Blockchain Based Protocol for IoT Device Authentication and Secure Communication

A dissertation submitted for the Degree of Master of Information Security

K. S. Dasun

University of Colombo School of Computing

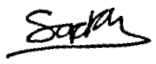
2019

DECLARATION

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name	:	K. S. Dasun
Registration Number	:	2015/MIS/004
Index Number	:	15770042



Signature — 01/09/2019 Date

This is to certify that this thesis is based on the work of

Mr. K. S. Dasun

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by	:	Dr. Kasun de Zoysa
Supervisor Name	:	Dr. Kasun de Zoysa

Signature — /09/2019 Date

ABSTRACT

The Internet of Things (IoT) has gained a huge popularity during the recent years expanding into many fields and industries. The massive generation and collection of data by IoT devices also come with severe threats to security and privacy aspects. After understanding the importance of securing IoT device networks, many researchers have focused on developing solutions to protect IoT networks. However, providing complete security in IoT domain was challenging due to resource constraints and the lack of standards among devices. This project aims to create a more secure IoT device platform based on blockchain technology to protect IoT device networks.

The principal objective of the project is divided into four sub-problems as IoT device authentication, data storage, data retrieval and secure communication. Then sub-problem solutions are designed according to industry accepted security standards and combined to create the complete solution based on blockchain technology. The solution has been implemented as a reusable platform using Java programming language with Spring framework to ensure that application is portable across all platforms that support Java Runtime Environment (JRE). The implemented aggregator application publishes a RESTful API which is consumed by connected IoT devices and other aggregators in the network. The aggregator applications are connected to build a distributed network which maintains the blockchain used for data storage.

Finally, a comprehensive evaluation process has been performed by developing a prototype IoT network. The results obtained during the tests are analyzed to find the limitations of the solution and possible future enhancements. This thesis explains the background of IoT security domain and the design of a novel approach to secure IoT networks.

ACKNOWLEDGEMENT

I would like to take this opportunity to thank many people who helped me to make this project a success with great pleasure. At the beginning, I must say that even though only a few names are mentioned here, I have been lucky enough to receive help and support from many people who I recall with a great pleasure while writing this. I am very thankful to all of them.

Great thanks especially to Dr. Kasun de Zoysa, Senior Lecturer at University of Colombo School of Computing, for taking time out of his busy schedule to perform the duty as the supervisor. Without his guidance, support, and motivation this project would not have been possible.

Special thanks go to Dr. Manjusri Wickramasinghe, Lecturer at University of Colombo School of Computing, for guiding us throughout the project.

Next, I am grateful to the lecture panel at University of Colombo School of Computing, for all their hard work in teaching us. The support and resources they provided was a large part of what made this project successful.

Finally, I would like to extend my deepest gratitude to my family for their never-ending support and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACE – Authentication and Authorization for Constrained Environments

AES – Advanced Encryption Standard

API – Application Programming Interface

CIA – Confidentiality, Integrity and Availability

DES – Data Encryption Standard

DSA – Digital Signature Algorithm

IP – Internet Protocol

IoT – Internet of Things

MAC – Message Authentication Code

NTP – Network Time Protocol

OSCAR – Object Security Architecture for the Internet of Things

PIR – Passive Infrared Sensor

PKCS – Public Key Cryptography Standards

PoW – Proof of Work

RNG – Random Number Generator

RSA – Rivest–Shamir–Adleman

SDN – Software Defined Networking

SHA – Secure Hash Algorithm

SQL – Structured Query Language

SSP – Single Sign-On

TCP – Transmission Control Protocol

TLS – Transport Layer Security

UDP – User Datagram Protocol

WLAN – Wireless Local Area Network

CHAPTER 1 INTRODUCTION

1.1 INTRODUCTION

IoT, which stands for Internet of Things, refers to an ever-growing network of devices, which includes home appliances, vehicles, devices with embedded sensors and the communication medium, which enables the devices to connect and exchange information. IoT extends the Internet beyond standard devices such as desktops, laptops, and mobile phones to a wide verity of connected computing devices. Based on the statistics, the IoT devices are growing exponentially by integrating with all aspects of the physical world. The use of IoT devices results in direct economic benefits, improved living standards, and improved efficiency of industrial processes, and leads to a more connected world. The extensive application set of IoT technology can be categorized as consumer devices and industrial devices at a higher level.

Consumer applications: Connected vehicles, wearable devices, health monitoring tools, smart home devices, security cameras and remote monitoring equipment's, etc.

Industrial applications: Road monitoring, transport vehicle monitoring, automated manufacturing supply chains to respond dynamic product demands, environmental monitoring, connected weather stations, redefined agricultural processes, etc.

IoT devices are physical objects that use sensors to collect data and various application programming interfaces (APIs) to exchange data over the Internet. Furthermore, IoT devices are also collaborating with other IoT devices to perform their functions. Most of the IoT devices communicate using wireless mediums. However, the connectivity protocols and the processing capabilities, of IoT devices largely depend on the application. The non-standard nature of the IoT devices and the use of proprietary protocols create many challenges when it comes to integrating IoT devices from different sources.

The growing number of IoT devices have raised numerous challengers in security and privacy aspects. The researchers have demonstrated the possibility of exploiting IoT device networks to gain illegal access and to perform unauthorized modifications to information collected and processed by IoT devices. The difficulty of adding common security measures to resource constrained IoT devices and the lack of standards among devices have made the IoT devices

vulnerable to many security attacks. The security measures used in conventional Internet applications to protect information and privacy tend to be inappropriate to protect IoT device networks. However, if appropriate security measures have not been implemented these connected devices could provide a much larger surface for attackers to target home and industrial networks.

1.2 MOTIVATION

The exciting idea of adding intelligence to objects were found in many early projects even though their progress was slow due to the lack of affordable technology. In today's IoT networks, many devices with network connectivity are collecting and sharing data with each other. Thanks to cheap hardware components such as sensors, processors, network connectivity chips, increasing availability of broadband connections, and wireless networks, it is possible to group almost any set of devices to form an IoT network. This merges the physical and digital worlds by allowing the devices to send and receive, real-time data without human intervention. In most of the consumer applications of IoT, the devices connected to the IoT network can be controlled and monitored from a remote location either by another device or through an interface like a mobile phone. A light bulb which is turning on and off based on data received from a PIR (passive infrared sensor) sensor, which also allows to be controlled through a smart phone can be considered as an example IoT application. An IoT device can be simple as a toy or complex as an autonomous vehicle which can navigate without any aid from a human.

However, with the rapid growth of IoT devices which link devices to the Internet opens them up to several vulnerabilities if they are not fully protected. In most of IoT applications, the security of devices or their communication networks has not always been the top priority during IoT product design phases. From manufactures point of view, security is found as an expensive thing which is slowing down the development while reducing the usability of their products. IoT networks are created using different operating systems, programming languages, hardware components, and communication protocols. There are many cases, where the IoT appliances are integrated with legacy systems, which are using non-standard interfaces. There is a high chance that these legacy systems are not updated to protect against modern security threats.

Most of the IoT devices are resource constrained and do not contain resources required to implement strong security mechanisms. Moreover, they are not intelligent to perform other tasks, which are outside of their intended purpose. Implementing strong cryptography-based algorithms

on those devices will slow down their embedded processors and can negatively affect the user experience. Furthermore, securing all the devices in the network individually is difficult because the attack surface is enormous. The IoT devices may be distributed throughout a wide geographical area using the connectivity medium as the Internet. IoT networks store and transmit sensitive and financially valuable data, which entails huge potential gains and, hence attracting attackers. Most of the IoT devices reside in vulnerable environments, which are within a reachable distance by attackers and the natures of the deployments make protecting individual devices extremely difficult.

Regardless of their security concerns IoT application are becoming more and more popular in every day among both consumer users as well as among industrial environments. The growth of IoT networks is mostly driven by efficiency, cost savings, and enhanced consumer experience. However, one of the key concerns of successful adoption of IoT is having sufficiently strong security mechanisms employed to protect the integrity and privacy of data stored and transmitted in the network. For a consumer to use IoT device network with confidence, the network should be able to ensure that it is not compromised, and all the data stored in the network as well as the data communication channels used by the components are secure. Hence implementing security measures becomes critical to ensure protecting privacy and safety of IoT device networks.

Many researchers have attempted to build secure IoT networks by focusing on identifying the root causes for security weaknesses observed in small to large scale existing IoT networks and developed solutions, which are tailored to specific environments. However, most of these proposed solutions are limited to individual applications of IoT and they are not generally usable as a framework to secure common IoT networks. Furthermore, some of the developed solutions rely on proprietary protocols which are not compatible with others making them difficult to integrate. Due to the lack of common generic framework, most of the IoT device networks completely ignore the security or rely only on basic security features. Hence, if there is a such framework, it will help to secure IoT networks as well as to reduce the development time required when implementing IoT data communication methods from the scratch.

1.3 PROBLEM DEFINITION

There are many secure communication protocols exists for standard Internet devices to achieve protection against unauthorized interception and modifications while providing authentication for communicating ends (e.g.: TLS/SSL). Firewalls, antivirus programs, intrusion detection/prevention systems (IDS/IPS), security auditing tools and many other utilities have helped to keep malicious activities off the devices and communication networks. The similarity of the devices has made developing tools and protocols relatively easy. Over the years, these protocols and utilities combined with limited physical access to the critical devices have been proved to be successful in protecting end user information and privacy.

IoT devices are capturing and communicating sensitive and critical data that must be protected from unauthorized access. However, the standard Internet protocols are unsuitable to secure IoT devices and communication networks due to the resource constraints and the lack of adherence to common standards among different IoT devices. Most of the IoT devices do not have necessary computing resources required to use general-purpose secure communication protocols, which demands a considerable amount of processing power in order to encrypt/decrypt a steady stream of packets without a delay. Furthermore, most of the traditional communication protocols depends on a central server, which facilitate authentication, authorization and secure communication, while providing services to the connected clients. The centralized nature of such communication protocols can make IoT networks more vulnerable to denial of service attacks.

IoT devices typically register with the network themselves and configure automatically without human intervention. Traditional security mechanisms are built on top of the assumption that, the sensitive devices are not physically accessible to the attacker. However, many IoT devices reside in places where the attackers can gain physical access. Spoofing attacks and impersonating are common on IoT networks due to this fact. Hence, establishing immutable identity for the devices is a very important aspect in protecting IoT networks.

The purpose of this project is to research and implement a more secure method to authenticate IoT devices and protect the integrity of data collected, processed, and transmitted by IoT devices through insecure networks.

1.4 OBJECTIVES

The main objective of this project is to research and implement a more secure IoT device authentication and communication protocol. The developed solution will facilitate secure enrollment of IoT devices to the network. After the enrollment, the IoT devices must authenticate before they can transmit or receive information. The authentication process should involve more than one aggregator device to ensure that the connected device is not compromised. Then only the IoT device is authorized to perform privileged functions.

IoT networks typically generate and communicate sensitive data. Hence the solution is developed to protect confidentiality of the data communicated between IoT devices, to preserve the integrity of information and provide the ability to verify the integrity of data stored in the network. A security measures will be implemented to avoid reply attacks in the IoT device network. In order to make sure that, all devices in the IoT network running the latest version of the software the developed solution can push updates securely to connected devices. Moreover, this research expects to implement the above security measures in a reusable framework, which can be used in IoT communication networks to support connected devices while gaining protection against malicious attacks.

1.5 SCOPE

This project will propose a suitable blockchain based protocol to authenticate IoT devices, preserve the confidentiality and integrity of information exchanged between IoT devices. The proposed solution will be implemented as a general communication platform for IoT devices. The solution should facilitate high availability of the network in where the IoT devices can be continuously operational even during connected device failures. Furthermore, a control API will be implemented to monitor and interact with the IoT device network. The developed control API is also capable of allowing the devices and users to verify the integrity of data stored in the system.

Moreover, a prototype IoT device network will be implemented using the developed framework and evaluated to for resource consumption and the security. The results obtained during the evaluation will be summarized to measure the suitability of the implemented solution to protect IoT device networks.

1.6 STRUCTURE OF THE DISSERTATION

The dissertation will provide an overall idea about the process used to design solution and develop the prototype of the secure IoT platform. The dissertation has divided into six main chapters.

CHAPTER 2 – LITERATURE REVIEW

Literature review gathers information related to the project being developed. The literature review chapter provides a summary of related technologies and discusses the suitability of selected technologies.

CHAPTER 3 – DESIGN OF SOLUTION

The design of solution chapter describes the architecture and the methods to be used to implement the solution. Moreover, this chapter shows how information gathered in the literature review can be transformed into complete detailed solution design that focuses on how to deliver the expected features.

CHAPTER 4 – IMPLEMENTATION

Implementation chapter explains the implementation process of the solution developed in the design chapter. This chapter explains the implementation environment, used software tools and technologies.

CHAPTER 5 – RESULTS AND EVALUATION

The evaluation is performed to measure how far the developed solution meets its expected requirements. This chapter shows how the system was tested with various scenarios and the results of each test. In addition to that the errors found while testing and solutions made for them are explained in this chapter.

CHAPTER 6 – CONCLUSION AND FUTURE WORK

As the last chapter of the thesis, this chapter will include the critical evaluation of the proposed approach and suggestions for any future enhancement.

CHAPTER 2 LITERATURE REVIEW

2.1 INTRODUCTION

IoT networks and their applications have received a great deal of attention in recent years. IoT networks are employed in many civilian applications such as transportation, agriculture, environmental monitoring and home automation. This technology has dramatically changed the home and industrial processes. However, along with the exponential growth of IoT applications, security threats have also been raised and pose a more serious threat to privacy than ever before. Most of the enterprises and individuals lack the awareness of security, and manufacturers are more focused on implementing core functions of their products while ignoring security. This chapter explains the technologies that can be used to develop a more secure IoT framework. Many researchers have focused on investigating new techniques and improving existing solutions to secure IoT networks after understanding the importance of security. This chapter also provides details about approaches proposed to secure IoT networks and related existing work.

2.2 SECURE SYSTEMS

With the widespread of computerized systems and data communication networks, they are also increasingly becoming a target for attacks. Standard computing devices such as computers and mobile phones are protected using special hardware components, software, and domain-specific policies, against unauthorized access and modifications. Overall, these measures are designed to limit the exposure of the protected system to authorized users and to perform certain actions when an intrusion is detected or during a violation of the defined security policy. Effective security solution protects the valuable resources from verity of threats associated with unauthorized access, use, disclosure, disruption, modification, or destruction with minimum impact to the usability and the system performance. Any successful deployment of security solution should provide a balanced protection against possible treats without degrading the productivity of system users. The process which defines security controls in order to protect information systems is referred to as information security management. This process involves identification of assets, threats, known vulnerabilities, potential impacts, and possible controls, followed by an assessment of the effectiveness of the applied solution.

All security measures deployed by such a solution can be categorized into three goals. These goals serve as the basis for designing security measures and must be considered during the planning phase.

1. Confidentiality
2. Integrity
3. Availability

The above goals are also identified as three fundamental principles of security and commonly referred to as CIA triad. Figure 2.1 shows a graphical representation of CIA triad.

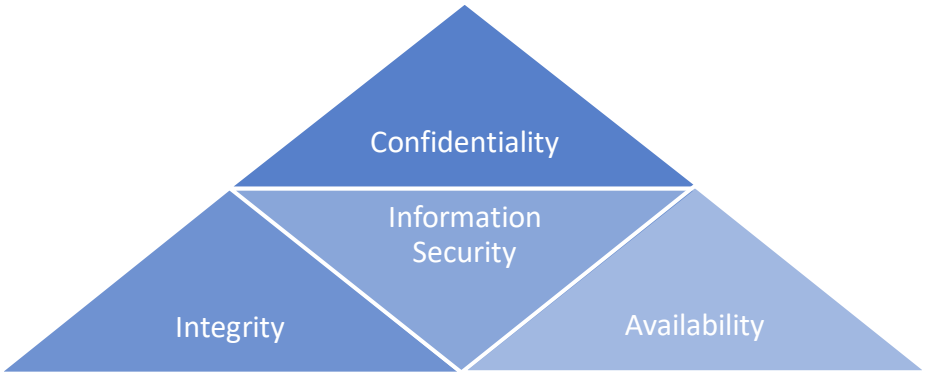


Figure 2.1: Security Triad

Any organization or system can stay protected by identifying the reasons for security breaches that can occur in any of the above goals and employing countermeasures to minimize the possibility of identified malicious actions while mitigating the risk associated with a security attack. These issues could also include natural disasters, hardware malfunctions and physical theft of devices. The deployed security measures may apply multiple layers of defenses each implementing different policies and controls to ensure that the system is fully secured against all the identified threats. The level of protection required for each category does depend on the application, its operating environment, and the business use cases. There are several standards and policies, developed by academics and professionals to guide the information security management process. These standards enforce laws and regulations that control how sensitive information should be accessed, processed, stored, and exchanged between different parties.

2.3 CONFIDENTIALITY

In today's world, information always has a value. Confidentiality is a property of data or information system, which ensures that data is not disclosed or made available to unauthorized entities. In most cases, a failure to maintain the confidentiality of information means that some external people have managed to access it, through intentional behavior or by an accidental event and such a failure of confidentiality typically cannot be undone. Hence, protecting information is a critical task in designing a secure system. The security measures deployed to achieve confidentiality property of an information system are designed to prevent sensitive information from leaking into the hands of wrong people at the same time allowing the authorized people to use it whenever they need to. Confidentiality can be also considered as a component of privacy. In most of the cases, confidentiality is achieved by categorizing information into several different levels based on their importance, sensitivity, and the impact the information leakage can cause to the system. Once categorized, access rules are defined to allow certain authorized users to access the information inside each category. This will make sure that, the only the right people have access to the right amount of data, they need to perform their tasks.

Data encryption which transforms data into another format called ciphered data is a common method of protecting confidentiality. Currently, encryption is one of the most effective methods employed by organizations and information systems, in order to protect data. Figure 2.2 shows the encryption process.

Figure 2.2: Data Encryption Process

Once encrypted using an adequately secure method, the source data which is referred as plaintext cannot be retrieved by an unauthorized user even, they gained access to the encrypted data. Mainly there are two types of encryption algorithms used today.

1. Asymmetric encryption (Public key encryption)
2. Symmetric encryption

2.3.1 Asymmetric Encryption

Asymmetric encryption algorithms which are also known as public key encryption algorithms use two different keys to perform encryption and decryption. The Keys used by asymmetric key algorithms are referred as public key and private key. A plaintext message encrypted using public key can be decrypted only by using its corresponding private key and a plaintext message encrypted using private key can be decrypted only by using its corresponding public key. In these algorithms, the security of the public key does not matter, and it can be freely distributed among the senders who want to send messages to the receiver without compromising security. However, the private key must be kept secret as anyone with the private key can decrypt messages encrypted with the public key.

Asymmetric encryption/decryption process is shown in the Figure 2.3.

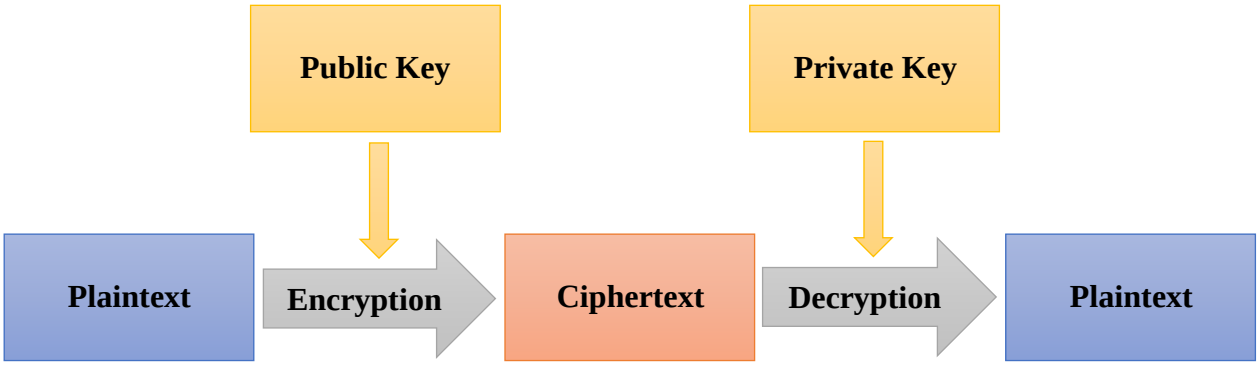


Figure 2.3: Asymmetric Encryption / Decryption Process

Public key algorithms are widely used in modern crypto systems and following are some popular algorithms.

- RSA (Rivest–Shamir–Adleman)
- DSA (Digital Signature Algorithm)
- Elliptic curve cryptography algorithms
- PKCS (Public Key Cryptography Standards)
- ElGamal

RSA Algorithm

RSA algorithm which is the most widely used public key cryptography algorithm is developed based on the difficulty of factorizing large integers. In RSA algorithm, the public key consists of two numbers where one number is the multiplication of two large prime numbers, and the private key is also derived from the same two prime numbers. The strength of the RSA encryption depends on the key size and can be increased exponentially by multiplication of the key size. Keys generated using RSA algorithm are typically 1024 or 2048 bits long.

Steps

1. Generate two large prime numbers: p and q (approximately equal size)
    - n = pq, n is the required bit length (e.g. 1024 bits)
2. Calculate n = pq and $\phi = (p-1)(q-1)$.
3. Select an integer e in range $1 < e < \phi$, such that $\gcd(e, \phi)=1$.
4. Calculate the secret exponent d in range $1 < d < \phi$, such that $ed \equiv 1 \bmod \phi$.

- (n, e) – The public key
- (d, p, q) – The private key

RSA algorithm is used in digital signatures to verify the authenticity of messages. However, it is less efficient than symmetric key algorithms when encrypting large messages.

DSA Algorithm

DSA algorithm was proposed by the National Institute of Standards and Technology (NIST) in August 1991 for use in their Digital Signature Standard (DSS). The algorithm is based on the algebraic properties of the modular exponentiations, together with the discrete logarithm problem which is computationally intractable.

Elliptic Curve Cryptography Algorithms

Elliptic-curve cryptography (ECC) is based on the algebraic structure of elliptic curves over finite fields. ECC can provide equivalent security to other non-EC cryptography algorithms by using smaller keys. For example, a 256-bit key in ECC can offer about the same security as 3072-bit key using RSA. Due to this fact, ECC is more suitable for resource constrained devices.

2.3.2 Symmetric Encryption

Symmetric-key algorithms use the same key which is a shared secret between two or more parties for both encryption of plaintext and decryption of ciphertext. This requirement that both parties have access to the secret key is one of the main drawbacks of symmetric key encryption, in comparison to asymmetric encryption algorithms.

There are two main types of symmetric encryption algorithms.

