Bilkent University

Department of Computer Engineering

Senior Design Project

CrypDist

Project Analysis

Mehmet Furkan Şahin
Oğuz Demir
Turan Kaan Elgin
Onur Uygur
Gizem Çaylak

Supervisor: Can Alkan
Jury Members: İbrahim Körpeoğlu and Ercüment Çiçek

Analysis Report
November 7, 2016

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2

Table of Contents

Project Analysis

CrypDist

1. Introduction

Data management has a great importance for most of the companies to maintain their business. Since technology is developing and companies are growing every day, it creates a demand to manage a huge amount of data. Managing huge amount of data is still an on-going research field and engineers are working on this field to provide secure and reliable systems with faster response time to people. One of the methods to store the data is data distribution which distributes the data on different servers to manage the data. Data encryption and data replication can be used together with data distribution to ensure the secure management of distributed data and the reliability of the system.

Our project, CrypDist, aims for maximizing the performance of the delivery of data and providing fast access for the users. The main purpose of our project is providing genotype and phenotype data to the researchers to enable them to conduct advanced research by analyzing and updating the data. We use blockchain to ensure safety and reliability of continuously growing data.

2. Current System

The blockchain is the key element of the digital currency system bitcoin. The blockchain is a distributed database. Bitcoin has a decentralized system because blockchain stores the data across its nodes of network. Bitcoin uses blockchain to record the bitcoin transactions between users. Network of communicating nodes provide maintenance (replication of the transactions and sending the transaction information to other nodes) for the blockchain. When a new transaction is added to a node, this transaction is broadcasted to other nodes. Each node in the network stores its own copy of blockchain. Although bitcoin uses blockchain to store the information about transactions, there is no current system keeping abstract data of human genome in its blockchain.

3. Proposed System

3.1. Overview

CrypDist is a blockchain based distribution application that will be created with Java on the both client and server side. It is also going to use Akamai to handle the distribution part as an off the shelf component. CrypDist is going to use NetStorage facility of Akamai to store the raw data and it is going to distribute it with the help of Akamai Edge servers.

On the client side, CrypDist will be able to provide a summary data in form of a blockchain so that the user will be able to understand the content of the raw data itself without looking at the exact data. Blockchain structure is a data structure used for storing continuously growing list of records without tampering or revision. Hence, by the help of blockchain structure, CrypDist will be able to maintain the data dynamically and the synchronization will be handled automatically. Clients will be able download whatever part of the data they need, manipulate it and upload it back to servers to be reached in future as a new version. However, the data in the blockchain cannot be manipulated by anyone. In case of data addition, a block containing the abstract information will be added synchronously to blockchain on clients and the actual data will be distributed among servers. Thus, the encryption of the data will be already done by blockchain meaning that the user will never know where the data sent but he is always going to have the information in his own summary, blockchain. However, the raw data will already be open to anyone who has the blockchain. Here, CrypDist classifies the raw data to two different classes; one will include the genomics data which will be open and the second will include phenotype of the owner of the data which will be kept encrypted because it contains sensitive personal data. Thus, when a client wants to reach phenotype data, he will need be authenticated by our services.

The main functionality of CrypDist is that it avoids the centralized database architecture so that we can have a democratized data distribution system which avoids any block and always up to deliver the data as possible as quickly thanks to Akamai services. Our first and main focus is to manage genomics data since demand to work on that specific data is huge such that many researchers are working to avoid future possible mutations by examining the outputs of the past research.

The problem which CrypDist is planning to solve is currently tried to be handled with dbGaP system, which provides centralized database of Genotypes and Phenotypes. Using dbGaP has several disadvantages: Firstly, using one centralized server for taking data will be slow, when number of clients and size of the data is considered. Secondly, dbGaP stores all the data secure and getting access to this data requires several diplomatic steps, which takes very long time. However, the accessibility of Genotypes data does not need to be controlled because it does not contain any personal information and it is needed for statistical data. As a result, currently researchers are having problem for accessing the secure Genotypes data, which should have been public. Lastly, since the data is stored in centralized database, crashes, in-tolerated faults or political issues (e.g. shut down applications by governments) on the servers may cause loss of data, which the Wellcome Trust users had encountered in the U.K.

Previously, blockchain structure was used in genomics for distributing CPU load between machines by exchanging bitcoins with process power (http://dnadigest.org/a-new-multi-centralized-cryptocurrency-coinami/). However, CrypDist is the first application using blockchain for data distribution.

3.2. Functional Requirements

- Each client will have a copy of the blockchain.
- Each client will be able to display content of the blockchain.
- Each client will be able to add data to the blockchain.
- Each client can upload raw data or update already exist data via version control.
- Each client will be able to specify the type of the data as confidential or public to upload the data.
- Each client will be able to specify total number of copies of data.
- Each client will be able to write a SQL query to request data.
- Each client will be able to access public data.
- Only permitted clients will be able to access confidential data via authentication system.
- Client will be able to list query results and select one of the results to download data.
- Depending on the location of the request and content of the requested data, the client will be able to get data from the most suitable server, in terms of distance and availability of data.

- After processing the downloaded data, the client will be able to update the data as upper version of old data.
- During an erroneous situation on a server, the client will be able to continue its operation without corruption by providing the flow of data from other servers.

3.3. Non-Functional Requirements

Availability: The system service shall be available for use 24 hours per day, 365 per year owing to the distributed architecture compared to centralized architecture via distribution and replication of data to servers and quick recovery procedure when a failure is detected.

Continuity: The system shall be able to handle major interruptions such as power outages of a server by continuing operation on other services.

Portability: Since the system will be written in a high-level programming language, it shall be usable in different environments.

Recoverability: The system shall restore the data in the event of corruption or loss by copying the replicated data from other servers.

Response time: Via distributed architecture, response time shall be decreased compared to centralized architecture by locating the request location and sending the data to requester from nearest server.

Reusability: The system shall be reused across multiple products to store and retrieve the data.

Robustness: The system shall be able to cope with the errors during execution time.

Scalability: The system shall be practical and efficient when applied to large input data by adding new resources such as server.

Security: Security problem of the distributed systems shall be resolved by encrypting the data. Only permitted clients shall be able to access to confidential data by authentication system.

