計算。
- 每個節點獨立驗證新的區塊，並組裝到區塊鏈中
- 通過工作流程證明，每個節點獨立選擇具有最多累積計算量證明的鏈

在接下來的幾節中，我們將研究這些流程以及它們如何相互作用創建網路共識的自發性，以允許任何比特幣節點組裝自己的權威的、信任的、公共的全局賬本的副本。

### 獨立交易驗證

在 [transactions] 中，我們看到了錢包軟體如何通過收集UTXO創建交易，提供適當的解鎖腳本，然後構建分配給新所有者的新輸出。然後將產生的交易發送到比特幣網路中的相鄰節點，以便它可以在整個比特幣網路上傳播。

但是，在向鄰居轉發交易之前，接收交易的每個比特幣節點都將首先驗證交易。確保只有有效的交易通過網路傳播，無效的交易會被遇到它們的第一個節點丟棄。

每個節點根據一個很長的標準檢查清單驗證每筆交易：

- 交易的語法和資料結構必須正確
- 輸入和輸出列表都不為空
- 所有交易的字節大小小於 MAX\_BLOCK\_SIZE.
- 每個輸出值，和總的輸出值，都必須在允許的範圍區間（小於2100萬比特幣，大於 dust 閾值）
- 任何輸入的hash不等於0，N不等於-1 (幣基交易不應該被傳播)
- nLocktime 等於 INT\_MAX, 或者 nLocktime 和 nSequence 的值滿足 MedianTimePast 的要求
- 每筆交易的字節大小大於或等於 100
- 交易中包含的簽名操作 (SIGOPS) 小於簽名操作限制
- 解鎖腳本( scriptSig ) 只能向堆疊中壓入數值, 鎖定腳本 (scriptPubkey) 必須匹配 IsStandard 格式 (拒絕非標準的交易).
- 交易池或主分支的一個區塊中必須存在這筆交易
- 對於每個輸入，如果引用的輸出存在於池中的任意一筆其他交易中，則這筆交易被拒絕

- 對於每個輸入，查找主分支和交易池以找到引用的輸出的交易。如果任何輸入的輸出交易丟失，這將成為一筆孤兒交易。如果匹配的交易沒在池中，添加它到孤兒交易池
- 對於每個輸入，如果引用的交易是幣基輸出，它必須有至少 COINBASE\_MATURITY (100) 次確認
- 對於每個輸入，引用的輸出必須存在且未被花費
- 使用引用的輸出交易來獲得輸入值，檢查每個輸入值，以及總和，在允許的範圍中（大於0，小於2100萬）。
- 如果輸入值的總和小於輸出值的總和，拒絕
- 如果交易費太低（`minRelayTxFee`），拒絕
- 每個輸入的解鎖腳本必須與對應的輸出的鎖定腳本匹配

這些條件的詳情可以在 Bitcoin Core 中的 `AcceptToMemoryPool`, `CheckTransaction`, 和 `CheckInputs` 中看到。注意，條件是隨時變更的，以應對新的拒絕服務攻擊類型，或者放鬆規則以包含更多類型的交易。

通過在交易到達後，向外傳播前，獨立驗證，每個節點都經歷了一個有效（但未確認）的交易池，稱為 *transaction pool*, *memory pool* 或 *mempool*。

## 挖礦節點

Jing，上海的電腦工程學生，是一個比特幣礦工。Jing通過運營一個“鑽井平臺”來獲得比特幣，這是針對比特幣設計的專業電腦硬體系統。Jing的這套專業系統連接到一個完整比特幣節點伺服器。和 Jing 的做法不同，一些礦工在沒有完整節點的情況下挖礦，我們將在 [礦池](#) 中看到。和每個完整節點一樣，Jing的節點在比特幣網路上接收和傳播未確認的交易，也將這些交易聚合到新的區塊中。

Jing的節點和所有節點一樣，監聽在比特幣網路上傳播的新區塊。但是，新區塊的出現對挖礦節點有特殊的意義。礦工之間的競爭實際上以一個新的區塊的傳播而結束，這個區塊的作用是宣佈一個贏家。對礦工來說，得到一個有效的新區塊意味著其他人贏得了競爭，而他們輸了。然而，一輪比賽的結束也是下一輪比賽的開始。新的區塊不只是一個方格旗，標誌著比賽的結束；它也是下一個區塊競賽的發令槍。

## 將交易聚合到區塊中

在驗證交易之後，比特幣節點將把它們添加到 *memory pool* 或 *transaction pool* 中，在那裡等待交易被包含(挖掘)到一個區塊中。Jing的節點收集、驗證和轉發新的交易，就像其他節點一樣。然而，與其他節點不同的是，Jing的節點將這些交易聚合到 *candidate block* 中。

我們來看看Alice在Bob's Cafe買咖啡時創建的區塊(見 [\[cup\\_of\\_coffee\]](#) )。Alice的交易包含在277,316區塊中。為了演示本章的概念，讓我們假設該區塊是由Jing的採礦系統挖掘的，並跟蹤Alice的交易，是如何成為這個新區塊的一部分的。

Jing的挖礦節點維護區塊鏈的本地副本。當Alice買咖啡的時候，Jing的節點已經裝配了一個鏈到277,314。Jing的節點監聽交易，試圖挖掘一個新區塊，也監聽其他節點發現的區塊。當Jing的節點在挖掘時，它通過比特幣網路接收到區塊277315。這個區塊的到來標誌著第277315區塊競賽的結束，以及第277316區塊競賽的開始。

在之前的10分鐘裡，Jing的節點搜索277,315區塊的解決方案時，它也在收集交易，併為下一個區塊做準備。到目前為止，它已經在Memory pool中收集了幾百個交易。在接收到第277315區塊並進行驗證之後，Jing的節點還將它與Memory pool中的所有交易進行比較，並刪除第277315區塊中包含的任何交易。留在Memory pool中的交易都是未確認的，並等待在新的區塊中記錄。

Jing的節點立即構造一個新的空區塊，作為277,316區塊的候選。這個區塊被稱為 *candidate block*，因為它還不是一個有效的區塊，不包含有效的工作證明。只有當礦機成功找到PoW的解決方案時，該區塊才有效。

當Jing的節點將Memory pool中的所有交易彙總時，新的候選區塊有418筆交易，總交易費用為0.09094928比特幣。你可以使用Bitcoin Core客戶端命令行接口在區塊鏈中看到這個區塊，如 [Using the command line to retrieve block 277,316](#) 所示。

Example 3. Using the command line to retrieve block 277,316

```
$ bitcoin-cli getblockhash 277316  
  
00000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4  
  
$ bitcoin-cli getblock 00000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

幣基交易

區塊中的第一筆交易是一筆特殊的交易，叫做 **幣基交易** *coinbase transaction*。這筆交易是 Jing 的節點創建的，包含對他的挖礦努力的獎勵。

Note 當區塊 277,316 被挖出時，獎勵是每個區塊25比特幣。在其之後，一個“減半”週期已經過去。區塊獎勵在2016年7月變為12.5比特幣。2020年，到達210000區塊時，將再次減半。

Jing的節點創建了coinbase交易，對自己錢包的支付：“支付給Jing的地址25.09094928比特幣”。Jing開採一個區塊所收取的獎勵總額是幣基獎勵(25個新比特幣)和該區塊所有交易的費用 (0.09094928) 之和，如 [Coinbase transaction](#) 所示。

#### Example 4. Coinbase transaction

```
$ bitcoin-cli getrawtransaction  
d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f 1
```

```

    "vout" : [
        {
            "value" : 25.09094928,
            "n" : 0,
            "scriptPubKey" : {
                "asm" : "02aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b210P_CHECKSIG",
                "hex" : "2102aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21ac",
                "reqSigs" : 1,
                "type" : "pubkey",
                "addresses" : [
                    "1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N"
                ]
            }
        }
    ]
}

```

與一般交易不同，幣基交易不消耗（花費）UTXO作為輸入。它只有一個輸入，叫做 *coinbase* 幣基，從無創造比特幣。幣基交易有一筆輸出，可支付給礦工自己的比特幣地址。幣基交易的輸出將 25.09094928 比特幣發送到礦工的比特幣地址；在這個例子中，是：1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N。

## 幣基獎勵和費用

為了創建幣基交易，Jing的節點首先通過累計418筆交易的輸入和輸出計算所有交易費用。計算方式如下：

```
Total Fees = Sum(Inputs) - Sum(Outputs)
```

在區塊 277,316 中，總的交易費用為 0.09094928 比特幣。

接下來，Jing的節點計算新區塊的正確獎勵。獎勵是根據區塊的高度來計算的，從每區塊50比特幣開始，每21萬個區塊減少一半。因為這個方區塊的高度是277,316，正確的獎勵是25比特幣。

在 Bitcoin Core 客戶端的 GetBlockSubsidy 方法中可以看到，如 [Calculating the block reward —— Function GetBlockSubsidy, Bitcoin Core Client, main.cpp](#) 所示：

Example 5. Calculating the block reward —— Function GetBlockSubsidy, Bitcoin Core Client, main.cpp

```

CAmount GetBlockSubsidy(int nHeight, const Consensus::Params& consensusParams)
{
    int halvings = nHeight / consensusParams.nSubsidyHalvingInterval;
    // Force block reward to zero when right shift is undefined.
    if (halvings >= 64)
        return 0;

    CAmount nSubsidy = 50 * COIN;
    // Subsidy is cut in half every 210,000 blocks which will occur approximately every 4 years.
    nSubsidy >>= halvings;
    return nSubsidy;
}

```

最初的獎勵以 聰（satoshis）為單位，通過50乘以 COIN 常數（100,000,000 聰）。這將初始獎勵設置為 50億 satoshis。

然後，方法計算 halvings（減半）的次數，當前區塊高度除減半區間（SubsidyHalvingInterval），在這個例子中，是 277,316 / 210,000，結果為1。

最大的減半次數為 64，所以如果超過 64 次減半，程式碼返回 0（只獎勵費用）獎勵。

接下來，該函數使用二進制右移運算符將獎勵（nSubsidy）分為兩半。在277,316區塊的情況下，對50億 satoshis 的獎勵進行二元右移（一次減半），結果為 25億 satoshis 或25個比特幣。使用二進制右移運算符是因為它比多次除法更有效率。為了避免潛在的錯誤，位移操作在63次減半後跳過，補貼設置為0。

最後，幣基獎勵 (nSubsidy) 被加到交易費上 (nFees)，返回總和。

## Tip

如果Jing的挖礦節點寫出了coinbase交易，那麼Jing是不是可以“獎勵”他自己的100或1000比特幣？答案是，錯誤的獎勵會導致該區塊被其他人認為是無效的，從而浪費了Jing用於工作證明的電力。只有該區塊被大家接受，Jing才能花費獎勵。

## 幣基交易的結構

通過這些計算，Jing的節點通過支付其自己 25.09094928 比特幣構建幣基交易。

如你在 [Coinbase transaction](#) 中看到，幣基交易有特殊的格式。不同於指定一個要花費的之前的UTXO的交易輸入，它有一個 "coinbase" 輸入。我們在 [\[tx\\_in\\_structure\]](#) 中檢查交易輸入。讓我們比較一下普通交易輸入和幣基交易輸入。[The structure of a "normal" transaction input](#) 展示了普通交易輸入的結構，[The structure of a coinbase transaction input](#) 展示了幣基交易輸入的結構。

Table 1. The structure of a "normal" transaction input

Size	Field	Description
32 bytes	交易的Hash	指向包含要花費的UTXO的交易的指針
4 bytes	輸出的索引	要花費的UTXO的索引號，第一個從0開始
1-9 bytes (VarInt)	解鎖腳本大小	接下來的解鎖腳本的長度（字節）
Variable	解鎖腳本	滿足UTXO鎖定腳本條件的腳本
4 bytes	序列號	目前禁用的 Tx-replacement 功能，設置為 0xFFFFFFFF

Table 2. The structure of a coinbase transaction input

Size	Field	Description
32 bytes	交易的Hash	所有位都是0：沒有要引用的交易
4 bytes	輸出的索引	所有位都是1: 0xFFFFFFFF
1-9 bytes (VarInt)	幣基數據大小	幣基數據的長度，2 到 100 字節
Variable	幣基數據	用於額外隨機數和採礦標籤的任意數據。在v2區塊中，必須從區塊高開始
4 bytes	序列號	設為 0xFFFFFFFF

在幣基交易中，前兩個欄位被設置為不引用UTXO的值。不同於“交易的Hash”，第一個欄位填充32個字節，全部設置為零。“輸出索引”填充4個字節，全部設置為0xFF（十進制255）。“解鎖腳本”（scriptSig）被幣基數據（Coinbase Data）取代，這是礦工使用的數據欄位，我們將在下面看到。

## 幣基數據 Coinbase Data

幣基交易沒有解鎖腳本（scriptSig）欄位。這個欄位被 coinbase data 替代，該欄位必須包含 2-100 個字節。除了前幾個字節，其他的可由礦工填充任意數據。

例如，中本聰在創世區塊的 coinbase data 中加入了文本 "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks"，將它作為日期的證明，並傳達了一個訊息。現在，礦工使用 coinbase data 存放額外的隨機值，和識別礦池的字串。

幣基的前幾個字節以前是隨機的，但現在不是了。根據BIP-34，version-2區塊（版本號為2）必須在幣基交易欄位的開頭包含區塊高度作為腳本 push 操作。

在區塊 277,316 中我們看到的幣基（參見 [Coinbase transaction](#)），位於交易輸入的解鎖腳本或 scriptSig 欄位，包含十六進制值 03443b0403858402062f503253482f。讓我們解碼它。

第一個字節 03 指示腳本執行引擎將接下來的三個字節壓入腳本堆疊(參見 [\[tx\\_script\\_ops\\_table\\_pushdata\]](#))。接下來的三個字節，0x443b04，是用小端序編碼的區塊高度(倒序，低位字節優先)。反轉字節的順序，結果是 0x043b44，十進制是277,316。

接下來的幾個十六進制數據（0385840206），編碼了額外的隨機數（參見 [額外隨機數解決方案](#)），用於找到PoW的解決方案。

幣基數據的最後部分（2f503253482f）是ASCII編碼的字串 /P2SH/，表明挖到這個區塊的節點支持 BIP-16 中定義的 P2SH 交易。P2SH 能力的引入要求礦工認可BIP-16或BIP-17。支持BIP-16的人將在他們的coinbase數據中包含 /P2SH/。支持P2SH的BIP-17實現的人將字串 p2sh/CHV 包含在他們的coinbase數據中。BIP-16勝出，許多礦工繼續在他們的coinbase中包含字串 /P2SH/，以表示對該特性的支持。

[Extract the coinbase data from the genesis block](#) 使用 [\[alt\\_libraries\]](#) 中介紹的 libbitcoin 庫從創世區塊中提取幣基數據，展示中本聰的訊息。注意，libbitcoin 庫包含一份創世區塊的靜態副本，所以示例程式碼可以直接從庫中檢索到創世區塊。

#### Example 6. Extract the coinbase data from the genesis block

```
link:code/satoshi-words.cpp[]
```

我們使用 GNU C++ 編譯器編譯和運行：

#### Example 7. Compiling and running the satoshi-words example code

```
$ # Compile the code
$ g++ -o satoshi-words satoshi-words.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Run the executable
$ ./satoshi-words
The Times 03/Jan/2009 Chancellor on brink of second bailout for banks
```

## 構建區塊頭

要構建區塊頭，挖礦節點需要填充6個欄位，在 [The structure of the block header](#) 列出：

Table 3. The structure of the block header

Size	Field	Description
4 bytes	Version	最終軟體/協議更新的版本號
32 bytes	Previous Block Hash	引用鏈中上一個區塊（父區塊）的雜湊值
32 bytes	Merkle Root	該區塊中交易的merkle樹的根Hash
4 bytes	Timestamp	區塊的大概創建時間（ Unix 紀元以來的秒數 ）
4 bytes	Target	該區塊的 PoW 演算法目標

4 bytes	Nonce	PoW 演算法使用的計數器
---------	-------	---------------

在 277,316 區塊被挖掘時，區塊結構的版本號是2，以小端序編碼為四字節是 0x02000000.

接下來，挖礦節點需要添加 "Previous Block Hash" ( 稱為 prevHash )。這是 277,315 區塊的雜湊值，是 Jing 的節點從網路收到並接受的作為候選區塊 277,316 的父區塊。277,315 區塊的雜湊值是：

```
00000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569
```

Tip

通過選擇特定的父區塊(由候選區塊頭中的 Previous Block Hash 欄位所指示)，Jing 將其挖掘能力用於擴展以該特定區塊結束的鏈。從本質上說，這就是 Jing 使用他的挖礦力量為最長難度的有效鏈“投票”。

下一步是使用merkle樹彙總所有交易，以便將merkle根添加到區塊頭中。coinbase交易被列為區塊中的第一個交易。然後，在它之後又添加了418個交易，總共在區塊中添加了419個交易。如我們在 [\[merkle\\_trees\]](#) 中看到的，樹中必須有偶數個“葉子”節點，因此最後一個交易被複制，創建420個節點，每個節點都包含一個交易的雜湊。然後將交易雜湊成對地組合在一起，創建樹的每一級，直到將所有交易彙總為樹的“根”節點。merkle樹的根將所有交易彙總為單個32字節的值，你可以看到 [Using the command line to retrieve block 277,316](#) 中列出的“merkle root”：

```
c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e
```

Jing 的挖礦節點然後將添加 4字節的時間戳，編碼為 Unix 紀元時間戳，表示從 UTC/GMT時間 1970年1月1日 零點 以來的秒數，1388185914 等於 Friday, 27 Dec 2013, 23:11:54 UTC/GMT 。

然後 Jing 的節點填充目標 (target) 欄位，定義了使其成為一個有效區塊所需的PoW。target 在區塊中以 "target bits" 矩陣儲存，這是目標的尾數-指數 (mantissa-exponent) 編碼。編碼有1字節的指數，緊接3字節的尾數 (係數) 。例如，在區塊 277,316 中，target bits 的值是 0x1903a30c。第一部分 0x19 是一個十六進制指數，後面的部分，0x03a30c，是係數。target 的概念和 target bits 的表示分別在 [重新設定目標調整難度](#) 和 [目標 \(Target\) 的表示](#) 中說明。

最後一個欄位是隨機數 (nonce) ，初始化為0。

隨著所有其他欄位被填充，區塊頭現在已經完成，挖礦過程開始。目標是找到一個隨機數的值，使區塊頭的雜湊值小於 target。在找到合適的隨機值前，挖礦節點可能需要嘗試數十億，或數萬億次。

## 挖掘區塊

用最簡單的術語來說，挖礦是重複雜湊區塊頭的過程，不斷更改參數，直到生成的雜湊值與特定目標相匹配。雜湊函數的結果不能預先確定，也不能創建產生特定雜湊值的模式。雜湊函數的這種特性意味著產生匹配特定目標的雜湊結果的唯一方法是反覆嘗試，隨機修改輸入，直到偶然出現所需的結果。

## 工作量證明演算法 Proof-of-Work Algorithm

密碼雜湊演算法的關鍵特徵是，在計算上不可能找到產生相同指紋的兩個不同輸入（稱為 碰撞 *collision*）。作為推論，除了嘗試隨機輸入之外，通過選擇輸入以產生期望的指紋的方式實際上也是不可能的。

使用SHA256，無論輸入是什麼，輸出總是256位的。在 [SHA256 example](#) 中，我們使用Python解釋器計算 "I am Satoshi Nakamoto." 的SHA256雜湊值。

Example 8. SHA256 example

```
$ python
```

```
Python 2.7.1
>>> import hashlib
>>> print hashlib.sha256("I am Satoshi Nakamoto").hexdigest()
5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e
```

[SHA256 example](#) 展示了 "I am Satoshi Nakamoto" 的雜湊結果：

5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e. 這個 256位的數字是這句話的 雜湊 hash 或 摘要 digest，基於這句話的每部分。添加一個字母、標點符號，或其他任何字符都將產生不一樣的雜湊值。

現在，如果我們改變語句，會會看到完全不同的雜湊值。讓我們使用 [SHA256 script for generating many hashes by iterating on a nonce](#) 中簡單的Python腳本嘗試在尾部添加陣列。

Example 9. SHA256 script for generating many hashes by iterating on a nonce

```
link:code/hash_example.py[]
```

運行它將產生幾個短語的雜湊，通過在文本末尾添加一個數字來使其不同。通過增加數字，我們可以得到不同的雜湊，如 [SHA256 output of a script for generating many hashes by iterating on a nonce](#) 所示。

Example 10. SHA256 output of a script for generating many hashes by iterating on a nonce

```
$ python hash_example.py
```

```
I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbab...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```

每個短語產生一個完全不同的雜湊結果。它們看起來完全是隨機的，但是你可以在使用Python的任何電腦上再次生成這個示意中的結果，並看到相同的雜湊。

在這種場景中用作變數的數字稱為nonce。nonce用於改變加密函數的輸出，在本例中，是為了改變短語的SHA256指紋。

要對這個演算法提出挑戰，我們來設置一個目標:找到一個短語，它生成一個以0開頭的十六進制雜湊。幸運的是,這並不難! [\[sha256\\_example\\_generator\\_generator\\_output\]](#) 顯示："I'm Satoshi Nakamoto13" 這個短語產生的是一個符合我們的標準的雜湊值：0ebc56d59a34f5082aaef3d661696c2b618e6243272169531041a5。花了13次才找到它。在概率方

打個簡單的比方，想像這樣一個遊戲：玩家不斷地擲一副骰子，試圖擲得比指定的目標少。在第一輪，目標是12個。除非你擲雙六，否則你就能贏。下一輪的目標是11。玩家必須投出10或更少的數值才能獲勝。這同樣是一項簡單的任務，假設幾輪之後，目標是5。現在，超過一半的結果將超過目標而無效。隨著目標降低，要想贏得勝利，擲骰子的次數要成倍增加。最終，當目標是2（可能的最小值）時，每36次投擲中只有一次，或其中的2%，會產生一個勝利的結果。

從一個知道骰子遊戲的目標是2的觀察者的角度來看，如果有人成功地投出了一個成功的結果，那麼可以假設他們平均嘗試了36次。換句話說，一個人可以從目標設定的難度中估計成功所需要的工作量。當演算法是基於確定性函數（如SHA256）時，輸入本身就構成了證明 *proof*，證明做了一定量的工作 *work* 才產生低於目標的結果。所以稱為, *Proof-of-Work*。

**Tip** 即使每次嘗試都會產生隨機結果，任何可能結果的概率都可以提前計算。因此，特定難度的結果構成特定工作量的證明。

在 [SHA256 output of a script for generating many hashes by iterating on a nonce](#) 中，獲勝的“nonce”是13，這個結果可以被任何人獨立的驗證。任何人都可以在短語“我是中本聰”後面加上數字13，然後計算雜湊，驗證它是否小於目標。成功的結果也是工作的證明，因為它證明我們做了工作去發現那一次。雖然只需要一次雜湊計算就可以驗證，但我們需要13次雜湊計算才能找到一個有效的nonce。如果我們有一個更低的目標(更高的難度)，那麼需要更多的雜湊計算才能找到一個合適的nonce，但是對於任何人來說，只有一個雜湊計算需要驗證。此外，通過了解目標，任何人都可以使用統計數據來估計困難程度，從而知道需要做多少工作才能找到這樣一個nonce。

**Tip** PoW必須生成一個小於目標的雜湊值。更高的目標意味著找到低於目標的雜湊值要容易得多。較低的目標意味著更難在目標以下找到雜湊值。目標和難度是反比的。

比特幣的PoW與 [SHA256 output of a script for generating many hashes by iterating on a nonce](#) 所展示的挑戰非常相似。礦機構造一個充滿交易的候選區塊。接下來，挖礦程序計算這個區塊頭的雜湊，看看它是否小於當前的 *target*。如果雜湊值不小於目標，那麼礦機將修改nonce(通常只將其遞增1次)，並再次嘗試。在比特幣網路目前的難度下，礦工必須嘗試千萬億次，才能找到一個能產生足夠低的區塊頭雜湊的nonce。

一個非常簡單的PoW在 Simplified Proof-of-Work implementation 中以Python實現

#### Example 11. Simplified Proof-of-Work implementation

[link:code/proof-of-work-example.py](#) [1]

運行這段程式碼，你可以設置所需的難度(以位為單位，有多少位前導位必須為零)，並查看電腦需要多長時間才能找到解決方案。在 `[pow example output]` 中，你可以看到它在普通的筆記本上是如何工作的。

Example 12. Running the Proof-of-Work example for various difficulties

```
$ python proof-of-work-example.py*
```

Difficulty: 1 (0 bits)

[ ... ]

Difficulty: 8 (3 bits)

Starting search...

