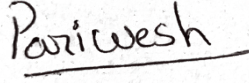




University  
of Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

## MASTER'S THESIS

Study program/specialization: Computer Science	Spring semester, 2020  Open
Author: Pariwesh Subedi	 ..... (author signature)
Instructor: Hein Meling Supervisor(s): Hein Meling	
Title of Master's Thesis:  Deploying Credentials using the Libra Blockchain: Design, Implementation and Evaluation	
ECTS : 30	
Keywords: Blockchain, Credential, Libra, Permissioned	Number of pages: 83 .....  + supplement material/other - Appendix A - Appendix B  Stavanger, June 28, 2020

# Deploying Credentials using the Libra Blockchain: Design, Implementation and Evaluation

*Author:*

Pariwesh Subedi

*Supervisor:*

Hein Meling



Department of Electrical Engineering and Computer Science  
Faculty of Science and Technology

June 28, 2020

# *Abstract*

Credential verification possess a unique challenge involving multi-disciplinary domains throughout the process starting from certification to verification of credentials. Even though technology has transformed many traditional implementations, credential verification is one of such implementation that still widely adopts traditional paper-based approaches. This tend to become time-consuming, administration-controlled and vulnerable to security threats that is evidently present in the system. Standards like W3C's verifiable credential and Mozilla Open Badges has been drafted to approach digitization of credential. These standards can be incorporated using blockchain solutions to address the need of transparency and tamper-evident records in this multi-party credential verification domain. This thesis looks into challenges of implementing a certification system using blockchain technology and proposes a solution for on-chain verification for off-chain digital credentials. The designed solution not only proves the achievement of credentials but also proves a recipient's association with a certifying authority while addressing security challenges present in paper-based approaches. Furthermore, features such as credential revocation and timed credential makes it adaptable into multiple certification scenarios.

# *Acknowledgements*

I would like to thank my supervisors Hein Meling and Rodrigo Saramago for their valuable guidance and constant feedback throughout the course of this thesis.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgements</b>	<b>ii</b>
<b>Abbreviations</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 The BBChain Project . . . . .	2
1.2 Scope . . . . .	3
1.3 Contributions . . . . .	4
1.4 Outline . . . . .	5
<b>2 Background</b>	<b>6</b>
2.1 Blockchain . . . . .	6
2.2 Consensus . . . . .	7
2.2.1 Consensus in permissionless blockchain . . . . .	8
2.2.2 Consensus in permissioned blockchain . . . . .	9
2.3 Libra Blockchain . . . . .	9
2.3.1 Ledger State . . . . .	10
2.3.2 Libra Protocol . . . . .	11
2.3.3 Move Programming Language . . . . .	14
2.4 Related Works . . . . .	18
<b>3 System Design and Extensions</b>	<b>21</b>
3.1 System Overview . . . . .	21
3.1.1 The off-chain layer . . . . .	23
3.1.2 The blockchain layer . . . . .	25
3.2 Extensions . . . . .	29
3.2.1 Permissioned issuers consortium . . . . .	29
<b>4 Implementation with Libra Blockchain</b>	<b>31</b>
4.1 Libra System Components . . . . .	31

---

4.1.1	Modules . . . . .	32
4.1.2	Resources . . . . .	35
4.2	System Features and Algorithms . . . . .	41
4.2.1	Issuer . . . . .	42
4.2.2	Owner . . . . .	45
4.2.3	Holder . . . . .	46
4.2.4	Verifier . . . . .	47
4.3	Limitations . . . . .	48
4.4	Extensions . . . . .	49
4.4.1	Permissioned issuer consortium . . . . .	49
4.4.2	Unique credential digest . . . . .	50
4.5	Use-case Analysis . . . . .	50
4.5.1	Driving Licence Certification . . . . .	51
<b>5</b>	<b>Experimental Evaluation</b>	<b>54</b>
5.1	Experimental Setup . . . . .	54
5.1.1	Libra Network . . . . .	56
5.1.2	Test Client . . . . .	57
5.2	Experimental Results . . . . .	59
5.2.1	Gas usage . . . . .	59
5.2.2	Performance Evaluation . . . . .	61
5.3	Discussion . . . . .	62
<b>6</b>	<b>Conclusion and Future Directions</b>	<b>64</b>
6.1	Conclusion . . . . .	64
6.2	Future Directions . . . . .	66
	<b>List of Figures</b>	<b>66</b>
	<b>List of Tables</b>	<b>68</b>
	<b>A Experimental Data</b>	<b>70</b>
	<b>B Attachments</b>	<b>71</b>
	<b>Bibliography</b>	<b>72</b>

# Abbreviations

<b>BLOB</b>	<b>B</b> inary <b>L</b> arge <b>O</b> bject
<b>BFT</b>	<b>B</b> yzantine <b>F</b> ault <b>T</b> olerance
<b>DID</b>	<b>D</b> istributed <b>I</b> Dentifier
<b>DoS</b>	<b>D</b> enial of <b>S</b> ervice
<b>IPFS</b>	<b>I</b> nter <b>P</b> lanetary <b>F</b> ile <b>S</b> ystem
<b>LAN</b>	<b>L</b> ocal <b>A</b> rea <b>N</b> etwork
<b>MVIR</b>	<b>M</b> ove <b>I</b> ntermediate <b>C</b> ode
<b>OBI</b>	<b>O</b> pen <b>B</b> adge <b>I</b> nterface
<b>PBFT</b>	<b>P</b> ractical <b>B</b> yzantine <b>F</b> ault <b>T</b> olerance
<b>PoS</b>	<b>P</b> roof of <b>S</b> take
<b>PoW</b>	<b>P</b> roof of <b>W</b> ork
<b>QC</b>	<b>Q</b> orum <b>C</b> ertificate
<b>RCN</b>	<b>R</b> esearch <b>C</b> ouncil of <b>N</b> orway
<b>VM</b>	<b>V</b> irtual <b>M</b> achine
<b>W3C</b>	<b>W</b> orld <b>W</b> ide <b>W</b> eb <b>C</b> onsortium
<b>XFT</b>	<b>C</b> ross- <b>F</b> ault <b>T</b> olerance

# Chapter 1

## Introduction

Certification is the process of attesting achievement or status using official documentation. Certification awards a holder with a credential, also referred to as a certificate, that can be presented as a proof of successful completion of the certification process [1]. Similarly, verification is the process of verifying that the presented credential is valid or not. This process of certification and verification is widely adopted in different sectors in the form of educational degrees, driving licence or any government-issued certificate [2].

In many scenarios, holders are presented with paper-based credentials that are easy for handling from the holder's perspective but should also be taken proper care when storing it for a long period of time. For verification, this type of credentials are generally equipped with security features that make it possible for a verifier to check their presence to distinguish a valid credential from an invalid one. While these securities features helps to verify a valid credential, there are fraudulent businesses counterfeiting these documents [3] for monetary gain.

Difficulty follows when presenting paper-based credentials for verification as it requires additional validations to prove its authenticity. In general practices, for a strong document verification, holder starts by verifying copies of the original credential with a valid authority who sign and seal these documents. This sealed copy is then delivered to a verifier who should be aware of the standards and practices necessary to verify the authenticity of the presented document. Any damages to the seal during this process renders the copies invalid. This results in a lengthy process and involves multiple parties to present and verify a certificate on top of an already cumbersome certification process [1].



Certifying authorities have also opted for digital certifications to ease this process. For this, a certificate is signed by the issuer and is considered valid as long as the signature in the digital certificate is verifiable. This form of certification makes it easier to issue and validate a certificate over the traditional approach. But, this approach also requires the issuer to add and maintain resources while a digitally signed certificate requires continuous resigning to keep it valid for a prolonged period of time. Moreover, this approach centralizes the process creating a single point of failure in the system.

While there is no easy way for a third-party verifier to verify the authenticity of certifications [4, 5], it also poses difficulties to the certifying authorities and holders in this process of issuing and presenting a certificate. Moreover, traditional paper-based credentials summarizes different information of issuer and the holder which in many cases reveals more information than what is actually required [3]. This defines a need for a solution that respects users' privacy and eases the process of for all involved parties.

We explore a blockchain-based alternative for the certification-verification problem. A blockchain-based distributed ledger provides transparency, and tamper-evident records in the system which elevates trust in the system's certification procedure and reliability between all involved parties in this process [6].

This thesis aims to answer the following research questions:

- Can Libra blockchain provide decentralized, transparent certification and verification of digital credentials?
- Can selective disclosure be adopted to present the least information during certificate verification?
- Can document counterfeiting be discouraged and detected through blockchain application?

## 1.1 The BBChain Project

BBchain project is a research project funded by the Research Council of Norway(RCN). The goal of this project is to design a distributed systems solution that prevents loss and fraud of diplomas and potentially any other kind of credentials,

while increasing the transparency and trust on the institution's procedures [7]. The BBChain project aims are defined as the following:

- Efficiency: Limit data circulation in the blockchain system to improve efficiency.
- Security: Provide security guarantee with selective data disclosure.
- Trustworthiness: Build user credibility by linking a user's biometric data with blockchain.

Initial efforts to achieve these goals is being researched through smart contracts [8] in Ethereum based solution [7]. Smart Contracts in Ethereum and Modules in Libra blockchain are similar in several aspects but a major difference between them lies in how data is defined and stored in these platforms. This brings an opportunity to explore a different system design and evaluate the pros and cons of the different implementation.

## 1.2 Scope

While the thesis focuses on solutions to the document verification problems, it assumes that the actual credentials are stored off-chain and the blockchain works only as an issuing and verifying platform. Also, other correlated problems such as, associating blockchain address to a person or distributing and storing Libra addresses of the trusted issuers are relevant but not within the scope of this thesis.

In regards to the *BBchain* project, the work done in this thesis revolves around the efficiency and security aspect. While adding trust through biometrics is important for the *BBChain* project, the scope for this thesis work doesn't explore this dimension. We focus on exploring alternatives of the Ethereum solution, by designing a blockchain application on Libra blockchain, devising testing platform and evaluating its features and performance metrics.

This thesis scrutinizes certification in an educational setting and proposes a blockchain-based solution that can also be employed in other scenarios too. We use Libra blockchain and *Move* programming language to develop this solution.

## 1.3 Contributions

Following are some of the contributions made during the period of the thesis work.

- Define credential verification interface for BBchain project so that issuing and verifying clients can connect to the blockchain without being concerned about the underlying architecture/platform of the blockchain technology.
- Define credential account as a possible solution to store issued credential with Holder.
- Define Libra resources to prove the holder's enrolment with issuer. Moreover, earmarking approach of credential registration allows receivers of the credential to prove their association with the issuing authority.
- Propose an extension for permissioned issuer registration.

Besides these primary contributions to the BBchain project, we have made contributions to the Libra open source projects and developed some tools to simplify development, test and host on a custom Libra network through the following:

- Addition of test shell scripts to open source Libra blockchain developer tools repository [9] that helps easier development and testing of modules and scripts developed in Move programming language.
- A bug in the core Libra project was identified and reported to the development team [10].
- Different components, such as the validator, faucet service were published as docker containers with the capability to publish custom modules and scripts that might help developers to test publishing modules outside the Libra network.
- Kubernetes[11] configuration scripts to setup Libra network was developed and published [12].

## 1.4 Outline

Next chapters in the thesis are outlined as follows:

**Chapter 2** introduces background about blockchain and presents a comparative analysis of related works.

**Chapter 3** highlights system design and presents extensions to it for permissioned inclusion of authoritative nodes as issuers.

**Chapter 4** presents the system architecture and implementation details specific to Libra blockchain.

**Chapter 5** presents experimental setup, evaluation procedures and their results.

**Chapter 6** concludes and presents a suggestion for future work.

# Chapter 2

## Background

This chapter presents the background related to this thesis work. We start by presenting the core concept relating to blockchain, specifically, Libra blockchain that will help to understand the following chapters. Further, we discuss related works done in the field of document verification using blockchain discussing different features offered by these solutions.

### 2.1 Blockchain

A blockchain is an ever-growing digitized and decentralized data registry. Blockchain provides an efficient way of storing data in a transparent and tamper-evident ledger, making sure that everyone has access to the entire history of data at any point in time. A blockchain definition needs an introduction to hash function, hash-chain and Merkle tree data structures first.

**Hash Function** A cryptographic hash function  $H$ ,  $H(x) \rightarrow y$  is a deterministic function that maps input ( $x$ ) of arbitrary size to a output of defined length [13].

**Hash-chain** A hash chain is a data structure where data is stored in the form of blocks. These blocks are stored as a linked list, where pointers to previous blocks are the cryptographic hash of those blocks [14]. For every block  $b$  in the hash-chain there exists  $b.prev$  that references the preceding block, a field  $b.h_{-1}$  that refers to

---

hash of the previous block and  $b.data$  that refers the data in the block. Then, hash of block  $b$  can be computed as  $H(b.h_{-1}||b.data)$ .

**Merkle Tree** Merkle tree provides a method to hash multiple data values such that they can be combined into a single representation[13]. It is a binary tree where every node in the tree represents a hash value  $h$  such that, for a leaf node,  $h$  represents the hash of data stored in the node,  $h = H(D)$  and, for inner nodes,  $h$  is the hash of concatenated hash of its children,  $h = H(leftchild||rightchild)$ .

Formally, blockchain can be defined as a hash-chain where the data entered in each block is the root of a merkle tree [13]. We can broadly divide blockchain technology into two different types, permissioned and permissionless [15].

A permissionless blockchain is a type of blockchain system that is open and decentralized. Since memberships are not managed, any node in this blockchain network can participate or disassociate at any point in time [6]. The popular Bitcoin [13] and Ethereum [8] are example of such blockchain system. A permissioned blockchain, on the other hand, are controlled by a single entity or federation. Memberships in the network are managed by the central authority and this governs the participation of nodes in the network. Hyperledger [16] and Libra [17] are examples of such permissioned blockchain.

## 2.2 Consensus

Consensus, grammatically, refers to a general agreement. In distributed systems, the term consensus applies when two or multiple processes in a distributed network need to reach an agreement on a value that is needed for computation of a common task [18]. Blockchain systems usually run a consensus algorithm to agree on the data to be stored in the system. A consensus algorithm should follow the following properties:

- Safety: a consensus algorithm is safe if all involved nodes produces the same valid outputs [19].
- Fault Tolerance: a consensus algorithm is fault-tolerant if it can recover even if nodes involved in consensus fails [19].

- Liveness: a consensus algorithm is live if all non-faulty nodes involved in consensus eventually produce a value [19].

In the following section, we briefly introduce consensus algorithms in different blockchain architecture.

### 2.2.1 Consensus in permissionless blockchain

Choices of consensus algorithm in permissionless blockchain defines usability and adaptability of the system. Fork in blockchain, consensus failure, poor performance and dominance of nodes are some risks that a consensus protocol must address [19]. Following we present two widely adopted approaches for permissionless consensus: Proof of Work(PoW) and Proof of Stake(PoS).

**Proof of Work** PoW restrict the ability of a node to update the blockchain ledger by requiring nodes to compete on solve a cryptographic puzzle. As an incentive of solving the problem, the node that solves it receives a reward in the form of a native cryptocurrency. Some prominent blockchain system like Bitcoin and Ethereum employs PoW as their consensus algorithm.

An example of PoW consensus algorithm can be defined as, for an integer  $d$ , the PoW function with difficulty  $d$  takes data item and returns a hash value( $h_{PoW}$ ) and a nonce value of random bits. In this case, a PoW solution is valid if 1) the hash of data item combined with the nonce value equals the  $h_{PoW}$ , and 2) the first  $d$  bits of  $h_{PoW}$  are 0. This form of consensus is identified to be costly in terms of energy usage [20] thus a shift towards PoS is seen [21].

**Proof of Stake** The Proof of Stake approach solves the problem of high energy cost in PoW. This consensus approach selects random stake-holders in the system to append block into the blockchain. One of such implementations is the Follow-The-Satoshi algorithm, where a random native currency, is selected at random and the owner of that currency can append to the blockchain and thus receive a block reward for it. Generally, if users misbehave when adding a block in the system, their stake of coins deposited are slashed and users are punished [22].

---

## 2.2.2 Consensus in permissioned blockchain

A permissioned blockchain platform allows only known participants to participate in the consensus to append blocks in the blockchain. Nodes in the network are usually semi-trusted and are comparatively fewer than in permissionless architecture and hence allows a higher number of transactions in a unit time than compared with the permissionless architecture. A permissioned block can run different alternatives of consensus mechanisms. Algorithms such as Paxos, RAFT and different Byzantine Fault Tolerant(BFT) algorithms are known to solve the consensus problem in such architecture [14, 15].

One of the popular permissioned blockchain is the Hyperledger Fabric [16]. Transactions in this blockchain system are almost immediate as it trusts the nodes in the system and holds identifiers for each node. It supports different consensus algorithms such as Practical Byzantine Fault Tolerance(PBFT), SIEVE, Cross-Fault Tolerance(XFT), etc to run consensus in the network. It is best suited for a limited number of nodes in the system and cannot be scaled up efficiently if required [19].

**Byzantine Fault Tolerant Protocol - Hotstuff** Byzantine Fault Tolerant refers to the ability of a system to manoeuvre through unpredictable failures of its components. Hotstuff is one of such leader-based BFT protocol [23]. It is a partially synchronous BFT protocols allowing a leading node to run consensus at the speed of actual network delay, and, with a communication complexity that is linear to the number of nodes running the protocol. The Libra blockchain that we discuss in this thesis work uses a variant of the Hotstuff [23] protocol to run consensus amongst the nodes in its network.