Transparency: The block chain part of the system shall be transparent while the resources where actual data is stored shall be hidden from client.

Usability: %95 of the novice users can learn to operate major use cases without outside assistance.

Resource constraints*:* To meet the work load of the system minimum hardware requirements for each machine should be:

- *Minimum disk space:* 40 GB
- *RAM:* 4 GB
- *CPU:* 2 cores

3.4. Pseudo Requirements

- The bandwidth requirements of the blockchain must be lower than the capacities of about 75% of the network nodes in the system. (If the requirements are too high, only some of the nodes will be able to process blocks which will lead to centralization of control.)
- The timestamp contained in each block, which indicates its creation time, must have at most three hours of deviation.
- There must be a uniform access among all network nodes such that no network node must wait for another node to transmit more than one block.
- The blockchain must not accept a transaction which has more data than the maximum size of a block.
- The blockchain must not accept a block which does not contain a particular key which indicates its validity.
- The time required for creating a new block must be less than 20 minutes.
- The block size must not be more than 1 MB.
- The blockchain size must not be more than 40 GB.
- The contents of the transactions must not be open to third parties.
- There must not be any method for decrypting the cryptography algorithm used for transactions except for brute force attack.
- Brute force attack to decrypt the cryptography algorithm must theoretically require at least 10000 years to execute.
- The speed of encryption algorithm must be at least 20 MB/s.
- Blocks should not be deleted or altered by the clients.
- Only authenticated clients should be able to access the confidential data.
- The user passwords should be encrypted by MD5 algorithm.
- The blockchains of each client should be notified within 2 seconds to be updated when new data added to the chain by any client.
- The time required for querying a server and returning the results must be less than 15 minutes.
- During an erroneous situation on a server, the latency of response time should not exceed 15 seconds.
- The query to request data should be written in SQL language.

3.5. System Models

3.5.1. Scenarios

Query Confidential Data

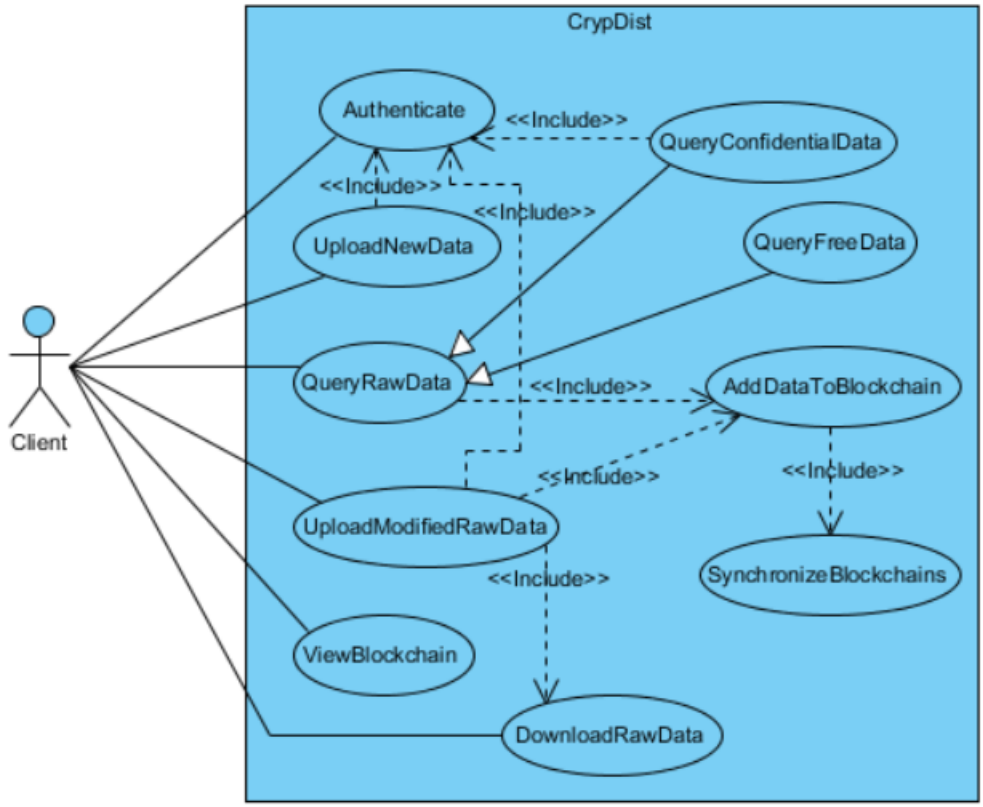
Client selects the authentication option in the main menu. Then System opens the authentication screen. Client enters his information and System checks the information. After confirmation, System returns to main menu and client chooses the option to query data. Then System opens the data query screen. Client enters an SQL command to the text area in the screen. System parses the command and executes it in the server. After that, System returns the data in a binary file.

Upgrade Data

After authentication, Client selects the option to update data. System opens the data update screen. Then, Client selects the option to download the data by using the results of query. System returns the data. After that, Client selects the option to upload the modified data.

3.5.2. Use Cases

Use Case Model:



Use Case 1:

Use case name: Authenticate

Participating actor: Client

Pre-condition: Client must be in the main menu.

Post-condition: System authenticates the client.

Entry condition: System opens the authentication screen.

Exit conditions:

- Client receives the confirmation.
- Client receives the rejection.

Main flow of events:

- Client chooses the option to authenticate.
- System displays the authentication screen.
- Client enters his information.
- System checks the database and give the message **accordingly**.

Use Case 2:

Use case name: QueryConfidentialData

Participating actor: Client

Pre-conditions:

- Client must have been authenticated to the system.
- Client must be in the main menu.

Post-condition: System returns the queried data.

Entry condition: System opens the querying screen.

Exit condition:

- System displays the data.
- System gives an SQL syntax error.

Main flow of events:

- Client chooses the option to query data.

- System displays the querying screen.
- Client enters the SQL command.
- System executes the command and returns the data.

Alternative flows:

- System displays SQL syntax error if the command includes it.

Use Case 3:

Use case name: QueryFreeData

Participating actor: Client

Pre-condition: Client must be in the main menu.

Post-condition: System returns the queried data.

Entry condition: System opens the querying screen.

Exit condition:

- System displays the data.
- System gives an SQL syntax error.

