5钱包

The word "wallet" is used to describe a few different things in bitcoin. At a high level, a wallet is an application that serves as the primary user interface. The wallet controls access to a user’s money, managing keys and addresses, tracking the balance, and creating and signing transactions. More narrowly, from a programmer’s perspective, the word "wallet" refers to the data structure used to store and manage a user’s keys.
“钱包”在比特币中有多重含义：

- 广义：钱包是一个应用程序，作为主要的用户接口。钱包控制访问用户的钱、管理密钥和地址、跟踪余额、创建和签名交易。
- 狭义：从程序员的角度看，钱包是指数据结构，用于存储和管理用户的密钥。

In this chapter we will look at the second meaning, where wallets are containers for private keys, usually implemented as structured files or simple databases.
本节介绍狭义的钱包，即钱包是私钥的容器，一般实现为结构化文件或简单数据库。

5.1钱包技术概述

In this section we summarize the various technologies used to construct user-friendly, secure, and flexible bitcoin wallets.
在本节中，我们总结了“用于构建友好、安全、灵活的比特币钱包”的各种技术。

A common misconception about bitcoin is that bitcoin wallets contain bitcoin. In fact, the wallet contains only keys. The "coins" are recorded in the blockchain on the bitcoin network. Users control the coins on the network by signing transactions with the keys in their wallets. In a sense, a bitcoin wallet is a *keychain*.
一个常见误解是：比特币钱包里有比特币。
事实上，钱包里只有密钥。比特币被记录在比特币网络的区块链中。
用户使用钱包中的密钥对交易进行签名，从而来控制网络上的比特币。
在某种意义上，比特币钱包是钥匙链（keychain）。ddk：钥匙链的意思是它是用来保存钥匙的。

Tip：Bitcoin wallets contain keys, not coins. Each user has a wallet containing keys. Wallets are really keychains containing pairs of private/public keys (see [private_public_keys]). Users sign transactions with the keys, thereby proving they own the transaction outputs (their coins). The coins are stored on the blockchain in the form of transaction outputs (often noted as vout or txout).
提示：比特币钱包中只有密钥，没有比特币。
每个用户有一个钱包，其中包含了密钥。钱包只是包含私钥/公钥的钥匙链。
用户用密钥对交易进行签名，以证明他们拥有交易输出（他们的比特币）。
比特币以交易输出的形式存储在区块链中（通常记为vout或txout）。

There are two primary types of wallets, distinguished by whether the keys they contain are related to each other or not.
有两种主要类型的钱包，区别是：钱包包含的密钥之间是否相互关联。

The first type is a *nondeterministic wallet*, where each key is independently generated from a random number. The keys are not related to each other. This type of wallet is also known as a JBOK wallet from the phrase "Just a Bunch Of Keys."
第1种类型是：非确定性钱包（nondeterministic wallet）。
其中每个密钥都是用一个随机数独立生成的，密钥之间没有关联。
这种钱包也称为JBOK钱包。（JBOK：Just a Bunch Of Keys一堆密钥）

The second type of wallet is a *deterministic wallet*, where all the keys are derived from a single master key, known as the *seed*. All the keys in this type of wallet are related to each other and can be generated again if one has the original seed. There are a number of different *key derivation* methods used in deterministic wallets. The most commonly used derivation method uses a tree-like structure and is known as a *hierarchical deterministic* or *HD* wallet.
第2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，这个主密钥称为种子（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许多不同的密钥导出方法。
最常用的导出方法使用树状结构，称为“分层确定性（HD）钱包”。

Deterministic wallets are initialized from a seed. To make these easier to use, seeds are encoded as English words, also known as *mnemonic code words*.
确定性钱包是用一个种子进行初始化。
为了更容易使用，种子被编码为英文单词，也称为“助记词”。

The next few sections introduce each of these technologies at a high level.
下面几节将在一个高层次上介绍每种技术。

5.1.1非确定性（随机）钱包

In the first bitcoin wallet (now called Bitcoin Core), wallets were collections of randomly generated private keys. For example, the original Bitcoin Core client pregenerates 100 random private keys when first started and generates more keys as needed, using each key only once.
在第一个比特币钱包中（Bitcoin Core），钱包是随机生成的私钥的集合。
例如，最初的Bitcoin Core客户端在第一次启动时生成100个随机私钥，当需要时在生成更多密钥，每个密钥只使用一次。

Such wallets are being replaced with deterministic wallets because they are cumbersome to manage, back up, and import. The disadvantage of random keys is that if you generate many of them you must keep copies of all of them, meaning that the wallet must be backed up frequently. Each key must be backed up, or the funds it controls are irrevocably lost if the wallet becomes inaccessible. This conflicts directly with the principle of avoiding address reuse, by using each bitcoin address for only one transaction. Address reuse reduces privacy by associating multiple transactions and addresses with each other.
这种钱包已经被替换为确定性钱包，因为它们难以管理、备份和导入。
随机密钥的缺点是，如果你生成了很多密钥，你必须保存所有的副本，这就意味着，必须经常备份钱包。
每个密钥都必须备份，否则一旦钱包丢失，资金也就丢了。
这直接与避免地址重复使用的原则相冲突——每个比特币地址只用于一次交易。
地址重用将多个交易和地址关联在一起，从而减低了隐私性。

A Type-0 nondeterministic wallet is a poor choice of wallet, especially if you want to avoid address reuse because it means managing many keys, which creates the need for frequent backups. Although the Bitcoin Core client includes a Type-0 wallet, using this wallet is discouraged by developers of Bitcoin Core. Type-0 nondeterministic (random) wallet: a collection of randomly generated keys shows a nondeterministic wallet, containing a loose collection of random keys.
Type-0非确定性钱包是很差的钱包选择，特别是如果你想避免地址重用，因为它要管理许多密钥，这样就要经常备份。
虽然Bitcoin Core客户端包含Type-0钱包，但Bitcoin Core的开发者并不鼓励使用这种钱包。
图1显示了一个非确定性钱包，包含一些松散的随机密钥。

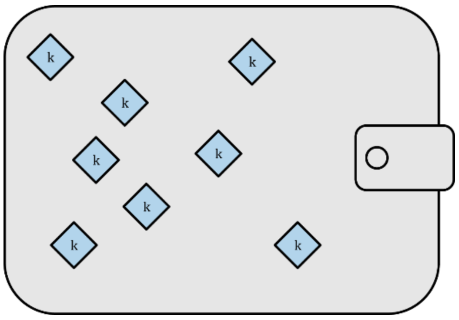


Figure 1. Type-0 nondeterministic (random) wallet: a collection of randomly generated keys
图1：Type-0非确定性（随机）钱包，包含了随机生成的密钥。

Tip: The use of nondeterministic wallets is discouraged for anything other than simple tests. They are simply too cumbersome to back up and use. Instead, use an industry-standard–based *HD wallet* with a *mnemonic* seed for backup.
提示：除了简单测试之外，不要使用非确定性钱包。
它们难以备份和使用。推荐使用基于业界标准的HD钱包，它用一个助记词种子来备份。

5.1.2 确定性（种子）钱包

Deterministic, or "seeded," wallets are wallets that contain private keys that are all derived from a common seed, through the use of a one-way hash function. The seed is a randomly generated number that is combined with other data, such as an index number or "chain code" (see HD Wallets (BIP-32/BIP-44)) to derive the private keys.
确定性（种子）钱包包含的所有密钥可以这样导出：通过使用一个单向哈希函数从一个公共种子得到。
种子是一个随机生成的数字，这个数字与其它数字合并，例如一个索引号或“链码”,从而导出私钥。

In a deterministic wallet, the seed is sufficient to recover all the derived keys, and therefore a single backup at creation time is sufficient. The seed is also sufficient for a wallet export or import, allowing for easy migration of all the user’s keys between different wallet implementations. Type-1 deterministic (seeded) wallet: a deterministic sequence of keys derived from a seed shows a logical diagram of a deterministic wallet.
在确定性钱包中，种子足以恢复所有导出的密钥，所以备份种子就够了。
种子对于钱包导出或导入也是足够了，这样这就很容易在不同的钱包之间转移用户的密钥。
图2显示了确定性钱包的一个逻辑图。

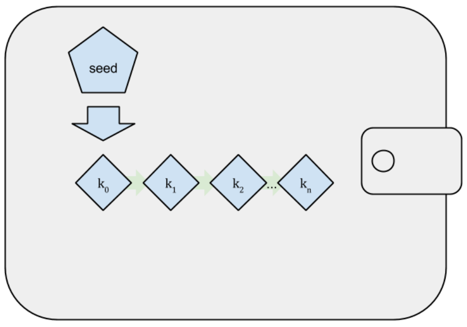


Figure 2. Type-1 deterministic (seeded) wallet: a deterministic sequence of keys derived from a seed
图2：Type-1确定性（种子）钱包：从一个种子导出一系列确定的密钥

5.1.3 分层确定性（HD）钱包

Deterministic wallets were developed to make it easy to derive many keys from a single "seed." The most advanced form of deterministic wallets is the HD wallet defined by the BIP-32 standard. HD wallets contain keys derived in a tree structure, such that a parent key can derive a sequence of children keys, each of which can derive a sequence of grandchildren keys, and so on, to an infinite depth. This tree structure is illustrated in Type-2 HD wallet: a tree of keys generated from a single seed.
确定性钱包可以很容易从一个种子中导出许多密钥。
确定性钱包的最高级形式是BIP-32定义的HD钱包。
HD钱包包含以树状结构导出的密钥，这样，父密钥可以导出一系列子密钥，每个子密钥又可以导出一系列孙密钥，以此类推，无限导出。图3显示了这种树状结构。

