Blockchains for Coldchains (BC4CC)

An Innosuisse – Swiss Innovation Agency (formerly Commission for Technology and Innovation, CTI) Project, Grant Agreement No. 26996.1 PFES-ED

Final Report

The BC4CC Consortium

Universität Zürich UZH, Communication Systems Group CSG, Zürich, Switzerland
modum.io AG, Zürich, Switzerland

For more information on this document or the BC4CC project, please contact:

Prof. Dr. Burkhard Stiller
Universität Zürich, CSG@IfI
Binzmühlestrasse 14
CH—8050 Zürich
Switzerland

Phone: +41 44 635 4569
Fax: +41 44 635 6809
E-mail: stiller@ifi.uzh.ch

Document Control

Title: Final Report

Type: Public

Editor(s): Burkhard Stiller, Bruno Rodrigues, Eder Scheid, Geetha Parangi

E-mail: stiller@ifi.uzh.ch, rodrigues@ifi.uzh.ch, scheid@ifi.uzh.ch, parangi@ifi.uzh.ch

Author(s): (in alphabetical order) Timo Hegnauer, Alexander Hofmann, Andreas Knecht, Daniel Lacik, Geetha Parangi, Bruno Rodrigues, Eder Scheid, Burkhard Stiller, Tim Strasser, Sacha Uhlmann, Patrick Widmer

Doc ID: BC4CC-Final-Report-v3.docx

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
V0.1	July 18, 2019	Burkhard Stiller	Template and structure
V0.2	September 19, 2019	Geetha Parangi	Economic Analysis
V0.3	September 25, 2019	Eder Scheid	Initial table of content and organization
V0.4	September 29, 2019	Burkhard Stiller	Updates and suggestions
V0.5	November 4, 2019	Eder Scheid	Accepted Bruno's review. Reorganization, and preliminary text input references
V0.6	November 28, 2019	Eder Scheid	Added the main technical (Framework, Policy-based Selection, and Interoperability API)
V0.7	December 11, 2019	Bruno Rodrigues	Technical Basis (Background on BC, DLT, and PBNM)
V0.8	December 14, 2019	Eder Scheid	Review of Economic Evaluation, organizational issues and additions of security analysis and thesis contents.
V0.9	February 19, 2020	Sacha Uhlmann and Lisa Dössegger	Modum input
V1.0	February 22, 2020	Eder Scheid, Burkhard Stiller	Added Missing sections and appendix, fixed references, tables, and figures

Legal Notices

Table of Content

1 Executive Summary

This final report of the BC4CC project presents all developments as well as detailed and overall results of the prototypical design and evaluation of a Blockchain (BC)-based approach toward providing the interoperability between BCs and allowing users to specify their requirements in form of policies, regarding the usage of the BC that will be used to store their data. With such an approach in place, the need for detailed technical knowledge to select a dedicated BC is reduced and the interoperability achieved reduces the vendor lock-in into a single BC. Thus, the BC4CC approach allows for a flexible and dynamic BC interaction.

2 Introduction

Blockchains (BC) and Distributed Ledger Technology (DLT) have been experiencing a fast development in recent years due the extensive media attention revolving crypto-currencies and the deployment of the technology in different areas besides the financial domain, such as for supply-chains [1], Internet-of-Things (IoT) [2], cybersecurity [3], and many others [4]. This fast development led to a spawn of startup companies implementing different BC solutions to address issues of particular industry niches. For instance, considering only the supply-chain sector, there are several startups (*e.g.,* modum.io [5] – the BC4CC implementation partner, Everledger [6], Provenance [7]) as well as companies (*e.g.,* IBM [8], Microsoft [9], and Oracle [10]) providing BC-based solutions for increasing transparency and traceability.

The availability of many different solutions for the same application area creates, as a natural consequence, a dispersion of solutions used in the market, thus, a new problem is introduced to decide not only which BC-based solution is appropriate [11], but also whether this solution is interoperable with other solutions in the market [12]. This issue becomes evident considering that different BCs have different characteristics and different usage scenarios. For example, one BC design may favor data privacy over higher transaction rates, whereas another design guarantees faster transactions rates, but no data privacy. In this context, there is a non-trivial trade-off that has to be considered when deciding which BC to use.

Making BCs interoperable is becoming a key to connect independent BC networks [13]. For example, interoperability would mean that one could send Bitcoins and another person would be able to receive an equivalent amount in Ethers without the need for a third party, such as an exchange. Alternatively, it is possible to imagine a hospital, which has its medical records on its BC interacting with the social security BC to validate the identity of a patient. Furthermore, users would be able to access a wide range of features natively of each chain, without the need to download large files for each BC that they might want to use. Further, in the supply-chain context, interoperability means that different stakeholders (*e.g.,* producer, logistic, and retailer) are able to interact with different BCs without the need to change their business logic. However, enabling such a cross-chain communication is not a straightforward task. For instance, although Bitcoin [14] and Ethereum [15] are cryptocurrencies, they have different characteristics and are implemented in different programming languages, which turns the simple task of sending Bitcoins and receiving Ethers into a complex task.

2.1 Project Goals

The two main goals were proposed to be achieved at the conclusion of the BC4CC project: (a) an BC-agnostic approach for applications and (b) open interfaces for interoperable BCs. They involve technical challenges and present measurable innovations in these areas, such as BC interoperability and data management in the BC. The first goal (described in Section 2.1.1) relates to the creation of an BC interoperability Application Programming Interface (API), and the second goal (described in Section 2.1.2) concerns the development of a BC-agnostic approach where users are able to transparently interact and select the most suitable BC based on policies.

2.1.1 Open API for IoT Data in the BC

The goal is to create an API that can be used by newly onboard customers without specific BC knowledge or understanding. The API requirements from potential partners need to be collected and based on these, a selection of the most promising requirements needs to be chosen. These requirements need to align with the modum.io infrastructure. For example, if the API for the temperature storage requires an InterPlanetary File System (IPFS) storage location, then it should be supported by the Smart Contract as well. Quantifiable goals include:

- Open API developed with successfully tested with 2 Swiss customers,
- 1 security related white-paper,
- 1 scientific peer-reviewed paper (*e.g.,* BCC 2017, IEEE BC Summit),
- API documentation, and,
- Measurable business impact with 2000 transactions per day with one customer by end of 2018.

2.1.2 A BC-agnostic Approach

Modum.io selected Ethereum (the second largest BC in terms of market capitalization as of December 2019 [16]) for their current system because of its large developer base, cutting-edge technology, strong growth, high security and availability. However, unforeseen developments in the BC space could, nevertheless, have negative effects on modum.io's business and their customers, who could be potentially locked into a rapidly evolving system with worldwide, divergent stakeholders. modum.io's future customers are expected to be diverse with respect to their requirements, *e.g.,* performance, regulatory compliance, and volumes. BC agnosticism could permit modum.io and their customers various commercial options to select their optimal BC, and provide the potential technical capabilities for migration. In particular, transaction speed, readiness to store proof-of-existence type data (for immutability and auditability) and low mining fees are particularly sensitive to a commercial success of modum.io. Novel extensions of public BCs such as RSK and Lightning might offer smaller block times, more data processing/storage, or significantly lower mining fees, which could be a decisive factor customer onboarding and, therefore, crucial towards modum.io's success.

In this sense, a BC-agnostic approach would allow modum.io's customers to select, through high-level abstractions of these low-level BC characteristics, the ideal BC for their business. Quantifiable goals of such system include:

- Implementation of the modum.io system including data migration to at least 1 other BC,
- Evaluation of its technical impact and customer feedback,
- Acquisition (Pilot Study) of at least 1 customer on an alternative implementation, and
- Peer reviewed scientific papers (*e.g.,* BCC 2017, IEEE BC Summit).

2.2 Document Outline

This document defines seven major sections, in which all key aspects, technologies, design choices, evaluations, and achievements are discussed.

Section 3 details the fundamental background to the technologies and concepts employed to achieve the goals of the project, *e.g.,* BC and Policy-based Management (PBM), and presents market and economic basis to support the industry view.

Then, the main section of the this report, Section 4, presents the design and the Proof-of-Concept (PoC) implementation of the two solutions (*i.e.,* Policy-based Blockchain Agnostic Framework and the Modular Blockchain Interoperability API) that address the report's goals.

Consolidating industry and academy, together, Section 5 details a case study of the integration of the BC4CC project with modum.io's system.

Section 6 and discusses the role of BC in the industry and offers details regarding the conducted security, technical, and feasibility evaluations on the prototype.

