

# A Verifiable Data Store and Exchange for an IoT Environment

## Abstract

A recent rise in the role which IoT devices play, in order to deem the cities as 'smart' has led to heightened access to these devices for the general public. This meteoric rise in the number of IoT devices is abreast with enormous amounts of essential, marketable data generated by them. While previously, bleeding-edge tech hubs and opportunistic governments used, governed and administered these sophisticated devices, now they are becoming increasingly accessible to the public for usage as well as ownership, and influence the lives of all the stakeholders. Thus, in the modern smart city ecosystem, it is pertinent to protect the data generated by the various IoT devices, to make it immutable, tamper-proof and more importantly verifiable, with no single authority governing and opaquely managing it. In this project, we hence propose a decentralized, trust-less data storage architecture which tackles the problem of data verifiability and makes any data stored in it verifiable and immutable by practice.

## 1 Introduction

Numerous IoT devices revolve around smart cities. These are essentially micro-controller boards installed intuitively at various avenues like public transport, traffic lights, public parks, research laboratories weather stations throughout the city. With the advancement in the fields of networking, a high bandwidth and stable connection of these devices to a cloud are now conceivable and in fact, implemented. Thus, devices are more than capable of generating their own data and have numerous sensors embedded within them. This live data specific to the city is a very essential and marketable entity. It can be of extensive use to the research community, to study the patterns of various walks of life, public habits, as well as the environmental and climatic fluctuations.

It is interesting to note, that access of technology to the common people has increased at a dramatic pace with the low cost of developing controller chips. While earlier, it was bleeding edge tech companies, private firms or the government itself which managed and interacted with these devices, now the common public is an equally important stakeholder. This decade has been marked by a sharp rise of the startup culture, and as a result, small groups of individuals now have ownership of various IoT devices, and themselves have turned into data producers and consumers. This has led to a new, unpredicted problem - data verifiability. With individuals turning into data producers, it is difficult to prove the genuineness of data, hence there is a pressing need for a mechanism or practice which makes it possible to verify data. Also, data once generated should be tamper-proof, and stored inside an immutable data storage. Additionally, the data should not be under the control of a single entity in order to make each stakeholder equally powerful in the eco-system. Along with data, it is equally important to have an audit trail of the ownership of devices, and their configuration. With the expanding digital market, there has emerged a need for digital representation of hardware entities. With most of the transactions involving ownership of even hardware now regulated through interconnected networks, there is a necessity of a complete audit trail of their activity.

Blockchain, a technology which emerged to power cryptocurrencies has seen a meteoric rise in its usage in various domains and is no longer restricted to cryptocurrencies. Also, it supports data immutability, and decentralization by design. Ethereum [1], in particular, has taken a strong initiative to make the development of Ethereum powered web apps called dApps (distributed apps) feasible. Since the storage on blockchain is expensive, various off-chain storage alternatives which complement the blockchain technology like - InterPlanetaryFileSystem [2] have emerged which allow storing the data off-chain yet retaining its immutability. However, for our use case, wherein we also need to verify the genuineness of the data - i.e. it's actual origin is the one which is being claimed, the transactions and calculations need to be very fast, without any latency as IoT devices

may be generating large volumes of data simultaneously in short intervals - too short for blockchain as it requires mining and there is a significant time gap before each data point is written - and since we require each data point to be verifiable, there would have to be a separate transaction for each of the data points, something which would be very computationally expensive and time-consuming on a traditional blockchain. Furthermore, querying for the data on the blockchain is not as efficient as a traditional database can offer, for various applications. BigchainDB [3], a project recently funded by the European Union has emerged as a technological breakthrough in the field of immutable storage, which along with being tamper-proof is absolutely scalable, with a latency comparable to traditional client-server systems while at the same time, featuring the properties of a blockchain. In this project, we build a marketplace model on top of BigchainDB along with data verification based upon a signature based public-private key infrastructure - Ed25519 [4] and featuring a complete audit trail of a device ownership and configuration using the concept of digital twins - a feature which BigchainDB supports natively. The marketplace shall allow individual owners of data and devices to register themselves, create digital twins of their devices and make the entire process from data generation to data storage and ownership completely transparent and verifiable.

## 2 Motivation

Preceding section discussed how data is produced by various stakeholders with increased access to a plethora of IoT devices, and how it is a prime commodity which has extensive use in several domains. However, the genuineness of data is equally, if not more, important. Any research which is carried out generates the results on the basis of the data fed into it, in the preliminary stages, and predictions rely solely on the validity of data. Fake data can wreak havoc and have disastrous consequences for the scientific and corporate community alike. Due to the increase in the number of data producers, it is very easy to fake data, generating enormous amounts of random data to populate datasets is very much conceivable. However, if strict regulations are put in place and stringent rules are imposed by the authorities, it will hamper the data production by the masses, something which is undesirable in order to realize the true potential of a holistic data-driven smart society, in which all the stakeholders contribute their part. Thus, we propose to develop a verifiable data store and exchange, in which it shall not only be possible to verify each single data point stored inside an immutable storage, but also to keep track of the ownership of the IoT devices and their various parameters at all times.

## 3 Problem Definition

The discussion in preceding sections stresses the importance of all the stakeholders of a smart city eco-system having an equal authority and be eligible to become data producers without having to worry about providing the genuineness of the data. To tackle this problem, we proposed developing an exchange/marketplace embedded with a data verification module, built on top of BigchainDB - a highly scalable database solution with blockchain like properties. The marketplace to be developed shall fulfill the following requirements.

### 3.1 Immutability

The data once stored should be completely secure - authenticity, confidentiality, and integrity should be maintained, and if tampered, it should be easily detectable.

### 3.2 Scalability

A smart city network can contain thousands of IoT devices, all sending simultaneous streams of data. The system developed should thus have a very low latency and be scalable in general to accommodate the number as well as the large size of data. Also once the data is stored, it should be query-able with the same efficiency as a traditional database like MySQL, or MongoDB.

### 3.3 Verifiability

The data should be credible in order to make any use of it, and hence it should be possible to verify each and every data point using an intuitive verification mechanism.