## 2.3 Libra Blockchain

Libra blockchain is a permissioned blockchain system that uses the Libra protocol to govern the workflow of different components in the system while Move programming language defines the core application workflow of the blockchain. All the permissioned node, known as validators, jointly maintain a single-versioned distributed database that stores resources created by the blockchain applications.



Libra blockchain aims to develop as a permissionless system over time, but, the initial governance that is needed in the system is provided through the member of Libra association acting as validators in the system. This makes it a permissioned system in this initial stage. It has interfaces available to add a new validator or remove and anyone in the network can propose themselves for the position of a validator but they are only given a validator privilege if the majority of validators agree to add them into the validators consortium. The transition into a permissionless architecture is planned to use PoS based consensus protocol giving validators with voting rights that are equivalent to their possession of Libra coins [24].

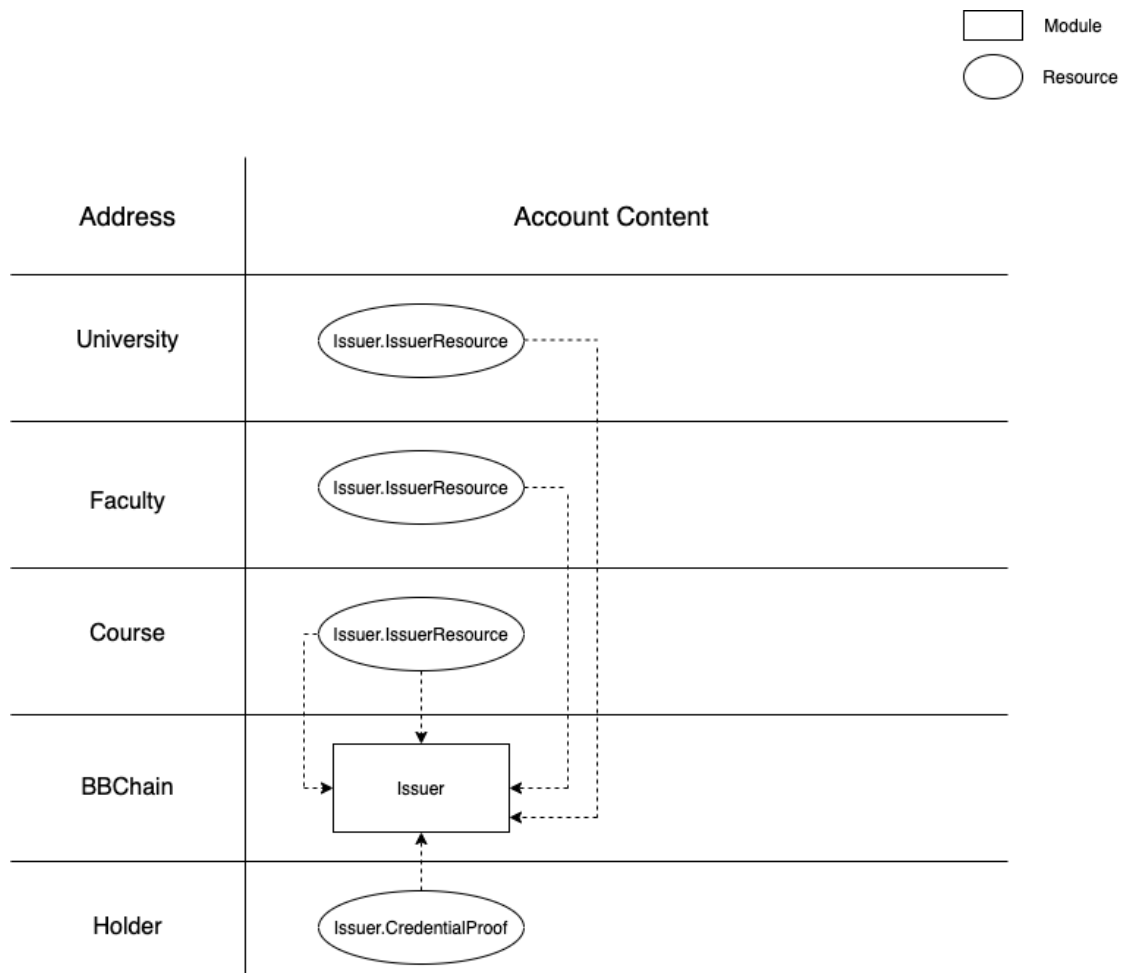
In its core, consensus in Libra in the initial stage is carried out using a Byzantine Fault Tolerant(BFT) protocol, referred to as the Libra Byzantine Fault Tolerance(LibraBFT). The consensus agrees on a state of the distributed database. Similarly, applications in the blockchain is defined using Move which allows users to define custom programming logic as modules and data created in the database through the modules as resources. Resources are also referred to as programmable resources as it adheres to rules defined by developers in the modules that create them. The distributed database holds modules and programmable resources within Libra accounts and are authenticated by public-key cryptography [17, 25].

Accounts in Libra has a private key, used to sign a transaction and a public verification key. Transactions submitted by libra accounts are recorded in an ever-growing Merkle tree where every leaf node in this tree represents a transaction. All transactions are secured through digital signature, which can be one of Ed25519 or MultiEd25519 signatures in the Libra ecosystem. Each digital signature scheme is associated with a unique identifier [17]. The hash value of public key combined with this signature scheme identifier generates the base for Libra address.

### **2.3.1 Ledger State**

The state of a blockchain at a particular database version is the ledger state. The ledger stores data embedded as key-value pairs where a Libra account address is the key and it is linked to values that represent the modules and resources present in the account. Validators should always be aware of the latest version of the ledger in order to execute a submitted transaction. Transactions submitted by clients are executed against this ledger state updating the database to a newer version.

An example of ledger state can be seen in figure 2.1. Here, *BBChain* represents a Libra address that published the *Issuer* module. Other addresses, *University*, *Faculty*, *Course* and *Holder* don't have any modules published but store resources created from the *Issuer* module. The dotted arrow line represents logical link between module and the resources created from the module.



**Figure 2.1:** Example of ledger state defining how modules and resources are embedded as values of the Libra addresses key.

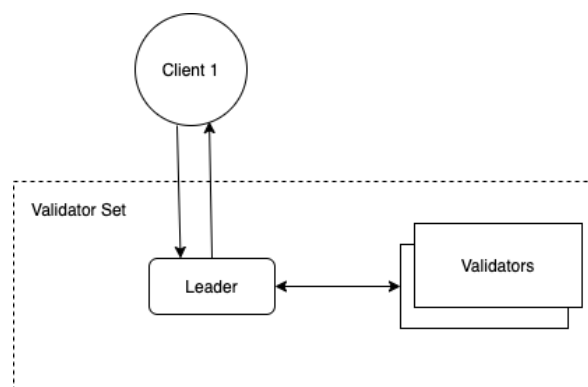
### 2.3.2 Libra Protocol

Libra protocol defines interaction and communication between validators and clients of the blockchain system. Validators behaviour is governed by procedures that are defined in the module that creates them. Modules published by a particular address are unique and are stored in the database under the account which publishes them. Similarly, resources are also unique based on the module that creates them. Resources are also stored within a Libra account address that holds them but

are separately stored than the modules. This differs from the popular Ethereum architecture, where the procedures and the data created are stored together [8].

## Validators

Validators are the nodes in the Libra blockchain that are defined as the part of the validator set. Nodes in this validator set are responsible to carry out consensus using the LibraBFT protocol and agree upon the transition of blockchain state from  $\delta$  to  $\delta_1$  based on the new transaction  $T_1$ . Any transactions that are agreed upon by the validators are stored into the ledger and are immutable. Hence, a validator node does not always need to hold a copy of the entire ledger but only the most recent ledger state to move the blockchain from a current state to a new state [17].



**Figure 2.2:** Workflow in Libra protocol

Validators are managed through a Move module and are equipped with an ability to vote to add a new validator into the validators set or to remove existing validator from the validator set. Even though this will not be possible in the initial stages when blockchain is released, it is planned to be rolled out in phases [24].

**Libra Byzantine Fault Tolerance Protocol** LibraBFT protocol is an adaptation of the Hotstuff protocol [23] with a goal to support 100 validators in the initial stage and 500-1000 in a longer period [24]. Some changes done over *Hotstuff* protocol are:

- Validators sign on the block rather than a sequence of transactions. This makes the protocol resistant to non-deterministic bugs and allows the client to authenticate Quorum Certificate (QC) while reading from the database.
- Removes the need of synchronized clocks among validators by adding a pacemaker that notifies validators of timeouts and a quorum of timeouts is expected before moving to the next round.
- Leader election is handled by a non-deterministic function that selects the next leader at random. This decreases the time which mitigates risks from a DoS attack on the leader.
- Employs aggregation of signature on a Quorum Certificate. This preserves the identity of validators signing it.

Validators receive transactions and share it with other validators in the system using the shared mempool protocol. In every round of the LibraBFT protocol, a validator node is assigned the role of a leader. This leader node is responsible to propose transactions as block. Any validator that receives this proposal checks its voting rules and determines if it should vote for the certification of the block or not. If it accepts to vote on the block then it executes the transaction separate from the blockchain system resulting in an authenticator that is sent to the leader along with the signed block. The leader collects these votes and broadcasts a QC once more than  $2f + 1$  validator votes are collected, where  $f$  represents the number of failed nodes. The validators vote on the authenticator as part of the consensus protocol. When a transaction is committed at a version  $i$ , the consensus emits a signature on the entire state of the database at that particular version. Then, a block is considered as a committed block once two other blocks are committed over it [17].

## **Clients**

Clients in the Libra blockchain are the nodes that submit transactions in the system. The transactions can be submitted to any validator and it will be processed by the leader node from the validator set. Read queries submitted to the validators receives response along with a signed authenticator for the latest version of the database known to the validator. Clients can also hold the copy of the entire database and

send responses to other clients if they query for the data too. The data returned from a validator or clients can always be verified with the authenticator from the response [25].

### 2.3.3 Move Programming Language

Move is a programming language developed for interaction in the Libra blockchain. It defines the three most basic components of the blockchain [17], which are

- Modules
- Transaction scripts
- Configurations and extension

Modules allow users to define custom code blocks and data types. Transaction script invoke these code blocks through a compiled move byte code. Similarly, configuration and extensions in the Libra blockchain defines the working of different core Libra modules that defines Libra accounts, validator governance and blockchain configuration and is only accessible to a predefined association's address [26].

Modules and transaction scripts can be better understood with an example of a transaction in the Libra network. Libra coin is an example of a Libra resource that is defined using *LibraAccount* module. So, if a Libra account wants to transfer Libra coins from their account to another account then this is triggered through a transaction script. This particular transaction script engulfs Move bytecode that invokes the transfer procedure in the *LibraAccount* module. This triggers the code block defined in the *LibraAccount* module resulting in the transfer of Libra Coin from one account to another. The simultaneously emits two transaction events, one when the balance is deducted from the sending account and another when the balance is added into the receiving account.

#### Modules

Move modules are responsible to define resources and procedures for changing or deleting the resource. A module declares resources and procedures to create, update or delete these resources [26]. A constraint that applies to modules published in

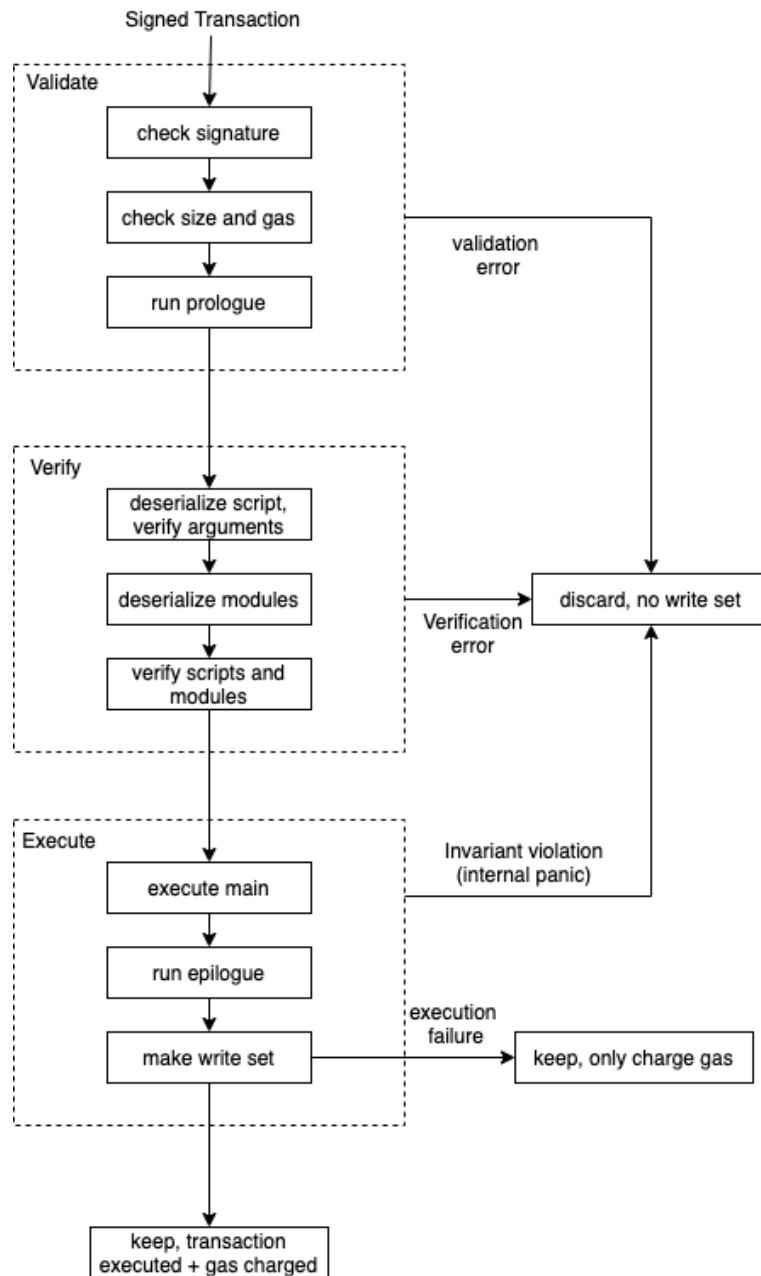
a Libra account is that a module should be uniquely named within an account. Considering figure 2.1 as an example, *BBChain* address cannot publish another module named *Issuer* but the *Course* can publish a module named as *Issuer*. Hence, it is important for the clients to know the address of the module publisher before they can interact with any of their published modules.

**Resource** A resource in Libra protocol is data owned by an account which has been authenticated using public-key cryptography. The type of each resource is declared by the module that defines it and it can be referred with the combination of the module type, address, and name of the resource [17]. Referring to figure 2.1, the type of resource with the *University* address is `BBchain.Issuer.IssuerResource`. To retrieve the same resource a client would request `University/resources/B-Bchain.Issuer.IssuerResource`. This architecture allows each account to store a maximum of one resource of a given type. Since a resource can be an arbitrarily complex data type, it can hold simple data types such as integer and string and other complex data types such as other resources and structs too allowing a single resource type to store a magnitude of data.

Any associated rules that modify, publish, or delete the resource are declared in the module. The safety and verification rules of Move prevents any other module to modify a resource that is not defined by it. Also, modification of resources can be restricted through procedure definition in a module.

### **Transaction Script**

A transaction script is also written in Move programming language. A compiled transaction script generates move bytecode which is sent along with arguments as a part of a transaction. A successful transaction then produces a writeset as its output and it is written to the ledger by the leader node only after a consensus agreement over it. An output of a transaction also results in an execution status code that defines if the transaction ran successfully or not, gas usage, which defines units of gas used in processing the transaction and an event list that holds all the events emitted during the transaction. A transaction sender signs the submitted transaction which undergoes validation and verification in a validator's Move Virtual Machine before it is published among other validators [27].



**Figure 2.3:** Processing transaction in Libra

**Transaction Fees** The Libra protocol charges transaction fee (defined by gas usage in a transaction) to manage the use of resources in the system. The fees are deducted after a transaction completes but a check on if an account can execute a transaction based on the owned Libra coins is done prior to executing a transaction. If the account runs out of coins during the execution, the transaction reverts but the gas used is not reverted [17, 27].

**Events** Events are outputs that are produced when running a transaction. They provide an effective measure to verify the successful execution of a transaction. In a system where transaction failure is a possibility, the event gives an overview of what happens with the transaction when processing it. All the emitted events are associated with a unique key that identifies the structure through where the event was emitted and the payload [27]. Once a transaction completes successfully, all the emitted events are stored in the ledger history which can prove successful execution of the transaction at any point in time after the execution.

### **Move Virtual Machine**

A transaction submitted to a validator first interacts with the Move Virtual Machine (Move VM). The Move VM validates the correctness of the submitted bytecode before it is processed by the receiving validator or broadcasted to other validators [28].

A module or script written in Move Intermediate Representation (MVIR) or Move programming language is compiled to Move bytecode first, This converts structured control flow to unstructured and complex expressions into smaller bytecode that manipulates an operand stack. This bytecode is then submitted to one of the validator node that validates this transaction and broadcasts it to other nodes so that the transaction is executed in the next epoch of the consensus algorithm [17].

