# 1密码学及加密货币概述

**ddk问题：**

* “货币”是什么，怎么运作的？
* 货币体系规则：发行、流通、价值？

|  |  |  |
| --- | --- | --- |
|  | **Currencies**  **（法定）货币** | **Cryptocurrencies**  **加密货币** |
| **发行** | 中央银行发行  使用防伪标识 | 使用“**密码学技术**”将“新货币创造规则”编码到数学协议中 |
| **流通** | 人们在交易中使用货币。  中央机构和执法部门进行管制，防止“货币体系规则”受到破坏。 | 使用“**密码学技术**”将“货币体系规则”编码到系统本身。  可防止对系统的干扰，并且能避免混淆（例如双重支付） |
| **价值** | ？？？ | ？？？ |

ddk问题：// 参考前言部分

* 技术和管理的发展和演变，是为了解决遇到的问题。
* 现有货币有什么问题？
* 加密货币要解决什么问题？如何解决？

**本章讨论“比特币”中使用“密码学技术”：**

* Hash 哈希算法
* digital signature 数字签名

## 1.1密码学哈希函数

**(1) 一般哈希函数（general hash function ）**

一般哈希函数的三个基本属性：

* 任意输入：任意大小的字符串
* 固定输出：固定大小，例如256位
* 有效计算：计算复杂度为O(n)

以一般哈希函数为基础，可以创建各种数据结构。

例如：hash table（哈希表），能够为快速搜索建立索引

**（2）加密哈希函数（cryptographic hash functions）**

为了实现加密货币的安全性，需要使用“加密哈希函数”。

例如：SHA-256

**【建模哈希函数】**

哈希函数是密码学的瑞士军刀：它们在各种各样的应用中找到了一席之地。

这种通用性的另一个方面是：为了确保安全性，不同的应用需要哈希函数的不同属性。

很难确定哈希函数的一个属性列表，能对于所有应用有可证明的安全性。

在本书中，我们选择了：对于比特币和其它加密货币所用的哈希来说，重要的三个属性。

即使在这个范围中，也不是所有这些属性对于每个哈希函数的使用都是必需的。

例如，“难题友好”只对比特币挖矿很重要。

安全系统的设计者通常会把哈希函数建模为：对每个可能的输入，输出一个独立的随机值。

使用这种“随机输出模型”来证明安全性，这在密码学中仍然存在争议。

不管在这场辩论中的立场如何，这都是对建立安全系统的有价值的智力练习：关于如何减少安全属性的推理。

本章就是帮你学习这个技能。

**【“加密哈希函数”需要三个额外属性】**

我们要解释为什么需要这样的属性，即这些属性有什么作用，怎么在加密货币中起作用。

|  |  |  |
| --- | --- | --- |
| **属性** | **属性的作用** | **属性的应用** |
| **碰撞阻力**  **collision-resistance** | 正向：对不同的输入，产生不同的输出。  **说明：**几乎不可能找到两个不同输入产生相同的输出 | 信息摘要  （message digest） |
| **隐秘性**  **hiding** | 反向：根据输出，无法知道输入。  **说明：**   * 碰撞阻力说明不同输入产生不同输出； * 但如果只有两个输入，那么就很容易根据输出猜到输入； * 隐秘性要解决的是：不能根据输出猜出输入，使用的方法是在输入中增加nonce（随机值）。 | 保证（commitment） |
| **谜题友好**  **puzzle-friendliness** | 搜索谜题：H(​ *id*​ ‖​ *x*​ ) ∈***Y***​  正向：id是随机生成的，如果只能通过随机选择x来尝试，那么这个H就是“谜题友好”。 | 比特币挖矿 |

