Proposal Demonstrator I

for Validicity

by Göran Krampe

Introduction

This document describes a design for an information system utilizing block chain technology to store immutable event data as part of a regular PostgreSQL database.

The goal is to create the basis of a system that uses block chains internally to give external parties the ability to independently and cryptographically fully verify the integrity of the data in the system. This property is known as being “tamper evident” which means that any tampered data should be detectable by all involved parties using the system.

Block chain systems can additionally have the stronger property “tamper proof” which means that the data is cryptographically protected against wrongful modification to begin with, but that is a much more complicated property to attain – it implies that the database must be distributed in full copies to all involved parties, and all changes to the data needs to be done in concensus by all parties. Such a system is in most situations impractical and very complex to build, operate and maintain.

We are instead aiming for a practical level of “tamper evident” combined with a flexible architecture:

- Using only “off the shelf” technology that is Open Source, no “lock in” with vendors such as Amazon
- Using a regular relational database backend making it easy to develop functionality and make integrations
- Implementing the block chain “aspects” ourselves, but using well proven and established libraries and algorithms for the cryptographical parts
- Keeping a subset of data entities structured in block chains, but not all data – only the data that really needs to be managed in such a way

The above also ensures ability to host at any cloud operator as well as run “on premise” as an internal system serving a larger organisation.

Block chains

Some subset of entities in the database are maintained in “chains of blocks”. This means each record in those tables is linked to the previous record, hashed and signed with a private key. Such tables can only grow with more records and the existing records can never be modified or deleted. Each record, since it’s signed, is also proven to have been created by the actor with the corresponding private key.

The database can have several such “block chain” tables. The first record in all such chain tables is anchored to a single first known hash, thus forming the single root of all chains in the system.

A third party that mirrors these chain tables into their own database can continuously verify that all new records are properly chained with the previous records, and thus maintains a copy of the immutable ledger.

In addition to the data stored in the actual blocks, selected parts of the rest of the database are “anchored” into the blocks using secure hashes. This way other parts of the database that are kept in regular tables or as JSON documents etc can be “captured” at specific points in time so that it can later be proven that they have not been manipulated afterwards.

Each chain table is also continuously logged using a Merkle tree, which is a binary tree of hashes. This Merkle tree enables the system to perform fast verifications without having to traverse the full chains. It also offers the ability to produce and hand out proofs (a small file) and verify such proofs.

For more explanations around how this works, QLDB has very good documentation:

https://docs.aws.amazon.com/qldb/latest/developerguide/verification.html

Technical Architecture

The following is my proposal for technical choices. I have used all of these components in multiple projects and systems are running in production.

- Ubuntu LTS Linux on server
- PostgreSQL as database
- Server side of system written in Dart using the Aqueduct framework
- Mobile application(s) written in Dart using Flutter
- Other client applications also written in Dart (like code running in RFID scanner device)
- Communication between clients and server is REST using OAuth2. This is again standard widely used and familiar technology.

Further I propose to use Upcloud as initial cloud vendor since I am very well versed in using them, but any other cloud vendor can also be used.

Also, the choice of PostgreSQL gives us maximum flexibility and a path to migrate later on to CockroachDB which is “on the wire compatible” with PostgreSQL but offers a much higher level of scalability world wide, if that turns out to be interesting.

Since the end result is meant to be a flexible system that is malleable and easily integrated with other systems and clients – the choice of a classic REST API using OAuth2 gives best options for clients.

Demonstrator I

The first demonstrator is strongly focused on the RFID scanning terminal device. After discussing this with Alex the following steps seems logical:

1. An example hardware setup combining an RFID reader with a Linux-based fanless small device (see components below) that can be connected with a USB cable to a stationary lab computer running Windows.
2. Ability to scan the bottle RFID tag using software in the device.
3. Ability to emulate keyboard strokes for the connected Windows computer, so that a reasonable value can be entered into the existing administrative system, as if it was typed on the keyboard.
4. Ability to generate and submit a block to the central server where it is verified and added to a block chain table.

Step 4 implies the existence of a server in the cloud running a first bare bone version of the database and the REST API, protected with OAuth2.

It also implies we have the basics in place for the block chain – key pair generation locally on the scanner device, registration of scanner device as a known actor in the server, initialization of the block chain and reception of the submitted block by verifying it against the public key and adding it to the chain.

The key pair for each actor should be created locally on the devices, for the scanner device this could mean at first startup. Creation of blocks is always done locally on the devices, in this case the scanner device. This means the device signs the block before sending it to the cloud. This way the private key never leaves the device.

I proposa that the cryptographical code for signing, key derivation and hashes etc is the same as the code being used in the Nano crypto currency which has a market cap today of $80.000.000 and to date 50 million blocks in the ledger and zero incidents from being in operation in 5 years. Nano is the fastest crypto currency to date. Using the same algorithms and code is a deliberate choice to ensure safety and reliability.

From another project I already have an RFID reader called the ACR122U from ACS (Advanced Card Systems), and a Beelink Z83II mini PC. This will work as a first setup – we can always change the mini PC to something else later on.

Timetable

The following is an estimate of time for the four steps in Demonstrator I:

1. **8 hours**. Investigating hardware, installing Linux and configuring RFID reader etc.
2. **8 hours**. Given that I have a bottle with an RFID tag I can use as an example, being able to read it using software should be doable in one day.
3. **16 hours**. This involves getting the scanner device to act as a so called HID Keyboard which can be done using “USB gadget mode” which the Linux kernel has support for. It can be done using both some of the Raspberry machines as well as a Beaglebone Black. Unsure of the Beelink Z83II, but two days is hopefully enough.
4. **32 hours**. This involves creating the server with database, server software, API etc, as well as writing the client and server code including the cryptographic parts. Quite a lot of software, but most of this is stuff I know very well that I can do quickly.

In summary, if we disregard time to demonstrate and other meetings etc, 8 full days should be enough to reach something that covers Demonstrator I.

Demonstrator II

This is a suggestion for Demonstrator II, only included as “food for thought”. The idea is to add a mobile application to the mix, including RFID scanning:

1. Ability to install mobile application from Playstore
2. Start mobile application and login
3. Register mobile application as an actor in the server
4. Scan RFID of bottle and create a timestamped “event” in the system by entering a comment
5. View event log of the scanned bottle

Another user can do the same and view the event log. Note that login can be simplistic at this point.

Other additional parts one could elect to include are:

- Integration of Chainpoint with anchoring against the Bitcoin chain.
- Ability to independently partly mirror the database chains and continuously verify their integrity. This would have to be an open source server software easily installed and operated.
- Ability to independently verify Chainpoint proofs using an open source client software. This would have to be an open source server software easily installed and operated.