

“钱包”一词在比特币中有多重含义。广义上，钱包是一个应用程序，为用户提供交互界面。钱包控制用户访问权限，管理密钥和地址，跟踪余额以及创建和签名交易。狭义上，即从程序员的角度来看，“钱包”是指用于存储和管理用户密钥的数据结构。我们将深入介绍第二层含义，本章中钱包是私钥的容器，一般是通过结构化文件或简单数据库来实现。

5.1 钱包技术概述

在本节中，我们总结了各种技术，它们为用户构建起友好，安全和灵活的比特币钱包。

一个常见误解是，比特币钱包里含有比特币。事实上，钱包里只含有钥匙。“钱币”被记录在比特币网络的区块链中。用户通过钱包中的密钥签名交易，从而来控制网络上的钱币。在某种意义上，比特币钱包是密钥链。

提示 比特币钱包只含有密钥，而不是钱币。每个用户有一个包含多个密钥的钱包。钱包只包含私钥/公钥对的密钥链（请参阅[私钥章节]）。用户用密钥签名交易，从而证明他们拥有交易输出（他们的钱币）。钱币以交易输出的形式存储在区块链中（通常记为vout或txout）。

有两种主要类型的钱包，区别在于它们包含的多个密钥是否相互关联。

第一种类型是非确定性钱包（nondeterministic wallet），其中每个密钥都是从随机数独立生成的。密钥彼此无关。这种钱包也被称为“Just a Bunch Of Keys（一堆密钥）”，简称JBOK钱包。

第二种类型是确定性钱包（deterministic wallet），其中所有的密钥都是从一个主密钥派生出来，这个主密钥即为种子（seed）。该类型钱包中所有密钥都相互关联，如果有原始种子，则可以再次生成全部密钥。确定性钱包中使用了許多不同的密钥推导方法。最常用的推导方法是使用树状结构，称为分级确定性钱包或HD钱包。

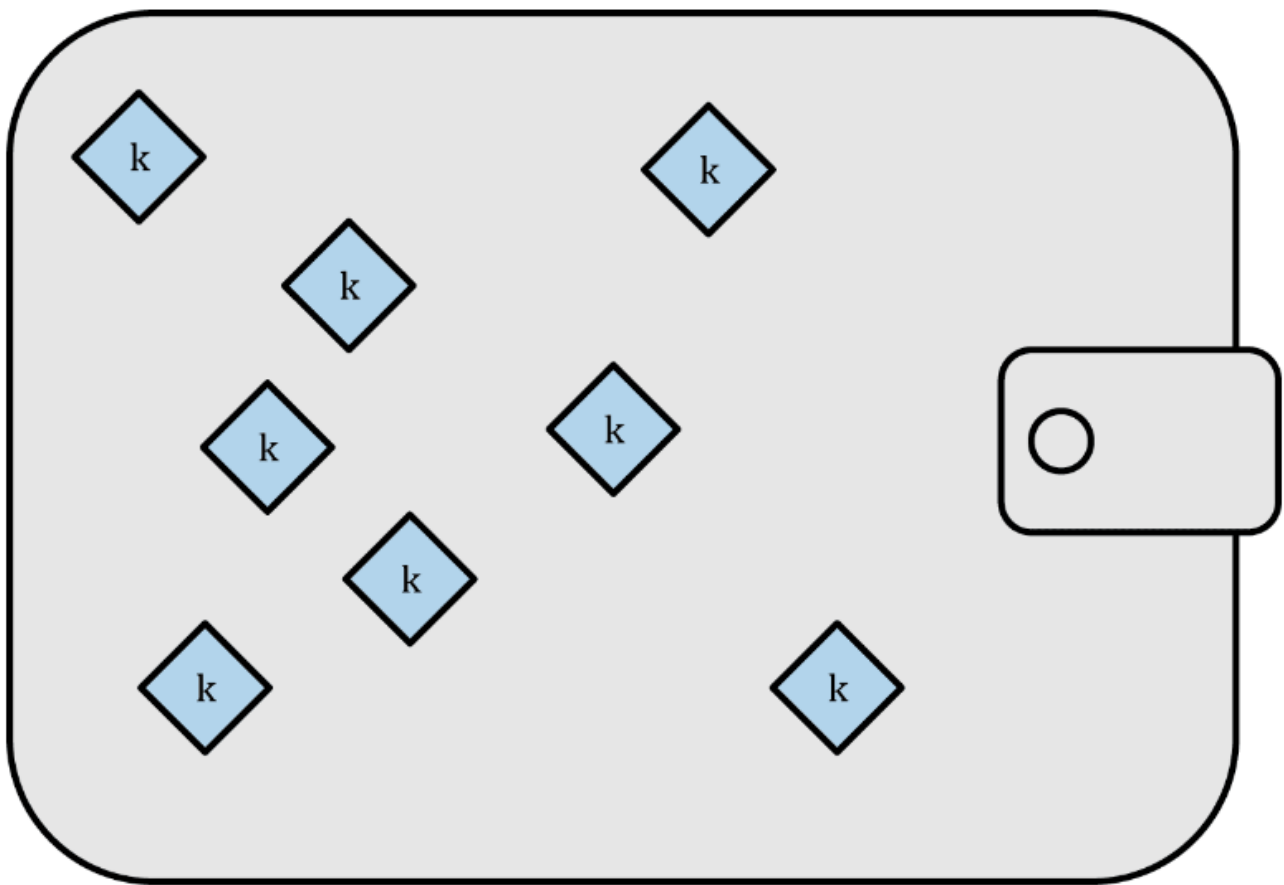
确定性钱包由种子衍生创造。为了便于使用，种子被编码为英文单词，也称为助记词。

接下来的几节将深入介绍这些技术。

5.1.1 非确定性（随机）钱包

在最早的一批比特币客户端中（Bitcoin Core，现在称作比特币核心客户端），钱包只是随机生成的私钥集合。这种类型的钱包被称作零型非确定钱包。举个例子，比特币核心客户端预先生成100个随机私钥，从最开始就生成足够多的私钥并且每个密钥只使用一次。这种钱包现在正在被确定性钱包替换，因为它们难以管理、备份以及导入。随机密钥的缺点就是如果你生成很多私钥，你必须保存它们所有的副本。这就意味着这个钱包必须被经常性地备份。每一个密钥都必须备份，否则一旦钱包不可访问时，钱包所控制的资金就付之东流。这种情况直接与避免地址重复使用的原则相冲突——每个比特币地址只能用一次交易。地址重复使用将多个交易和地址关联在一起，这会减少隐私。当你想避免重复使用地址时，零型非确定性钱包并不是好的选择，因为你要创造过多的私钥并且要保存它们。虽然比特币核心客户端包含零型钱包，但比特币的核心开发者并不鼓励大家使用。