**【在比特币中的应用】**

哈希指针：区块链、梅克尔树

### 1.1.1碰撞阻力

**【定义】**

**碰撞阻力：不同输入，产生不同输出**

**【说明】**

如果输出是256位，我们随机地选择输入，并计算哈希值，在检验第2256+1个输入之前便很可能找到碰撞。

实际上，如果我们随机选择2130+1个输入，找到至少两个相同哈希值的概率为99.8%。

仅仅通过检验可能输出数量的平方根次数，便大体能找到碰撞，这在概率论中称为生日悖论（birthday paradox）。

生日悖论是指，如果一个房价里不少于23个人，那么有两人生日相同的概率要大于50%。

这意味着，在一个典型的班级（30人）中，有两人生日相同的概率更高。

如果超过60个人，则这种概率要大于99%。

可以证明，没有哈希函数具有防碰撞特性。

但对于某些哈希函数，我们认为它具有碰撞阻力特性，因为我们还没有找到“识别碰撞的有效方法”。

我们依赖这些加密哈希函数，仅仅是因为还未找到有效碰撞的方法。

例如，对于MD5哈希函数，在多年的努力之后，最终找到了碰撞，使该函数在实践中被逐渐淘汰。

（ddk：防碰撞特性指完全没有碰撞，碰撞阻力指尽量减少碰撞。）

**【应用】**

**将哈希输出作为“信息摘要”（message digest）**

哈希函数对于一个信息生成固定长度的摘要，这为我们提供了一种有效方法：记住之前所见事物，并在今后认出这一事物。这样做，能极大降低存储要求。

保证主体不受其他实体的恶意行为侵害，这正是密码学的核心。

### 1.1.2隐秘性

**【定义】**

**隐秘性（非精确描述）：**

如果只知道哈希函数的输出y = H（x），没有可行的办法算出输入值x。

**隐秘性（精确的描述）：**

当其输入r选自一个大取值范围的概率分布，在给定H（r‖x）条件下来确定x是不可行的。

例如，r是从长度为256位的字符串中随意选出的，那么任意特定字符串被选中的概率为1/2256。

（注：双竖线‖为连接符号，代表把一系列事件联系起来）

**【应用】**

**“隐秘性”的应用：保证（commitment）**

**保证：**我保证一件事（meg）是真的，但在确认我的保证之前，我不公布这个meg，而且也保证我不会修改这个msg。

**“保证”怎么实现呢？**

* 我将这个“msg”与一个“随机数”（nonce，例如256位）合在一起，作为加密哈希函数的输入，输出一个哈希值（com）。com = H（msg‖nonce）
* 我公布这个com值，表示我做了一个保证：即保证msg是真实的。

之后，我无法改变msg，因为难以到找到另一个输入产生的输出为com。

* 任何人也无法根据我公布的com来推断出msg，这就是“隐秘性”。
* 当要验证我的保证时，我就公布msg和nonce，计算哈希值，与之前公布的com比较，就能确认我的保证是否为真。