Block algorithms - This type of algorithms divides data into blocks of specific size and then performs the encryption for each block separately.

Stream algorithms - This type of algorithms encrypts data as a stream of bits rather than dividing it into blocks before performing encryption.

Symmetric encryption process is presented in Figure 2.4.

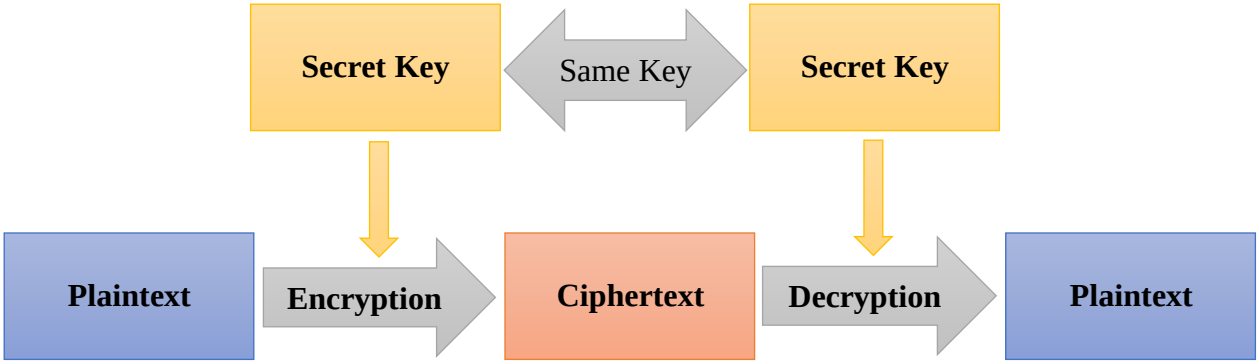


Figure 2.4: Symmetric Encryption / Decryption Process

Symmetric encryption algorithms are widely used in modern cryptosystems and following are some popular algorithms.

- Blowfish
- AES
- DES
- RC4 / RC5 / RC6

2.4 INTEGRITY

Data integrity can be defined as the assurance of the accuracy and consistency data over its entire lifecycle from data creation, communication, processing, and storage. The overall objective of data integrity is ensuring the data is the same as it was when originally recorded. Data integrity can be compromised due to various reasons such as:

- Viruses/malware, hacking, and other security threats
- Due to Malicious or unintentional modifications
- Data transfer errors and hardware failures
- Physical compromise to devices

Data which are modified unintentionally has a little use to the users. Hence data integrity is a core focus in designing any system, which stores, processes, or exchange data. There are several techniques employed by modern systems to ensure data integrity.

Access Control : This also includes limiting physical access to data as if users cannot access data, they cannot change it.

Data Auditing : Frequent data audits are important to ensure that data remains unchanged.

Hash Functions : Hash Functions generate a unique representation of data, which can be used later to compare the states of the data set.

Digital Signature : Like checksum functions, digital signatures create a one-way hash generated using a private key.

Backups : Backup process can be explained as keeping multiple copies of the data and having duplicates is never a bad idea as there is a possibility in failure of even backup media.

Maintenance : In most of the cases, physical device failures can be identified or expect before a complete failure happens. IT professionals can monitor and upgrade disks to ensure that they are operating as they should.

2.4.1 Cryptographic Hash Functions

Cryptographic hash functions accept an input of any length and produce an output of a fixed size called digest. They are designed to be one-way functions, so that, once the hash is generated it is infeasible to revert the result and get the original data back. Hash functions can be used to verify the data integrity and the identification of the sender or source of data.

Figure 2.5 shows the high-level process of generating a hash code using a hash function.

Figure 2.5: Hash Function

An ideal hash function will have following features.

Deterministic : The same messages always result in a similar hash

Quick : The hash value for any given message can be computed in short time.

One-way : It is impossible to generate the original message from its hash value except by trying all possible messages.

Change Distribution : A small change to a message should change the hash value completely so that the new hash value appears to be uncorrelated with the old hash value.

Unguessable : It is infeasible to find two different messages with the same hash value.

Cryptographic hash functions have many applications in information security context and following are some of them.

- Data integrity verification
- Digital signatures
- Message authentication codes (MACs)
- Authentication
- Data indexing
- Duplicate data detection

Following sections describe two common popular hash functions, which are in use today.

MD5 [1]

MD5 hash function produces a digest of 128 bits (16 bytes) and typically expressed as a 32-digit hexadecimal number. MD5 has been used in a wide variety of security applications and is also popular to check the integrity of files. However, later it has been found that MD5 hash function has certain flaws, which make is less useful as a cryptographic hash function. It can still be used as a checksum to verify data integrity of files but only against unintentional corruption.

e.g.: Plain text : This is a test.

MD5 : 120EA8A25E5D487BF68B5F7096440019

SHA [2]

The Secure Hash Algorithms are a family of cryptographic hash functions published by the US National Institute of Standards and Technology (NIST). There are hash algorithms in SHA family.

SHA-1 : A 160-bit hash function designed by the National Security Agency (NSA) to be part of the Digital Signature Algorithm. This algorithm was no longer approved for most cryptographic uses after 2010 due to cryptographic weaknesses.

SHA-2 : Includes two different algorithms, known as SHA-256 and SHA-512 each having different block sizes. They differ in the word size; SHA-256 use 32-bit words where SHA-512 uses 64-bit words. This algorithm works by first splitting data into pieces of 512 bits (64 bytes) and then producing its cryptographic "mixing" to finally issue a 256-bit hash value. SHA-256 is one of the strongest hash functions available has not yet been compromised in any way.

e.g.: Plain text: This is a test.

SHA-256: a8a2f6ebe286697c527eb35a58b5539532e9b3ae3b64d4eb0a46fb657b41562c

SHA-3 : A hash function formerly called Keccak, chosen in 2012 after a public competition among non-NSA designers. It supports the same hash lengths as SHA-2, and its internal structure differ significantly from the rest of the SHA family.

2.5 AVAILABILITY

Availability is the assurance that information system and data are accessible to authorized users when it is needed. It is important to ensure the computing systems used to process and store the information, and the communication channels used must be functioning correctly to achieve availability property of information systems. The information systems which are designed to improve availability are called as high availability systems. Those systems employ proactive methods such as redundancy, failover, RAID, and high-availability clusters to mitigate effects on system outages occur due to both intentional and unintentional malicious activities.

Following are some of the intentional attacks.

- Physical attacks – In this type of attack, the attackers gain physical access to the information system and open the system to variety of ways for hacking.
- DoS and DDoS attacks - In this type of attack, the information system and its services become unavailable to the authorized users, or they may experience a severe degrade of system performance. This attack is mostly accomplished by flooding the target machine with the service request and due to this, the targeted system gets overload and fails to respond to other users.
- SYN flood attacks – This is a type of Denial of Service attack that exploits the portion of the three-way handshake process of TCP protocol to make the targeted information system inaccessible.

Data redundancy is important when planning to minimize the system down time. Data redundancy is achieved by replicating data into multiple nodes in the network and allowing IoT devices to connect to more than one node. When a single node has been failed, the IoT devices can connect to another node and continue to operate as the data generated by the failed IoT device is replicated into the other nodes.

2.6 BLOCKCHAIN

A blockchain is a growing list of blocks, which are linked by each block storing the hash value generated for the previous block in the chain. Blockchain network is a peer to peer network with no central authority to manage the stored data or enforce rules. The network is maintained by many independent users refer to as nodes. Blockchain can also be considered as a new approach of developing distributed databases and there are many applications. Blockchains are so popular because of their immutable property, which is once data are written to the blockchain, it is nearly impossible to modify or delete it. At a minimum, all blocks in a blockchain contain, hash value of the previous block, timestamp and data.

2.6.1 Public Blockchains

Public blockchains are large distributed network of independent nodes, which are open for anyone to participate at any level. Most of the public blockchain implementations are open-source projects. The public blockchains are very popular due to their transparency and the involvement of participants to contribute the growth of the blockchain. The public blockchain networks typically have incentive mechanisms to encourage more participants to join the network. These networks can operate seamlessly in trustless networks due to their immutable nature of the records. Each node in a public blockchain, must solve a complex resource-intensive cryptographic problem called a proof of work (PoW) to ensure that recorded transactions are non-editable.

Public blockchains are widely used in crypto currency projects to create an unchangeable ledger of transactions. However, one of the main drawbacks of public blockchains is their PoW approach, which is very expensive in terms of resources required for calculations and time required for miners to solve the puzzle. For example, currently the average time required for a new block creation in popular Bitcoin network is 10 minutes and difficulty level of mining has been adjusted to maintaining this time. However, due to the increased difficulty and competition among minors, the task of adding a new block to the network became several times expensive than before. Due to this disadvantage, public blockchains are only suitable for applications where a large number of users participate in maintaining the blockchain.

e.g.: Bitcoin, Ethereum

Bitcoin

Bitcoin, a form of electronic cash, is a decentralized digital currency without a central bank. The currency can be exchanged between on a peer-to-peer bitcoin network without the need for a central authority or administrator. The transactions made using Bitcoins are recorded in a public blockchain. The Bitcoins are created as rewards for the process known as mining and can be exchanged for other products, and services. In Bitcoin network, the nodes can validate transactions and add them to their copy of the ledger while broadcasting the update to other nodes.

In Bitcoin blockchain, each block contains a SHA-256 hash of the previous block which is used to link blocks. A new block added to the blockchain will be accepted only if the block contains a proof-of-work (PoW) which is a system based on Hashcash. The PoW algorithm used in Bitcoin requires miners to find a number called a nonce, such that when the block content is hashed along with the nonce, the result is numerically smaller than the network's difficulty target. This proof can be validated by any node in the network, but extremely difficult to generate. The minors must try many different nonce values before meeting the difficulty target.

Ethereum

Ethereum is an open-source pubic blockchain platform which supports smart contract (scripting) functionality. When compared to Bitcoin, which is a peer to peer electronic cash system that enables online payments, the Ethereum blockchain focuses on running the programming code of any decentralized application. In Ethereum network, Ether is a token generated by the Ethereum platform which can be transferred between accounts and used to compensate participant mining nodes for computations performed. In the Ethereum blockchain, the miners work to earn Ether, which is a type of crypto token that fuels the network. Moreover, in Ethereum platform, there is a second type of token called gas that is used to pay miners fees for including transactions in their block, and every smart contract execution requires a certain amount of gas to be sent along with it to put it in the Ethereum blockchain. Ethereum platform also provides a decentralized virtual machine, the Ethereum Virtual Machine (EVM), which can execute scripts using a network of public nodes.

2.6.2 Permissioned Blockchains

The permissioned blockchains limit the parties that can perform certain transactions on the network. Each participant belongs to a set of roles, which defines the access rights of the corresponding user. The blockchains maintain an access control layer to allow certain actions to be performed only by certain authenticated users. The permissioned blockchains may or may not require proof of concept or some other requirement from nodes. This type of blockchains is a popular among industrial or business applications.

E.g.: Ripple

In general, permissioned blockchains have following advantages when compared to public blockchains.

Efficiency : Permissioned blockchains are more efficient and have better performance when compared to public blockchain networks. In permissioned blockchains, not all nodes are doing redundant validations of all blocks in the network and permissioned blockchains only require its member nodes to validate transactions.

Control : Nodes in a permissioned blockchains have pre-defined authorities in the network which is different from a public blockchain which allows any node to participate in any authorization level.

2.6.3 Private Blockchains

Blockchains were originally intent to be public networks, which are open to any user. The public blockchains can be considered as the opposite of private networks, which are strictly controlling the access to the network. Private blockchains are relatively smaller in size than the other two types of blockchains and have strict control over the authorization. The users in a private blockchain cannot read, write, or validate the blockchain if they do not have necessary permissions. Private blockchains are introduced to solve the problems identified in public blockchains and expand the scope to business applications.

e.g.: Hyperledger

Table 2.1 explain the differences between public and private blockchain networks.

Public Blockchains	Private Blockchains
Anyone can run a node and has access to all data in the network.	Access to blockchain network is restricted.
Anyone can perform transactions on the network.	Not all public users can perform transactions on the network.
Anyone can validate the blocks in the network.	Only the authorized uses can validate the blockchain.
Cost of a transaction is high.	Cost of transaction is low.
Slower and it takes a long time to approve and add a new block to the blockchain.	New blocks can be added to the blockchain almost instantly.
Nodes in the network can be independent and requires no trust.	The nodes in the network are known to each other.

Table 2.1: Difference between public and private blockchains

2.7 RELATED WORK

Motivated by the increased number of vulnerabilities, attacks, and information leaks, IoT device manufacturers and researchers have carried out experiments to propose solutions to secure IoT networks and analyze the effectiveness of existing solutions. The following sections present a summary of the work which has been done in related fields.

2.7.1 The Effect of IoT New Features on Security and Privacy: New Threats, Existing Solutions, and Challenges yet to be Solved [3]

In their research, Zhou, Wei, et al. have proposed the concept of "IoT features" and then analyzed the security and privacy effect of eight IoT features, which have most impact on security for the threats they cause, existing solutions and unsolved challenges. "IoT features" refer to the unique features of IoT devices and their applications, which are different from the applications of the Internet and computers. After defined "IoT features" they have analyzed each feature to find treats, challenges, solutions, and opportunities. They have followed the up-to-date work in the field of IoT and identified challenges while pointing out the need for further security improvements.

Furthermore, they have suggested that researchers need to discover the root causes behind new security threats and design more generic and practical protective measures.

The summary of the research is presented in Table 2.2.

Feature	Threat	Challenge	Opportunity
Interdependence	Bypassing static defenses, Overprivileged	Bypassing static defenses, Overprivileged	Context-based permission
Diversity	Insecure protocols	Fragmented	Dynamic analysis simulation platform, IDS
Constrained	Insecure systems	Lightweight defenses and protocols	Combining biological and physical characteristics
Myriad	IoT botnet, DDoS	Intrusion detection and prevention	IDS
Unattended	Remote attack	Remote verification	Remote attestation, Lightweight trusted execution
Intimacy	Privacy leak	Privacy protection	Homomorphic encryption, Anonymous protocols
Mobile	Malware propagation	Cross-domain identification and trust	Dynamic configuration
Ubiquitous	Insecure configuration	-	Safety consciousness

Table 2.2: Threats, Challenges and Opportunities of IoT Features

2.7.2 Survey of Security and Privacy Issues of Internet of Things [4]

Borgohain, Tuhin, Uday Kumar, and Sugata Sanyal have conducted a general survey of security issues of IoT devices along with an analysis of the privacy issues that an end-user may face when consuming applications of IoT. Most of the survey has focused on the security loopholes arising out of when IoT devices exchange information. First, they have discussed about various communication technologies using the Internet infrastructure for the exchange of information and then analyzed each method to identify privacy issues faced by the end users of each technology. After analyzing IoT from a security point of view, they have suggested that proper security measures must be taken in the initial phase itself before going further development of IoT for effective and widely accepted adoption.

2.7.3 Security of IoT Systems: Design Challenges and Opportunities [5]

In their research, they have mentioned, security is the most important requirement for the widespread use of IoT networks. They have identified that optimization intensive CAD techniques compounded with their traditional accurate modeling are naturally suited to enable the design of highly optimized IoT devices. They have considered energy and security as two main constraints for IoT devices and described how both constraints can be addressed well using CAD techniques.

2.7.4 New security architecture for IoT network [6]

This research identifies, traditional security mechanisms like firewalls, intrusion detection and prevention systems which are deployed at the Internet edge to protect the network from external attacks are no longer enough to secure the next-generation Internet and the border less architecture. The applications of IoT raises additional concerns over network access control and software verification. Recent advances in computer networking have introduced a new technology paradigm for future communication, which is Software Defined Networks (SDN) where a central software program, called SDN controller, manages the overall network behavior. Based on the SDN architecture, they have proposed a security model for the IoT which is designed to establish and protect both wired and wireless network infrastructure. Then they have extended the proposed architecture in order to include Ad-Hoc networks and network object things such as: sensors, tablets, smart phones, etc.

2.7.5 Security in the Internet of Things: A Review [7]

In their research, they have identified security and privacy as the key issues for IoT applications, which are still facing some enormous challenges. In order to facilitate this emerging domain, they have reviewed the research progress of IoT while paying attention to the security. By means of deeply analyzing the security architecture and features they discussed the research status of key technologies including encryption mechanism, communication security, protecting sensor data and cryptographic algorithms, and identified challengers in following areas.

1. Security Structure
2. Key Management
3. Security Law and Regulations
4. Requirements for Burgeoning Applications

2.7.6 Network Level Security and Privacy Control for Smart Home IoT Devices [8]

The increasing uptake of smart home appliances, such as lights, smoke-alarms, power switches, baby monitors, and weighing scales, raises privacy and security concerns at an unprecedented scale, allowing legitimate and illegitimate entities to snoop and intrude into the family's activities. In this paper, they have first illustrated these threats using real devices currently available in the market. Then they have argued that as more such devices emerge, the attack vectors increase, and ensuring security of the house become more challenging and advocated that device-level protections be augmented with network-level security solutions, that can monitor network activity to detect suspicious behavior. They have further proposed that software defined networking technology should be used to dynamically block devices, based on their activity and on the context within the house such as time-of-day or occupancy-level. They have prototyped their solution using open-source SDN platforms and evaluated its efficacy in protecting multiple smart-home devices.

2.7.7 Security Challenges in the IP-based Internet of Things [9]

The direct interpretation of the term Internet of Things refers to the use of standard Internet protocols for the human-to-thing or thing-to-thing communication in embedded networks. In this paper, they have discussed the applicability and limitations of existing Internet protocols and security architectures in the context of the Internet of Things. First, they have given an overview of the deployment model and general security needs and then presented challenges and requirements for IP-based security solutions highlighting specific technical limitations of standard IP security protocols. As a conclusion, they have mentioned that the security architecture proposed for IoT should fit the capabilities of the thing, and security protocols should further consider the resource-constrained nature of things and heterogeneous communication models. The security protocols should include lightweight security mechanisms that are feasible to be run on small things. They have also mentioned that the group security must be considered as well, since the IoT brings communication patterns that are unusual in traditional networks, and thus are not sufficiently supported by end-to-end Internet security protocols. The protocol design should further consider the effect of packet fragmentation on security, with focus on possible DoS attacks.

2.7.8 Towards an optimized blockchain for IoT [10]

This research proposes a lightweight Bitcoin based architecture for IoT that virtually eliminates the overheads of classic blockchain, while maintaining most of its security and privacy benefits. They have highlighted that IoT devices can benefit from a private immutable ledger managed centrally, to optimize energy consumption. In this solution, high resource devices are creating an overlay network to implement a publicly accessible distributed blockchain that ensures end-to-end security and privacy. The proposed architecture uses distributed trust to reduce the block validation time. They have explored the feasibility of their approach in smart home applications as a representative case study for broader IoT applications. The architecture is evaluated under common threat models highlighting its effectiveness in providing security and privacy for IoT applications. Furthermore, they have demonstrated that their method decreases packet and processing overhead significantly compared to the blockchain implementation used in Bitcoin.

2.7.9 IoTChain: A Blockchain Security Architecture for the Internet of Things [11]

This research has defined IoT as the integration of Internet Protocol (IP) enabled constrained devices with the existing Internet infrastructure and proposed IoTChain as a combination of Object Security Architecture for the Internet of Things (OSCAR) architecture and ACE authorization framework to secure access to IoT resources. IoTChain consists of two components, an authorization blockchain based on the ACE framework and the OSCAR object security model extended with a group key scheme. In their approach, a trust-less authorization blockchain has replaced the centralized authorization server in the ACE, and OSCAR used the public ledger to set up multi-cast groups for authorized clients. They have we have implemented the authorization blockchain on top of a private Ethereum network to evaluate the feasibility of the solution and reported on several experiments that assess the performance of different architecture components.

CHAPTER 3 DESIGN OF SOLUTION

3.1 INTRODUCTION

As mentioned in the first chapter, the main objective of this project is to design and implement a blockchain based protocol for IoT device authentication and secure communication. To achieve the objectives of the research a compressive study has been conducted. After the study, the most suitable technologies to build such a platform under the constraints of IoT devices are identified. The proposed solution utilizes a combination of techniques, which are selected from the fields of computer security, web API development, and hardware programming. This chapter describes the techniques used in the proposed solution as well as their appropriateness to achieve the objectives of the project.

3.2 DEVELOPMENT METHODOLOGY

The process of developing a software solution to a problem can be performed according to software process models, which describe approaches to a variety of tasks that take place during the process. The waterfall model is one such software process model in where the activities are performed as a linear sequence of phases where each phase depends on the previous phase. In contrast the iterative process model encourages the design, develop and test of features in iterations. The iterative development methodology was chosen to develop the solution mainly by considering its flexibility. One of the underlying principles of iterative development is that the solution can be refined according to the feedbacks received while evaluating the intermediate solutions. Further details about software process models and the reason for selecting iterative development methodology is explained in Appendix B.

3.3 SOLUTION BREAKDOWN

This research suggests a blockchain based approach to secure IoT networks by considering the common constraints of IoT networks. Many IoT devices perform critical and real time operations, which expects high availability of the device. The embedded processors on IoT devices have limited processing power and memory to implement conventional authorization protocols and protection software. In any usable IoT security solution, the security features implemented on the device should have minimum interruption to the device operation. The main objective of this project is decomposed into several sub problems as network architecture, device authentication, secure data storage, and control API. After developing sub problem solutions individually, they are linked together to form the final solution.

Following sections describe the overview of the proposed solution.

3.3.1 Network Architecture

When designing a solution to secure IoT networks, it is necessary to have a better understanding about IoT network architecture. Network architecture varies in both topology and architectural design, depending on the application. Most of the current IoT networks are implemented using the aggregator topology where many small IoT devices or sensor nodes are connected to an aggregator device, which is responsible for communicating with the connected nodes. In most cases, IoT devices connect to the aggregator device via wireless mediums such as Wi-Fi. The connected devices may perform a verity of tasks, which sometimes require intercommunication between devices in the same network. The aggregator device has enough processing power to collect information from the child IoT nodes and to perform application-based processing on the collected data. The aggregator is also capable of communicating with other aggregators as well as communicating with internet services to perform further processing.

Figure 3.1 shows the aggregator topology for IoT network.

