Bilkent University

Department of Computer Engineering

Senior Design Project

CrypDist

Project Low Level Design

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

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

Low Level Design Report
February 20, 2017

Table of Contents

1. Introduction

1.1. Object Design Trade-offs

1.1.1. Blockchain as a distribution system

There are several type of distribution systems which are mostly running in a centralised manner but in CrypDist we chose to move with Blockchain. Blockchain is mainly a database system which is used for secure money transactions throughout the years under the name of Bitcoin currency. It is a slow but free technique. Since it does not depend on any third party people during the transactions of data, it can be accepted as one of the most secure data transaction methods. Because of its direct structure it is not subject to any governmental control too. After the thorough consideration of above factors and thinking that our aim is to provide free data access, we decided to implement Blockchain even if it is comparably slower and more complex to implement than traditional centralised distribution systems.

1.1.2. SHA256 Hashing Function

SHA256 hashing algorithm is from the SHA2 crypto algorithm family which is mainly designed by the National Security Agency (NSA). It is widely used in processes requiring high security standards because of its high collision resistant nature. But these advantages come with some computation related disadvantages. For example, SHA256 is a comparable slow algorithm than the algorithms in SHA1 family and there is lack of support for SHA-2 on systems running Windows XP SP2 or older.[1] Additionally, when we compare the feasibility complexities of different algorithms including SHA256, we can conclude that it is relatively more complex than many other algorithms. But in any case, SHA256 is one of the most secure algorithms and the data that we give service of is very sensitive. Thus, when we consider that the security is one of the top priority in our application, SHA256 is the best fit.

1.2. Engineering Standards

For modelling the software architecture via diagrams, UML models are used. Class diagram is used for object design purposes.

2. Packages

For implementing the database for keeping block and privileged peer information, PostgreSQL package will be used. Privileged peer means the ones who have access to confidential information like phenotype data.

[1] https://en.wikipedia.org/wiki/SHA-2

3. Data Structures and Algorithms

3.1. Blockchain Structure and Block Mining

Blockchain is a data structure used for holding transactions. It consists of blocks linked together. Each block contains four transactions. The main contents of a block are the following:

- Block id
- Block hash: Unique hash key (256 bits) of the block that distinguishes it as a primary key
- Previous block hash: For linking to the previous block, each block contains its hash key as a pointer.
- Timestamp: Time of creation of the block
- Merkle tree: It is the tree that holds the signatures of transactions. A signature is the unique key which identifies a transaction.

Blocks are created by the people who are so called miners. In CrypDist, all peers are miners, which means all of them are responsible for creation of the blocks. Creation of a block is a slow process such that it requires finding a valid hash key. The hash key is produced by using the block metadata (id, timestamp, previous hash, merkle tree root), by using the SHA-256 cryptographic hash function, such that when its data is changed, the change can be detected from its hash key. However, it should be ensured that after changing the data, the hash key will not be the same. For this purpose, there is a constraint on the hash key which is its first 8 bits are zero among the total 256 bits. For satisfying this constraint, an integer value called nonce is added to the block metadata to produce its hash key. For finding an appropriate nonce, a miner should try all values up to some maximum number in a brute-force approach. The miner who finds the nonce value first has the right to add the block to the chain. The block is also broadcasted to other peers.

There is also a difficulty value of a block which indicates the difficulty of its mining process. The difficulty of the blockchain is computed by summing up all the difficulties of the blocks on its longest path. Longest path means there are some forks in the chain. So the chain is more likely a tree instead of a linked list. A fork happens when two peers attempt to add a block at the same time. The consensus protocol ensures that the blockchain is the same for all peers by using the longest path.

The blockchain is said to be immutable, however the real explanation is changing it is very difficult. When data of a block is changed, its unique hash key also changes depending on the merkle tree root. So the later block’s hash key also changes since it depends on the previous block’s hash key. By domino effect, the last block’s hash key changes. So, the blockchain of a peer is compared with the blockchains of the majority, and when a change is detected, system requires him to update his blockchain by pulling from the other peers. There is still a possibility of an attacker to create an alternative blockchain, but according to the recent research, the probability of it decreases exponentially as the length of chain grows.

3.2. Merkle Tree Structure

Merkle tree is a data structure where its leaves hold the signatures of transactions. The signatures are produced by double hashing their string representations by SHA-256 hash function. The internal nodes are generated by combining the leaves' signatures and by the same process, the root is generated in a bottom-up manner. The merkle tree root is used to detect a change in the transaction data such that it also changes the block hash.