例子：赌博

* 我押了一个结果（msg），公布它的哈希值（com），别人不知道我押的是什么。

说明，即使只有两个结果（例如硬币的正反），也能通过增加r，使得无法从哈希值推断出所押的结果。

* 当真实结果出来后，我就可以公布我的msg和nonce，从而证明我押的是否正确。

### 1.1.3谜题友好

**【定义】**

搜索谜题：H(​ *id*​ ‖​ *x*​ ) ∈***Y*** ​

id是随机生成的，如果只能通过随机选择x来尝试，那么这个H就是“谜题友好”。

**【应用】**

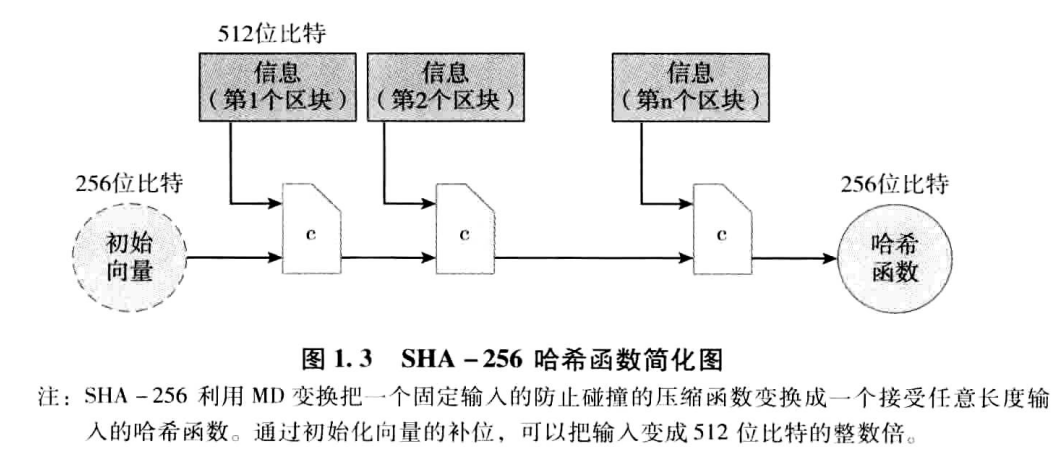
比特币挖矿：

使用当前交易的哈希（id），寻找一个x，使得H(​ *id*​ ‖​ *x*​ ) 在***Y***​的范围内，

Y的值代表了搜索的难度。

### 1.1.4 SHA-256

SHA ： Secure Hash Algorithm 安全哈希算法



ddk问题：

* “初始向量”是什么？初始向量有什么作用？
* 是不是一个固定输入，总是得到固定固定输出？（应该是，否则怎么验证？）

## 1.2哈希指针及数据结构

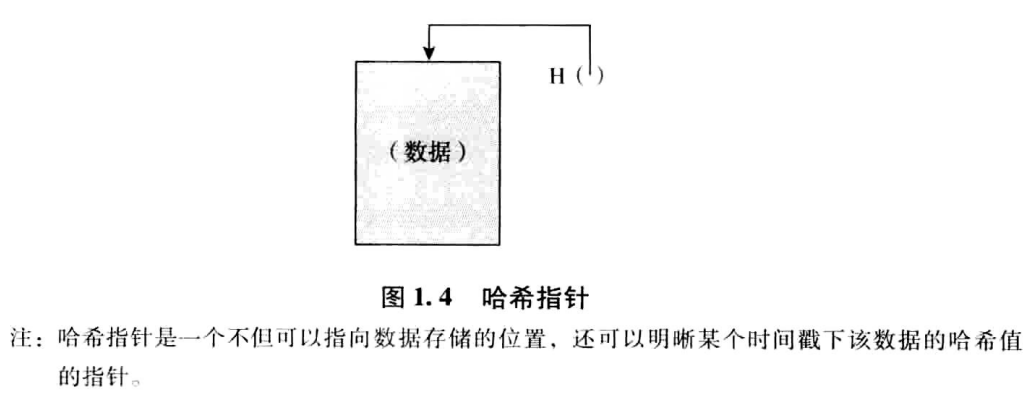
**普通指针：**指出数据存储的位置

**哈希指针：**指出数据的哈希值

**哈希指针：可以验证数据没有被篡改过。**

因为如果数据被篡改过，则哈希值必定发生变化。

所以，数据和哈希值必须同时修改，否则就能验证出不一致。



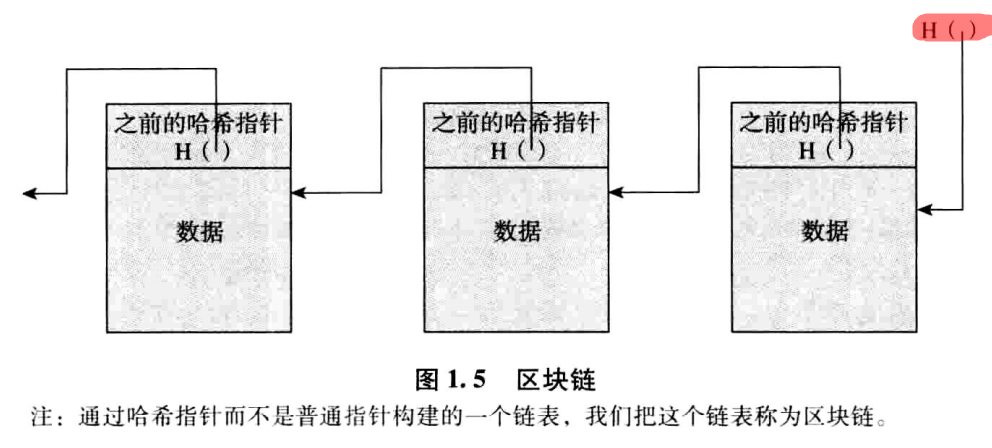
注：“某个时间戳下该数据”指的是“某一时刻的数据”。

可以利用哈希指针构建各种各样的数据结构，例如：

|  |  |  |
| --- | --- | --- |
|  | **链表** | **二叉树** |
| **普通指针** | 数据链表 | 二叉查找树 |
| **哈希指针** | 区块链（block chain） | 梅克尔树（Merkel tree） |

