目前情况：

1. 哈希值

在solidity中，keccak256可计算哈希值。

0.5版本之前：keccak256(…)，可接受任意个参数，计算其拼接后的哈希值

0.5版本以后：keccak256(abi.encodePacked(…))。由于keccak256只接受一个bytes类型的参数，因此需使用abi.encodePacked转换。注意：abi.encodePacked不接受1、2等整型常数，需要以变量形式传入。

uint类型相当于uint256，为32字节。address类型为20字节。

例如：a = 1; b = 2;

abi.encodePacked(a,b);

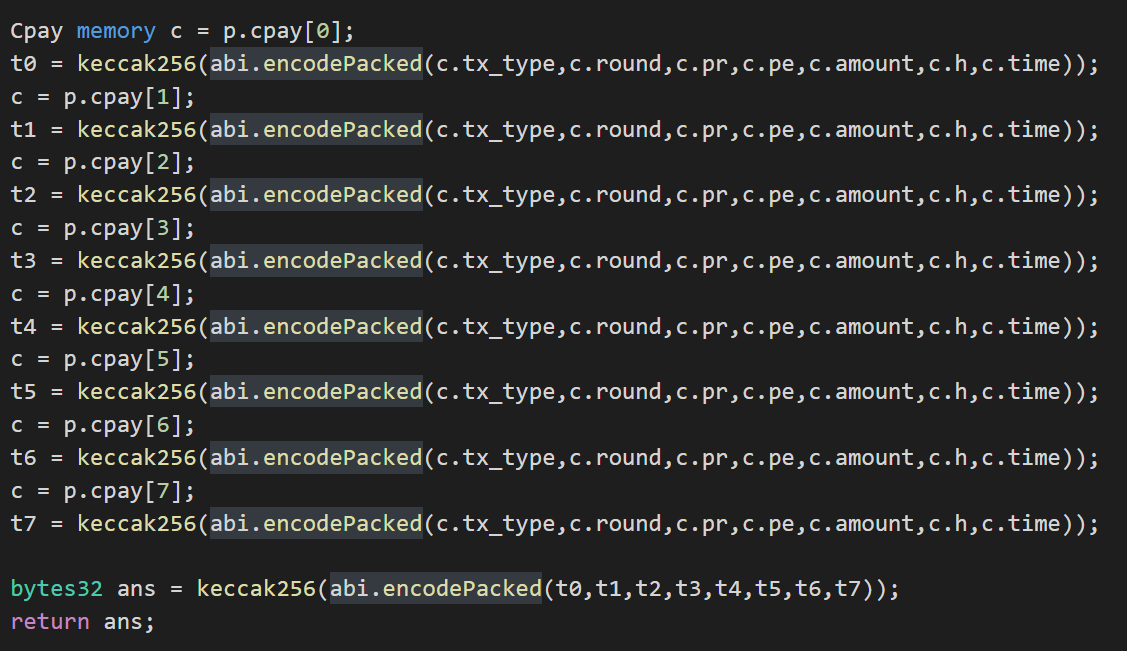
结果为0x00…00100…002（实际结果以byte形式存储）

在python中，可使用sha3.keccak\_256计算哈希值。

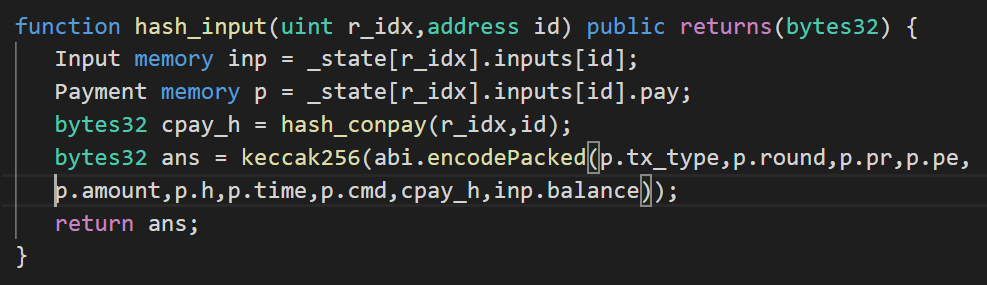
整数类型可通过(int).to\_bytes(32, byteorder='big')转化为byte类型，其中32表示长度。

