Project report:
Distributed Database

Team:
Anton Bozhedarov - Architecture design
Vsevolod Glazov - Cryptography
Ivan Matvienko - Rust developer
Nikolay Shvetsov - Cryptography
Andrey Vlasov - Rust developer
Anton Razzhigaev - Architecture design

1. Introduction

A distributed database is a database that consists of two or more files located in different sites either on the same network or on entirely different networks. Portions of the database are stored in multiple physical locations and processing is distributed among multiple database nodes.

Decentralized database helps to increase memory capacity of the system and makes it tolerant to failure of the some parts. But in this case one may face a problem of integrity of data and protection it from undesirable changes.

Blockchain technology allows to deal with mentioned problems. So every user of a system is able to check correctness of the data and make changes only with consent of other members. Moreover using secret sharing technique for encryption of the distributed pieces helps to organize system of data storage with no trust.

2. File encryption and distribution

To initialize the system we take the entire file that needs to be stored in the database and split it into the parts. Then we encrypt each of these parts and share among system's nodes with duplicates to increase the robustness in case some of the nodes will be disabled.

In this project for simplifying and optimizing the access time we work with a csv-table file, and share not only the encrypted part of this file but also a list of the rows contained in the file to reduces the processing time in case the part doesn't contain the rows to be modified. Also, storing the rows of the entire file helps to easily restore it.

To split the file we implemented the function that takes as an input parameter number of rows to be included in each part of the file and returns the set of the splitted parts.

The principal scheme of the initialization stage is presented on the figure 1.

After splitting the file we encrypt each of it's part independently with a randomly generated key using Advanced Encryption Standard (AES) with Cipher Block Chaining (CBC

mode). This way, each 16-bit ciphertext block depends on all plaintext blocks processed up to that point that increase the security and uniformity of encrypting.

The hash of each part we add to the initial block of the Blockchain. It allows to use the mechanism of verifying the congruence of the same part of entire file on different nodes of the distributed system at any time the file is being modified.

Same as distributing the parts of the file we split the randomly generated key using the Shamir's secret scheme among the system's verifiers to make it impossible for a user to edit the file alone. Shamir's scheme allows to adjust the number of participants that is necessary to edit the file. In our project we established that number to be a half of a number of all verifiers.