### 1.2.1区块链

**【定义】**



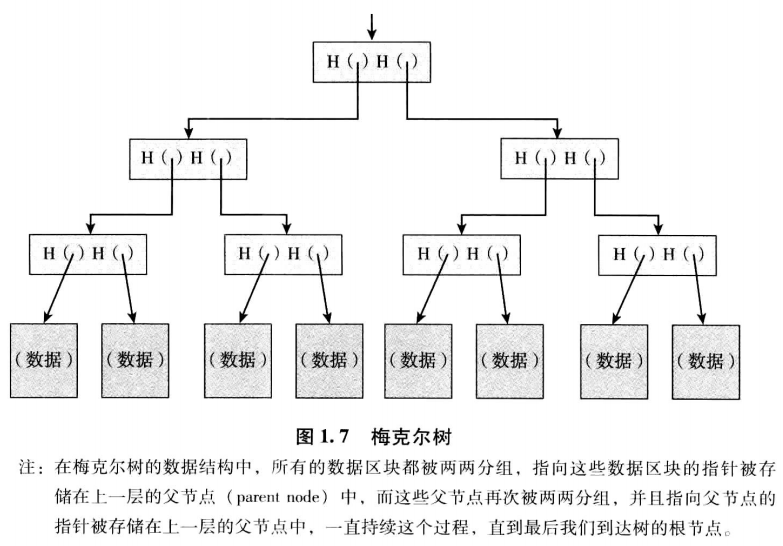
**【应用：防篡改日志】**

管理员只需要记住“区块链头部的哈希指针”（篡改者无法修改修改它），

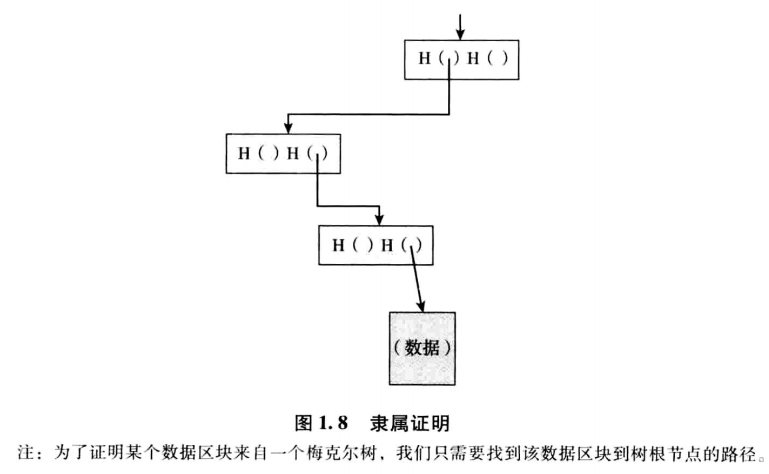
这样，如果有任何篡改，就能检测到了。

### 1.2.2梅克尔树

**【定义】**



**【应用：验证隶属和非隶属关系】**



一个排序梅克尔树是把底层的数据通过某些排序得到的梅克尔树，

这里的排序规则可以是字母排序、字典排序、数字化排序，或者其它约定的排序方式。

**排序梅克尔树：**可以验证某个数据块不属于梅克尔树。（非隶属关系）

可以在任何以指针为基础的数据结构中使用哈希指针，条件是数据结构不存在循环。

例如，可以建立一个哈希指针定向的非循环图，可以验证隶属关系，同时容易计算。

## 1.3数字签名

**“数字签名”是对“手写签名”的模拟。**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **手写签名** | **数字签名** | **签名的作用** |
| **针对某个文件** | 签名只对这个文件有效，文件应该不能被篡改，或者能够检查出来。 | 文件可以用“信息摘要”来代替，  这样篡改文件很容易被检测出来。 | 保证这个信息是准确的，没有被篡改 |
| **只有本人可以签名** | 手写签名很难伪造 | 使用私钥进行签名，  私钥只有本人知道 | 保证是我本人同意这个信息 |
| **任何人都可以验证签名的有效性** | 任何人都可以看到签名，认出（或通过技术手段）这是谁的签名 | 使用公钥验证签名，  公钥是公开的 | 任何人可以验证我的签名，从而确认：是我签署的，并且信息有效 |

|  |  |  |
| --- | --- | --- |
| **“数字签名方案”由三个算法构成** | | |
| **算法** | **函数** | **说明** |
