4 密钥和地址

You may have heard that bitcoin is based on *cryptography*, which is a branch of mathematics used extensively in computer security. Cryptography means "secret writing" in Greek, but the science of cryptography encompasses more than just secret writing, which is referred to as encryption. Cryptography can also be used to prove knowledge of a secret without revealing that secret (digital signature), or prove the authenticity of data (digital fingerprint). These types of cryptographic proofs are the mathematical tools critical to bitcoin and used extensively in bitcoin applications.
你可能听说过，比特币是基于密码学，密码学是在计算机安全中广泛使用的一个数学分支。
密码学在希腊语中的意思是“秘密书写”，但密码学不仅是秘密书写，还指加密。
密码学也可以用于：证明了解一个秘密，而不用揭示这个秘密（数字签名）；或证明数据的不会泄露秘密（数字签名），或证明数据的真实性（数字指纹）。
这些类型的密码学证明对比特币来说是关键的数学工具，并且在比特币应用中被广泛使用。

Ironically, encryption is not an important part of bitcoin, as its communications and transaction data are not encrypted and do not need to be encrypted to protect the funds. In this chapter we will introduce some of the cryptography used in bitcoin to control ownership of funds, in the form of keys, addresses, and wallets.
然而，加密不是比特币的重要部分，比特币的通信和交易数据都没有被加密，也不需要加密来保护资金。
在本章中，我们介绍一些在比特币中用于控制资金所有权的密码学，形式是密钥、地址和钱包。

4.1介绍

Ownership of bitcoin is established through *digital keys*, *bitcoin addresses*, and *digital signatures*.
比特币的所有权是通过下列建立的：数字密钥、比特币地址、数字签名。

The digital keys are not actually stored in the network, but are instead created and stored by users in a file, or simple database, called a *wallet*. The digital keys in a user’s wallet are completely independent of the bitcoin protocol and can be generated and managed by the user’s wallet software without reference to the blockchain or access to the internet. Keys enable many of the interesting properties of bitcoin, including decentralized trust and control, ownership attestation, and the cryptographic-proof security model.
数字密钥并不存储在网络中，而是由用户生成和存储，存储在文件或简单数据库（钱包）中。
用户钱包中的数字密钥完全独立于比特币协议，可由用户钱包软件生成和管理，而无需参考区块链或访问互联网。
密钥实现了比特币的许多有趣特性，包括去中心化信任和控制、所有权证明、密码证明安全模型。

Most bitcoin transactions require a valid digital signature to be included in the blockchain, which can only be generated with a secret key; therefore, anyone with a copy of that key has control of the bitcoin. The digital signature used to spend funds is also referred to as a *witness*, a term used in cryptography. The witness data in a bitcoin transaction testifies to the true ownership of the funds being spent.
大多数比特币交易都需要一个有效的数字签名才会被存储在区块链中，数字签名只能用私钥生成；
因此，任何拥有私钥的人都可以控制那个比特币。
用于花费资金的数字签名也被称为见证（witness），这是密码学中使用的一个术语。
比特币交易中的见证数据证明了对花费的资金拥有所有权。

Keys come in pairs consisting of a private (secret) key and a public key. Think of the public key as similar to a bank account number and the private key as similar to the secret PIN,

or signature on a check, that provides control over the account. These digital keys are very rarely seen by the users of bitcoin. For the most part, they are stored inside the wallet file and managed by the bitcoin wallet software.
密钥是成对出现的，一个私钥，一个公钥。
公钥就像银行的账户，私钥就像支票上的签名，它提供对账户的控制。。
比特币的用户很少直接看到这些数字密钥。
多数情况下，比特币钱包软件把它们存储在钱包文件中，并管理它们。

In the payment portion of a bitcoin transaction, the recipient's public key is represented by its digital fingerprint, called a *bitcoin address*, which is used in the same way as the beneficiary name on a check (i.e., "Pay to the order of"). In most cases, a bitcoin address is generated from and corresponds to a public key. However, not all bitcoin addresses represent public keys; they can also represent other beneficiaries such as scripts, as we will see later in this chapter. This way, bitcoin addresses abstract the recipient of funds, making transaction destinations flexible, similar to paper checks: a single payment instrument that can be used to pay into people's accounts, pay into company accounts, pay for bills, or pay to cash. The bitcoin address is the only representation of the keys that users will routinely see, because this is the part they need to share with the world.
在比特币交易的支付环节，收款人的公钥是用它的数字指纹来表示的，称为“比特币地址”，就像支票上的收款人姓名。
多数情况下，比特币地址由一个公钥生成，并与其对应。
然而，并非所有比特币地址都代表公钥，它们也可以代表其他支付对象，例如脚本，将在本章后面看到。
这样一来，比特币地址就是资金接收者的抽象，使得交易的目的地更加灵活，就像支票一样：可以支付给个人账户、公司账户、支付单支、支付现金。
比特币地址是用户经常看到的密钥的唯一表示，因为他们需要把这部分告诉其他人。

First, we will introduce cryptography and explain the mathematics used in bitcoin. Next, we will look at how keys are generated, stored, and managed. We will review the various encoding formats used to represent private and public keys, addresses, and script addresses. Finally, we will look at advanced use of keys and addresses: vanity, multisignature, and script addresses and paper wallets.
首先，我们将介绍密码学，并解释在比特币中使用的数学知识。
然后，了解密钥如何被产生、存储和管理。
我们将了解各种编码格式，用于表示私钥和公钥、地址、脚本地址。
最后，介绍密钥和地址的高级用途：比特币靓号、多签名、脚本地址、纸钱包。

4.1.1 公钥密码学和加密货币

Public key cryptography was invented in the 1970s and is a mathematical foundation for computer and information security.
公钥密码学发明于1970年代，它是计算机和信息安全的一个数学基础。

Since the invention of public key cryptography, several suitable mathematical functions, such as prime number exponentiation and elliptic curve multiplication, have been discovered. These mathematical functions are practically irreversible, meaning that they are easy to calculate in one direction and infeasible to calculate in the opposite direction. Based on these mathematical functions, cryptography enables the creation of digital secrets and unforgeable digital signatures. Bitcoin uses elliptic curve multiplication as the basis for its cryptography.
自从发明了公钥密码学之后，已经发现了一些合适的数学函数，例如：素数求幂和椭圆曲线乘法。
这些数学函数都是不可逆的，就是说，很容易在一个方向上计算，但无法在反方向上计算。
基于这些数学函数，密码学能够创建数字秘密和不可伪造的数字签名。
比特币使用椭圆曲线乘法作为其密码学的基础。

In bitcoin, we use public key cryptography to create a key pair that controls access to bitcoin. The key pair consists of a private key and—derived from it—a unique public key.

The public key is used to receive funds, and the private key is used to sign transactions to spend the funds.
在比特币中，我们用公钥密码学创建一个密钥对，用于控制比特币。
密钥对包括一个私钥和一个公钥（用私钥生成）。
公钥用于接收资金，私钥用于对交易进行签名，以花费资金。

There is a mathematical relationship between the public and the private key that allows the private key to be used to generate signatures on messages. This signature can be validated against the public key without revealing the private key.
公钥和私钥之间有一个数学关系，使得可以使用私钥对消息生成签名。
可以用公钥验证这个签名，而不需要泄露私钥。

When spending bitcoin, the current bitcoin owner presents her public key and a signature (different each time, but created from the same private key) in a transaction to spend those bitcoin. Through the presentation of the public key and signature, everyone in the bitcoin network can verify and accept the transaction as valid, confirming that the person transferring the bitcoin owned them at the time of the transfer.
当花费比特币时，比特币的当前所有者需要在交易中提供他的公钥和签名（签名每次都不同，但都是用同一私钥生成的）。
通过提交公钥和签名，比特币网络中的所有人都可以验证和承认这个交易是有效的，确认转钱的这个人对比特币有所有权。

Tip：In most wallet implementations, the private and public keys are stored together as a key pair for convenience. However, the public key can be calculated from the private key, so storing only the private key is also possible.
提示：在大多数钱包中，为了方便，是把私钥和公钥作为一个密钥对来存储的。
但是，可以用私钥计算得到公钥，所以也可以只存储私钥。

4.1.2 私钥和公钥

A bitcoin wallet contains a collection of key pairs, each consisting of a private key and a public key. The private key (k) is a number, usually picked at random. From the private key, we use elliptic curve multiplication, a one-way cryptographic function, to generate a public key (K). From the public key (K), we use a one-way cryptographic hash function to generate a bitcoin address (A).
比特币钱包中包含一些密钥对，每个密钥对有一个私钥和一个公钥。
私钥（k）是一个数字，通常是随机选择的数字。
有了私钥，就可以使用椭圆曲线乘法来生成公钥（K），椭圆曲线乘法是一个单向加密函数。
有了公钥（K），再使用一个单向加密哈希函数生成比特币地址（A）。

In this section, we will start with generating the private key, look at the elliptic curve math that is used to turn that into a public key, and finally, generate a bitcoin address from the public key. The relationship between private key, public key, and bitcoin address is shown in Private key, public key, and bitcoin address.
本节从生成私钥开始，讲述椭圆曲线运算如何用私钥生成公钥，最后在用公钥生成比特币地址。
下图显示了私钥、公钥、比特币地址之间的关系。

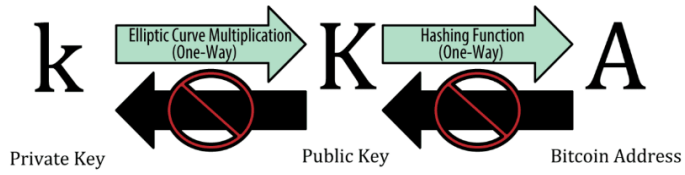


Figure 1. Private key, public key, and bitcoin address

Why Use Asymmetric Cryptography (Public/Private Keys)?

为什么使用非对称密码学（公钥/私钥）？

Why is asymmetric cryptography used in bitcoin? It’s not used to "encrypt" (make secret) the transactions. Rather, the useful property of asymmetric cryptography is the ability to generate *digital signatures*. A private key can be applied to the digital fingerprint of a transaction to produce a numerical signature. This signature can only be produced by someone with knowledge of the private key. However, anyone with access to the public key and the transaction fingerprint can use them to *verify* the signature. This useful property of asymmetric cryptography makes it possible for anyone to verify every signature on every transaction, while ensuring that only the owners of private keys can produce valid signatures.
为什么在比特币中使用非对称密码学?
非对称密码学不是用于对交易进行“加密”（使交易成为秘密）。
非对称密码学的有用特性是能够生成数字签名。可以对交易的数字指纹应用一个私钥，从而生成一个数字签名。这个签名只能由知道私钥的人生成。但是，任何知道公钥和交易指纹的人都可以使用它们来验证这个签名。
这个特性使得每个人都可以验证每笔交易的每个签名，同时保证：只有私钥的所有者能生成有效的签名。

4.1.3 私钥

A private key is simply a number, picked at random. Ownership and control over the private key is the root of user control over all funds associated with the corresponding bitcoin address. The private key is used to create signatures that are required to spend bitcoin by proving ownership of funds used in a transaction. The private key must remain secret at all times, because revealing it to third parties is equivalent to giving them control over the bitcoin secured by that key. The private key must also be backed up and protected from accidental loss, because if it’s lost it cannot be recovered and the funds secured by it are forever lost, too.
私钥只是随机选出的一个数字。
对私钥的所有权和控制，是对相应比特币地址关联的资金的控制的根本。
在花费比特币时，私钥用于创建签名，以证明对资金的所有权。
私钥必须始终保密，因为一旦被泄露，别人就能用它控制你的比特币了。
私钥还必须进行备份，以防意外丢失，因为一旦丢失，就无法复原，你的比特币也就丢了。

Tip：The bitcoin private key is just a number. You can pick your private keys randomly using just a coin, pencil, and paper: toss a coin 256 times and you have the binary digits of a random private key you can use in a bitcoin wallet. The public key can then be generated from the private key.
提示：比特币私钥只是一个数字。
你可以用硬币、铅笔和纸来随机选择你的私钥：掷硬币256次，正面为1，反面为0，用笔和纸记录袭来，你就得到了一个私钥，可以在比特币钱包中使用。
然后，用这个私钥可以生成对应的公钥。

Generating a private key from a random number
从一个随机数生成一个私钥

The first and most important step in generating keys is to find a secure source of entropy, or randomness. Creating a bitcoin key is essentially the same as "Pick a number between 1 and 2^{256}." The exact method you use to pick that number does not matter as long as it is not predictable or repeatable. Bitcoin software uses the underlying operating system’s random number generators to produce 256 bits of entropy (randomness). Usually, the OS random number generator is initialized by a human source of randomness, which is why you may be asked to wiggle your mouse around for a few seconds.
第一步也是最重要的一步，是找到一个安全的随机源（熵，平均信息量）。
创建一个比特币密钥实际上等同于：在1 到2^{256}之间选一个数字。
选择数字的方法并不重要，只要它不是可预测的或重复的。

比特币软件使用操作系统底层的随机数生成器来产生256位的随机数。
通常情况下，操作系统随机数生成器由一个人工随机源来进行初始化，这就是为什么你会被要求晃动鼠标几秒钟。

