Architecture Design

BBW
Jasper Hu, yjhu, 4356241
Naqib Zarin, nzarin, 4384474
Ashay Somai, asomai, 4366220
Ymte Jan Broekhuizen, yjbroekhuizen, 4246586
Luat Nguyen, tlnguyen, 4467574

June 23, 2017

Contents

1 Introduction

This document provides a high-level description of the final product and the system it uses. The end product exists out of a mobile application in the Android environment, which is developed in the Java programming language. Using the Android environment and the programming language Java, we are developing a blockchain-based web-of-trust. The idea is that you use blockchain to verify the authenticity of the user. Blockchain itself is like the name says a chain of blocks and each block contains the information to verify a user, normally this is a public key.
The Android environment is an open-source platform and operating system, which is developed by Google. This operating system is available for many devices. However, we are focusing on the mobile devices.

Hereafter, the paper will be composed of the following sections. The next section will be the design goals of this project. In the third part, there will be more information about the software architecture views of the project. Specifically, these views are Subsystem decomposition, Hardware/ software mapping, Persistent data management and Concurrency.

1.1 Design Goals

To let the project move in an orderly fashion we had to fulfill several design goals so that everyone could keep working on the project without letting it become complete chaos. When no strict checks are done for code aspects such as method name conventions, useless unit tests, or straight bad code, the code can quickly evolve into a unmanageable monstrosity. This is especially true when five enthusiastic coders work on a project with such vague specifications and an initially unclear design.

Since there are too many design goals, we split it in six different aspects: availability, manageability, performance, reliability, scalability, and securability. Consequently, these six aspects are elaborated in the following sections.

1.1.1 Availability

The idea is that there will be a working version after every sprint week. This way, the product owner can see the features, we are working on, every week. So in case, we are going the wrong way, we can still turn to go the right way during the meeting.

1.1.2 Manageability

To manage our program, we use the version control program Git, and it is stored on GitHub. In case there is an available update, you can just pull the working release version from the master and use that one instead. Our working or developing branch is the branch 'develop'.

1.1.3 Performance

The program should be able to be processed on a mobile device, so the blockchains that the program is using should not be too large. It should even be limited to a specific capacity, which could be a part of the capacity of the device. Since most of the user would like to use their mobile phone for other applications as well, it would be good to let the application not exceed about a couple GBs of memory. Most of the mobile applications have a capacity of less than 100MB and we would not like our program to use a lot of disk space, which limits the user to use their phone for other applications.

1.1.4 Reliability

To ensure the reliability of our program, we use several techniques. The first technique that we are using is Test Driven Development. This technique increase the probability that we completely understand what a specific feature should do. Since it improves the understanding of the feature, we can develop it more reliable and more efficiently. Testing is done using unit tests, integration tests and regression tests.
The next technique that we are using is Travis Continuous Integration. After every commit to our program, or in other words, after every update to our program, the program is automatically

tested. Consequently, it will let you know, whether the update contains any errors.
Testing is important because it helps to find bugs and preventing unwanted behavior.

1.1.5 Scalability

The whole program is decentralized to a mobile application. The persisted data is saved on the server and the server is the device itself. The calculation of some hashes only requires little processing power, so there won't be a bottleneck on the processing power. The only problem is that you have to save the blockchain to your device. So the only limit would be storage capacity of the device. However, a single block uses few storage space, so that the limit would be colossal for the current phones. There are no other bottlenecks, so the program scales pretty well. The program runs smoothly on multiple devices, like Chinese and Google Android devices.

1.1.6 Securability

The public and private key pair is generated using the ED25519 protocol. This protocol is very safe and is also used by banks like Bunq. When the public and private key pair have been generated, they can be saved to files. When they are saved, they are first encrypted using the AES-CTR protocol using the hex representation of the public key as password. This can and/ or should be changed later, to use another password.

The first/ outer layer is the password protected SQLite file, on which the persisted data is saved. The second/ inner layer of protection is that the (valuable) information, like IBAN or contact information of the owner of the blockchain, in a block could be saved using the AES-CBC encryption protocol using the public key of a contact as encryption key. This encryption protocol is chosen, because it is used a lot in the cryptography world and is assumed as safe. Next, it ensures that the data is not readable directly, but should be decrypted first, when read out of the database. Likewise, the data should be encrypted first, before it is stored in the database. This is not a impenetrable layer, but will make it harder to read out the data.
If all layers are penetrated, there is still an option to revoke a block. This way the block, containing information about another user will be invalid. Next to that, every block contains a trust value, representing the trustworthiness of a user. More information about the trust value can be found in the next subsection. Revoking a block can be done using the interface, where you select your block and press revoke afterwards. If you revoke a block, the block will be seen as invalid.
Finally, when you add a block to your own blockchain, there is first a signature generated using the byte array representation of the public key of the block that has been sent. And in the API, when the block is received, the signature is validated, before adding it to your blockchain.

