比特币源码研读系列3 创建tx(3)

上篇我们讲到了创建tx需要做下面这些事，并且对步骤1 Find my spendable money中的重要函数进行了解读。

1. **Find my spendable money**
2. **创建tx。**
3. **广播tx**

下面来说说步骤2：创建tx。CreateTransaction方法有点长，我们这篇就来讲讲该方法。

该方法入参vecSend保存着收款人相关信息包括收款金额和收款人的加密地址等，wtxNew就是即将创建的tx。

*bool CWallet::CreateTransaction(const std::vector<CRecipient>& vecSend, CWalletTx& wtxNew, CReserveKey& reservekey, CAmount& nFeeRet,*

*int& nChangePosInOut, std::string& strFailReason, const CCoinControl& coin\_control, bool sign)*

*{*

*CAmount nValue = 0;*

*int nChangePosRequest = nChangePosInOut;*

*unsigned int nSubtractFeeFromAmount = 0;*

***//检查收款人的信息是否有效。***

*for (const auto& recipient : vecSend)*

*{*

*if (nValue < 0 || recipient.nAmount < 0)*

*{*

*strFailReason = \_("Transaction amounts must not be negative");*

*return false;*

*}*

*nValue += recipient.nAmount;*

*if (recipient.fSubtractFeeFromAmount)****//交易费用是否从收款金额里扣除。从后面的代码我们可以看到，如果是从收款人中扣，则是所有收款人平摊该交易费用。***

*nSubtractFeeFromAmount++;*

*}*

*if (vecSend.empty())*

*{*

*strFailReason = \_("Transaction must have at least one recipient");*

*return false;*

*}*

*wtxNew.fTimeReceivedIsTxTime = true;*

*wtxNew.BindWallet(this);*

*CMutableTransaction txNew;*

***//设置nLocktime锁定时间，并做了一点随机性调整。***

*// Discourage fee sniping.*

*//*

*// For a large miner the value of the transactions in the best block and*

*// the mempool can exceed the cost of deliberately attempting to mine two*

*// blocks to orphan the current best block. By setting nLockTime such that*

*// only the next block can include the transaction, we discourage this*

*// practice as the height restricted and limited blocksize gives miners*

*// considering fee sniping fewer options for pulling off this attack.*

*//*

*// A simple way to think about this is from the wallet's point of view we*

*// always want the blockchain to move forward. By setting nLockTime this*

*// way we're basically making the statement that we only want this*

*// transaction to appear in the next block; we don't want to potentially*

*// encourage reorgs by allowing transactions to appear at lower heights*

*// than the next block in forks of the best chain.*

*//*

*// Of course, the subsidy is high enough, and transaction volume low*

*// enough, that fee sniping isn't a problem yet, but by implementing a fix*

*// now we ensure code won't be written that makes assumptions about*

*// nLockTime that preclude a fix later.*

*txNew.nLockTime = chainActive.Height();*

*// Secondly occasionally randomly pick a nLockTime even further back, so*

*// that transactions that are delayed after signing for whatever reason,*

*// e.g. high-latency mix networks and some CoinJoin implementations, have*

*// better privacy.*

*if (GetRandInt(10) == 0)*

*txNew.nLockTime = std::max(0, (int)txNew.nLockTime - GetRandInt(100));*

*assert(txNew.nLockTime <= (unsigned int)chainActive.Height());*

*assert(txNew.nLockTime < LOCKTIME\_THRESHOLD);****//这里我倒是有点困惑，如果大于岂不是会导致始终创建失败？***

*FeeCalculation feeCalc;*

*CAmount nFeeNeeded;***//nFeeNeeded表示因该tx的字节体积所产生的系统费用。目前是1KB字节需要3000** **satoshis的费用，1Btc=100,000,000satoshis.**

*unsigned int nBytes;*

*{*

*std::set<CInputCoin> setCoins;*

*LOCK2(cs\_main, cs\_wallet);*

*{*

*std::vector<COutput> vAvailableCoins;*

*AvailableCoins(vAvailableCoins, true, &coin\_control);*

*// Create change script that will be used if we need change*

*// TODO: pass in scriptChange instead of reservekey so*

*// change transaction isn't always pay-to-bitcoin-address*

*CScript scriptChange;*

***//设置我的找零地址scriptChange。***

*// coin control: send change to custom address*

*if (!boost::get<CNoDestination>(&coin\_control.destChange)) {*

*scriptChange = GetScriptForDestination(coin\_control.destChange);*

*} else { // no coin control: send change to newly generated address*

*// Note: We use a new key here to keep it from being obvious which side is the change.*