### 3.4 Decentralization

The system should be completely decentralized, without a single point of failure. The idyllic goal would be to develop a Byzantine fault tolerant system, so even a physically compromised node should not serve as a point of failure.

### 3.5 Audit trail of ownership and configuration

The system should allow the users to provide a complete audit trail of ownership and any configuration i.e. any abstract and non digital changes should have adequate digital representation.

## 4 Literature review

Internet of Things has swayed many areas, and there have been many IoT applications that have improved system performance as well as quality of life such as healthcare, manufacturing, monitoring and so on [5]. The wireless sensors play key role in integrating the IoT device with central controllers for further processing, moreover, they have to be "smarter" [6]. IoT devices generate data continuously that can help improve not only healthcare but also improve autonomous driving, surveillance systems and so on. Researches like Nyugen's[7], show that the industry as well as researchers are interested in the IoT data and need a model for storage and exchange of this data. It is estimated that IoT will reach 26 billion units by 2020, up from 0.9 billion in 2009, and will impact the information available to supply chain partners and how the supply chain operates. As a result, IoT would provide new opportunities as well as challenges [8]. Since the volume of data is large and is expected to increase rapidly, a data store that scales horizontally is necessary. Moreover, as the data would be from a plethora of sources, the data would be heterogeneous; a data storage system that could store heterogeneous data is required.

For the challenges mentioned above, a system that is horizontally scalable and could store heterogeneous data is required. Many traditional databases are based on structured relational model. Although, relational databases are of eminent importance, they are ineffective in storing and processing big data. In such scenarios distributed databases like NoSQL databases and Hadoop are gaining popularity. NoSQL databases provide horizontal scaling, dynamic modification of data schema, distributed index, etc. that relational databases don't have [9]. On the other hand, NoSQL databases lack in ACID properties. Many scholars have conducted researches on storing data with NoSQL database. A framework is also proposed that allows use of Structured Query Language (SQL) in relational as well as NoSQL databases [10]. Most of the researches have proposed storing data from a individual sources separately. In [11], unified data storage that stores structured as well as unstructured data from various sources is proposed. Although this does not provide transparency and verifiability that any other user who wants to use data would want for assurance, they store structured data in a database and the unstructured data in file system. Yet, the problem is not solved, the requirement is one storage system that is diversified yet not heterogeneous.

Another problem is data integrity. Data integrity is an exacerbated issue in cloud storage as data owners hardly control where their data are stored, who can actually access them, and how. Nevertheless, more and more public and private data is being made public as "it relieves the burden of maintenance cost as well as the overhead of storing data locally"[12]. Hence, it is an urgent need to address data integrity. Data integrity is generally handled using cryptographic tools and methods like encrypting data or signing it using cryptographic signatures. So for an attack to be effective it would need violation of secret keys, yet once realized these attacks are undetectable. Therefore, it is advisable to utilize rigorous data replication strategy to ensure anyhow data integrity. Data replication and distribution pose threat to data integrity, as a attacker must compromise all the data. This replication approach is widely used in cloud computing environments, where there is abundance of distributed storage resources. Although, replicating data in cloud increases burden of an attack in cloud environment, cloud providers could themselves collude with the attackers and violate data integrity. Thus, blind trust must be avoided, not even the cloud providers should be

given access to use or manipulate delegation of data. Such a system could be brought by using blockchain.

Nowadays, blockchain based databases are popular. Blockchain can be perceived as a replicated database distributed among thousands of diverse nodes. In its first conception, it has been used as a public ledger in BitCoin transactions [13]. The properties of blockchain make it important in any field. The data once stored in the blockchain cannot be deleted, this property of immutability ensures data integrity and consistency. The nodes are decentralized, thus there is no true owner of a blockchain and any transaction cannot be successful until it has gained a majority of support from the nodes in the network. The mining process used in BitCoin and Ethereum is still *proof-of-work* (PoW). It includes a computationally intensive hashing task that regulates the average time spent by a miner in generating a new block that is to be integrated to the chain. Once a miner generates a block it broadcasts the blocks to other miners. Although *proof-of-work* ensures data integrity, it has one major drawback: *performance*. This is due to braodcasting latency of blocks in the network and the time intensive task of PoW. In BitCoin, the average latency is 10 minutes and the throughput is about 7 transactions per second [14]. Another relevant concern is blockchain's stability, though, it has been well so far, there is no valid research that states why this is the case and for how long would this be continued [14]. Encryptions provide data verifiability, the owner can sign the data that is outsourced so that the source is no longer anonymous. Asymmetric encryption provides two way security as the data cannot be forged and it cannot be used by illegal user. Nonetheless, once tha keys are compromised, nothing remains safe, it is important that the keys be kept as much secure as possible. In [15] Gaetani et.al. propose a blockchain database that uses a rotation consensus mechanism instead of *proof-of-work* where a miner is selected as leader at each round. The leader is responsible for receiving new operations, signing them with its private key, and broadcasting them to other miners. All the miners add these operations to their local *ledger*,and apply these to their local replicas. Another similar approach is acquired by BitCoin-NG [16], a bitcoin protocol modified to improve performance.

Thus, it is found, that certain properties of a distributed system; like low latency, high throughput is required for storage and to secure the data stored it is essential that the database is immutable, decentralized and allows exchange of assets along with keeping track of them. A decentralized and distributed storage is the answer that has the best of the two worlds. BigChainDB [3] is one such blockchain database that works on same grounds as Bitcoin-NG [16] but employs a Byzantine Fault Tolerant blockchian engine that uses voting based on *proof-of-stake* rather than *proof-of-work* or rotational consensus mechanism that doesn't consider the stake of a data owner in order to store the data. BigchainDB allows malicious miners but isolates them and their nodes from harming the rest of the nodes. BigchainDB seems a promising database that makes the application stack (that comprises of application, processing, file storage and database) into a complete decentralized and distributed application stack that paves way to Web3.0.

## 5 Proposed Methodology

### 5.1 Data exchange and Verifiable data

We propose to build a data exchange, a web application which allows several device owners to register their devices.This concept represents the convergence of the physical and the virtual world where every industrial product will get a dynamic digital representation. Throughout the product development life cycle, right from the design phase to the deployment phase, organizations can have a complete digital foot print of their products.