Finally, Section 7 summarizes the project with two views, one from view from the industry standpoint and another from an academic standpoint. Further, Section 7 lists in an overview the project outcomes and presents future work directions.

Besides references, abbreviations, the lists of tables and figures, a set of two Annexes refer to relevant details of selected areas of work performed, which are available as separate PDF documents.

3 Fundamental Set-up and Background

This chapter discusses the theoretical and practical background on which this project is built on and its relevance for the industry. It mainly focuses on three aspects that are highly important for the development of BC4CC, namely the technical basis, the market basis and the economic basis.

3.1 The Industry Relevance of BC4CC

Generally, there is a lack of BC adoption in industries. While the topic of supply-chains has been identified as an important use case for BC technology, especially due to the possibility of document and persist relevant supply-chain details with time stamps in an immutable manner, actual productive BC-based systems remain a rarity. This is also true for temperature-controlled supply chains, despite its well-known advantage of improving transparency and trust. Although there is a certain resistance against changes and the adoption of new, early technology in general, two factors are especially problematic for the industry to adopt BCs: *(i)* transaction fees and *(ii)* solution lock-in.

Firstly, transaction fees accompanying every BC transaction can amount to quite a large sum, which depends on the BC used and the current value of this BC's coins. In addition to the implementation and maintenance costs of the BC, this can become quite expensive for a company. This is discussed in more detail in Section 6.5 of this report.

Secondly, companies fear a lock-in effect on one specific technology, if they choose to adopt a particular technology, in this case the BC, to build their entire solution upon. As a result, their system might not be interoperable with other systems, which do not use this technology. This can cause integration concerns when working with other companies, even if both companies are using BC technology since those can have quite different characteristics. Therefore, many companies do not want to commit to a dedicated BC in apprehension of choosing a technology that will not become the standard.

BC4CC did set the goal to provide means and mechanisms to overcome those disadvantages by providing the user control over how and dependent on which factors a user wants to use which BC; therefore, allowing users to easily adopt the new approach without risking to be locked-in or to spend more than required to benefit from the technology. Thus, the project offers a newly developed Open API to customers, suited for the upload of sensor data, and provides a BC-agnostic framework for the storage of data that can be configured based on customer-defined policies, which guide the selection of the most suitable BC on a case-to-case basis taking into consideration BC-related data, such as transactions costs, performance, and regulations. The BC-Agnostic Framework enables the automatic selection of the right BC based on predetermined preferences, while the flexible Open API allows for an easy integration of the system. BC4CC paves the way for the industry to move toward a network-based approach, where data integrity is a given, network-based applications are supported, and data can be shared not only between two individual actors of the supply chain, but also within logistics networks.

The supply-chain industry is highly competitive. Therefore, it is not only critical to show the potential business value of such a system, but also making sure that integration costs are limited. As with all new technologies, expertise in organizations is still limited.

Thus, providing an abstraction layer to interact with the new technology is further critical to promote BC adaptation. At the same time, certain organizations might require using a specific BC, might it be due to contractual or technical reasons.

BC4CC tackles all of these factors in one approach by enabling users to define their own cost thresholds and manage their spending individually as well as choosing their preferred BC. Thus, they are able to reduce costs of an BC application significantly. By building an API, the system becomes considerably more compatible and interoperable as well as user-friendly. By introducing BC-agnosticism, the BC-Agnostic Framework improves the scalability of the business case itself, because it facilitates the acquisition of new customers and the entry into new market segments. BC4CC being BC-agnostic does not only enable customers to choose their custom-made BC solutions, including life-cycle options such as onboarding, mapping of current processes and data migration, but also gives them control over the transaction costs specific to the different BCs. Therefore, with the help of the newly developed BC4CC mechanisms, logistics providers can build any chain solutions, such as cold chains or supply-chains that cater exactly to their needs and tackle their individual challenges. The industry is hence able to adopt the BC technology even with very little knowledge and more importantly, at very little cost.

3.2 Fields of Application

Until temperature-sensitive products (*e.g.,* drugs) reach their final destination, they must be handled by different stakeholders involved in the cold-chain supply-chain. These stakeholders are considered during the development of the proposed framework, because they have different points of views and requirements.

Each one of these stakeholders has different requirements that impact on the choice of the BC, resulting in different policies. For example, companies follow specific standards and regulations in their manufacturing line. Therefore, recording in the BCs that the goods left the company under such standards is crucial to avoid any disagreement or SLA violations. What impact on the BC choice for such a stakeholder is not the cost involved, but rather how fast can the records (*e.g.,* temperature sensor readings) be included in the BC and how secure is the BC implementation (*e.g.,* a private BC). Thus, a *maximum cost* parameter in the policy should be set to the highest possible so that the framework will store these records as fast as possible.

However, in the context of logistic services, the requirement that impacts the most is how much it will cost to append a new record to the BC, independent of the time that it will take. This is because the conditions (*e.g.,* temperature and humidity) within the storage compartment of vehicles must continuously be monitored to ensure that temperature-sensitive drugs produced were not affected. This active monitoring produces a considerable amount of BCs transaction. Thus, reducing the cost of data storage becomes crucial to this stakeholder. In this sense, it must define a policy with the *maximum cost* parameter set to the lowest possible value.

In the next items, it is described a list (non-exhaustive) of such stakeholders and their relation with the proposed framework.

- **Companies:** Producers (*e.g.,* pharmaceutical companies) follow specific standards and regulations in their manufacturing line. Therefore, recording in the BC that the goods left the company under such standards is crucial to avoid any disagreement or Service Level Agreement (SLA) violations.
- **Logistic Services**: The goods and their conditions (*e.g.,* temperature and humidity) within the storage compartment of vehicles must continuously be monitored to ensure that temperature-sensitive drugs were not affected during transport. Once the logistics service (*e.g.,* the Swiss Post in Switzerland) has received the good, the service can start to monitor its temperature using the required sensor.
- **Final Destination**: Wholesalers and pre-wholesalers distribute and maintain a massive volume of goods to smaller companies, such as pharmacies. Each good must be verified to check if it was transported following under regulations. If there was a single deviation during the transport, then the drug cannot be sold to end-customers.
- **Government:** Audits the monitored data to verify if the logistic services are being compliant with the current regulations. The BC technology is compelling to the government because it simplifies the auditing process by assuring that the stored data cannot be tampered.
- **BCs**: Although BCs are not stakeholders *per se*, they support all the interaction among stakeholders, providing data immutability and trust without the need of a Trusted Third Party (TTP). Thus, they were included in the stakeholders list. Examples of BC implementations are Bitcoin, Ethereum, and MultiChain.

3.3 Technical Basis

This section describes the two fundamental concepts that support the novelty and the development of the framework herein described, *(i)* Blockchain (BC) and Distributed Ledgers, and *(ii)* Policy-based Management (PBM). The novel BC and Distributed Ledger concepts are described in Section 3.3.1 and the consolidated PBM concept is described in Section 3.3.2.

3.3.1 Blockchains and Distributed Ledgers

In its purest form, a Blockchain (BC) acts as a decentralized and public ledger that transparently and immutably records blocks of transactions across a network of computers based using a consensus algorithm. Therefore, a BC is, as originally proposed, open to all its participants with respect to the rights of reading, writing and participation in the consensus mechanism. A Linked-List (LL), however, is a data structure that is traditionally managed by one or more trusted entities holding the write permissions. Thus, based on the process to compose new blocks of information (*i.e.,* the consensus mechanism), as well as the guarantees of immutability and transparency, it can be said that although the final outcome of a BC and LL is similar, the way in which these structures are composed is completely different.

From an abstract point-of-view, a BC resembles a LL (*cf.* Figure 1), which is an abstract data structure whose instances are logically interconnected by pointers. Thus, on a BC, transactions are stored in the form of a LL, sequentially ordering the blocks (of transactions). In a LL representation, blocks represent the nodes of a LL. However, the

resemblances of LLs and BCs end there, since the process of composing a chain is entirely different. For instance, the major differences lie on the processes of gathering information from the peer-to-peer network, the assembling of information (*i.e.,* transactions) into such abstract data structure (*i.e.,* blocks), as well as appending new blocks to the list of blocks (*i.e.,* consensus).

