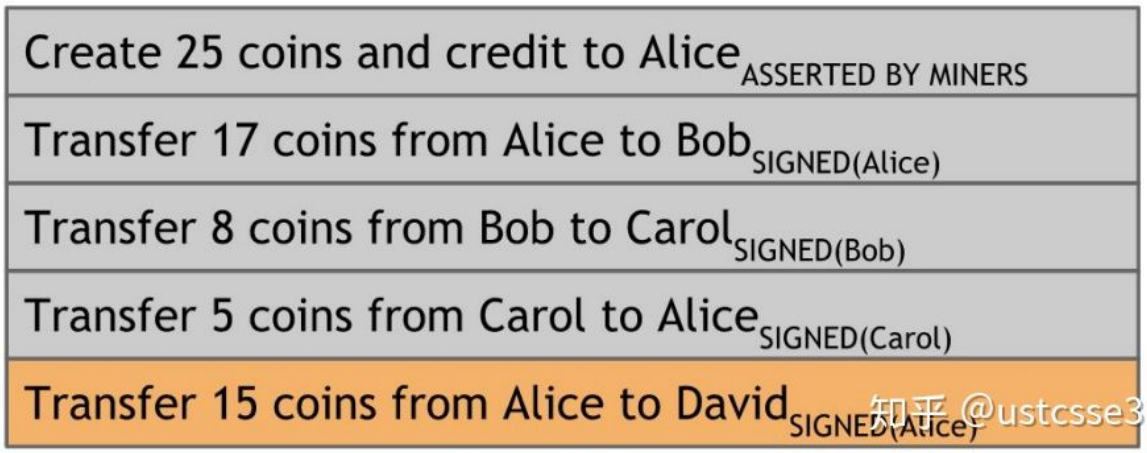
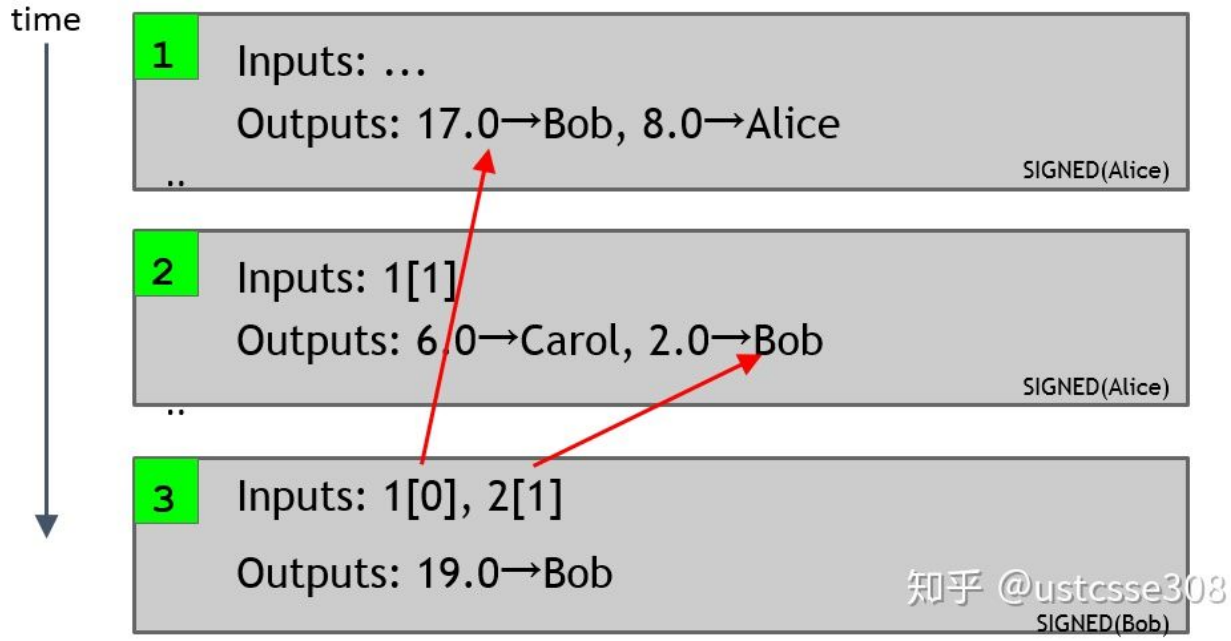