More precisely, the private key can be any number between 0 and n - 1 inclusive, where n is a constant (n = 1.1578 * 10^{77}, slightly less than 2^{256}) defined as the order of the elliptic curve used in bitcoin (see Elliptic Curve Cryptography Explained). To create such a key, we randomly pick a 256-bit number and check that it is less than n. In programming terms, this is usually achieved by feeding a larger string of random bits, collected from a cryptographically secure source of randomness, into the SHA256 hash algorithm, which will conveniently produce a 256-bit number. If the result is less than n, we have a suitable private key. Otherwise, we simply try again with another random number.
更准确地说，私钥可以是1和n-1之间的任何数字，n是一个常数（n=1.158 * 10^{77}，略小于2^{256}），被定义为由比特币所使用的椭圆曲线的阶。
为了生成这样一个私钥，我们随机选择一个256位的数字，并确认它是小于n。
在编程中，一般是这样实现的：输入一个很长的字符串（随机比特），它来自一个随机加密安全源，使用SHA256哈希算法将其转换为一个256位数字。如果运算结果小于n，我们就得到了一个私钥。否则，就用另一个随机数再试一次。

Warning：Do not write your own code to create a random number or use a "simple" random number generator offered by your programming language. Use a cryptographically secure pseudorandom number generator (CSPRNG) with a seed from a source of sufficient entropy. Study the documentation of the random number generator library you choose to make sure it is cryptographically secure. Correct implementation of the CSPRNG is critical to the security of the keys.
警告：不要自己写代码来生成随机数，或使用编程语言提供的简单随机数生成器来获得随机数。
使用加密安全伪随机数生成器（CSPRNG），并用来自一个足够随机源的一个种子。
研究一下你所选择的随机数生成器库的文档，确保它是加密安全的。
CSPRNG的正确实现对于密钥的安全至关重要。

The following is a randomly generated private key (k) shown in hexadecimal format (256 bits shown as 64 hexadecimal digits, each 4 bits):
下面是一个随机生成的私钥（k），显示的是十六进制格式（256位表示为64个十六进制数字）。

```
1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
```

Tip：The size of bitcoin's private key space, (2^{256}) is an unfathomably large number. It is approximately 10^{77} in decimal. For comparison, the visible universe is estimated to contain 10^{80} atoms.
提示：比特币私钥空间的大小是2^{256}，这是一个非常非常大的数字。
十进制表示大约是10^{77}，而据估计，可见宇宙只含有10^{80}个原子。

To generate a new key with the Bitcoin Core client (see [ch03_bitcoin_client]), use the getnewaddress command. For security reasons it displays the public key only, not the private key. To ask bitcoind to expose the private key, use the dumpprivkey command. The dumpprivkey command shows the private key in a Base58 checksum-encoded format called the *Wallet Import Format* (WIF), which we will examine in more detail in Private key formats. Here's an example of generating and displaying a private key using these two commands:
为了用Bitcoin Core客户端生成一个新的密钥，使用 getnewaddress 命令。
出于安全考虑，这个命令显示公钥，而不显示私钥。
如果让bitcoind显示私钥，使用 dumpprivkey 命令。
dumpprivkey 命令会把私钥显示为 Base58校验和编码格式，这种格式称为钱包导入格式（WIF）。
下面是使用这两个命令的例子。

```
$ bitcoin-cli getnewaddress
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy

$ bitcoin-cli dumpprivkey 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
```

```
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

The dumpprivkey command opens the wallet and extracts the private key that was generated by the getnewaddress command. It is not possible for bitcoind to know the private key from the public key unless they are both stored in the wallet.
dumpprivkey 命令打开钱包，提取由 getnewaddress 命令生成的私钥。
除非公钥和私钥都保存在钱包中，否则bitcoind不可能知道这个私钥。

Tip：The dumpprivkey command does not generate a private key from a public key, as this is impossible. The command simply reveals the private key that is already known to the wallet and which was generated by the getnewaddress command.
提示：dumpprivkey命令不能从公钥得到对应的私钥，因为这是不可能的。
这个命令只是显示钱包中已有的私钥，即由getnewaddress命令生成的私钥。

You can also use the Bitcoin Explorer command-line tool (see [appdx_bx]) to generate and display private keys with the commands seed, ec-new, and ec-to-wif:
你还可以使用Bitcoin Explorer命令行工具（参见附录中的[appdx_bx]），
使用命令seed、ec-new、ec-to-wif生成和显示私钥。

```
$ bx seed | bx ec-new | bx ec-to-wif
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

4.1.4 公钥

The public key is calculated from the private key using elliptic curve multiplication, which is irreversible: *K* = *k* * *G*, where *k* is the private key, *G* is a constant point called the *generator point*, and *K* is the resulting public key. The reverse operation, known as "finding the discrete logarithm"—calculating *k* if you know *K*—is as difficult as trying all possible values of *k*, i.e., a brute-force search. Before we demonstrate how to generate a public key from a private key, let's look at elliptic curve cryptography in a bit more detail.
使用椭圆曲线乘法，从私钥计算得到公钥，这是不可逆转的过程：`K = k*G`
其中，`k`是私钥，`G`是常数点（生成点），`K`是公钥。
其反向运算是非常困难的，被称为“寻找离散对数”，需要暴力搜索所有的可能的`k`值。
在说明如何从私钥得到公钥之前，我们先稍微看看椭圆曲线密码学的细节。

`Tip`: Elliptic curve multiplication is a type of function that cryptographers call a "trap door" function: it is easy to do in one direction (multiplication) and impossible to do in the reverse direction (division). The owner of the private key can easily create the public key and then share it with the world knowing that no one can reverse the function and calculate the private key from the public key. This mathematical trick becomes the basis for unforgeable and secure digital signatures that prove ownership of bitcoin funds.
提示：椭圆曲线乘法是一种函数，密码学家称之为“陷阱门函数”：在一个方向上（乘法）很容易实现，但反方向上很难实现（除法）。
私钥的所有者可以很容易得到公钥，然后提供给其他人，但没人能够根据公钥计算出私钥。
这个数学技巧成为不可伪造和安全的“数字签名”的基础，它能证明比特币资金的所有权。

4.1.5 椭圆曲线密码学的解释

Elliptic curve cryptography is a type of asymmetric or public key cryptography based on the discrete logarithm problem as expressed by addition and multiplication on the points of an elliptic curve.
椭圆曲线密码学是一种非对称或公钥加密法，它基于离散对数问题，
可以用对椭圆曲线上的点进行加法或乘法运算来表达。

An elliptic curve is an example of an elliptic curve, similar to that used by bitcoin.
下图是一个椭圆曲线的例子，类似于比特币所用的椭圆曲线。

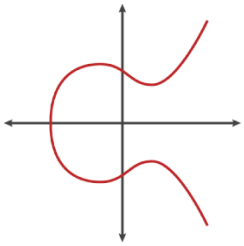


Figure 2. An elliptic curve

Bitcoin uses a specific elliptic curve and set of mathematical constants, as defined in a standard called secp256k1, established by the National Institute of Standards and Technology (NIST). The secp256k1 curve is defined by the following function, which produces an elliptic curve:
比特币使用了一个特定的椭圆曲线，和一套数学常量，定义在一个标准中（secp256k1），这个标准是由NIST制定的。
secp256k1曲线是由下面这个函数定义，该函数产生了一个椭圆曲线。

$$y^2 = (x^3 + 7)\} \text{ over } (F_p)$$

或

$$y^2 \bmod p = (x^3 + 7) \bmod p$$

The *mod p* (modulo prime number p) indicates that this curve is over a finite field of prime order *p*, also written as \(\(\mathbb{F}_p \)\), where p = $2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$, a very large prime number.
mod p（对素数p取模）表示，该曲线是在素数p的有限域内，也写作Fp，
其中p = $2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$， 这是个一个非常大的素数。

Because this curve is defined over a finite field of prime order instead of over the real numbers, it looks like a pattern of dots scattered in two dimensions, which makes it difficult to visualize. However, the math is identical to that of an elliptic curve over real numbers. As an example, Elliptic curve cryptography: visualizing an elliptic curve over F(p), with p=17 shows the same elliptic curve over a much smaller finite field of prime order 17, showing a pattern of dots on a grid. The secp256k1 bitcoin elliptic curve can be thought of as a much more complex pattern of dots on a unfathomably large grid.
因为这个曲线被定义在一个素数的有限域内，而不是定义在实数的范围内，所以，它的函数图像看起来是分散在二维平面上的离散的点，因此很难可视化。
但是，其中的数学原理与实数范围的椭圆曲线相同。
举个例子，下图显示的椭圆曲线是在F(p)上，p=17，显示了网格上点的模式。
secp256k1比特币椭圆曲线可以被想象成一个极大的网格上点的非常复杂的模式。

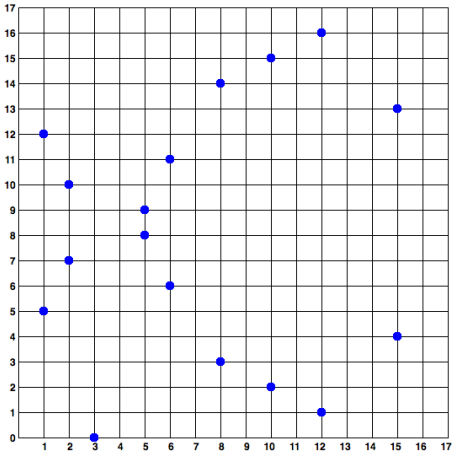


Figure 3. Elliptic curve cryptography: visualizing an elliptic curve over F(p), with p=17

So, for example, the following is a point P with coordinates (x,y) that is a point on the secp256k1 curve:
例如，P的坐标是(x,y)，它是 secp256k1 曲线上的一个点。

```
P = (55066263022277343669578718895168534326250603453777594175500187360389116729240,
32670510020758816978083085130507043184471273380659243275938904335757337482424)
```

Using Python to confirm that this point is on the elliptic curve shows how you can check this yourself using Python:
例1显示了如何使用Python检查这个。

Example 1. Using Python to confirm that this point is on the elliptic curve
例1：使用Python确认这个点在椭圆曲线上。

```
Python 3.4.0 (default, Mar 30 2014, 19:23:13)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> p = 115792089237316195423570985008687907853269984665640564039457584007908834671663
>>> x = 55066263022277343669578718895168534326250603453777594175500187360389116729240
>>> y = 32670510020758816978083085130507043184471273380659243275938904335757337482424
>>> (x ** 3 + 7 - y**2) % p
0
```

In elliptic curve math, there is a point called the "point at infinity," which roughly corresponds to the role of zero in addition. On computers, it's sometimes represented by x = y = 0 (which doesn't satisfy the elliptic curve equation, but it's an easy separate case that can be checked).
在椭圆曲线的数学中，有一个点称为“无穷远点”，这大致对应于0在加法中的作用。
在计算机中，它有时表示为x=y=0（它不满足椭圆曲线方程，但这是一个容易检查的特例）。

There is also a + operator, called "addition," which has some properties similar to the traditional addition of real numbers that gradeschool children learn. Given two points P_1 and P_2 on the elliptic curve, there is a third point $P_3 = P_1 + P_2$, also on the elliptic curve.
还有一个+运算符，称为“加法”，就像小学数学中的实数相加。
给定椭圆曲线上的两个点P_1 和P_2，则$P_3 = P_1 + P_2$也在椭圆曲线上。

Geometrically, this third point P_3 is calculated by drawing a line between P_1 and P_2. This line will intersect the elliptic curve in exactly one additional place. Call this point $P_3' = (x, y)$. Then reflect in the x-axis to get $P_3 = (x, -y)$.
在几何图形中，P_3是通过在P_1 和 P_2之间画一条线来计算。
这条直线恰好与椭圆曲线相交于另外一个地方。这个点就是 $P_3' = (x, y)$。
然后，反转x轴，就得到了$P_3 = (x, -y)$

There are a couple of special cases that explain the need for the "point at infinity."

