Blockchain in IoT with HyperLedger Fabric Framework

A report on project completed in partial fulfilment of the requirement of the degree of Master of Science

Alexander Vieth

Elin Dangol Maharjan

Sepideh Askarimarnani

Viet-Hoa Nguyen

Project Supervisors:

Prof. Dr. Martin Kappes

Johannes Bouche

Faculty of Computer Science and Engineering

Frankfurt University of Applied Science

Nibelungenplatz-1, 60318 Frankfurt am Main

Date: 31.03.2020

Abstract

The blockchain emerges as an advanced technology that has the potential broken out of its initial use in the advancement of cryptocurrency and bitcoin to non-financial purposes with its permissioned model, which can transform a lot of businesses. With blockchain still under constant development, it is important to start experimenting with different implementations before actually adopting it in the enterprise. The objective of this project is to develop a prototype using Hyperledger Fabric for quality management in the context of semiconductor manufacturing. This prototype acts as a proof of concept to evaluate the use of blockchain in realistic scenarios. The results show that the blockchain can be used as a decentralized database that provides additional layers of security and provides the auditing process with more transparency.

Keywords: blockchain; Hyperledger Fabric; sensor data integrity; quality management.

Acknowledgements

Working in this project was very interesting. During the developing process, we have learned a lot about blockchain technology, especially about Hyperledger Fabric. We would like to thank Prof.Dr Kappes for the chance to be involved in this project. We especially want to thank Johannes Bouche and Lukas Atkinson who guided us for the completion of the project and explained critical aspect of topics related to the project.

Contents

List of Figures

List of Tables

Introduction

Blockchain is the newest contribution to the technology after the internet of things, social media and cloud computing. Blockchain is a distributed ledger or data structure for all transactions which are taken place between different parties in a network. The main goal of this technology is to implement a decentralized platform that doesn't require the engagement of an authorized third party for validating transactions. Every member in network hold the same copy of the ledger. Since it is founded on a system of distributed consensus, thus guarantees integrity and security [5].

On the other hand, Internet of Things (IoT) is a fast-growing paradigm transforming our lives in terms of working, traveling, entertaining ourselves or interacting each other, and our life is changing in many ways. IoT requires building a network of interconnected daily objects collecting data from the environment and automating specific activities. IoT devices are known to have remarkable vulnerabilities to cyber-attacks. In fact, the shortage of intrinsic security mechanisms renders IoT prone to privacy concerns and security risks [6].

A significant majority of IoT platforms are centralized model based, so a central failure point is a risk that needs to be tackled. Information confidentiality and authentication is a sever threat in IoT, therefore data security is a core requirement. Data integrity is another concern in IoT. One of the main implementations of IoT relates to decision support systems in which timely decision are taken based on the collected data from several sensors. Hence, system should be protected by preventing the injection of wrong data which impact on decision making. Another challenge in IoT is establishing trust between entities without an authorized third-party to mitigate the issue of non-repudiation [6].

Furthermore, today there are growing number of use cases for data provenance systems in IoT. In summary, such systems offer information on different phases of data generation and modifications, who began and when and how they were created. It is crucial to store this information in a transparent and replicable manner in order to be trusted. Vaccine Supply Chains is an example of data provenance system in which temperature and locations of the vaccines are tracked and controlled. Therefore, any errors during the cold chain is identified and in this way it helps to develop and preserve trust in immunization application, by avoiding falsified vaccines from reaching the supply chain [7].

Blockchain (BC) can help address all these issues. Blockchain technology enables making decentralized IoT systems providing secured and trusted exchange of data along storing data in a shared ledger between all entities [8]. Many of IoT's architectural deficiencies can be alleviated by key characteristic of the blockchain technology including immutability, transparency, auditability, data encryption and operational resilience. Blockchain offers authenticity, non-repudiation, and integrity and facilitates authorization and automation for transactions by smart contracts [6].

Currently many researches are conducting in integration of Blockchain in IoT and there are many challenges to address specific demands in this area and developing a prototype using a blockchain technology in IoT, was a great chance for us to understand these topics to some extent and develop our practical knowledge regarding security aspect of the processing data in a blockchain network.

The rest of this report is organized as follows. In section 2, full description of the project and its scope is presented. Section 3 covers the main characteristics of the blockchain and Hyperledger Fabric framework. The following section 4 is the core of design and implementation of the project. User interfaces are presented in section 5. In section 6, project timeline and team organization are provided. We discuss project issues in section 7 and, finally, Section 8 concludes our work.

Project Description

Since the scientific field of the blockchain technology and its extensions, like smart contracts, is both vast and relatively unexplored, our team had to maneuver in a predefined project harness, which was provided to us by the project organizers. In the following chapters, we will describe the predefined scenario, the general goal of the project, as well as go into detail about the specific scenario that we've created, which fits into the given project harness. As a general note about the project - the following requirements/project harness was mandated to us by the project organizers. This obviously limited the amount of directions that we could have taken the project in. We will later see, that this resulted in some interesting inherent problems, when it comes to applying the blockchain technology to scenarios which are similar to our project.

2.1 Project Overview

The previously mentioned project harness consists of a list of requirements, which our project had to fulfill Among those are the technology to be used and a general scenario outline. The requirements can be summarized to these points:

- have a scenario with 2 or more business partners
- use sensor data (simulated or data from an actual sensor)
- use the hyperledger fabric framework
- realize some security benefit by using smart contracts
- have a platform to interact with the blockchain network

Given this harness, only a limited amount of meaningful and distinguishable scenarios are possible. Meaningful and distinguishable scenarios in this case are scenarios that can still be distinguished even if we change the names and/or types of the sensor data or amount of sensors, or peers. For example, a project which implements a smart contract that performs a boundary check on temperature data of a product in a manufacturing process is essentially the same as a project which implements a smart contract that performs a boundary check on radar data of multiple radar stations. While the amount of peers and the types of data are different, both projects boil down to a boundary check and lack any meaningful differences in terms of security benefits.

2.2 Purpose

With that context in mind, the goal of the project was to realize a solution to a scenario which fulfills the above mentioned requirements and increases the security when handling the sensor data, or increases the confidence in the data's integrity. Essentially any angle that improves the scenario in terms of it-security, while utilizing the required technologies. The secondary purpose is gaining experience in handling new topics of computer science in a scientific way, in a group. In later chapters we will go into further detail regarding the project and time management, as well as planning. Furthermore, the results of this project might help the IT-Security research lab where there are currently working on a blockchain related research project, in assessing possible directions/angles from which to tackle additional aspects of using blockchain for sensor/IoT data.

2.3 Scope

While the requirements for this project technically allow for the development of a fully fletched business application, the timeframe for the project and the general lack of experience of our team in the field of blockchain, meant, that we had to restrict the scope of the project to a proof-of-concept level application. This means we've defined certain assumptions about the project setup and the implementation of our ideas. For example, some of the assumptions are:

- key exchange for sensor data encryption happened over a secure channel
- the sensors and ledger run on different machines
- the web interface for the ledger and sensors follows security best practices

These points would be critical for a fully developed application but are not relevant regarding the topic of this project, which is handling sensor data with blockchain and smart contracts. In the next few chapters, we will go into detail on the specific scenario and setup that we've used. There we will mention and explain all the assumptions we took specifically.

2.4 Scenario

Let's assume that there exists a company BP1 which manufactures silicon wafer plates and delivers them to a different manufacturing plant, which is owned by another company BP2. Wafers are thin silicone plates on which microchips are produced [9]. A single wafer can contain hundreds of microchips, which are very sensitive to humidity and temperature. Let's further assume that the wafer plates are packaged in sealed containers, which contain a temperature and relative humidity sensor. The sensors are connected to a raspberry pi which has access to the internet. We assume that the raspberry pi has the minimum amount of attack surface, which means there are no additional services/programs running on the pi that could be used to compromise the device. We also assume that the pi has been configured with the best practices in mind, in the context of IT-security. This can include - but is not limited to: using a stable release of a Linux operating system distribution, with up-to-date security updates, proper SSH setup [10] and HTTPS capabilities. Both business partners BP1 and BP2 have an invested interest in knowing the state of the wafer plates. The state in our case is defined as a combination of temperature and relative humidity at a given point in time. The raspberry pi reads the sensor values every few seconds and packages them into a data bundle. The bundle contains a list of sensor values with a timestamp, as well as a bundle ID and the name of the sensor. If the bundle reaches a certain amount of sensor data entries, the raspberry pi encrypts the data and sends it to a webserver, which in turn is connected to the blockchain network.

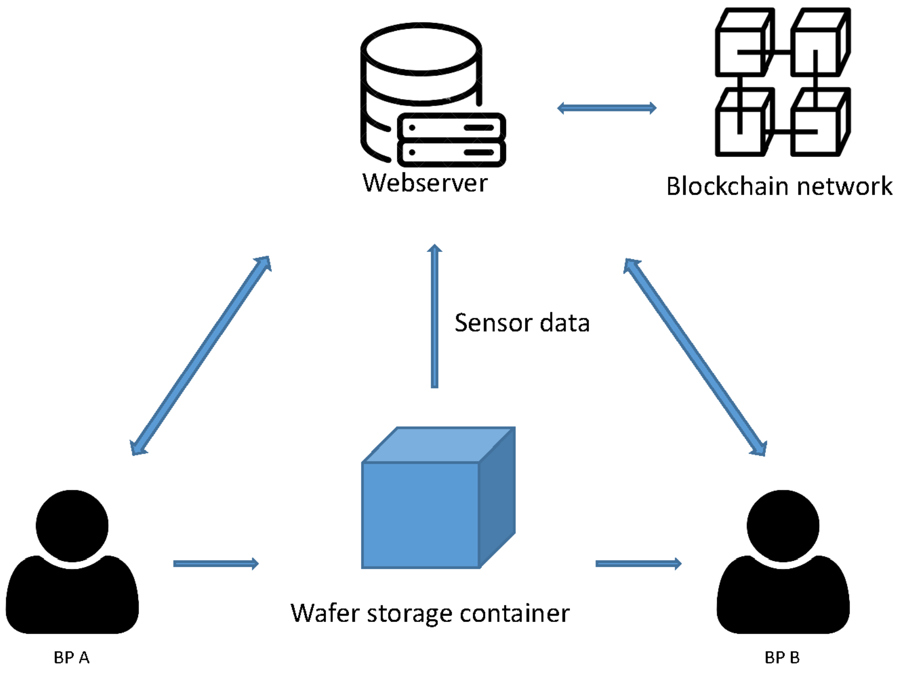


Figure 2.1: Project setup overview

There the data is decrypted and written into the blockchain. If the decryption fails, the data is ignored. We have defined a required format for the data, as well as the need for a valid username, which has to be attached to the packaged sensor data. We assume that the communication from the raspberry pi to the webserver is encrypted using HTTPS. We further assume that the webserver was setup with the best practices for secure web services [11]. This should include features like input validation, escaping inputs, sql injection prevention and CSRF protection. While some features like CSRF protection are not necessary when we have one-directional communication from the raspberry pi to the webserver, it becomes relevant when the same webserver handles the web interface which allows an administrator to register new sensors, and business partners to sign of on sensor data. While this is the scenario that we ended up with, it is useful to also take a look at some of our previous candidates and see how we arrived at our solution. We have considered implementing the following features:

- performing anomaly detection on sensor data
- marking blockchain entries with a trustworthiness score
- using a real raspberry pi and sensors

Anomaly detection would allow us to detect outliers, or abnormal behavior of the sensor. This could be a sign of a broken device, or the attempt to forge fake sensor data, perhaps to cover up manufacturing or transport mistakes. Detecting these situations might increase the confidence in the data for both business partners. For the sake of clarity, we will discuss the caveats, that come with these proposed features, in the next chapter. If we where to perform a qualitative check on the sensor data, we could then determine a level of suspiciousness regarding the data. The idea was to append that level as a numeric value to the blockchain entry. The business partners could then quickly take notice of a suspicious situation. Another idea was to do create a realistic representation of the scenario, using real sensor data and real raspberry pi's. This degree of realism would allow us to check for certain, how applicable the solution really would be. While not technically a feature, it would increase the complexity of the project, which was one of the characteristics that we have used, to assess if we should implement a feature or not. We've instead gone for the simulation route, in which we use a real world set of temperature and humidity data from the great UCI Machine Learning Repository[12]. The website provides a vast pool of datasets in a broad span of fields. We've decided to use the occupancy detection dataset [13], which contains data to detect if a room is occupied by a human or not. Luckily it also provides real sensor values for the fluctuating relative humidity and temperature (among other values). Since we assume that these values would not drastically differ, compared to when we would have used our own raspberry pi with real sensors, we've have decided to write a small simulator program in python. This also had the advantage, that each of our team members could access/read the sensor data for testing at any point, without having access to special equipment - which in this case would be the raspberry pi with the 2 sensors. We have written the simulator, to read the dataset in predefined intervals and parse out the temperature and humidity values. We then assigned our own timestamp, along with additional information, which we will cover in later chapters. The simulator then packages the data to a JSON, which is then sent via HTTP to our webserver for further handling.

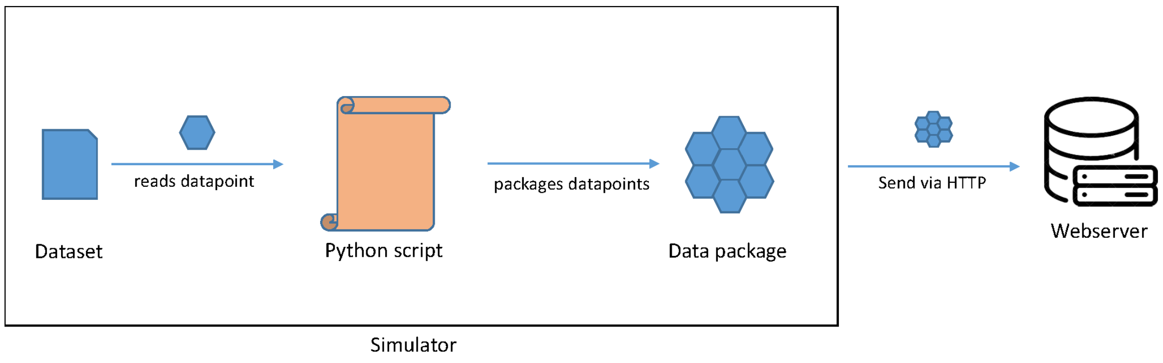


Figure 2.2: Sensor simulator setup

In a real world setting, there would be a real sensor and we would add another layer of security by using HTTPS. We have simplified the scenario to compensate for lack of time and the relevancy of these features in the context of the project goal, which is to use the hyperledger fabric blockchain and smartcontract technology.

2.4.1 Problem analysis

To understand why we eventually decided against these features, we need to look back at the goal of the project, which was to use hyperledger fabrics blockchain framework and it's support for smart contracts to improve the integrity and/or security of sensor data coming from IoT devices.

Looking at the above defined scenario, it is clear that no matter the output of the sensor, that we want to store the data in the blockchain. Under (almost) no circumstance, do we want to reject sensor data and therefore prevent it from being stored in the blockchain. The simple reason being, that both business partners want to have an uncompromised and unfiltered view of the current state of the wafer transportation container. If a sensor is damaged and starts recording unrealistic values, the business partners still have an invested interest in knowing that information.

Likewise, in a scenario where we have potentially forged, or otherwise anomalous data - even if we can categorize the data with a decent amount of precision, we still want both business partners to have that information, as opposed to rejecting the data point(s). Let's assume that we take the second approach, which is to still perform an anomaly detection process on the sensor data, but instead of rejecting it, we mark the data either with a binary value or levels of suspiciousness and then create a new blockchain entry. Even in a perfect scenario, where the categorization process is always accurate and both business partners could immediately see how trustworthy the data is, did we not actually used the features of the blockchain technology or the smart contract in any way. The same functionality can be realized without blockchain and smart contracts

by simply running the same anomaly detection on the sensor data somewhere else as the smart contract. There exists a bit of nuance in that scenario though, which we will show a little later. Having the before mentioned perfect anomaly detection being inside the smart contract, would guarantee that the detection process is always triggered for every transaction, which would be a desirable scenario.

An underlying problem arises though, if we take a more realistic view of anomaly detection systems. The previous thought experiment requires the assumption, that the sensor data has some characteristic that can be categorized as anomalous, and can be detected by running the data - or a set of data - through some code. With simple boundary checks, it is clear that this is the case. Define a valid boundary and then validate the given data against that. Even deviating from established patterns will work. Let's assume a scenario, in which the sensor is exposed to daylight. The recorded temperature would change periodically over the span of serval days, and by the nature of relative humidity, the relative humidity would also change in the same periodical fashion. Should the recorded data deviate too much from that pattern, we can categorize it as an anomaly. In regards to sensor data forgery, which is arguably the more sensitive topic, the answer is not so obvious. When planning the project and our approach to it, we've found that there exists a difficult problem of detecting forged sensor data.

A great real world example for a similar problem can be found in the 2010 Stuxnet worm [14]. The goal of the worm was to infect as many computers in the country of Iran, and more specifically the town if Natanz as well as its neighboring areas. The target was to infiltrate the uranium enrichment facility, located in Natanz via a portable storage device - such as an USB stick.

Once inside the facility, the worm destroyed the enrichment centrifuges by manipulating its rotary speeds, while simultaneously displaying acceptable sensor data in the control room. Because the Iranians trusted the source of the sensor data, there was no way to detect the forgery. The sensor values of course seemed real. This leads us to our scenario, where a similar attack could be performed. An attacker could torch the wafer storage container, while forging acceptable temperature and humidity values. If these sensor values are all the business partners have, to perceive the current state of the storage container, they will suffer the same results as the Iranians. The comparison to Stuxnet is not technically correct though, since Stuxnet had compromised the centrifuges control elements.

The correct analogy to our project would be an attack, in which the raspberry pi was compromised. In that case it would be game over anyways. The attack vector still exists though, if we don't check the authenticity of the sensors. This need for verification is one of the main points we want to propose in this project.

Our system would otherwise also be vulnerable to a man-in-the-middle attack[15], in which the attacker intercepts the real sensors data, changes the values and forwards his own message to our server.

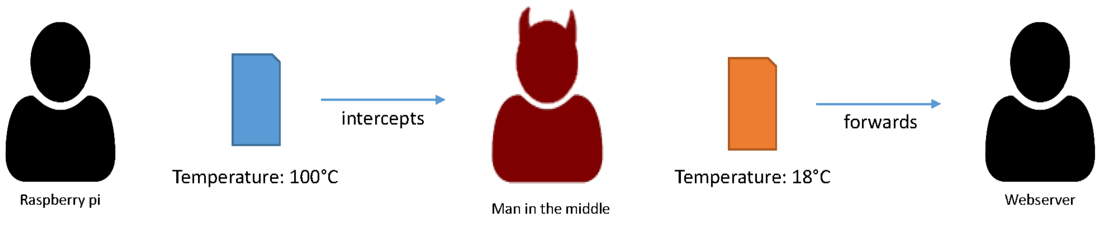


Figure 2.3: Man in the middle attack scenario

An approach to add security to our scenario is to use encryption, to secure the sensor values, as well as verify the sensors authenticity in a similar fashion as Nakamoto described for his Bitcoin crypto currency [16]. More on that later. Regarding the realistic scenario setup, in which we would setup and use a real raspberry pi and real sensors - we've found that this extra degree of realism doesn't actually add anything meaningful to the project.
We've found that we can use a preexisting dataset for humidity and temperature values to create a simulator for our sensor, which was more flexible and easier to use in our distributed working environment.

2.4.2 Sensor data trust

Given that context, we've deduced that by the nature of the given scenario, every solution that bases its accepting/rejecting decision on analysis of the sensor data, will be inherently flawed. We came to that conclusion, because in every case the business partners want to see and perhaps process the data coming from the sensor, even if we could for certain categorize ïnvalid̈data, regardless on how you would define ïnvalid̈. Anomaly detection system will produce false positives and false negatives. Given a margin of error for such a system, the benefits can evaporate quickly. That being the case - it results in us never rejecting data, which is one of the aspects that smart contract technology provides us with. There could be an argument made, that besides the inherit benefit of having a decentralized immutable storage of information, the features of the hyperledger fabric framework do not offer meaningful security additions to a scenario as was described in the project harness. Hyperledger does provide user/peer management, which helped us in setting up the permissioned blockchain network. Based on this information, we've decided that using the smart contract to perform authorization and data format checks, is a meaningful and beneficial usage of hyperledger frameworks smart contract technology. This means that we can, to a reasonable degree (depending on the encryption used), say, that no entry in the blockchain was read by a unauthorized third party.

2.4.3 Unusable benefits of Blockchain and Smartcontracts

While analyzing our approach to the subject, we've found that core beneficial concepts of other blockchain solutions are not applicable for logging sensor/IoT data. Lets take Bitcoin [16] as an example. Whoever Nakamoto might be, he utilized the validation aspect that comes with a decentralized immutable record of transactions, that are signed and verified using public/private keys. Since Bitcoin is a crypto currency, it deals with financial transactions between 2 peers in a network with an arbitrary amount of peers. Where there is money, there is fraud and theft. The blockchain and its distributed ledger, offer a clear benefit by preventing fake transactions. Let's assume Jay and Jungho are peers. Jay wants to transfer 10 EUR to the wallet of Jungho, so he creates the transaction and signs it with his private key. Jungho can then verify the validity of the transaction using Jays public key.