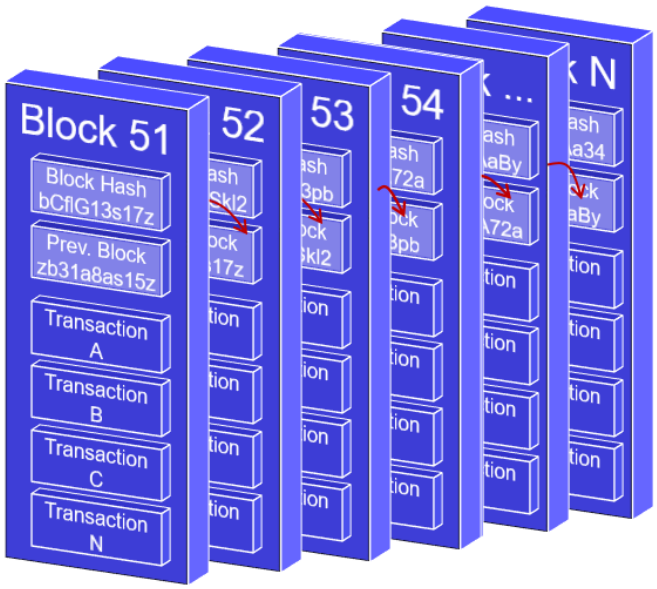

Figure 1 – Blockchain Example

Since every node has a copy of the list of blocks, a single main chain must be guaranteed, which is the responsibility of the consensus mechanism. Every node needs to verify the sequence of the blocks or transactions. Depending on the kind of consensus mechanism, the correct sequence of the main chain is chosen by the majority of nodes with the same result, or the node who solved a complex mathematical problem. Either way, once a correct sequence has been found, the changed state of the blockchain is broadcast to every participant in the network. Due to this, BC is suitable as a system for parties, who do not trust each other to have a trustworthy exchange.

Basically, a BC works similar to a traditional database. With the exception, that anything that is put into a BC cannot be deleted or changed, and, therefore, allows for permanent storage of information. Every node in the BC network has a full replication of the BC, traditional databases have a master/slave relationship, resulting in the slave database to synchronize to the master database. The consensus of a BC is being determined by the majority of the nodes agreeing on an outcome, whereas in the traditional way the transactions are simply distributed amongst participants. In a BC every peer can validate a transaction.

The capacity of BC to provide a trustworthy, decentralized, and publicly available data storage makes it an interesting opportunity for organizations to increase business agility and reduce costs by removing intermediaries in distributed applications. However, it is important to note that a BC can be implemented in different ways, being typically named Distributed Ledger (DL), by modifying permissions to read and write, as well as the

participation in the block-validation process. In this regard, a BCs are essentially public with respect to (i) read, (ii) write, and (iii) consensus participation, whereas DLs are named based on any modification of these parameters. Moreover, the different types (*cf.* Figure 2) can be classified according to varying read and write permissions:

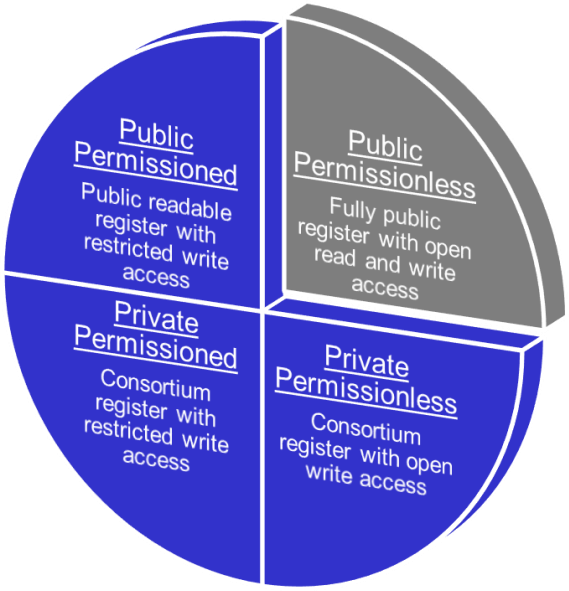

Figure 2: Deployment Types

- **Public Permissionless** BCs are the most prevalent type of BCs. Bitcoin [14] and Ethereum [15], and most forks are considered public permissionless BCs, because of its read and write permissions, as well as the participation in the consensus, are open to anyone with Internet access. Thus, public permissionless BCs are the standard type of BC deployment and most cryptocurrencies are implemented as such.
- **Public Permissioned** write permissions are restricted to selected entities, but anyone is able to read from the BC. For example, this deployment type can be used for use-cases where multiple trusted authorities want to publish public data, accessible to anyone (*e.g.,* publishing hashes of academic certificates).
- **Private Permissioned** BCs offer a trust model resembling traditional databases, where the read and write permissions are restricted and consequently data can only be read by authorized parties. Restricting the permissions creates a hierarchy between its participants (*e.g.,* role-based actions) where the main features of BC (*e.g.,* transparency, immutability, decentralization) may not make sense for a potential application.
- **Private Permissionless** BCs are comparable to public permissionless, but the notion of the reading access control is restricted to a certain group or community. Therefore, the writing and reading permissions are open to all participating members of this private group. A dedicated supply chain BC would be a possible example, where the exchanged information is only readable by its authorized members, but all members can issue transactions without boundaries.

Therefore, depending on the needs of the application domain, the inherent power of disintermediation can increase of trust through transparency among the stakeholders involved. Nonetheless, while BC's have started its widespread adoption within the FinTech domain, many other application areas, use cases, and specific BC types are emerging. However, it is important to observe that the BC applicability relies on a multitude of different facets, which are usually determined by dedicated application needs in terms of performance, security, and scalability, which have to be carefully considered in a long term analysis.

3.3.2 Policy-based Management (PBM)

As the complexity for managing networks and distributed systems increase, the use of policies to enforce several aspects such as Quality-of-Service (QoS) or control access became more necessary. Policies can be used to regulate the operation of different aspects of a BC-based system, including the coordination of which BC to use based on well-defined conditions determined in advance. In this regard, Policy-Based Network Management (PBNM) is a well-known approach that can comprise either QoS, access control, or sustainability-oriented policies, helping to address such management issues. PBNM presents several benefits, such as less manual countermeasures and errors, automated analysis and verification based on a formal foundation, dynamic inspection and adaption at runtime, without demanding changes in the underlying implementation. Nonetheless, PBNM is not yet widely commercially used. Its widespread use depends on an automated policies refinement process in order to translate high-level policies into machine-readable policies able to put the business strategies in practice.

A policy is a set of rules used to manage and control access to a set of ICT resources and services. According to [17] [18], a policy has three parts: an event, which triggers a specific rule; a condition, which contains the logic that defines if the action part is going to be run or not; an action, which defines what needs to be performed. This structure is called Event-Condition-Action (ECA), the general syntax which is "on event if condition then action".

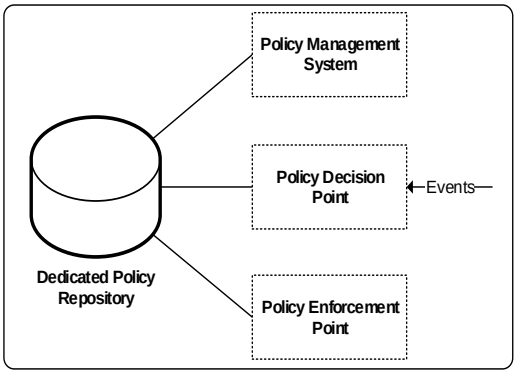

Figure 3: Policy-Based Network Management Basic Architecture

The general architecture for a PBNM system was described by IETF (Internet Engineering Task Force) [19] and is depicted in Figure 3. Policies are created, modified and stored by the Policy Management System (PMS); searched and retrieved by the Policy Decision Point (PDP); and enforced in by the Policy Enforcement Point (PEP).

3.4 Market Basis

As discussed in Section 3.2. of this report, various different stakeholders are involved in the cold chain distribution of medical products. In order to assess the scope of potential users and the actual need for a solution such as BC4CC, it is crucial to have an overview over the existing service providers in the industry. In Switzerland, there are around 100 pharmaceutical producers with a total annual transport volume of around 150'000 shipments. modum.io AG has conducted pilot studies with Schaer Pharma AG, Acino International AG, and Amino AG (total transport volume of 10'000 palettes a year) in 2016 and has a productive solution with Swiss Post since May 2019. The total of shipments per year and per customer is presented in Table 1.

