TrustChain Android Documentation

Release 0.0

Wilko Meijer, Rico Tubbing

Nov 13, 2017

Contents

CHAPTER 1

Introduction

TrustChain Android is a native Android app implementing the TU Delft style blockchain, called TrustChain. This app provides an accessible way to understand and to use TrustChain. The app is build as part of a Blockchain Engineering course of the TU Delft. It is meant as a basic building block to experiment with blockchain technology. This documentation should get you started in the workings of the app, however for thorough understanding, reading other documentation and looking at the source code is a necessity.

We have tried to make the code clear. However, this app was not build by Android experts so please don't hold any mistakes or weird structures for Android against us. Instead, please let us know what could be improved, or provide a fix yourself by submitting a pull request on GitHub.

CHAPTER 2

What is TrustChain?

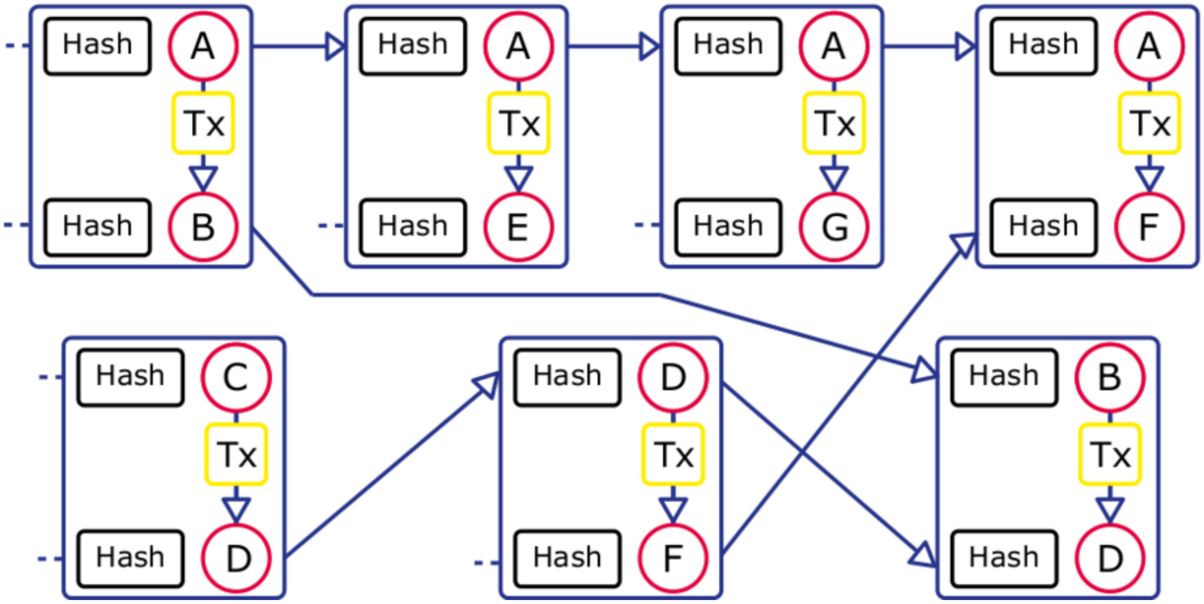
In this section a short explanation of TrustChain is given. For a more extensive and complete overview please refer to the documentation that can be found on the blockchain lab website.

TrustChain is designed to deliver scalable distributed trust. This is achieved by letting two parties agree to a deal or transaction and storing the proof of the transaction in an immutable way. The transaction is stored in a block and a series of blocks are chained together to form a blockchain. To achieve scalability, each peer keeps track of their own blockchain which holds all the transactions they participated in. So when two peers want to agree on a transaction peer A creates a block with the transaction, signs it, adds it to their chain and sends it to peer B. Then peer B creates a block based on the block it received from A, signs it, adds it to their chain and sends it back to peer A. Both parties now have a proof of their transaction in their chain and they also have a copy of the block from the other party. For a more detailed description of how a block is created see *Creating a block*.