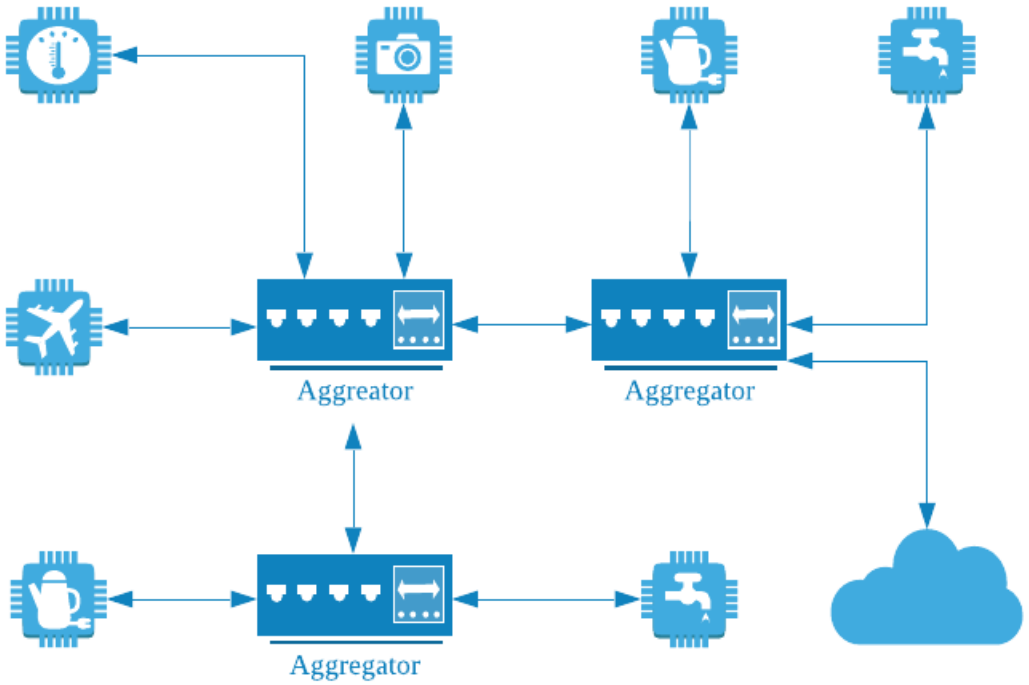


Figure 3.1: Aggregator Topology for IoT Network

In the proposed approach, the aggregator devices perform resource-consuming tasks such as maintaining the blockchain and hosting the control interface. In addition to the traditional aggregator function layer, a new hub function layer is added to facilitate establishing connections to more than one aggregator device. An IoT device or an aggregator may use the hub function to perform important operations such as authentication. Once the hub has received a request, it will forward the request to multiple other aggregators for collecting their responses. The aggregators to send the request are selected randomly from the list of available aggregators. Once all the responses are collected, the hub function compare the responses to find discrepancies between the received responses. The response which is agreed by most of the other aggregators will be sent to the service requester.

3.3.2 Data Storage

Blockchain stores immutable records of device identifications and data generated by the devices. Any device in the network can ensure the integrity of information by validating the blockchain. However, only the authorized devices can access the encrypted information. Each block in the

blockchain contains multiple fields, which are created to maintain the blockchain and for easy navigation between blocks.

The Table 3.1 describes the purpose of each field stored in a block.

Field	Description
Timestamp	Time zone independent UNIX timestamp of block added time.
Type	System has three predefined types: 1. AUTH – Blocks containing device authentication details. 2. CERT – Blocks containing public keys and device identifiers. 3. DATA – Blocks containing data records 4. UPDATE – Blocks containing device update information. Creation and use of custom block types are also supported.
Owner ID	Identifier of the aggregator added the block.
Hash	SHA-256 hash code generated for the block.
Previous Block Hash	Hash code of the previous block.
Data	Data stored in the block.

Table 3.1: Fields in a Block

Figure 3.2 shows the structure of the blockchain proposed in the solution.

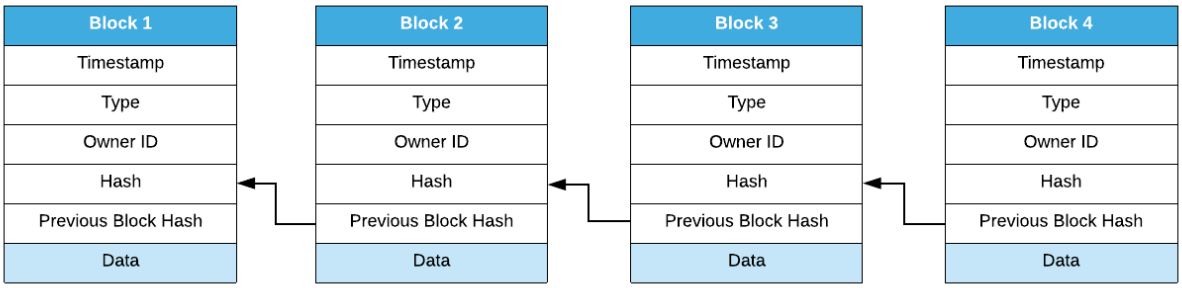


Figure 3.2: Structure of the Blockchain

Data added to a block is encrypted by the aggregator based on the request received from the provider. If the data stored in the block found to be encrypted, the aggregator will decrypt the data before transferring to the requester. However, some data such as public keys are stored as plain-text in the blockchain.

3.3.3 Device Authentication

One of the most important challenges that IoT networks face is authentication and verification of connected devices. The device authentication is performed in a decentralized way by involving several other randomly selected aggregators. The authenticity of the IoT device and the aggregator is validated before establishing the connection. The hub function will select a subset of the other aggregators to validate the response. Hence compromising several nodes in the network will not grant unauthorized access to the network. Figure 3.3 shows the sequence diagram of the authentication flow implemented in the solution.

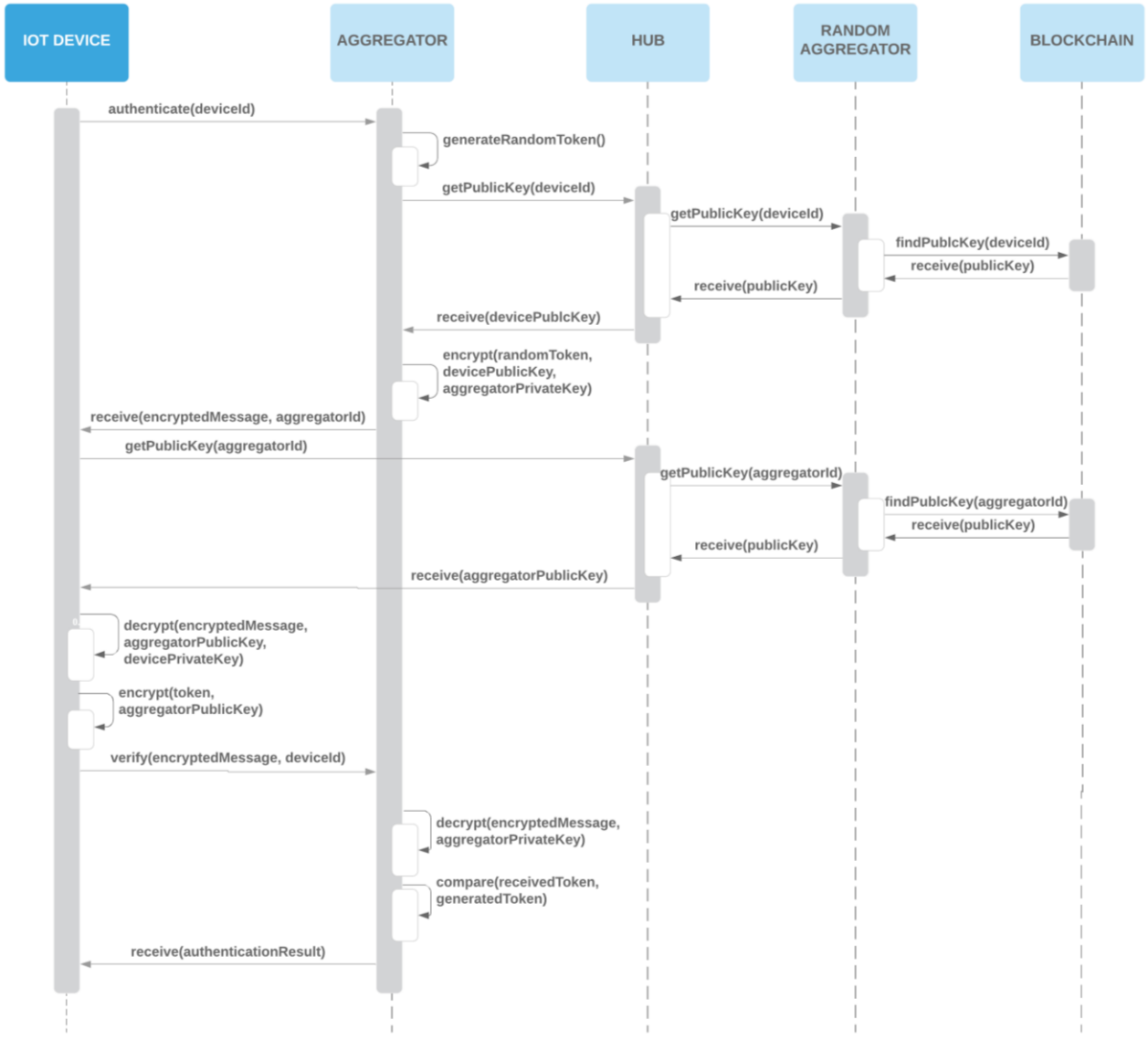


Figure 3.3: Sequence diagram for device authentication

3.3.4 Control API

All authorized devices, and user applications are interacting with the blockchain through the control API. Furthermore, the control API allows to perform operations through the hub layer to ensure that the provided information is accurate.

In general, the control API provides the features described in Table 3.2 to the consumers.

Feature	Description
Device authentication	The control API performs the device authentication according to the rules discussed in the previous section.
New device registration	The control API provides facilities to securely enroll new devices to the network. At the time of enrollment, a unique identifier and a public key/private key pair will be generated for each device. The generated identifier and the public key are stored in the blockchain to facilitate authentication of the newly created device.
Data storage	Devices will access the control API to store data. The device can decide whether to store data in plaintext format or in the encrypted format.
Data retrieval	Devices obtain data from the blockchain through control API. However, the node will gain access to the data only if data is not encrypted or the consumer possesses the correct private key to decrypt data.
Blockchain validation	The control API will provide the real time validation status of the blockchain by collecting information from multiple other nodes.
IoT network status checking	The control API provides an overview of the other registered devices found in the platform and provides facilities to manage registrations.

Table 3.2: Control API Features

3.3.5 System Update Distribution

Details about IoT system updates are stored in a special block of type “UPDATE”. Device specific data block which contains information about the system update is included in all device update blocks. At minimum, following information is stored in update data.

- Applicable device type
- System version information
- Hash code of the system update file
- Published location

The update checking frequency is decided by the IoT device. The devices can query for the matching updates and use the provided information in data field to download the system update and verify the integrity. The device will receive only the updates published by its owner. The owner is the aggregator which performed the device registration. Since the data stored in the blockchain is immutable, the IoT devices can guarantee that the update has not been modified in a malicious way. System updates usually contain large binary files and it is advisable to publish update files to an external location instead of storing them in the blockchain.

Figure 3.4 summarize the system update process of an IoT device.

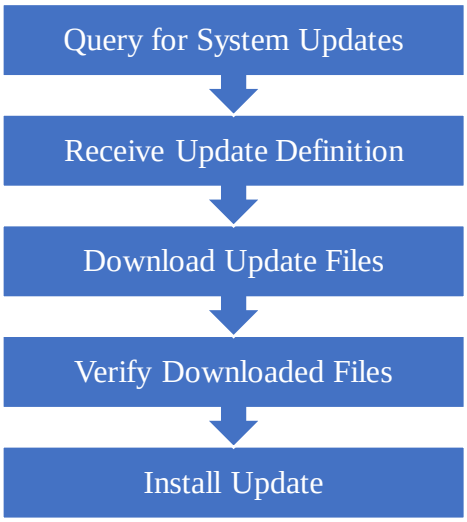


Figure 3.4: System Update Process

CHAPTER 4 IMPLEMENTATION

4.1 INTRODUCTION

Implementation process converts the designed solution into the actual system by developing the application programs. The output of this phase is a complete working solution, which consists of executable programs and hardware components developed to fulfill the requirements identified in the previous sections. In this phase, an appropriate programming language, hardware, and software-development tools were chosen to implement the solution that is designed in the previous phases. Furthermore, this phase relies on a mixture of technologies and design patterns, which are precisely chosen to fulfill the requirements efficiently.

4.2 IMPLEMENTATION ENVIRONMENT

The implementation environment of the solution is planned carefully to achieve the maximum outcomes while taking some important factors like cost, robustness, reliability and performance into consideration. Primarily the implementation environment is where the aggregator program and hub program operate. Moreover, the requirements for the implementation environment can be separated into software requirements and hardware requirements. The detail of the implementation environment is described in the following subsection.

4.2.1 Aggregator Environment

The web application of the implemented solution is hosted in the aggregator environment. The aggregator environment is planned to handle the communication links between the IoT devices and the blockchain in real-time. IoT devices mainly communicate with the aggregator device to retrieve, process and store data. Furthermore, the aggregator device act as a distributed streaming platform to synchronize data with other aggregator devices maintaining the blockchain. Each aggregator device hosts its own copy of the blockchain which is verified with other devices periodically to ensure the data integrity.

Table 4.1 summarize the minimum system requirements for the aggregator environment.

Hardware	Software
700 MHz processor 256 MB RAM 500 MB storage Network connectivity	Any Java supported operating system. Java Runtime Environment 8 or later.

Table 4.1: Aggregator Environment Minimum System Requirements

4.3 DEVELOPMENT TOOLS AND TECHNOLOGIES

A variety of tools and technologies were used in the development of the solution to save time while satisfying requirements identified during the previous phases. The selected tools and technologies are described in this section.

4.3.1 Development Tools

Following software tools were used to facilitate the rapid development of the system at the same time preserving the quality of the program language code produced.

Table 4.2 summarizes the tools used and the purpose.

Tool	Purpose
Java Development Kit	Main platform for aggregator application development.
IntelliJ IDEA	IDE for developing the aggregator and hub programs.
PlatformIO	Platform for developing IoT device program.
Visual Studio Code	Editor used to write IoT device program.
SoapUI	For developing load test projects used during the evaluation.
Git	Used to track the changes during the software development.

Table 4.2: Development Tools

4.3.2 Technologies

The system is developed using a mixture of well-matured and robust technologies, which enables the implementation of required functionalities. The solution is mainly developed using Java [12] language, which is a platform independent, object-oriented programming language developed for general-purpose programming. Java is a platform independent programming language meaning that compiled Java code can run on any platform that supports Java without the need for

recompilation. This solution has selected Java as the main programming language by considering the following features.

- Platform independent
- Object oriented
- Secure
- Complete standard library
- Support for multi-threaded programming

Spring Framework [13] is an open-source modular application framework for Java platform. The core features of the Spring framework can be used to develop desktop applications as well as web-based APIs. The framework uses existing technologies like ORM frameworks, logging frameworks, JEE, JSON, etc. to provide a comprehensive support framework. The Spring framework was selected to develop the proposed solution by mainly considering its ability to integrate with other technologies used for real time communication and features provided to develop RESTful APIs. A JSON format based RESTful API was developed in the solution by considering the wide acceptance of JSON as the main format used in RESTful APIs. RESTful API [14] is an application interface designed using HTTP requests such as GET, POST, DELETE, PUT, etc. to communicate. RESTful APIs can return data in XML, JSON or any other format depending on the context. By using a stateless protocol and standard HTTP operations, RESTful APIs deliver fast, reliable, and extensible APIs developed using components that can be updated independently without affecting the other systems.

The solution stores data in H2 Database [15] which is an open source relational database management system written in Java which can be embedded in Java applications. H2 database supports a subset of the SQL (Structured Query Language).

The main features of H2 are:

- Open source
- Fast JDBC API
- Embedded and server modes with support for in-memory databases
- Browser based console application
- Small footprint: around 2 MB file size

4.4 NETWORK ARCHITECTURE

The network architecture of the solution consists of aggregators, IoT devices and user applications, which are consuming the services provided by IoT network. Each aggregator device in the network has its own embedded database to store a copy of the blockchain. The aggregator device is accessed by many IoT devices at a time. The access rights for IoT devices are granted only after the device registration process and successful authentication. Figure 4.1 shows the network architecture of the solution.

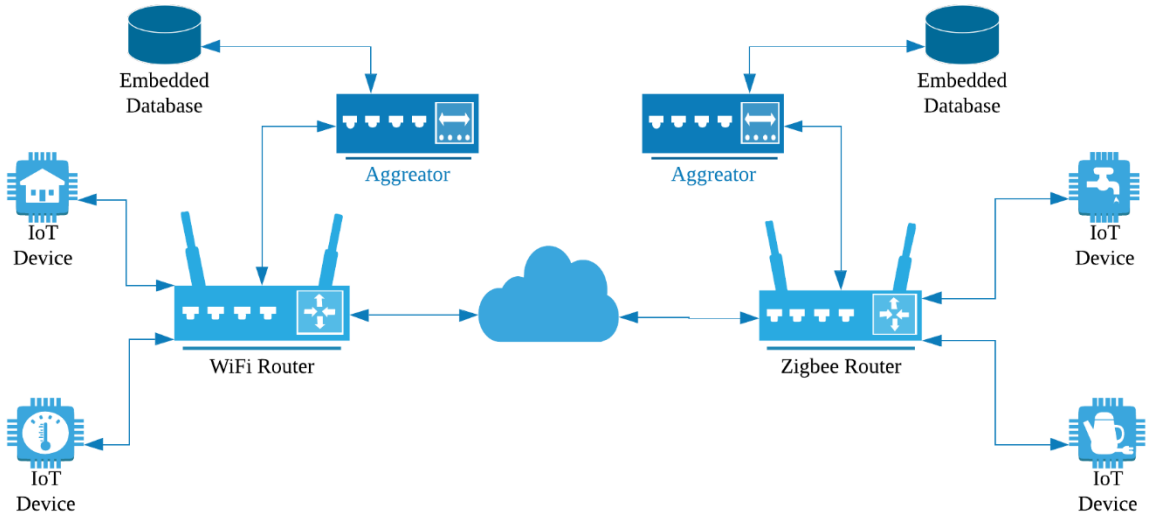


Figure 4.1: Network Architecture of the Proposed Solution

The solution can function independently without relying on any Internet technology. However, the Internet or any other wide area network (WAN) technology can be used to link remote IoT device environments together. In this scenario, the aggregator devices should be able to discover each other to form the required communication links. An IoT device can access any aggregator device available in its sub network for communication. The communications between IoT devices are handled through the aggregator device.

4.4.1 Communication Protocols

All components in the network, including IoT devices and aggregators are linked using a communication medium which supports HTTP protocol. The HTTP protocol is mainly used by IoT devices to access the API provided by the aggregator and the communication between aggregator devices. The implementation supports several popular communication mediums used by current IoT device networks. However, the communication medium used for the solution should have a low latency and adequate bandwidth for the application.

Zigbee [16] [17] is a Bluetooth like protocol, which has wide use in industrial and home IoT applications. Zigbee has several profiles, including ZigBee PRO [18] and ZigBee Remote Control [19] (RF4CE [20]) which are based on the IEEE802.15.4 protocol. Zigbee is operating at 2.4 GHz frequency targeting applications that require relatively infrequent data exchanges at low data-rates over a restricted area and within a 100m range such as in a home or building. ZigBee is popular technology for IoT communications as it has some significant advantages offering low-power operation, high security, reliability, and scalability. Zigbee works in a mesh network and becomes more powerful and stronger as more devices are added. The latest version of ZigBee is 3.0 [21], which is the unification of the various ZigBee wireless standards into a single standard.

- Standard: ZigBee 3.0 based on IEEE802.15.4
- Frequency: 2.4 GHz
- Range: 10-100 m
- Data Rates: 250 kbps

Zigbee IP [22] is an IPv6-based open standard supporting full wireless mesh networking solution and provides seamless Internet connections to control low-power, low-cost devices. It enables low-power devices to participate natively with other IPv6-enabled Ethernet, Wi-Fi and, HomePlug [23] devices. It supports standard Internet protocols, such as IPv6, TCP, TLS and UDP and end-to-end security using TLS1.2 protocol, link layer frame security based on AES-128-CCM algorithm [24] and support for public key infrastructure using standard X.509 v3 certificates and ECC-256 cipher suite.

Wi-Fi [25] is commonly used communication medium supported by the implementation for the wireless local-area networking (WLAN). Wi-Fi is based on the IEEE 802.11 family of standards and a popular choice for IoT device communication due to its wide existing availability in homes.

- Standard: Based on 802.11n
- Frequencies: 2.4 GHz and 5 GHz bands
- Range: Approximately 50 m
- Data Rates: Around 600 Mbps

Wi-Fi HaLow [26] is a technology based on the IEEE802.11ah standard and introduced to address the range and power concerns of IoT devices using Wi-Fi as the communication medium. Wi-Fi HaLow uses the 900 MHz band to provide extended range over a radius of one kilometer with low power requirements and power use is further optimized by using predefined wake/doze periods.

4.5 APPLICATION ARCHITECTURE

There are three main components exits in the developed solution as follows.

1. Device Registration and Authentication Component
2. Communication Hub Component
3. Blockchain Management Component

The IoT devices act as clients and connect to the aggregator server using the provided RESTful API. The aggregator is developed as a web service which can be started as a standalone application and once started publishes a RESTful API. The Blockchain management component uses an embedded database which stores blockchain data. The RESTful API was selected by considering its suitability to develop efficient, scalable, platform-independent applications.

Figure 4.2 shows the composition of the solution.

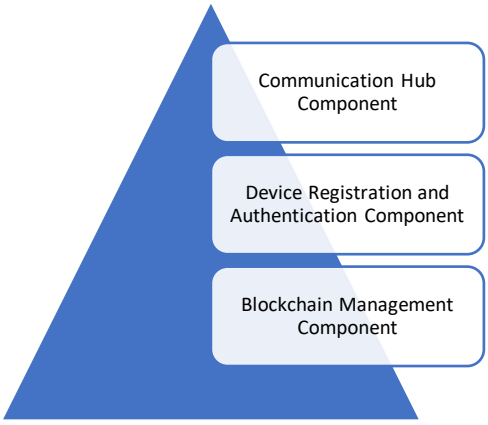


Figure 4.2: Application Structure of the Solution

4.6 FUNCTION IMPLEMENTATION

As presented in methodology chapter, the proposed solution can be broken down into nine subsystems as follows.

1. Blockchain Storage
2. Blockchain Search
3. Blockchain Synchronization
4. Blockchain Validation
5. Device Registration
6. Device Authentication
7. Device Information
8. Hub Function
9. Secure Communication
10. API Playground

The following sections discuss about the implementation details of each subsystem.

4.6.1 Blockchain Storage

The blockchain handling component can receive data from IoT devices or from aggregators in the network. The following API URI is exposed for devices to send data which will be stored in the blockchain.