| **生成公钥和私钥** | (sk,pk):= generateKeys( keysize ) | keysize （输入）  sk/pk （私钥/公钥）  generateKeys最好是随机的，因为它需要为不同的人生成不同的密钥 |
| **对一段消息进行签名** | sig := sign( sk, message ) | 用sk对message进行签名，返回签名值sig。 |
| **验证签名** | isValid := verfiy( pk, message, sig ) | 使用pk对sig进行解密，用解密值与message比较，如果相同，则sig有效，否则，sig无效。 |

**【实践中的考量】**

在实践中，由于签名算法（sign）本身的限制，导致能够签署的信息大小是有限的。

一个简单的解决方法：对信息的哈希值进行签名，而非对信息本身进行签名。

另一个技巧：对哈希指针进行签名

如果你签署了哈希指针，那么该签名保护了整个结构。

**【ECDSA椭圆曲线数字签名算法】**

|  |  |
| --- | --- |
| **ECDSA的参数** | **位数** |
| 私钥 | 256 |
| 公钥 | 512 （未压缩）  257 （压缩） |
| 待签名信息 | 256 |
| 签名 | 512 |

**【关于“随机性”】 // ddk：不太理解**

我们注意到generateKeys和sign都可以采用随机算法。

的确，generateKeys最好是随机的，因为它需要为不同的人生成不同的密钥，而verify则需要是确定的。

要将算法概念转化为现实中执行的数字签名机制，我们还需要考虑很多实际问题。

例如，很多签名算法是随机的（特别是比特币使用的算法），因此，我们需要随机性的良好来源，我们不能低估这一点的重要性，因为不良随机性会使你认为安全的算法变得不安全。

使用ECDSA时，确保随机性良好来源至关重要，因为不良来源将可能导致密钥信息的泄露。

这一点不难理解，如果你使用了不良随机来生成密钥，那么该密钥就可能不安全。

但是，ECDSA的古怪就在于，即使你仅仅只是在生成签名时使用了不良随机，而你使用的密钥完美无缺，你的个人密钥还是可能泄露。（熟悉DSA的人都知道这是DSA的古怪之处，但并不针对椭圆曲线）。