Every validator node is equipped with Move VM. The Move VM implements transaction in a safe and isolated environment away from the network first before broadcasting them. The virtual machine implements a bytecode verifier and an interpreter for the move bytecode. As presented in figure 2.3, the submitted transaction is first checked for a valid signature and a minimum amount for transaction execution is checked in the sender's account. After this validation, the submitted bytecode is verified and executed locally. Any error during the validation and verification process generates an error and discards the transaction while error in execution is recorded in the ledger and fees for executed statements are charged [17, 27, 28]. If no errors are encountered, the transaction is executed and a fee is charged for the execution of transaction.

The Move VM validates data structure in the submitted bytecode to be one of boolean, unsigned 64-bit integers, 256-bit addresses, unsigned 8-bit integers, structs



resources, or references. During the development of modules and scripts, the Virtual Machine also provides an interface for developers to run end-to-end tests on the source code [27].

## 2.4 Related Works

Majority of digitally signed documents are stored for several years. Validity of these signed documents is defined by the validity of the signer's public key. Another way of authorizing validity to a document is by time stamping the document when it is signed. This timestamp defines the period of validity for the document. There are many online services that offer to sign digital documents as a service, but they too re-sign documents periodically to retain its validity [29] for a longer period of time.

As blockchain is employed to verify a document's authenticity it is equally important to discuss how a credential is stored. There are generally two approaches to storing data in a blockchain. First is to store data embedded in the chain, and the second is to store the data outside the blockchain. The second approach, referred to as off-chain storage [29, 30] only stores a hash that represents the content stored outside the chain. Thomas Hepp [30] presents off-chain storage as a viable means of storing data because it can be referred to with a smaller data size and it removes the overhead computation for variable block sizes. This benefits a blockchain application by decreasing the size of each transaction which in turn decreases the cost per transaction and fits comparatively many transactions in the same block.

Using the first approach, the content of the blockchain is deemed valid based on the hash of the most recent block published on a public, third-party publication service [31]. After a few years, systems like Bitcoin allowed a different trustable approach of assuring signature timestamp on a signed document based on block mining time. Clark and Essex suggested using timestamps as a part of Proof of Work(PoW) in "CommitCoin" [32] which can later work as a proof for when a commitment was made. Similarly, Stavrou and Voas discuss combining trust and accuracy from a timestamping service with a blockchain system [33]. OpenTimestamps is one of such scalable project that allow timestamping through bitcoin.

In 2012, Peter Todd presented OpenTimestamp [34] that defines rules to create publicly verifiable timestamps that can be proved to have been created before the

Solution	Functionalities				
	Verification	Revocation	Selective Disclosure	Counterfeit protection	Adaptability
Blockcerts [36]	■	□	-	□	△
Hypercerts [37]	■	□	-	□	△
Cerberus [3]	■	■	■	■	△
UNIC [41]	□		-	□	△
UZHBC [42]	□		-	□	△

■ = *core feature*    - = no support    ▲ = *supports multiple certification platforms*  
□ = *partial support*    △ = *only supports education certification*

**Table 2.1:** Comparison of functionalities and supported certifications of different certification-verification solutions

time when it is verified. An extension to this protocol was presented by Brandoli with a solution for scalability problems that allows aggregation of documents in a single transaction using Merkle tree architecture [35].

Digital Certificates Project developed *Blockcerts* [1, 36] for blockchain-based certification verification and issuance system using Bitcoin blockchain. Blockcerts provided verification and revocation functionality but a limitation with this solution was that the revocation of certifications were centralized which posed a risk to the system. *Hypercerts* was proposed by Joao Santos as an extension for Blockcerts to handle the revocation mechanism using Ethereum blockchain [37] and InterPlanetary File System(IPFS) [38] to store revocation proofs. *Cerebrus* was then proposed in 2019 as a solution with on-chain revocation, off-chain storage and selective disclosure [3] for verification.

An education records verification solution was proposed by Hau and Li [39] in Ethereum using an external database to maintain off-chain educational records and aggregating a SHA256 hashed value of education record in a Merkle tree structure. In 2017, SAP developed *TrueRec* digital wallet leveraging the storage of academic certifications in Ethereum [40].

Several efforts have also been made to issue or verify education certificates by educational institutions using blockchain, the first being University of Nicosia(UNIC) that issued its first digital certificate for one its courses in 2015 [41]. University of Zurich(UZHBC) has also developed a blockchain solution [42] to verify diplomas issued by the university. Similarly, efforts to incorporate blockchain for educational credential verification in national level can be seen in India [43] and Malaysia [44].

W3C has drafted a standard for "Verifiable Credentials" as a digital and tamper-evident form of physical credentials. Verifiable credentials have the flexibility to be linked with real-world entities through Distributed Identifiers(DIDs) and can

support different types of credentials [45]. A verifiable credential can be presented for verification using a verifiable presentation that presents the holder's ownership of the credential through a digital signature scheme.

Similarly, In 2011, Mozilla and MacArthur foundations started working on the Open Badge Interface(OBI) to represent minute forms of credentials in the form of badges [46]. By the end of 2013, different implementations, such as Moodle [47], Blackboard [48], and Canvabadges [49] were created based on the OBI. Badges in this standard can be used to present comprehensive information about an achievement and work done to achieve it. Holder's who hold these badges can store it in a "Backpack" which allows them to import, maintain and scope these badges [50] from different platforms.

# Chapter 3

## System Design and Extensions

In this chapter, we discuss design decisions and methodology that were analysed and adopted in this thesis. We start by defining the system overview where we present how a certification authority and their organizational architecture can be mapped into the proposed blockchain application. Next, we discuss different system entities and functionalities that they are offered in the solution.

While our goal is to develop an application that can be adapted into different certification scenarios, we discuss this chapter with references to certification in an educational institution. This approach helps to better explain the design and present clear ideas by referencing it with real-world processes and entities.

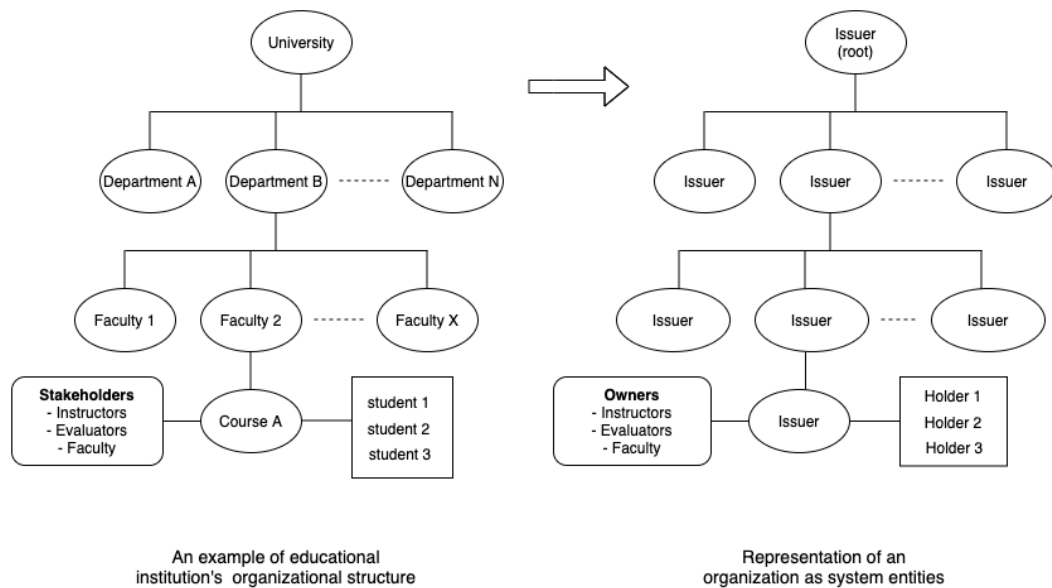
### 3.1 System Overview

The proposed system allows credential issuers, stakeholder of the issuing authority and recipient to jointly verify, validate and deploy credentials in the system. Our approach of on-chain verification assures that the values recorded as credentials are all registered by the issuer, signed by the issuing stakeholders and validated by the holder while increasing transparency in the organization's procedures for credential issuance.

Diverging from the traditional approach of summarizing all information in a single certificate, we break down certification into smaller, uniquely verifiable units referred to as *Credential* in the designed solution. These smaller units can then be combined into a single data structure that can be used to jointly prove achievement

of all the smaller units of credentials within it. This approach allows verification of a single credential or aggregated credentials as a single unit slashing the need of presenting all information about credentials stored in the aggregation.

We define 4 different entities in the proposed system 1) issuers, who collectively represent entity that registers earmarked data structures for their eventual holders 2) owners, who are the authoritative entity within an issuing organization and are responsible to verify the registered credentials and sign them 3) holders, who hold credentials awarded by the proposed system, and, 4) verifiers, who can be any address in the network that wants to verify a credential issued through the blockchain application.

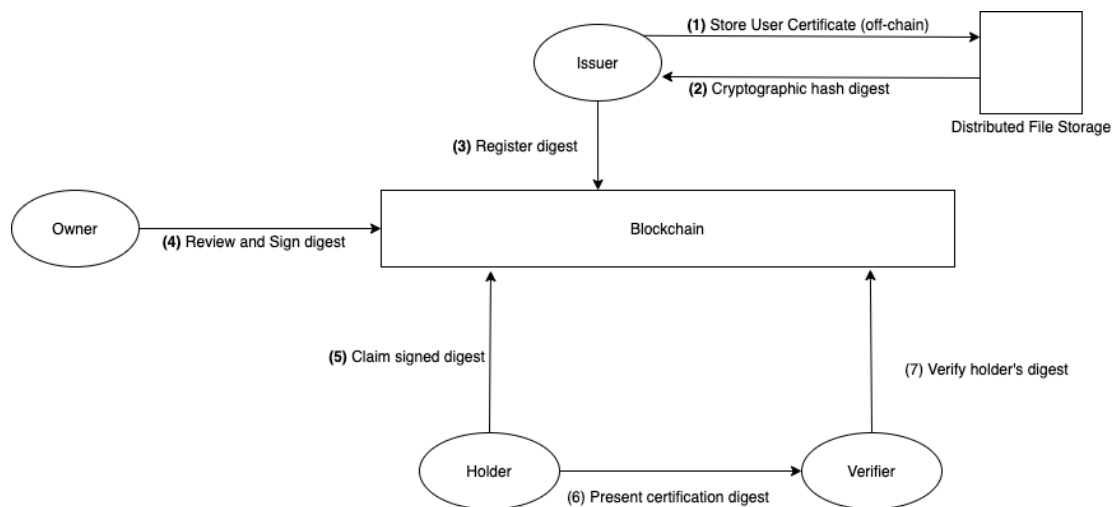


**Figure 3.1:** An example of educational organizational structure and its translation into different system entities

We present an example of translating different entities of a hypothetical educational institution into system entities in figure 3.1. Considering a hierarchical organizational structure, the topmost entity, the university, represent the root-issuer. Similarly, other issuers following the root-issuers are called sub-issuers. Here, department, faculty and courses are sub-issuers in the system, where the department is sub-issuer of the university, faculty is sub-issuer of the department and course is a sub-issuer of the faculty. This translation allows each entities at different levels to be identified as issuers in the system who have an ability to register earmarked data structures for holders in the system.

Similarly, each issuer entity can register its owners, who represent stakeholders responsible to verify the earmarked data structures. For a course, we see that instructor, evaluator and faculty are the owners while there are three registered holders representing students of the course. Here, students receive credentials after successful completion of each course work. Upon completion of all course works these credentials can then be aggregated into a credential proof representing student's achievement in the course. Similarly, aggregation of multiple credential proofs can be performed to generate a credential proof that represents a university degree. The credentials and credential proofs are stored in a holder's credential account data structure.

As shown in figure 3.2, entities in the system interact with blockchain as well as other resources outside the blockchain. Hence, we have divided the system into two layers of operation: off-chain layer, where crucial operations such as registering credential into off-chain storage is performed and the blockchain layer, where the data from the off-chain layer is incorporated into the blockchain application. Both of these layers depend on each other to complete a document issuance, authentication and verification workflow in the system.



**Figure 3.2:** High-level overview of interaction between system entities, off-chain components and the blockchain

### 3.1.1 The off-chain layer

We refer to operations performed outside the blockchain application as the off-chain layer. This includes workflows such as saving digital certification to an off-chain

storage and presenting verifiable data as a certification holder to the verifier. In figure 3.2, any arrow lines that do not interact with the blockchain are operations performed off-chain. These operations can be done with a client-side application that interacts with the blockchain. We discuss these operations below:

### **Distributed off-chain credential storage**

This thesis is scoped around building trust for an off-chain credential and not on defining the data structure of that credential. Hence, we briefly present how a credential might be stored off-chain and how it can be referenced in the blockchain application.

A credential that is issued to a holder is stored off-chain and can be in the form of verifiable credential [45], open badges [46] representation or a similar representation that presents information of the holder, issuer and subject for which a credential is awarded for among other details. These form of credentials are usually JSON-like data structure that can vary in terms of sizes. Storing these resources on-chain creates unpredictability in the data structure in the system while simultaneously increasing volume of data transferred in a transaction, increasing its cost and decreasing throughput of the system. This creates a need for a tamper-evident off-chain storage layer that stores these certifications which can be referred from the blockchain application.

We propose the use of distributed storage protocols such as the InterPlanetary File System (IPFS) [38] to add trust and tamper-evident mutability to any certification published off-chain. Protocol like IPFS provides similar guarantees as blockchain for tamper-evident record keeping. It divides a published certification file and all the blocks within it among peers in the network resulting in a cryptographic hash value that represents the data stored in the distributed storage. We refer to this hash value as the credential digest in the modelled blockchain.

**Credential Digest** Credential digest, or simply digest, is a hexadecimal value representation of a certification data that is stored off-chain. While we presented in the above section how a trusted cryptographic representation to an off-chain credential can be generated and registered in the form of on-chain credential digest, it is equally important to build trust around this value on-chain to prove its

authenticity and validity. Hence, embedding this as a part of the blockchain data structure allows different stakeholders to verify and vote for its authenticity and the issuer can control its validity by issuing or revoking it through on-chain procedure invocation.

### **Off-chain sharing of certification**

The process of presenting a certification received through the blockchain application to any real-world entity for verification is an off-chain process. The blockchain guarantees that the recipient can selectively fetch verifiable data from the application. This information can then be shared off the chain and be used by anyone with an address in the network to verify its authenticity and association with the holder.

### **3.1.2 The blockchain layer**

The blockchain layer of the proposed solution is made up of the blockchain application and the interfaces that can be used by different system entities to interact with the blockchain application. We briefly introduced different system entities in section 3.1, and, in the following text we further look into specific functionalities available to them.

### **System entities and functionalities**

#### **Issuer**

Issuers in the system are authoritative entities. They are responsible to handle holders enrolment and to earmark credential digest as part of credential for holders into the system.

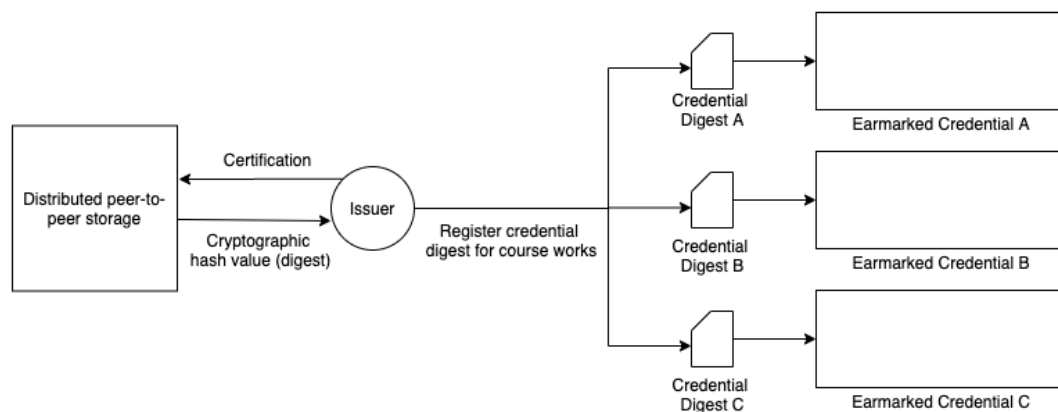
Issuer entity are defined such that the credentials issued through them automatically has features from the issuer if not specified manually. For example, owner definition in an issuer's resource dictates an access control mechanism for Libra accounts to sign a registered credential, quorum defines the number of owner verification needed to consider a credential as owner-verified. This information is translated into credentials as they are registered into the system. Thus, credential itself holds



information about the issuer, owners and signatures from the owners which will later aid in the credential verification workflow.

Different issuer-specific functionalities that the system provides are:

**Earmark and aggregate credentials** An issuer entity can either be a root issuer that represents the highest level authority in an organizational structure or a sub issuer that can be an issuer at any level below the root issuer. A root issuer is responsible to instantiate and earmark a credential account for holders upon enrolment and to generate the final credential proof that represents achievement of the holder. In an education setting, the aggregation performed by root issuer can be understood as the final aggregation of credential proofs that represents all courses to generate a university degree in the form of a new aggregation. Similarly, both root issuer and sub-issuers can register earmarked credentials into the system. A sub issuer holds information about its parent issuer and the root issuer allowing users to define a hierarchical structure with multiple levels of sub-issuers. We can represent issuers in an organization in a *Tree* data structure where a root issuer is at  $depth = 0$ . Any subsequent sub-issuer holds the information about its parent issuer and the root issuer.



