CISC 322

Assignment 2 Report

BitCoin Core: Secure Digital Wallet

March 23, 2023

Group 44 :

Victor Coombes (20vmwc@queensu.ca)

Osamu Adun (19osa2@queensu.ca)

Anton Gudonis (19adg8@queensu.ca)

Selena Kesidis (21smk13@queensu.ca)

Navin Pandey (17np1@queensu.ca)

Kyle Verma (18kv10@queensu.ca)

**Table of Contents**

**Abstract Pg 03**

**Introduction and Overview Pg 03**

**Architectural Style Pg 04**

**Architecture Overview Pg 05**

**Component Breakdown Pg 07**

**Interaction Breakdown Pg 08**

**Use Cases Pg 09**

**Inner Architecture of a Subsystem (Conceptual + Concrete) Pg 11**

**Reflexion Analysis for High-Level Architecture Pg 12**

**Reflexion Analysis for Subsystem Pg 13**

**Data Dictionary Pg 13**

**Naming Conventions Pg 13**

**Lessons Learned Pg 13**

**Conclusions Pg 13**

**References Pg 14**

**Abstract**

This report will examine the concrete architecture of Bitcoin Core as recovered through Bitcoin project source code. ­­­­­The purpose of this analysis is to identify and discuss divergences with respect to the conceptual architecture from our previous report. Upon grouping the top-level entities of the system into subsystems it is clear that there are several similarities and differences between the conceptual architecture and the actual implementation of Bitcoin Core. Nonetheless, the peer-to-peer architectural style remains as the foundation for Bitcoin Core's concrete architecture, while also integrating elements of both publish/subscribe and client-server styles in the system. This report will further discuss the observed top-level concrete subsystems of Bitcoin Core from the codebase through the process in which they were derived and how they interact with each other. In addition, the inner architecture of the Wallet subsystem will be analyzed to compare and contrast conceptual and concrete views. This will be conducted through reflection analysis on both the high level architecture and 2nd level subsystem. To further illustrate Bitcoin Cores concrete architecture, we will provide two sequence diagrams for relevant use cases and reflect on lessons learned from deriving the concrete architecture.

**Introduction and Overview**

In the previous report, the conceptual architecture of Bitcoin Core was discussed in thorough detail. This allowed for an understanding of meaningful relations within the software’s system. It provided a high-level, abstract design of Bitcoin Core and highlighted its overall vision and goals. It was discovered that Bitcoin Core operates in a peer-to-peer architecture style to provide users with a first-party, open-sourced wallet solution that additionally allows users to mine bitcoin if desired. Analyzing the conceptual architecture proved to be very valuable in the beginning of this project as it allowed for previously inexperienced developers to gain an idea of how the algorithms of the system works.

In this report, Bitcoin Core’s concrete architecture will be discussed. This refers to the detailed, specific implementation of a system including, actual technologies and software components used to build and deploy its system. Concrete architecture refers to the actual relations that exist within a system and is mainly expressed through a product’s source code and configuration files. While, a conceptual architecture provides a big-picture view of a system, a detailed concrete architecture report should provide the “nuts-and-bolts” details a developer would need to actually build and implement a system. By analyzing Bitcoin Core’s concrete architecture, developers are able to further their understanding of how its system works and functions. This has been done by installing the Understand tool and opening Bitcoin Core’s source code on it to view its various components.

Specifically, the subsystem relating to the wallet of Bitcoin Core was selected for analysis. This is due to the fact that the wallet subsystem plays a crucial role in the security and usability of Bitcoin by providing a secure and easy-to-use interface for managing a user’s funds on the network. Its primary focus is to securely store and manage a user’s private keys, which are used to sign transactions and prove ownership of Bitcoin addresses. Additional functions of the wallet subsystem includes, balance tracking, address book management and, backup/recovery.

From the conceptual architecture report, it was deduced that there were 13 main components within Bitcoin Core’s system: app/GUI, RPC, wallet, storage engine, headers, blocks, coins, validation engine, Mempool, miner, connection manager and peer discovery. Throughout the dissection of Bitcoin Core’s source code, it was found that these components exist within the system. However, some are compacted into a single component. In this case, peer-to-peer network, connection manager and, peer discovery are combined into the new component, network. There are also two newly discovered components that were previously unaccounted for named, utils and performance evaluation.