Should a forged transaction over 1000 EUR be send to Jungho, the decryption/ver-

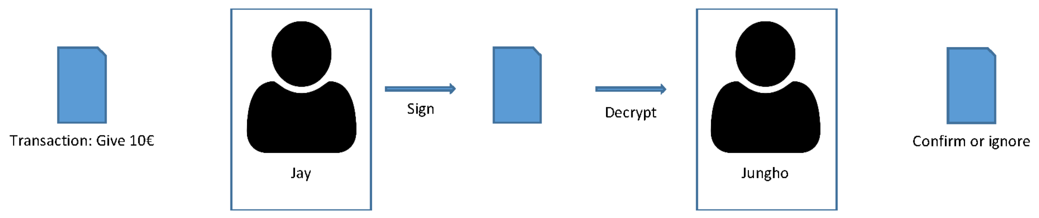


Figure 2.4: Basic transactions with bitcoin

ification would fail, unless Jay's private key was compromised. Therefor every transaction can be locally verified and only authorized transactions can make it into the blockchain. The chain itself can be used to reconstruct and verify transactions. Making the contents of the blockchain meaningful to future transactions and therefor have an impact on new blocks. With this real world example in mind, lets again take a look at our scenario. The values inside the blockchain don't influence how we want to handle the next data point, since we are only storing sensor data. A temperature value has no intrinsic meaning to the next temperature value, as a transaction over x EUR from A to B would, to a transaction over y EUR from B to C. Regarding the verification of transactions, using encryption and decryption - this concept is beneficial, even in our scenario. As long as we assure the safety of the private keys, we can claim with confidence, that only valid transactions and therefor sensor data from a trusted source makes it into our blockchain. As we will in later chapters see, we've isolated this aspect as one of the strongest arguments for using blockchain for storing sensor data.

Blockchain

3.1 Introduction

Blockchain is one of the latest innovations in our world today and many developments and works have just started on this technology. There are various blockchain-oriented applications, including financial services, reputation systems and the Internet of Things (IoT). Blockchain is a decentralized ledger or data structure for all transactions which are taken place between different parties in a network. It can be considered as chain of blocks where each block points to the block previous of it. After entering the data of transactions or events into the Blockchain, it is impracticable to modify the information that are accessed by members of the network. Transactions are verified in the Blockchain only after it is approved by the majority of the users involved in this process [1] [17].

3.2 Blockchain Architecture

As described before, sequence of blocks form a blockchain, and each block contains records of transactions and transaction counter in its block body and also a block header in which the calculated hash of the parent block and other details are stored. The first block is called the genesis block [1]. This is demonstrated in Figure 3.1.

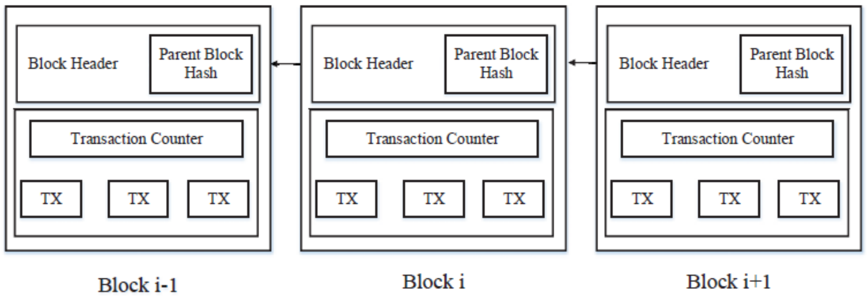


Figure 3.1: Blockchain as a continuous sequence of blocks [1]

Key Characteristics of Blockchain

There are several key characteristics of the blockchain summarized as bellow.

1. ***Decentralization***: In contrast to the centralized transaction systems in which each transaction should be validated by a trusted agency, in blockchain the centralized authority is not needed anymore [1].

2. ***Persistency***: Deleting or rollback transactions is almost impossible as they're used in the blockchain [1].

3. ***Anonymity***: Every participant owns an individual address to communicate within the blockchain, which doesn't disclose the user's true identity [1].

4. ***Auditability***: It is simple to verify and trace transactions in a blockchain [1].

5. ***Transparency***: The collected data in blockchain is transparent to all authorized members which results in preventing from modification and tampering [2].

3.3 Working Principle

The first step in using blockchain is to construct a peer-to-peer network between all interested nodes. Each node is assigned a Public/Private key pair for encryption and decryption of the exchanged messages respectively. Once a transaction is generated by a node, it get endorsed by private key of the node and is transmitted to other peers and consequently it is decrypted in receiver nodes using sender public key which is known to the network. In this way the authentication of sender is assured as well as the integrity of the messages, since if data is transferred incorrectly, decryption at the recipient peer is not possible anymore. In this way the messages are validated by each peer and transmitted, then miner nodes order and pack them into a timestamped block [4].

Next, these block are disseminated back to the network and nodes check the validity of transactions in the block and also by applying the corresponding hash function, verify that new block points out to its preceding block in the chain. The block will be ignored if theses conditions are not met. If two conditions are successfully satisfied, the nodes must append the block to the blockchain and update transactions [4]. The diagram in Figure 3.2 presents the workflow of the blockchain.

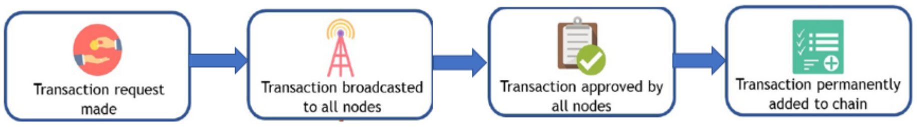
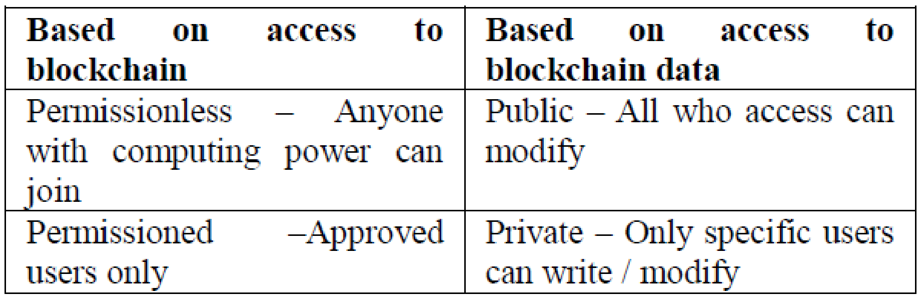

Transaction request made
Transaction broadcasted to all nodes
Transaction approved by all nodes
Transaction permanently added to chain

Figure 3.2: A common blockchain process flow [2]

3.4 Types

Based on the managed data, the accessibility of these data and the actions that user able to perform, there are various types of blockchains. Therefore, public and private blockchains, and permissioned and permissionless blockchains can be separated. In table below four type of Blockchains are categorized [4] [18].



Based on access to blockchain	**Based on access to blockchain data**
Permissionless – Anyone with computing power can join	Public – All who access can modify
Permissioned –Approved users only	Private – Only specific users can write / modify

Table 3.1: Blockchain Types [4]

It is however noted that public and permissionless terms are used interchangeably, and thus private and permissioned.

3.4.1 Public /Permissionless

Public blockchains are open for all. Normally, participants are encouraged to join. Everyone is able to enter without the approval of third-parties. Anyone can participate anonymously in posting transactions and may be involved in the mining and consensus phase of adding a new block of records to blockchain. Bitcoin is the most famous public blockchain [4] [18] [19].

3.4.2 Private

In a private blockchain access to network is restricted. Invitations are necessary to participate in a private blockchain. Private blockchains generally exist in a permissioned network to further constrain network membership [19].

3.4.3 Permissioned

Most private blockchains often control which users can execute transactions, smart contracts or serve as network miners. In permissioned blockchains participants are known and identified. These blockchains are typically designed for their particular business needs by organizations can provide interfaces through company's current applications [4].

Generally such blockchains offer a means of securing interactions between a group of individuals that have a shared goal but do not completely trust each other, such as businesses exchanging money, products or information [3]. Hyperledger-Fabric or Ripple are examples of private permissioned blockchains. It is also notable that not all private blockchains are necessarily permissioned. For example, a company may implement a private blockchain based on Ethereum which is not permissioned [4].

3.5 Hyperledger Fabric

3.5.1 Overview

Hyperledger Fabric, is an open-source blockchain platform optimized for use in enterprise environments [20]. It is among Hyperledger's projects based on the Linux platform. Example of use cases could be in fields of food safety, contract management, industrial logistics and digital currency settlement [3].

Fabric presents a new blockchain architecture that strives to be more resilient, flexible, salable and confidential. It has been designed as a modular architecture and implemented as a permissioned blockchain facilitating privacy and confidentiality of transactions and the smart contracts [20][3]. It is the first distributed operating system for permissioned blockchains in which all participants are identified in the whole network. Furthermore, distributed application developed in standard programming languages (e.g., Go, Java, Node.js) can be executed on many nodes in the Fabric network [3].

Fabric's architecture demonstrates a modern ***execution-order-validation*** model providing the execution of untrusted code in an untrustworthy distributed environment. It stores the history of executed transactions in the ledger database. Transaction flow is divided to three phases: 1) execution and endorsement by several peers 2) ordering 3) validation which will be discussed with more details in section Transaction Flow 3.5.5 [3] There are two core components in any application developed in Fabric framework:

1. ***Smart Contract***, also called chaincode is an executable program code containing the logic of the application and runs in the execution phase [3].

2. ***Endorsement policy***, in validation phase, functions as a static library for validation of the transaction in Fabric framework. Permission for defining the endorsement policies only is granted to Administrator. In a typical policy, a set of peers who should sign a transaction are defined in a logical expression

like "$(A \wedge B) \vee C$". In simple word, it specifies the ***endorsers*** which should digitally sign a transaction before being accepted by other members into their ledger instance [3].

Hyperledger Fabric even provides the possibility to establish ***channels*** to construct a specific transaction ledger for a community of participants [20].

3.5.2 Fabric Components

Fabric Network

A Set of nodes construct a network in the fabric bloackchain. Each node hold an identity granted by ***membership service provider (MSP)*** . There are three types of nodes in a Fabric network:

1. ***Clients*** through an application submit transaction proposal in execution phase, support organizing the execution process and transmit transaction to get ordered [3]

2. ***Peers*** are the core building block of a network hosting smart contracts and ledger. Client application should connect to peers in order to access the ledger [20]. Execution and validation of the proposed transactions are carried out by peers. A copy of the blockchain ledger is kept by all peers. However, only a number of peers called endorsers are allowed to participate in execution of the transaction proposal [3].

3. ***Orderers*** are the nodes which participate in the process of ordering the transactions, and a group of them form Ordering Service Nodes (OSN). These nodes do not take part in the validation and execution of the transactions [3].

