HD钱包

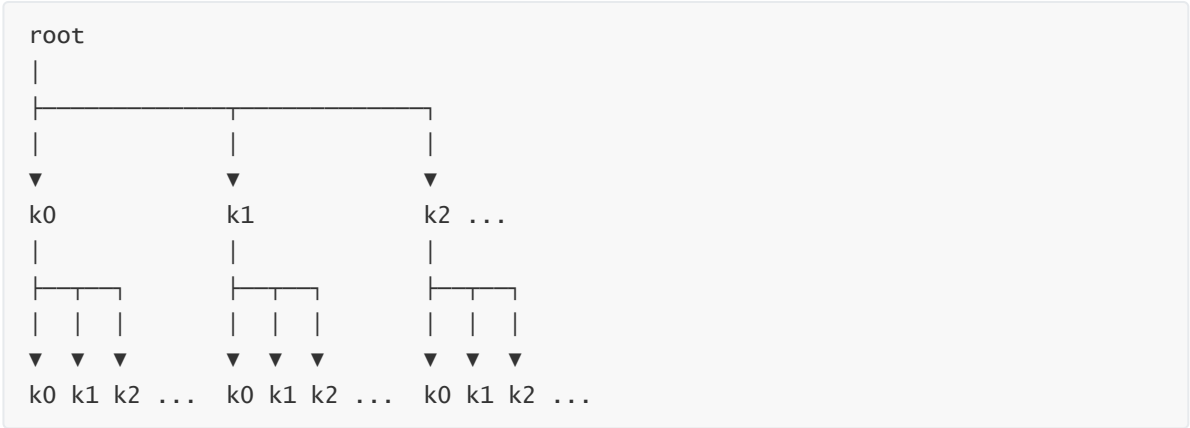
在比特币的链上，实际上并没有账户的概念，某个用户持有的比特币，实际上是其控制的一组UTXO，而这些UTXO可能是相同的地址（对应相同的私钥），也可能是不同的地址（对应不同的私钥）。

出于保护隐私的目的，同一用户如果控制的UTXO其地址都是不同的，那么很难从地址获知某个用户的比特币持币总额。但是，管理一组成千上万的地址，意味着管理成千上万的私钥，管理起来非常麻烦。

能不能只用一个私钥管理成千上万个地址？实际上是可以的。虽然椭圆曲线算法决定了一个私钥只能对应一个公钥，但是，可以通过某种确定性算法，先确定一个私钥k1，然后计算出k2、k3、k4……等其他私钥，就相当于只需要管理一个私钥，剩下的私钥可以按需计算出来。

这种根据某种确定性算法，只需要管理一个根私钥，即可实时计算所有“子私钥”的管理方式，称为HD钱包。

HD是Hierarchical Deterministic的缩写，意思是分层确定性。先确定根私钥root，然后根据索引计算每一层的子私钥：



对于任意一个私钥k，总是可以根据索引计算它的下一层私钥kn：

k_n=hdkey(k,\;n)$k_n = hdkey(k, n)$

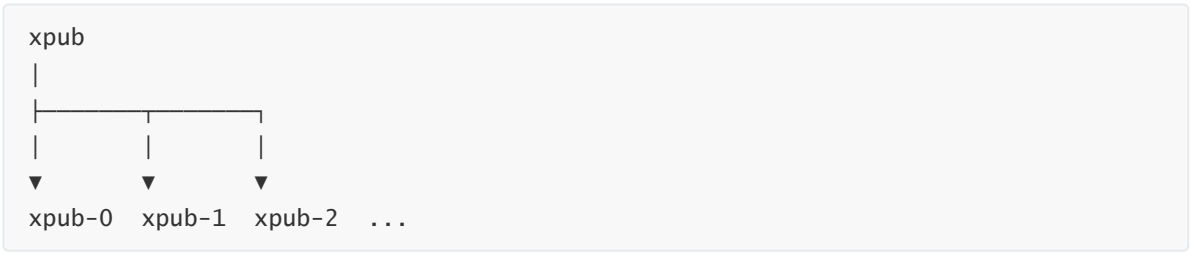
即HD层级实际上是无限的，每一层索引从0～232，约43亿个子key。这种计算被称为衍生（Derivation）。