1.2 Trust Value

The trust value is a floating-point value between 0 and 100. The trust value is initialized to 10. This value has been chosen, because the value should not be too high. Otherwise, the upper limit will be reached too quickly. Revoking a contact, will set the trust value to 0. There are 3 transactions: 'Successful transaction', 'Failed transaction' and 'Verified IBAN'. When these actions occur, the trust value gets updated as well. This update makes use of the distribution function: $100(1 - e^{-0.05x})$. This distribution function is based on an exponential cumulative distribution function and ensures that the trust value in the beginning increase more than when later on. When a successful transaction occurs, the corresponding parameter to the distribution function of the trust value of a block will be calculated and after that, it gets incremented by the regularization parameter for transactions and that has been initialized to 1. The process for a verified IBAN is almost the same, but the regularization parameter has been initialized to 3. These regularization parameters take care of the mutation growth of the trust value and have been initialized randomly and can be calibrated later on.

2 Software Architecture Views

The program is interconnected using different parts and the software architecture views elaborate on them. In the subsystem decomposition, it shows how the system is divided into subsystems.

Secondly, in the hardware/software mapping, it illustrates how the hardware subsystems are connected to the software subsystems. As third, in the persistent data management, it shows how and where the data is persisted. And finally, in the concurrency part, it shows how the resources, processes are shared and how the communication works between them.

2.1 Subsystem Decomposition

The client uses the Mobile Application to handle all requests. Or in other words, communicates with the server using the mobile application. This mobile application also processes all requests. The mobile application consequently communicates with the database using the Android SDK. Since the mobile application processes all requests by itself, there is no other communication between the database and the mobile application.

As an analogy, this application can be thought of as a storage company for valuables. The company is divided in four main components:

- Component 1: Storage, a place where all the valuables are stored safely and orderly.
- Component 2: Structure, the rules on how the valuables should be stored, arranged such that the ownership of each valuables are guaranteed not to be lost.
- Component 3: Logistic, the department where all the logistic is handled
- Component 4: Customer support: this is where customers are coming in with their valuables.
- Component 5: Real-life network crawler: in this component, our program reads in and processes the data crawled from the live Tribler network.

The structure of the application are divided in four main components:

- The Database Package: the door to the database where the information are stored safely and orderly such that we can be sure that the identity and trust value are correct when we access it later. This package represents the component 1 of the company.
- The Blockchain Package: the rules and frames on how the information should be shaped into blocks, chains and users so this information could be added to the database orderly and/or represented to the user properly. This package represents the component 2 of the company.
- The Controller Package: this is where all the logistic of the application happens, such as what and how a transaction should be handled, what happens when a user wants to remove a contact from his contact list. In short, this package is designed to carry out the request of the user. This package represents the component 3 of the company.
- The API Package: This package gives the user the right tools to make requests to the program. This package represents the component 4 of the company.
- The crawler package: This package represents the reading and processing of the data, crawled from the live Tribler network; when the crawled database has been read in, it gets converted to blocks and thus it can be used in our program.

We describe the description of different subsystems below.

2.1.1 Blockchain

This subsystem contains the implementation of the web-of-trust structure of the library, and utilizes several objects to fulfill this implementation.

- The user represents a real-life human, which can be yourself, a friend, or an unknown user. We make a conceptual distinction between friends and unknown user because the friend (classname Contact) is already trusted and has all information already reside in our own database. Once you try to pair with an unknown user, we call it an acquaintance. This type of user is not yet added to your database but is first checked for correctness. The acquaintance also keeps their data with them instead of storing it in the database so that it can easily be sent over the network. After it is trusted it becomes a regular user and is

stored inside the database. This subtle but important difference is both for convenience and for security.

We store but a few properties about all users:

- His name
- His unique IBAN number (bank number)
- His unique public key

The user could be either the direct user of the application or his contacts.