Main flow of events:

- Client chooses the option to query data.
- System displays the querying screen.
- Client enters the SQL command.
- System executes the command and returns the data.

Alternative flows:

- System displays SQL syntax error if the command includes it.

Use Case 4:

Use case name: UploadNewData

Participating actor: Client

Pre-condition: Client must be in the main menu.

Post-condition: System uploads the data.

Entry condition: System opens the upload screen.

Main flow of events:

- Client chooses the option to upload data.
- Client browses for the data.
- System uploads the data.

Use Case 5:

Use case name: DownloadRawData

Participating actor: Client

Pre-condition: Client must be in the main menu.

Post-condition: System returns the requested data.

Entry condition: System opens the update screen.

Main flow of events:

- Client chooses the option to update data.
- Client requests the data by using the results of the previous queries.
- System returns the data.

Use Case 6:

Use case name: UploadModifiedRawData

Participating actor: Client

Pre-condition: Client must have been downloaded some part of the data.

Post-condition: System uploads the modified data.

Entry condition: System opens the upload screen.

Exit condition: Client receives the confirmation message.

Main flow of events:

- Client browses for the data.
- System uploads the data and gives the confirmation message.

Use Case 7:

Use case name: AddDataToBlockchain

Participating actor: Client

Pre-condition: Client must have been uploaded or queried data.

Post-condition: System adds the data to the chain.

Main flow of events:

- System adds the data to the chain.

Use Case 8:

Use case name: ViewBlockchain

Participating actor: Client

Pre-condition: Client must be in the main menu.

Post-condition: System displays the block information.

Main flow of events:

- Client chooses the option to view the block information in the main screen.
- System displays the information.

Use Case 9:

Use case name: SynchronizeBlockchains

Participating actor: Client

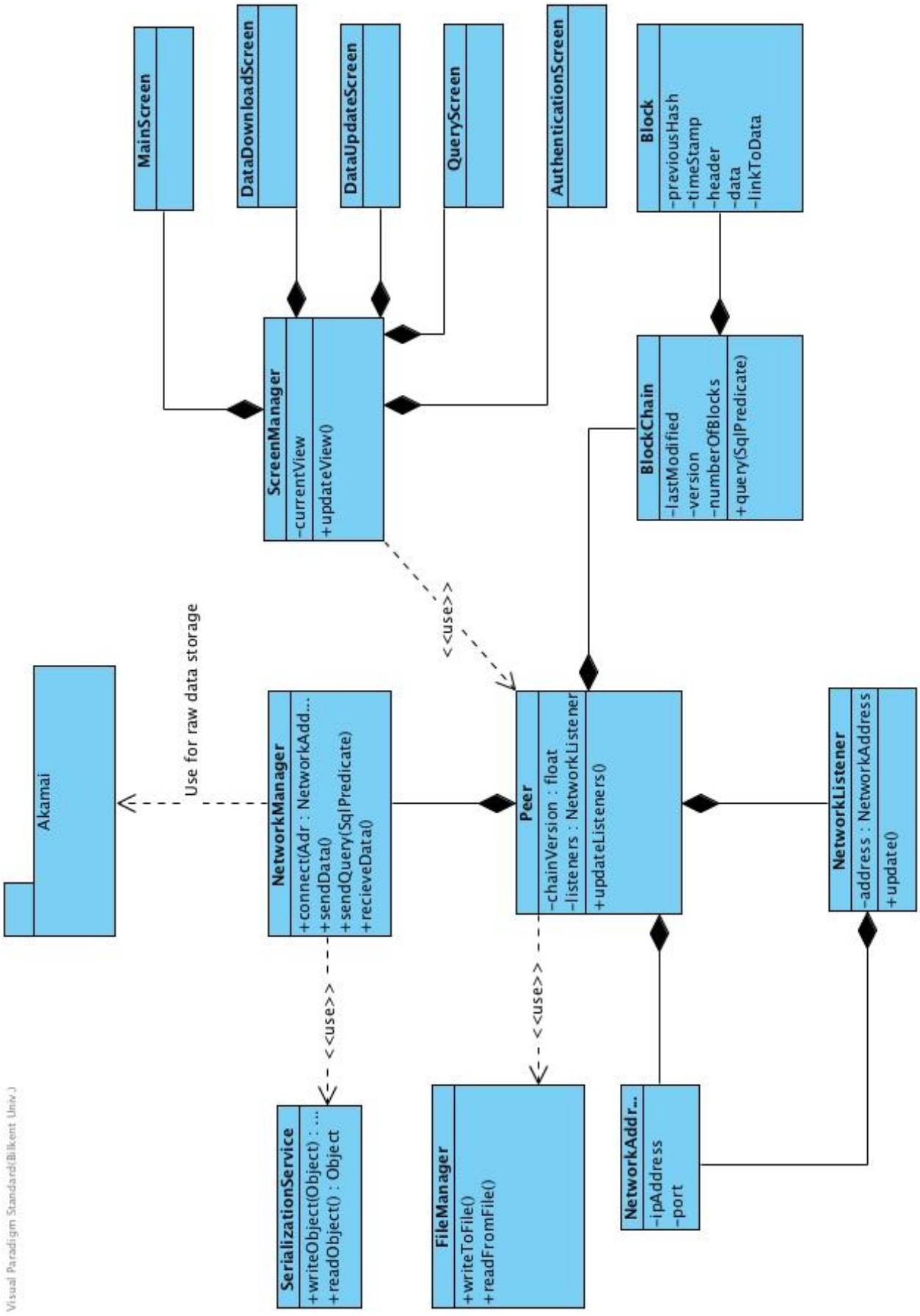
Pre-condition: System must have added data to the blockchain.

Post-condition: System synchronizes the blockchains.

Main flow of events:

- System synchronizes the blockchains.