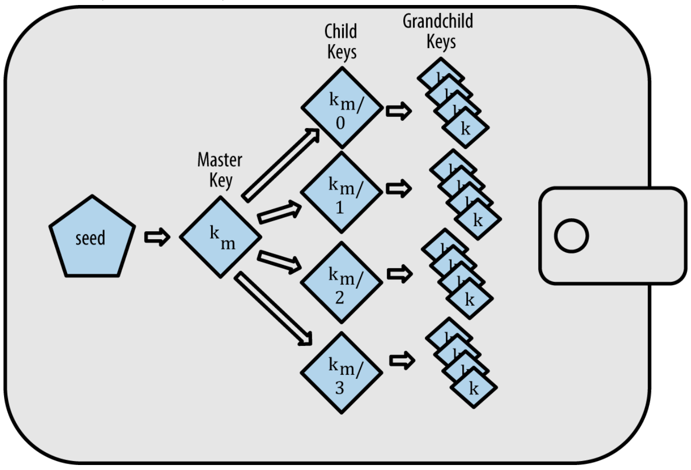


Figure 3. Type-2 HD wallet: a tree of keys generated from a single seed
图3：Type-2 HD钱包：从一个种子导出一个密钥树

HD wallets offer two major advantages over random (nondeterministic) keys.
相比随机（非确定）密钥，HD钱包有两个主要的优点。

First, the tree structure can be used to express additional organizational meaning, such as when a specific branch of subkeys is used to receive incoming payments and a different branch is used to receive change from outgoing payments. Branches of keys can also be used in corporate settings, allocating different branches to departments, subsidiaries, specific functions, or accounting categories.
第1，树状结构可以被用来表达组织含。
例如，一个子密钥分支用来接收收入支付，另一个子密钥分支用来接收找零。
密钥分支还可以用于企业环境，给部门、子公司、特定职能、会计类别分配不同的分支。

The second advantage of HD wallets is that users can create a sequence of public keys without having access to the corresponding private keys. This allows HD wallets to be used on an insecure server or in a receive-only capacity, issuing a different public key for each transaction. The public keys do not need to be preloaded or derived in advance, yet the server doesn't have the private keys that can spend the funds.
第2，用户可以创建一系列公钥，而不需要访问对应的私钥。
这样，HD钱包就能用在不安全的服务器上，或者用在接收环境中，为每个交易发布一个不同的公钥。
不需要预先放置或导出公钥，而且服务器也没有可用于来花费资金的私钥。

5.1.4种子和助记词

HD wallets are a very powerful mechanism for managing many keys and addresses. They are even more useful if they are combined with a standardized way of creating seeds from a sequence of English words that are easy to transcribe, export, and import across wallets. This is known as a *mnemonic* and the standard is defined by BIP-39. Today, most bitcoin wallets (as well as wallets for other cryptocurrencies) use this standard and can import and export seeds for backup and recovery using interoperable mnemonics.
HD钱包是管理许多密钥和地址的一种强大机制。
如果这样使用，甚至更有用：使用一种标准化的方法，从一系列英文单词中创建种子，这些英文单词很容易在钱包之间转录、导出和导入。
这称为助记词，BIP-39定义了这个标准。
现在，多数比特币钱包（以及其它加密货币的钱包）使用这个标准，并可以使用可互操作的助记词导入和导出种子，以进行备份和恢复。

Let’s look at this from a practical perspective. Which of the following seeds is easier to transcribe, record on paper, read without error, export, and import into another wallet?
我们从实际的角度来看看。
以下哪个种子更容易转录、记录在纸上、不会读错、导入导出到其它钱包中?

A seed for an deterministic wallet, in hex
确定性钱包的一个种子（使用16进制）：

```
0C1E24E5917779D297E14D45F14E1A1A
```

A seed for an deterministic wallet, from a 12-word mnemonic
确定性钱包的一个种子（使用12个单词）：

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

5.1.5钱包最佳实践

As bitcoin wallet technology has matured, certain common industry standards have emerged that make bitcoin wallets broadly interoperable, easy to use, secure, and flexible. These common standards are:

- Mnemonic code words, based on BIP-39
- HD wallets, based on BIP-32
- Multipurpose HD wallet structure, based on BIP-43
- Multicurrency and multiaccount wallets, based on BIP-44

由于比特币钱包技术已经成熟，出现了一些常用的业界标准，使得比特币钱包具备广泛互操作性，易于使用，安全和灵活。这些常用的标准是：

• 助记码	BIP-39
• HD钱包	BIP-32
• 多用途HD钱包结构	BIP-43
• 多币种和多帐户钱包	BIP-44

These standards may change or may become obsolete by future developments, but for now they form a set of interlocking technologies that have become the de facto wallet standard for bitcoin.
这些标准可能会变化，也可能被未来发展所替代，但目前，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。

The standards have been adopted by a broad range of software and hardware bitcoin wallets, making all these wallets interoperable. A user can export a mnemonic generated

on one of these wallets and import it in another wallet, recovering all transactions, keys, and addresses.
这些标准已被广泛的软件和硬件比特币钱包所采用，使所有这些钱包都可互操作。
用户可以导出在一个钱包中生成的助记符，将其导入另一个钱包，从而恢复所有交易、密钥和地址。

Some example of software wallets supporting these standards include (listed alphabetically) Breadwallet, Copay, Multibit HD, and Mycelium. Examples of hardware wallets supporting these standards include (listed alphabetically) Keepkey, Ledger, and Trezor.
支持这些标准的软件钱包有：Breadwallet、Copay、Multibit HD、Mycelium
支持这些标准的硬件钱包有：Keepkey、Ledger、Trezor

The following sections examine each of these technologies in detail.
以下小结将详细介绍每种技术。

Tip: If you are implementing a bitcoin wallet, it should be built as a HD wallet, with a seed encoded as mnemonic code for backup, following the BIP-32, BIP-39, BIP-43, and BIP-44 standards, as described in the following sections.
提示：如果你要实现一个比特币钱包，那么应该构建为HD钱包，把种子编码为助记词代码进行备份，遵循标准BIP-32/39/43/44。

5.1.6使用比特币钱包

In [user-stories] we introduced Gabriel, an enterprising young teenager in Rio de Janeiro, who is running a simple web store that sells bitcoin-branded t-shirts, coffee mugs, and stickers.
Gabriel是一个有进取心的少年，正在经营一家简单的网店，销售T恤、咖啡杯和贴纸。

Gabriel uses a Trezor bitcoin hardware wallet (A Trezor device: a bitcoin HD wallet in hardware) to securely manage his bitcoin. The Trezor is a simple USB device with two buttons that stores keys (in the form of an HD wallet) and signs transactions. Trezor wallets implement all the industry standards discussed in this chapter, so Gabriel is not reliant on any proprietary technology or single vendor solution.
Gabriel使用了一个Trezor比特币硬件钱包来安全地管理他的比特币。
Trezor是一个简单的USB设备，有两个按钮，用于存储密钥（以HD钱包的形式）和签署交易。
Trezor钱包实现了本章讨论的所有业界标准，因此Gabriel不依赖于任何专有技术或某个制造商方案。

Figure 4. A Trezor device: a bitcoin HD wallet in hardware
图4：一个Trezor设备：一个比特币HD硬件钱包

When Gabriel used the Trezor for the first time, the device generated a mnemonic and seed from a built-in hardware random number generator. During this initialization phase, the wallet displayed a numbered sequence of words, one by one, on the screen (see Trezor displaying one of the mnemonic words).
当Gabriel首次使用这个Trezor时，设备从内置的硬件随机数生成器生成一个助记词和种子。
在这个初始化阶段，钱包在屏幕上按顺序显示一些单词。

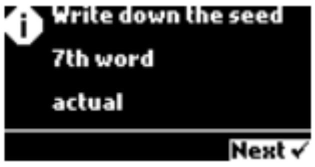
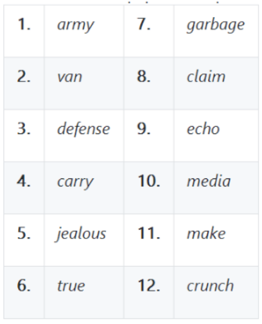

Write down the seed
7th word
actual
Next ✓

Figure 5. Trezor displaying one of the mnemonic words
图5：Trezor显示了一个助记词

By writing down this mnemonic, Gabriel created a backup (see Gabriel’s paper backup of the mnemonic) that can be used for recovery in the case of loss or damage to the Trezor device. This mnemonic can be used for recovery in a new Trezor or in any one of the many compatible software or hardware wallets. Note that the sequence of words is important, so mnemonic paper backups have numbered spaces for each word. Gabriel had to carefully record each word in the numbered space to preserve the correct sequence.
通过记下这个助记词，Gabriel就创建了一个备份，可以在Trezor设备丢失或损坏的情况下进行恢复。
这个助记词可用于在新的Trezor上恢复，或者在任何兼容软件和硬件钱包中恢复。
注意，单词的顺序很重要。Gabriel必须仔细记录每个单词的编号，以保证正确的顺序。

Table 1. Gabriel’s paper backup of the mnemonic
表1 Gabriel备份的助记词



1.	*army*	7.	*garbage*
2.	*van*	8.	*claim*
3.	*defense*	9.	*echo*
4.	*carry*	10.	*media*
5.	*jealous*	11.	*make*
6.	*true*	12.	*crunch*

Tip: A 12-word mnemonic is shown in Gabriel’s paper backup of the mnemonic, for simplicity. In fact, most hardware wallets generate a more secure 24-word mnemonic. The mnemonic is used in exactly the same way, regardless of length.
提示：为了简单，图1显示了12个单词的助记词。
实际上，多数硬件钱包生成更加安全的24个单词的助记词。
助记词的使用方法是一样的，不管长度如何。

For the first implementation of his web store, Gabriel uses a single bitcoin address, generated on his Trezor device. This single address is used by all customers for all orders. As we will see, this approach has some drawbacks and can be improved upon with an HD wallet.
为了第一次实现他的网店，Gabriel使用Trezor设备生成一个比特币地址。
所有客户的订单都使用这个地址。
我们将看到，这种方法有一些缺点，可以用HD钱包进行改进。