Ledger

In Hyperledger Fabric framework a ledger is a core concept. Information or so-called Facts related to the business objects are stored in the ledger. Two related parts in ledger are ***word state*** and ***blockchain***. ***Word state*** is a database which stores the current values of the objects in form of a key-value pairs. ***Blockchain*** has an append-only structure which holds the log of all transactions resulted in the current state of the object. Transactions are stored in a block and are added to the blockchain. This history is immutable and append-only, but the current state can be modified [20].

Fabric CA

Every active node in the blockchain is assigned a verifiable digital ***identity*** in the form of a X.509 digital certificate. These certificates are cryptographic credentials employed for specifying the access permission to resources and information in a blockachin [3].

Fabric framework has a built–in ***Certificate Authority*** known as Fabric CA responsible for issuing and managing digital certificates through generating a public and private key. It functions in two modes: offline and online. In offline mode, credentials are generated for all nodes. Peer and orderer nodes are registered only in this mode. In online mode, there is the possibility to create cryptographic credentials for client nodes [3].

Membership Service Provider (MSP)

It stores the identity of all active members (clients, peers and orders) in the network. In other word, it has a list of permissioned network participants. As Fabric is a permissioned private blockchain, therefore, all nodes communicate with each other through authenticated messages by means of digital certificate. Membership service is a process in which the identities are authenticated and authorized by other part of the network. It is implemented based on ***Public Key Infrastructure (PKI)*** mechanism [3]. MSP assigns a ***role*** to each identity and defines privilege for each role. Every user should be assigned a role, for example admin, client, peer, orderer. Considering the scope, MSP are categorized into two domains: ***local*** and ***channel***. In ***local*** mode, MSP is implemented on an active node and in ***channel*** mode it is configured for a channel. A local MSP folder holds certificates of the root CA and admin as well as private and public keys of the node [20].

Ordering Service

The main function of the *ordering service* is to organize batches of submitted transactions in a well-defined order and bundle them into blocks. Those blocks will be a part blockchain [20]. More details is explained in section 3.5.5

3.5.3 Smart Contract

In real world, when companies wants to interact with each other, there should be a set of contracts including the rules, common terms and conditions, data and defined process details. In this way, all communications between organizations are controlled through this *business model.* In blockchain technology, these contracts are transformed into executable programs, identified as *smart contract* in the industry [20].

In other world, all transaction logic that regulates the life cycle of objects in a blockchain, are specified in the *smart contract.* In Fabric framework, a group of smart contracts are bundled into ***chaincodes*** deployed to the network. *Smart contract* has access to world state and blockchain in ledger to update states and query the history of transactions respectively. It runs on a peer node in the network and invoked by a set of input parameters named as *transaction proposal* [20].

3.5.4 Architecture

As mentioned before, Fabric architecture follows the 'execute-order-validate' paradigm. This is shown in Figure 3.3. The whole transaction flow is discussed in the following section .

3.5.5 Transaction Flow

In this section, different steps of the transaction flow in Fabric is summarized and. These steps are shown in Figure 3.4

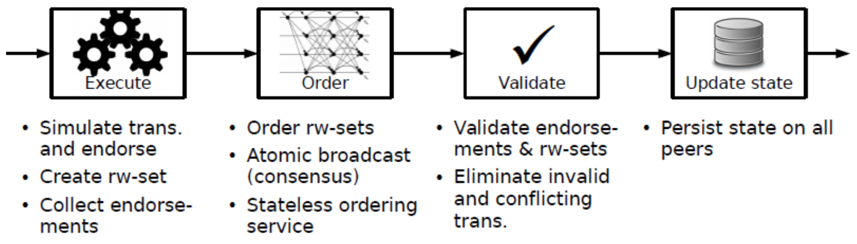

Execute
Order
Validate
Update state
• Simulate trans. and endorse
• Create rw-set
• Collect endorse-ments
• Order rw-sets
• Atomic broadcast (consensus)
• Stateless ordering service
• Validate endorse-ments & rw-sets
• Eliminate invalid and conflicting trans.
• Persist state on all peers

Figure 3.3: Execute-order-validate architecture of Fabric (rwset means a readset and writeset as explained [3]

Execution Phase

In this phase, client applications communicate with endorsing peers. First, based on the endorsement policy, a transaction proposal sent by application to required endorsers. Next, using this proposed transaction, every endorsing member separately executes the chaincode and a generates ***transnational proposal response***. The ledger is not updated in this step. Finally, endorsed proposal responses are collected and sent back to the client application. When a proposal response is endorsed, the digital signature of the endorser is added then using its private key, the whole payload is encrypted and signed. So, ***Authenticity*** and ***Non-repudiation*** is achieved [20].

Ordering Phase

In the second phase, endorsed transaction responses are submitted to the Ordering Service Nodes where the blocks of sequenced transactions are generated and prepared for distribution among peers. The essential point is that the ordering service positions transactions in a specific order, which is used by the peers during transactions validation and commitments [20].

Validation and Commit Phase

Phase three starts with distribution a copy of new blocks by orderer to all peer nodes connected to it in the same channel. This block is processed by every peer separately but in the same manner as other channel peers do. So, resulted in consistency of the ledger. Each transaction in the new block is examined by peers for checking the necessary endorsements based on the predefined policy in the chain code. This validation process ensures that all related organizations have achieved an identical outcome. After each single transaction has been validated successfully, it is committed to the peer ledger. Invalid transactions are kept for auditing purpose and marked with an indicator and do not update ledger [20].

Certain events are published during these processes including transaction events, block and chaincode events. So,client application can be subscribed to these event in order to get notified [20].

Consensus

The whole cycle of transaction workflow is termed ***consensus*** since all peers have come to an agreement on the sequence and content of transactions in a cycle facilitated by the orderers [20].

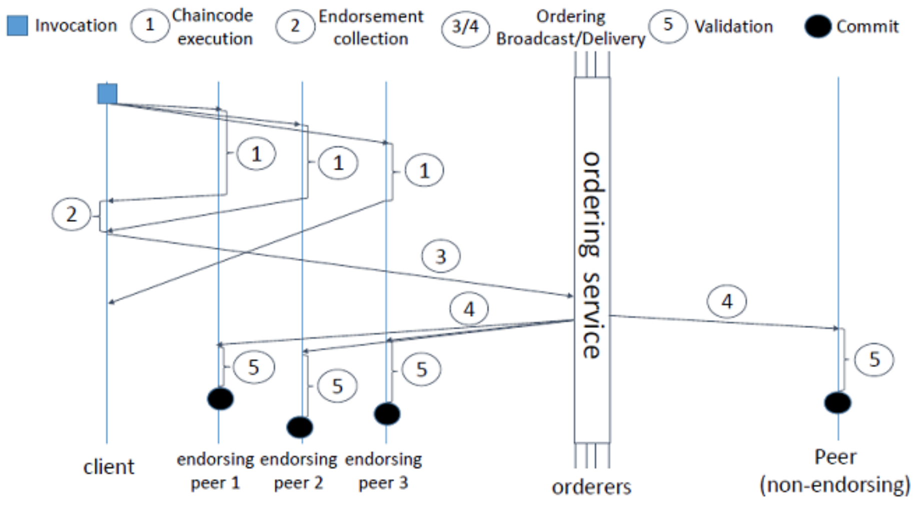


Figure 3.4: Fabric high level transaction flow [3]

Design

The design was basically done by generalizing a solution for a manufacturing process and supply chain scenario. Further, different parts of the solution were customized in relation to the project topic.

The following diagram shows in detail the process of data collection during the entire manufacturing process.

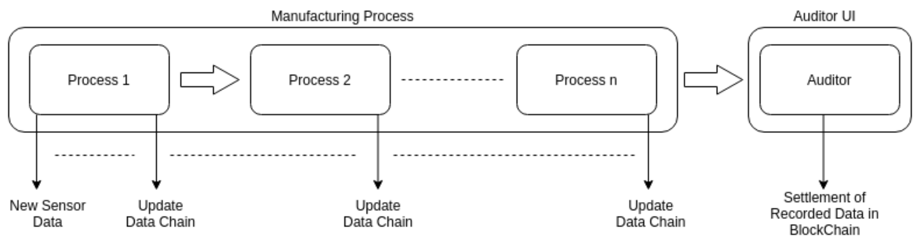


Figure 4.1: Sensor data push to the Blockchain during the manufacturing process

When a new manufacturing process cycle is started, a new transaction is raised with a unique batch number representing that particular manufacturing batch. The data are continuously collected throughout the manufacturing process. At some intervals, the collected data are packed into a data block and are pushed to the server under the initially created batch number. With this, we create a chain of data of particular batch under a single chain identified by their batch number. At the end when the manufacturing process ends, the auditors audits the collected data and provides their decision which is updated into the chain and thus marks the end of the insert/update operations for that batch. Further, the users are able to fetch the data from the blockchain. However, any editing of the settled batch data is not possible anymore.

4.1 Use Case Diagram

The use case diagram defines all the types of actors that is used to interact with the system. Also it shows what each type of those actors can do within the system.

Let us look into details about the actors and their respective roles in the system.

Admin: This represents the main admin user.

Sensor: This represents the sensor devices.

User: This represents a general user.

Auditor: This represents the auditors from the audit department of the auditing organization.

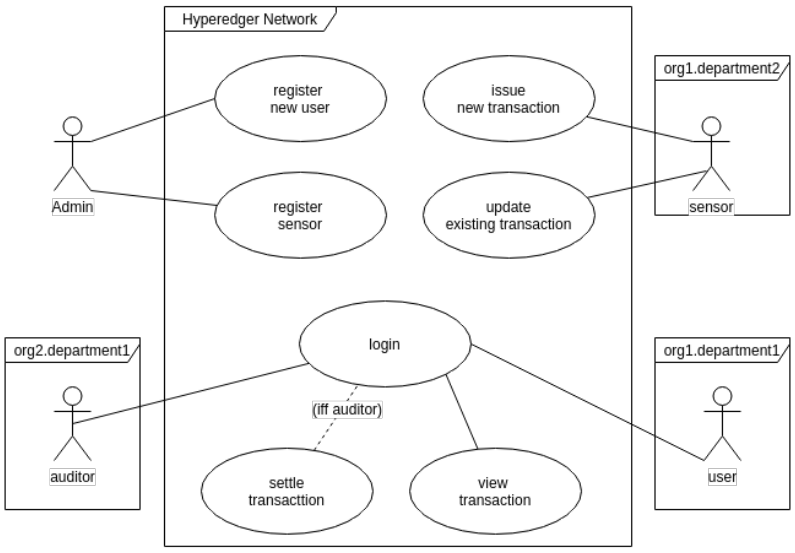


Figure 4.2: Use Case Diagram

An organizational structure comprising of two organizations were used for this project. An admin is assigned for these organizations. The first organization, *org1* is the manufacturing organization containing normal users and the sensor devices. The users and sensor devices are further assigned into two departments, *department1* and *department2*, within the *org1*. The second organization, *org2* comprises of auditor users in their separate department of auditors.