**Figure 3.3:** An example of issuer earmarking three credential hash representing certification of three courses into the system as three different credential

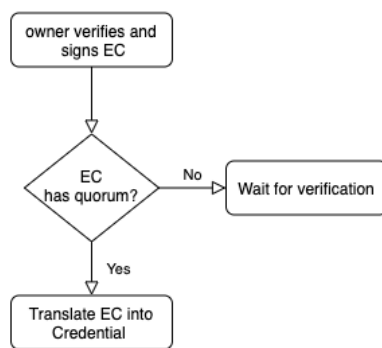
**Revoking an issued credential** An issuer can revoke an issued credential and save it as revoked resource. A list of revocations holds all the revoked resources and is registered with the issuer who issued the credential or aggregated them. This guarantees a cheap and safe revocation discovery during the process of credential

verification. Upon revocation, credential and credential proof resource is removed from the holder's credential account and moved into issuer's revoked resource.

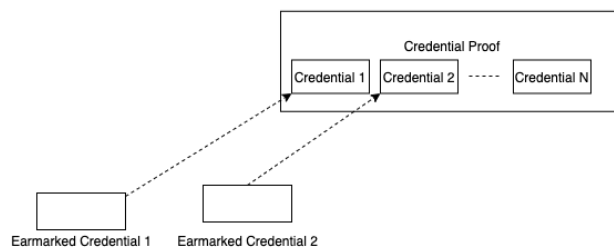
**Timed credential** Validity of an issued credential can be timed by defining a validity start and end time in the credential data structure. Any credential without this time definition is considered to be valid indefinitely. Creating a timed credential automatically sets this time on the credential proof data structure that aggregates this credential. These data structures lives within the credential account even after the credential validity expires as it represents a credential once achieved by a holder.

### Owner

Similar to instructors and evaluators of a course in an education system, we define owners to represent such authoritative stakeholders for an issuing entity. They are responsible to verify an earmarked resource and work as a validator in the process of translating an earmarked credential into an issued credential.



**Figure 3.4:** Quorum requirement for verification of Earmarked Credential(EC)



**Figure 3.5:** Translation of Earmarked Credentials into Credentials

**Verifying an earmarked resource** Once an issuer earmarks a credential with off-chain digest value for a holder, it is then the owner's responsibility to verify and validate the digest's association with the holder. This verification is an off-chain process and requires the check of the associated holder, issuer, time of validity of the off-chain credential with the on-chain earmarked credential. Upon verification and validation, owners sign the hash of combination from issuer address, holder

address, credential digest and validity start and end time generating a signature that represents the owner's vote towards the proof of validity and association of a credential with the defined holder. An earmarked credential waits until a quorum of signatures are collected in the credential before it is moved into an earmarked credential proof as shown in figure 3.4 and figure 3.5.

## **Holder**

Holder represent entities in the system to whom a signed credential is awarded. Upon successful completion of Issuers requirements to achieve a credential proof, holders can claim an aggregated credential proof from the Issuer.

As a holder, any address in the network cannot directly be linked with a sub-issuer without initially registering with the root issuer. Registering with the root issuer presents the holders with their credential account where all credential issued by the issuing authority towards the holder is registered. The credential account allows a signed credential to be safely stored with a holder and it is sufficient for the issuer to hold information on which digest-holder relation to avoid digest collision and for low-cost document verification.

Based on different data structures created and moved between entities in the system, a holder can present the following proofs based on the presence of different data structure with different entities.

**Proof of certification** Holders have an ability to extract digest values awarded by a defined issuer from their credential account. This allows holders to selectively present digest that is most relevant to the situation they are in. For an example, if a holder is to present certification for completion of a particular course, then, they can present the aggregated credential proof as the certification. Similarly, if they just have to present that they had submitted a particular coursework then they can submit the credential digest. The blockchain application can then be queried with this digest value and student's address to validate its validity and authenticity.

**Proof of association with issuing authority** There are two types of association that a holder can prove with an issuing authority:

- Association with root issuer: As a holder is registered with a root issuer, the root issuer creates a credential account for the holder. The existence

of this credential account created by a particular issuer signifies the fact of recognition from an issuer towards the holder as presented in Issuer's functionality of creating credential account. In the case of an educational institution, the existence of this data structure proves the holder's enrolment into the organization.

- Association with sub issuer: Once an issuer, registers a holder, it creates an empty earmarked credential proof data structure. This data structure works as a container of credentials in the later stages of the application but in the initial stage, the presence of this earmarked data structure for a holder proves issuer's association with the holder. In the case of educational institution, this data structure proves the enrolment of the holder in a course.

### **Verifier**

A verifier can be any address in the Libra network that wants to verify a credential using the developed interfaces. A verification query needs a digest value and the holder's address. This query is responded with an event in the blockchain system which defines the validity of the presented digest and its association with the presented holder's address.

## **3.2 Extensions**

In this section we propose some extensions to the blockchain application discussed previously in this chapter. The extension is aimed to discourage any adversary efforts in falsifying an issuing authority.

### **3.2.1 Permissioned issuers consortium**

While the process of certification is an authoritative process, the proposed permissionless blockchain application in section 3.1 allows any address in the network to instantiate an issuer resource. Nevertheless, sub issuer in the system can only be instantiated after permissioned inclusion of their addresses in the parent issuer's white list of sub issuers.

We propose an extension for creation of a group of permissioned issuers, referred to as issuer consortium. Starting with a certain number of permissioned member nodes, we propose a consensus voting among the members to add or remove issuers in the group. This structural constraint allows only authorized issuer's to become part of the consortium allowing them to register as a root issuer.

For  $N$  different issuer nodes in the issuer consortium( $IC$ ), all nodes have voting power equally divided among them. We define a public procedure that allows any public node( $N_o$ ) to propose entry into the issuer consortium. After registration of the proposal, it is defined with a voting period( $t$ ) that allows the members of the consortium to vote for inclusion of the node  $N_o$  until the end this period.

If  $\frac{N}{2} + 1$  votes are collected for the inclusion of the new node, then the node  $N_o$  is included into the consortium updating the total members in  $IC$  from  $N$  to  $N + 1$ .

The integrity of issuers in the system is then dependent upon the righteous use of voting amongst the issuers in the issuer consortium while allowing nodes to acquire a root issuer privilege through consensus voting.

# Chapter 4

## Implementation with Libra Blockchain

In this chapter, we present a practical implementation of the design presented in chapter 3 using the Libra blockchain. Further, we discuss permissioned group for issuers in the system as extension to the proposed application. Finally, we trace a real-world use case of the application in terms of certification scenario other than in educational settings.

### 4.1 Libra System Components

The proposed blockchain application can be divided into modules, resources and transaction scripts. These components are what builds a blockchain application in the Libra blockchain.

As presented in section 2.3, modules define logical operation at the core of Libra as well as custom user-defined codes. A module defines procedures that can be invoked using transaction scripts. Such procedures can be either publicly-accessible or private within a module. While procedures can operate with given input arguments, they rely on Libra resources to store data into ledger state and read from them whenever required.

An abstract data model that shows the relation between the developed modules and resources is presented as ledger state representation in figure 4.4 and figure 4.6.

This representation associates modules and resources for issuers and holders in the system. Other entities, such as owners and verifiers interact with the system through procedures defined in the modules but do not libra resources associated with their account during this process.

### 4.1.1 Modules

We introduced Libra modules in section 2.3.3 and we use such modules to create a logical data flow in the system.

For any Libra account to use a Move module, the module has to be published first using a Libra account and the same account should compile and generate the transaction script bytecode that interacts with the published modules. In the following sections, we consider the module publisher to be a Libra account, represented as *Module Owner*. Other entities in the system such as the issuer, owners, holder and verifier send transaction using the compiled script bytecode with arguments to interact with the published modules.

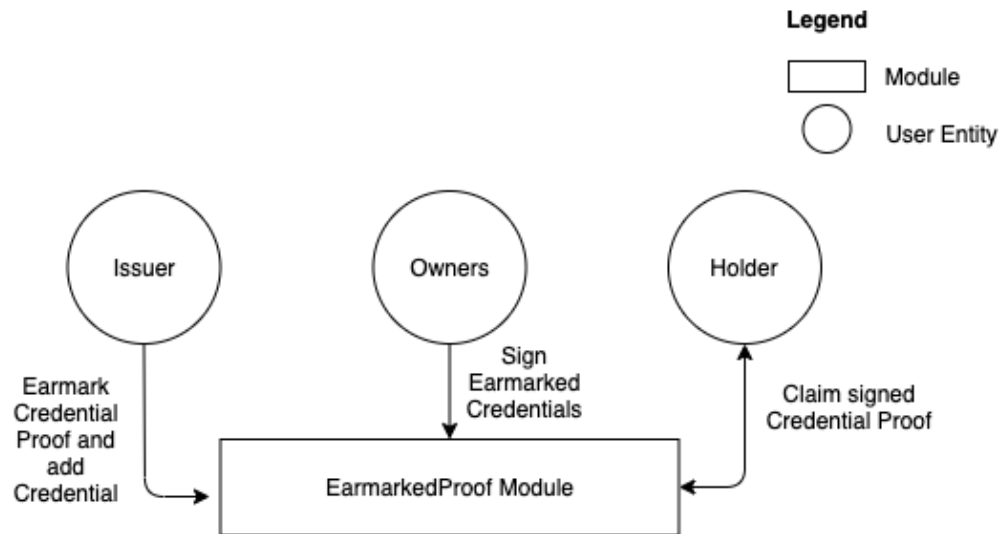
There are three distinct modules in the system and are defined as follows:

#### Proofs Module

*Proofs* module defines *Credential*, *CredentialProof*, and *CredentialAccount* resources and the procedures that can create or modify them. These are core resources in the system and are common between issuers and holders. Other modules depend on this module as a core resource provider. Resources defined in this module are thus assessed through *EarmarkedProofs* and *Issuer* modules as we define in the following sections. This design decision abstracts core resources away from the modules where users can directly interact with. Moreover, access checks for the defined procedure are defined based on the resources possessed by the transaction sender as each entity in the system will own different resources. Similarly, any address in the Libra blockchain network can perform the role of a verifier and hence this module also provides the verification procedure that can be invoked to verify a credential digest. Upon verification of the provided digest, the modules emit an event that can be queried to get the verification result.

## EarmarkedProofs Module

EarmarkedProofs module abstracts resources and procedures defined in the Proofs module and is used to define operations that involves (1) issuer, who earmark a credential and credential proof resources for a holder (2) owners, who sign the earmarked credentials and (3) holders, who can claim the signed earmarked credential.



**Figure 4.1:** Logical representation of entity interactions with the Earmarked-Proofs Module

As a credential is assigned by an issuer towards a holder, it is registered as an entry in the *LoggedProofs* resource of the Issuer. We propose a consensus voting-based approach on earmarked credentials that requires a defined quorum of signature from involved stakeholders before it is can be claimed by the designated holder.

Stakeholders, defined by *Owners* of the *Issuer* resource have access to verify an earmarked credential. As a credential receives quorum of signatures, it is then moved into an earmarked *CredentialProof* for the defined holder under the *LoggedProofs* resource of the issuer. Issuer of this *LoggedProofs* resource can then aggregate credentials present in the credential proof resource representing completion of credential issuance for the issuer.

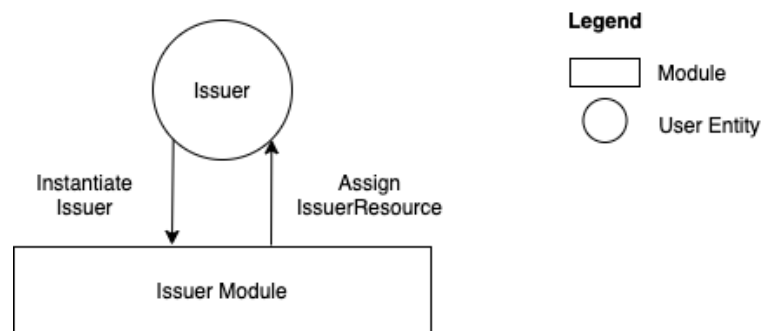
This workflow assigns a quorum-signed earmarked resource as logged proof and not directly to the holder. Similarly, as a holder claims this signed earmarked resource, the signatures on credentials and the aggregation of the credentials



in the credential proof are verified. This builds up a multi-party trust towards credentials and credential proof resource between the issuer, who initially registers the credential, owners who verify and sign the credential and holder who claims the signed credential through credential proof. Moreover, as the resource data cannot be mutated outside the procedures defined in the module, Libra guarantees that credentials once signed will remain signed and under the possession of a holder's credential account until it is revoked by the issuer.

The Earmarked module also defines the *DigestHolderProofs* which registers information about digest-holder association and the *RevocationProofs* resource that defines vectors for credential and credential proof that has been revoked. These resources help in the verification of the smallest units of credential during the verification workflow in the system and aids in selective disclosure of achieved credentials.

## Issuer Module

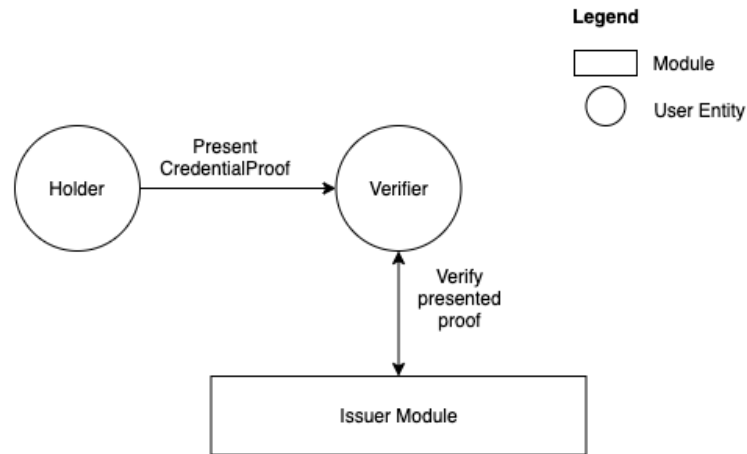


**Figure 4.2:** Issuer interaction with the Issuer module

The *Issuer* module defines issuer specific resource and relies on functionalities defined in *Proofs* and *EarmarkedProofs* modules.

Issuers are initiated through this module. An issuer, in terms of the developed blockchain application, are trusted Libra address that owns the resources defined in *Issuer*, *EarmarkedProofs* modules. We define the resources created through these modules collectively as the issuer resources. Similarly, a root-issuer can be distinguished from a sub-issuer based on the *parent\_address* variable present in *IssuerResource* resource.

This module is also responsible to define other issuer-specific functionalities, such as, a holder’s registration with the issuer, registration and creation of issuer and sub-issuer.



**Figure 4.3:** Verifier interaction with the Issuer module

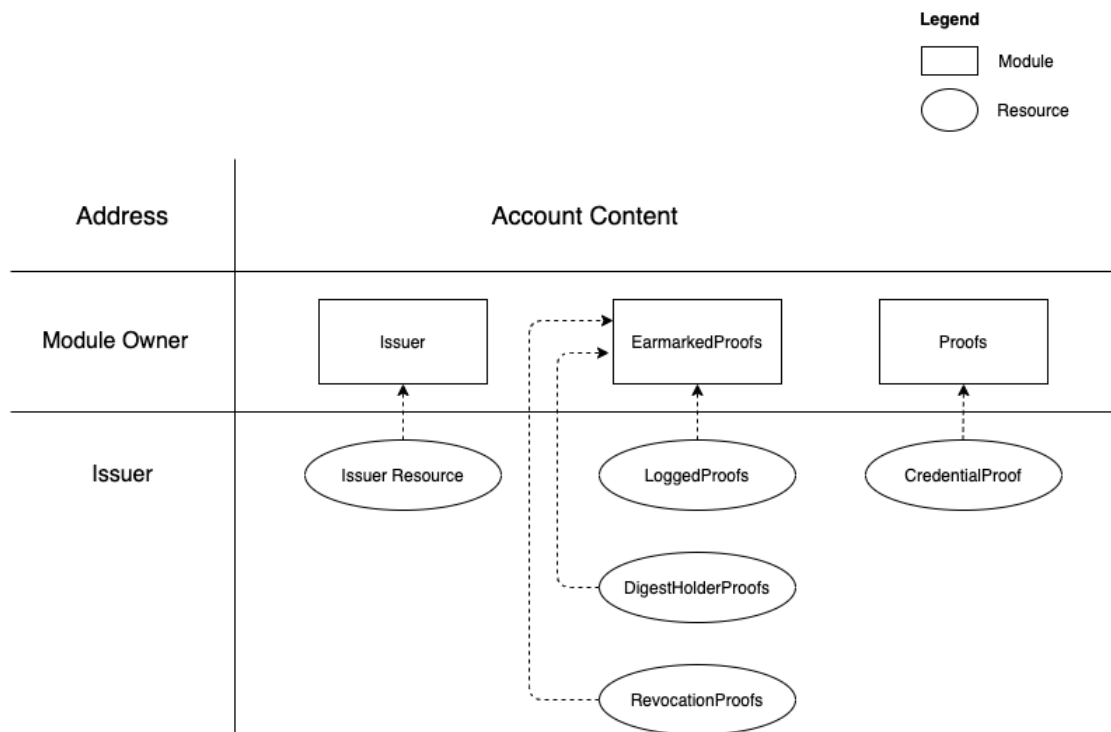
### 4.1.2 Resources

Resources in Libra ecosystem are data structures that can only be created, edited or deleted by the module that defines it. This guarantees that the core data structures cannot be used by other modules in the blockchain. This further allows the definition of resources to differentiate one system entity from another.

A resource created from a module can only be moved into a self-owned account and thus prevents adversary efforts to create resources and move it into a victim’s account. Earmarking approach of resources for holders in the proposed application is thus implemented at the core of the blockchain operations. Such earmarked resources has to be claimed by the receiving account to authenticate that they actually are willing to receive the earmarked resource.