图5-1展示的是一个非确定性钱包，其含有的随机密钥是个松散的集合。

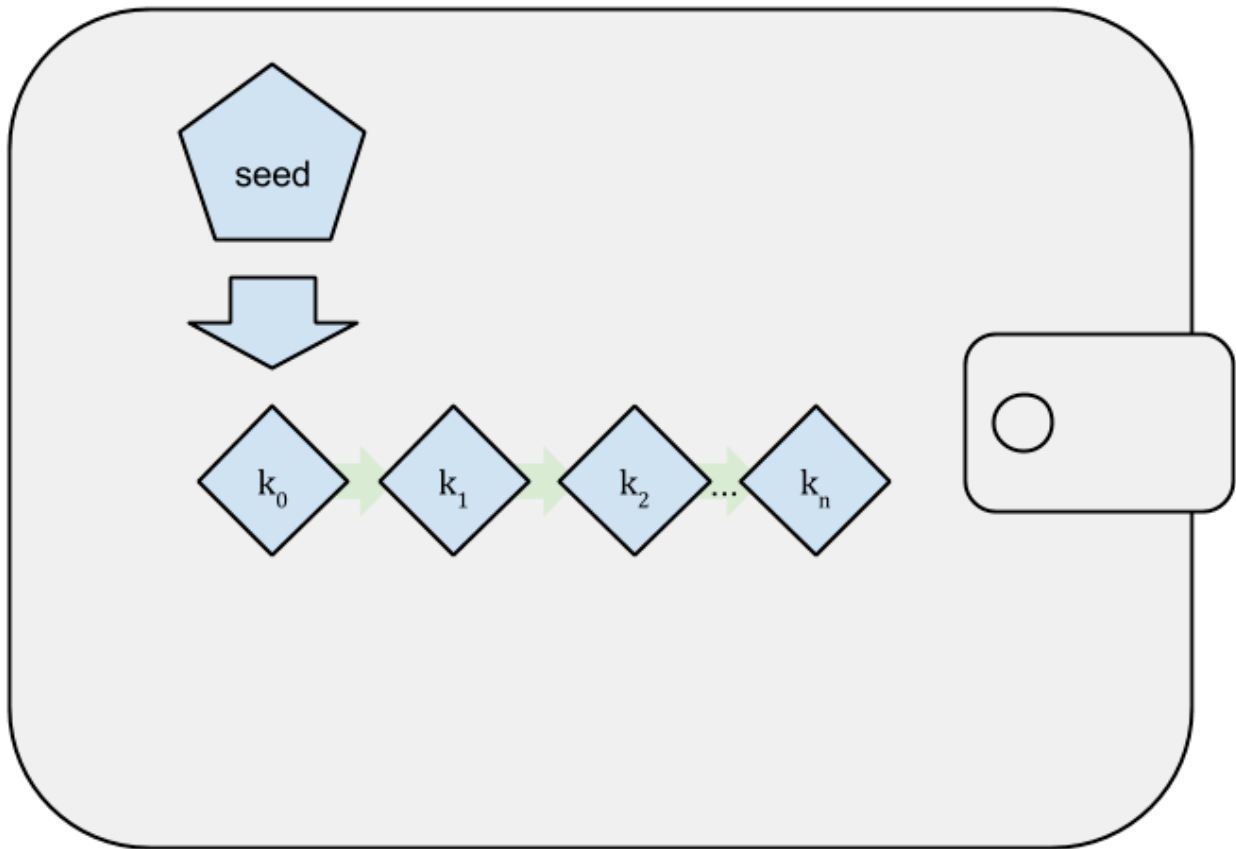


提示除了简单的测试之外，不要使用非确定性钱包。它们对于备份和使用来说太麻烦了。相反，推荐使用基于行业标准的HD钱包，可以用种子助记词进行备份。

5.1.2 确定性（种子）钱包

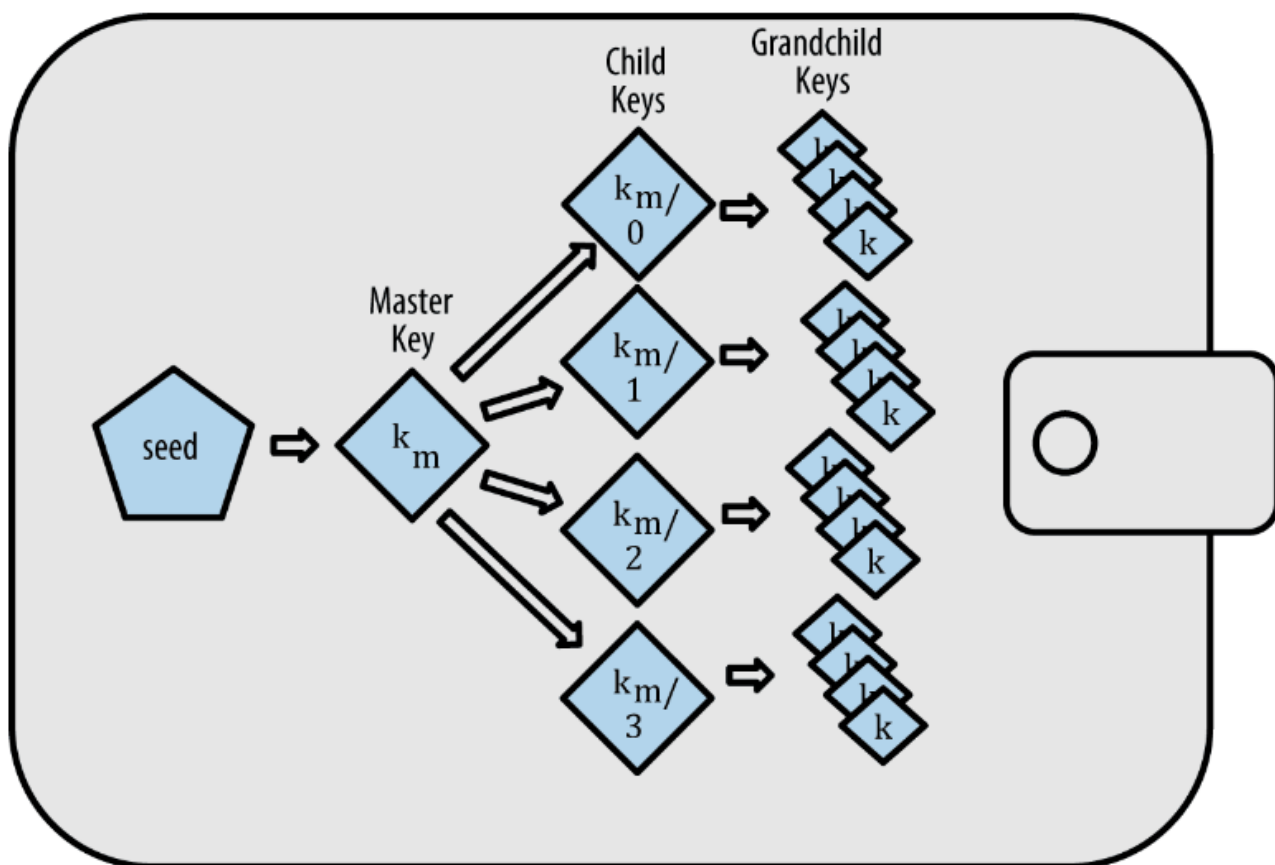
确定性，或者“种子”钱包包含通过使用单项离散函数从公共的种子生成的私钥。种子是随机生成的数字。这个数字也含有比如索引号码或者可生成私钥的“链码”（参见“分层确定性钱包（BIP0032/BIP0044）”一节）。在确定性钱包中，种子足够恢复所有的已经产生的私钥，所以只用在初始创建时的一个简单备份就足以搞定。并且种子也足够让钱包导入或者导出。这就很容易允许使用者的私钥在钱包之间轻松转移。

图5-2展示了确定性钱包的逻辑图。



5.1.3 分层确定性钱包（HD Wallets (BIP-32/BIP-44)）

确定性钱包被开发成更容易从单个“种子”中生成许多密钥。确定性钱包的最高级形式是通过BIP0032标准定义的HD钱包。HD钱包包含以树状结构衍生的密钥，使得父密钥可以衍生一系列子密钥，每个子密钥又可以衍生出一系列孙密钥，以此类推，无限衍生。图5-3展示了树状结构。



相比较随机（不确定性）密钥，HD钱包有两个主要的优势。第一，树状结构可以被用来表达额外的组织含义。比如当一个特定分支的子密钥被用来接收交易收入并且有另一个分支的子密钥用来负责支付花费。不同分支的密钥都可以被用在企业环境中，这就可以支配不同的分支部门、子公司、具体功能以及会计类别。

HD钱包的第二个好处就是它可以允许让使用者去建立一个公共密钥的序列而不需要访问相对应的私钥。这可允许HD钱包在不安全的服务器中使用或者在每笔交易中发行不同的公共钥匙。公共钥匙不需要被预先加载或者提前衍生，而在服务器中不需要可用来支付的私钥。