The admin user is allowed to register new users for the two organizations. The sensor users can raise a new transaction or update an existing transaction in the blockchain. The two types of users, user and auditor requires to login before making any further request to the system. Both types of users can view transactions made in the blockchain, however only the auditor users are allowed to settle the transaction after the manufacturing process is completed.

4.2 Architecture

The system was designed with a modular approach. The system is basically divided into four modules, i.e. hyperledger fabric network, API server, UI and sensors, as seen in the figure below:

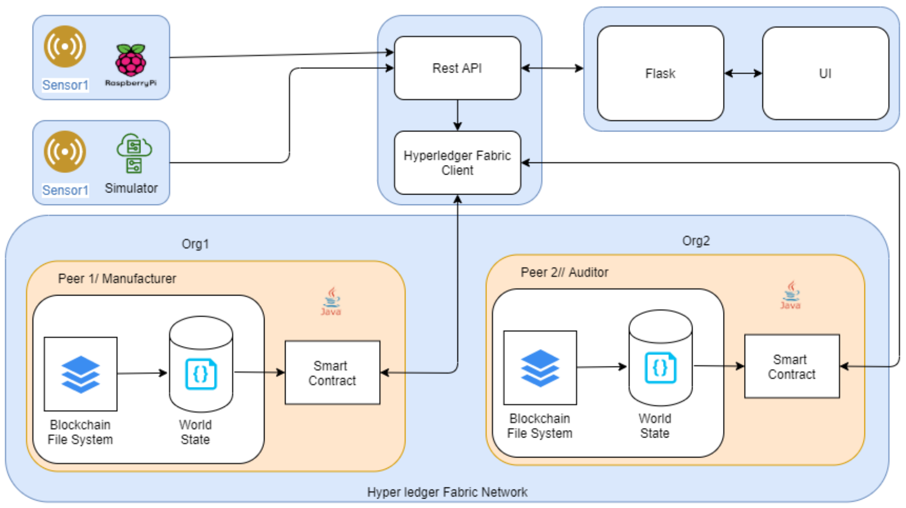


Figure 4.3: System Architecture Diagram

The core hyperledger fabric network was created in a private network infrastructure. However, it can also be implemented over a public network infrastructure, The second module is the API server. It consists of the RESTful API service for the end-users of the system and sensor devices, It also includes hyperledger fabric client component. The hyperledger fabric client component provides an interface to connect to the gateway and interact with the hyperledger fabric network. The web user interface and the sensor devices can interact with the hyperledger fabric network through the API endpoints.

4.3 Components

4.3.1 Hyperledger Fabric Network

For this project, Hyperledger Fabric[21] was used as the Blockchain framework. The two organizations were represented as a two different network hosted over docker containers. A number of docker containers were used to represent different peers, and endorsers for the two organizations. The docker containers were then collectively hosted in a cloud infrastructure in Digital Ocean Cloud Service.

To setup the network infrastructure, four peer nodes were created along with one ordering service node. The peer nodes were assigned, two each, to the two organizations, org1 and org2. Also a separate node was setup with the administrative privileges so that any deployment of smart contract or creation of new nodes in the network can be done through this particular node. Couchdb was used as world state database. The world state stores the most recent detail for every transaction stored in the blockchain. Lastly two Certificate Authority(CA) servers were also created in separate nodes for the two organizations.

To start the network, a channel was created in the network using the hyperledger fabric framework and all the peer and orderer nodes were assigned to the channel. The keys generated from the CA servers were used to validate the nodes while joining the channel.

Smart Contract Implementation

The smart contract for our project was written in java programming language. The core logic regarding data insertions and update into the blockchain was implemented in the smart contracts. The following diagram shows the flow of a transaction into different states in the blockchain. .

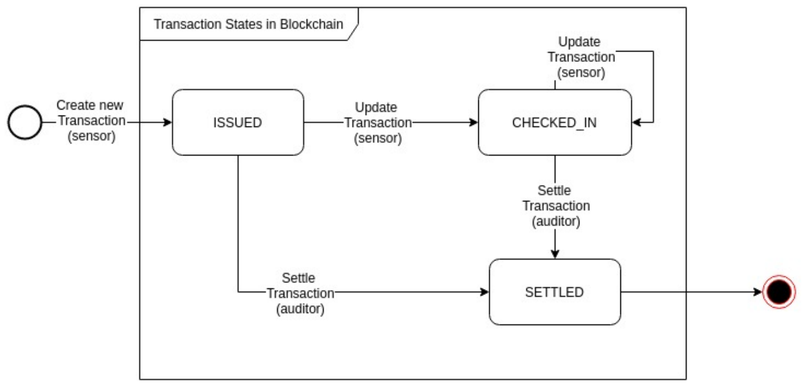

Transaction States in Blockchain
Create new Transaction (sensor)
ISSUED
Update Transaction (sensor)
CHECKED_IN
Update Transaction (sensor)
Settle Transaction (auditor)
SETTLED
Settle Transaction (auditor)

Figure 4.4: Smart Contract State Diagram

As seen in the figure above, a transaction can be in three different states, i.e. **ISSUED**, **CHECKED_IN** and **SETTLED**. When a new transaction is raised by the sensor at the start of a manufacturing process cycle, it is set to ISSUED state. For every updates received from the sensor device, the transaction is set to CHECKED_IN state. Finally, the transactions are set to SETTLED state at the end after the audit process is completed. The state of the transaction can to changed to SETTLED state only by a auditor user.

A transaction might be completed in a single step as well if all the data is sent in a single batch of data. Thus, it has been considered so that the auditor user is able to set the state from ISSUED to SETTLED directly as well.

4.3.2 API

An Application Programming Interface(API), written in nodejs, was used for interfacing between the Hyperledger Fabric Network and the client modules. This HTTP-based RESTful API enables the communication between the web client, sensor data simulator and the blockchain network.

The following table defines the API endpoints and there respective descriptions:

Sn	API Path	Description	Type	Content-Type
1	**/api/user/register**	Register new user	POST	application/json
2	**/api/sensor/ register**	Register new sensor device and returns the newly generated public key for the registered device upon successful registration.	POST	application/json
3	**/api/txn/issue**	Issue a new transaction	POST	application/json
4	**/api/txn/update**	Update and existing transaction	POST	application/json
5	**/api/txn/settle**	Settlement of existing transaction	POST	application/json
6	**/api/txn/list**	Gets the list of all the transactions	GET	application/json
7	**/api/txn/ {batchNumber}**	Gets details of a transaction	GET	application/json
8	**/api/txn/ {batchNumber}/ history**	Gets detailed history of a transaction	GET	application/json

Table 4.1: REST API endpoints

4.3.3 Sensor

A temperature and humidity sensor is used to record the data of the semiconductors during the manufacturing process. The sensor continuously records the temperature and humidity during the whole manufacturing process. The collected data is then, at regular intervals or whenever available, pushed to the blockchain.

Initially, a sensor was used with a Raspberry Pi to record the data and study the data formats and the sensor behavior. Thereafter, with the idea of how the collected data was like, a simulator was used to to simulate the whole manufacturing process.

Simulator

A simulator, written in python, was used to read the values from dataset of temperature and humidity values. The data in the dataset are then prepared into a block of data with random number of records. An initialization vector is created dynamically and the whole block is then encrypted using the key of the simulated sensor device. The encrypted data is then pushed to the blockchain through the RESTful api service(4.3.2).

Each cycle of manufacturing process is represented through a unique batch number. The prepared blocks of data for the given manufacturing process cycle are then pushed to the blockchain network under its respective batch numbers.

4.3.4 UI

The User Interface(UI) for the project was designed to interact and view the information stored in the blockchain. It provides a way for the auditor to view the stored data for their auditing process and then update the status of the transactions with the audit results. Following components were used for the UI.

Flask

Flask[22] is a light-weight web framework in python for web development. The UI was developed using flask since it is good at keeping the core functionalities simple but provides a good approach to extend any required functionalities when required.

HTML

A web based UI was used for viewing data and transaction settlement by the auditor user. Thus, Hyper-text Markup Language 5(HTML5) was used to create different components in the UI.

Bootstrap

To provide some visual styling to the UI, bootstrap, a styling library was used to provide a proper suitable design for the UI. Bootstrap, under its hood, uses Cascading Style Sheet 3 (CSS3) and javascript.

4.4 Data Encryption

Contemplating data security during the data transfer through the networks, data encryption algorithms were used to ensure that transfer of data in a secure way. A widely popular and sufficiently reliable Advanced Encryption Standard(AES) encrption algorithm was used to encrpt the data.

AES algorithm is a symmetric block cipher algorithm which utilizes a defined length initialization vector and a key to do the encryption. The encrpted data will be then send to the receiver along with the initialization vector to the receiver. The receiver then uses the initialization vector and its copy of the key to decrypt the message.

We used "aes-256-cbc" AES algorithm for our case. Our implementation of the algorithm uses 16-bits initialization vector and 32-bits encryption key of length 128-bits. The data from the sensor is encrypted using the key provided during the sensor device registration and the dynamically generated initialization vector. The data is send over the RESTful service to the node api server(4.3.2) where the data is decrypted and further passed to the core hyperledger network(4.3.1).

4.5 Client Authentication

Basic authentication method was used to authenticate the clients making requests to the hyperledger fabric network(4.3.1) through the RESTful api(4.3.2). An authentication header is sent with every request to the server. The header contains the authentication type, ”Basic” in our case, along with encoded user credentials.

For simplicity, basic authentication method was used. However a more sophisticated and secure authentication methods, applicable for the scenario, can be used as well. An example of such is Oauth2 with JSON Web Token(JWT).

4.6 Application Flow

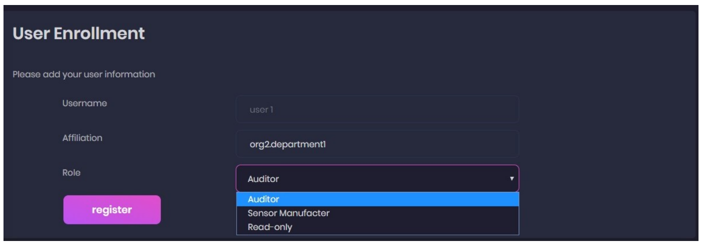
A registered sensor device starts collecting the temperature and humidity data once the manufacturing process starts. It continuously collects the data and prepares the data into blocks of data. The data is then encrypted using the secret key of the sensor device and then pushed to the API server.

The received data is decrypted at the API server. The data is then sent to the hyperledger fabric network through its client interface. The smart contract deployed at the hyperfabric network then validated the received data and issues the transaction to be stored into the blockchain. If the request is invalid or comes through unauthorized source of identity, the smart contract rejects the request.

At any given time, any authenticated users can access the data stored in the blockchain. The web UI is used for this purpose. When the manufacturing process is completed, the auditor users can then view the data and settle the transactions with the audit results. The transactions, after the settlement cannot be altered and can only be viewed as a reference data.

