你可能听说过比特币是基于密码学，这一在计算机安全中广泛使用的数学分支。 密码学在希腊语中是“秘密写作”的意思，但密码学这门科学不仅只包含被称之为秘密写作的加密学。 密码学也可以用来证明秘密的知识，而不会泄露秘密（数字签名），或证明数据的真实性（数字指纹）。 这些类型的加密证明是比特币中关键的数学工具并在比特币应用程序中被广泛使用。具有讽刺意味的是，加密不是比特币的重要组成部分，因为它的通信和交易数据没有加密，也不需要加密来保护资金。 在本章中，我们将介绍一些在比特币中用来控制资金的所有权的密码学，包括密钥，地址和钱包。

4.1简介

比特币的所有权是通过数字密钥、比特币地址和数字签名来确定的。数字密钥实际上并不存储在网络中，而是由用户生成之后，存储在一个叫做钱包的文件或简单的数据库中。存储在用户钱包中的数字密钥完全独立于比特币协议，可由用户的钱包软件生成并管理，而无需参照区块链或访问网络。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 有权认证和基于密码学证明的安全模型。

大多数比特币交易都需要一个有效的签名才会被存储在区块链。只有有效的密钥才能产生有效的数字签名，因此拥有~密钥副本就拥有了对该帐户的比特币的控制权。用于支出资金的数字签名也称为见证（witness），密码术中使用的术语。 比特币交易中的见证数据证明了所用资金的真正归谁所有。

密钥是成对出现的，由一个私钥和一个公钥所组成。公钥就像银行的帐号，而私钥就像控制账户的PIN码或支票的签名。比特币的用户很少会直接看到数字密钥。一般情况下，它们被存储在钱包文件内，由比特币钱包软件进行管理。

在比特币交易的支付环节，收件人的公钥是通过其数字指纹代表的，称为比特币地址，就像支票上的支付对象的名字 （即“收款方”）。一般情况下，比特币地址由一个公钥生成并对应于这个公钥。然而，并非所有比特币地址都是公钥； 他们也可以代表其他支付对象，譬如脚本，我们将在本章后面提及。这样一来，比特币地址把收款方抽象起来了，使得 交易的目的地更灵活，就像支票一样：这个支付工具可支付到个人账户、公司账户，进行账单支付或现金支付。比特币地址是用户经常看到的密钥的唯一代表，他们只需要把比特币地址告诉其他人即可。

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。然后我们将了解密钥如何被产生、存储和管理。我们将回顾用于代表私钥和公 钥、地址和脚本地址的各种编码格式。最后，我们将讲解密钥和地址的高级用途：比特币靓号，多重签名以及脚本地址和纸钱包。

4.1.1 公钥加密和加密货币

公钥加密发明于20世纪70年代。它是计算机和信息安全的数学基础。

自从公钥加密被发明之后，一些合适的数学函数被发现，譬如：素数幂和椭圆曲线乘法。这些数学函数都是不可逆的， 就是说很容易向一个方向计算，但不可以向相反方向倒推。基于这些数学函数的密码学，使得生成数字密钥和不可伪造 的数字签名成为可能。比特币正是使用椭圆曲线乘法作为其公钥加密的基础。

在比特币系统中，我们用公钥加密创建一个密钥对，用于控制比特币的获取。密钥对包括一个私钥，和由其衍生出的唯 一的公钥。公钥用于接收比特币，而私钥用于比特币支付时的交易签名。

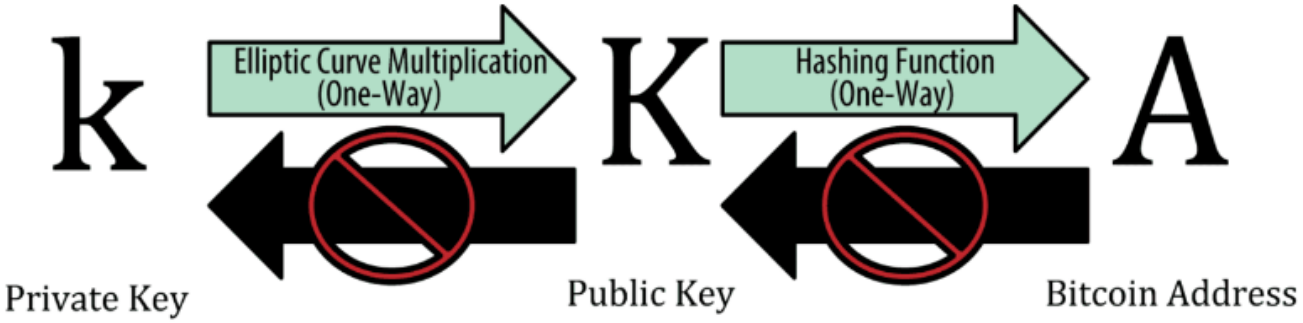
公钥和私钥之间的数学关系，使得私钥可用于生成特定消息的签名。此签名可以在不泄露私钥的同时对公钥进行验证。

支付比特币时，比特币的当前所有者需要在交易中提交其公钥和签名（每次交易的签名都不同，但均从同一个私钥生 成）。比特币网络中的所有人都可以通过所提交的公钥和签名进行验证，并确认该交易是否有效，即确认支付者在该时刻对所交易的比特币拥有所有权。

提示 大多数比特币钱包工具为了方便会将私钥和公钥以密钥对的形式存储在一起。然而，公钥可以由私钥计算得到， 所以只存储私钥也是可以的。

4.1.2 私钥和公钥

一个比特币钱包中包含一系列的密钥对，每个密钥对包括一个私钥和一个公钥。私钥（k）是一个数字，通常是随机选出的。有了私钥，我们就可以使用椭圆曲线乘法这个单向加密函数产生一个公钥（K）。有了公钥（K），我们就可以使 用一个单向加密哈希函数生成比特币地址（A）。在本节中，我们将从生成私钥开始，讲述如何使用椭圆曲线运算将私 钥生成公钥，并最终由公钥生成比特币地址。私钥、公钥和比特币地址之间的关系如下图所示。



为什么使用非对称加密（公钥/私钥）？

为什么在比特币中使用非对称密码术？ 它不是用于“加密”（make secret）交易。 相反，非对称密码学的有用属性是生成数字签名的能力。 可以将私钥应用于交易的数字指纹以产生数字签名。 该签名只能由知晓私钥的人生成。 但是，访问公钥和交易指纹的任何人都可以使用它们来验证签名。 这种非对称密码学的适用性使得任何人都可以验证每笔交易的每个签名，同时确保只有私钥的所有者可以产生有效的签名。

4.1.3 私钥

私钥就是一个随机选出的数字而已。一个比特币地址中的所有资金的控制取决于相应私钥的所有权和控制权。在比特币交易中，私钥用于生成支付比特币所必需的签名以证明对资金的所有权。私钥必须始终保持机密，因为一旦被泄露给第三 方，相当于该私钥保护之下的比特币也拱手相让了。私钥还必须进行备份，以防意外丢失，因为私钥一旦丢失就难以复原，其所保护的比特币也将永远丢失。

提示 比特币私钥只是一个数字。你可以用硬币、铅笔和纸来随机生成你的私钥：掷硬币256次，用纸和笔记录正反面并转换为0和1，随机得到的256位二进制数字可作为比特币钱包的私钥。该私钥可进一步生成公钥。

从一个随机数生成私钥 生成密钥的第一步也是最重要的一步，是要找到足够安全的熵源，即随机性来源。生成一个比特币私钥在本质上与“在1 到2^256之间选一个数字”无异。只要选取的结果是不可预测或不可重复的，那么选取数字的具体方法并不重要。比特币软件使用操作系统底层的随机数生成器来产生256位的熵（随机性）。通常情况下，操作系统随机数生成器由人工的随机源进行初始化，这就是为什么也可能需要不停晃动鼠标几秒钟。

更准确地说，私钥可以是1和n-1之间的任何数字，其中n是一个常数（n=1.158 * 10^77，略小于2^256），并被定义为由比特币所使用的椭圆曲线的阶（见椭圆曲线密码学解释）。要生成这样的一个私钥，我们随机选择一个256位的数字，并检查它是否小于n-1。从编程的角度来看，一般是通过在一个密码学安全的随机源中取出一长串随机字节，对其使用SHA256哈希算法进行运算，这样就可以方便地产生一个256位的数字。如果运算结果小于n-1，我们就有了一个合适的私钥。否则，我们就用另一个随机数再重复一次。

警告 不要自己写代码或使用你的编程语言提供的简易随机数生成器来获得一个随机数。使用密码学安全的伪随机数生成器（CSPRNG），并且需要有一个来自具有足够熵值的源的种子。使用随机数发生器的程序库时，需仔细研读其文档，以确保它是加密安全的。正确实施CSPRNG是密钥安全性的关键所在。

以下是一个随机生成的私钥（k），以十六进制格式表示（256位的二进制数，以64位十六进制数显示，每个十六进制数占4位）：

1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD

提示比特币私钥空间的大小是2^256，这是一个非常大的数字。用十进制表示的话，大约是10^77，而可见宇宙被估计只含有10^80个原子。

要使用比特币核心客户端生成一个新的密钥，可使用 getnewaddress 命令。出于安全考虑，命令运行后只 显示生成的公钥，而不显示私钥。如果要bitcoind显示私钥，可以使用 dumpprivkey 命令。 dumpprivkey 命令会把私钥以 Base58校验和编码格式显示，这种私钥格式被称为钱包导入格式（WIF，Wallet Import Format），在“私钥的格式”一节有详细讲解。下面给出了使用这两个命令生成和显示私钥的例子：

