chaincode

Wallet Development

Chaincode Residency, June 19th 2019

Fair notice

- This presentation is about the Bitcoin Core wallet
- May* contain traces† of C++

*will †oodles

What are a wallet’s functions?

- Key management
- Transaction construction
- Persistence

Key management

- Identify owned transactions
- Generate new addresses
- Determine how to sign transactions

Transaction construction

- Parse addresses and turn them into txOuts
- Coin selection and fee estimation
- Sign inputs
- Advanced features (batching, RBF, CPFP, etc)

Persistence

- Store keys
- Store UTXOs (coins)
- Store transaction history
- Store metadata
  - Labels
  - Blockchain progress
  - etc

Agenda

- Glossary
- Initialization and interfaces
- Code management
- Key management
- Transaction construction
- Persistence
- Future directions

Glossary

- `CPubKey` - a public key, used to verify signatures. A point on the secp256k1 curve.
- `CKey` - a private key, kept secret and used to sign data. In Bitcoin, private keys are scalars in the secp256k1 group.
- `CKeyID` - a key identifier, which is the RIPEMD160(SHA256(pubkey)). This is the hash used to create a P2PKH or P2WPKH address.
- `CTxDestination` - a txout script template with a specific destination. Stored as a variant variable. Can be a:
    - `CNoDestination`: no destination set
    - `CKeyID`: TX_PUBKEYHASH destination (P2PKH)
    - `CScriptID`: TX_SCRIPTHASH destination (P2SH)
    - `WitnessV0ScriptHash`: TX_WITNESS_V0_SCRIPTHASH destination (P2WSH)
    - `WitnessV0KeyHash`: TX_WITNESS_V0_KEYHASH destination (P2WPKH)
    - `WitnessUnknown`: Unknown segwit version (for future segwit upgrades)

Initialization and interfaces

Initialization

- The wallet component is initialized through the `WalletInitInterface`
- For builds with wallet, the interface is overridden in **src/wallet/init.cpp**
- For `--disable-wallet` builds, a dummy interface is defined in **src/dummywallet.cpp**
- The initiation interface methods are called during node initialization

Loading

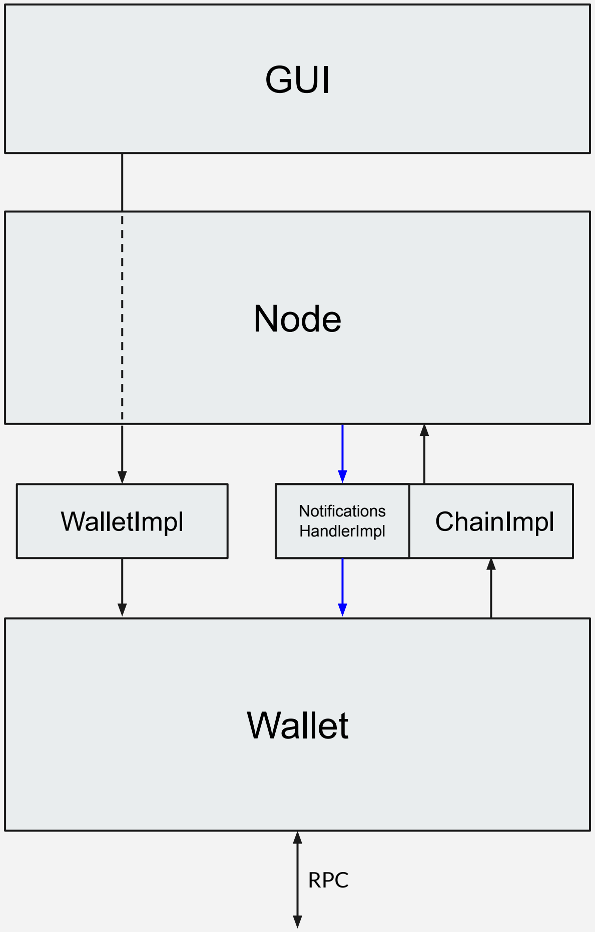
- `WalletInit::Construct()` adds a client interface for the wallet
- The node then tells the wallet to load/start/stop/etc through the `ChainClient` interface in **src/interfaces/wallet.cpp**
- Most methods in that interface call through to functions in **src/wallet/load.cpp**

