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 16, 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

Since the concept design goals are too large, it is easier to split it in six different aspects: availability, manageability, performance, reliability, scalability, and securability. Consequently, these six aspects are elaborated in the following sections.

1.2 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.

1.3 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'.

1.4 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.

1.5 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 behaviour.

1.6 Scalability

The whole program is decentralized to a mobile application. The persisted data is saved on the server and the server is saved on the device as well. The calculation of some hashes only require 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.

1.7 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, in the future it will send a message that the block will be invalid, so the others can revoke it as well.
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.

1.7.1 Trust Value

The trust value is a 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.

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.

Below the description of each package are presented:

- Blockchain Package
  Blockchain Components package contains all the classes which are necessary for the implementation of the web-of-trust structure of the application.
  - User
    The User class represents the data about a particular user/ participant of the application. An user object contains the following:
    * The name
    * The unique IBAN number (bank number)
    * The unique public key

    The user could be either the direct user of the application or his contacts.
  - Chain
    The Chain class is a representation of an user's non-erasable logbook, which notes all the contacts which it's owner has added (and removed from) into the contact list. This class, being strictly an information older, also serve the purpose of making the Blockchain Components more clear by making it explicit that each user has his/her own logbook (to be more technical, his/her own blockchain). A Chain object contains the following:
    * An user object which is the owner of this Chain

    * An ArrayList object which is used to contain the block objects
  - Block
    The Block class represents an information holder of a particular action/transaction in 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 the log book example. A Block object contains the following:
    * An User object representing the owner of the chain this block belongs to
    * An User object representing the contact which is saved into the chain
    * A BlockData which contains various information about the block
    * A Hash object representing the block's own unique hash key.
  - BlockData
    Since block has many fields so that the constructor became large and unwieldy, an intermediate class BlockData has been introduced which acts as the data storage for blocks. You can set any field in peace, and when you are done simply pass the BlockData object to the constructor to instantiate a Block. A BlockData object contains the following:
    * A BlockType object representing the type of the block (see class below)
    * An integer representing the order of this block in the chain. The more recent a block is added, the larger this number is.
    * A Hash object representing the hash key of the preceding block (the most recently added block) in the chain.
    * A Hash object representing the hash key of the block which contains the contact information that we want to add to the block chain. This is the block which is given by the user whom you wanted to pair with.
    * A double, which represents the trustworthiness of that block.
  - BlockType
    The BlockTpe enumeration represents the type of an block object and contains:
    * GENESIS type representing the first block of the chain, which is actually the chain owner
    * ADD_KEY type representing a block which concerns the addition of a contact from the chain
    * REVOKE_KEY type representing a block which concerns the removal of a contact from the chain it belongs
  - Hash
    The Hash class represents the hash key of a block. This class is simply a String holder and is invented for the purpose of making the application code more clear by not using String value as a parameter everywhere.
  - Acquaintance
    The Acquaintance class represents a person who you have paired with, but have not added into your own database yet. This generally happens after two participants have sent each other contact information via Bluetooth, but they are still deciding whenever to add the other person. This class inherits from the User class and contains:
    * A "multichain" which is a list of Chain representing the Acquaintance's database.

- Controllers
  The controllers consists of three parts, namely the BlockController, BlockVerificationController and the 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.

- Database
  All our users and blocks are stored in a SQLite 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 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.

  The Database class contains separate read and write functions. The read function only accepts ReadQueries and vice versa so that you do not accidentally perform a write function and will be able to see this at compile time. The read functions calls execute on the read query with SQLiteDatabase as argument; the default execute method creates a query with overridable filter, selected columns, table name, and order; then the overridable method parse is called on the read query. By extending the ReadQuery each subquery is able to select the filtered results they want, then parsing those results and making the results available via getters.