$ bitcoin-cli getnewaddress 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy $ bitcoin-cli dumpprivkey 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

dumpprivkey 命令打开钱包提取由 getnewaddress 命令生成的私钥。除非密钥对都存储在钱包里，否则bitcoind 的并不能从公钥得知私钥。 dumpprivkey 命令才有效。

提示 dumpprivkey命令无法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也就是由getnewaddress命令生成的私钥。

您还可以使用Bitcoin Explorer命令行工具（请参阅附录中的[appdx_bx]）使用命令seed，ec-new和ec-to-wif生成和显示私钥：

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

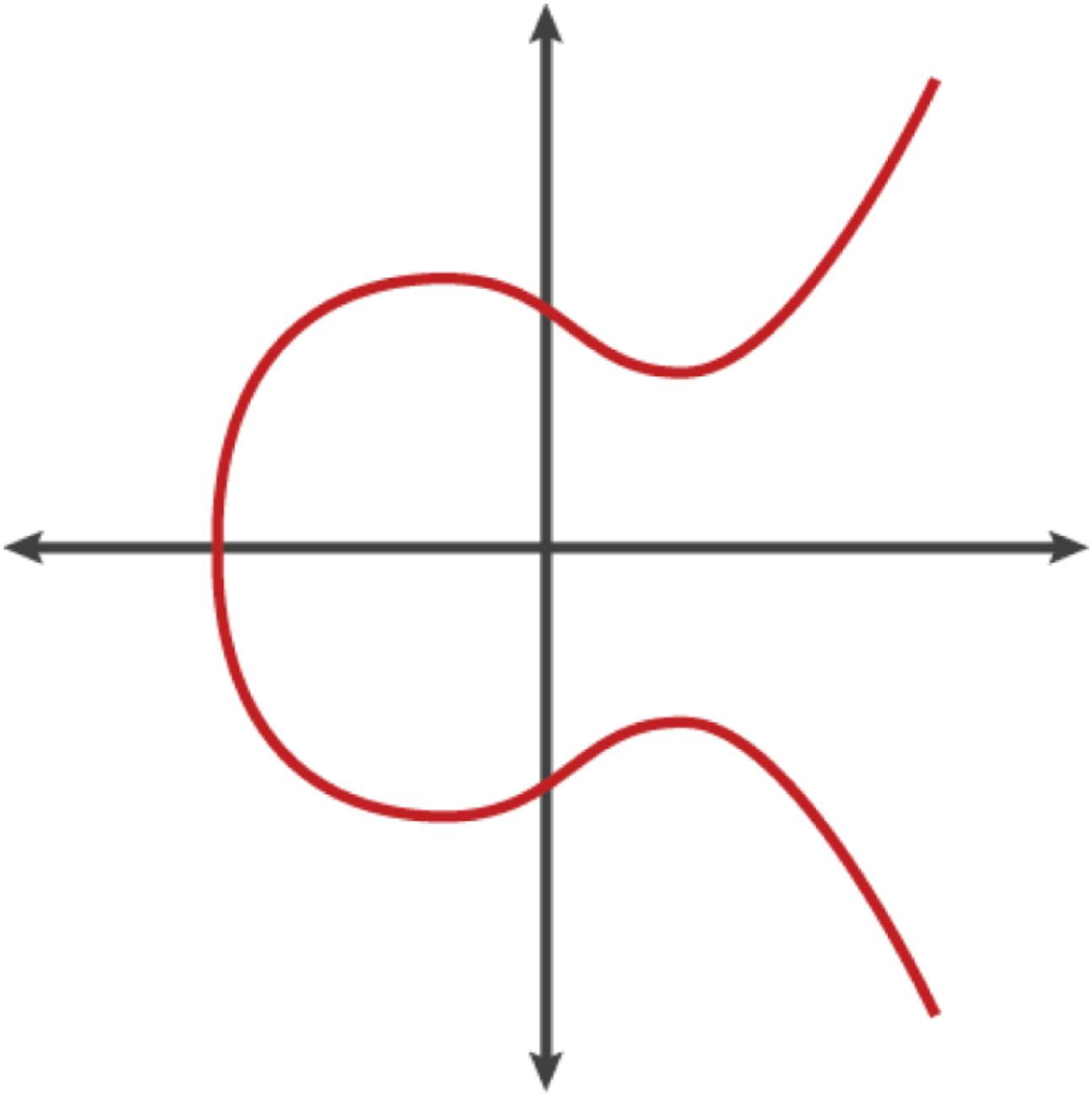
4.1.4 公钥

通过椭圆曲线乘法可以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是被称为生成点的常数点，而K是所得公钥。其反向运算，被称为“寻找离散对数”——已知公钥K来求出私钥k——是非常困难的，就像去试验所有可能的k值，即暴力搜索。在演示如何从私钥生成公钥之前，我们先稍微详细学习下椭圆曲线密码学。

提示 椭圆曲线乘法是密码学家称之为“陷阱门”功能的一种函数：在一个方向（乘法）很容易做，而不可能在相反的方向（除法）做。 私钥的所有者可以容易地创建公钥，然后与世界共享，知道没有人可以从公钥中反转函数并计算出私钥。 这个数学技巧成为证明比特币资金所有权的不可伪造和安全的数字签名的基础。

4.1.5 椭圆曲线密码学（Elliptic Curve Cryptography）解释

椭圆曲线加密法是一种基于离散对数问题的非对称加密法，可以用对椭圆曲线上的点进行加法或乘法运算来表达。下图是一个椭圆曲线的示例，类似于比特币所用的曲线。



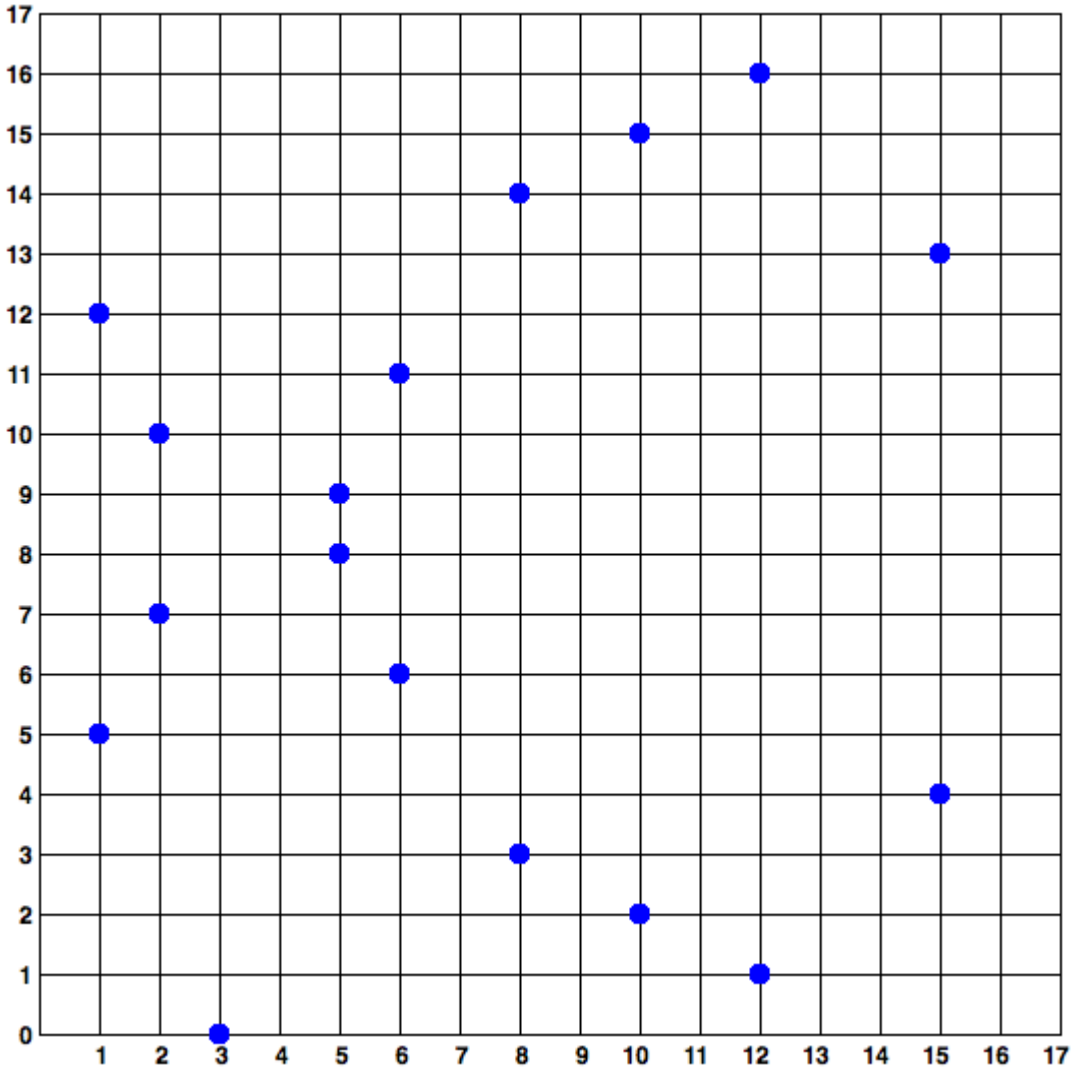
比特币使用了secp256k1标准所定义的一种特殊的椭圆曲线和一系列数学常数。该标准由美国国家标准与技术研究院 （NIST）设立。secp256k1曲线由下述函数定义，该函数可产生一条椭圆曲线：