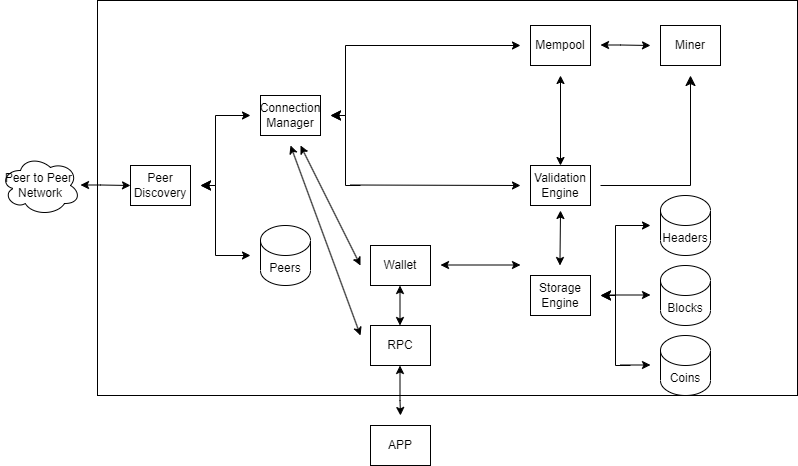
One of the most proficient uses of a concrete architecture is to compare it to its conceptual architecture and see if there are any unexpected changes to the relations of the components. The two main categories of differences are absences and divergences. Absences are known as relations that existed in the conceptual architecture but not in concrete while divergences are vice versa. In the case of Bitcoin Core, there were no notable absences and only a few divergences mostly due to the fact that multiple components are merged to create a network.

Thus, the analysis of Bitcoin Core’s concrete architecture and focus on the wallet subsystem has provided a deeper understanding of the implementation used to build and deploy Bitcoin Core. The discovery of new components and increased comprehension of older ones reinforce the value of examining concrete architecture in order to fully comprehend the inner workings of a system. Overall, our group has learned how complex the concrete derivation process can be when compared to conceptual architecture and have gained a more thorough understanding of Bitcoin Core’s implementation through analyzing the dependency file.

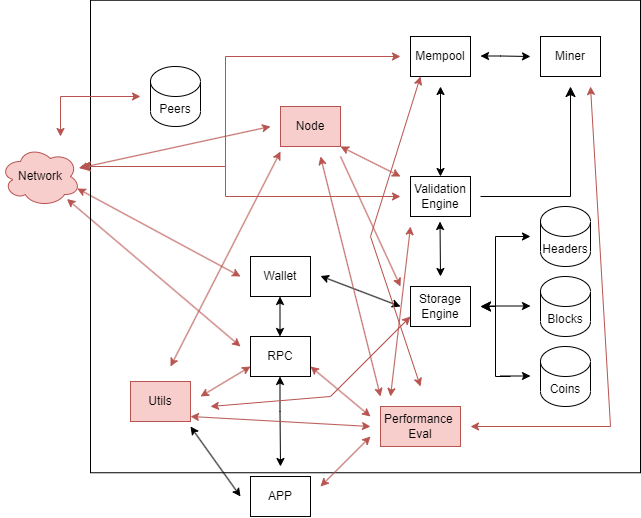
**Architectural Style**

The architecture of the Bitcoin Core system is a peer to peer style architecture. This style allows for a decentralized system that users and consumers expect from Bitcoin. The appeal of Bitcoin for many is that all aspects of it are decentralized and no one entity has control of the system. Bitcoin Core deals with Bitcoin and must offer the same protections that are native to it. The system is reliant on peers connecting with each other and exchanging information without any central server that stores ledgers or information. Users connect with other users and use the resources of each other.The architecture also has components of both Publish/Subscribe and Client Server in the system. Several of the components interact in a Publish/Subscribe system and the Client Server systems handle front to back end operations.

**Architecture Overview**

  
(Conceptual architecture of the Bitcoin Core system)

Through examination of both the conceptual architecture shown above and the provided source code of Bitcoin core, we derived a concrete architecture.