3.3. Security

Security of the network and the blocks is managed by the proof of work and SHA256 algorithms. Security of the network is handled by proof of work which is produced by hashing the block header to validate the blocks. Since the security of the network is handled by proof of work, not by access control, there is no need for encryption to secure the network traffic. The algorithm which finds the proof of work hashes the block header and a random number that is calculated with SHA256 cryptographic algorithm, several times until it finds a predetermined matching pattern. When a new block is created, it is not directly added to the block chain and called candidate block because it does contain proof of work yet. The block can be added to the blockchain if one of the miners comes up with a solution to the proof-of-algorithm. To perform the proof of work calculation for each block, the hash of the block header must be less than the target difficulty. The difficulty target is a 4-byte hexadecimal value stored in the block header. First two hexadecimal digits represent the exponent and the next six hex digits represent the coefficient. The difficulty target is calculated with the following formula:

target = coefficient * 2^(8 * (exponent – 3))

Blocks arriving at different nodes and different times can cause forks in the blockchain due to its decentralized structure. Proof of work value is also used to solve the forks in blockchain. Each node always chooses and expands the chain of blocks which has the largest cumulative difficulty chain. The total proof of work is calculated by summing the difficulties stored in each block. Therefore, forks appear between the different versions of blockchain temporarily. As new blocks are added, the nodes in the network chose the blockchain that has the greatest proof of work to add the new blocks and resolve the forks.

3.4. Secure Hash Algorithm 256 (SHA)

SHA is a cryptographic hash function which uses 32-byte words. Each block in the blockchain has a unique hash produced by using the SHA256 cryptographic hash algorithm on the header of the block. The block hash is a 32-byte digital fingerprint for each block. The SHA256 algorithm always produces 32-byte result regardless of the length of the data. Once the SHA256 function is used, it is difficult to obtain the data from the hash.

4. Pattern Applications

4.1. Façade Design Pattern

For encapsulating individual subsystems, Façade pattern is applicable. By providing the subsystem interface by only one class, other classes can be abstracted. For Blockchain subsystem, BlockchainManager class will provide the interface by managing Blockchain and Peer objects.

4.2. Adapter Design Pattern

For data distribution, CrypDist will use the services provided by Akamai technologies. Therefore, it should adapt the legacy system and use it without modification. For that purpose, an adapter pattern is applicable. The following is its model.

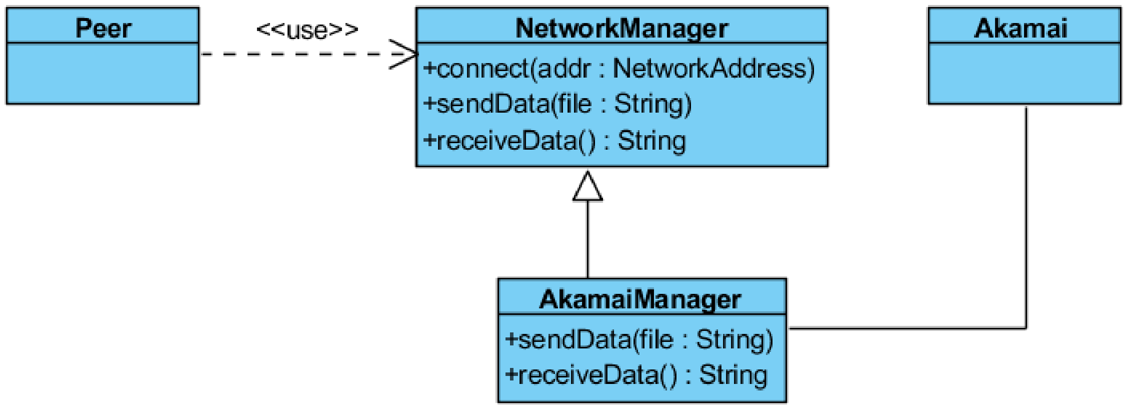


Figure 4.1 – Adapter Design Pattern

NetworkManager class is responsinle for managing the network between the peer and Akamai. By inheritance its methods are overridden by AkamaiManager which adapts those methods for the API of Akamai.