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

或

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

上述mod p（素数p取模）表明该曲线是在素数阶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， 这是个非常大的素数。 因为这条曲线被定义在一个素数阶的有限域内，而不是定义在实数范围，它的函数图像看起来像分散在两个维度上的散落的点，因此很难可视化。不过，其中的数学原理与实数范围的椭圆曲线相似。作为一个例子，下图显示了在一个小了很多的素数阶17的有限域内的椭圆曲线，其形式为网格上的一系列散点。而secp256k1的比特币椭圆曲线可以被想象成一个极大的网格上一系列更为复杂的散点。



下面举一个例子，这是 secp256k1 曲线上的点P，其坐标为(x，y)。可以使用Python对其检验：

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

在椭圆曲线的数学原理中，有一个点被称为“无穷远点”，这大致对应于0在加法中的作用。计算机中，它有时表示为X = Y = 0（虽然这不满足椭圆曲线方程，但可作为特殊情况进行检验）。

还有一个 + 运算符，被称为“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P1和P2，则椭圆曲线上必定有第三点 P3 = P1 + P2。 几何图形中，该第三点P3可以在P1和P2之间画一条线来确定。这条直线恰好与椭圆曲线相交于另外一个地方。此点记为 P3'= (x，y)。然后，在x轴做翻折获得 P3=(x，-y)。

下面是几个可以解释“穷远点”之存在需要的特殊情况。

若 P1和 P2是同一点，P1和P2间的连线则为点P1 的切线。曲线上有且只有一个新的点与该切线相交。该切线的斜率可用微积分求得。即使限制曲线点为两个整数坐标也可求得斜率！

在某些情况下（即，如果P1和P2具有相同的x值，但不同的y值），则切线会完全垂直，在这种情况下，P3 = “无穷远点”。

若P1就是“无穷远点”，那么其和 P1 + P2= P2。类似地，当P2是无穷远点，则P1+ P2 = P1。这就是把无穷远点类似于0的作用。 事实证明，在这里 + 运算符遵守结合律，这意味着(A+B)+C = A+(B+C)。这就是说我们可以直接不加括号书写 A + B + C，而不至于混淆。 因此，我们已经定义了椭圆加法，我们可以对乘法用拓展加法的标准方法进行定义。给定椭圆曲线上的点P，如果k是整数，则 kP = P + P + P + ...+ P（k次）。注意，在这种情况下k有时被混淆而称为“指数”。

4.1.6 生成公钥

以一个随机生成的私钥k为起点，我们将其与曲线上预定的生成点G相乘以获得曲线上的另一点，也就是相应的公钥K。生成点是secp256k1标准的一部分，比特币密钥的生成点都是相同的：

{K = k * G}

其中k是私钥，G是生成点，在该曲线上所得的点K是公钥。因为所有比特币用户的生成点是相同的，一个私钥k乘以G将 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。这就是可以把比特币地址（K的衍生） 与任何人共享而不会泄露私钥（k）的原因。

提示 因为其中的数学运算是单向的，所以私钥可以转换为公钥，但公钥不能转换回私钥。

为实现椭圆曲线乘法，我们 以之前产生的私钥k和与生成点G相乘得到公钥K：

K = 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD * G

公钥K 被定义为一个点 K = (x, y)：

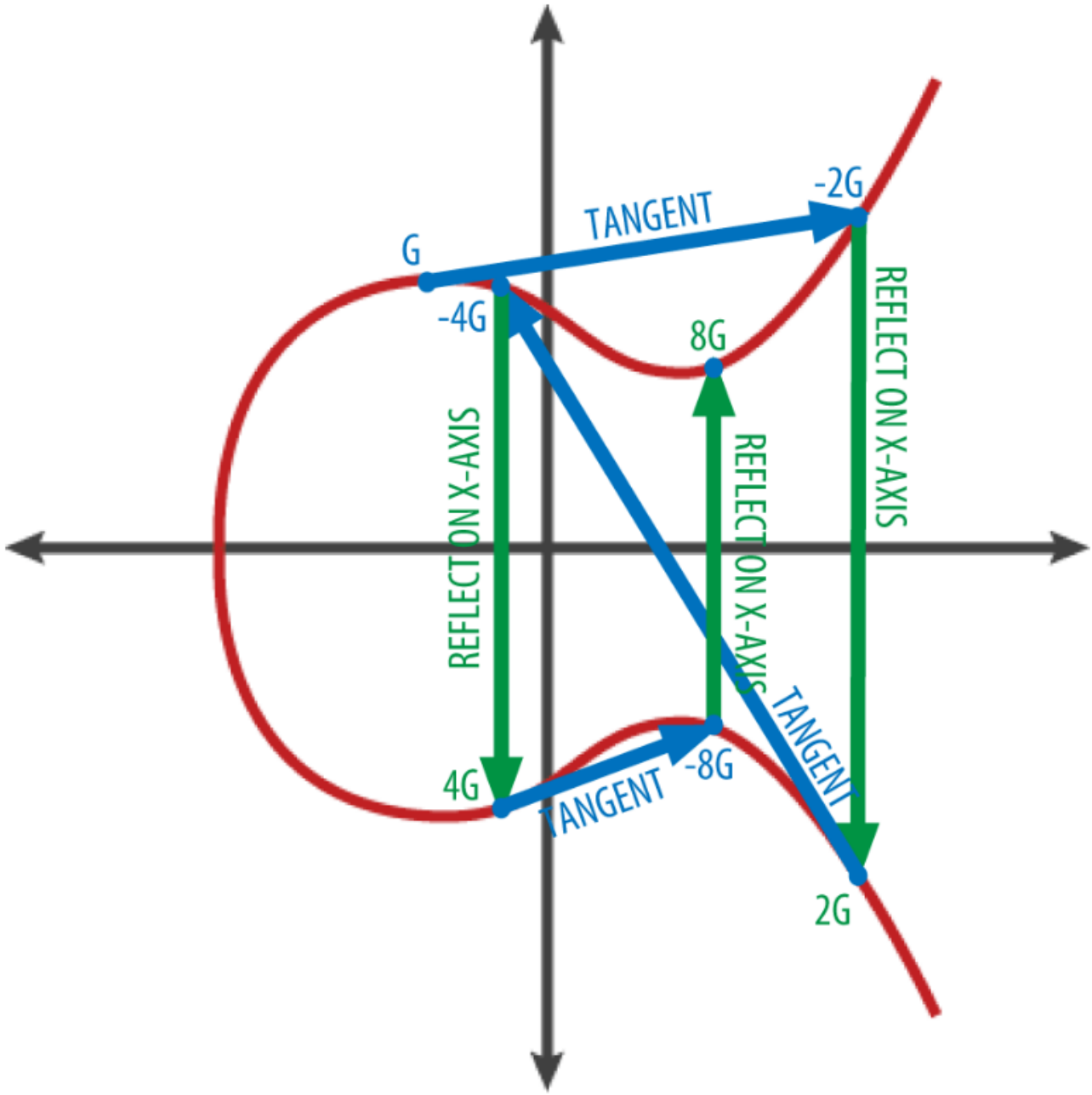
K = (x, y)

其中，

x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB

为了展示整数点的乘法，我们将使用较为简单的实数范围的椭圆曲线。请记住，其中的数学原理是相同的。我们的目标是找到生成点G的倍数kG。也就是将G相加k次。在椭圆曲线中，点的相加等同于从该点画切线找到与曲线相交的另一 点，然后翻折到x轴。

下图显示了在曲线上得到 G、2G、4G 的几何操作。



提示大多数比特币程序使用OpenSSL加密库进行椭圆曲线计算。例如，调用EC_POINT_mul() 函数，可计算得到公钥。

4.2 比特币地址

比特币地址是一个由数字和字母组成的字符串，可以与任何想给你比特币的人分享。由公钥（一个同样由数字和字母组 成的字符串）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

在交易中，比特币地址通常以收款方出现。如果把比特币交易比作一张支票，比特币地址就是收款人，也就是我们要写入收款人一栏的内容。一张支票的 收款人可能是某个银行账户，也可能是某个公司、机构，甚至是现金支票。支票不需要指定一个特定的账户，而是用一个抽象的名字作为收款人，这使它成为一种相当灵活的支付工具。与此类似，比特币地址使用类似的抽象，也使比特币交易变得很灵活。比特币地址可以代表一对公钥和私钥的所有者，也可以代表其 它东西，比如会在后面的“P2SH (Pay-to-Script-Hash)”一节讲到的付款脚本。现在，让我们来看一个简单的例子，由公钥生成比特币地址。

比特币地址可由公钥经过单向的加密哈希算法得到。哈希算法是一种单向函数，接收任意长度的输入产生指纹或哈希。加密哈希函数在比特币中被广泛使用：比特币地址、脚本地址以及在挖矿中的工作量证明算法。由公钥生成比特币地址时使用的算法是Secure Hash Algorithm (SHA)和the RACE Integ rity Primitives Evaluation Message Digest (RIPEMD)，具体地说是SHA256和RIPEMD160。