5.2钱包技术细节

Let’s now examine each of the important industry standards that are used by many bitcoin wallets in detail.
现在我们来详细看看每个重要的业界标准，它们被许多比特币钱包使用。

5.2.1助记词词汇（BIP-39）

Mnemonic code words are word sequences that represent (encode) a random number used as a seed to derive a deterministic wallet. The sequence of words is sufficient to re-create the seed and from there re-create the wallet and all the derived keys.
助记词是单词序列，它们表示（编码）一个随机数，用作一个种子来导出一个确定性钱包。
单词序列足以重新创建这个种子，从而重建钱包和所有导出密钥。

A wallet application that implements deterministic wallets with mnemonic words will show the user a sequence of 12 to 24 words when first creating a wallet. That sequence of words is the wallet backup and can be used to recover and re-create all the keys in the same or any compatible wallet application. Mnemonic words make it easier for users to back up wallets because they are easy to read and correctly transcribe, as compared to a random sequence of numbers.
用助记词实现确定性钱包的钱包应用程序在第一次创建一个钱包时，会给用户显示12-24个单词的序列。
这个单词序列就是钱包的备份，也用来在任何兼容钱包应用程序中恢复和重建所有密钥。
助记词使用户更容易备份钱包，因为相比随机数序列来说，它们更容易读和正确转录。

Tip: Mnemonic words are often confused with "brainwallets." They are not the same. The primary difference is that a brainwallet consists of words chosen by the user, whereas mnemonic words are created randomly by the wallet and presented to the user. This important difference makes mnemonic words much more secure, because humans are very poor sources of randomness.
提示：“助记词”经常与“脑钱包”相混淆。
他们不一样，主要区别是，脑钱包由用户选择的单词组成，而助记词是由钱包随机创建的，并呈现给用户。
这个重要区别使助记词更加安全，因为人的随机性比较差。

Mnemonic codes are defined in BIP-39 (see [appdxbitcoinimpproposals]). Note that BIP-39 is one implementation of a mnemonic code standard. There is a different standard, with a different set of words, used by the Electrum wallet and predating BIP-39. BIP-39 was proposed by the company behind the Trezor hardware wallet and is incompatible with Electrum’s implementation. However, BIP-39 has now achieved broad industry support across dozens of interoperable implementations and should be considered the de facto industry standard.
BIP-39定义了助记词代码。
注意，BIP-39是助记词代码标准的一个实现。还有一个不同的标准，使用一组不同的单词，被Electrum钱包使用，并且早于BIP-39。
Trezor硬件钱包背后的公司建议了BIP-39，它与Electrum实现并不兼容。
然而，BIP-39现在已经获得了广泛的业界支持，有数十个可互操作的实现，应被视为事实上的业界标准。

BIP-39 defines the creation of a mnemonic code and seed, which we describe here in nine steps. For clarity, the process is split into two parts: steps 1 through 6 are shown in Generating mnemonic words and steps 7 through 9 are shown in From mnemonic to seed.
BIP-39定义了助记词和种子的创建，我们这里用九个步骤来描述。
为了清晰，这个过程分为两部分：1-6步是生成助记词，7-9步是用助记词获得种子。

5.2.2生成助记词

Mnemonic words are generated automatically by the wallet using the standardized process defined in BIP-39. The wallet starts from a source of entropy, adds a checksum, and then maps the entropy to a word list:

1. Create a random sequence (entropy) of 128 to 256 bits.
2. Create a checksum of the random sequence by taking the first (entropy-length/32) bits of its SHA256 hash.
3. Add the checksum to the end of the random sequence.
4. Split the result into 11-bit length segments.
5. Map each 11-bit value to a word from the predefined dictionary of 2048 words.
6. The mnemonic code is the sequence of words.

助记词是由钱包自动生成的，使用的是BIP-39中定义的标准过程。
钱包从一个随机源开始，增加一个校验和，然后将这个随机数映射为一个单词列表：
1、创建一个随机序列（128到256位）
2、创建这个随机序列的校验和：取随机序列SHA25哈希的前32位
3、把校验和加到随机序列的末尾
4、将结果分成11比特长的分段
5、把每个11比特值映射为一个单词，单词来自预定义字典（2048个单词）
6、助记词就是这个单词序列

Generating entropy and encoding as mnemonic words shows how entropy is used to generate mnemonic words.
图6显示了如何用随机数生成助记词。

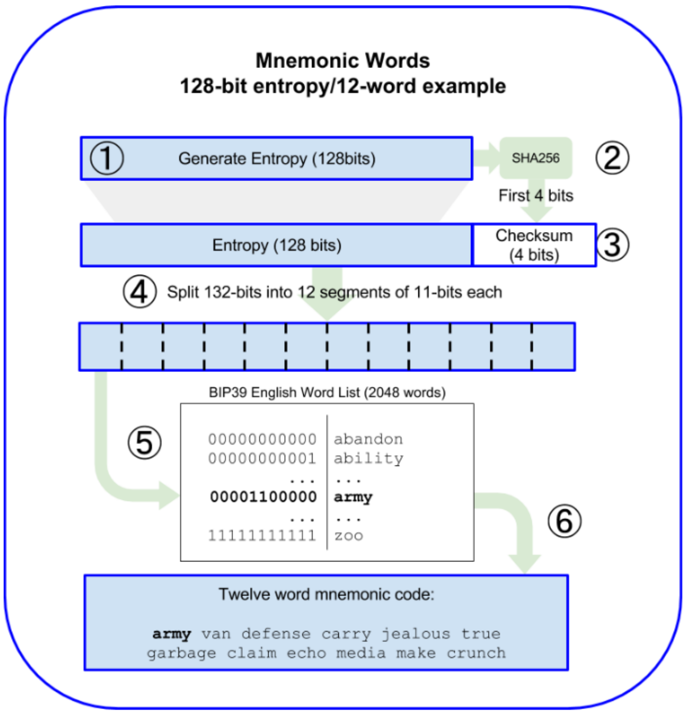
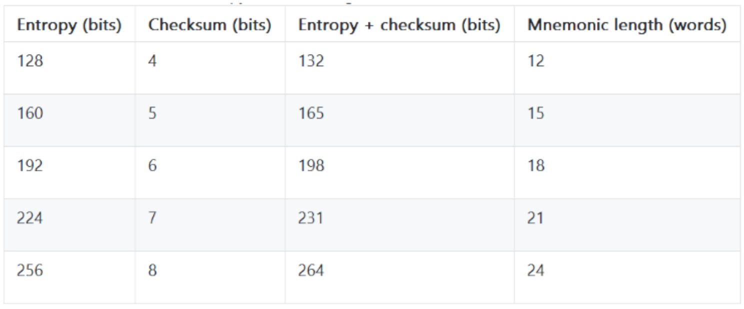

Mnemonic Words
128-bit entropy/12-word example
① Generate Entropy (128bits)
SHA256 ②
First 4 bits
Entropy (128 bits)
Checksum (4 bits) ③
④ Split 132-bits into 12 segments of 11-bits each
BIP39 English Word List (2048 words)
⑤
00000000000 abandon
00000000001 ability
... ...
00001100000 army
... ...
11111111111 zoo
⑥
Twelve word mnemonic code:
army van defense carry jealous true
garbage claim echo media make crunch

Figure 6. Generating entropy and encoding as mnemonic words
图6：生成随机数，编码为助记词

Mnemonic codes: entropy and word length shows the relationship between the size of the entropy data and the length of mnemonic codes in words.
表2显示了随机数据长度与助记词长度之间的关系

Table 2. Mnemonic codes: entropy and word length
表2：助记词：随机数和单词长度



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用助记词得到种子

The mnemonic words represent entropy with a length of 128 to 256 bits. The entropy is then used to derive a longer (512-bit) seed through the use of the key-stretching function PBKDF2. The seed produced is then used to build a deterministic wallet and derive its keys.
助记词表示128~256位的随机数。
然后用这个随机数调出一个更长的种子（512位），使用的是密钥延伸函数PBKDF2。
生成的种子用于构建一个确定性钱包和导出密钥。

The key-stretching function takes two parameters: the mnemonic and a *salt*. The purpose of a salt in a key-stretching function is to make it difficult to build a lookup table enabling a brute-force attack. In the BIP-39 standard, the salt has another purpose—it allows the introduction of a passphrase that serves as an additional security factor protecting the seed, as we will describe in more detail in Optional passphrase in BIP-39.
密钥延伸函数有两个参数：助记词、盐（salt）。
盐的目的是增加这个的困难性：构造一个查找表来实现暴力攻击。
在BIP-39中，盐有另一个目的：它允许引入密码短语（passphrase），作为保护种子的附加安全因素。

The process described in steps 7 through 9 continues from the process described previously in Generating mnemonic words:

7. The first parameter to the PBKDF2 key-stretching function is the *mnemonic* produced from step 6.
8. The second parameter to the PBKDF2 key-stretching function is a *salt*. The salt is composed of the string constant "mnemonic" concatenated with an optional user-supplied passphrase string.
9. PBKDF2 stretches the mnemonic and salt parameters using 2048 rounds of hashing with the HMAC-SHA512 algorithm, producing a 512-bit value as its final output. That 512-bit value is the seed.