Table 1: Total Transport Volume per Pharmeceutical Producer in 2016

Customer	**Total Shipments / Year**
Amino AG	5’220
Schaer Pharma	2’000
Acino Pharma	2’000
Total	**9’220**

The Swiss wholesaler segment is less fragmented, as it can be seen in Table 2. Five players split the volume of 5 million yearly shipments: Galexis, Voigt, Pharmafocus, Amedis, and Zur Rose. modum.io AG worked intensively with the pre-wholesalers Voigt and Voigt Industry Service which handle more than 1 million shipments per year, which is about 25% market share. modum.io AG has conducted successful pilots with Voigt, Provet, Amino, Schaer Pharma, Acino Pharma, Toppharm, Cavapro, Blutspende Zürich, which have a combined shipping volume of of 2.5 million shipments per year.

Table 2: Total Transport Volume per Swiss Wholesaler in 2016

Customer	**Total Shipments / Year**
Provet AG	104'400
Voigt Industrie Services	292'320
Voigt AG	1'252'800
Zur Rose Grossist (CH)	480'240
Zur Rose Versand (CH)	39'150
Amedis	208'800
Pharmafocus	208'800
Galexis	1'252'800
Total	**4’152’510**

The logistics service market can be split into international providers such as Panalpina, Kühne und Nagel, DPD, DHL, UPS and national providers like the Swiss Post and Galliker, and many of smaller companies providing more specialized services. modum.io AG has run pilots with Post, and DPD. The Swiss Post is considering an active collaboration with an expected shipping volume of 3 million medical goods per year.

3.5 Economic Basis

BCs are an element of distributed systems, in this sense, BC-supported business should be aware of potential cyber-attacks, such as the ones described in [20], which directly impact in the economics of the business, *e.g.,* reputational harm, financial losses, and service disruption. Thus, these companies should perform risk analysis, cybersecurity planning, definition of counter measures, and general best practices in security, which include personnel training and the acquisition of protection services [21]. However, the main goal of the project is to store data in the BC; thus, the economy basis focuses on transactions costs and fees.

In the BC4CC project, the goal is to develop methods for supporting the cold chain distribution process (supply chain) of medical drugs using BC technology. The commercial goal is to reduce the cost of delivery, while assuring a regulation-compliant temperature tracking using IoT-sensors. In this regard, Modum.io AG developed a BC-based solution to offer a passive monitoring solution, to improve efficiency, security and transparency in the pharmaceutical supply chain. Since the goods in the supply-chain follow a vast variety of complex processes, different needs for the BC technology arise [4]. Since each BC technology incorporates different protocols and technologies, their native cryptocurrency, the information about their state or events cannot be exchanged directly between two BCs. The BC interoperability is necessary to connect different BCs, exchanging information and assets.

As part of the BC4CC project requirement, a solution is developed to store and retrieve data on different BCs. The solution employs notary scheme, to allow connectivity to different BCs. The solution incorporates eight adapters to popular BC implementations, including Bitcoin, Ethereum, EOS, Hyperledger, Multichain, Stellar, Peercoin and IOTA. The solution introduces interoperability API implementation in the form of Python API, allowing users to store, retrieve, and migrate data on multiple BCs without the knowledge of their language and implementations.

3.5.1 Factors Affecting Transaction Fee

The BC transaction fee depend on several factors including network congestion, transaction size, transaction confirmation time, cryptocurrency value, the type of BC (public or private), and the number of operations performed in a transaction as shown in Figure 4. In all the public BCs the transaction fee is paid by the sender initiating the transaction.

3.5.1.1 Network Congestion

If the arrival rate of potential transactions is low, then the transactions without fees or low fees attached are written to the BC. However, if the arrival rate of potential transactions increases, the equilibrium shifts and only transactions with higher fees attached are written to the BC [22], [23].

3.5.1.2 Transaction Size

The BC allows a few kilobytes of data to be stored. However, if the user wants to store the larger files, than they need to split the files into small chunks of data and send it in each transaction, in such cases, the user has to pay the base price for each transaction that can be expensive. Moreover, the bigger the transaction size in terms of bytes, the more the transaction fee the sender needs to pay.

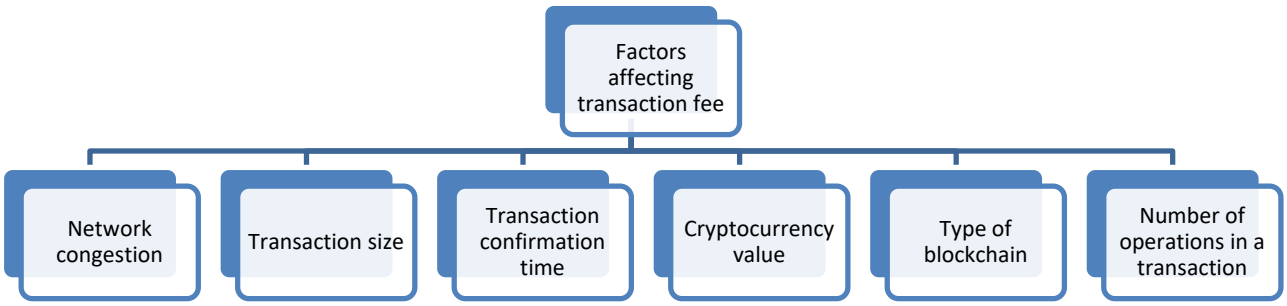

Figure 4: Factors Affecting Transaction Fee

3.5.1.3 Transaction Confirmation Time

The transaction fee paid by the sender affect the confirmation time. The transaction without fees or small fees are given low priority and likely to wait longer for confirmation. In the BC which employs PoW (Proof-of-Work) consensus mechanism, the sender creates the transaction and include a fee which are received by the miner who creates a block that stores the transaction. If the transaction contains no fees, then there is no incentive for miners to include the transaction in the block they are creating. Since the BC has the restricted block size limit, which further restricts the number of transactions that the block can hold, miners give higher priority to the transactions with a high fee. Therefore, the transaction confirmation time of transaction with small fee is much longer than those transactions with larger fee [24].

3.5.1.4 Cryptocurrency Value

The value of cryptocurrency effects the transaction fee. Higher the value of cryptocurrency will lead to increase in the conversion rate from cryptocurrency to fiat currency, which will subsequently increase the transaction fee. For example, if the price of 1 Bitcoin equals 10 USD and later it changes to 50 USD, the value of cryptocurrency and its conversion rate will impact the transaction fee. If the value of cryptocurrency increases, subsequently the transaction fee also increases.

3.5.1.5 Type of BC

In public BC, the transaction fees processed by public ledgers are higher than as compared to the private BC where the reduced number of high-processing nodes enables fee effective transactions [25]. In a private BC, the transaction validation is carried by the network creator, as a result there are no transaction fees for transaction validation.

3.5.1.6 Number of Operations in a Transaction

A number of operations can be performed in a given transaction and the transaction fee varies depending on the complexity of the transaction, the higher the complexity the higher is the transaction fee. For example, the Smart Contracts are the most important feature of certain BCs, for example Ethereum. To create the Smart Contract and to execute it, certain amount of gas in the Ethereum BC has to be invested [15].

3.5.2 BC4CC Public BC Transaction Fee Evaluation

Thus, an evaluation of costs is essential to indicate its relevance for a commercial setting. Therefore, for each of the available BC4CC public BCs, the function used to estimate the transaction fee is described in detail.

3.5.2.1 Bitcoin

Bitcoin is the first decentralized electronic cash system introduced by Satoshi Nakamoto to perform the monetary transfers or transactions without relying on central authority or intermediary. It employs PoW consensus mechanism. In the Bitcoin BC, the transactions are estimated to be around 3.3-7 TPS, the block time is approximately 10 minutes and it allows for a data storage of 80 Byte per transaction [26].

In the Bitcoin BC, miners bear the fee of solving the PoW puzzle and all its transaction partners benefit from the security and consistency of the BC. Bitcoin BC supports optional direct payments from transaction partners to the miners in terms of transaction fees [27]. The transaction fees are appended by the transaction partners into their Bitcoin transaction to ensure that their transactions are included in the block that the miner attaches to the BC. By custom, the spender is responsible for paying the defined Bitcoin transaction fees. During the heavy traffic situations, the transactions with higher fee get priority. Therefore, the transaction partners tend to bid up the transaction fees to gain priority over other transaction partners that pay a lower fee. The miner, in addition to the mining rewards, receives the transaction fees corresponding to all transaction in the block.

When the transaction partners initiate the Bitcoin transaction, they send details of the transaction, *e.g.,* the amount of payment, the source account(s) of the payer, the destination account(s) of the payee along with a fee, to a node in the Bitcoin network. This corresponding node will then spread the transaction to other nodes. When the transaction is known within the network, *i.e.,* inside the memory pool, miners are free to select any transactions from the memory pool to put into the new block. However, there is a preference towards the transaction with the highest fee. This creates a market mechanism to find the price of Bitcoin transactions [28]. The most important factor affecting how fast the transaction gets confirmed is the fee rate.