以公钥 K 为输入，计算其SHA256哈希值，并以此结果计算RIPEMD160 哈希值，得到一个长度为160位（20字节）的数字：

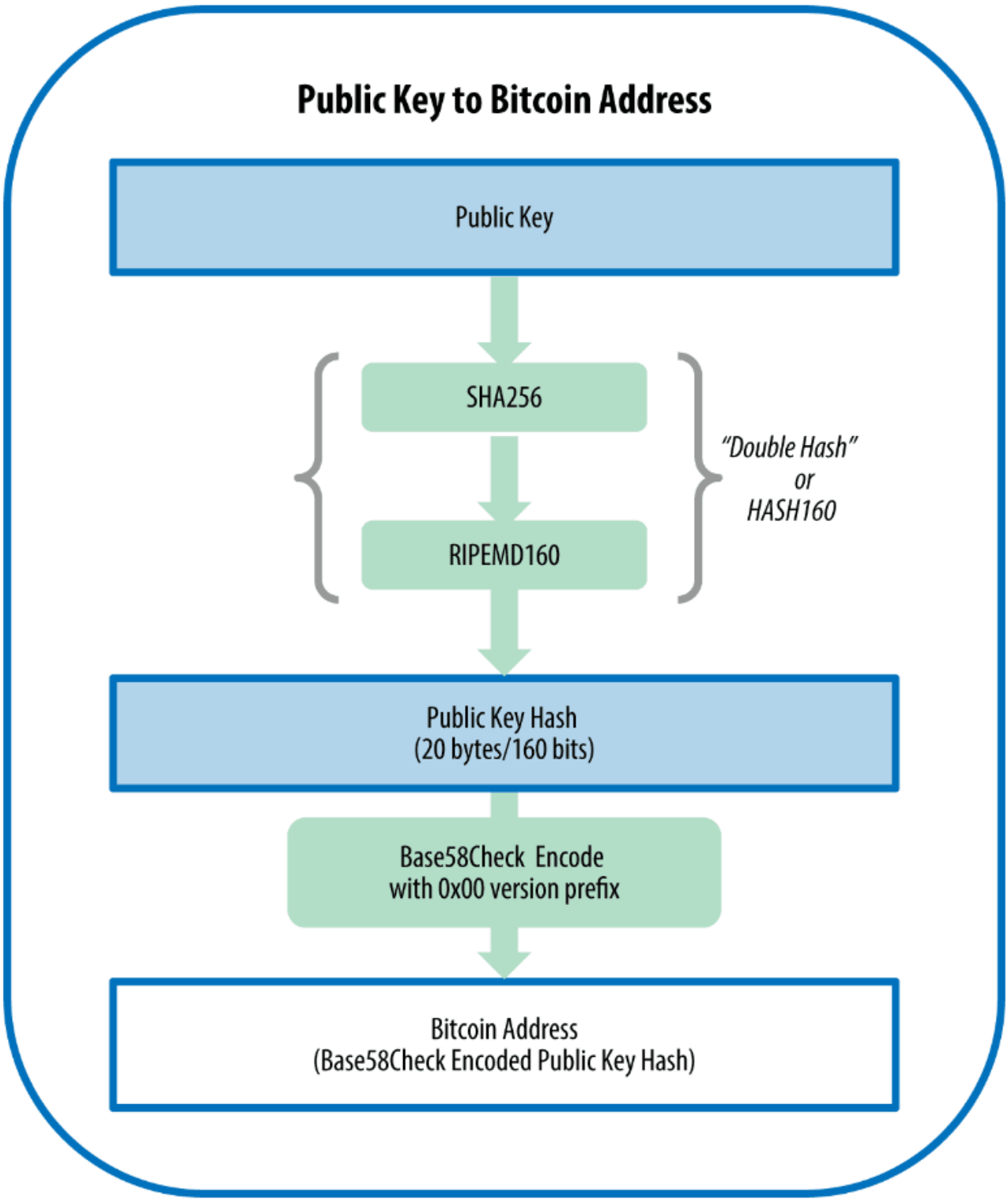
A = RIPEMD160(SHA256(K))

公式中，K是公钥，A是生成的比特币地址。

提示比特币地址与公钥不同。比特币地址是由公钥经过单向的哈希函数生成的。

通常用户见到的比特币地址是经过“Base58Check”编码的（参见“Base58和Base58Check编码”一节），这种编码使用了58个字符（一种Base58数字系统）和校验码，提高了可读性、避免歧义并有效防止了在地址转录和输入中产生 的错误。Base58Check编码也被用于比特币的其它地方，例如比特币地址、私钥、加密的密钥和脚本哈希中，用来提高可读性和录入的正确性。下一节中我们会详细解释Base58Check的编码和解码机制，以及它产生的结果。

下图描述了如何从公钥生成比特币地址。



4.2.1 Base58和Base58Check编码

为了更简洁方便地表示长串的数字，使用更少的符号，许多计算机系统会使用一种以数字和字母组成的大于十进制的表示法。例如，传统的十进制计数系统使用0-9十个数字，而十六进制系统使用了额外的 A-F 六个字母。一个同样的数字，它的十六进制表 示就会比十进制表示更短。甚至更加简洁，Base64使用了26个小写字母、26个大写字母、10个数字以及两个符号（例 如“+”和“/”），用于在电子邮件这样的基于文本的媒介中传输二进制数据。Base64通常用于编码邮件中的附件。Base58 是一种基于文本的二进制编码格式，用在比特币和其它的加密货币中。这种编码格式不仅实现了数据压缩，保持了易读 性，还具有错误诊断功能。Base58是Base64编码格式的子集，同样使用大小写字母和10个数字，但舍弃了一些容易错 读和在特定字体中容易混淆的字符。具体地，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“/”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。

例4-1 比特币的Base58字母表

123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz

为了增加防止打印和转录错误的安全性，Base58Check是一种常用在比特币中的Base58编码格式，比特币有内置的检查错误的编码。检验和是添加到正在编码的数据末端的额外4个字节。校验和是从编码的数据的哈希值中得到的，所以可以用来检测并避免转录和输入中产生的错误。使用Base58check编码时，解码软件会计算数据的校验和并和编码中自带的校验和进行对比。二者不匹配则表明有错误产生，那么这个Base58Check的数据就是无效的。一个错误比特币地址就不会被钱包软件认为是有效的地址，否则这种错误会造成资金的丢失。

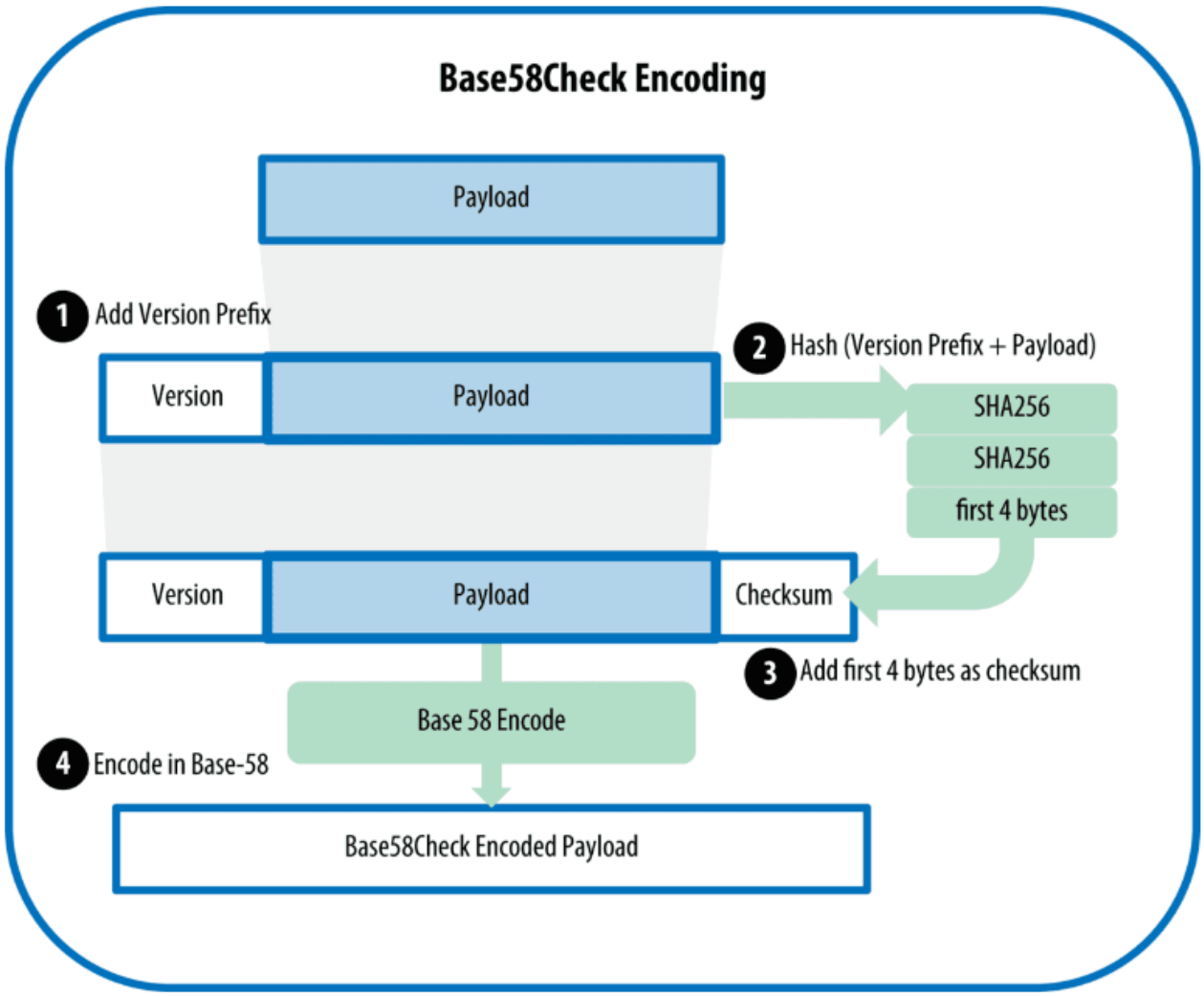
为了将数据（数字）转换成Base58Check格式，首先我们要对数据添加一个称作“版本字节”的前缀，这个前缀用来识别编码的数据的类 型。例如，比特币地址的前缀是0（十六进制是0x00），而对私钥编码时前缀是128（十六进制是0x80）。 表4-1会列出一些常见版本的前缀。