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：步骤6生成的助记词
8. PBKDF2密钥延伸函数的第二个参数是：盐。
   盐的组成：字符串常量“mnemonic”，后面跟一个可选的用户提供的密码字符串。
9. PBKDF2延伸了助记词和盐参数：使用了2048次哈希（HMAC-SHA512算法），生成一个512位的值。 这个值就是种子。

From mnemonic to seed shows how a mnemonic is used to generate a seed.
图7显示了如何使用助记词生成一个种子

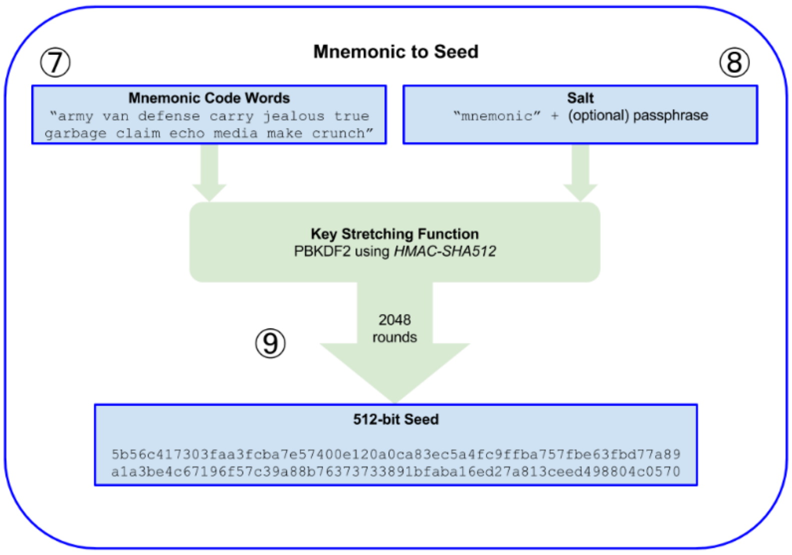

Mnemonic to Seed
⑦
⑧
Mnemonic Code Words
"army van defense carry jealous true garbage claim echo media make crunch"
Salt
"mnemonic" + (optional) passphrase
Key Stretching Function
PBKDF2 using HMAC-SHA512
⑨
2048 rounds
512-bit Seed
5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fbd77a89
a1a3be4c67196f57c39a88b76373733891bfaba16ed27a813ceed498804c0570

Figure 7. From mnemonic to seed

`Tip`: The key-stretching function, with its 2048 rounds of hashing, is a very effective protection against brute-force attacks against the mnemonic or the passphrase. It makes it extremely costly (in computation) to try more than a few thousand passphrase and mnemonic combinations, while the number of possible derived seeds is vast (2^{512}).
提示：密钥延伸函数，使用了2048次哈希，能够非常有效地防止针对助记词或密码短语的暴力攻击。它使攻击计算非常昂贵，要尝试超过几千个密码短语和助记词的组合，可能产生的种子数量非常大（2^{512}）。

Tables #mnemonic_128_no_pass, #mnemonic_128_w_pass, and #mnemonic_256_no_pass show some examples of mnemonic codes and the seeds they produce (without any passphrase).
表3~5显示了一些例子，助记词和它们生成的种子（没有密码短语）

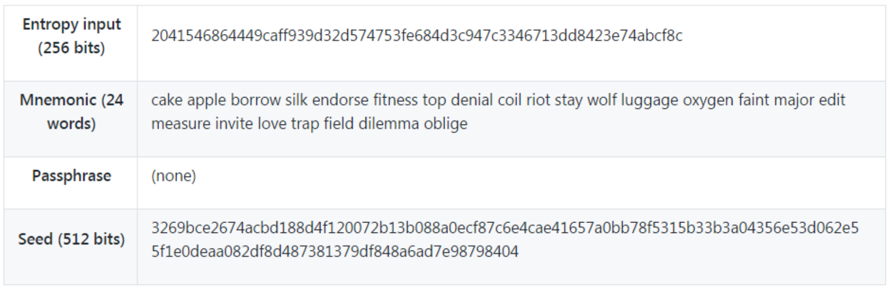
Table 3. 128-bit entropy mnemonic code, no passphrase, resulting seed
表3：128位随机助记词，没有密码短语，生成的种子

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)	5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fbd77a89a1a3be4c67196f57c39 a88b76373733891bfaba16ed27a813ceed498804c0570

Table 4. 128-bit entropy mnemonic code, with passphrase, resulting seed
表4：128位随机助记词，有密码短语，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	SuperDuperSecret
Seed (512 bits)	3b5df16df2157104cfdd22830162a5e170c0161653e3afe6c88defeefb0818c793dbb28ab3ab091897d0 715861dc8a18358f80b79d49acf64142ae57037d1d54

Table 5. 256-bit entropy mnemonic code, no passphrase, resulting seed
表5：256位随机助记词，没有密码短语，生成的种子



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)	3269bce2674acbd188d4f120072b13b088a0ecf87c6e4cae41657a0bb78f5315b33b3a04356e53d062e5 5f1e0deaa082df8d487381379df848a6ad7e98798404

5.2.4 BIP-39中的可选密码短语

The BIP-39 standard allows the use of an optional passphrase in the derivation of the seed. If no passphrase is used, the mnemonic is stretched with a salt consisting of the constant string "mnemonic", producing a specific 512-bit seed from any given mnemonic. If a passphrase is used, the stretching function produces a *different* seed from that same mnemonic. In fact, given a single mnemonic, every possible passphrase leads to a different seed. Essentially, there is no "wrong" passphrase. All passphrases are valid and they all lead to different seeds, forming a vast set of possible uninitialized wallets. The set of possible wallets is so large (2^{512}) that there is no practical possibility of brute-forcing or accidentally guessing one that is in use.
BIP-39允许在导出种子时使用一个可选的密码短语。
如果不使用密码短语，使用的盐只有“mnenonic”，这就从任何给定的助记词产生一个特定的种子。
如果使用密码短语，密钥延伸函数使用同样的助记词会产生不同的种子。
事实上，给定一个助记词，每一可能的密码短语都会生成一个不同的种子。
本质上，没有“错误”的密码短语。所有密码短语都是有效的，它们都会生成不同的种子，形成一大批可能未初始化的钱包。这批钱包的数量非常大（2^{512}），使得暴力破解或偶然猜测实际上不可能。

Tip: There are no "wrong" passphrases in BIP-39. Every passphrase leads to some wallet, which unless previously used will be empty.
提示：BIP-39中没有“错误的”密码短语。
每个密码短语都会生成一个钱包，除非以前用过，否则就是空钱包。

The optional passphrase creates two important features:
- A second factor (something memorized) that makes a mnemonic useless on its own, protecting mnemonic backups from compromise by a thief.
- A form of plausible deniability or "duress wallet," where a chosen passphrase leads to a wallet with a small amount of funds used to distract an attacker from the "real" wallet that contains the majority of funds.

这个可选密码短语有两个重要功能：
- 密码短语成为第二个因素（记忆），使得助记词不能单独使用，从而防止助记词备份被盗取利用。
- 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，使其不再关注有大额资金的“实际”钱包。

However, it is important to note that the use of a passphrase also introduces the risk of loss:
- If the wallet owner is incapacitated or dead and no one else knows the passphrase, the seed is useless and all the funds stored in the wallet are lost forever.
- Conversely, if the owner backs up the passphrase in the same place as the seed, it defeats the purpose of a second factor.

但是，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，这个种子就是无用的，所有存储在钱包中的资金都将永远丢失。（ddk：应该是助记词吧？）
- 如果将密码短语和种子备份在同一地方，则起不到作为第二个因素的目的。

ddk问题：种子是否写错了?

While passphrases are very useful, they should only be used in combination with a carefully planned process for backup and recovery, considering the possibility of surviving the owner and allowing his or her family to recover the cryptocurrency estate.
虽然密码短语非常有用，但它们应该与仔细规划的备份和恢复过程结合使用，考虑拥有者生存的可能性，允许其家人恢复加密货币资产。

5.2.5使用助记词

BIP-39 is implemented as a library in many different programming languages:
python-mnemonic

The reference implementation of the standard by the SatoshiLabs team that proposed BIP-39, in Python

bitcoinjs/bip39

An implementation of BIP-39, as part of the popular bitcoinJS framework, in JavaScript

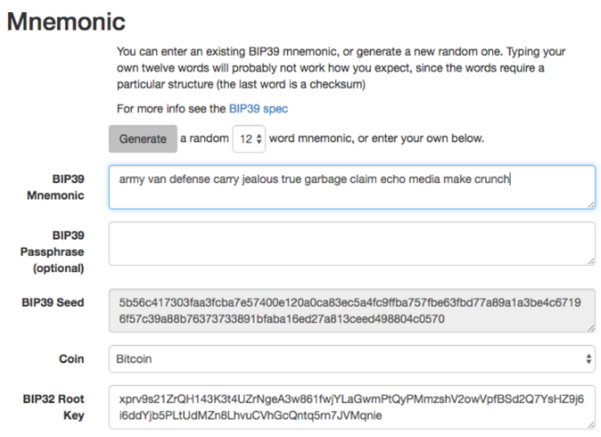
libbitcoin/mnemonic

An implementation of BIP-39, as part of the popular Libbitcoin framework, in C++

在许多编程语言中，BIP-39被实现为一个库：

python-mnemonic	SatoshiLabs团队的标准参考实现，他们提出了BIP-39，Python语言
bitcoinjs/bip39	它是流行的bitcoinJS框架的一部分，JavaScript语言
libbitcoin/mnemonic	它是流行的Libbitcoin框架的一部分，C ++语言

There is also a BIP-39 generator implemented in a standalone webpage, which is extremely useful for testing and experimentation. A BIP-39 generator as a standalone web page shows a standalone web page that generates mnemonics, seeds, and extended private keys.
还有一个BIP-39生成器，是在一个单独的网页中实现的，对于测试和实验非常有用。
图8显示了一个单独的网页，它可以生成助记词、种子、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: 5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fbd77a89a1a3be4c6719 6f57c39a88b76373733891bfaba16ed27a813ceed498804c0570

Coin: Bitcoin

BIP32 Root Key: xprv9s21ZrQH143K3t4UZrNgeA3w861fwjYLaGwmPtQyPMmzshV2owVpfBSd2Q7YsHZ9j6 i6ddYjb5PLtUdMZn8LhvuCVhGcQntq5rn7JVMqnie