Success with nonce 9

Hash is 1c1c105e65b47142f028a8f93ddf3dabb9260491bc64474738133ce5256cb3c1

```

Elapsed Time: 0.0004 seconds
Hashing Power: 25065 hashes per second
Difficulty: 16 (4 bits)
Starting search...
Success with nonce 25
Hash is 0f7becfd3bcd1a82e06663c97176add89e7cae0268de46f94e7e11bc3863e148
Elapsed Time: 0.0005 seconds
Hashing Power: 52507 hashes per second
Difficulty: 32 (5 bits)
Starting search...
Success with nonce 36
Hash is 029ae6e5004302a120630adccb808452346ab1cf0b94c5189ba8bac1d47e7903
Elapsed Time: 0.0006 seconds
Hashing Power: 58164 hashes per second

[...]

Difficulty: 4194304 (22 bits)
Starting search...
Success with nonce 1759164
Hash is 0000008bb8f0e731f0496b8e530da984e85fb3cd2bd81882fe8ba3610b6cefc3
Elapsed Time: 13.3201 seconds
Hashing Power: 132068 hashes per second
Difficulty: 8388608 (23 bits)
Starting search...
Success with nonce 14214729
Hash is 000001408cf12dbd20fcba6372a223e098d58786c6ff93488a9f74f5df4df0a3
Elapsed Time: 110.1507 seconds
Hashing Power: 129048 hashes per second
Difficulty: 16777216 (24 bits)
Starting search...
Success with nonce 24586379
Hash is 0000002c3d6b370fccd699708d1b7cb4a94388595171366b944d68b2acce8b95
Elapsed Time: 195.2991 seconds
Hashing Power: 125890 hashes per second

[...]

Difficulty: 67108864 (26 bits)
Starting search...
Success with nonce 84561291
Hash is 0000001f0ea21e676b6dde5ad429b9d131a9f2b000802ab2f169cbca22b1e21a
Elapsed Time: 665.0949 seconds
Hashing Power: 127141 hashes per second

```

如你所見，將難度增加1位會使找到解決方案所需的時間增加一倍。如果考慮整個256位的數字空間，每次將多一個位限制為0，搜索空間就減少了一半。在 [\[pow\\_example\\_output\]](#) 中，需要8400萬次雜湊才能找到一個nonce，它產生的雜湊有26個前導位為零。即使以每秒超過12萬次雜湊的速度，在筆記本上也需要10分鐘才能找到這個解決方案。

在編寫本文時，網路正在嘗試查找一個小於以下值的區塊頭雜湊：

如你所見，目標的開頭有很多0，這意味著可以接受的雜湊範圍要小得多，很難找到一個有效的雜湊值。網路要發現下一個區塊，平均每秒需要超過1.8 zeta-hashes(thousand billion billion hashes)。這似乎是一項不可能完成的任務，但幸運的是，網路有每秒產生3個exa-hashes(EH/sec)的處理能力，平均10分鐘就能找到一個block。

## 目標 (Target) 的表示

在 [Using the command line to retrieve block 277,316](#) 中，我們看到這個區塊包含了目標，以一個稱為“target bits”或是“bits”，在區塊277,316中的值為 0x1903a30c。該表示法將工作量證明的驗證目標表示為係數/指數格式，前兩個十六進制數字是指數，後六個十六進制數字是係數。因此，在這個區塊中，指數為 0x19，係數為 0x03a30c。

這種表達方式下計算難度目標的公式是：

- target = coefficient \*  $2^{(8*(exponent-3))}$

使用這個公式，和難度bits值 0x1903a30c，可以得到：

- target =  $0x03a30c * 2^{0x08*(0x19-0x03)}$
  - => target =  $0x03a30c * 2^{(0x08*0x16)}$
  - => target =  $0x03a30c * 2^{0xB0}$

十進制就是：

- => target =  $238,348 * 2^{176}$
  - => target =  
22.829.202.948.393.929.850.749.706.076.701.368.331.072.452.018.388.575.715.328

轉換為十六進制：



這意味著高度為277,316的有效區塊是區塊頭的雜湊值小於目標的區塊。在二進制中，該數字必須有超過60個前導位設置為零。有了這樣的難度，一個礦工每秒處理1萬億次雜湊，平均只能每8,496個區塊或每59天尋找到一次解決方案。

## 重新設定目標調整難度

正如我們所看到的，目標確定了難度，因此影響了找到工作證明演算法的解決方案所需的時間。這就引出了一個明顯的問題：為什麼困難是可以調整的，由誰來調整，以及如何調整？

比特幣的區塊平均每10分鐘生成一次。這是比特幣的心跳，支撐著貨幣發行的頻率和交易結算的速度。它必須保持不變，不僅是短期的，而是持續幾十年。在這段時間裡，預計電腦的能力將繼續快速增長。此外，參與挖礦的人和電腦的數目也不斷變化。為了保持10分鐘的生成時間，必須考慮這些變化調整挖掘的難度。事實上，工作證明的目標是一個動態參數，它定期調整以滿足10分鐘的區塊間隔目標。簡單地說，可以設置目標值，使當前的挖礦能力導致10分鐘的區塊間隔。

那麼，這種調整是如何在一個完全分散的網路中進行的呢？重新設定目標是獨立地在每個節點上自動進行的。每產生2016個區塊，所有節點都重新設定PoW目標。重新設定目標的公式衡量了找到最後2016個區塊所需的時間，並與預期的20160分鐘(2016個區塊乘以期望的10分鐘區塊間隔)進行了比較。計算實際時間間隔和期望時間間隔的比例，並對目標按比例進行調整(向上或向下)。簡單地說：如果網路發現區塊的速度比每10分鐘快，難度就會增加(目標範圍縮小)。如果區塊發現速度比預期的要慢，那麼難度就會降低(目標範圍增加)。

公式如下：

New Target = Old Target \* (Actual Time of Last 2016 Blocks / 20160 minutes)

[Retargeting the Proof-of-Work —— CalculateNextWorkRequired\(\) in pow.cpp](#) 展示了 Bitcoin Core 客戶端使用的程式碼。

#### Example 13. Retargeting the Proof-of-Work —— CalculateNextWorkRequired() in pow.cpp

```
// Limit adjustment step
int64_t nActualTimespan = pindexLast->GetBlockTime() - nFirstBlockTime;
LogPrintf(" nActualTimespan = %d before bounds\n", nActualTimespan);
if (nActualTimespan < params.nPowTargetTimespan/4)
    nActualTimespan = params.nPowTargetTimespan/4;
if (nActualTimespan > params.nPowTargetTimespan*4)
    nActualTimespan = params.nPowTargetTimespan*4;

// Retarget
const arith_uint256 bnPowLimit = UintToArith256(params.powLimit);
arith_uint256 bnNew;
arith_uint256 bnOld;
bnNew.SetCompact(pindexLast->nBits);
bnOld = bnNew;
bnNew *= nActualTimespan;
bnNew /= params.nPowTargetTimespan;

if (bnNew > bnPowLimit)
    bnNew = bnPowLimit;
```

#### Note

目標值的校準每2,016個區塊發生一次，由於原始的比特幣核心客戶端中出現了一個差一的錯誤，它是基於之前的2,015個區塊（而不是應該的2,016區塊）的總時間，導致重新設定的目標傾向於難度增加0.05%。

Interval (2,016 個區塊) 和 TargetTimespan (兩週時間，1,209,600 seconds) 兩個參數在 *chainparams.cpp* 中定義。

為了避免難度的極端波動，重新設定目標的調整必須小於每個週期4倍。如果所需的目標調整大於4倍，則調整為4倍而不是更多。任何進一步的調整都將在下一個重新設定目標期間完成，因為這種不平衡將持續到下一個2016個區塊。因此，雜湊算力和難度之間的巨大差異可能需要幾個2,016區塊週期來平衡。

#### Tip

挖掘比特幣區塊的難度大約需要整個網路“處理10分鐘”，根據挖掘前2,016個區塊所花費的時間，每2,016個區塊進行一次調整。通過降低或提高目標來實現。

請注意，目標與交易的數量或價值無關。這意味著雜湊的算力以及用於保障比特幣安全鎖消耗的電量也完全獨立於交易數量。比特幣可以擴大規模，實現更廣泛的應用，並保持安全，而不需要增加目前的雜湊算力水平。隨著新礦工進入市場競爭獎勵，雜湊算力的增加代表市場的力量。只要足夠的雜湊算力在礦工誠實追求獎勵的控制下進行，就足以防止“接管”攻擊，因此足以保證比特幣的安全。

挖礦的難度與電力成本相關，以及比特幣與用於支付電力的貨幣的匯率。高性能的挖礦系統在當前硅片製造方面儘可能地高效，將電力盡可能高地轉化為雜湊算力。對挖礦市場的主要影響是1千瓦小時的比特幣電價，因為這決定了挖礦的盈利能力，因此影響了進入或退出挖礦市場的選擇。

## 成功挖到區塊

如我們之前看到的，Jing的節點構建了一個候選區塊，並準備挖掘它。Jing有幾臺硬體挖掘設備和特定於應用的集成電路，其中幾十萬個集成電路以驚人的速度並行運行SHA256演算法。這些定製的機器通過USB或局域網連接到他的挖礦節點。接下來，在Jing的桌面上運行的挖礦節點將區塊頭髮送到他的挖礦硬體，開始每秒嘗試數萬億隨機數。因為隨機數只有32位，當遍歷完所有可能時（大概40億），挖礦硬體改變區塊頭（調整幣基的隨機數或時間戳）並重新測試隨機數，和新的組合。

開始挖掘區塊277,316後的大概11分鐘，一個硬體社保發現了結果，並將其發送回挖礦節點。

當插入到區塊頭後，隨機數 924,591,752 產生了以下區塊雜湊值：

0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bcd4

小於目標值：

000000000000003A30C000

Jing的挖礦節點立即將這個區塊發送到它的對等節點。它們接收，驗證，並傳播這個新的區塊。隨著這個區塊在網路上漣漪般傳播，每個節點都將其添加到自己的區塊鏈上，將區塊鏈的高度擴展到 277,316 個區塊。挖礦節點接收並驗證區塊，放棄自己嘗試挖掘相同區塊的努力，並立即開始計算鏈上的下一個區塊，將Jing的區塊作為“父區塊”。通過在Jing新發現的區塊之上構建，其他的礦工實質上使用它們的算力“投票”，認可Jing的區塊和它擴展的區塊。

在下一節，我們將看下每個節點驗證和選擇最長鏈的過程，從而創建了形成去中心化區塊鏈的共識。

## 驗證新的區塊

比特幣共識機制的第三步是網路中每個節點對每個新區塊進行獨立驗證。隨著新解決的區塊在整個網路中移動，每個節點在傳播給對等節點之前執行一系列測試來驗證它。這確保了只有有效的區塊在網路上傳播。獨立驗證還可以確保那些誠實行動的礦工將自己的區塊整合到區塊鏈中，從而獲得回報。那些不誠實行事的礦工被拒絕，不僅失去了獎勵，而且浪費了尋找工作證明解決方案的努力，導致電力成本沒有補償。

當一個節點收到一個新的區塊時，它將通過一長串檢查清單驗證它必須符合的條件；否則，拒絕該區塊。這些條件可以在 Bitcoin Core 客戶端的方法 `CheckBlock` 和 `CheckBlockHeader` 中看到：

- 區塊的資料結構語法正確
  - 區塊頭的雜湊值小於目標值
  - 區塊的時間戳小於未來2小時（允許時間錯誤）
  - 區塊的大小在可接受的限制範圍內
  - 第一筆（且只有第一筆）交易是幣基交易
  - 區塊中的所有交易是有效的，可以通過 **獨立交易驗證** 中的驗證

網路上每個節點對每個新區塊的獨立驗證確保礦工不會作弊。在之前的章節中，我們看到礦工如何寫出一筆交易，在該區塊內創建新的比特幣並獲得交易費用。為什麼礦工不會自己寫一千個比特幣的交易，而不是正確的獎勵呢？因為每個節點都根據相同的規則驗證區塊。無效的幣基交易會使整個區塊無效，導致該區塊被拒絕，因此該交易永遠不會成為分類賬的一部分。礦工必須根據所有節點遵循的共同規則構建一個完美的區塊，並通過正確的PoW解決方案來挖掘它。為此，他們在挖礦中耗費大量的電力，如果他們作弊，所有的電力和精力都被浪費掉了。這就是為什麼獨立驗證是去中心化共識的一個關鍵組成部分。

## 組裝和選擇區塊的鍊

區塊鏈去中心化共識機制的最後一個步驟是將區塊組裝到鏈中，並選擇最多Proof-of-Work的鏈。當一個節點驗證了一個新的區塊後，它將嘗試通過將區塊鏈接到現有的區塊鏈，來組裝鏈。

節點維護三組區塊：連接到主區塊鏈的區塊，形成主區塊鏈分支的（次級區塊鏈），最後，在已知的鏈中沒有父區塊的區塊（孤兒區塊）。無效的區塊一旦不滿足驗證條件即被拒絕，因此它們不包含在任何鏈中。

任何時候，“主鏈”都是無效鏈區塊中與其相關的最多累積工作量證明。在大多數情況下，這也是其中最多區塊的鏈條，除非有兩條等長鏈和一條有更多的工作量證明。主鏈也將有分支，這些分支是與主鏈上的區塊“兄弟姐妹”。這些區塊是有效的，但不是主鏈的一部分。他們被保留以供將來參考，以防其中一個連鎖店的業務延伸超過主鏈。在下一部分（區塊鏈分叉）中，我們將看到由於在同一高度上幾乎同時開採區塊體而出現次級鏈。

“主鏈”在任何時候都是有著最多的累計Proof-of-Work的區塊組成的 有效 鏈。大多數情況下，這也是擁有最多區塊的鏈，除非有兩個高度相同的鏈，其中一個有更多的Proof-of-Work。主鏈會有分支，分支上的區塊時主鏈上區塊的“兄弟姐妹”。這些區塊是有效的但不是主鏈的一部分。它們留作將來引用，以防這些鏈被擴展而超過主鏈。在下一節 [區塊鏈分叉](#) 中，我們將看到相同高度的區塊同時被挖掘而形成的次鏈。

當接收到一個新區塊時，節點將嘗試將其插入到現有的區塊鏈中。節點將查看區塊的“previous block hash”欄位，該欄位是對父區塊的引用。然後，節點將嘗試在現有的區塊鏈中找到父節點。大多數時候，父類將是主鏈的“頂部”，意味著這個新區塊擴展了主鏈。例如，新區塊277,316具有對其父區塊277,315的雜湊值的引用。大多數接收277,316的節點都已經有了277,315區塊作為主鏈的頂部，因此將連接新的區塊並擴展該鏈。

有時，正如我們在 [\[fork\]](#) 中看到的，新的區塊擴展了非主鏈的鏈。在這種情況下，節點將把新區塊附加到它所擴展的次級鏈上，然後將次級鏈的工作量與主鏈進行比較。如果次級鏈的累積工作量大於主鏈，則節點將在次級鏈上重新聚合，意味著它將選擇次級鏈作為其新的主鏈，使舊主鏈成為次級鏈。如果節點是一個礦工，那麼它現在將構造一個區塊來擴展這個新的、更長的鏈。

如果一個有效的區塊到達了，但沒有在已有的鏈中找到其父區塊，則這個區塊被認為是“孤兒”。孤兒區塊保存在孤兒區塊池中，直到父區塊到達。一旦父區塊到達並連接到已有的鏈上，孤兒區塊就會被取出，並連接到父區塊上，成為鏈的一部分。孤兒區塊通常在兩個區塊在一段很短的時間內被挖掘，但反序到達時發生（子區塊在父區塊之前到達）。

通過選擇最大累積工作量的有效鏈，所有節點最終都能實現網路範圍內的一致。隨著工作量的增加，鏈之間的暫時差異最終得到解決，從而擴展可能鏈中的一條。挖礦節點通過挖掘下一個區塊來選擇擴展哪個鏈，使用它們的挖礦能力“投票”。當他們挖掘一個新的區塊並擴展鏈時，新的區塊本身就代表了他們的選票。

在下一節中，我們將討論競爭鏈(分叉)之間的差異是如何通過 最大累積工作量 的鏈的獨立選擇來解決的。

## 區塊鏈分叉

因為區塊鏈是一個去中心化的資料結構，所以它的不同副本並不總是一致的。區塊可能在不同的時間到達不同的節點，導致節點具有不同的區塊鏈的視圖。為了解決這個問題，每個節點總是選擇並嘗試擴展表示最大工作量證明的區塊的鏈，也稱為最長鏈或最大累積工作量鏈。通過將記錄在鏈中的每個區塊中的工作量相加，節點可以計算創建鏈所花費的總工作量。只要所有節點都選擇最大累積工作量的鏈，全球比特幣網路最終就會收斂到一致的狀態。分叉作為區塊鏈版本之間的臨時不一致而出現，隨著其中一個分叉添加更多區塊時，將最終重新聚合並解決。

Tip

本節中描述的區塊鏈分叉由於全局網路中的傳輸延遲而自然發生。我們還將在本章後面討論故意誘導的分叉。

下面的幾張圖，我們跟蹤網路一個“分叉”事件。這些圖是簡化的比特幣網路表示。為方便說明，不同的區塊以不同的形狀表示。網路中的每個節點表示為圓圈。

每個節點都有自己的全局區塊鏈視角。每個節點從鄰居節點接收區塊，更新自己的區塊鏈副本，選擇最大累積工作量的鏈。為方便說明，每個節點包含一個代表當前主鏈的頂部的形狀。所以，你在節點中看到的星形，表示它是主鏈的頂部。

在第一張圖 [Before the fork —— all nodes have the same perspective](#) 中，網路對區塊鏈有統一的視角，星形 (star) 區塊代表主鏈的頂部。

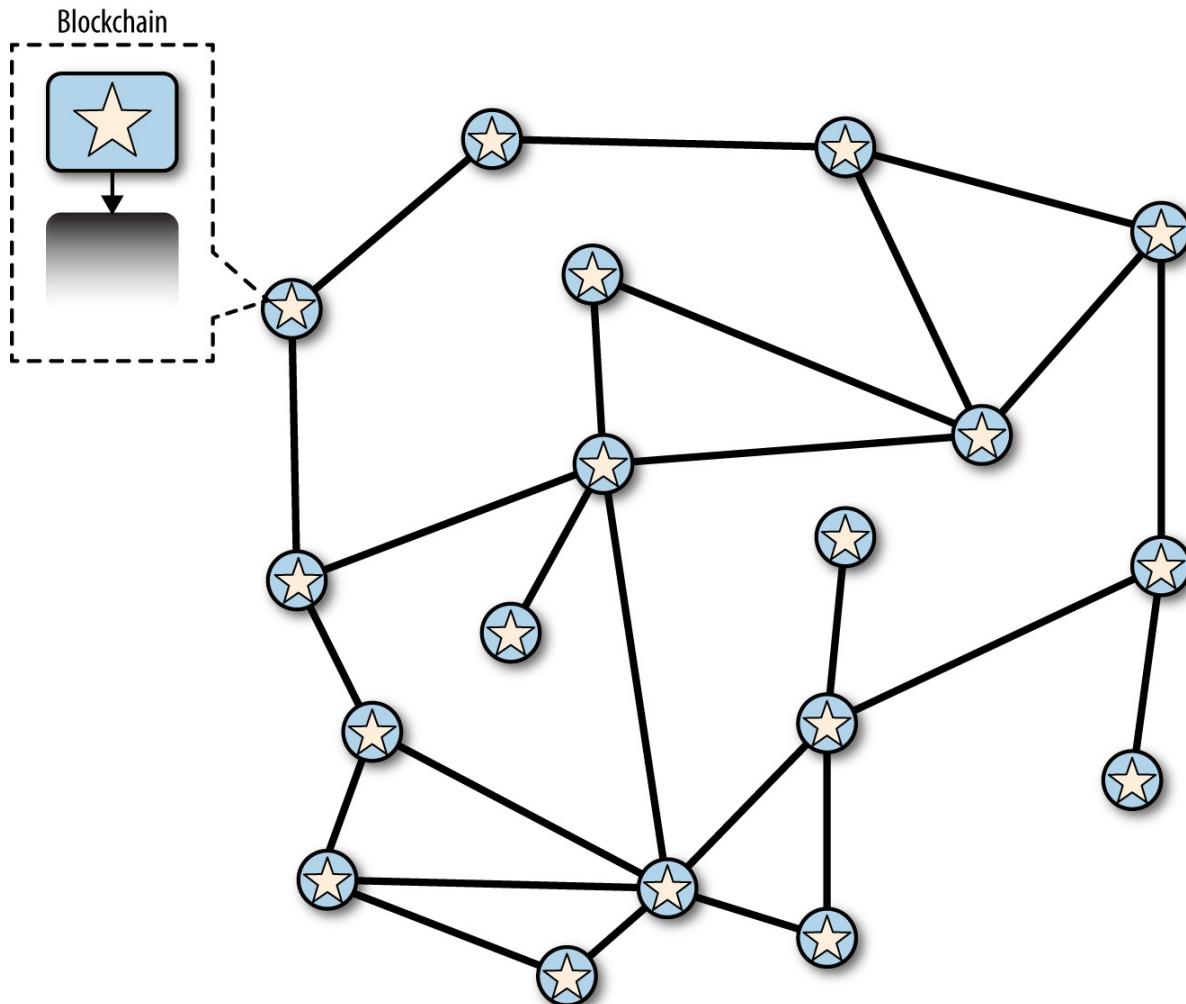


Figure 2. Before the fork — all nodes have the same perspective

當有兩個候選區塊爭奪形成最長鏈時，發生“分叉”。這通常在兩個礦工在相近的時間段內同時解決了Proof-of-Work演算法。兩個區塊發現它們的候選區塊的解決方案後，立即廣播它們的“獲勝的”區塊給鄰居節點，以使它們在網路上傳播。每個收到有效區塊的節點都將其整合進區塊鏈，將其擴展一個區塊。如果節點之後收到擴展相同父區塊的區塊，則將其視為次級鏈上的候選區塊。結果是，一些節點先看到第一個候選區塊，另一些則先看到第二個，這就形成了區塊鏈的兩個競爭版本。

在 [Visualization of a blockchain fork event: two blocks found simultaneously](#) 中，我們看到兩個礦工（節點X和節點Y）幾乎同時挖掘了兩個不同的區塊。這兩個區塊都是星形區塊的子區塊，在其之上擴展區塊鏈。為了便於我們追蹤，節點X產生的標記為三角形（triangle），節點Y產生的標記為倒三角（upside-down triangle）。

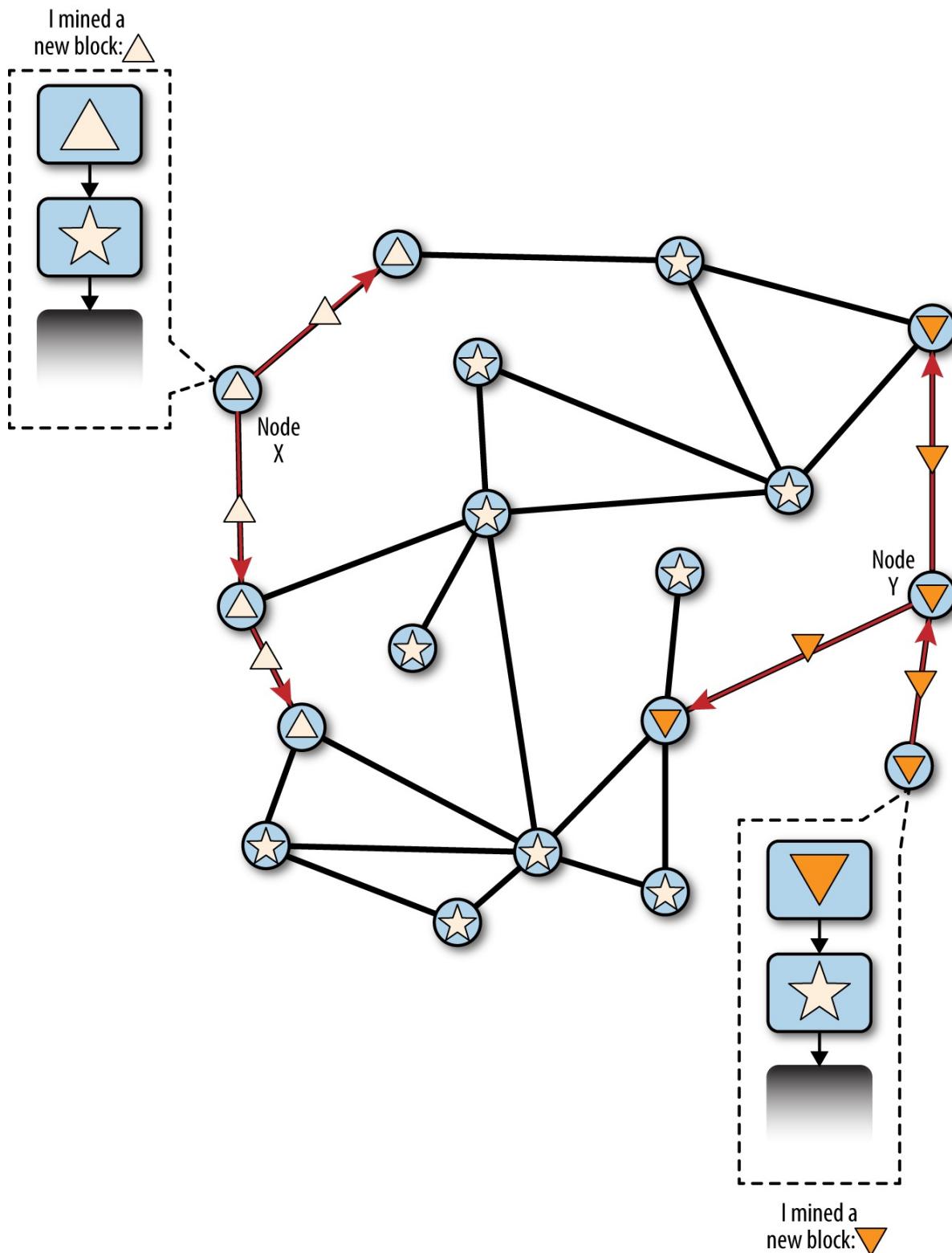


Figure 3. Visualization of a blockchain fork event: two blocks found simultaneously

例如，我們假設節點X為一個擴展區塊鏈的區塊“triangle”找到了一個PoW解決方案，構建在父區塊“star”之上。幾乎與此同時，同樣從“star”擴展鏈的節點Y找到了區塊“upside-down triangle”的解決方案，這是它的候選區塊。兩個區塊都是有效的，兩個區塊都包含一個有效的工作證明解決方案，並且兩個區塊都擴展了相同的父區塊(區塊“star”)。這兩個區塊可能包含大部分相同的交易，交易的順序可能只有很少的差異。

當兩個區塊傳播時，一些節點首先接收區塊到“triangle”，一些節點首先接收區塊“upside-down triangle”。如 [Visualization of a blockchain fork event: two blocks propagate, splitting the network](#)，網路分割為區塊鏈的兩種不同視角：一邊是 triangle，另一邊是 upside-down triangle。