接下来，我们计算“双哈希”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))

在产生的长32个字节的哈希值（两次哈希运算）中，我们只取前4个字节。这4个字节就作为检验错误的代码或者校验和。校验码会添加到数据之后。

结果由三部分组成：前缀、数据和校验和。这个结果采用之前描述的Base58字母表编码。下图描述了Base58Check编 码的过程。



在比特币中，大多数需要向用户展示的数据都使用Base58Check编码，可以实现数据压缩，易读而且有错误检验。Base58Check编码中的版本前缀是用来创造易于辨别的格式，在Base58里的格式在Base58Check编码的有效载荷的开始包含了明确的属性。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。例如我们可以看到他们的不同，Base58Check编码的比特币地址是以1开头的，而Base58Check编码的私钥WIF是以5开头的。表4-1展示了一些版本前缀和他们对应的Base58格式。

表4-1 Base58Check版本前缀和编码后的结果

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

我们回顾比特币地址产生的完整过程，从私钥、到公钥（椭圆曲线上某个点）、再到两次哈希的地址，到最终的Base58Check编码。例4-3的C++代码完整详细的展示了从私钥到Base58Check编码后的比特币地址的 步骤。代码中使用“3.3 其他客户端、资料库、工具包 ”一节中介绍的libbitcoin library来实现某些辅助功能。

例4-3.从私钥中创建Base58Check编码的比特币地址

link:code/addr.cpp[]

代码使用预定义的私钥在每次运行时产生相同的比特币地址，如下例所示

例4-3.编译并运行addr代码

Compile the addr.cpp code $ g++ -o addr addr.cpp $(pkg-config --cflags --libs libbitcoin) Run the addr executable $ ./addr Public key: 0202a406624211f2abbdc68da3df929f938c3399dd79fac1b51b0e4ad1d26a47aa Address: 1PRTTaJesdNovgne6Ehcdu1fpEdX7913CK

4.2.2 密钥的格式

公钥和私钥的都可以有多种格式。一个密钥被不同的格式编码后，虽然结果看起来可能不同，但是密钥所编码数字并没有改变。这些不同的编码格式主要是用来方便人们无误地使用和识别密钥。

4.2.2.1私钥的格式

私钥可以以许多不同的格式表示，所有这些都对应于相同的256位的数字。表4-2展示了私钥的三种常见格式。不同的格式用在不同的场景下。十六进制和原始的二进制格式用在软件的内部，很少展示给用户看。WIF格式用在钱包之间密钥的输入和输出，也用于代表私钥的二维码（条形码）。

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

表4-3 示例：同样的私钥，不同的格式

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

这些表示法都是用来表示相同的数字、相同的私钥的不同方法。虽然编码后的字符串看起来不同，但不同的格式彼此之 间可以很容易地相互转换。请注意，“raw binary”未显示在表4-3 示例中，根据定义此处显示的任何编码的格式，不是raw binary数据。

我们使用Bitcoin Explorer中的wif-to-ec命令（请参阅[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

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。 您可以使用Bitcoin Explorer在命令行上解码Base58Check格式。