有一些特例，解释了需要“无穷远点”。

If P_1 and P_2 are the same point, the line "between" P_1 and P_2 should extend to be the tangent on the curve at this point P_1. This tangent will intersect the curve in exactly one new point. You can use techniques from calculus to determine the slope of the tangent line. These techniques curiously work, even though we are restricting our interest to points on the curve with two integer coordinates!
如果P_1 和P_2是同一点，则P_1 和P_2的连线则为点P_1对这个曲线的切线。
这个切线与曲线相交于一点。
你可以使用微积分的技术来确定切线的斜率。
这些技术行为古怪，即使我们用两个整数坐标来限制曲线上的点！

In some cases (i.e., if P_1 and P_2 have the same x values but different y values), the tangent line will be exactly vertical, in which case P3 = "point at infinity."
在某些情况下（即，如果P_1 和P_2有相同的x值，但不同的y值），则切线会正好垂直，在这种情况下，P3 =“无穷远点”。

If P_1 is the "point at infinity," then $P_1 + P_2 = P_2$. Similarly, if P_2 is the point at infinity, then $P_1 + P_2 = P_1$. This shows how the point at infinity plays the role of zero.
It turns out that + is associative, which means that (A + B) + C = A + (B + C). That means we can write A + B + C without parentheses and without ambiguity.
Now that we have defined addition, we can define multiplication in the standard way that extends addition. For a point P on the elliptic curve, if k is a whole number, then kP = P + P + P + … + P (k times). Note that k is sometimes confusingly called an "exponent" in this case.
如果P_1是“无穷远点”，那么$P_1 + P_2 = P_2$。
如果P_2是“无穷远点”，那么$P_1 + P_2 = P_1$。
这就是把无穷远点看做0。
事实证明，这里+运算符遵守结合律，即(A+B)+C = A+(B+C)。
这就是说，我们可以直接写成A+B+C，不用括号也不会引起混淆。
既然我们已经定义了椭圆加法，我们可以以标准的方法定义乘法，它扩展了加法。
给定椭圆曲线上的点P，如果k是一个整数，则 kP = P + P + P + …+ P（k次）。
注意，在这种情况下，K有时被让人迷惑地称为“指数”。

ddk问题：本节没看懂

4.1.6 生成公钥

Starting with a private key in the form of a randomly generated number *k*, we multiply it by a predetermined point on the curve called the *generator point G* to produce another point somewhere else on the curve, which is the corresponding public key *K*. The generator point is specified as part of the secp256k1 standard and is always the same for all keys in bitcoin:
私钥是一个随机生成的数字k，我们将其与曲线上预定的生成点G相乘，得到曲线上的另一个点，它就是对应的公钥 K。
生成点是在secp256k1标准中规定的，对于比特币的所有密钥都是相同的。

K = k * G

where *k* is the private key, *G* is the generator point, and *K* is the resulting public key, a point on the curve. Because the generator point is always the same for all bitcoin users, a private key *k* multiplied with *G* will always result in the same public key *K*. The relationship between *k* and *K* is fixed, but can only be calculated in one direction, from *k* to *K*. That's why a bitcoin address (derived from *K*) can be shared with anyone and does not reveal the user's private key (*k*).
k是私钥，G是生成点，K是公钥。
因为生成点是固定的，所以私钥k乘以G，总是得到相同的公钥K。
k和K之间的关系是固定的，但只能单向运算，即从k得到K。
因此，可以把比特币地址（从K得到）提供给别人，但不会泄露私钥（k）。

Tip: A private key can be converted into a public key, but a public key cannot be converted back into a private key because the math only works one way.
提示：私钥可以转换为公钥，但公钥不能转换回私钥，因为这个数学计算是单向的。

Implementing the elliptic curve multiplication, we take the private key *k* generated previously and multiply it with the generator point G to find the public key *K*:
在实现椭圆曲线乘法中，我们用之前产生的私钥k和与生成点G相乘，得到公钥K。

```
K = 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD * G
```

Public key *K* is defined as a point K = (x,y):
公钥K 被定义为点 K = (x,y)。

```
K = (x, y)
where,
x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

To visualize multiplication of a point with an integer, we will use the simpler elliptic curve over real numbers—remember, the math is the same. Our goal is to find the multiple *kG* of the generator point *G*, which is the same as adding *G* to itself, *k* times in a row. In elliptic curves, adding a point to itself is the equivalent of drawing a tangent line on the point and finding where it intersects the curve again, then reflecting that point on the x-axis.
为了图示整数点的乘法，我们使用一个较为简单的椭圆曲线，它使用实数。记住，数学原理是相同的。
我们的目标是找到kG。也就是将G相加k次。
在椭圆曲线中，一个点加上本身等同于在这个点上画一条切线，它与曲线相交，然后在x轴上反转那个点。

Elliptic curve cryptography: visualizing the multiplication of a point G by an integer k on an elliptic curve shows the process for deriving *G*, *2G*, *4G*, as a geometric operation on the curve.
下图显示了在曲线上得到 G、2G、4G 的几何操作。

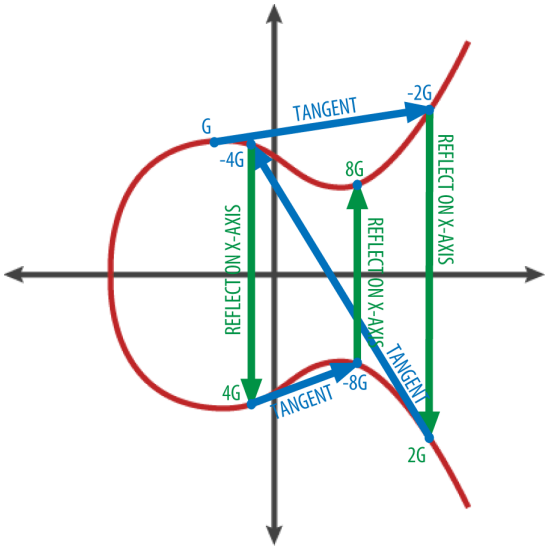


Figure 4. Elliptic curve cryptography: visualizing the multiplication of a point G by an integer k on an elliptic curve

`Tip`: Most bitcoin implementations use the OpenSSL cryptographic library to do the elliptic curve math. For example, to derive the public key, the function EC_POINT_mul() is used.
提示：大多数比特币实现使用`OpenSSL`加密库来做椭圆曲线计算。
例如，使用`EC_POINT_mul()` 函数可以得到得到公钥。

4.2 比特币地址

A bitcoin address is a string of digits and characters that can be shared with anyone who wants to send you money. Addresses produced from public keys consist of a string of numbers and letters, beginning with the digit "1." Here's an example of a bitcoin address:
比特币地址是一个由数字和字母组成的字符串，可以提供给任何要给你转钱的人。
由公钥生成的比特币地址以数字“1”开头。
下面是一个比特币地址的例子。

```
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
```

The bitcoin address is what appears most commonly in a transaction as the "recipient" of the funds. If we compare a bitcoin transaction to a paper check, the bitcoin address is the beneficiary, which is what we write on the line after "Pay to the order of." On a paper check, that beneficiary can sometimes be the name of a bank account holder, but can also include corporations, institutions, or even cash. Because paper checks do not need to specify an account, but rather use an abstract name as the recipient of funds, they are very flexible payment instruments.
比特币地址最长用作交易的资金收款方。
如果把比特币交易比作一张支票，比特币地址就是收款人，也就是我们在收款人一栏写的内容。
支票的收款人可能是银行账户持有者的名字，也可能是公司、机构，甚至现金。
因为支票不需要指定一个账户，而是用一个抽象的名字作为收款方，这使它成为一种灵活的支付工具。

Bitcoin transactions use a similar abstraction, the bitcoin address, to make them very flexible. A bitcoin address can represent the owner of a private/public key pair, or it can represent something else, such as a payment script, as we will see in [p2sh]. For now, let's examine the simple case, a bitcoin address that represents, and is derived from, a public key.
比特币地址使用了类似的抽象（比特币地址），使之非常灵活。
比特币地址可以表示一对公钥/私钥的所有者，也可以表示其它东西，例如支付脚本。
现在，让我们来看一个简单的例子，从一个公钥得到一个比特币地址。

The bitcoin address is derived from the public key through the use of one-way cryptographic hashing. A "hashing algorithm" or simply "hash algorithm" is a one-way function that produces a fingerprint or "hash" of an arbitrary-sized input. Cryptographic hash functions are used extensively in bitcoin: in bitcoin addresses, in script addresses, and in the mining Proof-of-Work algorithm.
使用单向加密哈希，可以从公钥得到比特币地址。
一个哈希算法是一个单向函数，它对一个任意大小的输入产生一个指纹或哈希。
加密哈希函数在比特币中被广泛使用：比特币地址、脚本地址、挖矿的工作量证明算法。

The algorithms used to make a bitcoin address from a public key are the Secure Hash Algorithm (SHA) and the RACE Integrity Primitives Evaluation Message Digest (RIPEMD), specifically SHA256 and RIPEMD160.
用于从公钥生成比特币地址的算法是SHA和RIPEMD，具体地说是SHA256和RIPEMD160。

Starting with the public key *K*, we compute the SHA256 hash and then compute the RIPEMD160 hash of the result, producing a 160-bit (20-byte) number:
以公钥 K作为输入，计算其SHA256哈希，然后对结果再计算RIPEMD160 哈希，得到一个长度为160位（20字节）的数字。

```
A = RIPEMD160(SHA256(K))
```

where *K* is the public key and *A* is the resulting bitcoin address.
K是公钥，A是生成的比特币地址。

Tip：A bitcoin address is *not* the same as a public key. Bitcoin addresses are derived from a public key using a one-way function.
提示：比特币地址不等于公钥。
比特币地址是用单向哈希函数对公钥计算后生成。

Bitcoin addresses are almost always encoded as "Base58Check" (see Base58 and Base58Check Encoding), which uses 58 characters (a Base58 number system) and a checksum to help human readability, avoid ambiguity, and protect against errors in address transcription and entry. Base58Check is also used in many other ways in bitcoin, whenever there is a need for a user to read and correctly transcribe a number, such as a bitcoin address, a private key, an encrypted key, or a script hash. In the next section we will examine the mechanics of Base58Check encoding and decoding and the resulting representations.
比特币地址几乎总是编码为Base58Check，它使用58个字符和一个校验和，用于可读性和避免歧义，并防止地址转录和输入时出现错误。
Base58Check还用在比特币的许多其它方面，在用户需要读取和正确转录一个数字的地方，例如比特币地址、私钥、加密密钥、脚本哈希。
下一节中，我们会详细解释Base58Check的编码和解码，以及它产生的表示。

Public key to bitcoin address: conversion of a public key into a bitcoin address illustrates the conversion of a public key into a bitcoin address.
下图显示了如何从公钥生成比特币地址。

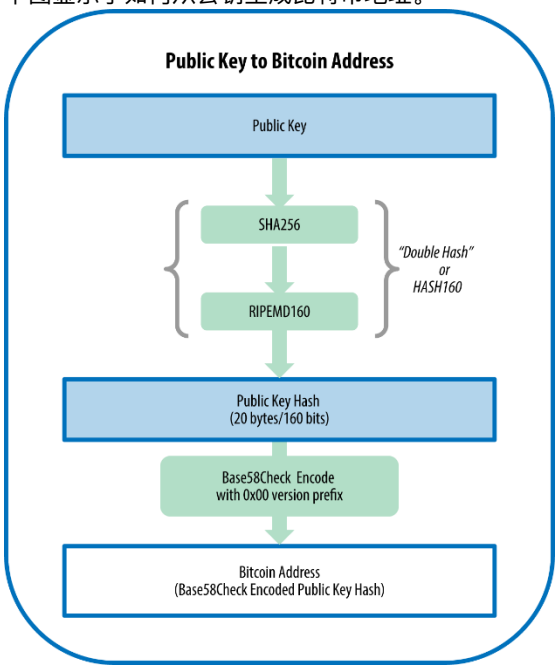


Figure 5. Public key to bitcoin address: conversion of a public key into a bitcoin address
图5：把公钥转换成比特币地址

4.2.1 Base58和Base58Check编码

In order to represent long numbers in a compact way, using fewer symbols, many computer systems use mixed-alphanumeric representations with a base (or radix) higher than 10. For example, whereas the traditional decimal system uses the 10 numerals 0 through 9, the hexadecimal system uses 16, with the letters A through F as the six additional symbols. A number represented in hexadecimal format is shorter than the equivalent decimal representation.
为了用简洁的方式表示长串的数字（使用更少的符号），许多计算机系统使用混合数字和字母来表示大于十进制的数字。

例如，十进制计数使用0–9，十六进制使用了 A–F六个字母。同样一个数字，它的十六进制表示就会比十进制表示更短。

Even more compact, Base64 representation uses 26 lowercase letters, 26 capital letters, 10 numerals, and 2 more characters such as “``” and "/" to transmit binary data over text-based media such as email. Base64 is most commonly used to add binary attachments to email. Base58 is a text-based binary-encoding format developed for use in bitcoin and used in many other cryptocurrencies. It offers a balance between compact representation, readability, and error detection and prevention. Base58 is a subset of Base64, using upper- and lowercase letters and numbers, but omitting some characters that are frequently mistaken for one another and can appear identical when displayed in certain fonts. Specifically, Base58 is Base64 without the 0 (number zero), O (capital o), l (lower L), I (capital i), and the symbols “``” and "/". Or, more simply, it is a set of lowercase and capital letters and numbers without the four (0, O, l, I) just mentioned. Bitcoin’s Base58 alphabet shows the full Base58 alphabet.
Base64使用26个小写字母、26个大写字母、10个数字、2个符号（例如+和/），用于在基于文本媒介（例如email）上传输二进制数据。Base64常用于在email中添加二进制附件。
Base58 是一种基于文本的二进制编码格式，用于比特币和许多其它加密货币。
这种编码格式提供了下列的平衡：简洁表示、可读性、错误检测和预防。
Base58是Base64的一个子集，也使用大小写字母和数字，但舍弃了一些容易读错和在特定字体中容易混淆的字符。具体来说，Base58不含Base64中的0（数字0）、O（大写字母o）、l（小写字母 L）、I（大写字母i），以及+和/这两个字符。简而言之，Base58就是由不包括（0，O，l，I）的大小写字母和数字组成。

Example 2. Bitcoin’s Base58 alphabet
例2：比特币的Base58字母表