(Concrete Architecture of the Bitcoin Core system)

**Architecture Derivation Process**

The Architecture Derivation Process for creating a concrete architecture from the source files of Bitcoin Core begins with a conceptual architecture. The conceptual architecture provides a high-level overview of the system and its components, defining their interactions and dependencies. This conceptual architecture was used as a base for the top level subsystems proposed for the concrete architecture. The next step is to analyze the source code of Bitcoin Core to identify its components, their functionalities, and their interactions. The mapping of the source files to the proposed top level subsystems consisted of identifiable naming conventions, official documentation, analysis of code to understand its purpose. Additionally, a presentation detailing the top level subsystems by a primary developer of bitcoin core helped refine the derivation process. Although initially the top level subsystems were intended to differ only slightly from the proposed conceptual architecture it was decided to merge certain subsystems due to key files performing actions for multiple subsystems. Additionally new top level subsystems were proposed to account for systems not considered within the conceptual architecture. Once the source files were mapped to the concrete architecture the dependencies between the top level subsystems were analyzed using the understand tool to identify crucial dependencies for the final iteration of the proposed top level concrete architecture.

It was decided to analyze the second level subsystem of the wallet component of bitcoin core. Using various online articles discussing the architecture of bitcoin wallets a set of second level subsystems was proposed. Once the second level subsystems were decided on, the files within the wallet top level subsystem were further mapped using the same methods discussed previously when deriving the top level concrete architecture.

**Component Breakdown**

***New components of concrete architecture***

**Network:** The Network is a component that serves the combinatorial functions of the Peer to Peer Network, Connection Manager, and P2P Discovery. Those three become subsystems of the Network, which replaces them in the concrete architecture.

**Utils:** The Utils component provides different utilities to various other systems, it compresses, decompresses, encrypts, and decrypts data

**Performance Evaluation:** The Performance Evaluation collects information on system performance and analyzes them to optimize the system

**Node:** The Node maintains and holds the state of the Bitcoin Network for validation of transactions

***Previously stated components (Assignment 1)***

**App:** The GUI allows users to interact with Bitcoin Core. It allows users to graphically interact with and manage their wallets, send transactions and view transactions, on the network.

**RPC:** The remote procedure call interface enables users to access the API’s provided by bitcoin core to retrieve additional information regarding the blockchain.

**Wallet:** The wallet component component is responsible for allowing users to manage their bitcoins. It is responsible for creating and managing passwords, keeping track of bitcoin amounts within the wallet, and keeping a record of sent and received transactions to the wallet.

**Storage Engine:**

The storage engine component of Bitcoin Core is responsible for storing and retrieving Bitcoin blockchain data, such as transaction records, blocks, headers, and other relevant information.

**Headers, Blocks, and Coins:**

The Headers, Blocks, and Coins components are responsible for storing and managing various blockchain data for the storage engine. Headers contain metadata about each block, such as its hash and the time of creation. Blocks contain the transaction data and are used to verify the integrity of the blockchain. Coins are the units of value that are transferred between Bitcoin addresses, and each coin is associated with a transaction output.

**Validation Engine:**

The Validation Engine component of Bitcoin Core is responsible for ensuring the integrity of the blockchain data.

**Mempool:**

The Mempool component of Bitcoin Core is responsible for storing the unconfirmed transactions that have not yet been included in a block.

**Miner:**

The Miner component of Bitcoin Core is responsible for creating new blocks in the blockchain.

**Interaction Breakdown and Discrepancies**

***New interactions of concrete architecture***

Util -> Storage Engine: Utils is responsible for both encrypting and decrypting data retrieved from the Storage Engine.

Util -> RPC: Utils needs access to the RPC so it can communicate with the servers to access user data that it can perform its utility functions on.

Util -> App: The App contains various types of data that the Util can perform actions on, from compressing data for easier storage later to encrypting sensitive data.

Performance Evaluation -> App/RPC/Util/Mempool/Miner/Validation Engine/Node: The Performance Evaluation needs to interact will all of the listed systems as it retrieves performance data with the goal of optimizing their respective functions from faster and more efficient processing of blockchain transactions to faster block creations and efficient use of computing power.