3.5.3. Object and Class Model

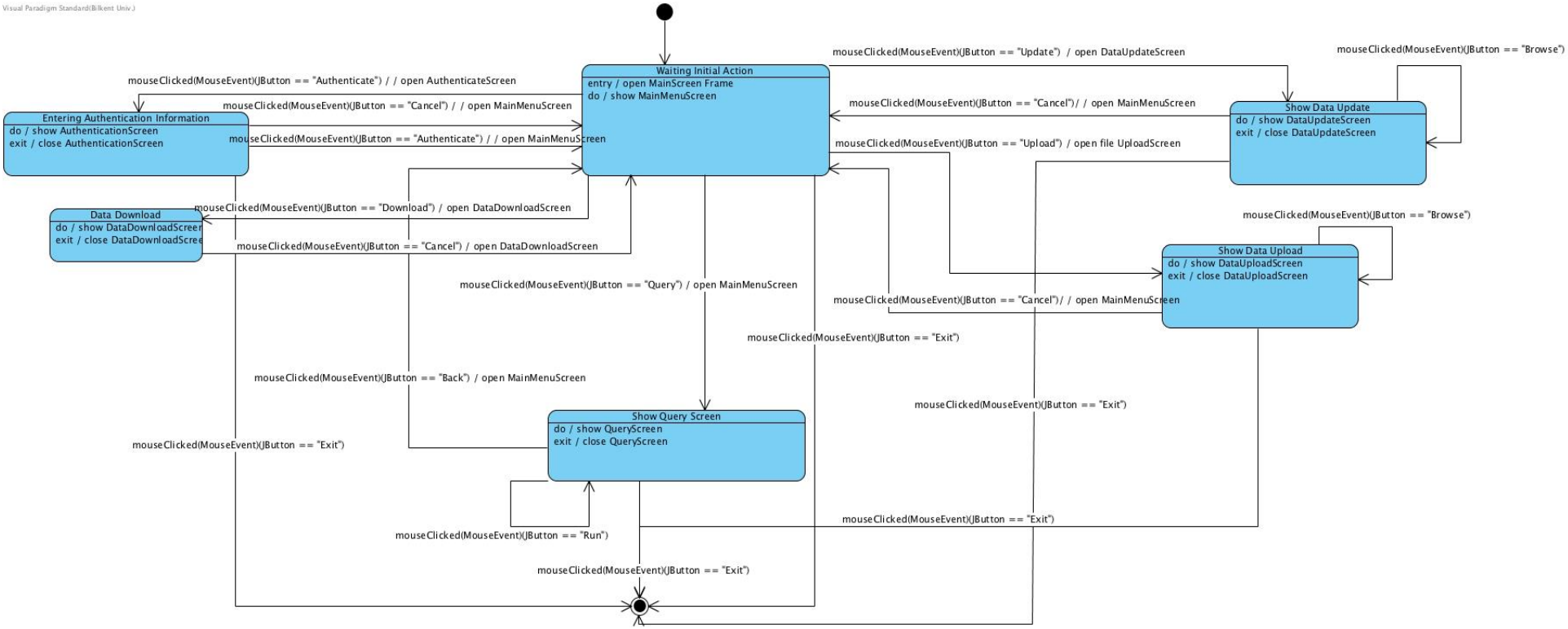


In class diagram, Peer mainly corresponds to a client in the network. This client has a copy of BlockChain, and notifies the other clients in case of an update in the BlockChain with the help of NetworkListener class. Also, for the File I/O, FileManager is used by the Peer class. NetworkManager establishes the connection between a client and server in order to provide raw data flow. The data to be transmitted from server to client or client to server is serialized with the help of SerializationService. In order to store the raw data, Akamai service is used.Lastly, showing the appropriate screen to user is handled by ScreenManager and screen classes.