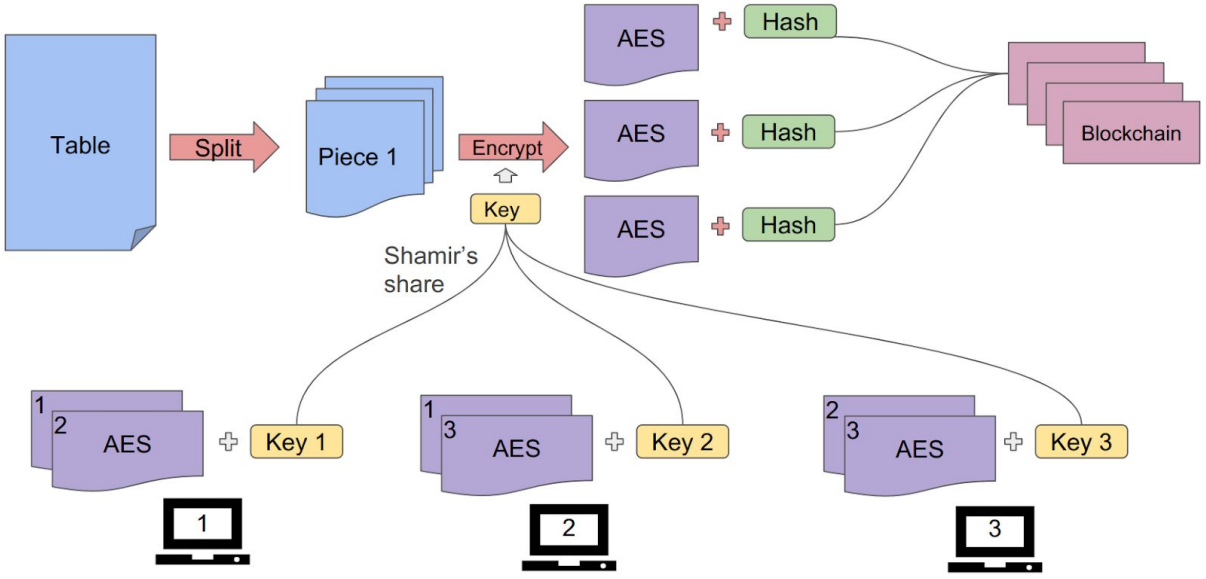


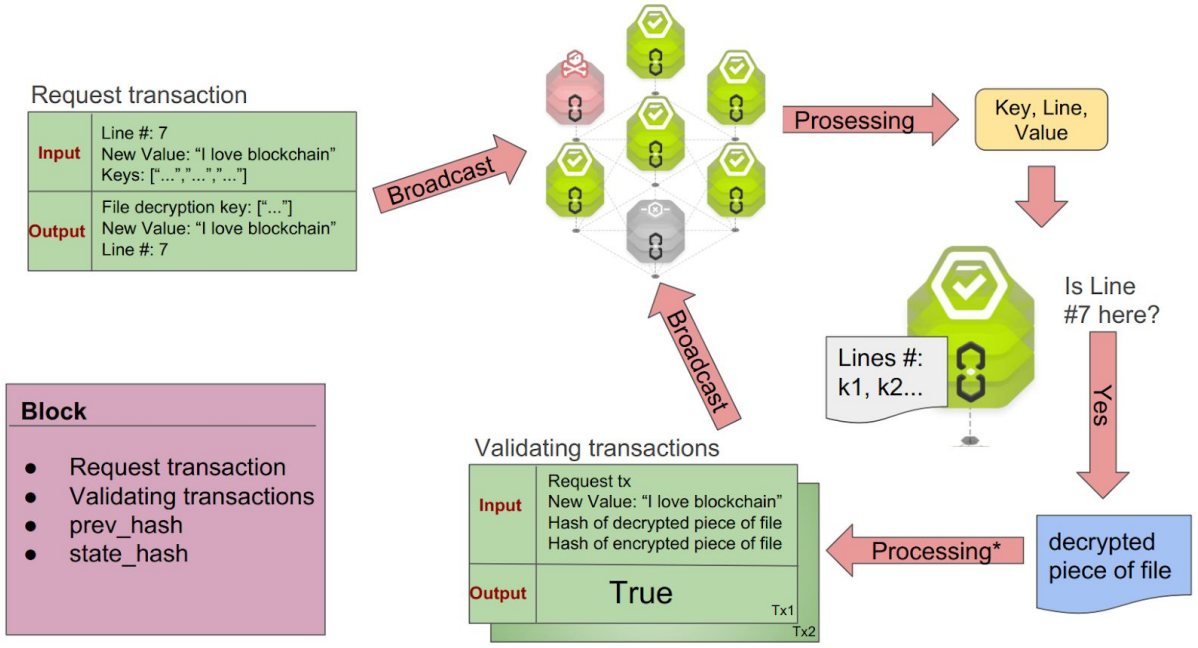
Figure 1: Initialization stage of the system

3. Blockchain network

We designed a blockchain based architecture to ensure users that logs of changings in the file are unchangeable and no editing could be done without agreement of a threshold number of maintainers. We decided to use Exonum framework for that purpose.

Now let's make a closer look on one particular editing of the file. Suppose Bob wants to edit file and write in the 7th line that "he loves EXONUM" therefore he creates a transaction with proposal of editing file. He asks other participants of the network to sign this transaction if they accept this changing in the file. Once sufficient number of keys were collected the transaction uploads to the network. Then if the transaction is correct and has sufficient number of keys it returns the initial common key of the file recovered via Shamir scheme.

Not all nodes of the network has the requested part of the file with the 7th line, then if a node has a requested part the processing of file begins and the node creates a validating transaction (about it we will tell you in the next section). One block in the blockchain consists of a request transaction, validating transactions from nodes that have the requested part of the file, a hash of previous block and a state hash. Only if block is correct the requested changing is accepted by the network. The principal scheme of this all mentioned above you can see in the picture 2.