Node -> Network: Node interacts with Network to retrieve the current status of the blockchain from the p2p network. This ensures the node is up to date so it can properly validate, send and receive transactions

Node -> Util: Node utilizes several utility functions related to encryption and decryption

Node -> Storage Engine: Node stores relevant data relating to the local node in the storage engine db. This includes the latest snapshots of the blockchain and its various blocks.

Node -> Validation Engine: Node crucially interacts with the validation engine to perform consensus checks towards the blockchain. Consensus checks are performed by all nodes on the p2p network to ensure the blockchain can not be manipulated by individual users. Additionally, node uses consensus checks to ensure when downloading new information from the blockchain that the data is valid and not manipulated.

***Previously stated interactions (Assignment 1)***

App -> RPC: The Bitcoin Core application provides the user interface for interacting with the RPC interface, which enables programmatic interaction with the Bitcoin Core software.

App -> Wallet: The Bitcoin Core application allows users to create and manage their Bitcoin wallets, which are used to store private keys and other information related to their Bitcoin transactions.

Wallet -> Storage Engine: The wallet component relies on the storage engine to store and retrieve blockchain data, including the transaction records, blocks, headers, and other relevant information.

Storage Engine -> Headers: The storage engine component stores the header information associated with each block in the blockchain.

Storage Engine -> Blocks: The storage engine component stores the full transaction data for each block in the blockchain.

Storage Engine -> Coins: The storage engine component stores information about the individual coins that are transferred between Bitcoin addresses.

Validation Engine -> Blocks: The validation engine component is responsible for ensuring that the blockchain data stored in the blocks is authentic and correctly formatted.

Validation Engine -> Coins: The validation engine component verifies the authenticity and validity of each coin transfer.

Mempool -> Validation Engine: The Mempool component sends unconfirmed transactions to the validation engine component for verification before they are included in a block.

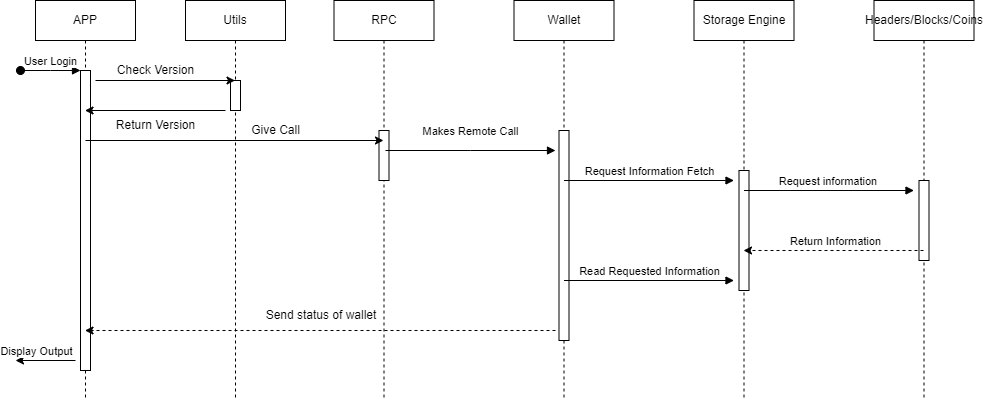
Miner -> Mempool: The miner component selects transactions from the Mempool to create a new block to add to the blockchain.

Miner -> Validation Engine: The miner component must validate the authenticity and format of the transactions it includes in a new block before adding them to the blockchain.