5.1.4种子和助记词（BIP-39）

HD钱包具有管理多个密钥和地址的强大机制。由一系列英文单词生成种子是个标准化的方法，这样易于在钱包中转移、导出和导入，如果HD钱包与这种方法相结合，将会更加有用。这些英文单词被称为助记词，标准由BIP-39定义。今天，大多数比特币钱包（以及其他加密货币的钱包）使用此标准，并可以使用可互操作的助记词导入和导出种子进行备份和恢复。

让我们从实际的角度来看以下哪种种子更容易抄录、阅读、导出以及导入。

16进制表示的种子：0C1E24E5917779D297E14D45F14E1A1A

助记词表示的种子：

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

5.1.5钱包最佳实践

由于比特币钱包技术已经成熟，出现了一些常见的行业标准，使得比特币钱包具备广泛互操作，易于使用，安全和灵活的特性。这些常用的标准是：

助记码，基于BIP-39

HD钱包，基于BIP-32

多用途HD钱包结构，基于BIP-43

多币种和多帐户钱包，基于BIP-44

这些标准可能会随着发展而改变或过时，但是现在它们形成了一套互锁技术，这些技术已成为比特币的事实上的钱包标准。

这些标准已被广泛的软件和硬件比特币钱包采用，使所有这些钱包互操作。用户可以导出在其中一个钱包上生成的助记符，并将其导入另一个钱包，实现恢复所有交易，密钥和地址。

列举支持这些标准的软件钱包，包括（按字母顺序排列）Breadwallet, Copay, Multibit HD和Mycelium。列举支持这些标准的硬件钱包，包括（按字母顺序排列）Keepkey, Ledger和Trezor。

以下部分将详细介绍这些技术。

提示如果您正准备开发一个比特币钱包，那么它应该被构建为一个HD钱包，一个种子被编码为助记词代码进行备份，遵循BIP-32, BIP-39, BIP-43和BIP-44标准，下面章节有所涉猎。

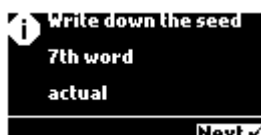
5.1.6使用比特币钱包

在[用户故事]中，我们介绍了Gabriel，里约热内卢是一个有进取心的少年，他正在经营一家简单的网络商店，销售比特币品牌的T恤，咖啡杯和贴纸。

Gabriel使用Trezor比特币硬件钱包（Trezor设备：硬件HD钱包）来安全地管理他的比特币。Trezor是一个简单的USB设备，具有两个按钮，用于存储密钥（以HD钱包的形式）和签署交易。Trezor钱包遵循本章讨论的所有行业标准，因此Gabriel不依赖于任何专有技术或单一供应商解决方案。



当Gabriel首次使用Trezor时，设备从内置的硬件随机数生成器生成助记词和种子。在这个初始化阶段，钱包在屏幕上按顺序逐个显示单词。



通过写下这个助记符，Gabriel创建了一个备份（参见表5-1），可以在Trezor设备丢失或损坏的情况下用于恢复。在新的Trezor钱包，或者任一种兼容的软件和硬件钱包中，助记词都可以用于恢复。 请注意，单词序列很重要，因此，记忆纸备份需要对每个单词都有空格。Gabriel必须仔细记录每个单词的编号，以保持正确的顺序。 表5-1Gabriel的助记器备份

1.	<i>army</i>	7.	<i>garbage</i>
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>

提示为了简单起见，Gabriel的助记词记录中显示了一个12个词。事实上，大多数硬件钱包生成更安全的24个词的助记符。助记词以完全相同的方式使用，不管长度如何。

作为网店的第一次实践，Gabriel使用他的Trezor设备生成一个比特币地址。所有客户的订单都使用此单一地址。我们将看到，这种方法有一些缺点，不过可以使用HD钱包进行改进。

5.2钱包技术细节

现在我们来深入了解被众多比特币钱包所使用的重要的行业标准。

5.2.1助记码词汇（BIP-39）

助记码词汇是英文单词序列代表（编码）用作种子对应所确定性钱包的随机数。单词的序列足以重新创建种子，并且从种子那里重新创造钱包以及所有私钥。在首次创建钱包时，带有助记码的，运行确定性钱包的钱包的应用程序将会向使用者展示一个12至24个词的顺序。单词的顺序就是钱包的备份。它也可以被用来恢复以及重新创造应用程序相同或者兼容的钱包的密钥。助记码词汇可以让使用者复制钱包更容易一些，因为相比较随机数字顺序来说，它们更容易地被阅读和正确抄写。

提示助记词经常与“脑钱包”混淆。他们不一样。主要区别在于脑钱包由用户选择的单词组成，而助记符是由钱包随机创建的，并呈现给用户。这个重要的区别使助记词更加安全，因为人类猜测随机数还是无能为力。

助记码被定义在比特币的改进建议39中（参见“附录 2 比特币改进协议[bip0039]”）。需要注意的是，BIP-39是助记码标准的一个实施方案。还有一个不同的标准，使用一组不同的单词，是由Electrum钱包使用，并且早于BIP-39。BIP-39由Trezor硬件钱包背后的公司提出，与Electrum的实施不兼容。然而，BIP-39现在已经在数十个可互操作的实践案例中获得了广泛的行业支持，应被视为事实上的行业标准。

BIP-39定义了助记符码和种子的创建，我们在这里描述了九个步骤。为了清楚起见，该过程分为两部分：

1-6步是创建助记词，7-9步是从助记词到种子。

5.2.2创建助记词

助记词是由钱包使用BIP-39中定义的标准化过程自动生成的。钱包从熵源开始，增加校验和，然后将熵映射到单词列表：

- 1、创建一个128到256位的随机序列（熵）。
- 2、提出SHA256哈希前几位（熵长/ 32），就可以创建一个随机序列的校验和。
- 3、将校验和添加到随机序列的末尾。
- 4、将序列划分为**包含11位**的不同部分。
- 5、将每个包含11位部分的值与一个已经预先定义2048个单词的字典做对应。
- 6、生成的有顺序的单词组就是助记码。