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
```

}

结果包含密钥作为有效载荷，WIF版本前缀128和校验和。

请注意，压缩密钥的“有效负载”附加了后缀01，表示导出的公钥要压缩：

$ bx base58check-decode KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

wrapper {

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

}

将十六进制转换为Base58Check编码

要转换成Base58Check（与上一个命令相反），我们使用Bitcoin Explorer的base58check-encode命令（请参阅[appdx_bx]），并提供十六进制私钥，其次是WIF版本前缀128：

bx base58check-encode 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd --version 128

5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn

将十六进制（压缩格式密钥）转换为Base58Check编码

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

生成的WIF压缩格式的私钥以字母“K”开头，用以表明被编码的私钥有一个后缀“01”，且该私钥只能被用于生成压缩格式 的公钥（参见“压缩格式公钥”一节）。

4.2.3.1公钥的格式

公钥也可以用多种不同格式来表示，最重要的是它们分为非压缩格式或压缩格式公钥这两种形式。

我们从前文可知，公钥是在椭圆曲线上的一个点，由一对坐标（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。其中一个256比特数字是公钥的x坐标，另一个256比特数字是y坐标。前缀04是用来区分非压缩格式公钥， 压缩格式公钥是以02或者03开头。

下面是由前文中的私钥所生成的公钥，其坐标x和y如下：

x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A

y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB

下面是同样的公钥以520比特的数字（130个十六进制数字）来表达。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：

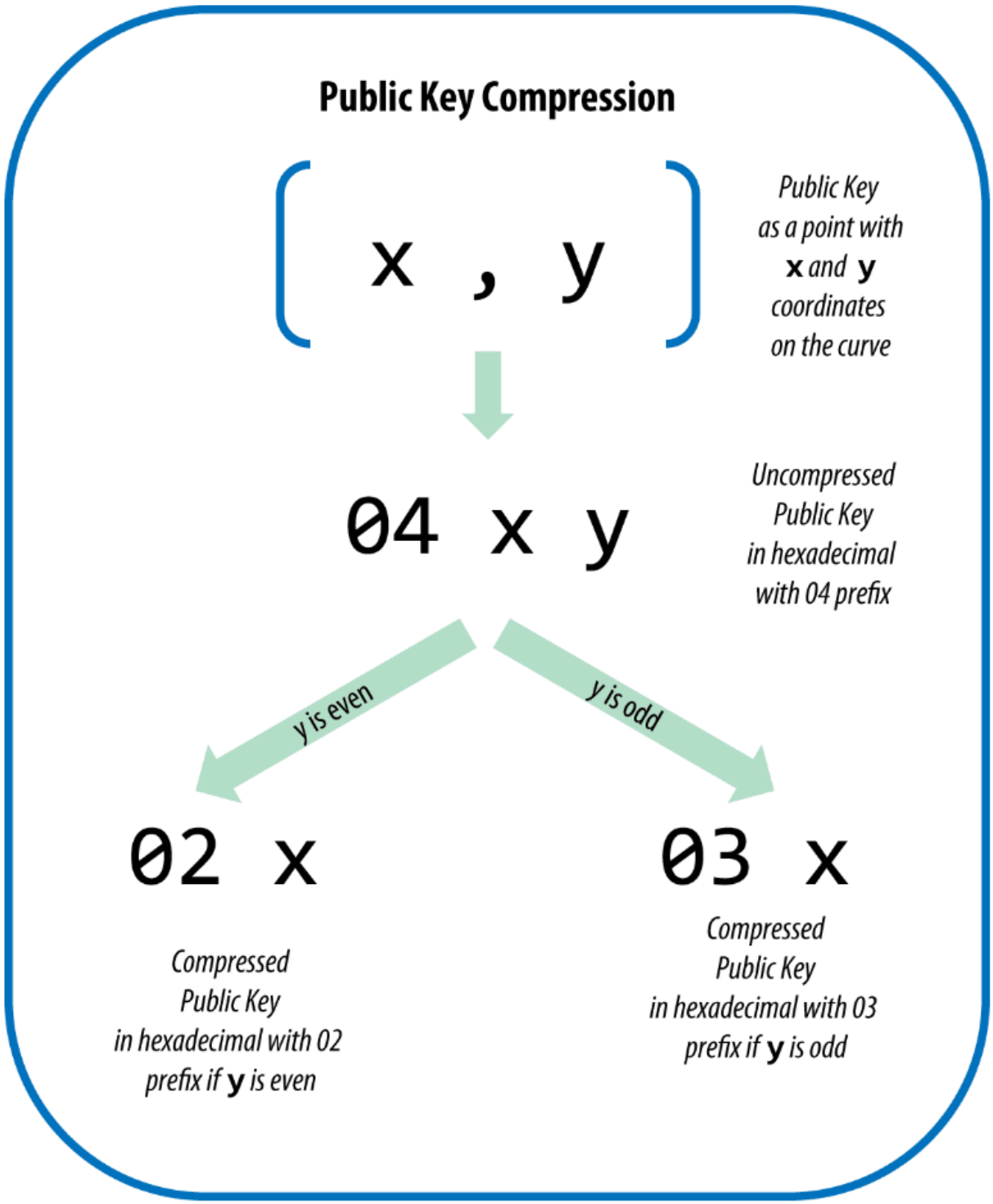
K =
04F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE 52DDFE2E505BDB

4.2.3.2压缩格式公钥

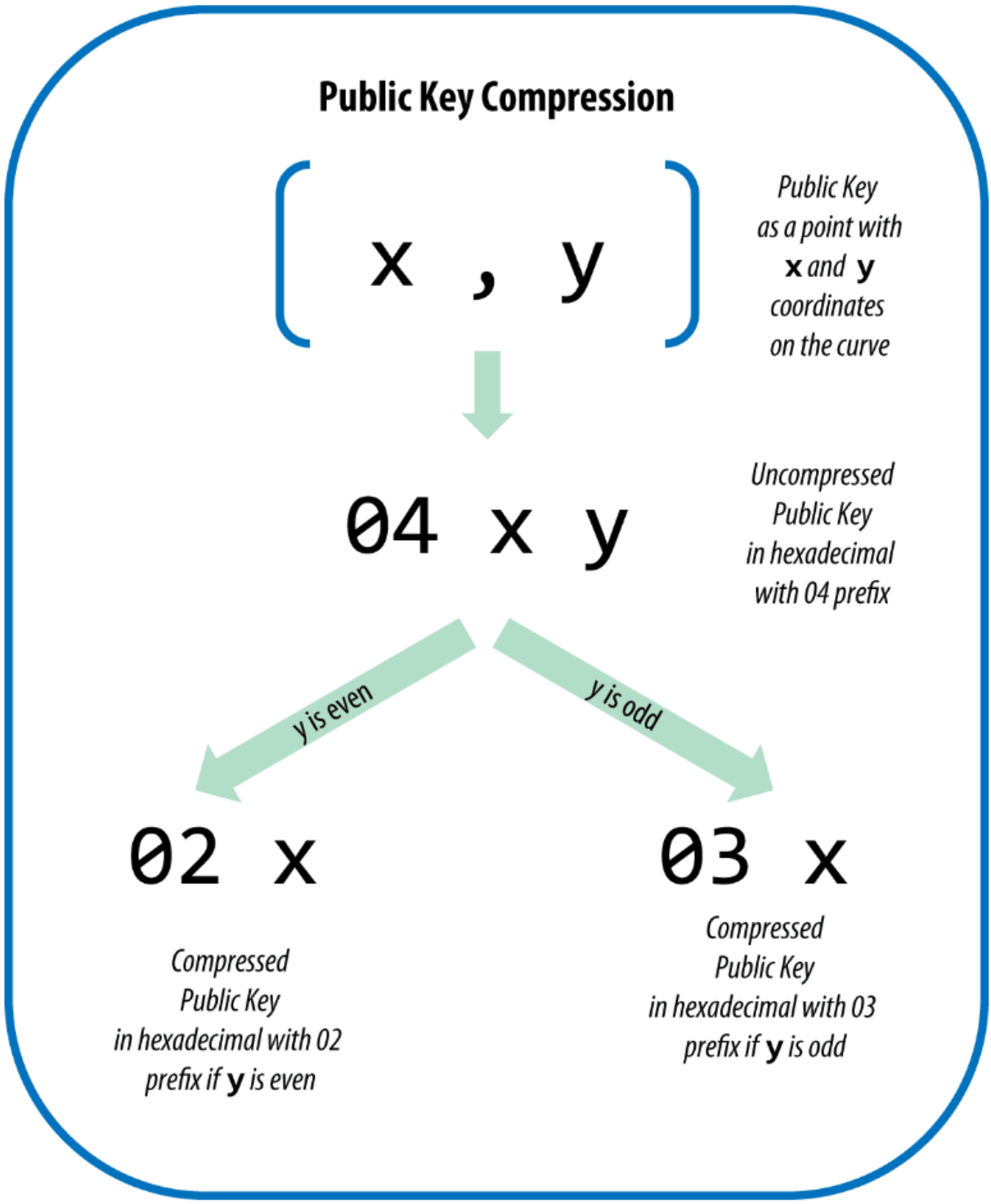
引入压缩格式公钥是为了减少比特币交易的字节数，从而可以节省那些运行区块链数据库的节点磁盘空间。大部分比特币交易包含了公钥，用于验证用户的凭据和支付比特币。每个公钥有520比特（包括前缀，x坐标，y坐标）。如果每个区块有数百个交易，每天有成千上万的交易发生，区块链里就会被写入大量的数据。

正如我们在“4.1.4 公钥”一节所见，一个公钥是一个椭圆曲线上的点(x, y)。而椭圆曲线实际是一个数学方程，曲线上的点实际是该方程的一个解。因此，如果我们知道了公钥的x坐标，就可以通过解方程y2 mod p = (x3 + 7) mod p得到y坐 标。这种方案可以让我们只存储公钥的x坐标，略去y坐标，从而将公钥的大小和存储空间减少了256比特。每个交易所 需要的字节数减少了近一半，随着时间推移，就大大节省了很多数据传输和存储。

未压缩格式公钥使用04作为前缀，而压缩格式公钥是以02或03作为前缀。需要这两种不同前缀的原因是：因为椭圆曲 线加密的公式的左边是y2 ，也就是说y的解是来自于一个平方根，可能是正值也可能是负值。更形象地说，y坐标可能在 x坐标轴的上面或者下面。从图4-2的椭圆曲线图中可以看出，曲线是对称的，从x轴看就像对称的镜子两面。因此，如果我们略去y坐标，就必须储存y的符号（正值或者负值）。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。当我们在素数p阶的有限域上使用二进制算术计算椭圆曲线的



时候，y坐标可能是奇数或者偶数，分别对应前面所讲的y值的正负符号。因此，为了区分y坐标的两种可能值，我们在生成压缩格式公钥时，如果y是偶数，则使用02作为前缀；如果y是奇数，则使用03作为前缀。这样就可以根据公钥中给定的x值，正确推导出对应的y坐标，从而将公钥解压缩为在椭圆曲线上的完整的点坐标。下图阐释了公钥压缩：



下面是前述章节所生成的公钥，使用了264比特（66个十六进制数字）的压缩格式公钥格式，其中前缀03表示y坐标是一个奇数：

K = 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A

这个压缩格式公钥对应着同样的一个私钥，这意味它是由同样的私钥所生成。但是压缩格式公钥和非压缩格式公钥看起来不同。更重要的是，如果我们使用双哈希函数(RIPEMD160(SHA256(K)))将压缩格式公钥转化成比特币地址，得到的地址将会不同于由非压缩格式公钥产生的地址。这种结果会让人迷惑，因为一个私钥可以生成两种不同格式的公钥——压缩格式和非压缩格式，而这两种格式的公钥可以生成两个不同的比特币地址。但是，这两个不同的比特币地址的私钥是一样的。

压缩格式公钥渐渐成为了各种不同的比特币客户端的默认格式，它可以大大减少交易所需的字节数，同时也让存储区块链所需的磁盘空间变小。然而，并非所有的客户端都支持压缩格式公钥，于是那些较新的支持压缩格式公钥的客户端就不得不考虑如何处理那些来自较老的不支持压缩格式公钥的客户端的交易。这在钱包应用导入另一个钱包应用的私钥的 时候就会变得尤其重要，因为新钱包需要扫描区块链并找到所有与这些被导入私钥相关的交易。比特币钱包应该扫描哪个比特币地址呢？新客户端不知道应该使用哪个公钥：因为不论是通过压缩的公钥产生的比特币地址，还是通过非压缩的公钥产生的地址，两个都是合法的比特币地址，都可以被私钥签名，但是他们是不同的比特币地址。

为了解决这个问题，当私钥从钱包中被导出时，代表私钥的WIF在较新的比特币钱包里被处理的方式不同，表明该私钥已经被用来生成压缩的公钥和因此压缩的比特币地址。这个方案可以解决导入私钥来自于老钱包还是新钱包的问题，同时也解决了通过公钥生成的比特币地址是来自于压缩格式公钥还是非压缩格式公钥的问题。最后新钱包在扫描区块链时，就可以使用对应的比特币地址去查找该比特币地址在区块链里所发生的交易。我们将在下一节详细解释这种机制是如何工作的。

4.2.3.3压缩格式私钥

实际上“压缩格式私钥”是一种名称上的误导，因为当一个私钥被使用WIF压缩格式导出时，不但没有压缩，而且比“非压缩格式”私钥长出一个字节。这个多出来的一个字节是私钥被加了后缀01，用以表明该私钥是来自于一个较新的钱包， 只能被用来生成压缩的公钥。私钥是非压缩的，也不能被压缩。“压缩的私钥”实际上只是表示“用于生成压缩格式公钥的私钥”，而“非压缩格式私钥”用来表明“用于生成非压缩格式公钥的私钥”。为避免更多误解，应该只可以说导出格式 是“WIF压缩格式”或者“WIF”，而不能说这个私钥是“压缩”的。

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

请注意，十六进制压缩私钥格式在末尾有一个额外的字节（十六进制为01）。 虽然Base58编码版本前缀对于WIF和WIF压缩格式都是相同的（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，考虑到对于Base58这是十进制编码100号和99号之间的差别。对于100是一个数字长于99的数字，它有一个前缀1，而不是前缀9。当长度变化 ，它会影响前缀。 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。

要注意的是，这些格式并不是可互换使用的。在实现了压缩格式公钥的较新的钱包中，私钥只能且永远被导出为WIF压 缩格式（以K或L为前缀）。对于较老的没有实现压缩格式公钥的钱包，私钥将只能被导出为WIF格式（以5为前缀）导 出。这样做的目的就是为了给导入这些私钥的钱包一个信号：是否钱包必须搜索区块链寻找压缩或非压缩公钥和地址。

如果一个比特币钱包实现了压缩格式公钥，那么它将会在所有交易中使用该压格式缩公钥。钱包中的私钥将会被用来在曲线上生成公钥点，这个公钥点将会被压缩。压缩格式公钥然后被用来生成交易中使用的比特币地址。当从一个实现了压缩格式公钥的新的比特币钱包导出私钥时，钱包导入格式（WIF）将会被修改为WIF压缩格式，该格式将会在私钥的后面附加一个字节大小的后缀01。最 终的Base58Check编码格式的私钥被称作WIF（“压缩”）私钥，以字母“K”或“L”开头。而以“5”开头的是从较老的钱包中 以WIF（非压缩）格式导出的私钥。

提示 “压缩格式私钥”是一个不当用词！私钥不是压缩的。WIF压缩格式的私钥只是用来表明他们只能被生成压缩的公钥和对应的比特币地址。相反地，“WIF压缩”编码的私钥还多出一个字节，因为这种私钥多了后缀“01”。该后缀是用来区分“非压缩格式”私钥和“压缩格式”私钥。

4.3 用Python实现密钥和比特币地址

最全面的比特币Python库是 Vitalik Buterin写的 pybitcointools。在例4-5中，我们使用pybitcointools库（导入为“bitcoin”）来生成和显示不同格式的密钥和比特币地址。

