世界上所有的法定貨幣都需要控制其總量以及加入不同的防偽技術來防止偽造。對於流通貨幣而言，中央銀行控制了貨幣的發行量並且在紙幣中加入防偽技術。這些技術提高了偽造的門檻，但是法定貨幣依然有可能被偽造，對於破壞法定貨幣系統規則的人，最終還是需要法律部門介入。

加密數位貨幣也需要安全手段來防止駭客擾亂系統，尤其需要防止“混淆”，意思是，對不同的人做出前後不一致的說辭。比如說 Alice 付了 Bob 一個數位貨幣，她就不能告訴 carol，她付給 carol 的是同一個數位貨幣。但與法定貨幣不同的是，這些安全措施全部用技術方法來實現而不依靠一個中央執法機構。

如上所述，加密貨幣很大程度上利用了密碼學的技術。密碼學提供了一套加密數位貨幣系統安全編碼的技術機制。我們不但可以利用密碼學防止篡改和混淆，還可以將貨幣的發行規則寫在數學協定中。在我們深刻理解加密數位貨幣系統之前，我們需要先深入探究加密數位貨幣系統所依賴的密碼學的基礎。

密碼學是一個高深的學術領域，用到了很多鮮為人知的數學理論，並且其理論也比較複雜。幸運的是，比特幣只運用到了密碼學中少量簡單且大家所知的一些理論。在本章中，我們將會具體研究密碼學中的雜湊函數 (Hash) 和數位簽章 (digital signature) 技術。這兩個基本概念對構建一個加密數位貨幣系統非常關鍵。後面的章節中，我們會介紹一些更複雜的密碼學理論，例如零知識證明 (zero-knowledge proofs)，零知識證明被應用到對比特幣網路的拓展和修改中。

一旦我們學習了一些必要的密碼學基礎，我們將討論如何用這些密碼學基礎構建一個加密數位貨幣系統。本章結束的時候，我們會透過設計一些簡單的加密貨幣來闡明我們在設計中遇到的挑戰。

1.1 密碼學雜湊 (hush)

第一個我們要理解的密碼學基礎是密碼學雜湊函數。密碼學雜湊函數是一個數學函數，具備以下三個特點：

◎ 它的輸入可以是任意大小的任意字串。

◎ 它的輸出大小是固定長度的字串，本書為了討論方便，我們假設輸出都是 256-bit，但是我們後面的討論也可以應用到其他大小的輸出，只要這個輸出長度夠大。

◎ 它可以被有效地計算。簡單來說就是：對於一個給定的字串，在合理的時間內，你可以透過 hash 函數計算其輸出。從技術上來解釋就是，對於一個 n-bit 的輸入，其計算複雜度為 O (n)。

這三個特性可以定義一個簡單的 hash 函數，透過 hash 函數的這三個特性，可以構建雜湊表。本書中，我們將會重點關注密碼學 hash 函數，對於一個需要達到密碼學安全目標的雜湊函數來說，我們需要其具備以下三個額外的特性：(1) 防碰撞性 (Collision-resistance) (2) 隱蔽性 (Hiding) (3) 不方便解密性 (puzzle-friendliness)。

我們下面會瞭解這三點意味著什麼，學過密碼學的朋友可能會感覺到這裡的論述與一般的密碼學課程會不同，尤其是第三點在密碼學裡不是一個常見的要求，但對加密數字貨幣確很有意義。

特徵 1：防碰撞性：密碼學雜湊函數的第一個特徵是它是可以避免碰撞。碰撞指的是兩個不同的輸入得到相同的輸出。我們說：如果沒有任何人可以找到一個碰撞，那麼這個 hash 函數 H (.) 是防碰撞的。

