Scalable Blockchain-assisted Log Storage System for Cloud-generated Logs

by

William Pourmajidi

Bachelor of Information Technology, York University, 2016

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the program of

Computer Science

Toronto, Ontario, Canada, 2018

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my dissertation may be made electronically available to the public.

Scalable Blockchain-assisted Log Storage System for Cloud-generated Logs

Master of Science 2018

William Pourmajidi

Computer Science

Ryerson University

Abstract

During the normal operation of a Cloud solution, no one pays attention to the logs except the technical department, which may periodically check them to ensure that the performance of the platform conforms to the Service Level Agreements. However, the moment the status of a component changes from acceptable to unacceptable, or a customer complains about accessibility or performance of a platform, the importance of logs increases significantly. Depending on the scope of the issue, all departments, including management, customer support, and even the actual customer, may turn to logs to find out what has happened, how it has happened, and who is responsible for the issue. The party at fault may be motivated to tamper the logs to hide their fault. Given the number of logs that are generated by the Cloud solutions, there are many tampering

opportunities. While tamper detection solution can be used to detect any changes in the logs, we argue that the critical nature of logs calls for immutability. In this thesis, we propose a blockchain-based log system, called Logchain, that collects the logs from different providers and avoids log tampering by sealing the logs cryptographically and adding them to a hierarchical ledger, hence, providing an immutable platform for log storage.

Acknowledgements

This master's thesis is submitted to fulfill the requirements of the MSc of Computer Science at Ryerson University in Toronto, Canada. The work carried out in this thesis was supervised by Dr. Andriy Miranskyy.

First and foremost, I would like to express my sincere gratitude to Dr. Miranskyy for his determined support and guidance throughout my study as a graduate student. Dr. Miranskyy has set an example of excellence as a researcher, professor, mentor and a role model and in all cases, whenever I have reached out for assistance, he has provided me with the direction and the required resources to pursue my academic goals. I could not have imagined having a better supervisor and mentor for my MSc studies.

Besides my supervisor, my sincere thanks goes to all the members of my thesis committee. I thank my fellow labmates in Ryerson AMiR Lab: Mujahid Sultan, Sravya Polisetty, Sheik Mamun, and Lei Zhang.

This journey would not have been possible without the support of my family. I would like to thank my wife Sara, whose love, support, and encouragement made this possible.

Dedication

To my wife, Sara, who has been a constant source of support during my graduate studies. Thank you for all your support and encouragement, for your faith and your confidence in me. You made this endeavour easier.

Contents

Declaration ii
Abstract iii
Acknowledgements v
Dedication vi
List of Tables xi
List of Figures xii

1 Introduction **1**
1.1 Terminology 3
1.1.1 Cloud Computing 3
1.1.2 Infrastructure as a Service (IaaS) 3
1.1.3 Platform as a Service (PaaS) 3
1.1.4 Software as a Service (SaaS) 3
1.1.5 Cloud Service Provider (CSP) 4
1.1.6 Cloud Service Consumer (CSC) 4
1.1.7 Cloud Operational Log (COL) 4
1.1.8 Log Tampering 4
1.1.9 Service Level Agreement (SLA) 4
1.1.10 Chain of Accountability 6
1.1.11 Blockchain 6
1.2 Cloud Computing and Cloud Monitoring 7
1.3 Cloud Monitoring and its Challenges 8
1.4 Trust Among Cloud Participants 9
1.5 Logs and Their Importance 10
1.6 Tamper-motivation and Log Tampering 11

1.7 Motivation . . . 13
1.8 Objective . . . 13
1.9 Proposed Solution . . . 14
1.10 Novelty and Contribution . . . 15
1.11 Outline . . . 16

2 Literature Review **17**
2.1 Cloud Logs and Related Challenges . . . 17
2.2 Digital Forensic and its Principles . . . 19
2.3 Digital Evidence and its Reliability . . . 19
2.4 Logs as Digital Evidence . . . 21
2.5 Log Tampering Prevention/Detection Solutions . . . 22
2.6 Blockchain . . . 23
2.7 Trustable Log Management Systems . . . 25
2.8 Blockchain Scalability Solutions . . . 27

3 Methodology **29**
3.1 Common Key Components of Blockchains . . . 30
3.1.1 Genesis Block (GB) . . . 30
3.1.2 Data Block (DB) . . . 30
3.1.3 Blockchain (BC) . . . 31
3.1.4 Mining and Hash Binding . . . 31
3.1.5 Blockchain Immutability Features . . . 35
3.1.6 Blockchain Characteristics . . . 36
3.2 Logchain as a Service (LCaaS) . . . 37
3.3 Key Components of Logchain as a Service (LCaaS) . . . 40
3.3.1 Absolute Genesis Block (AGB) . . . 40
3.3.2 Relative Genesis Blocks (RGB) . . . 40
3.3.3 Terminal Blocks (TB) . . . 41
3.3.4 Circled Blockchains (CB) . . . 44
3.3.5 Super Blocks (SB) . . . 44
3.3.6 Super Blockchain (SBC) . . . 46
3.4 LCaaS Implementation . . . 46

3.4.1 Programming Language, Libraries, Code Repository 46
3.4.2 LCaaS Modules 47
3.4.3 Configuration 50
3.4.4 Process Flow 51
3.4.5 Blocks and Their Types 54
3.4.6 Block Presentation 55
3.4.7 Persistent Storage of Blocks 56
3.5 LCaaS Application Programming Interface (API) 59
3.5.1 Submission Methods 59
3.5.2 Verification Methods 61
3.6 Applicability to other Blockchain Platforms 67

4 Evaluation **68**
4.1 Current Blockchain Platforms 68
4.1.1 Public Blockchains 69
4.1.2 Private Blockchains 69
4.2 Blockchain as a Service 71
4.3 Ethereum 71
4.4 LCaaS and Ethereum Integration 72
4.5 Ethereum Blockchain and Test Blockchain Environments 74
4.6 Implementation 74
4.6.1 Additional Python Libraries 74
4.6.2 Test Networks 75
4.6.3 Test Ether and Secure Vault 75
4.6.4 Publishing a Smart Contract Using Solidity 76
4.6.5 Interaction with Ethereum Ropsten and the Deployed Smart Contract 78
4.7 Results Verification 81

5 Conclusion and Future Work **83**
5.1 Conclusion 83
5.2 Future Work 84
5.2.1 LCaaS Improvement 85

5.2.2 Hierarchical Ledger Implementation on other Blockchains 86

Appendices **87**

A Index Module (index.py) **88**

B Logchain Module (LC.py) **111**

C Blockchain Module (blockchain.py) **116**

D Ethereum Module (ethereum.py) **121**

E Contract ABI (contract-abi.py) **126**

F Superblock Smart Contract (Superblock.sol) **129**

References **131**

Acronyms **143**

List of Tables

3.1 Python Libraries Used in LCaaS 47
3.2 Configuration Items for LCaaS 51
3.3 Indexes Used in `Logchain` Class to Keep Track of the Most Recent Added Block and its Details 52

4.1 Additional Configuration Item Introduced to LCaaS 79

List of Figures

1.1 The CSP and CSC Responsibilities Related to each Service Offering . . . 5
1.2 The Chain of Accountability and Escalation Path for SLAs 7

2.1 The Investigative Process for Digital Forensic as per [11] 20

3.1 Original Genesis Block and its Elements 30
3.2 Data Block and its Elements 31
3.3 Probability of Desired Outcome at Each Trial as Number of Zeros Grows 32
3.4 Hash Relationship Between Data Block i and Data Block $i+1$ 33
3.5 Key Characteristics of Blockchain 36
3.6 Two-level Hierarchy as Implemented by LCaaS 37
3.7 Architectural Components of LCaaS 38
3.8 Absolute Genesis Block (AGB) and Relative Genesis Block (RGB) in Circled Blockchain(0) and Circled Blockchain(1) 41
3.9 JavaScript Object Notation (JSON) Object in the Data Element of a Terminal Block 42
3.10 The Relation Between Terminal Block and All other Blocks in a Circled Blockchain 43
3.11 JSON Object in the Data Element of a Super Block 45
3.12 The Relationship Between Terminal Block and Super Block 45
3.13 Interconnection Among Classes in Logchain Module 49
3.14 Interconnection Among Classes in Blockchain Module 50
3.15 The Internal Logic of `create_new_block` Method 53
3.16 The Internal Logic of `create_new_block` Method 54
3.17 A Terminal Block `data` Element Shown as an Object 55

3.18 The Stringified `data` Element of a Terminal Block 56
3.19 JSON Objects Stored on the Firebase Database 57
3.20 The Dashboard of the Firebase Showing Buckets for Circled Blockchains and Super Blocks 58
3.21 An Example of `submit_raw` Method and the LCaaS HTTP Response . . 60
3.22 An Example of `submit_digest` Method and the LCaaS HTTP Response 61
3.23 An Example of `verify_raw` Method and the LCaaS HTTP Response . . 62
3.24 An Example of `verify_digest` Method and the LCaaS HTTP Response 63
3.25 The Internal Logic of the `verify_raw` and `verify_digest` Methods 64
3.26 An Example of `verify_tb` Method and the LCaaS HTTP Response . . . 65
3.27 The Internal Logic of the `verify_tb` Method 66

4.1 The Difference between Public and Private Blockchains 70
4.2 LCaaS and Ethereum Integration 73
4.3 Test Ether Purchase via MetaMask Faucet 75
4.4 Logchain Balance 76
4.5 Transaction for Publishing a Smart Contract 77
4.6 Superblock Smart Contract Successfully Added on Ethereum Blockchain 78
4.7 Sender Address Verification 80
4.8 Ethereum Smart Contract Receipt 81
4.9 Super Block Submission To Ethereum Blockchain 82

Chapter 1

Introduction

In the majority of Cloud offerings, there are two parties involved. The Cloud Service Provider (CSP) owns a shared pool of configurable computing resources and offers computing and storage services, at a predefined price, via Internet, to a Cloud Service Consumer (CSC).

During the normal operation of a Cloud platform, although many logs are being collected and stored, no one pays attention to collected logs except the technical operation department, which may check these logs periodically. The continuous monitoring of all resources on the Cloud is an effort by the CSP to ensure that the current performance of the Cloud platform and the Quality of Service (QoS) that is provided to CSC match the ones that are promised to them in the signed Service Level Agreement (SLA). When a technical issue arises, or a Cloud service delivery is interrupted, the collected logs become the most important source of the troubleshooting and tracing efforts by the technical operations department. Depending on the scope of the technical issue, some or many of the departments of the CSP will get involved to analyse the logs and to draw conclusions on important matters such as what has happened, how it has happened, and who is responsible for the incident. Cloud service delivery interruptions or outages can directly impact a CSC; in many cases, the CSC will be one the of parties that becomes interested in reviewing and assessing the logs.

Logs contain very sensitive information and details about offered services. For example, operational logs indicate how and at what capacity a system has been operating, and network logs include all incoming and outgoing packets of a deployed solution on the

Cloud platform. These logs hold the truth about the delivered QoS and can be used as legal evidence in the court of law [102].

Logs are generated and collected by various monitoring solutions that a CSP has deployed on the Cloud infrastructure. In fact, full access to all resources (e.g., bare-metal servers, networking components, cloud management platforms, virtualisation tools, etc.) is required to deploy holistic monitoring solutions [84], and such access is only available to the CSP. While the full control over monitoring systems and generated logs allow a CSP to monitor and maintain Cloud services efficiently, it gives them a controversial power over evidential resources that are significantly important to CSCs. That is, logs are generated and stored on a platform that is built,managed, and owned by the CSP. Hence, CSPs have read, write, and modify permissions on all collected logs.

The majority of CSPs provide a monitoring dashboard to their CSCs. These dashboards are used by CSCs to view, analyse, and export logs that may seem useful for generating reports or other technical tasks. While using these tools, the CSCs have to trust that the information provided to CSP is genuine and has full integrity, in other words, has not been tampered. Ironically, almost in all cases, the CSCs have no option to test and verify the integrity of the logs that are provided to them. Without an option to verify the integrity of the provided logs, CSCs are in a very weak position at the times of QoS disputes. Such disadvantage causes many trust related issues.

Acknowledging this issue, some researchers [120, 86, 65, 104] have suggested the use of a Trusted Third-Party (TTP) as a mediator between the CSPs and CSCs. This external trusted entity [99] solves the trust issue and can be used as an effective solution. However, this solution adds an additional layer that is a constraint. Additionally, trusting a TTP requires taking the risk of assuming that it will always act as an honest mediator [93].

Hence, finding and deploying a solution that can guarantee the integrity of the logs that are provided to the CSC, without relying on a TTP, is of paramount importance. Such a solution provides peace of mind to both parties and establishes a trustworthy relationship between them.

In this work, we propose a blockchain-based log storage system, called Logchain, that collects logs generated on a CSP platform and stores them in cryptographically sealed and immutable blocks that are linked together by a hash binding relationship, resulting in a blockchain. Logchain stores blocks in a hierarchical ledger to provide the capacity and performance that is needed for a storage system that deals with hundreds of transactions

per second. To make the proposed solution more accessible and to increase its usability, we propose an Application Programming Interface (API) that converts Logchain to an "as a service" delivery method and can be referred to as Logchain as a Service (LCaaS).

1.1 Terminology

Throughout the study the following **terminology** is used.

1.1.1 Cloud Computing

As defined by NIST [30], "cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.".

1.1.2 Infrastructure as a Service (IaaS)

This is the most advanced level of offering. In this model, the CSP shares all its raw resources to the CSC and the CSC is responsible for deploying Virtual Machine (VM), network and security settings for the created VMs. Obviously, this model requires the CSC to be tech-savvy and aware of all Cloud offerings.

1.1.3 Platform as a Service (PaaS)

In this service model, the CSP provides their customers with a computation platform (usually in the form of a VM) and a set of pre-installed software, application and development packages. The CSC can choose to develop, test, and deploy any type of application they wish and enjoy Cloud features such as scalability and elasticity.

1.1.4 Software as a Service (SaaS)

In this service model, the entire infrastructure, platform, and running environment are managed by the CSP. In this model, the CSCs are using thin clients (mainly web browsers) to interact with the provided software or application.

1.1.5 CSP

An organisation that creates a shared pool of configurable computing resources and provides it to Cloud consumer in a service model that is governed by a mutually agreed Service Level Agreement. Based on the required service, a Cloud provider can choose any of IaaS,PaaS, and SaaS offerings [67].

1.1.6 CSC

Any person, organisation or business entity that uses one or more of the offerings (in forms of IaaS, PaaS, SaaS) by a CSP. In addition to their financial obligation, paying monthly bills, CSCs are responsible for their usage of resources and the content they upload or generate on the Cloud platform. Figure 1.1 depicts the CSP and CSC responsibilities related to each service offering.

1.1.7 Cloud Operational Log (COL)

Logs are among the most important pieces of analytical data in Cloud-based services [76]. COL is generated during the operation of a Cloud platform and is stored for future analysis. COLs consist of hardware, infrastructure, network and application logs.

1.1.8 Log Tampering

Introduction of errors, losses, modifications or removal of one or all part of logs. Log tampering can happen at any time including before and after the storage of logs [52].

1.1.9 SLA

CSPs offer their service to CSCs at a predefined rate and with a predefined set of QoS characteristics. All these characteristics and their acceptable values will be defined in an SLA that legally binds the two parties. Any deviation from the SLA considered a breach of agreement and is subject to a legal action. The complex nature of the Cloud landscape calls for exhaustive and comprehensive SLAs [86].

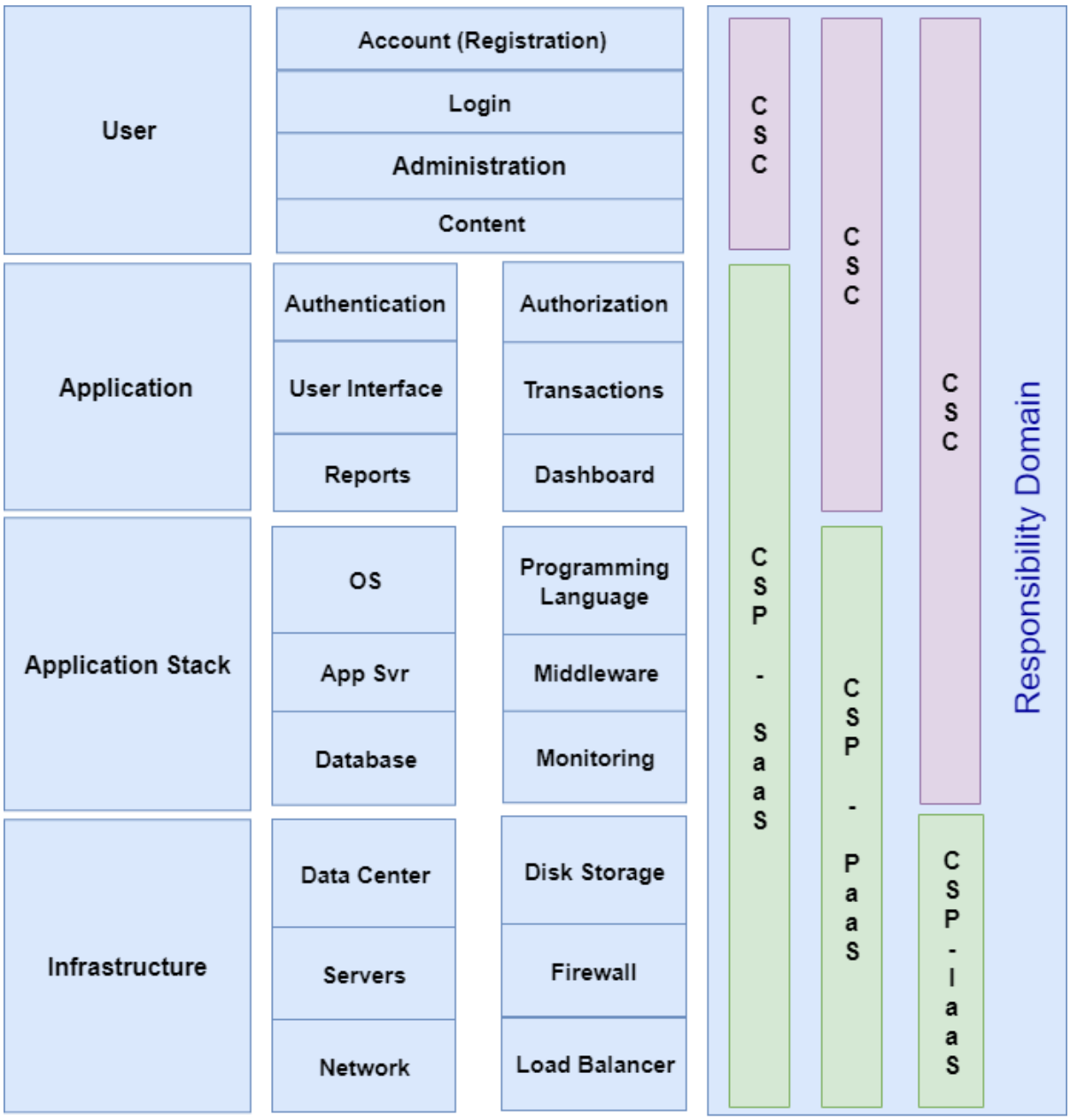

Figure 1.1: The CSP and CSC Responsibilities Related to each Service Offering

1.1.10 Chain of Accountability

To build a Cloud infrastructure, the CSPs need to combine the services and products of many providers. Hardware providers, network providers, operating system providers, and application providers are among vendors that CSPs use to create their offerings to the CSCs. Here are the key players that a CSP needs to interact with [98].

1.1.10.1 Infrastructure Supplier

Provides all required items for the cloud infrastructure such as Hardware, Software, Storage, Servers, network, and Hypervisors. Examples of major suppliers are IBM, Dell, HP, and Microsoft.

1.1.10.2 Internet Protocol (IP) Network Provider

Provides the required internet backbone for the cloud resources so that Cloud services can be served to the CSCs. Examples of major IP network providers are AT&T, Verizon, Bell, and BT.

While CSCs have a signed SLAs with CSPs, CSPs have many signed SLAs with their vendors. If a CSC escalates an issue to a CSP, the CSP may find out that it needs to escalate the reported issue to one of its vendors. Figure 1.2 shows the chain of accountability and escalation path for the SLAs.

1.1.11 Blockchain

Blockchains offer immutability by replying on a distributed digital ledger structure. With no central authority, blockchain stores data in blocks, and each block is linked to another block by a hash binding relationship. In other words, the hash of the current block, includes the hash of the previous block and so on. In a blockchain, any changes in the previous blocks will change their hashes, hence, breaking the chain.

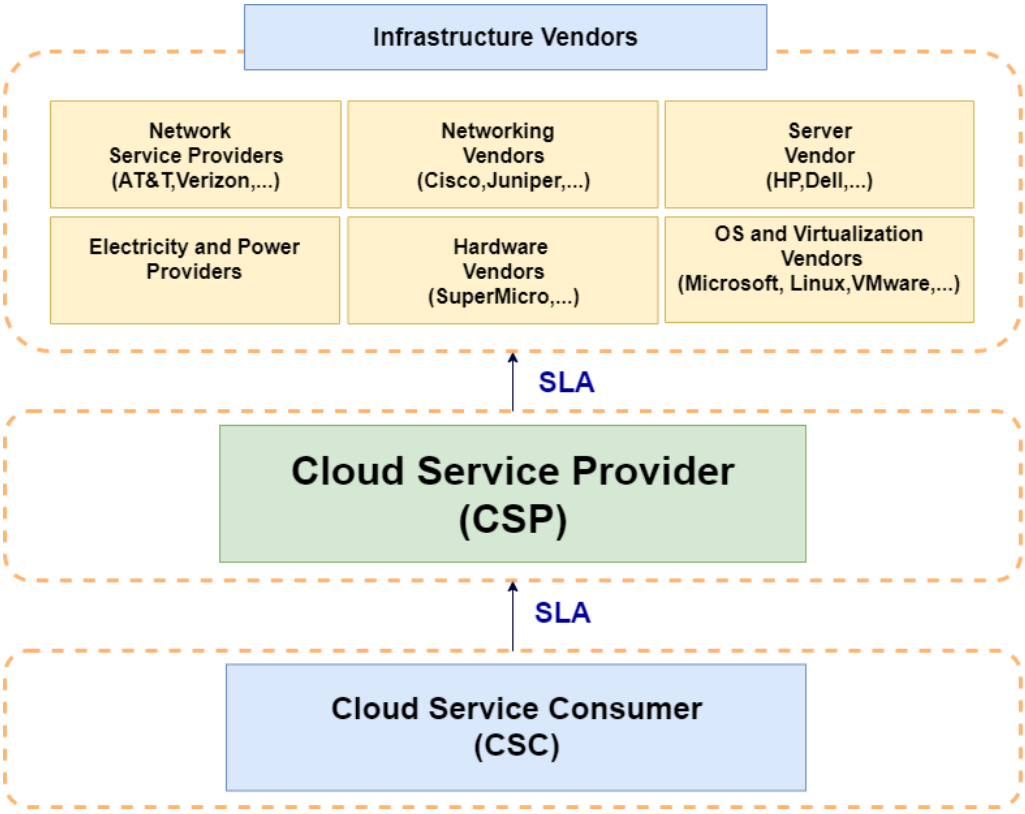

Figure 1.2: The Chain of Accountability and Escalation Path for SLAs

1.2 Cloud Computing and Cloud Monitoring

Cloud Computing, as an enabling model, is the most evolved form of deployment infrastructure. The term "Cloud", in its current sense, was first used by the CEO of Google (in 2006), Eric Schmidt. He described Cloud as a business model that allows companies to provide a wide range of services to their clients through the Internet [118]. Concurrent to this acknowledgement, advancements in virtualisation, the key technology behind Cloud computing, provided a higher utilisation ratio for hardware by breaking the physical limitation of allocating a hardware unit to a single user. In a virtualised environment, the same hardware is shared among multiple users, making it more utilised, resulting in a more appealing financial feasibility. On-premises deployment, as an alternative to Cloud computing, requires massive upfront capital investment and has massive operational cost.

The pay-as-you-go model of Cloud computing has made it an unbeatable alternative to on-premises deployments. As the most evolved form of deployment platform, Cloud Computing has tremendously changed the deployment options for the companies around the world. Elasticity, rapid deployment, and high scalability have made service and product delivery more feasible than before.

1.3 Cloud Monitoring and its Challenges

With many organisations choosing to move from their traditional infrastructure to Cloud, the reliability of services offered by the CSP becomes an important topic. A CSP needs to implement and maintain complex hardware,software, and network infrastructure. The CSPs design and implement this complex platform in several data centres full of homogeneous bare-metal servers equipped with hypervisors that take control of hardware resources, virtualise them, and share them in a configurable pool of resources. These servers and their virtual machines are connected via physical and Software Defined Network (SDN). To achieve high-availability, for every element, several redundant pairs are considered (e.g., UPSs, routers, switches, firewalls, storage components, and bare-metal servers). Needless to say, the CSPs require advanced monitoring tools that capture several metrics for every deployed component on the Cloud infrastructure. These monitoring tools have to be able to keep up with the number of generated logs in a Cloud platform. To provide an example of the scale of generated logs, a Cloud-based application such as Netflix generates more than 10 billion records a day [43].

Furthermore, unique characteristics and features of Cloud computing environments, such as elasticity and auto-scaling cause major challenges for the traditional monitoring tools. The number and nature of deployed resources in a pre-Cloud environment were mainly static, however, the elasticity of the Cloud results in a dynamic environment, in which, additional resources are dynamically added or removed. Hence, causing significant challenges for the traditional monitoring tools that are designed for static environments [111]. Another important challenge for Cloud monitoring is that a complete monitoring system requires complete access to the entire Cloud infrastructure and only CSPs have such level of access. As a result, almost all Cloud monitoring solutions are implemented by the CSPs. While this is beneficial to the CSPs, it leaves the CSCs with

no options to validate the accuracy of provided monitoring details and logs.

Cloud computing offers different types of delivery options. Infrastructure as a Service (IaaS) is the most comprehensive form of delivery whereCSCs are provided with full control over the entire lifecycle of provided resources. Therefore, IaaS requires the most advanced monitoring systems that can cover all the deployed components of an IaaS offering. As for the PaaS, the CSCs have control over one or more VMs and only require access to operating system logs. In contrast, in SaaS, the CSCs only use the software that are provided to them as a service and therefore need very limited monitoring resources. Another challenge for Cloud monitoring tools is to offer different monitoring options to CSCs, based on different delivery types [57].

In spite of above-indicated monitoring challenges, Cloud monitoring tools are mainly implemented by the CSPs which makes the trustworthiness of the monitoring data questionable [84]. Amazon CloudWatch [3], Google Stackdriver Monitoring [35] are examples of such monitoring systems. The CSPs use Cloud monitoring tools for two important reasons. Firstly, to monitor the status of all deployed components; and secondly, to feed the required details for Cloud charge-back system that converts resource usages into billable items. While the former is often used internally and within the jurisdiction of the Cloud provider, the latter has a significant impact on the customer. Hence, it is critical that the CSCs are aware of the collected metrics, their values, and how they are used to construct their payable invoices.

1.4 Trust Among Cloud Participants

As was indicated at the beginning of this chapter, while building in-house monitoring platforms provide full control over the monitoring systems to the CSPs, it puts them in a very powerful and autocratic position compared to the CSCs. This is because the accuracy of the provided data cannot be assessed by the CSCs as the full control is in the hands of the CSP [104]. The issue becomes more critical when generated metrics and their values are used as the basis for invoice generation. Similarly, if the CSC is complaining about a breach of the SLA, the issue of genuineness of logs become a critical matter and can damage the trust between these two parties.

In a Cloud computing environment, many different types of suppliers and users exist.

The CSPs (such as Amazon, IBM, Microsoft, and Google) are conglomerates which provide the actual Cloud environment. In the Business to Business (B2B) model, the CSCs (such as Netflix [29] and Dropbox [12]) use the services provided by the CSPs to re-package or offer on-demand services to their clients. At the same time, the CSP offers a Business to Consumer (B2C) model as well and sells Cloud services directly to end-users. While such diversity of business models has significantly contributed to the financial success of Cloud computing, it has also increased the importance of trust among Cloud participants.

1.5 Logs and Their Importance

Monitoring solutions are responsible for monitoring various resources in a deployed platform and generate useful insights based on generated values for all definable metrics. While the majority of monitoring systems are capable of generating graphical reports and sending alerts and notifications, the fundamental components of any monitoring system are the ones that collect and store logs. Here we are referring to raw data (generated by each Cloud hardware and software component) and stored for troubleshooting activities. In case of any technical issue, it does not matter which monitoring solution or approach has been used to collect the logs; the actual logs play the most significant role.

Logs are evidential documents [40]. They contain all the details and QoS metrics related to the operation of software, network components, servers, and Cloud platforms. As a key element in computer forensic investigations, logs are presentable in the court of law [95] only if they satisfy the legal requirement. These legal requirements are as follows:

1. Authentic,
2. Reliable,
3. Believable, and
4. Admissible.

As can be seen from the above, log's authenticity and reliability are among the key legal requirements, yielding to the importance of a tamper-proof log system.

1.6 Tamper-motivation and Log Tampering

In here, tamper-motivation is defined as the desire of one or more of the parties involved in a platform, infrastructure, or Cloud solution to access critical logs and to tamper the logs by adding, removing, or manipulating a part or the entire log. Given the number of logs that are generated by the Cloud monitoring solutions, there are many tampering possibilities. While tamper detection solutions can be used to detect any changes in the logs [100, 107], we argue that due to the critical nature of logs, tamper detection is not good enough; one should consider a storage option that offers immutability for all critical logs.

In the following paragraphs, we explore a few tamper-motivation situations that relate to private, community, and public Cloud.

A private Cloud is implemented, managed, and used by the same organisation. In a private Cloud, the customer of the Cloud services is the same organisation that manages the entire Cloud operation and, therefore, all stakeholders (more or less) have the same interests. In private Clouds, a special type of tamper-motivation may exist. Imagine a financial company that has established a private Cloud. The senior management team, based on the industry's common rules and regulations, have set extensive backup policies. One of such policies requires a real-time backup from all financial transactions. This requirement translates to a huge second-by-second backup for the IT department. The IT department configures their backup systems and ensures that the backup operation has started and there is enough space for continuous backup of transactions. A few months after the initial setup, the company's primary storage is affected by a hardware failure and the customer-facing agents of the company report that they have no access to any recent transaction. The IT department gets involved and starts a complete investigation. Given the importance of this matter, now all departments, including the senior management team are aware of the storage outage, and the IT department is under constant pressure. After a complete investigation, the IT department finds out that the real-time backup system has stopped working a few days ago and had sent several alerts to the members of the IT department, but no one in the IT department has checked these alerts as they have been busy with other issues. The IT department is the only department that has access to all the logs, including the alert logs. Hence, the department may be motivated

to take advantage of this access, remove or tamper the alert messages, and blame the issue on a hardware problem by showing the senior management the tampered version of logs, hence saving their jobs. In the above situation, the senior management team has almost no way to validate the authenticity of the logs.

A community Cloud [79] is a unique Cloud model in which all the components are provisioned for exclusive use by members of several organisations that construct a community due to the similarity of their goals. It falls between private and public Cloud [63] and offers a more feasible alternative for organisations as they will share the deployment and implementation cost among all the partners. To survive, a community Cloud requires a clear definition of responsibilities, maintenance tasks, operational tasks, and control. Each partner is responsible for a subset of Cloud elements and together, all partners, ensure that the community Cloud remains operational and available. In a community Cloud, there are additional tamper-motivations. In case of an unfortunate incident, the party at fault may be motivated to tamper the logs that identifies them as the party responsible for the issue or even worse, try to tamper the logs and fabricate a scenario in which another party becomes the main reason behind failure and, therefore, responsible for the caused damage. Having access to the logs for one or more of the parties in a community Cloud may cause trust issues that call for immutability of logs.

