BankChain - Architecture design

Chainable Technologies

H.G. van de Kuilen
rvandekuilen, 4226151
J.C. Kuijpers
jckuijpers, 4209915
E.J. Sennema
esennema, 4496582
M.R. Kok
mrkok, 4437659
I. Dijcks
idijcks, 4371151

May 2017

Contents

1 Introduction

In this document, a high-level description of our app is given, together with the design goals. There is a large focus on the subsystem decomposition and how practical problems, like persistence and concurrency are solved. The app being created is an additive to a web-of-trust on a mobile phone. It adds the possibility to verify a peer using their public key and bank account, by sending a cent to the peer with a challenge. The solved challenge is sent back using another cent. The product relies on the difficulty to get a bank account in a false name or on illegal markets.

1.1 Design goals

The product being created is foremost a research project and a proof-of-concept. This is the reason the three main design goals are security and code quality.

1.1.1 Secure

As the app is about getting verification of a peer in a secure, encrypted and trusted way, making the app secure is an important part of the design. Encrypted challenges over bank accounts is all nice and good, unless your private key can easily be stolen by a simple backdoor in the app.

1.1.2 High code quality

This project being foremost a proof-of-concept of an extension to a web-of-trust, code quality is very important. With high code quality, the processes, advantages but also the limitations can be easily explained to future developers (of any project) and researchers. In general, higher code quality also improves the development velocity (because of less bugs and more understanding of the code), product quality (because of less bugs and more reviews) and mindset of the developers (less issues, more understanding, more freedom in code changes due to testing).

To get higher code quality, a lot of the code must be tested and code must be reviewed by peers. The program needs a clear structure, both for testability (mocking), to be able to switch components and to be able to work on different parts in parallel.

2 Software architecture views

In this section we describe what the architecture of the code and the product is.

2.1 Subsystem decomposition

The system is divisable in two main parts: a GUI and all functionality. The functionality can be divided in many parts.

Bank interface

The connection to the bank is implemented using an set of classes that cover the API calls to the bank and handle all other bank related steps. This is an abstracted area, so multiple banks can be implemented. The main implementation is with the Bunq API. There is also a mock implementation of a bank so other code can be tested without actually calling into a bank. The Bunq implementation uses a HTTP library named Retrofit (Square, Inc., 2017b).

Challenge / Response system

The challenge / response system is responsible for creating a secure challenge, responding to the challenge and verifying the response with the original challenge. This is implemented using asymmetric cryptography, more specifically the ed25519 Elliptic Curve implementation of it. We plan to make the API of this package as simple as possible to hide this fault-prone code, and allow possibly implementing other asymmetric technology, e.g. RSA.

Blockchain interface

As we don't implement a blockchain but still need to talk to it, we will have a dummy blockchain implementation that uses memory and files instead. The API of this will be cooperated with the project that write an actual blockchain. For this reason, there will be multiple implementations of the blockchain interface: a mock, a dummy and possibly an actual blockchain.

The blockchain mock is also capable of creating a public and private keypair for the user. The public key is stored in blockchain, and the private key is stored on the device.

Verification logic

This system contains the business logic of the verification. It receives requests from the API, asks for data from the blockchain, creates a challenge and sends it with the bank interface.

GUI

The graphical user interface shows information to the user and accepts input. The GUI is never talked to by other components: the GUI only talks to them. To receive notifications and updates, it will need to listen on the bus. See the concurrency chapter.

2.2 Hardware/software mapping

This system uses both peer to peer and third party communication, the third party being a bank. All software written in this project resides on two peers, A and B. Both are Android phones and have the BankChain app installed. They are the same, equal peers, both server and client. A peer can ask another peer if it wants to be verified. This is done using the RequestClient to a RequestServer with peer-to-peer communication. A peer can send a transaction to a bank using the bank implementation. The whole flow is handled by the BankVerification subsystem. It calls into the other subsystems in order using direct calls or the BUS. It also listens on the bus for new actions to perform.

dit moet nog beter / anders

An android process is a single process with multiple threads. Multi-threading is done by a bus and runnables.

2.3 Persistent data management

There are a couple of persistent data items for BankChain: a list of previously verified IBANs, settings like the bank API code, a private key and previous verifications.

The verified IBANs would, when combined with the other project groups, be stored in the blockchain. The private key would be handled by group 1. The only things left is settings, and a list of verifications for display. This can be done using a file based storage.

As the app is not merged with other apps, at this point (or possibly at all), there is storage for a dummy blockchain and a private and public key pair. This is not in the scope of the project but is useful for testing and developing the application. Both will be stored in files.

2.4 Concurrency

As the GUI must remain usable, even when running longer operations, long operations should not be run by the main thread. To do this, Android can use threads and runnables. Together with anonymous function, one can run code in the background. However, to do any GUI updates, there must be a switch to the main thread again (Google, Inc., 2017). This gets very messy, especially when going multiple levels deep. To solve this issue, we make use of a bus (pipeline) that runs all its tasks on the main thread. For the bus we use a library (Square, Inc., 2017a). When a background runner updates some data, it notifies the GUI of the update using a message on the bus. No switch to another thread is necessary. All GUI code can then stay inside the GUI component, as it only needs to register for messages.

For larger asynchronous tasks, an AsyncTask can be used as well. This however needs subclassing. (Guy, 2009)

Glossary

API Application Programming Interface. 3

blockchain is a distributed database that maintains a continuously growing list of records, called blocks, secured from tampering and revision (Wikipedia, 2017). 3

bunq is a Dutch payment app and bank that provides a developer friendly and open programming interface.. 3

GUI Graphical User Interface. 3

HTTP HyperText Transfer Protocol. 3

RSA Cryptosystem named after Ron Rivest, Adi Shamir, and Leonard Adleman. 3

References

Google, Inc. (2017, March 21). *Processes and threads.* Retrieved May 1st, 2017, from `https://developer.android.com/guide/components/processes-and-threads.html`

Guy, R. (2009, May 6). *Painless threading.* Retrieved May 1st, 2017, from `https://android-developers.googleblog.com/2009/05/painless-threading.html`

Square, Inc. (2017a). *Otto.* Retrieved May 1st, 2017, from `http://square.github.io/otto/`

Square, Inc. (2017b). *Retrofit.* Retrieved May 1st, 2017, from `http://square.github.io/retrofit/`

Wikipedia. (2017). *Blockchain — wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/w/index.php?title=Blockchain&oldid=778628009` ([Online; accessed 4-May-2017])