图5-6展示了熵如何生成助记词。

Mnemonic Words 128-bit entropy/12-word example

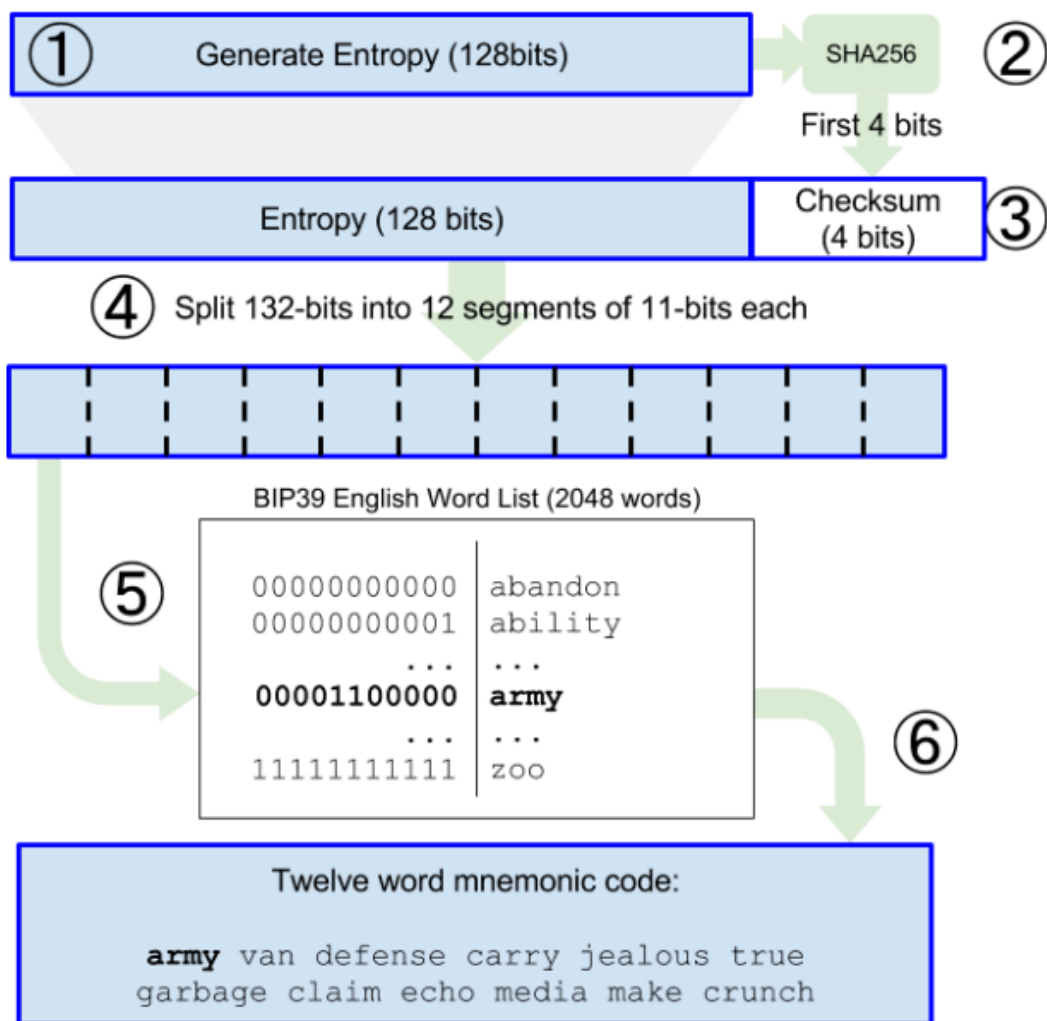


表5-2表示了熵数据的大小和助记词的长度之间的关系。

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

5.2.3从助记词生成种子

助记词表示长度为128至256位的熵。通过使用密钥延伸函数PBKDF2，熵被用于导出较长的（512位）种子。将所得的种子用于构建确定性钱包并得到其密钥。

密钥延伸函数有两个参数：助记词和盐。其中盐的目的是增加构建能够进行暴力攻击的查找表的困难度。在BIP-39标准中，盐具有另一目的，它允许引入密码短语（passphrase），作为保护种子的附加安全因素，我们将在BIP-39可选密码短语章节详细地描述。

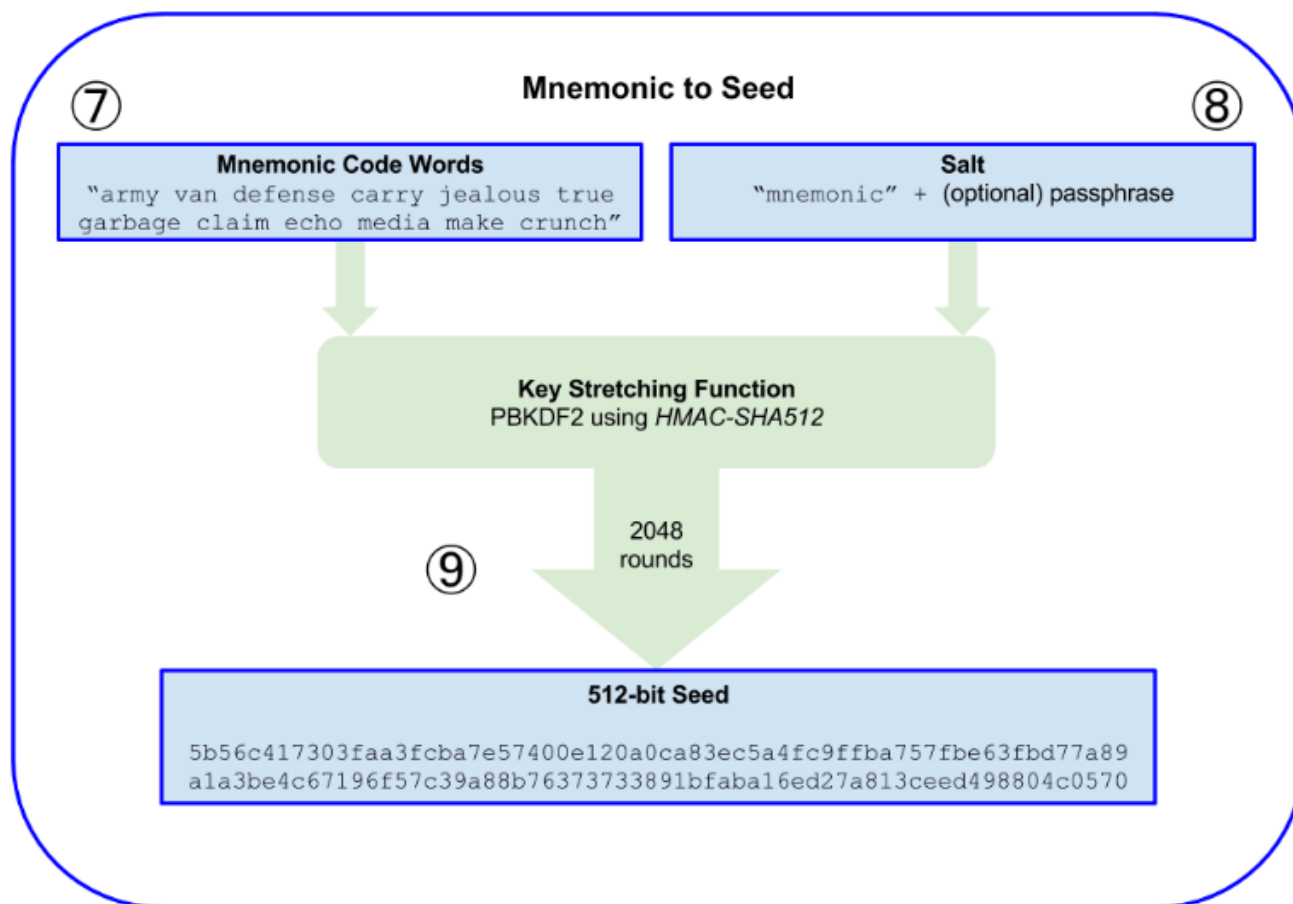
创建助记词之后的7-9步是：

7、PBKDF2密钥延伸函数的第一个参数是从步骤6生成的助记符。

8、PBKDF2密钥延伸函数的第二个参数是盐。由字符串常数“助记词”与可选的用户提供的密码字符串连接组成。

9、PBKDF2使用HMAC-SHA512算法，使用2048次哈希来延伸助记符和盐参数，产生一个512位的值作为其最终输出。这个512位的值就是种子。

图5-7显示了从助记词如何生成种子



提示 密钥延伸函数，使用2048次哈希是一种非常有效的保护，可以防止对助记词或密码短语的暴力攻击。它使得攻击尝试非常昂贵（从计算的角度），需要尝试超过几千个密码和助记符组合，而这样可能产生的种子的数量是巨大的（ 2^{512} ）。

表5-3、5-4和表5-5展示了一些助记码的例子和它所生成的种子。

Entropy input (128 bits)	0c1e24e5917779d297e14d45f14e1a1a
Mnemonic (12 words)	army van defense carry jealous true garbage claim echo media make crunch
Passphrase	(none)
Seed (512 bits)	5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fbd77a89a1a3be4c67196f57c39a88b76373733891bfaba16ed27a813ceed498804c0570