多个byte变量通过+即可拼接。

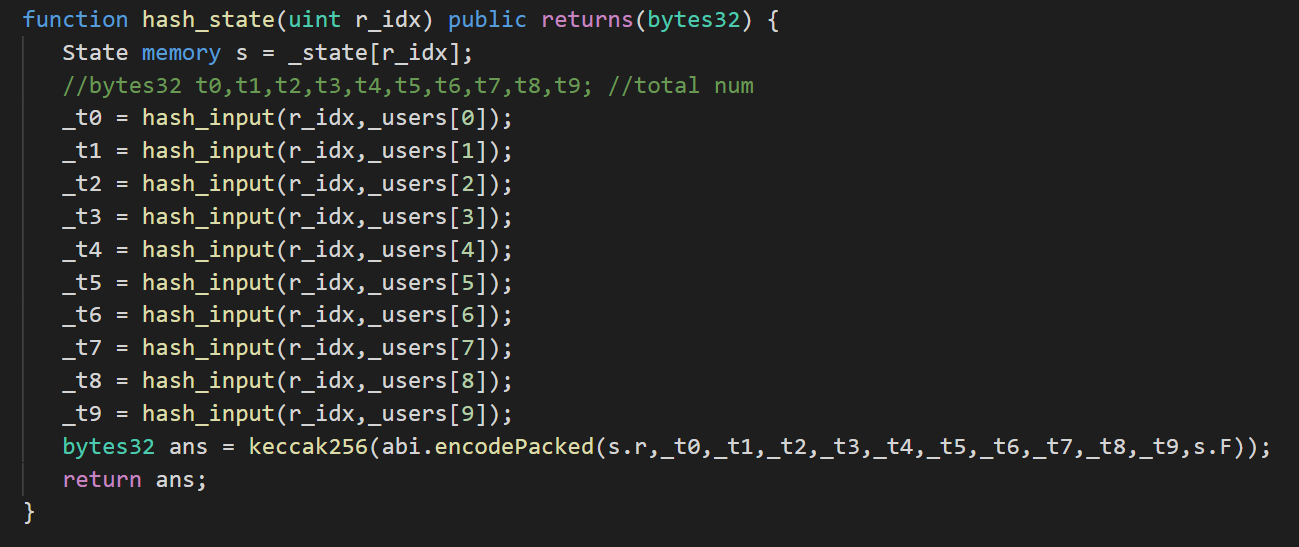
cpay的哈希值：在每个input中，cpay的大小固定为常数8，将每一个cpay的各个数据成员拼接后计算其哈希值，再将所有哈希值拼接起来计算一次哈希值。



input的哈希值：将input中的各数据成员与上述conpay的哈希值拼接后计算哈希值。



state的哈希值：将各个地址的input信息分别计算hash值，得到的结果与其余数据成员拼接，再计算哈希值。



由于gas计费的问题，函数内部无法定义数组，且对于变量的使用数量似乎也有限制，因此目前只能定义一些全局变量。

1. 签名

在solidity中，可使用ecrecover函数，根据签名和签名前的哈希值恢复地址信息，据此判断签名是否正确。

在python中，可使用sign\_recoverable函数进行签名。

总体流程为，在python中计算哈希值，对哈希值进行签名，更新到网络。solidity中验证时，重新计算哈希值，并验证签名是否正确。

验证分为：verify\_input，即验证input中用户的签名是否正确；verify\_state，即验证state中Custodian的签名是否正确。

1. gas值消耗

新建state：可能涉及到cpay的初始化，gas消耗约为500万。

verify\_input：单个用户验证约为21万，对于10个用户的小组，总消耗约为210万。

verify\_state：在使用call时可以正常运行，但使用transact时ganache会崩溃，因此无法观察gas消耗情况。目前还不清楚是代码的bug还是ganache的bug。

1. 总体情况

签名的验证基本完成（除了verify\_state的崩溃问题）

record等函数已写完框架，还有emit和send等内容需要完善。所有检验函数需要手编数据以验证实现是否正确。