防碰撞：我們說一個 hash 函數 H 是防碰撞的，當且僅當，對於兩個給定的輸入 x,y, 當 $x \neq y$，不可能找到 $H(x) = H(y)$

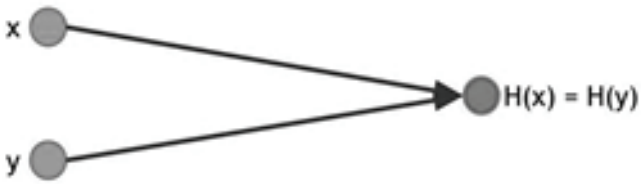

圖 1.1 雜湊碰撞。x 和 y 分別是不同輸入，當作為雜湊函數的輸入時，會產生相同的輸出。這時我們就說這個函數是雜湊碰撞的。

需要注意的是，我們只是說沒有人可以找到這樣的一個碰撞，但是我們並沒有說碰撞不存在，實際上，我們可以知道的是，碰撞確實存在，並且我們可以透過簡單的計算來證明碰撞的存在。雜湊的輸入集合包含了所有長度的所有字串，而輸出集合只包含了特定長度的字串。因為輸入集合比輸出集合包含更多的字串 (事實上，輸入集合是一個無限集合，輸出集合為一個有限集合)，因此，肯定有不同的輸入會被產生同一個輸出值。根據鴿籠原理 (Pigonhole Principle)，一定會有一個巨大的輸入集合，才有可能形成一個任意輸出的輸出集合。

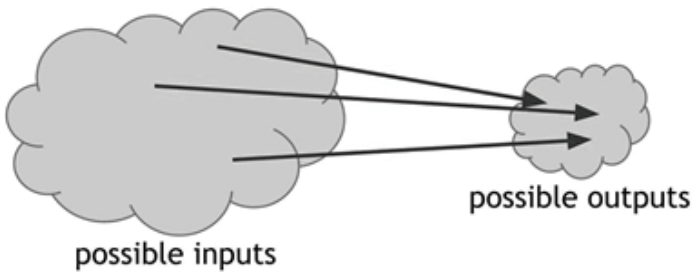

圖 1.2 因為輸入的數量超過輸出的數量，我們可以確定某一個輸出肯定對應多個輸入。

現在，我們考慮一下更糟糕的情形，我們說過不可能找到一個雜湊碰撞，但是還是有辦法去嘗試去尋找雜湊碰撞，考慮下面的這個雜湊函數，這個雜湊函數的輸出是：256 -bit，

輸入是 $(2^{256}+1)$ 個不同的輸入值，分別計算這 $(2^{256}+1)$ 個輸入值的輸出的雜湊值，並且檢查是否有任意兩個輸出值是相同的。因為我們的輸入集合遠遠大於輸出集合，我們可以確定肯定有某一個輸出值，對應了至少 2 個相同的輸入值，也就意味著，肯定有碰撞發生。

透過上述的方式，去尋找雜湊碰撞是肯定可以找到的，但是如果我們透過隨機的去選擇輸入，並且計算他們的雜湊值，我們將不用一一檢查 $(2^{256}+1)$ 個不同的輸入來確定某一個雜湊碰撞，實際上，只要我們隨機的取出 $(2^{130}+1)$ 個輸入，我們將有 99.8% 的機率去找到一個雜湊碰撞。

透過檢查可能輸出值的數目的平方根 (the square root of the number of possible outputs)，就找到碰撞的事實，在數學上，叫做生日悖論 (birthday paradox) 注解：生日悖論 (Birthday paradox) 是指，如果一個房間裡有 23 個或 23 個以上的人，那麼至少有兩個人的生日相同的機率要大於 50%。這就意味著在一個典型的標準小學班級 (30 人) 中，存在兩人生日相同的可能性更高。對於 60 或者更多的人，這種機率要大於 99%。

在本章的作業部分，我們會對此問題做更進一步的探討。

碰撞檢測演算法對任何一個雜湊函數都適用，但是，透過這種演算法來尋找雜湊碰撞會花費非常多的時間。例如，對於一個 256-bit 的輸出而言，為了尋找一個雜湊碰撞，最差的情況下，你有可能需要計算 $(2^{256}+1)$ 次，平均下來，為了找到這樣的雜湊碰撞，你平均需要計算：2^{128} 次。