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

The page (https://iancoleman.github.io/bip39/) can be used offline in a browser, or accessed online.
可以在浏览器中离线使用这个网页，或者在线访问它。
https://iancoleman.io/bip39/

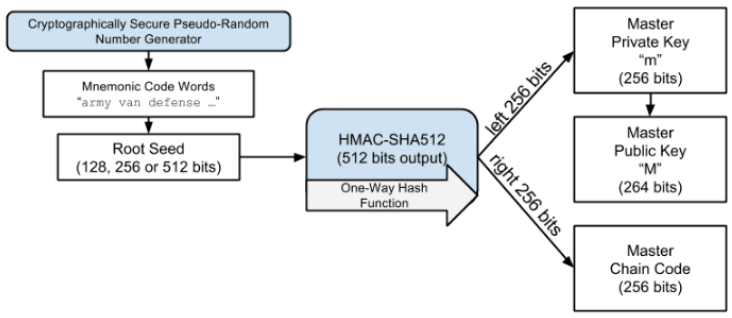
5.3用种子创建一个HD钱包

HD wallets are created from a single *root seed*, which is a 128-, 256-, or 512-bit random number. Most commonly, this seed is generated from a *mnemonic* as detailed in the previous section.
HD钱包用一个根种子（root seed）创建的，根种子是128/256/512位的随机数。
最常见的，这个种子是用前一章节介绍的助记词生成。

Every key in the HD wallet is deterministically derived from this root seed, which makes it possible to re-create the entire HD wallet from that seed in any compatible HD wallet. This makes it easy to back up, restore, export, and import HD wallets containing thousands or even millions of keys by simply transferring only the mnemonic that the root seed is derived from.
HD钱包的所有密钥都是用这个根种子导出的，可以在任何兼容的HD钱包中用这个种子重建HD钱包。
这样，对于包含大量密钥的HD钱包来说，很容易备份、恢复、导出、导读，只需要转移助记词，用它导出根种子。

The process of creating the master keys and master chain code for an HD wallet is shown in Creating master keys and chain code from a root seed.
图9显示了这个过程：为一个HD钱包创建主密钥和主链码

Figure 9. Creating master keys and chain code from a root seed
图9：用一个根种子创建主密钥和链码



The root seed is input into the HMAC-SHA512 algorithm and the resulting hash is used to create a *master private key* (m) and a *master chain code* (c).
根种子是HMAC-SHA512算法的输入，产生的哈希用于创建"主私钥(m) "和"主链码（c）"。

The master private key (m) then generates a corresponding master public key (M) using the normal elliptic curve multiplication process m * G that we saw in [pubkey].
然后，主私钥（m）生成一个对应的主公钥（M），方法是：使用椭圆曲线乘法m*G。

The chain code (c) is used to introduce entropy in the function that creates child keys from parent keys, as we will see in the next section.
链码（c）用于向函数引入随机数，这个函数用父密钥创建子密钥。

5.3.1导出子私密

HD wallets use a *child key derivation* (CKD) function to derive child keys from parent keys.
HD钱包使用一个CKD函数（子密钥导出），它用父密钥导出子密钥。

The child key derivation functions are based on a one-way hash function that combines:

- A parent private or public key (ECDSA uncompressed key)
- A seed called a chain code (256 bits)
- An index number (32 bits)

CKD函数是基于一个单项哈希函数，这个函数综合了：

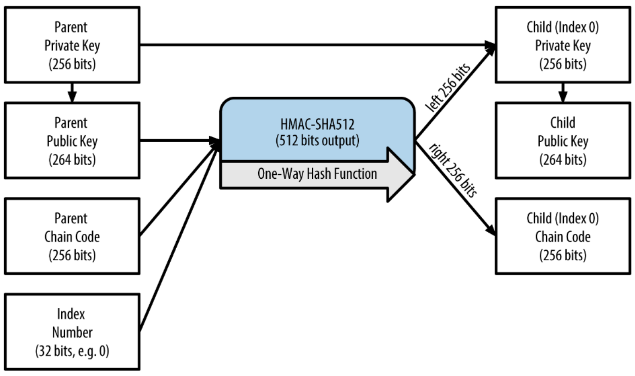
- 一个父私钥或父公钥 （ECDSA未压缩密钥）
- 一个种子（链码） （256位）
- 一个索引号 （32位）

The chain code is used to introduce deterministic random data to the process, so that knowing the index and a child key is not sufficient to derive other child keys. Knowing a child key does not make it possible to find its siblings, unless you also have the chain code. The initial chain code seed (at the root of the tree) is made from the seed, while subsequent child chain codes are derived from each parent chain code.
“链码”用来给这个过程引入确定性随机数据，这样，知道索引和子密钥，也不足以导出其它子密钥。
知道一个子密钥，并不能找到它的兄弟密钥，除非你也知道链码。
最初的链码种子（在树的根部）是来自种子，随后的子链码来自从各自的父链码。

These three items (parent key, chain code, and index) are combined and hashed to generate children keys, as follows.
这三个项目（父密钥、链码、索引）结合在一起，计算哈希，从而生成子密钥。

The parent public key, chain code, and the index number are combined and hashed with the HMAC-SHA512 algorithm to produce a 512-bit hash. This 512-bit hash is split into two 256-bit halves. The right-half 256 bits of the hash output become the chain code for the child. The left-half 256 bits of the hash are added to the parent private key to produce the child private key. In Extending a parent private key to create a child private key, we see this illustrated with the index set to 0 to produce the "zero" (first by index) child of the parent.
父公钥、链码、索引号结合在一起，用HMAC-SHA512算法计算哈希，生成一个512位的哈希。
这个512位的哈希分成两个256位数字。
右半部分成为子的链码。左半部分被加到父私钥上，生成子私钥。
在图10中，显示了把索引集设为0，生成第0个子。

Figure 10. Extending a parent private key to create a child private key
图10：扩展一个父私钥，生一个子私钥


Parent Private Key (256 bits)
Parent Public Key (264 bits)
Parent Chain Code (256 bits)
Index Number (32 bits, e.g. 0)
HMAC-SHA512 (512 bits output)
One-Way Hash Function
left 256 bits
right 256 bits
Child (Index 0) Private Key (256 bits)
Child Public Key (264 bits)
Child (Index 0) Chain Code (256 bits)

Changing the index allows us to extend the parent and create the other children in the sequence, e.g., Child 0, Child 1, Child 2, etc. Each parent key can have 2,147,483,647 (2^{31}) children (2^{31} is half of the entire 2^{32} range available because the other half is reserved for a special type of derivation we will talk about later in this chapter).
改变索引可以扩展父，创建序列中的其它子，例如0/1/2等等。
每个父密钥可以有2^{31} 个子。
2^{31}是2^{32}的一半，因为另一半是为特定类型的导出而保留的，我们将在本章稍后讨论。

Repeating the process one level down the tree, each child can in turn become a parent and create its own children, in an infinite number of generations.
向树下一层重复这个过程，每个子都可以成为一个父，创建它自己的子，可以创建无限代。

5.3.2使用导出的子密钥

Child private keys are indistinguishable from nondeterministic (random) keys. Because the derivation function is a one-way function, the child key cannot be used to find the parent key. The child key also cannot be used to find any siblings. If you have the n_{th} child, you cannot find its siblings, such as the n–1 child or the n+1 child, or any other children that are part of the sequence. Only the parent key and chain code can derive all the children. Without the child chain code, the child key cannot be used to derive any grandchildren either. You need both the child private key and the child chain code to start a new branch and derive grandchildren.
没法区分子私钥与非确定性（随机）密钥。
因为导出函数是单向函数，所以子密钥不能用于找到父密钥，也不能用来找到兄弟密钥。
如果你有第n个子密钥，你无法找到第n-1或n+1个子密钥。
只有父密钥和链码才能导出所有的子。没有子链码的话，子密钥也不能用来导出任何孙密钥。
你需要同时有子密钥和子链码，才能创建一个新的分支来导出孙密钥。

So what can the child private key be used for on its own? It can be used to make a public key and a bitcoin address. Then, it can be used to sign transactions to spend anything paid to that address.
那么，子私钥能用来干什么?
它可以用来生成公钥和比特币地址。
然后，它可用于签署交易，以花费向这个地址支付的比特币。

A child private key, the corresponding public key, and the bitcoin address are all indistinguishable from keys and addresses created randomly. The fact that they are part of a sequence is not visible outside of the HD wallet function that created them. Once created, they operate exactly as "normal" keys.
提示：子私钥、对应的公钥、比特币地址与随机创建的密钥和地址是无法区分的。
在HD钱包之外，看不出它们是一个系列。
一旦被创建出来，它们就和“普通”密钥一样使用。

5.3.3扩展密钥

As we saw earlier, the key derivation function can be used to create children at any level of the tree, based on the three inputs: a key, a chain code, and the index of the desired child. The two essential ingredients are the key and chain code, and combined these are called an *extended key*. The term "extended key" could also be thought of as "extensible key" because such a key can be used to derive children.
正如我们之前看到的，可以在树的任何层级上使用密钥导出函数来创建子。
它是基于三个输入：一个密钥，一个链码，索引。
两个基本的成分是密钥和链码，合称为扩展密钥。
“扩展”也被认为是“可扩展的密钥”，因为这个密钥可以用来导出子密钥。

Extended keys are stored and represented simply as the concatenation of the 256-bit key and 256-bit chain code into a 512-bit sequence. There are two types of extended keys. An extended private key is the combination of a private key and chain code and can be used to derive child private keys (and from them, child public keys). An extended public key is a public key and chain code, which can be used to create child public keys (*public only*), as described in [public_key_derivation].
扩展密钥可简单地储存和表示为：256位密钥和256位链码，合并为512位。
有两种类型的扩展密钥：