4.3. Command Design Pattern

Since this is a transaction system, encapsulating transactions with command pattern is useful. By that way, the transactions can be stored and executed without knowing their types and the system becomes extensible. In the future, new transaction types can be added by that way. The following model describes the pattern.

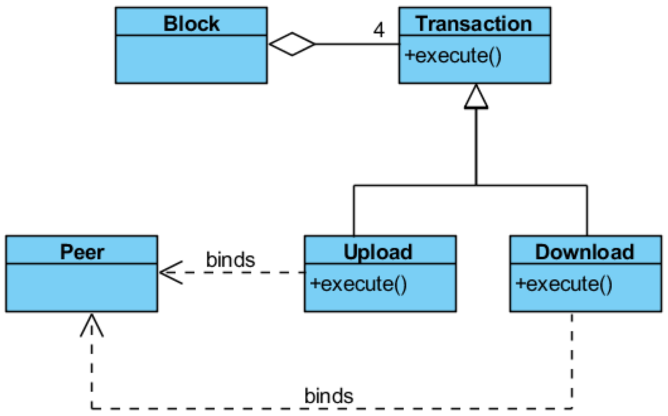


Figure 4.2 – Command Design Pattern

Block contains exactly four transactions. There are two types of transactions: Upload and Download. Both of them overrides the execute method of Transaction class. They belong to a single Peer object.

4.4. Singleton Design Pattern

Blockchain object will be replicated among the peers. So, each local program will contain exactly one Blockchain object. Containing multiple Blockchain objects is not valid and may cause unexpected situations. So by a singleton pattern, it will be ensured that there is exactly one instance of the Blockchain class by separating its instantiation from the constructor of the BlockchainManager class.

5. Class Interfaces

5.1. Subsystem Decomposition

Following component diagram shows the subsystem decomposition.