The `estimatesmartfee` function is released in the Bitcoin Core 0.15. The input to this function is the maximum number of blocks in which transaction is expected to be included and the output is the estimate fee value. The `estimatesmartfee` function is used for more intelligent estimates. It estimates the approximate fee per kiloByte in Bitcoin for a transaction to begin confirmation within *n* blocks. This function returns negative value if not enough transactions and blocks have been observed to make an estimate [29].

In this project, the `estimatesmartfee` function is used to estimate the fee per transaction. To calculate the fee per transaction in Bitcoin, the size of the transaction is multiplied with `estimatesmartfee` return value and divided by 1024, since `estimatesmartfee`, approximate fee per kByte in Bitcoin. The size of the transaction is the default transaction size plus the size of the data to be included in the BC. It was considered the default Bitcoin transaction size, *i.e.,* 106 Byte. The equations below describe how the transaction size and the fee to be paid for a transaction in Bitcoin are calculated.

$$TransactionSize = defaultTransactionSize + DataSize$$

$$BitcoinTransactioFee = (TransactionSize \times estimatesmartfeeOutput) \div 1024$$

3.5.2.2 Ethereum

Ethereum is a global, open-source platform for decentralized applications and unlike Bitcoin it is not just focused on providing the cryptocurrency. Ethereum offers Smart Contract capabilities too. The network has a throughput of 15-25 TPS and a block time of approximately 15 second [26]. Ether is the fuel for the Ethereum BC network. Miners validate and execute the transaction and every user has to pay for the computation regardless of transactions succeeds or fails. The transactions such as sending ether from one account to other, creating the Smart Contract or any other activities require computational power and are charged with fees [15].

The Ethereum protocol incorporates a pricing mechanism for the transaction. Each computational step in Ethereum is priced in units of gas and it is measured in *Wei*. The sender specifies the maximum amount of gas that the intended transaction is expected to consume (*i.e.,* the gas limit) and the price the sender wishes to pay per unit of gas (*i.e.,* the gas price). In Ethereum, the financial transfer will consume 21,000 unit of gas as a base fee for any transaction and plus a small unit of gas for attached data around 4 to 68 gas per byte [15].

The gas limit is the maximum number of units of gas the user is willing to spend on a transaction. The user must include enough units of gas to cover the computational resources or else the transaction will fail due to an "*Out of Gas Error*". If the user wants to spend less on a given transaction, he/she can do so lowering the amount for per unit of gas. Depending on the price the user pays for each unit of gas determines how quickly the transaction will be mined, equal to Bitcoin.

To calculate the fee per transaction, the `gasPrice` function and `estimategas` function provided by web3 API is used [30]. The price of a gas unit in *wei* is determined by the market. The function `web3.eth.gasPrice` is used, this function is the read only function and return the current gas price in *wei*. The price is determined by the x latest blocks median gas price [25]. To identify the amount of gas used in a transaction, the `estimateGas` function is used, which executes the transaction and return the amount of gas used.

To determine the fee per transaction in *wei*, the return value of `gasPrice` function and `estimategas` function is multiplied. The expression below returns the value in *wei*. In order to convert the value from *wei* to ether the function `fromWei` is used which converts

the above *wei* value into ether value. The value is returned as a decimal to ensure the precision down to the *wei*.

$$Fee\ per\ transaction = estimateGas \times gasPrice$$

3.5.2.3 EOS

EOS BC label itself as a BC architecture designed to enable vertical and horizontal scaling of decentralized applications [31]. It uses a combination of dPoS and asynchronous Byzantine Fault Tolerance as its consensus mechanism. EOS runs on only 21 nodes that validate and check the new transactions. It has a block time of 0.5 seconds and the maximum throughput of 3996 TPS. EOS enables the storage of 256 Byte of data per transaction [26].

In EOS, there is no gas or other transaction fees. However, when submitting an EOS transaction three EOS resources need to be considered, such as CPU, Network bandwidth, and RAM. CPU and Network bandwidth requires an EOS staking whereas the RAM needs to be purchased. RAM is needed only for the Smart Contract operations [32].

Network bandwidth is measured as the average consumption in bytes over the last 3 days [32]. Network bandwidth is temporarily consumed every time when the transaction is sent and decreases over time returning to 0. When we stake more tokens for net bandwidth, the more we get to use. We can unstake at any time to reclaim the EOS tokens. Similarly, the CPU bandwidth is measured as the average consumption in microseconds over the last 3 days [32]. CPU bandwidth is temporarily consumed every time when the transaction is sent and decreases over time returning to 0. The longer the transaction runs, the more CPU bandwidth it will consume and we can unstake at any time to reclaim the EOS tokens.

To initiate the transaction, the sender of the transaction needs to stake some EOS for CPU and for NET. If the sender stakes more EOS for CPU and NET, the more transactions and data he sends to the EOS BC in a day. The staked EOS value is not lost, but it is locked until we decide to unstake it. For example, if the sender stake 1 EOS for CPU then he/she can use 48.49 ms of CPU per day and if he stakes 1 EOS for NET, he/she can send 132.17 kByte of data per day in MEMOS in transactions.

To calculate the fee per transaction, Net usage fee as well as CPU usage fee should be calculated. As shown in the equation below, the net usage fee per transaction equals the 128 Byte plus the size of the data multiplied by current network price in `eosPerKiBPerDay`. The value of `eosPerKibPerDay` was obtained from [32].

$$Net\ usage\ fee\ per\ transaction = ((128 + size\ of\ data) \times current\ network\ price) \div 1024$$

The estimation the CPU usage for a transaction is not a trivial task. Based on past transactions the CPU usage value ranges from 200 µs to 700 µs. In order to fix a value, it was estimated 350 µs for each transaction. As shown in the below expression, CPU usage fee per transaction equals 350 micro seconds multiplied by current CPU price in EOS/ms/Day. The value of EOS/ms/Day was obtained from [32].

$$CPU\ usage\ fee\ per\ transaction = ((350\mu s) \times current\ cpu\ price) \div 1000$$

The total fee per transaction in EOS equals net usage cost per transaction plus the CPU usage fee per transaction as shown in the below expression.

$$Total\ fee\ per\ transaction\ in\ EOS = Net\ usage\ fee\ per\ transaction + CPU\ usgae\ fee\ per\ transaction$$

3.5.2.4 Stellar

The aim of the Stellar BC is to provide a cost-efficient platform to move money quickly and reliably. It has its own consensus protocol called Stellar consensus protocol and the name of the currency is Lumens (XLM). The throughput of the network is 1000 - 4000 TPS with a block time of around 5 seconds [26]. The Stellar BC requires small fees on transactions and the minimum balances on accounts. The transaction fee is the number of operations the transaction contains multiplied by the base fee which is currently 100 stroops (0.00001 XLM). The number of operations includes create account, payment, path payment, manage buy offer, manage sell offer, create passive sell offer, set options, change trust, allow trust, account merge, inflation, manage data, bump sequence.

$$Transaction\ fee\ per\ transaction = number\ of\ operations \times base\ fee$$

Accounts in the Stellar must maintain the minimum balance of lumens. If any transaction reduces the accounts balance to less than the minimum, then the transaction will be rejected with an insufficient balance error. Base reserve is currently 0.5 XLM and the entries include Trustlines, Offers, Signers, and Data entries. The minimum balance for an account is 2 × base reserve and each additional entry fee the base reserve [33]. The two special values used to calculate fees are a base fee and base reserve.

$$Minimum\ balance = (2 + number\ of\ entries) \times base\ reserve$$

3.5.2.5 Peercoin

Peercoin BC was launched in 2012. Peercoin consensus mechanism is Proof-of-Stake (PoS), where the coin owners have the influence over the network, produce new blocks and secure the network. Unlike Bitcoin, in Peercoin the transaction fees are not voluntarily set by the user. Instead in Peercoin there is a fixed fee per transaction. The static fee in Peercoin is set at 0.01 PPC per kByte. The fixed transaction fee eases the sender to determine how much they need to pay to the network to confirm their transactions. In addition to this, transactions in Peercoin are confirmed in terms of first come first serve [34]. Any small transaction under 1024 Byte pay at least 0.01 PPC to be included in the block and the transaction of 1025 Byte pay at least 0.02 PPC. The fixed price allows every transaction to be included in the very next block [35]. The size of the transaction is the default transaction size plus the size of the data to be included in the BC. We considered the default Peercoin transaction size is about 196 Byte. To calculate the fee per transaction, the size of the transaction multiplied with 0.01 will reveal the transaction fee in PPC.

$$Transaction\ size = default\ transaction\ size + size\ of\ the\ data$$

$$Fee\ per\ transaction\ in\ PPC = (Transaction\ size \times 0.01) \div 1024$$

3.5.3 Relationship Between Transaction Fee Factors

In Section 3.5.1, the various factors impacting the transaction fee in public BC were described. As an example, these factors impact the transaction fee in Bitcoin and Ethereum and they are examined. It should be noted that the values of cryptocurrencies fluctuate; thus, these values may differ. In Bitcoin, the transaction fee depends on transaction size and transaction confirmation time as shown in Table 3 and Figure 5. If the user wants to perform the transaction by including the data size of 111 Byte, depending on how fast the user needs the confirmation, the transaction fee varies. If the user needs the faster confirmation for example less than 20 min, then the transaction fee is higher as compared to late confirmation where the transaction fee would be lower.