现在问题来了：如何根据某个私钥计算下一层的子私钥？即函数`hdkey(k, n)`如何实现？

HD钱包采用的计算子私钥的算法并不是一个简单的SHA-256，私钥也不是普通的256位ECDSA私钥，而是一个扩展的512位私钥，记作xprv，它通过SHA-512算法配合ECC计算出子扩展私钥，仍然是512位。通过扩展私钥可计算出用于签名的私钥以及公钥。

简单来说，只要给定一个根扩展私钥（随机512位整数），即可计算其任意索引的子扩展私钥。扩展私钥总是能计算出扩展公钥，记作xpub：

从xprv及其对应的xpub可计算出真正用于签名的私钥和公钥。之所以要设计这种算法，是因为扩展公钥xpub也有一个特点，那就是可以直接计算其子层级的扩展公钥：



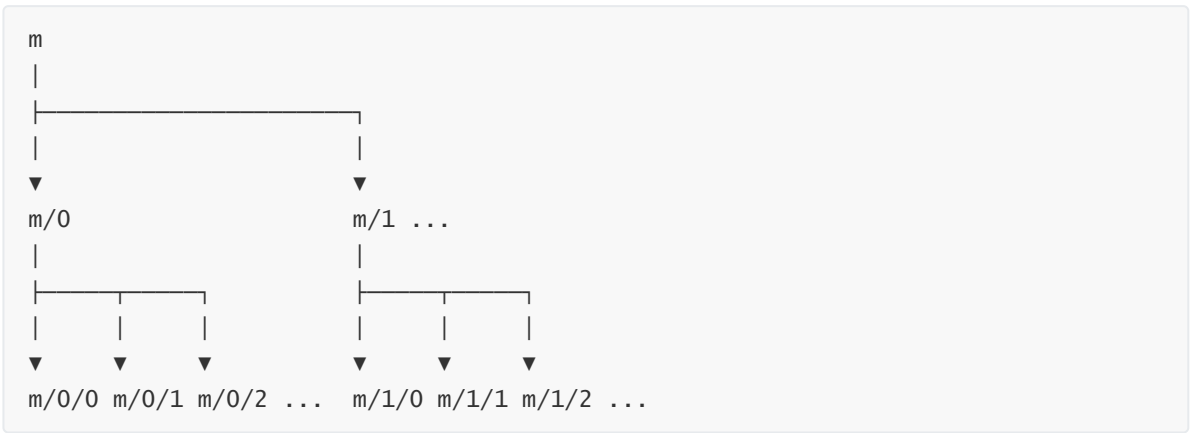
因为xpub只包含公钥，不包含私钥，因此，可以安全地把xpub交给第三方（例如，一个观察钱包），它可以根据xpub计算子层级的所有地址，然后在比特币的链上监控这些地址的余额，但因为没有私钥，所以只能看，不能花。

因此，HD钱包通过分层确定性算法，实现了以下功能：

- 只要确定了扩展私钥xprv，即可根据索引计算下一层的任何扩展私钥；
- 只要确定了扩展公钥xpub，即可根据索引计算下一层的任何扩展公钥；
- 用户只需保存顶层的一个扩展私钥，即可计算出任意一层的任意索引的扩展私钥。

从理论上说，扩展私钥的层数是没有限制的，每一层的数量被限制在0～232，原因是扩展私钥中只有4字节作为索引，因此索引范围是0～232。

通常把根扩展私钥记作`m`，子扩展私钥按层级记作`m/x/y/z`等：



例如，`m/0/2`表示从`m`扩展到`m/0`（索引为0）再扩展到`m/0/2`（索引为2）。

安全性

HD钱包给私钥管理带来了非常大的方便，因为只需要管理一个根扩展私钥，就可以管理所有层级的所有衍生私钥。

但是HD钱包的扩展私钥算法有个潜在的安全性问题，就是如果某个层级的xprv泄露了，可反向推导出上层的xprv，继而推导出整个HD扩展私钥体系。为了避免某个子扩展私钥的泄漏导致上层扩展私钥被反向推导，HD钱包还有一种硬化的衍生计算方式（Hardened Derivation），它通过算法“切断”了母扩展私钥和子扩展私钥的反向推导。HD规范把索引0～231作为普通衍生索引，而索引231～232作为硬化衍生索引，硬化衍生索引通常记作0'、1'、2'……，即索引0'=231，1'=231+1，2'=231+2，以此类推。

