交易有效性功能及代码说明

摘要

优仕链第一个版本主要实现了身份验证的功能 :链上的所有地址必须经过第三方机构的身份验证及审核，只有通过审核的地址才可以在链上进行交易。工作主要分为四个部分：证书服务器：负责第三方机构认证签名的审核以及用户证书的颁发；钱包客户端：负责实现用户认证、发送交易、查询余额的入口；认证合约：负责在链上记录通过审核的合法地址；交易数据层：负责对认证交易的有效性审核及普通交易的合法性审核。

目录

交易有效性功能及代码说明 1

一、 认证交易的打包 2

二、 认证交易的有效性校验和普通交易的合法性审核 2

1. 身份验证交易类型的确认 3

2. 身份验证交易的有效性校验 4

3. 普通交易地址合法性审核 5

4. 区块的过滤 7

5. 认证交易的 gas 消耗去除 7

一、 认证交易的打包

添加 console 新命令 sendAuthentication()，用于认证交易的发送

```
var sendAuthentication = new Method({
    name: 'sendAuthentication',
    call: 'eth_sendTransaction',
    params: 1,
    inputFormatter: [formatters.inputAuthenticationFormatter]
});
```

二、 认证交易的有效性校验和普通交易的合法性审核

节点在收到交易或者自身打包交易后，交易会进入交易缓冲区内，程序会对交易缓冲池内的所有交易进行校验(go-ethereum/core/tx-pool.go line591 validateTx())

除了采用以太坊架构中对交易大小、签名等信息校验，usechain 新加入了身份认证交易的签名校验和普通交易的地址合法性校验； 同时考虑到新地址不拥有 ust，无法支付第一笔身份验证交易的交易费，所以设置第一次身份验证交易的 gasPrice 为 0。

1. 身份验证交易类型的确认

身份认证的 payload 段数据结构：

level 表示身份认证等级， addressType 表示是主地址还是子地址。

如果 to（）的地址是 CA 合约地址，function hash 为 0x74f54c37，且长度

有效的交易会被视作身份认证交易，会进入 Tcert 校验等后续操作，如果校验失败，交易会被抛弃；而普通交易则不会进行身份认证方面的校验。

```go
//Another authentication implementation write in state_trasanction.go
func (tx *Transaction) IsAuthentication() bool {
    fmt.Println( a: "IsAuthentication func entered!")
    //The authentication tx payload must longer than 36 bytes
    //Added levelTag and address type
    if len(tx.Data()) <= 4 + 32 + 32 {
        return false
    }

    //leavel below, something wrong with !=
    if bytes.Compare(tx.Data()[:4], []byte{0x74, 0xf5, 0x4c, 0x37}) != 0 {
        return false
    }

    if !strings.EqualFold((*tx.To()).Hex(), common.AuthenticationContractAddressString) {
        fmt.Println( a: "Contract address doesn't match")
        return false
    }

    return true
}
```

2. 身份验证交易的有效性校验

一旦确认为身份验证交易，则会对 Tcert 的有效性进行校验，包括 level、Tag、address 是否吻合，证书是否过期，证书是否由 TCA 颁发。

```go
func checkCert (txCertData []byte, addrFrom string, levelTag int, addressTag int) error {

    var tcaCert *x509.Certificate
    var txCert *x509.Certificate

    tcaCert = readTcaEcdsa()
    if tcaCert == nil {
        return errors.New( text: "tcaCert missing")
    }

    txCert = parseEcdsaByte(txCertData)
    if txCert == nil {
        return errors.New( text: "txcert error")
    }
    fmt.Println( a: "checkCert")
    //解析level
    level, _ := strconv.Atoi(txCert.EmailAddresses[0][43:44])
    if levelTag != level {
        fmt.Println( a: "The authentication level doesn't match,level:",level,"levelTag:",levelTag)
        return errors.New( text: "The authentication level doesn't match")
    }

    //解析Tag
    addressType, _ := strconv.Atoi(txCert.EmailAddresses[0][44:45])
    if addressTag != addressType {
        fmt.Println( a: "The authentication address type doesn't match")
        return errors.New( text: "The authentication address type doesn't match")
    }

    //解析时间
    startTime := txCert.NotBefore
    endTime := txCert.NotAfter
    if time.Now().Before(startTime) || time.Now().After(endTime) {
        fmt.Println( a: "The certificate is out of date")
        return errors.New( text: "The certificate is out of date")
    }

    //解析地址
    addr := txCert.EmailAddresses[0]
    if len(addr) < 42 {
        fmt.Println( a: "The authentication address is none!")
        return errors.New( text: "The authentication address is none!")
    }

    if !strings.EqualFold(addrFrom, addr[:42]) {
        fmt.Println( a: "The tx transaction's addr is:", addr[:42], "from", addrFrom)
        return errors.New( text: "The tx source address doesn't match the address in tcert")
    }

    //验证签名
    err := txCert.CheckSignatureFrom(tcaCert)
    fmt.Println( a: "check txCert signature: ", err == nil)
    return err
}
```