Table 3: Relationship between Transaction Size, Fee and Confirmation Time in Bitcoin

Transaction size in bytes	**Transaction Confirmation Time in minutes**								
	<=20 min	**<=40 min**	**<=60 min**	**<=120 min**	**<=240 min**	**<=480 min**	**<=1440 min**	**<=5040 min**	**<=10080 min**
111 Byte	$0.65	$0.535	$0.509	$0.48	$0.39	$0.02	$0.0127	$0.0127	$0.0127
121 Byte	$0.709	$0.583	$0.55	$0.52	$0.43	$0.0277	$0.0138	$0.0138	$0.0138
151 Byte	$0.885	$0.728	$0.69	$0.659	$0.53	$0.0346	$0.017	$0.017	$0.017
166 Byte	$0.973	$0.8	$0.76	$40.72	$0.59	$0.038	$0.019	$0.019	$0.019
186 Byte	$1.09	$0.896	$0.85	$0.811	$0.661	$0.0426	$0.021	$0.021	$0.021

In Ethereum, the transaction fee depends on transaction size and transaction confirmation time as shown in Table 4 and Figure 6. If the user wants to perform the transaction by including the data size of 5 Byte, depending on how fast the user needs the confirmation, the transaction fee will vary. If the user needs the fastest confirmation, then the transaction fee is higher as compared to slowest confirmation where the transaction fee would be lower.

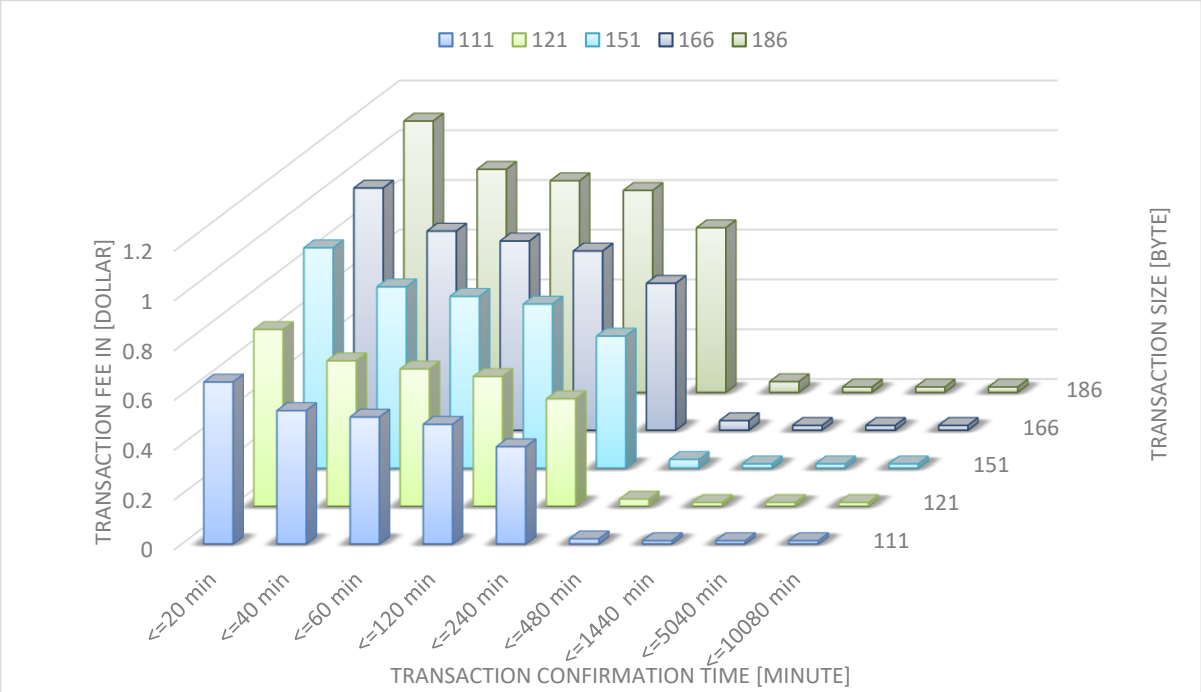

Figure 5: Graphical Representation of Relationship between Transaction Size, Fee, and Confirmation Time in Bitcoin

Table 4: Relationship between Transaction Size, Fee, and Confirmation Time in Ethereum

		Transaction Confirmation Mode					
Transaction Size in Bytes		**Slowest**	**Slow**	**Average**	**Above-Average**	**Faster**	**Fastest**
	5 Byte	$0.00662	$0.01328	$0.06637	$0.13273	$0.1991	$0.33184
	15 Byte	$0.00684	$0.01368	0.06848	$0.13696	$0.20545	$0.34241
	45 Byte	$0.0075	$0.01496	$0.07483	$0.14965	$0.22448	$0.37413
	60 Byte	$0.00781	$0.01561	$0.078	$0.156	$0.234	$0.38999
	80 Byte	$0.00821	$0.01645	$0.08223	$0.16446	$0.24669	$0.41114

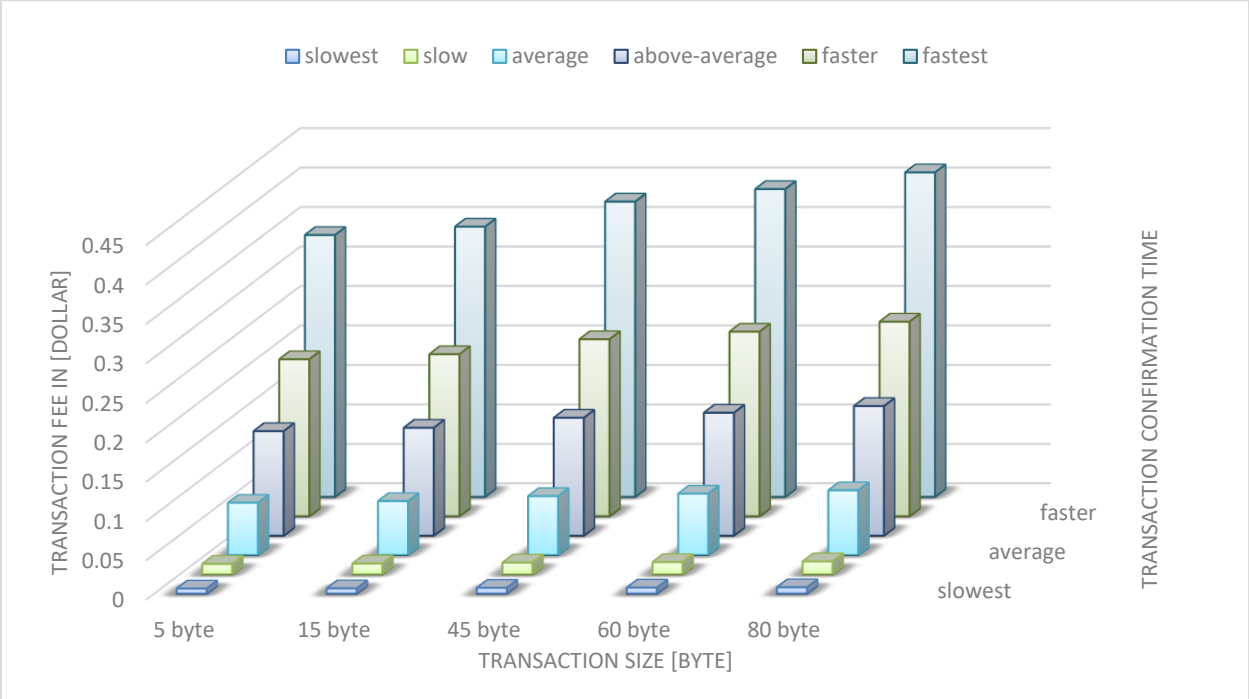

Figure 6: Graphical Representation of Relationship between Transaction Size, Fee and Confirmation Time in Ethereum

4 BC-Agnostic Framework – Design, Methodology, and Prototype

In order to achieve the goals of the BC4CC project, two main components were implemented and act together in the BC-Agnostic Framework, *(i)* the *Policy-based BC Selection Framework* (namely PleBeuS), and (ii) the *Modular BC Interoperability API* (namely Bifröst). These components allow for the performing of the cross-chain communication in an agnostic and transparent way. Thus, the BC Agnostic Framework is not only capable of dealing with BC interoperability, but ease the management of such framework through PBM enabling users and applications to operate over different BCs transparently.