In contrast to private and community Cloud, a public Cloud is available for all users and resources are offered as a service, usually based on a pay-per-use charge-back system [63]. In any public Cloud offering, there is a very distinct line between the CSP and the CSC. The CSPs rent their computational resources to CSCs through various Cloud offering models. The most well-known of such models are IaaS, PaaS, and SaaS. As a result, many different types of business relationships can be constructed on a public Cloud. While the public Cloud is the most feasible form of Cloud offerings for the CSCs, it requires the complete trust to the CSP, almost in all aspects. Namely, the operational health of the platform, its performance, its generated metrics, and even charge-back reports that are consolidated in the form of monthly invoices to the CSCs. Without having access to the actual logs or the actual infrastructure, CSCs of the public Clouds are in a very unfair, dependent position. Consider a scenario in which a CSC deploys an application on an elastic Cloud environment. The CSC enables auto-scaling feature provided by the Cloud provider and defines a rule that when the memory usage exceeds `80%`, the CSP should allocate `20%` extra memory space to the deployed application.

Imagine that the CSC receives complaints related to the application performance from its users. The CSC suspects inadequate performance of the auto-scaling function as a root cause of the problem and asks the CSP to send a detailed report of the elastic memory allocation. The IT team of the Cloud provider checks their logs and finds out that the auto-scaling feature has worked intermittently, hence the performance issue. If the CSC finds out the truth, there may be a potential lawsuit on the horizon. Thus, the IT team may be motivated to tamper the log before sending it to the CSC.

1.7 Motivation

As mentioned in Section 1.2, for any Cloud platform that generates logs, there are parties which may be motivated to tamper them. If the party succeeds, the incident that they are trying to hide, most likely, will forever remain a cold case. Thus, the importance of a tamper-proof log system for Cloud solutions is significant. In other words, a traceable, verifiable, and immutable log system is required to establish trust among Cloud participants in any of the private, community, or public Cloud.

Log tampering may affect CSCs financially and technically. If a CSP tampers the logs related to resource usage and charge the customer with a higher amount or if a CSP hides the breach of one or more criteria of an SLA, the CSC is in immediate need of finding a method or a tool to verify the integrity of provided data by the CSP.

Given the existence of many tamper-motivations on the Cloud platforms, the cost of log tampering for the CSCs, and the inadequacy of existing monitoring solutions, we come to the conclusion that an immutable log system that is capable of storing the logs and verifying their integrity can be genuinely beneficial for the CSCs and can be used to establish trust among Cloud participants.

1.8 Objective

The **primary** objective of this study is to address the inadequacy of current log system by alleviating the required trust to the cloud providers or a third-party. We achieve this goal by developing and deploying a prototype of a tamper-proof log system, called Logchain, that can store and verify logs and their integrity. The proposed solution

employs immutability characteristic of blockchain technology and guarantees the integrity of the stored logs. Usage of the solution leads to establishing trust among all cloud participants.

The **secondary** objective is to reduce the computational complexity of blockchains, specifically, in the area of verification. We achieve this objective by introducing a hierarchical ledger structure, in which logs are stored in blocks, and a unique set of blocks reside in a circled blockchain. Information about circled blockchains is stored in a higher level blockchain. Verification of a higher level block, confirms the integrity of all the blocks in a lower-level blockchain. We show how this hierarchical structure reduces the computational complexity that is required to verify the integrity of a series of blocks in a blockchain.

The **tertiary** objective is to address the accessibility challenges related to the proposed prototype by designing and implementing an API that can be used to interact with Logchain. Users can use the API to send data (logs) and verify the stored data (logs).

Last but not least, we show that Logchain can be configured on top of Ethereum [15] as one of the most popular blockchains solutions.

1.9 Proposed Solution

To address the trust issue of the current Cloud log storage solutions, we propose[1] the use of immutable storage to preserve logs. We chose blockchain as our data storage model because it has all the required features such as immutability, and support for storing any type of data. The proposed solution collects logs from various platforms and stores them in blocks of blockchains.

Cloud platforms consist of a large number of hardware and software components [47, 80], generating a large volume of logs and metrics data. It is important that the proposed solution is scalable and can verify stored logs efficiently and with minimal computational power. Data verification in current blockchain platforms is a CPU and Memory intensive task as one has to generate a hash and nonce for each block to confirm the integrity of the stored data in that block. To make data verification more feasible, we propose a hierarchical ledger. Logs are stored in blocks and blocks are stored in the blockchain.

[1]The proposal has been presented at the IEEE CLOUD 2018 conference [90].

We then take critical information of blockchain and store it in a higher level blockchain. Therefore, each block of a higher level blockchain contains a "fingerprint" of the lower-level blockchain. Thus, verifying one block of a higher level block verifies the integrity of all the logs that are stored in the lower-level blockchain that the block under verification represents.

To make the Logchain more accessible and make it easier to use, we designed and implemented an API that can be used to interact with Logchain. The API offers an interface for users to send their logs and verifies their integrity.

To assess the compatibility of the proposed solution, we implemented a hierarchical ledger on top of Ethereum platform.

Our prototype constructs a LCaaS and receives logs and stores them in an immutable hierarchical ledger; clients can use its API to interact with the solution and send, verify, and retrieve data from this immutable storage. The complete source code of the prototype can be accessed via [89].

1.10 Novelty and Contribution

To the best of our knowledge, there is no other work that uses blockchain as a storage system for logs that are generated by Cloud platforms. Although blockchains are used as an immutable storage option for many transaction-related data, their usage in the non-financial area is still in infancy. We employ blockchain and use blocks to store logs. To implement our prototype successfully, we had to provide a solution for the main scalability limitation of the blockchain, namely, the number of computational resources that are needed to verify the integrity of each block.

In addition to using essential characteristics of blockchain technology, we have introduced the following new additional features that will increase functionality, scalability, and usability of blockchain technologies for tamper-proof log storage: (i) Absolute Genesis Block (AGB), (ii) Relative Genesis Block (RGB), (iii) Terminal Block (TB) (iv) Circled Blockchain (CB), (v) Super Block (SB), and (vi) Super Blockchain (SBC).

We design the proposed solution as a service so many customers can use it concurrently. Moreover, given that CSCs are already using many "X-as-a-Service" solutions, they are accustomed to such service offering model. To ensure that the proposed solution

can be used in conjunction with the existing blockchain solutions, we implemented the proposed hierarchical ledger on top of Ethereum.

The significant contributions of this work can be summarised as enhancing the underlying blockchain structure and extending its capabilities by adding additional elements, listed above. Furthermore, we implement a hierarchical blockchain ledger to achieve faster verification process without altering the structure of a standard blockchain and its key components. We also introduce an API as an interface for the platform so that CSPs and CSCs can interact with it programmatically.

1.11 Outline

The rest of the thesis is structured as follows. In Chapter 2, we provide related works and a brief literature review on both Cloud computing and blockchain. In Chapter 3, we introduce the methodologies that we use to build Logchain, our prototype, and the approaches for enhancing blockchain and its capability, followed by the implementation of hierarchical ledger. Furthermore, we introduce API, its signatures, and its responses. In Chapter 4 we present the analysis and the results of application of LCaaS model on existing blockchain platforms. Finally, in Chapter 5, we provide a summary of this study, as well as a conclusion and a direction towards future work.

Chapter 2

Literature Review

In this chapter, in Section 2.1 we review the related research and work that highlights the importance of Cloud monitoring and challenges specific to Cloud monitoring, such as log collection and storage on the Cloud environment. In Section 2.2, we review the research and work related to the digital forensic investigation and its associated challenges. Section 2.3 provides a review of literature about the authenticity and reliability of digital evidence. In Section 2.4, we review logs and their importance as digital evidence. Solutions that are commonly used for tamper prevention or detection are discussed in Section 2.5. We provide an overview of blockchain and its characteristics in Section 2.6 and assess the solutions for building a trustable log management system in Section 2.7. In Section 2.8, we conclude this chapter by providing a review of the literature that identifies blockchain capacity limitations and offers solutions to overcome such limitations.

2.1 Cloud Logs and Related Challenges

The CSPs and CSCs leveraging Cloud offerings should continuously monitor the performance of their products to ensure the health and efficiency of their services. Quality dimensions, such as availability, reliability, and performance, are critical components of SLAs. Thus, monitoring is an essential component of a Cloud platform, and the generated data by monitoring platforms play a vital role and have to be stored with absolute care. Companies that offer their services to a large customer base, , like Netflix, deploy large and scalable solutions on the Cloud. The metrics that need to be monitored can

quickly produce more than 10 billion records a day [43] making the data set large enough to be classified as Big Data [81, 80]. The high volume of generated logs, and monitoring data cause various challenges. The storage of such large amount of data requires extensive storage capacity. Furthermore, processing the collected data is also computationally expensive [112, 80]. Hence, monitoring large scale deployed platforms is one of the major challenges of Cloud monitoring [112].

In addition to the scale of generated logs, unique characteristics of Cloud platforms cause various monitoring challenges. The CSPs use Cloud orchestration tools to automate the allocation of resources and their placement, fault management, and storage management [74]. Many monitoring tools need to be aware of the existence of a resource before they can provide a monitoring service for it. Hence, conventional monitoring tools cannot be used in an environment that offers dynamic resource allocation [112] empowered by Cloud orchestration tools. These challenges are not limited to hardware resources such as CPU and RAM, and the dynamic nature of Cloud networks also causes difficulties for the network monitoring tools [88].

Cloud platforms consist of several hierarchical layers [109]. The layer at the bottom is the hardware level and consists of data centre components. Many software-defined layers are implemented on top of this layer [66]. These layers are the infrastructure for the application layer which is the closest one to the CSCs. While there are clear boundaries among these layers, an issue in a lower layer can easily affect upper layers. A common challenge for CSPs is to trace an issue and find out the layer (and a component specific to that layer) that is the root cause of a given problem. Addressing this challenge requires extensive traceability and a holistic view of how layers are interrelated. Interestingly, this layered architecture opens the door for many log tampering opportunities as the logs at each layer are only accessible to the technical team responsible for that layer. As for CSCs, the layered structure imposes traceability issues, as CSCs cannot trace issues beyond the application layer, and they are often left wondering whether the outage is caused by their settings or is caused by the CSP.

Above-indicated issues affect both the CSPs and CSCs. It is important to mention that the issues related to establishing trust among Cloud participants are specifically related to the secure collection, storage, and transfer of COLs.

2.2 Digital Forensic and its Principles

The legal system relies on a range of forensic investigation and identification. Traditionally, all such analysis were carried out against physical evidence, such as DNA or fingerprints. However, in the digital era, digital evidence such as operational logs, transaction logs, and usage logs replace physical evidence. These new type of evidence, require new forensic investigation methods [60]. Digital forensics is the collection of methodologies that encompasses the forensic analysis of all types of digital crimes.

Digital Forensic Science is defined by Palmer [85] as "The use of scientifically de-rived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorised actions shown to be disruptive to planned operations.".

As a new branch of science, digital forensics is dealing with some issues, such as lack of standardisation, different practices among different involved parties, and the lack of interconnection between academia and law enforcement [96]. Caloyannides, in his book "Privacy Protection and Computer Forensics" [51], reviews the challenges of digital forensics and digital evidence. In an attempt to regulate and standardise digital forensics practices, the Digital Forensics Research Workshop (DFRWS) [11] has developed a forensic framework. The framework focuses on the interrelated steps of identification, preservation, collection, examination, analysis, presentation, and decision [115]. The framework identifies preservation as a crucial step and indicates that it must be a guarded principle across "forensic" categories. Figure 2.1 depicts the investigative process for digital forensic science created by the participants of the DFRWS [11].

2.3 Digital Evidence and its Reliability

Digital evidence is defined as any type of data that is stored or transferred using a computer that support a theory of how an offence has occurred [53]. Here, data can be a combination of text, images, audio, video, and logs. As a complex form of evidence, digital evidence is often overlooked, collected inaccurately, or analysed ineffectively. As

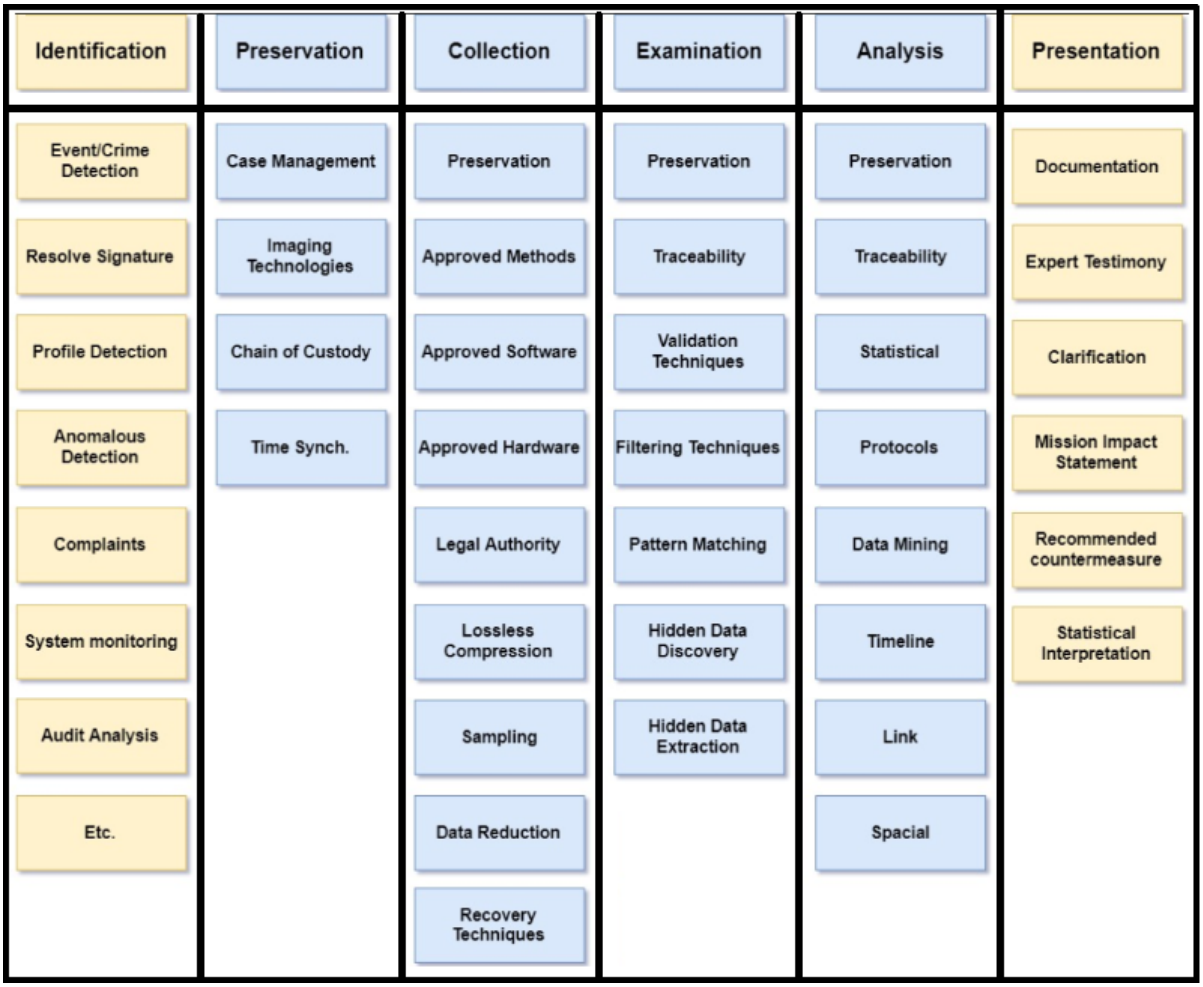

Figure 2.1: The Investigative Process for Digital Forensic as per [11]

indicated by Caloyannides, preservation and traceability of digital evidence are critical components of digital forensics [50]. In regards to the use of digital evidence in court, a common question is whether recovered digital evidence is authentic and with no alteration. Any reasonable doubt regarding the reliability of digital evidence significantly reduces the weight assigned to the evidence. In some cases, if the evidence is tampered, it makes it inadmissible. Almost in all cases, the only people who have access to the digital evidence and are familiar with its content can tamper it. This requirement causes many computer crimes by employees and internal members of organisations [53].

Another major challenge related to authenticity and reliability of digital evidence is

that digital data are much more easily tampered compared to physical evidence. Many software tools allow users to change the content and all associated characteristics of digital evidence. For example, not only the content of a file can be changed, but also the user has control over changing its metadata, such as author, creation date, modification data, and its last accessed time. Furthermore, just transferring or viewing digital evidence requires translation and transformation through many software and network layers. As a result, a significant trust issue exists for any digital evidence that is captured from a digital source.

In regards to the above challenge and its importance, we argue that in respect to the lifecycle of the digital evidence, the earlier a tamper-resistant storage option is introduced, the lower is the chance of digital data tampering. In fact, the best possible option would be to store the data in tamper-resistant storage right after that data are generated.

2.4 Logs as Digital Evidence

Almost all components involved in a Cloud computing infrastructure generate logs. It is essential to understand that logs record the dynamics of a system, rather than a static snapshot. A timestamped log can allow one to reconstruct the entire system interaction with all complex chain of events that occur during the operation of Cloud-based solutions. Because of this characteristic, logs can be easily used as the primary source to shed light on many legal disputes. To be admissible as digital evidence, logs have to fulfill several legal requirements. Both, the security architecture of the log collection and the methods that are used to protect logs against tampering, play important roles. Lack of either of them can cause logs to become inadmissible [40].

Logs are among the most critical pieces of information for practices, such as debugging forensics and security and incident detection. While many of these practices heavily rely on log analysis, there are, in general, some common challenges related to collection and analysis of logs. Logs are collected in various forms and with different time intervals. Logs are usually collected in a decentralised fashion and each component stores logs in a predefined location and format which may or may not be accessible to the rest of the Cloud [77]. The existence of various log formats impose another challenge as carrying out a forensic investigation may involve analysing and cross-referencing several files, each

of which may be in a particular format. In some cases, there may be no log to begin with, or logs are collected at a log-level that do not capture critical information that is required for the investigation.

To effectively use logs as digital evidence, one has to address these issues. To overcome these issues, many have recommended the use of a Log Management System (LMS) [77, 94, 69, 54]. The majority of LMSs promise a set of desirable features, such as accuracy, tamper-resistant, verifiability, confidentiality, and privacy [94].

2.5 Log Tampering Prevention/Detection Solutions

Digital forgery and tampering of digital artefacts and files long existed. Many solutions have been proposed to detect or prevent such undesired activities. Since the majority of logs are stored in files, it is vital to explore solutions that are offered for file tamper detection or prevention. Various file verification techniques exist to ensure that the file at hand is not tampered.

More than five decades ago, Peterson et al. described the use of cyclic codes to verify that a given array of bits is original or a change has happened [87]. Based on similar principles of cyclic codes, checksum has been widely known and used [55, 106]. In particular, checksums are used by many file systems to validate the integrity of files and their related operations, such as copy and move. Checksums are generated by using a checksum function that takes a block of digital data of arbitrary size and in return generates a fixed-size value. The primary issue related to checksumming data is that generating and verifying checksum values will slow down the I/O process [106], a process that is already known to be the slowest among all other processes in a computer [105].

One of the modern favourite hashing techniques is a family of Secure Hash Algorithm (SHA) [61] which is used as a means to verify content, authorship, or a property of a digital artefact. As an example, the source code management system git [20] generates a SHA-1 [61] signature for a commit and uses it to trace the commit throughout the entire lifecycle of the source code [48]. In this solution, SHA-1 is mainly used for traceability and points out to the person who committed the code and is not used as a means for tamper detection.

In recent years, many verification-as-a-service platforms offer integrity control for the

data that is uploaded by the user. Verification-or-integrity-as-a-service solutions, such as arXiv [7], provide a repository of electronic documents and ensure their integrity. Similarly, Data Integrity as a Service (DIaaS) uses the service-oriented architecture to release the burdens of data integrity management from users. In the suggested solution, an independent Integrity Management Service (IMS) is in charge of collecting and storing data with a minimal impact on the performance of data storage [83].

As for Cloud solutions, theoretically, many of the mentioned solutions are applicable. However, the complexity of Cloud environment (in particular, auto-scaling, redundant systems, and load balancers) and the scale of generated logs bring more challenges for the storage, access, and verification of the logs. Sharma [103] points out the complexity of mega-scale Cloud environment and suggests incorporation of various cryptographic algorithms and digital signature to achieve high integrity for storing critical information in the Cloud. Liu et al. [73] focus on the data storage integrity verification for Big Data in the areas of Cloud and Internet of Things (IoT), stating that data integrity is critical for any computation-related system.

Bharath and Rajashree [46] suggest the use of a mediator, known as a Third-Party Auditor (TPA), which verifies the integrity of the data and sends the integrity report to the users. However, this solution still requires trust in a third-party or central authority.

The main drawback of these services is that one must trust the central authority that is offering the service. The problem of trusting a third-party can be alleviated by a self-contained solution that does not rely on a TTP integrity verification service. We argue that **blockchain** is among the most promising solutions that can be used to replace the requirement for a TTP. It was initially designed to support a cryptocurrency known as Bitcoin that does not require a TTP (such as banks or other financial institutes) for verification or maintenance of financial transactions. A correctly implemented distributed blockchain is an adequate alternative to address the TTP issue [110, 75, 121].

2.6 Blockchain

Blockchain in the simplest form is a distributed database of records. Records are stored in each copy of the database, known as public ledger[1] [56]. One of they key characteristics

[1]Note that there exist other methods to create public ledgers.

of this database is that it does not provide support for normal Create, Read, Update, Delete (CRUD) operations. Instead, it only provides all its participants a write-only privilege. In contrast to a Database Management System (DBMS), this limitation is not a role-based setting. That is, all users, including the root (or admin), are faced with the same constraint. While this feature removes the support of update and delete operations for any inserted data, it results in desirable immutability. The most famous application of blockchain is providing the infrastructure of the most controversial currency in the world. While Bitcoin, as a realisation of a cryptocurrency, is tightly coupled with the blockchain, it is important to mention that Bitcoin is only one of the possible applications of blockchain technologies and the use of blockchain technologies is not limited to cryptocurrency or financial applications.

In 2008, an individual or a group of the researcher(s) under the alias of Satoshi Nakamoto published a paper titled "Bitcoin: A peer-to-peer electronic cash system" [82]. In this paper, the author(s) describe a system in which financial transactions can be sent from a sender to a recipient without relying on a trusted financial institute such as a bank. Nakamoto argue(s) that a purely peer-to-peer version of electronic cash transaction eliminates the needs of relying on a financial institution. A Payer needs to digitally sign a transaction to prove the authenticity of the transaction, and the receiver has to verify the transaction to prevent the double-spending problem. A significant component that allows transactions to be immutable is the use of a timestamp method to mark each transaction with a timestamp. A timestamp server takes a hash of a block of items to be timestamped and widely publishes the hash and its timestamp so that every participant get to know that the items must have existed at the time of the announcement [82].

Another critical component of blockchain is the use of Proof-of-Work (PoW). Similar to Hashcash [45], PoW involves running iterations for finding a particular value that when hashed in conjunction with other elements of a block, the calculated hash begins with a certain number of zero bits. The number of zeros is proportional to the time that is required to complete a PoW. The higher the number of zeros, the longer it will take to complete the PoW. Once the computational effort is dedicated and the hash value is found, all items along with the found value, known as `nonce`, are kept in a block. The content of a block cannot be changed unless the whole PoW process is repeated again. Chaining blocks together using hash binding or hash chaining [64, 101, 68] significantly increases the amount of computational effort that is needed for changing the content of

an earlier block. In a hash binding relationship, the current hash of the previous block is used as the previous hash of the current block. This chain makes any attempt to change the blockchain computationally unfeasible as one needs to re-process PoW for all the blocks of a blockchain in order to tamper with any of the earlier blocks [78].

Current blockchain implementations of the distributed ledgers already have notary proof-of-existence services [110]. For example, Poex.io [32], launched in 2013, verifies the existence of a computer file at a specific time, by storing a timestamp and the SHA-256 [61] of the respective file in a block that will be eventually added to a blockchain. The service is anonymous as the files are not stored or transferred to the provider's servers. Since the digital signature of the file is permanently stored in a decentralised blockchain, the provider can verify the integrity and existence of such a file (at a point of submission to the blockchain) anytime in the future. Characteristics of cryptographic hash function [97] allow a provider to claim, with high certainty, that if the document had not existed at the time when its hash was added to the blockchain, it would have been very difficult to embed its hash in the blockchain after the fact. Additionally, embedding a particular hash and then adopting a future document to match the embedded hash is also almost impossible [97].

2.7 Trustable Log Management Systems

Proof-of-existence solutions cannot be used as scalable LMSs, as they consider files individually, with no search function to locate the appropriate file or block. Moreover, Cloud solutions consist of thousands of hardware and software components, each of which generates a large volume of logs [91]. The current solutions are not designed to handle the scale that is required to store Cloud-generated logs. Furthermore, the current public blockchains can handle a limited number of concurrent transactions [110].

At its very core, blockchain requires a lot of computational resources to operate. For the creation of every block, on average, many iterations of hash generation are repeated until the desired outcome matches the required difficulty target. This requirement makes blockchain an expensive solution for storage of high volume data such as logs. Hence, using blockchain as a log management system, at least without modifications, is neither financially nor technically feasible.

To secure the collected logs, specifically audit logs, Waters et al. [113] build a platform that uses hash encryption to protect the content of audit logs from unauthorised parties by encrypting the content of them. Furthermore, the authors indicate that while encryption is required to preserve the logs and to make them immutable, it will cause many search issues. For instance, relying on traditional search techniques would require complete decryption for all the records at the time of the search. Such a requirement by itself creates room for potential unauthorised access. The authors present a design for a log encryption system that allows a designated trusted party, known as audit escrow agent to construct search capabilities and allow an investigator to decrypt entries matching a keyword.

Ko and Will [71] indicate that data are, arguably, the most important asset on the Cloud. The authors mention that the current data collection tools have four primary problems, namely, (i) the inability to provide a tamper-resistant environment, (ii) accurate timestamp synchronisation among all data collection servers, (iii) log space requirements, and (iv) efficient logging of root usage of the system. Ko and Will [71] offer a solution referred to as Logger and claim that Logger addresses all four issues. In regards to preventing log tampering, the authors acknowledge the complexity and difficulty of this requirement and address it by only allowing an internal process to access the file and forcing such limitation in all internal system calls. The authors explore additional options, such as the use of signatures, hash creation for records, or hash chaining, but at the end argue that such methods can cause significant performance issues.

Accorsi [41] presents a digital black box solution, named BBox, that provides authentic archiving in distributed systems. BBox exhibits characteristics that are required for a trustable log storage component. Namely, it offers a source validation technique and only allows authorised devices to insert records in the log files. Further, to ensure the immutability of the stored log, BBox uses hash chains. The solution encrypts all data, and there are no clear-text stored logs. Last but not least, BBox offers a keyword-based retrieval of records. Although records are encrypted, BBox supports simple keyword searches for log entries, by generating the so-called "log views". Therefore, the retrieval only requires the decryption of logs that match the entered keyword and in return, reduces the cost of log view generation.

As was indicated in Section 2.6, use of blockchain technology is not limited to financial industry. Azaria et al. [44] denote that heavy regulations and bureaucratic inefficiency

have severely slowed the innovation for Electronic Medical Record (EMR). The authors propose a blockchain-based solution, known as MedRec, that offers each patient with an immutable and comprehensive log and easy access to their medical records. Relying on key features of the blockchain, MedRec manages all crucial considerations that are required when dealing with sensitive data sets such as healthcare records. MedRec overcomes many inefficiencies that current EMRs have by employing advance blockchain techniques such as smart contracts. The smart contract was invented by Szabo [108] to to allow the decentralised nodes to run self-executing autonomous pieces of code.

2.8 Blockchain Scalability Solutions

Current blockchain consensus protocols require every node of the network to process every block of the blockchain, hence a significant scalability limitation. Although blockchain technology has great potential and can be used in many disciplines, it is dealing with a number of challenges. The scalability remains the most critical challenge [119]. Blockchain heavily relies on consensus algorithms, like PoW, and such algorithms are computationally expensive. To overcome the scalability issues, a novel cryptocurrency scheme is suggested by [49] where old transactions are removed from the blockchain, and a database holds the values of removed transactions. Although this solution reduces the size of the blockchain, it introduces the same trust issue that traditional DBMSs are suffering from.

In [59], Eyal et al. suggest redesigning the current structure of a blockchain. In the redesigned model, known as Bitcoin-NG (next generation), conventional blocks are decoupled into two parts, the key block, and microblocks. The key block is used for leader election, and the leader is responsible for microblock generation until a new leader appears. Once a node generates a key block, it becomes a leader and is allowed to generated microblocks. A microblock contains ledger entries and also a header that includes the reference to the previous block. Peculiarly, microblocks do not require PoW and only key blocks contribute to the length of the chain. While this approach is very different from current practices of the blockchain, the authors claim that Bitcoin-NG maintains all security features of the blockchain.

As indicated in 2.6, PoW plays an important role in blockchain. However, PoW requires a great deal of computing resources and higher cost for creating each block. To

address the cost associated with PoW, King and Nadal provide an alternative approach to PoW [70] and name it Proof-of-Stake (PoS). King and Nadal argue that the security of peer-to-peer cryptocurrency solutions such as Bitcoin does not have to depend on cost of energy consumption and one can mine or validate block transactions according to how many coins he or she holds. Compared to PoW, the proposed alternative works faster and cheaper. Confirming the ownership of a coin (or any digital asset) is fast and east and requires a digital signature. Given the higher efficiency of PoS, Ethereum has decided to migrate from PoW to PoS. At the time of writing this thesis, the migration is in process; currently, Ethereum runs on a hybrid PoW / PoS structure [9].

The PoS schemas are actively evolving. For example, Micali et al. [62] created fast, scalable, and secure PoS algorithm, called Algorand. The authors prove optimality of their solution and are currently working on the production implementation of the algorithm [2].

Chapter 3

Methodology

In this chapter, we introduce the methodologies that we use to build Logchain and explore the implementation of hierarchical ledger. Furthermore, we introduce Logchain API, its signatures, and its responses.

In Section 3.1, we present a brief overview of components that are common among all blockchain platforms and describe what each component does. We also provide an overview of mining and immutability of blocks in a blockchain and review blockchain characteristics. In Section 3.2, we provide a general overview of Logchain as a service, its details, and the additional settings that are added to the blockchain to enhance LCaaS' scalability. Moreover, we give details on the privacy options that we provide for log storage in LCaaS. Section 3.3 focuses on the additional components that are introduced to blockchain to make the implementation of Logchain possible. We review the details of each added component and describe what it does. Section 3.4 highlights details on the implementation of LCaaS, such as its modules, configurations, and process flow. Additionally, we review the way logs are converted to data blocks and stored permanently. In Section 3.5 we review the design and implementation of the LCaaS API and introduce API methods and their return values. We conclude this chapter by providing an overview of the applicability of LCaaS and its hierarchical ledger to other blockchain platforms in Section 3.6.

3.1 Common Key Components of Blockchains

Here we introduce the components that are common among all implementations of the blockchain. These components are native to the blockchain, and, without them, no blockchain can operate. Thus, they can be seen in any implementation of the blockchain, such as Bitcoin [82] and Ethereum [15].

3.1.1 Genesis Block (GB)

Genesis block is the first block of any blockchain. Genesis block has predefined characteristics. Its `index` and `previous_hash` are set to zero as there are no prior blocks. Its `data` element is set to null. The primary purpose of a genesis block is to indicate the start of a new blockchain and to act as an ancestor for all the following data blocks [58]. In our implementation of genesis blocks, we have extended the genesis block definition by creating two different types of genesis blocks (which we will explain in details in Section 3.3). Figure 3.1 shows the elements of the original GB.