3.5.4. Dynamic Models

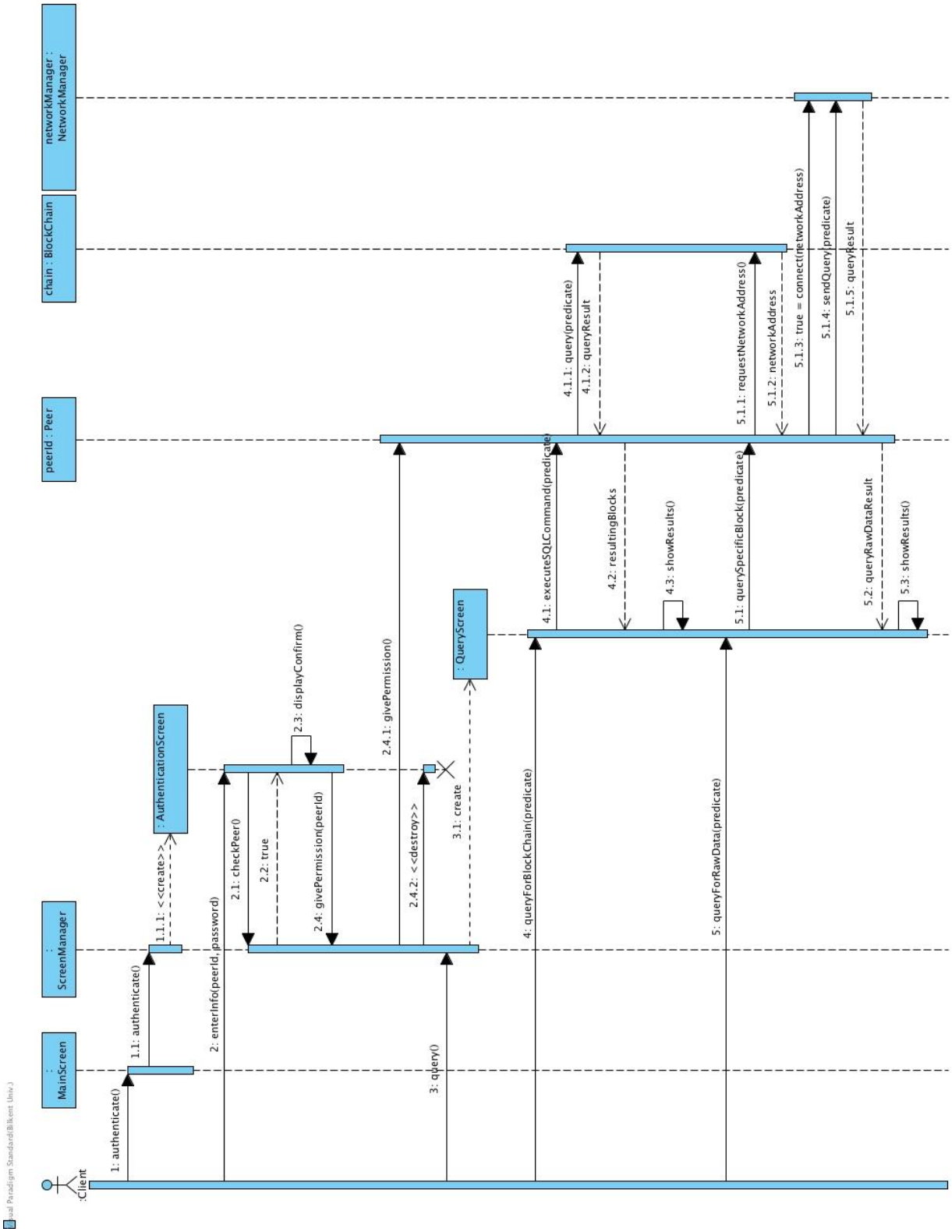
3.5.4.1. State diagram

In the diagram below, different states of ScreenManager object, which controls the navigation flow between screens, is described. Initially, when the program starts, the main screen is displayed. Main screen can be considered as a center to navigate all other screens. When "Authenticate" button is clicked, main screen is closed and authentication screen is displayed where user can enter its username and password. From authentication screen, by clicking "Authenticate" button, the ScreenManager navigates to main screen. From the main screen, via "Update" button, data update screen can be displayed and via "Upload" button, data upload screen can be displayed. By clicking on the "Query" button on the main screen, ScreenManager changes frame to query screen where user can write SQL query to search the data. Via "Download" button on the main screen, ScreenManager navigates to the data download screen. From all states except "Waiting Initial Action" state when the current screen is main screen, by clicking on the "Cancel" button, ScreenManager can navigate to main screen. Also, from all screens, ScreenManager allows to exit from the program.


Visual Paradigm Standard(Bilkent Univ.)
Waiting Initial Action
entry / open MainScreen Frame
do / show MainMenuScreen
Entering Authentication Information
do / show AuthenticationScreen
exit / close AuthenticationScreen
Data Download
do / show DataDownloadScreen
exit / close DataDownloadScree
Show Data Update
do / show DataUpdateScreen
exit / close DataUpdateScreen
Show Data Upload
do / show DataUploadScreen
exit / close DataUploadScreen
Show Query Screen
do / show QueryScreen
exit / close QueryScreen
mouseClicked(MouseEvent)(JButton == "Authenticate") / / open AuthenticateScreen
mouseClicked(MouseEvent)(JButton == "Cancel") / / open MainMenuScreen
mouseClicked(MouseEvent)(JButton == "Authenticate") / / open MainMenuScreen
mouseClicked(MouseEvent)(JButton == "Update") / open DataUpdateScreen
mouseClicked(MouseEvent)(JButton == "Browse")
mouseClicked(MouseEvent)(JButton == "Cancel")/ / open MainMenuScreen
mouseClicked(MouseEvent)(JButton == "Upload") / open file UploadScreen
mouseClicked(MouseEvent)(JButton == "Download") / open DataDownloadScreen
mouseClicked(MouseEvent)(JButton == "Cancel") / open DataDownloadScreen
mouseClicked(MouseEvent)(JButton == "Browse")
mouseClicked(MouseEvent)(JButton == "Query") / open MainMenuScreen
mouseClicked(MouseEvent)(JButton == "Cancel")/ / open MainMenuScreen
mouseClicked(MouseEvent)(JButton == "Exit")
mouseClicked(MouseEvent)(JButton == "Back") / open MainMenuScreen
mouseClicked(MouseEvent)(JButton == "Exit")
mouseClicked(MouseEvent)(JButton == "Exit")
mouseClicked(MouseEvent)(JButton == "Run")
mouseClicked(MouseEvent)(JButton == "Exit")
mouseClicked(MouseEvent)(JButton == "Exit")