Entropy input (128 bits)	0c1e24e5917779d297e14d45f14e1a1a
Mnemonic (12 words)	army van defense carry jealous true garbage claim echo media make crunch
Passphrase	SuperDuperSecret
Seed (512 bits)	3b5df16df2157104cfdd22830162a5e170c0161653e3afe6c88defeefb0818c793dbb28ab3ab091897d0715861dc8a18358f80b79d49acf64142ae57037d1d54

Entropy input (256 bits)	2041546864449caff939d32d574753fe684d3c947c3346713dd8423e74abcf8c
Mnemonic (24 words)	cake apple borrow silk endorse fitness top denial coil riot stay wolf luggage oxygen faint major edit measure invite love trap field dilemma oblige
Passphrase	(none)
Seed (512 bits)	3269bce2674acbd188d4f120072b13b088a0ecf87c6e4cae41657a0bb78f5315b33b3a04356e53d062e55f1e0deaa082df8d487381379df848a6ad7e98798404

5.2.4 BIP-39中的可选密码短语

BIP-39标准允许在推导种子时使用可选的密码短语。如果没有使用密码短语，助记词是用由常量字符串“助记词”构成的盐进行延伸，从任何给定的助记词产生一个特定的512位种子。如果使用密码短语，密钥延伸函数使用同样的助记词也会产生不同的种子。事实上，给予一个单一的助记词，每一个可能的密码短语都会导致不同的种子。基本上没有“错误”的密码短语，所有密码短语都是有效的，它们都会导致不同的种子，形成一大批可能未初始化的钱包。这批钱包非常之大（ 2^{512} ），使用暴力破解或随机猜测基本不可能。

提示BIP-39中没有“错误的”密码短语。每个密码都会导致一些钱包，只是未使用的钱包是空的。

可选密码短语带来两个重要功能：

（存储在大脑中的）密码短语成为第二个因素，使得助记词不能单独使用，避免了助记词备份盗取后被利用。起到掩人耳目的效果，把密码短语指向有小额资金的钱包，分散攻击者注意力，使其不在关注拥有大额资金的“真实”钱包。

然而，需要注意的是，使用密码短语也会引起丢失的风险：

如果钱包所有者无行为能力或死亡，没有人知道密码，种子是无用的，所有存储在钱包中的资金都将永远丢失。相反，如果所有者将密码短语与种子备份在相同的地方，则违反了上述第二个因素的目的。虽然密码是非常有用的，但它们只能与仔细计划的备份和恢复流程结合使用，考虑到所有者个人风险的可能性，应该允许其家人恢复加密资产。

5.2.5 使用助记符代码

BIP-39被做成函数库，支持多种编程语言：

[python-mnemonic](#)

SatoshiLabs团队在Python中提出了BIP-39标准的参考实现

[bitcoinjs/bip39](https://github.com/bitcoinjs/bip39)

作为流行的bitcoinJS框架的一部分，在JavaScript中实现了BIP-39

[libbitcoin/mnemonic](https://github.com/libbitcoin/mnemonic)

作为流行的Libbitcoin框架的一部分，在C++中实现了BIP-39

还有一个BIP-39生成器在独立的网页中实现，对于测试和实验非常有用。图5-8展示一个独立的网页，可以生成助记词、种子和扩展私钥。

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

5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fbd77a89a1a3be4c67196f57c39a88b76373733891bfaba16ed27a813ceed498804c0570

Coin

Bitcoin

BIP32 Root Key

xprv9s21ZrQH143K3t4UZrNgeA3w861fwjYLaGwmPtQyPMmzshV2owVpfBSd2Q7YsHZ9j6i6ddYjb5PLtUdMZn8LhvuCVhGcQntq5rn7JVMqnie

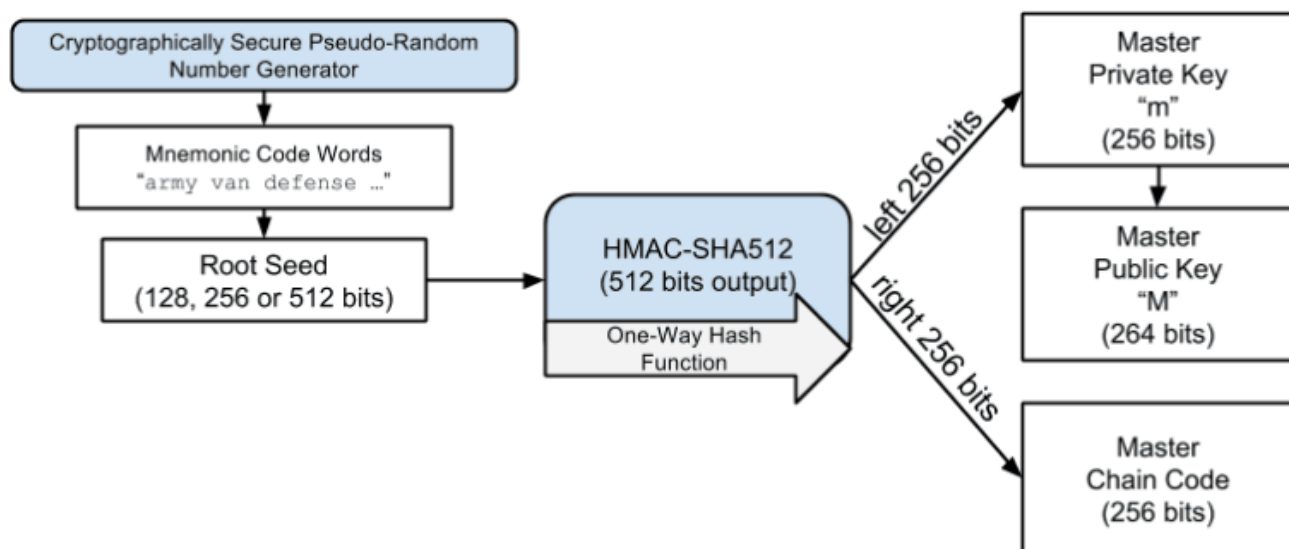
BIP-39生成器可以离线使用，也可以使用[这个在线地址](#).地址转向到[这儿了](#)【还得科学上网。译者注】

5.3从种子中创造HD钱包

HD钱包从单个根种子（root seed）中创建，为128到256位的随机数。最常见的是，这个种子是从助记符产生的，如上一节所述。

HD钱包的所有的确定性都衍生自这个根种子。任何兼容HD钱包的根种子也可重新创造整个HD钱包。所以简单的转移HD钱包的根种子就让HD钱包中所包含的成千上百万的密钥被复制，储存导出以及导入。

图5-9展示创建主密钥以及HD钱包的主链代码的过程。



根种子输入到HMAC-SHA512算法中就可以得到一个可用来创造主私钥(m) (master private key(m)) 和主链代码 (a master chain code) 的哈希。主私钥 (m) 之后可以通过使用我们在本章先前看到的那个普通椭圆曲线 $m * G$ 过程生来成相对应的主公钥 (M)。链代码用于从母密钥中创造子密钥的那个函数中引入熵。如下一节所示。

5.3.1私有子密钥的衍生

分层确定性钱包使用CKD (child key derivation)函数去从母密钥衍生出子密钥。

子密钥衍生函数是基于单项哈希函数。这个函数结合了：

一个母私钥或者公共钥匙 (ECDSA未压缩键)