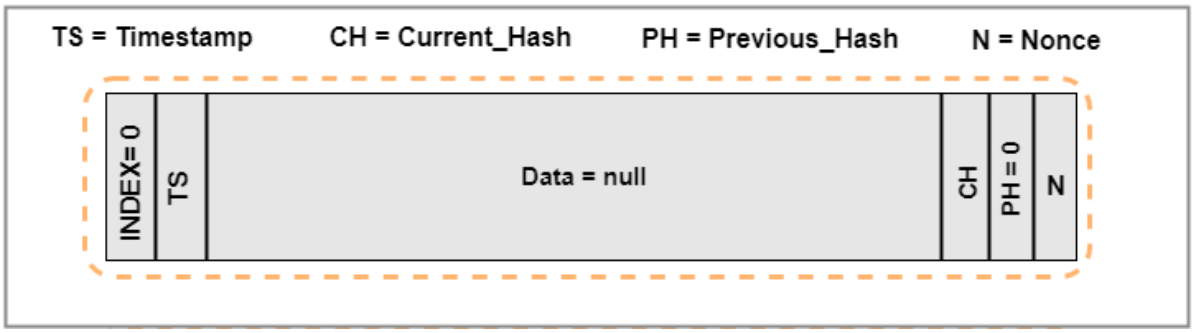

Figure 3.1: Original Genesis Block and its Elements

3.1.2 Data Block (DB)

Blocks are atomic units of storage. A data block, more commonly known as a block, contains the following variables: `index`, `timestamp`, `data`, `current_hash`, and `previous_hash`. The first element, `index`, is a unique sequential ID for each block; it uniquely identifies each block. The `timestamp` indicates the time at which the block is created and is usually

stored in the Coordinated Universal Time (UTC) format. The `data` is the most important element of a data block. It contains valuable information that blockchain has promised to keep immutable. The `nonce` is an arbitrary random number that is used to generate a specific `current_hash`. To achieve hash binding, each block includes a `previous_hash` element. The `previous_hash` is the exact duplicate of the `current_hash` of the previous block. In other words, the `current_hash` of the i-th block becomes the `previous_hash` of block $i+1$. Figure 3.2 illustrates a data block and its elements. Our implementation of blocks is similar to the existing blockchain solutions.

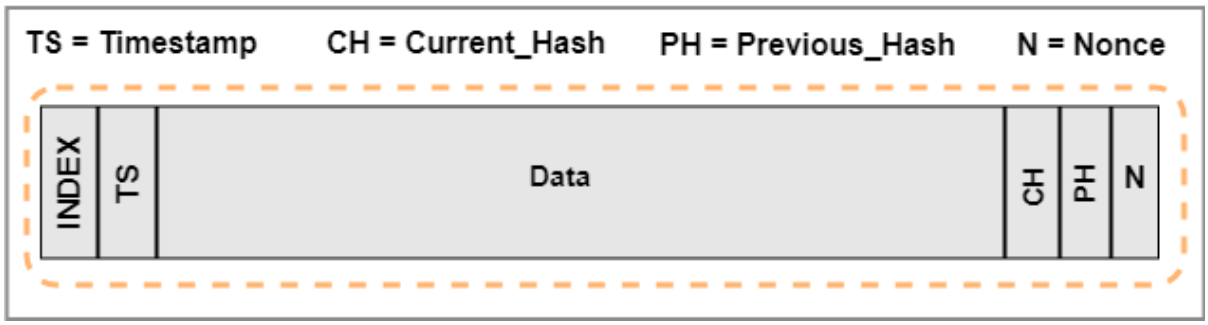

Figure 3.2: Data Block and its Elements

3.1.3 Blockchain (BC)

Blocks that are linked together via hash binding will result in a blockchain. If data in an earlier block (say, block m) are tampered, the link among all the subsequent blocks, from $m+1$ to the most recent block i will be broken. Then one has to recompute `current_hash` and `nonce` values of each block from block m to block i of the Blockchain (BC).

3.1.4 Mining and Hash Binding

Relying on key characteristics of cryptographic hash function [97], blocks are cryptographically sealed by using a method known as PoW. Comparable to Hashcash [45], PoW involves a computationally expensive process referred to as mining. Mining is the process running through all possible values of an integer variable[1], known as `nonce`, to

[1]Typically implemented as unsigned integer.

find a value for `nonce`, such that if the value is added to the rest of elements of a block and then hashed, the hash matches the imposed `difficulty_target`. Once the desired value for `nonce` is found, it resides in the `nonce` element of the block and the calculated hash resides in the `current_hash` element of a block. At this stage, the block is mined. The `difficulty_target` is often defined as the number of required zeros at the beginning of the desired hash. The more zeros, the more computational power is needed to generate a hash that matches the difficulty target. In practice, the probability of getting a prefix of zeros of length n bits, denoted by $P(\lambda_n)$, is calculated as

$$P(\lambda_n) = \left(\frac{1}{2}\right)^n. \tag{3.1}$$

Figure 3.3 shows how additional number of zeros in the desired difficulty target significantly reduces the probability of the desired outcome at each trial, hence, yielding (on average) to more trials and more computational power.

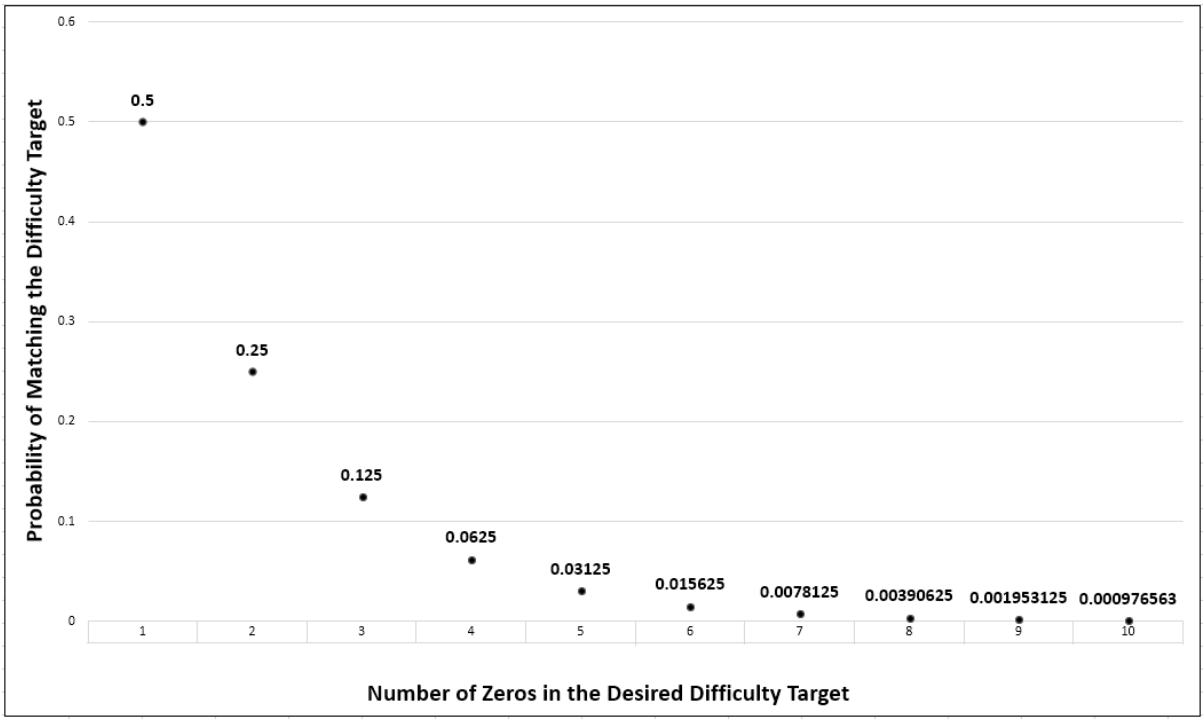

Figure 3.3: Probability of Desired Outcome at Each Trial as Number of Zeros Grows

One has to iterate through several values of `nonce`, to generate the `current_hash` for a given block that matches the defined `difficulty_target`. The target can be set during the initialisation of the LCaaS and may be adjusted later if needed.

Blocks are linked together based on a hash binding relationship. Our implementation of blockchains and mining operations have the same characteristics as any other blockchain. Figure 3.4 shows the hash binding relationship between block i and block $i+1$. Furthermore, we show the creation of the `current_hash` and `nonce` in Algorithm 1.

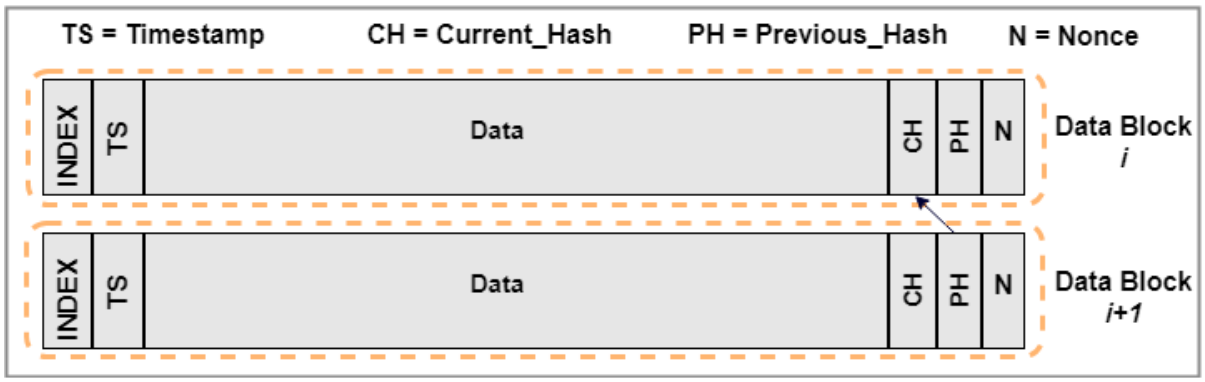

Figure 3.4: Hash Relationship Between Data Block i and Data Block $i+1$

Input : `block_index`, `timestamp`, `data`, `previous_hash`
Output: `current_hash`, `nonce`

```
1 content = concatenate(index, timestamp, data, previous_hash);
2 content = Hasher(content);                    // to speed up computing
3 nonce = 0;
4 repeat
5 |   nonce = nonce + 1;
6 |   current_hash = Hasher( concatenate(nonce, content) );
7 until prefix of current_hash = difficulty_target;
8 return current_hash, nonce;
```

Algorithm 1: Generation of hash and nonce for a block. Our implementation instantiates Hasher using SHA-256.

Once a block is mined, its content can no longer change unless the whole PoW process is repeated for every block in the blockchain.

Let us examine the computational complexity of Algorithm 1 line by line.

On line 1, we concatenate all elements of a block. The complexity of this operation will be proportional to the length of the variables. The variables `index`, `timestamp`, `current_hash` and `previous_hash` are of fixed length. The length of the `data`, denoted by d would vary. Therefore, we can say that the time needed to concatenate the elements is $O(d)$. From the log storage perspective (as will be discussed in details in Section 3.5) one can either store the raw log, in which case d can be large or one can store a "digest" of the log records (e.g., its hash value), in which case d will be of a small fixed length. In the latter case the $O(d)$ simplifies to $O(1)$.

Line 2 complexity, typically[2], will be proportional to the length of `content`. As discussed above, if we are storing raw logs then complexity of the operation would be $O(d)$, otherwise it would be $O(1)$.

Lines 3, 5, and 8 complexity is $O(1)$.

The number of iteration of the loop residing between lines 4 and 7 is a stochastic value. It would depend on satisfying the "prefix of `current_hash` = `difficulty_target`" constraint[3] on line 7. However, we do not know in advance how long this will take. In the best case scenario, we will need a single iteration to satisfy the constraint. In the worst case scenario, there exists no value of `nonce` satisfying the constraint, in which case we will need to iterate over all[4] possible values of `nonce` and fail.

On average, the number of iterations will be driven by the length of the prefix, as discussed at the beginning of this section. Assuming that the hash function outputs are uniformly distributed, we need to compute the mean (expected value) of the uniform distribution on the interval $[1, 2^n]$. Thus, as per the definition of the uniform distribution, the expected number of iterations E will be given by

$$E = \frac{2^n + 1}{2}. \tag{3.2}$$

Line 5 complexity would be proportional to the length of `nonce` and hashed `content`. Given that the lengths of `nonce` and hashed `content` are constants, we can say that the complexity of this line is $O(1)$.

Therefore, the cost of operations in the loop between lines 4 and 7 is constant and the

[2]As it depends on the actual hashing function.
[3]The complexity of computing the expression on Line 7 is $O(1)$.
[4]Given 64-bit unsigned integer, this would amount to $2^{64} - 1$ iterations.

cost of the loop itself will be driven only by the number of iterations. Therefore, based on Equation 3.2, the complexity of code between lines 4 and 7 is $O(2^{n-1})$.

Hence the time complexity of PoW for each block, denoted by $T(Block_{PoW})$ is calculated (if we are storing the raw log) as

$$T(Block_{PoW}) = \underbrace{O(d)}_{\text{Line 1}} + \underbrace{O(d)}_{\text{Line 2}} + \underbrace{O(1)}_{\text{Line 3}} + \underbrace{O(2^{n-1})}_{\text{Lines 4-7}} + \underbrace{O(1)}_{\text{Line 8}} = O(d) + O(2^{n-1}). \tag{3.3}$$

If we are storing a digest of the log, then the above equation simplifies to

$$T(Block_{PoW}) = O(1) + O(1) + O(1) + O(2^{n-1}) + O(1) = O(2^{n-1}). \tag{3.4}$$

The average number of operations grows linearly with the number of blocks. Therefore, assuming that we have m blocks, the complexity of computations for the chain, denoted by $Blockchain_{PoW}$ would be

$$T(Blockchain_{PoW}) = O(md) + O(m2^{n-1}) \tag{3.5}$$

for the case of raw logs and

$$T(Blockchain_{PoW}) = O(m2^{n-1}) \tag{3.6}$$

for digests of logs.

3.1.5 Blockchain Immutability Features

In addition to hash binding, blockchains take advantage of the hash function properties. Most cryptographic hash functions are designed to take an input of any size and produce a fixed-length hash value. Menezes et al. [42] indicate the following three basic properties of a hash function h with inputs x or x' and outputs y or y'.

1. "Preimage Resistance: for all predefined outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any preimage x' such that $h(x') = y$ for any y for which a corresponding input is not known. In other words, for a given hash, it would be computationally unfeasible to reverse the hash function and find the value that was hashed." [42]

2. "Second preimage resistance: it is computationally infeasible to find any second input which has the same output as any specified input, i.e, given x, to find a second preimage $x' \neq x$ such that $h(x) = h(x')$." [42]

3. "Collision resistance: it is computationally infeasible to find any two distinct inputs x and x' which hash to the same output, i.e., such that $h(x) = h(x')$. Collision resistance implies second preimage resistance but does not guarantee preimage resistance." [42]

3.1.6 Blockchain Characteristics

In addition to hash function features, blockchain has other unique characteristics that resulted in its widespread use and has made it one of the most promising technologies of cybersecurity [92]. The blockchain is a decentralised database. Each participating node holds a copy of digital ledger and ledgers are synced every time an item is added to a ledger. Storing several copies of the shared ledger eliminates the chances of data loss caused by a single point of failure. Unlike other databases, blockchain enforces append-only constraints. All data items are chronologically timestamped; thus, there is a concept of order, based on the time that each item is added to the ledger. Blocks cannot be removed and can be audited by anyone, as their details are publicly available in the shared ledger. Figure 3.5 depicts the key characteristics of the blockchain.

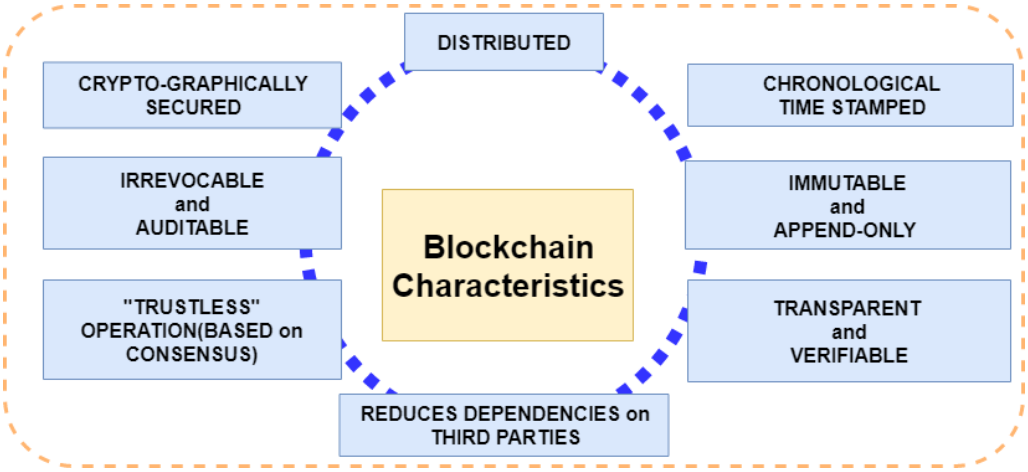

Figure 3.5: Key Characteristics of Blockchain

3.2 Logchain as a Service (LCaaS)

Current blockchain consensus protocols require every node of the network to process every block of the blockchain, hence a major scalability limitation. We overcome this limitation by segmenting a portion of a blockchain and locking-it-down in a block of a higher level blockchain, i.e., we create a two-level hierarchy of blockchains. Validating the integrity of a high-level block confirms the integrity of all the blocks of the lower-level blockchain and leads to a reduction of the number of operations needed to validate the chain. The LCaaS is a hierarchical blockchain framework, graphically shown in Figure 3.6. The figure depicts a two-level hierarchy, but the number of levels can be increased if a use case requires it.

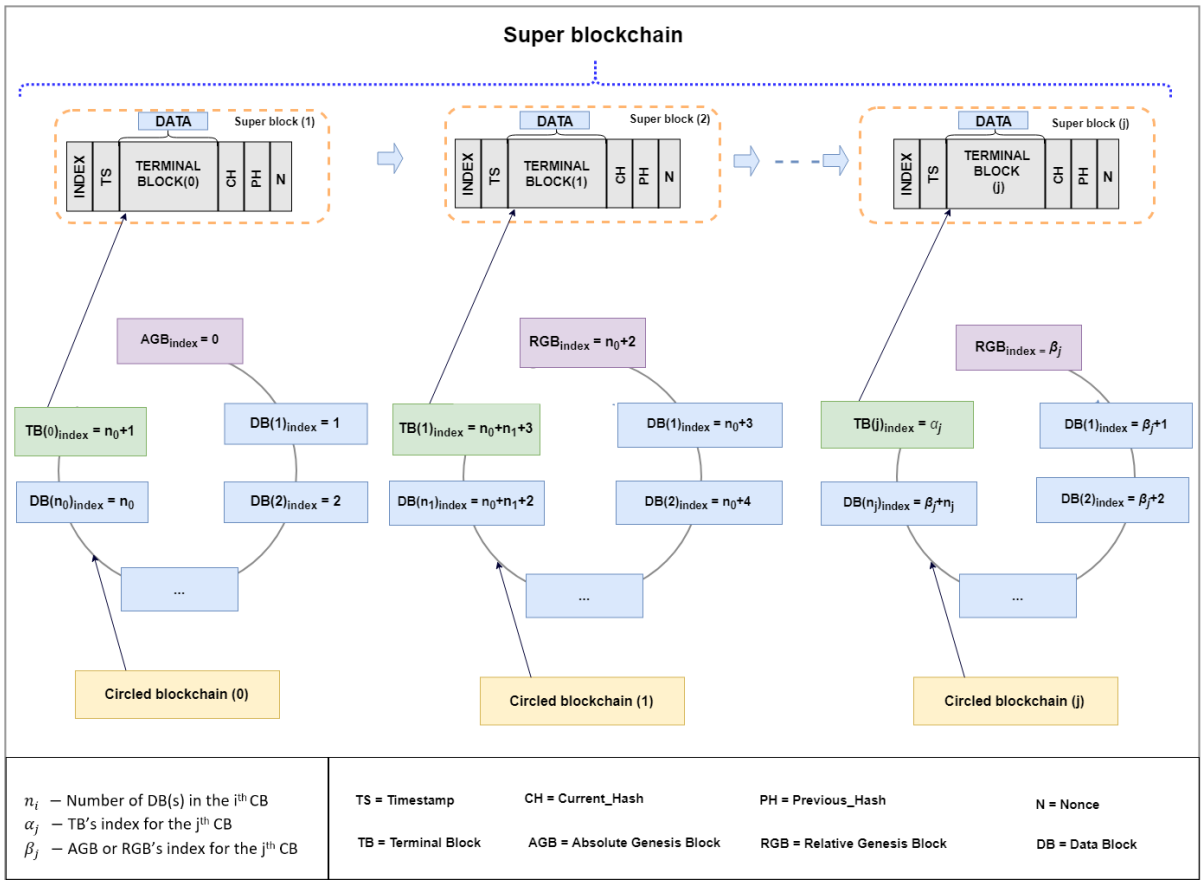

Figure 3.6: Two-level Hierarchy as Implemented by LCaaS

As mentioned in the legend of Figure 3.6, n_i refers to the number of data blocks in the i-th circled blockchain and α_j is the index of the terminal block of the j-th circled blockchain. β_j is the index for the absolute or relative genesis block of the j-th circled blockchain. The value of α_j will be calculated by

$$\alpha_j = \begin{cases} n_0 + 1, & \text{if } j = 0 \\ n_0 + 1 + \sum_{i=1}^{j} (n_i + 2) = n_0 + 1 + 2j + \sum_{i=1}^{j} n_i, & \text{if } j \geq 1 \end{cases} \tag{3.7}$$

and the value of β_j will be calculated by

$$\beta_j = \begin{cases} 0, & \text{if } j = 0 \\ \alpha_{j-1} + 1, & \text{if } j \geq 1 \end{cases}. \tag{3.8}$$

Logchain resides on top of a basic blockchain and converts it to a hierarchical ledger. Our primary goal is to bring scalability to blockchain for the situations in which the number of data items that need to be stored in a blockchain is large (e.g., operational logs of a Cloud platform). Figure 3.7 depicts the high-level architecture of the implemented platform.

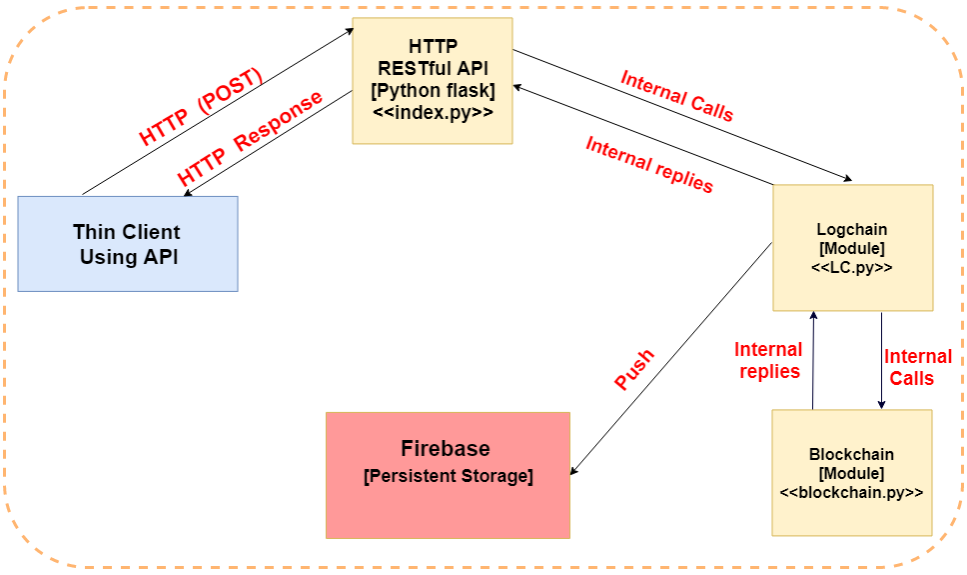

Figure 3.7: Architectural Components of LCaaS

At its current state, the prototype includes an entirely independent blockchain framework and receives data via APIs and stores them in blocks of the blockchain. Finally, the prototype can convert the blockchain to circled blockchains and forms a hierarchical ledger. The prototype supports the following features:

- Absolute genesis block;
- Relative genesis block;
- Data block;
- Terminal block;
- Circled blockchain;
- Super block;
- Super blockchain;
- Hierarchical ledger;
- API to receive raw data, or hash data from CSP or CSC;
- API to verify raw data, hash data, or TB;
- Control the number of blocks in a circled blockchain;
- Configurable difficulty target; and
- Configurable number of blocks in a circled blockchain.

As the name implies, the LCaaS offers the hierarchical ledger as a service. Cloud participants can create an account and receive a unique API key for all corresponding API calls. Clients also need to configure two main settings on their instance before they can use it.

The first key configuration is the `difficulty_target` which is defined as the number of required zeros at the beginning of an acceptable hash. The LCaaS will continue to generate new hashes and new `nonces` until a hash is generated that matches the difficulty target.

The second key configuration is defining a limit for the number of blocks in a circled blockchain. This constraint acts as size-limit and controls how many blocks are accepted in each circled blockchain. Once the limit is reached, the LCaaS takes the blockchain and pushes it to the hierarchical ledger. Let us now look at the key components of the LCaaS.

3.3 Key Components of LCaaS

While common key components of blockchains are necessary to implement a functional blockchain framework, our prototype requires additional components. Here we provide details of the components that the LCaaS has added to the blockchain framework. We have expanded the basic genesis block concept and introduced absolute genesis block and relative genesis blocks. Moreover, we have introduced new components, such as terminal block, super block, and super blockchains. These advancements allow the LCaaS to provide the hierarchical structure that improves the scalability of blockchains.

3.3.1 Absolute Genesis Block (AGB)

Absolute genesis block is placed as the first block of the first circled blockchain. An AGB is the first block that is created in the LCaaS and has the same characteristics as GB, with `index` and `previous_hash` set to zero and the `data` element set to null. Here is the summary of AGB's predefined settings:

- AGB's `current_hash` = SHA of all other elements of this block;
- AGB's `previous_hash` = 0;
- AGB's `index` = 0;
- AGB's `data` = null.

3.3.2 Relative Genesis Blocks (RGB)

Relative genesis block is placed at the beginning of every subsequent circled blockchain after the first circled blockchain. The `previous_hash` of an RGB is set to the `current_hash`

of the terminal block of the previous circled blockchain. Here is the summary of RGB's predefined settings:

- RGB's `current_hash` = SHA of all other elements of this block;
- RGB's `previous_hash` = current_hash of the previous TB;
- RGB's `index` = `index` of the previous terminal block + 1;
- RGB's `data` = null.

Figure 3.8 illustrates the connection between the AGB and its subsequent RGB.

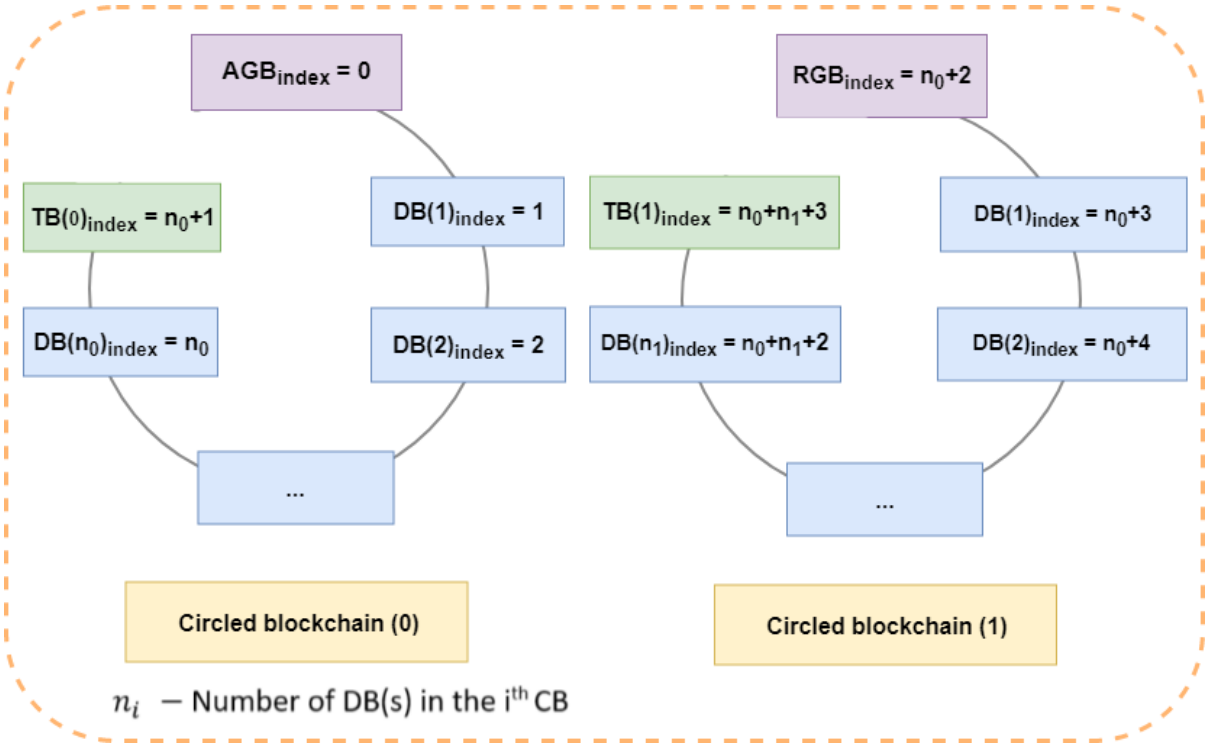

Figure 3.8: AGB and RGB in Circled Blockchain(0) and Circled Blockchain(1)

3.3.3 Terminal Blocks (TB)

Terminal Blocks are similar to genesis blocks, but they are added at the end of a blockchain to "close" it and produce a circled blockchain. The terminal block's `index`

and current_hash are calculated similarly to any other block. The part that differentiates a terminal block from a genesis block or a data block is its data element. The terminal block's data element stores a JavaScript Object Notation (JSON) object that contains details about the circled blockchain that it has terminated. The details are as follows. The aggr_hash is created by collecting and hashing current_hash values of all blocks in that circled blockchain, from the AGB or RGB to the block before the terminal block. The data element also store four additional values, namely timestamp_from, timestamp_to, block_index_from, and block_index_to. Here is the summary of TB's settings:

- TB's current_hash = SHA of all other elements of this block;
- TB's previous_hash = current_hash of the previous DB;
- TB's index = index of the previous data block + 1;
- TB's data = JSON object.

Figure 3.9 shows the JSON object that is stored in the data element of a terminal block.

```
{
    "aggr_hash": "",
    "timestamp_from": "",
    "timestamp_to": "",
    "block_index_from": "",
    "block_index_to": ""
}
```

Figure 3.9: JSON Object in the Data Element of a Terminal Block

In the above JSON, aggr_hash is constructed by collecting and hashing current_hash values of all the blocks in the circled blockchain that the terminal block is terminating. Storing the current_hash values of the blocks in the current circled blockchain forms a hash tree and can be used to efficiently verify the integrity of all the blocks of the circled blockchain by verifying the integrity of the terminal block for that circled

blockchain. The other components of JSON object stored in the data element of the terminal block are used for verification proposes. The `timestamp_from` and `timestamp_to` show the time span that circled blockchain has covered and the `block_index_from` and and `block_index_to` can be used to find if a block with a particular ID resides in a given circled blockchain.

Figure 3.10 shows the relationship between a terminal block and all other blocks in a circled blockchain. All elements of a super block are identical to the ones of a data block. Thus, it can be implemented by any other blockchain framework.