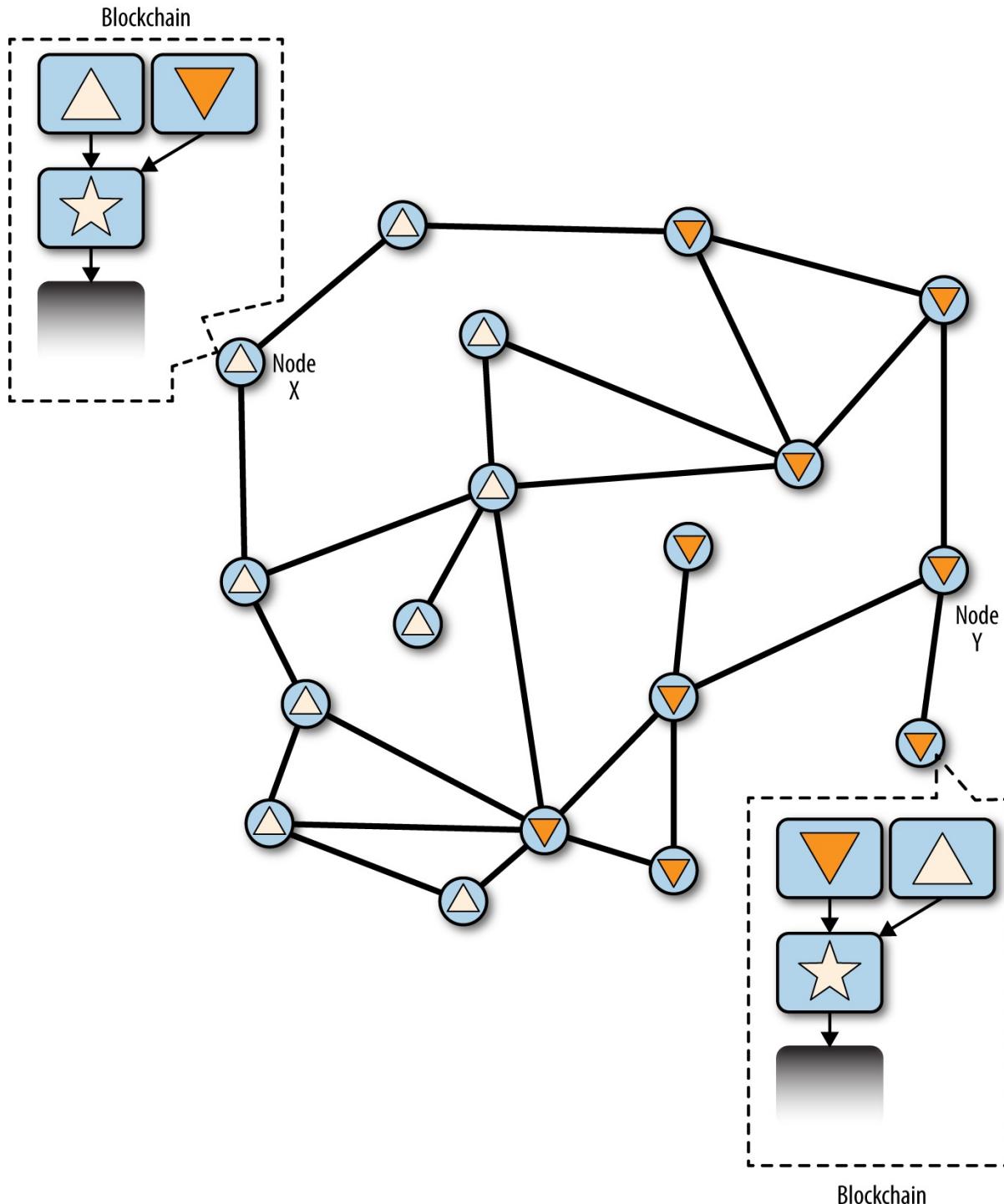


Figure 4. Visualization of a blockchain fork event: two blocks propagate, splitting the network

在圖中，一個隨機選擇的“節點X”首先接收到triangle區塊，並用它擴展star鏈。節點X選擇有“triangle”區塊的鏈作為主鏈。之後，節點X也接收到“upside-down triangle”區塊。由於它是第二名，被認為已經“輸掉”了比賽。然而，“upside-down triangle”區塊並沒有被丟棄。它與“star”區塊父鏈相連，形成一個次級鏈。雖然節點X假設它已經正確地選擇了獲勝鏈，但它保留了“失敗的”鏈，這樣它就有必要的訊息，如果“失敗的”鏈最終“獲勝”，則需要重新聚合。

在網路的另一端，節點Y基於自己對事件序列的看法構建區塊鏈。它首先接受了“upside-down triangle”，並選擇了那個鏈條作為“贏家”。當它後來接收到“triangle”區塊時，它將它作為一個次級鏈連接到“star”父區塊。

哪一邊都不是“正確的”，或者“錯誤的”。兩個都是有效的區塊鏈視角。之後只有一個會勝出，這取決於這兩個相互競爭的鏈如何被後續的工作量擴展。

挖礦視角類似於節點X的節點將立即開始挖掘一個候選區塊，該區塊以“triangle”作為其頂端擴展鏈。通過將“triangle”鏈接為候選區塊的父元素，它們使用雜湊算力投票。他們的投票支持了他們選出來的主鏈。

挖礦視角類似於節點Y的節點都將開始以“upside-down triangle”為父節點構建候選節點，擴展他們認為是主鏈的鏈。所以，比賽又開始了。

分叉幾乎總是在一個區塊中解決。雖然網路雜湊算力的一部分在“triangle”的頂部構建，而另一部分在“upside-down triangle”的頂部構建。即使雜湊算力幾乎是平均分配的，也很有可能在一組礦工找到任何解決方案之前，另一組礦工可能已經找到解決方案並傳播它。例如，假設在“triangle”頂部建造的礦工找到了一個新的區塊“rhombus”（菱形），它擴展了鏈（例如，star-triangle-rhombus）。它們立即傳播這個新區塊，整個網路將其視為有效的解決方案，如 [Visualization of a blockchain fork event: a new block extends one fork, reconverging the network](#) 所示。

所有在前一輪中選擇“triangle”作為獲勝者的節點，只需將鏈再延長一個block。然而，選擇“upside-down triangle”作為獲勝者的節點現在將看到兩條鏈：star-triangle-rhombus 和 star-upside-down-triangle。star-triangle-rhombus 鏈現在比其他鏈長（累積工作量更多）。因此，這些節點將 star-triangle-rhombus 鏈作為主鏈，將 star-upside-down-triangle 鏈轉化為二級鏈，如 [Visualization of a blockchain fork event: the network reconverges on a new longest chain](#) 所示。這是一個鏈的重新收斂，因為這些節點被迫修改他們對區塊鏈的看法，以納入更長的鏈的新證據。任何致力於將鏈基於 upside-down triangle 進行擴展的礦工現在都會停止工作，因為他們的候選區塊是一個“孤兒”，因為它的父區塊“upside-down triangle”不再是長鏈。不屬於“triangle”的“upside-down triangle”內的交易被重新插入到mempool中，以便在下一個區塊中包含，從而成為主鏈的一部分。整個網路在一個區塊鏈 star-triangle-rhombus 上重新收斂，“rhombus”是鏈中的最後一個區塊。所有的礦工立即開始在以“rhombus”作為父區塊的候選區塊上工作，以擴展 star-triangle-rhombus。

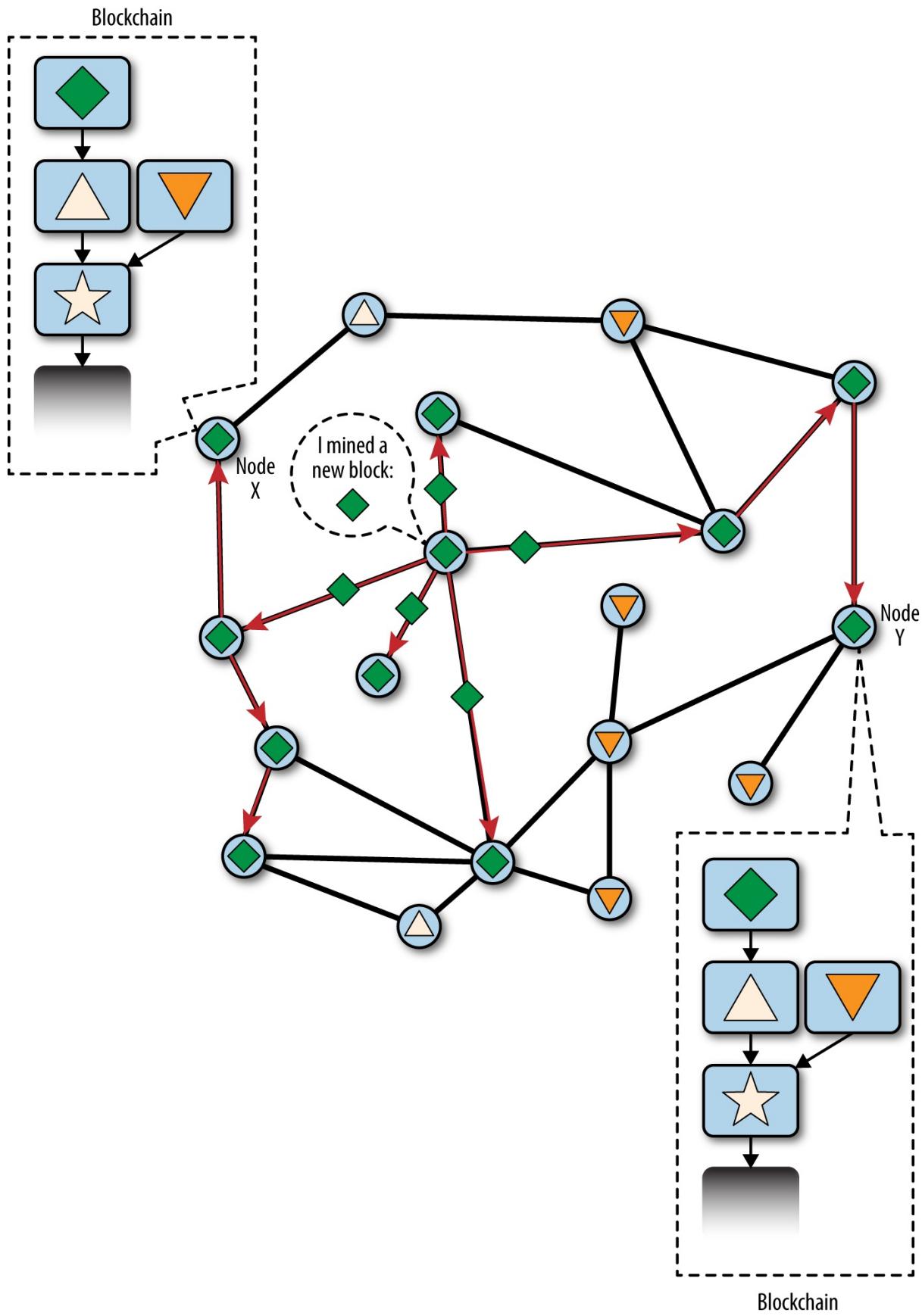


Figure 5. Visualization of a blockchain fork event: a new block extends one fork, reconverging the network

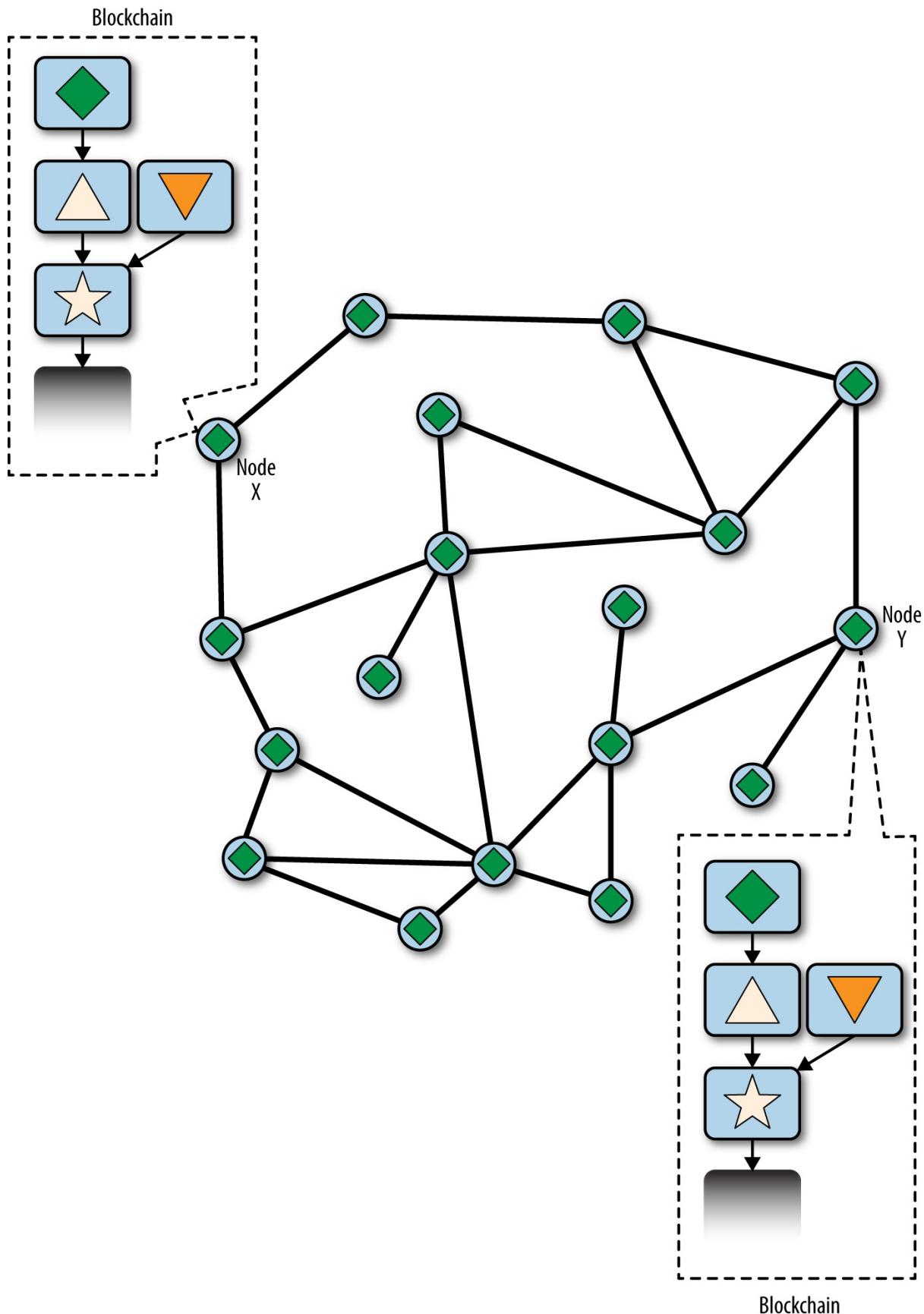


Figure 6. Visualization of a blockchain fork event: the network reconverges on a new longest chain

理論上，如果礦工在前一個分叉的相對面上幾乎同時發現了兩個區塊，那麼這個分叉可能延伸到兩個區塊。然而，發生這種情況的可能性很小。而一個區塊的fork可能每天都出現，而兩個區塊的fork則最多每隔幾周出現一次。

比特幣區塊的10分鐘間隔是快速確認時間（交易結算）和分叉概率之間的設計折中。更快的區塊時間會使交易清除更快，但會導致更頻繁的區塊鏈分叉，而較慢的區塊時間會減少分叉數量，但會降低交易速度。

## 挖礦和雜湊競賽 Mining and the Hashing Race

比特幣開採是一個極具競爭力的行業，比特幣存在以來雜湊算力每年都會成倍地增加。有些年份的增長反映了技術的完全變化，例如2010年和2011年，當時許多礦工從使用CPU採礦轉向GPU和現場可編程門陣列（FPGA）。2013年，通過將SHA256功能直接應用於專門用於採礦的硅芯片，ASIC採礦的引入帶來了採礦能力的又一次巨大飛躍。首批這樣的芯片可以提供比2010年整個比特幣網路更多的採礦能力。

以下列表顯示了比特幣網路在前8年的運行總雜湊算力：

### 2009

0.5 MH/sec–8 MH/sec (16× growth)

### 2010

8 MH/sec–116 GH/sec (14,500× growth)

### 2011

116 GH/sec–9 TH/sec (78× growth)

### 2012

9 TH/sec–23 TH/sec (2.5× growth)

### 2013

23 TH/sec–10 PH/sec (450× growth)

### 2014

10 PH/sec–300 PH/sec (30× growth)

### 2015

300 PH/sec–800 PH/sec (2.66× growth)

### 2016

800 PH/sec–2.5 EH/sec (3.12× growth)

在 [Total hashing power, terahashes per second \(TH/sec\)](#) 的圖表中，我們可以看到比特幣網路的雜湊算力在過去兩年中有所增加。如你所見，礦工之間的競爭和比特幣的增長導致雜湊算力（網路中每秒的總雜湊計算數量）呈指數級增長。

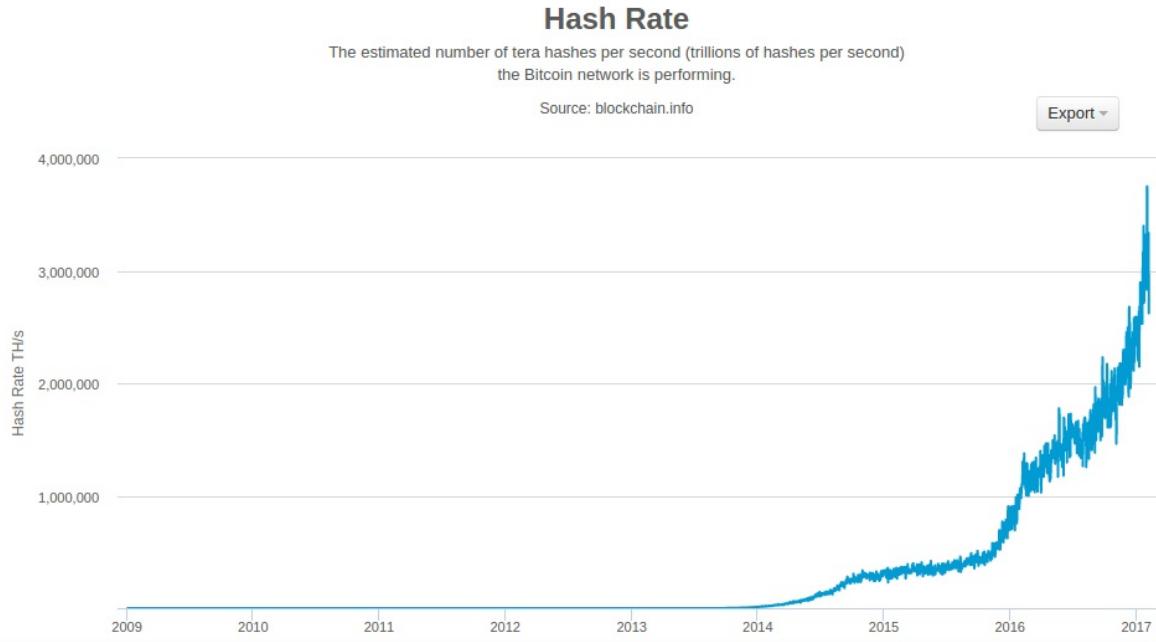


Figure 7. Total hashing power, terahashes per second (TH/sec)

隨著比特幣挖礦的雜湊算力大幅增加，難度也隨之增加。在 [Bitcoin's mining difficulty metric](#) 顯示的圖表中的難度以當前難度相對於最小難度（第一區塊的難度）的比率來衡量。

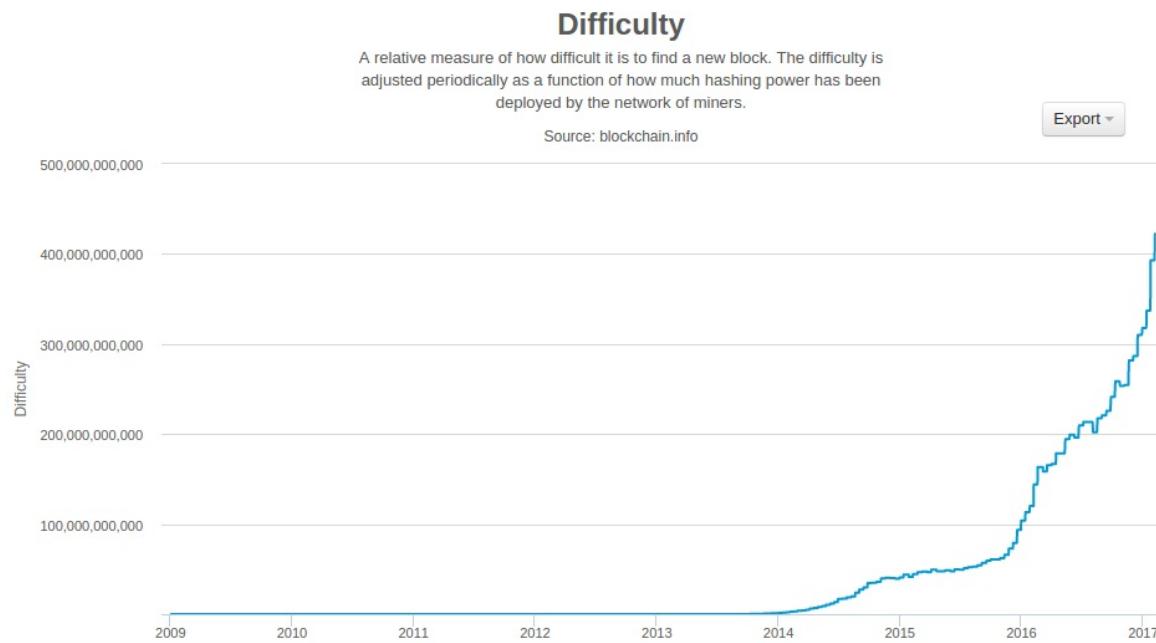


Figure 8. Bitcoin's mining difficulty metric

在過去的兩年中，ASIC採礦芯片變得越來越密集，接近硅片製造的極限，功能尺寸（分辨率）為16納米（nm）。目前，ASIC製造商的目標是超越通用CPU芯片製造商，設計14納米功能尺寸的芯片，因為挖礦的盈利能力比通用計算更快地推動這個行業的發展。比特幣礦業沒有更大的飛躍了，因為該行業已經達到摩爾定律的邊緣，該定律指出計算密度大約每18個月翻一番。儘管如此，隨著高密度芯片的競賽與可以部署數千個芯片的高密度數據中心的競賽相匹配，網路的挖掘能力仍然以指數級的速度增長。現在不再關心一區塊芯片可以挖多少礦，而是可以將多少芯片裝入一個設備中，依然可以散熱並提供足夠的算力。

## 額外隨機數解決方案

自2012年以來，比特幣挖掘已經發展到解決區塊頭結構的根本性限制。在比特幣的早期，礦工可以通過遍歷nonce來找到一個區塊，直到產生的雜湊值低於目標值。隨著困難的增加，礦工經常循環遍歷所有40億個臨時值，而沒有發現區塊。但是，通過更新區塊時間戳以考慮已用時間，這很容易解決。由於時間戳是標題的一部分，所以更改將允許礦工重新遍歷隨機數的值，並獲得不同的結果。但是，一旦採礦硬體超過 4GH/秒，這種方法變得越來越困難，因為nonce值在不到一秒鐘內就被耗盡了。隨著ASIC採礦設備開始推進並超過TH /秒雜湊算力，挖掘軟體需要更多的空間來儲存nonce值，以便找到有效的區塊。時間戳可能會稍微延長，但將其變為將來的值會導致該區塊無效。區塊頭中需要新的“更改”來源。解決方案是使用幣基交易作為額外的隨機值的來源。因為幣基腳本可以儲存2到100個字節的數據，所以礦工開始使用這個空間作為額外的nonce空間，允許他們探索更大範圍的區塊標題值以找到有效的區塊。幣基交易包含在merkle樹中，這意味著幣基腳本中的任何更改都會導致merkle根發生更改。八個字節的額外隨機數加上4個字節的“標準”隨機數允許礦工在不必修改時間戳的情況下總共探索  $2^{96}$  (一個8，後接28個0) 個可能性。如果將來，礦工可以運行所有這些可能性，他們可以修改時間戳。幣基腳本中還有更多空間用於將來擴展額外的隨機數空間。

## 礦池

在這個競爭激烈的環境中，獨自工作的礦工（也被稱為獨立礦工）沒有機會。他們發現一個區塊來抵消他們的電力和硬體成本的可能性非常低，所以它代表著賭博，就像玩彩票一樣。即使是最快的ASIC採礦系統也無法跟上在水力發電附近的巨型倉庫中堆疊數以萬計這些芯片的商業系統。礦工們現在合作形成採礦池，彙集他們的算力，在數千名參與者中分享獎勵。通過參與礦池，礦工獲得的整體獎勵份額較小，但通常每天都會得到獎勵，從而減少不確定性。

我們來看一個具體的例子。假定礦工已經購買了具有 14,000 千兆位/秒 (GH/s) 或 14TH/s 的雜湊算力的採礦硬體。2017 年，這臺設備的成本約為 2,500 美元。硬體在運行時耗電 1375 瓦 (1.3 千瓦)，每天耗電 33 千瓦時，電費非常低，每天花費 1 美元到 2 美元。在目前的比特幣難度下，礦工大概每隔 4 年可以獨自開礦 1 次。我們如何計算出這個概率？它基於 3EH/sec (2017 年) 的網路雜湊算力和這個礦工的 14TH/sec 算力：

$$\bullet \quad P = (14 * 10^{12} / 3 * 10^{18}) * 210240 = 0.98$$

...其中 210240 是每 4 年的區塊數量。礦工每 4 年找到一個區塊的概率為 98%，基於當時的全球雜湊算力。

如果礦工在該時間段內確實找到一個區塊，則可獲得 12.5 比特幣（每比特幣約 1,000 美元），一次性收入 12,500 美元，約 7,000 美元的淨利潤。但是，在 4 年內發現區塊的概率取決於礦工的運氣。他可能會在 4 年內找到兩個區塊並賺取鉅額利潤。或者他可能在 5 年內找不到一個區塊並遭受更大的經濟損失。更糟糕的是，在當前雜湊算力增長率下，比特幣工作量證明演算法的難度可能會顯著增加，這意味著礦工在硬體過時，必須被更強大的採礦硬體取代之前最多隻有一年的時間。如果這位礦工參加礦池，而不是等待四年一度的 12,500 美元的暴利，他將能夠每週賺取大約 50 美元至 60 美元。來自礦池的定期收入將幫助他分攤硬體和電力的成本，而不會承擔巨大的風險。硬體在一兩年後仍會過時，風險仍然很高，但在這段時間內收入至少是穩定可靠的。在經濟上，這隻有在非常低的電力成本（每千瓦小時不到 1 美分）和非常大的規模時才有意義。