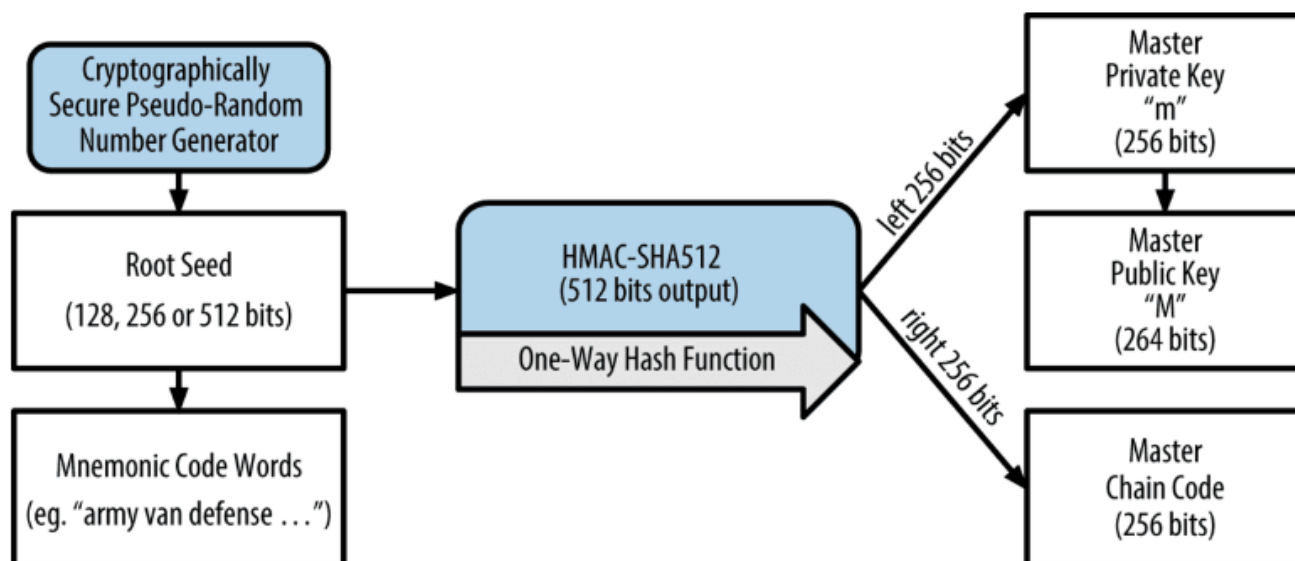
一个叫做链码 (256 bits) 的种子

一个索引号 (32 bits)

链码是用来给这个过程引入确定性随机数据的，使得索引不能充分衍生其他的子密钥。因此，有了子密钥并不能让它发现自己的姊妹密钥，除非你已经有了链码。最初的链码种子（在密码树的根部）是用随机数据构成的，随后链码从各自的母链码中衍生出来。

这三个项目（母私钥，链码，索引）相结合并散列可以生成子密钥，如下。

母公共钥匙——链码——以及索引号合并在一起并且用HMAC-SHA512函数散列之后可以产生512位的散列。所得的散列可被拆分为两部分。散列右半部分的256位产出可以给予链当链码。左半部分256位散列以及索引码被加载在母私钥上来衍生子私钥。在图5-10中，我们看到这个说明——索引集被设为0去生产母密钥的第0个子密钥（第一个通过索引）。



改变索引可以让我们延长母密钥以及创造序列中的其他子密钥。比如子0，子1，子2等等。每一个母密钥可以有2,147,483,647 (2^{31}) 个子密钥。 2^{31} 是整个 2^{32} 范围可用的一半，因为另一半是为特定类型的推导而保留的，我们将在本章稍后讨论。

向密码树下一层重复这个过程，每个子密钥可以依次成为母密钥继续创造它自己的子密钥，直到无限代。

5.3.2使用衍生的子密钥

子私钥不能从非确定性（随机）密钥中被区分出来。因为衍生函数是单向的，所以子密钥不能被用来发现他们的母密钥。子密钥也不能用来发现他们的相同层级的姊妹密钥。如果你有第 n 个子密钥，你不能发现它前面的（第 $n-1$ ）或者后面的子密钥（ $n+1$ ）或者在同一顺序中的其他子密钥。只有母密钥以及链码才能得到所有的子密钥。没有子链码的话，子密钥也不能用来衍生出任何孙密钥。你需要同时有子密钥以及对应的链码才能创建一个新的分支来衍生出孙密钥。

那子私钥自己可被用做什么呢？它可以用来做公钥和比特币地址。之后它就可以被用在那个地址来签署交易和支付任何东西。

提示子私钥、对应的公钥以及比特币地址都不能从随机创造的密钥和地址中被区分出来。事实是它们所在的序列，在创造他们的HD钱包函数之外是不可见的。一旦被创造出来，它们就和“正常”密钥一样运行了。

5.3.3扩展密钥

正如我们之前看到的，密钥衍生函数可以被用来创造密钥树上任何层级的子密钥。这只需要三个输入量：一个密钥，一个链码以及想要的子密钥的索引。密钥以及链码这两个重要的部分被结合之后，就叫做扩展密钥（extended key）。术语“extended key”也被认为是“可扩展的密钥”，因为这种密钥可以用来衍生子密钥。

扩展密钥可以简单地被储存并且表示为简单的将256位密钥与256位链码所并联的512位序列。有两种扩展密钥。扩展的私钥是私钥以及链码的结合。它可被用来衍生子私钥（子私钥可以衍生子公钥）。公钥以及链码组成扩展公钥，它可以用来扩展子公钥，见“生成公钥”章节。

想象一个扩展密钥作为HD钱包中密钥树结构的一个分支的根。你可以衍生出这个分支的剩下所有部分。扩展私钥可以创建一个完整的分支，而扩展公钥只能够创建一个公钥的分支。

提示一个扩展密钥包括一个私钥（或者公钥）以及一个链码。一个扩展密钥可以创造出子密钥并且能创造出密钥树结构中的整个分支。分享扩展密钥就可以访问整个分支。

扩展密钥通过Base58Check来编码，从而能轻易地在不同的BIP-32兼容钱包之间导入导出。扩展密钥编码用的Base58Check使用特殊的版本号，这导致在Base58编码字符中，出现前缀“xprv”和“xpub”。这种前缀可以让编码更易被识别。因为扩展密钥是512或者513位，所以它比我们之前所看到的Base58Check编码串更长一些。

以下面的扩展私钥为例，其使用的是Base58Check编码：

```
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i6GoNMKUga5biW6Hx4tws2six3b9c
```

这是上面扩展私钥对应的扩展公钥，同样使用Base58Check编码：

```
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVKZBPLrtJunSDMstweyLXhRgPxdp14sk9tjPW9
```