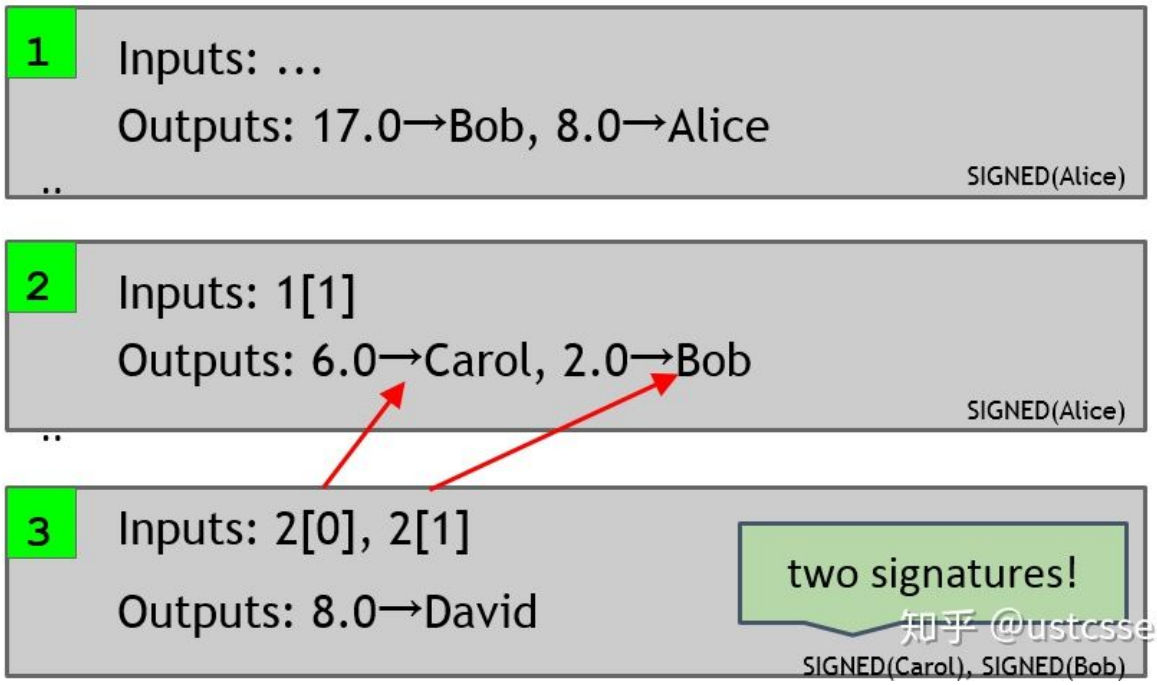
传统的记账形式