POST /blocks/save

This URI accepts the following request in the body of the HTTP POST request.

```
{
  "data": "data",
  "deviceId": "d076c6a4-a61a-4eab-a7bc-90deb51e689d",
  "encrypted": true,
  "encryptionKey": "encryption_key",
  "salt": "salt",
  "type": "DATA"
}
```

Blockchain data save request parameters are described in Table 4.3.

Parameter	Description
data	[OPTIONAL] Data to be saved in the blockchain.
deviceId	[REQUIRED] Device identifier of the data producer.
encrypted	[OPTIONAL] If this parameter is true, the data will be encrypted before saving into the blockchain. The default value of this parameter is false.
encryptionKey	[OPTIONAL] The key which is to be used for data encryption.
salt	[OPTIONAL] This parameter will be used in combination with the encryption key to strength the encryption. Adding salt to the encryption key will mitigate the risk of dictionary attacks.

Table 4.3: Blockchain Data Save Request Parameters

If the data has been successfully saved, this URI will return the HTTP status 200 OK.

The data save operation is performed by the API by following steps shown in Figure 4.3.

Figure 4.3: Data Storage Process

Once data received; this API operation first performs the validation of the received data using Hibernate Validator Framework. Then the hash code of last block will be obtained from the blockchain.

Data Encryption

If the encrypt flag is true, the data will be encrypted using 256-bit AES encryption algorithm.

```
public byte[] encrypt(byte[] bytes) {
   synchronized (this.encryptor) {
      byte[] iv = this.ivGenerator.generateKey();
      initCipher(this.encryptor, Cipher.ENCRYPT_MODE, this.secretKey,
            this.alg.getParameterSpec(iv));
      byte[] encrypted = doFinal(this.encryptor, bytes);
      return this.ivGenerator != NULL_IV_GENERATOR ? concatenate(iv,
encrypted)
            : encrypted;
   }
}
```

During the encryption process, a 16-byte initialization vector is generated using the secure random number generation function provided by JDK. The Java implementation provides a cryptographically strong random number generator (RNG) [27].

```
public byte[] generateKey() {
   byte[] bytes = new byte[keyLength];
   random.nextBytes(bytes);
   return bytes;
}
```

Then the cipher is initialized for the generated initialization vector.

```
/**
 * Initializes the Cipher for use.
 */
public static void initCipher(Cipher cipher, int mode, SecretKey secretKey,
      AlgorithmParameterSpec parameterSpec) {
   try {
      if (parameterSpec != null) {
         cipher.init(mode, secretKey, parameterSpec);
      }
      else {
         cipher.init(mode, secretKey);
      }
   }
   catch (InvalidKeyException e) {
      throw new IllegalArgumentException(
            "Unable to initialize due to invalid secret key", e);
   }
   catch (InvalidAlgorithmParameterException e) {
      throw new IllegalStateException(
            "Unable to initialize due to invalid decryption parameter spec",
e);
   }
}
```

Finally, during the encryption process, the actual encryption is performed by calling the *doFinal* method of the initialized *cipher*.

Once the encryption is completed, the following additional parameters are added to the block.

- Timestamp
- Type
- Owner ID
- Previous Block Hash

Then the hash of the block is calculated using SHA-256 algorithm and stored in the same block.

```
public void calculateHash() {
    try {
        String value = new ObjectMapper().writeValueAsString(this);
        this.hash = Hashing.sha256().hashString(value,
StandardCharsets.UTF_8).toString();
    } catch (JsonProcessingException e) {
        e.printStackTrace();
    }
}
```

After populating all fields inside the block, the newly generated block is stored into the H2 embedded database.

Finally, the new block added notification is sent to the other aggregators using the Block synchronization functionality.

4.6.2 Blockchain Search

Blockchain search feature is provided for consumers to search data stored in the blockchain. The function also supports a query API which can be used to query blocks based on the data stored inside the block.

Fetch All Data

Following API URI is provided for consumers, who want to obtain a copy of the blockchain. Since, the blockchain may contain a large amount of data, this method will not be used frequently.

```
GET /blocks/all
```

The API URI does not accept any parameter and provide the complete list of blocks stored in the blockchain as the response.

Example response:


```
[{
    "id": "9dd451bf-a299-4a1b-a8aa-76dd39952629",
    "type": "APP_START",
    "deviceId": "d076c6a4-a61a-4eab-a7bc-90deb51e689d",
    "encrypted": false,
    "hash": "1caf2b188cdc1c2ad25db6e36ad8062e9c4843fd21a3fecf09f93658821f584a",
    "previousHash": null,
    "data": "{\"type\":\"APP\",\"deviceId\":\"d076c6a4-a61a-4eab-a7bc-
90deb51e689d\",\"hostname\":\"192.168.56.1\",\"port\":8080,\"timestamp\":15531
72755934}",
    "addedTime": 1553172755979
  },{
    "id": "0c0c4a58-2ecc-4836-b9af-efaea12f24f1",
    "type": "APP_START",
    "deviceId": "d076c6a4-a61a-4eab-a7bc-90deb51e689d",
    "encrypted": false,
    "hash": "98112e285fe7005f9a7a7d924cc0a9187edbcc2f33f64a9edc37b0ecc2660023",
    "previousHash":
"1caf2b188cdc1c2ad25db6e36ad8062e9c4843fd21a3fecf09f93658821f584a",
    "data": "{\"type\":\"APP\",\"deviceId\":\"d076c6a4-a61a-4eab-a7bc-
90deb51e689d\",\"hostname\":\"192.168.56.1\",\"port\":8080,\"timestamp\":15531
72904927}",
    "addedTime": 1553172904974
  }]
```


Query Blockchain Data

Support for querying the data stored in the blockchain is provided to consumers through the following URI.

POST /blocks/search

This URI accepts the following request in the body of the HTTP POST request.


```
{
  "encryptionKey": "encryption_key",
  "salt": "salt",
  "path": "path/to/field",
  "value": "
}
```


Blockchain search request parameters are described in Table 4.4.

Parameter	Description
encryptionKey	[OPTIONAL] The key which is to be used for data decryption.
salt	[OPTIONAL] This parameter will be used in combination with the encryption key to strength the encryption. Adding salt to the encryption key will mitigate the risk of dictionary attacks.
path	JSONPath to select the parameter
value	query string value

Table 4.4: Blockchain Search Request Parameters

JSONPath

Like XPath for XML, JSONPath querying JSON documents with expressions. JSONPath expressions refer to a JSON document in the same way as XPath. However, unlike XML, a JSON document may or may not have a root element. Hence, JSONPath use $ as the root element of the document.

e.g.: *$.home.lights[0].brightness*

This request returns the data matching with the query from the blockchain. If a block is encrypted, the provided encryption key and salt will be used to decrypt the data inside the block. Following is an example output generated by this API method.


```
[{
    "id": "d4a44f3b-3869-4efc-a7d3-2de950e20039",
    "type": "APP_START",
    "deviceId": "d076c6a4-a61a-4eab-a7bc-90deb51e689d",
    "encrypted": false,
    "hash": "8f1671b7f6d6…..730128aa696cd7ab77a3d",
    "previousHash": "da0b3bfff3e1685053…8fbcba3e40c3f95fdf5c73ce0723",
    "data": "{\"type\":\"APP\",\"deviceId\":\"d076c6a4-a61a-4eab-a7bc-
90deb51e689d\",\"hostname\":\"192.168.56.1\",\"port\":8080,\"timestamp\":15568
00367007}", "addedTime": 1556800367019
  },{
    "id": "46b4a24b-cbb0-48f0-9a0e-731935fe169f",
    "type": "APP_START",
    "deviceId": "d076c6a4-a61a-4eab-a7bc-90deb51e689d",
    "encrypted": false,
    "hash": "876fb0870985d3c488522…..df7492a97b69008e9534aea51d289",
    "previousHash": "8f1671b7f6d6fd318f33c5e73…30128aa696cd7ab77a3d",
    "data": "{\"type\":\"APP\",\"deviceId\":\"d076c6a4-a61a-4eab-a7bc-
90deb51e689d\",\"hostname\":\"192.168.56.1\",\"port\":8080,\"timestamp\":15568
09480022}",
    "addedTime": 1556809480069
  }]
```

Query Blockchain Data by Block Id

If the block identifier is known to the device, the following URI can be used to fetch the block from the blockchain.

GET /blocks/get

This API URI accepts the block identifier as a GET parameter and returns the matching block if exists. This API URI always returns one block.

Following is an example response generated by the URI.


```
{
  "id": "8f21f9cb-0cf0-4389-aa69-052bde33dd64",
  "type": "APP_START",
  "deviceId": "d076c6a4-a61a-4eab-a7bc-90deb51e689d",
  "encrypted": false,
  "hash": "b1333e1122094a38d0d11…49c00e5fd73a0b108895111483449cbb9a33156",
  "previousHash": "98112e285fe7005f9a7a7d…2f33f64a9edc37b0ecc2660023",
  "data": "{\"type\":\"APP\",\"deviceId\":\"d0…90deb51e689d\",\"hostname\":
\"192.168.56.1\",\"port\":8080,\"timestamp\":1553173054142}",
  "addedTime": 1553173054203
}
```


4.6.3 Blockchain Sync

Initially, each aggregator maintains a list of other known aggregators called seeds. When aggregator is started, it selects a random aggregator from the seed list or from the existing blockchain in order to obtain the updated blocks. This process is important as the other aggregators may have generated new blocks while the considered aggregator is offline. The aggregator will not respond to the queries until its synchronization process is successfully completed.

Following code snippet is used to perform the synchronize function.

```
@EventListener(ApplicationReadyEvent.class)
public void sync() {
    String hash = blockSearchService.getPreviousHash();
    LOGGER.info("Obtaining updated blocks through hub service");
    obtainUpdatedBlocks(hash);
    LOGGER.info("Obtained updated blocks through hub service");
    boolean valid = validationService.validate();
    if (!valid) {
        LOGGER.info("Validation status is invalid.");
        blockRepository.deleteAll();
        obtainUpdatedBlocks("");
        LOGGER.info("Completed rebuilding local storage");
    }
}
```

First, the hash of the last block is obtained from the local blockchain. Then the blocks generated after the obtained hash is requested from other aggregators.

```
private void obtainUpdatedBlocks(String hash) {
    List<BlockDTO> blockDTOS = hubService.getAfter(hash);
    if (blockDTOS != null) {
        LOGGER.info("Store updated blocks");
        blockDTOS.forEach(blockDTO -> blockRepository.save(new
Block(blockDTO)));
    }
}
```

The blocks are obtained through the Hub Service which is communicating with multiple aggregators at the same time. The Hub Service will return only the most agreed response for a request.

Once all blocks are received, the blockchain will be validated by the aggregator to ensure that the local blockchain is still valid. If the local blockchain validation has been failed, the blocks will be requested again from other aggregators. The process continues, until a configurable number of times or until the blockchain validated successfully. After this process, the aggregator can ensure that its local blockchain is up to date with the blockchains of other aggregators.

4.6.4 Blockchain Validation

Blockchain validation is performed by aggregator to ensure the integrity of the data stored in the blockchain. During the validation, the aggregator starts from the initial block stored in the blockchain and continues until the last block in the blockchain. Every time the aggregator is started, this validation is performed.

Manual Block Validation

In addition to the automatic validation, the validation function can be triggered manually at any time by calling the following URI.

```
GET /validation/validate
```

This URI does not accept any parameter and returns the validation status (true or false) as the response.

Block validation is performed sequentially using the following code snippet.

```
private boolean isValid() {
    List<Block> blocks; int page = 0; String previousHash;
    do {
        blocks = blockRepository.findAllByOrderByAddedTime(
                        PageRequest.of(page++, pageSize));
        if (!blocks.isEmpty()) {
            previousHash = blocks.get(0).getHash();
            boolean first = true;
            for (Block block : blocks) {
                if (first) {
                    first = false;
                    continue;
                }
                if (!previousHash.equals(block.getPreviousHash())) {
                    return false;
                }
                previousHash = block.getHash();
            }
        }
    } while (blocks.size() == pageSize);
    return true;
}
```

During the validation, the blocks are loaded into the memory in batches as the complete blockchain may not fit into the memory available in the aggregator. The number of blocks loaded into the memory in one batch can be configured using the following system configuration.

```
block.load.page.size = 1000
```

This manual validation process may take some time to complete based on the number of blocks stored in the blockchain.

Check Validation Status

The following URI is provided to check the last validation status of the blockchain without running the complete validation process again.

```
GET /validation/lastStatus
```

This URI does not accept any parameters and return the last validation status of the blockchain without a re-validation. Hence, this method is expected to run faster than a manual validation.

However, the status returned from this URI may not represent the current validation status of the blockchain.

View Previous Validation Results

Another URI is provided to view the status of previous validations performed by the aggregator. The output of this URI is useful to analyze when the blockchain became invalid. The response can be obtained by calling the following URI.

GET /validation/all

This URI does not accept any parameters and produces a list of validation status records with each record having following fields.

- Validation Status
- Validation Start Time
- Validation End Time


```
[
  {
    "id": "0143fecc-8faf-4015-867b-ad3b761aae7f",
    "startedTime": 1553171775881,
    "completedTime": 1553171776187,
    "valid": true
  },
  {
    "id": "6bcc5698-76b8-41bf-b01c-e29c27643eca",
    "startedTime": 1553172173019,
    "completedTime": 1553172173475,
    "valid": true
  }
]
```


4.6.5 Device Registration

A new device can be registered to the blockchain using the following URI.

GET devices/register

This URI does not accept any parameters and produces the following response if the device registration is successful. The response should be transferred to the device in a secure manner.


```
{
  "deviceId": "d47ae6e6-ba8d-4f63-a36b-a704f2cf5027",
  "publicKeyAlgorithm": "RSA",
  "privateKeyAlgorithm": "RSA",
  "publicKey": "MIIBIjANBgkqhkiG...7i+BQIDAQAB",
  "privateKey": "MIIEvQIBADANBgkw0B...b7uOFgsxqkvjuog="
}
```

The response contains the following parameters.

deviceId : Unique device id generated for the device. The device Id is a UUID generated using a cryptographically strong pseudo random number generator.

publicKeyAlgorithm : The algorithm used to generate the public key.

privateKeyAlgorithm: The algorithm used to generate the private key.

publicKey : The public key.

privateKey : The private key.

The public key and private key are generated for a device using the RSA algorithm with key size of 2048. The following code snippet is used to generate the keys.

```
KeyPair keyPair = null;
try {
    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
    keyPairGenerator.initialize(2048);
    keyPair = keyPairGenerator.generateKeyPair();
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
}
return keyPair;
```

The KeyPairGenerator that generates public/private key pairs for the specified algorithm is created using the following code snippet. This method iterates the available list of registered security providers starting with the most preferred provider and returns a KeyPairGenerator object encapsulating the KeyPairGeneratorSpi implementation from the first provider that supports the specified algorithm.

```
public static KeyPairGenerator getInstance(String algorithm)
        throws NoSuchAlgorithmException {
    Objects.requireNonNull(algorithm, "null algorithm name");
    List<Service> list =
            GetInstance.getServices("KeyPairGenerator", algorithm);
    Iterator<Service> t = list.iterator();
    if (t.hasNext() == false) {
        throw new NoSuchAlgorithmException
            (algorithm + " KeyPairGenerator not available");
    }
    // find a working Spi or KeyPairGenerator subclass
    NoSuchAlgorithmException failure = null;
    do {
```

```
        Service s = t.next();
        try {
            Instance instance =
                GetInstance.getInstance(s, KeyPairGeneratorSpi.class);
            if (instance.impl instanceof KeyPairGenerator) {
                return getInstance(instance, algorithm);
            } else {
                return new Delegate(instance, t, algorithm);
            }
        } catch (NoSuchAlgorithmException e) {
            if (failure == null) {
                failure = e;
            }
        }
    } while (t.hasNext());
    throw failure;
}
```

Then the KeyPairGenerator is initialized using 2048 to generate the keys. The following code snippet is used to generate the Keys using the KeyPairGenerator.

```
public KeyPair generateKeyPair() {
    // accommodate odd key sizes in case anybody wants to use them
    int lp = (keySize + 1) >> 1;
    int lq = keySize - lp;
    if (random == null) {
        random = JCAUtil.getSecureRandom();
    }
    BigInteger e = publicExponent;
    while (true) {
        // generate two random primes of size lp/lq
        BigInteger p = BigInteger.probablePrime(lp, random);
        BigInteger q, n;
        do {
            q = BigInteger.probablePrime(lq, random);
            // convention is for p > q
            if (p.compareTo(q) < 0) {
                BigInteger tmp = p;
                p = q;
                q = tmp;
            }
            // modulus n = p * q
            n = p.multiply(q);
            // even with correctly sized p and q, there is a chance that
            // n will be one bit short. re-generate the smaller prime if so
        } while (n.bitLength() < keySize);
        // phi = (p - 1) * (q - 1) must be relative prime to e
        // otherwise RSA just won't work ;-)
        BigInteger p1 = p.subtract(BigInteger.ONE);
        BigInteger q1 = q.subtract(BigInteger.ONE);
        BigInteger phi = p1.multiply(q1);
        // generate new p and q until they work. typically
        // the first try will succeed when using F4
        if (e.gcd(phi).equals(BigInteger.ONE) == false) {
```

```
                continue;
            }
            // private exponent d is the inverse of e mod phi
            BigInteger d = e.modInverse(phi);

            // 1st prime exponent pe = d mod (p - 1)
            BigInteger pe = d.mod(p1);
            // 2nd prime exponent qe = d mod (q - 1)
            BigInteger qe = d.mod(q1);
            // crt coefficient coeff is the inverse of q mod p
            BigInteger coeff = q.modInverse(p);

            try {
                PublicKey publicKey = new RSAPublicKeyImpl(rsaId, n, e);
                PrivateKey privateKey = new RSAPrivateCrtKeyImpl(
                    rsaId, n, e, d, p, q, pe, qe, coeff);
                return new KeyPair(publicKey, privateKey);
            } catch (InvalidKeyException exc) {
                // invalid key exception only thrown for keys < 512 bit,
                // will not happen here
                throw new RuntimeException(exc);
            }
        }
    }
```

The generated public key is stored in the blockchain with the device id which is used to authenticate the device. With this information, a new block is added to the blockchain with the type as "DEVICE_REGISTER".


```
{
    "id": "7812b8bc-53c6-495f-86a9-d06aacceb699",
    "type": "DEVICE_REGISTER",
    "deviceId": "7ffa9781-25fb-41fa-afd1-f3ef49d4df6e",
    "encrypted": false,
    "hash": "45ac0cb54b2336a34…0e7bba7961a29fba4bcb3a4e",
    "previousHash": "7f262c1daf84517…0826965f42a22f0c6a705c72c",
    "data": "{\"deviceId\":\"7ffa9781-25fb-41fa-afd1-
f3ef49d4df6e\",\"publicKey\":\"MIIBIjANB...4bGuz4I+7i+BQIDAQAB\"}",
    "addedTime": 1556944537829
}
```


Device id inside the data field is the deviceId of the newly created device, and the deviceId appear in the block level is the deviceId of the aggregator who created the device. The device registration block stored in the blockchain is stored without encryption and can be accessed by any device in the network.

Once the new device is registered, the newly added block is broadcasted to the other aggregators in the network to perform device authentication with any aggregator without depending on a single aggregator where the device is registered. Figure 4.4 shows a summary of the steps performed during device registration.

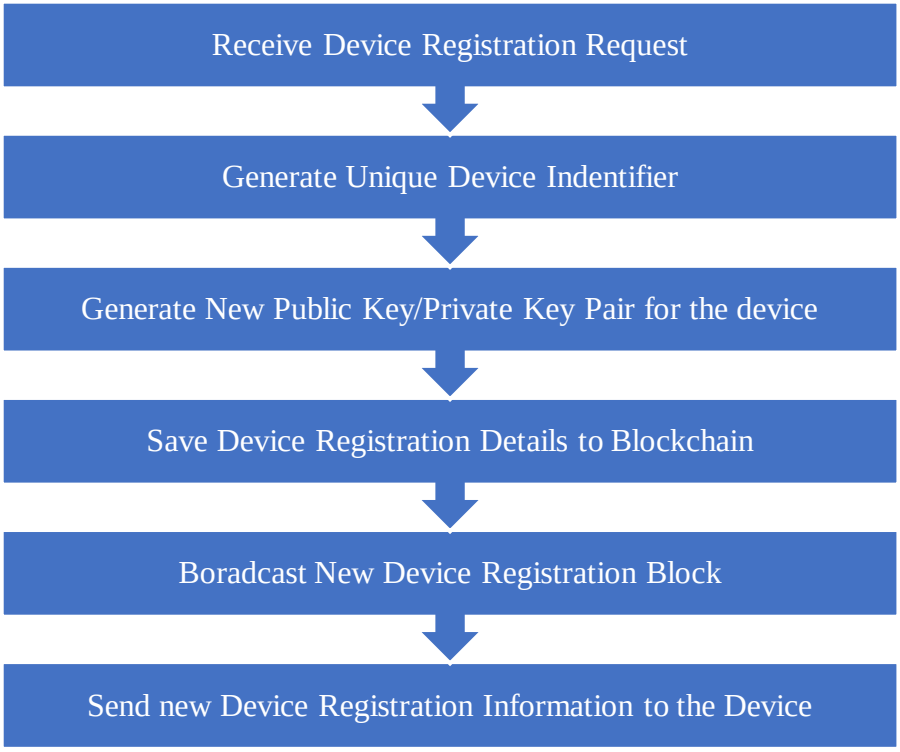


Figure 4.4: New Device Registration Steps

The private key is not stored in the blockchain and provided to the registered device in the response. The device should store the private key securely in order to perform authentication in the future before sending any data.

4.6.6 Device Authentication

Device authentication is performed using a three-step process with three different URIs in the following order.

1. Request authentication (/authenticator/authenticate)
2. Get aggregator public key (/devices/getPublicKey)
3. Verify device identity (/authenticator/verify)

Request Authentication

Following URI is called with the deviceId as a POST parameter to initiate the authentication process.

/authenticator/authenticate

Then operation perform the steps shown in Figure 4.5 to generate and return the authentication challenge token to the device.