Many transactions by many peers thus create a complicated network of entangled chains. However, the size of the chain of one peer is entirely dependent on the amount of transactions they participate in, so the whole system scales very well. The following picture tries to make the entanglement more clear. Each unique letter in the image represents a unique peer.

2.1 Dispersy

For peer discovery and handling the crawl requests for obtaining knowledge about peers, Dispersy is used in the ipv8 implementation of TrustChain. Dispersy is not implemented in this Android project and discussing it is out of scope for this documentation. For more information read the Dispersy readthedocs page.


Hash
A
Tx
Hash
B
Hash
A
Tx
Hash
E
Hash
A
Tx
Hash
G
Hash
A
Tx
Hash
F
Hash
C
Tx
Hash
D
Hash
D
Tx
Hash
F
Hash
B
Tx
Hash
D

CHAPTER 3

Creating a block

In order to complete a transaction with a peer, we need to create a block. A block in TrustChain is a little different than in bitcoin-style blockchains. In bitcoin-style blockchains, a block is a collection of transactions that happened in the network. A block is created by a node and is propagated through the network. All connected nodes validate the block and the transactions. In TrustChain a block is formed by two peers who wish to agree on a transaction. Therefore a TrustChainBlock only has one transaction.

Both parties need to agree on a transaction, so there has to be some interaction between peers. The way this is done in TrustChain is to first create an incomplete block, called a half block. This half block is then send to the second peer, which creates a full block from the half block and sends it back to the first peer. This process is explained in more detail below.

3.1 Structure of blocks

A block has the following attributes:

- `public_key` - The public key of the peer that created this block
- `sequence_number` - Represents the position this block has in the chain of the creating peer
- `link_public_key` - The public key of the other party
- `link_sequence_number` - The position the connected block has in the chain of the other party
- `previous_hash` - A hash of the previous block in the chain
- `signature` - The signature of the hash of this block
- `transaction` - The data that both parties need to agree on, this can be anything, from text to documents to monetary transactions

Note that `link_sequence_number` will be unknown for the first half block created, because peer A won't be sure when peer B inserts the linked block in his chain. This will stay unknown, as updating a block already in the chain is not desirable, since it might invalidate later blocks. When an interaction is completed peer A will have the block of peer B in its database as well, so it can always find out the position of the linked block in peer B's chain.

3.2 Create block

There are two situation that require creating a block. Initiating the creation of a transaction with another peer and completing a block send to you by another peer. This is both done by calling the `signBlock` method in Communication.java. This method calls the `createBlock` method in TrustChainBlock.java, signs the block, and validates the correctness of the block, before it gets added to the chain and send.

3.2.1 Initiating a transaction

When you want to initiate a transaction, you need to provide the bytes of the transaction, your public key, and the public key of the other party, and a link to the database containing your chain to `createBlock`. The latest block in your chain will be retrieved from the database, to be able to set `sequence_number` and `prev_hash`. The other attributes will be set according to the input, `signature` will remain empty for now.

3.2.2 Received a half block

A half block was received and it contains a transaction that we agree with. In this Android implementation we always want to complete the block, regardless of the transaction, so we don't need to check the transaction. The attributes are again set according to the input, with as difference that we now retrieve `transaction` and `link_sequence_number` from the linked block. `signature` will again remain empty.

3.3 Sign block

The next step is signing the block. This is as simple as creating a sha256 hash of the block and giving this digest to the build-in signing function of the crypto library.