User Interface

User Registration


User Enrollment

Please add your user information

Username user1

Affiliation org2.department1

Role Auditor

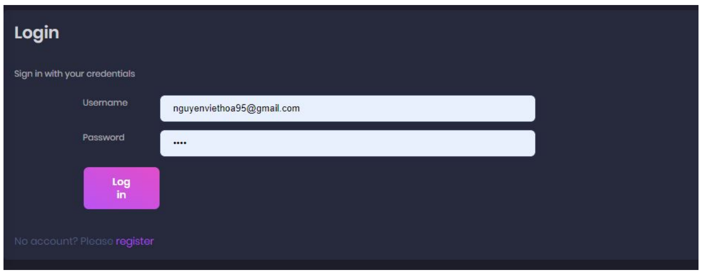
Auditor
Sensor Manufacter
Read-only

register

Figure 5.1: User Registration

The user registration form is used to get the required details for registration of a new user. Also a drop down menu is used to provide a list of available departments under the organization to which the new user can be registered to.

User Login


Login

Sign in with your credentials

Username nguyenviethoa95@gmail.com

Password ••••

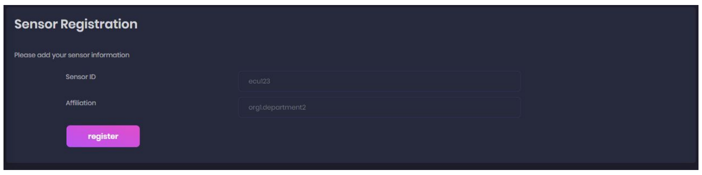
Log in

No account? Please register

Figure 5.2: Login Screen

Any user who needs to access the system need to login first.

Device Registration


Sensor Registration

Please add your sensor information

Sensor ID ecu123

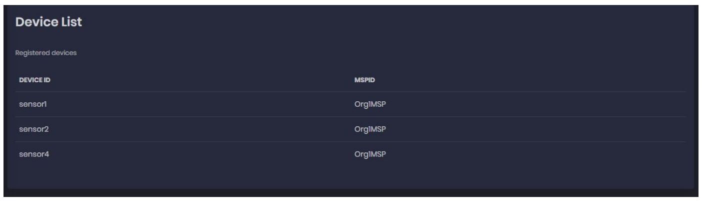
Affiliation org1.department2

register

Figure 5.3: Device Registration

Registration of the sensor devices can be done through this form. Unlike user registration, a secret key generated from the CA server of the respective organization is returned upon successful registration of the device. the key is required for device identification and encryption process during the data transmission.

Device List



Device List

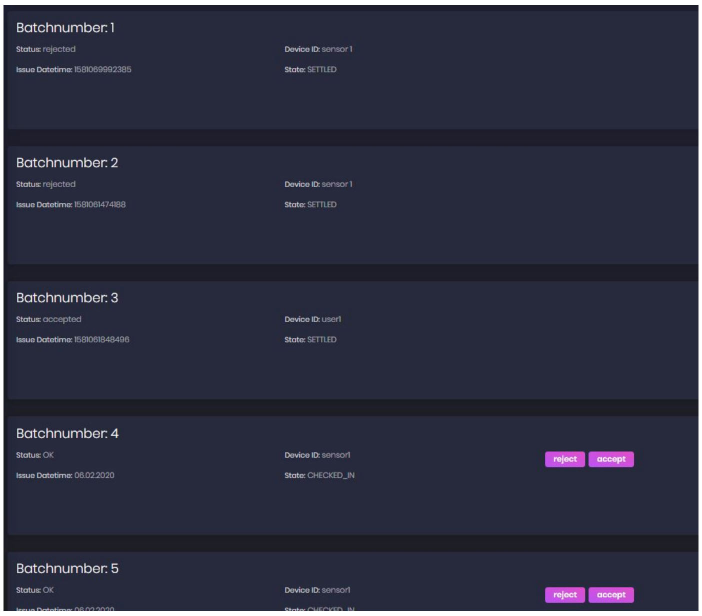
Registered devices

DEVICE ID	MSPID
sensor1	Org1MSP
sensor2	Org1MSP
sensor4	Org1MSP

Figure 5.4: List of Registered Devices

This interface shows the list of devices registered in the system along with their Membership Service Providers(MSP).

Transaction List



Batchnumber: 1

Status: rejected | Device ID: sensor 1

Issue Datetime: 1581069992385 | State: SETTLED

Batchnumber: 2

Status: rejected | Device ID: sensor 1

Issue Datetime: 1581061474188 | State: SETTLED

Batchnumber: 3

Status: accepted | Device ID: user1

Issue Datetime: 1581061848496 | State: SETTLED

Batchnumber: 4

Status: OK | Device ID: sensor1

Issue Datetime: 06.02.2020 | State: CHECKED_IN

Batchnumber: 5

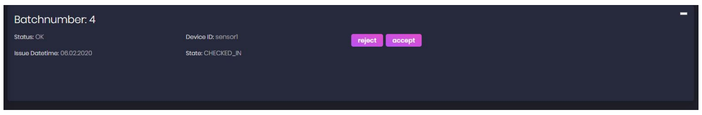
Status: OK | Device ID: sensor1

Issue Datetime: 06.02.2020 | State: CHECKED_IN

Figure 5.5: List of Transaction

This interface shows the list of all the transactions related to the particular organization which the logged in user is registered to. In the list, status of each transaction is displayed along with different state of the transactions in the blockchain.

Transaction Details



Batchnumber: 4

Status: OK

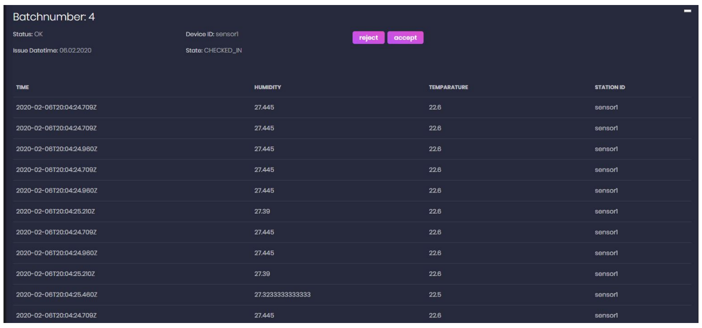
Device ID: sensor1

Issue Datetime: 06.02.2020

State: CHECKED_IN

Figure 5.6: Transaction Details

If an audit user is logged in, additional options to settle an transaction is available for those batch numbers which have completed the manufacturing process and pending for audit. The audit user can update the transaction status to accept or reject during settlement process.



Batchnumber: 4

Status: OK

Device ID: sensor1

Issue Datetime: 06.02.2020

State: CHECKED_IN

TIME	HUMIDITY	TEMPARATURE	STATION ID
2020-02-06T20:04:24.709Z	27.445	22.6	sensor1
2020-02-06T20:04:24.709Z	27.445	22.6	sensor1
2020-02-06T20:04:24.960Z	27.445	22.6	sensor1
2020-02-06T20:04:24.709Z	27.445	22.6	sensor1
2020-02-06T20:04:24.960Z	27.445	22.6	sensor1
2020-02-06T20:04:25.210Z	27.39	22.6	sensor1
2020-02-06T20:04:24.709Z	27.445	22.6	sensor1
2020-02-06T20:04:24.960Z	27.445	22.6	sensor1
2020-02-06T20:04:25.210Z	27.39	22.6	sensor1
2020-02-06T20:04:25.460Z	27.3233333333333	22.5	sensor1
2020-02-06T20:04:24.709Z	27.445	22.6	sensor1

Figure 5.7: Transaction Details Expanded View

Each transaction in the transaction list can be further expanded to view into details, the whole dataset of the manufacturing process for that particular batch number.

Project Timeline

The project was structured into four main parts as follows research and studying the literatures. analyzing the requirements and possible use cases, designing, development and testing. Like any other proof of concept projected most of the work are invested in getting understanding of the new topics and the new technology. By the end of this phrase, the team members had been able to read through the official documents of Hyperledger Fabrics and understand the key concepts. In the next step, based on the overall project definition provided by the professor, the team has defined the concrete use cases for the blockchain app.

6.1 Gantt Chart

A Gantt chart displaying the project timeline as shown in figure 6.1. Gantt charts are used to identify when the projects milestones should be completed so the project is delivered on time. A breakdown of the task, the order in which they should be competed and their time allocation is shown below.

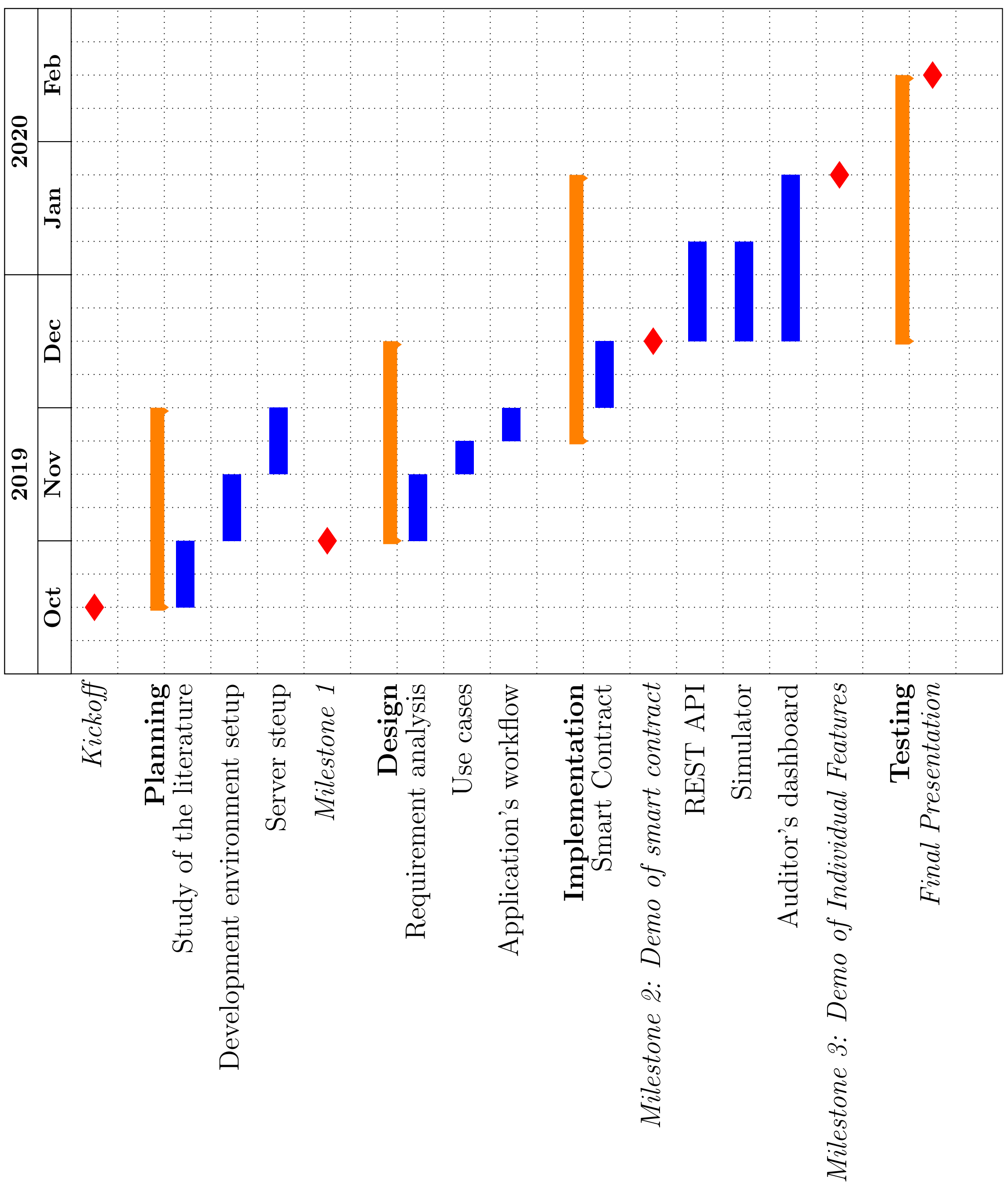


Figure 6.1: Project timeline

6.2 Individual Contribution

The individual contributions in our project are described in the table 6.1. Besides the individual contribution, the following tasks are distributed fairly between the team members:

- preparation and planning of milestones presentation
- preparation planning and writing of report.