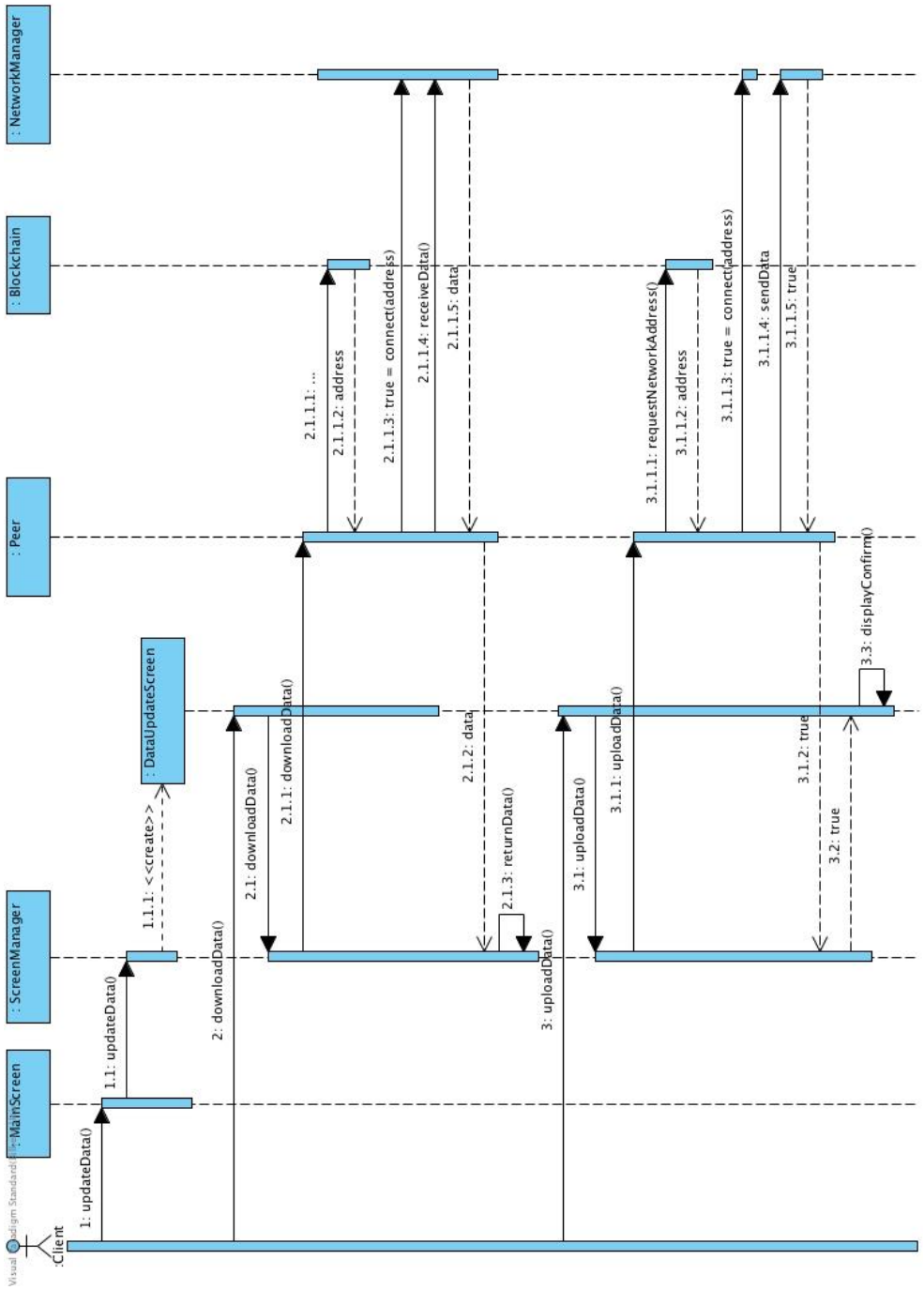
3.5.4.2. Sequence Diagrams

3.5.4.2.1. Query Confidential Data



In the diagram, the actor is the client of the system. MainScreen is the boundary class which client interacts with. It sends the request of the client to the ScreenManager class which is control class responsible from the screen transitions. It creates new boundary classes which client can interact with. Peer object is the record of the client in the system and the SQL query is processed via its blockchain and network manager.

3.5.4.2.2. Upgrade Data

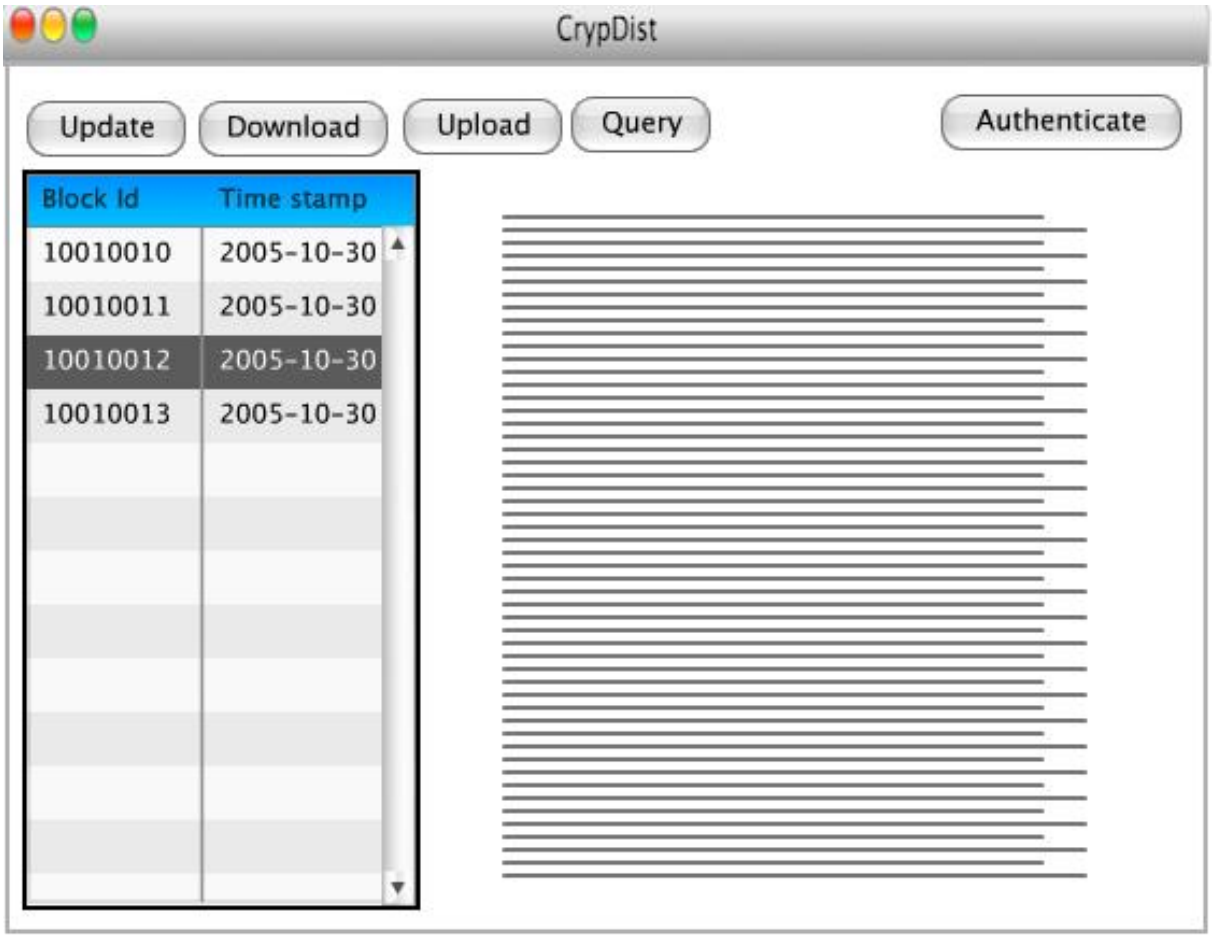


Authentication is also required in this scenario, but it is not shown again since it is the same as the previous one. In the diagram, the actor is the client, main boundary class is MainScreen, and main control class is ScreenManager which also creates other boundary objects. The logic of the connections is the same as the query diagram except in this diagram, the system sends and receives data via network instead of querying.