這個當然是一個巨大的數字：如果一台電腦的計算速度是 10000 次 / 秒，那麼要花費 10^{27} 年來計算 2^{128} 個不同值，我們換一種方式來說明這個問題，我們可以這樣來論述，如果所有人類生產的電腦在宇宙剛開始

的時候，就被發明了，並從那時開始就一直計算，一直計算到今天，這麼長的時間裡面，它能發現碰撞的機率還是非常非常小，小到比地球在接下來的 2 秒鐘被一個巨大的流星撞擊，並且毀滅了的機率還要小。

透過上述的論述，我們可以了解到，透過通用的演算法，我們很難找出任何雜湊的碰撞值，但是更進一步我們可以思考：是否有其他的方式，可以對特定的雜湊函數，有更高效的雜湊碰撞尋找方式？換句話說，儘管通用的碰撞尋找演算法不切實際，但是還有其他的演算法可以更高效的針對某一類雜湊函數來找出碰撞值。

考慮下面的雜湊函數：

$$H(x) = x \bmod 2^{256}$$

這個函數符合我們的要求：輸入是任意長度，輸出是 256-bit，並且對於任意一個輸入都可以計算出其輸出。但是這個雜湊函數有一個高效的方式來尋找碰撞。需要注意的是這個函數只會返回輸入值的 256-bit，一個可能的碰撞值是：3 和 $3+2^{256}$

注意：3 和 $3+2^{256}$ 對 2^{256} 求餘數之後，都是 3，這個例子可以說明，儘管我們透過一般的碰撞檢測演算法在實際中仍不可行，但是針對某些特定的雜湊函數，確實存在一些更高效的雜湊碰撞尋找演算法。

但是對於其他的雜湊函數，我們不知道是否這種高效的雜湊碰撞尋找演算法存在。我們假設他們是很難發生雜湊碰撞的，然後，並沒有 hash 函數可以證明是防止雜湊碰撞的。密碼學上實際使用的函數只是非常非常難以找到雜湊碰撞，並且從來沒有找到過這樣的碰撞，但是這不能證明碰撞不存在。在一些過去的例子中，例如 MD5 雜湊函數，經過多年的努力，終於找到了一個可行的演算法，可以找出碰撞值，並且導致了這個函數在實際使用中大大減少，甚至消失掉了。因此我們選擇相信在密碼學上使用的雜湊函數是很難發生雜湊碰撞的。

具體應用：資訊摘要 (Message digests)。現在我們知道了防止碰撞 (collision-resistance) 是什麼了，那麼下一個問題就是防止碰撞可以有什麼實際的用途？ 下面就是一個應用：如果我們把兩個不同的輸入 x 和 y，分別輸入到一個防止碰撞的 hash 函數裡面，那麼我們可以基本上判斷，得到的輸出值 H(x) 和 H(y) 也是不同的 --- 如果輸入 x 和 y 不同，但是得到了相同的 hash 值，則違反這是一個不具備防止碰撞特性的 hash 函數。

這個論述可以讓我們把雜湊函數的輸出作為資訊的摘要，考慮以下的例子：SecureBox 是一個需要認證的線上檔案存儲系統，允許使用者上傳自己的檔案，並且當用戶下載的時候，可以保證檔的完整性。假設 Alice 上傳了一個非常大的檔案，並且希望以後下載的時候可以驗證下載的檔就是她當初上傳的檔案。一種可行的方式是，透過將檔案儲存在自己的電腦上，然後將這個檔案與從網站上下載的檔案做比較。雖然這樣方式可以對比出，檔案是否一致，但是另外一個問題是，這也會使得一開始將檔案上傳到雲端沒有意義，因為如果 Alice 需要，她大可以直接使用儲存在自己電腦上的檔案。