3.4 Validate block

Block validation is the most important step here, as this ensures the validity of the blockchain. The validation function is located in TrustChainBlock.java. There are 6 different validation results:

- `VALID`
- `PARTIAL` - There are gaps between this block and the previous and next
- `PARTIAL_NEXT` - There is a gap between this block and the next
- `PARTIAL_PREVIOUS` - There is a gap between this block and the previous
- `NO_INFO` - We know nothing about this block, it is from an unknown peer, so we can say nothing about its validity
- `INVALID` - It violates some rules

The validation function starts of with a valid result and will update the validity result according to whether the rules hold for the block. Validation consists of six steps:

- Step 1: Retrieving all the relevant blocks from the database if they exist (previous, next, linked, this block)
- Step 2: Determine the maximum validity level according to the blocks retrieved in the previous step
- Step 3: Check whether the block is created correctly, e.g. whether it has a sequence number that comes after the sequence number of the genesis block

- Step 4: Check if we already know this block, if so it should be the same as we have in our database
- Step 5: Check if we know the linked block and check if their relation is correct
- Step 6: Check the validity of the previous and next block

For a more detailed explanation of the validation function, please take a look in the code and try to understand what happens there.

3.5 Links to code

- Block structure in ProtocolBuffers (Message.proto)
- All block related methods (TrustChainBlock.java)
- Sign block method (Communication.java)
- Validation result (ValidationResult.java)

Also see the readme on the ipv8 github

CHAPTER 4

Connection to a peer

In order to send blocks to other people, you will need to find peers. In this simple app, this has to be done manually. In IPv8, this is done with help of Dispersy.

There are two ways to connect to a peer: either via local network or bluetooth (note that bluetooth is not working perfectly). Therefore connecting to a peer is very simple. Find out the IP-address of a peer who has opened the app, listed at the top in the main screen. Fill in the ip address and the port number (default hardcoded as 8080) and press connect. Now the app will go through the steps as explained in *Creating a block*.

Sending a transaction to another peer via bluetooth requires you to pair the devices via the Android bluetooth manager. After they are paired, the app will list the devices your device is paired with. To initiate a transaction, press on one of the devices.

4.1 Connection

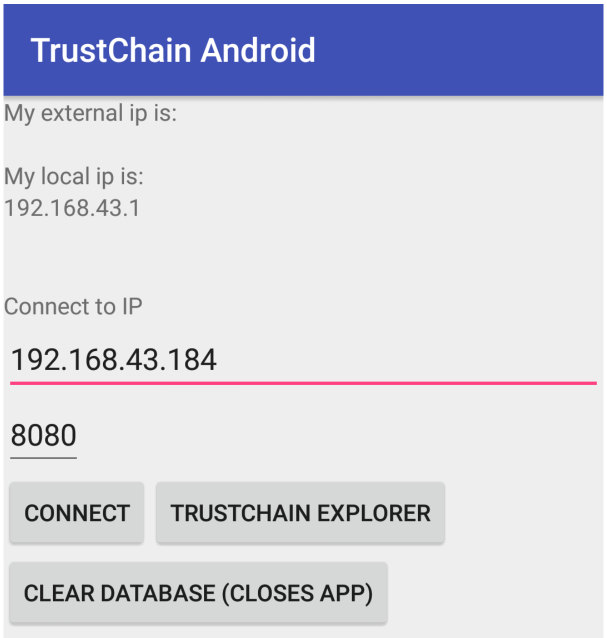
The class *Communication* is responsible for handling the data that is received either to bluetooth or WiFi. This class is abstract so that both type of connections use the same logic when a message is received. The classes *BluetoothConnection* and *NetworkConnection* both have as parent *Communication*. The most imporant function of these two classes is *sendMessage*, which sends a message to a peer. A WiFi connection will create a new *ClientTask* and a bluetooth connection will create a *ConnectThread*, which will both send the message to the peer.

4.2 Wifi

The connection is made by using the ServerSocket class. The implementation in TrustChain Android is done by a client-server style model. Although, since every device is a server can't really be seen as an exact client-server model. All outgoing messages, like sending a crawl request or half block, is done via client tasks. The server handles the incoming messages. It checks whether it has received a half block or a crawl request and calls handles the response by calling either `synchronizedReceivedHalfBlock` or `receivedCrawlRequest`.