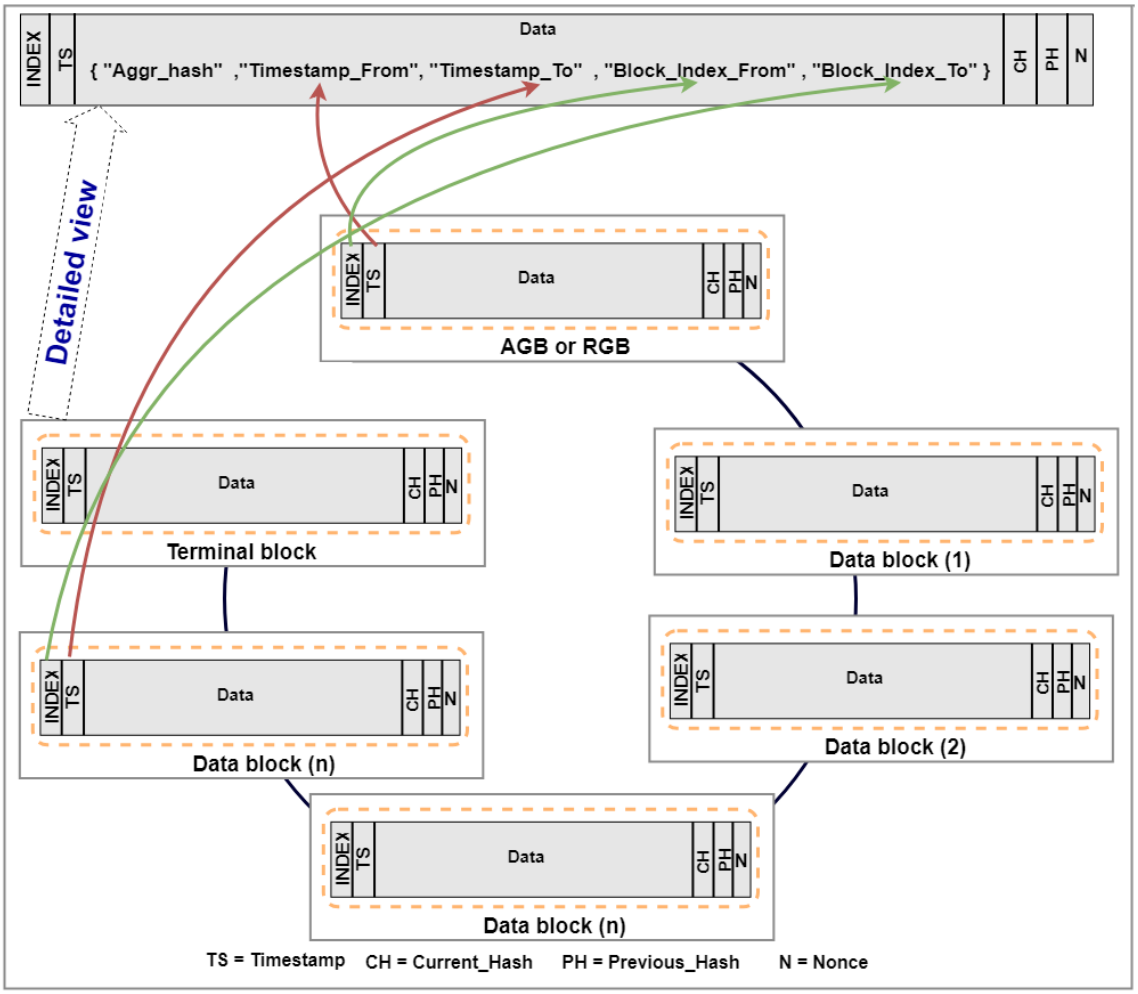

Figure 3.10: The Relation Between Terminal Block and All other Blocks in a Circled Blockchain

3.3.4 Circled Blockchains (CB)

Circled blockchains are blockchains that are capped. In other words, there is a limit on the number of blocks that they can include before a terminal block "caps" the blocks. Once a circled blockchain is terminated by a terminal block, it can no longer accept any new block.

3.3.5 Super Blocks (SB)

Super blocks exhibit the features of regular data blocks and have `nonce`, `index`, `timestamp`, `data`, `previous_hash`, and `current_hash`. The only difference between a super block and data block is that super block's `data` element stores all of the field of a terminal block of a circled blockchain. In order to accept terminal block elements, the `data` element consists of a JSON object. The elements of this JSON object are as follows: `index`, `timestamp`, `data`, `current_hash`, `previous_hash`, and `nonce`. Here is the summary of SB's settings:

- SB's `current_hash` = SHA of all other elements of this block;
- SB's `previous_hash` = `current_hash` of the previous SB;
- SB's `index` = `index` of the previous data block + 1;
- SB's `data` = JSON object.

Figure 3.11 shows the JSON object that is stored in the `data` element of a super block. Figure 3.12 depicts the relationship between a terminal block and the `data` element of a super block.

```
{
    "index": "",
        "timestamp": "",
        "Data": {
                "aggr_hash": "",
                "timestamp_from": "",
                "timestamp_to": "",
                "block_index_from": "",
                "block_index_to": ""
        },
        "current_hash": "",
        "previous_hash": "",
        "nonce": ""
}
```

Figure 3.11: JSON Object in the Data Element of a Super Block

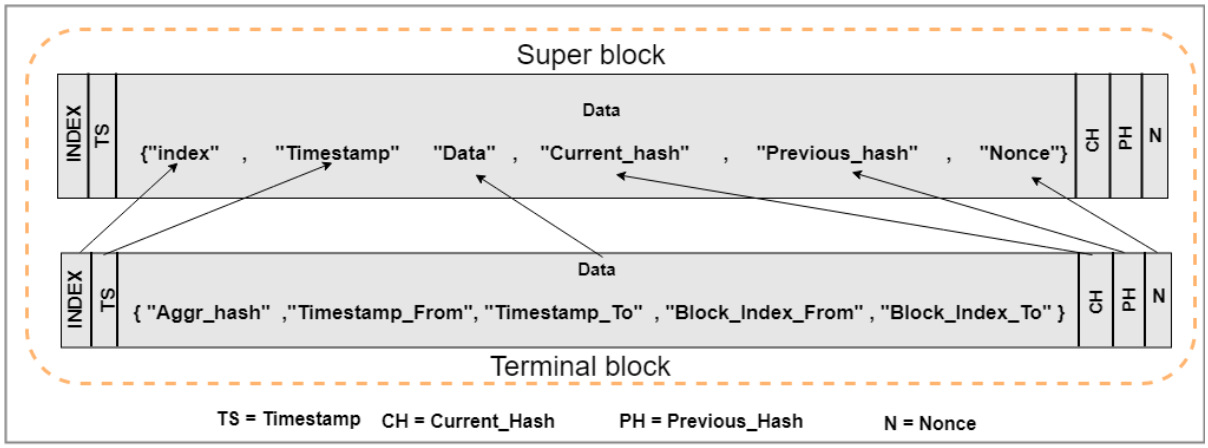

Figure 3.12: The Relationship Between Terminal Block and Super Block

3.3.6 Super Blockchain (SBC)

Super blockchain is a blockchain where each of its blocks is a super block. The super blocks are "chained" together by hash binding. In other words, super blocks that are linked together will result in a super blockchain. An i-th super block in a super blockchain relies on `current_hash` of its previous super block. If data in an earlier super block, say, super block m is tampered, the link among all the subsequent super blocks, from $m+1$ to the most recent super block, denoted by super block i, will be broken. Then one has to recompute `current_hash` and `nonce` values of each super block from super block m to super block i.

Considering that a super block's data element includes all the elements of a terminal block, changing any block in a circled blockchain, not only breaks the circled blockchain but also breaks the super blockchain. The above structure provides a hash tree structure and enhances the immutability of data stored in blocks of circled blockchains, while decreasing the computational resources required to verify blocks in a circled blockchain.

These novel enhancements (introduced in Section 3.3) allow the LCaaS to provide the hierarchical structure that is needed to overcome scalability limitations of the blockchains. In the following section, we dive into the implementation details of the LCaaS, its logic, and flow.

3.4 LCaaS Implementation

Here we describe the flow of LCaaS and how it interacts with the user-provided data. The source code for the prototype is available on the LCaaS GitHub repository [89] as well as in the appendices of this thesis.The LCaaS is equipped with an API that can be used to send data to LCaaS or to verify the integrity of data at hand by comparing it with the version of data that is stored in blocks of the LCaaS. The details of the LCaaS API are presented in Section 3.5.

3.4.1 Programming Language, Libraries, Code Repository

We have implemented the LCaaS based on the Object Oriented (OO) model. The prototype is implemented in Python 3.6. Table 3.1 shows libraries that we have used to

deliver LCaaS and describe their usage.

Python Library Name	Usage
datetime	To implement timestamp feature in blocks, terminal blocks, and super blocks
hashlib	To calculate the hash of a given input and return its hexdigest value
json	To work with JSON objects (e.g., parse, read and write in JSON format)
Flask	To implement the RESTful API that allows LCaaS customers to interact with it
pyrebase	To implement interaction with Google Firebase

Table 3.1: Python Libraries Used in LCaaS

3.4.2 LCaaS Modules

It is important to mention that we have not used any library, SDK, or third-party code to handle the logic for blockchain and blocks. Instead, all required functionality of the blockchain is developed from scratch. The reason is that we wanted to ensure that our proposed solution applies to a basic blockchain framework without relying on advanced features. Furthermore, keeping the framework simple makes it easier for one to understand the logic introduced by the LCaaS.

We have created three separate modules. Advanced care is carried out to achieve a fine balance of cohesion and coupling in class designs. That is, classes are designed as independent as possible, but whenever possible, we have used the OO generalisation techniques (Superclass and Subclass) to prevent redundant logic and code.

The customer-facing module is known as “index”, its logic is implemented in “index.py”, and its code is available in Appendix A. It uses Python Flask [19] and creates a Flask-based RESTful API interface that listens on the TCP port 5000 and interacts with the user using HTTP protocol. It uses Flask internal web server and bounds its web service to the available local interface. For heavier deployments and production environments, we recommend the internal Flask web server to be replaced by a more scalable

and heavy-duty web server such as Apache Web Server [6].

Upon running, the LCaaS platform instantiates an object from the class "Logchain" and associates it with a customer by a unique customer ID. Moreover, it constructs a connection to the persistent data storage component that uses Google Firebase real-time database [18]. The Firebase database is a Cloud-based database that stores data in JSON format and synchronises them in real-time to all clients that are connected to the same schema. The real-time synchronisation can be used in the future to create a distributed ledger in which all nodes are connected to the same schema and automatically receive updates with the newest items in the LCaaS. Once the database connection is established, the component waits for a request from the user. Upon receiving the request, the index module will look at the configurations to find the actions that are needed to construct the next data block. Based on the number of allowed blocks in a circled blockchain and the state of the latest added block, the new data block will be added to the current circled blockchain or a new circled blockchain will be created and then the data block is added to it. In the latter case, a new RGB is needed before the data block can be added to the new circled blockchain. In any case, the index module will rely on the functionalities that are provided by the second module.

The second module is known as "Logchain" and is delivered in "lc.py"; its code is available in Appendix B. This module implements the logic of converting the incoming data (via "index" module) to data blocks and puts data blocks in circled blockchains and generate a terminal block for each circled blockchain. The module also contains the logic for `Logchain` class. The `Logchain` class connects circled blockchains and their terminal blocks to a super block and eventually to a super blockchain. Moreover, this module implements the logic of creating super block and super blockchains. This module contains all additional components that we have added on top of regular blockchain features, namely, extension to the genesis block, terminal block, circled blockchain, super block, and super blockchains. The final part of this module introduces the composite data type for the terminal block's `data` element. As it was indicated in Section 3.3.3, the terminal block's `data` element consists of aggregated hashes of all the blocks in the circled blockchain that is terminating. Additionally, the `data` element includes `timestamp_from`, `timestamp_to`, `block_index_from`, and `block_index_to`. Figure 3.13 shows the interconnection among classes in Logchain module.

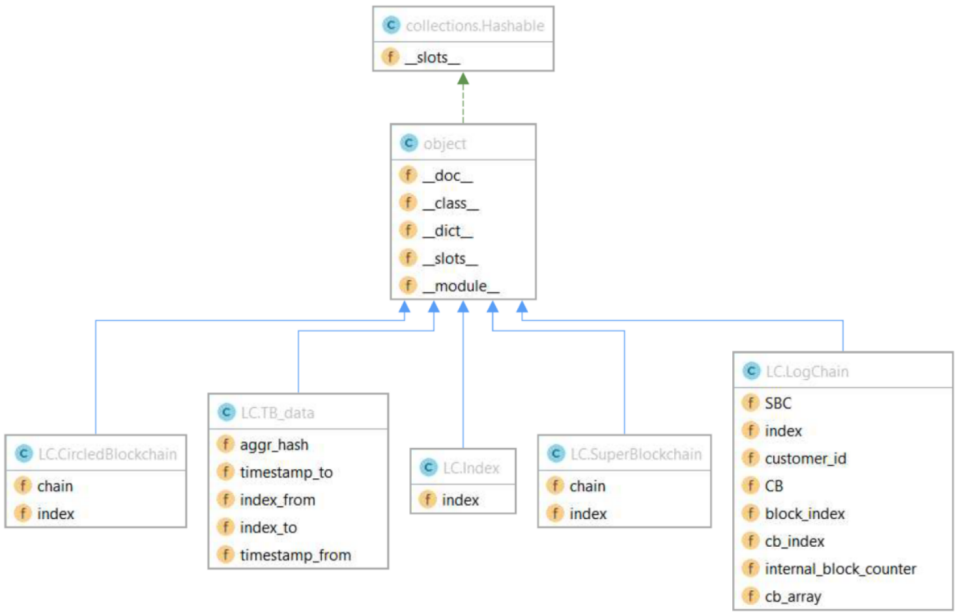

Figure 3.13: Interconnection Among Classes in Logchain Module

The third module referred to as "Blockchain" is delivered in "blockchain.py" and its code is available in Appendix C. It handles the internal affairs of a blockchain and acts as a blockchain framework. The module includes the Block class and all its methods that can be called to create genesis blocks, data blocks, and terminal blocks. It also takes care of hash generation and mining of blocks by generating a `nonce` and `current_hash` that match the configured `difficulty_target`. While this module provides the framework for blockchain, it contains no logic for converting data to data block and assigning data blocks to the circled blockchains. This design is intentional, as we wanted to ensure that our blockchain framework is as basic as possible and can be replaced by another similar blockchain platform. Figure 3.14 shows the interconnection among classes in the blockchain module.

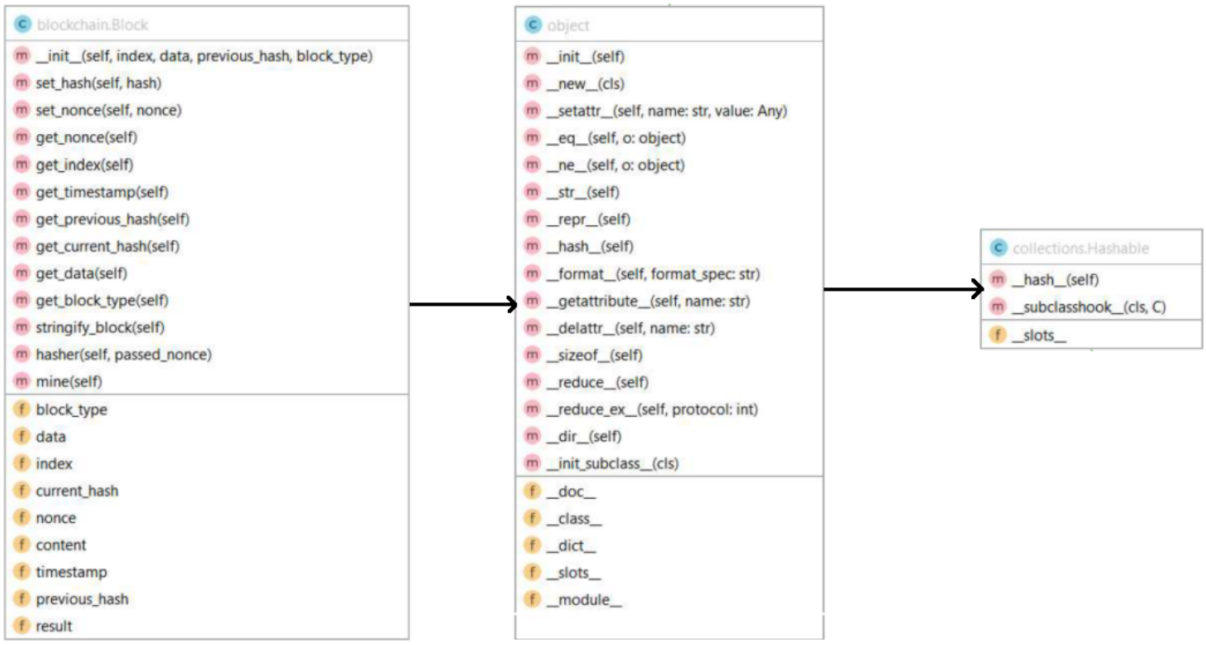

Figure 3.14: Interconnection Among Classes in Blockchain Module

3.4.3 Configuration

The LCaaS is designed as a configurable software. It loads various configuration from a JSON configuration file. Table 3.2 provides a list of configurable items in "config.json" file.

Configuration Parameter	Usage
DIFFICULTY_TARGET	Set the acceptable difficulty target for the generated current_hash. By default it is set to 000, where a represents a character rather than bit.
GENESIS_HASH	This is the default value of previous_hash for AGB.
MAX_NUMBER_OF_BLOCKS_ IN_CIRCLED_BLOCKCHAIN	Defines the capacity of a circled blockchain. In other words, it defines the number of blocks in each circled blockchain before a terminal block is needed for that circled blockchain.

Table 3.2: Configuration Items for LCaaS

3.4.4 Process Flow

In this section, we define the process flow of the LCaaS from the time that a customer sends a request to store data in the blockchain all the way to the processes that put the data in blockchain and return the confirmation to the customer using HTTP response calls.

Upon running the LCaaS, the application instantiates a new instance of `Logchain` class. This object has four indexes, each of which is instantiated from the `Index` class. Table 3.3 lists the indexes of `Logchain` class. These indexes keep track of the position of the most current block and its type, its `current_hash`, and its `index`. Such details are needed to create the next block and to implement hash binding between a block and its consecutive block. Moreover, these indexes are used to decide whether the next data block has to be added to the current circled blockchain or a to new one.

Configuration Parameter	Usage
`block_index`	This is used as the main index counter for blocks in the lifecycle of instances of the `Logchain` class. This index is associated with number of blocks in a blockchain. It starts from 0 and can go to m for a blockchain with m blocks.
`cb_index`	This is used as the main index counter for circled blockchains in the lifecycle of instances of `Logchain` class. It represents the number of circled blockchain in a LCaaS instance. It starts from 0 and can go to k for a LCaaS that has k circled blockchains.
`internal_block_counter`	This counter keeps track of the number of blocks inside of each circled blockchain and it gets reset to 0 when a new circled blockchain is created.
`sbc_index`	This counter keeps track of the index of the most recent super block that is added to the super blockchain.

Table 3.3: Indexes Used in `Logchain` Class to Keep Track of the Most Recent Added Block and its Details

In addition to the above indexes, the `LCaaS` class includes an array `cb_array`. This array of objects stores circled blockchain objects. Each circled blockchain object has a set of attributes that can be accessed by a set of defined "setters" and "getters" methods. Once the LCaaS object is instantiated all of the above indexes are set to 0 and the `cb_array` is set to an empty array. Upon receiving a request from a user, the LCaaS uses the following logic to determine the steps that are needed to convert the received data to blocks in the blockchain. It is important to mention that the procedure is heavily dependent on the position of the latest added block. A new block can be the first block of the LCaaS and, therefore, requires an AGB before it can be stored in the blockchain. Another scenario could be that the new block requires a new circled blockchain and a new RGB. These scenarios are handled by the `blockify` method. Figure 3.15 presents

the internal logic of this method.

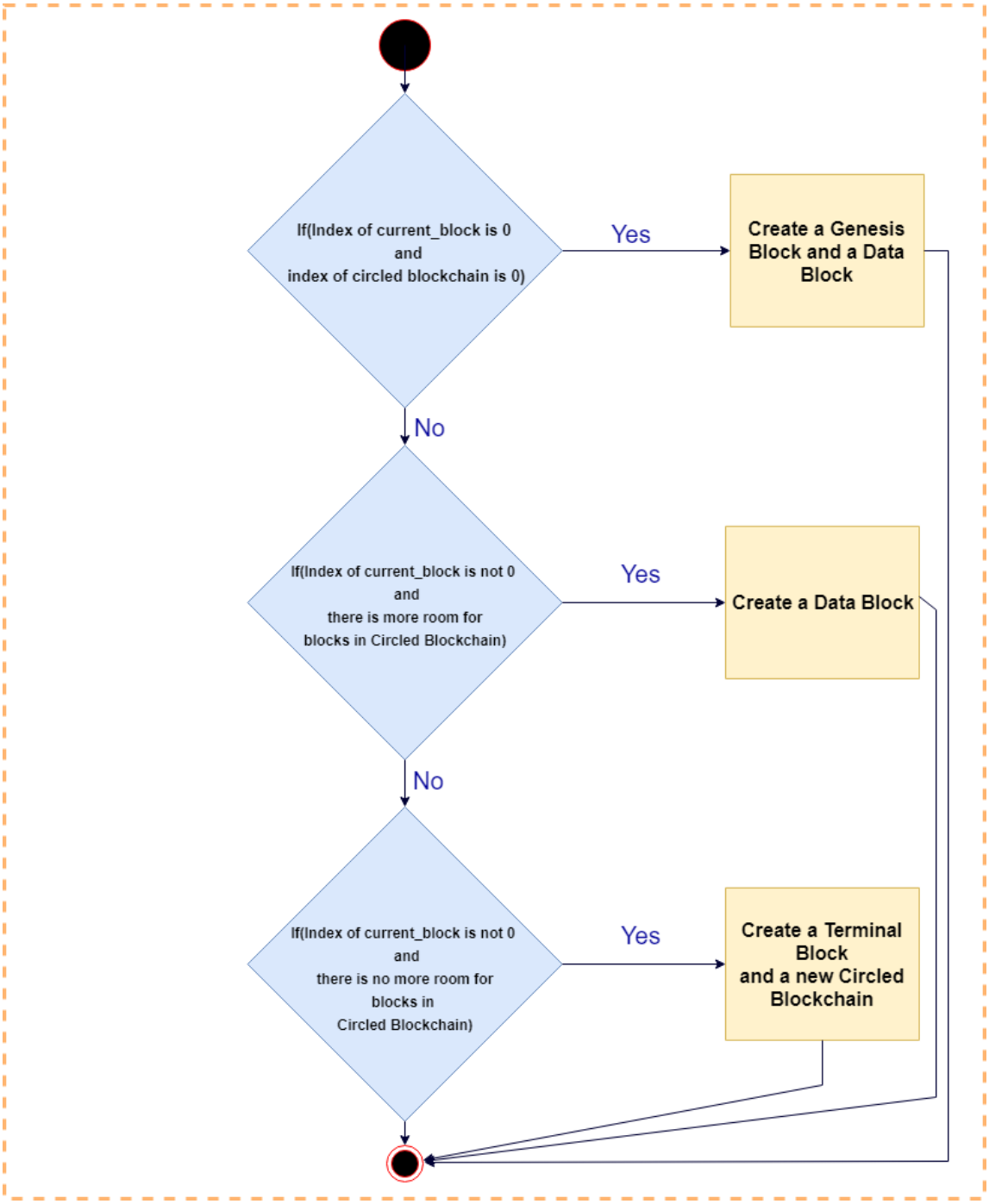

Figure 3.15: The Internal Logic of `create_new_block` Method

3.4.5 Blocks and Their Types

The blockchain module is responsible for all types of blocks that are created. The Logchain module calls the `create_new_block` method of this module and passes the `block_type` parameter into this method. Based on the passed value, the method uses its internal logic to create an appropriate block. Figure 3.16 presents the internal logic of this method.

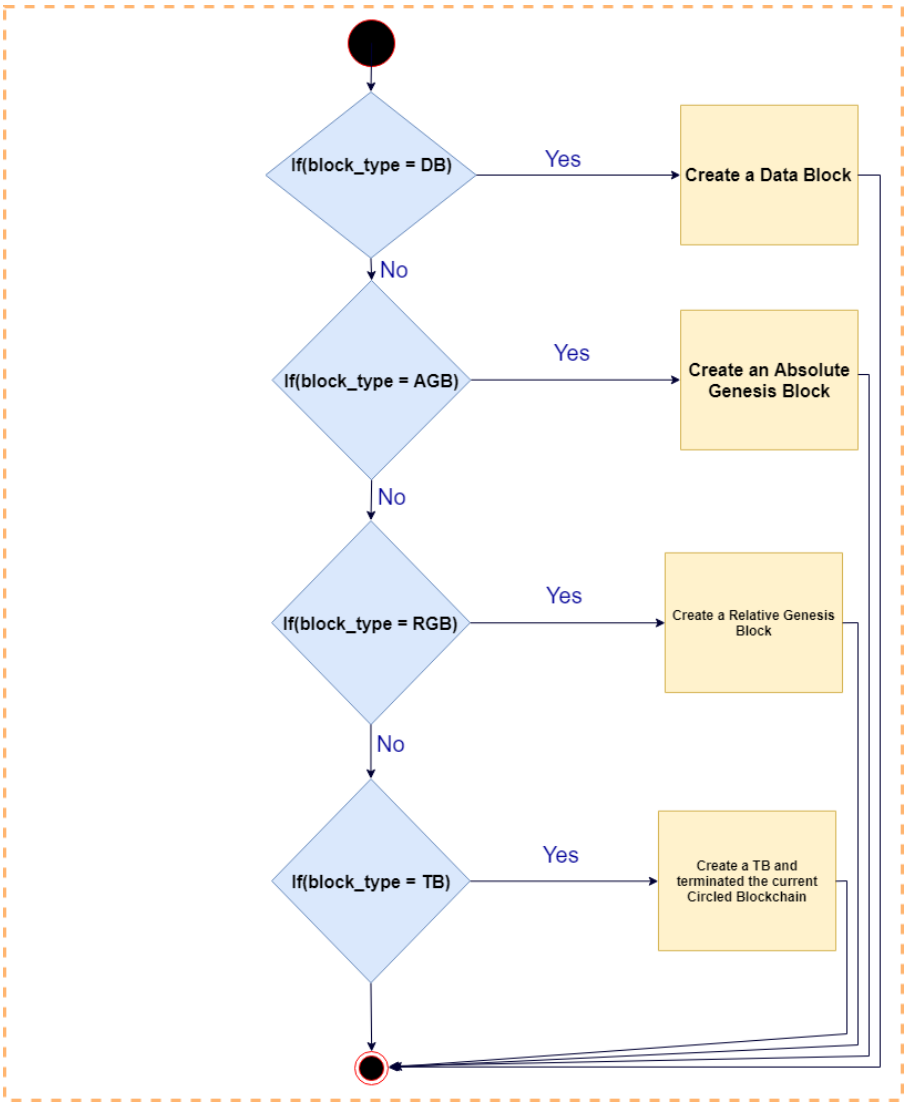

Figure 3.16: The Internal Logic of `create_new_block` Method

3.4.6 Block Presentation

Once LCaaS generates a new block, a copy of the block object is stored in memory of the Python environment. Saving blocks as objects, allow the LCaaS application to access their values at any time using `object.attribute` commands. Furthermore, proper logs are added to the Python console for each operation of the LCaaS. This internal log allows the operators of the LCaaS to ensure that Logchain is operating normally and all blocks are added correctly. Part of these logs are trimmed, annotated, and used as the HTTP responses. The HTTP responses will be sent to the client who has sent the request via API to inform them about the result of their request. It is important to mention that to increase the readability of blocks in logs, in the HTTP responses, as well as at the persistent storage, we convert objects to their string representation. The following example depicts the reason behind this conversion.

Figure 3.17 presents a terminal block in raw format with data element as a composite data type that is instantiated from `TB_Data` class. The first element in the figure is the `index` and the second element is the `timestamp`. The third element is the object reference as opposed to its content. Hence, if it is saved in the persistent storage, it does not include the details that are needed for future verification.

```
1 (4, '2018-06-27T02:36:07.458670',
2 <LC.TB_data object at 0x0000022F0B7EDF28>,
3 '0002fe60d37f40c1ae39202c999fdae7dfc5cdaced301484ae40d4c7c87ab3d1',
4 '0002dc6e4a47737a0f81340f423e126414f13eb0faf6dbaf0cd250f89feadf69',
5 801, 'TB')
```

Figure 3.17: A Terminal Block `data` Element Shown as an Object

To overcome this issue, a `stringify` method is defined that converts the object to its attributes and their values. The stringified version of the same terminal block is presented below. As it can be seen in Figure 3.18, the object is no longer presented with its reference, and the terminal block includes all the detailed of the circled blockchain that it has terminated in the string format.

```
1 (4, '2018-06-27T02:36:07.458670',
2  'aggr_hash:
3  dea3fffccea81f685fed03d0fa6b5c73714ec20b633adabd17928b52a9efc7b5',
4  'timestamp_from: 2018-06-27T02:36:05.282661',
5  'timestamp_to: 2018-06-27T02:36:06.886907',
6  'index_from: 0', 'index_to: 3',
7  '0002fe60d37f40c1ae39202c999fdae7dfc5cdaced301484ae40d4c7c87ab3d1',
8  '0002dc6e4a47737a0f81340f423e126414f13eb0faf6dbaf0cd250f89feadf69',
9  801, 'TB')
```

Figure 3.18: The Stringified `data` Element of a Terminal Block

3.4.7 Persistent Storage of Blocks

Once blocks are generated, the LCaaS will push them to the persistent storage. We use Google Firebase Real-time Database to store all types of blocks.

Figure 3.19 shows a sample of JSON objects stored in the Firebase database. The first object is shown in lines 1 to 6. The `Index` and `Type` elements of each object are just for enhanced visibility for user and search. The main content of the block is stored in the `content` element. Within the `content` element, the first item is the `index` of a block. The second item is the `timestamp`. The third item is the `current_hash` and the forth item is the `previous_hash`. The next item is `nonce` and the last item is the block type.

The hash binding relationship can be seen between the AGB at `index:0` and the first data block with `index:1`. The `previous_hash` of the block with `index:1` is equal to the `current_hash` of the block with `index:0`. Furthermore, as the capacity of circled blockchain in this example is set to 5, the block with `index:4` is a terminal block and terminates the circled blockchain. At this stage, The LCaaS has to create a new circled blockchain and generate a new RGB for it. The block with `index:5` shows the newly created RGB block. As it can be seen in line 33, the `previous_hash` of the RGB is equal to the codecurrent_hash of the terminal block (shown in line 27).

```
"{"Index": 0, "Type": "AGB", "Content": [0,"2018-06-26T20:24:55.803085",
"Absolute Genesis Block",
"000e5e2237f21b5c35a9c7368166cf7daf692d95a1fafaba69424f5d438ce715",
"0000000000000000000000000000000000000000000000000000000000000000",
3070, "AGB"]}",
"{"Index": 1, "Type": "DB", "Content": [1,"2018-06-26T20:24:55.995085",
{"Log": "User William formatted C Drive at 7:35 AM ,June 10th, 2018"},
"00073c9719c1468b5f8129b7df24e2a7af5087a71ba372f083122f8b116e462b",
"000e5e2237f21b5c35a9c7368166cf7daf692d95a1fafaba69424f5d438ce715",
4976, "DB"]}",
"{"Index": 2, "Type": "DB", "Content": [2,"2018-06-26T20:25:11.540994",
{"Log": "User Sara formatted C Drive at 10:35 AM ,June 11th, 2018"},
"00091859ca8ffdc2c07a76dd13f1f8842ac3d44e2582f178643e72befcaeed20",
"00073c9719c1468b5f8129b7df24e2a7af5087a71ba372f083122f8b116e462b",
4810, "DB"]}",
"{"Index": 3, "Type": "DB", "Content": [3,"2018-06-26T20:25:25.436190",
{"Log": "User Mamoosh formatted C Drive at 10:55 AM ,June 12th, 2018"},
"000fc7c8deaeadf7de19f22bb56b010ab18a2148a49a07a023b079ad2f90eb56",
"00091859ca8ffdc2c07a76dd13f1f8842ac3d44e2582f178643e72befcaeed20",
13267, "DB"]}",
"{"Index": 4, "Type": "TB", "Content": [4,"2018-06-26T20:25:37.282460",
"aggr_hash:
2aaeeac4349dce2a02c4146d9b7c88a23d7dd3702fd5440d7253245f4759753c",
"timestamp_from: 2018-06-26T20:24:55.803085",
"timestamp_to: 2018-06-26T20:25:25.436190",
"index_from: 0", "index_to: 3",
"0001141607c9c07e473e955b2ac97cfc83225f380bb00e3bc11d874928921c31",
"000fc7c8deaeadf7de19f22bb56b010ab18a2148a49a07a023b079ad2f90eb56",
2116, "TB"]}",
"{"Index": 5, "Type": "RGB", "Content": [5,"2018-06-26T20:25:51.977921",
"Relative Genesis Block",
"000adabfc998d9ac3ba68ce62a2de6d68b4b96520b341e0942b4bbb556336bed",
"0001141607c9c07e473e955b2ac97cfc83225f380bb00e3bc11d874928921c31",
8718, "RGB"]}",
```


Figure 3.19: JSON Objects Stored on the Firebase Database

To make data more readable for CSPs and CSCs, we have provided different buckets for each circled blockchain. Furthermore, we have introduced a bucket strictly for the storage of super blocks. Figure 3.20 presents the dashboard of Firebase and shows how blocks are stored in various circled blockchains. As it can be seen, a dedicated bucket for each circled blockchain is used. Moreover, a dedicated bucket for super blocks is created. In this example, the maximum number of blocks in each circled blockchain is set to 5 and that is why each circled blockchain consists of 5 blocks including the AGB or RGB, data blocks, and a terminal block at the end of each circled blockchain. The last part of this figure, shows a Superblock bucket and the genesis block of the super blockchain and the two super blocks that are created based on the terminal blocks of the two circled blockchain above it.