因此，`m/44'/0`表示的子扩展私钥，它的第一层衍生索引`44'`是硬化衍生，实际索引是231+44=2147483692。从`m/44'/0`无法反向推导出`m/44'`。

在只有扩展公钥的情况下，只能计算出普通衍生的子公钥，无法计算出硬化衍生的子扩展公钥，即可计算出的子扩展公钥索引被限制在0～231。因此，观察钱包能使用的索引是0～231。

BIP-32

比特币的BIP-32规范详细定义了HD算法原理和各种推导规则，可阅读此文档以便实现HD钱包。

小结

HD钱包采用分层确定性算法通过根扩展私钥计算所有层级的所有子扩展私钥，继而得到扩展公钥和地址；

可以通过普通衍生和硬化衍生两种方式计算扩展子私钥，后者更安全，但对应的扩展公钥无法计算硬化衍生的子扩展公钥；

通过扩展公钥可以在没有扩展私钥的前提下计算所有普通子扩展公钥，此特性可实现观察钱包。

钱包层级

HD钱包算法决定了只要给定根扩展私钥，整棵树的任意节点的扩展私钥都可以计算出来。

我们来看看如何利用bitcoinjs-lib这个JavaScript库来计算HD地址：

```
const bitcoin = require('bitcoinjs-lib');

let
    xprv =
'xprv9s21ZrQH143K4EKMS3q1vbJo564QAbs98BfXQME6nk8UCrnXnv8vWg9qmtup3kTug96p5E3AvarBhPMScQDqMhEEm41rpYEdXBL8qzVZtwz',
    root = bitcoin.HDNode.fromBase58(xprv);

// m/0:
var m_0 = root.derive(0);
console.log("xprv m/0: " + m_0.toBase58());
console.log("xpub m/0: " + m_0.neutered().toBase58());
console.log(" prv m/0: " + m_0.keyPair.toWIF());
console.log(" pub m/0: " + m_0.keyPair.getAddress());

// m/1:
var m_1 = root.derive(0);
console.log("xprv m/1: " + m_1.toBase58());
console.log("xpub m/1: " + m_1.neutered().toBase58());
console.log(" prv m/1: " + m_1.keyPair.toWIF());
console.log(" pub m/1: " + m_1.keyPair.getAddress());
```

```
xprv m/0:
xprv9vV8wfuBFqPwqk1QRQ6W8NWXV7fnTQmf56r5dYXz4YNDafrhGeg6N2dVF7MPhrPUomzdQSSZAj5X
XLCirjqKuEVSnbbqWd1BvkxkGPiNxY9
xpub m/0:
xpub69UVMBS56CxF4E5sXRdWVWTG39WGrsVWSKmgRvwbcsuCTUBqpBzLupwy6P98uDM99qVb6Y32drbS
Fz7iaj5TDYPuRgoHL4U1bQRFHygiMZC
 prv m/0: L3sZcLifBMDkutRFKy7HKNJ5hjJo8vKt8WrHW5mqenSf2e47SHaT
 pub m/0: 1EwGHVkkRNB98FZyhwpmxU9Ax9fVpnBCcZ
xprv m/1:
xprv9vV8wfuBFqPwqk1QRQ6W8NWXV7fnTQmf56r5dYXz4YNDafrhGeg6N2dVF7MPhrPUomzdQSSZAj5X
XLCirjqKuEVSnbbqWd1BvkxkGPiNxY9
xpub m/1:
xpub69UVMBS56CxF4E5sXRdWVWTG39WGrsVWSKmgRvwbcsuCTUBqpBzLupwy6P98uDM99qVb6Y32drbS
Fz7iaj5TDYPuRgoHL4U1bQRFHygiMZC
 prv m/1: L3sZcLifBMDkutRFKy7HKNJ5hjJo8vKt8WrHW5mqenSf2e47SHaT
 pub m/1: 1EwGHVkkRNB98FZyhwpmxU9Ax9fVpnBCcZ
```