It is possible, to model the hardware device, completely wherein, the the user is the owner of the device, the device itself has the ownership of the sensors. The notion of maintaining a log of ownership of hardware equipments, is a long standing tradition, not introduced with the introduction of digital twins; however, traditional logs are explicit, in the sense that at first, the ownership or configuration state changes, and then that change is logged manually, usually by a third party. It is easily possible for someone with malicious intent, to log the data incorrectly, and there is no means to verify the data. On the other hand, the novelty of our approach lies in the key fact that ownership, and other configuration state changes, update their counterpart, automatically, without the requirement of executing a manual procedure or involving any interaction with users. And since BigchainDB is an immutable database, there is no editing or *UPDATE* operation in database terminology possible. Hence, each time a change occurs, a new transaction record is

created. This results in an immutable audit trail of all the previous devices, at a fraction of the computational cost of a traditional blockchain.

Features	Blockchain	Distributed Database	BigchainDB
Immutability	✓	✗	✓
No Central Authority	✓	✗	✓
Assets over network	✓	✗	✓
High Throughput	✗	✓	✓
Low Latency	✗	✓	✓
Queryable	✗	✓	✓

Table 1: Features of BigchainDB.

We intend to use BigchainDB [3] which uses some clever engineering techniques that make it an enterprise-grade distributed system that incorporates a blockchain infrastructure. It is a hybrid system that has the best of both, a blockchain and distributed system and hence, overcomes problems generally faced by the two. Table 1 offers a comprehensive comparison of bigchaindb with other technologies. It offers traditional benefits of blockchain, like asset transfer, tamper resistance and decentralized storage and control. It has capabilities of low latency, high throughput, full-featured NoSQL query language, high capacity and many others that a NoSQL database like MongoDB can provide. Tendermint [17], a Byzantine Fault Tolerant blockchain virtual engine provides plug-and-play consensus. It uses *proof-of-stake* rather than traditional *proof-of-work* for validation and consensus voting. Figure 1 shows the components of a BigchainDB nodes. For Byzantine Fault Tolerance, BigchainDB requires a cluster of minimum of 4 nodes, which can communicate with each other. We intend to use docker containers to deploy such a cluster. BigchainDB exposes an HTTP API, we intend to build our own custom API on top of it, which will allow our data exchange platform to interact with BigchainDB. Also, the custom API will allow us to successfully query the data with the same efficiency as an industry grade database management system, as the support for querying in BigchainDB is limited.

## 5.2 Verification

The data verification uses the cryptographic algorithm Ed25519 which uses an asymmetric public-private key pair. Unlike traditional approaches, raw data is not sent directly to the BigchainDB server, rather it is at first, one-way encrypted using SHA-3 256 Algorithm, and a hash is obtained. This hash is then signed by the private key, using the crypto functions exposed by the Ed25519 library. The reason any form of raw data is first converted to a hash using SHA-3 is that, it helps in maintaining the process efficiently and is of constant time. If raw data which is highly variable in size is converted directly, it will prove to be computationally expensive to sign it, if it's large in size, whereas SHA-3 ensures that, the generated hash is constant in size (64 bits). The signing of hash, generates a signature.

It is important to note that the integrity of private key must be maintained, i.e., it should be inside tamper-proof hardware, whose seal if broken should render the node as compromised and the servers should refuse accepting data from the given node. This problem of hardware integrity lies in the domain of remote attestation. [18]. *Trusted Platform Module* or TPM is an international standard which is used for cryptographically securing devices. TPM chips are used to cryptographically secure hardware devices.

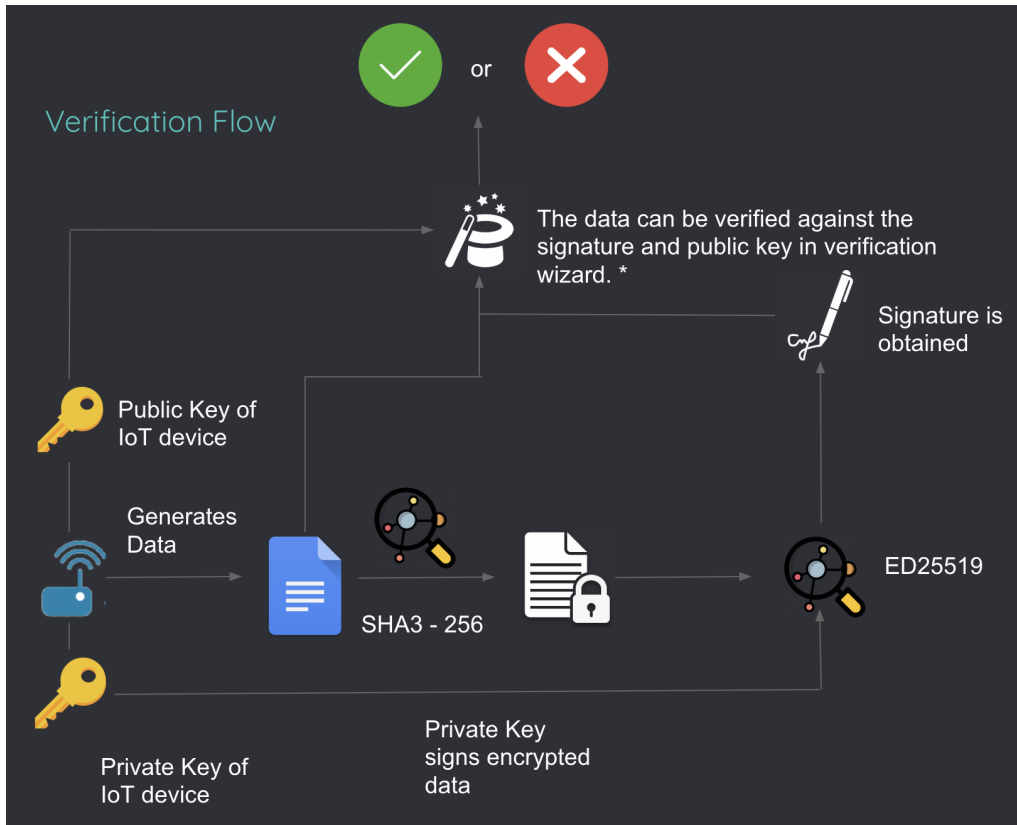
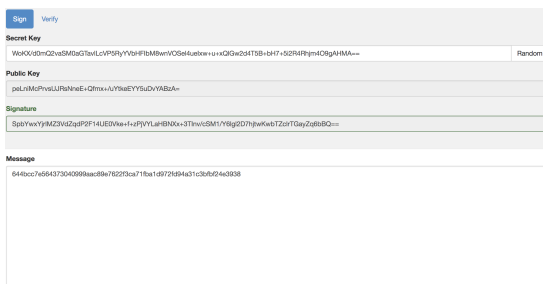
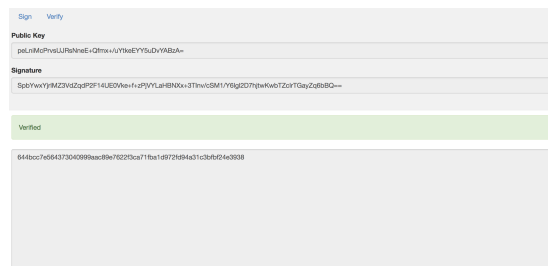