Node <-> Wallet Interface

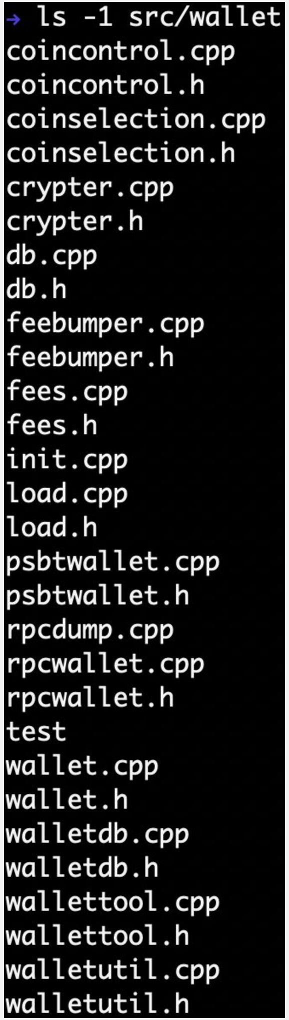
- The node holds a `WalletImpl` interface to call functions on the wallet.
- The wallet holds a `ChainImpl` interface to call functions on the node.
- The node notifies the wallet about new transactions and blocks through the `CValidationInterface`

Why?!

- There are no functional calls between the node and wallet
- Well-defined interface is easier to reason about
- Individual components can be tested in isolation
- Separate wallet into a different process
- Potential for different wallet implementations


GUI
Node
WalletImpl
Notifications HandlerImpl
ChainImpl
Wallet
RPC

Code Management

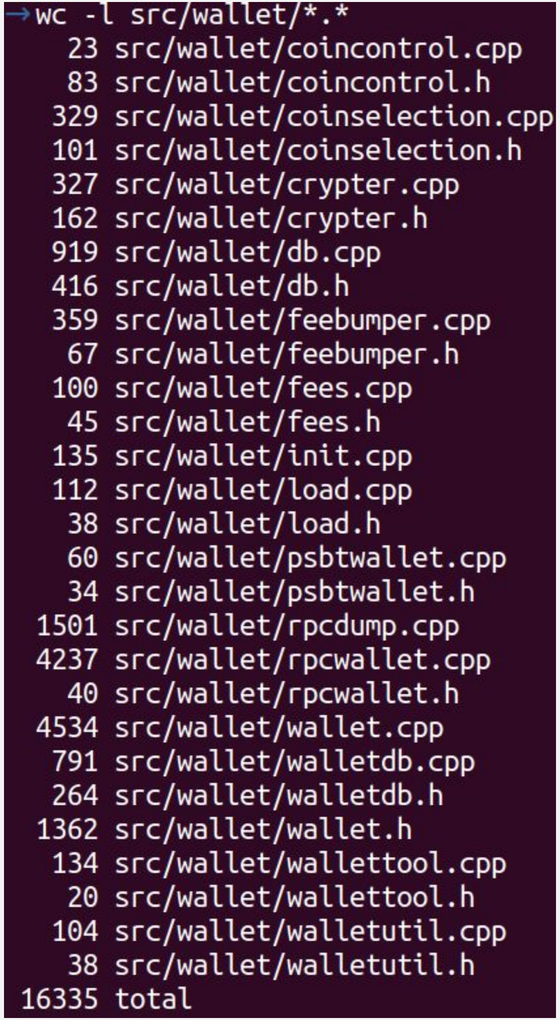


```
→ ls -1 src/wallet
coincontrol.cpp
coincontrol.h
coinselection.cpp
coinselection.h
crypter.cpp
crypter.h
db.cpp
db.h
feebumper.cpp
feebumper.h
fees.cpp
fees.h
init.cpp
load.cpp
load.h
psbtwallet.cpp
psbtwallet.h
rpcdump.cpp
rpcwallet.cpp
rpcwallet.h
test
wallet.cpp
wallet.h
walletdb.cpp
walletdb.h
wallettool.cpp
wallettool.h
walletutil.cpp
walletutil.h
```