礦池通過專門的礦池採礦協議協調成百上千的礦工。在創建了池中的帳戶後，各個礦工將他們的採礦設備配置為連接到池伺服器。他們的採礦硬體在採礦時仍與池伺服器連接，與其他礦工同步工作。因此，礦池礦工共同努力挖掘一區塊，然後分享獎勵。

成功的區塊支付到礦池的比特幣地址，而不是單獨的礦工。池伺服器將定期向礦工的比特幣地址付款，當他們的獎勵份額達到一定的閾值時。通常，池伺服器為提供池採礦服務收取獎勵的百分比的費用。

加入礦池的礦工將尋找候選區塊解決方案的工作拆分，根據他們的挖礦貢獻贏得“分成”。礦池為獲得股份設定了更廣的目標（較低的難度），通常比比特幣網路的目標要容易 1000 倍以上。當礦池中的某人成功地開採了一個區塊時，獎勵將由礦池賺取，然後與所有參與努力的礦工按比例分成。

礦池開放給任何礦工，無論大小，專業或業餘。因此，一個礦池將有一些參與者擁有一臺小型採礦機，而其他人則擁有一個裝滿高端採礦硬體的車庫。一些將採用幾十千瓦的電力進行採礦，另一些將運行耗用兆瓦級電力的數據中心。礦池如何衡量個人的貢獻，以公平分配獎勵，而不會有作弊的可能性？答案是使用比特幣的Proof-of-Work演算法來衡量每個礦池礦工的貢獻，但設置的難度較小，因此即使是最小的礦池礦工也能夠頻繁贏得一些份額，以便為礦池做出貢獻。通過設定較低的賺取份額的難度，該池衡量每個礦工完成的工作量。每當一名礦池礦工發現一個區塊頭雜湊值小於池目標

時，她就證明她已完成雜湊工作找到了結果。更重要的是，尋求份額的工作以統計上可測量的方式貢獻於總體努力，以找到比比特幣網路的目標更低的雜湊值。成千上萬試圖尋找小雜湊值的礦工最終會找到一個足以滿足比特幣網路目標的礦工。

讓我們回到骰子遊戲的比喻。如果骰子玩家投擲骰子的目標是投擲出少於4（整體網路難度）的值，則遊戲池將設定更容易的目標，計算遊戲池玩家投擲結果小於8的次數。當選手擲出少於8（池目標）的值時，他們將獲得份額，但是他們沒有贏得比賽，因為他們沒有達到比賽目標（少於4）。池玩家可以更頻繁地獲得更容易的池目標，即使他們沒有實現贏得比賽的更難的目標，也可以非常經常地贏得他們的份額。無論何時，其中一名球員將擲出少於4的值，並且贏得比賽。然後，收入可以根據他們獲得的份額分配給池玩家。儘管8或更少的目標沒有獲勝，但這是測量玩家擲骰子的公平方法，偶爾會產生少於4的投擲。

類似地，礦池將設置（更高和更容易）的池目標，以確保個體池礦工能夠找到經常低於池目標的區塊頭雜湊值，從而獲得份額。每隔一段時間，這些嘗試中的一個會產生一個比比特幣網路目標小的區塊頭雜湊值，使其成為一個有效的區塊，並且整個池都會贏。

### 管理型礦池

大多數礦池都是“被管理”的，這意味著有一個公司或個人運行池伺服器。池伺服器的所有者稱為礦池運營者 *pool operator*，他向池中礦工收取收入的一定比例費用。

池伺服器運行專門的軟體和池採礦協議來協調池內礦工的活動。池伺服器還連接到一個或多個完整的比特幣節點，並可直接訪問區塊鏈資料庫的完整副本。這允許池伺服器代表池礦工驗證區塊和交易，從而減輕他們運行完整節點的負擔。對於池礦工而言，這是一個重要的考慮因素，因為完整節點需要具有至少100至150 GB持久性儲存（硬碟）和至少2至4GB內存（RAM）的專用電腦。此外，運行在完整節點上的比特幣軟體需要經常進行監控，維護和升級。由於缺乏維護或缺乏資源而導致的任何停機都會損害礦工的盈利能力。對於許多礦工而言，可以在不運行完整節點的情況下進行挖掘是加入托管池的另一大好處。

池礦工使用採礦協議（如Stratum（STM）或GetBlockTemplate（GBT））連接到池伺服器。自2012年年底以來，名為GetWork（GWK）的較舊標準大多已經過時，因為它不能輕易支持以4GH/s以上的雜湊速率進行挖掘。STM和GBT協議都創建了包含候選區塊頭模板的區塊模板。池伺服器通過聚合交易構建候選區塊，添加幣基交易（帶有額外的nonce空間），計算merkle根，並鏈接到前一個區塊的雜湊值。然後將候選區塊的頭作為模板發送給每個泳池礦工。然後每個礦池礦工使用區塊模板進行開採，目標比比特幣網路的目標更廣，並將任何成功的結果發送回池伺服器以賺取份額。

### 對等礦池（P2Pool）

被管理的池可能會導致池運營者作弊，他可能會指使礦池努力進行雙重支付交易或使區塊無效（請參閱 [共識攻擊](#)）。此外，集中池伺服器代表單點故障。如果池伺服器關閉或由於拒絕服務攻擊而放慢速度，池中的礦工將無法開採。2011年，為解決這些集中化問題，提出並實施了一種新的池式挖掘方法：P2Pool，一個沒有中央運營商的對等礦池。

P2Pool通過分散池伺服器的功能工作，實現了稱為股份鏈 *share\_chain* 的並行的類似區塊鏈的系統。股份鏈是比比特幣區塊鏈難度更低的區塊鏈。股份鏈允許礦池礦工通過以每30秒一個區塊的速度挖掘鏈上的份額，在去中心化的池中進行合作。股份鏈上的每個區塊都會為參與工作的池礦工記錄相應的股份回報，並將股份從前一個股份區塊中向前移動。當其中一個股份區塊也實現比特幣網路目標時，它會被傳播并包含在比特幣區塊鏈中，獎勵所有為獲勝股份區塊之前的所有股份作出貢獻的礦池礦工。本質上，與池伺服器跟蹤池礦工的股份和獎勵不同，股份鏈允許所有池礦工使用類似比特幣的區塊鏈共識機制的去中心化共識機制來跟蹤所有股份。

P2Pool挖掘比池挖掘更復雜，因為它要求池礦工運行具有足夠硬碟空間，內存和互聯網帶寬的專用電腦，以支持完整的比特幣節點和P2Pool節點軟體。P2Pool礦工將他們的採礦硬體連接到他們的本地P2Pool節點，該節點通過向採礦硬體發送區塊來模擬池伺服器的功能。在P2Pool上，個人池礦工構建他們自己的候選區塊，像獨立礦工一樣聚集交易，然後在股份鏈上進行協作。P2Pool是一種混合型方法，與單獨挖礦相比，具有更精細的支出優勢，也不會像管理型礦池那樣給池操作員太多控制權。

儘管P2Pool降低了礦池運營商的權力集中度，但可以想象，股份鏈本身有51%攻擊的可能性。P2Pool的廣泛採用並不能解決比特幣本身的51%攻擊問題。相反，P2Pool使比特幣整體更加強大，作為多元化挖礦生態系統的一部分。

## 共識攻擊

至少從理論上講，比特幣的共識機制很容易受到礦工(或礦池)的攻擊，因為他們試圖利用他們的“雜湊”算力，來實施不誠實或破壞性的目的。如我們所看到的，共識機制取決於大多數礦工出於自身利益誠實行事。然而，如果一名礦工或一群礦工能夠獲得相當大的雜湊算力份額，他們就可以攻擊共識機制，從而破壞比特幣網路的安全性和可用性。

重要的是要注意到，共識攻擊只能影響未來的共識，或者至多影響最近的過去(幾十個區塊)。隨著時間的流逝，比特幣的賬本變得越來越不可改變。雖然在理論上，分叉可以在任何深度上實現，但在實踐中，強制執行一個非常深的分叉所需的計算能力是巨大的，這使得舊的區塊實際上是不可變的。共識攻擊也不影響私鑰和簽名演算法(ECDSA)的安全性。共識攻擊不能竊取比特幣，不能在沒有簽名的情況下使用比特幣，不能改變比特幣的方向，也不能改變過去的交易或所有權記錄。共識攻擊只能影響最近的區塊，並在創建未來區塊時導致拒絕服務中斷。

針對共識機制的攻擊場景稱為“51%攻擊”。在這種情況下，一群控制著整個網路的雜湊算力(51%)的礦工勾結起來攻擊比特幣。有了挖掘大部分區塊的能力，攻擊的挖礦人員可以在區塊鏈和雙重支付交易中引起有意的“分叉”，或者對特定的交易或地址執行拒絕服務攻擊。fork/double-spend攻擊是指攻擊者通過在其下方分叉並在另一個鏈上重新聚合而導致先前確認的區塊無效的攻擊。有了足夠的能力，攻擊者可以使一行中的六個或多個區塊失效，從而導致被認為不可變的交易(六個確認)失效。注意，雙重支付只能在攻擊者自己的交易上執行，攻擊者可以為此生成有效的簽名。如果通過使交易無效，攻擊者可以獲得不可逆的兌換付款或產品，而無需為此付費，那麼重複使用自己的交易就是有利可圖的。

讓我們來看一個51%攻擊的實際例子。在第一章中，我們看了Alice和Bob之間的一筆交易，即一杯咖啡。咖啡館老闆Bob願意接受咖啡的付款，而不需要等待確認(在一個區塊內開採)，因為與快速客戶服務的便利性相比，兩次消費一杯咖啡的風險較低。這類似於咖啡店的做法，即接受低於25美元的信用卡支付而無需簽名，因為信用卡退款的風險較低，而延遲交易獲得簽名的成本相對較大。相比之下，以比特幣賣出一件更昂貴的商品，就會有雙重支付攻擊的風險，即買家廣播一項相互競爭的交易，該交易使用相同的輸入(UTXO)，並取消支付給商家的款項。雙重支付攻擊可以以兩種方式發生：要麼在確認交易之前，要麼在攻擊者利用區塊鏈fork撤消多個區塊之前。51%攻擊允許攻擊者在新鏈中重複使用自己的交易，從而取消舊鏈中的相應交易。

在我們的例子中，惡意攻擊者Mallory來到Carol的畫廊，購買了一幅美麗的三聯畫，將中本聰描繪為普羅米修斯。Carol以25萬美元的價格將《The Great Fire》的畫作賣給了Mallory。Carol沒有在交易中等待6個或更多的確認，而是在只有一次確認後將畫作交給了Mallory。Mallory與一個共犯Paul一起工作，Paul經營著一個大的礦池。一旦Mallory的交易被包含在一個區塊內，共犯就會發起51%攻擊。Paul指導礦池重新挖掘與包含Mallory交易的區塊高度相同的區塊，將Mallory對Carol的付款替換為使用相同輸入的雙重支付交易。這種雙重支付交易消耗了相同的UTXO，並將其歸還給Mallory的錢包，而不是支付給Carol，實質上是讓Mallory保留比特幣。然後，Paul引導礦池挖掘一個額外的區塊，從而使包含雙重支付交易的鏈比原來的鏈長(在包含Mallory交易的區塊下產生一個fork)。當區塊鏈分叉被解決為支持新的(更長的)鏈時，重複使用的交易將替換為Carol的原始付款。Carol現在丟失了畫作，也沒有收到比特幣付款。在所有這些活動中，Paul的礦池參與者可能不知道重複支付的嘗試，因為他們使用自動的礦工進行挖掘，不能監視每個交易或區塊。

為了防止這種攻擊，出售大價值商品的商人必須至少等待6次確認才能將產品交給買方。或者，商戶應該使用託管多簽名帳戶，在託管帳戶得到資金後再等待幾個確認。確認越長，就越難以執行51%攻擊使交易無效。對於高價值的物品，即使買家需要等待24小時才能交貨，用比特幣付款仍然是方便和有效的，這相當於大約144個確認。

除了雙重支付攻擊之外，共識攻擊的另一個場景是拒絕向特定的比特幣參與者(具體的比特幣地址)提供服務。擁有大部分挖礦能力的攻擊者可以簡單地忽略特定的交易。如果它們包含在由另一個礦機挖掘的區塊中，攻擊者可以故意對該區塊進行fork和重新挖掘，再次排除特定的交易。這種類型的攻擊可以導致對特定地址或地址集的持續拒絕服務，只要攻擊者控制了大部分挖掘能力即可。

儘管名字為51%攻擊，實際上並不需要51%的雜湊算力。事實上，這樣的攻擊可以嘗試使用較小比例的雜湊算力。51%僅僅是這種攻擊幾乎可以保證成功的水平。達成共識的攻擊本質上是挖掘下一個區塊的集團間的拔河，“更強大”的集團更有可能獲勝。使用較少的雜湊算力，成功的可能性就會降低，因為其他礦工用他們“誠實”的採礦力量控制某些區塊的生成。可以這樣看，攻擊者的雜湊算力越強，他故意創建的分叉越長，他在最近的一段時間內可以使更多的區塊失效，或者在未來可以控制更多的區塊。安全研究小組使用統計建模來宣稱，只要30%的雜湊算力，各種類型的共識攻擊都是可能的。

可以說，比特幣的雜湊算力大幅增加，使得它不會受到任何一個礦工的攻擊。對於一個單獨的礦工來說，不可能控制超過總開採能力的一小部分。然而，由礦池引起的集中控制帶來了礦池運營商進行盈利性攻擊的風險。託管池中的池運營商控制候選區塊的構造，並控制包含哪些交易。這使池運營商能夠排除交易或引入雙重支付交易。如果這種濫用權力的行為是以一種有限而微妙的方式進行的，那麼池運營商可以在不被注意的情況下從共識攻擊中獲利。

然而，並非所有的攻擊者都是受利潤驅動的。一個潛在的攻擊場景是，攻擊者打算破壞比特幣網路，而不可能從這種破壞中獲利。破壞比特幣的惡意攻擊將需要鉅額投資和祕密計劃，可以想象，攻擊者可能是資金充裕、最有可能是政府支持的攻擊者。另外，資金充裕的攻擊者可以通過同時收集挖掘硬體、勾結池運營商、以及拒絕服務攻擊其他池來攻擊比特幣的共識。所有這些設想在理論上都是可能的，但隨著比特幣網路的總體雜湊能力繼續呈指數級增長，它們越來越不切實際。

毫無疑問，一場嚴重的共識攻擊將在短期內削弱人們對比特幣的信心，可能導致比特幣價格大幅下跌。然而，比特幣網路和軟體一直在不斷髮展，因此，比特幣社區會立即應對輿論攻擊，使比特幣更加穩健。

## 共識規則的改變

共識規則決定了交易和區塊的有效性。這些規則是所有比特幣節點之間協作的基礎，並負責將所有本地視圖匯聚到整個網路的一個一致的區塊鏈中。

雖然共識規則在短期內是不變的，並且必須在所有節點之間保持一致，但是從長期來看，它們並不是不變的。為了進化和發展比特幣系統，規則必須不時變化，以適應新的特性、改進或bug修復。然而，與傳統的軟體開發不同，對共識系統的升級要困難得多，需要所有參與者之間的協調。

## 硬分叉

在 [fork] 中，我們研究了比特幣網路可能會有短暫的分歧，在短時間內，在區塊鏈的兩個不同分支的網路中有兩個部分。我們看到這個過程是如何自然地發生的，作為網路正常運行的一部分，以及在挖掘一個或多個區塊之後，網路如何在公共區塊鏈上重新收斂。

在另一種情況下，網路可能會分化成以下兩個鏈：共識規則的改變。這種類型的分叉稱為 硬分叉 *hard fork*，因為在分叉之後，網路不會重新聚合到單個鏈上。相反，這兩條鏈是獨立進化的。當網路的一部分在一組不同於網路其他部分的共識規則下運行時，就會出現硬分叉。這可能是由於錯誤或共識規則執行過程中故意更改而導致的。

硬分叉可以用來改變共識規則，但是它們需要系統中所有參與者之間的協作。任何不升級到新的共識規則的節點都不能參與共識機制，並在硬分叉時被迫進入一個單獨的鏈。因此，硬分叉引入的更改可以被認為是不“向前兼容”的，因為在非升級的系統中不能再處理新的共識規則。

讓我們通過一個例子來查看硬分叉的結構。

[blockchainwithfork] 展示了區塊鏈和兩個分叉，在區塊高度為4的地方，出現一個單區塊分叉。這是我們在 [fork] 中看到的自發性分叉類型。在第5區塊的挖掘中，網路在一個鏈上重新聚合，分叉被解決。

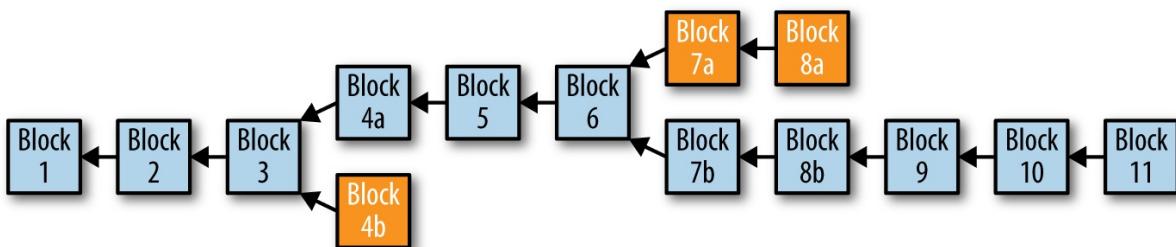


Figure 9. A blockchain with forks

然而，稍後，在區塊高6處，出現硬分叉。讓我們假設客戶端的新實現隨共識規則的更改而釋放。從區塊高度7開始，運行這個新實現的礦工將接受一種新的數字簽名類型，我們稱之為“Smores”簽名，它不是基於ECDSA的。緊接著，運行新實現的節點將創建一個包含Smores簽名的交易，並運行最新軟體的礦工，挖掘包含該交易的區塊7b。

任何沒有升級軟體以驗證Smores簽名的節點或礦工現在都無法處理第7b區塊。從他們的角度來看，包含Smores簽名的交易和包含該交易的block 7b都是無效的，因為它們是基於舊的共識規則進行評估的。這些節點將拒絕交易和區塊，不會傳播它們。任何使用舊規則的挖掘人員都不會接受第7b區塊，並將繼續挖掘父區塊為第6區塊的候選區塊。事實上，如果使用舊規則的礦工連接到的所有節點都遵守舊規則，因此不傳播該區塊，那麼他們甚至可能不會接收到block 7b。最終，他們將能夠挖掘第7a區塊，該區塊在舊規則下是有效的，不包含任何帶有Smores簽名的交易。

這兩條鏈從這一點開始繼續偏離。“b”鏈上的礦工將繼續接受和挖掘包含Smores簽名的交易，而“a”鏈上的礦工將繼續忽略這些交易。即使block 8b不包含任何經過Smores簽名的交易，“a”鏈上的礦工也不能處理它。對他們來說，它似乎是一個孤立的區塊，因為它的父“7b”不被認為是一個有效的區塊。

## 硬分叉：軟體，網路，挖礦，和鏈

對於軟體開發人員來說，“fork”這個詞還有另外一個含義，給“hard fork”這個詞增加了混淆。在開放源碼軟體中，當一組開發人員選擇遵循不同的軟體路線圖並啟動一個開放源碼項目的競爭性實現時，就會出現一個fork。我們已經討論了導致硬分叉的兩種情況：共識規則中的錯誤和共識規則的故意修改。在故意改變共識規則的情況下，軟分叉優先於硬分叉。然而，對於這種類型的硬分叉，必須開發、採用和啟動共識規則的新軟體實現。

試圖改變共識規則的軟體分叉的例子包括Bitcoin XT、Bitcoin Classic，以及最近的Bitcoin Unlimited。然而，這些軟體的分叉都沒有帶來一個硬的叉。雖然軟體fork是一個必要的先決條件，但它本身並不足以出現hard fork。要實現一個硬的fork，必須採用競爭的實現，以及由礦商、錢包和中間節點激活的新規則。相反，有許多Bitcoin Core的替代實現，甚至是軟體分支，它們不會改變共識規則，也不會排除bug，它們可以在網路上共存並互操作，而不會造成硬分叉。

共識規則在交易或區塊的驗證中可能以明顯和明確的方式存在差異。這些規則還可能在更微妙的方面有所不同，比如在適用於比特幣腳本或數字簽名等密碼基元的共識規則的實現上。最後，由於系統限制或實現細節所造成的隱式共識約束，共識規則可能會因未預期的方式而有所不同。在將比特幣核心0.7升級到0.8的過程中，在意料之外的硬分叉中可以看到後者的一個例子，這是由於用於儲存區塊的Berkley DB實現的限制造成的。

從概念上講，我們可以把硬分叉看作是分四個階段發展的：軟體分叉、網路分叉、挖礦分叉和鏈分叉。

當開發人員創建修改了共識規則的客戶端替代實現時，流程就開始了。

當這個分叉實現部署到網路中時，一定比例的礦工、錢包用戶和中間節點可以採用並運行這個實現。產生的分叉將取決於新的共識規則是否適用於區塊、交易或系統的其他方面。如果新的共識規則與交易相關，那麼在新規則下創建交易的錢包可能會生成一個網路分叉，然後當交易被挖掘到一個區塊時，會出現一個硬分叉。如果新規則涉及到區塊，那麼當在新規則下挖掘區塊時，硬分叉過程將開始。

首先，網路將會分叉。基於共識規則的原始實現的節點將拒絕在新規則下創建的任何交易和區塊。此外，遵循原始共識規則的節點將臨時禁止並斷開與發送這些無效交易和區塊的任何節點的連接。因此，網路將被劃分為兩個部分：舊節點只會繼續連接舊節點，新節點只會連接到新節點。基於新規則的單個交易或區塊將在網路中產生連鎖反應，並導致分成兩個網路。

一旦一名礦工使用新規則開採一個區塊，開採算力和鏈也將分叉。新礦工將在新區塊的頂部採礦，而老礦工將在舊規則的基礎上開採一個單獨的鏈條。分割後的網路將使在獨立的共識規則下操作的礦工不可能接收到彼此的區塊，因為他們連接到兩個獨立的網路。

## 分叉的礦工和難度

當礦工們分別開採兩個不同的鏈條時，它們的雜湊算力就會被分割開來。在這兩個鏈之間，可以按任何比例分割採礦算力。新規則可能只會被少數人遵守，或者被絕大多數算力的礦工遵守。

例如，我們假設 80% —— 20% 分割，大多數挖礦算力使用新的共識規則。假設分叉在重新設定目標的階段後立即開始。

這兩個鏈將從重新設定目標階段繼承難度。新的共識規則將有80%的先前的算力委託給他們。從這條鏈的角度來看，與前一時期相比，挖礦算力突然下降了20%。平均每12.5分鐘就會發現一些區塊，這代表了延伸這條鏈的挖礦算力下降了20%。這種區塊發行速度將持續下去（除非雜湊功率發生變化），直到2016年區塊被開採，這將需要大約25,200分鐘

(每區塊12.5分鐘) , 或17.5天。17.5天後, 重新設定目標, 難度調整 (減少20%) , 基於該鏈中雜湊算力的減少, 再次生成10分鐘的區塊。

在舊規則下, 只有20%的雜湊算力, 將面臨更加困難的任務。在這條鏈條上, 現在平均每50分鐘開採一區塊。接下來2016區塊礦的難度將不會調整, 這將需要100,800分鐘, 或者大約10周。假設每個區塊有固定的容量, 這也將導致交易容量減少5倍, 因為每小時記錄交易的區塊更少。

## 有爭議的硬分叉

這是共識軟體開發的曙光。正如開源開發改變了軟體的方法和產品, 並隨之創造了新的方法、新的工具和新的社區, 共識軟體開發也代表了電腦科學的一個新的前沿。從比特幣發展路線圖的辯論、實驗和磨難中, 我們將看到新的開發工具、實踐、方法和社區出現。

硬分叉被認為是有風險的, 因為它們迫使少數人要麼升級, 要麼留在少數人的鏈條上。許多人認為, 將整個系統分割成兩個相互競爭的系統的風險是不可接受的。因此, 許多開發人員不願意使用硬分叉機制來實現對共識規則的升級, 除非整個網路幾乎一致支持。任何沒有得到一致支持的硬分叉提議都被認為太“有爭議”, 不可能在不危及系統分割的情況下嘗試。

硬分叉的問題在比特幣開發界引起了極大的爭議, 尤其是當它涉及對控制最大區塊大小限制的共識規則的任何擬議更改時。一些開發人員反對任何形式的硬分叉, 認為這樣做風險太大。另一些人認為, 硬分叉機制是升級共識規則的重要工具, 以避免“技術債務”, 並與過去徹底決裂。最後, 一些開發人員認為硬分叉是一種應該很少使用的機制, 需要進行大量的預先規劃, 而且只有在達成幾乎一致的意見下才能使用。

我們已經看到了解決“硬分叉”風險的新方法的出現。在下一節中, 我們將討論軟分叉, 以及BIP-34和BIP-9方法, 用於通知和激活共識的修改。

## 軟分叉

並非所有共識規則的改變都會導致硬分叉。只有共識的變更是向前不兼容的, 才會產生一個分叉。如果更改的實現方式使得未修改的客戶端仍然認為交易或區塊在以前的規則下是有效的, 那麼更改可以在沒有分叉的情況下發生。

引入*soft fork*這一術語, 將這種升級方法與“硬分叉”區分開來。實際上, 軟分叉根本不是分叉。軟分是對共識規則的向前兼容更改, 允許未升級的客戶端繼續與新規則一致地運行。

軟分叉的一個方面並不明顯, 即軟分叉升級只能用於約束共識, 而不能擴展它們。為了向前兼容, 在新規則下創建的交易和區塊也必須在舊規則下有效, 反之亦然。新規則只能限制什麼是有效的; 否則, 在舊規則下被拒絕時, 它們將觸發硬分叉。

軟分叉可以通過多種方式實現——這個術語不指定特定的方法, 而是一組方法, 它們都有一個共同點: 不需要所有節點進行升級, 也不需要強制不升級的節點退出共識。

### 軟分叉重定義的 NOP 操作碼

基於對NOP操作碼的重新解釋, 比特幣已經實現了許多軟分叉。比特幣腳本中有10個操作碼, 從NOP1到NOP10。根據共識規則, 在腳本中存在這些操作碼被解釋為無效的操作符, 這意味著它們沒有效果。執行在NOP操作碼之後繼續, 就好像它不在那裡一樣。