**Use Cases**

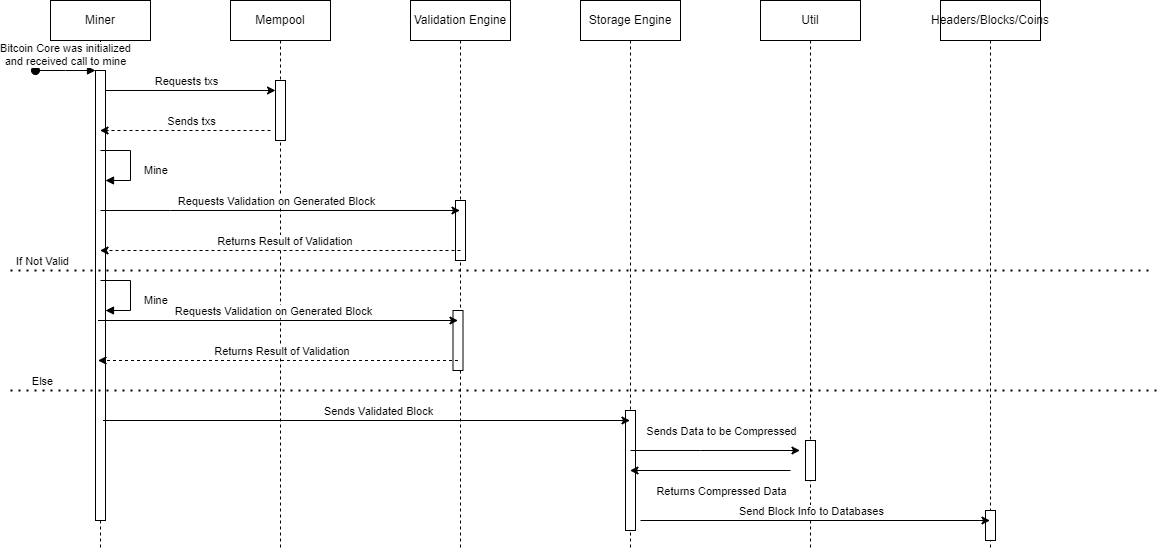
The two use cases we will be investigating are:

1. A user wants to log in and access their wallet and view the information inside.



When a user wants to login, the App sends Util to check version and other utility functions, once that is returned, App sends the RPC a call that it wants the RPC to give the Wallet. The Wallet requests an information fetch from the Storage Engine, which fetches the requested information from the correct database. The Storage Engine receives that information, which is read by the Wallet and sent back to the App to be displayed to the User.

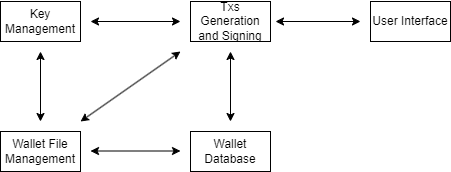
1. A user wants to mine and mint a new Bitcoin.



When a user wants to mine and mint a new coin, the Miner receives a call to mine. The Mempool receives a txs request and returns it back to the Miner. The Miner then performs the mining action, until it needs validation on the created block. It requests validation from the Validation Engine and the validation status is returned. If it's invalid, the process repeats and then validation is checked again. When it's valid, the block is sent to the Storage Engine. The Storage Engine sends the data to Utils, which compresses the data for more efficient storage in the databases. The Storage Engine receives the compressed data back and sends it to the appropriate database for storage.

**Inner Architecture of a Subsystem (Conceptual + Concrete)**

The Wallet is the subsystem that we have decided to investigate further.



(Conceptual Architecture of the Wallet Subsystem)

Key Management subsystem: The Key Management subsystem is responsible for maintaining and managing the keys that associate with the Bitcoin addresses of the user. It can generate new keys, import pre existing keys, and securely store keys in memory or on disk.

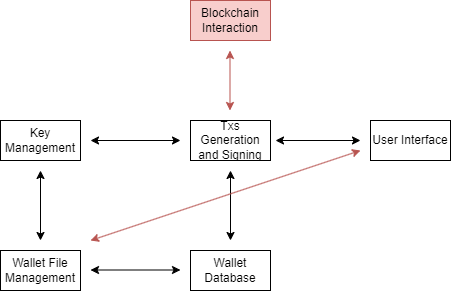
Transaction Generation and Signing subsystem: The Transaction Generation and Signing subsystems are responsible for generating and signing all transactions using the private keys of the users Bitcoin addresses. It can select appropriate inputs to send, sign transactions with the relevant keys, and build transactions inputs and outputs.

Blockchain Interaction subsystem: The Blockchain Interaction subsystem interacts with the Bitcoin network and the blockchain to obtain user wallet information like transaction history and balance. It also updates the user's wallet according to incoming and outgoing transactions, monitors for new transactions, and requests information from the blockchain about transactions.

Wallet Database subsystem: The Wallet Database subsystem manages and stores user wallet data. The data stored varies from transaction history to balance information. The other functionalities include creating new wallet files, backing up wallet files in event of data loss, and decrypting wallet files.

User Interface subsystem: The User Interface subsystem provides the user a simple and compelling user interface that they use to interact with the wallet and its components. It can display the wallet balance, the address book, and transaction history, while also displaying features responsible for sending and receiving Bitcoin.