5.3.4公共子密钥推导

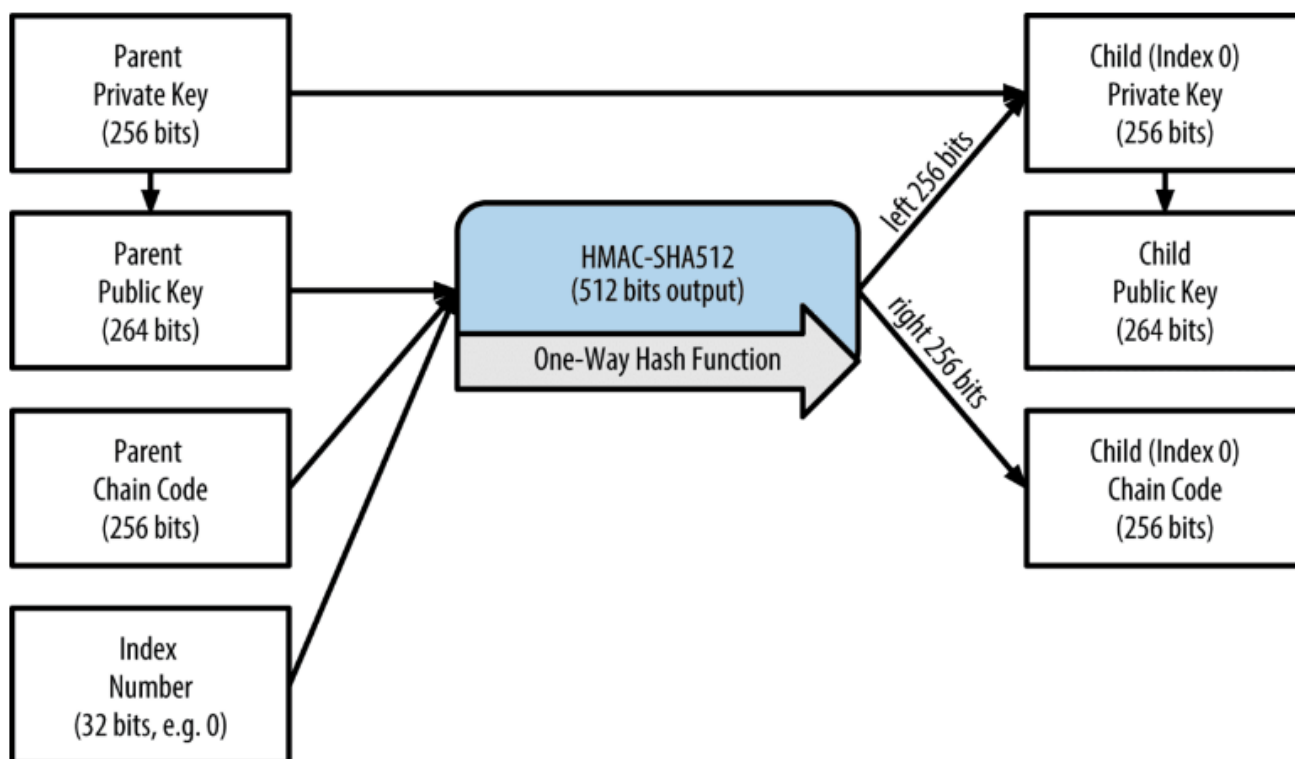
正如之前提到的，分层确定性钱包的一个很有用的特点就是可以不通过私钥而直接从公共母密钥派生出公共子密钥的能力。这就给了我们两种衍生子公钥的方法：或者通过子私钥，再或者就是直接通过母公钥。

因此，扩展密钥可以在HD钱包结构的分支中，被用来衍生所有的公钥（且只有公钥）。

这种快捷方式可以用来创造非常保密的只有公钥配置。在配置中，服务器或者应用程序不管有没有私钥，都可以有扩展公钥的副本。这种配置可以创造出无限数量的公钥以及比特币地址。但是发送到这个地址里的任何比特币都不能使用。与此同时，在另一种更保险的服务器上，扩展私钥可以衍生出所有的对应的可签署交易以及花钱的私钥。

这种方案的常见应用是安装扩展公钥电商的网络服务器上。网络服务器可以使用这个公钥衍生函数去给每一笔交易（比如客户的购物车）创造一个新的比特币地址。但为了避免被偷，网络服务商不会有任何私钥。没有HD钱包的话，唯一的方法就是在不同的安全服务器上创造成千上万个比特币地址，之后就提前上传到电商服务器上。这种方法比较繁琐而且要求持续的维护来确保电商服务器不“用光”公钥。

这种解决方案的另一种常见的应用是冷藏或者硬件钱包。在这种情况下，扩展的私钥可以被储存在纸质钱包中或者硬件设备中（比如 Trezor 硬件钱包），与此同时扩展公钥可以在线保存。使用者可以根据意愿创造“接收”地址而私钥可以安全地在线下被保存。为了支付资金，使用者可以使用扩展的私钥离线签署比特币客户或者通过硬件钱包设备（比如 Trezor）签署交易。图5-11阐述了扩展母公钥来衍生子公钥的传递机制。



5.3.5在网店中使用扩展公钥（xpub）

继续Gabriel网店的故事，让我们看看Gabriel是如何使用HD钱包。

Gabriel在一个网络上的托管服务器上建立一个简单的WordPress页面，作为他的网上商店。它的网店非常简单，只有几个页面和一张带有一个比特币地址的订单。

Gabriel使用他的Trezor设备生成的第一个比特币地址作为他的商店的主要比特币地址。这样，所有收到的付款都将支付给他的Trezor硬件钱包所控制的地址。

客户可以使用表格提交订单，并向Gabriel发布的比特币地址发送付款，触发一封电子邮件，其中包含Gabriel的订单详细信息。每周只几个订单，这个系统运行得很好。

然而，这个小型网络商店变得相当成功，并吸引了很多来自当地社区的订单。Gabriel很快就不堪重负。由于所有订单都支付相同的地址，因此很难正确匹配订单和交易，特别是当同一数量的多个订单紧密相连时。

HD钱包可以在不知道私钥的情况下获取公共子密钥，该能力为Gabriel提供了更好的解决方案。Gabriel可以在他的网站上加载一个扩展公钥（xpub），这可以用于为每个客户订单导出唯一的地址。Gabriel可以花费他在Trezor里资金，但加载在网站上的xpub只能生成地址并收到资金。HD钱包的这个功能非常安全。Gabriel的网站不包含任何私钥，因此不需要高级别的安全性。

为了导出xpub，Gabriel将基于Web的软件与Trezor硬件钱包配合使用。必须插入Trezor设备才能导出公钥。请注意，硬件钱包永远不会导出私钥，这些密钥始终保留在设备上。图5-12显示了Gabriel用于导出xpub的Web界面。

BasicHomescreenAdvanced

LabelGabriel's TrezorChange label


PIN protectionEnabledChange PIN

Total balance0.00 BTC

Account public keys (XPUB)

xpub6Cy7dUR4ZKF22HEuVq7epRgRsoXfL2MK1RE81CSvp1ZySySoYGXk5PUY9y9Cc5ExpnSwXyi mQAsVhyypDNDrfj4xjDsKZJNYgsHXoEPNCYQ

Be careful with your XPUBs. When you give them to a third party, you allow it to see your whole transaction history. Learn more