- 扩展私钥：是私钥和链码的结合。可用于导出子私钥和子公钥。
- 扩展公钥：是公钥和链码的结合。可用于创建子公钥。

Think of an extended key as the root of a branch in the tree structure of the HD wallet. With the root of the branch, you can derive the rest of the branch. The extended private key can create a complete branch, whereas the extended public key can *only* create a branch of public keys.
把扩展密钥想象成HD钱包的树结构中的一个分支的根。
有了这个分支的根，你可以导出这个分支的其它部分。
扩展私钥可以创建一个完整的分支，而扩展公钥只能够创造一个公钥分支。

`Tip`：An extended key consists of a private or public key and chain code. An extended key can create children, generating its own branch in the tree structure. Sharing an extended key gives access to the entire branch.
提示：一个扩展密钥包括一个私钥（或公钥）和一个链码。
扩展密钥可以创建子，在树结构中生成它自己的分支。
分享扩展密钥就可以访问这个分支。

Extended keys are encoded using Base58Check, to easily export and import between different BIP-32–compatible wallets. The Base58Check coding for extended keys uses a special version number that results in the prefix "xprv" and "xpub" when encoded in Base58 characters to make them easily recognizable. Because the extended key is 512 or 513 bits, it is also much longer than other Base58Check-encoded strings we have seen previously.
使用Base58Check来编码扩展密钥，这样就容易在不同的BIP-32兼容钱包之间导入和导出。
扩展密钥的 Base58Check编码使用了特殊的版本号，从而在Base58编码中生成了前缀xprv和xpub。
这样可以很容易识别它们。
因为扩展密钥是512或513位，所以它比我们之前看到的Base58Check编码字符串更长一些。

Here’s an example of an extended *private* key, encoded in Base58Check:
下面是一个扩展私钥的例子，用的是Base58Check编码。

```
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i6GoNMKUga5biW6Hx4tws2si
x3b9c
```

Here’s the corresponding extended *public* key, encoded in Base58Check:
下面是对应的公钥，也是Base58Check编码。

```
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVKZBPLrtJunSDMstweyLXhRgPxdp14sk9
tJPW9
```

5.3.4子公钥导出

As mentioned previously, a very useful characteristic of HD wallets is the ability to derive public child keys from public parent keys, *without* having the private keys. This gives us two ways to derive a child public key: either from the child private key, or directly from the parent public key.
HD钱包的一个很有用的特点是：没有私钥，也可以从父公钥导出出子公钥。
这就有了两种导出子公钥的方法：用子私钥导出子公钥，或用父公钥导出子公钥。

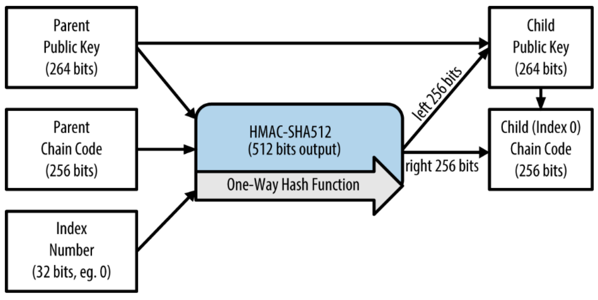
An extended public key can be used, therefore, to derive all of the *public* keys (and only the public keys) in that branch of the HD wallet structure.
因此，扩展公钥可用于导出HD钱包结构的那个分支中的所有公钥（只有公钥）。

This shortcut can be used to create very secure public key–only deployments where a server or application has a copy of an extended public key and no private keys whatsoever. That kind of deployment can produce an infinite number of public keys and bitcoin addresses, but cannot spend any of the money sent to those addresses. Meanwhile, on another, more secure server, the extended private key can derive all the corresponding private keys to sign transactions and spend the money.
这个方法可以用来创建非常安全的公钥部署：服务器或应用程序有一个扩展公钥，而没有私钥。
这种部署可以生成无限多的的公钥和比特币地址，但不能花费发给这些地址的比特币。
同时，在另一种更安全的服务器上，扩展私钥可以导出所有对应的私钥，来签署交易和花费比特币。

One common application of this solution is to install an extended public key on a web server that serves an ecommerce application. The web server can use the public key derivation function to create a new bitcoin address for every transaction (e.g., for a customer shopping cart). The web server will not have any private keys that would be vulnerable to theft. Without HD wallets, the only way to do this is to generate thousands of bitcoin addresses on a separate secure server and then preload them on the ecommerce server. That approach is cumbersome and requires constant maintenance to ensure that the ecommerce server doesn't "run out" of keys.
这个方案的一个常见应用是：在一个web服务器上安装一个扩展公钥，这个服务器作为电商应用。
这个web服务器可以使用公钥导出函数为每个交易创建一个新的比特币地址。
这个web服务器没有私钥，从而避免了被盗。
如果没有HD钱包，实现这个的唯一方法是：在一个隔离的安全的服务器上生成很多比特币地址，然后把它们预先放在电商服务器上。
这个方法非常笨拙，需要持续的维护来确保电商服务器没有用完公钥。

Another common application of this solution is for cold-storage or hardware wallets. In that scenario, the extended private key can be stored on a paper wallet or hardware device (such as a Trezor hardware wallet), while the extended public key can be kept online. The user can create "receive" addresses at will, while the private keys are safely stored offline. To spend the funds, the user can use the extended private key on an offline signing bitcoin client or sign transactions on the hardware wallet device (e.g., Trezor). Extending a parent public key to create a child public key illustrates the mechanism for extending a parent public key to derive child public keys.
这个方案的另一个常见应用是冷藏或硬件钱包。
在这种情况下，扩展私钥可以被储存在纸质钱包或硬件设备（如Trezor硬件钱包），同时，扩展公钥可以在线保存。
用户可以根据需要创造接收地址，而私钥被安全地离线保存。
为了花费资金，用户可在“离线签名比特币客户端”使用扩展私钥，或在“硬件钱包设备”上签署交易。

Figure 11. Extending a parent public key to create a child public key
图11：扩展一个父公钥来创建一个子公钥


Parent Public Key (264 bits)
Parent Chain Code (256 bits)
Index Number (32 bits, eg. 0)
HMAC-SHA512 (512 bits output)
One-Way Hash Function
left 256 bits
right 256 bits
Child Public Key (264 bits)
Child (Index 0) Chain Code (256 bits)

5.3.5在网店上使用扩展公钥

Let's see how HD wallets are used by continuing our story with Gabriel's web store.
继续Gabriel网店的故事，我们看看Gabriel如何使用HD钱包。

Gabriel first set up his web store as a hobby, based on a simple hosted Wordpress page. His store was quite basic with only a few pages and an order form with a single bitcoin address.
Gabriel首先基于一个简单的托管Wordpress页面建立它的网店。
网店非常简单，只有几个页面和一个订单表格，有一个比特币地址。

Gabriel used the first bitcoin address generated by his Trezor device as the main bitcoin address for his store. This way, all incoming payments would be paid to an address controlled by his Trezor hardware wallet.

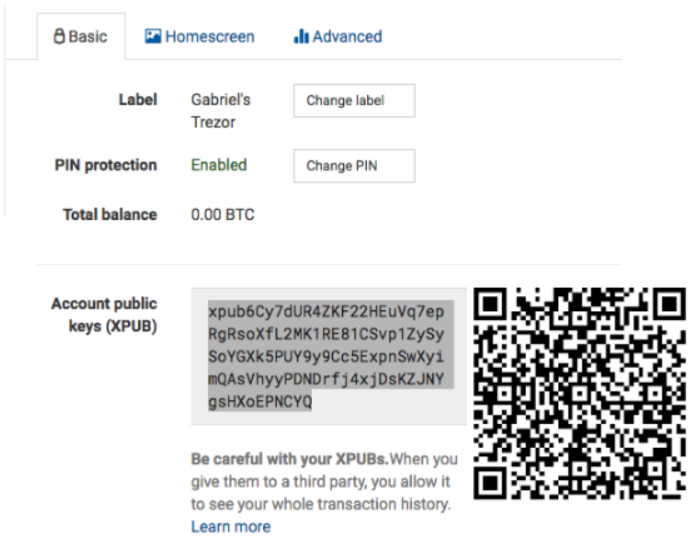
Gabriel使用Trezor设备生成的第一个比特币地址，作为网店的主比特币地址。
这样，所有收到的付款都将支付给他的Trezor硬件钱包控制的一个地址。

Customers would submit an order using the form and send payment to Gabriel’s published bitcoin address, triggering an email with the order details for Gabriel to process. With just a few orders each week, this system worked well enough.
客户可以使用表格提交订单，向Gabriel发布的比特币地址发送付款，触发一封电子邮件，其中包含订单详细信息。每周只有几个订单，这个系统运行得很好。

However, the little web store became quite successful and attracted many orders from the local community. Soon, Gabriel was overwhelmed. With all the orders paying the same address, it became difficult to correctly match orders and transactions, especially when multiple orders for the same amount came in close together.
这个小网店变得相当成功，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到相同的地址，因此很难正确匹配订单和交易，特别是当多个订单同时来，而且金额相同时。

Gabriel’s HD wallet offers a much better solution through the ability to derive public child keys without knowing the private keys. Gabriel can load an extended public key (xpub) on his website, which can be used to derive a unique address for every customer order. Gabriel can spend the funds from his Trezor, but the xpub loaded on the website can only generate addresses and receive funds. This feature of HD wallets is a great security feature. Gabriel’s website does not contain any private keys and therefore does not need high levels of security.
Gabriel的HD钱包提供了一个更好的方案，在不知道私钥的情况下导出子密钥。
Gabriel可以在他的网站上上传一个扩展公钥（xpub），可用于为每个订单导出一个唯一的地址。
Gabriel可以从Trezor花费资金，但网站上的xpub只能生成地址和并接受资金。
HD钱包的这个功能是一个非常安全的功能。
Gabriel的网站不包含任何私钥，因此不需要高级别的安全。

