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。

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，而是用一个普通的名字作为收 款人，这使它成为一种相当灵活的支付工具。与此类似，比特币地址的使用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摘要。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 Integrity Primitives Evaluation Message Digest (RIPEMD)， 特 别 是 SHA256 和RIPEMD160。

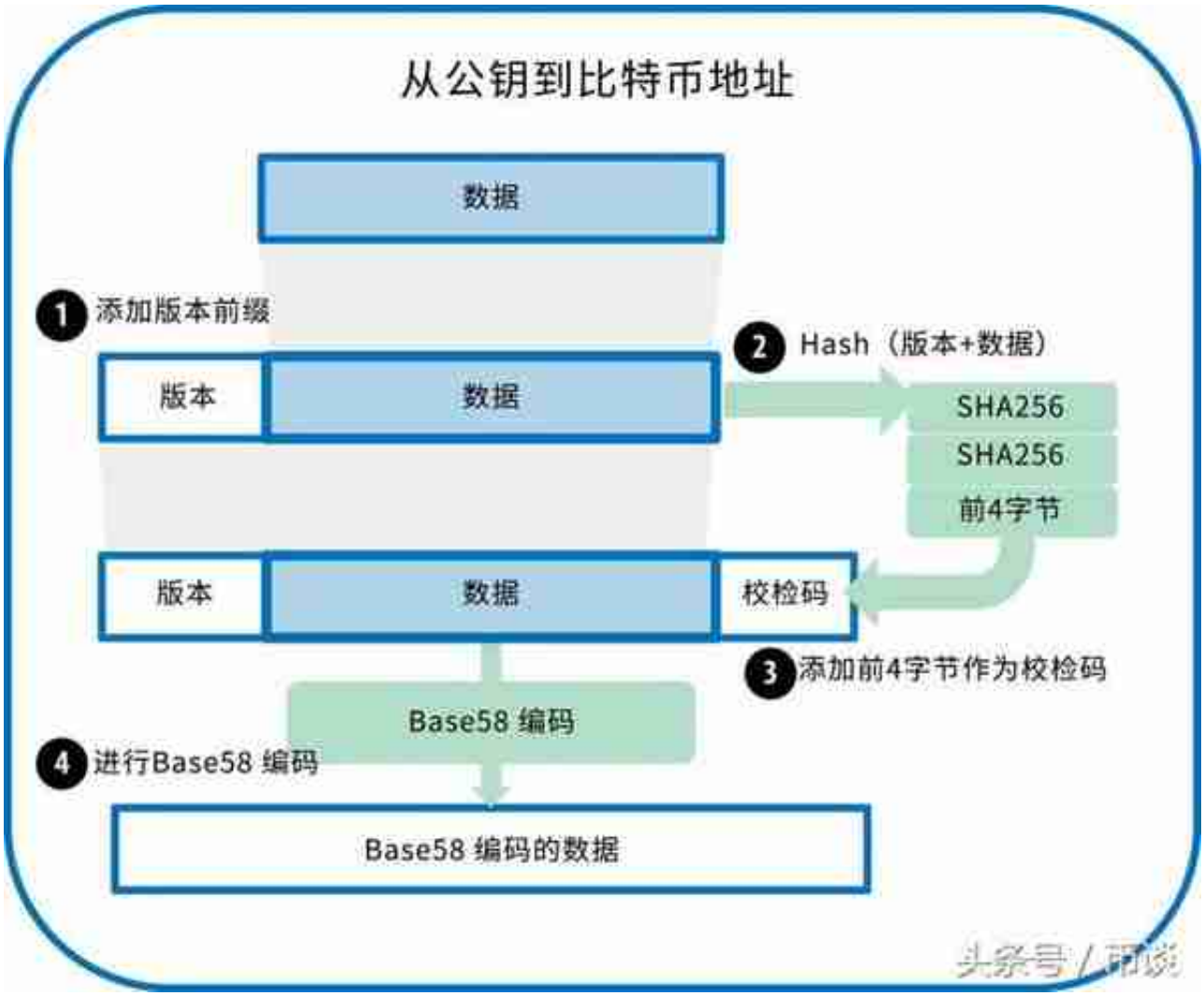
以公钥 K 为输入，计算其 SHA256 哈希值，并以此结果计算 RIPEMD160 哈希值，得到一个长度为 160 比特（20 字节）的数字：

A = RIPEMD160(SHA256(K))

公式中，K 是公钥，A 是生成的比特币地址。

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，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。



为了更简洁方便地表示长串的数字，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。更进一步，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123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxy z

Base58Check 是一种常用在比特币中的 Base58
编码格式，增加了错误校验码来检查数据在转录中出现的错误。 校验码长 4 个字节，添加到需要编码的数据之后。校验码是从需要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 编码格式时，编码软件会计算原始数据的校验码并和结果数据中自带的校验码进行对比。二者不匹配则表明有错误产生，那么这个 Base58Check 格式的数据就是无效的。例如，一个错误比特币地址就不会被钱包认为是有效的地址，否则这种错误会造成资金的丢失。为了使用 Base58Check 编码格式对数据（数字）进行编码，首先我们要对数据添加一个称作"版本字节"的前缀，这个前缀用来明确需要编码的数
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）。
表会列出一些常见版本的前缀。

接下来，我们计算"双哈希"校验码，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
个字节就作为校验码。校验码会添加到数据之后。

结果由三部分组成：前缀、数据和校验码。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。

Base58Check 编码：一种 Base58 格式的、有版本的、经过校验的格式，可以明确的对比特币数据编码的编码格式，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 的版本前缀是数据的格式易于辨别，编码之后的数据头包含了明确的属性。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种类 版本前缀 (hex)Base58

格式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 BIP38 Encrypted Private Key0x0142 6P BIP32 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，最终产生 Base58Check
格式的比特币地址。之前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"其他客户端、资料库、工具包 "中介绍的 libbitcoinlibrary 来实现某些辅助功能。

从私钥产生一个 Base58Check 格式编码的比特币地址

#include

int main() {

// Private secret key.

bc::ec_secret secret = bc::decode_hash(

"038109007313a5807b2ECCc082c8c3fbb988a973cacf1a7df9ce725c3 1b14776");

// Get public key. bc::ec_point public_key =

bc::secret_to_public_key(secret);

std::cout << "Public key: " << bc::encode_hex(public_key)

<< std::endl;

// Create Bitcoin address.

// Normally you can use:

```
// bc::payment_address payaddr;

// bc::set_public_key(payaddr, public_key);

// const std::string address = payaddr.encoded();

// Compute hash of public key for P2PKH address. const bc::short_hash
hash =

bc::bitcoin_short_hash(public_key);

bc::data_chunk unencoded_address; // Reserve 25 bytes

// [ version:1 ]

// [ hash:20 ]

// [ checksum:4 ] unencoded_address.reserve(25);

// Version byte, 0 is normal BTC address (P2PKH).
unencoded_address.push_back(0);

// Hash data bc::extend_data(unencoded_address, hash);

// Checksum is computed by hashing data, and adding 4 bytes from hash.
bc::append_checksum(unencoded_address);

// Finally we must encode the result in Bitcoin's base58 encoding
assert(unencoded_address.size() == 25);

const std::string address = bc::encode_base58(unencoded_address);

std::cout << "Address: " << address << std::endl; return 0;

}
```

正如编译并运行 addr 代码中展示的，由于代码使用预定义的私钥，所以每次运行都会产生相同的比特币地址。

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