```
123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz
```

To add extra security against typos or transcription errors, Base58Check is a Base58 encoding format, frequently used in bitcoin, which has a built-in error-checking code. The checksum is an additional four bytes added to the end of the data that is being encoded. The checksum is derived from the hash of the encoded data and can therefore be used to detect and prevent transcription and typing errors. When presented with Base58Check code, the decoding software will calculate the checksum of the data and compare it to the checksum included in the code. If the two do not match, an error has been introduced and the Base58Check data is invalid. This prevents a mistyped bitcoin address from being accepted by the wallet software as a valid destination, an error that would otherwise result in loss of funds.
为了增加额外的安全性（防止输入和转录错误），Base58Check是一种Base58编码格式，常用于比特币，它有内置的错误检查码。
检验和是额外的四字节，加在被被编码数据的末尾。
校验和是从被编码数据的哈希中得到，所以可以用来检测和防止转录和输入中的错误。
使用Base58check编码时，解码软件计算数据的校验和，并与编码中自带的校验和进行比较。
如果二者不匹配，表明有错误产生，那么这个Base58Check数据就是无效的。
这防止了钱包软件把错误比特币地址当成有效地址，否则可能会造成资金的丢失。

To convert data (a number) into a Base58Check format, we first add a prefix to the data, called the "version byte," which serves to easily identify the type of data that is encoded. For example, in the case of a bitcoin address the prefix is zero (0x00 in hex), whereas the prefix used when encoding a private key is 128 (0x80 in hex). A list of common version prefixes is shown in Base58Check version prefix and encoded result examples.
为了将数据（数字）转换成Base58Check格式，首先给这个数据加上一个前缀（版本字节），这个前缀用来识别被编码的数据的类型。
例如，比特币地址的前缀是0（0x00），私钥的前缀是128（0x80）。
表1列出了一些常见的版本前缀。

Table 1. Base58Check version prefix and encoded result examples
表1：Base58Check版本前缀和编码结果例子

Type	Version prefix (hex)	Base58 result prefix
Bitcoin Address	0x00	1
Pay-to-Script-Hash Address	0x05	3
Bitcoin Testnet Address	0x6F	m or n
Private Key WIF	0x80	5, K, or L
BIP-38 Encrypted Private Key	0x0142	6P
BIP-32 Extended Public Key	0x0488B21E	xpub

Next, we compute the "double-SHA" checksum, meaning we apply the SHA256 hash-algorithm twice on the previous result (prefix and data):
接下来，计算“双SHA”校验和，意味着，要对之前的结果（前缀和数据）运行两次SHA256哈希算法。

```
checksum = SHA256(SHA256(prefix+data))
```

From the resulting 32-byte hash (hash-of-a-hash), we take only the first four bytes. These four bytes serve as the error-checking code, or checksum. The checksum is concatenated (appended) to the end.
在生成的32字节哈希中，我们只取前4个字节。
这4个字节作为错误检测码（或校验和）。
这个校验和被附加到尾部。

The result is composed of three items: a prefix, the data, and a checksum. This result is encoded using the Base58 alphabet described previously. Base58Check encoding: a Base58, versioned, and checksummed format for unambiguously encoding bitcoin data illustrates the Base58Check encoding process.
这个结果由三部分组成：前缀、数据、校验和。
这个结果使用之前描述的Base58字母表编码。
下图显示了Base58Check编码过程。

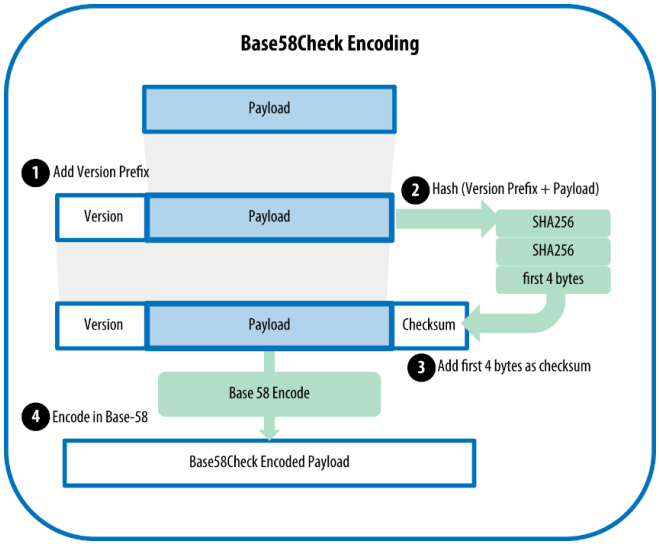


Figure 6. Base58Check encoding: a Base58, versioned, and checksummed format for unambiguously encoding bitcoin data

In bitcoin, most of the data presented to the user is Base58Check-encoded to make it compact, easy to read, and easy to detect errors. The version prefix in Base58Check encoding is used to create easily distinguishable formats, which when encoded in Base58

contain specific characters at the beginning of the Base58Check-encoded payload. These characters make it easy for humans to identify the type of data that is encoded and how to use it. This is what differentiates, for example, a Base58Check-encoded bitcoin address that starts with a 1 from a Base58Check-encoded private key WIF that starts with a 5. Some example version prefixes and the resulting Base58 characters are shown in Base58Check version prefix and encoded result examples.
在比特币中，给用户展示的多数数据是Base58Check编码，使之更简洁、易读、容易检测错误。
Base58Check编码中的版本前缀用于创建易于辨别的格式，它在Base58Check编码负载前面包含特定字符。
这些属性使用户可以轻松明确被编码的数据的类型，以及如何使用它们。
例如，Base58Check编码的比特币地址是以1开头，而Base58Check编码的私钥WIF是以5开头。
图1显示了一些例子版本前缀和生成的Base58字符。

4.2.2 密钥的格式

Both private and public keys can be represented in a number of different formats. These representations all encode the same number, even though they look different. These formats are primarily used to make it easy for people to read and transcribe keys without introducing errors.
公钥和私钥的都可以用一些不同的格式来表示。
这些格式都编码了相同的数字，虽然它们看起来不同。
这些格式主要用于使人们容易读，在转录密钥时避免引入错误。

4.2.2.1私钥的格式

The private key can be represented in a number of different formats, all of which correspond to the same 256-bit number. Private key representations (encoding formats) shows three common formats used to represent private keys. Different formats are used in different circumstances. Hexadecimal and raw binary formats are used internally in software and rarely shown to users. The WIF is used for import/export of keys between wallets and often used in QR code (barcode) representations of private keys.
私钥可以用许多不同的格式表示，都对应相同的256位数字。
表2显示了私钥的三种常见格式。不同的格式用于不同的场景。
十六进制和二进制格式用在软件内部，很少展示给用户。
WIF格式用在钱包之间导入和导出密钥，经常用于私钥的二维码（条形码）表示。

Table 2. Private key representations (encoding formats)

Type	Prefix	Description
Raw	None	32 bytes
Hex	None	64 hexadecimal digits
WIF	5	Base58Check encoding: Base58 with version prefix of 128- and 32-bit checksum
WIF-compressed	K or L	As above, with added suffix 0x01 before encoding

Example: Same key, different formats shows the private key generated in these three formats.
例子：显示了这三种格式生成的私钥

Table 3. Example: Same key, different formats

Format	Private key
Hex	1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
WIF-compressed	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

All of these representations are different ways of showing the same number, the same private key. They look different, but any one format can easily be converted to any other format. Note that the "raw binary" is not shown in Example: Same key, different formats as any encoding for display here would, by definition, not be raw binary data.
这些表示法都是用来表示相同的数字、相同的私钥。
虽然看起来不同，但不同格式之间很容易转换。

We use the wif-to-ec command from Bitcoin Explorer (see [appdx_bx]) to show that both WIF keys represent the same private key:
我们用Bitcoin Explorer中的wif-to-ec命令（参阅[appdx_bx]）来显示两个WIF键代表相同的私钥。

```
$ bx wif-to-ec 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd

$ bx wif-to-ec KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
```

4.2.3从Base58Check解码

The Bitcoin Explorer commands (see [appdx_bx]) make it easy to write shell scripts and command-line "pipes" that manipulate bitcoin keys, addresses, and transactions. You can use Bitcoin Explorer to decode the Base58Check format on the command line.
Bitcoin Explorer命令（参见[appdx_bx]）使得编写shell脚本和命令行管道变得容易，这些方式可以处理比特币密钥、地址和交易。
你可以使用Bitcoin Explorer来解码命令行上的Base58Check格式。

We use the base58check-decode command to decode the uncompressed key:
我们使用base58check-decode命令解码未压缩的密钥：

```
$ bx base58check-decode 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
wrapper
{
    checksum 4286807748
    payload 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
    version 128
}
```

The result contains the key as payload, the WIF version prefix 128, and a checksum.
结果包含的密钥有：负载、WIF版本前缀128、校验和。

Notice that the "payload" of the compressed key is appended with the suffix 01, signalling that the derived public key is to be compressed:
注意，压缩密钥的负载附加了后缀01，表示导出的公钥要被压缩：

```
$ bx base58check-decode KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
wrapper
{
```

```
    checksum 2339607926
    payload
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01
    version 128
}
```

将十六进制编码为Base58Check

To encode into Base58Check (the opposite of the previous command), we use the base58check-encode command from Bitcoin Explorer (see [appdx_bx]) and provide the hex private key, followed by the WIF version prefix 128:
要转换成Base58Check（与上一个命令相反），我们使用base58check-encode命令，提供十六进制私钥和WIF版本前缀128：

```
bx base58check-encode
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd --version
128
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

将十六进制（压缩密钥）编码为Base58Check

To encode into Base58Check as a "compressed" private key (see Compressed private keys), we append the suffix 01 to the hex key and then encode as in the preceding section:
要将压缩格式的私钥编码为Base58Check（参见“压缩格式私钥”一节），我们需在十六进制私钥的后面添加后缀01，然后使用跟上面一样的方法：

```
$ bx base58check-encode
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01 --version
128
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

The resulting WIF-compressed format starts with a "K." This denotes that the private key within has a suffix of "01" and will be used to produce compressed public keys only (see Compressed public keys).
生成的WIF压缩格式以“K”开头，这表示私钥有后缀“01”，将之用于生成压缩的公钥（参见“压缩格式公钥”一节）。

4.2.3.1公钥的格式

Public keys are also presented in different ways, usually as either *compressed* or *uncompressed* public keys.
公钥也可以用多种格式来表示，通常是压缩或未压缩公钥。

As we saw previously, the public key is a point on the elliptic curve consisting of a pair of coordinates (x,y). It is usually presented with the prefix 04 followed by two 256-bit numbers: one for the *x* coordinate of the point, the other for the *y* coordinate. The prefix 04 is used to distinguish uncompressed public keys from compressed public keys that begin with a 02 or a 03.
我们从前文可知，公钥是在椭圆曲线上的一个点，由坐标（x,y）组成。
公钥通常表示为：前缀04，紧接着两个256比特的数字。
一个数字是x坐标，另一个数字是y坐标。
前缀04表示未压缩公钥，前缀02或03表示压缩公钥。

Here’s the public key generated by the private key we created earlier, shown as the coordinates x and y:
下面是由前文中的私钥所生成的公钥，显示为坐标x和y：

```
x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

Here's the same public key shown as a 520-bit number (130 hex digits) with the prefix 04 followed by x and then y coordinates, as 04 x y:
下面是同样的公钥，表示为520比特的数字（130个十六进制数字），
这个520比特的数字以前缀04开头，紧接着是x及y 坐标，组成格式为04 x y。

```
K =  04F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
      07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

4.2.3.2压缩公钥

Compressed public keys were introduced to bitcoin to reduce the size of transactions and conserve disk space on nodes that store the bitcoin blockchain database. Most transactions include the public key, which is required to validate the owner's credentials and spend the bitcoin. Each public key requires 520 bits (prefix + x + y), which when multiplied by several hundred transactions per block, or tens of thousands of transactions per day, adds a significant amount of data to the blockchain.
引入压缩公钥是为了减少交易的字节数，节省节点的磁盘空间。
多数交易包含了公钥，用于验证用户的凭据和支付比特币。
每个公钥有520比特，如果每个区块有数百个交易，每天有成千上万的交易发生，区块链里就会被写入大量的数据。

As we saw in the section Public Keys, a public key is a point (x,y) on an elliptic curve. Because the curve expresses a mathematical function, a point on the curve represents a solution to the equation and, therefore, if we know the *x* coordinate we can calculate the *y* coordinate by solving the equation $y^2 \bmod p = (x^3 + 7) \bmod p$. That allows us to store only the *x* coordinate of the public key point, omitting the *y* coordinate and reducing the size of the key and the space required to store it by 256 bits. An almost 50% reduction in size in every transaction adds up to a lot of data saved over time!
正如我们在"公钥"一节所见，公钥是椭圆曲线上的一个点(x,y)。
因为曲线表达了一个数学函数，曲线上的一个点表示这个方程的一个解，
因此，如果我们知道x坐标，就能计算y坐标，方法是求：$y^2 \bmod p = (x^3 + 7) \bmod p$。
这样，只用保存x坐标，忽略y坐标，就能减少密钥的大小，只需要256位来保存。
每个交易就可以减少一半的空间，这样就可以省去大量要存储的数据。

Whereas uncompressed public keys have a prefix of 04, compressed public keys start with either a 02 or a 03 prefix. Let's look at why there are two possible prefixes: because the left side of the equation is y^2, the solution for *y* is a square root, which can have a positive or negative value. Visually, this means that the resulting *y* coordinate can be above or below the x-axis. As you can see from the graph of the elliptic curve in An elliptic curve, the curve is symmetric, meaning it is reflected like a mirror by the x-axis. So, while we can omit the *y* coordinate we have to store the *sign* of *y* (positive or negative); or in other words, we have to remember if it was above or below the x-axis because each of those options represents a different point and a different public key.
未压缩公钥有前缀04，而压缩公钥的前缀是02或03。
需要两种前缀的原因是：方程的左侧是y^2，解y是平方根，可以有有一个正值或负值。
形象地说，y坐标可以在x轴上面或下面。
从图4-2的椭圆曲线图中可以看出，曲线从x轴看是对称的。
因此，如果我们略去y坐标，就必须储存y的符号（正或负）。
换句话说，对于给定的x值，我们需要知道y值在x轴的上面还是下面，因为它们代表椭圆曲线上不同的点，即不同的公钥。

When calculating the elliptic curve in binary arithmetic on the finite field of prime order p, the *y* coordinate is either even or odd, which corresponds to the positive/negative sign as explained earlier. Therefore, to distinguish between the two possible values of *y*, we store a compressed public key with the prefix 02 if the *y* is even, and 03 if it is odd, allowing the software to correctly deduce the *y* coordinate from the *x* coordinate and uncompress the public key to the full coordinates of the point. Public key compression is illustrated in Public key compression.

当我们使用二进制算术在素数p的有限域上计算椭圆曲线时，y坐标是奇数或偶数，对应前面讲正负。
因此，为了区分y的两种可能值，我们在生成压缩公钥时，如果y是偶数，使用前缀02；如果y是奇数，使用前缀03。
这样，软件就可以正确地从x坐标得出y坐标，从而把公钥解压为点的完整坐标。
下图阐释了公钥压缩。

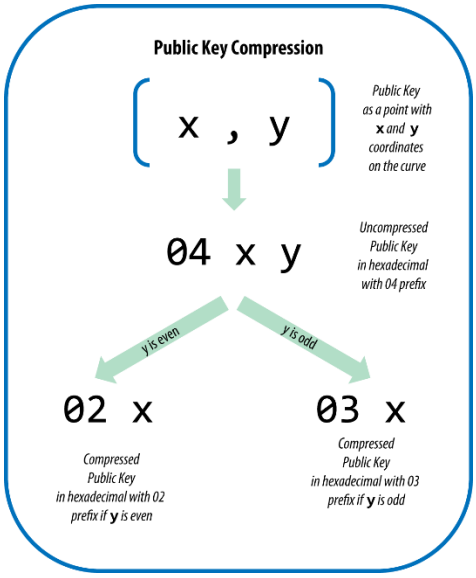


Figure 7. Public key compression

Here's the same public key generated previously, shown as a compressed public key stored in 264 bits (66 hex digits) with the prefix 03 indicating the *y* coordinate is odd:
下面是前面产生的相同的公钥，显示为264比特（66个十六进制数字）的压缩公钥，前缀是03，表示y坐标是奇数。