接着，游戏结束了，如果你的个人密钥泄露，对手就可以伪造你的签名。

因此，我们在实践中特别注意使用良好的随机来源，使用不良随机来源是安全系统的一个常见缺陷。

**【注意】**

比特币中并没有使用任何“加密”，因为没有加密的需要。

“加密”只是现代密码学中众多技术中的一个，很多技术（例如保证）在某种程度上隐藏信息，但与“加密”有所不同。

## 1.4公钥即身份

pk == 身份 // 能以pk身份发声的人，必须知道相应的sk

可以随时生成新的身份（pk）：用generateKeys程序生成密钥对sk和pk

**【实践中的考虑】**

在实践中，你可能会使用“pk的哈希”作为你的“身份”，这是因为公钥很大。

pk == pk的哈希 == 身份

为了验证消息是你发的，需要验证：

1. 你提供的身份是“pk的哈希”，然后，你提供pk来验证这是你的身份。 // 参见比特币的实现
2. 消息经过pk验证

**【去中心化身份管理】**

使用sk和pk，可以实现：去中心化的身份管理。

身份可以由你自己随时生成，想要多少就生成多少，没有人知道这些身份是谁生成的。

这就是比特币对待身份的方式，这些身份称为“地址”。

**去中心化身份管理的“匿名性”和“隐秘性”：**

在比特币中，你不需要注册或暴露你的真实身份，但你的行为模式可能是可识别的。

这就是比特币的基本隐秘性问题，第6章将专门讨论这个问题。

ddk：参见1.1.2，隐秘性是指，根据输出y不能获得输入x。

**【关于统计理论】**

初学者的直觉：概率系统是不可预测的和难以推理的。

统计理论：使我们能够精确地量化事件的可能性，并对系统的行为做出自信的断言。

## 1.5两种简单的加密货币

从“密码学”过渡到“加密货币”

### 1.5.1高飞币（GoofyCoin）

**高飞币只有两个规则：**

1. 高飞可以随时创建新币，且这些新币都属于他。
2. 拥有此币的人可以将其转给其他人

**高飞生成一个新货币：**

* 字符串：CreateCoin[ CoinID ] // 这是新建的货币
* 高飞使用sk对字符串进行数字签名 // 由高飞创建，可用高飞的pk验证

**说明：**

高飞的pk应该是众所周知，以证明这是高飞创建的货币。

之后转移过程中会指明接收者的pk。

**高飞将货币转移给爱丽丝：**

* 字符串：将“此币（哈希指针）”转给“爱丽丝（pk）”
  + 此币 ：该币的哈希指针
  + 爱丽丝：爱丽丝的pk
* 高飞使用sk对这个字符串进行“数字签名”
  + 此前高飞拥有这个货币。

确认方法：用哈希指针找到之前的货币，证明货币是高飞创建的（ CreateCoin[ CoinID ]和高飞的数字签名 ）。

* + 当前货币也是高飞签名的。

确认方法：使用高飞的pk可以确认签名，表示高飞同意转移货币。

* + 确认完成之后，就可以确认这个声明（字符串）是合法的，所以此币现在属于爱丽丝（pk）。
  + 爱丽丝可以向任何人证明这个货币是她的，因为记录有爱丽丝的pk，只有用爱丽丝的sk才能转移这枚货币。

不可能冒名转移这枚货币，因为只有爱丽丝有pk对应的sk，使用sk才能转移这枚货币。如果有人使用其它sk进行签名，则货币中的pk将证明其签名无效。

**爱丽丝将货币转移给鲍勃：**

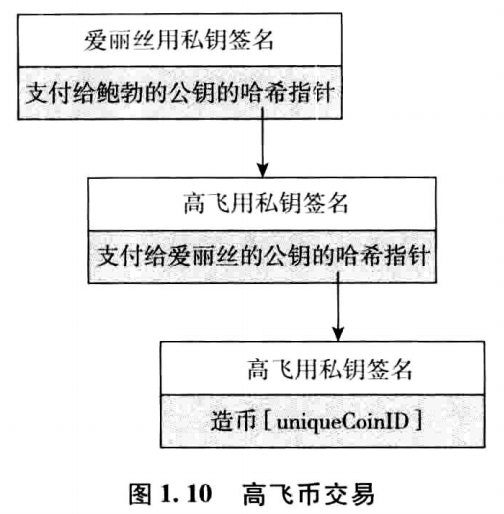
* 字符串：将“此币（哈希指针）”转给“鲍勃（pk）”
  + 此币：该币的哈希指针
  + 鲍勃：鲍勃的pk
* 爱丽丝使用sk对这个字符串进行数字签名
  + 此前爱丽丝拥有这个货币。

确认方法：用哈希指针找到之前的货币，证明货币被转移给了爱丽丝。

* + 当前货币也是爱丽丝签署的。

确认方法：使用爱丽丝的pk可以确认。

* + 都确认完成之后，就可以确认其声明是合法的，此币现在属于鲍勃。
  + 鲍勃同样可以向任何人证明这个货币是他的。（同爱丽丝的方法）
  + 任何人都可以验证一个币的有效性，跟随哈希指针追溯到它是由高飞创建的，并验证过程中的所有签名。