例4-5 使用pybitcointools库的密钥和比特币地址的生成和格式化过

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

例4-6 上例输出如下：

$ 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,
163889351287812384055267104667247415937610851208643314490666586224003393621 66L)

Public Key (hex) is:

045c0de3b9c8ab18dd04e3511243ec2952002dbfadc864b9628910169d9b9b00ec↲
243bcefdd4347074d44bd7356d6a53c495737dd96295e2a9374bf5f02ebfc176

Compressed Public Key (hex) is:

025c0de3b9c8ab18dd04e3511243ec2952002dbfadc864b9628910169d9b9b00ec

Bitcoin Address (b58check) is:

1thMirt546nngXqyPEz532S8fLwbozud8

Compressed Bitcoin Address (b58check) is: 14cxpo3MBCYYWCgF74SWTdcmxipnGUsPw3

例4-7是另外一个示例，使用的是Python ECDSA库来做椭圆曲线计算而非使用bitcoin的库。

link:code/ec-math.py[]

例4-8是上述脚本的输出。

注意： \ 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: 38090835015954358862481132628887443905906204995912378278060168703580660294000 EC point: (70048853531867179489857750497606966272382583471322935454624595540007269312627, 105262206478686743191060800263479589329920209527285803935736021686045542353380)

BTC public key: 029ade3effb0a67d5c8609850d797366af428f4a0d5194cb221d807770a1522873

4.4 高级密钥和地址

在以下部分中，我们将看到高级形式的密钥和地址，诸如加密私钥、脚本和多重签名地址，靓号地址，和纸钱包。

4.4.1 加密私钥（BIP0038）

私钥必须保密。私钥的机密性需求情况是，在实践中相当难以实现，因为该需求与同样重要的安全对象可用性相互矛盾。当你需要为了避免私钥丢失而存储备份时，会发现维护私钥私密性是一件相当困难的事情。通过密码加密存有私钥的钱包可能要安全一点，但那个钱包也需要备份。有时，例如用户因为要升级或重装钱包软件，而需要把密钥从一个钱包转移到另一个。私钥备份也可能需要存储在纸张上（参见"后面纸钱包"一节）或者外部存储介质里，比如U盘。但如果一旦备份文件失窃或丢失呢？这些矛盾的安全目标推进了便携、方便、可以被众多不同钱包和比特币客户端理解的加密私钥标准BIP0038的出台（BIP-38详细可参见附录部分）。

BIP0038提出了一个通用标准，使用一个口令加密私钥并使用Base58Check对加密的私钥进行编码，这样加密的私钥就可以安全地保存在备份介质里，安全地在钱包间传输，保持密钥在任何可能被暴露情况下的安全性。这个加密标准使 用了AES，这个标准由NIST建立，并广泛应用于商业和军事应用的数据加密。

BIP0038加密方案是：输入一个比特币私钥，通常使用WIF编码过，base58chek字符串的前缀"5"。此外BIP0038加密方案需要一个长密码作为口令，通常由多个单词或一段复杂的数字字母字符串组成。BIP0038加密方案的结果是一个由 base58check编码过的加密私钥，前缀为6P。如果你看到一个6P开头的的密钥，这就意味着该密钥是被加密过，并需要一个口令来转换（解码）该密钥回到可被用在任何钱包WIF格式的私钥（前缀为5）。许多钱包APP现在能够识别 BIP0038加密过的私钥，会要求用户提供口令解码并导入密钥。第三方APP，诸如非常好用基于浏览器的Bit Address ， 可以被用来解码BIP00038的密钥。

最通常使用BIP0038加密的密钥用例是纸钱包——一张纸张上备份私钥。只要用户选择了强口令，使用BIP0038加密的私钥的纸钱包就无比的安全，这也是一种很棒的比特币离线存储方式（也被称作"冷存储"）。

在bitaddress.org上测试表4-5中加密密钥，看看如何输入密码以得到加密密钥。

表4-5 BIP0038加密私钥例子

Private Key (WIF)	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Passphrase	MyTestPassphrase

4.4.2 P2SH (Pay-to-Script Hash)和多重签名地址

正如我们所知，传统的比特币地址从数字1开头，来源于公钥，而公钥来源于私钥。虽然任何人都可以将比特币发送到 一个1开头的地址，但比特币只能在通过相应的私钥签名和公钥哈希值后才能消费。

以数字3开头的比特币地址是P2SH地址，有时被错误的称谓多重签名或多重签名地址。他们指定比特币交易中受益人为哈希的脚本，而不是公钥的所有者。这个特性在2012年1月由BIP0016引进，目前因为BIP0016提供了增加功能到地址本身的机会而被广泛的采纳。不同于P2PKH交易发送资金到传统1开头的比特币地址，资金被发送到3开头的地址时，需要的不仅仅是一个公钥的哈希值和一个私钥签名作为所有者证明。在创建地址的时候，这些要求会被指定在脚本中，所有对地址的输入都会被这些要求阻隔。

一个P2SH地址从交易脚本中创建，它定义谁能消耗这个交易输出（后面"P2SH（Pay-to-Script-Hash）"一节对此有详细的介绍）。编码一个P2SH地址涉及使用一个在创建比特币地址用到过的双重哈希函数，并且只能应用在脚本而不是公钥：

script hash = RIPEMD160(SHA256(script))

产生的脚本哈希由Base58Check编码前缀为5的版本、编码后得到开头为3的编码地址。一个P2SH地址例子是3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM。可以使用Bitcoin Explorer命令脚本编码获得，比如sha256, ripemd160, and base58check-encode，举例如下：

$ echo dup hash160 [89abcdefabbaabbaabbaabbaabbaabbaabbaabba] equalverify checksig > script

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

3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM

提示P2SH 不一定是多重签名的交易。虽然P2SH地址通常都是代表多重签名，但也可能是编码其他类型的交易脚本。

4.4.2.1 多重签名地址和P2SH

目前，P2SH函数最常见的实现是多重签名地址脚本。顾名思义，底层脚本需要多个签名来证明所有权，此后才能消费资金。设计比特币多重签名特性是需要从总共N个密钥中需要M个签名（也被称为"阈值"），被称为M-N多签名，其 中M是等于或小于N。例如，第一章中提到的咖啡店主Bob使用多重签名地址需要1-2签名，一个是属于他的密钥和一个属于他同伴的密钥，以确保其中一方可以签署消费一笔锁定到这个地址的输出。这类似于传统的银行中的一个"联合账户"，其中任何一方配偶可以单独签单消费。或就像Bob雇佣的网页设计师Gopesh， 创立一个网站，可能为他的业务需要一个2-3的多签名地址，确保除非至少两个业务合作伙伴签署签名交易才可以进行支付消费。

我们将会在第五章节探索如何使用P2SH地址创建交易用来消费资金。

4.4.3 比特币靓号地址

靓号地址包含了**人类**可读信息的有效比特币地址。例如，1LoveBPzzD72PUXLzCkYAtGFYmK5vYNR33就是包含了Base-58 字母love的。靓号地址需要生成并通过数十亿的候选私钥测试，直到一个私钥能生成具有所需图案的比特币地址。虽然有一些优化过的靓号生成算法，该方法必须涉及随机~~士~~选择一个私钥，生成公钥，再生成比特币地址，并检查是否与所要的靓号图案相匹配，重复数十亿次，直到找到一个匹配。

一旦找到一个匹配所要图案的靓号地址，来自这个靓号地址的私钥可以和其他地址相同的方式被拥有者消费比特币。靓号地址不比其他地址具有更多或更少的安全性。它们依靠和其他地址相同的ECC和SHA。你无法比任何别的地址更容易的获得一个靓号图案开头的地址的私钥。

在第一章中，我们介绍了Eugenia，一位在菲律宾工作的儿童慈善总监。我们假设Eugenia组织了一场比特币募捐活动，并希望使用靓号比特币地址来宣传这个募捐活动。Eugenia将会创造一个以1Kids开头的靓号地址来促进儿童慈善募捐的活动。让我们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

认识到比特币地址不过是由Base58字母代表的一个数字是非常重要的。搜索“1kids”开头的图案我们会发 现从1Kids11111111111111111111111111111到1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzz的地址。这些以“1kid”开头的地址范围中大约有58的29次方地址（1.4 * 10^51）。表4-6显示了这些有“1kids”前缀的地址。