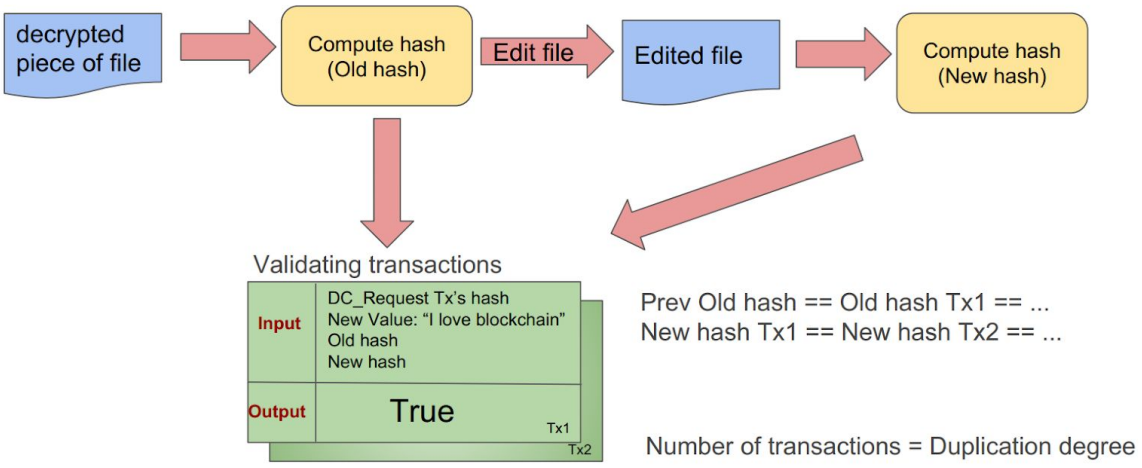


Picture 2: Blockchain network

4. Validation of the file editing

As it was mentioned above all available nodes that contain the requested part of the file create a validating transaction after processing the editing request. Every piece of file is duplicated and thus stores on several nodes. On every computer which has a requested piece of file the recovered key from transaction uses for decrypting that piece of file. Then a hash of this part is calculated for including into validating transaction for comparison with hashes from other validating transactions from nodes and an old hash from previous block.

Then after a requested editing is done a New hash is calculated and also included into transaction for further comparison during validation of block. The block is correct only if all (or majority) hashes from validating transactions are equal which means that all nodes recovered the same file and made the same editing. Also the hash of decrypted file compares with the one from previous block. And only if all this comparisons are OK the block adds to blockchain and an editing request accepted by the system. On the scheme below you can see the process of creating a validating transaction.



Picture 3: Creating a validating transaction

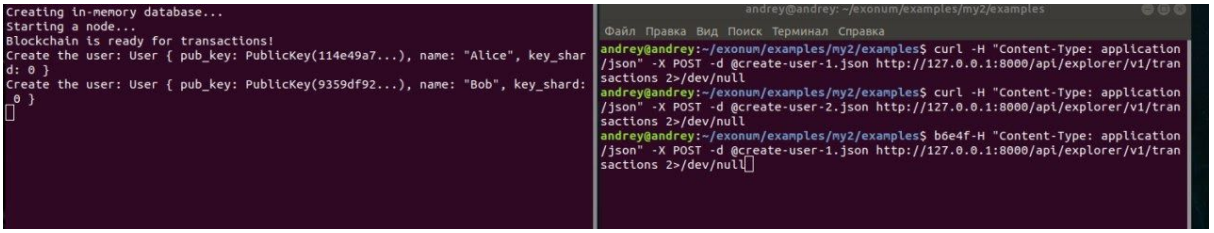
5. Exonum

We decided to work with Exonum - framework for building private blockchains. In order to increase the process of writing code on Rust, we decided to use built-in Exonum example "Cryptocurrency" as a base for our project. The essence of using exonum is that you only need to write business logic of Smart Contract or Services as they are called in Exonum. Other things are already implemented (fault tolerance protocol, merkle trees, e.t.c). In Service we implemented the data structures, that are stored in blockchain, the transactions and its logic, and APIs.

We have decided to implement 3 types of Transactions:

- TxCreateUser
- TxChangeDocument - for changing the document
- TxValidateRequest - for Validating all the requests

To store something in the Blockchain, there is the special data structure "User" in which we store user name, his Public Key, and his part of common secret we call "key shard". So, to initialize the user, we commit the "TxCreateUser" transaction. In our concept, every user is a validator of some particular node. The example of such transaction you can see below.



The Exonum itself doesn't have access to outer world, so to get access to the file, we want to modify, we need to write corresponding file API, and to make some queries to the Exonum via HTTP, there is and REST API pre implemented, by we have slightly modified it for our case. We weren't able to implement the file API due to the lack of time. But the modification script is written in python.

As it was mentioned before, we decided to use the Shamir's secret sharing scheme. Initially, we implemented it on python, but a little bit later, we decided to use the one, which is written on Rust, because it is much easier to embed in exonum.

There is an instruction file "readme.pdf", how to run the project. We have tested it only on 1 machine, so if you would like to test it on 2 computers, for example, you should make slight modification in the launching process.

6. Results

As a result of our project we created a design of distributed secure database architecture, including file distribution and encryption with AES and Shamir secret share scheme for key distribution among maintainers. Also we deployed an Exonum private blockchain network for secure data editing according to the opinion of the threshold number of users through mechanism of transactions and services.

The blockchain guaranties that nothing in the history of editing file could be changed and that any editing of the file will be validated only if predefined number of users agree with it. The source code for the file encryption, distribution and Exonum network could be found in the corresponding assignment.