Wallet File Management system: The Wallet File Management system is responsible for managing the wallet files. It adds new wallets, loads previously interacted with wallets, and maintains all files associated with the wallet.

****

(Concrete Architecture of the Wallet Subsystem)

**Reflection Analysis for High-Level Architecture**

The concrete architecture of Bitcoin Core was designed using a combination of the conceptual architecture from assignment one and the tools provided to analyze the source code of the system. Despite initially mapping directly to the conceptual architecture a number of gaps were observed specifically relating to the purpose of several files, this resulted in the creation of various new top level subsystems to represent these files. Additionally, the organization and purpose of several files resulted in the decision to merge multiple subsystems into one encompassing subsystem named Network. This was primarily due to the fact that several network based files completed actions related to managing connections, discovering peers and managing the p2p network all in a single file which made splitting the source code into subsystems impossible. On another note, the top level subsystems named Util and Performance Evaluation were also created. Despite this gap the explanation is reasonable as testing, logging and utility functions of the systems were not considered key components for the applications function during the creation of the conceptual architecture. Regardless, these files compose a large portion of files within the source code and therefore the new top level subsystems must be created. Lastly, a large majority of the source code revolved around maintaining the local node resulting in the need for an additional top level subsystem named node. This gap between the concrete and conceptual architecture is also explainable as during the creation of the conceptual architecture the maintenance of the local node was assumed to be handled by the various different subsystems in parts. Conversely, the source code contained dedicated files for maintaining the node and due to the large amount of such files a new subsystem was dedicated to this function.

**Reflection Analysis for Subsystem**

The top level wallet subsystem contained few gaps between the conceptual and concrete architecture. Specifically, a missing subsystem and one dependency. Within the conceptual architecture the Blockchain Interaction subsystem was not considered as it was proposed that the transaction generation and signing subsystem would perform interaction with the blockchain due to it being a small component. On the contrary, source files were located purely dedicated to interaction with the blockchain and therefore the new subsystem was created within the concrete architecture. Additionally, a dependency between user interface and wallet file management was overlooked. It was proposed that the user interface would purely interact with transactions and that the system/app would handle the management of wallet files. Conversely, analyzing the code within the files directly showed correlation/interaction between the two systems for the purpose of creating and loading new wallets from the database.

**Data Dictionary**

ZMQ: ZeroMQ is a messaging library that provides an asynchronous message queue.

QT: QT is an application for creating graphic user interfaces.

**Naming Conventions**

In our report, the phrase “tx” stands for transaction. “P2P” stands for peer to peer, while “GUI” stands for graphical user interface. Additionally, “RPC” stands for remote procedure call.

**Lessons Learned**Our group has learned that the concrete architecture and all of its subsystems are far more complex than the conceptual architecture. We have learned how much goes into a software architecture, how many files, subsystems, and systems have to interact to create a seamless user experience, regardless of software. A concrete architecture is a lot to get your head around, even with software to help. There was a lot to learn from Assignment 1 to Assignment 2. A1 gave a much more broad architecture than we realized, and the depths and intricacies of the concrete architecture are proof.

**Conclusions**

To conclude, we have compared and contrasted the conceptual and concrete architectures of Bitcoin Core and discovered key dependencies that help us understand the software architecture as a whole. The new roles adopted by the Network component in the concrete architecture show us that it is a crucial aspect of the software. Additionally, the new dependencies discovered through the new components help us conceptualize the Bitcoin Core system even better than before. Overall, we have further analyzed and understood the Bitcoin Core software architecture.

**References**

Bitcoin Core Website:

<https://bitcoin.org/en/bitcoin-core/>

Review of Bitcoin Core wallet:

<https://wallets.com/bitcoin-core-review/>

Bitcoin Architectural Overview:

<https://dl.gi.de/bitstream/handle/20.500.12116/16570/DFN-Forum-Proceedings-001.pdf?sequence=1&isAllowed=y>

An Architectural Overview of the Bitcoin Blockchain  
<https://dl.gi.de/bitstream/handle/20.500.12116/16570/DFN-Forum-Proceedings-001.pdf?sequence=1&isAllowed=y>