Table 6.1: List of individual contribution.

Name	**Tasks**
Alexander Vieth	• Identifying and defining the application's use cases • Design, implementation and testing of the simulator
Elin Dangol Maharjan	• Setup the core network • Design and implementation of the Hyperledger Fabric network and chaincode • Design, implementation and testing of the REST API for client application
Sepideh Askarimarnani	• Functional testing of the application • Research for methods of data encryption
Viet-Hoa Nguyen	• Design,implementation and testing of the dashboard for auditors • Research for cloud hosting solution and setup of the virtual machine.

Project Issues

In the previous chapter the implemented blockchain application was described. In this chapter the issues encountered in process of developing the application will be identified and discussed. It also provides the solutions made by the team in order to overcome these problems. Suggestions for future developments are also presented in order to improve the application and the development process.

7.1 Issues Faced

7.1.1 Network setup problem

Setting up the framework and the network

As the start the first problem faced by the team was able to setup the network. Due to the newness of the technology, it is worth noting that the lack of technical documents and community support acted as a big barrier in the first phrase. The official documentations on readthedoc does provide a relatively good explanation about the key concepts and the overall architecture. Nonetheless, there is still opportunities for enhancement, since the documentations does not explain in depth how the basic concepts they are exactly implemented in the code, and immediate example to showcase the key concepts. For example, with change in the configuration how the behaviors of the system will be affected.

The second issue with setting up the network is to get Hyperledger Fabric network running on personal development environment in our case personal computers, this emerged from the fact that Hyperledger Fabric works better with Linux-based environments and not very compatible with Windows OS, which is used by most of the team members. Our team has been struggled for a long time to get the sample network up-and-running for all of the team members. After a lot of attempts to install Hyperledger Fabric on Windows and comparing to the same process on linux-based machine, we concluded that Windows are obviously a bad choice and we should move to another solution on linux-based machines in order to save time.

Because Hyperledger Fabric is constantly being enhanced with new features in each of the updates, it should be kept in mind that the previous versions might not work with the new version. Therefore, the lesson to be taken with is that it is important to keep only one version over the team in order to avoid unwanted incompatibility problems, from which our team has been suffered.

Understanding the configuration

The problem of configuration is particularly challenging in the context of the first network example because of the fact that this example is docker based, which provide a fairly simple starting point keeping the developer from having to deal with the architecture complexity. This example does provide a project with the network setup and a smart contracted deployed called fast car example. The simple application is structured with each node (i.e. peers, ordering service, channels) implemented as a docker container. The configuration of the nodes is defined in a config yaml file. Thus, the adjustments to be made on the network such as updating the network components involved heavily with docker commands, which are systematically defined in the read the doc.

However, this kind of low-level approach has its own drawback, since it the adjustments made per container can have unexpected impacts on operation of the whole network, which can be traced and fixed. In other words, a Docker-contained program can be fast to get up and running, but it can also be non-deterministic if the programmer makes some mistakes. Thus, the attempts to make experimental changes on simple first network example are heavily discouraged thorough the development.

In addition, the lack of ergonomic developer experience on building the network from core does not help the developer to build intuition and gain trust with the core network components. This also results into the lack of experiments with the Fabric core network of the fabcar example.

Programming language constraints

In Hyperledger Fabric, chaincode is the piece of code that runs on top of the network peers to implement the business logic of how applications interact with the ledger. When a transaction is proposed, it triggers chaincode that decides what state change should be applied to the ledger. Thus, chaincode is one of the most important components in decentralized applications on Hyperledger Fabric. The writing of chaincode is supported in variety of popular programming languages such as Golang, Java, JavaScript via Nodejs and Python. Nonetheless, with respect to functionalities, developer productivity (assuming minimal experience in each language) and community support, writing chaincode in JavaScript (Nodejs) can be easier, faster and are backed with a large source of mature documentations. This observation also applies to the development of our simple REST API, which allows a client web application to interact with the chaincode.

7.1.2 Security problem

In the existing client-server applications, the identity models rely heavily on a centralized data repository of identity. As a result, due to the decentralized nature of blockchain application, several unique challenges are introduced with blockchain application,

User authentication

Since Hyperledger Fabric is a permissioned blockchain and uses Public Key Infrastructure, every user has to be granted a credential, user credentials such as usernames and password are replaced by asymmetric cryptography, which consists of a key pair of private key and public key. In our use case, we have three different roles for users, this introduce the challenge in being able to combine this security feature of Hyperledger Fabric with a role-based authentication system.

7.1.3 Conclusions

Apparently, developing and application with Hyperledger Fabric requires a great amount of in-depth understanding in a broad range of topics such as network security, system networking as well as excellent technical skills and web application development. Even experienced developers still need time to have a clear understanding of the blockchain platforms.

7.2 Solutions

7.2.1 Solutions to network configuration issues

As mentioned earlier, at the beginning there is a lot of problems regarding to the installation of Hyperledger Fabric on different OS environments. In order to overcome the hardware constraints, we have done research to find an economical and efficient options to enable efficient collaboration between team members.

One of the most promising approach is to rent server from a cloud service provider and deploy our Fabric blockchain onto that server. This approach will alleviate our team from having to maintaining our own server, therefore, we can focus more on developing our application's core functionalities. In addition to the outright increasing in productivity, cloud solution is faster than having each team member having to install and running his/her own network, rather than we have a single networking run and maintained by only one team members. With this approach, all team members will be able to work with the latest version of the network and can contribute freely to the projects.

By doing the research, the team has come across a lot of options for cloud services providers and different models of cloud services such as Microsoft Azure, Google Cloud and Amazon Web Services. It is also worth to mention that many cloud computing providers, such as Amazon Web Services (AWS Managed Blockchain) and Microsoft Azure (Azure Blockchain Services) are now offering Blockchain as a Service (BaaS),

these solutions might be interesting the enterprise application context, since they offer a complex back-end setup for the blockchain app, functions and smart contracts with a certain fee.
However, these off-the-shell solution does have some infrastructure constraints that does not allow our team to freely implemented our own network. Having eliminated the use of BaaS, the team has decided to use a simple virtual machine from Digital Ocean. Digital Ocean Droplets was our choice of cloud computing due to its competitive price (cheapest among the common cloud service providers) and the ease to start with.
Thorough the development this solution has be proven to be a good fit. The virtual machine allows the team to work seamlessly together with zero downtime. It also provides us the possibility to easily scaling out the infrastructure when being needed such as using more server and deploy them with the network.

7.2.2 Solutions to security issues

As mentioned earlier in the use case in the section, there is a need to have a role-based authentication system. We came up with a solution in which the user identity is managed in two separated databases. The first database is credential store inside Fabric network, after the user was created, a wallet for the user is created to hold the user identity generated (public and private key). The second database is the user account database of Flash dashboard application. Each user account is given a set of unique username and self-given password (first given by the admin) and a copy version of the private key created by the MSP. The process of user registration can be describe as follows:
1. The new user is assigned with an initial username and password by the admin. The user uses this username and password to login into the dashboard application.
2. A copy of the private and public key of the user is saved in the Flask's application user identity database.
3. Each transaction proposal (approve/reject data) in form of API calls must be

signed with the auditor's private key.

4. The REST API servers receives the requests and checks the user's private key against its the version in the Fabric's credential store to determines if the user is granted with the role to perform a particular transaction proposal before invoking the smart contract.

7.3 Remaining Problems

In this section, we will focus on the remaining security challenges of deploying a blockchain with authorized participant. Even though our implementation of the user authentications has fulfilled the operational requirements of the Hyperledger Fabric network, with potentially more complex use cases, there are some drawbacks in our solution such as data fragmentation and identity theft.

Data Fragmentation

The first unsolved problem is concerned with data fragmentation. Because the user identity is fragmented among two databases, in order to keep the two databases (web application and Fabric network) synchronized in various of use cases such as password backup, password reset if the users forget their login password, user blacklist or user deactivate. Moreover, the poor off-chain password management policies can result in the key pair being the soft spot for hackers. The private and the public key of the user has to be stored off-chain in the client application, they could be exposed to , since the key pairs can be stolen and be used to perform malicious transaction on the blockchain. Therefore, there is a need for a more secured way to store the key and key exchange process.

Automate auditing process in the smart contract

In our current implementation of the smart contract, transaction details are saved into the blockchain. In order to perform audit activity such as accept and reject the sensor data entries, the auditors have to look into the data by logging in and manually check each entry for the audit and then finally also update the status at the end of the audit process. However, this process can be automated and thus eliminate the role of an auditor completely. In the future, the automation can be implemented in the following manners. First, the auditor organization define their own logic for auditing contract. Afterwards, this set of logics can be written into a separate smart contract and deployed into the network as a subcontract under the main smart contract. The main smart contract will invoke that smart contract which is independently deployed by the auditor organization. This approach will avoid a lot of manual work, since auditing logic is implemented in the smart contract and once the transaction is done, the audit report status is automatically updated to the blockchain.

Enables https on the servers

One of the further steps which can be taken in term of security is enabling https on the server we are using to deploy our Fabric network and the dashboard application. It means that the REST API has to be configured with HTTPS and TLS, in order to keep all data transfer between the REST server and all of the REST clients encrypted. Thus, we can lower the security risks by preventing attacks such as man-in-the-middle attack, or malicious request injection. This will also prevent attackers to injecting scripts, images, or contents onto the dashboard application.