To export the xpub, Gabriel uses the web-based software in conjunction with the Trezor hardware wallet. The Trezor device must be plugged in for the public keys to be exported. Note that hardware wallets will never export private keys—those always remain on the device. Exporting an xpub from a Trezor hardware wallet shows the web interface Gabriel uses to export the xpub.
为了导出xpub，Gabriel将基于web的软件与Trezor硬件钱包。
为了导出公钥，必须插入Trezor设备。
注意，硬件钱包不会导出私钥，私钥始终在这个设备中。
图12显示了Gabriel导出xpub所用的web界面。



Label: Gabriel's Trezor

PIN protection: Enabled

Total balance: 0.00 BTC

Account public keys (XPUB):
```
xpub6Cy7dUR4ZKF22HEuVq7ep
RgRsoXfL2MK1RE81CSvp1ZySy
SoYGXk5PUY9y9Cc5ExpnSwXyi
mQAsVhyyPDNDrfj4xjDsKZJNY
gsHXoEPNCYQ
```
Be careful with your XPUBs. When you give them to a third party, you allow it to see your whole transaction history. Learn more

Figure 12. Exporting an xpub from a Trezor hardware wallet

图12：从Trezor硬件钱包导出一个xpub

Gabriel copies the xpub to his web store’s bitcoin shop software. He uses *Mycelium Gear*, which is an open source web-store plugin for a variety of web hosting and content platforms. Mycelium Gear uses the xpub to generate a unique address for every purchase.
Gabriel把这个xpub拷贝到网店的比特币购物软件中。
他使用Mycelium Gear，这是一个开源web网店插件，用于各种网络托管和内容平台。
Mycelium Gear使用这个xpub为每各订单生成一个唯一的地址。

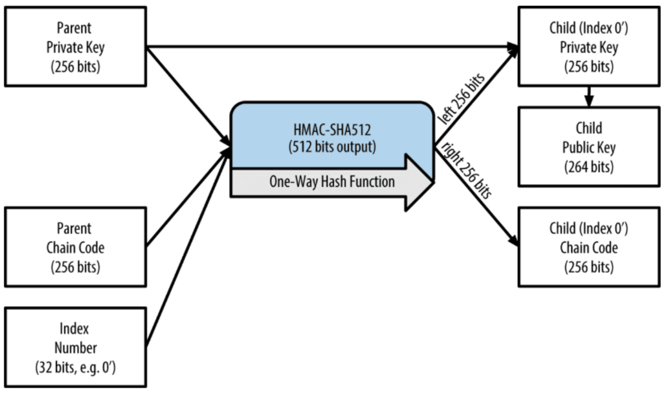
5.3.6强化子密钥导出

The ability to derive a branch of public keys from an xpub is very useful, but it comes with a potential risk. Access to an xpub does not give access to child private keys. However, because the xpub contains the chain code, if a child private key is known, or somehow leaked, it can be used with the chain code to derive all the other child private keys. A single leaked child private key, together with a parent chain code, reveals all the private keys of all the children. Worse, the child private key together with a parent chain code can be used to deduce the parent private key.
用xpub导出一个分支的公钥，这个能力很有用，但也有潜在的风险。
访问xpub不会得到子私钥。但是，因为xpub包含链码，如果知道了一个子私钥，链码和这个子私钥就可以导出其它子私钥。
泄露一个私钥和一个父链码，可以暴露所有的子密钥。
更糟糕的是，子私钥与父链码可以推导出父私钥。

To counter this risk, HD wallets use an alternative derivation function called *hardened derivation*, which "breaks" the relationship between parent public key and child chain code. The hardened derivation function uses the parent private key to derive the child chain code, instead of the parent public key. This creates a "firewall" in the parent/child sequence, with a chain code that cannot be used to compromise a parent or sibling private key. The hardened derivation function looks almost identical to the normal child private key derivation, except that the parent private key is used as input to the hash function, instead of the parent public key, as shown in the diagram in Hardened derivation of a child key; omits the parent public key.
为了应对这种风险，HD钱包使用了另一个导出函数（强化导出），它打破了父公钥和子链码之间的关系。

这个强化导出函数使用父私钥来导出子链码，而不是用父公钥。
这就在父子序列中创造了一道“防火墙”，有一个链码，但并不能得到父私钥或兄弟私钥。
这个强化导出函数看起来与一般的子私钥导出几乎相同，不同的是，父私钥用作这个哈希函数的输入，而不是用父公钥。如图13所示。

Figure 13. Hardened derivation of a child key; omits the parent public key
图13：子密钥的强化导出，忽略了父公钥



When the hardened private derivation function is used, the resulting child private key and chain code are completely different from what would result from the normal derivation function. The resulting "branch" of keys can be used to produce extended public keys that are not vulnerable, because the chain code they contain cannot be exploited to reveal any private keys. Hardened derivation is therefore used to create a "gap" in the tree above the level where extended public keys are used.
当使用强化私钥导出函数时，得到的子私钥和链码，与使用一般导出函数所得到的完全不同。
得到的密钥“分支”可用来生成扩展公钥，这些扩展公钥不易被攻击，因为它们所含的链码不能被用来暴露任何私钥。
因此，强化导出被用来在使用扩展公钥的层级上创建一个间隔。

In simple terms, if you want to use the convenience of an xpub to derive branches of public keys, without exposing yourself to the risk of a leaked chain code, you should derive it from a hardened parent, rather than a normal parent. As a best practice, the level-1 children of the master keys are always derived through the hardened derivation, to prevent compromise of the master keys.
简单来说，如果你想使用xpub来导出公钥分支，又不想有泄露链码的风险，你应该从一个强化父公钥来导出，而不是从一个普通父公钥导出。
最好的方法是，主密钥的第一级子密钥都是通过强化导出的，以防止泄露主密钥。

5.3.7普通导出和强化导出的索引号

The index number used in the derivation function is a 32-bit integer. To easily distinguish between keys derived through the normal derivation function versus keys derived through hardened derivation, this index number is split into two ranges. Index numbers between 0 and 2^{31}–1 (0x0 to 0x7FFFFFFF) are used *only* for normal derivation. Index numbers between 2^{31} and 2^{32}–1 (0x80000000 to 0xFFFFFFFF) are used *only* for hardened derivation. Therefore, if the index number is less than 2^{31}, the child is normal, whereas if the index number is equal or above 2^{31}, the child is hardened.
导出函数中使用的索引号码是32位整数。
为了区分普通导出函数和强化导出函数生成密钥，这个索引号被分为两个范围。
索引号在0和2^{31}–1 之间，是普通导出。
索引号在2^{31}和2^{32}–1之间，是强化导出。
因此，如果索引号小于2^{31}，子密钥是普通的，否则强化的。

To make the index number easier to read and display, the index number for hardened children is displayed starting from zero, but with a prime symbol. The first normal child key is therefore displayed as 0, whereas the first hardened child (index 0x80000000) is displayed as 0&#x27;. In sequence then, the second hardened key would have index 0x80000001 and would be displayed as 1&#x27;, and so on. When you see an HD wallet index i&#x27;, that means 2^{31}+i.
为了让索引号更容易被读和显示，强化子密钥的索引号显示的字符串以0开始，但有一个小撇号。
因此，第一个普通子密钥显示为0，第一个强化子密钥（索引号0x80000000）显示为0'。
第二个强化子密钥索引号是0x80000001，显示为1'，以此类推。
当你看到HD钱包索引号i'，意思是2^{31}+i。

5.3.8 HD钱包密钥标识（路径）

Keys in an HD wallet are identified using a "path" naming convention, with each level of the tree separated by a slash (/) character (see HD wallet path examples). Private keys derived from the master private key start with "m." Public keys derived from the master public key start with "M." Therefore, the first child private key of the master private key is m/0. The first child public key is M/0. The second grandchild of the first child is m/0/1, and so on.

HD钱包中的密钥使用一个“路径”来标识，每一级之间用/来分隔。
由主私钥导出的私钥以“m”打头。
由主公钥导出的公钥以“M“打头。
因此，父私钥导出的第一个子私钥是m/0，第一个公钥是M/0。
第一个子私钥的第二个子私钥是m/0/1，以此类推。

The "ancestry" of a key is read from right to left, until you reach the master key from which it was derived. For example, identifier m/x/y/z describes the key that is the z-th child of key m/x/y, which is the y-th child of key m/x, which is the x-th child of m.
一个密钥的“祖先”是从右向左读，直到主密钥。
例如，标识m/x/y/z描述了一个密钥，它是子密钥m/x/y的第z个子密钥，
子密钥m/x/y是m/x的第y个子密钥，m/x是m的第x个子密钥。

Table 6. HD wallet path examples
表6：HD钱包路径例子

HD路径	**Key described**
m/0	The first (0) child private key from the master private key (m) 主私钥（m）的第一个个子私钥
m/0/0	The first grandchild private key from the first child (m/0) 第一个子私钥（m/0）的第一个私钥
m/0'/0	The first normal grandchild from the first *hardened* child (m/0') 第一个强化子私钥（m/0'）的第一个普通子私钥
m/1/0	The first grandchild private key from the second child (m/1) 第二个子私钥（m/1）的第一个子私钥
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

5.3.9 浏览HD钱包树结构

The HD wallet tree structure offers tremendous flexibility. Each parent extended key can have 4 billion children: 2 billion normal children and 2 billion hardened children. Each of those children can have another 4 billion children, and so on. The tree can be as deep as you want, with an infinite number of generations.
HD钱包树状结构提供了极大的灵活性。
每一个父扩展密钥可以有40亿个子密钥：20亿个普通子密钥，20亿个强化子密钥。
而每个子密钥又会有40亿个子密钥，以此类推。
只要你愿意，这个树结构可以无限延伸。