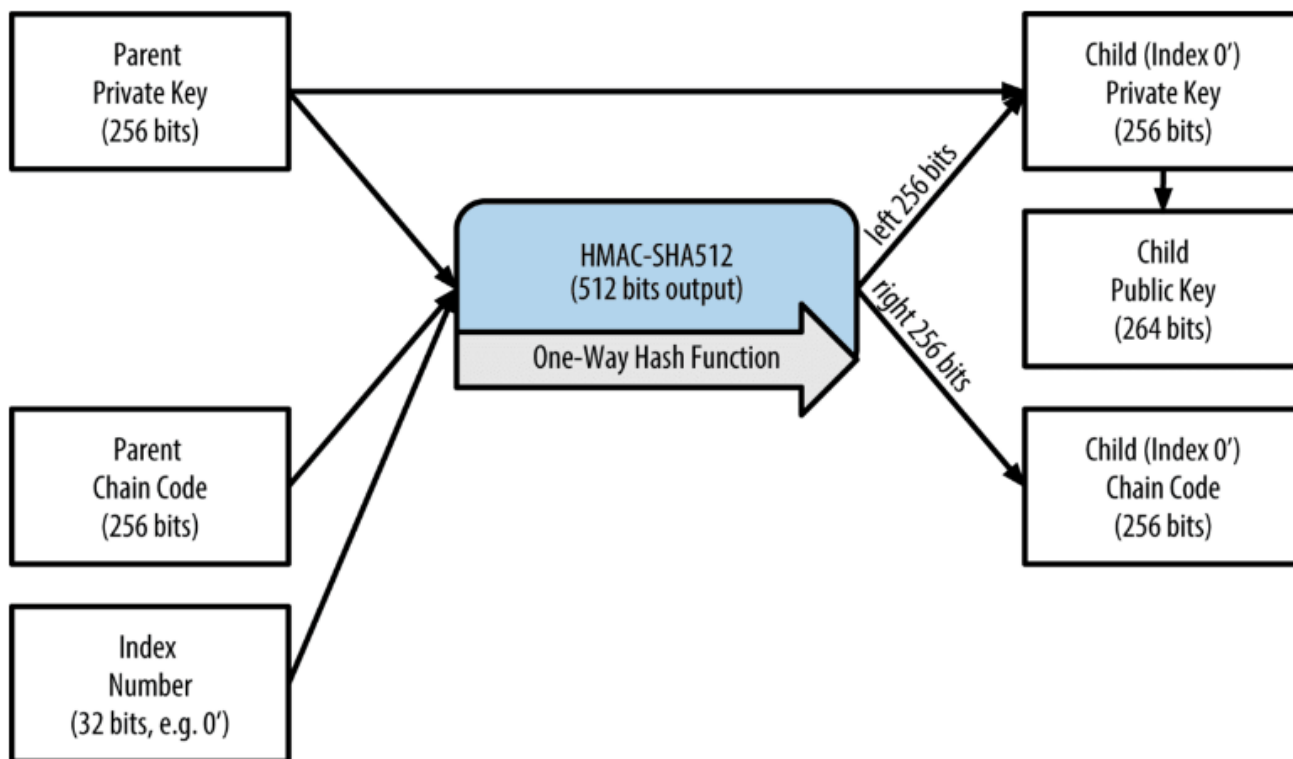


Gabriel将xpub复制到他的网店的比特币购物软件中。他使用的软件是Mycelium Gear，它是一个网店的开源插件，用于各种网络托管和内容平台。Mycelium Gear使用xpub为每次购买生成一个唯一的地址。

5.3.6硬化子密钥的衍生

从扩展公钥衍生一个分支公钥的能力是很重要的，但牵扯一些风险。访问扩展公钥并不能得到访问子私钥的途径。但是，因为扩展公钥包含链码，如果子私钥被知道或者被泄漏的话，链码就可以被用来衍生所有的其他子私钥。简单地泄露的私钥以及一个母链码，可以暴露所有的子密钥。更糟糕的是，子私钥与母链码可以用来推断母私钥。

为了应对这种风险，HD钱包使用一种叫做硬化衍生(hardened derivation)的替代衍生函数。这就“打破”了母公钥以及子链码之间的关系。这个硬化衍生函数使用了母私钥去推导子链码，而不是母公钥。这就在母/子顺序中创造了一道“防火墙”——有链码但并不能够用来推算子链码或者姊妹私钥。强化衍生函数看起来几乎与一般的衍生的子私钥相同，不同的是母私钥被用来输入散列函数中而不是母公钥，如图5-13所示。



当强化私钥衍生函数被使用时，得到的子私钥以及链码与使用一般衍生函数所得到的结果完全不同。得到的密钥“分支”可以被用来生产不易被攻击的扩展公钥，因为它所含的链码不能被用来开发或者暴露任何私钥。强化衍生也因此被用在上一层级，使用扩展公钥的密钥树中创造“间隙”。

简单地来说，如果你想要利用扩展公钥的便捷来衍生公钥的分支而不将你自己暴露在泄露扩展链码的风险下，你应该从强化母私钥衍生公钥，而不是从一般的母私钥来衍生。最好的方式是，为了避免了推到出主密钥，主密钥所衍生的第一层级的子密钥最好使用强化衍生。

5.3.7 正常衍生和强化衍生的索引号码

用在衍生函数中的索引号码是32位的整数。为了区分密钥是从正常衍生函数中衍生出来还是从强化衍生函数中产出，这个索引号被分为两个范围。索引号在0和 $2^{31}-1$ (0x0 to 0x7FFFFFFF)之间的是只被用在常规衍生。索引号在 2^{31} 和 $2^{32}-1$ (0x80000000 to 0xFFFFFFFF)之间的只被用在强化衍生。因此，索引号小于 2^{31} 就意味着子密钥是常规的，而大于或者等于 2^{31} 的子密钥就是强化型的。

为了让索引号码更容易被阅读和展示，强化子密钥的索引号码是从0开始展示的，但是右上角有一个小撇号。第一个常规子密钥因此被表述为0，但是第一个强化子密钥（索引号为0x80000000）就被表示为0'。第二个强化密钥依序有了索引号0x80000001，且被显示为1'，以此类推。当你看到HD钱包索引号i'，这就意味着 $2^{31}+i$ 。

5.3.8 HD钱包密钥识别符（路径）

HD钱包中的密钥是用“路径”命名的，且每个级别之间用斜杠 (/) 字符来表示（见表5-6）。由主私钥衍生出的私钥起始以“m”打头。由主公钥衍生的公钥起始以“M”打头。因此，母密钥生成的第一个子私钥是m/0。第一个公钥是M/0。第一个子密钥的子密钥就是m/0/1，以此类推。

密钥的“祖先”是从右向左读，直到你达到了衍生出的它的主密钥。举个例子，标识符m/x/y/z描述的是子密钥m/x/y的第z个子密钥。而子密钥m/x/y又是m/x的第y个子密钥。m/x又是m的第x个子密钥。

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 of the first child (m/0)
m/0'/0	The first normal grandchild of the first <i>hardened</i> child (m/0')
m/1/0	The first grandchild private key of the second child (m/1)
M/23/17/0/0	The first great-great-grandchild public key of the first great-grandchild of the 18th grandchild of the 24th child

5.3.9 HD钱包树状结构的导航

HD钱包树状结构提供了极大的灵活性。每一个母扩展密钥有40亿个子密钥：20亿个常规子密钥和20亿个强化子密钥。而每个子密钥又会有40亿个子密钥并且以此类推。只要你愿意，这个树结构可以无限类推到无穷代。但是，又由于有了这个灵活性，对无限的树状结构进行导航就变得异常困难。尤其是对于在不同的HD钱包之间进行转移交易，因为内部组织到内部分支以及亚分支的可能性是无穷的。

两个比特币改进建议（BIPs）提供了这个复杂问题的解决办法——通过创建几个HD钱包树的提议标准。BIP-43提出使用第一个强化子索引作为特殊的标识符表示树状结构的“purpose”。基于BIP-43，HD钱包应该使用且只用第一层级的树的分支，而且有索引号码去识别结构并且有命名空间来定义剩余的树的目的地。举个例子，HD钱包只使用分支m/i'是为了表明那个被索引号“i”定义的特殊为目的地。

在BIP-43标准下，为了延长的那个特殊规范，BIP-44提议了多账户结构作为“purpose”。所有遵循BIP-44的HD钱包依据只使用树的第一个分支的要求而被定义：m/44'/. BIP-44指定了包含5个预定义树状层级的结构：

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

第一层的purpose总是被设定为44'。

第二层的“coin_type”特指币种并且允许多元货币HD钱包中的货币在第二个层级下有自己的亚树状结构。目前有三种货币被定义：Bitcoin is m/44'/0'、Bitcoin Testnet is m/44'/1'，以及 Litecoin is m/44'/2'。

树的第三层级是“account”，这可以允许使用者为了会计或者组织目的，而去再细分他们的钱包到独立的逻辑性亚账户。举个例子，一个HD钱包可能包含两个比特币“账户”：m/44'/0'/0'和m/44'/0'/1'。每个账户都是它自己亚树的根。

第四层级就是“change”。每一个HD钱包有两个亚树，一个是用来接收地址一个是用来创造找零地址。注意无论先前的层级是否使用强化衍生，这一层级使用的都是常规衍生。这是为了允许这一层级的树可以在不安全环境下，输出扩展公钥。

被HD钱包衍生的可用的地址是第四层级的子级，就是第五层级的树的“address_index”。比如，第三个层级的主账户收到比特币支付的地址就是 M/44'/0'/0'/0/2。表5-7展示了更多的例子。

HD 路径	主要描述
M/44'/0'/0'/0/2	第三个收到公共密钥的主比特币账户
M/44'/0'/3'/1/14	第十五改变地址公钥的第四个比特币账户
m/44'/2'/0'/0/1	为了签署交易的在莱特币主账户的第二个私钥