因此, 軟fork可以修改NOP程式碼的語義, 從而賦予它新的含義。例如, BIP-65 ( CHECKLOCKTIMEVERIFY ) 重新解釋了NOP2操作碼。實現BIP-65的客戶端將NOP2解釋為OP\_CHECKLOCKTIMEVERIFY, 並對UTXO強加一個絕對鎖時間共識規則, 在他們的鎖定腳本中包含這個操作碼。這種更改是一個軟分叉, 因為在BIP-65下有效的交易對於任何不實現(不知道)BIP-65的客戶端也是有效的。對於舊的客戶端, 該腳本包含一個NOP程式碼, 該程式碼將被忽略。

### 軟分叉升級的其他方法

對NOP操作碼的重新解釋是有計劃的，也是共識升級的明顯機制。然而，最近又引入了另一種軟分叉機制，它不依賴於NOP操作碼進行非常特定的共識更改。這在 [segwit] 中有更詳細的討論。Segwit是對交易結構的架構更改，它將解鎖腳本（見證）從交易內部移動到外部資料結構（隔離它）。Segwit最初被設想為硬分叉升級，因為它修改了基本結構（交易）。2015年11月，一位致力於Bitcoin Core的開發人員提出了一種機制，可以將segwit作為一種軟分叉引入。用於此的機制是對在segwit規則下創建的UTXO的鎖定腳本的修改，這樣未修改的客戶端就可以通過任何解鎖腳本贖回鎖定腳本。因此，可以引入segwit，而不需要每個節點升級或從鏈上拆分：這是一個軟分叉。

很有可能還有其他的，但有待發現的機制，通過這些機制，升級可以以一種向前兼容的方式作為軟分叉。

## 針對軟分叉的批評

基於NOP操作碼的軟分叉相對來說是沒有爭議的。NOP操作碼放在比特幣腳本中，明確的目標是允許非破壞性升級。

然而，許多開發人員擔心其他的軟分叉升級方法會導致不可接受的折衷。對軟分叉變化的常見批評包括：

### 技術債務 *Technical debt*

因為軟分叉比硬分叉升級更復雜，他們引入了 *technical debt*，這個術語指的是由於過去的設計權衡而增加了未來的程式碼維護成本。程式碼的複雜性反過來又增加了錯誤和安全漏洞的可能性。

### 驗證放鬆 *Validation relaxation*

未修改的客戶端認為交易是有效的，而不評估修改後的共識規則。實際上，未修改的客戶端不使用所有共識規則進行驗證，因為它們對新規則視而不見。這適用於基於NOP的升級，以及其他軟分叉升級。

### 不可逆的更新 *Irreversible upgrades*

因為軟分叉創建具有附加共識約束的交易，它們在實踐中成為不可逆轉的升級。如果一個軟分叉升級在激活後被撤銷，根據新規則創建的任何交易都可能導致舊規則下的資金損失。例如，如果CLTV交易是在舊規則下計算的，則沒有timelock約束，可以在任何時候使用它。因此，批評人士認為，由於錯誤而不得不取消的失敗軟分叉幾乎肯定會導致資金損失。

## 使用區塊版本的軟分叉信號

由於軟分叉允許未修改的客戶端繼續在共識範圍內運行，因此“激活”軟分叉的機制是通過礦工發出準備就緒的信號：大多數礦商必須同意他們已經準備好並願意執行新的共識規則。為了協調他們的行動，有一個信號機制允許他們表示對共識規則變更的支持。該機制於2013年3月引入BIP-34激活，2016年7月被BIP-9激活取代。

## BIP-34 信號和激活

在BIP-34中，第一個使用了區塊版本號欄位的實現，允許礦工發出信號，表示已經準備好進行特定的共識規則更改。在BIP-34之前，區塊版本號根據約定設置為“1”，不以共識執行。

BIP-34定義了一個共識規則變更，要求幣基交易的coinbase欄位（輸入）包含區塊高度。在BIP-34之前，這個欄位可以包含任意數據。激活BIP-34之後，有效的區塊必須在coinbase的起始處包含特定的區塊高度，並用大於或等於“2”的版本號進行標識。

為了表明BIP-34的變化和激活，礦工將區塊版本設置為“2”而不是“1”。這並沒有立即使版本“1”區塊無效。一旦激活，版本“1”區塊將變得無效，並且所有版本“2”區塊將被要求在coinbase欄位中包含區塊高度才有效。

BIP-34基於1000區塊的滾動窗口定義了兩步激活的機制。礦工將通過構建具有“2”作為版本號的區塊來表示他或她對BIP-34的個人準備狀態。嚴格地說，這些區塊並不需要遵守在幣基交易中包含區塊高度的新共識規則，因為共識規則尚未激活。共識規則分兩步啟動：

- 如果75%（最近1000個區塊中的750個）標有版本“2”，則版本“2”區塊必須在幣基交易中包含區塊高度，否則它們將被視為無效。版本“1”區塊仍然被網路接受，並且不需要包含區塊高度。新舊共識規則在此期間共存。

- 當95%（最近1000個區塊中的950個）為版本“2”時，版本“1”區塊不再被視為有效。版本“2”區塊只有在幣基中包含區塊高時（按照之前的閾值）才有效。此後，所有區塊必須符合新的共識規則，並且所有有效區塊必須包含在幣基交易中的區塊高度。

在BIP-34規則下成功發送信號和激活後，該機制被再次使用兩次以激活軟分叉：

- [BIP-66](#) 嚴格的簽名DER編碼通過BIP-34類型的發信號以區塊版本“3”激活並且使版本“2”的區塊無效。
- [BIP-65](#) CHECKLOCKTIMEVERIFY +被BIP-34風格的信號激活，區塊版本為“4”，並使版本“3”的區塊失效。

在激活BIP-65後，BIP-34的信號傳導和激活機制退役並被接下來描述的BIP-9信號傳導機制取代。

標準定義在 [BIP-34 \(Block v2, Height in Coinbase\)](#) 中。

## BIP-9 信號和激活

BIP-34，BIP-66和BIP-65使用的機制成功地激活了三個軟分叉。但是，它被替換了，因為它有幾個限制：

- 通過使用區塊版本的整數值，一次只能激活一個軟分叉，因此需要軟分叉建議之間的協調，並就其優先級和排序達成一致。
- 此外，由於區塊版本增加，該機制沒有提供直接的方式來拒絕更改，然後提出不同的更改。如果老客戶端仍在運行，他們可能會錯誤地將信號發送給新的更改，作為先前拒絕的更改的信號。
- 每次新的更改不可避免地減少了未來更改的可用區塊版本。

BIP-9 被提出以克服這些挑戰，提高實施未來變革的速度和容易度。

BIP-9 將區塊版本解釋為位域而不是整數。由於區塊版本最初被用作整數使用，版本號1至4，因此有29位可用的位，可用於獨立同時發出29個不同提案的準備就緒信號。

BIP-9 還設置信號和激活的最長時間。這樣，礦工不需要永遠發出信號。如果提案在 TIMEOUT 期限內（在提案中定義）未激活，則提議被視為被拒。該建議可能會重新提交以用不同的位進行信號傳輸，從而更新激活期。

此外，在超過 TIMEOUT 並且某個功能被激活或拒絕之後，信號位可以被重新用於其他功能而不會造成混淆。因此，多達29個變化可以並行發送，TIMEOUT 之後可以“循環”以提出新的更改。

### Note

雖然信號位可以重複使用或循環使用，但只要投票期不重疊，BIP-9的作者建議只有在必要時才能重新使用位；由於舊軟體中的錯誤而可能發生意外行為。總之，我們不應該期望看到重用，直到所有29位被使用過一次。

建議的更改由包含以下欄位的資料結構標識：

### *name*

用於區分提案的簡短說明。大多數情況下，描述該建議的BIP為“bipN”，其中N是BIP編號。

### *bit*

0 到 28, 矿工用來表示批准該提案的區塊版本中的位。

### *starttime*

信號開始的時間（基於過去中值時間，MTP），在此之後，該位的值被解釋為該提議的信號準備就緒。

### *endtime*

如果更改未達到激活閾值，則認為拒絕的時間點（基於MTP）。

與BIP-34不同，BIP-9基於2016個區塊的難度重新設定目標階段計算整個區間的激活信號。對於每個重新定位週期，如果支持提案的信號的區塊的總數超過95%（2016區塊中1916區塊），則該提議將在下一個重新設定目標階段內激活。

BIP-9 提供了一個提案狀態圖，以說明提案的各個階段和狀態轉換，如 [BIP-9 state transition diagram](#) 中所示。

一旦參數在比特幣軟體中已知（定義），提案就以 DEFINED 狀態開始。對於MTP在開始時間之後的區塊，提案狀態將轉換為 STARTED。如果在重新設定目標週期內超過投票閾值並且沒有超時，則提案狀態轉換為 LOCKED\_IN。一個重新設定目標週期後，提案變為 ACTIVE。一旦它們達到該狀態，建議一直保持 ACTIVE 狀態。如果在達到投票閾值之前超時，提案狀態更改為 FAILED，表示被拒絕的提案。FAILED 提案永遠處於該狀態。

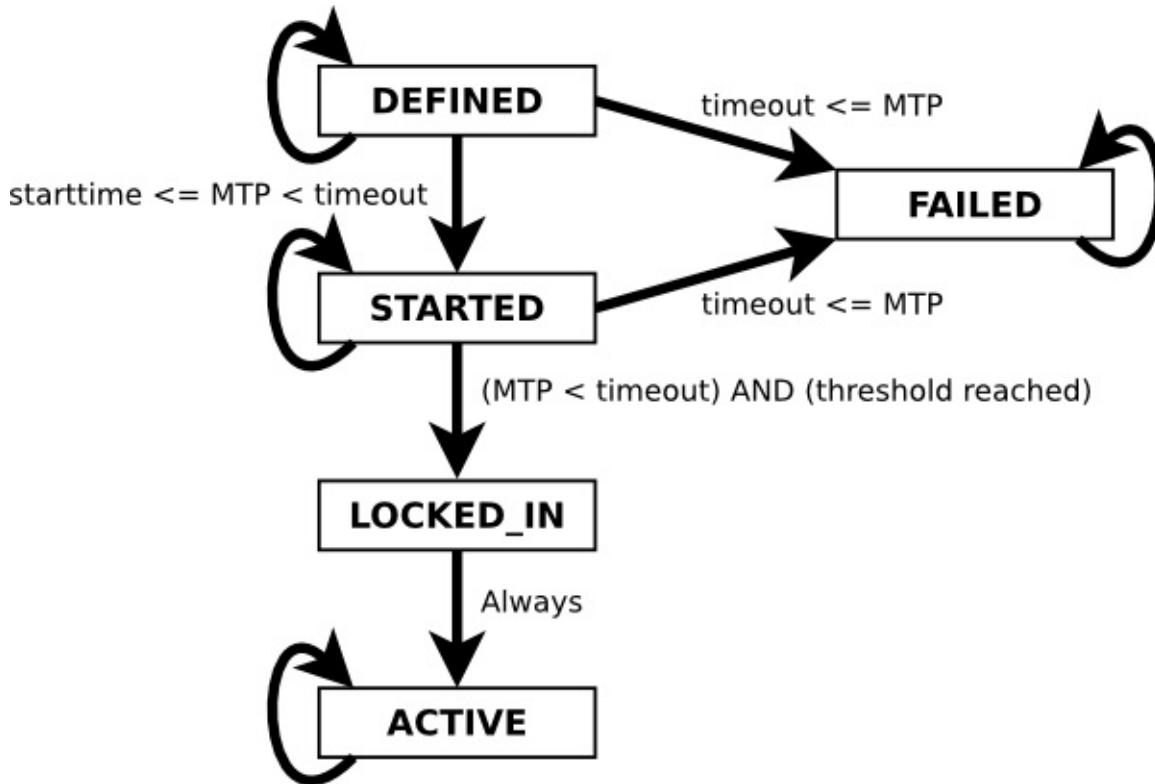


Figure 10. BIP-9 state transition diagram

BIP-9 首先用於激活 CHECKSEQUENCEVERIFY 和相關的 BIP (68,112,113)。名為“csv”的提案已於2016年7月成功啟用。

標準定義在 [BIP-9 \(Version bits with timeout and delay\)](#) 中。

## 共識軟體開發

共識軟體不斷髮展，關於改變共識規則的各種機制的討論很多。就其本質而言，比特幣在協調和共識變化方面設置了非常高的標準。作為一個分散的系統，它沒有“權威”，可以將其意志強加給網路的參與者。權力分散在礦工，核心開發人員，錢包開發人員，交易所，商戶和最終用戶等多個選區之間。任何這些選區都不能單方面做出決定。例如，儘管礦工理論上可以通過簡單的大多數（51%）改變規則，但他們受到其他選區同意的限制。如果他們單方面行事，其他參與者可能會拒絕跟隨他們，從而使經濟活動停留在少數鏈中。如果沒有經濟活動（交易，商人，錢包，交易所），礦工們將會挖到毫無價值的硬幣。權力的分散意味著所有的參與者必須協調，否則就不能做出改變。現狀是這個系統的穩定狀態，如果大多數人達成共識，則系統處於只有很少變化的穩定狀態。軟分叉的95%閾值反映了這一現實。

認識到共識的發展沒有完美的解決方案是很重要的。硬分叉和軟分叉都涉及權衡。對於某些類型的變化，軟分叉可能是更好的選擇；對於其他人來說，硬分叉可能是更好的選擇。沒有完美的選擇；都有風險。共識軟體開發的一個不變特徵是變革難度大，共識強制妥協。

一些人認為這是共識體系的弱點。隨著時間的推移，你可能會像我一樣，認為它是這個系統最強大的力量。

## 比特幣的安全

確保比特幣安全具有挑戰性，因為比特幣不是抽象的價值參考，就像銀行賬戶的餘額。比特幣非常像數字現金或黃金。你可能聽到過這樣的表達：“佔有是法律的十分之九”。那麼，在比特幣中，佔有是法律的十分之一。擁有解鎖比特幣的密鑰相當於擁有現金或貴重金屬。你可能會失去它，放錯位置，被盜，或者不小心把錯誤的數字給別人。在每一種情況下，用戶都沒有追索權，就好像他們在公共人行道上放棄了現金一樣。

但是，比特幣具有現金，黃金和銀行賬戶不具備的能力。包含您的密鑰的比特幣錢包可以像任何檔案一樣備份。它可以儲存在多個副本中，甚至可以打印在紙張上進行硬拷貝備份。您不能“備份”現金，黃金或銀行賬戶。比特幣與之前的任何東西都不相同，我們需要以一種新穎的方式來思考比特幣安全。

### 安全原則

比特幣的核心原則是去中心化，它對安全性有重要影響。傳統的銀行或支付網路等集中模式依賴於訪問控制和審查，以防止不良行為者。相比之下，像比特幣這樣的分散系統將責任和控制交給了用戶。由於網路的安全性基於工作量證明，而不是訪問控制，因此網路可以公開，並且比特幣流量不需要加密。

在諸如信用卡系統的傳統支付網路中，支付是開放式的，因為它包含用戶的私人標識符（信用卡號碼）。在初始收費後，任何可以使用該標識符的人都可以“拉取”資金並一次又一次地向擁有者收費。因此，支付網路必須通過加密進行端對端的安全保護，必須確保沒有竊聽者或中間人可以危害付款流量，包括傳輸中或儲存時（靜止時）。如果一個不良行為者獲得了訪問系統的權限，他可以危害當前的交易，以及可用於創建新交易的支付令牌。更糟糕的是，當客戶數據遭到破壞時，客戶面臨身份盜用的風險，必須採取措施防止被盜賬戶的欺詐性使用。

比特幣有很大的不同。比特幣交易僅向特定收件人授予特定的價值，不能偽造或修改。它不會透露任何隱私訊息，例如當事人的身份，也不能用於授權額外付款。因此，比特幣支付網路不需要加密或防止竊聽。事實上，您可以通過開放的公共頻道（例如不安全的WiFi或藍牙）來廣播比特幣交易，同時不會降低安全性。

比特幣的去中心化安全模式將大量權力交給用戶。有了這個權力就有責任對密鑰進行保密。對於大多數用戶來說，這並不容易，特別是在通用計算設備上，例如連接互聯網的智能手機或筆記本電腦。雖然比特幣的去中心化模式可以防止向信用卡那樣出現大規模危機，但許多用戶無法充分保護他們的密鑰，並一個接一個地被黑客入侵。

### 安全地開發比特幣系統

比特幣開發者最重要的原則是去中心化。大多數開發人員都熟悉集中式安全模型，並可能試圖將這些模型應用到他們的比特幣應用程式中，從而帶來災難性後果。

比特幣的安全性依賴於對密鑰的去中心化控制以及礦工的獨立交易驗證。如果你想利用比特幣的安全性，你需要確保你保持在比特幣安全模型中。簡而言之：不要將密鑰從用戶手中接管，也不要將交易從區塊鏈中移除。

例如，許多早期的比特幣交易所將所有用戶資金集中在單個“熱”錢包中，密鑰儲存在單個伺服器上。這種設計消除了用戶的控制權，並集中控制單個系統中的密鑰。許多此類系統已被黑客入侵，對客戶造成災難性後果。

另一個常見的錯誤是，為了減少交易費用或加速交易處理，錯誤地將交易“脫離區塊鏈”。一個“離線的區塊鏈”系統將在內部的集中式賬本上記錄交易，並且偶爾將它們同步到比特幣區塊鏈上。這種做法再次以專有和集中的方式取代了去中心化的比特幣安全性。當交易離開區塊鏈時，安全措施不當的集中賬本可能被偽造，導致資金流失和儲備枯竭，而不被人注意到。

除非您準備在運營安全，多層訪問控制和審計（如傳統銀行所做的那樣）上投入大量資金，否則在將資金用於比特幣的去中心化環境外之前，應該仔細考慮。即使你有足夠的資金和規則來實施強大的安全模式，這樣的設計也只是複製了傳統金融網路的脆弱模式，這種模式受到身份盜用，腐敗和貪汙的困擾。為了利用比特幣獨特的去中心化安全模式，您必須避免可能會感到熟悉但最終顛覆比特幣安全性的集中式體系結構的誘惑。

### 信任根 Root of Trust

傳統的安全體系結構基於一個稱為 信任根 *root of trust* 的概念，它是一個可信的核心，用作整個系統或應用程式安全的基礎。安全架構是圍繞信任根發展起來的，是一系列同心圓，如洋蔥中的層，從中心向外擴展信任。每層使用訪問控制，數字簽名，加密和其他安全基元構建更可信的內層。隨著軟體系統變得越來越複雜，它們更容易包含錯誤，這使得它們容易受到安全危害。結果，軟體系統越複雜，就越難保證安全。信任根的概念可以確保大多數信任被放置在系統中最不復雜的部分，系統中最不易受影響的部分，更復雜的軟體則圍繞著它。這種安全體系結構在不同規模上重複出現，首先在單個系統的硬體中建立信任根，然後通過操作系統將信任根擴展到更高級的系統服務，最後跨多個伺服器減少信任。

比特幣安全架構不同。在比特幣中，共識系統創建了一個完全分散的可信公共賬本。經過正確驗證的區塊鏈使用創世區塊作為信任根，建立一個直至當前區塊的信任鏈。比特幣系統可以也應該使用區塊鏈作為他們的信任根。在設計由許多不同系統上的服務組成的複雜比特幣應用程式時，您應仔細檢查安全架構，以確定信任的位置。最終，唯一應該明確信任的是完全驗證的區塊鏈。如果您的應用程式明確或隱含地信任除區塊鏈之外的任何其他訊息，那麼應該引起關注，因為它會引入漏洞。評估應用程式的安全體系結構的一個好方法是考慮每個單獨的組件並評估一個假想的場景，其中該組件完全受到攻擊並處於惡意行為者的控制之下。依次評估應用程式的每個組件受到危害時對整體安全性的影響。如果您的應用程式在組件受損時不再安全，則表明您錯誤地信任這些組件。一個沒有漏洞的比特幣應用程式應該只會損害比特幣共識機制，這意味著它的信任根源是比特幣安全架構中最強大的部分。

眾多被黑客入侵的比特幣交易所的例子有助於強調這一點，因為即使在最偶然的審查下，其安全架構和設計也會失敗。這些集中式實施已經將信任明確地投入到比特幣區塊鏈以外的眾多組件中，比如熱錢包，集中化賬本資料庫，易受攻擊的加密密鑰以及類似的模式。

## 用戶安全最佳實踐

人類已經使用了數千年的物理安全控制。相比之下，我們在數字安全方面的經驗還不到50年。現代通用操作系統並不是非常安全，並不適合儲存數字貨幣。我們的電腦通過永遠在線的互聯網連接不斷暴露於外部威脅下。他們運行數百個作者的數千個軟體組件，通常無限制地訪問用戶的檔案。在您的電腦上安裝的數千個軟體中，一個流氓軟體，可能會危及您的鍵盤和檔案，竊取儲存在錢包應用程式中的任何比特幣。保持電腦無病毒和無木馬所需的電腦維護技能超出了大多數電腦用戶的水平。

儘管有數十年來訊息安全方面的研究和進步，但數字資產仍然很容易受到堅定的對手的攻擊。即使是在金融服務公司，情報機構和國防承包商中受到高度保護和限制的系統也經常遭到侵犯。比特幣創造的數字資產具有內在價值，可以被盜取，並立即不可撤銷地轉移給新的所有者。這為黑客創造了一個巨大的誘因。到目前為止，黑客在入侵之後不得不將身份訊息或賬戶令牌（如信用卡和銀行賬戶）轉換為價值。儘管銷贓和洗錢很困難，但我們發現盜竊案日益增多。比特幣升級了這個問題，因為它不需要被銷贓或洗錢；這是數字資產的內在價值。

幸運的是，比特幣也創造了提高電腦安全性的動力。之前電腦被入侵的風險是模糊和間接的，比特幣使這些風險變得清晰明瞭。在電腦上持有比特幣有助於將用戶的注意力集中在提高電腦安全性的需求上。作為比特幣和其他數字貨幣的激增和採用的直接結果，我們看到了黑客技術和安全解決方案的升級。簡而言之，黑客現在有了一個非常肥美的目標，而用戶也有了明確的動機來保護自己。

在過去的三年中，作為比特幣應用的直接結果，我們看到了以硬體加密，密鑰儲存和硬體錢包，多重簽名技術以及數字託管等形式在訊息安全領域的巨大創新。在下面的章節中，我們將研究實際用戶安全的各種最佳實踐。

## 物理比特幣儲存

讓比特幣離線稱為 *cold storage*，它是最有效的安全技術之一。冷儲存系統是在離線系統（從未連接到互聯網）上生成密鑰並在紙上或數字媒體（如USB記憶棒）上離線儲存的系統。

## 硬體錢包

從長遠來看，比特幣安全越來越多地採取硬體防篡改錢包的形式。與智能手機或臺式電腦不同，比特幣硬體錢包只有一個目的：安全地持有比特幣。沒有通用軟體的危害並且接口有限，硬體錢包可以為非專業用戶提供幾乎萬無一失的安全級別。我期望看到硬體錢包成為比特幣儲存的主要方法。有關這種硬體錢包的示例，請參閱 [Trezor](#)。

## 平衡風險

雖然大多數用戶都對比特幣失竊有正確的擔憂，但風險更大。數據檔案一直在丟失。如果他們包含比特幣，損失會更加痛苦。為了保證他們的比特幣錢包的安全，用戶必須非常小心，不要太過分，而最終會失去比特幣。2011年7月，一個眾所周知的比特幣意識和教育項目損失了近7,000比特幣。為了防止盜竊，擁有者們實施了一系列複雜的加密備份。最後，他們意外地丟失了加密密鑰，使得備份毫無價值並且失去了一筆財富。就像把錢埋在沙漠中一樣，如果你過於仔細地保護你的比特幣，你可能無法再找到它。

## 分散風險

你會把你的全部資產以現金形式放入你的錢包嗎？大多數人會認為魯莽，但比特幣用戶經常把所有的比特幣放在一個錢包裡。相反，用戶應該將風險分散到多種多樣的比特幣錢包中。謹慎的用戶將比特幣的一小部分，或許不到5%的比特幣保留在在線或移動錢包中作為“零錢”。剩下的應該分成幾種不同的機制儲存，比如桌面錢包和離線（冷儲存）。

## 多重簽名和治理

每當公司或個人儲存大量比特幣時，他們應該考慮使用多重簽名比特幣地址。多重簽名通過要求多個簽名進行付款來解決資金安全問題。簽名密鑰應儲存在多個不同位置，並由不同人員控制。例如，在公司環境中，密鑰應該由多個公司管理人員獨立生成並保存，以確保任何人都不會損害資金。多重簽名地址也可以提供冗餘，即一個人擁有多個儲存在不同位置的密鑰。

## 生存性

經常被忽視的一個重要的安全因素是可用性，特別是在密鑰持有者無能力或死亡的情況下。比特幣用戶被告知使用複雜的密碼，並保證他們的密鑰安全和私密，而不與任何人分享。不幸的是，如果用戶無法解鎖，那麼這種做法幾乎不可能讓用戶的家人恢復任何資金。事實上，在大多數情況下，比特幣用戶的家族可能完全不知道比特幣資金的存在。

如果你有很多比特幣，你應該考慮與受信任的親戚或律師分享訪問細節。可以通過專門的稱為“數字資產執行者”的律師，使用多重簽名訪問和資產規劃設立更復雜的生存性計劃。

## 總結

比特幣是一種全新的，前所未有的複雜技術。隨著時間的推移，我們將開發更好的安全工具和實踐，使非專業人員更容易使用。目前，比特幣用戶可以使用這裡討論的許多技巧來享受安全且無故障的比特幣體驗。

## 區塊鏈應用

現在讓我們通過將其視為 應用平臺 *application platform* 來構建我們對比特幣的理解。如今，許多人使用術語“區塊鏈”來指任何共享比特幣設計原則的應用平臺。這個術語經常被濫用，並且被應用於許多未能交付比特幣區塊鏈提供的主要功能的東西。