注意到以`xprv`开头的`xprv9s21ZrQH...`是512位扩展私钥的Base58编码，解码后得到的就是原始扩展私钥。

可以从某个xpub在没有xprv的前提下直接推算子公钥：

```
const bitcoin = require('bitcoinjs-lib');

let
    xprv =
'xprv9s21ZrQH143K4EKMS3q1vbJo564QAbs98BfXQME6nk8UCrnXnv8vWg9qmtup3kTug96p5E3Avar
BhPMScQDqMhEEm41rpYEdXBL8qzVZtwz',
    root = bitcoin.HDNode.fromBase58(xprv);

// m/0:
let
    m_0 = root.derive(0),
    xprv_m_0 = m_0.toBase58(),
    xpub_m_0 = m_0.neutered().toBase58();

// 方法一：从m/0的扩展私钥推算m/0/99的公钥地址：
let pub_99a = bitcoin.HDNode.fromBase58(xprv_m_0).derive(99).getAddress();

// 方法二：从m/0的扩展公钥推算m/0/99的公钥地址：
let pub_99b = bitcoin.HDNode.fromBase58(xpub_m_0).derive(99).getAddress();

// 比较公钥地址是否相同：
console.log(pub_99a);
console.log(pub_99b);
```

```
1GvmxzNG5i8EqXQuD75XaGjqf2oCEmJ87M
1GvmxzNG5i8EqXQuD75XaGjqf2oCEmJ87M
```

但不能从xpub推算硬化的子公钥：

```
const bitcoin = require('bitcoinjs-lib');

let
```

```
    xprv =
'xprv9s21ZrQH143K4EKMS3q1vbJo564QAbs98BfXQME6nk8UCrnXnv8vWg9qmtup3kTug96p5E3Avar
BhPMScQDqMhEEm41rpYEdXBL8qzVZtwz',
    root = bitcoin.HDNode.fromBase58(xprv);

// m/0:
let
    m_0 = root.derive(0),
    xprv_m_0 = m_0.toBase58(),
    xpub_m_0 = m_0.neutered().toBase58();

// 从m/0的扩展私钥推算m/0/99'的公钥地址:
let pub_99a =
bitcoin.HDNode.fromBase58(xprv_m_0).deriveHardened(99).getAddress();
console.log(pub_99a);

// 不能从m/0的扩展公钥推算m/0/99'的公钥地址:
bitcoin.HDNode.fromBase58(xpub_m_0).deriveHardened(99).getAddress();
```

```
1PVoHWkWFk6uJYDERskP5HSwG7WHfYLacF
TypeError: Could not derive hardened child key
```

BIP-44

HD钱包理论上有无限的层级，对使用secp256k1算法的任何币都适用。但是，如果一种钱包使用`m/1/2/x`，另一种钱包使用`m/3/4/x`，没有一种统一的规范，就会乱套。

比特币的BIP-44规范定义了一种如何派生私钥的标准，它本身非常简单：

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

其中，`purpose`总是`44`，`coin_type`在SLIP-44中定义，例如，`0=BTC`，`2=LTC`，`60=ETH`等。`account`表示用户的某个“账户”，由用户自定义索引，`change=0`表示外部交易，`change=1`表示内部交易，`address_index`则是真正派生的索引为0～231的地址。

例如，某个比特币钱包给用户创建的一组HD地址实际上是：

- m/44'/0'/0'/0/0
- m/44'/0'/0'/0/1
- m/44'/0'/0'/0/2
- m/44'/0'/0'/0/3
- ...

如果是莱特币钱包，则用户的HD地址是：

- m/44'/2'/0'/0/0
- m/44'/2'/0'/0/1
- m/44'/2'/0'/0/2
- m/44'/2'/0'/0/3
- ...