防止碰撞的雜湊函數為這個問題提供了一種優雅並且高效的解決方案。Alice 只需要記住原始檔案的雜湊值就可以了，當她稍後從 SecureBox 下載檔案時，她只需要算出下載檔案的雜湊值是多少，並且和最初的雜湊值做比較，如果兩個雜湊值是一樣的，她可以確信，下載的檔案就是她當初上傳的檔案，如果兩個雜湊值不同，Alice 可以確定她上傳的檔案已經被修改過了。透過對比雜湊值，可以允許 Alice 知道檔案是否在傳輸過程中發生損壞，或者在雲端被修改，或者被伺服器故意修改。面對各種惡意行為時的一種保證和保障，也是密碼學給我們提供的一種核心價值所在。

這裡的雜湊值是一段固定長度的資訊或是明確的摘要，可以讓我們有效的記住我們過去看到的東西，並且在下次遇到的時候，可以很容易認出來。然後，整個檔案可能是好幾 GB 那麼大，但是在我們的例子中，雜湊輸出值是固定的 256bit，這就大大降低了我們的存儲要求。稍後的章節中，大家可以看到透過把雜湊值做為資訊摘要的實際應用案例。

性質 2：隱蔽性 (Hiding) 我們要求的雜湊函數還需要具備第二個特性，即隱蔽性。隱蔽性意味著 如果我們知道某個 hash 函數 y= H(x) 的輸出值 y，我們不可能找到輸入值 x。問題是，上述的表示形式不一定是正確的。考慮到下面非常簡單的例子：如果我們隨機的拋一枚硬幣，如果結果是 heads (正面)，我們就會公佈“heads” (這個字串) 的雜湊值，如果拋出來的結果是 tails(反面)，我們就會公佈“tails”反面這個字串的雜湊值。

然後，我們問我們的對手 (這個對手沒有看到拋硬幣的過程，但是看到了 heads 和 tails 字串的雜湊值) 這樣一個問題，透過給定的雜湊值，來猜測出輸入的字串是什麼？作為回應，對手可以很容易的計算字串“heads”的雜湊值和字串“tails”的雜湊值，並且對比我們給他的雜湊值，這樣經過幾步的計算，對手就很容易猜測出我們輸入的字串是什麼了。

對手很容易猜測出輸入的字串是什麼的原因是我們只有兩個不一樣的輸入 x，他們透過 2 次嘗試之後，就可以計算出來。為了使我們的雜湊函數具備隱蔽性，我們需要輸入 x 不具備特殊性，也就是說 x 的取值範圍非常廣泛，如果 x 取值於一個非常廣泛的集合，那麼透過嘗試幾個 x 值來找到輸出值的方式將不會發生了。

那麼現在的問題是：當一個雜湊函數的輸入並不是來自於一個分散分佈的集合的時候 (例如只來自於拋硬幣的“正面”和“反面”的時候)，我們是否能讓這個雜湊函數具備隱蔽性？答案是：是的！我們可以把一

個不是分散分佈的集合隱藏在一個分散分佈的集合裡面，從而取得隱蔽性。現在我們可以更準確的定義一下“隱蔽性”的意思了 (雙分隔號 ‖ 代表把一系列事件、事情等聯繫起來)。

隱蔽性：當一個秘密值 r 來自於一個最小熵的機率分佈的時候，如果在 H(r ‖ x) 中，不可能找到 x 的值，那麼我們說這樣的一個雜湊函數 H 是具備隱蔽性的

在資訊理論中，最小熵是一個結果的可預測性的量度，高階最小熵的直覺含意就是一個分佈 (例如隨機變數) 非常分散化，我們這樣描述的意思是：當我們從一個最小熵的分佈裡面，取一個樣本的時候，不大可能會取到特定的某個值。一個具體的例子就是：如果 r 取自所有 256-bit 長度的字串，那麼取到任意一個特定字串的機率是 $1/2^{256}$，這個機率值非常小。