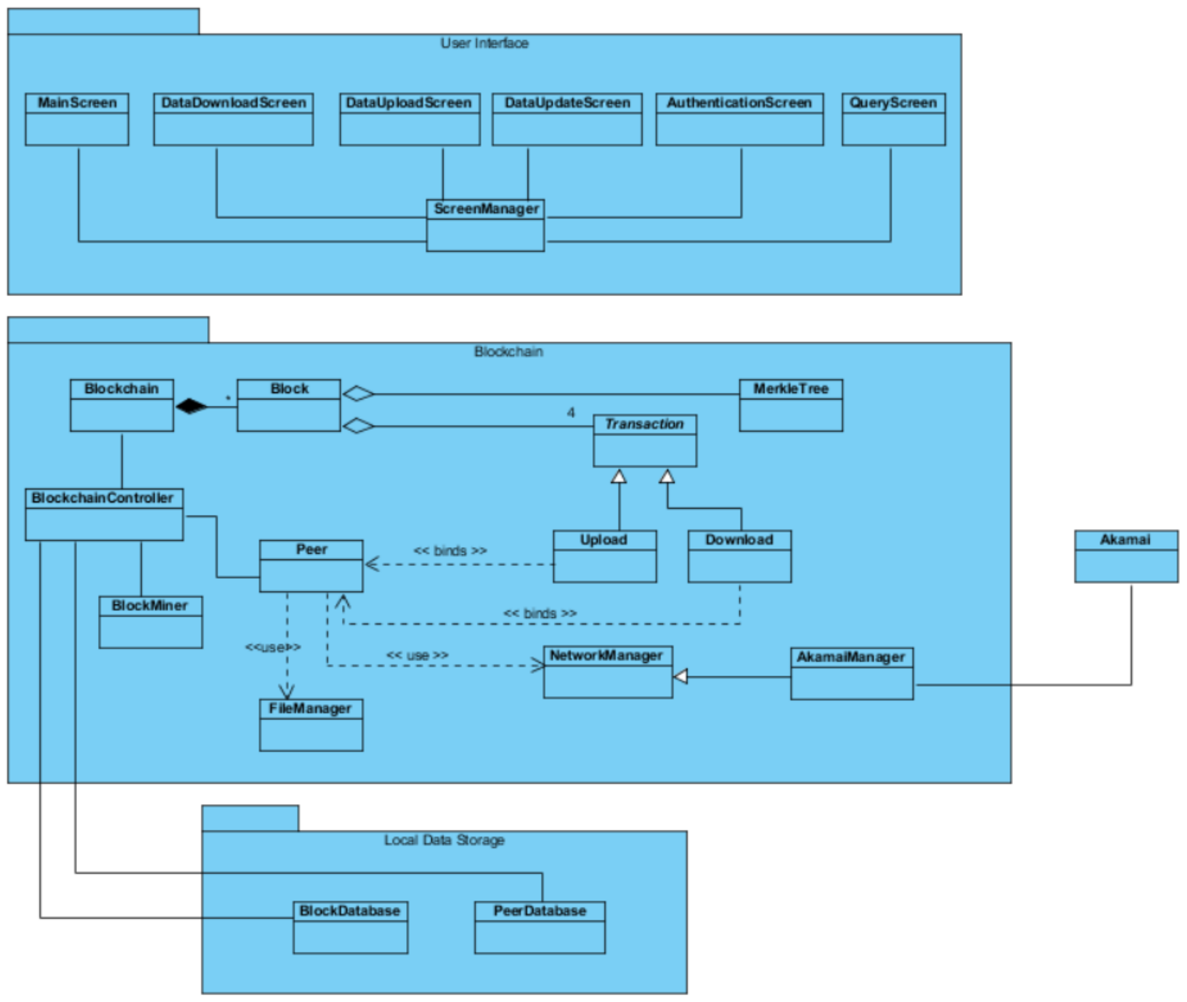


Figure 5.1.1 – Subsystem Decomposition

5.2. User Interface Subsystem

Following is the class diagram of GUI subsystem.

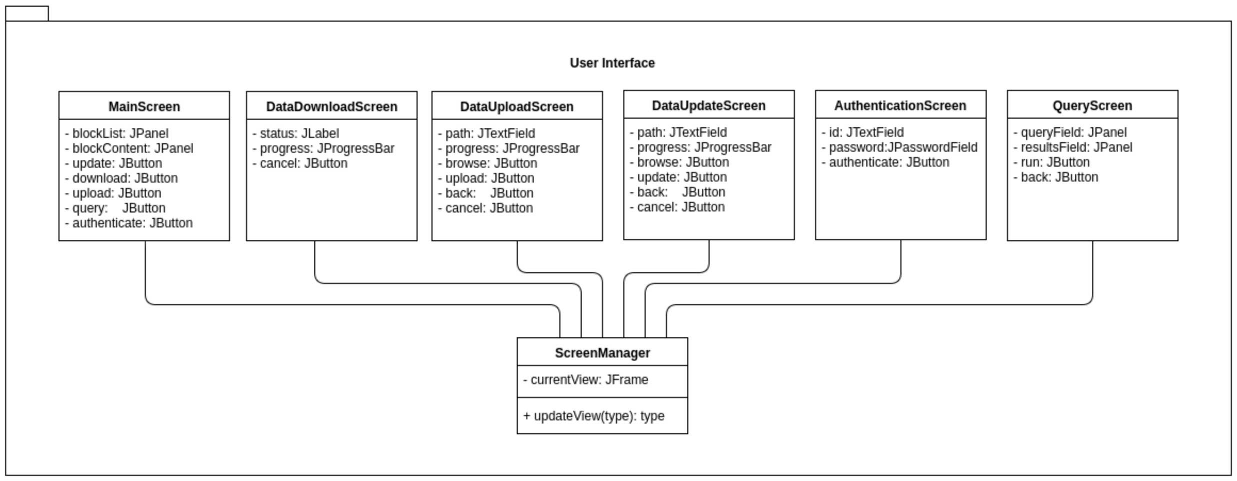


Figure 5.2.1 – User Interface Subsystem

Graphical User Interface Subsystem is the “Presentation” layer of 3-Tier Architecture and provides interaction with the system to the users. The information about BlockChain is visualized in interface and users can perform actions through static stable interface components and a management component. The subsystem is composed of 5 view classes and a controller. Each view class, contains the required instances for its purpose.

Main screen: It encounters the user when the program is initialized. Via the main screen, the user can access all functionalities of the system. Also, contents of the blockchain is visualized on a panel on the mainscreen.

DataDownloadScreen: Via download button in the main screen, the user can navigate to download screen which consists of a progress bar, showing how far along he/she is in the process to access data in servers.

DataUploadScreen: Via upload button in the main screen,the user can navigate to upload screen where the user can add a completely new data block to the system by selecting a file via browse button.