Code layout

- **coinselection.cpp|h** - Coin selection algorithm
- **crytper.cpp|h** - encrypting the wallet's private keys
- **[wallet]db.cpp|h** - interface to wallet's database for persistent storage
- **init.cpp** - initializing the wallet module
- **load.cpp|h** - loading/starting/stopping individual wallets
- **rpc*.cpp|h** - wallet's RPC interface
- **wallettool.cpp|h** - standalone wallet tool binary
- **wallet.cpp|h** - EVERYTHING ELSE

- **test/***



```
→wc -l src/wallet/*.*
    23 src/wallet/coincontrol.cpp
    83 src/wallet/coincontrol.h
   329 src/wallet/coinselection.cpp
   101 src/wallet/coinselection.h
   327 src/wallet/crypter.cpp
   162 src/wallet/crypter.h
   919 src/wallet/db.cpp
   416 src/wallet/db.h
   359 src/wallet/feebumper.cpp
    67 src/wallet/feebumper.h
   100 src/wallet/fees.cpp
    45 src/wallet/fees.h
   135 src/wallet/init.cpp
   112 src/wallet/load.cpp
    38 src/wallet/load.h
    60 src/wallet/psbtwallet.cpp
    34 src/wallet/psbtwallet.h
  1501 src/wallet/rpcdump.cpp
  4237 src/wallet/rpcwallet.cpp
    40 src/wallet/rpcwallet.h
  4534 src/wallet/wallet.cpp
   791 src/wallet/walletdb.cpp
   264 src/wallet/walletdb.h
  1362 src/wallet/wallet.h
   134 src/wallet/wallettool.cpp
    20 src/wallet/wallettool.h
   104 src/wallet/walletutil.cpp
    38 src/wallet/walletutil.h
 16335 total
```

Key Management

Identifying owned transactions

- When a transaction is added to the mempool or a block is connected, the wallet is notified through the `CValidationInterface`
- The wallet needs to know if the transaction belongs to it. That happens in `SyncTransaction()`, which calls `AddToWalletIfInvolvingMe()`
- The magic happens in `IsMine()`
- This takes the scriptPubKey, interprets it as a Destination type, and then checks whether we have the key(s) to watch/spend the coin.
- This is overly complicated, inefficient due to pattern matching, not selective, and not scalable.

Generating Keys

- The Bitcoin Core wallet was originally a collection of unrelated private keys
- If a new address was required, a new private key would be generated
- Giving an address out and then restoring from a backup loses funds!

Keypools

- Introduced by Satoshi in 2010
- Cache (100) private keys before they're needed
- When a new public key is needed (either for address or change), draw it from the keypool and refresh the pool
- (Also allows an encrypted wallet to give out an address without unlocking)

HD Wallets

- A minimal HD wallet implementation was added to Bitcoin Core in 2016
- A new HD seed is set on first run or when upgrading the wallet
- Restoring old backups can no longer definitively lose funds (since all private keys can be rederived)
- However, if many addresses were used since the backup, then the wallet may not know how far ahead in the HD chain to look for its addresses
- The keypool essentially became an address look-ahead pool. It is used to implement a 'gap limit'