3.5.5. User interface

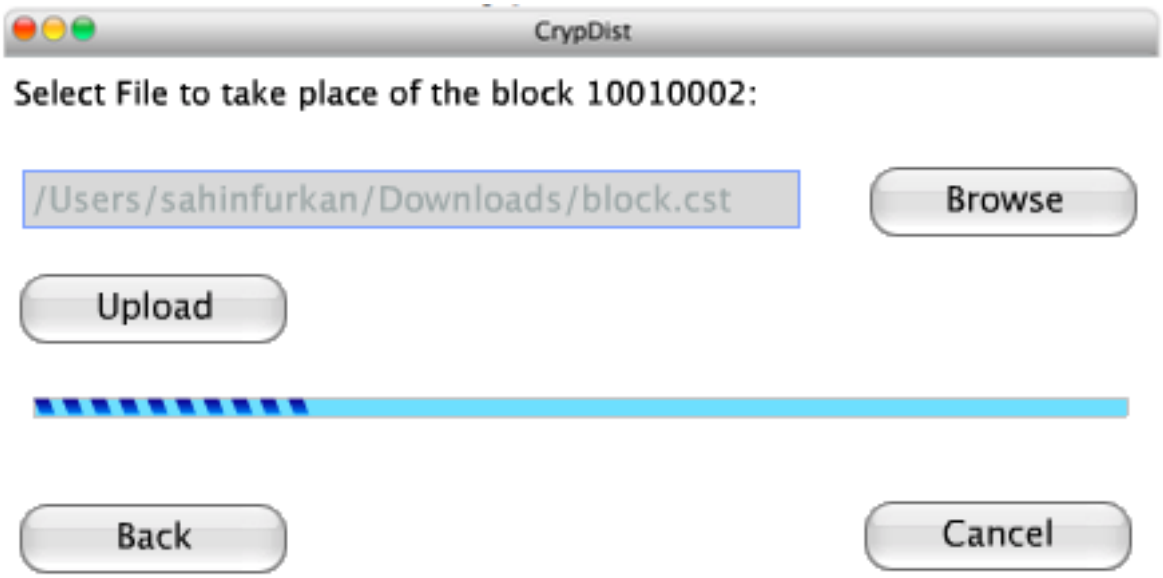
3.5.5.1. Main Screen

When user starts the application, Main screen shows up as the first screen. Main Menu is the screen providing access to all functionalities of CrypDist. He is also able to examine the blockchain and see the content stored in it by the provided panels. Mock-up visualization can be seen below.



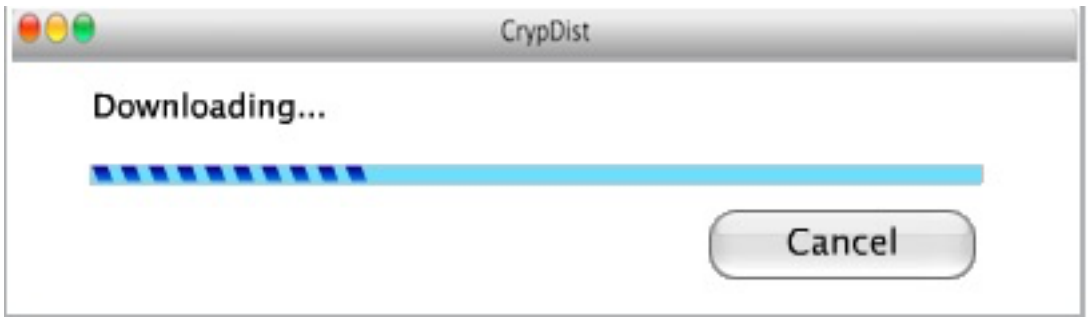
3.5.5.2. Update

Update screen allows user to upload a new data block instead of the chosen block in main screen. After he highlights the block to be replaced in main screen by clicking and choosing the update from the above menu, he gets redirected to the upload screen. In this screen, he locates the data to be uploaded and by choosing the upload button, he starts to upload the new data block but he always has the option to cancel the procedure in case of a mistaken choose.



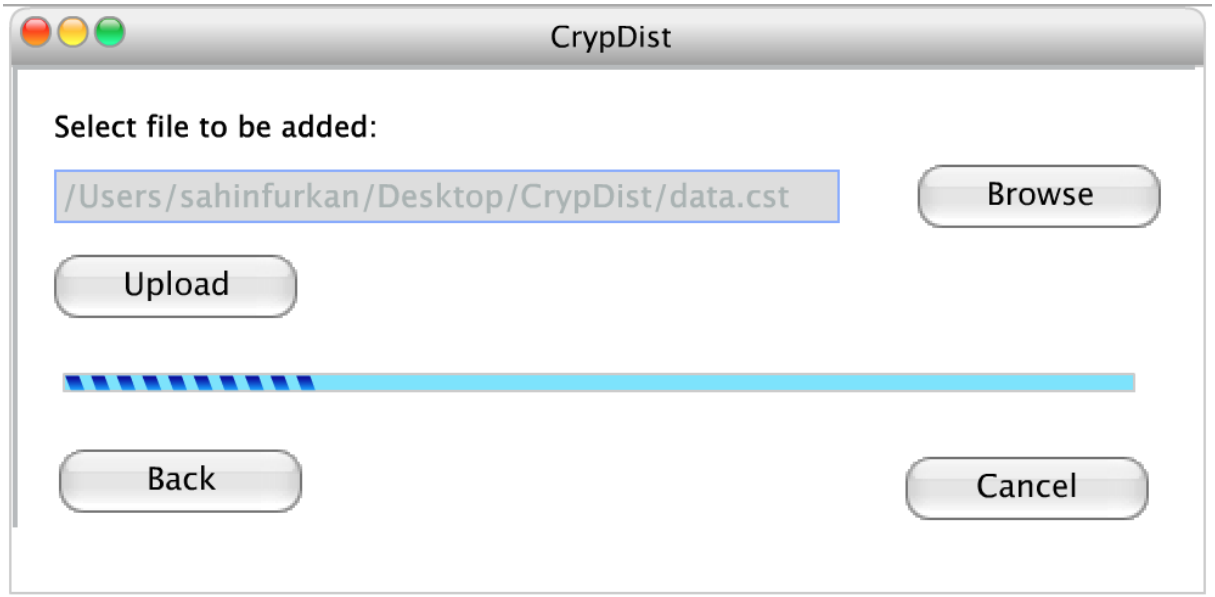
3.5.5.3. Download

Here we provide user the access to the raw data in servers. After he selects the block to be downloaded, he chooses the download option and the data is downloaded from the closest Akamai edge servers to his local.