隱蔽性應用：承諾協議 (commitment) 現在讓我們看一個隱蔽性性質的具體應用，這個應用是密碼學中經常探討的承諾協議 (commitment)，一個承諾協議的具體表述類似下面的動作，任意取一個值，並把這個值放到一個信封裡面，並把信封放到一個人人可以看到的桌子上。當你做這個動作的時候，你自己知道信封裡面封裝的值是什麼，但是信封並沒有打開，所以儘管你自己知道信封裡面的值，但是對剩下的所有人來說，信封裡面的值都是一個未知的值。然後你可以把信封打開，並公佈你之前放進去的值。

承諾協議：一個承諾協定方案包含以下兩種演算法

◎ com:=commit (msg, nonce) 承諾函數取一段訊息與一個亂數 (nonce) 作為輸入，輸出就是一個“承諾”(commitment)。

◎ isValid:=verify (com, msg, nonce) 驗證函數把一個“承諾”、訊息、及亂數 (nonce) 作為輸入，如果 com == commit(msg, nonce)，那麼返回 true，否則返回 false。

在上述承諾協議中，我們要求其具備以下兩個性質

◎ 隱蔽性：若是只知道 com，不可能找到 msg

◎ 綁定性：不可能找到兩組 (msg, nonce) 和 (msg’, nonce’)，如果 msg ≠ msg’，而 commit(msg, nonce) == commit(msg’, nonce’)

為了使用一個承諾協定架構，我們首先需要產生一個隨機的亂數，然後我們把承諾函數用於這個亂數和訊息，從而可以得到一個“承諾”。這個階段相當於把密封的信封放到桌子上。接下來，如果我們想要揭開之前放入的訊息是什麼，我們公佈當時我們選擇的亂數和當時選擇的訊息，那麼任何一個人都可以驗證我們公佈的訊息確實是之前放入的訊息。這個階段相當於用打開信封的動作。

每次你承諾一個具體的值，非常重要的一點是需要重新選擇一個亂數。在密碼學中，Nonce 代表只能被使用一次的亂數，或者叫做不重覆性的亂數。

在承諾協議中，隱蔽性和綁定性非常類似於密封信封和打開信封的過程，首先，知道某一個“承諾”的過程類似於可以看到信封但是無法知道信封裡面是什麼的過程。第二個性質是：綁定性，這個類似於當你把某一個值放到信封裡面的時候，你後面不可能改變它。這個是說，你不可能找到兩條不同的訊息，並且你承諾了一條訊息，然後你說你承諾了另外一條訊息。

我們如何在承諾協議中保證隱蔽性和綁定性這兩個性質成立呢？在我們討論這一點之前，我們需要討論我們如何實現一個承諾協議架構。我們可以透過一個密碼學的雜湊函數來實現這一點，考慮下面的承諾協議實現方案：

Commit (msg, nonce)：=H (nonce ‖ msg)，這裡 nonce 是一個長度 256 位的隨機值

為了承諾一段訊息，我們產生一個 256 位的亂數，然後我們把這個亂數和一段訊息聯繫起來，並把這個整體的雜湊值，作為一個“承諾”，為了驗證，可以透過計算綁定這段訊息的亂數的雜湊值來驗證。並且檢查這個雜湊值是否和他們之前看到的“承諾”相等。

現在我們再看一下我們在承諾協議中提到的兩個性質，如果我們把承諾函數的例子換成雜湊函數，那麼提到的兩個性子可以如下表述：

◎ 隱蔽性：知道 H(nonce ‖ msg), 不可能找到 msg

◎ 綁定性：不可能找到 (msg, nonce) 和 (msg’, nonce’)，如果 msg ≠ msg’ 且 H(nonce ‖ msg) == H(nonce’ ‖ msg’)

承諾協定中的隱蔽性也是我們在雜湊函數中需要的隱蔽性，如果我們隨機選取 256 位的某一個值 (key)，並且把這個值和一段訊息的聯繫作為整體作為雜湊函數的輸入，那麼透過雜湊函數的輸出不可能恢復得到輸入的一段訊息是什麼。並且綁定的性質也和雜湊函數的防碰撞性不謀而合，如果一個雜湊函數是防碰撞的，那麼不可能找到兩個不同的 msg