7.4 Suggestion for further improvements

Hyperledger Fabric is undeniably one of the most popular projects under Hyperledger. However, it is worth noting that Hyperledger is not a single framework but an umbrella project where there are several existing projects and projects which are still actively under development. From the technical standpoint, it would be beneficial to have a look into other Hyperledger frameworks in the search of solution for our remaining problems, since these frameworks are guaranteed to be compatible with Hyperledger Fabric. In addition, the strategy in solution finding is to prioritize fully developed solutions that can cover multiple problems at the same time. For that reason, it can be boiled down to other frameworks in the Hyperledger ecosystem. In this section, two other Hyperledger projects namely Hyperledger Indy and Hyperledger Composer, which are gaining a lot of business interests will be presented along with their benefits in order to explain how these technologies will enhance the existing application.

Rapid prototyping with Hyperledger Composer

Hyperledger Fabric is still a hard-to-grasp framework, as mentioned earlier in the section. A good framework to start with developing application on Fabric is to use Hyperledger Composer, an open-source application development framework built on Hyperledger Fabric blockchain infrastructure and runtime. With Hyperledger Composer, a developer with limited knowledge about blockchain can easily build and configure core components of the blockchain which includes the network's digital asset, transaction logic, participants and access controls.

It also supports use cases and real-time testing, which can even be performed through web-based Composer playground with the need for local installations. Compared to Fabric, Hyperledger Fabric is easier to be understood and gives the possibility of creating a minimal viable product in a small amount of time. In addition, Hyperledger Composer enables developers to quickly create REST API to expose logic to web application. Another advantage of Hyperledger Composer that Yeoman(a generic

scaffolding tool) is also integrated, it can create a skeleton Angular App used as a starting point to the applications.
A major benefit of Hyperledger Composer is that it also supports data integration call Loopback to connect our existing Fabric system on Hyperledger Composer so that we can continue to work and improve the old system without having to build the core network from the beginning. We firmly believe that with using Hyperledger Composer complementarily to Fabric can support developing, testing new use cases and operating the network far more efficiently by not having to deal with the complicated coding process when using Composer's web interface Playground .

Self-sovereign identity with Hyperledger Indy

As discussed earlier, the better solution to secure the private key in association with a username/id on the blockchain or similar storage for authorization and authentication should be considered. With the emerging of blockchain as a new computing paradigm for decentralized database system, self-sovereign identity (SSI) has gained a lot of attention. SSI is based on the premise that identifier, such as usernames should be replaced with IDs that are self-owned and independent. This approach requires a decentralized identity ecosystem without administrative authorities. Hyperledger Indy is a foundation technology that enables SSI and supports independent identities on distributed ledgers and offers tools, libraries, and reusable components for providing digital identities based on blockchains or other distributed ledgers. It is based on the premise that blockchain can be used as an authentication provider. Using their key-pair, the users register their identity on the blockchain. The benefit of Hyperledger Indy is that it is interoperable across multiple domains and application. Thus, it eliminates the need for any other solutions to be incorporate into other organization's platform. Furthermore, Hyperledger Indy allows a easy-to use- password-less authenticating with fingerprint for examples from the mobile device.

Conclusion

The project primarily showed, that while Hyperledger Fabric, as permissioned blockchain framework with support for smart contracts, can be used in many different scenarios to implement features, which are beneficial to the security and trust in the given data, it is by no means applicable for every scenario.

As we've showed in our project, the potential of one of the flagship features - the easy implementation of smart contracts - would not be fully ultilized, in a scenario, in which one or more business partners want to store data from IoT devices, like temperature sensors.

We've deduced, that the problem in the proposed scenario lies in the interests of the parties involved. Because sensors are the means by which the business partner's asses the status of their product, there is invested interest in seeing that data at all time.
This includes illogical data in the case of malfunctioning sensors, anomalous data, like an unusually sharp rise in temperature, in a situation where the given product is on fire. While the data coming from the sensors in these cases are certainly unusual, that is an even more important reason to log and store the data, to both analyze and react to potentially broken sensors. The smart contract system would allow us to query every peer in the network if they endorse a transaction or not, but since there is almost no reason not to endorse sensor data, that feature becomes almost meaningless in our project. The smart contract could be used to detect spam, or perform checks on the format of the transaction/sensor data, but these checks could also be implemented by a webserver.

However, we find that storing and distributing sensor data in a blockchain, improves the transparency and trust in the handling of the sensor data from the point of every business partner involved. The distributed manner in which the data is stored, prevents one of the business partners to have a monopoly on the data, which can damper the trust, that the data is not being tempered with. The immutable characteristics of the blockchain also provides an unfalsifiable record of the sensor data, which we imagine is beneficial in a court of law, but certainly is useful when trying to reconstruct events regarding the sensors.

Another conclusion that we've come to, is that following Nakamotos example [16] of using encryption to verify the validity and authenticity of transactions, is one of the best approaches to enhance the integrity of the blockchain. Encryption, in this case, is just the tool for the actual feature that is beneficial, which is user/sensor authentication.

One of the biggest weaknesses in purely storing every sensor data package, without a rejection policy, besides checking if the format of the data is correct, is that we cannot differentiate between forged or real sensor values. Anomaly detection systems always operate with a degree of fuzziness, when it comes to categorizing data. Fuzziness in this case relates to a margin of error, with both false positives and false negatives. In the problem analysis chapter, we've discussed the point, where anomaly detection is useful. There we came to the conclusion, that theoretically, with a perfect anomaly detection system, you could gain beneficial additional information regarding the suspiciousness of a given data point. While such a system can theoretically detect forgeries, by sensing tiny deviations from a detected pattern, in reality such a system is a minefield on many levels. Anomaly detection systems can, in most cases, be described as a statistical hypothesis test [23] and therefor will contain false positives and false negatives. Depending on that margin of error, the benefit of using anomaly detection to mark data, is greatly reduced. If anomaly detection where to be used as a basis to reject data from being stored, the margin of error can be fatal for the business partners involved. Important, but anomalous data, could then be invisible to the involved business partners. Just anomaly detection, even in a perfect scenario, also would not detect forged data, that resembles the regular values and patterns.

In the problem analysis chapter, we've described a scenario, in which an attack similar to Stuxnet is used. An attacker that can forge sensor data to cover up the real state of the sensors, will probably not be detected by using anomaly detection.

We've found that adding encryption helps to prevent these kind of man-in-the-middle attacks. We assume that the raspberry pi was not compromised, since in that scenario, there is not much that one can do. We have found, that ensuring the authenticity of the provider of the sensor data is a crucial factor, which has to be implemented with care, when handling IoT/sensor data (although that statement would also be true for almost every system).

The combination of using a permissioned blockchain, in addition to encrypting the sensor data produces a reasonable degree of data integrity and protection against forgery. While not all of Hyperledger Fabrics features add values to the proposed scenario, it does provide an easy implementation that can be used to add additional layers of security, when handling sensor data.

Bibliography

[1] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *2017 IEEE International Congress on Big Data (BigData Congress)*, pp. 557–564, June 2017.

[2] A. Siyal, A. Junejo, M. Zawish, K. Ahmed, A. Khalil, and G. Soursou, "Applications of blockchain technology in medicine and healthcare: Challenges and future perspectives," *Cryptography*, vol. 3, p. 3, 01 2019.

[3] E. Androulaki, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, and et al., "Hyperledger fabric," *Proceedings of the Thirteenth EuroSys Conference on - EuroSys '18*, 2018.

[4] T. M. Fernández-Caramés and P. Fraga-Lamas, "A review on the use of blockchain for the internet of things," *IEEE Access*, vol. 6, pp. 32979–33001, 2018.

[5] M. Memon, S. S. Hussain, U. A. Bajwa, and A. Ikhlas, "Blockchain beyond bitcoin: Blockchain technology challenges and real-world applications," in *2018 International Conference on Computing, Electronics Communications Engineering (iCCECE)*, pp. 29–34, 2018.

[6] A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, "Blockchain and iot integration: A systematic survey," *Sensors*, vol. 18, no. 8, p. 2575, 2018.

[7] M. Sigwart, M. Borkowski, M. Peise, S. Schulte, and S. Tai, "Blockchain-based data provenance for the internet of things," in *Proceedings of the 9th International Conference on the Internet of Things*, IoT 2019, (New York, NY, USA), Association for Computing Machinery, 2019.

[8] M. Crosby, P. Pattanayak, S. Verma, and V. Kalyanaraman, "Blockchain technology: beyond bitcoin. appl. innov. rev.(2016)," *Sutardja Center for Entrepreneur-*

ship & Technology Report, Berkeley University. http://scet. berkeley. edu/wp-content/uploads/AIR-2016-Blockchain. pdf, 2016.

[9] "Wafer manufacturing — how are semiconductors made?." `https://www.waferworld.com/wafer-manufacturing-process/`. Accessed: 12.03.2020.

[10] "Secure secure shell." `https://stribika.github.io/2015/01/04/secure-secure-shell.html`. Accessed: 12.03.2020.

[11] P. D. Monika Chakraborty *et al.*, "Owasp web application security quick reference guide 0.2," *The OWASP Foundation*, 2013.

[12] "Uci machine learning repository." `https://archive.ics.uci.edu/ml/index.php`. Accessed: 12.03.2020.

[13] "Occupancy detection data set." `https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+`. Accessed: 12.03.2020.

[14] N. Falliere, L. O. Murchu, and E. Chien, "W32. stuxnet dossier," *White paper, Symantec Corp., Security Response*, vol. 5, no. 6, p. 29, 2011.

[15] "Man-in-the-middle attack." `https://owasp.org/www-community/attacks/Man-in-the-middle_attack`. Accessed: 12.03.2020.

[16] S. Nakamoto *et al.*, "A peer-to-peer electronic cash system," *Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf*, 2008.

[17] R. Chatterjee and R. Chatterjee, "An overview of the emerging technology: Blockchain," in *2017 3rd International Conference on Computational Intelligence and Networks (CINE)*, pp. 126–127, Oct 2017.

[18] S. Thakur and V. Kulkarni, "Blockchain and its applications – a detailed survey," *International Journal of Computer Applications*, vol. 180, pp. 29–35, 12 2017.

[19] C. Pirtle and J. Ehrenfeld, "Blockchain for healthcare: The next generation of medical records," *Journal of Medical Systems*, vol. 42, no. 9, p. 172, 2018.

[20] Hyperledger, "hyperledger-fabricdocs documentation." `https://hyperledger-fabric.readthedocs.io/_/downloads/en/release-2.0/pdf/`, March 2020. version:2.0, Accessed:26-03-2020.

[21] Hyperledger, "Hyperledger project." `https://www.hyperledger.org/`, March 2020. version:2.0, Accessed:2020-03-02.

[22] Flask, "Flask web development." `https://flask.palletsprojects.com/en/1.1.x/`, March 2020. version:2.0, Accessed:26-03-2020.

[23] A. Soule, K. Salamatian, and N. Taft, "Combining filtering and statistical methods for anomaly detection," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pp. 31–31, 2005.