With all that flexibility, however, it becomes quite difficult to navigate this infinite tree. It is especially difficult to transfer HD wallets between implementations, because the possibilities for internal organization into branches and subbranches are endless.
但是，因为有了这个灵活性，浏览这个无限树就变得非常困难。
特别是在实现之间转移HD钱包尤其困难，因为分支的内部组织的可能性是无穷的。

Two BIPs offer a solution to this complexity by creating some proposed standards for the structure of HD wallet trees. BIP-43 proposes the use of the first hardened child index as a special identifier that signifies the "purpose" of the tree structure. Based on BIP-43, an HD wallet should use only one level-1 branch of the tree, with the index number identifying the structure and namespace of the rest of the tree by defining its purpose. For example,

an HD wallet using only branch m/i&#x27;/ is intended to signify a specific purpose and that purpose is identified by index number "i."
有两个BIP为这个复杂性提供了一个方案：为HD钱包树创建一些推荐的标准结构。
BIP-43建议把第一个强化子索引用作一个特殊的标识，它表示这个树结构的目的。
基于BIP-43，HD钱包应该只使用这个树的一个一级分支，通过定义它的目的，用这个索引号识别这个树的其它部分的结构和命名空间。
例如，只使用分支m/i'/的HD钱包表示一个特定目的，这个目的是由索引i标识的。

Extending that specification, BIP-44 proposes a multiaccount structure as "purpose" number 44' under BIP-43. All HD wallets following the BIP-44 structure are identified by the fact that they only used one branch of the tree: m/44'/.
BIP-44扩展了BIP-43，BIP-44建议了一个多账户结果，作为BIP-43中的目的号44'。
所有遵循BIP-44的HD钱包依据只使用一个分支：m/44'/。

BIP-44 specifies the structure as consisting of five predefined tree levels:
BIP-44规定这个结构包含5个预定义树层级。

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

The first-level "purpose" is always set to 44'.
第一层purpose'：总是44'。

The second-level "coin_type" specifies the type of cryptocurrency coin, allowing for multicurrency HD wallets where each currency has its own subtree under the second level. There are three currencies defined for now: Bitcoin is m/44'/0', Bitcoin Testnet is m/44&#x27;/1&#x27;, and Litecoin is m/44&#x27;/2&#x27;.
第二层coin_type'：指定加密货币的类型，这允许多币种HD钱包，每种货币有自己的子树。
目前定义了三种货币：比特币（m/44'/0'）、Bitcoin Testnet（m/44'/1'）、Litecoin（m/44'/2'）。

The third level of the tree is "account," which allows users to subdivide their wallets into separate logical subaccounts, for accounting or organizational purposes. For example, an HD wallet might contain two bitcoin "accounts": m/44&#x27;/0&#x27;/0&#x27; and m/44&#x27;/0&#x27;/1&#x27;. Each account is the root of its own subtree.
第三层account'：允许用户把钱包划分成逻辑子账户，用于记账或组织目的。
例如，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的子树的根。

On the fourth level, "change," an HD wallet has two subtrees, one for creating receiving addresses and one for creating change addresses. Note that whereas the previous levels used hardened derivation, this level uses normal derivation. This is to allow this level of the tree to export extended public keys for use in a nonsecured environment.
第四层change：一个HD钱包有两个子树，一个是用来创建接收地址，一个是用来创建找零地址。
注意，无论前面层级是否使用强化导出，这一层使用的都是常规导出。这是为了允许这一层的树可以导出扩展公钥，以用在不安全环境中。

Usable addresses are derived by the HD wallet as children of the fourth level, making the fifth level of the tree the "address_index." For example, the third receiving address for bitcoin payments in the primary account would be M/44&#x27;/0&#x27;/0&#x27;/0/2. BIP-44 HD wallet structure examples shows a few more examples.
HD钱包导出的可以使用的地址是第四层的子，所以第五层是数的address_index。
表7给出了一下BIP-44 HD钱包结构的例子。
例如，M/44'/0'/0'/0/2是：m/44'/比特币/主账号/接收付款/第3个地址。

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

Table 7. BIP-44 HD wallet structure examples

表7: BIP-44 HD钱包结构例子。

HD path	**Key described**
M/44‘/0‘/0‘/0/2	The third receiving public key for the primary bitcoin account m/44'/比特币/主账户/收款地址/第3个地址
M/44‘/0‘/3‘/1/14	The fifteenth change-address public key for the fourth bitcoin account m/44'/比特币/第4个账户/找零地址/第15个地址
m/44‘/2‘/0‘/0/1	The second private key in the Litecoin main account, for signing transactions m/44'/莱特币/主账户/收款地址/第2个地址

5.4总结（ddk）

BIP-39：

- 使用一个随机数生成助记词
- 使用助记词和盐生成种子（512位）
  盐：mnemonic +（可选）密码短语

图9：用一个根种子创建主密钥和链码

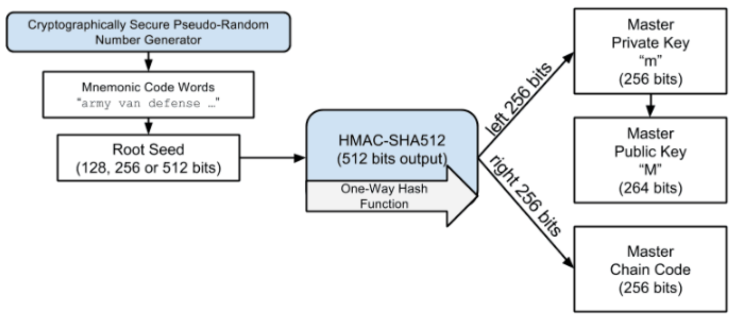
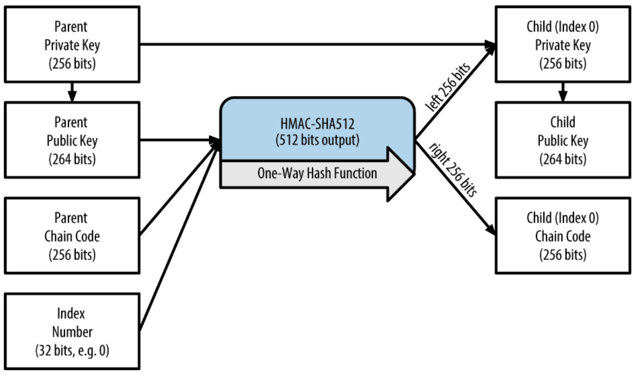

Cryptographically Secure Pseudo-Random Number Generator
Mnemonic Code Words "army van defense ..."
Root Seed (128, 256 or 512 bits)
HMAC-SHA512 (512 bits output)
One-Way Hash Function
left 256 bits
right 256 bits
Master Private Key "m" (256 bits)
Master Public Key "M" (264 bits)
Master Chain Code (256 bits)

图10：扩展一个父私钥，生一个子私钥


Parent Private Key (256 bits)
Parent Public Key (264 bits)
Parent Chain Code (256 bits)
Index Number (32 bits, e.g. 0)
HMAC-SHA512 (512 bits output)
One-Way Hash Function
left 256 bits
right 256 bits
Child (Index 0) Private Key (256 bits)
Child Public Key (264 bits)
Child (Index 0) Chain Code (256 bits)

扩展密钥可简单地储存和表示为：256位密钥和256位链码，合并为512位。
有两种类型的扩展密钥：

- 扩展私钥：是私钥和链码的结合。可用于导出子私钥和子公钥。
- 扩展公钥：是公钥和链码的结合。可用于创建子公钥。

扩展密钥的 Base58Check编码使用了特殊的版本号，从而在Base58编码中生成了前缀xprv和xpub。

图11：扩展一个父公钥来创建一个子公钥

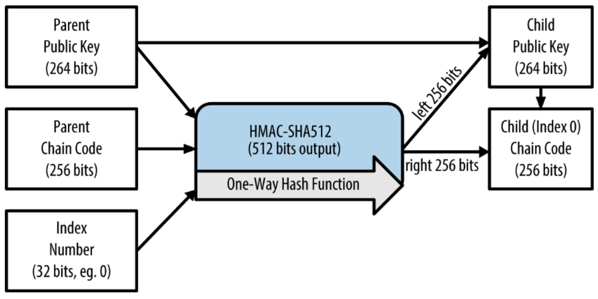
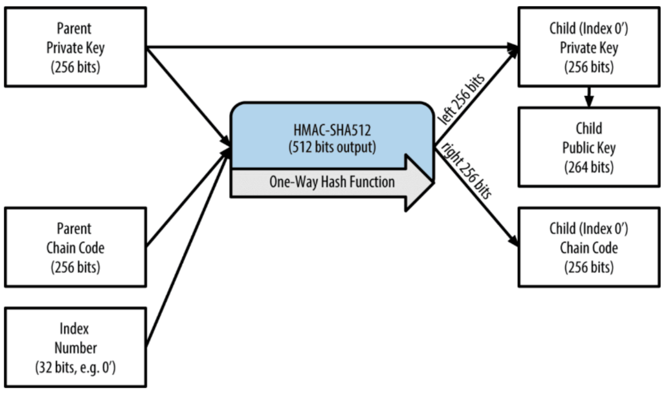

Parent Public Key (264 bits)
Parent Chain Code (256 bits)
Index Number (32 bits, eg. 0)
HMAC-SHA512 (512 bits output)
One-Way Hash Function
left 256 bits
right 256 bits
Child Public Key (264 bits)
Child (Index 0) Chain Code (256 bits)

图13：子密钥的强化导出，忽略了父公钥


Parent Private Key (256 bits)
Parent Chain Code (256 bits)
Index Number (32 bits, e.g. 0′)
HMAC-SHA512 (512 bits output)
One-Way Hash Function
left 256 bits
right 256 bits
Child (Index 0′) Private Key (256 bits)
Child Public Key (264 bits)
Child (Index 0′) Chain Code (256 bits)