*// The drawback is that by not reusing a previous key, the change may be lost if a*

*// backup is restored, if the backup doesn't have the new private key for the change.*

*// If we reused the old key, it would be possible to add code to look for and*

*// rediscover unknown transactions that were written with keys of ours to recover*

*// post-backup change.*

*// Reserve a new key pair from key pool*

*CPubKey vchPubKey;*

*bool ret;*

*ret = reservekey.GetReservedKey(vchPubKey, true);*

*if (!ret)*

*{*

*strFailReason = \_("Keypool ran out, please call keypoolrefill first");*

*return false;*

*}*

*scriptChange = GetScriptForDestination(vchPubKey.GetID());*

*}*

*CTxOut change\_prototype\_txout(0, scriptChange);*

*size\_t change\_prototype\_size = GetSerializeSize(change\_prototype\_txout, SER\_DISK, 0);*

*CFeeRate discard\_rate = GetDiscardRate(::feeEstimator);*

*nFeeRet = 0;* **//nFeeRet表示该tx在付完全部收款金额后的剩余金额。该余额还要扣除到tx的系统费用才是我的找零.**

*bool pick\_new\_inputs = true;*

*CAmount nValueIn = 0;*

*// Start with no fee and loop until there is enough fee*

*while (true)****//注意该循环的结束语句***

*{*

*nChangePosInOut = nChangePosRequest;*

*txNew.vin.clear();*

*txNew.vout.clear();*

*wtxNew.fFromMe = true;*

*bool fFirst = true;*

*CAmount nValueToSelect = nValue;*

*if (nSubtractFeeFromAmount == 0)*

*nValueToSelect += nFeeRet;*

*// vouts to the payees*

*for (const auto& recipient : vecSend)*

*{*

*CTxOut txout(recipient.nAmount, recipient.scriptPubKey);*

*if (recipient.fSubtractFeeFromAmount)*

*{*

*assert(nSubtractFeeFromAmount != 0);*

*txout.nValue -= nFeeRet / nSubtractFeeFromAmount; // Subtract fee equally from each selected recipient*

*if (fFirst) // first receiver pays the remainder not divisible by output count*

*{*

*fFirst = false;*

*txout.nValue -= nFeeRet % nSubtractFeeFromAmount;*

*}*

*}*

*if (IsDust(txout, ::dustRelayFee))****//如果收款金额太小，达到dustRelayFee级别，而且还打算从收款人中扣除交易费用，则肯定扣完就没有了，所以创建tx失败。***

*{*

*if (recipient.fSubtractFeeFromAmount && nFeeRet > 0)*

*{*

*if (txout.nValue < 0)*

*strFailReason = \_("The transaction amount is too small to pay the fee");*

*else*

*strFailReason = \_("The transaction amount is too small to send after the fee has been deducted");*

*}*

*else*

*strFailReason = \_("Transaction amount too small");*

*return false;*

*}*

*txNew.vout.push\_back(txout);*

*}*

*// Choose coins to use*

*if (pick\_new\_inputs) {*

*nValueIn = 0;*

*setCoins.clear();*

*if (!SelectCoins(vAvailableCoins, nValueToSelect, setCoins, nValueIn, &coin\_control))****//使用步骤1中找到的coins。***

*{*

*strFailReason = \_("Insufficient funds");*

*return false;*

*}*

*}*

*const CAmount nChange = nValueIn - nValueToSelect;*

*if (nChange > 0)****//如果有找零并且金额不是太小，则会放到outputs中。***

*{*

*// Fill a vout to ourself*

*CTxOut newTxOut(nChange, scriptChange);*

*// Never create dust outputs; if we would, just*

*// add the dust to the fee.*

*if (IsDust(newTxOut, discard\_rate))****//如果找零金额非常小，达到dust级别则不找零了，赠送给挖矿者。***

*{*

*nChangePosInOut = -1;*

*nFeeRet += nChange;*

*}*

*else*

*{*

*if (nChangePosInOut == -1)*

*{*

*// Insert change txn at random position:*

*nChangePosInOut = GetRandInt(txNew.vout.size()+1);*

*}***//nChangePosInOut指的是找零在vout中的位置**

*else if ((unsigned int)nChangePosInOut > txNew.vout.size())*

*{*

*strFailReason = \_("Change index out of range");*

*return false;*

*}*

*std::vector<CTxOut>::iterator position = txNew.vout.begin()+nChangePosInOut;*

*txNew.vout.insert(position, newTxOut);*

*}*

*} else {*

*nChangePosInOut = -1;*

*}*

***//挑选coins完成后，接下来放入inputs并用我的私钥签名授权使用。***

*// Fill vin*

*//*

*// Note how the sequence number is set to non-maxint so that*

*// the nLockTime set above actually works.*

*//*

*// BIP125 defines opt-in RBF as any nSequence < maxint-1, so*

*// we use the highest possible value in that range (maxint-2)*

*// to avoid conflicting with other possible uses of nSequence,*

*// and in the spirit of "smallest possible change from prior*

*// behavior."*

*const uint32\_t nSequence = coin\_control.signalRbf ? MAX\_BIP125\_RBF\_SEQUENCE : (CTxIn::SEQUENCE\_FINAL - 1);*