Similarly, deleting a resource is possible with the module that creates it. But, the developed application restrain from resource deletion. We move resources into different data structures to represent a change in its state. For an example: a credential proof once claimed by a holder can be revoked. This revocation doesn’t delete the claimed credential proof but moves it under issuer’s revoked resources thereby changing the state of the same data structure from a valid credential

to revoked credential. Similarly, a newly registered credential is introduced as an untrusted value, which after receiving a quorum of signatures is moved into credential proof, thereby, changing its definition from an untrusted to trusted by quorum of owners.



**Figure 4.4:** Issuer Modules and Resources

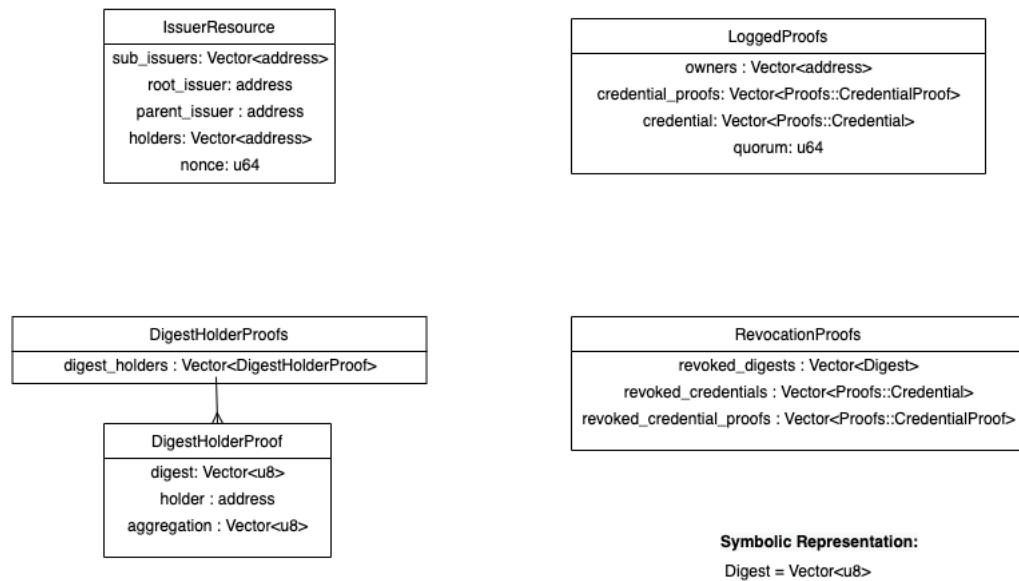
In the proposed application, it is either the issuer or the holder that acquires resources created through the application differentiating roles of these entities. Hence, we have categorized different resources as issuer and holder resources.

### Issuer Resources

Issuers in the system has two major roles. First, is their interaction with holder entity, which might involve registering of credential or their registration with the issuer, and, the second is to register an organizational structure that best involves stakeholders at different levels of the organization. This information in the blockchain is driven by four different resources, created through *Proofs* and *EarmarkedProofs* modules, collectively defined as issuer resources.

*IssuerResource*, *LoggedProofs*, *DigestHolderProofs* and *RevocationProofs* defines the four issuer resources and figure 4.5 present their data structure. Similarly,

the relation between these resources and their corresponding modules is as shown in figure 4.4. Dividing issuer centric information into different resources allows abstraction of data present in one resource from the other and allows unambiguous procedure definition to access them.



**Figure 4.5:** Overview of issuer resources and their data structure

**IssuerResource** An *IssuerResource* is a Libra resource that is moved into an issuer’s account upon their registration. As shown in figure 4.5, this resource consists of information about an issuer’s organizational structure, which includes, sub issuer, parent issuer, holders and the nonce value. The nonce value is a global counter for the number of actions performed by the issuer in the blockchain application.

**LoggedProofs** A *LoggedProofs* resource is at the core of earmarked credential registration and owner verification of these resources. An earmarked resource is logged into this issuer’s resource and its definition of owners allows the owners to sign the logged credentials. Presence of a Credential in the logged proof represents that the credential was generated by the issuer but still has to be signed by the owner. Once the required number of owners defined by the *quorum* value sign the *Credential*, it is moved into earmarked *credential\_proofs* vector of *LoggedProofs* resource. This value recorded in the *credential\_proofs* is the resource that a holder can claim after the issuer aggregates credentials present in them.

**DigestHolderProofs** Once a holder claims a *CredentialProof*, a record on the aggregated hash value from credentials of the *CredentialProof* is saved with the Issuer under this resource. At the issuer's end, this resource helps to relate a credential digest to the holder.

**RevocationProofs** A credential proof that is revoked is moved away from holder's account and placed as a part of Issuer's *RevocationProof*. While the credential proof resources are held as a part of the revocation proof, an issuer also maintains a list of revoked digests that defines the revocation of aggregated credential digests which was once issued as a certified credential proof.

## Holder Resources

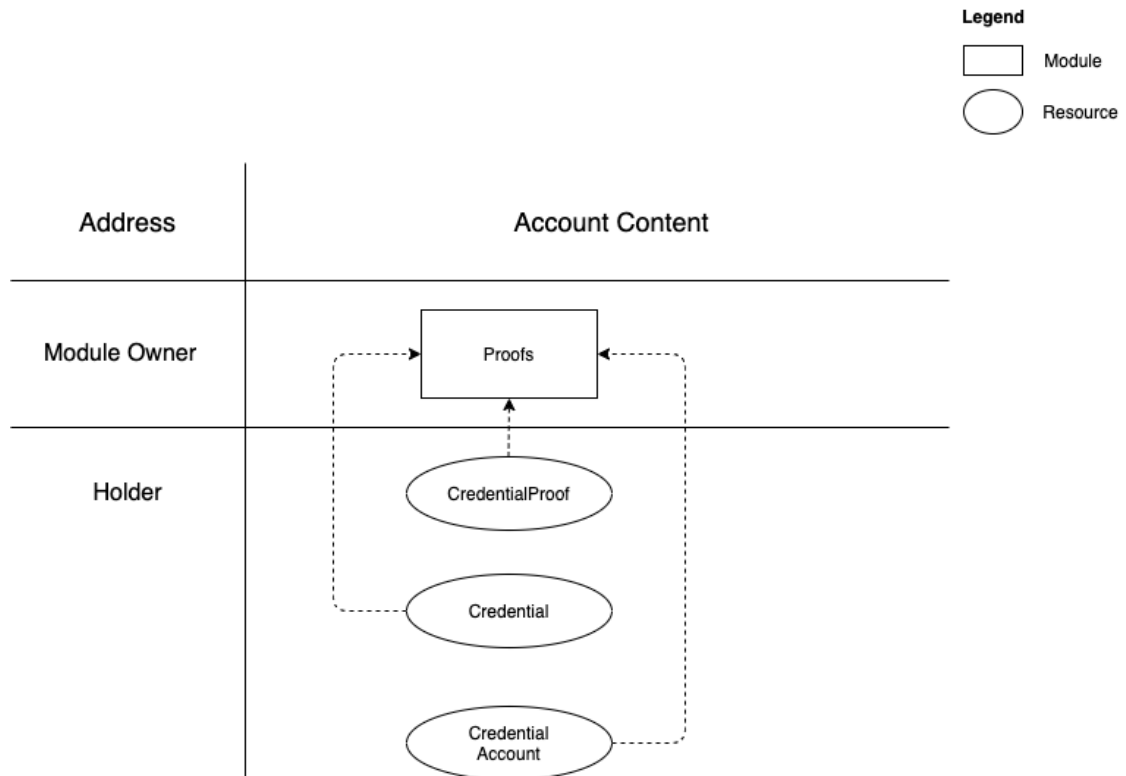
In different certification use cases, when a credential is awarded to a holder it is the holder's copy of the certified fact. Similarly, we propose certifications, defined by *Credential* and *CredentialProof* as shown in figure 4.6, as a holder's certified resource that is stored under their account in the Libra network. This represents a system that is similar to real-world certification where a holder possesses a valid copy of the credential.

Holders interactions can broadly be categorized into two operations, first is the process of adding trust from holder's perspective towards any resource earmarked for them and second is the interaction with their credential account. These operations are based on the resources that a holder's Libra account possess. All the holder's resources are created from the *Proofs* Module by an issuer and later claimed by the holder.

Figure 4.8 shows the organization of credentials, credential proofs and credential account as a holder's resource.

## Credential

Credentials are the smallest unit of certification and prove the fact of achievement of Issuer defined requirements for achieving a credential. For a course that is an issuer, then, a course work or a lab examination can be an example of credential. It is represented as a Libra resource that holds the external certified fact as byte array referred to as digest or credential digest.



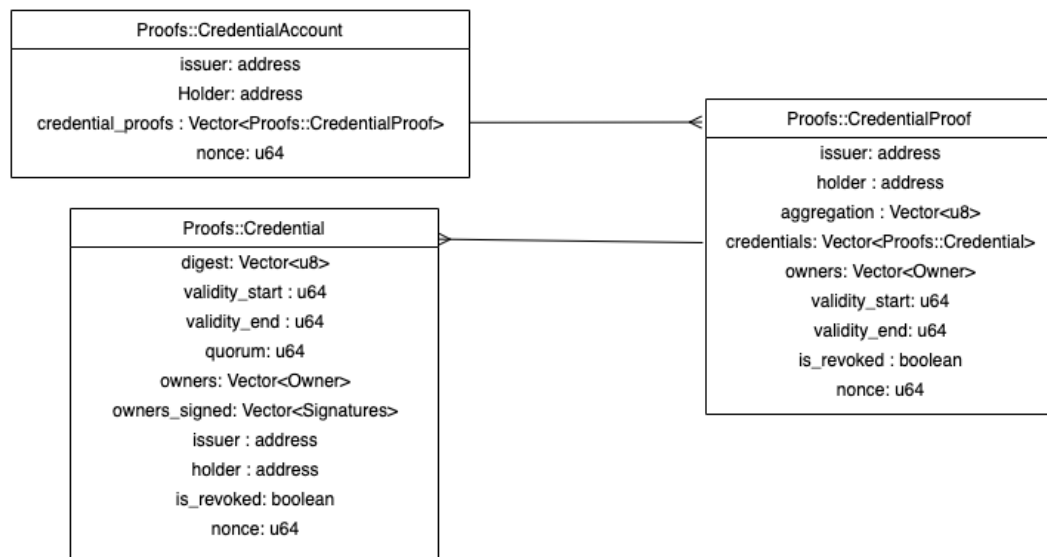
**Figure 4.6:** Holder Modules and Resources

### Credential Proof

A credential proof, defined by *CredentialProof* resource is a Libra resource and a wrapper around credentials assigned to a holder under a particular issuer. Initially, as a holder is registered with a sub-issuer, an empty *CredentialProof* resource is earmarked and saved into *LoggedProofs* resource which represents the holder's enrolment with the issuer. As the holder completes issuer's requirements to achieve a certificate, the digests of underlying credentials are aggregated by the issuer resulting in an aggregated hash value. This hash value is stored in credential proof and verifies that the holder meets all the requirements to have been presented with this certification. Now, a *CredentialProof* is ready to be claimed by the designated holder.

### Credential Account

A credential account resource is issued towards a holder's Libra account by an *Issuer* module when registering the holder with the issuer. This resource asserts trust in the holder address and represents a proof of enrolment for the holder to verify the association between them and the issuer. Upon achievement of credential



**Symbolic Representations:**

Owner = struct { address : Vector<address>, public\_key : Vector<u8>}  
 Signatures = Vector<Vector<u8>>

**Figure 4.7:** Overview of holder resources and their data structure

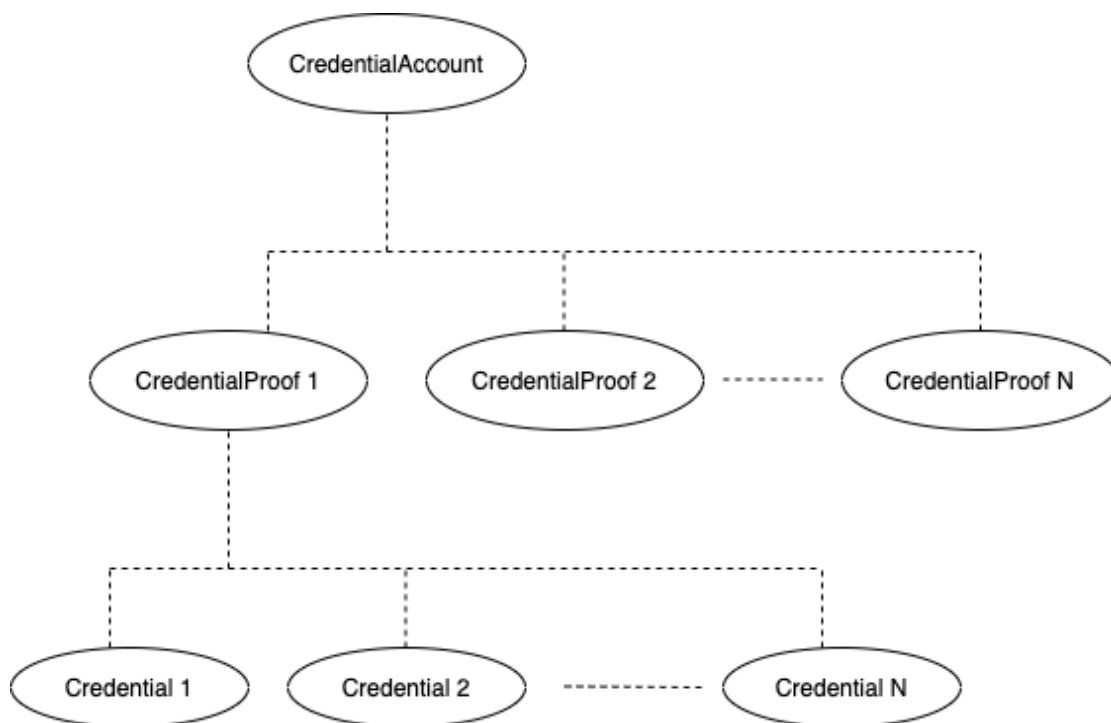
proofs, this proof of certification is moved into a holder’s credential account and stored with the holder’s account.

A credential account works as a wrapper for all the credential proofs issued towards and claimed by a holder. It imitates real-world certification where holders possess a valid copy of a certification which can be directly presented to a verifier or be verified through the issuer.

### Owner and Verifier

The proposed solution doesn’t associate resources for a verifier or an owner. Owners of for an issuer is defined by the issuer during its registration, hence, it is expected that the list of owners are verified and it doesn’t require any extra on-chain verification. Similarly, credential verifiers can be any entity in the network that verifies a digest through the proposed solution. Hence, no resources are allocated to these entities.

Even though an owner doesn’t possess any on-chain resource, it is responsible to hold a private and a public key. The private key stored with the owner and is used



**Figure 4.8:** Logical organization of credentials and credential proofs in a credential account

to sign credential digest saving the signature in the credential and the public key should be registered with the respective issuer prior to their instantiation of the issuer resources. Thus, owners are defined by the combination of Libra address and its public key in the issuer’s resource and it can be referenced whenever there is a need to verify a signed data value in a credential.

## 4.2 System Features and Algorithms

Different operations that an entity can run on the blockchain are the features of the blockchain application. In this section, we present technical specifications of related features and algorithms specific to the developed Libra blockchain application.

We divide different features into entity-based groups and present algorithms to define a high-level interaction and operations in different workflows. In these algorithms, we use procedure calls defined with ‘get’ prefix, such as *getIssuerResource* and *getIssuerLoggedProofs*, to denote invocation of related Libra resources. Similarly, *get\_txn\_sender* is a Move procedure call that returns the address of the transaction sender.



## 4.2.1 Issuer

### Issuer Registration

Public scoping of procedures in Libra modules allows any node in the network to register as a root issuer. Registering as an issuer moves issuer-specific resources into the Libra account of the transaction sender allowing the sender account to work as an issuer. An extension to this is the permissioned issuer registration that we discussed in section 3.2.1.

**Sub-issuer Registration** A root issuer in the system can further allocate sub-issuer privilege to other nodes in the network by including them as a *sub\_issuer*. Sub issuers can then register under the defined root issuer. This is a permissioned method of sub issuer registration under an issuer because we do not want any nodes in the network to be able to instantiate itself as a sub-issuer of an issuer without their approval. Sub-issuers of an issuer is tracked under *IssuerResource* as a part of *sub\_issuers* vector.

---

#### Algorithm 4.1 Registering a sub-issuer

---

```

1: function registerSubIssuer(address)
2:   ir = getIssuerResource(get_txn_sender())           ▷ get issuer resources
3:   if address not in ir.sub_issuers then
4:     ir.sub_issuers[] ← address                       ▷ register sub-issuer address
5:   end if
6: end function

```

---



---

#### Algorithm 4.2 Initiating as sub-issuer

---

```

1: function initSubIssuer(address)
2:   ir = getIssuerResource(address)
3:   sender = get_txn_sender()                          ▷ get transaction sender address
4:   if sender in ir.sub_issuers then                  ▷ check if sub-issuer is registered
5:     if !(exists(sender, IssuerResources)) then
6:       move(IssuerResources, sender)                ▷ create resources for sub-issuer
7:     end if
8:   end if
9: end function

```

---

## Holder Registration

Registering a holder with an issuer refers to the process of adding trust in the system from issuer towards the holder. In the libra implementation, as a root issuer registers a holder, it creates an empty entry of *CredentialAccount* for the holder. Once claimed by the holder, the issuing organization's credential account entry is moved to a holder's credential account if it exists, or, it creates a credential account with the new entry for the holder if the credential account doesn't exist.