小结

实现了BIP-44规范的钱包可以管理所有币种。相同的根扩展私钥在不同钱包上派生的一组地址都是相同的。

助记词

从HD钱包的创建方式可知，要创建一个HD钱包，我们必须首先有一个确定的512bit（64字节）的随机数种子。

如果用电脑生成一个64字节的随机数作为种子当然是可以的，但是恐怕谁也记不住。

如果自己想一个句子，例如`bitcoin is awesome`，然后计算SHA-512获得这个64字节的种子，虽然是可行的，但是其安全性取决于自己想的句子到底有多随机。像`bitcoin is awesome`本质上就是3个英文单词构成的随机数，长度太短，所以安全性非常差。

为了解决初始化种子的易用性问题，BIP-39规范提出了一种通过助记词来推算种子的算法：

以英文单词为例，首先，挑选2048个常用的英文单词，构造一个数组：

```
const words = ['abandon', 'ability', 'able', ..., 'zoo'];
```

然后，生成128~256位随机数，注意随机数的总位数必须是32的倍数。例如，生成的256位随机数以16进制表示为：

```
179e5af5ef66e5da5049cd3de0258c5339a722094e0fdbbbe0e96f148ae80924
```

在随机数末尾加上校验码，校验码取SHA-256的前若干位，并使得总位数凑成11的倍数，即。上述随机数校验码的二进制表示为`00010000`。

将随机数+校验码按每11 bit一组，得到范围是0~2047的24个整数，把这24个整数作为索引，就得到了最多24个助记词，例如：

```
bleak version runway tell hour unfold donkey defy digital abuse glide please omit
much cement sea sweet tenant demise taste emerge inject cause link
```

由于在生成助记词的过程中引入了校验码，所以，助记词如果弄错了，软件可以提示用户输入的助记词可能不对。

生成助记词的过程是计算机随机产生的，用户只要记住这些助记词，就可以根据助记词推算出HD钱包的种子。

注意：不要自己挑选助记词，原因一是随机性太差，二是缺少校验。

生成助记词可以使用bip39这个JavaScript库：

```
const bip39 = require('bip39');

let words = bip39.generateMnemonic(256);
console.log(words);

console.log('is valid mnemonic? ' + bip39.validateMnemonic(words));
```

```
leaf mansion night switch bread more dutch pigeon various humor cancel history
laundry divorce switch pretty canal tail deliver grape old child quit proud
is valid mnemonic? true
```

运行上述代码，每次都会得到随机生成的不同的助记词。

如果想用中文作助记词也是可以的，给`generateMnemonic()`传入一个中文助记词数组即可：

```
const bip39 = require('bip39');

// 第二个参数rng可以为null:
var words = bip39.generateMnemonic(256, null,
bip39.wordlists.chinese_simplified);
console.log(words);
```

```
亚 自 影 惩 兼 氯 粒 割 床 底 棒 润 铁 蒋 粒 到 畅 官 闷 抽 麻 腹 天 乡
```

注意：同样索引的中文和英文生成的HD种子是不同的。各种语言的助记词定义在[bip-0039-wordlists.md]。

根据助记词推算种子

根据助记词推算种子的算法是PBKDF2，使用的哈希函数是Hmac-SHA512，其中，输入是助记词的UTF-8编码，并设置Key为`mnemonic`+用户口令，循环2048次，得到最终的64字节种子。上述助记词加上口令`bitcoin`得到的HD种子是：

```
b59a8078d4ac5c05b0c92b775b96a466cd136664bfe14c1d49aff3ccc94d52dfb1d59ee628426192
eff5535d6058cb64317ef2992c8b124d0f72af81c9ebfaaa
```

该种子即为HD钱包的种子。

> 要特别注意：用户除了需要记住助记词外，还可以额外设置一个口令。HD种子的生成依赖于助记词和口令，丢失助记词或者丢失口令（如果设置了口令的话）都将导致HD钱包丢失！

用JavaScript代码实现为：

```
const bip39 = require('bip39');

let words = bip39.generateMnemonic(256);
console.log(words);

let seedBuffer = bip39.mnemonicToSeed(words);
let seedAsHex = seedBuffer.toString('hex');
// or use bip39.mnemonicToSeedHex(words)
console.log(seedAsHex);
```

```
tired blur mango myth bomb amount absurd travel box faint guess uniform kitten
crime spring orchard vital flee sustain fantasy squeeze flavor recall electric
293756ce6a417237c6907e0a41540Od59d1bbdfeeb3c7522b6ffccca60df60df5b758670b30bc10f
3fada6952415efa62654674a95efe5343c80eb8827a8944c
```