*for (const auto& coin : setCoins)*

*txNew.vin.push\_back(CTxIn(coin.outpoint,CScript(),*

*nSequence));*

*// Fill in dummy signatures for fee calculation.*

*if (!DummySignTx(txNew, setCoins)) {*

*strFailReason = \_("Signing transaction failed");*

*return false;*

*}*

*nBytes = GetVirtualTransactionSize(txNew);*

*// Remove scriptSigs to eliminate the fee calculation dummy signatures*

*for (auto& vin : txNew.vin) {*

*vin.scriptSig = CScript();*

*vin.scriptWitness.SetNull();*

*}*

*nFeeNeeded = GetMinimumFee(nBytes, coin\_control, ::mempool, ::feeEstimator, &feeCalc);****//计算该tx所需的系统费用***

*// If we made it here and we aren't even able to meet the relay fee on the next pass, give up*

*// because we must be at the maximum allowed fee.*

*if (nFeeNeeded < ::minRelayTxFee.GetFee(nBytes))*

*{*

*strFailReason = \_("Transaction too large for fee policy");*

*return false;*

*}*

*if (nFeeRet >= nFeeNeeded) {****//如果余额不足以支付系统费用则创建tx失败***

*// Reduce fee to only the needed amount if possible. This*

*// prevents potential overpayment in fees if the coins*

*// selected to meet nFeeNeeded result in a transaction that*

*// requires less fee than the prior iteration.*

*// If we have no change and a big enough excess fee, then*

*// try to construct transaction again only without picking*

*// new inputs. We now know we only need the smaller fee*

*// (because of reduced tx size) and so we should add a*

*// change output. Only try this once.*

*if (nChangePosInOut == -1 && nSubtractFeeFromAmount == 0 && pick\_new\_inputs) {*

*unsigned int tx\_size\_with\_change = nBytes + change\_prototype\_size + 2; // Add 2 as a buffer in case increasing # of outputs changes compact size*

*CAmount fee\_needed\_with\_change = GetMinimumFee(tx\_size\_with\_change, coin\_control, ::mempool, ::feeEstimator, nullptr);*

*CAmount minimum\_value\_for\_change = GetDustThreshold(change\_prototype\_txout, discard\_rate);*

*if (nFeeRet >= fee\_needed\_with\_change + minimum\_value\_for\_change) {*

*pick\_new\_inputs = false;*

*nFeeRet = fee\_needed\_with\_change;*

*continue;*

*}*

*}*

*// If we have change output already, just increase it*

*if (nFeeRet > nFeeNeeded && nChangePosInOut != -1 && nSubtractFeeFromAmount == 0) {*

*CAmount extraFeePaid = nFeeRet - nFeeNeeded;*

*std::vector<CTxOut>::iterator change\_position = txNew.vout.begin()+nChangePosInOut;*

*change\_position->nValue += extraFeePaid;*

*nFeeRet -= extraFeePaid;*

*}*

*break; // Done, enough fee included.*

*}*

*else if (!pick\_new\_inputs) {*

*// This shouldn't happen, we should have had enough excess*

*// fee to pay for the new output and still meet nFeeNeeded*

*// Or we should have just subtracted fee from recipients and*

*// nFeeNeeded should not have changed*

*strFailReason = \_("Transaction fee and change calculation failed");*

*return false;*

*}*

*// Try to reduce change to include necessary fee*

*if (nChangePosInOut != -1 && nSubtractFeeFromAmount == 0) {*

*CAmount additionalFeeNeeded = nFeeNeeded - nFeeRet;*

*std::vector<CTxOut>::iterator change\_position = txNew.vout.begin()+nChangePosInOut;*