Similarly, when a sub issuer registers a holder, they create a empty *CredentialProof* resource and save it as part of its *LoggedProofs* resource. Holders for an issuer are simply a vector of addresses. This is an append-only vector and entries are added only when a holder is registered with the issuer.

---

### Algorithm 4.3 Registering Holder with Sub-issuer

---

```

1: function registerHolder(address)
2:   issuer_address = get_txn_sender()
3:   ir = getIssuerResource(issuer_address)
4:   lp = getIssuerLoggedProofs(issuer_address)
5:   if address not in ir.holders then
6:     ir.holders[] ← address           ▷ record holder address with issuer
7:     cp = newCredentialProof(address) ▷ Create earmarked credential
      proof
8:     lp.credential_proofs[] ← cp     ▷ register a logged resource
9:   end if
10: end function

```

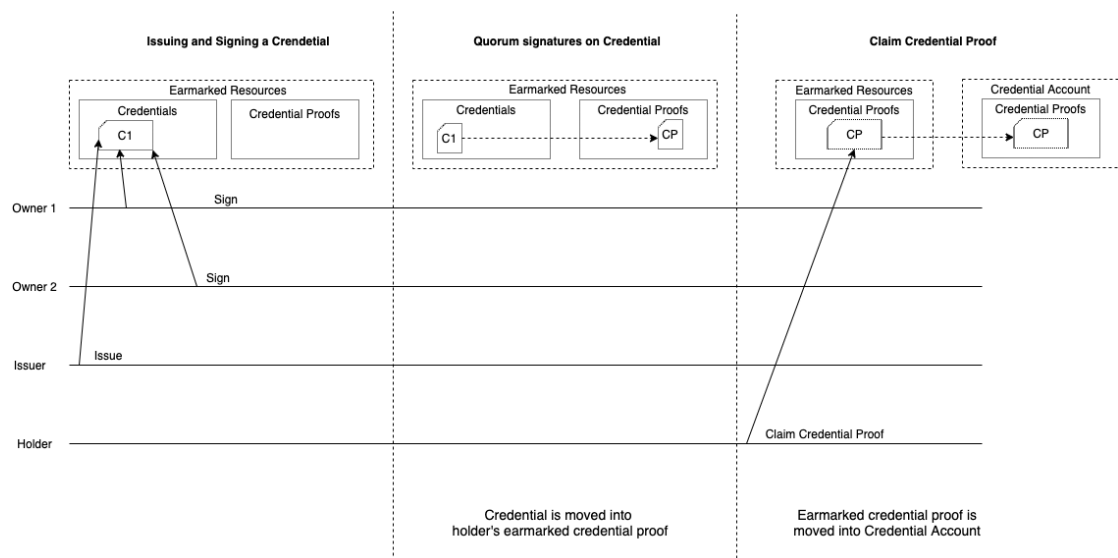
---

## Credential Registration

Once a holder achieves a credential, it should be recorded to an off-chain storage resulting in a value that references to the off-chain certificate as defined in section 3.1.1. This value is registered as *digest* and as a part of the *Credential* resource where other values such as the issuer, owners and the owners signatures are also defined as shown in figure 4.5.

Upon a successful registration of this credential, it is saved as a part of *LoggedProofs* resource and is an earmarked resource. The owners can now verify and sign the credential digests from this resource.

**Timed credential** A credential is a timed credential, if its *validity\_start* or *validity\_end* time are defined. These values are 64-bit unsigned integer values that represent a Unix timestamp. If *validity\_start* is not defined on a credential, then it defaults to the time when the credential is earmarked into *LoggedProofs* resource. Similarly, *validity\_end* is set to null if not defined resulting in no limits for validity period for the credential. *validity\_end* on a *CredentialProof* resource is based on the smallest *validity\_end* period of the constituting credential. Similarly, *validity\_start* is based on the largest *validity\_start* value of the constituting credential. If these values are defined, they are checked when a credential digest is verified in the verification workflow.



**Figure 4.9:** A timeline representation of credential registration, signing and claim

## Credential Aggregation

Upon completion of requirements for achieving a final credential, an issuer must aggregate all the credentials within the credential proof. This produces a single hash value that represents a combination of the digest and other constituting variables from the credentials in the credential proof. This aggregation represents a Merkle tree structure and the resulting aggregation represents the root of the tree.

Let us consider,  $C_1, C_2, \dots, C_N$  to be quorum verified credentials registered into a holder's *LoggedProofs* under *CredentialProof(CP)* resource for an issuer  $I$ . If these credentials define the completion of all requirements of the issuer to issue a final certificate, then, these credentials are aggregated by the issuer  $I$ .

This aggregation hashes the credential digest defined as the leaf nodes first and consequently combines and hashes the resulting inner nodes of the tree to produce a final root node. The value is then registered as *CP.aggregation*. Once, this aggregation is computed, the earmarked holder for *CP* can claim this resource into their credential account.

## Credential Revocation

Revoking an issued credential from the holder's credential account involves two steps. First, we move the credential and associated credential proof to the Issuer's *RevocationProofs* resource, and secondly, we record all the associated digests from these data structures as *revoked\_digests* in the *RevocationProofs* resource. This is queried whenever a new credential digest is registered or someone verifies a digest.

### 4.2.2 Owner

#### Sign Credential

Signing of credential is one of the vital steps in the document verification process. This is where owners can verify that the registered digest is valid or not. We assume that a manual verification of digest is done off-chain and if valid then only signed by each owner. Once, a quorum of owner signs a credential, the credential is then moved to the respective *CredentialProof* of the designated holder.

Let us consider an issuer with  $O_N$  number of owners and  $n$  is the defined quorum requirement for earmarked credential. Furthermore, if,  $C$  represents a credential earmarked by and issuer,  $O_{iPK}$  represents a private key of the  $i^{th}$  owner,  $C.quorum$  represents the value inherited from the issuer as quorum requirement, then, the value hash value  $H$  is signed using  $O_{iPK}$  and saved into  $C.signatures$  field with the operation  $C.signatures \leftarrow sign(H, O_{iPK})$  where  $H$  represents hash of combination of  $C.digest$ ,  $C.holder$ ,  $C.issuer$ . After this operation, a credential is checked for a quorum of signatures and the is moved into earmarked credential proof if the check is valid.

---

**Algorithm 4.4** Sign Earmarked Credential
 

---

```

1: function signCredential(issuer, digest, holder,  $O_{iPK}$ )
2:   return if digest is not registered with the issuer
3:    $ER = IRM.getEarmarkedResource(issuer)$ 
4:   for credential in  $ER.Credentials$  do
5:     if  $credential.digest == digest$  and  $credential	holder == holder$  then
6:        $h = Hash(credential.digest || credential	holder || credential.issuer)$ 
7:        $credential.signatures \leftarrow sign(h, O_{iPK})$ 
8:     end if
9:   end for
10:
11:   if  $len(credential.signatures) \geq credential.quorum$  then
12:      $cp \leftarrow EPM.getHolderCredentialProof(credential	holder, issuer)$ 
13:      $cp.credentials \leftarrow move(credential)$ 
14:   end if
15: end function

```

---

### 4.2.3 Holder

As a holder enrolls with an issuing authority, they can claim a *CredentialAccount* resource that will later be used to insert credential proofs from completion of requirements posed by the issuer towards receiving credentials from them.

#### Claim Credential Proof

If an issuer has a *CredentialProof* earmarked and all the credential within it is aggregated, then the designated holder can claim the credential proof. The credential proof data structure is created as soon as an issuer registers a holder and resides in the *LoggedProofs* resource. Upon a successful claim, the *CredentialProof* that belongs to the holder is moved to the holder's *CredentialAccount*

---

**Algorithm 4.5** Holder Claim for earmarked Credential Proof
 

---

```

1: function claimCredentialProof(issuer)
2:    $ep = getIssuerEarmarkedResource(issuer)$ 
3:    $credential\_account = getCredentialAccount(get\_txn\_sender())$ 
4:    $cp = ep.getHolderCredentialProof(caller\_address)$ 
5:   if  $exists(cp)$  and  $(cp.aggregation == aggregate(cp.credentials))$  then
6:      $credential\_account.credential\_proofs[] \leftarrow cp$ 
7:   end if
8: end function

```

---

## 4.2.4 Verifier

### Verify Digest

As shown in algorithm 4.6, once a verifier submits a digest and a corresponding holder address for verification. Then the holder's credential account is checked for existence of the provided digest in the form of credential or credential proof. If it exists then the resource found is checked for time and signatures validity. In case if the digest belongs to a credential proof, then the underlying credentials are aggregated and the aggregation is validated as well. Checking for digest in credential account works here because any resource that is revoked would already have been moved out of a credential account.

---

#### Algorithm 4.6 Verification of digest

---

```

1: function verifyDigest(digest, holder)
2:   credentialAccount = getCredentialAccount(holder)
3:   isCP = false;
4:   if digest ∈ credentialAccount.credentialproof_digests then
5:     resource = getCredentialProofByDigest(credentialAccount, digest)
6:     isCP = true;
7:   else if (digest ∈ credentialAccount.credential_digests) then
8:     resource = getCredentialByDigest(credentialAccount, digest)
9:   end if
10:  if resource then                                ▷ Process only if the resource is known
11:    if EPM.isValidResource(resource, resource.issuer, isCP) then
12:      emit VerificationEvent(true);
13:    end if
14:  end if
15:  emit VerificationEvent(false);
16: end function
17:
18: function EPM.isValidResource(resource, issuer, isCP)
19:  assert(credential.digest.issuer.RevocationProofs.revoked_digests) ▷ check if revoked
20:  assert(credential.digest.validity_start < now())                ▷ check start time validity
21:  if resource.digest.validity_end then
22:    assert(now() < digest.validity_end)                          ▷ check end time validity
23:  end if
24:  if isCP then                                                ▷ check if Credential Proof or Credential
25:    if validateSignatures(resource.owners_signed) then        ▷ validate signatures
26:      return true;
27:    end if
28:  else
29:    validSignatures, aggregation = aggregate(resource.credentials) ▷ validate
signatures and aggregate credential digests
30:    if (resource.aggregation == aggregation & validSignatures) then
31:      return true;
32:    end if
33:  end if
34:  return false;
35: end function

```

---

---

## 4.3 Limitations

Libra is still in its development phase and a public network is only available for basic Libra functionality. While publishing user-defined modules is possible at its core, it is currently disabled in the publicly available network. We discovered some limitations that comes with the use of Libra and Move programming language. These limitations are listed as follows:

**Limitation with data structure** There are limitation to how multiple values can be stored in a Libra resource's data structure. Move programming language does not support data types such as *Hashmap* that can be used to read values in  $O(1)$  time complexity. It employs *Vector* types to store multiple values. Hence, operations such as signing or verifying a digest needs to loop through the vector to find the provided value. This increases the time complexity of such implementations to  $O(N)$  limiting performance in blockchain application.

**Limitation with updating a published Libra module** During the time of development of the thesis project, once a module is published on Libra network, there are no easy options to update it. The only way this can be achieved is through a hard fork in the blockchain. A workaround on updating module is through depending on modules but the operations are limited to defined procedures in the original module. So, in case a larger update is required in the module then a new module has to be published under a different module name. This limits the ability to update a published module and procedure definition within it.

**Limitation with decentralized resources** Libra's architecture defines distributed resources at its core as resources are stored with accounts and not the modules that define them. There are cases where a global repository of data are needed for global verification. For example, operations such as assigning and revoking credentials are issuer-specific operations and the registered values are only known to the issuer and the holder who receives them. Moreover, as the corresponding digest values are generated from the off-chain layer the system cannot guarantee its global uniqueness unless there is a global data structure that tracks all the registered digests. This adds storage overhead and results in a slightly expensive query.

---

## 4.4 Extensions

In the following section, we present some extensions that can be applied to the proposed Libra solution in the above text. These extensions are specific to Libra and are presented according to the logical structuring of modules and resources in the system.

### 4.4.1 Permissioned issuer consortium

The presented Libra module defines public procedure for registration as a root issuer. This allows any account in the network to invoke this method to create an issuer's resource. Even though this won't allow the issuer account to create resources for holder accounts or issue credentials without the holders claiming them, an adversary might still invest efforts in imitating an issuing authority to gain the trust of holders. If holders fail to recognize an authority's address from the adversary it is possible for an adversary to present false credentials and the holder's to claim it.

While this can be prevented if a holder is aware of the issuer to which they register into and similarly for the verifiers, where they know addresses of authoritative entities. Another approach could also be to define a trusted group of root issuers. Then, only the members of this group will be allowed to register as a root issuer in the application as presented in section 3.2.1. We refer to this trusted group of issuers as the issuer consortium. This extension can be adopted if issuers are known and trusted among each other.

For this implementation, we propose a module, *IssuersConsortium*, that creates a membership resource *IssuerMembership* in an issuer's account if they are part of the issuer consortium. The resource holds issuer's public key related to the Consortium membership while a private key has to be kept safe with the issuer. The module defines a public procedure *joinIssuerConsortium* that any address in the Libra network can invoke and request to become a member of the consortium. A submitted request has a validity period of  $t$  in which other members can vote for the inclusion of the new node in the system. If in case consensus of votes are collected, the proposed address and it's public key is registered into *IssuerMembership* resource and assigned to the new member. The new member can now use their private key to sign and vote for inclusion of members address in the consortium.



Following this extension requires issuers to be trusted, functional and unbiased towards any other members in the consortium.

#### 4.4.2 Unique credential digest

Credential digests are values that are created off-chain and registered in the chain through an issuer. A benefit that comes with Libra architecture is that the data created by modules lives within Libra accounts. While this data organization leverages some use cases such as holder's proof of association and certification as defined in section 3.1.2, it becomes expensive to perform an operation that has global ordering.

The proposed application doesn't guarantee uniqueness of digest registered into the system because of the association of resource data with Libra account and absence of knowledge regarding digests registered throughout the system. To provide this guarantee, we must be able to track all the digests registered into the system.

A digest  $d_0$ , registered through a root issuer  $I_0$ , for a holder  $H_0$ , can also be registered through another issuer,  $I_1$  for a different holder  $H_1$  in the absence of global knowledge about issued credential digests. We propose a data-structure( $D$ ) that holds all credential digests( $N$ ) registered in the system as  $D = \{d_0, d_1, d_2, \dots, d_N\}$ . This data structure lives within the module owner's account and is referred from the digest registration procedure to validate the uniqueness of the registered digest.

### 4.5 Use-case Analysis

The solutions design discussed in previous sections exemplifies certification scenario in an educational institution. In this section, we explore a different use-case scenario where the proposed framework is adopted to a hypothetical driving licence certification scenario.

### 4.5.1 Driving Licence Certification

A driving licence is a type of credential that usually has validity for a given time period. Such time-framed credentials can be represented as units of a timed credential into the system.

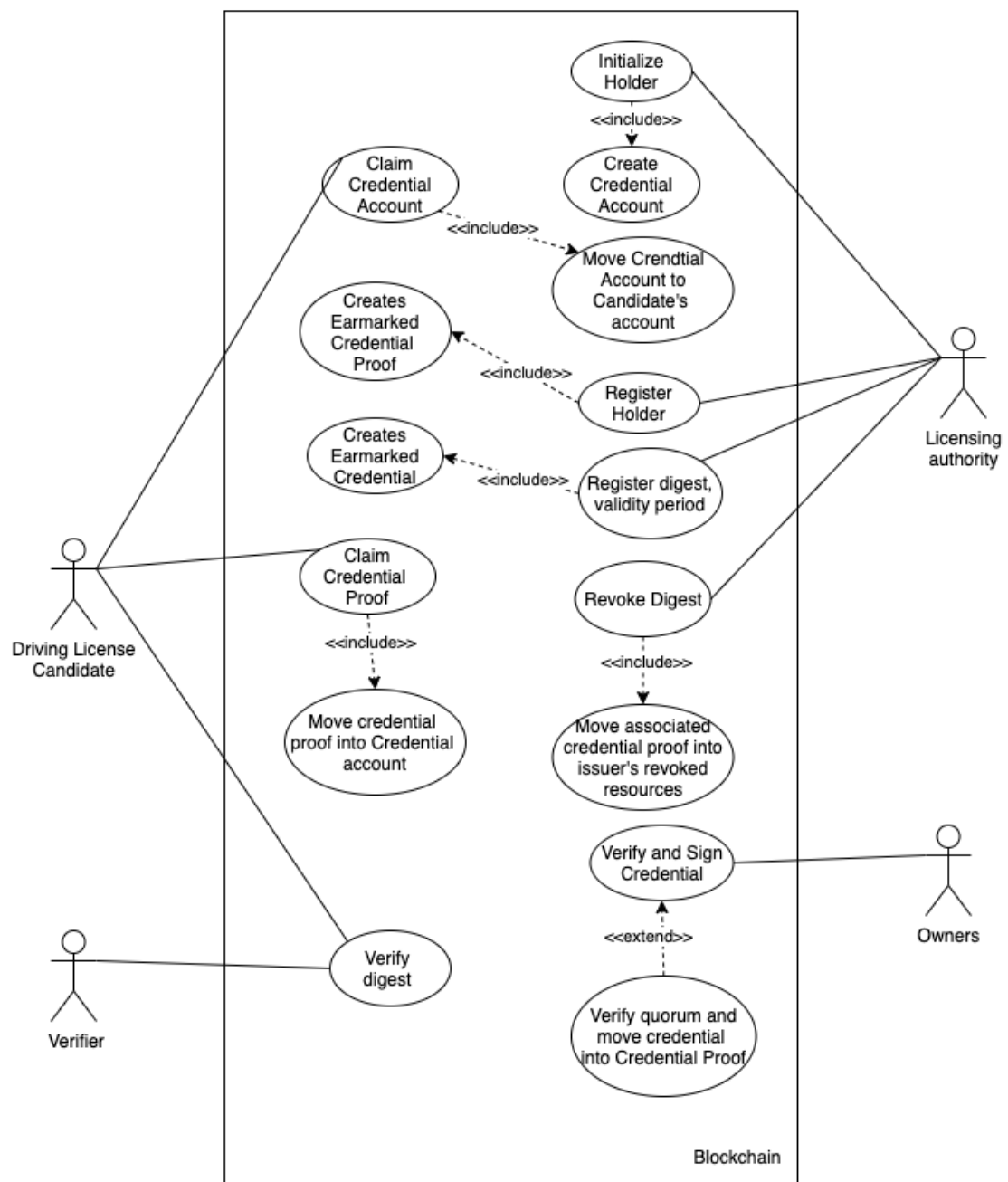
Starting with the registration of the licensing authority, we can represent such authority as the root issuer, while different licensing categories as sub issuers in the system. By licensing categories, we refer to licensing for 2-wheelers, heavy vehicles, etc. This structure allows registering results from different tests(written, driving test, etc) as credentials and aggregate them as credential proof representing the completion of exams in a licensing category.

