Philip Glazman
Senior Project

Atlas – C++ Bitcoin Wallet Implementation

Introduction

The aim of this project is to create a user-friendly bitcoin wallet implementation that encourages self-ownership of bitcoins and the use of bitcoin's Script language. This is done by providing the end user the following:

- A bitcoin wallet software that manages the keys and addresses.
- A friendly-to-use graphical interface.
- A dashboard to track analytics about payments and fees.
- A dashboard to configure custom scripts.

Background

Bitcoin is a digital money developed in 2009 where each node participating in the network can independently validate transactions and propagate them throughout the network using software similar to bittorrent. The protocol relies on public-key cryptography to create public addresses for the end-user. In terms of bitcoin, a wallet software manages the private keys that are associated with each public address. These keys gives users ownership in spending transactions and bitcoin. In bitcoin, the wallet is an abstraction that allows the end-user to send and receive payments.

This project aims to create a new wallet following security standards that can give the end-user an ability to create new transactions using a custom script dashboard as well as track payments using an analytics dashboard.

More on bitcoin: https://bitcoin.org/en/

Specification

The design of the wallet will follow object-oriented principles. It will mostly be written in C++ with the exception of embedded python code used in the analytics dashboard described above. The Qt framework will be used to write a cross-platform GUI. Libbitcoin Library and Boost Library be dependencies. I am including Libbitcoin for its useful math and cryptographic methods. Boost is being included for its useful methods and as a dependency for Libbitcoin.

Boost: http://www.boost.org/
Libbitcoin: https://github.com/libbitcoin/libbitcoin/wiki
Qt: https://www.qt.io/

Base Wallet

Components	Description
Mnemonic Code Words	Following a wallet standard, generated entropy will translate to 12 English words from a set. These words in addition to salt, will lead to a seed that creates a unique wallet. These 12 words could be written down and entered into the wallet to create this same unique wallet.
HD Wallet	A HD wallet, or deterministic wallet, is a wallet that creates a keychain based on a 512 bit seed. This is a standard in current Bitcoin wallets.
Bloom Filter	Bloom Filter is a standard privacy feature that allows the user to query for transactions without revealing to the network the specific transactions that he/she is asking for.
Peer Networking	Wallet will connect to peers, do hand-shaking, and ask for transaction data. Most of the low-level work will be handled by Libbitcoin library.
Payments	Allow user to send bitcoin and see the amount of bitcoin received. Allow user to generate new unique address for when receiving bitcoin
Graphical User Interface	Wallet will have a Qt built GUI.

Analytics Dashboard

Components	Description
Fee Estimation	Query network to determine low, high, and median transaction fees. Provide recommendation to user for fee cost and when to send transaction.
Spend Analysis	Provide user with information on where bitcoin have been spent.

Script Dashboard

Components	Description
Console	User can write their own bitcoin scripts and send them as transactions. Bitcoin Script language is stack-based language with limited OP codes.
Script debugger	Include debugger to help catch any errors in the user's script.

Schedule

This is the schedule I will use to regulate myself with in order to complete the project by late April. Week date is the week ending Sunday date.

Week Date	Completed Item
2/11	Algorithms and models should be written. UML Diagram and class definitions.
2/18	Wallet should have core functional features. Implement: • HD wallet design. • Seed generation. • Mnemonic Code Words.
2/25	Wallet should allow user to send and receive bitcoin. Implement: • Payments.
3/4	Wallet should have privacy standards and connect with peers. Implement: • Bloom Filter. • Peer Networking.
3/11	Wallet should have script console and debugger completed. User can begin writing scripts. Implement: • Script Debugger. • Script Console.
4/1	User can interact entirely with core wallet using a graphical user interface. Implement: • GUI for Core Wallet
4/8	User can write scripts using a graphical user interface. Implement: • GUI For Script Dashboard.
4/15	User can interact with analytics dashboard: Implement: • Fee Estimation. • Spend Analysis. • Embedded Python Scripts in GUI.
4/22	All bugs should be resolved or noted. Wallet should be fully functioning and ready for use.