在本章中，我們將看看比特幣區塊鏈作為應用程式平臺提供的功能。我們將考慮應用程式構建 原語 *primitives*，構成了區塊鏈應用程式的構建模塊。我們將看看使用這些原語的幾個重要應用程式，例如彩色幣（Colored coins），支付（狀態）通道和路由支付通道（閃電網路 Lightning Network）。

### 簡介

比特幣系統被設計成一個去中心化的貨幣和支付系統。然而，它的大部分功能都來源於更低層次的構造，可用於更廣泛的應用程式。比特幣不是由諸如賬戶，用戶，餘額和支付等組件構建的。相反，它使用具有低級加密函數的交易腳本語言，就像我們在 [transactions] 中看到的那樣。正如帳戶，餘額和支付的更高級概念可以從這些基本原語中派生出來一樣，許多其他複雜的應用程式也是如此。因此，比特幣區塊鏈可以成為一個應用平臺，為智能合約等應用提供信託服務，遠超數字貨幣和支付的原始目的。

### 構建模塊（原語）

長期正確運行時，比特幣系統提供了一定的保證，可用作構建模塊來創建應用程式。這些包括：

#### 無雙重支付 *No Double-Spend*

比特幣去中心化共識演算法的最基本的保證，確保UTXO不會花費兩次。

#### 不變性 *Immutability*

一旦交易記錄在區塊鏈中，並且後續的區塊已經添加了足夠的工作量，交易的數據就變得不可變。不變性由能源保證，因為重寫區塊鏈需要耗費能源來生產工作量證明。所需的能量以及不變程度隨著包含交易的區塊頂部的工作量而增加。

#### 中立性 *Neutrality*

去中心化比特幣網路傳播有效交易，無論這些交易的來源或內容如何。這意味著任何人都可以創建一個有足夠費用和信任的有效交易，他們可以隨時傳輸該交易並將其包含在區塊鏈中。

#### 安全時間戳 *Secure Timestamping*

共識規則拒絕任何有離得太遠的過去或未來的時間戳的區塊。這確保區塊上的時間戳可以被信任。區塊上的時間戳意味著對所有包含的交易的輸入的未使用保證。

#### 授權 *Authorization*

數字簽名經過去中心化網路驗證，可提供授權保證。包含數字簽名要求的腳本不能在腳本中隱含的私鑰持有者未經授權的情況下執行。

#### 可審計性 *Auditability*

所有交易都是公開的，可以進行審計。所有的交易和區塊都可以鏈接到創世區塊。

#### 會計 *Accounting*

在任何交易中（除了coinbase交易），輸入的價值等於輸出加上費用的價值。在交易中創建或銷燬比特幣值是不可能的。輸出不能超過輸入。

#### 不過期 *Nonexpiration*

有效的交易不會過期。如果它今天有效，只要輸入沒有被消耗並且共識規則沒有改變，它將在不久的將來也有效。

#### 完整性 *Integrity*

使用 SIGHASH\_ALL 簽署的比特幣交易或由另一種 SIGHASH 類型簽署的交易的部分不能在未使簽名無效的情況下進行修改，修改將導致交易本身無效。

交易原子性 Transaction Atomicity:: 比特幣交易是原子的。它們要麼是有效並被確認的（挖掘），要麼不是。不完整的交易不能開採，交易沒有臨時狀態。在任何時候，交易都是開採或不開採。

#### 離散（不可分）單位的價值 *Discrete (Indivisible) Units of Value*

交易的輸出是離散的和不可分割的價值單位。它們可以全部用完或不使用，不能分開或部分使用。

#### 控制法定人數 *Quorum of Control*

腳本中的多重簽名限制了多重簽名方法中預定義的法定權限。M-N要求由共識規則強制執行。

#### 時間鎖/老化 *Timelock/Aging*

包含相對或絕對時間鎖的任何腳本語句只能在其年齡超過指定時間後才能執行。

#### 複製 *Replication*

區塊鏈的分散儲存確保了在交易開始時，經過充分的確認後，它可以在整個網路中進行複制，是持久化的，並且對斷電，數據丟失等具有恢復能力。

#### 防偽 *Forgery Protection*

交易只能花費現有的已驗證輸出。不可能創造或偽造價值。

#### 一致性 *Consistency*

在沒有礦工分區的情況下，記錄在區塊鏈中的區塊會根據它們被記錄的深度而受到重組或不一致的可能性呈指數下降。一旦深刻記錄，改變所需的計算和能量使得改變幾乎不可行。

#### 記錄外部狀態 *Recording External State*

交易可以通過 OP\_RETURN 提交一個數據值，表示一個外部狀態機器中的狀態轉換。

#### 可預見的發行 *Predictable Issuance*

不到2100萬比特幣將以可預測的速度發行。

構建模塊的列表並不完整，會隨每一項引入到比特幣的新功能而增加。

## 構建模塊的應用

比特幣提供的構建模塊是可用於構成應用程式的信任平臺的組成部分。以下是當前存在的應用程式示例以及它們使用的構建區塊：

#### 存在性證明（數字公證） *Proof-of-Existence (Digital Notary)*

不變性時間戳持久化。數字指紋可以通過交易提交給區塊鏈，證明檔案在記錄時存在（時間戳）。指紋不能在事後修改（不可變性），並且證明將被永久保存（持久化）。

#### 眾籌平臺（閃電網路） *Kickstarter (Lighthouse)*

一致性原子性完整性。如果你簽署一項輸入和一項眾籌交易的輸出（完整性），其他人可以為籌款捐款，但直到目標（輸出價值）達到（一致性）後才能支付（原子性）。

#### 支付通道 *Payment Channels*

控制法定人數時間鎖定無雙重支付不過期審查阻力+授權。具有時間鎖定（時間鎖定）作為支付通道的“結算”交易的多重簽名2-of-2（控制法定人數）可以由任何一方（授權）在任何時間（審查阻力）持有（不過期）。然後，雙方可以創建承諾交易，在較短的時間間隔（Timelock）上雙重支付花費（無雙重支付，雙重支付可使之前的交易無效）“結算”。

## 彩色幣

我們將要討論的第一個區塊鏈應用是 彩色幣 *colored coins*。

彩色幣是指使用比特幣交易記錄比特幣以外的資產的創建，所有權和轉讓的一組類似技術。所謂“外部”，我們指的不是直接儲存在比特幣區塊鏈中的資產，不是比特幣本身，這是區塊鏈固有的資產。

彩色幣用於追蹤數字資產以及第三方持有的有形資產，並通過彩色幣進行所有權交易。數字資產彩色幣可以代表無形資產，如股票證書，許可證，虛擬財產（遊戲物品）或大多數任何形式的許可知識產權（商標，版權等）。有形資產的彩色幣可以代表商品（金，銀，油），土地所有權，汽車，船隻，飛機等的所有權證書。

這個術語來源於“著色”或標記比特幣的名義數量的想法，例如單一的satoshi，代表比特幣價值本身以外的其他東西。作為一個類比，考慮在1美元鈔票上加上一個訊息，說明“這是ACME的股票證書”或“這張鈔票可以兌換1盎司白銀”，然後交易1美元鈔票作為其他資產所有者的證書。第一個彩色幣的實現，名為 *Enhanced Padded-Order-Based Coloring* 或 *EPOBC*，將外部資產分配到1-satoshi輸出。通過這種方式，這是一個真正的“彩色幣”，因為每個資產都被添加為一個單獨的屬性（顏色）。

最近的彩色幣實現使用 OP\_RETURN 腳本操作碼在交易中儲存元數據，與將元數據關聯到特定資產的外部數據儲存一起使用。

如今兩個最出色的彩色幣實現是 [Open Assets](#) 和 [Colored Coins by Colu](#)。這兩個系統使用不同的方法來處理彩色幣，且不兼容。在一個系統中創建的彩色幣不能在另一個系統中看到或使用。

## 使用彩色幣

彩色幣通常在特殊錢包中創建，轉移和查看，這些幣可以解釋附加到比特幣交易的彩色幣協議元數據。必須特別注意避免在常規比特幣錢包中使用與彩色幣相關的密鑰，因為常規錢包可能會破壞元數據。同樣，不應將彩色幣發送到由常規錢包管理的地址，只能發送給由可識別彩色幣的錢包管理的地址。Colu和Open Assets系統都使用特殊的彩色幣地址來降低這種風險，並確保彩色硬幣不會發送給未知的錢包。

對於大多數通用區塊鏈瀏覽器來說，彩色幣也是不可見的。相反，你必須使用彩色幣資源管理器來解析彩色幣交易的元數據。

可以在 [coinprism](#) 找到與 Open Assets 兼容的錢包應用程式和區塊鏈瀏覽器。

Colu彩色幣兼容的錢包應用程式和區塊鏈瀏覽器可以在這裡找到：[Blockchain Explorer](#).

[Colored Coins Copay Addon](#) 是一個 Copay 錢包插件。

## 發行彩色幣

每個彩色幣的實現都有不同的創建彩色幣的方式，但它們都提供了類似功能。創建彩色幣資產的過程稱為 發行 *issuance*。初始交易，*issuance transaction* 將資產註冊在比特幣區塊鏈中，並創建一個用於引用資產的 *asset ID*。一旦發佈，可以使用 *transfer transactions* 在地址之間轉移資產。

作為彩色幣發行的資產可以有多個屬性。它們可以是 *divisible* 或 *indivisible*，意味著傳輸中資產的數量可以是整數（例如5）或小數（例如4.321）。資產也可以有 *fixed issuance*，也就是說一定金額只發行一次，或者 *reissued*，意味著資產的新單位可以在初始發行後由原發行人發行。

最後，一些彩色幣支持 *dividends*，允許將比特幣付款按所有權比例分配給著色貨幣資產的所有者。

## 彩色幣交易

給彩色幣交易賦予意義的元數據通常使用 OP\_RETURN 操作碼儲存在其中一個輸出中。不同的彩色幣協議對 OP\_RETURN 數據的內容使用不同的編碼。包含 OP\_RETURN 的輸出稱為 標記輸出 *marker output*。

輸出的順序和標記輸出的位置在彩色硬幣協議中可能有特殊含義。例如，在 Open Assets 中，標記輸出之前的任何輸出均表示資產發放，之後的任何輸出都代表資產轉移。標記輸出通過引用交易中的順序來為其他輸出分配特定的值和顏色。

作為對比，在 Colu 中，標記輸出對決定元數據解釋方式的操作碼進行編碼。操作碼0x01至0x0F指示發行交易。發行操作碼通常後面跟著一個資產ID或其他標識符，可用於從外部來源（例如，bittorrent）檢索資產訊息。操作碼0x10到0x1F表示轉移交易。轉移交易元數據包含簡單的腳本，通過參考其索引，將特定數量的資產從輸入轉移到輸出。輸入和輸出的排序對於腳本的解釋非常重要。

如果元數據太長以至於無法放入 OP\_RETURN，彩色幣協議可能會使用其他“技巧”在交易中儲存元數據。示例包括將元數據放入贖回腳本中，然後加上 OP\_DROP 操作碼以確保腳本忽略元數據。另一種使用的機制是1-of-N 多重簽名腳本，其中只有第一個公鑰是真正的公鑰，可以花費輸出，隨後的“密鑰”被編碼的元數據替代。

為了正確解釋彩色硬幣交易中的元數據，你必須使用兼容的錢包或區塊瀏覽器。否則，交易看起來像是一個帶有 OP\_RETURN 輸出的“普通”比特幣交易。

作為一個例子，我使用彩色幣創建併發布了MasterBTC資產。MasterBTC資產代表本書免費副本的代金券。這些優惠券可以使用彩色幣兼容的錢包進行轉讓，交易和兌換。

對於這個特定的例子，我使用了 <https://coinprism.info> 上的錢包和瀏覽器，它使用Open Assets彩色幣協議。

The issuance transaction as viewed on coinprism.info 使用Coinprism區塊瀏覽器展示了發行交易：

<https://www.coinprism.info/tx/10d7c4e022f35288779be6713471151ede967caaa39eec35296aa36d9c109ec>

## Transaction

Hash	10d7c4e022f35288779be6713471151ede967c...	Transaction confirmed
Date	Sunday, August 17, 2014 5:42:41 PM	Confirmations 137057 confirmations
Fee paid	0.0001 BTC	Time Sunday, August 17, 2014 5:...
Assets transacted	1	Block 00000000000000000150ab5...
		Height 316117
<hr/>		
Bitcoin		
 ↘ akTnsDt5uzpioRST76VFRQM8q8sBF...	-0.0001	Fees 0.0001
Free copy of "Mastering Bitcoin"		AcuRVsoa81hoLHmVTNxrd8KpTqUxe...
 + Issued assets -20 ↗ akTnsDt5uzpioRST76VFRQM8q8sBFnQ...	20	20

Figure 1. The issuance transaction as viewed on coinprism.info

如你所見，coinprism展示了20個“Mastering Bitcoin”比特幣的免費副本”MasterBTC資產發佈到一個特殊的彩色幣地址：

akTnsDt5uzpioRST76VFRQM8q8sBFnQiwcX

Warning

發送到此地址的任何資金或有色資產將永遠丟失。不要將價值發送到這個示例地址！

發行交易的交易ID是“正常”的比特幣交易ID。[The issuance transaction on a block explorer that doesn't decode colored coins](#) 在不能解析彩色幣的區塊瀏覽器中顯示相同的交易。我們將使用blockchain.info：

<https://blockchain.info/tx/10d7c4e022f35288779be6713471151ede967caaa39eec35296aa36d9c109ec>

## Transaction

View information about a bitcoin transaction

10d7c4e022f35288779be6713471151ede967caaa39eecd35296aa36d9c109ec		
1HpyyIGaXLq7ZCHk3s9EkVFoG15oLn2Us (0.01 BTC - Output)	1HpyyIGaXLq7ZCHk3s9EkVFoG15oLn2Us - (Unspent) Unable to decode output address - (Unspent)	0.000000 BTC 0 BTC 0.009894 BTC 0.00999999999999999 BTC
		0.0099 BTC

Figure 2. The issuance transaction on a block explorer that doesn't decode colored coins

如你所見，blockchain.info不會將其識別為彩色幣交易。實際上，它用紅色字母標記第二個輸出為“無法解碼輸出地址”。

如果你選擇 "Show scripts & coinbase"，你會看到交易的更多訊息 ([The scripts in the issuance transaction](#))。

### Output Scripts

OP_DUP OP_HASH160 b895201a7cfdd91a9bff3b42cd114d42e3a634d2 OP_EQUALVERIFY OP_CHECKSIG	OK
OP_RETURN 4f41010001141b753d68747470733a2f2f6370722e736d2f466f796b777248365559 (decoded) "OA_____u=https://cpr.sm/FoykwrH6UY	Strange
OP_DUP OP_HASH160 b895201a7cfdd91a9bff3b42cd114d42e3a634d2 OP_EQUALVERIFY OP_CHECKSIG	OK

Figure 3. The scripts in the issuance transaction

blockchain.info 還是不理解第二個輸出。它用紅色字母中的“Strange”標記。但是，我們可以看到標記輸出中的一些元數據是人類可讀的

```
OP_RETURN 4f41010001141b753d68747470733a2f2f6370722e736d2f466f796b777248365559
(decoded) "OA_____u=https://cpr.sm/FoykwrH6UY
```

讓我們使用 bitcoin-cli 檢索交易：

```
$ bitcoin-cli decoderawtransaction `bitcoin-cli getrawtransaction
10d7c4e022f35288779be6713471151ede967caaa39eecd35296aa36d9c109ec`
```

剔除交易的其他部分，第二個輸出如下所示：

```
{
  "value": 0.00000000,
  "n": 1,
  "scriptPubKey": "OP_RETURN 4f41010001141b753d68747470733a2f2f6370722e736d2f466f796b777248365559"
}
```

前綴 4F41 表示字母 "OA"，表示 "Open Assets"，幫我們通過Open Assets協議識別接下來的元數據。下面的ASCII編碼字串是資產定義的鏈接：

```
u=https://cpr.sm/FoykwrH6UY
```

如果我們檢索這個URL，我們得到一個JSON編碼的資產定義，如下所示：

```
{
  "asset_ids": [
    "AcuRVsoa81hoLHmVTNxRd8KpTqUxeqwgH"
  ],
  "contract_url": null,
  "name_short": "MasterBTC",
  "name": "Free copy of \"Mastering Bitcoin\"",
  "issuer": "Andreas M. Antonopoulos",
```

```

"description": "This token is redeemable for a free copy of the book \"Mastering Bitcoin\"",
"description_mime": "text/x-markdown; charset=UTF-8",
"type": "Other",
"divisibility": 0,
"link_to_website": false,
"icon_url": null,
"image_url": null,
"version": "1.0"
}

```

## 合約幣 Counterparty

合約幣（Counterparty）是一個建立在比特幣之上的協議層。合約幣協議類似於彩色幣，可以創建和交易虛擬資產和代幣。另外，合約幣提供資產的去中心化交易所。合約幣也正在實施基於以太坊虛擬機（EVM）的智能合約。

像彩色硬幣協議一樣，Counterparty在比特幣交易中嵌入元數據，使用OP\_RETURN操作碼或1-of-N多重簽名地址在公鑰的位置對元數據進行編碼。使用這些機制，Counterparty實現了一個以比特幣交易編碼的協議層。附加協議層可以被支持合約幣的應用程式解釋，例如錢包和區塊鏈瀏覽器，或者使用Counterparty庫構建的任何應用程式。

合約幣可以用作其他應用程式和服務的平臺。例如，Tokenly是一個基於Counterparty構建的平臺，它允許內容創作者，藝術家和公司發佈表示數字所有權的標記，並可用於租用，訪問，交易或購買內容，產品和服務。利用合約幣的其他應用包括遊戲（創世紀法術）和網格計算項目（摺疊硬幣）。

Counterparty的更多訊息可以在<https://counterparty.io>找到，開源項目位於<https://github.com/CounterpartyXCP>.

## 支付通道和狀態通道 Payment Channels and State Channels

支付通道 *Payment channels* 是在比特幣區塊鏈之外，雙方交換比特幣交易的去信任機制。這些交易如果在比特幣區塊鏈上結算，將變為有效的，而不是作為最終批量結算的普通票據。由於交易沒有結算，因此可以在沒有通常的結算等待時間的情況下進行交換，從而實現極高的交易吞吐量，低（亞毫秒級）的延遲以及精細的（satoshi水平）粒度。

其實，*channel*這個詞是一個比喻。狀態通道是虛擬結構，由區塊鏈之外的兩方之間的狀態交換來表示。本身沒有“通道”，底層的數據傳輸機制不是通道。我們使用術語“通道”來代表區塊鏈之外的雙方之間的關係和共享狀態。

為了進一步解釋這個概念，考慮一個TCP流。從更高級協議的角度來看，它是連接互聯網上的兩個應用程式的“套接字”。但是如果你查看網路流量，TCP流只是IP數據包上的虛擬通道。TCP流的每個端點序列化並組裝IP包以創建字節流的幻覺。在下面，它是所有斷開的數據包。同樣，支付通道只是一系列交易。如果排序正確並且連接良好，即使你不信任通道的另一端，他們也會創建可信任的可兌換義務。

在本節中，我們將看看各種支付通道。首先，我們將研究用於構建計量微支付服務（例如視頻流）的單向（單向）支付通道的機制。然後，我們將擴大這種機制，並引入雙向支付通道。最後，我們將研究如何在路由網路中點對點連接雙向通道以形成多跳通道，首先以 *Lightning Network* 的名字提出。

支付通道是狀態通道更廣泛概念的一部分，代表了狀態的脫鏈改變，並通過區塊鏈中的最終結算來保證。支付通道是一種狀態通道，其中被更改的狀態是虛擬貨幣的餘額。

### 狀態通道 —— 基本概念和術語

通過在區塊鏈上鎖定共享狀態的交易，雙方建立狀態通道。這被稱為 存款交易 *funding transaction* 或 鎚點交易 *anchor transaction*。這筆交易必須傳輸到網路並開採以建立通道。在支付通道的示例中，鎖定狀態是通道的初始餘額（以貨幣計）。

然後雙方交換簽名的交易，稱為 承諾交易 *commitment transactions*，它改變了初始狀態。這些交易是有效的交易，因為它們可以提交給任何一方進行結算，但是在通道關閉之前，它們會被各方關閉。狀態更新可以創建得儘可能快，因為每個參與方都可以創建，簽署和傳輸交易給另一方。實際上，這意味著每秒可以交換數千筆交易。

在交換承諾交易時，雙方也會使以前的狀態無效，以便最新的承諾交易永遠是唯一可以兌換的承諾交易。這樣可以防止任何一方通過單方面關閉通道並以過期的先前狀態作為對當前狀態更有利的通道進行作弊。我們將研究在本章其餘部分中可用於使先前狀態無效的各種機制。

在通道的整個生命週期內，只有兩筆交易需要提交區塊鏈進行挖礦：存款和結算交易。在這兩個狀態之間，雙方可以交換任何其他人從未見過的承諾交易，也不會提交區塊鏈。

[A payment channel between Bob and Alice, showing the funding, commitment, and settlement transactions](#) 說明了 Bob 和 Alice 之間的支付通道，顯示了存款，承諾和結算交易。

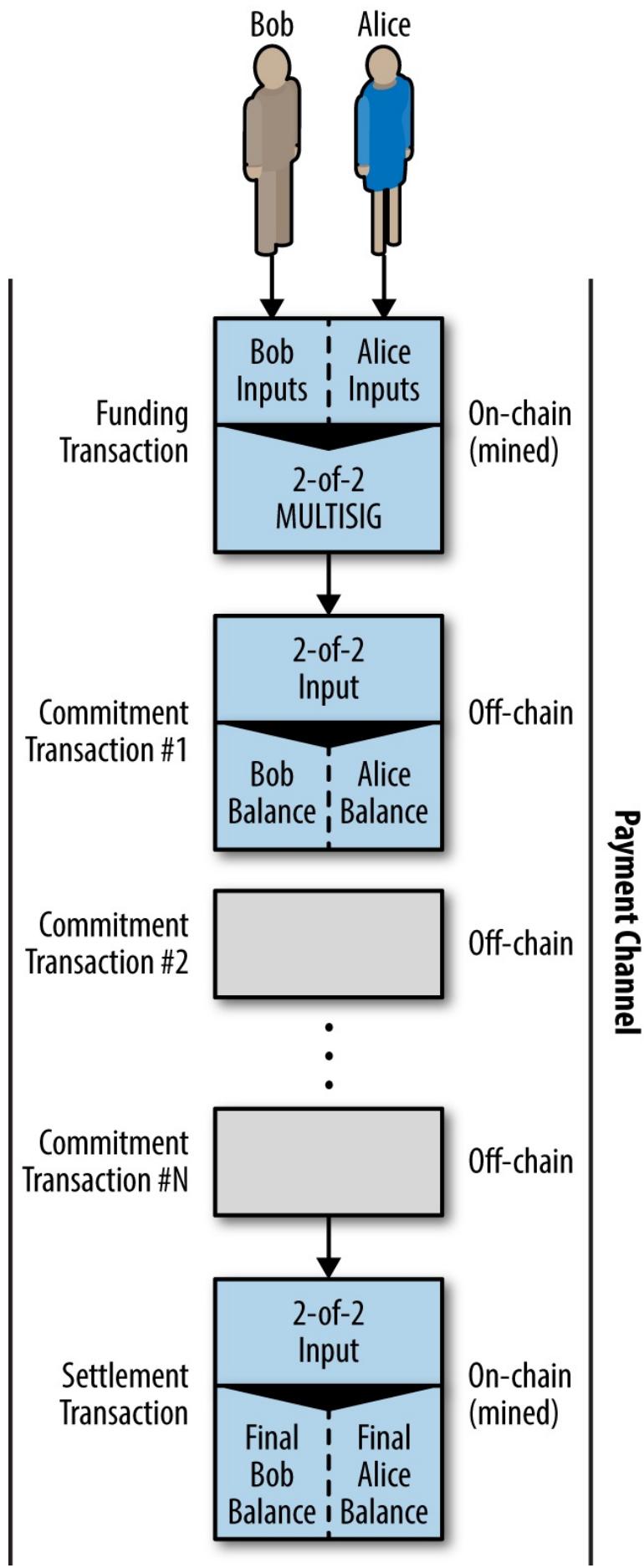


Figure 4. A payment channel between Bob and Alice, showing the funding, commitment, and settlement transactions

## 簡單支付通道示例

為了解釋狀態通道，我們從一個非常簡單的例子開始。我們展示了一個單向通道，意味著價值只在一個方向流動。我們也將從天真的假設開始，即沒有人試圖欺騙，保持簡單。一旦我們解釋了基本的通道想法，我們就會看看如何讓它變得去信任的，使得任何一方都不會作弊，即使他們想要作弊。

對於這個例子，我們將假設兩個參與者：Emma和Fabian。Fabian提供了一個視頻流媒體服務，使用微型支付通道按秒收費。Fabian每秒視頻收費0.01毫比特幣（0.00001 BTC），相當於每小時視頻36毫比特幣（0.036 BTC）。Emma是從Fabian購買此流視頻服務的用戶。[Emma purchases streaming video from Fabian with a payment channel, paying for each second of video](#) 顯示了Emma使用支付通道從Fabian購買視頻流媒體服務。