Moreover, creating sub-issuer allows separating owner roles defined by the authoritative requirements of the issuer. For an example, an authoritative in charge of a written test might not hold the same position for driving exams. Hence, dividing these exams into different sub-issuers leverages the difference in authority in real-world organizational structure.

When a licensing authority registers a candidate as a holder, they create an entry in earmarked credential account. This credential account entry has to be claimed by the candidate to reflect their enrolment with this particular licensing authority. This process, in turn, verifies that both the parties, i.e the candidate and the licensing authority trust each other. Furthermore, sub-issuers can only register credentials that writes to the credential account created by the root issuer.

Candidate's registration with sub issuer registers a credential proof for the holder. The credential proof at this point is saved as an earmarked resource and not directly into the credential account because we distinguish credential account to possess certification that is verified by all stakeholders and claimed by the candidate adding a level of trust to an issued credential from the student's perspective.

As the candidate completes an examination off the chain, we assume that the achievement is recorded off-chain and a hash to the particular credential is recorded into the system along with the period of validity as earmarked credential. A credential earmarked by an issuer is similar to credential proof and will not be immediately associated with the student's credential account. When a credential is earmarked, it adds trusts from the issuer side but a credential needs to be trusted by a consensus, a defined number of owners, by verifying the digest in the credential.



**Figure 4.10:** A use-case diagram for driving licensing and verification

After a successful off-chain verification of the digest, course owners can sign the earmarked credential. Once a quorum of signatures is collected for the credential, it is moved into its respective credential proof container.

A credential proof holding different credentials proves that the credentials defined within it has been issued and verified by the course owners. After completion

of all the exams, issuers aggregate the credentials within the credential proof. This information can finally be claimed by the holder moving it from the issuer's resource into the credential account. This process of claiming a credential proof is a validation from the student side to confirm that the assigned credentials are valid from their perspective. The information about the claimed resource and its holder is recorded into the issuer's resource. This is later useful when verifying validity and association of a credential digest with the candidate.

If a credential once issued is to be revoked by the issuer, then digest of the credential and credential proof is recorded as issuer's resource and the credential proof that is present in the student's credential account is moved into the issuer's revoked credential proofs resource.

From this use case analysis, we believe that the proposed system can also be adopted into different certification scenarios due to flexibility added through issuer organization to adapt to different organizational structure.

# Chapter 5

## Experimental Evaluation

We have proposed and built a blockchain application that supports authenticating and verifying credentials. In this chapter, we evaluate the cost of different user functionalities and evaluate Libra's performance metrics in handling transactions with the developed modules.

### 5.1 Experimental Setup

Experiments in this thesis project was carried out on a Local Area Network(LAN) at the University of Stavanger. A LAN is characterised by low latency network as the connected computers in the network are limited in a small area. We refer to each individual computers as a node and jointly refer them as a cluster. We employ Kubernetes[11] to manage, deploy and scale nodes in this test cluster.

Building a distributed systems infrastructure from scratch is a time consuming task. Kubernetes provides an interface to define such infrastructure as code which allows easy and maintainable deployments, scalability, fault-tolerance and provisioning of nodes in a cluster. This code defines the infrastructure and can be stored in a configuration file and it can be defined using YAML definition or in the JSON format. This formally defined configuration file provides a declarative input to Kubernetes API that parses the stored information and configures nodes as defined in the declarations. Kubernetes further relies on *Docker* and docker images to run nodes in a cluster. A docker image can be understood as a set of read-only instructions that allows creating operational nodes in the network.

Infrastructure configuration file defines the source of the docker images and allows provisioning the deployed nodes. We use YAML definitions in configuration file and a define shell scripts that dynamically invokes Kubernetes API to start a defined number of nodes in the network. These configurations and deployment scripts are made available as a part of the thesis source code repository[12].

**Table 5.1:** Specifications for different nodes in the experiment cluster.

Nodes	Number of nodes	Node type
Validator	4	Core Libra nodes
Prometheus	1	Monitoring node
Grafana	1	Visualizing node
Client	1	Test node

As shown in table 5.1 and figure 5.1, we deploy four validators, two monitoring and a test client in the cluster.

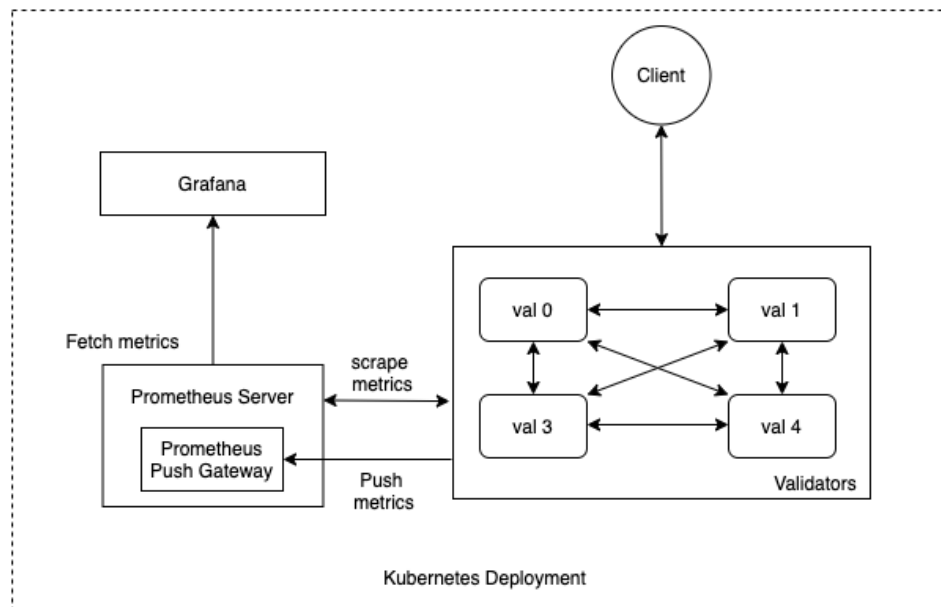
Running a distributed system, presents unique challenges in node monitoring as data from all the nodes needs to be monitored to insure their proper working. For the same, we use *Prometheus* to collect different metrics from validator nodes and *Grafana* for aggregated monitoring using of these metrics. Prometheus maintains a time-series database and either scrapes the validator nodes for default metrics or we push custom counter variables to measure performance of each node over a given period of time.

Similarly, the test client is used to publish the developed modules and to trigger transaction script executions that invokes different procedures in the published modules. While we tracked number of transactions, gas usage and latency through the test client, the monitoring services also provided an user-friendly interface to validate our findings which were compared throughout the processes of gathering test results.

All nodes in the cluster run on different physical machines and collectively form experimental Libra network. As recommended by Libra[17], validator nodes in the cluster are high provisioned with 12 CPUs and 32GiB memory each.

## Application Configurations

Libra supports using SHA-2 256-bits and SHA-3 256-bits encryption standards. For the experimentation, we use SHA3-256 bit encryption to perform hash operations.



**Figure 5.1:** Overview of test setup deployed using Kubernetes that consists of four validators, Prometheus, Grafana and a test client node.

Similarly, all the credentials introduced into the experimental network are of 256-bits in length.

### 5.1.1 Libra Network

The Libra Network in the experimental configuration is made up of validators and monitoring nodes. We created docker images[51] for validator nodes with an ability to publish custom user modules. Similarly, docker images for monitoring tools are open-source and were easily integrated into the developed solution. We define operation of different nodes in the libra network as follows:

**Validators** In the network, each validator is a single physical machine and are collectively responsible to carry out consensus using LBFT protocol through their access controlled ports. The test client submits transactions to one of these validators. Validator nodes run the open-source Libra code with customized functionalities that allows publishing modules and running arbitrary transaction script which are both currently disabled in the publicly available test Libra network.

The definition of Kubernetes configuration file further allows updating core validator configuration, such as, transactions per block and seed value for access control using environment variable definition. This allows easier update and test of the validators in the test network.

**Monitoring and visualizing tools** Monitoring tools provides the fastest way to verify a working network and helps to verify different test scenarios. *Prometheus* node is responsible to collect data from validators and the *Grafana* service helps to plot real-time metrics.

**Prometheus and push gateway** As we submit requests to the validators, it is equally important to collect metrics from different nodes in the distributed architecture to assert its correct functioning. Thus, to handle the same, we use Prometheus. Each validator in the network is configured to emit different count based logs through an endpoint that Prometheus can scrape to record them. Similarly, custom counter logs are recorded into Prometheus using the Prometheus push gateway where each validator sends requests to the gateway service that logs the submitted values.

**Grafana** The Grafana tool provides a user-friendly interface to aggregate and visualize different metrics recorded from the *Prometheus* monitoring tool. It further provides an interface that allows running complex rate based operations on the collected metrics.

### 5.1.2 Test Client

A test client was built using the Rust programming language. This test service submits multi-threaded asynchronous transaction requests on the developed application to all validator nodes. We use this client to submit transactions and gather metrics on gas usage, system throughput and latency of the developed solution. We present an operational overview of the test client in figure 5.2. The test client runs the following operation in order:





**Build requests per organization** Before any transactions are submitted to the blockchain, we build an ordered list of transactions that defines the creation of test organization structure, register, sign, revoke and verify credentials.

**Start workers to send requests** The goal of the test client is to test the performance of the developed module in the Libra blockchain network. Hence, the ordered list of transactions per organization from the previous step is asynchronously submitted to all validators in the network. The test client then creates several worker threads per validator instance and sends the request built in the previous step to Libra validators.

## 5.2 Experimental Results

Libra runs its transactions as a metered transaction. Each transaction thus entails a fees based on the processing and storage that it occupies. With the Libra network and test client setup, we tested gas usage of each operations in the system. Similarly, we also evaluated how Libra would perform under high number of parallel transactions that invokes custom user defined modules.

### 5.2.1 Gas usage

After deploying Libra modules in the test setup, we used compiled Libra transaction scripts to introduce transactions in the system. The gas usage on these functions were profiled and are presented in the following section.

**Table 5.3:** Holder procedures gas usage

Procedure	Gas Usage
Claim credential account	46484 gas
Claim credential proof	73620 gas for credential proof with 2 credentials, 209112 gas for credential proof with 5 credentials

**Table 5.4:** Owner procedures gas usage

Procedure	Gas Usage
Sign credential	61073 gas

**Table 5.5:** Verifier procedures gas usage

Procedure	Gas Usage
Verify credential	18812 gas
Verify credential proof	58237 gas for credential proof with 2 credentials 101715 gas for credential proof with 5 credentials

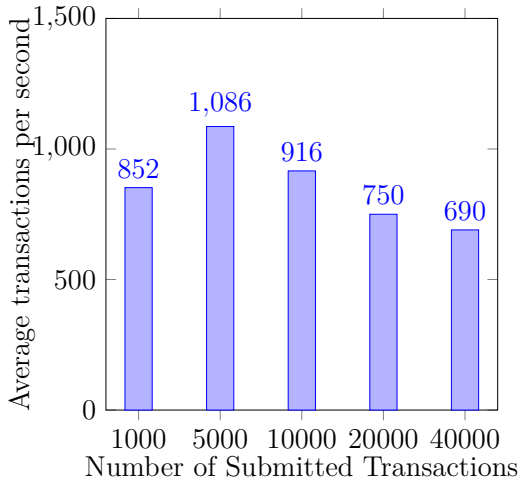
**Table 5.2:** Issuer procedures gas usage

Procedure	Gas Usage
Issuer registration	118838 gas when invoked with 2 owners 135465 gas when invoked with 4 owners
Sub-issuer registration	147008 gas when invoked with 2 owners 182587 gas when invoked with 4 owners
Holder registration	58382 gas when invoked as a root-issuer (creates earmarked credential account), 52573 gas when invoked as a sub-issuer (creates earmarked credential proof)
Credential Registration	28527 gas
Credential Aggregation	45474 gas when aggregating with 2 credentials 87906 gas when aggregating with 5 credentials
Credential Revocation	32185 gas

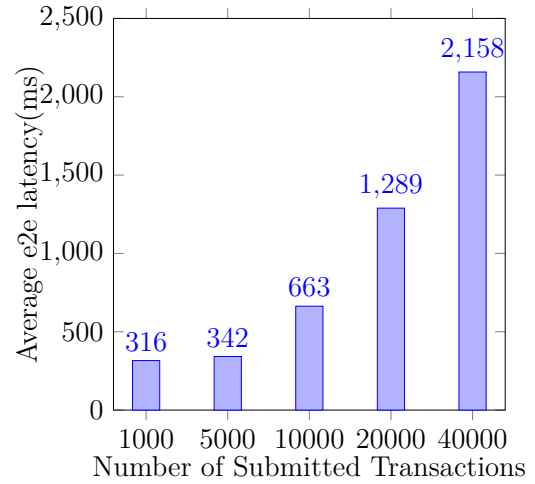
## 5.2.2 Performance Evaluation

For this evaluation, we have four validator nodes in the network and a test client that is responsible to spawn worker thread. Each thread is then responsible to build transaction requests responsible to create issuers, sub-issuers, owners, courses, students and credentials specific to an organization. These requests are accumulated in an ordered list before they are sent to the blockchain network for processing.

To track a transaction status, we track the sequence number and poll the blockchain network to check for the updated sequence number in a defined period. Any account that doesn't reach the expected sequence number is considered to have expired transactions, and we ignore those transactions in the result computation. The transactions we submit is based on the compiled scripts and each represent an individual task in the application's workflow.



**Figure 5.3:** Number of Submitted Transactions Vs Average Transaction(per sec)



**Figure 5.4:** Number of Submitted Transactions Vs Average e2e latency (ms)

In our experiment, we have  $W$  workers and each worker is assigned a defined number of accounts that they can use to create transactions in the system. These workers build and send transactions asynchronously, and we track the number of transactions as  $txns_{wi}$ , time taken for all requests to be submitted and response to be gathered as  $time_{wi}$ . Then, we calculate the average transaction per second per worker as  $tpsw$  and an average transactions per second as  $tps$ .

$$tpsw = \frac{txns_{wi}}{time_{wi}} \quad (5.1a)$$

$$tps = \frac{\sum_{i=1}^W tpsw_i}{W} \quad (5.1b)$$

Similarly, end-to-end(e2e) latency of transactions is based on total time taken to process them. As all the requests are sent asynchronously, we track submitted transactions based on their sequence number. For an Libra account  $A$  with sequence number  $n$ , if  $X$  transactions are submitted then the new expected sequence number will be  $n + X$ . Hence, the total time taken for transactions submitted by the account is the difference between the time at the start of transaction submission and the time at which the expected sequence number is reached. Mathematically,

$$e2e \text{ latency} = \text{Transaction end time} - \text{Transaction start time} \quad (5.2)$$

We compute an average of this e2e latency over all transaction to get the average e2e latency. All the evaluation results presented in the following sections are averaged results based on the five different experiments.

### 5.3 Discussion

Based on the results presented in experimental results, findings show that creating issuer entity costs the most in-terms of gas usage. Though this cost varies based on number of owners defined as the part of an issuer, the cost is most probably high because of different resource created for the issuers during this process. Similarly, the gas usage in verification and aggregation of credential proof depends on number of credentials to be processed within them.

Fees of a transaction is the product of gas used and gas price. As we performed these tests, the Libra association is working on defining gas pricing rules in the network thus a transaction fee cannot be defined during this period. But, the gas cost can be used as a measure to estimate the associated fees with different workflows as a transaction fee is given by the formula,

$$\text{Transaction fee} = \text{gas used} \times \text{gas price} \quad (5.3)$$

Discussing the performance metrics from experimental Libra network, we see that Libra can process about 800-900 transactions per second at approximately

300-500ms latency with four validators. But, the latency seems to deteriorate with increasing number of transactions in the system. Libra plans to start with approximately 100 validators in its initial stages [24] but we believe that the throughput of the system might further decrease than our findings because with the growth in number of validators in the consensus protocol, the throughput and latency is impacted [25].