If, from looking at the source code, it is not yet clear how the connection is made, please look into other Android server/client tutorials that can be found online.

The simplest way for connecting via IP, which does not have to deal with possible NAT puncturing or port forwarding is connecting to a local IP on the same WiFi network. A guaranteed method involves setting up a WiFi hotspot on one of the devices and letting the other peer connect to this hotspot. WiFi networks, which make use of IPv6 are not guaranteed to work.



My external ip is:

My local ip is:
192.168.43.1

Connect to IP

192.168.43.184

8080

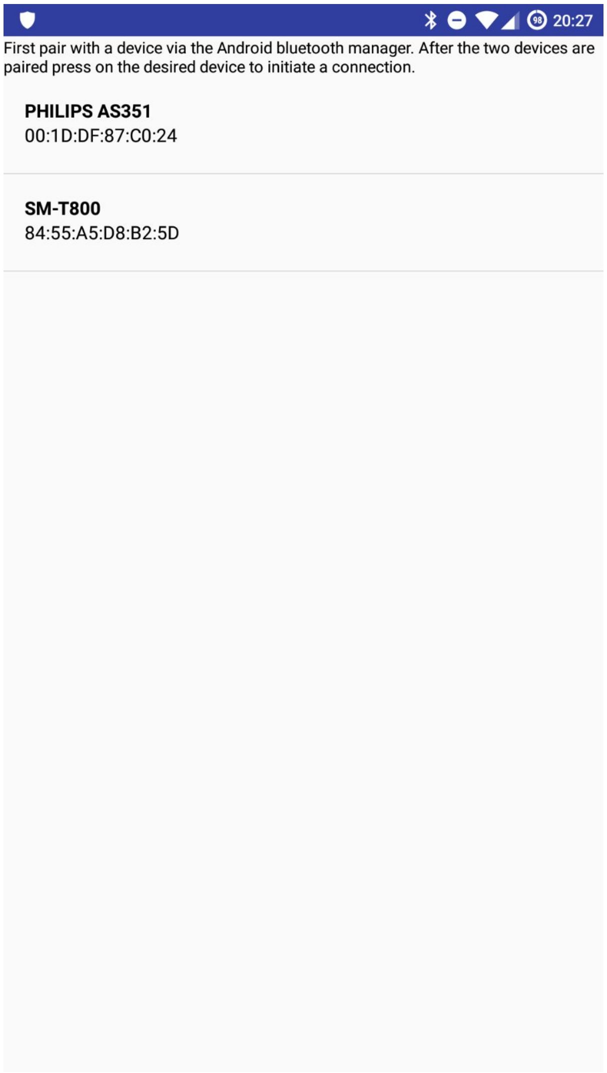
Example of connecting to a peer using on a device using a WiFi hotspot.

4.3 Bluetooth

Bluetooth works similar to WiFi: a server is created which is listening for messages, and a client send messages to server. However, with WiFi the messages are sent through two different ports and thus two different connections. This is not possible with Bluetoooth so the client will reuse the socket.

4.4 Links to code

- ClientTask implementation (ClientTask.java)
- Server implementation (Server.java)
- ConnectThread implementation (ConnectThread.java)
- AcceptThread implementation (AcceptThread.java)



First pair with a device via the Android bluetooth manager. After the two devices are paired press on the desired device to initiate a connection.

PHILIPS AS351
00:1D:DF:87:C0:24

SM-T800
84:55:A5:D8:B2:5D

CHAPTER 5

Message structure (Protocolbuffers)

Creating a network of TrustChain peers which only run a Java version of TrustChain is not very useful. Therefore the TrustChain blocks and messages should be compatible with many platforms, so cross-platform connection is possible. For the storage of the chain this is achieved by using SQLite, which has implementation for many platforms. For sending messages (blocks and crawlrequests) this compatibility can be achieved by using Google's Protocolbuffers, which is a cross-platform data serialization mechanism. Note that this Android implementation was not implemented for compatibility with other TrustChain implementation and thus will not be compatible out of the box.