- The second important object is the block concept. It holds information about a particular action or transaction and is stored inside the database. For each transaction/action such as adding a contact into the list, removing contact from the list, a block is added into the chain of an User, much like an entry in a log book. A Block contains the following information:
  - One user representing the owner of the chain this block belongs to
  - One user representing the contact which is saved into the chain
  - A hash value linking this block to the previous block in its chain
  - Another hash value linking this block to a block in the chain of a contact of which this block contains information
  - Its own hash which takes into account all properties

2.1.2 Controllers

There are three controllers in our library: the BlockController, BlockVerificationController and TrustController. These controllers handle the interaction between the Database and the API by using the BlockChainComponents.
In the BlockController the creation, addition and revocation of blocks is controlled. The API call is forwarded to this class and is executed in this class and eventually is stored in the Database.
The TrustController is made for the transactions between users. It is named TrustController because after a transaction, which can be for example a successful one, the trust changes in a block. The call to the TrustController is made in the API, when this call is made, the TrustController returns the updated block and this is updated in the Database using the BlockController. BlockVerificationController is used for checking whether blocks do actually exist in the Database, also whether a block does not differs from the original one which currently exists in the Database.

2.1.3 Database

The Database class contains separate read and write functions. The read function only accepts reading queries and vice versa so that you do not accidentally perform a write while trying to read some data and that you will be able to see this at compile time. Each specific read query contains specific logic on how to execute this query so that the database only contains generic read, write and parsing code.

Example read queries are "get a user with this key", "get the chain of this user", or "retrieve the entire multichain". Example write queries are "add this block" or "add this user". The implementation of these queries are trivial with every query being simple - e.g. no join operations or subqueries.

2.1.4 API

The API is a class that heavily depends on other classes to manage its functionality. All its methods are small proxy methods that call into the relevant parts of the system, usually blockController, to provide the functionality it promises. It is able to throw a relevant error when the functions are not called correctly, such as when a block already exists or when a hash mismatch occurs.

2.2 Hardware / software mapping

The mobile application runs on a mobile device, which has to use the Android Operating System. The mobile application makes use of the Android SDK to connect to the database. The database also runs on the mobile device itself. The Android SDK uses the SQL language to communicate with the database. You can find the class diagram in the sprint files folder in the root of the GitHub repository.

Our library uses the file system, stored on the internal memory or on the SD-card, to store its database and the encrypted private key file. There is no bluetooth, USB, or networking hardware used, instead providing mostly computational functionality. Some amount of RAM is (obviously) used for the database and for our application. The screen is used in the demo to show a simple tree with some demo functionality; this uses the standard Android GUI libraries and a custom library to show a tree view. These libraries use the GPU of the mobile phone.

2.3 Persistent data management (file/ database, database design)

The blockchain of a user is persisted in the SQLite database since we do not want to lose the data upon closing the application. The blockchain consists of blocks, and since the blockchain is persisted, the blocks are persisted too. Next, to that, The general information about the user, that we need for later, is saved in the database as well.

3 Persistent Data Management

Since our application should in theory be able to handle thousands of blocks in the blockchain we have chosen to use a database from the beginning of the development of our application. All our users and blocks are stored in this database; this is a fast solution, both in terms of development speed as well as in execution speed: the database software is included in each android distribution and there is a ton of resources available online. It also took the burden from us to develop a form of serialization that supported multiple lookup methods without loading everything into memory. In return we had to invest some time into setting it up and developing the queries necessary to create tables and to read and write records.

Our earlier design included one class, DatabaseHandler, which contained a method for each query. This god class handled database initialization, query construction, error handling, and result parsing. The new design splits all query logic from the database logic by letting subclasses derive from a base class, Read- or WriteQuery, that in turn can be fed into a fresh Database class which executes it and gives it opportunity to organize each result.

3.1 Database Design

On first application startup the database is created which is then persisted through the lifetime of the app. In the database there are two tables: one for the blocks, one for the users. The block table, in which all blocks are stored, looks as follows:

- Sequence number, starting from one for the genesis block
- SHA-256 hash value of the block for verification
- Hash values of two other blocks: the previous block in our own blockchain, and the hash of the last block of the contact of this block. If not applicable - when there is no sender or in case of the genesis block - the value "N/A" is used.
- Public keys of the owner and the contact of the current block, or "N/A" if not applicable.

There is another table for the user, which associates public keys with names and IBANs. This information is calculated into the block hash; not only the public key of the users is used. This table uses the public keys of the users as primary database key.

3.2 Query Execution

All queries inside our app are simple queries without subqueries or joins. Typical queries are "get all blocks owned by this user" or "get the information on the user with this public key". All these queries can be done without inner queries or joins by just filtering on columns. Some queries use the order-by statement; for example the get blocks from user sorts the block on sequence number so that a sequential list can be made later. We do not use any indices as the app is fast enough as it is; however indices are easy to add in SQLite if the query speed becomes too slow later on.