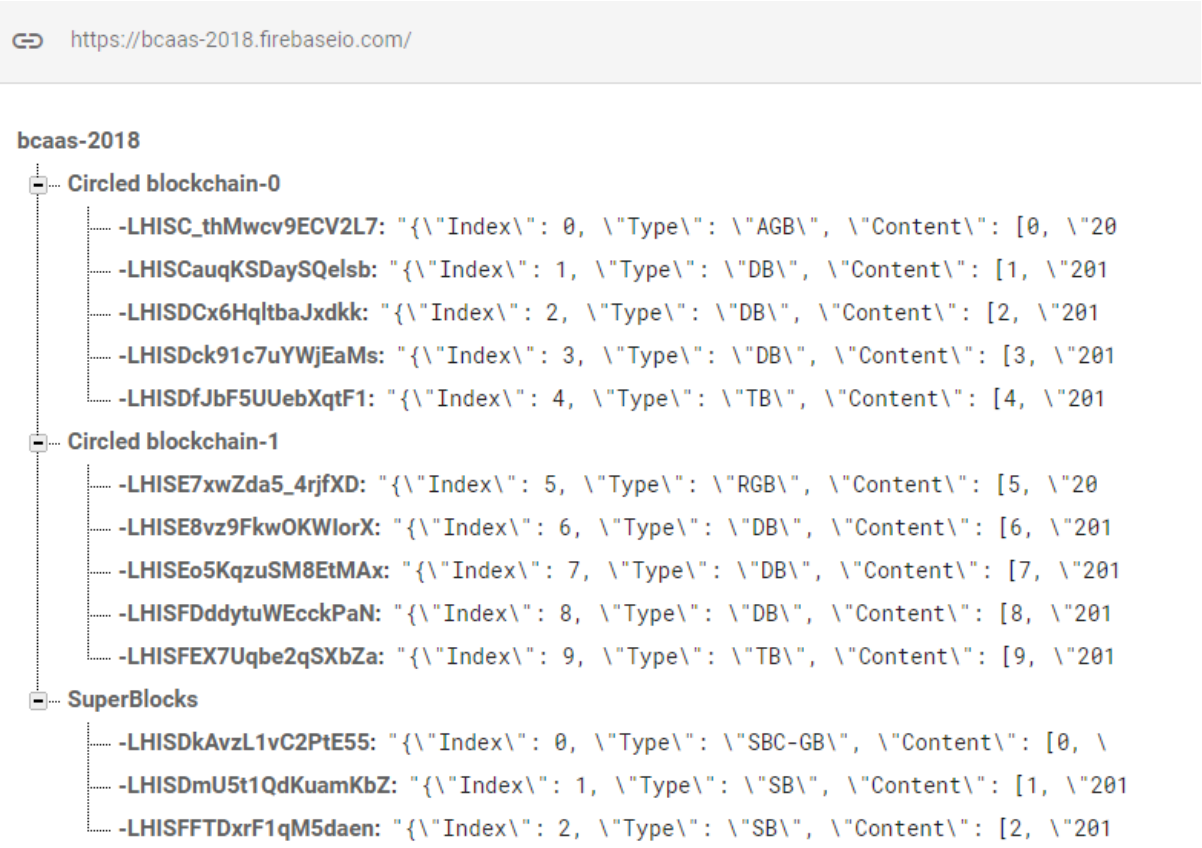

Figure 3.20: The Dashboard of the Firebase Showing Buckets for Circled Blockchains and Super Blocks

3.5 LCaaS API

To simplify the interaction with the Logchain platform, we have introduced a RESTful API that converts the Logchain to the LCaaS. The CSPs can efficiently use this API and interconnect the LCaaS with their monitoring systems and store all their logs, or the hash of their logs, in the Logchain. Similarly, CSCs can search and verify provided logs against the data in the Logchain and, therefore, be assured that the logs provided by the CSPs are not tampered. In the current implementation, the application receives logs or their hashes, adds them to the data blocks and mines the blocks by finding a `nonce` Like all other blockchains, our implementation links the blocks to their previous blocks by inserting the `current_hash` of the previous block into the `previous_hash` of the current block. The following section provides more details on the implementation of the LCaaS API. The LCaaS RESTful API has the following three characteristics:

1. **Client-Server**: There is a clear business logic separation between the tasks carried out by the API server and the responsibilities of the client
2. **Stateless**: Each request from a client contains all the information required by the LCaaS API to handle the request. In other words, the LCaaS API treats each request separately.
3. **Layered Structure**: Communication between a client and the LCaaS API is handled by the "index" module and no other modules are exposed to the clients.

The API is designed and implemented using Flask [19], a micro-framework for Python. The API methods are described below.

3.5.1 Submission Methods

There are two data submission methods: `submit_raw` and `submit_digest`. The former allows the client to submit the actual logs while the latter – just the file's digest (e.g., SHA-based digest computed using OpenSSL `dgst` [31]), thus, preserving the privacy of the log and reducing the amount of transmitted data. Both methods return, on success, `timestamp`, `block_index` and other details of the created block and, on failure, details of the error.

3.5.1.1 Submit_raw Method

This method is used by a CSPs or CSCs to submit their logs, in actual format, to the LCaaS. Here is an example for submitting a log snippet to the LCaaS using `submit_raw` submission method. Figure 3.21 shows the submission of raw log data through `submit_raw` method and the LCaaS response.

```
# Submission

curl -X POST \
  http://127.0.0.1:5000/submit_raw \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -H 'postman-token: ef8c9f1b-84ba-f43d-1503-f27ee1a58e9b' \
  -d '{"Log": "William formatted C Drive at 7:35 AM,June 10th,2018" }'

# HTTP Response from LCaaS

content-length:627
content-type:text/html; charset=utf-8
date:Fri, 13 Jul 2018 12:14:14 GMT
server:Werkzeug/0.14.1 Python/3.6.5
An AGB was created for the new circled block. AGB details are as follows:
(0, '2018-07-13T12:14:13.442103', 'Absolute Genesis Block',
'0005bdfed91ad4c04eccfbce9518e2218d44c59186b8fe73d1a0c6167c1c99bb',
'0000000000000000000000000000000000000000000000000000000000000000',
1869, 'AGB')
The new record has been successfully received and added to Logchain
with following details:
(1, '2018-07-13T12:14:13.944758',
{'Log': 'William formatted C Drive at 2:35 AM ,June 29th,2018'},
'000f9c99c25771b6940d1607de506a0357278d3f0608bea5d7cc0af22bb1903f',
'0005bdfed91ad4c04eccfbce9518e2218d44c59186b8fe73d1a0c6167c1c99bb',
6598, 'DB')
```

Figure 3.21: An Example of `submit_raw` Method and the LCaaS HTTP Response

3.5.1.2 Submit_digest Method

For privacy reasons the CSPs or CSCs may decide to generate a digest locally and submit it to LCaaS by using the `submit_digest` method. Here is an example for submitting digest of a log to LCaaS using `submit_digest` submission method. Figure 3.22 shows the submission of digest log data through `submit_digest` method and the LCaaS response.

```
# Submission

curl -X POST \
 http://127.0.0.1:5000/submit_digest \
 -H 'cache-control: no-cache' \
 -H 'content-type: application/json' \
 -H 'postman-token: 2b76fd3a-bebc-54a2-5a7b-ac76d491ceb4' \
 -d '{"digest":
 "50E721E49C013F00C62CF59F2163542A9D8DF02464EFEB615D31051B0FDDC326"}'

# HTTP Response from LCaaS

content-length:419
content-type:text/html; charset=utf-8
date:Fri, 13 Jul 2018 12:17:39 GMT
server:Werkzeug/0.14.1 Python/3.6.5
The new record has been successfully received and added to
Logchain with following details:
(1, '2018-06-27T14:53:37.881769', {'digest':
'50E721E49C013F00C62CF59F2163542A9D8DF02464EFEB615D31051B0FDDC326'},
'00073273f7957c99d666d0b63ad5db123e6238209c16ae38aa6da0087e6f14ed',
'0000bc96289a710506f00a8c99ca9822274b371d0586aa9ff14b9e2e45639646',
 9765, 'DB')
```

Figure 3.22: An Example of `submit_digest` Method and the LCaaS HTTP Response

3.5.2 Verification Methods

There are three verification methods: `verify_raw`,`verify_digest`, and `verify_tb`. The first one allows the client to verify the existence of actual logs in the Logchain, while the second allows the client to verify the digest of the logs. The last method allows user

to verify the existence of a terminal block with a specified hash in the Logchain, hence, proving the integrity of all the blocks in the circled blockchain of the submitted terminal block with on verification operation. All methods return, on success, the details of the found values in the Logchain and, on failure, details of the error.

3.5.2.1 Verify_raw Method

For the verification of the actual log content, one should use method `verify_raw`. The method would return the status of submission and number of blocks that match the submitted data; if no block is found, the API will return a message informing the user that no match has been found. In case of an error, the API will return the failed status along with the error's description. Figure 3.23 shows an example for log content verification request and response in LCaaS by using `verify_raw` verification method.

```
# Submission

curl -X POST \
  http://127.0.0.1:5000/verify_raw \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -H 'postman-token: bfbe3bfb-88e0-fbbe-2721-570712537d81' \
  -d '{"Log": "William formatted C Drive at 2:35 AM ,June 29th,2018" }'

# HTTP Response from LCaaS

content-length:439
content-type:text/html; charset=utf-8
date:Fri, 13 Jul 2018 12:18:31 GMT
server:Werkzeug/0.14.1 Python/3.6.5
An exact match for the submitted value has been found
(1, '2018-06-29T16:04:09.692996',
{'Log': 'William formatted C Drive at 2:35 AM ,June 29th,2018'},
'0008164d4115ae60c50ab8e0904bb29ff7cfdfeb852ede8d50e1c08cee972c8c',
'0005c7ea1a3dec05a6113b19a8648254e34604f0423386e878585c717f829fc2',
4312, 'DB')
```

Figure 3.23: An Example of `verify_raw` Method and the LCaaS HTTP Response

3.5.2.2 Verify_digest Method

For the verification of the digest of logs, one should use `verify_digest` method. The method would return the status of submission and number of blocks that match the submitted data; if no block is found, the API will return a message informing the user that no match has been found. In case of an error, the API will return the failed status along with the error's description. Figure 3.24 shows an example for digest verification request and the LCaaS response using `verify_digest` verification method:

```
# Submission

curl -X POST \
  http://127.0.0.1:5000/verify_digest \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -H 'postman-token: a8782a24-cdf9-3437-b4a3-cb6346007feb' \
  -d '{"digest":
  "57F9EE875542D6A64B48FC482B07DE146EA54685FB3F63DAC68D71DC1329FF82" }'

# HTTP Response from LCaaS

content-length:481
content-type:text/html; charset=utf-8
date:Fri, 13 Jul 2018 12:28:35 GMT
server:Werkzeug/0.14.1 Python/3.6.5
An exact match for the submitted value has been found
(2, '2018-06-29T16:07:41.996813', {'digest':
'57F9EE875542D6A64B48FC482B07DE146EA54685FB3F63DAC68D71DC1329FF82'},
'000f1d3d10d39edad059bddb2e8c7497a69730e6b8d000eb7159ae88e9c77cca',
'0008164d4115ae60c50ab8e0904bb29ff7cfdfeb852ede8d50e1c08cee972c8c',
494, 'DB')
```

Figure 3.24: An Example of `verify_digest` Method and the LCaaS HTTP Response

The internal logic for the the verification methods of LCaaS, namely,`verify_raw` and `verify_digest` methods are very close to each other. Figure 3.25 presents this logic.

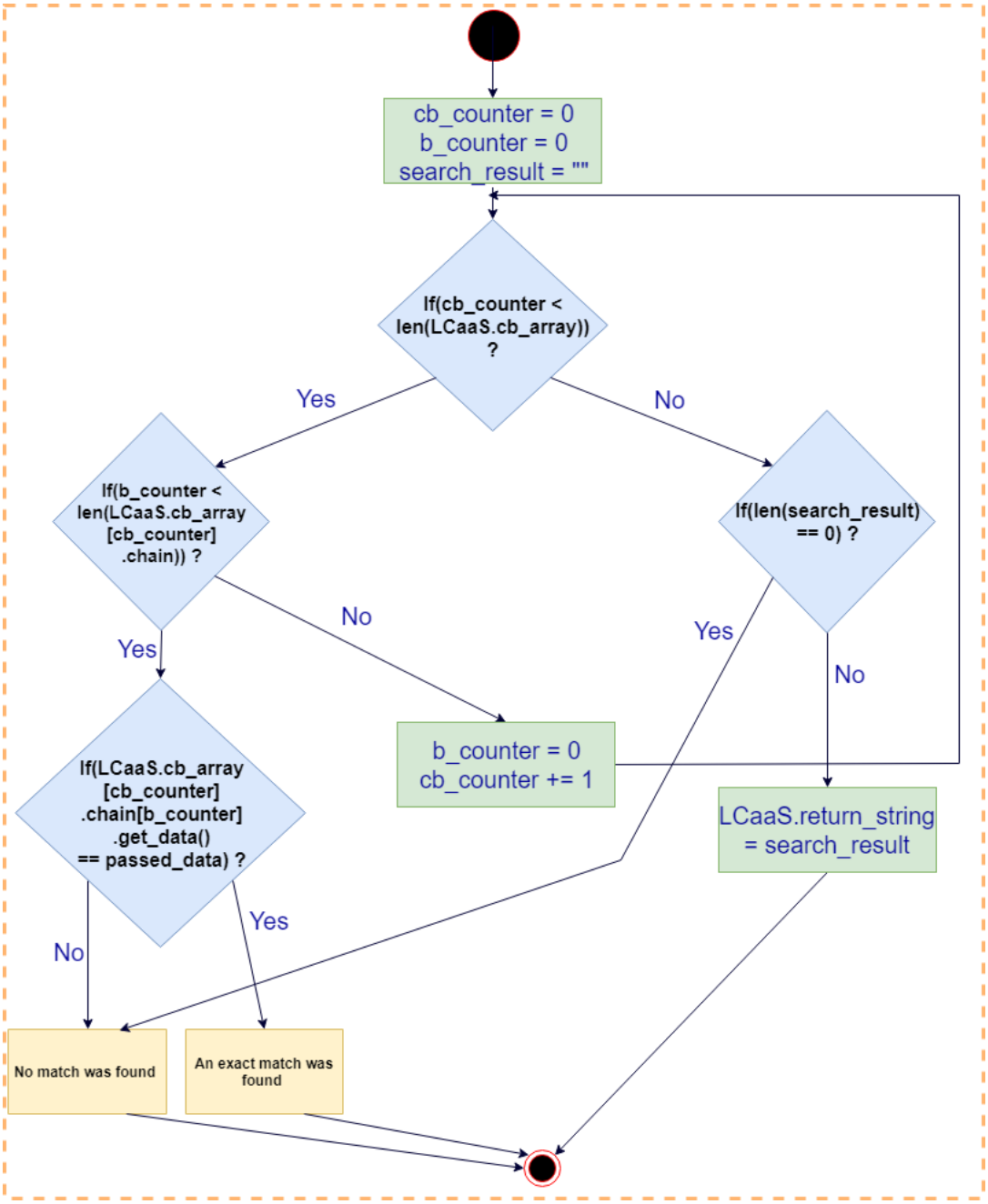

Figure 3.25: The Internal Logic of the `verify_raw` and `verify_digest` Methods

3.5.2.3 Verify_tb Method

To improve the scalability of our solution, we introduced the `verify_tb` method. It provides an assurance (in the cryptographic sense [97]) that the sequence of blocks, from `index_from` to `index_to` are not tampered. As it can be seen by generating a hash of all the `current_hash` values of all the blocks of a circled blockchain and compare it with the `aggr_hash` value of the data element of a TB, one can verify the integrity of all the blocks in the circled blockchain. Figure 3.26 is an example of hash verification for LCaaS terminal blocks by using `verify_tb` verification method. Figure 3.27 presents the internal logic of the `verify_tb` method.

```
# Submission
 curl -X POST \
  http://127.0.0.1:5000/verify_tb \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -H 'postman-token: c8381daf-25d4-a7d9-531d-c0efc4664e9a' \
  -d '{"tb_hash":
  "cb15f40cecc5db4541befe805133d700dd5d46c4bcab498251f8139d75972d02" }'

# HTTP Response from LCaaS
content-length:607
content-type:text/html; charset=utf-8
date:Fri, 13 Jul 2018 12:50:11 GMT
server:Werkzeug/0.14.1 Python/3.6.5
An exact match for the submitted value has been found
(9, '2018-06-29T16:39:58.666829',
'aggr_hash:
 cb15f40cecc5db4541befe805133d700dd5d46c4bcab498251f8139d75972d02',
'timestamp_from: 2018-06-29T16:07:53.683950',
'timestamp_to: 2018-06-29T16:39:57.843766',
'index_from: 5', 'index_to: 8',
'000276b9b3bc41f5517ea2b66ba951704cfd052f433a72e10b71bbca5a7d740a',
'000cf54ca70db4f9944712ebb928b215c118c19772080198cef7c8f5bf78e041',
2344, 'TB')
```

Figure 3.26: An Example of `verify_tb` Method and the LCaaS HTTP Response

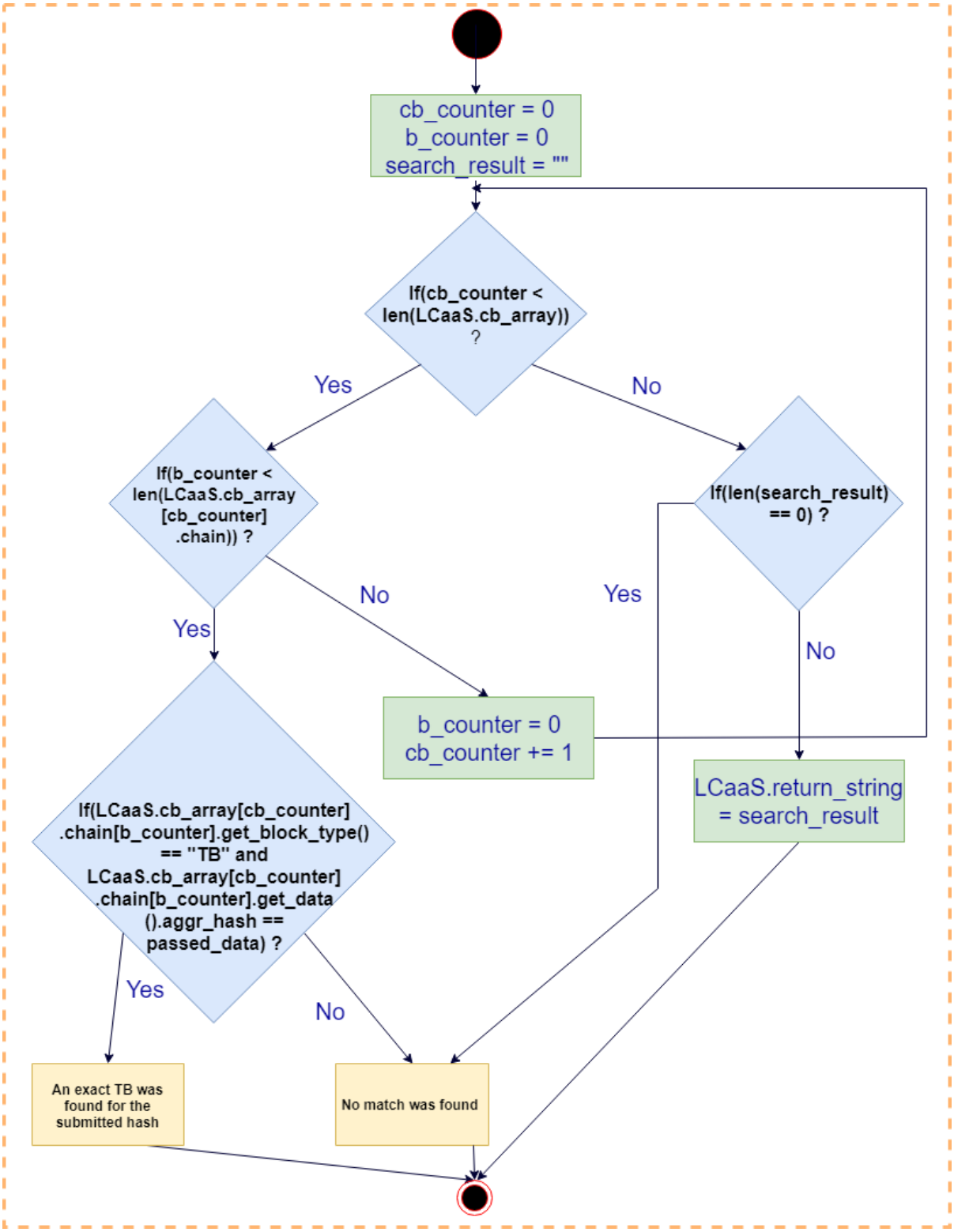

Figure 3.27: The Internal Logic of the `verify_tb` Method

3.6 Applicability to other Blockchain Platforms

As mentioned in Section 3.3, the following components were added on top of the existing blockchain components to make LCaaS possible:

- Absolute genesis block
- Relative genesis block
- Terminal block
- Circled blockchain
- Super block
- Super blockchain

It is important to mention that while we have introduced these additional components, we have tried not to alter the key element of the actual blocks. That is, our blocks, like any other blockchain's block, have the following key elements:

- `index`
- `timestamp`
- `data`
- `current_hash`
- `previous_hash`
- `nonce`

Avoiding any alteration on block's structure is intentional, because any modification in the blocks format (e.g., adding new elements) will result in a proprietary implementation of blocks and blockchains and will reduce the applicability of the proposed hierarchical structure to other existing blockchain platforms. In the next chapter, we will assess the applicability of the proposed solution to Ethereum blockchain platform.

Chapter 4

Evaluation

In this chapter, we assess the applicability of the proposed solution to other blockchain platforms. In other words, we will examine whether LCaaS can be implemented on top of the existing blockchains. We review the current existing blockchain platforms in Section 4.1 and then provide an overview of the Blockchain as a Service (BaaS) in Section 4.2. In Section 4.3, we review the rationale behind choosing Ethereum [15] as the selected blockchain platform to implement the LCaaS. In Section 4.4, we define integration points between the LCaaS and Ethereum, set the boundaries of each platform, and specify interfaces. Section 4.5 provides details of Ethereum blockchain and existing test networks that can be used to test the Ethereum blockchain without using the live blockchain. Section 4.6 provides implementation details. Section 4.7 concludes the chapter by reviewing and analysing the results of successful integration of the LCaaS and Ethereum blockchain.

4.1 Current Blockchain Platforms

In recent years, the concept of blockchain, its role, and use cases have become increasingly popular. Bitcoin, the oldest and the most famous application of blockchain, has reached the capital market of over 250 billion dollars in 2018 [36]. While the majority of blockchains have the same structure, they are generally categorised based on permission model [117] which defines how access to blockchain is configured and who can access it. The most common permission categories are public blockchains and private blockchains.

4.1.1 Public Blockchains

Public blockchains offer a shared and public ledger in which all records are visible to the public and everyone could take part in the consensus process. The first generation of blockchains and their applications, like Bitcoin, provided a public ledger to store cryptographically-signed transactions [116]. Public blockchains are often implemented over decentralised networks and offer permission-less read and write. That is, all public members can be a part of the distributed network and participate in mining process and, therefore, are allowed to add new blocks without permission from authorities. Public blockchains are also the oldest type of blockchains: they were introduced in 2009 by publishing the first transaction on the Bitcoin blockchain.

While a number of platforms (such as Bitcoin and Litecoin [24]) have focused on financial transactions, many companies have decided to offer non-financial services on top of existing public blockchains. The most well-known example, Ethereum, is the provider of the decentralised platform which allows developers to publish distributed applications. Ethereum has become increasingly famous as they have developed and released Solidity [34], a contract-oriented, high-level language for implementing smart contracts.

An example of a non-financial service over public blockchain is Factom [17]. As a provider of record management solutions, Factom stores the world's data on public blockchain platforms.

4.1.2 Private Blockchains

A private blockchain is regarded as a centralised network, because it is fully controlled by one or a group of organisations. Participants of a private blockchain have to be approved before they can act as a node in the blockchain. Moreover, nodes have to request read and write permissions. Participants have known identities, and the overall performance of the blockchain is higher than public blockchains due to a smaller number of nodes and smaller size of blockchains. In contrast to public blockchains, a central authority has the power to change the rules, smart contracts, and accept or reject participants. Private blockchains are the most flexible type of blockchains, as an organisation that owns the blockchain has full control over the configuration and implementation of the

blockchain. Multichain [27] is among the most well-known private blockchain platforms and allows the organisation to implement and set up a private blockchain. To standardise private blockchain offerings, Linux Foundation in 2016 started Hyperledger project [22] which contains a number of reference open source implementations of distributed ledgers created using blockchain technology.

It is important to mention that in recent years, a combination of public and private blockchain has been created. Known as the hybrid or consortium blockchain, where the control is extended to a certain number of people or nodes and can be beneficial in situations when a group of organisation/firms need to achieve a goal by collaborating with each other. Figure 4.1 depicts the difference between public and private blockchains.

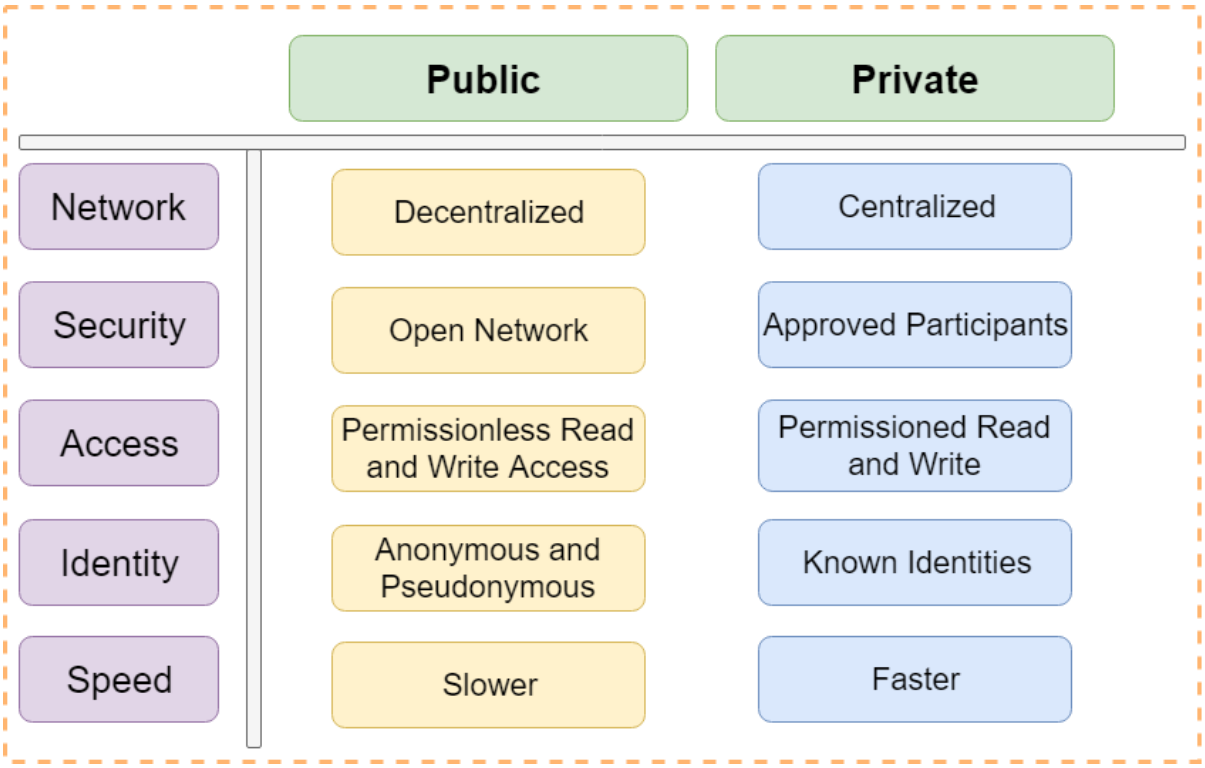

Figure 4.1: The Difference between Public and Private Blockchains

4.2 Blockchain as a Service

Blockchains are complicated platforms and require an extensive set of hardware and software components. Hence, deploying a reliable and scalable blockchain need both the capital and the human resources. However, many companies may need to implement a few prototypes and assess the pros and cons of blockchain solutions for the use cases they have in mind. To reduce the barriers to entry, many big players in the Cloud industry, have started to offer blockchain as part of their service offering. Public cloud providers, such as IBM and Microsoft, have all the required ingredients and therefore, have begun providing various level of BaaS to their CSCs.

To offer such services, CSPs had to partner with existing blockchain providers. For example, Microsoft has partnered with ConsenSys [10] to offer Ethereum Blockchain as a Service (EBaaS) on Microsoft Azure, and IBM has partnered with Hyperledger [21] to offer BaaS to its customers on IBM Cloud. Amazon has also started offering BaaS to its clients [8]. At the time of writing this thesis, Google has not officially launched any BaaS offering.

4.3 Ethereum

As mentioned above, while the focus of public blockchains (such as Bitcoin and Litecoin) is on financial transactions, platforms such as Ethereum try to provide a generalised platform for many different applications and use cases. Moreover, Ethereum provides the developer with an end-to-end system for building various distributed applications [114].

Another major focus of Ethereum is the smart contracts. The smart contracts are autonomous pieces of code [108]. They are deployed over the blockchain and are stored in blocks. Upon being called, like a function, smart contracts can interact with the user or data stored in the blockchain. Ethereum has developed Solidity [34], a high-level language for developers who want to build and deploy smart contracts on Ethereum blockchain. Additionally, to make the programming of smart contracts simpler, Ethereum team has provided Remix [33]: an online/offline Integrated Development Environment (IDE) for coding and compiling Solidity code.

Ethereum works based on the Ether currency. Ether is a necessary element for op-

erating Ethereum and is a form of payment made by the clients of the platform to the machines executing the requested operations (mainly smart contracts). All financial transactions, such as incentives and the transaction fees, are paid in Ether. Ether also allows Ethereum to remain a healthy network as it ensures that developers write quality applications as the inefficient code will cost more to run [13].

Given the popularity of Ethereum, its wide range of use cases, and its widely available development tools, we have selected Ethereum as the blockchain platform for integration with the LCaaS. In the following section, we provide details of this integration.

4.4 LCaaS and Ethereum Integration

As it was indicated in Section 3.4, we developed an internal blockchain to store logs that are sent to the LCaaS. Needless to say, this blockchain is a private blockchain and in order to replace it with a public blockchain (e.g., Ethereum) integration points have to be designed. We propose a composite structure in which receiving logs and converting them to blocks happens at the LCaaS side and storing the hashes and digitally signing them happens over the Ethereum blockchain. Using blockchain terminology, data collection and blockification of logs happen off-chain, and the block storage on the Ethereum blockchain is handled by Ethereum smart contract and will be on-chain.

