### 实验3 Script—基于堆栈的智能合约

1.背景

**OP\_HASH160** **4d5a9a16f711b32eed6527646df617b21d052350 OP\_EQUALVERIFY**

图1：script示例

实验2中曾经介绍过，交易产生的每个output都有一个**script（脚本）**字段，该字段的用途是“锁”住这个output，使其只能被特定用户使用。例如，图1中展示的是区块高度为1中的第0个output的脚本内容，其含义是只有提供一个HASH160值（一个特定的哈希函数）等于4d5a9a16f711b32eed6527646df617b21d052350的字符串，才可以使用这个output。**注意：哈希函数具有单向性**，知道HASH160(x)的值，我们几乎不可能找到对应的x。

发布此交易的用户先选择一个字符串*x*，计算其哈希值并放到脚本中。

假设后续某个交易的input尝试使用这个output，这个input的scriptSig内容必须为字符串*x*。在验证交易时，系统将input的scriptSig和所引用的output的script合并，得到下面的字符串序列（注意*x*是一个字符串而非字母x）：

*x* OP\_HASH160 4d5a9a16f711b32eed6527646df617b21d052350 OP\_EQUALVERIFY

这个字符串序列构成一个完整的脚本，其执行过程基于堆栈，如下所示。

**1、**{x}

**2、**{4d5a9a16f711b32eed6527646df617b21d052350}

**3、**{4d5a9a16f711b32eed6527646df617b21d052350, 4d5a9a16f711b32eed6527646df617b21d052350}

**4、**

初始栈为空

**1、**读入*x*，入栈

**2、**读入 OP\_HASH160，取出栈顶元素，计算HASH160值，入栈

**3、**读入下一个字符串，入栈

**4、**读入OP\_EQUALVERIFY，弹出栈顶两个元素，判断是否相等，若相等则验证通过

**操作 堆栈情况**

运行完毕，验证通过，说明这个input可以使用相应output。

基于脚本，比特币实现了非常简单的智能合约（smart contract），用户借助密码学技术，不但可以实现安全转账，还能给转账附带复杂的条件，比如要求某个output必须在一定时刻（区块高度）之后才能被使用，要求某个output必须要得到多人授权才可以被使用等，甚至还出现了基于比特币脚本的博彩游戏（SatoshiDice）。

本实验要求基于栈实现简单的比特币脚本解析和交易script验证。

2．目的

* 设计并实现基于栈的脚本解析和执行。
* 掌握基于栈的算法设计思想。

1. 问题描述

* 在实验2的基础上，增加验证交易的Script检测实验数据中交易合法性：验证交易input与其使用的output的完整脚本，若通过则该input可以使用此output，否则不能。
* 给定一组区块，以及一组交易，执行这些交易所引用的脚本，并输出脚本运行结果。

脚本类型说明如下：

**脚本1（puzzle，提供一个字符串得到指定哈希）：**

Output: OP\_HASH160 4389813341121533778 OP\_EQUALVERIFY

Input: HELLOBLOCKCHAIN

此处 OP\_HASH160 可以调用 std::hash 来计算。

**脚本2（算术题）：**

Output: OP\_BEGIN\_CALC 1 + 2 OP\_END\_CALC OP\_EQUALVERIFY

Input: 3

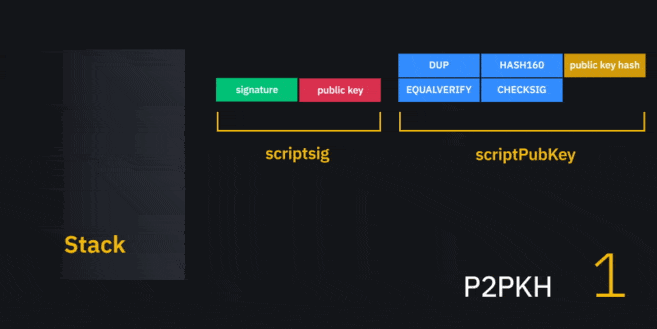
说明：这是为实验需要而设计的SCRIPT，不能起到安全保护效果，在真实的比特币系统不会使用。上述脚本的含义是要求Input给出在Output里面四则运算表达式 1 + 2 的正确答案。OP\_BEGIN\_CALC和OP\_END\_CALC分别定义了一个四则运算表达式的开始和结束。*OP\_BEGIN\_CALC 四则运算表达式 OP\_END\_CALC* 在执行完毕之后为四则运算表达式的值。

**脚本3（P2PKH交易）：**

Output: OP\_DUP OP\_HASH160 cbc20a7664f2 OP\_EQUALVERIFY OP\_CHECKSIG

Input: <signature> < public key>

关于P2PKH交易类型的脚本执行方式，可参考<https://zhuanlan.zhihu.com/p/553299013>中“支付到公钥(P2PK)”之后紧接着的“支付到公钥哈希(P2PKH)”，观看其中的“P2PKH”GIF动画，如下所示：



还可参考：理解比特币脚本<https://zhuanlan.zhihu.com/p/25461051>、深入理解比特币交易的脚本<https://zhuanlan.zhihu.com/p/24838810>等。

关于公私钥和数字签名，建议阅读：<https://blog.51cto.com/u_9634496/3075157>。

关于操作码可参考：<https://developer.bitcoin.org/reference/transactions.html#opcodes>。本次实验所用到的操作码全部在下面的表一中。

表一 实验3所涉及的操作码

|  |  |
| --- | --- |
| 操作码 | 含义 |
| OP\_BEGIN\_CALC | 定义一个四则运算表达式的开始 |
| OP\_END\_CALC | 定义一个四则运算表达式的结束，计算表达式并将结果压入栈中 |
| OP\_HASH160 | 弹出栈顶的数，调用std::hash计算，并将结果压入栈中 |
| OP\_EQUALVERIFY | 弹出栈顶两个操作数，判断它们是否相等 |
| OP\_DUP | 复制（非弹出）栈顶元素的值，并将复制的结果压入栈中 |
| OP\_CHECKSIG | 弹出栈顶两个操作数，其中第一个必须是公钥，第二个是对公钥的签名。验证该签名是否有效 |

1. 辅助代码说明

助教实现了经典的RSA，即**rsa.zip**，用于签名和验证（提示：里面还有使用std::hash的例子）。请查看main函数中的调用例子。本代码仅用于实验，对RSA做了很多简化，不能用于实际的加密场景。

1. 实验数据说明

(1) 采用更新后的demo数据集，即“**实验3数据.zip”**，可以自行补充更多的output和input。

(2) 仍需检查所引用的output是否被使用过，以及所引用的output的value之和是否大于等于本交易的output的value之和。

(3) 四则运算表达式的token之间均以空格隔开，运算数值均为整数。

(4) 不同标准库的std::hash实现可能不一样，所以同一个输入调用hash用不同编译器编译运行后的结果可能会不一样：<https://stackoverflow.com/questions/51145320/does-stdhash-give-same-result-for-same-input-for-different-compiled-builds-and>。可以先使用辅助代码验证上方 HELLOBLOCKCHAIN的哈希值是否为4389813341121533778，若不同，可以自己修改一下数据。