DataUpdateScreen: Via update button in the main screen, the user can navigate to update screen where the user upload a new data block instead of the chosen block in main screen by selecting a file via browse button.

AuthenticationScreen: Via authenticate button in the main screen, the user can navigate to authentication screen which provides a basic login panel the user can enter its id and password.

QueryScreen: Via query button in the main screen, the user can navigate to query screen which provides a panel that user can view the results of its query taken from the server.

ScreenManager: It is the controller for user interface components. It holds the object that creates current user interface. With the proper mouse inputs, ScreenManager changes the objects that create the user interfaces.

5.3. Blockchain Subsystem

Following is the class diagram of blockchain subsystem.

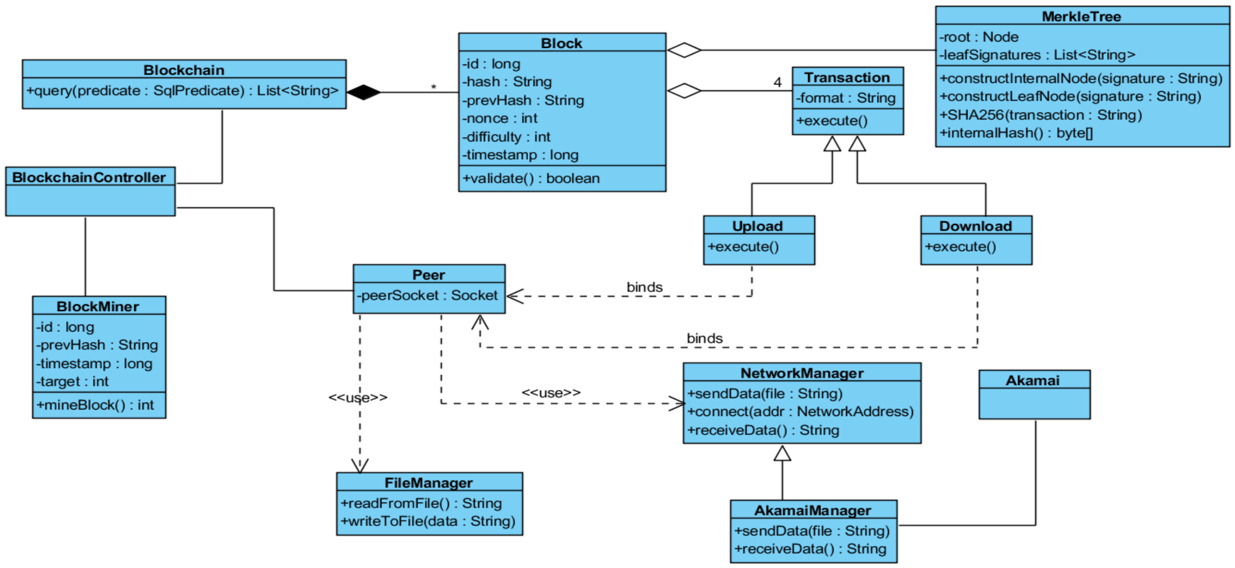


Figure 5.3.1 – Blockchain Subsystem

Block Class

Attributes:

- **private long id:** Id of the block
- **private String hash:** Unique hash key of the block
- **private String prevHash:** Hash key of the previous block
- **private int nonce:** The value which is used in the block mining process to generate the distinguishable hash key
- **private long timestamp:** Creation time of the block
- **private int difficulty:** Difficulty of finding the hash key
- **private MerkleTree data:** The data structure which holds the transaction signatures
- **private ArrayList<Transaction> transactions:** Transactions in the block

Operations:

- **public boolean validate():** Checks if the block structure is valid for adding to the blockchain.

Blockchain Class

Attributes:

- **private ArrayList<ArrayList<String>> chains:** Holds the chains in the structure. Each chain contains block hashes. Since there are forks, there are multiple chains.
- **private HashMap<String, Block> blockMap:** Maps hash keys to blocks.

Operations:
- **public boolean addBlock(Block block):** Adds a new block to the chain if it is valid and returns the receipt.
- **public List<String> query(SqlPredicate predicate):** Executes a query for the chain.