Within the Ethereum blockchain, the economics are controlled by execution fee called gas. The gas is paid by Ether, the Ethereum intrinsic currency [114]. Gas measures, in computation resource terms, the effort that is needed to process the transaction. A smart contract consists of one or more operations and each operation has an associated gas cost which is defined by the Ethereum protocol [1]. For instance, a SHA3 operation costs 30 units of gas. The gas price is the amount paid per unit of gas and is defined by the initiator of the transaction. The gas limit is the maximum amount of units of gas expenditure per block [114]. The higher the gas price, the more appealing the transaction would become for the miners. Hence, if a transaction needs be executed faster, the higher gas cost will motivate a miner to consider the transaction and mine it in the upcoming block. The current implementation of LCaaS does not use a lot of computational resources for each block that is pushed to the blockchain, hence, the only real bottleneck is at the blockchain provider side. As indicated above, one can increase the performance of the

blockchain by increasing the gas price for the desired transaction.

In the light of the above economics, and the fact that each transaction incurs cost, we limited the submissions to the Ethereum blockchain to super blocks. Super blocks include complete elements of a terminal block of a circled blockchain and can be used to verify the integrity of all the blocks in that circled blockchain. Based on the "Preimage" Resistance property of hash functions described in Section 3.1.5, it would be computationally infeasible to construct an entire circled blockchain such that its hash matches the `current_hash` of a super block. However, if one desires, minor changes to the current implementation of the "ethereum" module can be made to allow the LCaaS to push data blocks to the Ethereum blockchain as well.

Figure 4.2 depicts the relationship between LCaaS and Ethereum.

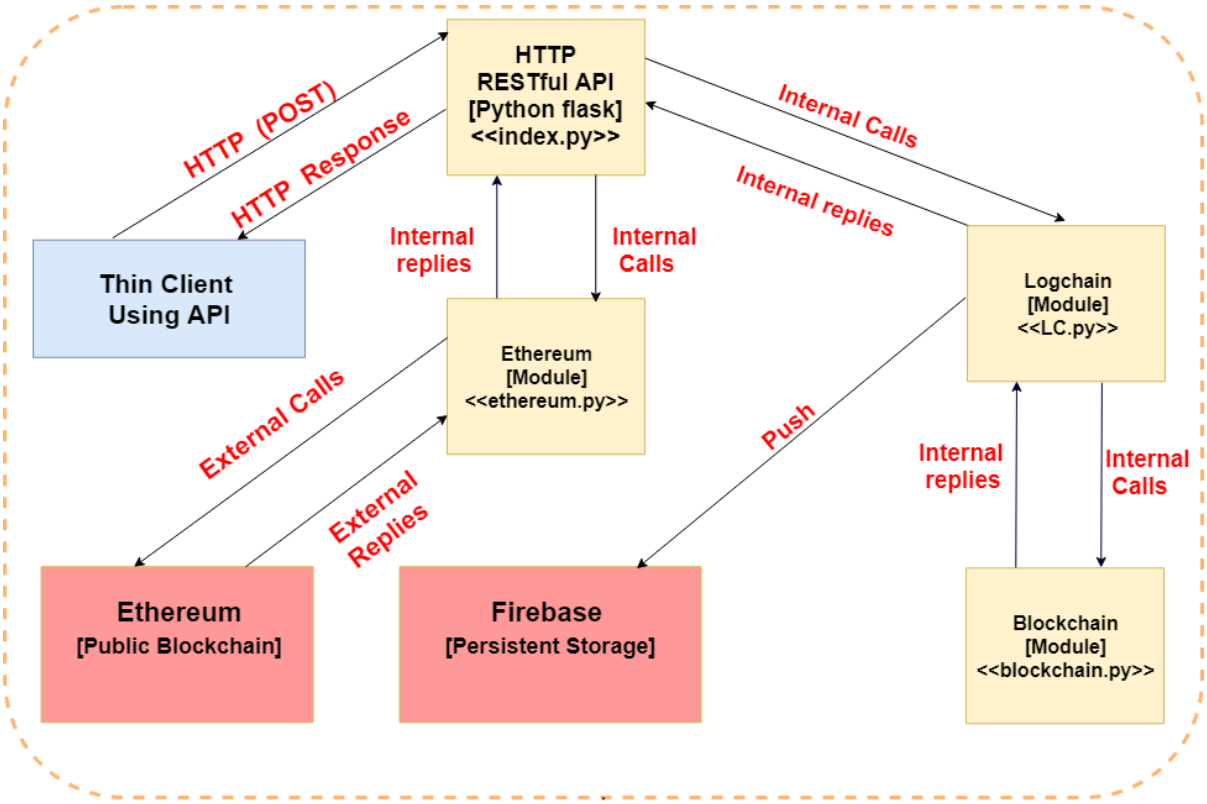

Figure 4.2: LCaaS and Ethereum Integration

4.5 Ethereum Blockchain and Test Blockchain Environments

As it was shown in Figure 4.2, blocks are created in the LCaaS and stored on the Ethereum blockchain. Since the main Ethereum blockchain is used for production and real transactions, pushing data and smart contracts to this network requires actual payment. To allow developers to test and interact with Ethereum blockchain, Ethereum has provided a real, live, and decentralised test blockchain that can be used throughout the entire development lifecycle and can be easily replaced by the real blockchain network when the developer decides to do so. There are two options to take advantage of Ethereum test network [38]. The first alternative, known as local, allows one to setup a node locally on a personal computer or a hosted server and run an isolated node that simulates nodes that are on the public Ethereum blockchain. The second alternative, known as the public, uses a decentralised test network that all developers around the world are constantly using. This live test network is more similar to the real Ethereum blockchain as it is a publicly available decentralised blockchain. Since, we need to use a test network that resembles all characteristics of the real Ethereum blockchain, we chose the second alternative to test drive the integration of the LCaaS and Ethereum.

4.6 Implementation

4.6.1 Additional Python Libraries

To interact with the Ethereum test networks, we use "web3.py" library [39]. This python library is inspired by the famous "web3" JavaScript library, and it allows Python applications to interact with the Ethereum blockchain. The "web3.py" supports all the features that we require to integrate the LCaaS and Ethereum blockchain. Namely, it supports sending Ether to Ethereum blockchain and allows one to publish a smart contract on the blockchain and interact with the functions of the published contract. We chose "web3.py" as it has a vibrant development community and is used extensively in multiple production-level projects. That is, it keeps current with latest features of the Ethereum blockchain.

4.6.2 Test Networks

As it was indicated in Section 4.3, we selected Ethereum as our public blockchain platform to assess the applicability of the LCaaS. At the time of writing this thesis, there are three Ethereum test networks [16]: Ropsten, Kovan,and Rinkeby. While all of them can be used for integration with the LCaaS, we chose Ropsten because it is the only Ethereum test network that works based on the PoW and, therefore, is more similar to the commonly used blockchains in the market.

4.6.3 Test Ether and Secure Vault

Considering the cost of mining and adding a transaction to a block, public blockchain platforms offer their services at a price. Customers, who are willing to interact with the public blockchain, should pay the required fee for every transaction. Interestingly, the Ethereum test networks follow the same logic. In fact, any interaction other than submitting a query against the test network requires a payment in Ether currency. However, a special type of Ether is designed for transactions on Ethereum test network. Known as test ether, the currency does not have any real monetary value, yet, it resembles the real Ether on the live Ethereum blockchain. To obtain test Ether, we use MetaMask Ether Faucet [37] and obtain 1.0 test Ether. Figure 4.3 depicts the purchase of a test Ether.

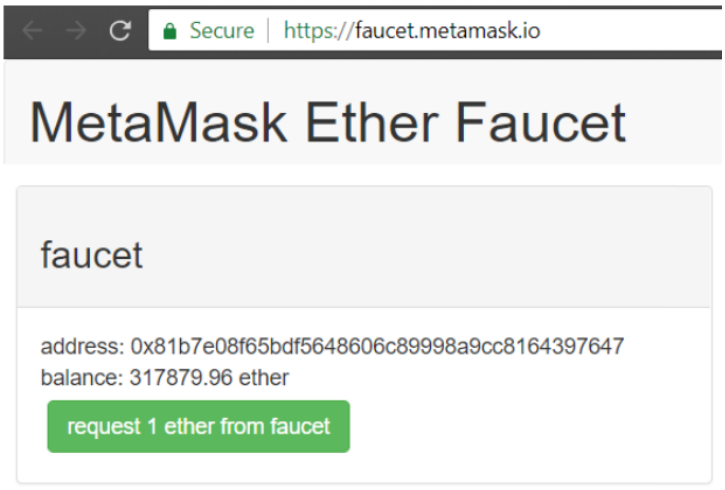

Figure 4.3: Test Ether Purchase via MetaMask Faucet

We use MetaMask [25] Chrome web browser extension to keep our test Ether in a secure vault. The MetaMask also allows users to choose the network they want to to connect to. We chose the Ropsten. Figure 4.4 shows MetaMask settings.

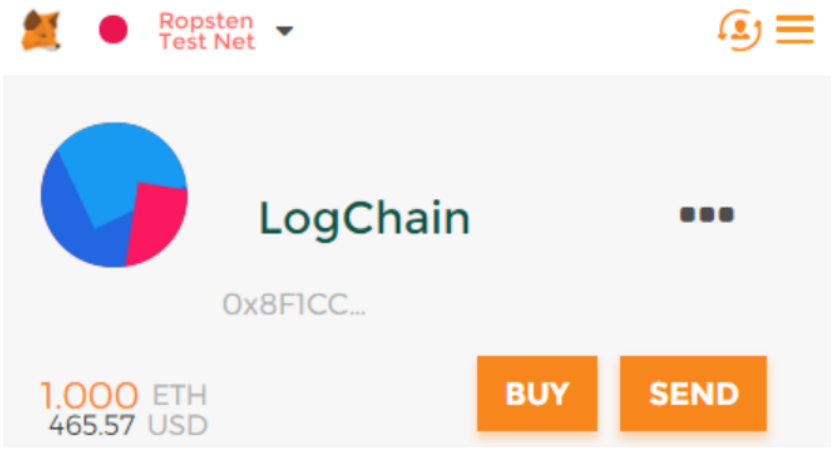

Figure 4.4: Logchain Balance

4.6.4 Publishing a Smart Contract Using Solidity

Smart contracts on the Ethereum blockchain allow users to interact with the blockchain and send data to blockchain. In other words, any interaction with the blockchain has to be managed by published smart contracts. The storage of super blocks in the blockchain requires a smart contract. It is important to mention that a super block's data element contains the terminal block of a circled blockchain. Hence, super blocks are the most efficient candidate to be stored in a public blockchain, as one can easily verify the integrity of a super block and conclude the integrity of all the blocks in the circled blockchain that the terminal block stored in that super block has terminated.

To publish our smart contract, we use Solidity [34], as it is the recommended language for developers to build and deploy smart contracts on Ethereum blockchain. Furthermore, we use Remix [33] as an IDE for coding and compiling solidity code. The smart contract implements the logic of receiving super blocks and pushing them into the Ethereum blockchain. We name this smart contract `Superblock.sol`[1]. The source code of the

[1]The `.sol` extension indicates that this file is a solidity file and can be compiled as a smart contract.

smart contract is available in Appendix F. Once the smart contract is developed using solidity IDE, it has to be published on the Ethereum blockchain. We use the Remix to publish the smart contract. It is important to mention that publishing a smart contract on the blockchain is considered a transaction and is a chargeable service. Thus, we use the test Ether that we have stored in MetaMask vault to pay the transaction fee. Figure 4.5 depicts the smart contract publish transaction.

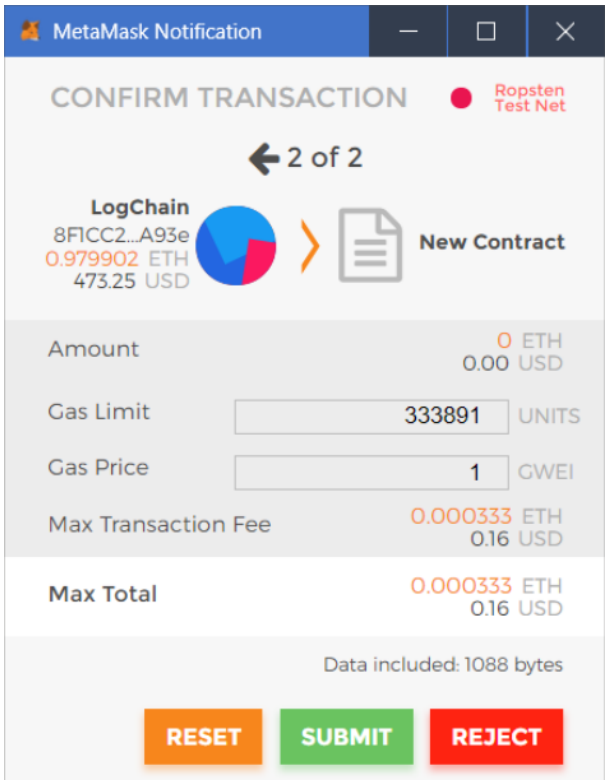

Figure 4.5: Transaction for Publishing a Smart Contract

By clicking on the `Submit` button, the contract developer agrees to pay the required fees for publishing the smart contract. Upon paying the fees, Remix provides a confirmation that the contract has been successfully compiled and published on the Ethereum blockchain. The published smart contract has a unique address that can be used to reference the contract and call its functions. Figure 4.6 shows the confirmation of successful deployment of the smart contract on Ethereum blockchain.

Figure 4.6: Superblock Smart Contract Successfully Added on Ethereum Blockchain

4.6.5 Interaction with Ethereum Ropsten and the Deployed Smart Contract

We introduce a new Python module and dedicate this module to interaction with the Ethereum blockchain. The new module is referred to as "ethereum", and its code is available in the "ethereum.py" file (available in Appendix D). In this module, interaction with Ropsten and smart contract is implemented.

For this module, to be able to communicate with the published smart contract, we need an interface. The most common interface for Ethereum is Application Binary Interface (ABI). An Ethereum smart contract is a bytecode deployed on the Ethereum blockchain. There could be several functions in a contract, and an ABI is needed to define which function in the contract can be called and what kind of signatures are needed to be sent to the functions of a contract. Furthermore, the ABI guarantees that the function will return data in the format expected by the application. We store the ABI of the deployed smart contract as a Python file and import it to newly added module. A copy of the "contract-abi.py" can be found in Appendix E. Smart contracts deployed on Ethereum blockchain are protected with immutability feature of blockchain and can no longer be changed. If any modification is required, the entire process has to be repeated,

and a new smart contract has to be published.

At this stage, we have the secure wallet and the required test Ether. The next step of the implementation is to connect the "ethereum" module to Ropsten. As indicated in Section 4.5, we do not use an a local node for Ethereum test network. Hence, we need a provider that provides the required connection to Ropsten. We use Infura [23] as an access provider and obtain a URL that can be used to interact with Ropsten.

Furthermore, to make Ethereum integration configurable, we introduce a new configuration item in addition to items described in Section 3.4.3. Table 4.1 provides details for this configuration parameter.

Configuration Parameter	Usage
PUSH_TO_ETHEREUM	If set to "Yes", a copy of every generated super block will be stored on the Ethereum blockchain. If set to "No", the feature is disabled.

Table 4.1: Additional Configuration Item Introduced to LCaaS

To interact with Ropsten, we instantiate an object from the Web3 class provided by the "Web3.py" library and link it to the unique URL provided by Infura. The instantiated object allows interaction with Ropsten and the deployed smart contract. The LCaaS platform is a pay-as-you-go platform with a flat membership fee. Hence, the premium features such as integration with Ethereum are only available to users who pay the premium membership fee. Therefore, we introduce additional logic in the smart contract to only accept transactions from users who have paid their membership fee. Once a user pays the membership fee, their address is added to the allowed list of senders and can push super block to the smart contract. Figure 4.7 presents the logic of checking the validity of a user to push super blocks to Ethereum.

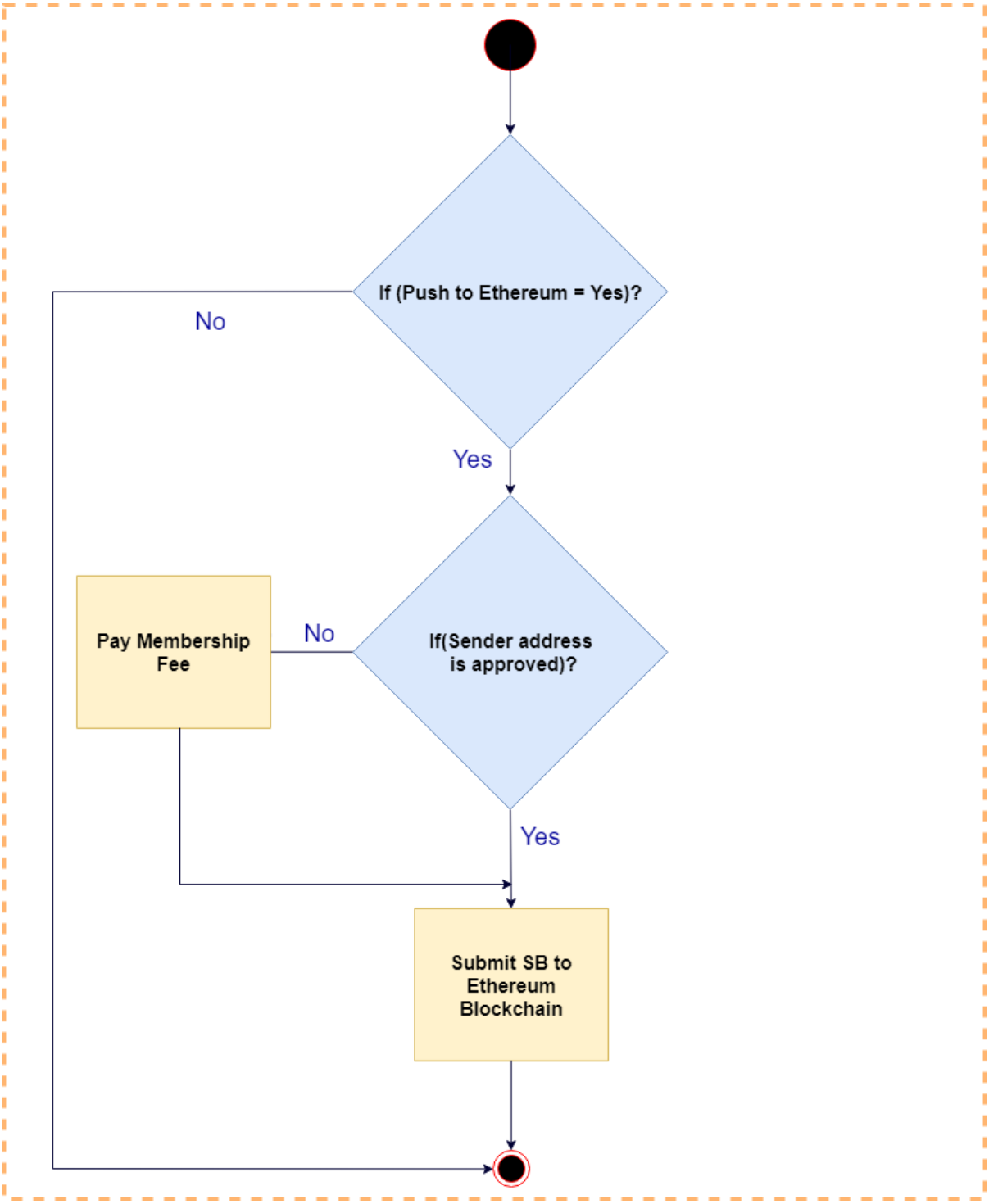

Figure 4.7: Sender Address Verification

4.7 Results Verification

Once the user's address is added to the allowed list, they can push super blocks into the published smart contract. The smart contract, stores the super block on the Ethereum blockchain and, upon successful submission, returns a receipt back to the "ethereum" module. The receipt includes details of the transaction such as the sender address, the content, the transaction hash, and the block number. Figure 4.8 depicts an example of the receipt sent back from the smart contract to the "ethereum" module.

```
{'status': 'added', 'processed_receipt': (AttributeDict({'args':
AttributeDict({'_sender': '0x3F4f9bb697F84A26fBc85883F2ff4d31a36ed83c',
'_superblock': "(0, '2018-07-10T11:31:53.938607',
'A Genesis Block for Super blockchain',
'000042d5f7c68c5223ac16d2ebb77f5008ad8acec56e88cd949ae6814938d6e9',
'0000000000000000000000000000000000000000000000000000000000000000',
4959, 'SBC-GB')"}),
'event': 'SuperblockSubmission',
'logIndex': 0, 'transactionIndex': 0, 'transactionHash': HexBytes
('0x03c8d2c6698e1a7296a08ffa42eea4b2d9d2b43ab948f75fee8ed327f507835e'),
'address': '0x6C6bF111B5D9D9060e53C5d967E0A7389D15634B',
'blockHash': HexBytes
('0xfcba2346909db2ace2e6c7cadf918624186ec4f088d4c5c03ad0ceb8c77b2e1a'),
'blockNumber': 3609334}),)}
```

Figure 4.8: Ethereum Smart Contract Receipt

All interactions with the Ethereum blockchain can be traced using Etherscan [14], a web dashboard connected to Ethereum blockchain. Known as Ethereum block explorer, Etherscan allows anyone to look up transactions details by using the sender or recipient address, transaction hash, or block number.

Using the Ethereum `blockNumber` field as well as the sender address one can easily look up a transaction on Etherscan and verify the transaction. If transaction is found and the submitted super block chain matches with the one at hand, the integrity of the

terminal block in the data element of the super block is confirmed, hence the integrity of all blocks in the circled blockchain that the terminal block has terminated. Figure 4.9 depicts the search result for the above transaction on Etherscan.

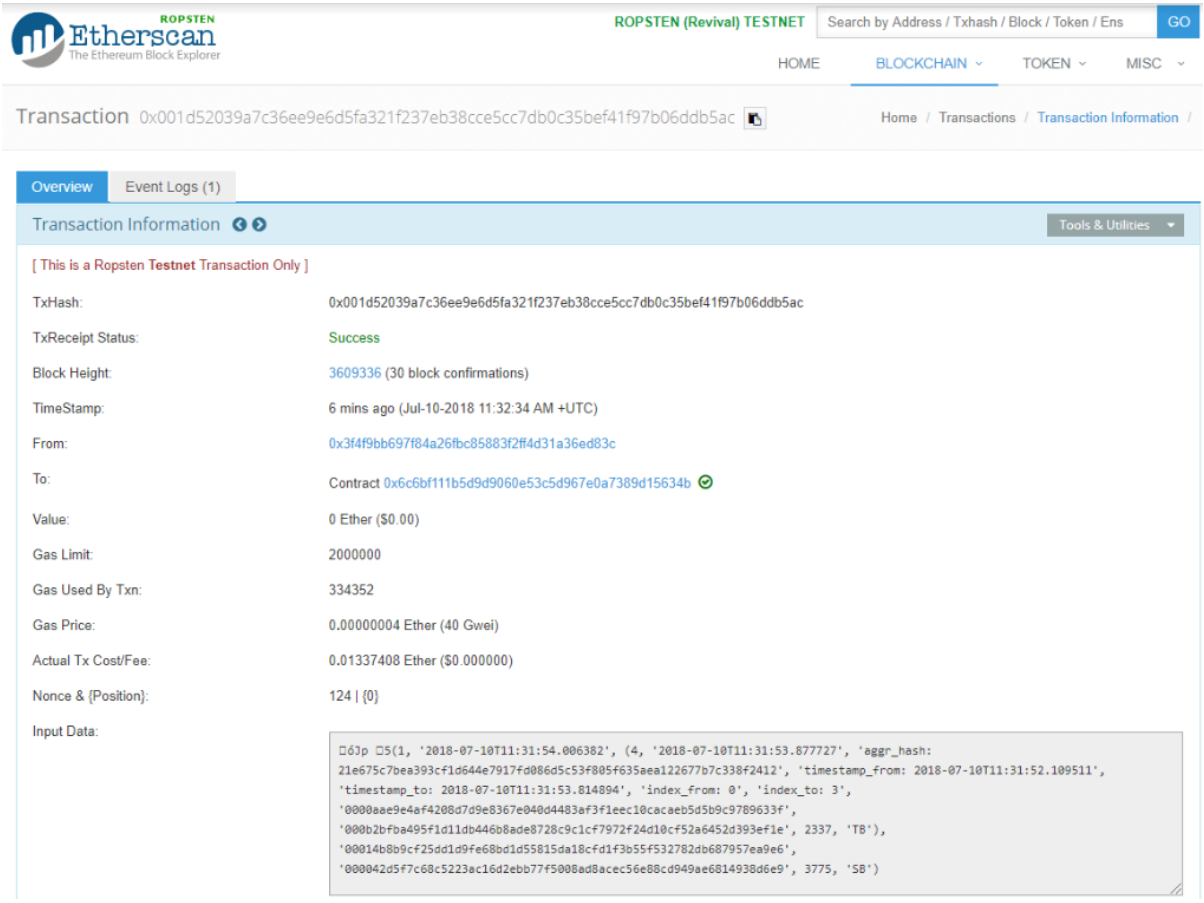

Figure 4.9: Super Block Submission To Ethereum Blockchain

Successful submission of a super block to Ethereum blockchain confirms that LCaaS' hierarchical ledger can be applied to other blockchain platforms such as Ethereum. Moreover, we conclude that the hierarchical ledger can be applied to all other Ethereum-based blockchains such as AWS Blockchain. If the logic of creating TBs, CBs, and SBs are migrated to smart contracts, one can drop the LCaaS entirely and instead, use its suggested hierarchical ledger on top of any blockchain platform.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we focused on building a tamper-proof log storage system.

The **primary** objective of this study was set to address the inadequacy of modern log systems by alleviating the required trust in a Cloud provider or a third-party. We addressed this objective by introducing LCaaS framework, called Logchain, that allows the storage of logs in tamper-resistant blockchains. We also showed that LCaaS can be implemented as a smart contract using existing blockchain platforms (namely, Ethereum).

The **secondary** objective was set to reduce the computational complexity of the Logchain. We addressed this objective by introducing a hierarchical ledger structure, where circled blockchains limit the number of blocks in each blockchain.

The **tertiary** objective was set to make stored logs accessible. We addressed this objective by designing and implementing an API that can be used to interact with the Logchain.

The Logchain exhibits the following characteristics.

Immutability: hash of each block is created as per pseudo code shown in Algorithm 1. It incorporates the hash of a previous block; thus, any changes to the previous blocks would "break" the blockchain[1].

[1]It is important to mention that blockchains are not immune to security challenges. Namely, security threats (such as majority attack and fork attack [72]) have been known and acknowledged in the blockchain community. We indicate that the intention of the proposed solution is not to address any of such security issues.

Cryptographically sealed: nonce-based proof-of-work method is used to ensure that generated hashes meet the configured difficulty target.

Scalability: relying on super blocks, many super blocks can be generated at the same time and then added to a super blockchain at the same time. This will bring parallel processing feature for situations where multiple sources of data are generating data that need to be put in the blockchain. For example, a platform may consist of twenty servers and each server can be associated with one super blockchain.

Accessibility: API-based verification is added to the hierarchical ledger so that users can submit raw data or digest values to check the consistency of their data.

Privacy: to improve privacy, an entire super blockchain is reserved for a client to ensure that blockchains from different clients are not mingled. Furthermore, a user will need to send only the terminal block to the LCaaS to verify the integrity of the entire circled blockchain. Moreover, the option to store the hash (digest) value of log data (as opposed to the actual data) would bring additional confidentially to the clients.

If LCaaS is implemented on a public blockchain, all transaction records are publicly available. This becomes a major privacy concern if the users of LCaaS decide to push their raw logs into the blockchain. Privacy can be improved if the users of LCaaS send the digest of their logs. If higher level of privacy is needed, one can implement LCaaS on a private blockchain.

The proposed LCaaS acts as a hierarchical ledger and repository for all logs generated by the Cloud providers and can be accessed by all the Cloud participants to establish trust among them. Using the provided API, a client can verify the logs provided by the Cloud provider against the records in the LCaaS and find out if the logs were tampered with or remained intact.

5.2 Future Work

To evolve LCaaS, we will adopt the two-pronged approach: improving the LCaaS itself and porting LCaaS to other blockchain platforms. The details of this approach are given below.

5.2.1 LCaaS Improvement

The current version of the LCaaS is the minimum viable product[2]. The following requirements will be considered for the next version of the LCaaS.

Database Improvement: The implemented version of the LCaaS only supports Google Firebase. We plan to introduce support for additional NoSQL databases (such as MongoDB [26], CouchDB [5], and DynamoDB [4]) and relational databases (such as MySQL [28]).

Search Improvement: The current search method of LCaaS is limited to objects stored in memory of the Python environment. We plan to expand our search method to support queries against the persistent storage.

Configuration Improvement: The current configuration setting of LCaaS are stored in a JSON configuration file. We plan to expand current configuration settings to include additional configuration points and allow users to choose the domain of applicability of each setting.

Performance Improvement: The current implementation of LCaaS uses serial processing for pushing terminal blocks to super blocks. By introducing concurrency in the LCaaS, users can plug several log streams into the platform to create a centralised immutable log storage system.

Reliability Improvement: The implemented version of LCaaS does not have any graceful restart feature. For instance, if LCaaS crashes during a transaction, the data and the state for all the live transactions are lost. We plan to implement fault-tolerance features for each module of the LCaaS in order to overcome this reliability issue.

Encryption Improvement: The current implementation of the LCaaS does not allow users to choose their desired encryption algorithm. We plan to make encryption algorithms configurable so that users can select their preferred encryption method.

API Improvement The search API of the LCaaS is currently limited to the five methods (described in Section 3.5). We plan to expand the LCaaS API to allow users to interact with all internal components, such as super blocks and super blockchains. As for the search API, we plan to expand the API functionality to include submitted data to the Ethereum network as well.

[2]I.e, the product has enough features that would satisfy initial clients. These clients would generate feedback that will guide future development of the product.

5.2.2 Hierarchical Ledger Implementation on other Blockchains

In the future, we plan to test LCaaS with other Blockchain as a Service solutions and find integration points that can be used to implement LCaaS on top of such solutions. Last but not least, we plan to convert the internal logic of LCaaS to smart contracts. Using smart contracts to handle the business logic of LCaaS makes the framework completely independent of the implemented solution and make it portable to any blockchain framework that supports smart contracts.