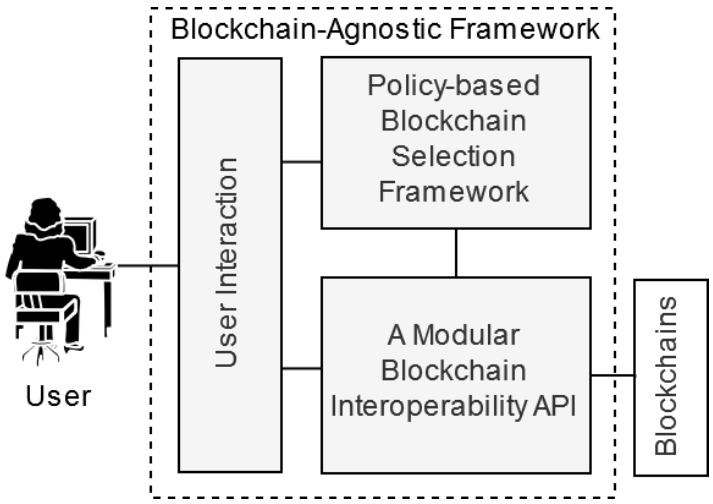

Figure 7: BC-Agnostic Framework Overview and Components

Figure 7 presents the design of the proposed BC-Agnostic Framework. The Policy-based Blockchain Selection Framework (described in Section 4.2) is based on the paradigm of Policy-based Management (PBM) [17], and automatically selects the most suitable BC based on user requirements and sends this information to the Modular Blockchain Interoperability API. This API (described in Section 4.3) is designed as a server-side component based on the notary interoperability approach. In practice, this means that the information sent to it will be further sent as a BC transaction to the selected BC transparently. The *User Interaction* component is responsible for the internal communication between the components and the external communication with the user.

4.1 Supported Blockchains

Table 5 presents the summary of the BC and their characteristics that are currently supported by the BC-agnostic Framework. These values are used for the Policy-based Blockchain Selection Framework to filter the BCs according to user requirements.

Table 5: Summary of BCs and Characteristics

Blockchain	Type	Tps	Block Time [s]	Data	Turing-Complete	Fees
Bitcoin	Public	4 – 7	600	80 Byte	No	Variable
Ethereum	Public	15 – 25	15	46 Kbyte	Yes	Variable
Stellar	Public	1000 – 4000	5	28 Byte	No	Base Fee
EOS	Public	250 – 3996	0.5	256 Byte	Yes	Variable
IOTA	Public	500 – 800	60	1300 Byte	No	None
Hyperledger	Private	Variable	20 (default)	20 Byte	Yes	None
Multichain	Private	Variable	15 (default)	80 Byte	No	None

4.2 Policy-based Blockchain Agnostic Framework

This section presents the newly developed Policy-based Blockchain Selection framework, called PleBeuS, to aid the selection of the most appropriate BC to store data based on user requirements. PleBeuS allows users to represent their requirements in the form of policies which are used as inputs for filtering and selection algorithms. Next sections present the PleBeuS framework, detailing architectural components, goals, policy parameters, and the employed policy switching mechanism and selection algorithms.

4.2.1 Design

The architecture of PleBeuS follows the PBM concept; thus, it is divided in the Policy Management Tool (PMT), the Policy Decision Point (PDP), and the Bifröst API

acting as the Policy Enforcement Point (PEP). Figure 8 depicts the PleBeuS architecture and its components. The components depicted using a dashed line, such as the transaction costs service, are external solutions and were not implemented.

- *API Request Handler*: Allows to configure and manage policies, which are then stored in the database. It is a Web service that interacts with a Graphical User Interface (GUI), which executes REST API calls in the background. The GUI allows users to create new policies, delete, or edit existing policies and presents cost and transaction statistics. The aforementioned statistics are retrieved from the database and served via the REST API. The database contains user information, policies, information about the supported BCs and all the transactions that have been made.
- *Database:* The database stores users, policies, BC information (see Section 4.1), and the transactions that were enforced by the PEP.
- *Policy Selector*: Upon receiving a request to store data in a BC, this component retrieves the active policy that matches with the time frame and the user that sent the request. The retrieved policy is then forwarded to the Blockchain Selector to execute the selection algorithms.

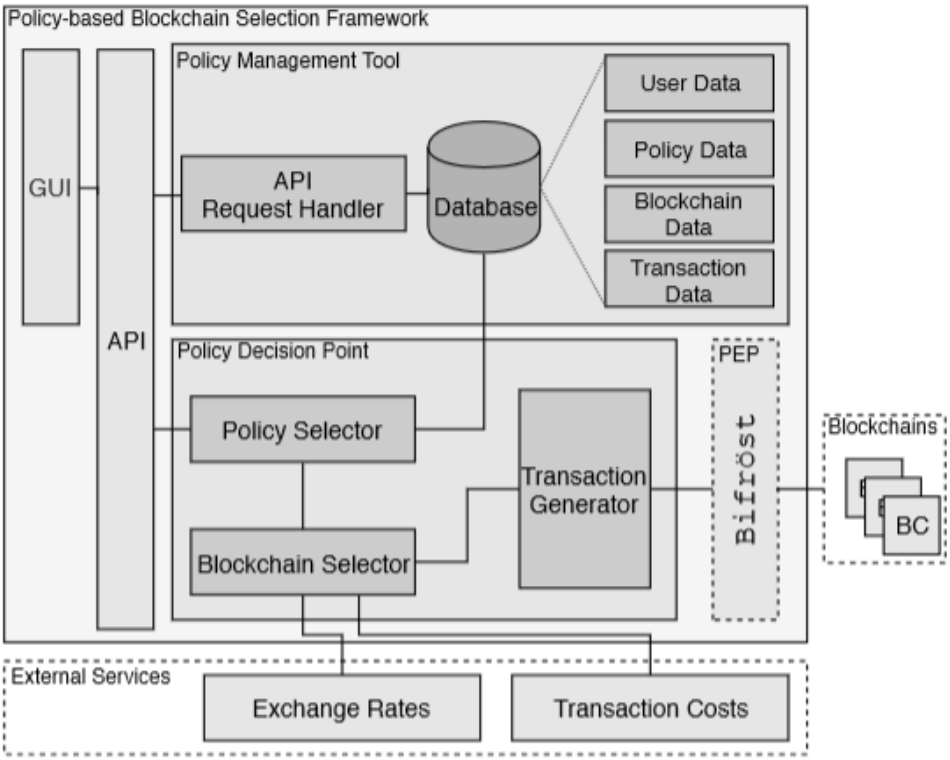

Figure 8: PleBeuS Architecture

- *Blockchain Selector*: This component implements the selection algorithms (cf. Section 4.2.4), being responsible for retrieving the transactions costs for each BC and their exchange rate to calculate the costs thresholds in the currency defined by the user. Moreover, once a BC or more were selected, it sends the selected BC to the *Transaction Generator*. This component is responsible for selecting the BC based on the active policies for the user. When the user makes a call to the API including the data to be stored in the BC, PleBeuS executes its selection process to

find the appropriate BC to be used for the provided data. First, the currently active policy is determined, depending on the parameters which the user has configured beforehand (*e.g.,* cost thresholds). When the active policy has been determined, all BCs that comply to that policy, are selected, before the result is returned to the user. The BC Selector fetches static BC data from the database and uses external services to determine transaction costs of public BC implementations. The response of the API call returns the extracted data from the user-provided input, the hashed value of the extracted data, the cost of the transaction, the BC implementation chosen for each transaction and additional information used for providing statistics about the transactions.

- *Transaction Generator*: This component receives the selected BC and constructs a transaction in the format required by the Bifröst API, *i.e.,* an HTTP request containing the data to be stored and the selected BC.