比特币中的记账形式：





使用掉谁的钱就要跟上谁的signature

下图是比特币交易的语法：

在上图中有两个需要注意的变量：

- `scriptSig`：这个`scriptSig`就是一个凭证，证明交易的创建者确实有使用这个输出的权利
- `scriptPubKey`：`scriptPubkey`相当于是一把锁（lock），交易Tx1的创建者（如Alice）指定了只有Bob才能拿走交易Tx1的输出，那么scriptPubkey一定要能够保证确实是Bob才能使用；而Bob在创建交易Tx2的时候也必须提供锁的钥匙，就是scriptSig，证明自己。当然，scriptPubkey的锁可能是各种各样的，可以是特定身份的人；也可以是一个问题的答案，不论是谁，只要回答出来，就可以拿走output。可以看下面的例子。

```
2 7 OP_ADD 3 OP_SUB 1 OP_ADD 7 OP_EQUAL
```

对于上面的scriptPubkey，scriptSig应该是什么？ 12

简单来讲验证过程：

在比特币中，证明个人身份的形式是通过**公钥的哈希值**

当一个人想要用钱的时候，他只需要提供2个信息：

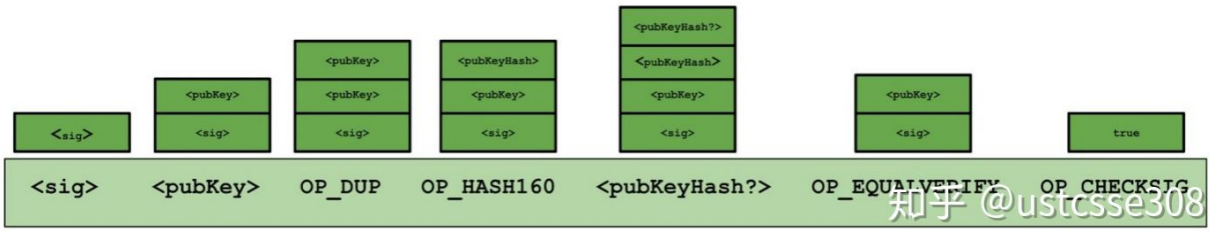
1. 完整的公钥 (pubkey)
2. 签名 (sig)，这个是通过私钥进行加密的，即签名是使用私钥对交易的签名

当矿工拿到这两个信息之后，他会：

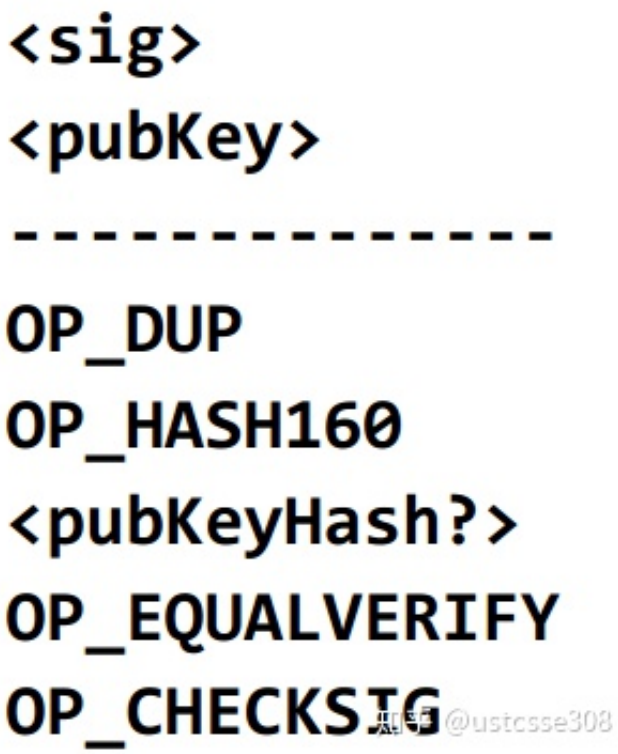
1. 使用完整的公钥(pubkey)做一下hash，看看是否和上一个交易的pubkey一样
2. 如果一样的话，则使用公钥(pubkey)可不可以解开签名(sig)

详细的验证过程：

验证算法的执行栈如下图所示： 很重要



其中栈中的数据如下所示：

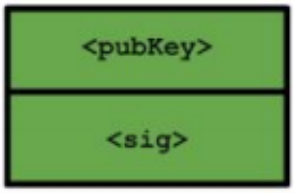


顾名思义，OP_DUP是duplicate复制，OPHASH160是进行哈希，69e0....串是指定的地址，OP_EQUALVERIFY是验证是否相等，以及OP_CHECKSIG是进行签名验证。

执行过程如下：

首先是指令，这是数据指令，是来自于Bob的签名，也即Tx2的中scriptSig的第一部分，入栈。

接下来是，同样是数据指令，是来自于Bob的完整公钥，也即Tx2的中scriptSig的第二部分，入栈。

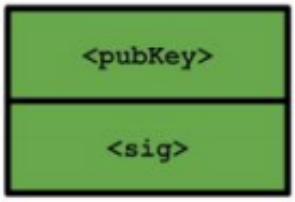


第三条指令是OP_DUP，这是来自Tx1的Alice的输出的scriptPubKey，OP_DUP添加到堆栈，因为是复制，所以把当前栈顶的数据复制一份放到栈顶，这样把BOB提供的公钥复制了一份。