Figure 1: The verification algorithm.

As Figure 1 shows, these three components, the *public key*, the *signature*, and the *hash* which was signed can then be used to verify the validity of data. For instance, the SHA-3 hash for *hello world* is `644bcc7e564373040999aac89e7622f3ca71fba1d972fd94a31c3bfbf24e3938`. Figure 11, shows the signing and verification procedure for the given string using a javascript implementation of Ed25519. Figure 2a shows signing with the private key, which results in a signature, while Figure 2b shows the verification taking place. Figure 3 shows the connections of the Raspberry Pi to achieve the result.



(a) Signature generation



(b) Data Verification

Figure 2: An example of data verification



Figure 3: Raspberry Pi - Set Up

## 6 Hardware and Software requirements

### 6.1 BigchainDB

BigchainDB is like a usual database with added characteristics of blockchain. The properties of blockchain that are included are decentralization, owner-controlled assets, immutability. The properties of database include high transaction rate, indexing and querying of structured data, low latency. The data in BigchainDB is stored in transactions. For querying in BigchainDB, a node operator can use the power of MongoDB's query engine to search and query all stored data, including all transactions, assets and metadata. The node operator can decide for themselves how much of that query power they expose to external users. Figure 6 shows the internal structure of a BigchainDB node, and the various components present inside it.

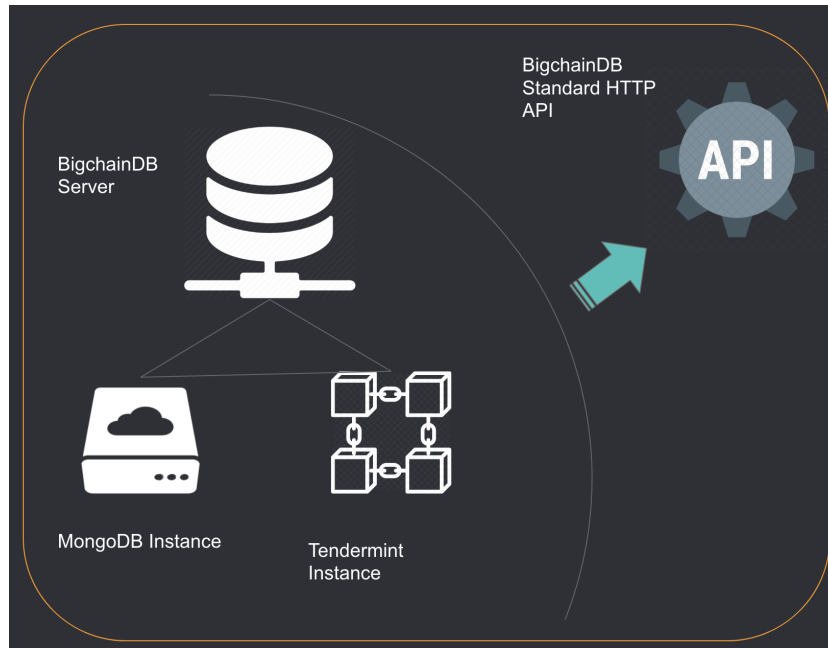


Figure 4: The Components of a bigchainDB node.

In BigchainDB, **transactions** are the smallest records and are used to register, create, issue or transfer things (e.g. assets). The ID of a transaction is a unique hash that identifies a transaction. There are two kinds of transactions: CREATE transactions and TRANSFER transactions.

### 6.1.1 CREATE Transactions

The CREATE transaction creates a new asset on the network. It is the start of a whole chain of transactions that handle this asset. There can be only one CREATE transaction per asset, and an asset can never be altered, it is immutable.

### 6.1.2 TRANSFER Transactions

TRANSFER transaction is used to update the information of an asset and to transfer the ownership of an asset. TRANSFER transactions build upon previous transactions. A TRANSFER transaction can only transfer shares of one asset at a time.

In BigchainDB, data is structured as **assets**. An asset can characterize any physical or digital object that you can think of like a car, a data set or an intellectual property right. These assets can be registered on BigchainDB either by users in CREATE transactions or transferred (or updated) to other users in TRANSFER transactions. At BigchainDB, the primary focus is assets (e.g. a client order can be an asset that is then tracked across its entire lifecycle). This switch in perspective from a process-centric towards an asset-centric view influences much of how we build applications.

An **Input** to a transaction in BigchainDB is a pointer to an output of a previous transaction. It specifies to whom an asset belonged before and it provides a proof that the conditions required to transfer the ownership of that asset are fulfilled. In a CREATE transaction, there is no previous owner, so an input in a CREATE transaction simply specifies who the person registering the object is. In a TRANSFER transaction, an input contains a proof that the user is authorized to "spend" (transfer or update) this particular output. In practical terms, this means that with the input, a user is stating which asset (e.g. the bike) should be transferred.