3.3 Personal Keypair Storage

While our own keypair - public and private key - could in theory be added to the database, it is cleaner to store the keys in separate files. The keys are first encoded using the PKCS8 protocol and converted to a byte array. Afterwards, the files are encrypted using the AES-CTR encryption protocol and stored with a password. Our API provides the possibility to change this password.

4 Concurrency

Our library does not currently spawn multiple threads. The only two files that need to be read are read at startup; there is no network or bluetooth activity so our library does not need or support callbacks or multi-threading. The only thing that would profit from concurrency is the database part. All calls to our API block until the involved query is complete or until an exception is thrown; this is unavoidable since the underlying SQLite database does not provide asynchronous support.

However, parts of our API may still be called from other threads to achieve concurrency. All read queries - retrieving blocks, chains and users - may be called simultaneously and will be executed in parallel. As of yet only one thread may write to the database at the same time. It may be possible to configure SQLite Android to allow multithreading, however this seems not that important as most uses of our library will be checking which contacts have which keys with only the occasional update as you meet new contacts.

5 GUI

When we open the app we will enter the so called main page. In this page we will have three options. First, we can go to the contacts page where a list of contacts and some more information will be displayed. The second option is to pair with a new friend. The third (temporary, for testing purposes) option is to clear the database. When we click on the pair button, we will see an overview of contacts that we are able to pair with (see figure 3). When we click on a person, say TestSubject1, we go to the Friends Page (see figure 2). The Friends Page will contain the name of the person you just connected with, his or her public key and his or her iban. These values will be retrieved once those blocks are made (when clicked on that person in the Pair Page) and passed on to the Friends Page. Also his trustworthy rating will be shown and finally, you will have the option to add this person to your contact list or visit his contact list.

6 Glossary

- AES
  An abbreviation for Advanced Encryption Standard, an encryption protocol.
- Android
  Open-source operating system mainly targeting mobile phones; developed by Google.
- Android SDK
  The Software Development Kit used for creating and debugging applications for the Android operating system.
- Block
  Element from a blockchain that contains a user-key mapping. Can also be used to revoke a key. Information is hashed so that is hard to create fake blocks with the same hash.

- Blockchain
  Tamper-proof list of blocks. Tamper-proof because it uses a one-way hash function to link its blocks, making it nearly impossible to create a chain with the same hash.
- Bluetooth
  Open standard for wireless communication between devices within a short range.
- CBC
  Abbreviation for Cipher Block Chain. This protocol ensures that the data is divided into several blocks using a blockchain structure with a XOR (exclusive or) to the previous block.
- Cryptographic hashing function
  One-way function that takes any amount of data and generates a hash (fixed-size string). It is nearly impossible to find some other data than the original while still keeping the same hash.
- Git
  Fast Version Control System that keeps track of current and previous modifications of source files while allowing multiple collaborators work on the same files.
- GitHub
  Website that provides Git storage allowing code reviews and integration with diverse applications.
- Google
  Large search company that mainly focuses on Internet software and mobile application development.
- Hash
  Fixed-size result of a (Cryptographic) hashing function used for verification of correctness of data.
- IBAN
  Abbreviation for International Bank Account Number, used to wire money between associations and people.
- Integration test
  Test for testing whether multiple components can cooperate as expected.
- Java
  Object-oriented programming language for developing software.
- Middleware
  The layer that enables a subsystem to communicate with another subsystem. One of these subsystems does not have to be in the application itself.
- Native application
  An application specifically built for one device. Often compiled to machine code making it a generally fast and small application.
- Open-source
  Term meaning that the source code of a program is available to the public.
- Private key
  Piece of information that only you have. Often an associated public key is generated at the same time.
- Public Key
  Piece of information that you can make available to the public, then prove to someone that you own the corresponding private key.
- Regression test
  Test that checks whether new bugs have been introduced when code is added or updated.

- Server
  Remote computer, which could be used to persist data or handle any other requests.
- SHA-256
  Cryptographic hashing function that is unbroken and widely used today.
- SQL
  Abbreviation for Structured Query Language. Used for interactions with databases.
- SQLite
  SQLite is a database engine which works with the SQL language.
- Test Driven Development
  A way of developing software where you first write unittests and later write the code providing the functionality. Has an advantage of making you explicitly think about what you want without worrying about implementation details.
- Unit test
  A small test to check a single software component for correctness.
- Web application
  Application in the form of a website.
- Web-of-trust
  A concept in cryptography to indicate a network of users who trust each other and own each others public keys.