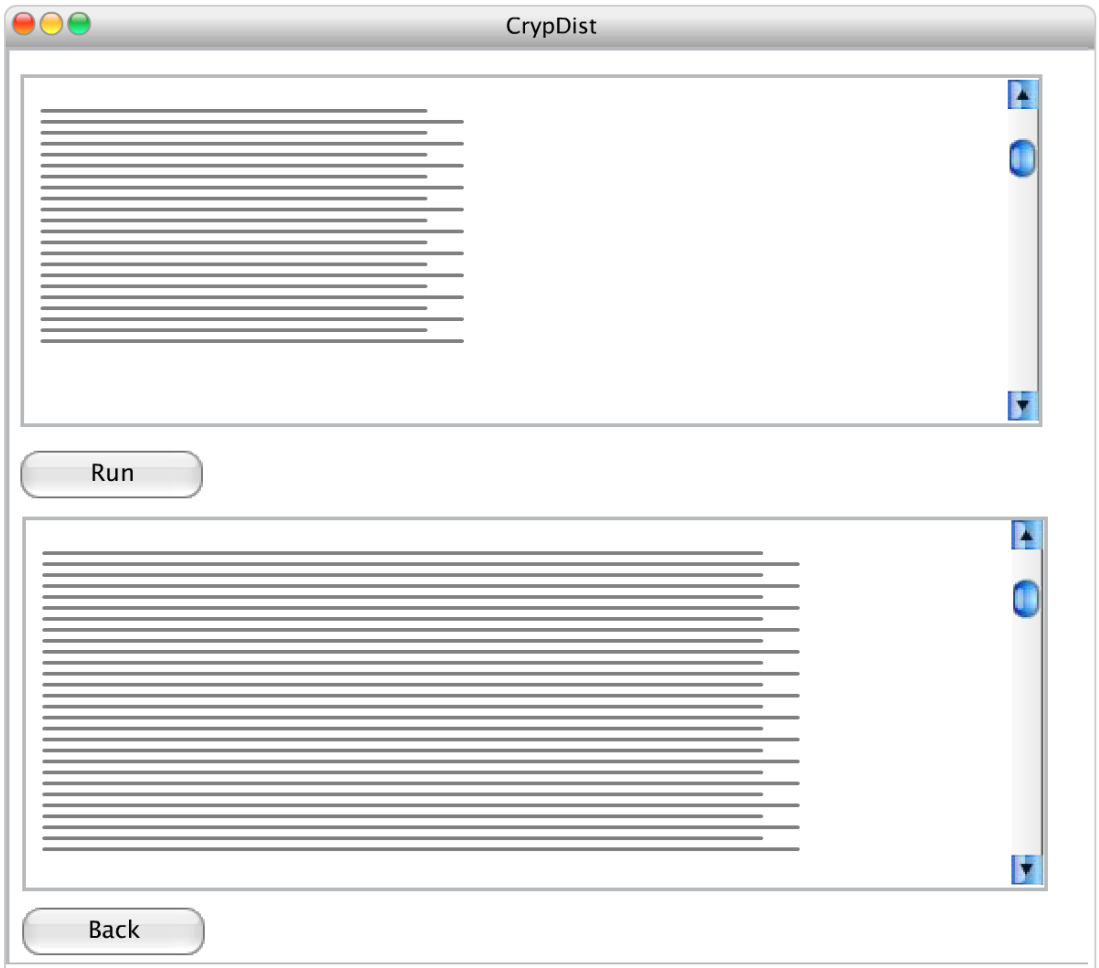
3.5.5.4. Upload

Upload screen is used to add a completely new data block to the system. Accordingly, blockchain is also updated and all the other pairs are notified by the change. User simply chooses the data he will upload and clicks upload, If he is authenticated to do so. According to the use cases of the system, some arguments might be also added to this screen but in a robust manner it is as shown below.



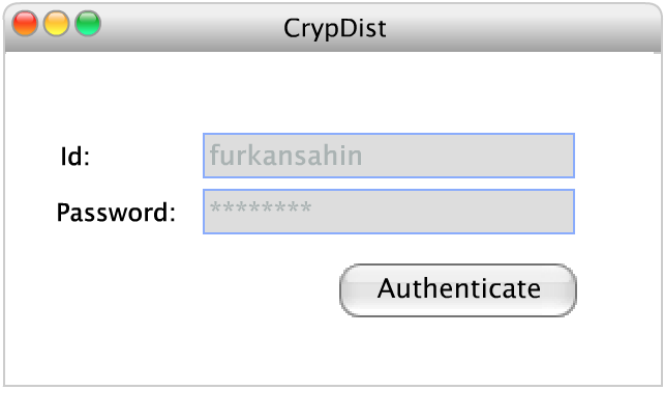
3.5.5.5. Query

CrypDist provides a query screen so that the user can run the queries on raw data. By the help of this screen he is able to understand the content of the server deeply and according to the results of the queries, he can demand to download a specific data. Query is typed in the provided screen and the results taken from the server are shown in the same screen.



3.5.5.6. Authenticate

CrypDist requires authentication for update, upload and phenotype data download purposes. Users will be able to use these functionalities if and only if they are one of our trusted users who will have an account to use these services. Authentication screen is a basic login panel which you can see below.



4. Glossary

Blockchain: Blockchain is a distributed database which aims to manage the blocks inside it.
Block: Blocks are continuously-growing list of records which are safe from revision and tempering.

Raw data: Raw data is the actual data that clients will use to conduct their research.

Akamai: Akamai is a company providing content delivery network and cloud services to its customers.

NetStorage: NetStorage is a cloud based storage which aims to enhance the quality and the performance of the delivery.

Genomics data: Genomics data contains information about the human genome such as DNA sequence, mutations and genetic disorders.

Phenotype data: Phenotype data contains information about the observable characteristics of human such as eye color, hair color and height.

Centralized database: Centralized database is a database where all the data are stored and maintained in one physical location.

dbGaP: dbGaP is a database containing genotypes and phenotypes data in a centralized database. It enables users to conduct advanced research by analyzing the data.