A transaction **Output** specifies the conditions that need to be fulfilled to change the ownership of a specific asset. For instance: to transfer a bicycle, a person needs to sign the transaction with his or her private key. This also implicitly contains the information that the public key associated with that private key is the current owner of the asset. A transaction can also have multiple outputs. These are called divisible assets.



The **metadata** field allows users to add additional data to a transaction. This can be any type of data, like the age of a bicycle or the kilometers driven. The good thing about the metadata is that it can be updated with every transaction. In contrast to the data in the asset field, the metadata field allows to add new information to every transaction.

The **Transaction ID** is a unique hash that identifies a transaction. It contains all the information about the transaction in a hashed way.

## 6.2 Tendermint

Tendermint is software for securely and consistently replicating an application on many machines. By securely, we mean that Tendermint works even if up to 1/3 of machines fail in arbitrary ways. By consistently, we mean that every non-faulty machine sees the same transaction log and computes the same state. Secure and consistent replication is a fundamental problem in distributed systems; it plays a critical role in the fault tolerance of a broad range of applications, from currencies, to elections, to infrastructure orchestration, and beyond.

Tendermint is designed to be easy-to-use, simple-to-understand, highly performant, and useful for a wide variety of distributed applications.

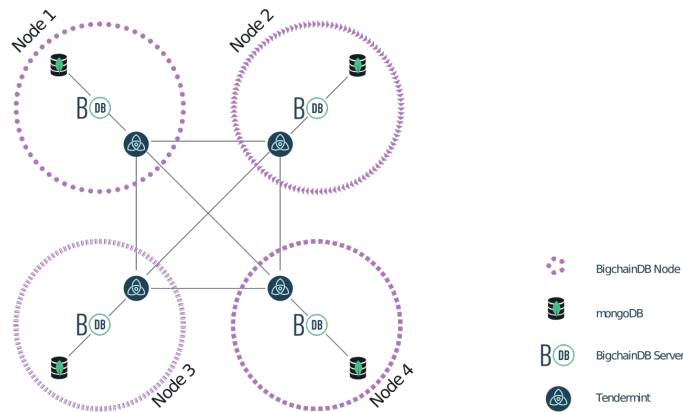


Figure 5: Communication between Bigchaindb nodes

Figure 5 shows how BigchainDB nodes communicate with each other.

The nodes in BigchainDB communicate with each other using Tendermint [17] wire protocols. There's a local MongoDB database in every BigchainDB node, but they're all independent. Messing with the MongoDB on one node won't affect any of the others. Another difference is that all the replication, voting, and consensus logic is done by Tendermint, not MongoDB or BigchainDB. Tendermint is a software that connects the nodes and gets them all agreeing on the current state. It will get them to agree even if up to 1/3rd of the nodes fail in arbitrary ways. However, Tendermint has no notion of what is inside the node. Tendermint doesn't implement the nodes. It leaves that up to like BigchainDB.

Tendermint is known to be quite fast, at least compared to many other operational blockchain systems (such as Bitcoin or Ethereum). Tests have achieved transaction rates of thousands of transactions per second. The latency from the time a transaction is submitted to the time it's included in a finalized block is around one second, for a global network.

## 6.3 Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks

to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

Docker is able to share the host OS across multiple “Containers” rather than requiring each one to have and run its own full operating system. This allows you to encapsulate your application into a reusable module that can be plugged in and run on any machine where resources are available. This allows for more fine grained resource allocation and can minimize the amount of wasted CPU or memory resources.

And importantly, Docker is open source. This means that anyone can contribute to Docker and extend it to meet their own needs if they need additional features that aren’t available out of the box.

## 6.4 Ed25519

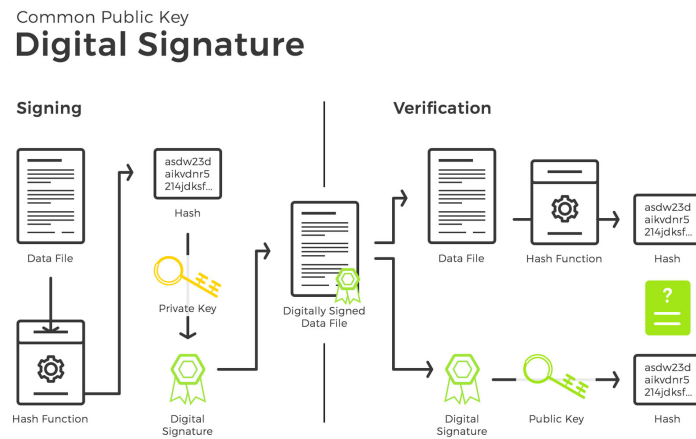


Figure 6: Digital Signature Verification

Figure 6 is a flowchart that shows how the data is first signed digitally and then verified.

Ed25519 is a public key verification system with remarkable fast key generation and signing mechanisms [4]. It boasts a high security level with collision to resistance to hash collisions. Ed25519 is featured in several python and C packages, which provide the users an abstract layer, and allows them to sign and verify the data without dwelling into complexities of cryptography.

## 6.5 Raspberry Pi

The Raspberry Pi is a low cost, small sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a little device that enables people to explore computing, and to learn how to program in languages like Scratch and Python. It’s capable of doing everything you’d expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing.

The following figure shows how the connections are implemented in the Raspberry Pi.