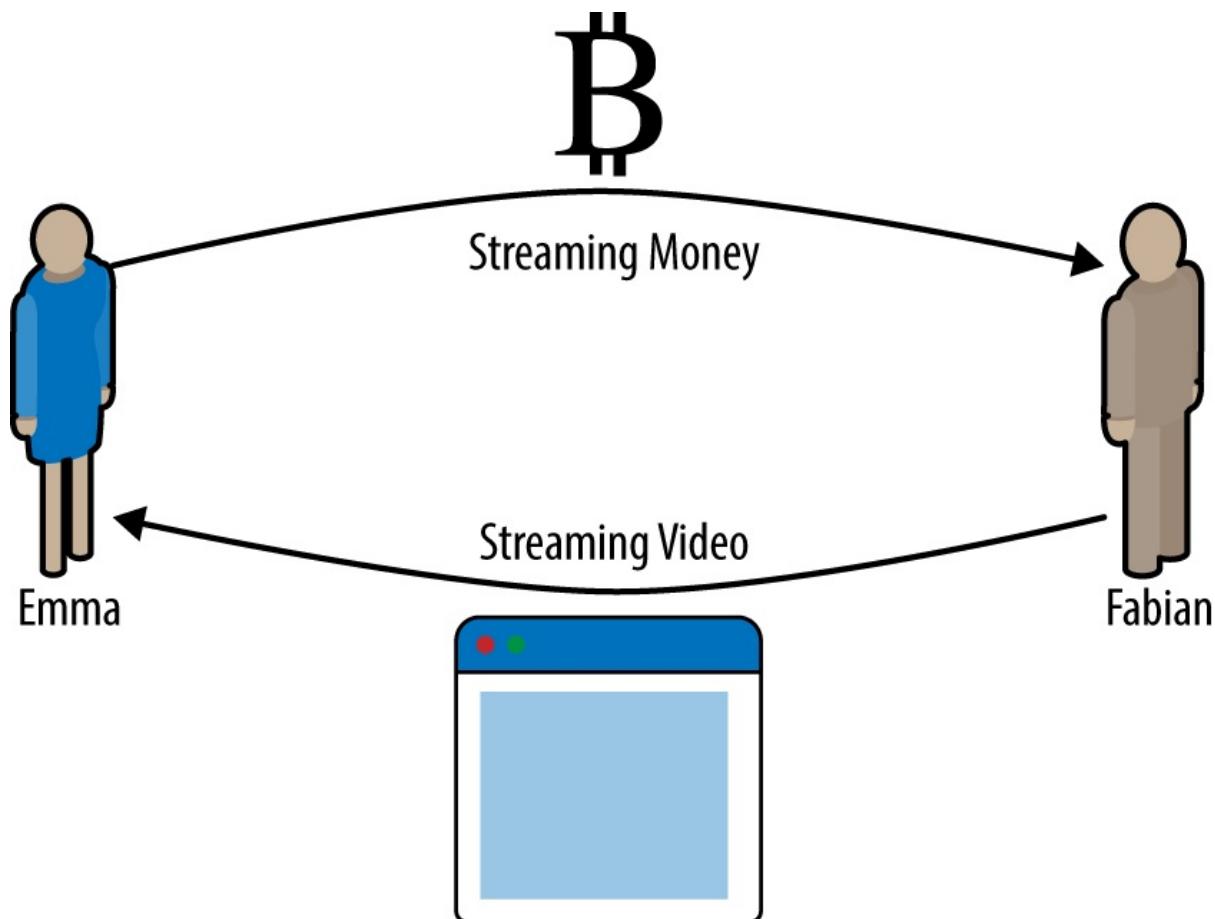


Figure 5. Emma purchases streaming video from Fabian with a payment channel, paying for each second of video  
在這個例子中，Fabian和Emma正在使用特殊的軟體來處理支付通道和視頻流。Emma在瀏覽器中運行該軟體，Fabian在伺服器上運行該軟體。該軟體包括基本的比特幣錢包功能，並可以創建和簽署比特幣交易。這個概念和術語“支付通道”對用戶來說都是完全隱藏的。他們看到的是按秒付費的視頻。

為了建立支付通道，Emma和Fabian建立了一個2-of-2多重簽名地址，每個地址都有一個密鑰。從Emma的角度來看，她瀏覽器中的軟體提供了一個帶有P2SH地址的QR碼（以“3”開頭），並要求她提交長達1小時視頻的“存款”，地址由Emma進行存款。支付給多重簽名地址的Emma的交易是支付通道的存款或錨定交易。

對於這個例子，假設Emma建立了36毫比特幣（0.036 BTC）的通道。這將允許Emma使用流式視頻最多1小時。在這種情況下，存款交易可通過*channel capacity* 設置在此通道中傳輸的最大金額。

資金交易消耗Emma錢包的一個或多個輸入，來創建存款。它為Emma和Fabian之間聯合控制的多重簽名2地址創建了一個36毫比特幣的輸出。可能會有找零的輸出返回Emma的錢包。

一旦存款交易得到確認，Emma可以開始觀看流式視頻了。Emma的軟體創建並簽署了一項承諾交易，該交易將通道餘額改為給Fabian地址0.01mBTC，並退還給Emma 35.99mBTC。Emma簽署的交易消耗了資金交易產生的36mBTC輸出，併產生兩個輸出：一個用於她的退款，另一個用於Fabian的付款。交易只是部分簽署 - 它需要兩個簽名（2個2），但只有艾瑪的簽名。當Fabian的伺服器接收到這個交易時，它會添加第二個簽名（用於2的2輸入）並將其返回給Emma以及1秒的視頻。現在雙方都有完全簽署的承諾交易，可以兌換，代表通道正確的最新餘額。任何一方都不會將此交易廣播到網路。

在下一輪中，Emma的軟體創建並簽署了另一個承諾交易（承諾 # 2），該交易消耗了資金交易中的2-of-2輸出。第二個承諾交易為Fabian的地址分配一個0.02毫比的輸出和一個35.98毫比的輸出返回Emma的地址。這項新的交易是視頻累計秒數的付款。Fabian的軟體簽署並返回第二個承諾交易，再加上一秒視頻。

通過這種方式，Emma的軟體繼續將承諾交易發送給Fabian的伺服器以換取流式視頻。隨著Emma消費更多的視頻，通道的餘額逐漸積累，以支付Fabian。假設Emma觀看視頻600秒（10分鐘），創建和簽署了600個承諾交易。最後一次承諾交易（# 600）將有兩個輸出，將通道的餘額，6 mBTC分給Fabian，30 mBTC 紿Emma。

最後，Emma點擊“Stop”停止視頻流。Fabian或Emma現在可以傳輸最終狀態交易以進行結算。最後一筆交易是結算交易，並向Fabian支付所有Emma消費的視頻費用，將剩餘的資金交易退還給Emma。

[Emma's payment channel with Fabian, showing the commitment transactions that update the balance of the channel](#) 顯示Emma和Fabian之間的通道以及更新通道餘額的承諾交易。

最終，在區塊鏈上只記錄兩筆交易：建立通道的存款交易和在兩個參與者之間正確分配最終餘額的結算交易。

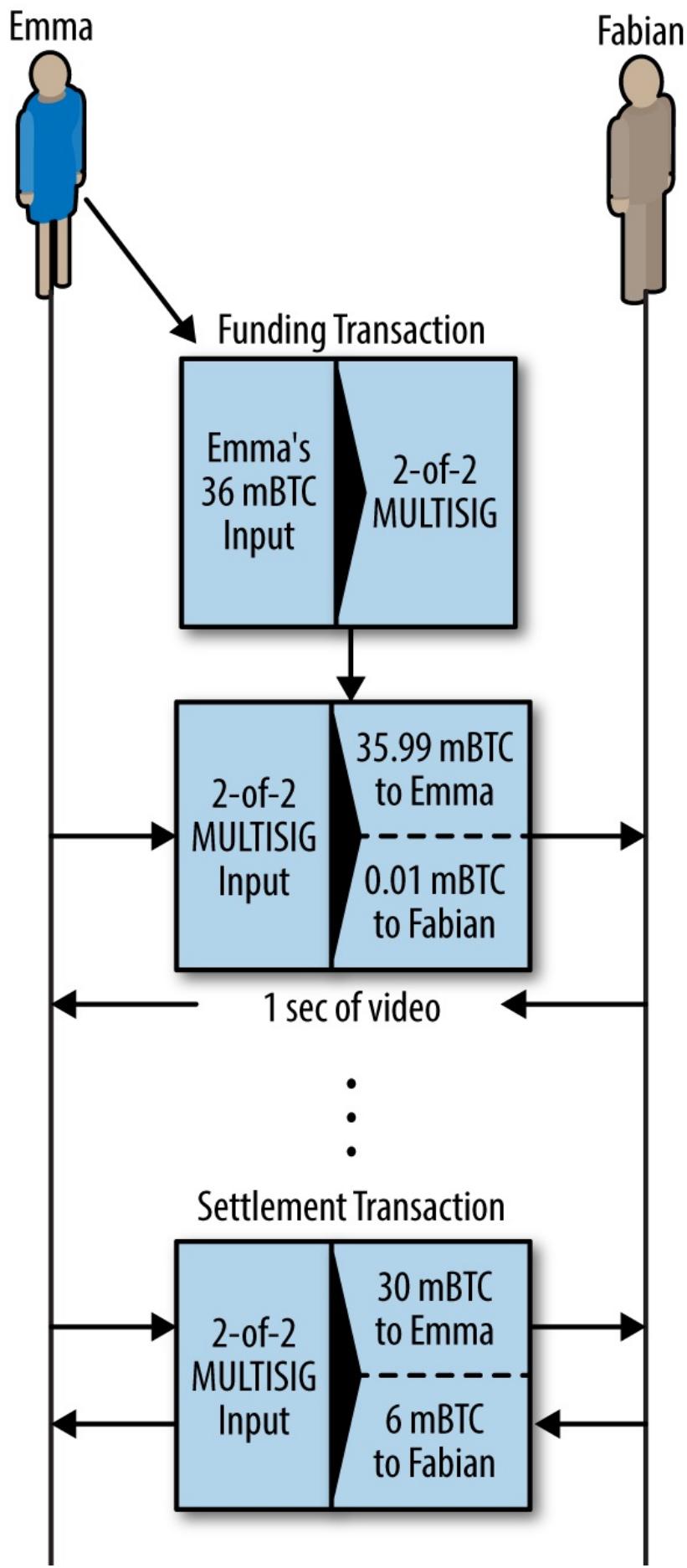


Figure 6. Emma's payment channel with Fabian, showing the commitment transactions that update the balance of the channel

## 建立無需信任的通道

我們剛才描述的通道是有效的，但只有雙方合作，沒有任何失敗或欺騙企圖。我們來看看一些破壞這個通道的情景，看看需要什麼來解決這些問題：

- 一旦存款交易發生，Emma需要Fabian的簽名才能獲得任何退款。如果Fabian消失，艾瑪的資金被鎖定在2-of-2交易中，並且實際上已經丟失了。如果其中一方在至少有一個由雙方簽署的承諾交易之前斷開連接，則此通道的存款會丟失。
- 在通道運行的同時，Emma可以接受Fabian已經簽署的任何承諾交易，並將其中一個交易給區塊鏈。為什麼要支付600秒的視頻，如果她可以傳輸承諾交易 # 1並且只支付1秒的視頻費用？該通道失敗，因為Emma可以通過播出對她有利的事先承諾而作弊。

這兩個問題都可以通過timelocks來解決，我們來看看如何使用交易級時間鎖 (nLocktime)。

除非有退款保障，否則Emma不能冒險支付2-of-2的多重簽名交易。為了解決這個問題，Emma同時構建存款和退款交易。她簽署了存款交易，但並未將其轉交給任何人。Emma只將退款交易轉交給Fabian並獲得他的簽名。

退款交易作為第一筆承諾交易，其時間鎖確定了通道的生命上限。在這種情況下，Emma可以將 nLocktime 設置為未來30天或4320個區塊。所有後續承諾交易的時間鎖必須更短，以便在退款交易前兌換。

現在Emma已經完全簽署了退款交易，她可以自信地傳輸已簽署的資金交易，因為她知道她可以最終在時限到期後即使Fabian消失也可以贖回退款交易。

在通道生命週期中，各方交換的每一筆承諾交易將被鎖定在未來。但是對於每個承諾來說，延遲時間會略短，所以最近的承諾可以在其無效的先前承諾前贖回。由於 nLockTime，雙方都無法成功傳播任何承諾交易，直到其時間鎖到期。如果一切順利，他們將通過結算交易優雅地合作和關閉通道，從而不必傳輸中間承諾交易。否則，可以傳播最近的承諾交易以結算賬戶並使所有之前的承諾交易無效。

例如，如果承諾交易 # 1被時間鎖定到將來的第4320個區塊，承諾交易 # 2時間鎖定到將來的4319個區塊。在承諾交易 # 1有效之前的600個區塊時，承諾交易 # 600可以花費。

[Each commitment sets a shorter timelock, allowing it to be spent before the previous commitments become valid](#) 展示了每個承諾交易設置一個更短的時間段，允許它在先前的承諾變得有效之前花費。

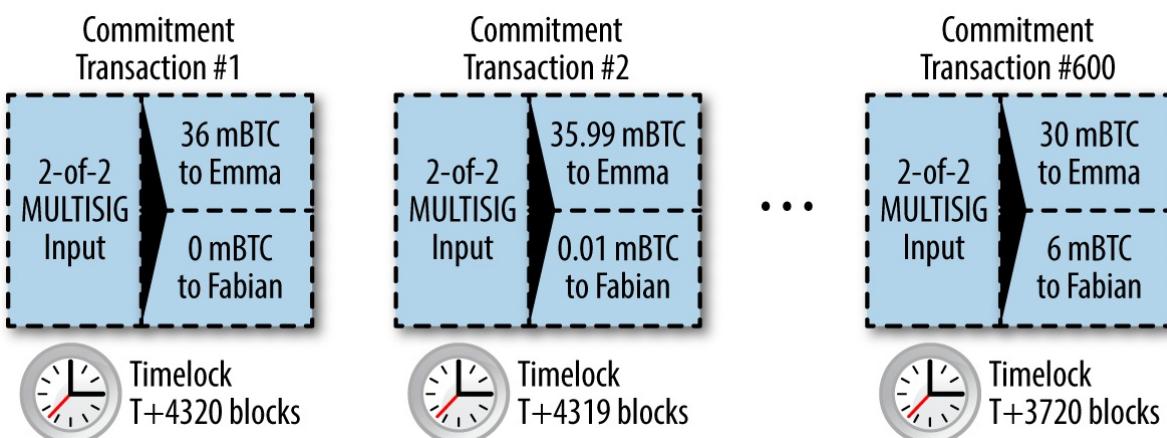


Figure 7. Each commitment sets a shorter timelock, allowing it to be spent before the previous commitments become valid

每個後續承諾交易都必須具有較短的時間鎖，以便可以在其前任和退款交易之前進行廣播。先前廣播承諾的能力確保它能夠花費資金輸出並阻止任何其他承諾交易通過花費輸出。比特幣區塊鏈提供的擔保，防止雙重支付和強制執行時間鎖，有效地允許每筆承諾交易使其前任者失效。

狀態通道使用時間鎖來實施跨時間維度的智能合約。在這個例子中，我們看到時間維度如何保證最近的承諾交易在任何先前的承諾之前變得有效。因此，可以傳輸最近的承諾交易，花費輸入並使先前的承諾交易無效。具有絕對時間鎖保護的智能合約的執行可防止一方當事人作弊。這個實現只需要絕對的交易級時間鎖（*nLocktime*）。接下來，我們將看到如何使用腳本級時間鎖 *CHECKLOCKTIMEVERIFY* 和 *CHECKSEQUENCEVERIFY* 來構建更靈活，更實用，更復雜的狀態通道。

單向支付通道的第一種形式在2015年由阿根廷開發團隊演示為視頻流應用原型。你可以在 [streamium.io](https://streamium.io) 看到。

時間鎖不是使先前承諾交易無效的唯一方法。在接下來的部分中，我們將看到如何使用撤銷密鑰來實現相同的結果。時間鎖是有效的，但它們有兩個明顯的缺點。通過首次打開通道時建立最大時間鎖，它們會限制通道的使用壽命。更糟糕的是，他們強迫通道的實現在允許長期通道和迫使其中一個參與者在過早關閉的情況下等待很長的退款時間之間取得餘額。例如，如果你允許通道保持開放30天，通過將退款時間鎖定為30天，如果其中一方立即消失，另一方必須等待30天才能退款。終點越遠，退款越遠。

第二個問題是，由於每個後續承諾交易都必須減少時間間隔，因此對雙方之間可以交換的承諾交易數量有明確的限制。例如，一個30天的通道，在未來設置一個4320區塊的時間段，在它必須關閉之前，只能容納4320箇中間承諾交易。將時間鎖承諾交易的間隔設置為1個區塊存在危險。通過將承諾交易之間的時間間隔設置為1個區塊，開發人員為通道參與者創造了非常高的負擔，這些參與者必須保持警惕，保持在線和觀看，並隨時準備好傳輸正確的承諾交易。

現在我們理解了如何使用時間鎖定來使先前的承諾失效，我們可以看到通過合作關閉通道和通過廣播承諾交易單方面關閉通道的區別。所有承諾交易都是時間鎖定的，因此廣播承諾交易總是需要等待，直到時間鎖已過。但是，如果雙方就最終餘額達成一致並知道它們都持有承諾交易並最終實現這一餘額，那麼它們可以在沒有時間鎖表示同樣餘額的情況下構建結算交易。在合作關係中，任何一方都採取最近的承諾交易，並建立一個結算交易，除了省略時間鎖之外，交易在每個方面都是相同的。雙方都可以簽署這筆結算交易，因為他們知道沒有辦法作弊並獲得更有利的餘額。通過合作簽署和轉交結算交易，他們可以關閉通道並立即贖回餘額。最差的情況下，其中一方可能會小心謹慎，拒絕合作，並強迫對方單方面使用最近的承諾交易關閉。但如果他們這樣做，他們也必須等待他們的資金。

## 不對稱可撤銷承諾 Asymmetric Revocable Commitments

處理先前承諾狀態的更好方法是明確撤銷它們。但這不容易實現。比特幣的一個關鍵特徵是，一旦交易有效，它保持有效狀態且不會過期。取消交易的唯一方法是在開採之前將其輸入與其他交易雙重支付。這就是為什麼我們在上面的簡單支付通道示例中使用時間鎖的原因，確保在較早的承諾有效之前可以花費最近的承諾。但是，按時間排列承諾產生了一些限制，使支付通道難以使用。

即使交易無法取消，也能以不希望使用它的方式構建交易。方法是給每一方一個 撤銷密鑰 *revocation key*，如果他們試圖欺騙，可以用來懲罰對方。這種撤銷先前承諾交易的機制最初是作為閃電網路（Lightning Network）的一部分提出的。

為了解釋撤銷鑰匙，我們將在Hitesh和Irene運營的兩個交易所之間構建一個更復雜的支付通道。Hitesh和Irene分別在印度和美國經營比特幣交易所。Hitesh印度交易所的客戶經常向Irene的美國交易所的客戶付款項，反之亦然。目前，這些交易發生在比特幣區塊鏈上，但這意味著要支付費用並等待幾個區塊進行確認。在交易所之間建立支付通道將顯著降低成本並加快交易流程。

Hitesh和Irene通過合作構建存款交易來啟動通道，每一方都向通道支付5比特幣資金。最初的餘額是Hitesh的5比特幣和Irene的5比特幣。資金交易將通道狀態鎖定為2-of-2的多重簽名，就像簡單通道的例子一樣。

存款交易可能有來自Hitesh的一個或多個輸入（加起來5比特幣或更多），以及來自Irene的一個或多個輸入（加起來5比特幣或更多）。輸入必須略高於通道容量才能支付交易費用。該交易有一個輸出，將10比特幣鎖定為由Hitesh和Irene控制的2-of-2多重簽名地址。交易也可能有一個或多個產出，如果他們的輸入超過了他們預期的通道貢獻，則會向Hitesh和Irene返回零錢。這是由雙方提供並簽署輸入的單一交易。它必須由各方合作建立並由各方簽字，然後才能傳送。

現在，Hitesh和Irene不創建雙方簽署的單一承諾交易，而是創建兩個 不對等 *asymmetric* 的承諾交易

Hitesh有兩項輸出的承諾交易。第一個輸出 立即 支付Irene她5比特幣。第二個輸出向Hitesh支付5比特幣，但是在1000區塊的時間鎖之後。交易輸出如下所示：

```

Input: 2-of-2 funding output, signed by Irene

Output 0 <5 bitcoin>:
    <Irene's Public Key> CHECKSIG

Output 1 <5 bitcoin>:
    <1000 blocks>
    CHECKSEQUENCEVERIFY
    DROP
    <Hitesh's Public Key> CHECKSIG

```

Irene有兩個輸出的不同承諾交易。第一個輸出立即向Hitesh支付他5比特幣。第二個輸出支付Irene她5比特幣，但是在1000區塊的時間段之後。 Irene持有的承諾交易（由Hitesh簽名）如下所示：

```

Input: 2-of-2 funding output, signed by Hitesh

Output 0 <5 bitcoin>:
    <Hitesh's Public Key> CHECKSIG

Output 1 <5 bitcoin>:
    <1000 blocks>
    CHECKSEQUENCEVERIFY
    DROP
    <Irene's Public Key> CHECKSIG

```

通過這種方式，每一方都有承諾交易，花費2-of-2的存款交易的輸出。該輸入由另一方簽名。在任何時候擁有交易的一方也可以簽署（完成2-of-2）和廣播。但是，如果他們廣播承諾交易，會立即付款給對方，而他們不得不等待一個短的時間鎖。通過延遲其中一項輸出的贖回，我們使各方在選擇單方面廣播承諾交易時處於輕微劣勢。但僅有延時的話就不足以鼓勵公平行為。

[Two asymmetric commitment transactions with delayed payment for the party holding the transaction](#) 展示了兩個不對稱承諾交易，其中支付給承諾持有人的輸出被延遲。

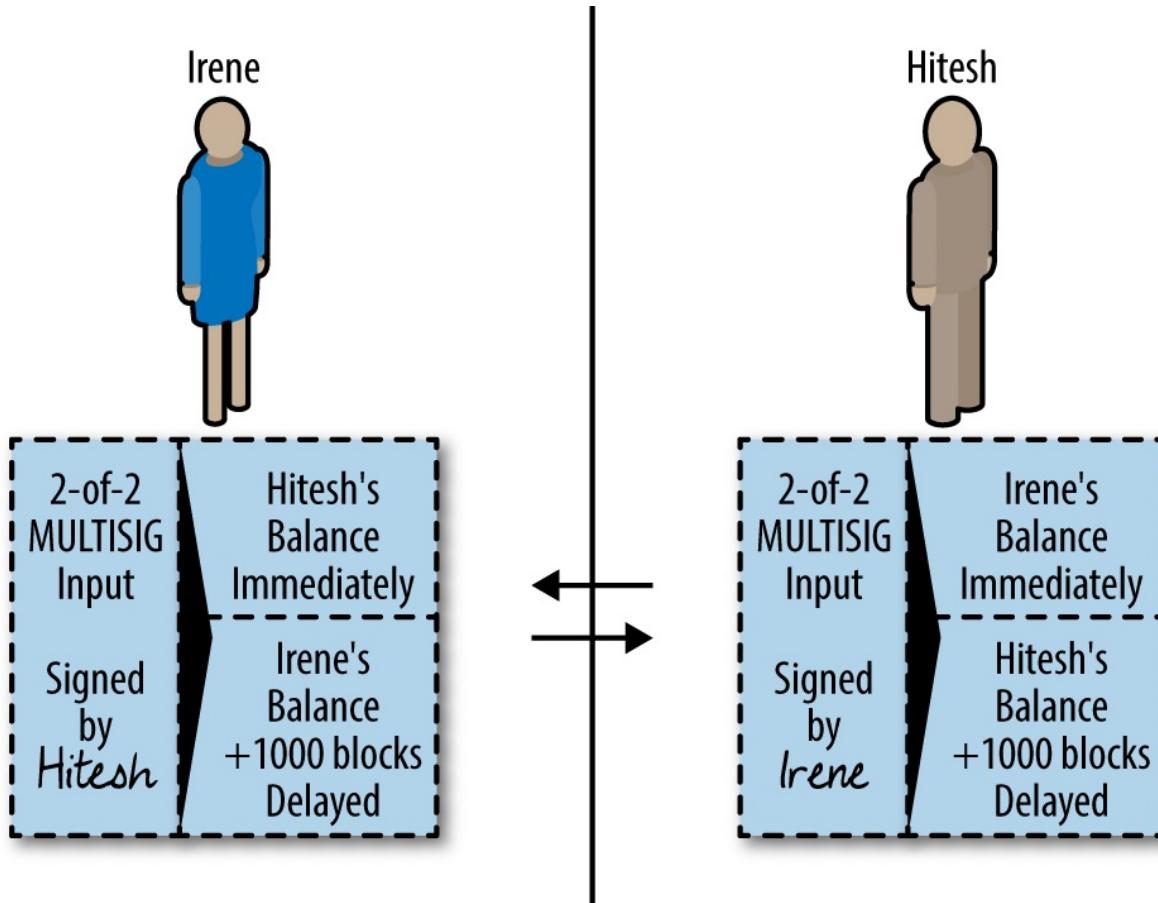


Figure 8. Two asymmetric commitment transactions with delayed payment for the party holding the transaction  
現在我們介紹這個方案的最後一個元素：一個可以防止作弊者廣播過期承諾的撤銷密鑰。撤銷密鑰允許受騙方通過佔用整個通道的餘額來懲罰作弊者。

撤銷密鑰由兩個密鑰組成，每個密鑰由每個通道參與者獨立生成。它類似於一個2-of-2多重簽名，但是使用橢圓曲線演算法構造，雙方都知道撤銷公鑰，但是每一方只知道撤銷私鑰的一半。

在每一輪中，雙方向對方公開其一半的撤銷密鑰，從而如果此次撤銷的交易被廣播，可以給予另一方（現在擁有兩半）用於要求罰款輸出的手段。

每個承諾交易都有一個“延遲的”輸出。該輸出的兌換腳本允許一方在1000個區塊之後兌換它，或者如果擁有撤銷密鑰，另一方可以贖回它，從而懲罰已撤銷承諾的傳輸。

因此，當Hitesh創建一筆讓Irene簽署的承諾交易時，他將第二個輸出在第1000個區塊之後支付給自己，或者支付給撤銷公鑰（其中他只知道一半的密鑰）。Hitesh構造了這個交易。只有當他準備轉移到新的通道狀態並想要撤銷這一承諾時，他才會向Irene展示他這一半的撤銷密鑰。

第二個支出的腳本如下：

```

Output 0 <5 bitcoin>:
<Irene's Public Key> CHECKSIG

Output 1 <5 bitcoin>:
IF
# Revocation penalty output
<Revocation Public Key>
ELSE
<1000 blocks>

```

```

CHECKSEQUENCEVERIFY
DROP
<Hitesh's Public Key>
ENDIF
CHECKSIG

```

Irene可以自信地簽署這筆交易，因為如果這筆交易被傳送，它會立即向她支付她應得的東西。 Hitesh持有該交易，但如果他通過單方面通道關閉傳輸，他將不得不等待1000個區塊才能獲得報酬。