BlockchainController Class

Attributes:
- **private Blockchain blockchain**
- **private Peer peer**

Operations:
- **public void addBlockToChain(Block block): Adds a block to the blockchain**

BlockMiner Class

Attributes:
- **int target:** Target difficulty of finding a hash key
- **int maxNonce:** Maximum value of the nonce value
- The others are the same as block except hash key

Operations:
- **public int mineBlock():** Finds a hash key for the block by trying different nonce values and returns the minimum appropriate one

MerkleTree Class

Attributes:
- **Node root:** Root node of the tree
- **List<String> leafSigs:** Signatures of the leaves

Constructor:
- **public MerkleTree(ArrayList<Transaction> transactions)**

Operations:
- **public void constructInternalNode(String signature)**
- **public void constructLeafNode(String signature)**
- **public String SHA256(String transaction):** Computes the signature of the transaction by the SHA256 algorithm.
- **public byte[] internalHash(String leftSignature, String rightSignature):** Computes the signature of the internal node from the child nodes.

Peer Class

Attributes:
- **private Socket peerSocket:** End-point of the peer which contains its IP address and port.

- **private FileManager fileManager:** Reads and writes the data into local files.
- **private NetworkManager networkManager:** Establishes a connection between the peer and the Akamai servers.

FileManager Class

Operations:
- **public String readFromFile(String fileName):** Reads the data from the file.
- **public void writeToFile(String data):** Writes the data to the file.

NetworkManager Class

Operations:
- **public void sendData(String fileName, String link):** Send data in the file to the appropriate link.
- **public String receiveData(String link):** Gets the data from the appropriate link.
- **public void connect(NetworkAddress addr):** Connects to the network address

AkamaiManager Class

Base Class: NetworkManager
Operations:
- **public void sendData(String fileName):** Send data to the file adapted to Akamai API.
- **public String receiveData(String link):** Gets the data from the appropriate link adapted to Akamai API.

Transaction Class (Abstract Class)

Attributes:
- **private Peer peer:** Peer associated with the transaction
- **private String format:** String representation of transaction for producing its signature

Operations:
- **public void execute():** Executes the transaction.

Upload Class

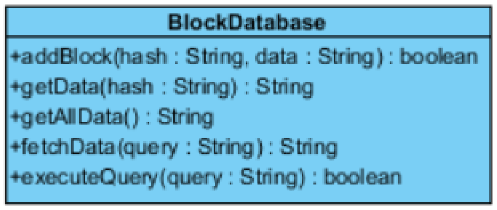
Operations:
- **public void execute():** Executes the transaction.

Download Class

Operations:
- **public void execute():** Executes the transaction.

5.4. Local Data Storage Subsystem

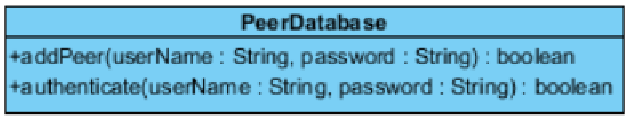
BlockDatabase Class


BlockDatabase
+addBlock(hash : String, data : String) : boolean
+getData(hash : String) : String
+getAllData() : String
+fetchData(query : String) : String
+executeQuery(query : String) : boolean

Operations:

- **public boolean addBlock(String hash, String data):** Adds a new block.
- **public String getData(String hash):** Gets the data of the block with the given hash.
- **public String getAllData():** Gets all data in the database.
- **public String fetchData(String query):** Executes query and returns resulting data.
- **public Boolean executeQuery(String query):** Executes update.

PeerDatabase Class


PeerDatabase
+addPeer(userName : String, password : String) : boolean
+authenticate(userName : String, password : String) : boolean

Operations:

- **public boolean addPeer(String userName, String password):** Adds a new peer.
- **public boolean authenticate(String userName, String password):** Checks if the peer data exists.

6. Glossary

Akamai Technologies: an American content delivery network (CDN) and cloud services.

Genotype: the particular type and arrangement of genes that each organism has.

Phenotype: the physical characteristics of something living, especially those characteristics that can be seen.

PostgreSQL: an object-relational database management system.

Transaction: a movement of data between a source and destination in the system.

Hash function: a function which maps arbitrary-sized data, known as a value, to a fixed-sized data, known as key.

Cryptographic hash function: a hash function which is a mathematical algorithm, which has a range of fixed-sized bit strings, and which is one-way (not invertible).

7. References

[1] “SHA-2”, https://en.wikipedia.org/wiki/SHA-2. [Accessed: 19.02.2017]

[2] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System, https://bitcoin.org/bitcoin.pdf. [Accessed: 19.02.2017]