Appendices

Appendix A

Index Module (index.py)

```
1
2
3
4  # This is the Main file for the LCaaS project
5  # Designed and implemented by William Pourmajidi - 2018 - Canada Ontario
6
7
8  from LC import *
9  from flask import Flask, jsonify, request
10 import pyrebase
11 from ethereum import *
12
13 ### Firebase Settings ####
14 # Link: https://bcaas-2018.firebaseio.com/Blocks.json
15 # Link: https://console.firebase.google.com/u/0/project/bcaas-2018/database
16     /bcaas-2018/data
17 # Link: https://passwordsgenerator.net/sha256-hash-generator/
18 # You will need to change the following settings to your own Firebase
19     instance
20
21 config = {
22     "apiKey": "AIzaSyAmXGisFxk0xJmAT_KpFDvCmfqH-YBP_04",
```

```
23      "authDomain": "bcaas-2018.firebaseapp.com",
24      "databaseURL": "https://bcaas-2018.firebaseio.com",
25      "storageBucket": "",
26      "messagingSenderId": "568088402855",
27      "serviceAccount": "serviceAccountCredentials.json"
28  }
29
30  firebase = pyrebase.initialize_app(config)
31  auth = firebase.auth()
32  # # authenticate a user
33  user = auth.sign_in_with_email_and_password("william.pourmajidi@gmail.com",
34       "bcaas2018Pass")
35
36  # user['idToken']
37  db = firebase.database()
38
39  app = Flask(__name__)
40
41  with open('config.json', 'r') as f:
42      config = json.load(f)
43  data_storage_option = config['BLOCK'][
44      'DATA_STORAGE_OPTION'] # option to store actual data in the block or
45          store hash of data(more privacy)
46  max_number_of_blocks_in_circledblockchain = config['BLOCKCHAIN'][
47      'MAX_NUMBER_OF_BLOCKS_IN_CIRCLED_BLOCKCHAIN'] # Capacity of a
48          Blockchain
49  push_to_ethereum = config['BLOCK']['PUSH_TO_ETHEREUM']
50
51  # Instantiate a new object from LogChain
52  LCaaS = LogChain(500747320)
53
54
55  @app.route('/')
```

```
56 def displayStatus():
57     return '<h2>Logchain-as-a-Service (LCaaS)has been succesfully initiated
58         ! Use our RESTful API to interact with it!</h2>'
59
60
61 @app.route('/submit_raw', methods=['POST']) # handles submit_raw method
62 def submit_raw():
63     # print("We received: ", request.get_json())
64     received_data = (request.get_json())
65
66     blockify(LCaaS.block_index.get_current_index(), LCaaS.cb_index.
67         get_current_index(), received_data)
68     # return_string = str(" new record has been successfully received and
69         added to LogChain" + "\ncurrent CB_Index: " + str(LCaaS.cb_index.
70         get_current_index())+ "\ncurrent Block_Index: " + str(LCaaS.
71         block_index.get_current_index()))
72     return LCaaS.return_string, 202
73
74
75 @app.route('/submit_digest', methods=['POST']) # handles submit_digest
76     method
77 def submit_digest():
78     # print("We received: ", request.get_json())
79
80     received_data = request.get_json()
81     print(received_data)
82     passed_digest_value = received_data['digest']
83     print(passed_digest_value)
84     if (len(passed_digest_value) == 64):
85         blockify(LCaaS.block_index.get_current_index(), LCaaS.cb_index.
86             get_current_index(), received_data)
87         return LCaaS.return_string, 202
88     else:
```

```
89          LCaaS.return_string = "Received data is not in correct SHA256 format
90              "
91
92          return LCaaS.return_string, 202
93
94
95  @app.route('/verify_raw', methods=['POST']) # handles verify_raw method
96  def verify_raw():
97      # print("We received: ", request.get_json())
98      received_data = (request.get_json())
99      search_b(received_data)
100     return LCaaS.return_string, 202
101
102
103 @app.route('/verify_digest', methods=['POST']) # handles verify_digest
104     method
105 def verify_digest():
106     received_data = (request.get_json())
107     search_b(received_data)
108     return LCaaS.return_string, 202
109
110
111 @app.route('/verify_tb', methods=['POST']) # handles verify_tb method
112 def verify_tb():
113     received_data = (request.get_json())
114     passed_tb_hash_value = received_data['tb_hash']
115     search_tb(passed_tb_hash_value)
116
117     return LCaaS.return_string, 202
118
119
120 def blockify(current_block_index_value, current_cb_index_value, data): #
121     Helper function
```

```
122
123     if (LCaaS.sbc_index.get_current_index()<1):
124         blockname = "Circled blockchain-" + str(LCaaS.sbc_index.
125             get_current_index())
126     else:
127         blockname = "Circled blockchain-" + str(LCaaS.sbc_index.
128             get_current_index()-1)
129     if ((current_block_index_value == 0) and (
130             current_cb_index_value == 0)): # we need to generate an absolute
131                 genesis block first
132         print("Log: A new CircledBlockchain and a an Absolute Genesis Block
133             (AGB) is needed")
134         LCaaS.create_new_CircledBlockchain(LCaaS.cb_index.get_current_index
135             ())
136         # create a circled blockchain using index of cb
137         absolute_genesis_block = create_new_block(type="AGB")
138         LCaaS.cb_array[LCaaS.cb_index.get_current_index()].add_block_to_CB(
139             absolute_genesis_block) # add absolute genesis block to the
140                 current CB
141
142         # db.child("Circled blockchain-0").push(json.dumps({'Index': LCaaS.
143             block_index.get_current_index(), 'Type': "AGB",
144
145         db.child(blockname).push(json.dumps({'Index': LCaaS.block_index.
146             get_current_index(), 'Type': "AGB",
147                                              'Content': LCaaS.cb_array[LCaaS.
148                                                  cb_index.get_current_index()].
149                                                  chain[
150                                                  LCaaS.internal_block_counter.
151                                                      get_current_index()].
152                                                      stringify_block()}),
153                                 user['idToken']) # push data to Firebase
154
```

```
155         print("Log: The current CB index is : ", LCaaS.cb_index.
156             get_current_index())
157         print("Log: The current block index is : ", LCaaS.block_index.
158             get_current_index())
159         print("Log: The current internal block counter index is : ", LCaaS.
160             internal_block_counter.get_current_index())
161         print(LCaaS.cb_array[LCaaS.cb_index.get_current_index()].chain[
162                 LCaaS.internal_block_counter.get_current_index()].
163                     stringify_block())
164 
165         LCaaS.block_index.increase_index()
166         LCaaS.internal_block_counter.increase_index()
167 
168         previous_block = absolute_genesis_block
169 
170         new_block_data_element = data
171         new_block = create_new_block("DB", previous_block,
172             new_block_data_element)
173         LCaaS.cb_array[LCaaS.cb_index.get_current_index()].add_block_to_CB(
174             new_block) # add the first data block to the current CB
175         # db.child("Circled blockchain-0").push(json.dumps({'Index': LCaaS.
176             block_index.get_current_index(), 'Type': "DB",
177 
178         db.child(blockname).push(json.dumps({'Index': LCaaS.block_index.
179             get_current_index(), 'Type': "DB",
180                                              'Content': LCaaS.cb_array[LCaaS.
181                                                  cb_index.get_current_index()].
182                                                  chain[
183                                                  LCaaS.internal_block_counter.
184                                                      get_current_index()].
185                                                      stringify_block()}),
186                                 user['idToken']) # push data to firebase
187 
```

```
188         LCaaS.return_string = str(
189             "An AGB was created for the new circle block. AGB details are as
190                  follows:\n" + str(
191                 absolute_genesis_block.stringify_block()) + "\nThe new
192                     record has been successfully received and added to
193                     LogChain with following details:\n" + str(
194                 new_block.stringify_block()))
195 
196         print("Log: The current CB index is : ", LCaaS.cb_index.
197             get_current_index())
198         print("Log: The current block index is : ", LCaaS.block_index.
199             get_current_index())
200         print("Log: The current internal block counter index is : ", LCaaS.
201             internal_block_counter.get_current_index())
202         print(LCaaS.cb_array[LCaaS.cb_index.get_current_index()].chain[
203                   LCaaS.internal_block_counter.get_current_index()].
204                       stringify_block())
205         LCaaS.block_index.increase_index()
206         LCaaS.internal_block_counter.increase_index()
207 
208 
209     elif ((current_block_index_value != 0) and (len(LCaaS.cb_array[
210             LCaaS.cb_index.get_current_index()].chain) < (
211             max_number_of_blocks_in_circledblockchain - 2))): # we just need
212                  to generate data block for the current Circled Blockchain
213 
214         print("Log: A new data block is needed in the same CircledBlockchain
215             ")
216         previous_block = LCaaS.cb_array[LCaaS.cb_index.get_current_index()].
217             chain[
218             LCaaS.internal_block_counter.get_current_index() - 1]
219         new_block_data_element = data
```

```
220         new_block = create_new_block("DB", previous_block,
221             new_block_data_element)
222         LCaaS.cb_array[LCaaS.cb_index.get_current_index()].add_block_to_CB(
223             new_block) # add data block to the current CB
224 
225         db.child(blockname).push(json.dumps({'Index': LCaaS.block_index.
226             get_current_index(), 'Type': "DB",
227                                              'Content': LCaaS.cb_array[LCaaS.
228                                                  cb_index.get_current_index()].
229                                                  chain[
230                                                  LCaaS.internal_block_counter.
231                                                      get_current_index()].
232                                                      stringify_block()}),
233 
234                                  user['idToken']) # push data to Firebase
235         LCaaS.return_string = str(
236             "The new record has been successfully received and added to
237                 LogChain with following details:\n" + str(
238                 new_block.stringify_block()))
239 
240         print("Log: The current CB index is : ", LCaaS.cb_index.
241             get_current_index())
242         print("Log: The current block index is : ", LCaaS.block_index.
243             get_current_index())
244         print("Log: The current internal block counter index is : ", LCaaS.
245             internal_block_counter.get_current_index())
246         print(LCaaS.cb_array[LCaaS.cb_index.get_current_index()].chain[
247                 LCaaS.internal_block_counter.get_current_index()].
248                     stringify_block())
249         LCaaS.block_index.increase_index()
250         LCaaS.internal_block_counter.increase_index()
251 
252 
```

```
253     elif ((current_block_index_value != 0) and (len(LCaaS.cb_array[
254                                 LCaaS.cb_index.get_current_index()].chain)
255                                     < (
256 
257                                 max_number_of_blocks_in_circledblockchain -
258                                      1))):
259                                 # we need to generate the last data block
260                                     and a TB
261         print("Log: Last block of this CB needs to be created and added")
262         print("Log: A new Terminal block is needed")
263         # create the last data block for this CB
264 
265         previous_block = LCaaS.cb_array[LCaaS.cb_index.get_current_index()].
266             chain[
267             LCaaS.internal_block_counter.get_current_index() - 1]
268 
269         new_block_data_element = data
270         new_block = create_new_block("DB", previous_block,
271             new_block_data_element)
272         LCaaS.cb_array[LCaaS.cb_index.get_current_index()].add_block_to_CB(
273             new_block) # add the last data block to the current CB
274 
275         db.child(blockname).push(json.dumps({'Index': LCaaS.block_index.
276             get_current_index(), 'Type': "DB",
277                                              'Content': LCaaS.cb_array[LCaaS.
278                                                 cb_index.get_current_index()].
279                                                 chain[
280                                                 LCaaS.internal_block_counter.
281                                                     get_current_index()].
282                                                     stringify_block()}),
283                                 user['idToken']) # push data to firebase
284 
285         LCaaS.block_index.increase_index()
```

```
286         LCaaS.internal_block_counter.increase_index()
287 
288         # create a terminal block as the last block of this CB
289 
290         concatenated_hashes = ""
291         count = 0
292 
293         while (count <= len(LCaaS.cb_array[LCaaS.cb_index.get_current_index
294             ()].chain) - 1):
295             if (count <= len(LCaaS.cb_array[LCaaS.cb_index.get_current_index
296                 ()].chain) - 2):
297                 concatenated_hashes = concatenated_hashes + LCaaS.cb_array[
298                     LCaaS.cb_index.get_current_index()].chain[
299                     count].get_current_hash() + ","
300                 count += 1
301             else:
302                 concatenated_hashes = concatenated_hashes + LCaaS.cb_array[
303                     LCaaS.cb_index.get_current_index()].chain[
304                     count].get_current_hash()
305                 count += 1
306 
307         print("Log: Aggregated_hash for all blocks of this CB is ",
308             concatenated_hashes)
309 
310         timestamp_from = LCaaS.cb_array[LCaaS.cb_index.get_current_index()].
311             chain[
312             0].get_timestamp()
313 
314         timestamp_to = LCaaS.cb_array[LCaaS.cb_index.get_current_index()].
315             chain[
316             -1].get_timestamp()
317 
```

```
318         block_index_from = LCaaS.cb_array[LCaaS.cb_index.get_current_index()
319             ].chain[
320             0].get_index()
321 
322         block_index_to = LCaaS.cb_array[LCaaS.cb_index.get_current_index()].
323             chain[
324             -1].get_index()
325 
326         # Here we make a hash of all hashes in the current CB
327 
328         hash_of_hashes = (hashlib.sha256(str(concatenated_hashes).encode('
329             utf-8'))).hexdigest()
330 
331         previous_block = LCaaS.cb_array[LCaaS.cb_index.get_current_index()].
332             chain[-1]
333         new_TB_data = TB_data(hash_of_hashes, timestamp_from, timestamp_to,
334             block_index_from, block_index_to)
335         new_TerminalBlock = create_new_block("TB", previous_block,
336             new_TB_data)
337 
338         # let's add the TB to the CB
339         print("Log: Terminal block is : ", stringify_terminalblock(
340             new_TerminalBlock))
341 
342         LCaaS.cb_array[LCaaS.cb_index.get_current_index()].add_block_to_CB(
343 
344             new_TerminalBlock) # add terminal block to CB
345 
346         db.child(blockname).push(json.dumps({'Index': LCaaS.block_index.
347             get_current_index(), 'Type': "TB",
348                                              'Content': stringify_terminalblock
349                                                  (new_TerminalBlock)}),
350                                  user[
```

```
351                                 'idToken']) # push terminal block to
352                                     Firebase (it is stringied so it can be
353                                      viewed properly)
354
355         LCaaS.block_index.increase_index()
356         LCaaS.internal_block_counter.increase_index()
357
358         # add terminal block content to the data element of a SB and add the
359              SB to the SBC
360
361         if (len(LCaaS.SBC.superchain) == 0):
362             SBC_gensis = create_new_block("SBC-GB") # create a genesis block
363                  for the SBC
364             LCaaS.SBC.add_block_to_SBC(SBC_gensis) # ad the SBC-GB to SBC
365             SB_GB_submission = ""
366             SB_submission = ""
367
368             db.child("SuperBlocks").push(
369                 json.dumps(
370                     {'Index': LCaaS.SBC.superchain[LCaaS.sbc_index.
371                         get_current_index()].get_index(), 'Type': "SBC-GB",
372                      'Content': LCaaS.SBC.superchain[LCaaS.sbc_index.
373                          get_current_index()].stringify_block()}),
374                 user['idToken']) # push super block to Firebase
375
376             LCaaS.sbc_index.increase_index()
377
378             previous_super_block = SBC_gensis
379             new_super_block_data_element = stringify_terminalblock(
380                 new_TerminalBlock) # adding the entire terminal block as
381                     data element for superblock
382             new_super_block = create_new_block("SB", previous_super_block,
383                 new_super_block_data_element)
```

```
384             LCaaS.SBC.add_block_to_SBC(new_super_block) # add the super
385                 block to the SBC
386 
387             print("Log: a new SB is created: " + str(
388                 LCaaS.SBC.superchain[LCaaS.sbc_index.get_current_index()].
389                     stringify_block()))
390             db.child("SuperBlocks").push(
391                 json.dumps(
392                     {'Index': LCaaS.SBC.superchain[LCaaS.sbc_index.
393                         get_current_index()].get_index(), 'Type': "SB",
394                      'Content': LCaaS.SBC.superchain[LCaaS.sbc_index.
395                          get_current_index()].stringify_block()}),
396                 user['idToken']) # push super block to Firebase
397 
398             ## Code for Ethereum integration ##
399 
400             if (push_to_ethereum == 'Yes'):
401 
402                 LCE = LC_Ethereum
403 
404                 if (LCE.check_whether_address_is_approved(0
405                     x3f4f9bb697f84a26fbc85883f2ff4d31a36ed83c)):
406                     print(
407                         "Log: The client has already paid the membership fee
408                             and is authorized to use LogChain and Ethereum
409                             connection")
410                     SB_GB_submission = "\nThe Gensis Superblock is added to
411                         the Ethereum network " + str(
412                         LCE.submit_a_superblock(str(SBC_gensis.
413                             stringify_block()), 0.1))
414                     print(SB_GB_submission)
415                     SB_submission = "\nThe Superblock is added to the
416                         Ethereum network " + str(
```

```
417                     LCE.submit_a_superblock(str(new_super_block.
418                         stringify_block()), 0.1))
419                 print(SB_submission)
420 
421             else:
422                 LCE.send_ether_to_contract(0.03) ## membership fee
423                 print(
424                     "Log: The membership fee is now paid and the client
425                         is authorized to use LogChain and Ethereum
426                         connection")
427                 SB_GB_submission = "\nThe Gensis Superblock is added to
428                     the Ethereum network " + str(
429                     LCE.submit_a_superblock(str(SBC_gensis.
430                         stringify_block()), 0.1))
431                 print(SB_GB_submission)
432                 SB_submission = "\nThe Superblock is added to the
433                     Ethereum network " + str(
434                     LCE.submit_a_superblock(str(new_super_block.
435                         stringify_block()), 0.1))
436                 print(SB_submission)
437 
438 
439 
440         LCaaS.return_string = str(
441             "The last data block for this CB is generated:\n" + str(
442                 new_block.stringify_block()) + "\nA Terminal Block have
443                     been successfully created and added to LogChain with
444                     following details\n" + str(
445                 stringify_terminalblock(new_TerminalBlock)) + "\n A new
446                     Super block has been created\n" + str(
447                 LCaaS.SBC.superchain[LCaaS.sbc_index.get_current_index()
448                     ].stringify_block()) + str(
449                 SB_GB_submission) + str(SB_submission))
```

```
450
451             LCaaS.sbc_index.increase_index()
452
453
454         else:
455
456             previous_super_block = LCaaS.SBC.superchain[
457                 LCaaS.sbc_index.get_current_index() - 1] # this is the
458                     previous superblock in the superblockchain for this
459                     instance of Logchain
460             new_super_block_data_element = stringify_terminalblock(
461                 new_TerminalBlock) # adding the entire terminal block as
462                     data element for superblock
463
464             new_super_block = create_new_block("SB", previous_super_block,
465                 new_super_block_data_element)
466             LCaaS.SBC.add_block_to_SBC(new_super_block) # add the super
467                 block to the SBC
468
469             ## Code for Ethereum integration ##
470             SB_submission = ""
471
472             if (push_to_ethereum == 'Yes'):
473
474                 LCE = LC_Ethereum
475                 # LTest.send_ether_to_contract(0.03)
476
477                 if (LCE.check_whether_address_is_approved(0
478                     x3f4f9bb697f84a26fbc85883f2ff4d31a36ed83c)):
479                     print(
480                         "Log: The client has already paid the membership fee
481                             and is authorized to use LogChain and Ethereum
482                             connection")
```

```
483                 SB_submission = "\nThe Superblock is added to the
484                     Ethereum network " + str(
485                     LCE.submit_a_superblock(str(new_super_block.
486                         stringify_block()), 0.1))
487                 print(SB_submission)
488 
489             else:
490                 LCE.send_ether_to_contract(0.03) ## membership fee
491                 print(
492                     "Log: The membership fee is now paid and the client
493                         is authorized to use LogChain and Ethereum
494                         connection")
495                 SB_submission = "\nThe Superblock is added to the
496                     Ethereum network " + str(
497                     LCE.submit_a_superblock(str(new_super_block.
498                         stringify_block()), 0.1))
499                 print(SB_submission)
500 
501 
502 
503         print("Log: " + str(LCaaS.SBC.superchain[LCaaS.cb_index.
504             get_current_index()].stringify_block()))
505         db.child("SuperBlocks").push(
506             json.dumps(
507                 {'Index': LCaaS.SBC.superchain[LCaaS.sbc_index.
508                     get_current_index()].get_index(), 'Type': "SB",
509                  'Content': LCaaS.SBC.superchain[LCaaS.sbc_index.
510                      get_current_index()].stringify_block()}),
511             user['idToken']) # push super block to Firebase
512 
513         LCaaS.return_string = str(
514             "The last data block for this CB is generated:\n" + str(
```

```
515                 new_block.stringify_block()) + "\nA Terminal Block have
516                     been successfully created and added to LogChain with
517                     following details\n" + str(
518                 stringify_terminalblock(new_TerminalBlock)) + "\n A new
519                     Super block has been created\n" + str(
520                 LCaaS.SBC.superchain[LCaaS.sbc_index.get_current_index()
521                     ].stringify_block()) + str(SB_submission))
522
523         LCaaS.sbc_index.increase_index()
524
525
526 elif ((current_block_index_value != 0) and (len(LCaaS.cb_array[
527                     LCaaS.cb_index.get_current_index()].chain) ==
528                         max_number_of_blocks_in_circledblockchain)):
529
530     print("Log: A new CircledBlockchain and a Relative Genesis Block (
531         RGB) is needed")
532     print("Log: The previous CB index is : ", LCaaS.cb_index.
533         get_current_index())
534     print("Log: The previous CB length is : ", len(LCaaS.cb_array[LCaaS.
535         cb_index.get_current_index()].chain))
536     print("Log: The current block index is : ", LCaaS.block_index.
537         get_current_index())
538     print("Log: The internal block counter index is : ", LCaaS.
539         internal_block_counter.get_current_index())
540
541     LCaaS.cb_index.increase_index() # increase the index for CB
542     LCaaS.internal_block_counter.reset_current_index() # reset the
543         internal counter to 0 as new CB needs index to be 0
544
545     print("Log: The new CB index is : ", LCaaS.cb_index.
546         get_current_index())
```

```
547         print("Log: The current block index is : ", LCaaS.block_index.
548             get_current_index())
549         print("Log: The current internal block counter is : ",
550               LCaaS.internal_block_counter.get_current_index())
551 
552         LCaaS.create_new_CircledBlockchain(LCaaS.cb_index.get_current_index
553             ())
554 
555         print("Log: The current CB length is : ", len(LCaaS.cb_array[
556                                                     LCaaS.cb_index.
557                                                         get_current_index()
558                                                         ].chain))
559         previous_block = LCaaS.cb_array[LCaaS.cb_index.get_current_index() -
560              1].chain[-1]
561         print("Log: The CB index is : ", LCaaS.cb_index.get_current_index())
562         print("Log: The block index is : ", LCaaS.block_index.
563             get_current_index())
564         relative_genesis_block = create_new_block("RGB", previous_block)
565 
566         LCaaS.cb_array[LCaaS.cb_index.get_current_index()].add_block_to_CB(
567             relative_genesis_block) # add relative genesis block to the
568                 current CB
569         db.child(blockname).push(json.dumps({'Index': LCaaS.block_index.
570             get_current_index(), 'Type': "RGB",
571                                              'Content': LCaaS.cb_array[LCaaS.
572                                                  cb_index.get_current_index()].
573                                                  chain[
574                                                  LCaaS.internal_block_counter.
575                                                      get_current_index()].
576                                                      stringify_block()}),
577                                 user['idToken']) # push data to Firebase
578 
579         print(LCaaS.cb_array[LCaaS.cb_index.get_current_index()].chain[
```

```
580                 LCaaS.internal_block_counter.get_current_index()].
581                     stringify_block())
582
583         print("Now the length of CB is", len(LCaaS.cb_array[
584                                             LCaaS.cb_index.
585                                                 get_current_index()].chain)
586                                                 )
587
588         LCaaS.block_index.increase_index()
589         LCaaS.internal_block_counter.increase_index()
590
591         # LCaaS.cb_array[LCaaS.cb_index].internal_index.increase_index()
592
593         print("Log: The CB index is : ", LCaaS.cb_index.get_current_index())
594         print("Log: The block index is : ", LCaaS.block_index.
595             get_current_index())
596         print("*After*Log: Length of CB ", len(LCaaS.cb_array[LCaaS.cb_index
597             .get_current_index()].chain))
598         previous_block = relative_genesis_block
599         new_block_data_element = data
600         new_block = create_new_block("DB", previous_block,
601             new_block_data_element)
602         LCaaS.cb_array[LCaaS.cb_index.get_current_index()].add_block_to_CB(
603             new_block) # add data block to the current CB
604
605         db.child(blockname).push(json.dumps({'Index': LCaaS.block_index.
606             get_current_index(), 'Type': "DB",
607                                             'Content': LCaaS.cb_array[LCaaS.
608                                                 cb_index.get_current_index()].
609                                                 chain[
610                                                 LCaaS.internal_block_counter.
611                                                     get_current_index()].
612                                                     stringify_block()}),
```

```
613                                 user['idToken']) # push data to Firebase
614
615         LCaaS.return_string = str(
616             "An RGB was created for the new circle block. RGB details are as
617                  follows:\n" + str(
618                 relative_genesis_block.stringify_block()) + "\nThe new
619                    record has been successfully received and added to
620                    LogChain with following details:\n" + str(
621                 new_block.stringify_block()))
622
623         print(LCaaS.cb_array[LCaaS.cb_index.get_current_index()].chain[
624                  LCaaS.internal_block_counter.get_current_index()].
625                     stringify_block())
626
627         LCaaS.block_index.increase_index()
628         LCaaS.internal_block_counter.increase_index()
629
630
631 def search_b(passed_data):
632     cb_counter = 0
633     b_counter = 0
634     search_result = ""
635
636     while (cb_counter < len(LCaaS.cb_array)):
637         while (b_counter < len(LCaaS.cb_array[cb_counter].chain)):
638             if (LCaaS.cb_array[cb_counter].chain[b_counter].get_data() ==
639                 passed_data):
640                 print("An exact match for submitted raw data has been found:
641                     ")
642                 print(LCaaS.cb_array[cb_counter].chain[b_counter].
643                     stringify_block())
644                 search_result += "\nAn exact match for the submitted value
645                     has been found\n" + str(
```

```
646                     LCaaS.cb_array[cb_counter].chain[b_counter].
647                         stringify_block())
648
649                 b_counter += 1
650
651             else:
652                 print("No match was found for the received data!!!\n")
653                 b_counter += 1
654                 continue
655             # search_result = "No match was found for the received data!!!"
656
657         b_counter = 0
658         cb_counter += 1
659
660     if (len(search_result) == 0):
661         LCaaS.return_string = "No match was found for the received data!!!"
662     else:
663         LCaaS.return_string = search_result
664
665
666 def search_tb(passed_data):
667     cb_counter = 0
668     b_counter = 0
669     search_result = ""
670
671     # LCaaS.cb_array[cb_counter].chain[b_counter].get_data().aggr_hash ==
672         passed_data and
673
674     while (cb_counter < len(LCaaS.cb_array)):
675         while (b_counter < len(LCaaS.cb_array[cb_counter].chain)):
676             if (LCaaS.cb_array[cb_counter].chain[b_counter].get_block_type()
677                 == "TB" and
```

```
678                 LCaaS.cb_array[cb_counter].chain[b_counter].get_data().
679                     aggr_hash == passed_data):
680             print("An exact TB for the submitted hash data has been
681                 found:")
682 
683             # print(LCaaS.cb_array[cb_counter].chain[b_counter].
684                 stringify_block())
685             # search_result += "\nA matching Terminal Block for the
686                 submitted hash has been fou" \
687             # "nd\n" + str(
688             # LCaaS.cb_array[cb_counter].chain[b_counter].
689                 stringify_block())
690 
691             search_result += "\nAn exact match for the submitted value
692                 has been found\n" + str(
693                 stringify_terminalblock(LCaaS.cb_array[cb_counter].chain[
694                     b_counter]))
695             b_counter += 1
696         else:
697             print("No match was found for the received data!!!\n")
698             b_counter += 1
699             continue
700             search_result = "No match was found for the received data!!!
701                 "
702 
703     b_counter = 0
704     cb_counter += 1
705 
706 if (len(search_result) == 0):
707     LCaaS.return_string = "No match was found for the received data!!!"
708 else:
709     LCaaS.return_string = search_result
710 
```

```
711
712 # data element of TB is the hash of all CB block current_hashes
713 if __name__ == '__main__':
714     app.run()
715
```

Appendix B

Logchain Module (LC.py)

```
1
2
3   from blockchain import *
4   import datetime as date
5
6
7   class Index:
8       def __init__(self):
9           self.index = 0
10
11      def increase_index(self):
12          self.index += 1
13
14      def get_current_index(self):
15          return self.index
16
17      def reset_current_index(self):
18          self.index = 0
19          return self.index
20
21
22  class LogChain:
```

```
23      block_index = Index() # This will be used as main index counter for
24          blocks in the lifecycle of instances of this class
25      cb_index = Index() # This will be used as main index counter for
26          Circled blockchains in the lifecycle of instances of this class
27      sbc_index = Index() # TBD
28      internal_block_counter = Index() # This will hold the internal counter
29          for the count of existing blocks in a CB
30      cb_array = [] # This array holds the indexes for all circled
31          blockchains in this class
32      return_string = ""
33
34      def __init__(self, cid):
35          """
36
37          :rtype: object
38          """
39          self.customer_id = cid
40          self.SBC = SuperBlockchain(index=0)
41          # SBC_gensis = create_new_block("SBC-GB")
42          # self.SBC.add_block_to_SBC(SBC_gensis)
43
44      def create_new_CircledBlockchain(self, index):
45          self.index = index
46          self.CB = CircledBlockchain(index)
47          self.cb_array.append(self.CB)
48
49
50  class CircledBlockchain:
51
52      def __init__(self, index):
53          self.index = index
54          self.chain = []
55
```

```
56      def add_block_to_CB(self, passed_block):
57          self.chain.append(passed_block)
58
59
60  class SuperBlockchain:
61
62      def __init__(self, index):
63          self.index = index
64          self.superchain = []
65
66      def add_block_to_SBC(self, passed_superblock):
67          self.superchain.append(passed_superblock)
68
69
70  class TB_data():
71
72      def __init__(self, aggr_hash, timestamp_from, timestamp_to, index_from,
73           index_to):
74          self.aggr_hash = aggr_hash
75          self.timestamp_from = timestamp_from
76          self.timestamp_to = timestamp_to
77          self.index_from = index_from
78          self.index_to = index_to
79
80      def get_tb_data_aggr_hash(self):
81          return self.aggr_hash
82
83
84  def stringify_terminalblock(passed_block):
85      terminalblock_string = (
86          passed_block.get_index(), passed_block.get_timestamp().isoformat(),
87          "aggr_hash: " + passed_block.get_data().aggr_hash,
88          "timestamp_from: " +
```

```
89          passed_block.get_data().timestamp_from.isoformat(),
90          "timestamp_to: " + passed_block.get_data().timestamp_to.isoformat(),
91          "index_from: " + str(passed_block.get_data().index_from),
92          "index_to: " + str(passed_block.get_data().index_to),
93          passed_block.get_current_hash(), passed_block.get_previous_hash(),
94              passed_block.get_nonce(),
95          passed_block.get_block_type())
96      return terminalblock_string
97
98  # class TerminalBlock(Block): # Main class for defining Terminal Blocks (TB
99      ) and all their attributes and methods
100 # def __init__(self, index, data, previous_hash, block_type, aggr_hash,
101     timestamp_from, timestamp_to,
102 # block_index_from, block_index_to):
103 # self.nonce = int
104 # self.index = index
105 # self.timestamp = date.datetime.utcnow()
106 # self.data = TB_data()
107 # self.previous_hash = previous_hash
108 # self.current_hash = str
109 # self.aggr_hash = aggr_hash
110 # self.timestamp_from = timestamp_from
111 # self.timestamp_to = timestamp_to
112 # self.block_index_from = block_index_from
113 # self.block_index_to = block_index_to
114 # self.block_type = block_type # Not included in the content for hash
115     generation
116 # self.content = str(self.index).encode('utf-8') + str(self.timestamp).
117     encode('utf-8') + \
118 # str(self.data).encode('utf-8') + str(self.previous_hash).encode('utf-8')
119     + str(
120 # self.aggr_hash).encode('utf-8') + \
```

```
121 # str(self.timestamp_from).encode('utf-8') + str(self.timestamp_to).encode
122     ('utf-8') + \
123 # str(self.block_index_from).encode('utf-8') + str(self.block_index_to).
124     encode('utf-8')
125 #
126 #
127
```