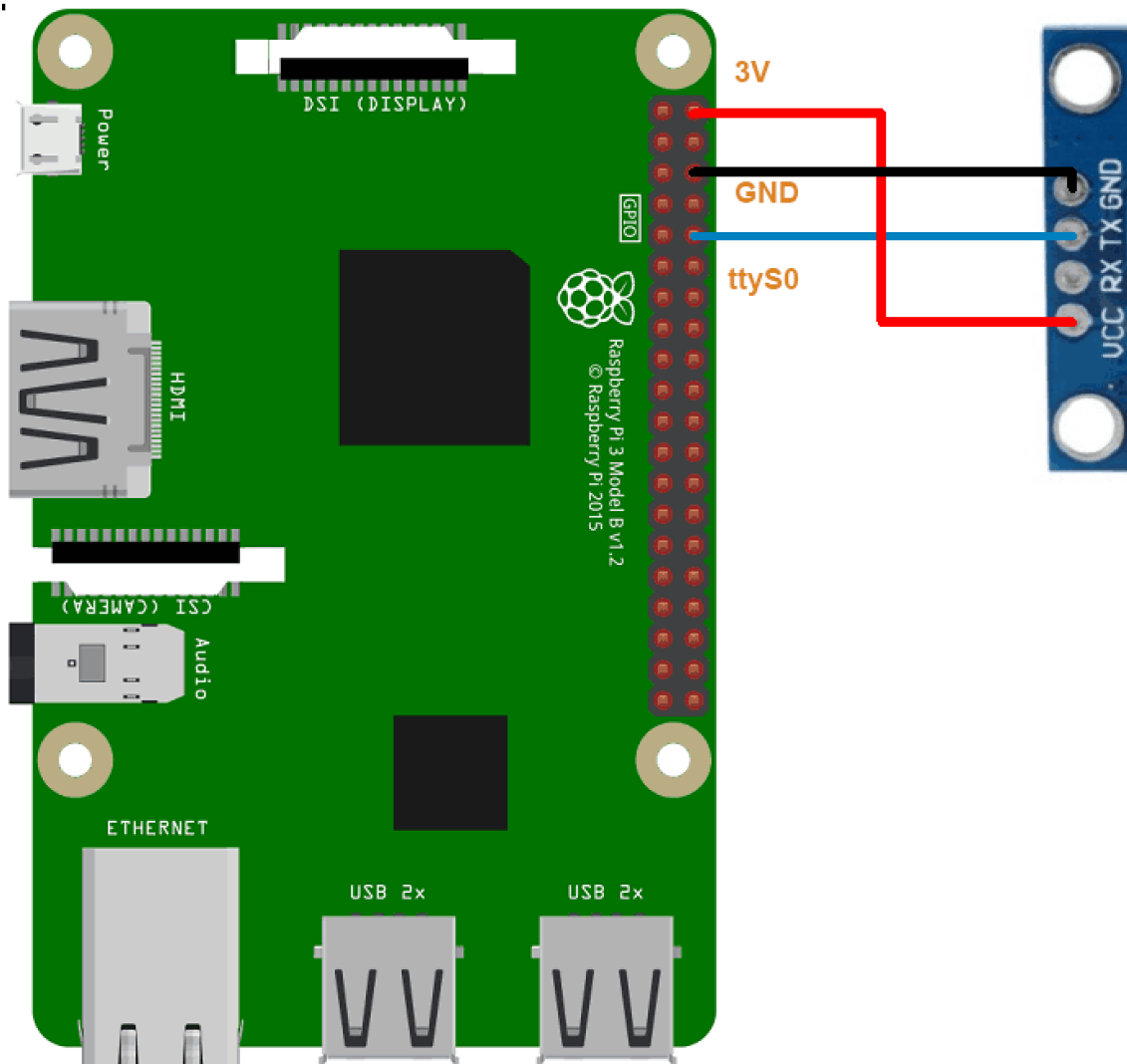


Figure 7: Raspberry Pi - Connections

Raspberry Pi has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines to weather stations and birdhouses with infra-red cameras. The Raspberry Pi was designed for the Linux operating system, and many Linux distributions now have a version optimized for it.

## 6.6 IoT Sensors

A sensor is a device that is able to detect changes in an environment. By itself, a sensor is useless, but when we use it in an electronic system, it plays a key role. A sensor is able to measure a physical phenomenon (like temperature, pressure, and so on) and transform it into an electric signal.

The Internet of Things is one of the most important and promising technologies today. Around us, there are smartphones, wearables, and other devices, all of which use sensors. Nowadays, sensors play an important role in our life and as well as in IoT. Sensors monitor our health status (e.g. a heartbeat), air quality, home security, and are widely used in the Industrial Internet of Things (IIoT) to monitor production processes.

IoT sensor devices are the key ingredient in the overall IoT system. IoT sensor devices are sensors that can communicate their readings to internet cloud services for further aggregation and trend analysis.

### 6.6.1 SKG 13BL GPS Engine Module

The SKG13BL is a complete GPS engine module that features super sensitivity, ultra low power and small form factor. The GPS signal is applied to the antenna input of module, and a complete serial data message with position, velocity and time information is presented at the serial interface with NMEA protocol or custom protocol. The small form factor and low power consumption make the module easy to integrate into portable device like PNDs, mobile phones, cameras and vehicle navigation systems.

## 6.7 Node.js

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser. Node.js lets developers use JavaScript to write Command Line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user’s web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server side and client side scripts.

Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications (e.g., real-time communication programs and browser games).

## 6.8 Google Cloud Platform

Google Cloud Platform, offered by Google, is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products. Alongside a set of management tools, it provides a series of modular cloud services including computing, data storage, data analytics and machine learning. Google Cloud Platform provides Infrastructure as a service, Platform as a service, and Serverless computing environments.

## 6.9 Monit

Monit is a small Open Source utility for managing and monitoring Unix systems. Monit conducts automatic maintenance and repair and can execute meaningful causal actions in error situations.

The BigchainDB and Tendermint processes are managed together using Monit. BigchainDB and Tendermint are managed together, because if BigchainDB is stopped (or crashes) and is restarted, Tendermint won’t try reconnecting to it. (That’s not a bug. It’s just how Tendermint works.)

Monit will run the BigchainDB and Tendermint processes and restart them when they crash. If the root bigchaindb process crashes, Monit will also restart the Tendermint process.

# 7 Implementation

## 7.1 Virtual Machines and BigchainDB nodes

Google cloud platform(GCP) was used to create 4 instances of virtual machines powered by Ubuntu 18 LTS. The 4 instances have external public IP Addresses via which they establish communication to allow for BigchainDB to replicate data.

Milestone Achieved	Deadline
Virtual CPUs	1 core
Random Access Memory	4 GB
Storage Instance	10 GB
Operating System	Ubuntu 18.04 LTS

Table 2: Timeline.