  A large number of queries have been implemented:

  - BlockExistQuery checks if a block with a given owner and containing the public key of a given contact already resides inside the database to avoid duplicates. It also checks the revoke boolean of a block so that you can check if certain blocks are revoked.
  - ChainSizeQuery retrieves the chain size of an owner, useful for determining the sequence number for the next block to be created.
  - LatestBlockQuery retrieves the last block of the chain of a certain user, useful for linking a new block to the rest of the chain.
  - GetChainQuery retrieves all blocks inside a certain owner, useful for a chain needs to be sent over bluetooth.
  - GetUserQuery retrieves a user's info based on its public key. This is useful because a block object in our code needs an entire User object for its owner and contact attributes, while the database only stores the public keys of those user. An additional two queries are thus needed to obtain the full user objects, stored inside the user table.
  - UserExistQuery checks if a user exists.
  - DatabaseEmptyQuery checks if the database is empty. When the app starts, it needs this query to check if any genesis block has to be generated. This query actually only checks the user table, but since every block needs a corresponding user and every user is thus added before any blocks, this is no problem.
  - BlockAddQuery adds a block database. It throws a DatabaseException when a block with the same hash is already inside the database.
  - UserAddQuery works the same as BlockAddQuery, but adds a user instead of a block.
  - UpdateTrustQuery takes a block, and modifies the corresponding block in the database with an updated trust value. This is the only part of the block that is mutable: the trust value does not have any influence on the block hash and is instead used for a comprehensible history of the trust you have in another user given their transaction history.

2.2 Hardware/software mapping (mapping of sub-systems to processes and computers, communication between computers)

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.

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.

2.4 Concurrency (processes, shared resources, communication between processes, deadlocks prevention)

The users of the blockchain will have to communicate with each other, using a communication protocol. Currently, we are simulating a communication protocol, but in the future we will use a Bluetooth protocol. Every user keeps their blockchain, and every other user can see this blockchain. The other user now can add the initial user to his own blockchain, and extend his chain. The users should also be aware of their own keys. In case these get stolen, the user should revoke it.

As for the shared resources, the idea is to keep everything decentralised. This means that there is no central server which keeps track of the blockchain; the central server is the device itself. Every user keeps track of their own blockchain, and as stated earlier, the communication between users gives users the opportunity to know about the blockchain of others.

3 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.

4 Glossary

- AES
  AES is an abbreviation for Advanced Encryption Standard and is an encryption protocol.
- Android
  Android is an open-source operating system, which is developed by Google.
- Android SDK
  Android SDK is the software development kit, which is used for creating applications for the Android environment.
- Block
  A block contains information about another user. This information could be hashed.

- Blockchain
  A blockchain is a chain of blocks, of which a block contains the hash of the previous block.
- Bluetooth
  Bluetooth is an open standard for wireless connections between devices on a short distance.
- CBC
  CBC is an 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
  A cryptographic hash function is a hashing protocol, but it has some more properties, which ensure more security. For example, it is a one-way protocol; it is not feasible to find the input of the data using the result of the hashing function.
- Git
  Git is a way to control the versions of your program. You could update, delete or edit your version of the program using this.
- GitHub
  GitHub is a website that allows storage and parallel development of code for programming teams.
- Google
  Google is a large IT company which mainly focuses on developing software and hardware.
- Hash
  A Hash is a protocol to map any information to a particular arbitrary size.
- IBAN
  IBAN is an abbreviation for International Bank Account Number.
- Integration test
  An integration test tests whether multiple components can cooperate as expected.
- Java
  Java is a object-oriented programming language for developing software.
- Middleware
  Middleware is 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
  A native application is an application, which is built with the SDK of the device and therefore is operating system dependent.
- Open-source
  Open-source means that the source code of a program is available to use to the public.
- Public Key
  A public key is one of the two keys used in cryptography to encode information.
- Regression test
  A regression test tests whether new bugs have been introduced due to updated code.
- Server
  A server is a remote computer, which could be used to persist data or handle any other requests.
- SHA-256
  SHA-256 is a cryptographic hashing function of the second SHA family.
- SQL
  SQL stands for Structured Query Language and is used to make mutations to a database.

- SQLite
  SQLite is a database engine which works with a SQL database.
- Test Driven Development
  Test Driven Development is a way of developing software, of which you write the tests first and build the program around it.
- Unit test
  A unit test tests a single component to verify its behaviour.
- Web application
  A web application is an application in the form of a website.
- Web-of-trust
  Web-of-trust is a concept in the cryptography to create the trustworthiness between a user and its public key.