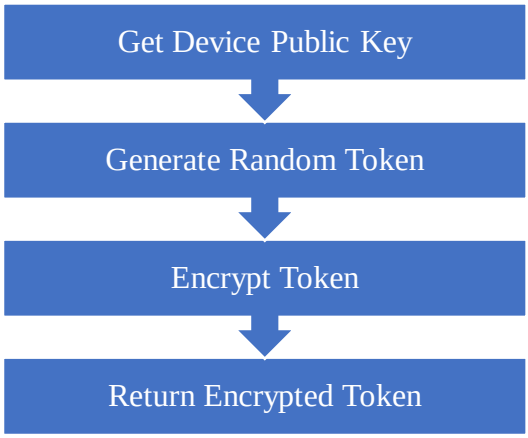


Figure 4.5: Authentication Challenge Generation Process

Get Device Public Key – The public key of the device is obtained from the blockchain using the following code snippet.

```
public String getPublicKey(String deviceId) throws IOException {
    BlockDTO blockFound = blockSearchService.findOne(block ->
            (BlockType.APP_REGISTER == block.getType() ||
BlockType.DEVICE_REGISTER == block.getType())
                    && deviceId.equals(block.getDeviceId()
            ));
    if (blockFound != null) {
        DeviceInfoDTO deviceInfoDTO =
objectMapper.readValue(blockFound.getData(), DeviceInfoDTO.class);
        return deviceInfoDTO.getPublicKey();
    }
    return null;
}
```

Generate Random Token – A random UUID is generated for the authentication process.

Encrypt Token – The generated random token is encrypted using the public key obtained from the blockchain.

Return Encrypted Token – Once the token is encrypted, the encrypted value is returned as the response to the device requesting to initiate the authentication.

Get Aggregator Public Key

The response returned for the authenticate request contains the identifier of the aggregator generated the authentication challenge. Then the device will request the public key of the aggregator involved in the authentication process from the following URI by passing the aggregator device id as a GET parameter.

```
GET /devices/getPublicKey
```

This URI can be requested from any aggregator. Once the request is received, the hub layer of the aggregator may request this information from several other aggregators to ensure that the returned public key is correct.

Verify Identity

The device next tries to decrypt the challenge received from the URI using the aggregator public key obtained from the above step. Once successful, the device will get the decrypted challenge. Then the device tries to solve the challenge using its own private key stored in the device. Finally, when the device was successfully able to solve the challenge, it will encrypt the challenge one more time using the aggregator device public key and sent to the aggregator. Then the encrypted solution is sent to the aggregator by calling the following URI.

```
/authenticator/verify
```

If the authentication process is successful, the response contains the authentication token which must be sent with all future requests from the endpoint. Then authentication token issued by the aggregator is encrypted using the device public key before sending the token. Hence, capturing the token in-between the aggregator and device will not allow an interceptor to gain unauthorized access.

The authentication token issued by the aggregator is valid only a configurable time duration. After this duration, the device should obtain a new token from the aggregator.

A device can obtain more than one token from the aggregator and implement a token pool which is maintained by a background worker. This approach is recommended to make sure that tokens in the pool are always valid. The number of token issues per device can be configured using the below system configuration.

max.concurrent.autth.token.count =100

The device should decrypt the token using the aggregator public key and encrypt it using the device private key before sending again. The encrypted authentication token should be sent using "Authorization" HTTP header with the "Bearer" authorization scheme.

Example:

```
GET /validation/lastStatus HTTP/1.1

Host: localhost

Authorization: Bearer 7ffa9781f3....ef49d4df6e
```

The Bearer authentication scheme which is used by the solution is registered in IANA [28] and originally defined in the RFC 6750 [29] for the OAuth 2.0 [30] authorization framework.

If this token was not present in the request header or not validated successfully, the aggregator may produce authentication failure and return the HTTP status 401 to the device.

4.6.7 Device Information

Several URIs are implemented in the solution to provide information about devices in the network.

List Started Applications

The following URI provides the list of started applications. This URI does not accept any parameters.

GET /devices/getStartedApplications

Check If Aggregator Has Started

Devices in the network can check if an aggregator is started using the following URI which accept deviceId of the aggregator as the POST parameter.

/devices/isStarted

The URI provides true/false as the response based on the aggregator started status. If the requested device is not found in the network, the URI will provide a response with the HTTP status code 404.

Retrieve Device Registration Record

The following URI accept the deviceId as a GET parameter and return the device registration block.

GET /devices/get

Following is an example response generated by the above URI.


```
{
  "id": "04927fde-d6a2-4191-9dab-0372808ac47f",
  "type": "DEVICE_REGISTER",
  "deviceId": "cf1f46c5-882a-4ed4-8556-95a91e070dc9",
  "encrypted": false,
  "hash": "907c065cb1d3f477837bb832e93065f83ccad2b9383b7d1a9d8100315fad9823",
  "previousHash":
"91dfb2da5c1e3579f51b8dae6553a9ce21a9c129d9ab830516eeb0be7e721e4a",
  "data": "{\"deviceId\":\"cf1f46c5-882a-4ed4-8556-
95a91e070dc9\",\"publicKey\":\"MIIBIjANBgkqhkiG9w...oXBlwIDAQAB\"}",
  "addedTime": 1556944278850
}
```


If the requested device is not found in the network, a response will be returned with the HTTP status code 404.

Retrieve All Device Registration Records

Registration records of all devices found in the network can be obtained by calling the following URI. This URI accepts the device type as a GET parameter and returns the list of device registration records.

GET /devices/all

The type parameter of this URI can be either of the following two values.

- APP – To obtain the list of aggregators.
- DEVICE – To obtain the list of devices without aggregators.

Following is a sample response generated by this URI.


```
[{
    "id": "04927fde-d6a2-4191-9dab-0372808ac47f",
    "type": "DEVICE_REGISTER",
    "deviceId": "cf1f46c5-882a-4ed4-8556-95a91e070dc9",
    "encrypted": false,
    "hash": "907c065cb1d…83b7d1a9d8100315fad9823",
    "previousHash": "9ce21a9c129d9…b830516eeb0be7e721e4a",
    "data": "{\"deviceId\":\"cf1f46c5-882a-4ed4-8556-
95a91e070dc9\",\"publicKey\":\"MIIBIjANBg…wIDAQAB\"}",
    "addedTime": 1556944278850
  },
  {
    "id": "7812b8bc-53c6-495f-86a9-d06aacceb699",
    "type": "DEVICE_REGISTER",
    "deviceId": "7ffa9781-25fb-41fa-afd1-f3ef49d4df6e",
    "encrypted": false,
    "hash": "45ac0cb54b23…09553420e7bba7961a29fba4bcb3a4e",
    "previousHash": "7f262c1b051daf8…f47892a22f0c6a705c72c",
    "data": "{\"deviceId\":\"7ffa9781-25fb-41fa-afd1-
f3ef49d4df6e\",\"publicKey\":\"MIIBIjA…4I+7i+BQIDAQAB\"}",
    "addedTime": 1556944537829
  }]
```


Total Number of Devices Registered in the Network

The total number of devices registered in the network can be obtained by calling the following URI which does not accept any parameters.

```
GET /devices/total
```

4.6.8 Hub Function

The hub function allows the consumers to request results from other aggregators and find the most accepted result in the network. Due to this function, compromising few nodes in the network will not grant a malicious user to manipulate all data stored in the network.

Devices can request API calls to provide from hub function by setting the HTTP header X-VALIDATE-RESULT.

The hub function is implemented in according to the process shown in Figure 4.6.

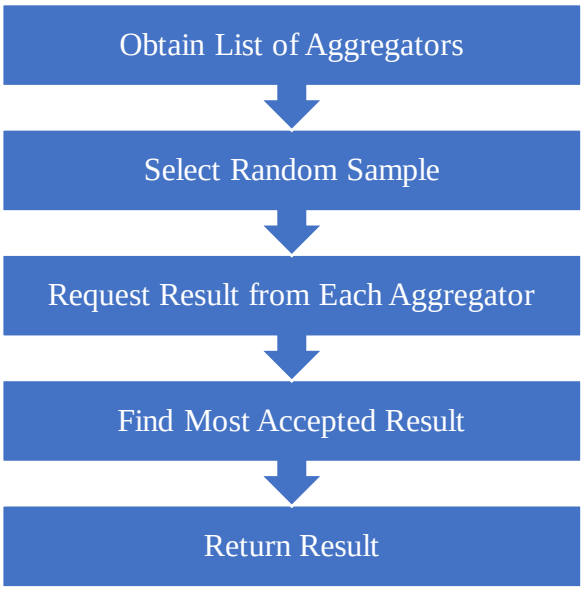


Figure 4.6: Hub Function Steps

Obtain List of Aggregators

The aggregator list is obtained by filtering the blockchain based on the device registration record type. Only the aggregators that are currently in the running status is added to the list. The following code snippet is used to obtain the list of aggregators.

```
public List<AppStatusDTO> getStartedApplications() {
    List<AppStatusDTO> appStatusDTOS = new ArrayList<>();
    blockSearchService.find(block -> block.getType() ==
BlockType.APP_REGISTER).forEach(blockDTO -> {
        BlockDTO lastStartedBlock =
getLastStatusBlock(blockDTO.getDeviceId());
        if (lastStartedBlock != null && lastStartedBlock.getType() ==
BlockType.APP_START) {
            try {

appStatusDTOS.add(objectMapper.readValue(lastStartedBlock.getData(),
AppStatusDTO.class));
            } catch (IOException e) {
                LOGGER.error("Error in reading application status", e);
            }
        }
    });
    return appStatusDTOS;
}
```

Select Random Sample

After obtaining the list of active aggregators, a random sample is selected using the following code snippet.

```
private List<AppStatusDTO> selectVerifiers() {
    List<AppStatusDTO> appStatusDTOS = deviceService.getStartedApplications();
    Collections.shuffle(appStatusDTOS);
    int toIndex = verifiesCount > appStatusDTOS.size() ? appStatusDTOS.size()
: verifiesCount;
    return appStatusDTOS.subList(0, toIndex);
}
```

The shuffle function randomly permutes the specified list in a way that all permutations occur with approximately equal likelihood. The implementation of this function traverses the list backwards, from the last element up to the second, repeatedly swapping a randomly selected element into the "current position". Then the elements are randomly selected from the portion of the list that runs from the first element to the current position, inclusive.

The number of aggregators to select in the sample can be configured using the following system configuration.

```
hub.verifiers.count=5
```

Request Result from Each Aggregator

Once the random sample of aggregators are selected, each aggregator is contacted to obtain the result for the request. Following code snippet is used to perform this function.

```
private <T> List<T> getResults(String request, List<NameValuePair> parameters,
Class<T> type) {
    List<T> results = new ArrayList<>();
    selectVerifiers().forEach(appStatusDTO -> {
        String uri = String.format(appEndPointPattern,
appStatusDTO.getHostname(),
                appStatusDTO.getPort(),
                request, URLEncodedUtils.format(parameters,
Charset.defaultCharset()));
        LOGGER.info("Obtaining results for " + uri);
        long startTime = System.currentTimeMillis();
        T result = restTemplate.getForEntity(uri, type).getBody();
        LOGGER.info(String.format("Obtained results for %s in %dms", uri,
(System.currentTimeMillis() - startTime)));
        results.add(result);
    });
    return results;
}
```

Find Most Accepted Result

Finally, the most accepted result is selected by considering results obtained from all aggregators. The result will be returned only if the acceptance percentage of the result is above a configurable threshold. If a result cannot be found with above the accepted threshold, an error is returned from the URI. The below system configuration can be used to configure the threshold used in the aggregator.

```
hub.accept.percentage=.5
```

The below code snippet is used to find the common result out of the list of results.

```
private <T> T getCommonResult(List<T> results) {
    if(results == null || results.isEmpty()) {
        return null;
    }
    Map<T, Integer> commonResultCounts = new HashMap<>();
    results.forEach(result -> {
        if (commonResultCounts.containsKey(result)) {
            commonResultCounts.put(result, commonResultCounts.get(result) +
1);
        } else {
            commonResultCounts.put(result, 1);
        }
    });
    int totalResults = results.size();
    LOGGER.info("Total number of results found: " + totalResults);
    List<Map.Entry<T, Integer>> entries =
commonResultCounts.entrySet().stream()
            .sorted(Map.Entry.comparingByValue())
            .collect(Collectors.toList());
    LOGGER.info("Number of different results found: " + entries.size());
    for (Map.Entry<T, Integer> entry : entries) {
        float resultAcceptedPercentage = (float) entry.getValue() /
totalResults;
        LOGGER.info("Result accepted with percentage: " +
resultAcceptedPercentage);
        if (resultAcceptedPercentage > acceptPercentage) {
            return entry.getKey();
        }
    }
    throw new NoAcceptableResultException();
}
```

4.6.9 Secure Communication

The devices can encrypt the payload before sending to the aggregator using the token provided during the authentication token provided from aggregator. If the request is encrypted, the HTTP header X-ENCRYPTED must be set in order to decrypt the request correctly by the aggregator. Furthermore, if this header is set in the request, the response provided by the aggregator will also be encrypted using the same token.

Reply Attack Protection

A replay attack occurs in a network, when an attacker intercepts the communication on the network and fraudulently manipulates requests to delay or repeat the intended action in order to cause damage.

To mitigate this attack, the aggregator sends a random sequence initialization number to the device in the authentication successful result. Then the device should increment this number by one every time a new request is created. This sequence should be set as the value of HTTP header X-SEQUENCE. If two requests are received with the same sequence number, the aggregator will reject the request by returning a response with the HTTP status code 403 (Forbidden).

4.6.10 API Playground

A web-based UI is provided in order to experiment with the API published by the solution. This API Playground tool is hosted by the aggregator and can be accessed through the following URI.

GET /api-playground

This initial UI summarizes all the API calls provided by the aggregator as shown in the Figure 4.7.

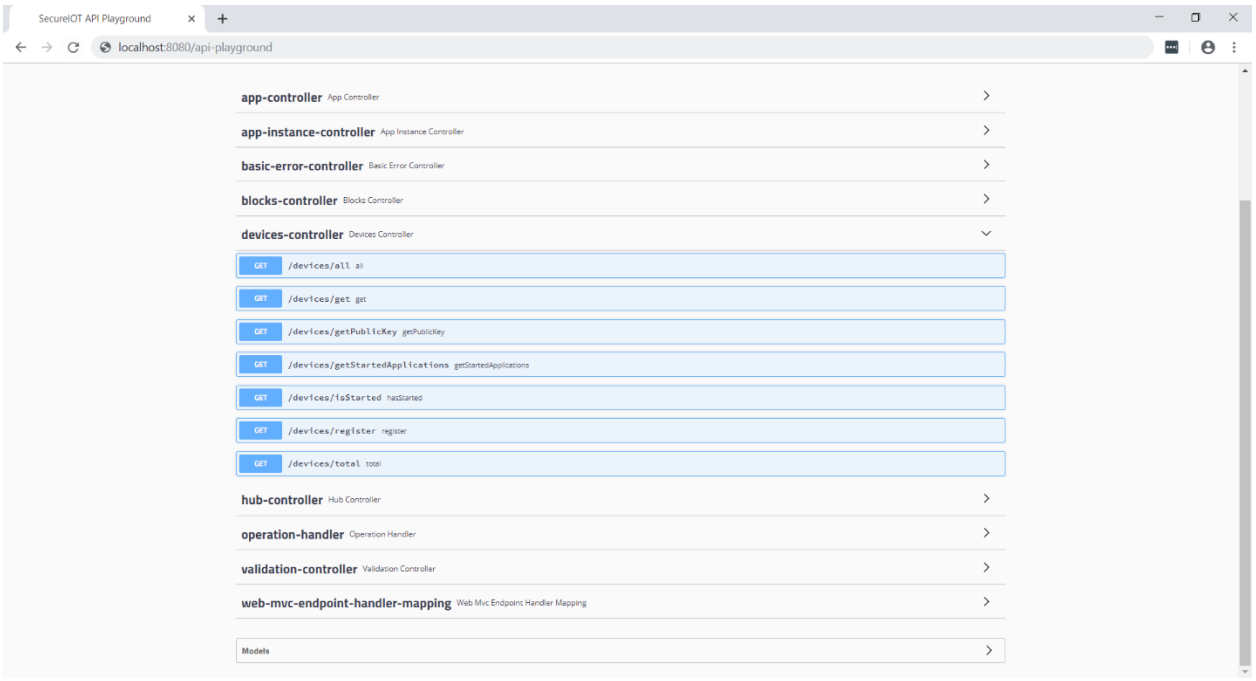


Figure 4.7: API Call Summary Screen in API Playground

The URIs are categorized into several controllers based on their function and can be expanded to view more details.

The “Try it Out” feature in the API playground allows the users to compose and send a request to the URI and obtain the response as shown in Figure 4.8.

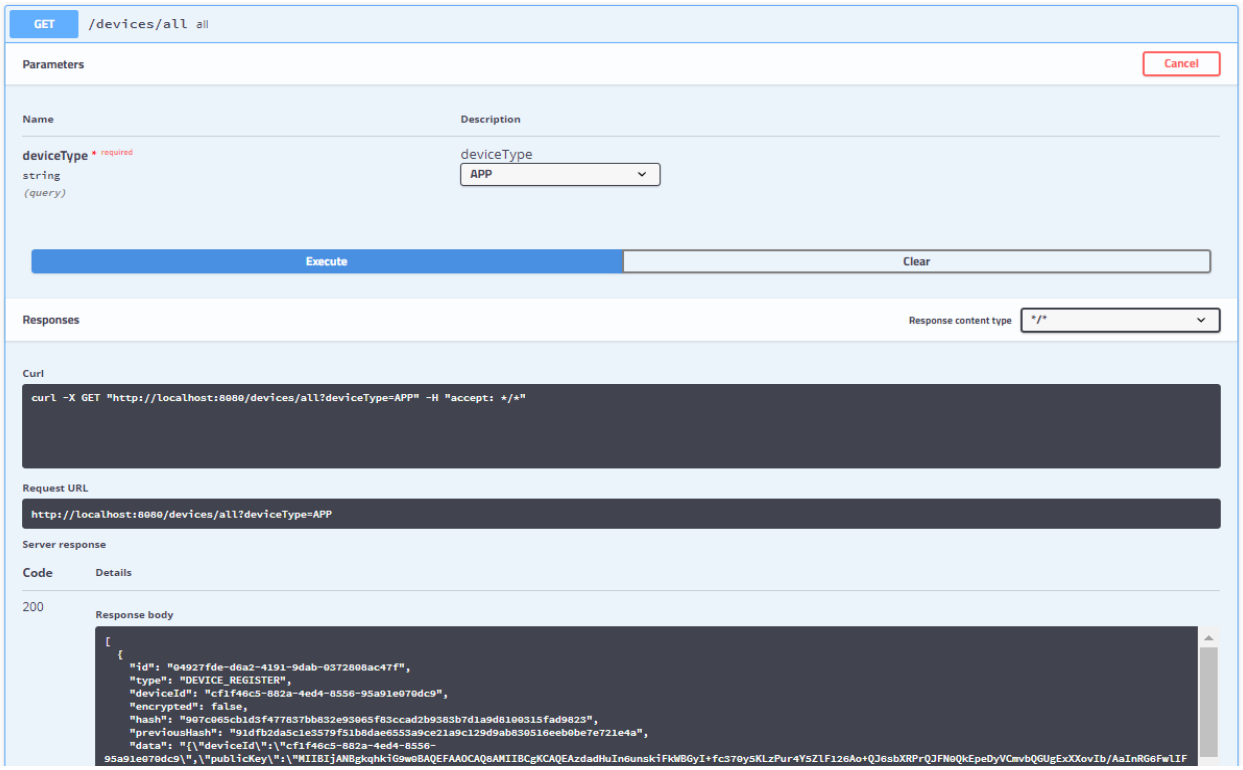


Figure 4.8: Try it Out Feature in API Playground

4.7 COMPLETE SOLUTION

After implementing the subsystems, they are integrated to form the complete solution. The complete solution is packaged as a WAR file [31] which can be executed as a standalone program using the command: *java -jar secureiot.war*

Figure 4.9 shows the output of running the SecureIoT application.

Figure 4.9: Running SecureIoT Application

CHAPTER 5 RESULTS AND EVALUATION

5.1 INTRODUCTION

Evaluation is a process conducted to measure the how far the implemented solution meets its expected requirements. Testing is performed by operating the solution under controlled conditions while performing a pre-determined set of verification and validation tasks. The goal of the evaluation process is to establish confidence that the solution is "fit for purpose". This chapter summarizes the results obtained by evaluating the secure IoT framework developed according to the proposed solution described in previous sections.

5.2 EVALUATION ENVIRONMENT

The testing environment of the solution was selected so that the solution can be fully tested under a common environment. Following sections describe the software/hardware specification of the testing environment used to evaluate the solution.

5.2.1 Aggregator Evaluation Environment

Table 5.1 describes the specifications of the aggregator device used to test the developed solution. Since all the processing tasks are mainly performed by the aggregator, a reasonably high-power device is required.

Hardware	Software
SOC: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit **CPU**: 1.4 GHz 64-bit quad-core ARM Cortex-A53 CPU **Memory**: 1 GB LPDDR2 SDRAM **Storage**: 32 GB **Wi-Fi**: Dual-band 802.11ac and Bluetooth 4.2 **Power**: 5V/2.5A DC power input	**Operating System**: Raspbian **Runtime**: Java 11 **Web Server**: Tomcat 9

Table 5.1: Aggregator Application Evaluation Environment

5.2.2 IoT Device Evaluation Environment

IoT device environment is planned to create a prototype device which can send/receive data to/from the aggregator device in enough data rate for testing. Table 5.2 summarize the specification of the selected IoT device environment.

Hardware	Software
CPU: Tensilica Xtensa LX6 microprocessor @ 160 or 240 MHz **Memory**: 520 KiB SRAM **Wi-Fi**: 802.11 b/g/n/d/e/i/k/r (2.4 GHz) **Power**: 3.3 V DC	**Wi-Fi Modes**: Station / softAP / SoftAP+station / P2P **Security**: WPA/WPA2/WPA2-Enterprise/WPS **Encryption**: AES/RSA/ECC/SHA

Table 5.2: IoT Device Evaluation Environment

5.2.3 IoT Device Evaluation Program

An IoT device which can send/receive data in a pre-defined data rate has been developed to evaluate the solution. The source code of the program can be found in Appendix D. Figure 5.1 shows the menu appear in the program after connecting to an existing Wi-Fi network.

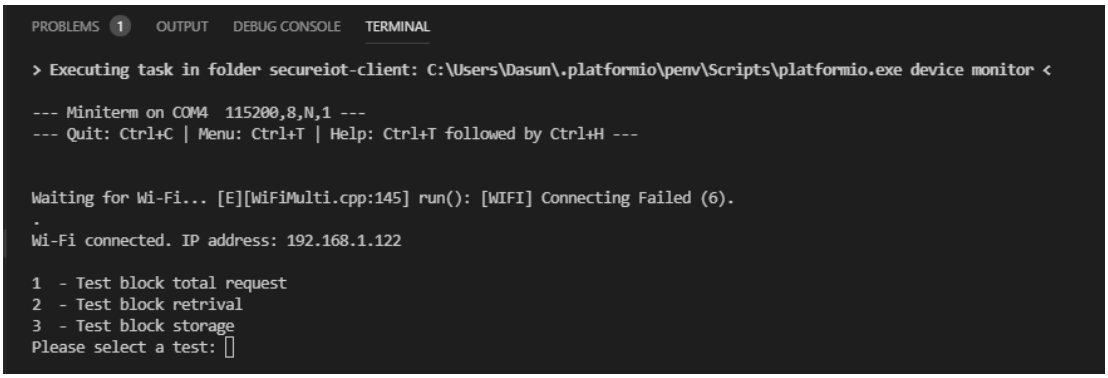


Figure 5.1: Running IoT Program for Evaluation Tests

5.2.4 Complete Evaluation Environment

The complete evaluation environment designed for the solution consists of two Raspberry PI [32] modules which are acting as aggregators and two ESP32 [33] modules which are acting as IoT devices connected to the aggregator. This evaluation environment uses Wi-Fi as the communication medium to establish the connection between aggregators and IoT devices. The architectural diagram of the complete evaluation environment is shown in Figure 5.2.