Appendix C

Blockchain Module (blockchain.py)

```
1
2
3  # This file contains all the Business Logic for Block and Implements Block
4      Class
5  import datetime as date
6  import hashlib
7  import json
8
9  # Loading configuration
10 with open('config.json', 'r') as f:
11     config = json.load(f)
12
13 difficulty_target = config['BLOCK']['DIFFICULTY_TARGET'] # Difficulty
14     target for blocks
15 genesis_hash = config['BLOCK']['GENESIS_HASH'] # Genesis block value for
16     blocks
17 max_number_of_data_blocks_in_circledblockchain = config['BLOCKCHAIN'][
18     'MAX_NUMBER_OF_BLOCKS_IN_CIRCLED_BLOCKCHAIN'] # Capacity of a
19         Blockchain
20
21
```

```
22 class Block: # Main class for defining Blocks and all their attributes and
23     methods
24     result = str
25 
26     def __init__(self, index, data, previous_hash, block_type):
27         self.nonce = int
28         self.index = index
29         self.timestamp = date.datetime.utcnow()
30         self.data = data
31         self.previous_hash = previous_hash
32         self.current_hash = str
33         self.block_type = block_type # Not included in the content for hash
34             generation
35         self.content = str(self.index).encode('utf-8') + str(self.timestamp)
36             .encode('utf-8') + \
37                     str(self.data).encode('utf-8') + str(self.
38                         previous_hash).encode('utf-8')
39 
40     # Setters
41     def set_hash(self, hash):
42         self.current_hash = hash
43 
44     def set_nonce(self, nonce):
45         self.nonce = nonce
46 
47     # Getters
48 
49     def get_nonce(self):
50         return self.nonce
51 
52     def get_index(self):
53         return self.index
54 
```

```
55      def get_timestamp(self):
56          return self.timestamp
57
58      def get_previous_hash(self):
59          return self.previous_hash
60
61      def get_current_hash(self):
62          return self.current_hash
63
64      def get_data(self):
65          return self.data
66
67      def get_block_type(self):
68          return self.block_type
69
70      def stringify_block(self):
71          block_string = (
72              self.index, self.timestamp.isoformat(), self.data, self.
73                  current_hash, self.previous_hash, self.nonce,
74              self.block_type)
75          return block_string
76
77      def hasher(self, passed_nonce):
78          self.nonce = passed_nonce
79          Hash_object = hashlib.sha256(self.content + str(self.nonce).encode('
80              utf-8'))
81          return Hash_object.hexdigest()
82
83      def mine(self):
84          potential_nonce = 0
85          result = self.hasher(potential_nonce)
86          # print(result)
87          while (str(result).startswith(difficulty_target) != True):
```

```
88              potential_nonce = potential_nonce + 1
89              result = self.hasher(potential_nonce)
90              # print(result)
91          self.current_hash = result
92          self.nonce = potential_nonce
93
94
95  def create_new_block(type, lastblock=None, passed_data=None):
96      block_type = type
97
98      if block_type == "DB": # creates a data block
99          new_index = lastblock.index + 1
100         new_data = passed_data
101         new_previous_hash = lastblock.current_hash
102         newBlock = Block(new_index, new_data, new_previous_hash, block_type)
103         newBlock.mine()
104         return newBlock
105
106     elif block_type == "AGB": # creates an Absolute Genesis Block (AGB)
107         new_index = 0
108         new_data = str("Absolute Genesis Block")
109         new_previous_hash = genesis_hash
110         newBlock = Block(new_index, new_data, new_previous_hash, block_type)
111         newBlock.mine()
112         return newBlock
113
114     elif block_type == "SBC-GB": # creates a Genesis Block (GB) for a
115         Superblolchain
116         new_index = 0
117         new_data = str("A Genesis Block for Superblockchain")
118         new_previous_hash = genesis_hash
119         newBlock = Block(new_index, new_data, new_previous_hash, block_type)
120         newBlock.mine()
```

```
121         return newBlock
122 
123     elif block_type == "RGB": # creates a Relative Genesis Block (RGB)
124 
125         new_index = lastblock.index + 1
126         new_data = str("Relative Genesis Block")
127         new_previous_hash = lastblock.current_hash
128         newBlock = Block(new_index, new_data, new_previous_hash, block_type)
129         newBlock.mine()
130         return newBlock
131 
132     elif block_type == "TB": # creates a terminal block
133         new_index = lastblock.index + 1
134         new_data = passed_data
135         new_previous_hash = lastblock.current_hash
136         newBlock = Block(new_index, new_data, new_previous_hash, block_type)
137         newBlock.mine()
138         return newBlock
139 
140     elif block_type == "SB": # creates a supoer block
141         new_index = lastblock.index + 1
142         new_data = passed_data
143         new_previous_hash = lastblock.current_hash
144         newBlock = Block(new_index, new_data, new_previous_hash, block_type)
145         newBlock.mine()
146         return newBlock
147 
```

Appendix D

Ethereum Module (ethereum.py)

```
1
2  import time
3
4  from web3 import Web3, HTTPProvider
5  import contract_abi
6
7
8  ##### details that will be used to send a transaction to ethereum test
9      blockchain
10
11 class LC_Ethereum:
12
13     def send_ether_to_contract(amount_in_ether):
14         # The address for the published contract on ethereum test blockchain
15             (Ropsten test network)
16         contract_address = "0x6c6bf111b5d9d9060e53c5d967e0a7389d15634b"
17
18         # sender private key (can be
19         obtained from MetaMask plug-ins and connected
20         wallet_private_key = "4f5ae03e520e54a18ff4d7d50b2e85d705
21         eeb2e2cc154e4318fd9cb65d354cc3"
22         wallet_address = "0x3f4f9bb697f84a26fbc85883f2ff4d31a36ed83c"
```

```
23
24          # the w3 object
25
26          w3 = Web3(HTTPProvider("https://ropsten.infura.io/
27              GoumPwW0PttpedP5fdnG"))
28
29          contract_address = w3.toChecksumAddress(contract_address)
30          wallet_address = w3.toChecksumAddress(wallet_address)
31
32          w3.eth.enable_unaudited_features()
33
34          contract = w3.eth.contract(address=contract_address, abi=
35              contract_abi.abi)
36          amount_in_wei = w3.toWei(amount_in_ether, 'ether');
37
38          nonce = w3.eth.getTransactionCount(wallet_address)
39
40          txn_dict = {
41              'to': contract_address,
42              'value': amount_in_wei,
43              'gas': 2000000,
44              'gasPrice': w3.toWei('40', 'gwei'),
45              'nonce': nonce,
46              'chainId': 3
47          }
48
49          signed_txn = w3.eth.account.signTransaction(txn_dict,
50              wallet_private_key)
51
52          txn_hash = w3.eth.sendRawTransaction(signed_txn.rawTransaction)
53
54          txn_receipt = None
55
```

```
56          count = 0
57          while txn_receipt is None and (count < 30):
58              txn_receipt = w3.eth.getTransactionReceipt(txn_hash)
59              print(txn_receipt)
60
61              time.sleep(10)
62
63          if txn_receipt is None:
64              return {'status': 'failed', 'error': 'timeout'}
65
66          return {'status': 'added', 'txn_receipt': txn_receipt}
67
68      def check_whether_address_is_approved(sender_address):
69          w3 = Web3(HTTPProvider("https://ropsten.infura.io/
70              GoumPwW0PttpedP5fdnG"))
71          contract_address = "0x6c6bf111b5d9d9060e53c5d967e0a7389d15634b"
72          contract_address = w3.toChecksumAddress(contract_address)
73          contract = w3.eth.contract(address=contract_address, abi=
74              contract_abi.abi)
75          wallet_address = w3.toChecksumAddress(sender_address)
76          return contract.functions.isApproved(wallet_address).call()
77
78      def submit_a_superblock(submission, amount_in_ether):
79          contract_address = "0x6c6bf111b5d9d9060e53c5d967e0a7389d15634b"
80
81          # sender private key (can be obtained from MetaMask plug-ins and
82              connected
83          wallet_private_key = "4f5ae03e520e54a18ff4d7d50b2e85d705
84          eeb2e2cc154e4318fd9cb65d354cc3"
85          wallet_address = "0x3f4f9bb697f84a26fbc85883f2ff4d31a36ed83c"
86
87          # the w3 object
88
```

```
89          w3 = Web3(HTTPProvider("https://ropsten.infura.io/
90              GoumPwW0PttpedP5fdnG"))
91
92          contract_address = w3.toChecksumAddress(contract_address)
93          wallet_address = w3.toChecksumAddress(wallet_address)
94
95          w3.eth.enable_unaudited_features()
96
97          contract = w3.eth.contract(address=contract_address, abi=
98              contract_abi.abi)
99          amount_in_wei = w3.toWei(amount_in_ether, 'ether');
100
101         nonce = w3.eth.getTransactionCount(wallet_address)
102
103         txn_dict = contract.functions.sendSuperblock(submission).
104             buildTransaction({
105             'chainId': 3,
106             # 'value': amount_in_wei,
107             'gas': 2000000,
108             'gasPrice': w3.toWei('40', 'gwei'),
109             'nonce': nonce,
110         })
111
112         signed_txn = w3.eth.account.signTransaction(txn_dict, private_key=
113             wallet_private_key)
114
115         result = w3.eth.sendRawTransaction(signed_txn.rawTransaction)
116
117         tx_receipt = w3.eth.getTransactionReceipt(result)
118
119         count = 0
120         while tx_receipt is None and (count < 30):
121             time.sleep(10)
```

```
122
123             tx_receipt = w3.eth.getTransactionReceipt(result)
124
125             print(tx_receipt)
126
127         if tx_receipt is None:
128             return {'status': 'failed', 'error': 'timeout'}
129
130         processed_receipt = contract.events.SuperblockSubmission().
131             processReceipt(tx_receipt)
132
133         print(processed_receipt)
134
135         output = "Address {} Submitted a Superblock to ethereum : {}" \
136             .format(processed_receipt[0].args._sender, processed_receipt[0].
137                 args._superblock)
138         print(output)
139
140         return {'status': 'added', 'processed_receipt': processed_receipt}
141
```

Appendix E

Contract ABI (contract-abi.py)

```
abi = """
 [
       {
              "constant": false,
              "inputs": [
                     {
                            "name": "_superblock",
                            "type": "string"
                     }
              ],
              "name": "sendSuperblock",
              "outputs": [
                     {
                            "name": "success",
                            "type": "bool"
                     }
              ],
              "payable": false,
              "stateMutability": "nonpayable",
              "type": "function"
       },
```

```
	{
		"anonymous": false,
		"inputs": [
			{
				"indexed": false,
				"name": "_sender",
				"type": "address"
			},
			{
				"indexed": false,
				"name": "_superblock",
				"type": "string"
			}
		],
		"name": "SuperblockSubmission",
		"type": "event"
	},
	{
		"payable": true,
		"stateMutability": "payable",
		"type": "fallback"
	},
	{
		"inputs": [],
		"payable": false,
		"stateMutability": "nonpayable",
		"type": "constructor"
	},
	{
		"constant": true,
		"inputs": [],
		"name": "getSuperblock",
		"outputs": [
```

```
                        {
                                "name": "",
                                "type": "string"
                        }
                ],
                "payable": false,
                "stateMutability": "view",
                "type": "function"
        },
        {
                "constant": true,
                "inputs": [
                        {
                                "name": "_sender",
                                "type": "address"
                        }
                ],
                "name": "isApproved",
                "outputs": [
                        {
                                "name": "approved",
                                "type": "bool"
                        }
                ],
                "payable": false,
                "stateMutability": "view",
                "type": "function"
        }
]

 """
```

Appendix F

Superblock Smart Contract (Superblock.sol)

```
1
2  pragma solidity ^0.4.0;
3  contract Superblock {
4  // The 'dict' of addresses that are approved to submit SBs
5      mapping (address => bool) approvedSender;
6      string SB;
7
8      // The event to announce a SB on the blockchain
9      event SuperblockSubmission(address _sender, string _superblock);
10     function Superblock() public {
11     }
12     // The 'payable' and it will be called whenever ether is sent to the
13        contract address.
14     function() public payable{
15         // Contains information about the transaction
16         if (msg.value > 20000000000000000) {
17             //if the value sent greater than 0.02 ether (in Wei)
18             // then add the sender's address to approvedSender list and now
19                the can submit SBs
20             approvedSender[msg.sender] = true;
```

```
        }
    }

    // The function to check whether a specified address is approved to
       post SBs.
    function isApproved(address _sender) public view returns (bool approved
       ) {
        return approvedSender[_sender];
    }

    // Read-only function that returns the current SB
    function getSuperblock() public view returns(string) {
        return SB;
    }
//The function that submit the SB to the blockchain
    function sendSuperblock(string _superblock) public returns (bool
       success) {
        // Check if the sender is verified
        if (approvedSender[msg.sender]) {

            SB = _superblock;
            emit SuperblockSubmission(msg.sender, SB);
            return true;

        } else {
            return false;
        }

    }
}
```

References

[1] Account types, gas, and transactions — ethereum homestead 0.1 documentation. http://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html. (Accessed on 07/24/2018).

[2] Algorand. https://www.algorand.com/. (Accessed on 07/22/2018).

[3] Amazon cloudwatch - application and infrastructure monitoring. https:https://aws.amazon.com/cloudwatch. (Accessed on 07/14/2018).

[4] Amazon dynamodb - non relational db. https://aws.amazon.com/dynamodb/. (Accessed on 07/14/2018).

[5] Apache couchdb. http://couchdb.apache.org/. (Accessed on 07/14/2018).

[6] Apache http server project. https://httpd.apache.org/. (Accessed on 07/14/2018).

[7] arxiv.org e-print archive. https://arxiv.org/. (Accessed on 07/14/2018).

[8] Aws blockchain templates. https://aws.amazon.com/blockchain/templates/. (Accessed on 07/18/2018).

[9] Casper version 1 implementation guide · ethereum/research wiki. https://github.com/ethereum/research/wiki/Casper-Version-1-Implementation-Guide. (Accessed on 07/20/2018).

[10] Consensys – harness the power of ethereum. https://new.consensys.net/. (Accessed on 07/18/2018).

[11] Digital forensics research workshop. https://www.dfrws.org/. (Accessed on 07/14/2018).

[12] Dropbox. https://www.dropbox.com/. (Accessed on 07/18/2018).

[13] Ether and ethereum. https://www.ethereum.org/ether. (Accessed on 07/14/2018).

[14] Ethereum (eth) blockchain explorer. https://etherscan.io/. (Accessed on 07/14/2018).

[15] Ethereum project. https://www.ethereum.org/. (Accessed on 07/14/2018).

[16] Ethereum testnet selection. https://testnet.etherscan.io/. (Accessed on 07/14/2018).

[17] Factom - making the world's systems honest. https://www.factom.com/. (Accessed on 07/14/2018).

[18] Firebase. https://firebase.google.com/. (Accessed on 07/14/2018).

[19] Flask (a python microframework). http://flask.pocoo.org/. (Accessed on 07/14/2018).

[20] Git - distributed version control system. https://git-scm.com/. (Accessed on 07/14/2018).

[21] Home - hyperledger. https://www.hyperledger.org/. (Accessed on 07/18/2018).

[22] Hyperledger. https://www.hyperledger.org/. (Accessed on 07/24/2018).

[23] Infura - scalable blockchain infrastructure. https://infura.io/. (Accessed on 07/14/2018).

[24] Litecoin - open source p2p digital currency. https://litecoin.org/. (Accessed on 07/14/2018).

[25] Metamask. https://metamask.io/. (Accessed on 07/14/2018).

[26] Mongodb for giant ideas — mongodb. https://www.mongodb.com/. (Accessed on 07/17/2018).

[27] Multichain — open source blockchain platform. https://www.multichain.com/. (Accessed on 07/14/2018).

[28] Mysql. https://www.mysql.com/. (Accessed on 07/14/2018).

[29] Netflix canada - watch tv shows online, watch movies online. https://www.netflix.com/ca/. (Accessed on 07/18/2018).

[30] Nist sp 800-145, the nist definition of cloud computing. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. (Accessed on 07/19/2018).

[31] Openssl dgst. https://www.openssl.org/docs/manmaster/man1/dgst.html. (Accessed on 07/14/2018).

[32] Poex.io - the original blockchain notary service. https://poex.io/. (Accessed on 07/14/2018).

[33] Remix — solidity ide. https://remix.ethereum.org. (Accessed on 07/14/2018).

[34] Solidity — contract-oriented language. http://solidity.readthedocs.io/en/v0.4.24/. (Accessed on 07/14/2018).

[35] Stackdriver Monitoring - Hybrid Cloud Monitoring.

[36] State of blockchain 2018 - coindesk. https://www.coindesk.com/research/state-blockchain-2018/?slide=13. (Accessed on 07/24/2018).

[37] Test ether faucet. https://faucet.metamask.io/. (Accessed on 07/14/2018).

[38] Test networks — ethereum. http://ethdocs.org/en/latest/network/test-networks.html#details. (Accessed on 07/14/2018).

[39] Web3.py — python library for interacting with ethereum. https://web3py.readthedocs.io/en/stable/. (Accessed on 07/14/2018).

[40] Rafael Accorsi. Log data as digital evidence: What secure logging protocols have to offer? In *33rd Annual IEEE Int. Computer Software and Applications Conference, COMPSAC'09.*, volume 2, pages 398–403. IEEE, 2009.

[41] Rafael Accorsi. Bbox: A distributed secure log architecture. In *European Public Key Infrastructure Workshop*, pages 109–124. Springer, 2010.

[42] Paul C. van Oorschot Alfred J. Menezes and Scott A. Vanstone. *Handbook of applied cryptography.* CRC press, 1996.

[43] Ali Anwar, Anca Sailer, Andrzej Kochut, and Ali R Butt. Anatomy of cloud monitoring and metering: A case study and open problems. In *Proceedings of the 6th Asia-Pacific Workshop on Systems*, page 6. ACM, 2015.

[44] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. Medrec: Using blockchain for medical data access and permission management. In *Open and Big Data (OBD), International Conference on*, pages 25–30. IEEE, 2016.

[45] Adam Back et al. Hashcash-a denial of service counter-measure. 2002.

[46] P Bharathi and S Rajashree. Secure file access solution for public cloud storage. In *Information Communication and Embedded Systems (ICICES), 2014 International Conference on*, pages 1–5. IEEE, 2014.

[47] A. Bhattacharyya, S. A. J. Jandaghi, S. Sotiriadis, and C. Amza. Semantic aware online detection of resource anomalies on the cloud. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 134–143. IEEE, Dec 2016.

[48] Christian Bird, Peter C Rigby, Earl T Barr, David J Hamilton, Daniel M German, and Prem Devanbu. The promises and perils of mining git. In *6th Int. Working Conf. on Mining Software Repositories, MSR'09.*, pages 1–10. IEEE, 2009.

[49] JD Bruce. The mini-blockchain scheme, 2014. *URL: http://cryptonite. info.*

[50] Michael A Caloyannides. *Computer forensics and privacy.* Artech House Publishers, 2001.

[51] Michael A Caloyannides. *Privacy protection and computer forensics*. Artech House, 2004.

[52] Eoghan Casey. Error, uncertainty, and loss in digital evidence. *International Journal of Digital Evidence*, 1(2):1–45, 2002.

[53] Eoghan Casey. *Digital evidence and computer crime: Forensic science, computers, and the internet*. Academic press, 2011.

[54] Anton Chuvakin, Kevin Schmidt, and Chris Phillips. *Logging and log management: the authoritative guide to understanding the concepts surrounding logging and log management*. Newnes, 2012.

[55] Fred Cohen. A cryptographic checksum for integrity protection. *Computers & Security*, 6(6):505–510, 1987.

[56] Michael Crosby, Pradan Pattanayak, Sanjeev Verma, and Vignesh Kalyanaraman. Blockchain technology: Beyond bitcoin. *Applied Innovation*, 2:6–10, 2016.

[57] Guilherme Da Cunha Rodrigues, Rodrigo N Calheiros, Vinicius Tavares Guimaraes, Glederson Lessa dos Santos, Marcio Barbosa De Carvalho, Lisandro Zambenedetti Granville, Liane Margarida Rockenbach Tarouco, and Rajkumar Buyya. Monitoring of cloud computing environments: concepts, solutions, trends, and future directions. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 378–383. ACM, 2016.

[58] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

[59] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *NSDI*, pages 45–59, 2016.

[60] Hany Farid. Image forgery detection. *IEEE Signal processing magazine*, 26(2):16–25, 2009.

[61] Patrick Gallagher. Secure hash standard (shs). *FIPS PUB*, pages 1–27, 2008.

[62] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[63] Sumit Goyal. Public vs private vs hybrid vs community-cloud computing: A critical review. *International Journal of Computer Network and Information Security*, 6(3):20, 2014.

[64] Stuart Haber and W Scott Stornetta. How to time-stamp a digital document. In *Conference on the Theory and Application of Cryptography*, pages 437–455. Springer, 1990.

[65] Wassim Itani, Ayman Kayssi, and Ali Chehab. Privacy as a service: Privacy-aware data storage and processing in cloud computing architectures. In *Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on*, pages 711–716. IEEE, 2009.

[66] H. Kang, M. Le, and S. Tao. Container and Microservice Driven Design for Cloud Infrastructure DevOps. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 202–211, April 2016.

[67] Michael J Kavis. *Architecting the cloud: design decisions for cloud computing service models (SaaS, PaaS, and IaaS)*. John Wiley & Sons, 2014.

[68] John Kelsey and Bruce Schneier. Minimizing bandwidth for remote access to cryptographically protected audit logs. In *Recent Advances in Intrusion Detection*, pages 9–9, 1999.

[69] Karen Kent and Murugiah Souppaya. Guide to computer security log management. *NIST special publication*, 92, 2006.

[70] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19, 2012.

[71] Ryan KL Ko and Mark A Will. Progger: An efficient, tamper-evident kernel-space logger for cloud data provenance tracking. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 881–889. IEEE, 2014.

[72] Iuon-Chang Lin and Tzu-Chun Liao. A survey of blockchain security issues and challenges. *IJ Network Security*, 19(5):653–659, 2017.

[73] Chang Liu, Chi Yang, Xuyun Zhang, and Jinjun Chen. External integrity verification for outsourced big data in cloud and iot: A big picture. *Future Generation Computer Systems*, 49:58–67, 2015.

[74] Changbin Liu, Yun Mao, Jacobus Van der Merwe, and Mary Fernandez. Cloud resource orchestration: A data-centric approach. In *Proceedings of the biennial Conference on Innovative Data Systems Research (CIDR)*, pages 1–8. Citeseer, 2011.

[75] Michael Mainelli and Mike Smith. Sharing ledgers for sharing economies: an exploration of mutual distributed ledgers (aka blockchain technology). *Journal of Financial Perspectives*, 3(3):38–58, 2015.

[76] Raffael Marty. Cloud application logging for forensics. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 178–184. ACM, 2011.

[77] Raffael Marty. Cloud application logging for forensics. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 178–184, New York, NY, USA, 2011. ACM.

[78] Roman Matzutt, Martin Henze, Jan Henrik Ziegeldorf, Jens Hiller, and Klaus Wehrle. Thwarting unwanted blockchain content insertion. In *Cloud Engineering (IC2E), 2018 IEEE International Conference on*, pages 364–370. IEEE, 2018.

[79] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[80] Andriy Miranskyy, Abdelwahab Hamou-Lhadj, Enzo Cialini, and Alf Larsson. Operational-log analysis for big data systems: Challenges and solutions. *IEEE Software*, 33(2):52–59, Mar 2016.

[81] Audris Mockus. Engineering big data solutions. In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pages 85–99, New York, NY, USA, 2014. ACM.

[82] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[83] Surya Nepal, Shiping Chen, Jinhui Yao, and Danan Thilakanathan. Diaas: Data integrity as a service in the cloud. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 308–315. IEEE, 2011.

[84] T. A. B. Nguyen, M. Siebenhaar, R. Hans, and R. Steinmetz. Role-based templates for cloud monitoring. In *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, pages 242–250, Dec 2014.

[85] Gary Palmer. A road map for digital forensics research-report from the first digital forensics research workshop (dfrws). *Utica, New York*, 2001.

[86] Pankesh Patel, Ajith H Ranabahu, and Amit P Sheth. Service level agreement in cloud computing. 2009.

[87] William Wesley Peterson and Daniel T Brown. Cyclic codes for error detection. *Proceedings of the IRE*, 49(1):228–235, 1961.

[88] Panita Pongpaibool, Aimaschana Niruntasukrat, Chavee Issariyapat, Koonlachat Meesublak, Chumpol Mokarat, and Premrudee Aiumsupucgul. Netham-nano: A robust and scalable service-oriented platform for distributed monitoring. In *Proceedings of the AINTEC 2014 on Asian Internet Engineering Conference*, AINTEC '14, pages 51:51–51:57, New York, NY, USA, 2014. ACM.

[89] Willaim Pourmajidi and Andriy Miranskyy. Williampourmajidi/lcaas. `https://github.com/WilliamPourmajidi/LCaaS`. (Accessed on 07/14/2018).

[90] William Pourmajidi and Andriy Miranskyy. Logchain: Blockchain-assisted log storage. In *Proceedings of IEEE International Conference on Cloud Computing (CLOUD 2018), to appear*, 2018.

[91] William Pourmajidi, John Steinbacher, Tony Erwin, and Andriy Miranskyy. On challenges of cloud monitoring. In *2017 Conf. of the Center for Adv. Studies on Collaborative Research (CASCON)*, pages 259–265. IBM Corp., 2017.

[92] Deepak Puthal, Nisha Malik, Saraju P Mohanty, Elias Kougianos, and Chi Yang. The blockchain as a decentralized security framework. *IEEE Consumer Electronics Magazine*, 7(2):18–21, 2018.

[93] Rohit Ranchal, Bharat Bhargava, Lotfi Ben Othmane, Leszek Lilien, Anya Kim, Myong Kang, and Mark Linderman. Protection of identity information in cloud computing without trusted third party. In *Reliable Distributed Systems, 2010 29th IEEE Symposium on*, pages 368–372. IEEE, 2010.

[94] Indrajit Ray, Kirill Belyaev, Mikhail Strizhov, Dieudonne Mulamba, and Mariappan Rajaram. Secure logging as a service—delegating log management to the cloud. *IEEE systems journal*, 7(2):323–334, 2013.

[95] D Reilly, Chris Wren, and Tom Berry. Cloud computing: Forensic challenges for law enforcement. In *Int. Conference for Internet Technology and Secured Transactions (ICITST)*, pages 1–7. IEEE, 2010.

[96] Mark Reith, Clint Carr, and Gregg Gunsch. An examination of digital forensic models. *International Journal of Digital Evidence*, 1(3):1–12, 2002.

[97] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *International workshop on fast software encryption*, pages 371–388. Springer, 2004.

[98] Abdul Salam, Zafar Gilani, and Salman Ul Haq. *Deploying and Managing a Cloud Infrastructure: Real-World Skills for the CompTIA Cloud+ Certification and Beyond: Exam CV0-001*. John Wiley & Sons, 2015.

[99] Nuno Santos, Krishna P Gummadi, and Rodrigo Rodrigues. Towards trusted cloud computing. *HotCloud*, 9(9):3, 2009.

[100] Masaya Sato and Toshihiro Yamauchi. Vmm-based log-tampering and loss detection scheme. *Journal of Internet Technology*, 13(4):655–666, 2012.

[101] Bruce Schneier and John Kelsey. Cryptographic support for secure logs on untrusted machines. In *USENIX Security Symposium*, volume 98, pages 53–62, 1998.

[102] Siti Rahayu Selamat, Robiah Yusof, and Shahrin Sahib. Mapping process of digital forensic investigation framework. *International Journal of Computer Science and Network Security*, 8(10):163–169, 2008.

[103] Shweta Sharma. A strongly trusted integrity preservance based security framework for critical information storage over cloud platform. *International Journal of Applied Information Systems*, 11(6):3–7, Nov 2016.

[104] Melanie Siebenhaar, Ronny Hans, Ralf Steinmetz, et al. Role-based templates for cloud monitoring. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 242–250. IEEE Computer Society, 2014.

[105] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating system concepts essentials*. John Wiley & Sons, Inc., 2014.

[106] Gopalan Sivathanu, Charles P Wright, and Erez Zadok. Enhancing file system integrity through checksums. Technical report, Citeseer, 2004.

[107] Richard T Snodgrass, Shilong Stanley Yao, and Christian Collberg. Tamper detection in audit logs. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 504–515. VLDB Endowment, 2004.

[108] Nick Szabo. Nick szabo – the idea of smart contracts. `http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/idea.html`. (Accessed on 07/14/2018).

[109] W. T. Tsai, X. Sun, and J. Balasooriya. Service-oriented cloud computing architecture. In *2010 Seventh International Conference on Information Technology: New Generations*, pages 684–689, April 2010.

[110] Sarah Underwood. Blockchain beyond bitcoin. *Commun. ACM*, 59(11):15–17, October 2016.

[111] Jonathan Stuart Ward and Adam Barker. Self managing monitoring for highly elastic large scale cloud deployments. In *Proceedings of the sixth international workshop on Data intensive distributed computing*, pages 3–10. ACM, 2014.

[112] Jonathan Stuart Ward and Adam Barker. Self managing monitoring for highly elastic large scale cloud deployments. In *Proceedings of the Sixth International*

Workshop on Data Intensive Distributed Computing, DIDC '14, pages 3–10, New York, NY, USA, 2014. ACM.

[113] Brent R Waters, Dirk Balfanz, Glenn Durfee, and Diana K Smetters. Building an encrypted and searchable audit log. In *NDSS*, volume 4, pages 5–6, 2004.

[114] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

[115] Collective work of all DFRWS attendees. A road map for digital forensic research. `http://dfrws.org/sites/default/files/session-files/a_road_map_for_digital_forensic_research.pdf`, 8 2001. (Accessed on 06/03/2018).

[116] Xiwei Xu, Ingo Weber, Mark Staples, Liming Zhu, Jan Bosch, Len Bass, Cesare Pautasso, and Paul Rimba. A taxonomy of blockchain-based systems for architecture design. In *Software Architecture (ICSA), 2017 IEEE International Conference on*, pages 243–252. IEEE, 2017.

[117] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. *Draft NISTIR*, 8202, 2018.

[118] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

[119] Zibin Zheng, Shaoan Xie, and Hong-Ning Dai. Blockchain challenges and opportunities: A survey. 2016.

[120] Dimitrios Zissis and Dimitrios Lekkas. Addressing cloud computing security issues. *Future Generation computer systems*, 28(3):583–592, 2012.

[121] Guy Zyskind, Oz Nathan, et al. Decentralizing privacy: Using blockchain to protect personal data. In *Security and Privacy Wkshps (SPW), 2015 IEEE*, pages 180–184. IEEE, 2015.

Acronyms

AGB Absolute Genesis Block. 15, 40–42, 51, 52, 56, 58

API Application Programming Interface. 3, 14–16, 29, 39, 46, 47, 55, 59–66, 83–85

B2B Business to Business. 10

B2C Business to Consumer. 10

BaaS Blockchain as a Service. 68, 71

BC Blockchain. 31

CB Circled Blockchain. 15, 82

COL Cloud Operational Log. 4, 18

CRUD Create, Read, Update, Delete. 24

CSC Cloud Service Consumer. 1–6, 8–10, 12, 13, 15–18, 39, 58–61, 71

CSP Cloud Service Provider. 1–6, 8–10, 12, 13, 16–18, 39, 58–61, 71

DB Data Block. 30, 42

DBMS Database Management System. 24, 27

DFRWS Digital Forensics Research Workshop. 19

DIaaS Data Integrity as a Service. 23

EBaaS Ethereum Blockchain as a Service. 71

EMR Electronic Medical Record. 27

GB Genesis Block. 30

IaaS Infrastructure as a Service. 3, 4, 9, 12

IDE Integrated Development Environment. 71, 76, 77

IMS Integrity Management Service. 23

IoT Internet of Things. 23

IP Internet Protocol. 6

JSON JavaScript Object Notation. 42–45, 47, 48, 50, 56, 57, 85

LCaaS Logchain as a Service. 3, 15, 16, 29, 33, 37–48, 50–52, 55, 56, 59–68, 72–75, 79, 82–86

LMS Log Management System. 22, 25

OO Object Oriented. 46, 47

PaaS Platform as a Service. 3, 4, 9, 12

PoW Proof-of-Work. 24, 25, 27, 28, 31, 33, 35, 75

QoS Quality of Service. 1, 2, 4, 10

RGB Relative Genesis Block. 15, 40–42, 48, 52, 56, 58

SaaS Software as a Service. 3, 4, 9, 12

SB Super Block. 15, 44, 82

SBC Super Blockchain. 15

SDN Software Defined Network. 8

SHA Secure Hash Algorithm. 22, 25, 33, 40–42, 44, 59, 72

SLA Service Level Agreement. 1, 4, 6, 7, 9, 13, 17

TB Terminal Block. 15, 39, 41, 42, 65, 82

TPA Third-Party Auditor. 23

TTP Trusted Third-Party. 2, 23

VM Virtual Machine. 3, 9