*// Only reduce change if remaining amount is still a large enough output.*

*if (change\_position->nValue >= MIN\_FINAL\_CHANGE + additionalFeeNeeded) {*

*change\_position->nValue -= additionalFeeNeeded;*

*nFeeRet += additionalFeeNeeded;*

*break; // Done, able to increase fee from change*

*}*

*}*

*// If subtracting fee from recipients, we now know what fee we*

*// need to subtract, we have no reason to reselect inputs*

*if (nSubtractFeeFromAmount > 0) {*

*pick\_new\_inputs = false;*

*}*

*// Include more fee and try again.*

*nFeeRet = nFeeNeeded;*

*continue;*

*}*

*}*

*if (nChangePosInOut == -1) reservekey.ReturnKey(); // Return any reserved key if we don't have change*

***//对tx引用的inputs进行签名***

*if (sign)*

*{*

*CTransaction txNewConst(txNew);*

*int nIn = 0;*

*for (const auto& coin : setCoins)*

*{*

*const CScript& scriptPubKey = coin.txout.scriptPubKey;*

*SignatureData sigdata;*

*if (!ProduceSignature(TransactionSignatureCreator(this, &txNewConst, nIn, coin.txout.nValue, SIGHASH\_ALL), scriptPubKey, sigdata))*

*{*

*strFailReason = \_("Signing transaction failed");*

*return false;*

*} else {*

*UpdateTransaction(txNew, nIn, sigdata);*

*}*

*nIn++;*

*}*

*}*

*// Embed the constructed transaction data in wtxNew.*

*wtxNew.SetTx(MakeTransactionRef(std::move(txNew)));*

*// Limit size*

*if (GetTransactionWeight(wtxNew) >= MAX\_STANDARD\_TX\_WEIGHT)*

*{*

*strFailReason = \_("Transaction too large");*

*return false;*

*}*

*}*

*if (gArgs.GetBoolArg("-walletrejectlongchains", DEFAULT\_WALLET\_REJECT\_LONG\_CHAINS)) {*

*// Lastly, ensure this tx will pass the mempool's chain limits*

*LockPoints lp;*

*CTxMemPoolEntry entry(wtxNew.tx, 0, 0, 0, false, 0, lp);*

*CTxMemPool::setEntries setAncestors;*

*size\_t nLimitAncestors = gArgs.GetArg("-limitancestorcount", DEFAULT\_ANCESTOR\_LIMIT);*

*size\_t nLimitAncestorSize = gArgs.GetArg("-limitancestorsize", DEFAULT\_ANCESTOR\_SIZE\_LIMIT)\*1000;*

*size\_t nLimitDescendants = gArgs.GetArg("-limitdescendantcount", DEFAULT\_DESCENDANT\_LIMIT);*

*size\_t nLimitDescendantSize = gArgs.GetArg("-limitdescendantsize", DEFAULT\_DESCENDANT\_SIZE\_LIMIT)\*1000;*

*std::string errString;*

*if (!mempool.CalculateMemPoolAncestors(entry, setAncestors, nLimitAncestors, nLimitAncestorSize, nLimitDescendants, nLimitDescendantSize, errString)) {*

*strFailReason = \_("Transaction has too long of a mempool chain");*

*return false;*

*}*

*}*

*LogPrintf("Fee Calculation: Fee:%d Bytes:%u Needed:%d Tgt:%d (requested %d) Reason:\"%s\" Decay %.5f: Estimation: (%g - %g) %.2f%% %.1f/(%.1f %d mem %.1f out) Fail: (%g - %g) %.2f%% %.1f/(%.1f %d mem %.1f out)\n",*

*nFeeRet, nBytes, nFeeNeeded, feeCalc.returnedTarget, feeCalc.desiredTarget, StringForFeeReason(feeCalc.reason), feeCalc.est.decay,*

*feeCalc.est.pass.start, feeCalc.est.pass.end,*

*100 \* feeCalc.est.pass.withinTarget / (feeCalc.est.pass.totalConfirmed + feeCalc.est.pass.inMempool + feeCalc.est.pass.leftMempool),*

*feeCalc.est.pass.withinTarget, feeCalc.est.pass.totalConfirmed, feeCalc.est.pass.inMempool, feeCalc.est.pass.leftMempool,*

*feeCalc.est.fail.start, feeCalc.est.fail.end,*

*100 \* feeCalc.est.fail.withinTarget / (feeCalc.est.fail.totalConfirmed + feeCalc.est.fail.inMempool + feeCalc.est.fail.leftMempool),*

*feeCalc.est.fail.withinTarget, feeCalc.est.fail.totalConfirmed, feeCalc.est.fail.inMempool, feeCalc.est.fail.leftMempool);*

*return true;*

*}*

到此步骤2 创建tx结束，下一篇我们来看看这个新生成的tx是如何广播的。