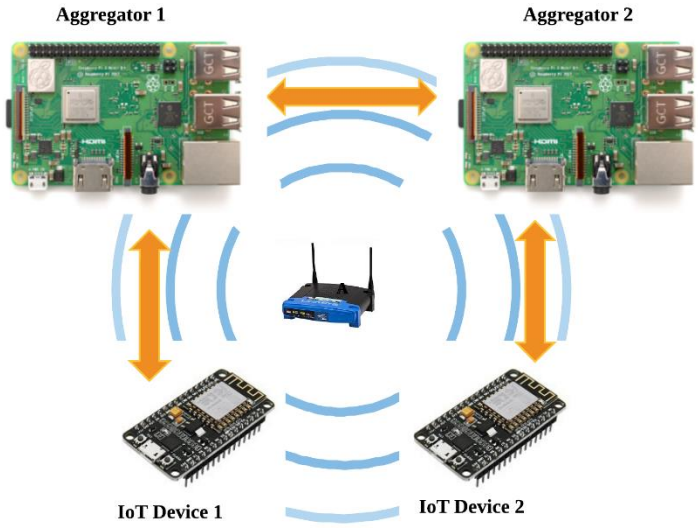


Figure 5.2: Complete Evaluation Environment

The prototype IoT network developed to evaluate the implemented secure IoT framework is shown in Figure 5.3.

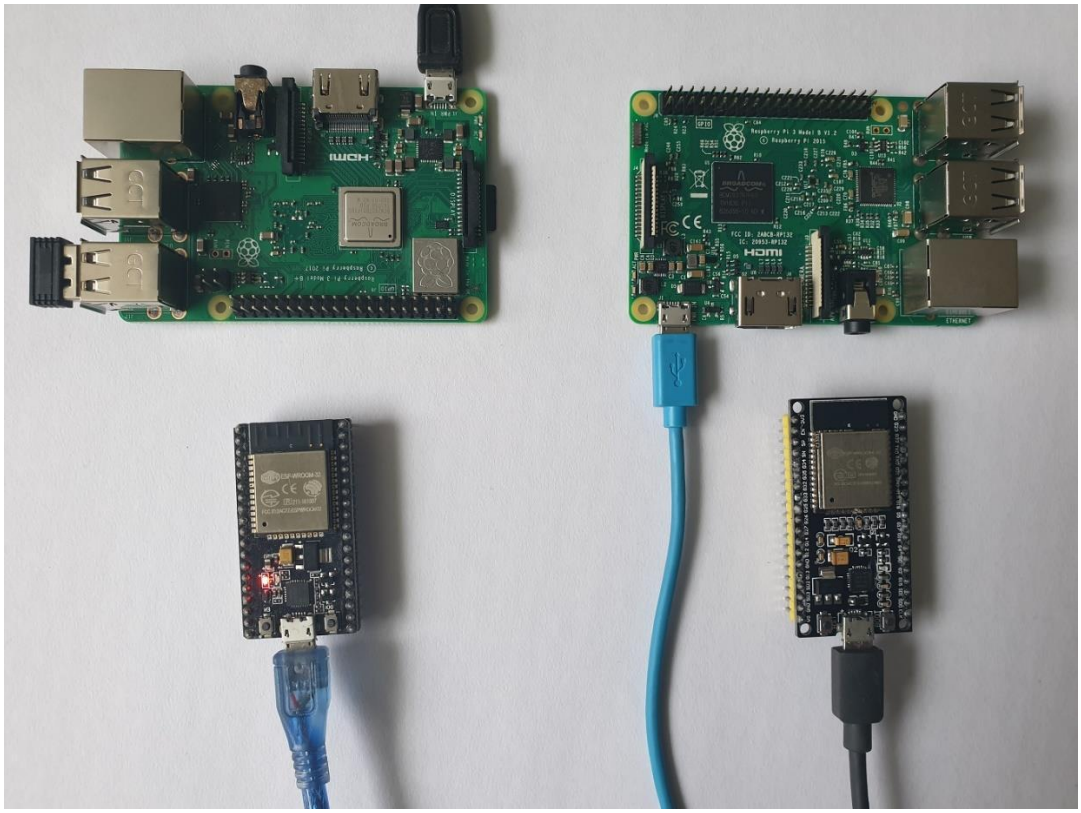


Figure 5.3: Solution Evaluation Prototype

5.3 SECURITY EVALUATION

The principal objective of this project was to design and implement a more secure generic IoT device platform, which can be used to protect wide range of IoT applications. The objective is achieved by developing, a platform which can perform secure enrollment, device authentication, authorization, confidentiality protection, replay attack protection, integrity verification, and secure update delivery. The security measures deployed to achieve the objectives must be strong enough according to the accepted standards to ensure that the platform provides adequate security to practical IoT applications. However, these measures must be implemented within the identified constraints of IoT devices. This section evaluates how far the implemented solution fulfill these objectives.

The developed solution mandates that any connected IoT device must register and authenticate before consuming functions provided by the platform. The device registration and authentication functions are performed using public key cryptography. The public key and private key are generated for a device using the RSA algorithm with key size of 2048 bits, which is accepted as secure enough according to modern security standards. The NIST (National Institute of Standards and Technology) recommends 2048-bit keys for RSA and expect to be sufficient until 2030. Once authenticated, the devices can perform privileged functions of the network and receive the ability store encrypted information on the network. If an IoT device attempt to access the API without authentication, the aggregator produces the following response and rejects the request.

HTTP status code: 403


```
{
  "timestamp": 1566697553678,
  "status": 403,
  "error": "Forbidden",
  "message": "Access Denied",
  "path": "/blocks/all"
}
```


Due to the strong encryption used, the attackers cannot intercept the messages and read the content. However, they can still capture and repeat previous messages in order to perform harmful actions on the IoT device network. To prevent this vulnerability, the IoT device negotiates a sequence with the aggregator device after the authentication. Then the aggregator validates the sequence before

performing any function. Due to this feature, the implemented system is secure against reply attacks. If the sequence is not validated properly or previous sequence number, which is already consumed is received, the aggregator will produce an error response instead of performing the requested function.

HTTP status code: 403


```
{
  "timestamp": 1566698151358,
  "status": 403,
  "error": "Forbidden",
  "message": "Invalid Sequence",
  "path": "/blocks/all"
}
```


The data stored in developed solution are saved to a blockchain network hosted in multiple aggregators. In a blockchain, the blocks are linked using the hash code of the previous block which is calculated by considering all the important fields in the block. Hence, if data or fields in one block changed, it can be easily noticed while validating the blockchain. The implemented solution validates the blockchain during the startup. In addition to the initial validation, the IoT devices or users can request the aggregators to validate their local blockchains anytime they want. Due to this feature, the implemented solution protects the integrity of the stored data which is an inherent property of blockchain based systems.

The developed solution consists of more than one aggregator device and data stored in the local blockchain of one aggregator device are also replicated to several other aggregator devices. When new block is added to the local blockchain of one aggregator, the others are automatically notified to update their blockchains. An IoT device can connect to any other aggregator device and obtain the same state of the data. As typical IoT applications employ several aggregator devices, there are chances of happening unexpected failures in one or more aggregator devices. Due to the data replication feature of the developed solution, no data will be lost in the event of aggregator device failure. The IoT device network can continue to operate by connecting to other aggregator devices, and this feature is important to maintain high availability of the network.

In most of the IoT applications, the aggregator devices are often located in different networks or different physical locations. Even with all the security measures deployed, it is possible an attacker to compromise an IoT aggregator device by gaining unauthorized access. The developed solution

can request data from multiple aggregator devices and returns only the most accepted response to the receiver. This feature is referred to as the hub function in the solution. Due to this feature, compromising few aggregator devices will not affect the reliability of the data received from aggregator devices. In the case of the data stored in the affected aggregator device is corrupted, it can easily sync with the other aggregators during the startup by clearing the local blockchain storage. In the case, the hub function is unable to obtain an acceptable response, it will produce the following response.


```
{
  "timestamp": 1566713476888,
  "status": 500,
  "error": "Error",
  "message": "No acceptable response found"
}
```


In order to ensure that, IoT devices are well protected and to enhance their functionality, the delivery of system updates is essential. However, the attackers can infect the system updates with malicious programs to gain unauthorized access to the system. For this, the attackers might gain control of the connected aggregator or modify the update files during the transit. The developed solution facilitates storing update information securely in the blockchain network. The IoT devices can download the system updates using the information stored in update blocks and verify their integrity with the blockchain before installing them. This allows IoT devices to perform system updates without worrying about malicious modifications.

By considering all those solutions, it is confirmed that developed solution is secure enough to mitigate the risks identified in the objectives of the research.

5.4 AGGREGATOR APPLICATION PERFORMANCE EVALUATION

Unlike general purpose computers and communication networks, the IoT devices and their networks are designed to perform more application specific tasks. The tasks performed by IoT device networks are different from one application to another. However, regardless of the differences, the data storage and communication facilities are used as the foundation to build other high-level tasks. This section and the next section evaluate the usability of the platform with regards to the performance mainly by considering those two basic tasks. The performance and efficiency

of the aggregator device when running the developed Secure IoT solution has been evaluated under several tests. Following sections summarize the results obtained during each test.

5.4.1 Data Storage Response Time

Under this test, the average time taken for the aggregator to complete a data storage request has been measured while increasing the number of data storage requests sent per second. This test has been performed to get an idea about how the implementation behaves when the number of data storage requests are increased either due to the increased number of IoT devices connected or due to the devices which are sending large number of requests while operating. The size of the data stored in each request is set to 5 KB and the average time taken to perform 1000 requests including the network delay has been measured in each test. Figure 5.4 and Figure 5.5 shows the summary of the response times observed in two scenarios.

Without Encryption

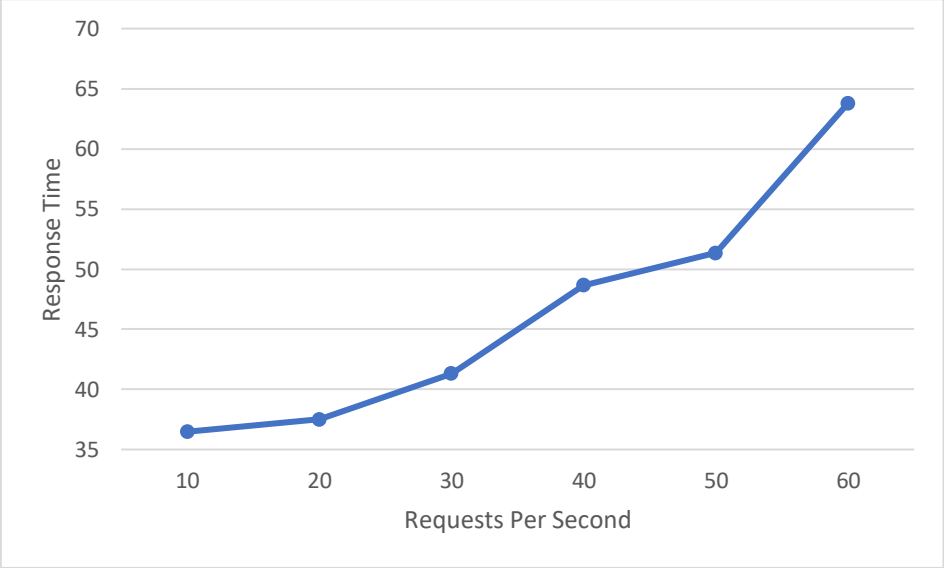


Figure 5.4: Data Storage without Encryption Response Time

With Encryption

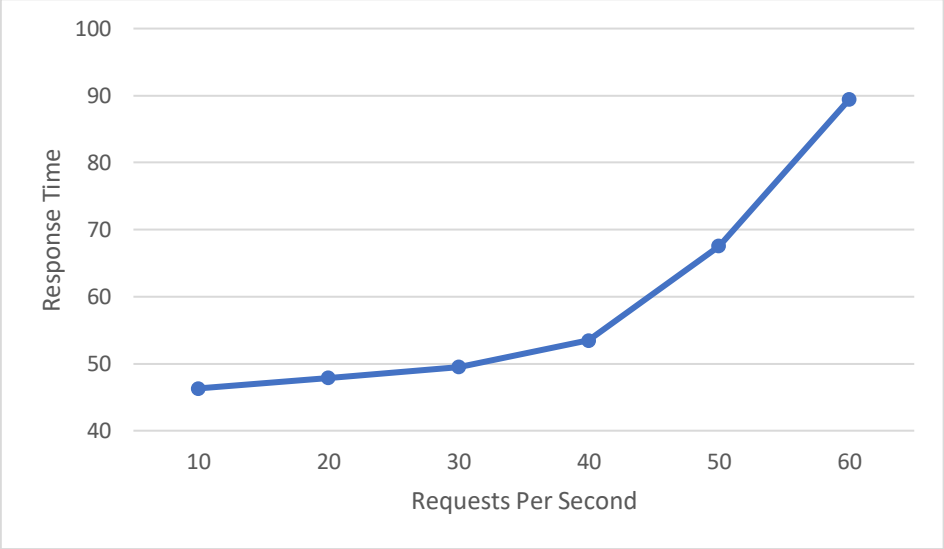


Figure 5.5: Data Storage with Encryption Response Time

The CPU and memory usage of the aggregator device while processing 50 requests per second is shown in Figure 5.6 and Figure 5.7.

Without Encryption

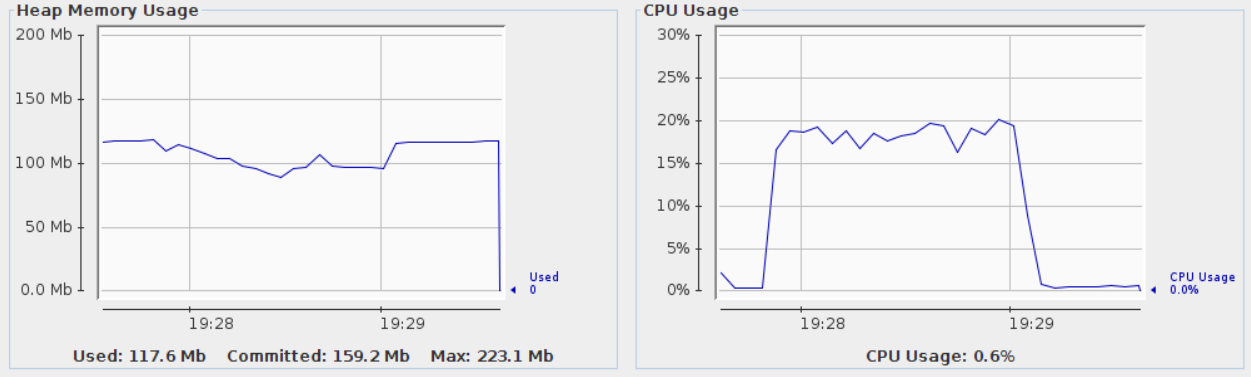


Figure 5.6: Data Storage without Encryption - Memory/CPU Usage

With Encryption

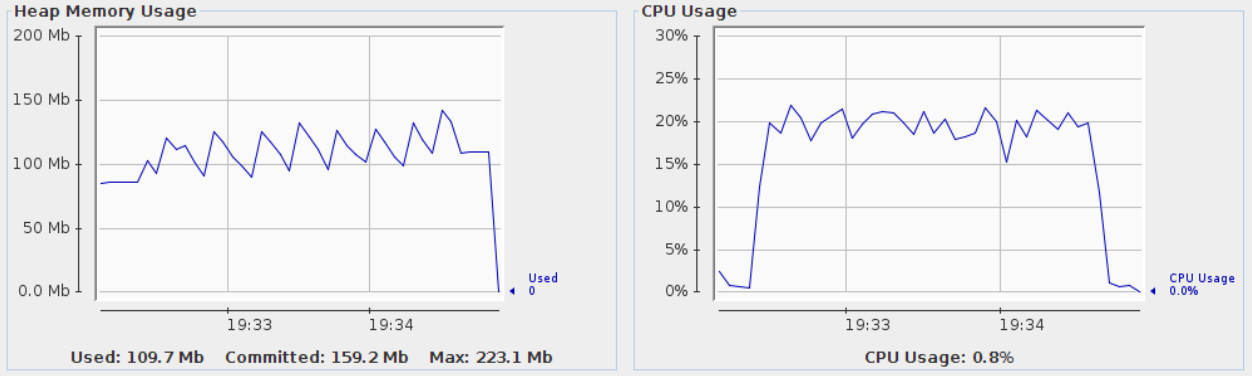


Figure 5.7: Data Storage with Encryption - Memory/CPU Usage

5.4.2 Data Retrieval Response Time

This test has been performed to measure the aggregator response time variation when the number of data retrieval requests are increased. This test allows to get an idea of how the implementation behaves when the number of data retrieval requests increased either due to the increased number of IoT devices connected or due to the devices which are sending large number of requests while operating.

The graph shown in Figure 5.8 is created by measuring the time takes to retrieve 5 KB of data. The average time taken to perform 1000 requests has been measured under each test. The response time has been measured including the network delay.

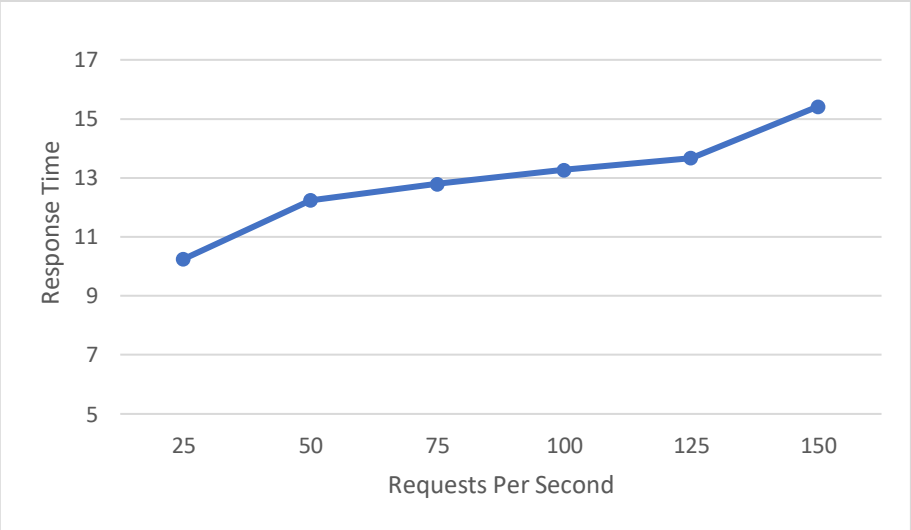


Figure 5.8: Data Retrieval Response Time

The CPU and memory usage of the aggregator device while processing 50 requests per second is shown in Figure 5.9.

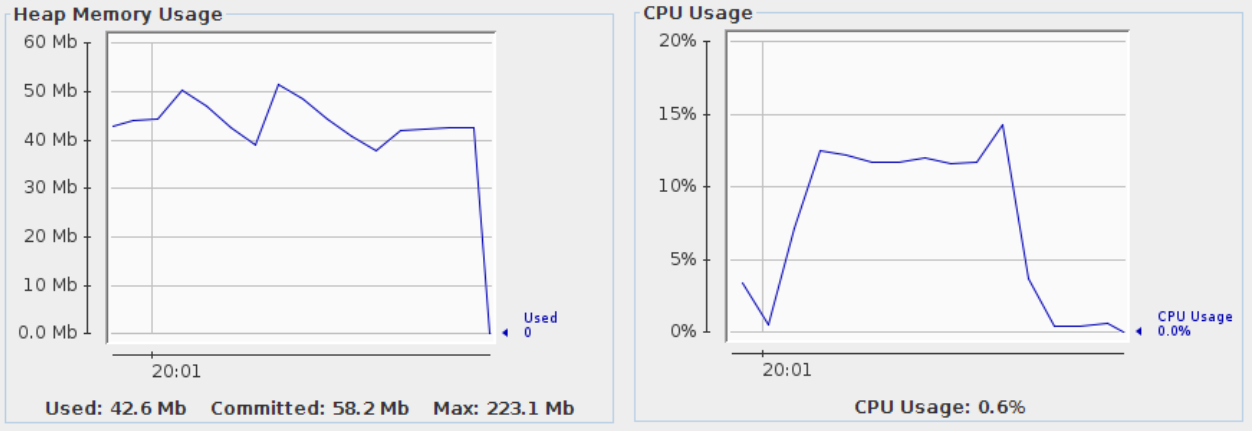


Figure 5.9: Data Retrieval - Memory/CPU Usage

5.4.3 Blockchain Validation Time

This test has been performed to evaluate the time taken to validate the complete blockchain as the number of blocks stored in the aggregator increased.

In following tests, each block contains 5 KB of data and the validation request has been sent 10 times to find the average response time. Figure 5.10 summarizes the results obtained by repeating the test while increasing the number of blocks stored in the blockchain.

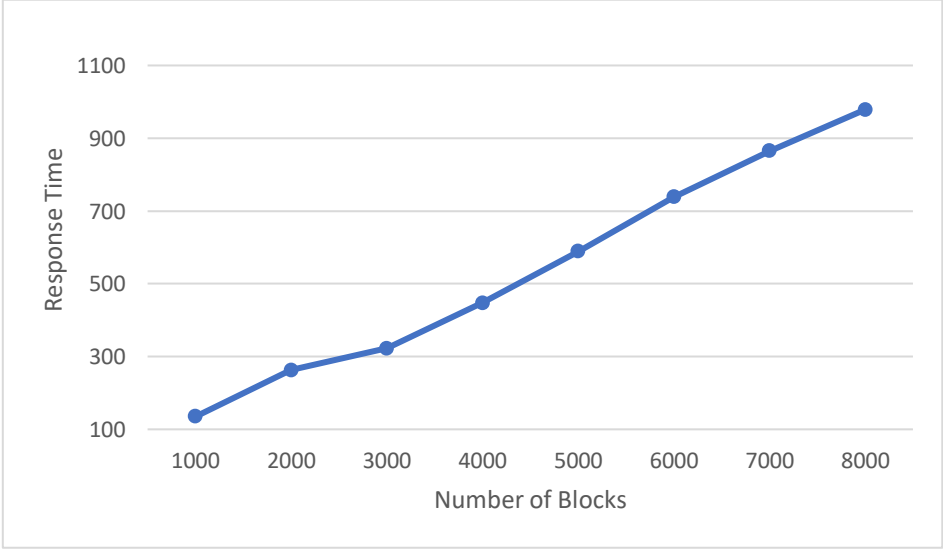


Figure 5.10: Blockchain Validation Time

The CPU and memory usage of the aggregator device while validating 6000 blocks is shown in Figure 5.11.