The developed solution provides security features similar to that of *Cerberus* [3]. In its core, adaptability with different digital certification standards and flexibility in defining organizational structure makes the developed solution much more adaptable in different certification scenarios than the works presented in section 2.4. It allows different stakeholders within an issuing authority to participate and collectively certify a holder. This helps in mitigating threats related with corrupt staffs in a certifying institution.

As we approach certification from a holder's perspective, the developed credential account mimics a real-world scenario where holders always hold their own credentials. Credentials are individually equipped with validity period and owner signatures that validates authenticity of data present in the credential. This allows a holder to selectively present credentials to verifiers with limited information. It aids in user's privacy as minimal information is provided for verification. If in case a credential is revoked, then, it can be easily detected as all revocations are maintained on-chain and referenced during verification. Credential accounts and earmarked credential proofs can also be used to present a proof of holder's association with an issuing authority.

# Chapter 6

## Conclusion and Future Directions

This chapter presents conclusion for this thesis and proposes direction for future work.

### 6.1 Conclusion

The main objective of the thesis was to design and develop a blockchain application that addresses the need of transparency in organizational procedures while different forms of digital credentials are issued and verified through the platform. This has been completed and an operational blockchain solution has been developed.

The developed application works on top of Libra blockchain and effectively implements credential issuance by involving all involved stakeholders in the process. *Issuers* earmark credential for its eventual holder, *owners* at the issuing authority verify and validate the earmarked credential and *holders* claim the earmarked resource after a verification from their side. This earmarking approach to resource registration and claiming such resources are not just application-defined but enforced through the Libra blockchain system. It guarantees that any critical action performed in moving resources from one account to another is agreed by all involved parties and governed by the Libra architecture.

While we issue a credential thorough on-chain reference, the actual credential is stored off-chain. This unique fingerprint generated from the off-chain storage is registered as part of an on-chain credential resource. As actions on the registered credentials are performed and recorded in the blockchain, it adds transparency in

the organizational procedures. If any changes are made to credentials published off-chain, the tamper-evident property of the such storage updates its fingerprint thereby disassociating the updated credential from the one referenced in the blockchain. This guarantees a tamper-evident link between a credential stored off-chain and the credential referenced in the blockchain. Moreover, this approach of separating off-chain storage from on-chain verification is the key to the system's adaptability towards different standards for digital credential. This also best uses a blockchain platform by limiting data transferred in the system and thereby improving the throughput and limiting costs in the developed application.

A credential issued to a holder is stored in their credential account. Move language safety constraint prevents any modification to the stored information and allows easy read-only access to the holders and guarantees immutability of credentials stored with the holder. Moreover, holders can safely store these credentials in offline applications and present minimal information to verifiers through selective disclosure. The credential account also verifies a holder's association with an issuing authority which can be used as proof of association in an organizational setup where this information is valuable to the holder.

Furthermore, the proposed extension for permissioned issuer registration adds initial trust to any issuer registered in the system. If any fraudulent actions by a permissioned issuer is encountered, then the issuer in the discussion can have their access revoked through consensus voting, leading to the revocation of its issued credentials. This extension can be applied only if the majority of issuers perform righteous actions in the permissioned group.

Other key advantages of the proposed solution includes credential revocation and timed credential which can be adopted in multiple certification scenarios and primarily addresses the needs of time-critical certification systems such as driving licences or government-issued certificates such as passports.

From a security point of view, detecting counterfeit credential is convenient as all credentials issued through blockchain applications are stored on-chain and are checked during verification. So, any attempt in presenting counterfeit credential will fail. Similarly, as the solution involves multiple entities for credential issuance, if at least a single entity of an issuing authority perform correct actions, frauds can be easily detected.



---

## 6.2 Future Directions

This section presents ideas on improvements to the proposed solution in this thesis. We define them as follows:

**Zero Knowledge proof** Selective disclosure helps in presenting minimal information to verify credentials in the system. An interesting implementation would be to use zero-knowledge proofs in the blockchain system for verification of credentials. With zero-knowledge proofs, we believe that holders wouldn't have to present any information about credential to entities verifying the credential.

**Planning Monetary Overhead** While the earmarking approach for credential issuance build trust from all involved parties in the system, it also simultaneously increases cost in the system as each operation is executed as an independent transaction. If we can plan a method for cost distribution among all the entities then that might give an added benefit for system adaptation.

**Hierarchical Issuer Structure** Issuer structuring in the system is hierarchical. This restricts several actions to be initiated by the root issuer, for an example, a credential account entry can only be created by the root issuer requiring all issuers to register with the root issuer of an organization first. We can distribute such capabilities among all issuers in an organization which will help in minimizing the number of transactions by decoupling transaction from root issuer to all issuers.

**Revocation and Updating Credentials** Revocation in the proposed solution removes all associated resource into revoked resources. It might be convenient to revoke a single credential resource in scrutiny instead of all associated resource as revoked resource. In this case, we can remove the revoked credential and other associated resources can be moved back into issuer's earmarked resources. This will also allow issuers to update credentials in the same workflow by revoking the credential to be updated and introducing new credential as an update. As all credentials registered into the system has to go through the same workflow, the new registered credential has to be approved by all stakeholders

# List of Figures

2.1	Example of ledger state defining how modules and resources are embedded as values of the Libra addresses key. . . . .	11
2.2	Workflow in Libra protocol . . . . .	12
2.3	Processing transaction in Libra . . . . .	16
3.1	An example of educational organizational structure and its translation into different system entities . . . . .	22
3.2	High-level overview of interaction between system entities, off-chain components and the blockchain . . . . .	23
3.3	An example of issuer earmarking three credential hash representing certification of three courses into the system as three different credential . . . . .	26
3.4	Quorum requirement for verification of Earmarked Credential(EC) . . . . .	27
3.5	Translation of Earmarked Credentials into Credentials . . . . .	27
4.1	Logical representation of entity interactions with the Earmarked-Proofs Module . . . . .	33
4.2	Issuer interaction with the Issuer module . . . . .	34
4.3	Verifier interaction with the Issuer module . . . . .	35
4.4	Issuer Modules and Resources . . . . .	36
4.5	Overview of issuer resources and their data structure . . . . .	37
4.6	Holder Modules and Resources . . . . .	39
4.7	Overview of holder resources and their data structure . . . . .	40
4.8	Logical organization of credentials and credential proofs in a credential account . . . . .	41
4.9	A timeline representation of credential registration, signing and claim . . . . .	44
4.10	A use-case diagram for driving licensing and verification . . . . .	52
5.1	Overview of test setup deployed using Kubernetes that consists of four validators, Prometheus, Grafana and a test client node. . . . .	56
5.2	Overview of test client workflow . . . . .	58
5.3	Number of Submitted Transactions Vs Average Transaction(per sec) . . . . .	61
5.4	Number of Submitted Transactions Vs Average e2e latency(ms) . . . . .	61

# List of Tables

2.1	Comparison of functionalities and supported certifications of different certification-verification solutions . . . . .	19
5.1	Specifications for different nodes in the experiment cluster. . . . .	55
5.3	Holder procedures gas usage . . . . .	60
5.4	Owner procedures gas usage . . . . .	60
5.5	Verifier procedures gas usage . . . . .	60
5.2	Issuer procedures gas usage . . . . .	60
A.1	Number of Transaction Vs Number of transaction(per second) . . . .	70
A.2	Number of Transaction Vs Average Transaction latency(ms) . . . .	70

# List of Algorithms

4.1	Registering a sub-issuer . . . . .	42
4.2	Initiating as sub-issuer . . . . .	42
4.3	Registering Holder with Sub-issuer . . . . .	43
4.4	Sign Earmarked Credential . . . . .	46
4.5	Holder Claim for earmarked Credential Proof . . . . .	46
4.6	Verification of digest . . . . .	47

# Appendix A

## Experimental Data

This appendix contains the complete experimental data for average throughput and e2e latency of transactions in Chapter 5. The results have been rounded to nearest whole number.

**Table A.1:** Number of Transaction Vs Number of transaction(per second)

Number of transactions.	Number of transaction(per second)				
	Exp. 1	Exp. 2	Exp. 3	Exp. 4	Exp. 5
1000	929	922	948	912	870
5000	1024	960	1253	1039	1156
10000	893	787	890	934	759
20000	717	709	678	754	892
40000	688	672	703	710	680

**Table A.2:** Number of Transaction Vs Average Transaction latency(ms)

Number of transactions.	Average Transaction Latency(ms)				
	Exp. 1	Exp. 2	Exp. 3	Exp. 4	Exp. 5
1000	296	305	292	287	301
5000	312	356	359	300	384
10000	663	654	672	659	667
20000	1053	1583	1374	1287	1149
40000	2504	2387	1958	1893	2152

# Appendix B

## Attachments

This appendix contains instruction to run tests on a LAN network with the developed test module with rust programming language.

- Components of libra network are containerized and published as docker images in docker hub. These containerized components allows users to publish custom module and transaction script in the network.
  - Libra Validator image : <https://hub.docker.com/r/pariwesh/thesis>
  - The combination of stable docker images by their version number are as follows:
    - \* Validator image: `pariwesh/thesis:libra_validator_dynamic-2.0.1`
    - \* Safety rules image: `pariwesh/thesis:libra_safety_rules-2.0.0`
    - \* Initialization container image : `pariwesh/thesis:libra_init-2.0.0`
- Kubernetes configuration file to start the custom libra network is available as shell scripts.
  - Repository : <https://github.com/pariweshsubedi/thesis-project/tree/bbchain-network-2.0.0/kube/libra>
- Implementation of testing framework created using Rust is available as part of github repository.
  - Repository : <https://github.com/pariweshsubedi/libra-bbchain-port/tree/master/testsuite/bbchain-test/src>

# Bibliography

- [1] Digital certificates project. <http://certificates.media.mit.edu/>. Accessed: 2020-03-02.
- [2] Gilles Grolleau, Tarik Lakhal, and Naoufel Mzoughi. An introduction to the economics of fake degrees. *Journal of Economic Issues*, 42(3):673–693, 2008.
- [3] Aamna Tariq, Hina Binte Haq, and Syed Taha Ali. Cerberus: A blockchain-based accreditation and degree verification system. *arXiv preprint arXiv:1912.06812*, 2019.
- [4] Meng Han, Lei Li, Ying Xie, Jinbao Wang, Zhuojun Duan, Ji Li, and Mingyuan Yan. Cognitive approach for location privacy protection. *IEEE Access*, 6: 13466–13477, 2018.
- [5] Liyuan Liu, Meng Han, Yan Wang, and Yiyun Zhou. Understanding data breach: A visualization aspect. In *International Conference on Wireless Algorithms, Systems, and Applications*, pages 883–892. Springer, 2018.
- [6] Karl Wüst and Arthur Gervais. Do you need a blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54. IEEE, 2018.
- [7] Bbchain | trustworthy distributed document verification system. <http://bbchain.no>. Accessed: 2020-01-10.
- [8] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.
- [9] Libra developer tools. <https://github.com/pariweshsubedi/libra-developer-tools>, . Accessed: 2020-05-20.

- 
- [10] Libra developer community: Push\_back on a vector with other vectors. <https://community.libra.org/t/push-back-on-a-vector-with-other-vectors/2706>, . Accessed: 2020-05-20.
- [11] Kubernetes. <https://kubernetes.io/>. Accessed: 2020-03-02.
- [12] Libra for bbchain github repo. <https://github.com/pariweshsubedi/thesis-project/tree/bbchain-network-2.0.0/kube>, . Accessed: 2020-03-02.
- [13] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.
- [14] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Consensus in the age of blockchains. *arXiv preprint arXiv:1711.03936*, 2017.
- [15] Michael Crosby, Pradan Pattanayak, Sanjeev Verma, Vignesh Kalyanaraman, et al. Blockchain technology: Beyond bitcoin. *Applied Innovation*, 2(6-10):71, 2016.
- [16] Christian Cachin et al. Architecture of the hyperledger blockchain fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers*, volume 310, page 4, 2016.
- [17] Zachary Amsden, Ramnik Arora, Shehar Bano, Mathieu Baudet, Sam Blackshear, Abhay Bothra, George Cabrera, Christian Catalini, Konstantinos Chalkias, and Evan Cheng. The libra blockchain. 2019. URL <https://developers.libra.org/docs/assets/papers/the-libra-blockchain/2019-09-26.pdf>.
- [18] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.
- [19] Arati Baliga. Understanding blockchain consensus models. *Persistent*, 2017 (4):1–14, 2017.
- [20] Matteo Benetton, Giovanni Compiani, and Adair Morse. Cryptomining: Local evidence from china and the us. Technical report, Working paper, 2019.



- 
- [21] Proof of stake. <https://eth.wiki/en/concepts/proof-of-stake-faqs>.
- [22] Fahad Saleh. Blockchain without waste: Proof-of-stake. *Available at SSRN 3183935*, 2020.
- [23] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.
- [24] Libra: The path forward. <https://developers.libra.org/blog/2019/06/18/the-path-forward>. Accessed: 2020-01-25.
- [25] Jiashuo Zhang, Jianbo Gao, Zhenhao Wu, Wentian Yan, Qize Wo, Qingshan Li, and Zhong Chen. Performance analysis of the libra blockchain: An experimental study. In *2019 2nd International Conference on Hot Information-Centric Networking (HotICN)*, pages 77–83. IEEE, 2019.
- [26] Getting started with move. <https://developers.libra.org/docs/move-overview>, .
- [27] Life of a transaction - libra. <https://developers.libra.org/docs/life-of-a-transaction>, .
- [28] Virtual machine. <https://developers.libra.org/docs/crates/vm>, .
- [29] Tomasz Hyla and Jerzy Pejaś. Long-term verification of signatures based on a blockchain. *Computers & Electrical Engineering*, 81:106523, 2020.
- [30] Thomas Hepp, Matthew Sharinghousen, Philip Ehret, Alexander Schoenhals, and Bela Gipp. On-chain vs. off-chain storage for supply-and blockchain integration. *it-Information Technology*, 60(5-6):283–291, 2018.
- [31] Henri Massias, X Serret Avila, and J-J Quisquater. Design of a secure timestamping service with minimal trust requirement. In *the 20th Symposium on Information Theory in the Benelux*. Citeseer, 1999.
- [32] Jeremy Clark and Aleksander Essex. Commitcoin: Carbon dating commitments with bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 390–398. Springer, 2012.
- [33] Angelos Stavrou and Jeffrey Voas. Verified time. *Computer*, 50(3):78–82, 2017.

- [34] P Todd. Opentimestamps: Scalable, trust-minimized, distributed timestamping with bitcoin. *Peter Todd [Internet]*, 15, 2016.
- [35] Andrea Brandoli. Blockchain notarization: extensions to the opentimestamps protocol. 2019.
- [36] Blockcerts : The open standard for blockchain credentials. <https://www.blockcerts.org/>. Accessed: 2020-01-25.
- [37] Joao Santos. Hypercerts: A non-siloed blockchain-based certification service. 2017.
- [38] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.
- [39] Meng Han, Zhigang Li, Jing He, Dalei Wu, Ying Xie, and Asif Baba. A novel blockchain-based education records verification solution. In *Proceedings of the 19th Annual SIG Conference on Information Technology Education*, pages 178–183, 2018.
- [40] Meet truerec by sap. <https://news.sap.com/2017/07/meet-truerec-by-sap-trusted-digital-credentials-powered-by-blockchain/>. Accessed: 2020-03-02.
- [41] Academic certificates on the blockchain. <https://digitalcurrency.unic.ac.cy/free-introductory-mooc/self-verifiable-certificates-on-the-bitcoin-blockchain/academic-certificates-on-the-blockchain/>. Accessed: 2020-01-27.
- [42] Jerinas Gresch, Bruno Rodrigues, Eder Scheid, Salil S Kanhere, and Burkhard Stiller. The proposal of a blockchain-based architecture for transparent certificate handling. In *International Conference on Business Information Systems*, pages 185–196. Springer, 2018.
- [43] India readies its biggest deep tech bet yet: a upi-like blockchain platform. <https://factordaily.com/india-readies-upi-like-blockchain-platform/>. Accessed: 2020-02-20.
- [44] University consortium set up to authenticate degrees using blockchain technology. <https://www.nst.com.my/news/nation/2018/11/429615/university-consortium-set-authenticate-degrees-using-blockchain>. Accessed: 2020-02-20.

- 
- [45] M. Sporny, D. C. Burnett, D. Longley, and G. Kellogg. Verifiable credentials data model 1.0. <https://w3c.github.io/vc-data-model/>. Accessed: 2020-01-10.
- [46] Mozilla open badges. <https://wiki.mozilla.org/Badges>, . Accessed: 2020-01-12.
- [47] Moodle: Openbadges user documentation. [https://docs.moodle.org/dev/OpenBadges\\_User\\_Documentation](https://docs.moodle.org/dev/OpenBadges_User_Documentation), . Accessed: 2020-01-13.
- [48] Blackboard - open badges in higher education. <https://sites.google.com/site/openbadgesinhighereducation/blackboard>, . Accessed: 2020-01-13.
- [49] Canvabadges. [https://www.edu-apps.org/edu\\_apps/index.html?tool=canvabadges](https://www.edu-apps.org/edu_apps/index.html?tool=canvabadges), . Accessed: 2020-01-13.
- [50] Mozilla backpack is now badgr backpack. <https://backpack.openbadges.org.badgr.io/>, . Accessed: 2020-01-13.
- [51] Libra for bbchain docker hub. <https://hub.docker.com/repository/docker/pariwesh/thesis>. Accessed: 2020-03-02.