第四个指令是OP_HASH160，入栈，对下面的数据，也即Bob的公钥进行两次哈希（SHA-256以及RIPEMD-160），把自己替换掉。这样就获得了Bob公钥的哈希值。

第五条指令是数据指令，，同样来自于交易Tx1，Alice指定的的输出地址，入栈。这样栈顶就有两份哈希值了。

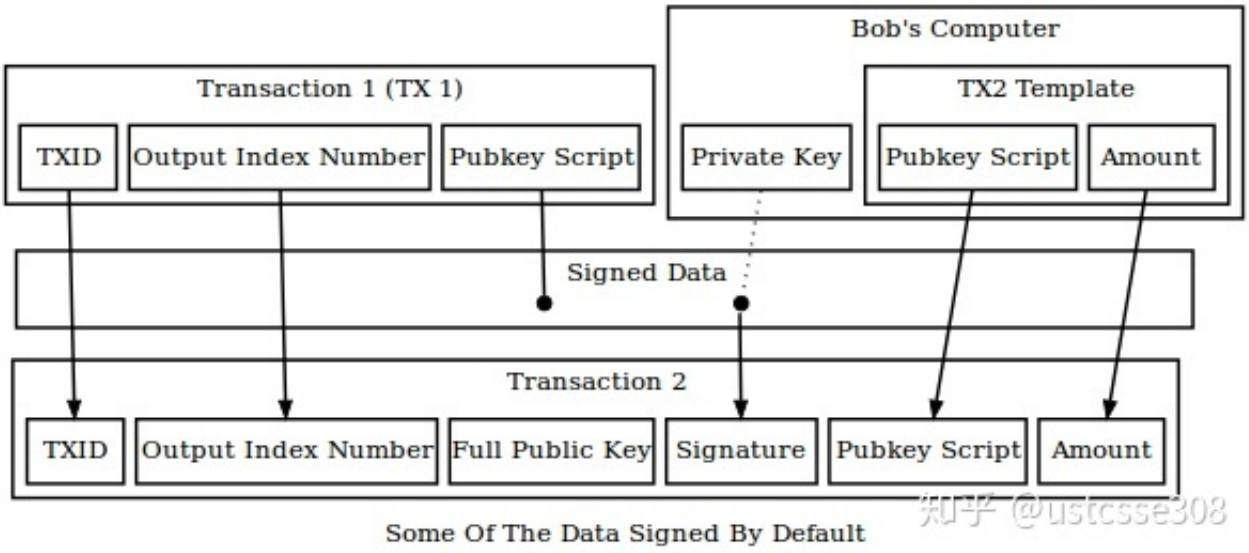
下一条指令稍微复杂点，OP_EQUALVERIFY，入栈，相当于展开成EQUAL和VERIFY两个操作。EQUAL的操作是检查它下面的两个值是否相等，这里，也即检查Alice指定的地址（栈顶）和Bob提供的完整公钥生成的哈希（栈顶第二个）是否相等。EQUAL会得到0(false)或者1(true)。VERIFY检查EQUAL的返回值，如果是false，则交易非法，如果是true，则将自己和true出栈。这里，是为true的情况。



最后一条指令是OP_CHECKSIG入栈，对栈中的两个元素进行检查，当前栈中的数据实际就是Bob的输入中提供的完整公钥和对应私钥的签名，如果验证通过则True入栈。

签名的内容(使用私钥进行加密的内容)：

假设有两个交易TX1和TX2，TX2是新的交易，Bob是交易的发起人，那么整个TX2即为需要被加密的内容



确切地说，就是交易Tx2的除了签名部分之外的内容（可以通过标志位对进行签名的交易内容进行简化，在这个例子中，使用pubkey Script替代Signature做填充）。上图共有三个部分，中间是Signed Data，也即被签名的数据。上部是Tx1和Bob自己的数据，下部分是Bob最终形成的Tx2。从上部分和中间部分

形成了下部。（签名的另一个好处是，Tx2的明文部分也不能被攻击者随意篡改）