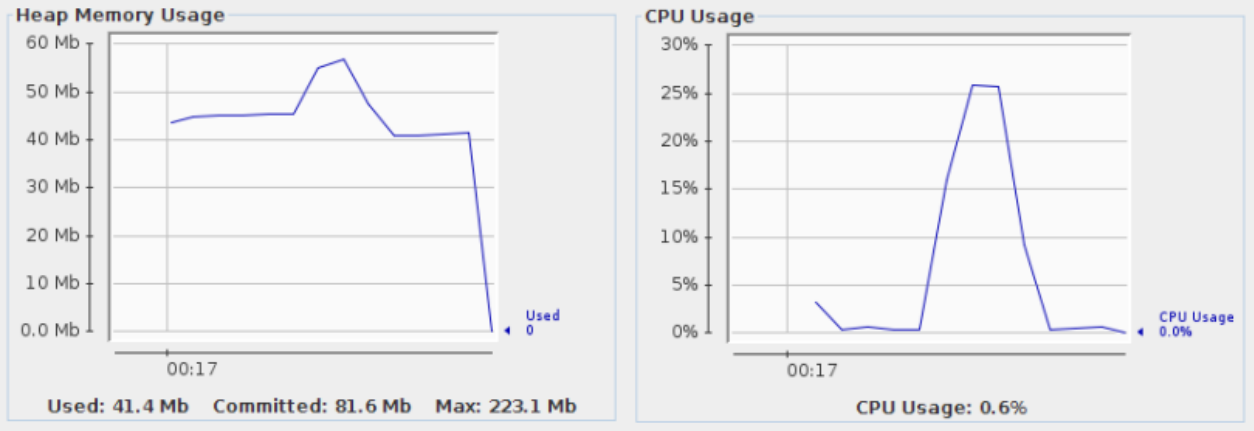


Figure 5.11: Blockchain Validation - Memory/CPU Usage

5.4.4 Aggregator Application Power Consumption

In this test, the power consumption of the aggregator device is measured while increasing the number of transactions made per second. Figure 5.12 shows the setup used to measure the power consumption.

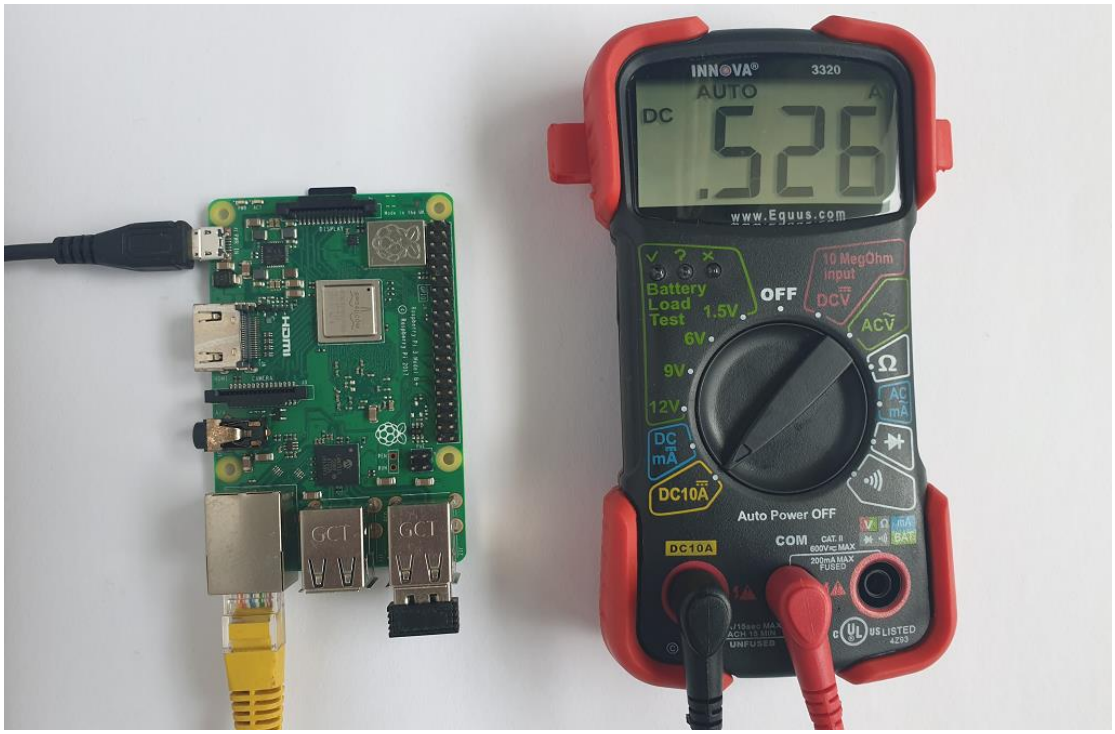


Figure 5.12: Aggregator Device Power Consumption Measurement Setup

The graph shown in Figure 5.13 summarizes the power consumption of the IoT device when the number of block retrieval requests made per second is increased.

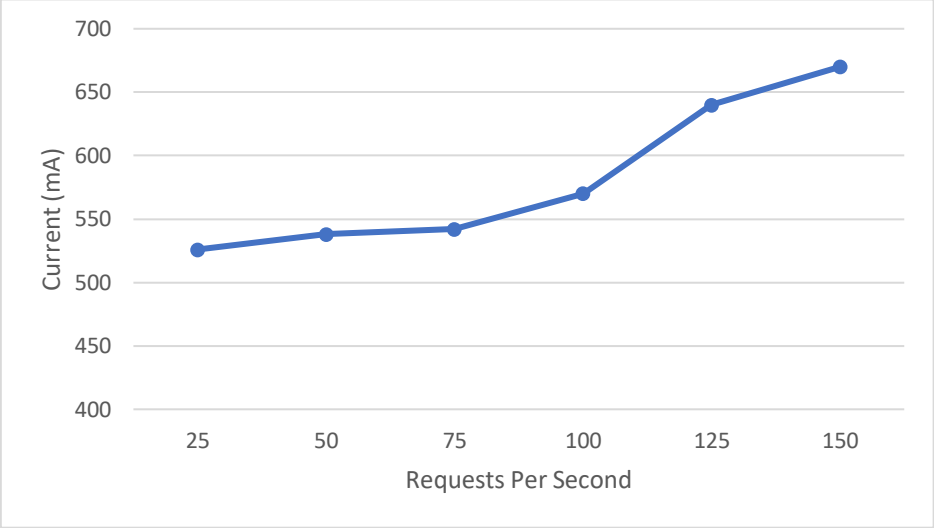


Figure 5.13: Aggregator Device Power Consumption

5.5 IoT DEVICE PERFORMANCE EVALUATION

Several test scenarios have been planned to measure the performance and efficiency of the IoT device when interacting with the aggregator device running the developed Secure IoT solution. The following sections describe each test scenario and summarize the obtained results.

5.5.1 IoT Device Power Consumption for Data Storage

Under this test, the average power consumption of the IoT device has been measured while increasing the number of storage requests generated per second. Each storage request has been designed to send 5 KB of data to the aggregator. The obtained results are summarized in the graph shown in Figure 5.14.

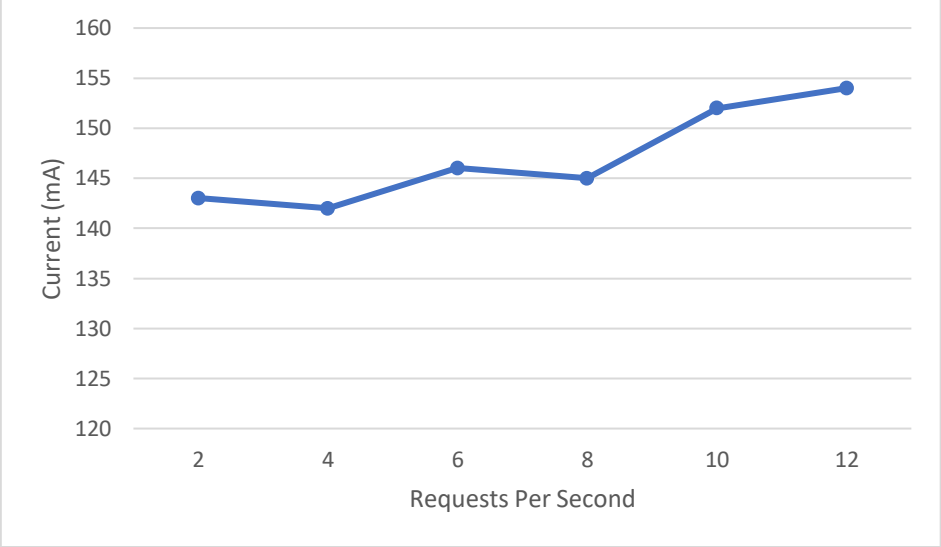


Figure 5.14: IoT Device Power Consumption for Data Storage

5.5.2 IoT Device Power Consumption for Data Retrieval

Under this test, the average power consumption of the IoT device has been measured while increasing the number of data retrieval requests generated per second. Each data retrieval request has been designed to receive 5 KB of data from the aggregator. The obtained results are summarized in the graph shown in Figure 5.15.

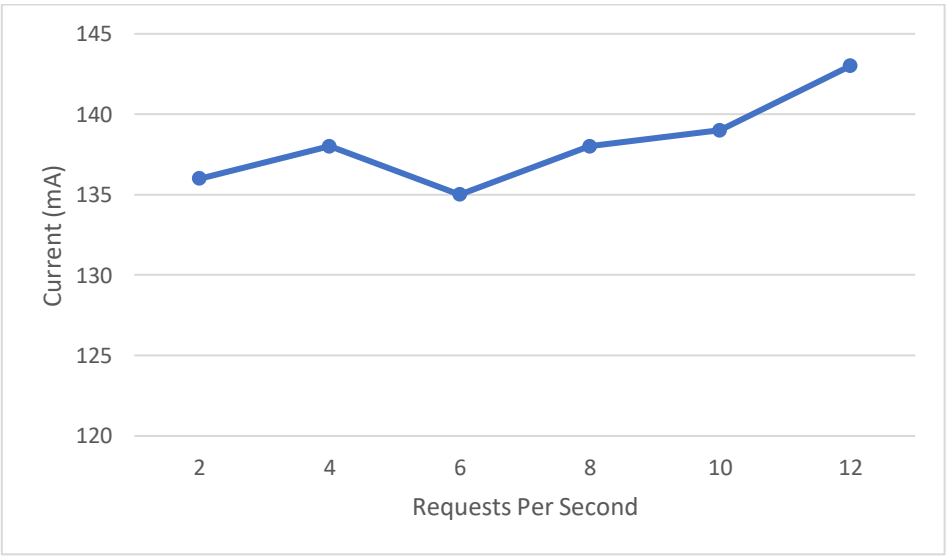


Figure 5.15: IoT Device Power Consumption for Data Retrieval

CHAPTER 6 CONCLUSION AND FUTURE WORK

6.1 OVERVIEW

The IoT devices and networks are playing a major role across various industries by leveraging efficient data collection and communication methods. However, the wide spread of IoT technologies have resulted in many concerns about security and privacy. IoT devices are deployed in many ways and their unique limitations have made the use of conventional security measures to secure IoT device networks less effective. After identifying the importance of security in IoT domain, this project was started to design and develop a generic platform to secure IoT device networks. The main objective of this project is achieved by implementing a platform with following ten sub functions.

1) Blockchain Storage
2) Blockchain Search
3) Blockchain Synchronization
4) Blockchain Validation
5) Device Registration
6) Device Authentication
7) Device Information
8) Hub Function
9) Secure Communication
10) API Playground

The sub function solutions are carefully planned according to industry accepted security requirements and characteristics of IoT devices. After analyzing the current status of each respective research area, a unique solution has been proposed to solve the problem within identified constraints. Then the solution has been implemented for individual sub function, and finally the sub function solutions are combined to propose the complete solution. The proposed solution suggested a blockchain based approach to secure IoT device networks. Blockchain technology which is basically a distributed database has been identified as an emerging technology which can bring integrity and availability properties to IoT networks. The solution is based on a private

blackchin implementation and designed mainly to bring the following three properties of secure systems to IoT networks.

1) Confidentiality
2) Integrity
3) Availability

Each property is achieved within the following unique properties of IoT devices.

- Resource Constraints
- Ad-hoc nature
- Huge variation of devices in IoT networks

The solution has been developed by modelling the IoT device network according to aggregator topology in where multiple low-power IoT devices are connected to an aggregator. The designed solution has been implemented as a re-usable framework which can be used to secure any IoT device network. The application developed in the solution should be implemented in the aggregator as generally the aggregator possesses enough resources to run the program. The aggregator application stores the data received from IoT devices in a blockchain, and data is replicated to other aggregators using blockchain synchronization function. The aggregator can automatically validate the blockchain data and request data from other aggregators in order to update the local blockchain while fixing discrepancies.

When an aggregator has received a data retrieval request, a feature has been implemented to redirect the same request to multiple other aggregators and return the most commonly accepted result to the consumer. Due to this feature, compromising few aggregators in the blockchain network will not affect the reliability of the entire network. A comprehensive RESTful API has been developed for IoT devices and users to interact with any aggregator in the network. The API publishes many useful operations which can be invoked by IoT devices to perform their operations including the authentication. The implementation also provides utilities to handle practical IoT issues like the secure system update handling.

Finally, the developed solution has been evaluated using a prototype IoT network built using Raspberry PI development board and ESP32 modules. The evaluation environment is one of the most common IoT device platforms used by real-world applications. The evaluation process of the

developed platform analyzed the security of the solution as well as the usability of the solution in practical IoT applications. The performance of the IoT platform has been measured to ensure that the implemented security features has minimum impact to the functionality of IoT devices. The results obtained from the evaluation have shown that the solution meets its expected requirements adequately and can be used as a reliable platform to secure IoT networks.

6.2 LIMITATIONS AND KNOWN ISSUES

The following are the limitations and issues identified in the developed solution. These limitations mainly arise due to reasons like duration and scope of the project.

Time Synchronization

There are several functions in the implemented solution (e.g.: Block creation) which use the system time. Each block stored in the blockchain contains a field to store time zone independent UNIX timestamp of block creation time. For these functions, having a synchronized time between aggregators is essential. However, there is no requirement for all aggregators to reside in the same time zone since the system performs operations using time zone independent UNIX timestamp. The designed solution assumes the aggregators have been synchronized their time using an external time synchronization utility. Since most of the operating systems have their built-in time synchronization utilities, this is a valid assumption in initial prototypes of the solution. However, having a built-in time synchronization feature helps to reduce dependencies on external functionalities and becomes important as the application is ported into different platforms in the future.

Fixed Authentication Session

There are some IoT applications where the IoT device is moving from one aggregator to another aggregator while operating. In the current solution, the IoT devices authenticate with one aggregator and authentication session is stored only in the aggregator who performed the authentication process. Hence if the IoT device is moving from one aggregator to another, the device must perform the authentication process again with the new aggregator before sending/receiving data. If the application expects more responsive interaction with the aggregator, re-authentication may not be desirable. To overcome this limitation, “Single sing-on” [34] feature has been described in “Future Improvements” section.

Limited Encryption Algorithm Support

Currently, the implemented system support encryption only using 256-bit AES encryption algorithm. This algorithm is selected by considering its common use in IoT platforms and the availability of shared libraries. Furthermore, the 256-bit AES encryption algorithm is considered as a strong encryption algorithm. However, there can be IoT applications, which prefer specific encryption algorithm over another due to some domain-specific constraints. The solution was implemented by considering this requirement as well, and it is easily extensible to support other cryptographic encryptions.

Blockchain Validation Slowness

With the increased number of blocks stored in the blockchain, the validation process may take a considerable time to complete. A complete blockchain validation is performed when an aggregator application is started, and the aggregator will reject requests until this validation process is completed. As the growth of the data stored in the blockchain, this initialization time may be more noticeable. However, as aggregator devices are not restarted frequently, it is assumed that this has little impact to the usability of the solution.

Compatibility

The aggregator application developed in this solution is platform independent and expose a RESTful API which can be accessed by any program running on any device. During the evaluation phase of the project, the implemented solution was tested in one popular IoT hardware platform. However, the application should be tested in other platforms as well in order to ensure that, the it can maintain the same performance.

6.3 FUTURE IMPROVEMENTS

After performing the solution evaluation, several extra functionalities and improvements which can be used to expand the proposed solution have been identified. The following are some future expansions that can be made to the solution.

Time Synchronization of Aggregator Devices

As described in the "Limitations and Known Issues" section, having a built-in time synchronization mechanism helps to simplify the application deployment on various platforms by reducing external dependencies. There are several widely used protocols exists today to support time synchronization. The Network Time Protocol (NTP) [35] is one such protocol for clock synchronization with UTC time between computer systems over packet-switched, variable-latency data networks. NTP protocol can do so with a high degree of accuracy and a high degree of stability. NTP operates over the User Datagram Protocol (UDP) [36] and uses the concepts of server and client. A server is a source of time information, and a client is a system that is attempting to synchronize its clock to a server. Furthermore, if any of the aggregator device own a built-in GPS [37], it can be used as a more precise time for source.

Single Sign-On for IoT Devices

Single sign-on feature can be implemented in the solution to allow IoT devices to move from one aggregator to another without authenticating again. There are two options has been identified to achieve this functionality.

1) **Shared Authentication Token**: In this approach, the authentication token which is currently stored only in one aggregator should be shared with the other aggregators. Once the authentication token has been shared, those aggregators can also verify the validity of the authentication token independently from the aggregator which performed the initial authentication. The aggregators which has access to the authentication token can be assigned to a one group and the authentication token can be encrypted with a key known only to the group.
2) **Mutual Authentication Token Validation**: The format of the authentication token can be modified to include the device-id of the aggregator performed the authentication. When an aggregator received a new request with an authentication token which is not in the local authentication token pool, the aggregator should extract the device aggregator which performed the authentication from the authentication token. Then the device-id can be used to establish a connection to the corresponding aggregator to validate authentication token. In this approach, the authentication token only resides in the aggregator who performed the initial authentication, and it will answer authentication token validation requests coming

from other aggregators. The authentication token validation results can be cached until a pre-defined time to minimize redundant network calls.

Blockchain Data Indexing

After the blockchain has been successfully validated, the data stored in the blockchain can be indexed using tools like Elasticsearch [38]. Elasticsearch is a search engine developed using Java and based on the Lucene library. Elasticsearch supports full-text search with an HTTP web interface and schema-free JSON documents. When a new block is added to the blockchain, the data inside the block should be added to the index. Since the blockchain stores immutable records, only the insertions are performed on the index. Building an index will help to respond to data search queries much faster than the current implementation. Furthermore, it is expected that the use of an indexer will help to optimize the aggregator resource usage.

Further Compatibility Testing

The developed solution can be tested in other popular IoT platforms, which have enough processing power to function as an aggregator. Following are some of them.

- BeagleBone Blue/Black [39] [40]
- Intel Edison Development Board [41]
- Orange Pi 3 [42]
- Raspberry Pi Zero [43]

The solution can be easily installed and tested in any Java supported platform.

APPENDIX A SYSTEM DOCUMENTATION

This chapter explains the process that should be followed to install the implementation of this project which is the SecureIoT application. The instructions provided in this chapter will be very useful for anyone who is performing the SecureIoT installation or modifications.

SecureIoT solution consists of three main parts, which are the aggregator server, IoT devices and the web client for accessing the API. The web clients do not need any special configuration other than having a compatible web browser. The web server should be installed on a computer or a development board, which has enough capacity to bear the workload caused by IoT devices.

Minimum Hardware Requirements	Minimum Software Requirements
700 MHz processor 256 MB RAM 500 MB storage Network connectivity	Any Java supported operating system. Java Runtime Environment 8 or later.

Table A. 1: Minimum Hardware and Software Requirements to Run the Aggregator Application

ADDITIONAL SOFTWARE REQUIREMENTS

The following software tools will be required to do any modification or extensions to the system.

- Java Development Kit (JDK) 8
- IntelliJ IDEA IDE
- PlatformIO and Visual Studio Code
- Web Browser

APPLICATION INSTALLATION

An easy to use single click runner has been created to start the aggregator server application. To run the SecureIoT aggregator application, copy “secureiot” directory located at [SecuireIoT application folder]\secureiot\ to the storage and run start.bat file. After the SecureIoT application is started it can be tested by accessing the URL http://localhost:8080/api-playground via the web browser. Please note that Java Cryptography Extension (JCE) [44] should be installed before running the application.

APPENDIX B SOFTWARE PROCESS MODELS

There are two main strategies exist when designing solutions for software systems as top-down approach and bottom-up approach. A top-down approach is the process of breaking down a system into subsystems and gaining more understanding about its compositional sub-systems. In bottom-up approach, subsystems are developed individually and then linked together to form the complete top-level system. The solution was developed using a bottom-up approach where the main problem is divided into sub problems and sub problems are individually solved using appropriate technologies under the constraints of the problem. Finally, the sub problem solutions are integrated into a single solution and evaluated to assess the suitability to meet the expected objectives.

The process of developing a software solution to a problem can be performed according to software process models, which describe approaches to a variety of tasks that take place during the process. Even though there are many software process models, two of the most popular process models are described below.

Waterfall Model

Because of the cascade from one phase to another, this model is known as the waterfall. In this model, the following phase should not start until the previous phase has completed. The waterfall model has the advantages of simplicity of implementation and well documented systems. However, waterfall model is inflexible in partitioning the project into distinct stages. Moreover, commitments must be made at an early stage in the process, which makes it difficult to change the system according to evaluation results. Therefore, the waterfall model should only be used when the solution is well understood and unlikely to change frequently during system development.

Steps in waterfall model are shown in Figure B.1.