根据助记词和口令生成HD种子的方法是在`mnemonicToSeed()`函数中传入password：

```
const bip39 = require('bip39');
let words = bip39.generateMnemonic(256);
console.log(words);

let password = 'bitcoin';

let seedAsHex = bip39.mnemonicToSeedHex(words, password);
console.log(seedAsHex);
```

```
shiver perfect during enlist van arm peasant funny pulp cherry soccer fix embody
moment until cushion clock vault increase way tumble chuckle twist survey
8e3cd4205660b5d772f5c3e79e7701f84bfdcbd1f8d38ac1eb04379ede7e28f1e912c82c137dc125
6c0a7bd61b5ffe137816a2e82cc01189b03caf55943aa302
```

从助记词算法可知，只要确定了助记词和口令，生成的HD种子就是确定的。

如果两个人的助记词相同，那么他们的HD种子也是相同的。这也意味着如果把助记词抄在纸上，一旦泄漏，HD种子就泄漏了。

如果在助记词的基础上设置了口令，那么只知道助记词，不知道口令，也是无法推算出HD种子的。

把助记词抄在纸上，口令记在脑子里，这样，泄漏了助记词也不会导致HD种子被泄漏，但要牢牢记住口令。

最后，我们使用助记词+口令的方式来生成一个HD钱包的HD种子并计算出根扩展私钥：

```
const
    bitcoin = require('bitcoinjs-lib'),
    bip39 = require('bip39');

let
    words = 'bleak version runway tell hour unfold donkey defy digital abuse
glide please omit much cement sea sweet tenant demise taste emerge inject cause
link',
    password = 'bitcoin';

// 计算seed:
let seedHex = bip39.mnemonicToSeedHex(words, password);
console.log('seed: ' + seedHex); // b59a8078...c9ebfaaa

// 生成root:
let root = bitcoin.HDNode.fromSeedHex(seedHex);
console.log('xprv: ' + root.toBase58()); // xprv9s21ZrQH...uLgyr9kF
console.log('xpub: ' + root.neutered().toBase58()); // xpub661MyMwA...oy32fcRG

// 生成派生key:
let child0 = root.derivePath("m/44'/0'/0'/0/0");
console.log("prv m/44'/0'/0'/0/0: " + child0.keyPair.toWIF()); //
KzuPk3PXKdnd6QwLqUCK38PrXoqJfJmACzxTaa6TFKzPJR7H7AFg
console.log("pub m/44'/0'/0'/0/0: " + child0.getAddress()); //
1PwKkrF366RdTuYsS8KWEbGxfP4bikegcS
```

```
seed:
b59a8078d4ac5c05b0c92b775b96a466cd136664bfe14c1d49aff3ccc94d52dfb1d59ee628426192
eff5535d6058cb64317ef2992c8b124d0f72af81c9ebfaaa
xprv:
xprv9s21ZrQH143K4ATirFwVJe2ZvmmBACdKcK1cWXtDdMhLFViGUDBDWJnWHLYGz5624M8zFsYKGoGt
8UVcPMwu9MR5gveP98ZoRq7uLgyr9kF
xpub:
xpub661MyMwAqRbcGeYBxHUVfmyJUobfZfMAyXwDJvHqBhEK8J3R1kVU476z8dvaQYCvwWPTQFzVZUuN
TAWWdhNQVY2TjTiFseh8j9goy32fcRG
prv m/44'/0'/0'/0/0: KzuPk3PXKdnd6QwLqUCK38PrXoqJfJmACzxTaa6TFKzPJR7H7AFg
pub m/44'/0'/0'/0/0: 1PwKkrF366RdTuYsS8KWEbGxfP4bikegcS
```

可以通过https://iancoleman.io/bip39/在线测试BIP-39并生成HD钱包。请注意，该网站仅供测试使用。生成正式使用的HD钱包*必须*在*可信任的离线环境*下操作。

小结

BIP-39规范通过使用助记词+口令来生成HD钱包的种子，用户只需记忆助记词和口令即可随时恢复HD钱包。

丢失助记词或者丢失口令均会导致HD钱包丢失。