5.1 Making changes

Protocolbuffers is used to create the structure of both a `TrustChainBlock` and `CrawlRequest`, both can be found in Message.proto With Protocolbuffers the corresponding Java classes can then be compiled. Making changes and recompiling the Java classes is quite easy, just follow the tutorial of ProtocolBuffers and you should be fine. When making changes, don't forget to also update the database structure.

5.2 Links to code

- Structure of message (Message.proto)

CHAPTER 6

Local chain storage (database)

Every half block correctly created gets saved locally on the device. Also all the incoming blocks of other peers, either as a response to a crawlrequest or in other ways, are saved locally when validated correctly. The blocks are saved using an SQLite database. Android has code in place to handle all the complicated parts, so using the database after setup consists mainly of writing queries. Please refer to the Android tutorials for an explanation on how to use SQLite databases in Android.

The database is set up in the same way as ipv8 python code. So the database from implementation of the python code can be imported. Notice that the columns correspond to the attributes of the Protocol Buffers object, so for inserting it simply needs to get the relevant data from the object. Note that it when receiving a raw message it always has to be passed to a Protocol Buffers object first, to ensure that data was received correctly.

6.1 Links to code

- Creation of database, inserting blocks (TrustChainDBHelper.java)
- Retrieving blocks (TrustChainBlock.java)
- IPv8 (database.py)

CHAPTER 7

Crypto

Trustchain uses the Curve25519 elliptic curve, which is normally used for key agreement, but in trustchain it is used for DSA. The security provider that is used is SpongyCastle, which has a great variety of implemented curves.

The class Key is a helper class, which can be used to perform crypographic functions. Most of the functions in this class are static functions, so that they can be called from every place. These functions can be used to easily load a public/private key, create a signature, create a new key pair, and verify a signature.

On initial start-up, the app creates a public private key pair which are then stored in the local storage of the device.

CHAPTER 8

Installation Instructions

If you want to quickly test the basic version of TrustChain Android follow *Installing TrustChainAndroid APK*, if you want to create a version of TrustChain Android with your own features follow *Setting up the Android Project*.

8.1 Installing TrustChainAndroid APK

In order to install an APK of Android TrustChain as found on GitHub. The only thing you need to do is download the APK to your phone and open it to install it. The current minimum version of Android you need is 17 (4.2), however not all features will be available to this version. The recommended version is 21 (5.0) or higher. To manually clear the database on lower versions, simply clear the data of the app in options.

8.2 Setting up the Android Project

Follow the steps below if you want to make alterations to the project and add your own functions. If you are already familiar with developing Android native apps and GitHub, this will be trivial.

- Download and install Android Studio
- Make yourself familiar with how Android projects are set up, by reading the Android Getting Started Guide
  - Note that the guide makes use of the Layout Editor, however writing the xml files directly will give you much better control
- Clone the repository to your work station `git clone https://github.com/wkmeijer/CS4160-trustchain-android.git`
- Import the project in Android Studio by `File>Open` and search for the cloned repository
- Start editing

Note that connecting to an emulator will often not work, so for proper testing you will need two phones.

CHAPTER 9

Contact and links

9.1 Contact

For students of the CS4160 Blockchain Engineering course of the TU Delft. You can contact us with questions about the code via e-mail.

- Wilko Meijer (w.k.meijer@student.tudelft.nl)
- Rico Tubbing (r.tubbing@student.tudelft.nl)

9.2 Useful links

- App source code on GitHub
- TrustChain source code on IPv8 GitHub
- Dispersy ReadTheDocs
- Tribler GitHub
- Blockchain Lab TU Delft
- Paper | TrustChain: A Sybil-resistant scalable blockchain