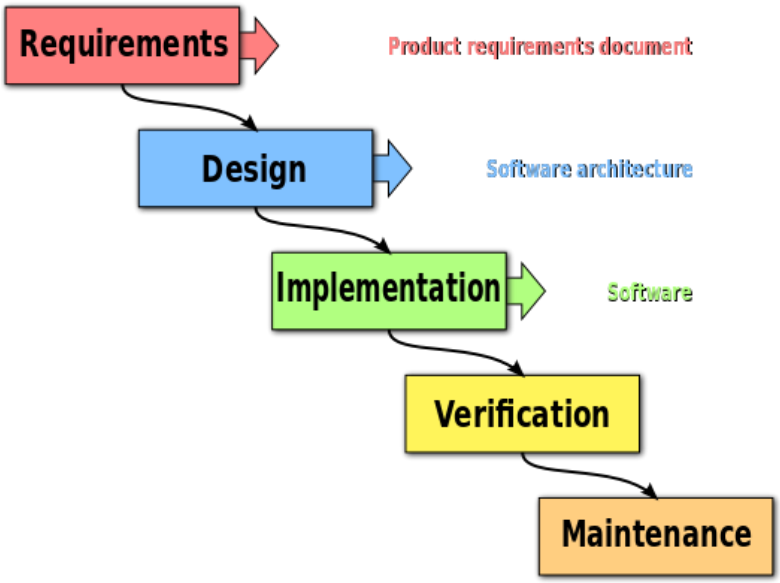


Figure B.1: Waterfall Model for Software Development [45]

Iterative Development

Iterative development is based on the idea of developing an initial implementation, evaluating implementation, and refining it through many versions until an adequate system has been developed. This model is suitable when there is no fixed set of requirements at the beginning of the development process. The main advantage of iterative development is its ability make changes to improve the solution while continuously evaluating the current solution. The specification of the solution can also be developed incrementally. However, the developed solution may be poorly structured due to continuous changes.

Iterative software development process is shown in Figure B.2.

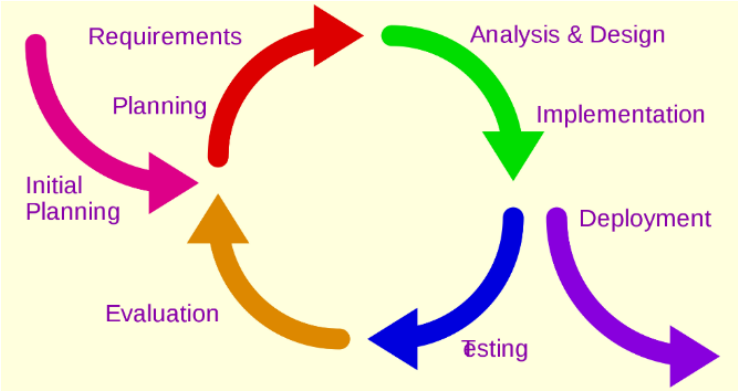


Figure B.2: Iterative Software Development [46]

APPENDIX C DIAGRAMS

CLASS DIAGRAM [47]

The class diagram in the Unified Modelling Language (UML) [48] is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. This appendix presents the class structure of the SecureIoT aggregator application.

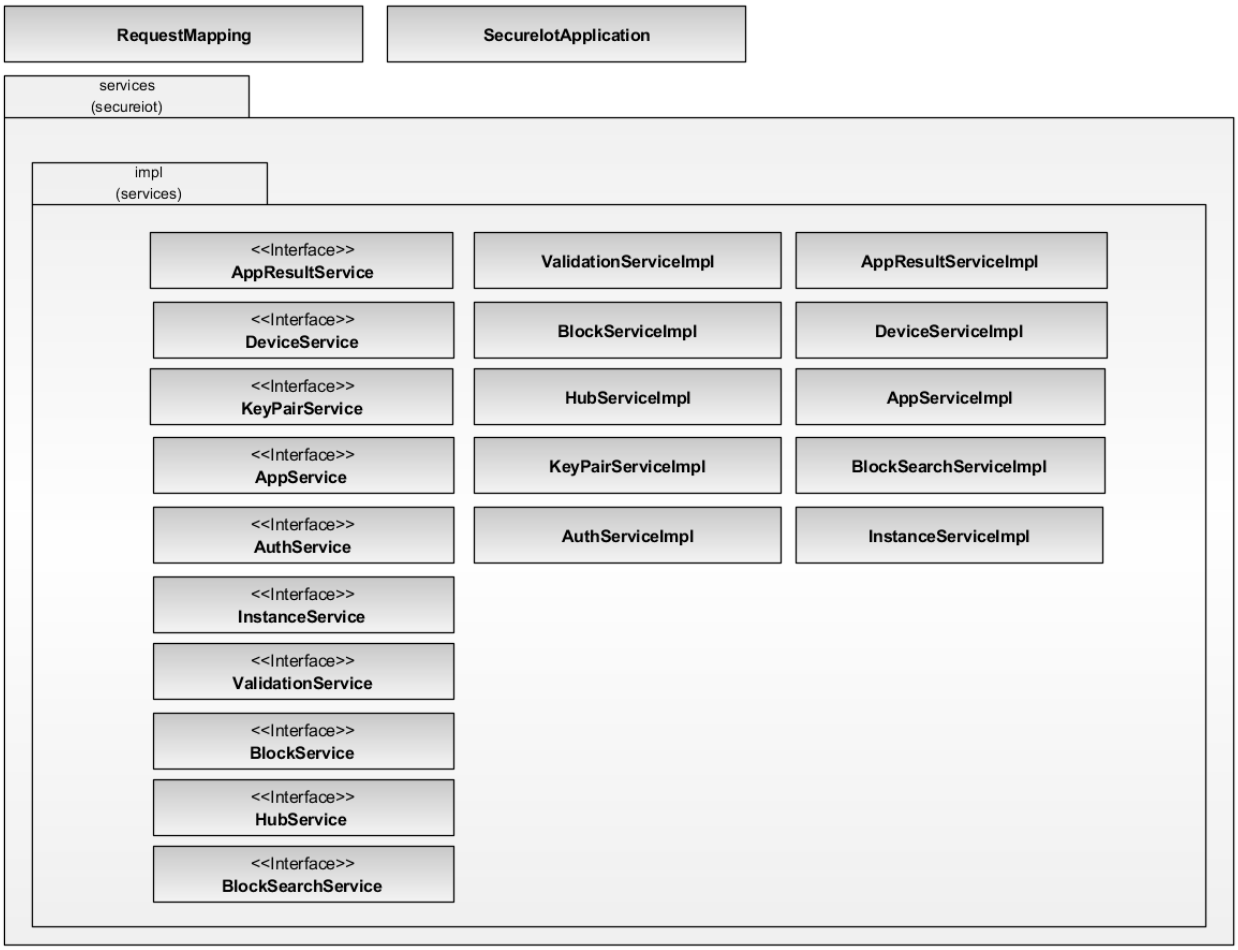


Figure C.1: Class Diagram of Secure IoT Application – Part 1

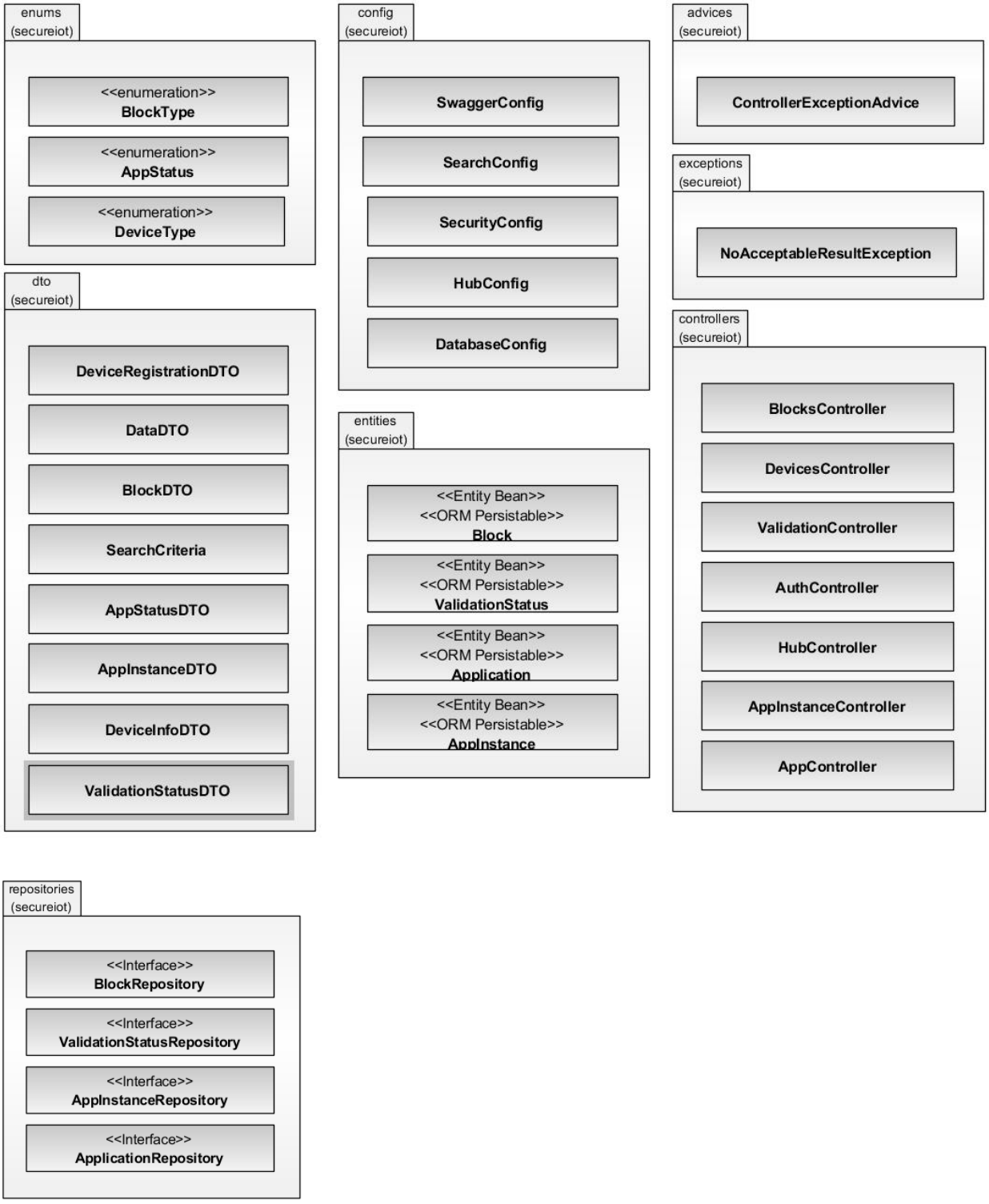


Figure C.2: Class Diagram of Secure IoT Application – Part 2

APPENDIX D IoT DEVICE EVALUATION PROGRAM

Following is the source code of the IoT device program developed to test the solution.

```
#include <Arduino.h>
#include <WiFi.h>
#include <WiFiMulti.h>
#include <HTTPClient.h>

WiFiMulti WiFiMulti;

// Wi-Fi connection details
const char *SSID = "ssid";
const char *PASSWORD = "password";

// Aggregator connection details
const String HOST = "http://192.168.1.118:8080";
const String ENDPOINT_TOTAL_BLOCKS = HOST + "/blocks/total";
const String ENDPOINT_GET_BLOCK = HOST + "/blocks/get?id=4d549b5a-0e34-461a-
bdc2-2a18d52e2fe8";
const String ENDPOINT_STORE_BLOCK = HOST + "/blocks/save";

// Number of requests per second and test limit
const int TPS = 20;
const int TOTAL_REQUESTS = 1000;

// Tests
const byte TOTAL_BLOCKS = 1;
const byte DATA_RETRIVE = 2;
const byte DATA_STORE = 3;

byte selectedTest = 0;

int run = 0;

String DATA_STORE_REQUEST = "{\"data\": \"aa…aa\",\"deviceId\": \"cf1f46c5-
882a-4ed4-8556-95a91e070dc9\",\"encrypted\": false,\"encryptionKey\":
\"aaaaaa\",\"salt\": \"aaaaaa\",\"type\": \"DATA\"}";
void getTotalBlocks()
{
    HTTPClient http;
    long startTime = millis();
    http.begin(ENDPOINT_TOTAL_BLOCKS);
    if (http.GET() == HTTP_CODE_OK)
    {
        String payload = http.getString();
        Serial.println("[" + String(millis() - startTime) + " ms] [" +
ENDPOINT_TOTAL_BLOCKS + "] Total blocks: " + payload);
    }
    http.end();
}

void getBlock()
```

```
{
    HTTPClient http;
    long startTime = millis();
    http.begin(ENDPOINT_GET_BLOCK);
    if (http.GET() == HTTP_CODE_OK)
    {
        String payload = http.getString();
        Serial.println("[" + String(millis() - startTime) + " ms] [" +
ENDPOINT_GET_BLOCK + "] Received block size: " + payload.length());
    }
    http.end();
}

void storeBlock()
{
    HTTPClient http;
    long startTime = millis();
    http.begin(ENDPOINT_STORE_BLOCK);
    http.addHeader("Content-Type", "application/json");
    int status = http.POST(DATA_STORE_REQUEST);
    if (status == HTTP_CODE_OK)
    {
        String payload = http.getString();
        Serial.println("[" + String(millis() - startTime) + " ms] [" +
ENDPOINT_STORE_BLOCK + "] Block saved");
    }
    http.end();
}

// Select a test and run
void peformTest()
{
    switch (selectedTest)
    {
    case TOTAL_BLOCKS:
    {
        getTotalBlocks();
        break;
    }
    case DATA_RETRIVE:
    {
        getBlock();
        break;
    }
    case DATA_STORE:
    {
        storeBlock();
        break;
    }
    }
}

void setup()
{
    Serial.begin(115200);
    delay(10);
    WiFiMulti.addAP(SSID, PASSWORD);
```

```
    Serial.println();
    Serial.println();
    Serial.print("Waiting for Wi-Fi... ");

    while (WiFiMulti.run() != WL_CONNECTED)
    {
        Serial.print(".");
        delay(500);
    }
    Serial.println();
    Serial.print("Wi-Fi connected. IP address: ");
    Serial.println(WiFi.localIP());

    delay(500);
}

void loop()
{
    // Ask to select a test
    if (selectedTest < 1 || selectedTest > 3)
    {
        Serial.println();
        Serial.println("1  - Test block total request");
        Serial.println("2  - Test block retrival");
        Serial.println("3  - Test block storage");
        Serial.print("Please select a test: ");
        while (!Serial.available())
        {
        }
        selectedTest = Serial.parseInt();
        Serial.println();
        Serial.print("Test starting: ");
        Serial.println(selectedTest);
        Serial.println("=====================================");
        Serial.println();
        delay(3000);
    }
    else
    {
        if (run > TOTAL_REQUESTS)
        {
            Serial.println("Test Completed!");
            run = 0;
            selectedTest = 0;
        }
        peformTest();
        run++;
        delay(1000 / TPS);
    }
}
```

REFERENCES

[1] "MD5," Wikipedia, 26 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/MD5. [Accessed 26 05 2019].

[2] "Secure Hash Algorithms," Wikipedia, 26 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/Secure_Hash_Algorithms. [Accessed 26 05 2019].

[3] W. Zhou, Y. Jia, A. Peng, Y. Zhang and P. Liu, "The effect of iot new features on security and privacy: New threats, existing solutions, and challenges yet to be solved," *IEEE Internet of Things Journal*, 2018.

[4] T. Borgohain, U. Kumar and S. Sanyal, "Survey of security and privacy issues of internet of things," *arXiv preprint arXiv:1501.02211*, 2015.

[5] T. Xu, J. B. Wendt and M. Potkonjak, "Security of IoT systems: Design challenges and opportunities," in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, 2014.

[6] F. Olivier, G. Carlos and N. Florent, "New security architecture for IoT network," *Procedia Computer Science*, vol. 52, pp. 1028--1033, 2015.

[7] H. Suo, J. Wan, C. Zou and J. Liu, "Security in the internet of things: a review," in *2012 international conference on computer science and electronics engineering*, 2012.

[8] V. Sivaraman, H. H. Gharakheili, A. Vishwanath, R. Boreli and O. Mehani, "Network-level security and privacy control for smart-home IoT devices," in *IEEE 11th International conference on wireless and mobile computing, networking and communications*, 2015.

[9] T. Heer, O. Garcia-Morchon, R. Hummen, S. L. Keoh, S. S. Kumar and K. Wehrle, "Security Challenges in the IP-based Internet of Things," *Wireless Personal Communications*, vol. 61, pp. 527--542, 2011.

[10] A. Dorri, S. S. Kanhere and R. Jurdak, "Towards an optimized blockchain for IoT," in *Proceedings of the second international conference on Internet-of-Things design and implementation*, 2017.

[11] O. Alphand, M. Amoretti, T. Claeys, S. Dall'Asta, A. Duda, G. Ferrari, F. Rousseau, B. Tourancheau, L. Veltri and F. Zanichelli, "IoTChain: A blockchain security architecture for the Internet of Things," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2018.

[12] "Java (programming language)," Wikipedia, 26 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/Java_(programming_language). [Accessed 26 05 2019].

[13] "Spring Framework," Pivotal Software, 26 5 2019. [Online]. Available: https://spring.io/. [Accessed 26 5 2019].

[14] "Representational state transfer," Wikipedia, 26 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer. [Accessed 26 05 2019].

[15] "H2 Database Engine," H2 Database Engine, 01 05 2019. [Online]. Available: https://www.h2database.com/html/main.html. [Accessed 01 05 2019].

[16] "Zigbee," Wikipedia, 01 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/Zigbee. [Accessed 01 05 2019].

[17] "Zigbee Alliance," Zigbee Alliance, 01 05 2019. [Online]. Available: https://www.zigbee.org/. [Accessed 01 05 2019].

[18] "Zigbee PRO," Zigbee Alliance, 01 05 2019. [Online]. Available: https://www.zigbee.org/zigbee-for-developers/zigbee-pro/. [Accessed 01 05 2019].

[19] "Zigbee Remote Control," Zigbee Alliance, 01 05 2019. [Online]. Available: https://www.zigbee.org/zigbee-for-developers/applicationstandards/zigbee-remote-control/. [Accessed 01 05 2019].

[20] "Zigbee RF4CE," Zigbee Alliance, 01 05 2019. [Online]. Available: https://www.zigbee.org/zigbee-for-developers/network-specifications/zigbeerf4ce/. [Accessed 01 05 2019].

[21] "Zigbee 3.0," Zigbee Alliance, 01 05 2019. [Online]. Available: https://www.zigbee.org/zigbee-for-developers/zigbee-3-0/. [Accessed 01 05 2019].

[22] "Zigbee IP and 920IP," Zigbee Alliance, 1 5 2019. [Online]. Available: https://www.zigbee.org/zigbee-for-developers/network-specifications/zigbeeip/. [Accessed 1 5 2019].

[23] "HomePlug," Wikipedia, 01 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/HomePlug. [Accessed 01 05 2019].

[24] "CCM mode," Wikipedia, 01 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/CCM_mode. [Accessed 01 05 2019].

[25] "Wi-Fi," Wikipedia, 01 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/Wi-Fi. [Accessed 01 05 2019].

[26] "Wi-Fi HaLow," Wi-Fi Alliance, 01 05 2019. [Online]. Available: https://www.wi-fi.org/discover-wi-fi/wi-fi-halow. [Accessed 01 05 2019].

[27] "SecureRandom," Oracle Corporation, 01 05 2019. [Online]. Available: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/security/SecureRandom.html. [Accessed 01 05 2019].

[28] "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry," Internet Assigned Numbers Authority (IANA), 04 05 2019. [Online]. Available: https://www.iana.org/assignments/http-authschemes/http-authschemes.xhtml. [Accessed 04 05 2019].

[29] "The OAuth 2.0 Authorization Framework: Bearer Token Usage," Internet Engineering Task Force (IETF) , 04 05 2019. [Online]. Available: https://tools.ietf.org/html/rfc6750. [Accessed 04 05 2019].

[30] "OAuth," Wikipedia, 04 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/OAuth. [Accessed 04 05 2019].

[31] "WAR (file format)," Wikipedia, 04 05` 2019. [Online]. Available: https://en.wikipedia.org/wiki/WAR_(file_format). [Accessed 04 05 `2019].

[32] "Raspberry Pi," RASPBERRY PI FOUNDATION, 05 05 2019. [Online]. Available: https://www.raspberrypi.org/. [Accessed 05 05 2019].

[33] "ESP32," ESPRESSIF SYSTEMS, 05 05 2019. [Online]. Available: https://www.espressif.com/en/products/hardware/esp32/overview. [Accessed 05 05 2019].

[34] "Single sign-on," Wikipedia, 06 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/Single_sign-on. [Accessed 06 05 2019].

[35] "Network Time Protocol," Wikipedia, 06 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/Network_Time_Protocol. [Accessed 06 05 2019].

[36] "User Datagram Protocol," Wikipedia, 06 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/User_Datagram_Protocol. [Accessed 06 05 2019].

[37] "Global Positioning System," Wikipedia, 05 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/Global_Positioning_System. [Accessed 05 05 2019].

[38] "Elasticsearch," Elastic, 06 05 2019. [Online]. Available: https://www.elastic.co/products/elasticsearch. [Accessed 06 05 2019].

[39] "BeagleBone Blue," BeagleBoard.org Foundation , 06 05 2019. [Online]. Available: https://beagleboard.org/blue. [Accessed 06 05 2019].

[40] "BeagleBone Black," BeagleBoard.org Foundation , 06 05 2019. [Online]. Available: https://beagleboard.org/black. [Accessed 06 05 2019].

[41] "Intel Edison," Wikipedia, 06 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/Intel_Edison. [Accessed 06 05 2019].

[42] "Orange Pi 3," Orange Pi, 06 05 2019. [Online]. Available: http://www.orangepi.org/Orange%20Pi%203/. [Accessed 06 05 2019].

[43] "Raspberry Pi Zero," Raspberry Pi Foundation, 06 05 2019. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-zero/. [Accessed 06 05 2019].

[44] "Java Cryptography Extension (JCE) Unlimited Strength," Oracle, 26 05 2019. [Online]. Available: https://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html. [Accessed 26 05 2019].

[45] "Waterfall model," Wikipedia, 25 5 2019. [Online]. Available: https://en.wikipedia.org/wiki/Waterfall_model. [Accessed 25 5 2019].

[46] "Iterative and incremental development," Wikipedia, 25 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/Iterative_and_incremental_development. [Accessed 25 05 2019].

[47] "Class diagram," Wikipedia, 26 5 2019. [Online]. Available: https://en.wikipedia.org/wiki/Class_diagram. [Accessed 26 5 2019].

[48] "Unified Modeling Language," Wikipedia, 26 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/Unified_Modeling_Language. [Accessed 26 05 2019].

[49] "HTTP Status Code Definitions," W3C, 04 05 2019. [Online]. Available: https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html. [Accessed 04 05 2019].

[50] "JAR (file format)," Wikipedia, 04 05 2019. [Online]. Available: https://en.wikipedia.org/wiki/JAR_(file_format). [Accessed 04 05 2019].