**【总结】**

|  |  |  |
| --- | --- | --- |
|  | **高飞币的实现** | **问题** |
| **货币发行** | 由高飞发行  可用高飞的pk进行验证 | 中心化问题：高飞可以随意发行货币 |
| **货币转移** | 1. 使用货币拥有者的sk对声明进行签名，   声明是将货币转移给接收者的pk   1. 使用“区块链”记录了货币转移历史记录 | 货币转移不是中心化，  但有“双重支付”问题 |

**双重支付（double spending）是任何加密货币都需要解决的主要问题之一。**

高飞币没有解决这个问题，所以不安全，不合适作为加密货币。

### 1.5.2财奴币（ScroogeCoin）

**【基本原理】**

Scrooge负责公布包含所有发生过的交易历史记录，它是一个只增账目（ledger）。

要求所有交易在被接受前都要写入账目，就可以用这账目防止“双重支付”。

因为，如果这个币之前已经转一个接收者，大家都能看到。

Scrooge使用“区块链”实现这个账目。

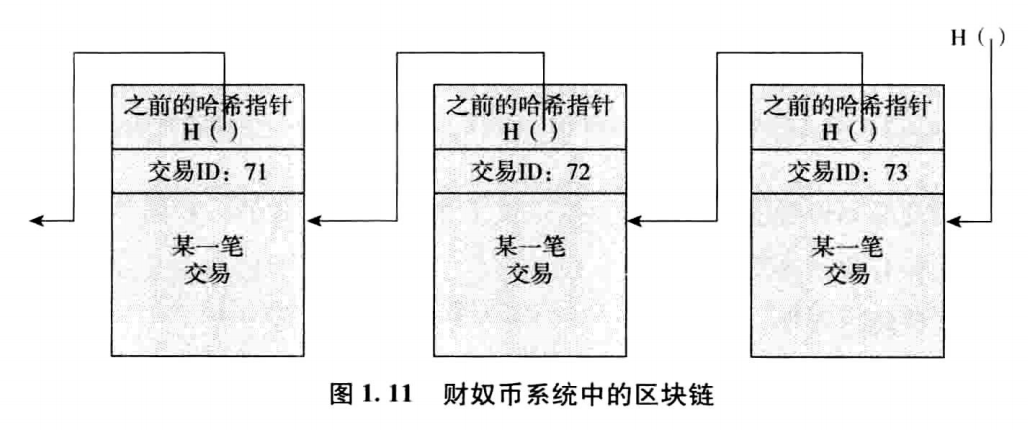
每个数据块包含一次交易，Scrooge针对最后一个哈希指针进行数字签名（它约束整个结构中所有的数据），并将签名与区块链一起发布。