當通道進入下一個狀態時，Hitesh必須在Irene同意簽署下一個承諾交易前撤銷此承諾交易。要做到這一點，他所要做的就是將他的一半 *revocation key* 發給Irene。一旦Irene擁有將這項承諾的兩半密鑰，她就可以自信地簽署下一個承諾。因為她知道如果Hitesh試圖通過公佈先前的承諾來作弊，她可以使用撤銷鑰匙來贖回Hitesh的延遲輸出。如果Hitesh作弊，Irene會得到兩個輸出。同時，Hitesh只有該撤銷公鑰的一半撤銷密鑰，在後續1000個區塊之前無法贖回輸出。Irene將能夠在1000個區塊到達之前贖回輸出懲罰Hitesh。

撤銷協議是雙邊的，這意味著在每一輪中，隨著通道狀態的前進，雙方交換新的承諾，為以前的承諾交換撤銷密鑰，並簽署對方的新的承諾交易。當他們接受一個新的狀態時，他們通過給予對方必要的撤銷密鑰來懲罰任何作弊行為，使先前的狀態無法使用。

我們來看一個它如何工作的例子。Irene的一位客戶希望將2比特幣發送給Hitesh的客戶之一。為了在通道中傳輸2比特幣，Hitesh和Irene必須推進通道狀態以反映新的餘額。他們將承諾一個新的狀態（狀態2號），其中10個比特幣被分割，7個比特幣給Hitesh，3個比特幣給Irene。為了推進通道狀況，他們將各自創建新的承諾交易，體現新的通道餘額。

和以前一樣，這些承諾交易是不對稱的，因此每一方的承諾交易都會迫使他們在兌換時等待。至關重要的是，在簽署新的承諾交易之前，他們必須首先交換撤銷密鑰以使先前的承諾失效。在這種特殊情況下，Hitesh的興趣與通道的真實狀態保持一致，因此他沒有理由廣播先前的狀態。然而，對於Irene來說，狀態1給她的餘額高於狀態2。當Irene將她的先前承諾交易（狀態1）的撤銷密鑰給Hitesh時，她也撤銷了她通過倒退通道獲利的能力。因為有了撤銷鑰匙，Hitesh可以毫不拖延地贖回先前承諾交易的兩個輸出。這意味著，如果Irene廣播先前的狀態，Hitesh可以行使他的權利拿走所有輸出。

重要的是，撤銷不會自動發生。雖然Hitesh有能力懲罰Irene的作弊行為，但他必須認真觀察區塊鏈是否存在作弊跡象。如果他看到先前的承諾交易被廣播，他有1000個區塊時間採取行動，使用撤銷密鑰來阻止Irene的作弊行為，並通過取得全部10個比特幣來懲罰她。

具有相對時間鎖的不對稱可撤銷承諾（CSV）是實施支付通道的更好方式，也是該技術非常重要的一項創新。通過這種構造，通道可以無限期地保持開放，並且可以擁有數十億的中間承諾交易。在Lightning Network的原型實現中，承諾狀態由48位索引標識，允許任何單個通道有超過 $281\text{萬億次}$  ( $2.8 \times 10^{14}$ ) 狀態轉換！

## 雜湊時間鎖合約 Hash Time Lock Contracts (HTLC)

支付通道可以通過特殊類型的智能合約進一步擴展，允許參與者將資金髮送到可贖回的密鑰上，並有過期時間。此功能稱為 *Hash Time Lock Contract* 或 *HTLC*，並用於雙向支付通道和路由支付通道。

我們先來解釋HTLC的“雜湊”部分。要創建HTLC，付款的預期接收人將首先創建一個密鑰 R。然後他們計算這個密鑰的雜湊值 H：

```
H = Hash(R)
```

產生的雜湊值 H 可以包含在輸出的鎖定腳本中。知道這個密鑰的人可以用它來贖回輸出。密鑰 R 也被稱為雜湊函數的原象 *preimage*。原象只是用作雜湊函數輸入的數據。

HTLC的第二部分是“時間鎖定”。如果密鑰未透露，HTLC的付款人可以在一段時間後獲得“退款”。這是通過使用 *CHECKLOCKTIMEVERIFY* 進行絕對時間鎖定實現的。

實現了 HTLC 的腳本看起來是這樣的：

```

IF
    # Payment if you have the secret R
    HASH160 <H> EQUALVERIFY
ELSE
    # Refund after timeout.
    <locktime> CHECKLOCKTIMEVERIFY DROP
    <Payer Public Key> CHECKSIG
ENDIF

```

任何知道密鑰 R 的人，當雜湊值等於 H 時，可以通過行使 IF 流的第一個子句來贖回該輸出。

如果密鑰未透露，HTLC聲稱，在一定數量的區塊之後，付款人可以使用 IF 流程中的第二個條款要求退款。

這是HTLC的基本實現。這種類型的HTLC可以由具有密鑰 R 的 任何人 兌換。對腳本稍作修改，HTLC可以採取許多不同的形式。例如，在第一個子句中添加一個 CHECKSIG 運算符和一個公鑰，將雜湊的兌換限制為一個指定的收款人，該收款人還必須知道密鑰 R 。

## 路由支付通道（閃電網路）

閃電網路是一個提議端到端連接的雙向支付通道路由網路。像這樣的網路可以允許任何參與者在無需信任任何中間人的情況下將支付從通道發送到通道。閃電網路 <https://lightning.network/lightning-network-paper.pdf> [由Joseph Poon和Thadeus Dryja於2015年2月首先描述]，建立在許多其他人提出和闡述的支付通道的概念上。

“閃電網路”是指用於路由支付通道網路的特定設計，現在已經由至少五個不同的開源團隊實現。獨立實現由一組互操作性標準進行協調：<http://bit.ly/2rBHeoL>[*Basics of Lightning Technology (BOLT) paper*]。

閃電網路的原型實現已由多個團隊發佈。目前，這些實現只能在testnet上運行，因為它們使用segwit，而沒有在主比特幣區塊鏈（mainnet）上激活。

閃電網路是實施路由支付通道的一種可能方式。還有其他幾個旨在實現類似目標的設計，例如Teechan和Tumblebit。

## 基本閃電網路示例

讓我們看下它如何工作。

在這個例子中，有五個參與者：Alice，Bob，Carol，Diana和Eric。這五位參與者相互開設了支付通道，兩兩相連。Alice 與 Bob，Bob 與 Carol，Carol 與 Diana，Diana 與 Eric。為了簡單起見，我們假設每個參與者為每個通道提供2比特幣，每個通道的總容量為4比特幣。

[A series of bidirectional payment channels linked to form a Lightning Network that can route a payment from Alice to Eric](#) 展示了閃電網路中的五位參與者，通過雙向支付通道進行關聯，這些通道可以連接起來以支持 Alice 支付到 Eric (路由支付通道（閃電網路）)。

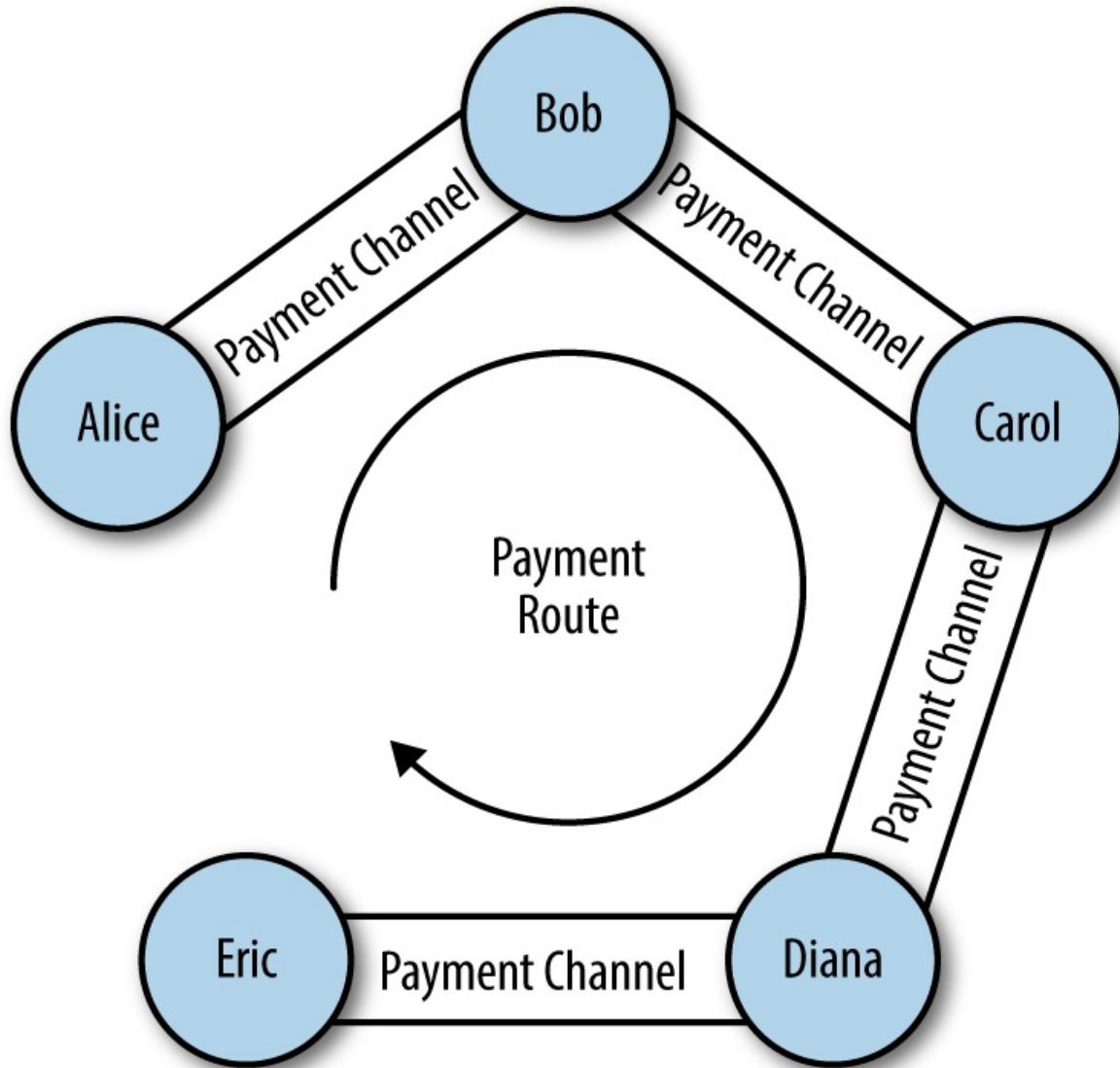


Figure 9. A series of bidirectional payment channels linked to form a Lightning Network that can route a payment from Alice to Eric

Alice 想要支付 Eric 1比特幣。但是，Alice 沒有通過支付通道與 Eric 連接。創建一個支付通道需要一筆資金交易，這筆交易必須交給比特幣區塊鏈。Alice 不想開設新的支付通道並承諾更多的資金。有沒有間接支付Eric的方法？

[Step-by-step payment routing through a Lightning Network](#) 展示了通過連接參與者的支付通道上的一系列 HTLC 承諾，從 Alice 支付到 Eric 的分步過程。

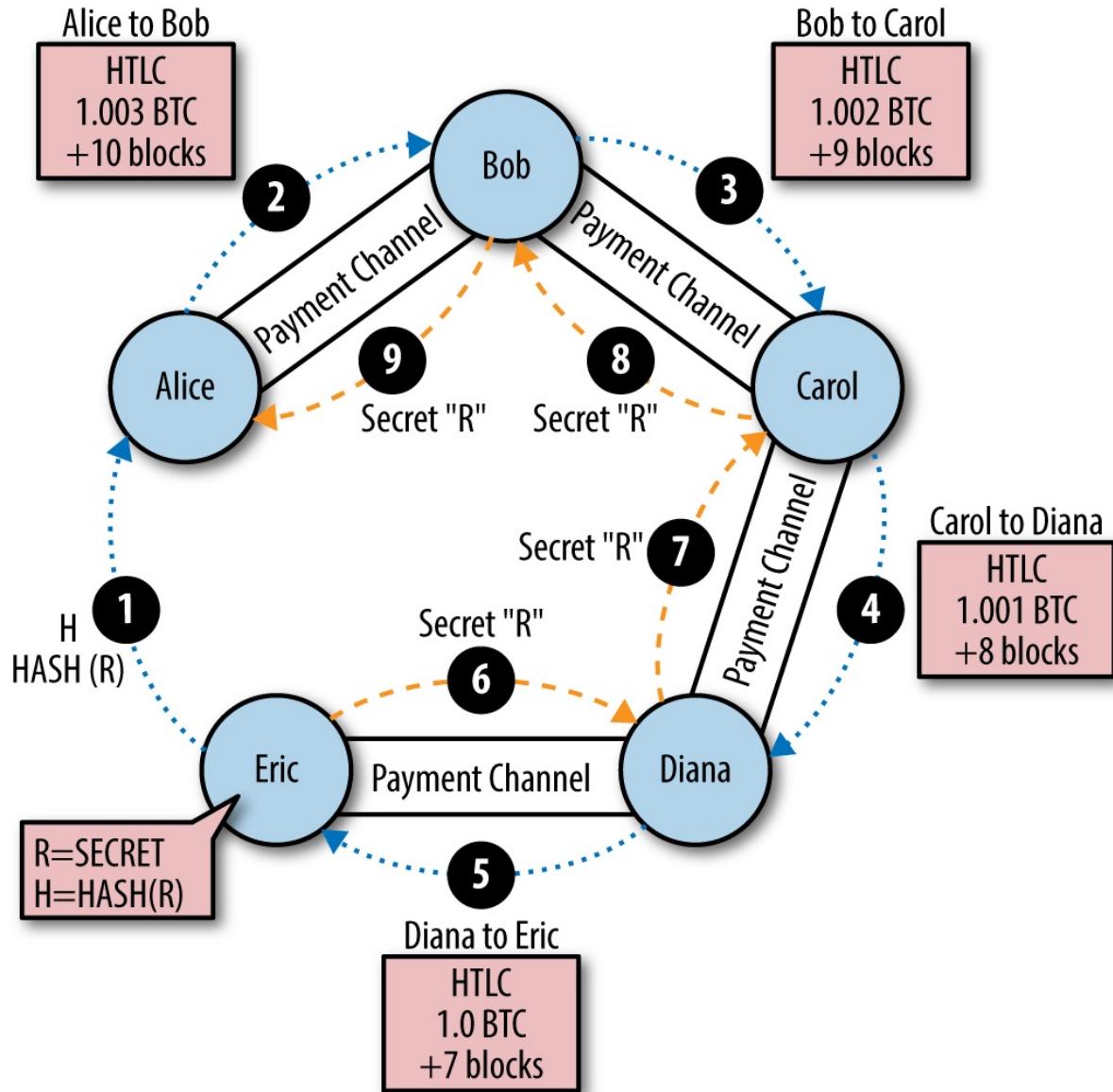


Figure 10. Step-by-step payment routing through a Lightning Network

Alice 正在運行一個閃電網路（LN）節點，該節點跟蹤她和Bob的支付通道，並且能夠發現支付通道之間的路線。Alice 的 LN 節點還可以通過互聯網連接到 Eric 的 LN 節點。Eric 的 LN 節點使用隨機數生成器創建一個密鑰  $R$ 。Eric 的節點並沒有向任何人透露這個密鑰。Eric 的節點計算密鑰  $R$  的雜湊  $H$  並將該雜湊傳送給 Alice 的節點（參見 [Step-by-step payment routing through a Lightning Network](#) 步驟1）。

現在，Alice 的 LN 節點構造了 Alice 的 LN 節點和 Eric 的 LN 節點之間的路線。所使用的路由演算法將在後面更詳細地討論，但現在讓我們假設 Alice 的節點可以找到有效的路由。

然後，Alice 的節點構建一個HTLC，支付給雜湊值  $H$ ，並有10個區塊的超時退款（當前區塊 + 10），金額為1.003比特幣（參見 [Step-by-step payment routing through a Lightning Network](#) 步驟2）。額外的0.003將用於補償參與此支付路線的中間節點。Alice 向 Bob 提供這個 HTLC，從 Bob 的通道餘額中扣除 1.003 比特幣並將其交給 HTLC。HTLC具有以下含義：“如果鮑勃知道密鑰，則 Alice 將1.003的通道餘額付給Bob，或者如果經過10個區塊，則退還到 Alice 的餘額。”Alice 和 Bob 之間的通道餘額現在是由三項輸出的承諾交易表示：Bob的2比特幣，Alice的0.997比特幣，Alice的HTLC的1.003比特幣。Alice向HTLC承諾的金額減少了Alice的餘額。

Bob現在有一個承諾，如果他能夠在接下來的10個區塊內獲得密鑰  $R$ ，他可以獲得被Alice鎖定的1.003。有了這個承諾，Bob的節點在Carol的支付通道上構建了一個HTLC。Bob的HTLC承諾了9個區塊超時的1.002比特幣給  $H$ ，如果有密鑰  $R$ ，Carol可以贖回（參見 [Step-by-step payment routing through a Lightning Network](#) 步驟3）。Bob知道，如果Carol可以獲得他的HTLC，她必須有  $R$ 。如果Bob在9個區塊時間內得到  $R$ ，他可以用它來向Alice索要Alice的HTLC。

他還通過在9個區塊時間內貢獻他的通道餘額獲得了0.001比特幣。如果Carol不能索要他的HTLC，他不能索要Alice的HTLC，那麼一切都會恢復到之前的通道餘額，沒有人會不知所措。Bob和Carol之間的通道餘額現在是：Carol的2，Bob的0.998，Bob到HTLC的1.002。

Carol現在有一個承諾，如果她在接下來的9個區塊時間內獲得R，她可以索要由Bob鎖定的1.002比特幣。現在，她可以在她與Diana的通道上做HTLC承諾。她將一個1.001比特幣的HTLC提交給雜湊H，8個區塊過期時間，如果有密鑰R，Diana可以贖回（參見 [Step-by-step payment routing through a Lightning Network](#) 步驟4）。從Carol的角度來看，如果這樣做的話，她能獲得0.001比特幣更好，如果沒有，她什麼都不會丟失。她到Diana的HTLC只有在R被揭示時才是可行的，在這一點上，她可以向Bob索取HTLC。Carol和Diana之間的通道餘額現在是：Diana的2，Carol的0.99，Carol對HTLC承諾的1.001。

最後，Diana可以向Eric提供一個HTLC，將7個區塊超時時間內支付1比特幣給雜湊H（參見 [Step-by-step payment routing through a Lightning Network](#) 步驟5）。Diana和Eric之間的通道餘額現在是：Eric的2，Diana的1，Diana到HTLC的1。

但是，在這條路線上，Eric擁有密鑰R。因此，他可以索要Diana提供的HTLC。他將R發送給Diana並索要1比特幣，將其添加到他的通道餘額中（參見 [Step-by-step payment routing through a Lightning Network](#) 步驟6）。通道餘額現在是：Diana的1，Eric的3。

現在，Diana有密鑰R。因此，她現在可以從Carol那獲得HTLC。Diana將R發送給Carol，並將1.001比特幣添加到她的通道餘額中（參見 [Step-by-step payment routing through a Lightning Network](#) 步驟7）。現在，Carol和Diana之間的通道餘額是：Carol的0.999，Diana的3.001。Diana參與這條支付路線“贏得”了0.001。

沿著路線返回，密鑰R允許每個參與者索要HTLC。Carol向Bob索要1.002，將他們的通道中的餘額設置為：Bob的0.998，Carol的3.002（參見 [Step-by-step payment routing through a Lightning Network](#) 步驟8）。最後，Bob索要來自Alice的HTLC（參見 [Step-by-step payment routing through a Lightning Network](#) 步驟9）。他們的通道餘額更新為：Alice的0.997，Bob的3.003。

Alice在沒有與Eric開通通道的情況下就向Eric支付了1比特幣。支付路徑中的任何中間人都不需要互相信任。將他們的資金在通道中用於短期承諾，他們可以賺取一小筆費用，唯一的風險是如果通道關閉或路由支付失敗，退款會有小幅延遲。

## 閃電網路傳輸和路由

LN節點之間的所有通信都是點對點加密的。另外，節點有一個長期的公鑰，<http://bit.ly/2r5TACm>[用來向彼此授權]。

每當一個節點希望將支付發送給另一個節點時，它必須首先通過連接具有足夠容量的支付通道來通過網路構建路徑 *path*。節點公佈路由訊息，包括他們已經打開了哪些通道，每個通道有多少容量，以及他們收取的路由支付費用。路由訊息可以以各種方式共享，隨著閃電網路技術的發展，可能會出現不同的路由協議。一些閃電網路實現使用IRC協議作為節點公佈路由訊息的便利機制。路由發現的另一個實現使用P2P模型，其中節點將通道公告傳播給他們的同伴，採用“泛洪”模式，類似於比特幣傳播交易的機制。未來的計劃包括名為 [Flare](#) 的提案，這是一種具有本地節點“鄰居”和更遠距離信標節點的混合路由模型。

在我們前面的例子中，Alice的節點使用這些路由發現機制之一來查找將她的節點連接到Eric節點的一條或多條路徑。一旦Alice的節點構建了一條路徑，她將通過網路傳播一系列加密和嵌套指令，連接每個相鄰的支付通道，初始化該路徑。

重要的是，這條路只有Alice的節點才知道。支付路線中的所有其他參與者只能看到相鄰的節點。從Carol的角度來看，這看起來像是Bob到Diana的付款。Carol並不知道Bob實際上是在轉發Alice支付的一筆款項。她也不知道Diana會向Eric轉賬。

這是閃電網路的一個重要特徵，因為它確保了付款隱私，並且使得應用監視，審查或黑名單非常困難。但是，Alice如何建立這種支付路徑，而不向中間節點透露任何東西？

閃電網路根據稱為 [Sphinx](#) 的方案實施洋蔥路由（onion-routed）協議。此路由協議可確保付款發起人可以通過Lightning Network 構建和傳遞路徑，以便：

- 中間節點可以驗證和解密路由訊息中屬於他們的部分並找到下一跳。

- 除了上一跳和下一跳之外，他們無法瞭解路徑中的任何其他節點。
- 他們無法識別付款路徑的長度，或他們在該路徑中的位置。
- 路徑的每個部分都被加密，使得網路層的攻擊者無法將來自路徑不同部分的數據包相互關聯。
- 與Tor（互聯網上的洋蔥路由匿名協議）不同，沒有可以置於監控之下的“出口節點”。付款不需要傳送到比特幣區塊鏈；節點只是更新通道餘額。

使用這種洋蔥路由協議，Alice將路徑中的每個元素都封裝在一個加密層中，從結尾開始並向後工作。她用Eric的公鑰將一條訊息加密給Eric。此訊息包裹在一封加密給Diana的訊息中，將Eric標識為下一個收件人。發給Diana的訊息包裹在一封加密給Carol公鑰的訊息中，並將Diana確定為下一個收件人。給Carol的訊息被加密到Bob的密鑰。因此，Alice已經構建了這種加密的多層“洋蔥”訊息。她將此發送給Bob，他只能解密和解包外層。在裡面，Bob發現一封給Carol的信，他可以轉發給Carol，但不能自己破譯。沿著路徑，訊息被轉發，解密，轉發等，一直到Eric。每個參與者只知道每跳中的前一個和下一個節點。

路徑的每個元素都包含有關必須擴展到下一跳的HTLC訊息，正在發送的金額，要包含的費用以及使HTLC過期的CLTV鎖定時間（以區塊為單位）。隨著路由訊息的傳播，這些節點將HTLC承諾轉發到下一跳。

此時，你可能想知道節點為何不知道路徑的長度及其在該路徑中的位置？畢竟，他們收到一條訊息並將其轉發到下一跳。根據它是否變短了，他們能夠推斷出路徑大小和位置？為了防止這種情況，路徑總是固定為20跳，並填充隨機數據。每個節點都會看到下一跳和一個固定長度的加密訊息來轉發。只有最終收件人看到沒有下一跳。對於其他人來說，總是還有20跳。

## 閃電網路的好處

閃電網路是次層路由技術。它可以應用於任何支持一些基本功能的區塊鏈，例如多重簽名交易，時間鎖定和基本智能合約。

如果閃電網路位於比特幣網路之上，那麼比特幣網路可以在不犧牲無中介無信任運轉原則的情況下，大幅提升容量，隱私，粒度和速度：

### 隱私 *Privacy*

閃電網路支付比比特幣區塊鏈上的支付私有得多，因為它們不公開。雖然路線中的參與者可以看到通過其通道傳播的付款，但他們不知道發件人或收件人。

### 可互換性 *Fungibility*

閃電網路使得在比特幣上應用監視和黑名單變得更加困難，從而增加了貨幣的可互換性。

### 速度 *Speed*

使用Lightning Network的比特幣交易以毫秒為單位進行結算，而不是以分鐘為單位，因為在不提交交易給區塊的情況下清算HTLC。

### 粒度 *Granularity*

閃電網路可以使支付至少與比特幣“灰塵”限制一樣小，可能甚至更小。一些提案允許subsatoshi（次聰）增量。

### 容量 *Capacity*

閃電網路將比特幣系統的容量提高了幾個數量級。閃電網路路由的每秒支付數量沒有實際的上限，因為它僅取決於每個節點的容量和速度。

### 無信任運作 *Trustless Operation*

閃電網路在節點之間使用比特幣交易，節點之間作為對等運作而無需信任。因此，閃電網路保留了比特幣系統的原理，同時顯著擴大了其運行參數。

當然，如前所述，閃電網路協議並不是實現路由支付通道的唯一方式。其他提議的系統包括Tumblebit和Teechan。但是，目前閃電網路已經部署在測試網路上。幾個不同的團隊開發了競爭性的LN實現，並正在朝著一個通用的互操作性標準（稱為BOLT）努力。Lightning Network很可能將成為第一個在生產環境中部署的路由式支付通道網路。

## 總結

我們只研究了一些可以使用比特幣區塊鏈作為信任平臺構建的新興應用。這些應用將比特幣的範圍擴展到支付範圍和金融工具之外，涵蓋了信任至關重要的許多其他應用。通過分散信任的基礎，比特幣區塊鏈成為了一個平臺，將在各行各業產生許多革命性的應用。