```
K = 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
```

This compressed public key corresponds to the same private key, meaning it is generated from the same private key. However, it looks different from the uncompressed public key. More importantly, if we convert this compressed public key to a bitcoin address using the double-hash function (RIPEMD160(SHA256(K))) it will produce a *different* bitcoin address. This can be confusing, because it means that a single private key can produce a public key expressed in two different formats (compressed and uncompressed) that produce two different bitcoin addresses. However, the private key is identical for both bitcoin addresses.
这个压缩公钥对应同一私钥，即，它是由同一私钥所生成。
但是，它与未压缩公钥看起来不同。
更重要的是，如果用RIPEMD160(SHA256(K))将压缩公钥转化成比特币地址，会产生不同的比特币地址。

这让人迷惑，因为这意味着，一个私钥可以生成两种不同格式的公钥（压缩格式和未压缩格式），而生成两个不同的比特币地址。但是，这两个比特币地址对应同一个私钥。

Compressed public keys are gradually becoming the default across bitcoin clients, which is having a significant impact on reducing the size of transactions and therefore the blockchain. However, not all clients support compressed public keys yet. Newer clients that support compressed public keys have to account for transactions from older clients that do not support compressed public keys. This is especially important when a wallet application is importing private keys from another bitcoin wallet application, because the new wallet needs to scan the blockchain to find transactions corresponding to these imported keys. Which bitcoin addresses should the bitcoin wallet scan for? The bitcoin addresses produced by uncompressed public keys, or the bitcoin addresses produced by compressed public keys? Both are valid bitcoin addresses, and can be signed for by the private key, but they are different addresses!

压缩公钥渐渐成为了比特币客户端的默认格式，它可以大大减少交易的大小，节省磁盘存储空间。
但是，并非所有客户端都支持压缩公钥。
支持压缩公钥的新客户端需要考虑不支持压缩公钥的老客户端。
在这种情况下特别重要：一个钱包应用从另一个钱包应用中导入私钥。
因为新钱包需要扫描区块链来找到对应这个私钥的交易。
比特币钱包应该扫描哪些比特币地址？未压缩公钥产生的比特币地址，还是压缩公钥产生的比特币地址？
两种都是有效的比特币地址，都可以用私钥签名，但是它们是不同的比特币地址。

To resolve this issue, when private keys are exported from a wallet, the WIF that is used to represent them is implemented differently in newer bitcoin wallets, to indicate that these private keys have been used to produce *compressed* public keys and therefore *compressed* bitcoin addresses. This allows the importing wallet to distinguish between private keys originating from older or newer wallets and search the blockchain for transactions with bitcoin addresses corresponding to the uncompressed, or the compressed, public keys, respectively. Let's look at how this works in more detail, in the next section.
为了解决这个问题，当从钱包中导出私钥时，新钱包中的WIF的实现是不同的，
以指示这些私钥被用于生成压缩公钥和压缩地址。
这使导入钱包能区分源于老钱包和新钱包的私钥，并分别使用对应于未压缩或压缩公钥的比特币地址来搜索区块链中的交易。我们将在下一节详细解释。

4.2.3.3压缩私钥

Ironically, the term "compressed private key" is a misnomer, because when a private key is exported as WIF-compressed it is actually one byte *longer* than an "uncompressed" private key. That is because the private key has an added one-byte suffix (shown as 01 in hex in Example: Same key, different formats), which signifies that the private key is from a newer wallet and should only be used to produce compressed public keys. Private keys are not themselves compressed and cannot be compressed. The term "compressed private key" really means "private key from which only compressed public keys should be derived," whereas "uncompressed private key" really means "private key from which only uncompressed public keys should be derived." You should only refer to the export format as "WIF-compressed" or "WIF" and not refer to the private key itself as "compressed" to avoid further confusion
“压缩私钥”是一个误称，因为当私钥被导出为WIF压缩格式时，实际比未压缩私钥还多了一个字节。
这是因为私钥增加了一个自己的后缀（图4中显示为01），用以表示该私钥是来自于新钱包，应该用于生成压缩公钥。
私钥本身未被压缩，也不能被压缩。“压缩私钥”表示“从这个私钥只应导出压缩公钥”，而“非压缩私钥”表示“从这个私钥只应导出未压缩公钥”。
为避免进一步的混乱，应该只说导出格式是“WIF压缩”或“WIF”，而不能说私钥本身是“压缩”的。

Example: Same key, different formats shows the same key, encoded in WIF and WIF-compressed formats.
图4显示了同一密钥，以WIF和WIF压缩格式编码。

Table 4. Example: Same key, different formats
表4：示例，相同的密钥，不同的格式

Format	Private key
Hex	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Hex-compressed	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD01
WIF-compressed	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

Notice that the hex-compressed private key format has one extra byte at the end (01 in hex). While the Base58 encoding version prefix is the same (0x80) for both WIF and WIF-

compressed formats, the addition of one byte on the end of the number causes the first character of the Base58 encoding to change from a 5 to either a *K* or *L*. Think of this as the Base58 equivalent of the decimal encoding difference between the number 100 and the number 99. While 100 is one digit longer than 99, it also has a prefix of 1 instead of a prefix of 9. As the length changes, it affects the prefix. In Base58, the prefix 5 changes to a *K* or *L* as the length of the number increases by one byte.
注意，十六进制压缩私钥格式在末尾有一个额外的字节（0x01）。
虽然对于WIF和WIF压缩格式来说，Base58编码版本前缀都是相同的（0x80），但在数字末尾添加一个字节会导致Base58编码的第一个字符从5变为K或 L。
把它看作是数字100和数字99之间十进制编码差异的Base58等价物。
100比99的位数长，它有一个前缀1，而不是前缀9。当长度变化时，它会影响了前缀。
在Base58中，前缀5改变为K或L，因为数字的长度增加一个字节。

Remember, these formats are *not* used interchangeably. In a newer wallet that implements compressed public keys, the private keys will only ever be exported as WIF-compressed (with a *K* or *L* prefix). If the wallet is an older implementation and does not use compressed public keys, the private keys will only ever be exported as WIF (with a 5 prefix). The goal here is to signal to the wallet importing these private keys whether it must search the blockchain for compressed or uncompressed public keys and addresses.
记住，这些格式并不是可互换使用的。
在实现了压缩公钥的新钱包中，私钥只会导出为WIF压缩（以K或L为前缀）。
对于老的没有实现压缩公钥的钱包，私钥将只会导出为WIF（以5为前缀）。
这样做的目的就是为了告诉钱包，导入的这些私钥是要搜索区块链中的压缩或未压缩公钥和地址。

If a bitcoin wallet is able to implement compressed public keys, it will use those in all transactions. The private keys in the wallet will be used to derive the public key points on the curve, which will be compressed. The compressed public keys will be used to produce bitcoin addresses and those will be used in transactions. When exporting private keys from a new wallet that implements compressed public keys, the WIF is modified, with the addition of a one-byte suffix 01 to the private key. The resulting Base58Check-encoded private key is called a "compressed WIF" and starts with the letter *K* or *L*, instead of starting with "5" as is the case with WIF-encoded (noncompressed) keys from older wallets.
如果比特币钱包实现了压缩公钥，那么它会在所有交易中使用压格公钥。
钱包中的私钥将用来导入曲线上的公钥点，这个公钥是压缩的。
压缩公钥被用来生成比特币地址，并用于交易中。
当从一个实现了压缩公钥的新钱包中导出私钥时，WIF被修改，为私钥增加一个自己的后缀01。
生成的Base58Check编码私钥被称作“压缩WIF”，以字母K或L开头，而以5开头的是来自老钱包的WIF（未压缩）密钥。

Tip: "Compressed private keys" is a misnomer! They are not compressed; rather, WIF-compressed signifies that the keys should only be used to derive compressed public keys and their corresponding bitcoin addresses. Ironically, a "WIF-compressed" encoded private key is one byte longer because it has the added 01 suffix to distinguish it from an "uncompressed" one.
提示：“压缩私钥”是一个误称！
私钥没有被压缩。WIF压缩表示这个私钥只应用于生成压缩公钥和对应的比特币地址。
“WIF压缩”私钥还多出了一个字节，因为多了后缀“01”，用来区分“非压缩私钥”和“压缩私钥”。

4.3 用C++实现密钥和地址

Let's look at the complete process of creating a bitcoin address, from a private key, to a public key (a point on the elliptic curve), to a double-hashed address, and finally, the Base58Check encoding. The C++ code in Creating a Base58Check-encoded bitcoin address from a private key shows the complete step-by-step process, from private key to Base58Check-encoded bitcoin address. The code example uses the libbitcoin library introduced in [alt_libraries] for some helper functions.
我们来看看创建一个比特币地址的完整过程，从私钥到公钥（椭圆曲线上的一个点），再到双哈希地址，最后是BASE58CHECK编码。
C++代码显示了每一步的过程，从私钥到Base58Chech编码比特币地址。
这个代码例子使用了[alt librraries]中介绍的libbitcion库。

Example 3. Creating a Base58Check-encoded bitcoin address from a private key
例3：从私钥生成一个Base58Check编码的比特币地址

```
link:code/addr.cpp[]
```

The code uses a predefined private key to produce the same bitcoin address every time it is run, as shown in Compiling and running the addr code.
每次运行时，这个代码使用了一个预定义的私钥，来生成相同的比特币地址。

Example 4. Compiling and running the addr code
例4：编译和运行addr代码

```
# Compile the addr.cpp code
$ g++ -o addr addr.cpp -std=c++11 $(pkg-config --cflags --libs libbitcoin)

# Run the addr executable
$ ./addr
Public key:
0202a406624211f2abbdc68da3df929f938c3399dd79fac1b51b0e4ad1d26a47aa
Address    : 1PRTTaJesdNovgne6Ehcdu1fpEdX7913CK
```

Tip：The code in Compiling and running the addr code produces a bitcoin address (1PRTT...) from a *compressed* public key (see Compressed public keys). If you used the uncompressed public key instead, it would produce a different bitcoin address (14K1y...).
提示：这个代码使用一个压缩公钥生成一个比特币地址（1PRTT…）。
如果你使用未压缩公钥，它会生成一个不同的比特币地址（14K1y…）。

4.4用Python实现密钥和地址

The most comprehensive bitcoin library in Python is pybitcointools by Vitalik Buterin. In Key and address generation and formatting with the pybitcointools library, we use the pybitcointools library (imported as "bitcoin") to generate and display keys and addresses in various formats.
Python的最全面的比特币库是 Vitalik Buterin编写的 pybitcointools。
在例5中，我们使用这个库（导入为bitcoin），来生成和显示各种格式的密钥和地址。

Example 5. Key and address generation and formatting with the pybitcointools library
例5 ： 用pybitcointools库生成和格式化密钥和地址

```
link:code/key-to-address-ecc-example.py[]
```

Running key-to-address-ecc-example.py shows the output from running this code.
例6显示了运行这个代码的输出。

Example 6. Running key-to-address-ecc-example.py
例6：运行例子代码。