Each of the VMs represent one of the nodes of the BigchainDB cluster. Each node was configured to support an instance of MongoDB, Tendermint, and BigchainDB server each. All three of

the components need to execute simultaneously in order for them to act as a BigchainDB node. This is achieved through monit. Monit daemon automatically starts all three of the processes as soon as the VM machine boots.

## 7.2 Raspberry Pi and SKG 13BL GPS Module


On the client side, a Raspberry Pi is interfaced with a SKG13BL GPS Module as show in figure. A system service is created, which runs a javascript script as soon as the system boots. The GPS receiver module uses Universal Asynchronous Receiver/Transmitter(UART) communication to communicate with controller or PC terminal. The hardware interface for GPS units is designed to meet the National Marine Electronics Association (NMEA) requirements. We extract Latitude, Longitude and time information from string received from GPS module using Python. After extraction, an Asset object is created as as required for a CREATE transaction. The transaction object is signed by the private key of the Raspberry Pi device, which is hardcoded into it. A subsequent POST request is sent to one of the 4 BigchainDB nodes, to commit the transaction. A successful transaction results in replication of data to all 4 of the nodes instantaneously.

## 8 Results


To fulfill the notion of the verifiability introduced at the beginning, a verification wizard, aided by the consortium of BigchainDB nodes is generated by leveraging the Crypto Modules of Python3. For verification of a given data point, three entities are essential - the SHA-256 hash of the data point, the public key of the IoT Node which generated the data point, and the signature generated when the hash was signed by the corresponding private key of the IoT Node. Hence we designed an intuitive, user-friendly datapoint verification wizard.

It has been devised to aid the end-user, even if totally unaware of crypto or validation processes, to audit data independently, through a graphical interface, with an easy point-and-click experience.


**Data Point Verification Wizard**



**Description**  
See Public Key, Singature, Message



**Hashing**  
Generate or enter your own hash



**Verification**  
Verify the signature

**Data Point - Readings**

```

{
  "Temperature": 24.206257854839407,
  "Altitude": "19",
  "Longitude": "15.59541",
  "Latitude": "38.25947",
  "Date": "2018-07-15T00:13:06.020Z"
}

```

**Full Response**

▼ See the full response (The actual message signed and stored)

```

{"asset": {"data": {"Altitude": "19", "Date": "2018-07-15T00:13:06.020Z", "Latitude": "38.25947", "Longitude": "15.59541", "Temperature":

```

**Public Key**

**Signature**

**The following are needed for verifying a reading:**

- Message Hash:** The hashed (using SHA3-256) message which represents actual readings and is signed by private key of the IoT node. The hash is generated by encrypting the full response.
- Public Key:** The corresponding public key of the IoT node.
- Signature:** The signature generated after the encoded message is signed by the private key.

Figure 8: Data Verification Wizard Step 1

The wizard operates in three steps. First step as shown in Fig. 8 is an informative step. It shows the raw readings, with an option to see the entire response - the actual message which was encrypted as mentioned previously. It also makes it convenient for the user by extracting the signature from the fulfillment object, the public key is displayed as well. These three pieces of information are self-sufficient to verify the validity of the data.

The step 2 of the wizard which is shown in Fig. verification-2.png presents an option to the user to generate the SHA-256 hash of the full response and shows the data to be encrypted in a field which is intentionally editable, to allow the user to tamper with the data, and see the outcome of verification.

### Data Point Verification Wizard

✔

**Description**  
See Public Key, Singature, Message

#

**Hashing**  
Generate or enter your own hash

⚖

**Verification**  
Verify the signature

**Full Response**

▼ See the full response (The actual message signed and stored)

```
{"asset":{"data":{"Altitude":"19","Date":"2018-07-15T00:13:06.020Z","Latitude":"38.25947","Longitude":"15.59541","Temperature":
```

**Public Key**

```
GPCMV87ei5XewCmG7a1fibPYVNFgWHWf19aPuZyRe47
```

**Singature**

```
2CVHINKTu2kfhKLDyJ7ID8bpbcpI7a9pB5uhD1WANow2ivURm4Y13SSPajuS1MfxPwwcpzCS5xsbqWrgtDHckWw
```

**Data**

```
{"asset":{"data":{"Altitude":"19","Date":"2018-07-15T00:13:06.020Z","Latitude":"38.25947","Longitude":"15.59541","Temperature":24.206257854839408,"entity":"reading","resource_id":"23594d30-0b38-4967-97ef-c961fd9fdbbb","type":"Temperature"},"id":null,"inputs":{"fulfillment":null,"fulfills":null,"owners_before":["GPCMV87ei5XewCmG7a1fibPYVNFgWHWf19aPuZyRe47"]},"metadata":null,"operation":"CREATE","outputs":{"amount":"1","condition":{"details":{"public_key":"GPCMV87ei5XewCmG7a1fibPYVNFgWHWf19aPuZyRe47","type":"ed25519-sha-256"},"uri":"ni://sha-256:otq8G9vIHQYPuiYFuef30idT840rbxvdkVvGP43Cb4?fpI=ed25519-sha-256&cost=131072"},"public_keys":["GPCMV87ei5XewCmG7a1fibPYVNFgWHWf19aPuZyRe47"]},"version":"2.0"}
```

Go Back
Generate Hash
Exit

**This is how the hash is generated:**

- The full message is converted to binary before being encrypted by SHA-3 256.
- You can use an external tool to generate this hash, or use the one provided here. Either way proceed to next step to generate/enter the hash.

Figure 9: Data Verification Wizard Step 2

After generating the hash using the wizard, or entering the SHA3-256 hash generated from an external source (the hash field is also editable), the user lands on step 3, the final step of the verification wizard as shown in Fig. 10. On pressing the verify button the hash is verified against the public key and the signature if the data has not been tampered with, and the public key is indeed of the IoT node which signed the data hash, the verification succeeds, else it fails and the wizard shows the appropriate message as shown in Fig. 11a and Fig. 11b respectively. This last step again can be carried out independently and there is no compulsion to use this wizard for verifying the hash against the signature.