3. 普通交易地址合法性审核

在 validateTx()处添加地址合法性校验，校验 from 地址；如果 to（）地址是普通地址则进行合法性校验。如果 To()的地址为空或者为合约地址，则不做过滤。

```go
//check the certificate signature if the transaction is authentication Tx
func CheckAddrLegality(_db *state.StateDB, tx *types.Transaction, _from common.Address) error {
    if !_db.CheckAddrAuthenticateStat(_from) {
        return ErrIllegalSourceAddress
    }

    //Need check dest addr only except contract creation
    if tx.To() != nil {
        if !_db.CheckAddrAuthenticateStat(*tx.To()) && _db.GetCode(*tx.To()) == nil {
            return ErrIllegalDestAddress
        }
    }
    fmt.Println( a: "The normal transaction addr checking passed!")

    return nil
}
```

校验地址的合法性是通过访问 CA 合约的 statDB，检索地址是否已经被 CA 合约记录为合法地址

```go
func (self *StateDB) checkAuthenticateStat(_key []byte) bool{
    //res := self.GetState(_addr, common.HexToHash(hex.EncodeToString(_key)))
    keyString := "0x" + hex.EncodeToString(_key)

    //_addr := common.HexToAddress("0xDd7eF61b67EC080F0EC43b2257143D526c98ec25");
    _addr := common.HexToAddress(common.AuthenticationContractAddressString)
    fmt.Println( a: "GetstorageAt, address:", _addr.Hex(), "key:", keyString)
    res := self.GetState(_addr, common.HexToHash(keyString))
    fmt.Println( a: "The db get result:", res.Hex())

    i, err := strconv.Atoi(res.Hex()[2:])
    if err != nil || i == 0 {
        return false
    }
    return true
}

func (self *StateDB) CheckAddrAuthenticateStat(_addr common.Address) bool {
    key := calculateStatdbIndex(_addr.Hex()[2:],  paramIndex: "7");

    return self.checkAuthenticateStat(key)
}
```

另外需要注意对 internal 交易的修改，合约发起 internal 交易时，要确保 to（）地址不能是不合法的普通地址（可以为空，合约地址或者验证过的合法地址）。修改的地方放在 EVM 执行 transfer 的校验函数 CanTransfer（）函数处，除了校验余额足够的条件，添加地址合法性的校验。

```go
// CanTransfer checks wether there are enough funds in the address' account to make a transfer.
// This does not take the necessary gas in to account to make the transfer valid.
func CanTransfer(db vm.StateDB, addr common.Address, recipient *common.Address, amount *big.Int) bool {
    if recipient != nil {
        if !db.CheckAddrAuthenticateStat(*recipient) && db.GetCode(*recipient) == nil {
            fmt.Println( a: "The internal tx authentication check failed, the recipient:", (*recipient).Hex())
            return false
        }
    }
    return db.GetBalance(addr).Cmp(amount) >= 0
}
```

4. 区块的过滤

另外需要注意到，在交易缓冲池做校验能够避免正常节点抛弃掉不合法交易，但是如果恶意节点将无效交易打包成区块，再做广播，这时正常节点应该要能够识别到这个区块是无效的，不能被纳入到链中。

所以在接收到新区块，进行状态检测时，在 TransactionDB 函数处，添加身份认证交易的监测和普通交易地址合法性的监测

```
// TransitionDb will transition the state by applying the current message and
// returning the result including the the used gas. It returns an error if it
// failed. An error indicates a consensus issue.
func (st *StateTransition) TransitionDb() (ret []byte, usedGas uint64, failed bool, err error) {
    if err = st.preCheck(); err != nil {
        return
    }
    msg := st.msg
    sender := st.from() // err checked in preCheck

    homestead := st.evm.ChainConfig().IsHomestead(st.evm.BlockNumber)
    contractCreation := msg.To() == nil
    if contractCreation == false {
        transactionFormat := msgToTransaction(msg)

        if transactionFormat.IsAuthentication() {
            err = transactionFormat.CheckCertificateSig(msg.From())
            if err != nil {
                return ret: nil,  usedGas: 0,  failed: false, vm.ErrInvalidAuthenticationsig
            }
        }else  {
            if !st.state.CheckAddrAuthenticateStat(msg.From()) ||
                (!st.state.CheckAddrAuthenticateStat(*msg.To()) && st.state.GetCode(*msg.To()) == nil) {
                return ret: nil,  usedGas: 0,  failed: false, vm.ErrIllegalAddress
            }
        }
    }
```

5. 认证交易的 gas 消耗去除

用户新地址的余额 UST 为 0，无法支付矿工费，所以修改身份认证交易的 gasPrice 为 0；

```
// Drop non-local transactions under our own minimal accepted gas price
local = local || pool.locals.contains(from) // account may be local even if the transaction arrived from the network
if !local && pool.gasPrice.Cmp(tx.GasPrice()) > 0 && !tx.IsAuthentication() {
    return ErrUnderpriced
}
```

同时为了避免恶意发送大量身份验证交易对网络进行恶意攻击，在发现是身份认证交易时，如果绑定的地址已经验证成功过则不允许在此进行绑定（待商榷和完善，因为现在身份认证有等级，身份认证过后还可以允许继续做学

位验证？）

```
    //Check the address whether binded already
    if (*pool.currentState).CheckAddrAuthenticateStat(from) {
        return ErrAuthenticationDuplicated
    }
}else {
```

此外，证书服务器对同一个 ecert 可以颁发的 tcert 最大数量也做了限制（目前为 1000 个）