```
$ python key-to-address-ecc-example.py
Private Key (hex) is:
 3aba4162c7251c891207b747840551a71939b0de081f85c4e44cf7c13e41daa6
Private Key (decimal) is:
 26563230048437957592232553826663696440606756685920117476832299673293013768870
Private Key (WIF) is:
 5JG9hT3beGTJuUAmCQEmNaxAuMacCTfXuw1R3FCXig23RQHMr4K
Private Key Compressed (hex) is:
 3aba4162c7251c891207b747840551a71939b0de081f85c4e44cf7c13e41daa601
Private Key (WIF-Compressed) is:
 KyBsPXxTuVD82av65KZkrGrWi5qLMah5SdNq6uftawDbgKa2wv6S
Public Key (x,y) coordinates is:
 (41637322786646325214887832269588396900663353932545912953362782457239403430124L,
 16388935128781238405526710466724741593761085120864331449066658622400339362166L)
Public Key (hex) is:
 045c0de3b9c8ab18dd04e3511243ec2952002dbfadc864b9628910169d9b9b00ec↵
243bcefdd4347074d44bd7356d6a53c495737dd96295e2a9374bf5f02ebfc176
Compressed Public Key (hex) is:
 025c0de3b9c8ab18dd04e3511243ec2952002dbfadc864b9628910169d9b9b00ec
Bitcoin Address (b58check) is:
 1thMirt546nngXqyPEz532S8fLwbozud8
Compressed Bitcoin Address (b58check) is:
 14cxpo3MBCYYWCgF74SWTdcmxipnGUsPw3
```

A script demonstrating elliptic curve math used for bitcoin keys is another example, using the Python ECDSA library for the elliptic curve math and without using any specialized bitcoin libraries.
例7是另一个例子，使用Python ECDSA库来做椭圆曲线计算，没有使用任何专门的比特币库。

Example 7. A script demonstrating elliptic curve math used for bitcoin keys
例7：一个脚本，演示了用于比特币密钥的椭圆曲线计算。

```
link:code/ec-math.py[]
```

Installing the Python ECDSA library and running the ec_math.py script shows the output produced by running this script.
例8显示了运行这个脚本的输出。

Note：A script demonstrating elliptic curve math used for bitcoin keys uses os.urandom, which reflects a cryptographically secure random number generator (CSRNG) provided by the underlying operating system. Caution: Depending on the OS, os.urandom may *not* be

implemented with sufficient security or seeded properly and may *not* be appropriate for generating production-quality bitcoin keys.
注意：例7使用了os.urandom，它反应了底层操作系统提供的一个加密安全随机数生成器（CSRNG）。当心：依赖于操作系统，os.urandom的实现可能不够安全或正确的种子，对于生成实际应用的比特币密钥可能不合适。

Example 8. Installing the Python ECDSA library and running the ec_math.py script
例8：安装Python ECDSA库，运行ec_math.py脚本

```
$ # Install Python PIP package manager
$ sudo apt-get install python-pip
$ # Install the Python ECDSA library
$ sudo pip install ecdsa
$ # Run the script
$ python ec-math.py
Secret:
380908350159543588624811326288874439059062049959123782780601687035806602940
0
EC point:
(700488535318671794898577504976069662723825834713229354546245955400072693126
27,
105262206478686743191060800263479589329920209527285803935736021686045542353
380)
BTC public key:
029ade3effb0a67d5c8609850d797366af428f4a0d5194cb221d807770a1522873
```

4.5 高级密钥和地址

In the following sections we will look at advanced forms of keys and addresses, such as encrypted private keys, script and multisignature addresses, vanity addresses, and paper wallets.
在以下章节中，我们看看密钥和地址的高级形式，例如加密私钥、脚本和多签名地址、靓号地址、纸钱包。

4.4.1 加密私钥（BIP-38）

Private keys must remain secret. The need for *confidentiality* of the private keys is a truism that is quite difficult to achieve in practice, because it conflicts with the equally important security objective of *availability*. Keeping the private key private is much harder when you need to store backups of the private key to avoid losing it. A private key stored in a wallet that is encrypted by a password might be secure, but that wallet needs to be backed up. At times, users need to move keys from one wallet to another—to upgrade or replace the wallet software, for example.
私钥必须保密。
私钥机密性是老生常谈，在实践中很难实现，因为它与可用性是矛盾的，而可用性也是重要的安全目标。
当你需要为了避免私钥丢失而存储备份时，保持私钥机密更加困难。
保存在钱包中的私钥被密码加密保存，可能是安全的，但钱包需要备份。
有时，用户因为要升级或重装钱包软件，需要把密钥从一个钱包转移到另一个钱包。

Private key backups might also be stored on paper (see Paper Wallets) or on external storage media, such as a USB flash drive. But what if the backup itself is stolen or lost? These conflicting security goals led to the introduction of a portable and convenient standard for encrypting private keys in a way that can be understood by many different wallets and bitcoin clients, standardized by BIP-38 (see [appdxbitcoinimpproposals]).
私钥备份也可以存在纸上（参见“纸钱包”）或外部存储介质，比如U盘。
但如果备份失窃或丢失呢?
这些矛盾的安全目标导致了为加密私钥引入了一个可携带和方便的标准，许多不同的钱包和比特币客户端都能理解它，这个标准就是BIP-38。

BIP-38 proposes a common standard for encrypting private keys with a passphrase and encoding them with Base58Check so that they can be stored securely on backup media, transported securely between wallets, or kept in any other conditions where the key might be exposed. The standard for encryption uses the Advanced Encryption Standard (AES), a standard established by the NIST and used broadly in data encryption implementations for commercial and military applications.
BIP-38提出了一个通用标准：使用一个口令来加密私钥，使用Base58Check对其进行编码，这样，就可以在备份媒介上安全地存储它们，在钱包之间安全地传输它们，或者在密钥可能被保留的地方保存它。
加密标准使用了AES，这是NIST制定的标准，并广泛应用于商业和军事应用的数据加密。

A BIP-38 encryption scheme takes as input a bitcoin private key, usually encoded in the WIF, as a Base58Check string with the prefix of "5." Additionally, the BIP-38 encryption scheme takes a passphrase—a long password—usually composed of several words or a complex string of alphanumeric characters. The result of the BIP-38 encryption scheme is a Base58Check-encoded encrypted private key that begins with the prefix 6P. If you see a key that starts with 6P, it is encrypted and requires a passphrase in order to convert (decrypt) it back into a WIF-formatted private key (prefix 5) that can be used in any wallet. Many wallet applications now recognize BIP-38-encrypted private keys and will prompt the user for a passphrase to decrypt and import the key. Third-party applications, such as the incredibly useful browser-based Bit Address (Wallet Details tab), can be used to decrypt BIP-38 keys.
BIP-38加密方案是：输入一个比特币私钥，通常是WIF编码，base58chek字符串的前缀“5”。

此外，BIP-38加密方案需要一个口令（长密码），通常由多个单词或一段复杂的数字字母字符串组成。
BIP-38加密方案的结果是一个由 base58check编码的加密私钥，前缀为6P。
如果你看到一个密钥的前缀是6P，这就意味着，它被加密了，需要一个口令来解密成WIF格式的私钥（前缀是5），使得任何钱包都可以使用它。
许多钱包应用现在都识别 BIP-38加密私钥，会提示用户提供口令，以解码和导入密钥。
第三方应用可用于解密BIP-38密钥，例如基于浏览器的Bit Address。

The most common use case for BIP-38 encrypted keys is for paper wallets that can be used to back up private keys on a piece of paper. As long as the user selects a strong passphrase, a paper wallet with BIP-38 encrypted private keys is incredibly secure and a great way to create offline bitcoin storage (also known as "cold storage").
BIP-38加密密钥最长用于纸钱包，它把私钥卸载纸上。
只要用户选择了强口令，BIP-38加密私钥就非常安全，这是创建离线比特币存储的一个好方法（冷存储）。

Test the encrypted keys in Example of BIP-38 encrypted private key using bitaddress.org to see how you can get the decrypted key by entering the passphrase.
在表5中，使用bitaddress.org来测试加密密钥，看看输入口令如何得到解密密钥。

Table 5. Example of BIP-38 encrypted private key

Private Key (WIF)	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Passphrase	MyTestPassphrase
Encrypted Key (BIP-38)	6PRTHL6mWa48xSopbU1cKrVjpKbBZxcLRRCdctLJ3z5yxE87MobKoXdTsJ

4.4.2 P2SH 和多签名地址

As we know, traditional bitcoin addresses begin with the number "1" and are derived from the public key, which is derived from the private key. Although anyone can send bitcoin to a "1" address, that bitcoin can only be spent by presenting the corresponding private key signature and public key hash.
我们知道，传统的比特币地址以数字1开头，它源于公钥，而公钥源于私钥。
虽然任何人都可以将比特币发送到一个1开头的地址，但比特币只能通过提供相应的私钥签名和公钥哈希后才能花费。

Bitcoin addresses that begin with the number "3" are pay-to-script hash (P2SH) addresses, sometimes erroneously called multisignature or multisig addresses. They designate the beneficiary of a bitcoin transaction as the hash of a script, instead of the owner of a public key. The feature was introduced in January 2012 with BIP-16 (see [appdxbitcoinimpproposals]), and is being widely adopted because it provides the opportunity to add functionality to the address itself. Unlike transactions that "send" funds to traditional "1" bitcoin addresses, also known as a pay-to-public-key-hash (P2PKH), funds sent to "3" addresses require something more than the presentation of one public key hash and one private key signature as proof of ownership. The requirements are designated at the time the address is created, within the script, and all inputs to this address will be encumbered with the same requirements.
以数字3开头的比特币地址是P2SH地址，有时被误称为多签名地址。
它们指定比特币交易的收款人是一个脚本的哈希，而不是公钥的所有者。
这个特性在2012年1月由BIP-16引入，目前已被广泛采用，因为它为地址增加了功能。
不同于P2PKH交易发送资金到1开头的比特币地址，当资金被发送到3开头的地址时，需要的不仅仅是一个公钥哈希和一个私钥签名作为所有者证明。
在创建地址时在脚本中指定了要求，这个地址的所有输入将被用相同的要求来负担。

A P2SH address is created from a transaction script, which defines who can spend a transaction output (for more details, see [p2sh]). Encoding a P2SH address involves using

the same double-hash function as used during creation of a bitcoin address, only applied on the script instead of the public key:
一个P2SH地址用一个交易脚本创建的，它定义了谁能花费一个交易输出。
对一个P2SH地址进行编码涉及使用相同的双哈希函数（与创建一个比特币地址使用的相同），只应用于脚本，而不是应用于公钥。

```
script hash = RIPEMD160(SHA256(script))
```

The resulting "script hash" is encoded with Base58Check with a version prefix of 5, which results in an encoded address starting with a 3. An example of a P2SH address is 3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM, which can be derived using the Bitcoin Explorer commands script-encode, sha256, ripemd160, and base58check-encode (see [appdx_bx]) as follows:
产生的脚本哈希是用Base58Check编码，有一个版本前缀5，它会生成3开头的编码地址。
P2SH地址的一个例子是 3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM，
可以使用Bitcoin Explorer的下列命令获得它。

```
$ echo 'DUP HASH160 [89abcdefabbaabbaabbaabbaabbaabbaabbaabba] EQUALVERIFY CHECKSIG' >
script