### Data Point Verification Wizard

<b>Description</b> See Public Key, Signature, Message	<b>Hashing</b> Generate or enter your own hash	<b>Verification</b> Verify the signature
--	---	---

**Full Response**

▶ See the full response (The actual message signed and stored)

**Public Key**

GPCMV87ei5XewCmG7a1fbPYVNFgWHWf19aPuZyRe47

**Signature**

2CVHINKTu2kfHKLdyJ7ID8bpbcpI7aN9pB5uhD1WANow2ivURm4Y13SSPaju51MfxPwwcpzCS5xsqbWrgtDHckWw

**Hash**

b7da08f9075f668ce51f0f4ce0dcf369fc146efb58b48be970a7aa2f82f417dc

Go Back
Verify
Exit

**Note:**  
The hash visible in the field *Message Hash* is the hash generated using SmartME APIs. It is an editable field, and you may enter your own hash generated from an independent source.

Figure 10: Data Verification Wizard Step 3

#### Data Point Verification Wizard

<b>Description</b> See Public Key, Signature, Message	<b>Hashing</b> Generate or enter your own hash	<b>Verification</b> Verify the signature
--	---	---

**Full Response**

▶ See the full response (The actual message signed and stored)

**Public Key**

GPCMV87ei5XewCmG7a1fbPYVNFgWHWf19aPuZyRe47

**Signature**

2CVHINKTu2kfHKLdyJ7ID8bpbcpI7aN9pB5uhD1WANow2ivURm4Y13SSPaju51MfxPwwcpzCS5xsqbWrgtDHckWw

Go Back
Exit

**Success!!**  
The verification of the reading was successful against the public key and signature!

#### Data Point Verification Wizard

<b>Description</b> See Public Key, Signature, Message	<b>Hashing</b> Generate or enter your own hash	<b>Verification</b> Verify the signature
--	---	---

**Full Response**

▶ See the full response (The actual message signed and stored)

**Public Key**

GPCMV87ei5XewCmG7a1fbPYVNFgWHWf19aPuZyRe47

**Signature**

2CVHINKTu2kfHKLdyJ7ID8bpbcpI7aN9pB5uhD1WANow2ivURm4Y13SSPaju51MfxPwwcpzCS5xsqbWrgtDHckWw

Go Back
Exit

**Failure!!**  
The verification of the reading has failed against the public key and signature!

(a) Successful data verification

(b) Failure of data verification

Figure 11: Outcomes of verification wizard step-3

## 9 Conclusion

The preliminary work so far in the project has allowed us to find a suitable alternative for blockchain for getting an immutable data storage, without compromising the efficiency of an industrial grade database - BigchainDB. Also, in order to achieve the goal of data verifiability, Ed25519, has been tested in isolation, by feeding it a random feed of data, and can later be used with live data coming from sensors. The verification wizard successfully acts as a proofing mechanism for the Public Private Key based signature verification approach. The instantaneous reflection of the readings as soon as they are generated in the Raspberry Pi Module, showcase the feasibility of our work in a networking domain.

## References

- [1] Gavin Wood, Ethereum: A Secure Decentralised Generalised Transaction Ledger, 2014
- [2] Benet, Juan. "IPFS-content addressed, versioned, P2P file system." arXiv preprint arXiv:1407.3561 (2014).
- [3] Trent McConaghy, Rodolphe M., Andreas M., Dimitri De J., Troy McConaghy, Greg McMullen, Ryan H., Sylvain B., and Alberto G., BigchainDB: A Scalable Blockchain Database, June 8, 2016.
- [4] Bernstein, D.J., Duif, N., Lange, T., Schwabe, P. and Yang, B.Y., 2012. High-speed high-security signatures. Journal of Cryptographic Engineering, 2(2), pp.77-89.

- [5] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.
- [6] A. Biru, R. Minerva, and D. Rotondi, "Towards a definition of the Internet of Things (IoT)," IEEE Tech. Rep., 2015. [Online]. Available: <http://iot.ieee.org/definition.html>
- [7] Nguyen Cong Luong, Dinh Thai Hoang, Ping Wang, Dusit Niyato, Dong In Kim, and Zhu Han, "Data Collection and Wireless Communication in Internet of Things (IoT) Using Economic Analysis and Pricing Models: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, fourth quarter, 2016
- [8] Zeng D., S. Guo, Z. Cheng *The web of things: a survey* J. Commun., 6 (6) (2011), pp. 424-438
- [9] J. Guo, L. Xu, G. Xiao, and Z. Gong, "Improving multilingual semantic interoperability in cross-organizational enterprise systems through concept disambiguation," *IEEE Trans. Ind. Informat.*, vol. 8, no. 3, pp. 647–658, Aug. 2012.
- [10] O. Curé, R. Hecht, C. Duc, and M. Lamolle, "Data integration over NoSQL stores using access path based mappings," *Lect. Notes Comput. Sci.*, vol. 6860, pp. 481–495, 2011.
- [11] An IoT-Oriented Data Storage Framework in Cloud Computing Platform Lihong Jiang, Li Da Xu, Hongming Cai, Zuhai Jiang, Fenglin Bu, and Boyi Xu.
- [12] Mehdi Sookhak, Abdullah Gani, Hamid Talebian, Adnan Akhuzada, Samee U. Khan, Rajkumar Buyya, and Albert Y. Zomaya. Remote Data Auditing in Cloud Computing Environments: A Survey, Taxonomy, and Open Issues. *ACM Comput. Surv.*, 47(4):65:1–65:34, May 2015.
- [13] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. Available at <https://bitcoin.org/bitcoin.pdf>.
- [14] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121. IEEE, 2015.
- [15] Gaetani, E., Aniello, Leonardo, Baldoni, Roberto, Lombardi, Federico, Margheri, Andrea and Sassone, Vladimiro, Blockchain-based database to ensure data integrity in cloud computing environments, Italian Conference on Cybersecurity, Venice, Italy, 2017.
- [16] Ittay Eyal, Adem Efe Gencer, Emin G˘un Sirer, and Robbert Van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59, 2016.
- [17] Jae Kwon, Tendermint: Consensus without Mining, 2014.
- [18] Jain, L., Vyas, J. (2008). Security analysis of remote attestation. CS259 Project Report, Tech. Rep.