表4-6 “1Kids”靓号的范围

From	1Kids11111111111111111111111111
	1Kids11111111111111111111111112
	1Kids11111111111111111111111113
	...
To	1Kidszzzzzzzzzzzzzzzzzzzzzzzzz

我们把“1Kids”这个前缀当作数字，我们可以看看比特币地址中这个前缀出现的频率。如果是一台普通性能的桌面电脑， 没有任何特殊的硬件，可以每秒搜索大约10万个密钥。

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

正如你所见，Eugenia将不会很快地创建出以“1KidsCharity”开头的靓号地址，即使她有数千台的电脑同时进行运算。每增加一个字符就会增加58倍的计算难度。超过七个字符的图案通常需要专用的硬件才能被找出，譬如用户定制的具有多个图形处理单元（GPU）的台式机。那些通常是无法继续在比特币挖矿中盈利的钻机，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要快很多个量级。

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，如靓号矿池中的矿池。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。对小额的账单，Eugenia可以将搜索7位字符图案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。

生成一个靓号地址是一项通过蛮力的过程：尝试一个随机密钥，检查生成的地址是否和所需的图案相匹配，重复这个过程直到成功找到为止。例4-9是个靓号矿工的例子，用C++程序写的来寻找靓号地址的程序。这个例子运用到了我们在“其他替代客户端、资料库、工具包”一节介绍过的libbitcoin库。

例4-9 靓号挖掘程序

link:code/vanity-miner.cpp[]

注释编译和运行虚拟矿工示例使用std :: random_device（译者注：std :: random_device是均匀分布的整数随机数生成器，产生非确定性随机数）。 根据实施情况，可能会反映底层操作系统提供的CSRNG。 在类似Unix的操作系统（如Linux）中，它来自/ dev/urandom。 这里使用的随机数字生成器用于演示，并不适用于生成级别的比特币密钥，因为它没有以足够的安全性。

示例代码需要用C编译器链接libbitcoin库（此库需要提前装入该系统）进行编译。可以不带参数直接执行vanity-miner的可执行文件 （参见例4-10），它就会尝试找到以“1kid”开头的靓号地址。

例4-10编译并运行vanity-miner程序示例

\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

使用时间命令查看需要多久才能找到结果

$ time ./vanity-miner Found vanity address! 1KidPWhKgGRQWD5PP5TAnGfDyfWp5yceXM Secret: 2a802e7a53d8aa237cd059377b616d2bfcfa4b0140bc85fa008f2d3d4b225349

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

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”三个字符模板的结果。你可以尝试在源代码中改变search这一搜索模板，看一看如果是四个字符或者五个字符的搜索模板需要花多久时间！

4.4.3.2 靓号地址安全性

靓号地址既可以增加、也可以削弱安全措施，它们着实是一把双刃剑。用于改善安全性时，一个独特的地址使对手难以使用他们自己的地址替代你的地址，以欺骗你的顾客支付他们的账单。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可以让捐款人捐款到她宣布的一个随机生成地址（例如：1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy）。 或者她可以生成一个以“1Kids”开头的靓号地址以显得更独特。

在这两种情况下，使用单一固定地址（而不是每比捐款用一个独立的动态地址）的风险之一是小偷有可能会黑进你的网站，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 在付款之前用自己眼睛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 1j7mdg5rbqyuhenydx39wvwk7fslpeoxzy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4-8所示。

表4-8 生成匹配某随机地址的多个靓号

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

那靓号地址会不会增加安全性？如果Eugenia生成1Kids33q44erFfpeXrmDSz7zEqG2FesZEN的靓号地址，用户可能看到靓号图案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（或者靓号池付出的）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，攻击者将会被逼迫到10字符的境地，那将是个人电脑，甚至昂贵自定义靓号挖掘机或靓号池也无法生成。对Eugenia来说可承担的起支出，对攻击者来说则变成了无法承担支出，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

纸钱包是打印在纸张上的比特币私钥。有时纸钱包为了方便起见也包括对应的比特币地址，但这并不是必要的，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者线下存储比特币（即冷存储）的方式。作为备份机制，一个纸钱包可以提供安全性，以防在电脑硬盘损坏、失窃或意外删除的情况下造成密钥的的丢失。作为一个冷存储的机制，如果纸钱包密钥在线下生成并永久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更有安全性。

纸钱包有许多不同的形状，大小，和外观设计，但非常基本的原则是一个密钥和一个地址打印在纸张上。表4-14展现了纸钱包最基本的形式。

表4-9 比特币纸钱包的私钥和公钥的打印形式

Public address	Private key (WIF)
1424C2F4bC9JidNjjTUZCbUxv6Sa1Mt62x	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn

通过使用工具，就可以很容易地生成纸钱包，譬如使用bitaddress.org网站上的客户端Javascript生成器。这个页面包含所有生成密钥和纸钱包所必须代码，甚至在完全失去网络连接的情况下，也可以生成密钥和纸钱包。若要使用它，先将HTML页面保存在本地磁盘或外部U盘。从Internet网络断开，从浏览器中打开文件。更方便的，使用一个原始操作系统启动电脑，比如一 个光盘启动的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线在本地打印机上打印出 来，从而制造了密钥只存在纸张上而从未存储在在线系统上的纸钱包。将这些纸钱包放置在防火保险柜内，发送比特币到 对应的比特币地址上，从而实现了一个简单但非常有效的冷存储解决方案。图4-8展示了通过bitaddress.org 生成的纸钱包。

这个简单的纸钱包系统的不足之处是那些被打印下来的密钥容易被盗窃。一个能够接近这些纸的小偷只需偷走纸或者用把拍摄纸上的密钥，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BIP0038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，因为这些密钥从没有在线过，并且必须从保险箱或者其他物理的安全存储中导出。图4-9展示了通过bitaddress.org 生成的加密纸钱包。

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其中的总金额的话，有些钱包可能会生成一个找零地址。并且，你所用的电脑可能被病毒感染，那么就有可能泄露私钥。一 次性提走所有余款可以减少私钥泄露的风险，如果你所需的金额比较少，那么请把余额发送到相同交易的一个新的纸钱包里。

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通过某种方式隐藏私钥的保险箱内，或者用不透明的刮刮贴，或者折叠和防篡改的铝箔胶粘密封。图4-10至图4-12展示了几个不同安全和备份功能的纸钱包的例子。

图4-10 通过bitcoinpaperwallet.com生成的、私钥写在折叠袋上的纸钱包

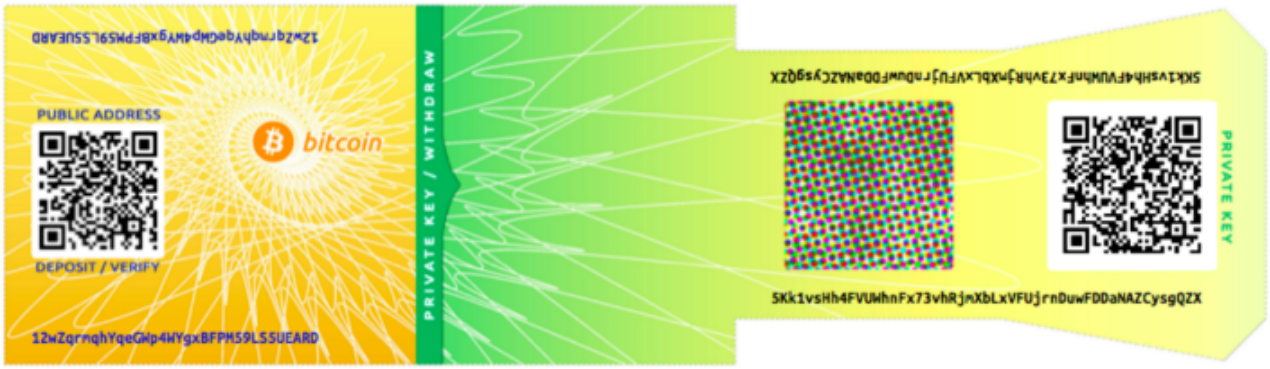


图4-11 通过bitcoinpaperwallet.com 生成的、私钥被密封住的纸钱包， 其他设计有密钥和地址的额外副本，类似于票根形式的可以拆卸存根，让你可以存储多个副本以防火灾、洪水或其他自然灾害。

图4-12 在备份“存根”上有多个私钥副本的纸钱包