$ bx script-encode < script | bx sha256 | bx ripemd160 | bx base58check-encode --version 5
3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM
```

Tip：P2SH is not necessarily the same as a multisignature standard transaction. A P2SH address *most often* represents a multi-signature script, but it might also represent a script encoding other types of transactions.
提示：P2SH 不一定就是多签名标准交易。
虽然P2SH地址最常表示一个多签名脚本，但它还可以表示一个编码其它类型交易的脚本。

4.4.2.1 多签名地址和P2SH

Currently, the most common implementation of the P2SH function is the multi-signature address script. As the name implies, the underlying script requires more than one signature to prove ownership and therefore spend funds.
目前，P2SH功能最常见的实现是多签名地址脚本。
顾名思义，脚本要求多个签名来证明所有权，此后才能花费资金。

The bitcoin multi-signature feature is designed to require M signatures (also known as the "threshold") from a total of N keys, known as an M-of-N multisig, where M is equal to or less than N. For example, Bob the coffee shop owner from [ch01_intro_what_is_bitcoin] could use a multisignature address requiring 1-of-2 signatures from a key belonging to him and a key belonging to his spouse, ensuring either of them could sign to spend a transaction output locked to this address. This would be similar to a "joint account" as implemented in traditional banking where either spouse can spend with a single signature. Or Gopesh, the web designer paid by Bob to create a website, might have a 2-of-3 multisignature address for his business that ensures that no funds can be spent unless at least two of the business partners sign a transaction.
比特币多签名特性是：总共有N个密钥，要求M个签名，称为M-N多签名，其中M<=N。
例如，Bob可以使用一个多签名地址，要求1-2签名，一个是他的密钥，另一个是同伴的密钥，以确保有一个人可以签署消费锁定到这个地址的输出。
这类似于银行中的一个"联合账户"，其中任何一方都可以单独签单消费。
或者，Bob雇佣Gopesh来创建一个网站，Gopesh可能有一个2-3多签名地址，确保至少有两个业务伙伴签署了一个交易，才可以花费资金。

We will explore how to create transactions that spend funds from P2SH (and multi-signature) addresses in [transactions].
我们将在第五章节了解如何创建花费P2SH（和多签名）地址中的资金的交易。

4.4.3 靓号地址

Vanity addresses are valid bitcoin addresses that contain human-readable messages. For example, 1LoveBPzzD72PUXLzCkYAtGFYmK5vYNR33 is a valid address that contains the letters forming the word "Love" as the first four Base-58 letters. Vanity addresses require generating and testing billions of candidate private keys, until a bitcoin address with the desired pattern is found. Although there are some optimizations in the vanity generation algorithm, the process essentially involves picking a private key at random, deriving the public key, deriving the bitcoin address, and checking to see if it matches the desired vanity pattern, repeating billions of times until a match is found.
靓号地址是有效的比特币地址，它包含了可读信息。
例如，1LoveBPzzD72PUXLzCkYAtGFYmK5vYNR33是一个有效地址，前四个字符包含了Love。
靓号地址需要生成和测试数十亿的候选私钥，直到找到一个有希望模式的比特币地址。
虽然在靓号生成算法中有一些优化，但该方法还是要涉及：随机选择一个私钥、生成公钥、生成比特币地址，检查是否与所要的靓号模式相匹配。重复数十亿次，直到找到一个匹配的。

Once a vanity address matching the desired pattern is found, the private key from which it was derived can be used by the owner to spend bitcoin in exactly the same way as any other address. Vanity addresses are no less or more secure than any other address. They depend on the same Elliptic Curve Cryptography (ECC) and SHA as any other address. You can no more easily find the private key of an address starting with a vanity pattern than you can any other address.
一旦找到一个匹配地址，花费方式就与其它地址相同。
靓号地址并不比其它地址具有更安全或更不安全。
它们依赖相同的ECC（椭圆曲线密码学）和SHA。
你无法比任何别的地址更容易的获得一个靓号模式开头的地址的私钥。

In [ch01_intro_what_is_bitcoin], we introduced Eugenia, a children's charity director operating in the Philippines. Let's say that Eugenia is organizing a bitcoin fundraising drive and wants to use a vanity bitcoin address to publicize the fundraising. Eugenia will create a vanity address that starts with "1Kids" to promote the children's charity fundraiser. Let's see how this vanity address will be created and what it means for the security of Eugenia's charity.
在第一章介绍过，Eugenia是在菲律宾工作的一位儿童慈善总监。
假设Eugenia组织了一场比特币募捐活动，他希望使用靓号比特币地址来宣传这个募捐活动。
Eugenia创建了一个以1Kids开头的靓号地址来促进儿童慈善募捐的活动。
我们来看看这个靓号地址如何被创建，这个靓号地址对Eugenia慈善募捐的安全性又意味着什么。

4.4.3.1 生成靓号地址

It's important to realize that a bitcoin address is simply a number represented by symbols in the Base58 alphabet. The search for a pattern like "1Kids" can be seen as searching for an address in the range from 1Kids111111111111111111111111111111 to 1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzzz. There are approximately 58^{29} (approximately $1.4 * 10^{51}$) addresses in that range, all starting with "1Kids." The range of vanity addresses starting with "1Kids" shows the range of addresses that have the prefix 1Kids.
重要的是认识到，比特币地址不过是由Base58字母代表的一个数字。
搜索“1kids”开头的模式可看做搜索从1Kids111111111111111111111111111111到1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzzz的地址。
这些以“1kid”开头的地址范围中大约有58^{29}个。
表6显示了有前缀“1kids”的地址。

Table 6. The range of vanity addresses starting with "1Kids"

From	1Kids111111111111111111111111111111
	1Kids111111111111111111111111111112
	1Kids111111111111111111111111111113
	...
To	1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzzz

Let's look at the pattern "1Kids" as a number and see how frequently we might find this pattern in a bitcoin address (see The frequency of a vanity pattern (1KidsCharity) and average search time on a desktop PC). An average desktop computer PC, without any specialized hardware, can search approximately 100,000 keys per second.
我们把模式"1Kids"看做一个数字，看看多快能在比特币地址中找到这个模式。
如果是一台普通桌面电脑，没有任何特殊的硬件，每秒可以搜索10万个密钥。

Table 7. The frequency of a vanity pattern (1KidsCharity) and average search time on a desktop PC
表7：桌面PC上搜索靓号模式的速度

Length	Pattern	Frequency	Average search time
1	1K	1 in 58 keys	< 1 milliseconds
2	1Ki	1 in 3,364	50 milliseconds
3	1Kid	1 in 195,000	< 2 seconds
4	1Kids	1 in 11 million	1 minute
5	1KidsC	1 in 656 million	1 hour
6	1KidsCh	1 in 38 billion	2 days
7	1KidsCha	1 in 2.2 trillion	3–4 months
8	1KidsChar	1 in 128 trillion	13–18 years
9	1KidsChari	1 in 7 quadrillion	800 years
10	1KidsCharit	1 in 400 quadrillion	46,000 years
11	1KidsCharity	1 in 23 quintillion	2.5 million years

As you can see, Eugenia won't be creating the vanity address "1KidsCharity" anytime soon, even if she had access to several thousand computers. Each additional character increases the difficulty by a factor of 58. Patterns with more than seven characters are usually found by specialized hardware, such as custom-built desktops with multiple GPUs. These are often repurposed bitcoin mining "rigs" that are no longer profitable for bitcoin mining but can be used to find vanity addresses. Vanity searches on GPU systems are many orders of magnitude faster than on a general-purpose CPU.
正如你所见，Eugenia不会很快地创建出以"1KidsCharity"开头的靓号地址，即使她有数千台的计算机同时进行运算。

每增加一个字符就会增加58倍的计算难度。超过七个字符的模式通常需要专用的硬件才能找出，例如用户定制的有多个GPU的台式机。那些通常是无法继续在比特币挖矿中盈利的矿机，被重新赋予了寻找靓号地址的任务。用GPU系统搜索靓号的速度比用通用CPU要快很几个量级。

Another way to find a vanity address is to outsource the work to a pool of vanity miners, such as the pool at Vanity Pool. A pool is a service that allows those with GPU hardware to earn bitcoin searching for vanity addresses for others. For a small payment (0.01 bitcoin or approximately $5 at the time of this writing), Eugenia can outsource the search for a seven-character pattern vanity address and get results in a few hours instead of having to run a CPU search for months.
另一种寻找靓号地址的方法是将工作外包给一个矿池里的靓号矿工们。
一个矿池是一种允许那些 GPU硬件通过为他人寻找靓号地址来获得比特币的服务。
对小额支付（本文写作时大约是5美元），Eugenia可以将搜索7位字符模式的靓号地址的工作外包，在几个小时内就可以得到结果，而不必用一个CPU搜索上几个月才得到结果。

Generating a vanity address is a brute-force exercise: try a random key, check the resulting address to see if it matches the desired pattern, repeat until successful. Vanity address miner shows an example of a "vanity miner," a program designed to find vanity addresses, written in C++. The example uses the libbitcoin library, which we introduced in [alt_libraries].
生成一个靓号地址是一个暴力搜索过程：尝试一个随机密钥，检查生成的地址是否和所需的模式相匹配，重复这个过程直到成功找到为止。
例9是个靓号矿工的例子，这是用C++程序写的，寻找靓号地址的程序。
这个例子运用到了libbitcoin库。

Example 9. Vanity address miner
例9：靓号地址矿工

```
link:code/vanity-miner.cpp[]
```

Note：Compiling and running the vanity-miner example uses std::random_device. Depending on the implementation it may reflect a CSRNG provided by the underlying operating system. In the case of a Unix-like operating system such as Linux, it draws from /dev/urandom. The random number generator used here is for demonstration purposes, and it is *not* appropriate for generating production-quality bitcoin keys as it is not implemented with sufficient security.
说明：例10使用了std::random_device。
根据实施情况，可能会反映底层操作系统提供的CSRNG。
在Unix操作系统（如Linux）中，它来自/dev/urandom。
这里使用的随机数字生成器用于演示，并不适用于生成实际使用的比特币密钥，因为它不够安全。

The example code must be compiled using a C++ compiler and linked against the libbitcoin library (which must be first installed on that system). To run the example, run the vanity-miner executable with no parameters (see Compiling and running the vanity-miner example) and it will attempt to find a vanity address starting with "1kid."
示例代码需要用C++编译器，链接libbitcoin库（需要提前装入该系统）。
为了运行这个例子，运行不带参数的vanity-miner，它就会尝试找到以“1kid”开头的靓号地址。

Example 10. Compiling and running the vanity-miner example
例10：编译并运行vanity-miner程序例子

```
$ # Compile the code with g++
$ g++ -o vanity-miner vanity-miner.cpp $(pkg-config --cflags --libs libbitcoin)

$ # Run the example
$ ./vanity-miner
Found vanity address! 1KiDzkG4MxmovZryZRj8tK81oQRhbZ46YT
Secret: 57cc268a05f83a23ac9d930bc8565bac4e277055f4794cbd1a39e5e71c038f3f

$ # Run it again for a different result
```

```
$ ./vanity-miner
Found vanity address! 1Kidxr3wsmMzzouwXibKfwTYs5Pau8TUFn
Secret: 7f65bbbbe6d8caae74a0c6a0d2d7b5c6663d71b60337299a1a2cf34c04b2a623

# Use "time" to see how long it takes to find a result
$ time ./vanity-miner
Found vanity address! 1KidPWhKgGRQWD5PP5TAnGfDyfWp5yceXM
Secret: 2a802e7a53d8aa237cd059377b616d2bfcfa4b0140bc85fa008f2d3d4b225349