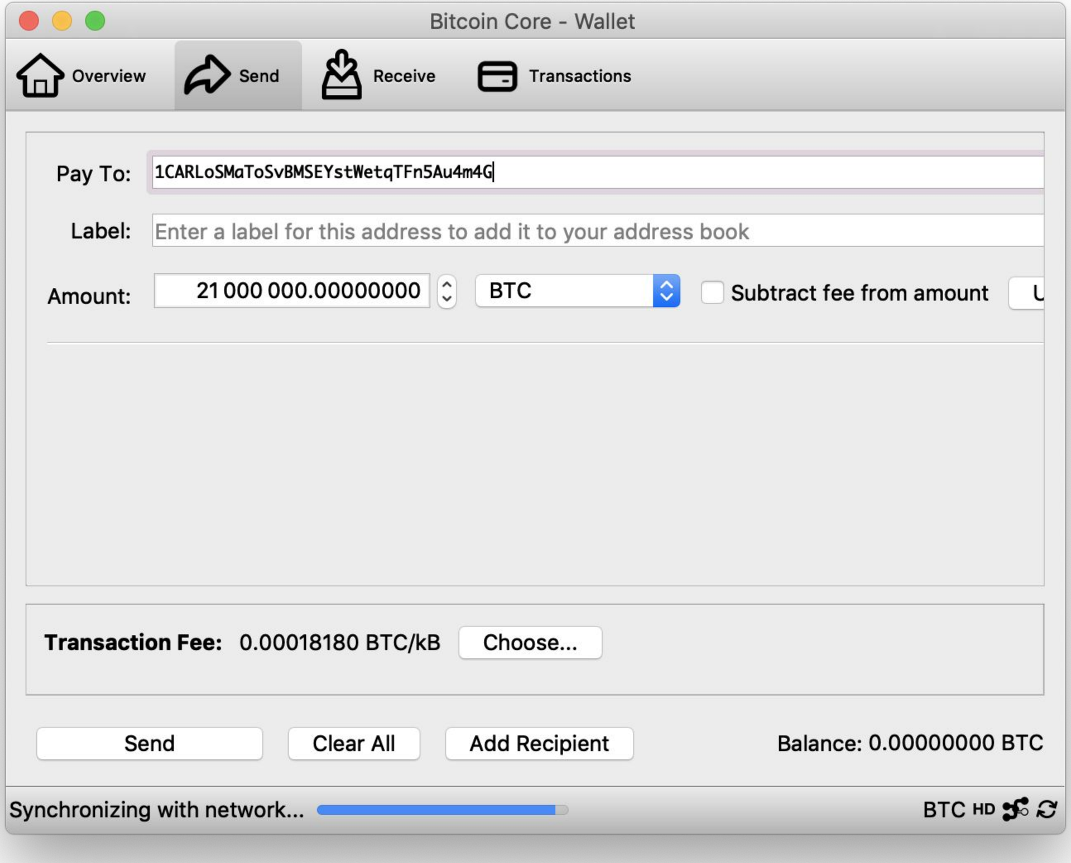
Generating keys (cont)

- For HD wallets, new keys are derived using the BIP32 HMAC derivation scheme
- For non-HD wallets, strong randomness is used to generate a new key
- In both cases, we test the new key by signing a message
- We save the key to the DB before using it

Transaction Construction

Constructing transactions

- Sending from the wallet happens through the RPC or GUI
  - `sendtoaddress`
  - `sendmany`
  - `{create,fund,sign,send}rawtransaction`



Pay To: 1CARLoSMaToSvBMSEYstWetqTFn5Au4m4G

Label: Enter a label for this address to add it to your address book

Amount: 21 000 000.00000000 BTC ☐ Subtract fee from amount

Transaction Fee: 0.00018180 BTC/kB

Balance: 0.00000000 BTC

Synchronizing with network...

BTC HD

Constructing Transactions (cont)

- The address is decoded into a `CDestination`
- Other parameters can be added for finer control (RBF, fees, etc)
- The wallet creates the transaction in `CreateTransaction()`

Coin Selection

- By default, coin selection is automatic
- The logic starts in `CWallet:SelectCoins()`
- By preference, we choose coins with more confirmations
- The actual logic for selecting which UTXOs to use is in **coinselection.cpp**, which implements the branch and bound algorithm
- If that fails, we fall back to using the old `KnapsackSolver`
- Manual coin selection (Coin Control) is possible. See the `CCoinControl` structure

Signing Inputs

- Signing is (almost) the last step in `CreateTransaction()`
- The `CWallet` is an implementation of the `SigningProvider` interface
- The signing logic for the `SigningProvider` is all in **src/script/sign.cpp**

Sending Transactions

- The wallet saves and broadcasts the wallet in `CommitTransaction()`
- The transaction is added to the mempool over the `submitToMemoryPool()` interface method and relayed on the network in the `relayTransaction()` interface method

Persistence

Persistence

- Bitcoin Core wallet uses berkeley db for storage
- **db.cpp|h** is for the low-level interaction with bdb:
  - setting up environment
  - opening/closing database
  - batch writes
  - etc
- **walletdb.cpp|h** is for higher-level database read/write/erase operations.
- bdb is a key-value store:
  - The keys is a type (eg “tx”) followed by an identifier (eg txid)
  - The value is the serialized data
- Object serialization code is in **wallet.h** and **walletdb.h**
- Additional deserialization logic in **walletdb.cpp**

Future Directions

Future Directions

- Descriptor-based wallets
- Hardware wallet integration
- Improve wallet<->node interface
- Process separation
- Different backend storage?
- Re-implementation??

Questions?
Comments?