ddk: 签名表示这是Scrooge亲自确认的（可用pk验证），并且数据有效的（不会被被篡改）。

只有Scrooge签名的区块链的交易才是有效的。

任何人都核查Scrooge对入口区块的签名，以验证一个交易是由Scrooge签名的。

由Scrooge来确保不会出现“双重支付”。



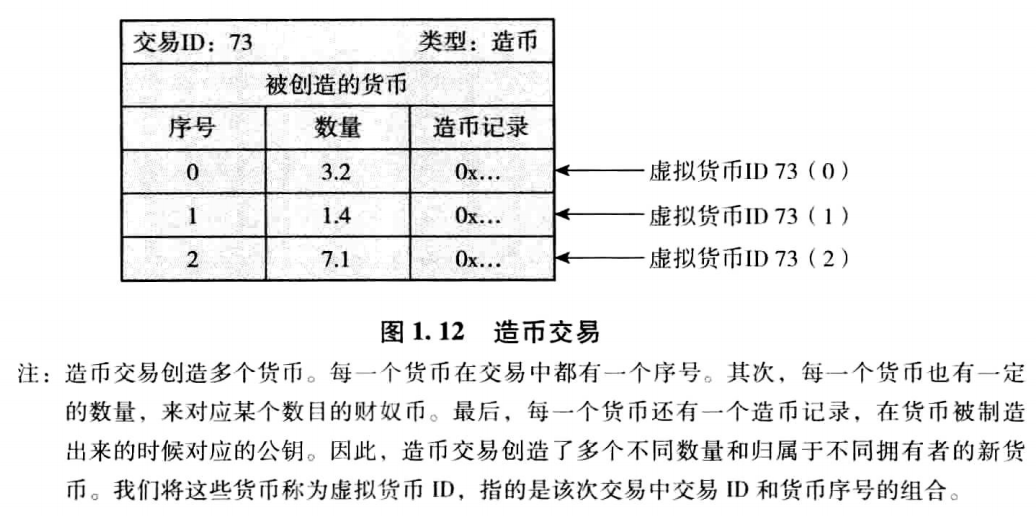
哈希指针用于保证只增特性

因为Scrooge有可能试图修改交易记录，有了哈希指针，只要有人监督Scrooge发布的最新哈希指针，变化会很明显，可以轻易被发现。

**【造币交易】**

**造币交易（CreateCoins）：**Scrooge在一次交易中可以创造多个币

如果造币交易是由财奴签署，则它就是有效的。



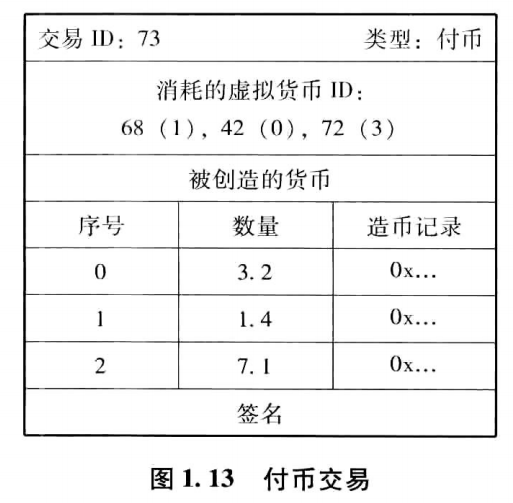
**【付币交易】**

**付币交易（PayCoins）**：这一交易会消耗币，并创建具有相同价值的新币。

这一交易必须由每个支付该币的人来进行签署，表示同意花掉这个币。

付币交易的有效性要求：

* 被消耗的币是有效货币
* 被消耗的币没有被双重支付
* 本次交易产生的币值量等于消息的币值量
* 本地交易被消耗的所有币均有所有者的有效签名



所有条件满足，则付币交易有效，Scrooge会接受交易，并将其附加到区块链上。

之后，所有人都可以看见交易发生了，这时，参与者才可接受交易实际发生了。

**【总结】**

|  |  |  |
| --- | --- | --- |
|  | **财奴币的实现** | **问题** |
| **货币发行** | 由Scrooge发行  可用Scrooge的pk进行验证 | 中心化问题：   * Scrooge可以随意发行新币 |
| **货币转移** | 所有交易由Scrooge判决有效，写入“只增账目”，并公布于众。  这样，人们可以看见哪些币是有效的，从而可防止“双重支付”。  Scrooge不能创建虚假交易（因为无法伪造其他人的签名）。 | 中心化问题：   * 可以停止其他用户的交易 * 强制收取交易费 * 停止更新区块链 |

**这里的问题是“中心化”，那么是否能让系统“去中心化”？**

即，很多方面像财奴币一样运作的加密货币，但没有中央信任机构。

**需要解决的问题：**

* 货币发行：通过去中心化的方式控制新币的产生
* 货币转移：
  + 所有用户必须一致同意：一个公开的“**区块链**”（已经发生的交易的历史记录）
  + 所有用户必须一致同意：哪些交易的有效的？哪些交易已经实际发生了？

如果可以解决这些问题，那就可以创建一个如同财奴币的货币，同时没有中心化的机构。

这样的系统就与比特币非常像了。

财奴币是一种以账本为基础的记账式加密货币。

**但其突出问题是“中心化”，所以第二章将介绍“比特币如何做到去中心化”。**