real      0m8.868s
user      0m8.828s
sys       0m0.035s
```

The example code will take a few seconds to find a match for the three-character pattern "kid," as we can see when we use the time Unix command to measure the execution time. Change the search pattern in the source code and see how much longer it takes for four- or five-character patterns!
正如我们运行Unix命令所测出的运行时间所示，示例代码要花几秒钟来找出匹配“kid”模式的结果。
你可以尝试在源代码中改变搜索模式，看看如果是四个字符或五个字符的搜索模式需要花多久时间！

4.4.3.2 靓号地址安全性

Vanity addresses can be used to enhance *and* to defeat security measures; they are truly a double-edged sword. Used to improve security, a distinctive address makes it harder for adversaries to substitute their own address and fool your customers into paying them instead of you. Unfortunately, vanity addresses also make it possible for anyone to create an address that *resembles* any random address, or even another vanity address, thereby fooling your customers.
靓号地址既可以增加和削弱安全措施，它们着实是一把双刃剑。
用于改善安全性时，一个与众不同的地址使对手更难用他们自己的地址来欺骗你的客户，向他们支付，而不是向你支付。
不幸的是，靓号地址也可能使得任何人都能创建一个类似于随机地址的地址，甚至另一个靓号地址，从而欺骗你的客户。

Eugenia could advertise a randomly generated address (e.g., 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy) to which people can send their donations. Or, she could generate a vanity address that starts with 1Kids, to make it more distinctive.
Eugenia可以让捐款人捐款到她宣布的一个随机生成地址。
或者她可以生成一个以“1Kids”开头的靓号地址以显得更独特。

In both cases, one of the risks of using a single fixed address (rather than a separate dynamic address per donor) is that a thief might be able to infiltrate your website and replace it with his own address, thereby diverting donations to himself. If you have advertised your donation address in a number of different places, your users may visually inspect the address before making a payment to ensure it is the same one they saw on your website, on your email, and on your flyer. In the case of a random address like 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy, the average user will perhaps inspect the first few characters "1J7mdg" and be satisfied that the address matches. Using a vanity address generator, someone with the intent to steal by substituting a similar-looking address can quickly generate addresses that match the first few characters, as shown in Generating vanity addresses to match a random address.
在这两种情况下，使用单一固定地址（而不是每笔捐款用一个单独的动态地址）的风险之一是，小偷有可能会攻入你的网站，用他自己的地址取代你的地址，从而将捐赠转移给他自己。
如果你在不同的地方公布了你的捐款地址，你的用户可以在付款之前检查以确保这个地址跟在你的网站、邮件和传单上看到的地址是同一个。
在随机地址的情况下，普通用户可能会只检查头几个字符“1j7mdg”，就认为地址匹配。
使用靓号地址生成器，那些想通过替换类似地址来盗窃的人可以快速生成与前几个字符相匹配的地址，如表8所示。

Table 8. Generating vanity addresses to match a random address

Original Random Address	1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
Vanity (4-character match)	1J7md1QqU4LpctBetHS2ZoyLV5d6dShhEy
Vanity (5-character match)	1J7mdgYqyNd4ya3UEcq31Q7sqRMXw2XZ6n
Vanity (6-character match)	1J7mdg5WxGENmwyJP9xuGhG5KRzu99BBCX

So does a vanity address increase security? If Eugenia generates the vanity address 1Kids33q44erFfpeXrmDSz7zEqG2FesZEN, users are likely to look at the vanity pattern word *and a few characters beyond*, for example noticing the "1Kids33" part of the address. That would force an attacker to generate a vanity address matching at least six characters (two more), expending an effort that is 3,364 times (58 × 58) higher than the effort Eugenia expended for her 4-character vanity. Essentially, the effort Eugenia expends (or pays a vanity pool for) "pushes" the attacker into having to produce a longer pattern vanity. If Eugenia pays a pool to generate an 8-character vanity address, the attacker would be pushed into the realm of 10 characters, which is infeasible on a personal computer and expensive even with a custom vanity-mining rig or vanity pool. What is affordable for Eugenia becomes unaffordable for the attacker, especially if the potential reward of fraud is not high enough to cover the cost of the vanity address generation.

那靓号地址会不会增加安全性?

如果Eugenia生成一个靓号地址，用户可能看到靓号模式的字母和一些字符在上面，例如在地址部分中注明了1Kids33。这样就会迫使攻击者生成至少6个字母相匹配的的靓号地址（比之前多2个字符），就要花费比Eugenia多3364倍的努力。

本质上，Eugenia付出的努力（或靓号池付出的）迫使攻击者不得不生成更长的靓号图案。

如果Eugenia花钱请矿池生成8个字符的靓号地址，攻击者将会被迫到生成10个字符，那将是个人电脑，甚至昂贵的定制矿机或靓号池也无法生成的。

对Eugenia来说，可以承担这笔支出，对攻击者来说，则无法承担这笔支出，特别是如果欺诈的潜在回报不足以支付生成靓号地址所需的费用。

4.4.4 纸钱包

Paper wallets are bitcoin private keys printed on paper. Often the paper wallet also includes the corresponding bitcoin address for convenience, but this is not necessary because it can be derived from the private key. Paper wallets are a very effective way to create backups or offline bitcoin storage, also known as "cold storage." As a backup mechanism, a paper wallet can provide security against the loss of key due to a computer mishap such as a hard-drive failure, theft, or accidental deletion. As a "cold storage" mechanism, if the paper wallet keys are generated offline and never stored on a computer system, they are much more secure against hackers, keyloggers, and other online computer threats.
纸钱包是打印在纸上的比特币私钥。
经常，为了方便，纸钱包也包括对应的比特币地址，但这不是必须的，因为地址可以从私钥中导出。
纸钱包是一个非常有效的建立备份或离线存储比特币（即冷存储）的方法。
作为备份机制，纸钱包可以提供安全性，防止在电脑硬盘损坏、失窃或意外删除的情况下造成密钥丢失。
作为一个冷存储机制，如果纸钱包密钥在线下生成，并永远不在电脑系统中存储，他们在应对黑客攻击、键盘记录器，或其他在线电脑威胁时更加安全。

Paper wallets come in many shapes, sizes, and designs, but at a very basic level are just a key and an address printed on paper. Simplest form of a paper wallet—a printout of the bitcoin address and private key shows the simplest form of a paper wallet.
纸钱包有许多不同的形状、大小和设计，但非常基本的是在纸上打印一个密钥和一个地址。
表9展现了最简单的纸钱包。

Table 9. Simplest form of a paper wallet—a printout of the bitcoin address and private key

Public address	Private key (WIF)
1424C2F4bC9JidNjjTUZCbUxv6Sa1Mt62x	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn

Paper wallets can be generated easily using a tool such as the client-side JavaScript generator at *bitaddress.org*. This page contains all the code necessary to generate keys and paper wallets, even while completely disconnected from the internet. To use it, save the HTML page on your local drive or on an external USB flash drive. Disconnect from the internet and open the file in a browser. Even better, boot your computer using a pristine operating system, such as a CD-ROM bootable Linux OS. Any keys generated with this tool while offline can be printed on a local printer over a USB cable (not wirelessly), thereby creating paper wallets whose keys exist only on the paper and have never been stored on any online system. Put these paper wallets in a fireproof safe and "send" bitcoin to their bitcoin address, to implement a simple yet highly effective "cold storage" solution. An example of a simple paper wallet from bitaddress.org shows a paper wallet generated from the bitaddress.org site.
通过使用工具，可以很容易地生成纸钱包，例如bitaddress.org的客户端JavaScript生成器。
这个页面包含了生成密钥和纸钱包的所有代码，甚至在完全断开网络连接的情况下，也可以使用。
若要使用它，先将HTML页面保存在本地磁盘或外部U盘。
断开互联网，从浏览器中打开文件。
更好的是，使用一个原始操作系统启动电脑，例如光盘启动的Linux系统。
任何在脱机情况下使用这个工具所生成的密钥，都可以通过USB线（不是无线）在本地打印机上打印出来，从而创建了纸钱包，它的密钥只在纸上，不会存储在任何在线系统上。
将这些纸钱包放置在保险柜中，发送比特币到对应的比特币地址，从而实现了一个简单但非常有效的冷存储解决方案。图8展示了通过bitaddress.org 生成的纸钱包。

Figure 8. An example of a simple paper wallet from bitaddress.org
图8：简单纸钱包的例子

The disadvantage of a simple paper wallet system is that the printed keys are vulnerable to theft. A thief who is able to gain access to the paper can either steal it or photograph the keys and take control of the bitcoin locked with those keys. A more sophisticated paper wallet storage system uses BIP-38 encrypted private keys. The keys printed on the paper wallet are protected by a passphrase that the owner has memorized. Without the passphrase, the encrypted keys are useless. Yet, they still are superior to a passphrase-protected wallet because the keys have never been online and must be physically retrieved from a safe or other physically secured storage. An example of an encrypted paper wallet from bitaddress.org. The passphrase is "test." shows a paper wallet with an encrypted private key (BIP-38) created on the bitaddress.org site.
这个简单的纸钱包系统的缺点是，被打印的密钥容易被盗窃。
能够接近这些纸的小偷只需偷走纸或拍摄纸上的密钥，就能控制被这些密钥锁定的比特币。
一个更复杂的纸钱包存储系统使用BIP-38加密的私钥。
打印在纸钱包上的这些私钥被其所有者记住的一个口令保护起来。
没有口令，这些被加密过的密钥也是毫无用处的。
但它们仍旧优于用口令保护的钱包，因为这些密钥从没有在线过，并且必须从保险箱或其他物理的安全存储中得到。图9展示了通过bitaddress.org 生成的加密纸钱包。

Figure 9. An example of an encrypted paper wallet from bitaddress.org. The passphrase is "test."
图9：加密纸钱包的例子，口令是“test”

Warning: Although you can deposit funds into a paper wallet several times, you should withdraw all funds only once, spending everything. This is because in the process of unlocking and spending funds some wallets might generate a change address if you spend less than the whole amount. Additionally, if the computer you use to sign the transaction is compromised, you risk exposing the private key. By spending the entire balance of a paper wallet only once, you reduce the risk of key compromise. If you need only a small amount, send any remaining funds to a new paper wallet in the same transaction.
警告：虽然你可以多次存款到纸钱包中，但是你最好一次性提取里面所有的资金。
因为如果你提取的金额少于总额的话，有些钱包可能会生成一个找零地址。
并且，你所用的计算机可能被病毒感染，那就有可能泄露私钥。

一次性提走所有余款可以减少私钥泄露的风险，如果你只需要一笔小金额，那么就把余额发送到另一个新的纸钱包中。

Paper wallets come in many designs and sizes, with many different features. Some are intended to be given as gifts and have seasonal themes, such as Christmas and New Year's themes. Others are designed for storage in a bank vault or safe with the private key hidden in some way, either with opaque scratch-off stickers, or folded and sealed with tamper-proof adhesive foil. Figures #paper_wallet_bpw through #paper_wallet_spw show various examples of paper wallets with security and backup features.
纸钱包有许多设计和大小，并有许多不同的特性。
有些作为礼物送给他人，有季节性的主题，像圣诞节和新年主题。
另外一些则是设计保存在银行金库或保险箱内，用某种方式隐藏了私钥，可以是用不透明的刮刮贴，或折叠和密封的防篡改的铝箔。图10~12展示了几个具有安全和备份特性的纸钱包。

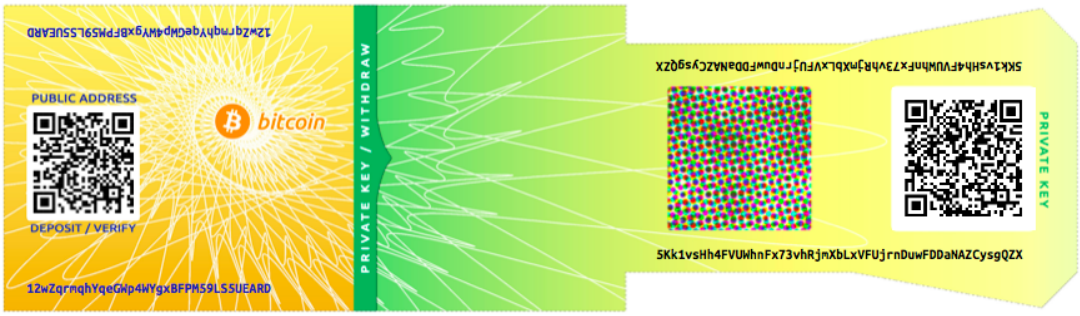


Figure 10. An example of a paper wallet from bitcoinpaperwallet.com with the private key on a folding flap
图10：来自bitcoinpaperwallet.com的纸钱包例子，在折叠式襟翼上有私钥。

Figure 11. The bitcoinpaperwallet.com paper wallet with the private key concealed
图11：bitcoinpaperwallet.com纸钱包，有被隐藏的私钥

Other designs feature additional copies of the key and address, in the form of detachable stubs similar to ticket stubs, allowing you to store multiple copies to protect against fire, flood, or other natural disasters.
其它设计增加了密钥和地址的额外副本，以可拆卸的存根的形式，类似于票根存根，允许你存储多个副本，以防止火灾、洪水或其它自然灾害。

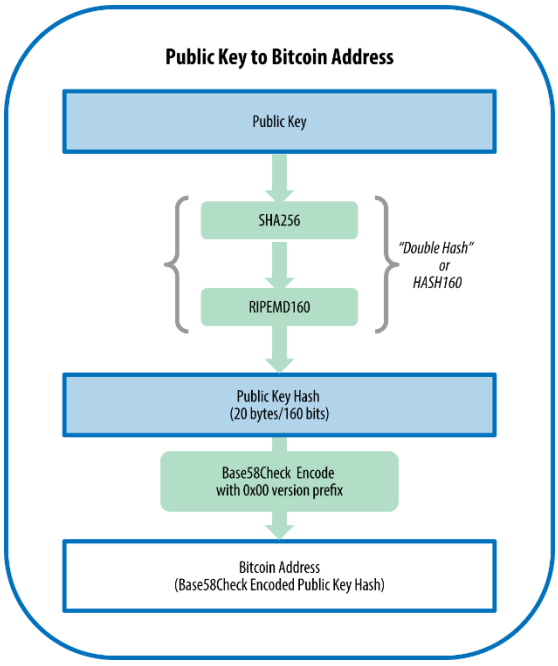
Figure 12. An example of a paper wallet with additional copies of the keys on a backup "stub"

图12：纸钱包例子，在备份stub中有额外的密钥备份

4.6本章总结（ddk）

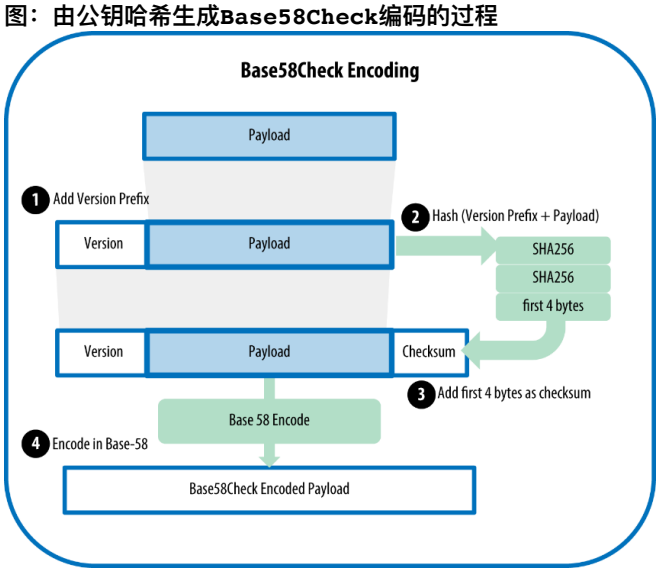
私钥k：256位（32字节）
椭圆曲线算法：$K = k * G$　　// G是椭圆曲线上的一个固定点
公钥k：椭圆曲线上的点坐标(x,y)　// x和y都是256位

图：由公钥生成比特币地址的过程



公钥哈希：RIPEMD160(SHA256(K))
比特币地址：Base58Check编码

图：由公钥哈希生成Base58Check编码的过程



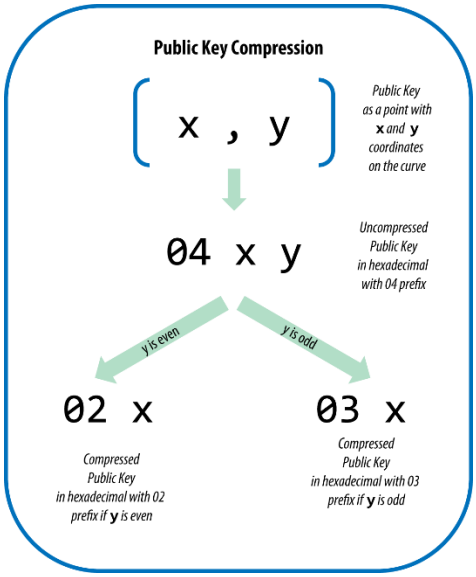
表：Base58Check版本前缀和编码结果前缀

Type	Version prefix (hex)	Base58 result prefix
Bitcoin Address	0x00	1
Pay-to-Script-Hash Address	0x05	3
Bitcoin Testnet Address	0x6F	m or n
Private Key WIF	0x80	5, K, or L
BIP-38 Encrypted Private Key	0x0142	6P
BIP-32 Extended Public Key	0x0488B21E	xpub

表：私钥表示

Type	Prefix	Description
Raw	None	32 bytes
Hex	None	64 hexadecimal digits
WIF	5	Base58Check encoding: Base58 with version prefix of 128- and 32-bit checksum
WIF-compressed	K or L	As above, with added suffix 0x01 before encoding

图：公钥表示



公钥通常表示为：（04，x，y）
前缀04表示未压缩公钥，前缀02或03表示压缩公钥。

表：相同的私钥，不同的格式

Format	Private key
Hex	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Hex-compressed	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD01
WIF-compressed	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

表：BIP-38加密私钥（使用AES）

Private Key (WIF)	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Passphrase	MyTestPassphrase
Encrypted Key (BIP-38)	6PRTHL6mWa48xSopbU1cKrVjpKbBZxcLRRCdctLJ3z5yxE87MobKoXdTsJ