4.2.2 Goals

The goal of this prototype is to provide a framework for defining policies, which are used for selection of an appropriate BC implementation, depending on the use case that it is needed for. The core problem the framework solves, is that it abstracts the strengths and shortcomings of different BC implementations. In turn, it also automates the BC selection process. Using a defined set of configurable parameters, a fitting BC is selected on transaction execution. During the design and implementation of this prototype, there was an emphasis creating a *robust*, *extendable*, and *flexible* solution, while providing *user-friendly* set up and usage.

Robustness of the prototype was provided by always ensuring that only one policy at a time can be active, while also restricting the user from creating any policy conflicts. Additionally, the framework ensures, that each defined policy always returns at least a single blockchain implementation that fits the chosen parameters. Thus, a consistent output is always to be expected, when API calls are made towards the framework. Finally, during configuration of the policies, the framework informs the user in case some conflicting configurations are about to be made. This effectively eliminates invalid or conflicting policy configurations even before a policy has been created.

At the moment, PleBeuS prototype supports seven BC implementations, which have been introduced in Section 4.1. Additionally, as a *fallback*, it also supports a traditional database (PostgreSQL) in case no BC fits the desired characteristics. Adding additional BC implementations is straightforward, one would need to add the implementation to the database, including the characteristics (as shown in Listing 1), and extend the code with the respective cost model. This makes the framework *extendable* with additional BCs and features.

```
{
    "_id" : ObjectId("5ca359bcd1297088c8adf971"),
    "nameShort" : "BTC",
    "name" : "Bitcoin",
    "type" : "public",
    "tps" : 4,
    "blockTime" : 600,
    "maxTrxSize" : 80,
    "turingComplete" : false
}
```

Listing 1: Example of a BC Implementation as Stored in the PleBeuS Database

While the prototype is configured to work in conjunction with the Modular Blockchain Interoperability API, it is also usable as a standalone application. This makes the proposed solution *flexible* in its usage. In case the framework is used independent of the Modular Blockchain Interoperability API, the user is able to define policies and use the response payload returned by the framework, as a recommendation for which BC implementation to use. This is also why the call to the Interoperability API is not made directly after selection and the output is returned instead. Decoupling the two APIs enables a flexible solution that can still be used in conjunction with each other, if one wishes to do so.

Additionally, the framework opens up the possibility to extract information about the transactions. The API used for passing data to the framework returns all necessities for a transaction (as shown in Listing 2). This could be used for analysis purposes if, for instance, one wants to compare costs created by using the framework with the regular cost by simply using a single BC.

```
[
    {
        "username": "TestUser",
        "blockchain": "Ethereum",
        "dataHash": "hashed data using SHA-256",
        "data": "the plain data",
        "cost": 0.01,
        "policyId": "5ce16ca8fb2adb4b443ae2b9",
        "costProfile": "performance",
        "interval": "daily"
    }
]
```

Listing 2: Example of a Transaction Payload

Finally, as for the usage of the framework, a GUI has been created for the management of policies enabling to easily create, edit or delete policies. This GUI is based on a REST API the framework exposes. Rather than using application-specific interfaces, REST manipulates resources, using a uniform set of “stateless” operations [36].This means, that the operations performed on the API, which are used for managing the framework, are not dependent on each other’s state. In turn, it makes using the GUI for management completely optional. This opens up further opportunities for automation, making it possible to manage policies and in turn the whole framework, via external scripts. Interacting with the framework is possible with any programming language, which lets the user choose in what context the framework should be used.

Due to the nature of the framework being a single web application, the setup is user-*friendly*, as it is easier to use than a Command Line Interface (CLI), for instance. Accompanying the framework is a *docker* setup. Docker enables hardware virtualization and isolates applications by using containers *docker*. By using the *docker* setup, it is only necessary to set the needed environment variables, before the application can be set up with a `docker-compose up` command, which installs the dependencies and populates the database with the necessary information.

4.2.3 Policy Parameters

Users can define policies, which provide the key component for this framework. These policies represent the set of rules the framework abides by, for selecting the appropriate BC, given the state of different factors (*e.g.,* time of the day or cost thresholds). The framework's policy parameters and filters can be split into two different categories. One of them being *BC-specific characteristics* and the other being *externally-driven factors*. Next sections present an in-depth explanation about parameters that a user can configure for a policy and which consequences these decisions entail.

4.2.3.1 Blockchain-specific Parameters

These parameters are bound to characteristics the different BC implementations exhibit. The characteristics are statically stored in the database as they do not change and are not dependent on any external factors. Hence, the characteristics of the BC implementations are mapped one-to-one as chosen parameters for a policy. They are used to filter BC implementations based on their characteristics and are not part of the reason why a policy changes. This means that they are responsible for selecting BC implementation within a policy.

Public vs. Private: Users may choose whether the data to be stored should be open to the public or not. In case the transactions contain sensitive data, storing into a publicly accessible manner is not an option. When the user makes the choice of whether a public or a private BC should be chosen, the framework retrieves data from the database, where the information of whether a BC implementation is public or private is stored. In the Policy Management Tool, the user can select, whether the BC should be public, private or if there is an indifference about the type. During BC selection, this information is then used to filter the BC selection pool for the active policy.

Blockchain Throughput: The minimum throughput that needs to be supported by the implementation is another chosen parameter. It is labeled as a number, which represents the amount of Transactions per Second (TPS) the BC implementation

supports. In case the user needs fast transmissions and has to send out a lot of transactions, the parameter's value can be set to a high number, which in turn eliminates all implementations from the selection pool that do not meet that TPS threshold. In the Policy Management Tool, the parameter's values can be chosen from a list, which contains the values from the respective BC implementations.

Block Time: While the BC throughput states how many transactions can be made in a second, it does not take in to account how long it takes for the transaction to be confirmed. This is why the block time is selectable for users as well. Similarly, to the throughput, in the Policy Management Tool, the value for this parameter can be chosen from a list of available values.

Data Size: Most BCs are not designed to store a large amount of data. Depending on how much data has to be stored in the transaction, this might be an important factor for the user. In the use case of cold-chain monitoring, the size of the transaction does not matter much, since the data does not get stored into the BC directly. Instead, only a hash of the data is stored and the data itself is stored somewhere off-chain. Nonetheless, this is a selectable parameter in case the use case demands more data to be stored in the chain. These size values conform to the limitations imposed by the Interoperability API. Just as with the previous parameters, the user can choose from the available values and if the value does not meet the threshold, it is removed from the BC selection pool.

Turing Completeness: Depending on the use case of the user, Turing completeness of the BC implementation is a criterion that might be needed for the BC implementation. This is important if Smart Contracts want to be used in the transactions. Even BCs that are focused on providing transactions can have limited SC capabilities, though their lack of Turing completeness limits their complexity. To be able to SC-focused BCs, this parameter concerns the Turing completeness and not general Smart Contract capabilities.

In the context of this project, SC capabilities are not necessary as the transactions are already verified and just need to be stored in the BC. Though, it provides an additional characteristic that differentiates the implementations. For other use cases this might be helpful information. In the Policy Management Tool, the user can choose if a Turing complete BC is needed or not. Should this be the case, BC implementations, which do not have these capabilities, are removed from the selection pool during the selection process.

4.2.3.2 Externally-Driven Parameters

These set of parameters are not static and are susceptible to external factors, for instance the time of the day. They do not map to any BC characteristics per se and map user preferences instead. These are the parameters that are responsible for policy changes.

Cost Thresholds: Users can set a maximum amount of costs they are willing to spend in a certain interval, as one of the policy parameters. They can specify any amount as their maximum value in a specified currency (CHF, EUR or USD), they want to spend for their transactions on a daily, weekly, monthly or yearly basis. As soon as the cost within the interval has been reached, the currently active policy switches to the next highest interval. Costs are calculated by different means. For BCs that have variable transaction costs, the information is fetched via multiple external APIs that provide information about transaction costs respectively [37] [31] [27] [38]. Since those external sources deliver the

fees in the currency of the respective BC, an additional call is made to CoinMarketCap [16]. This converts the transaction costs to the previously chosen currency. Since the value of cryptocurrencies is so volatile, this is done with each request to the transaction endpoint of the framework. This makes sure that the costs are always as up to date as possible.

As an example of how this parameter influences the policy selection: When the user provides a maximum cost threshold of CHF 10.- daily and the daily cost has already been reached, the framework switches to the next highest interval or to the economic cost profile of the same interval (this is further explained). If cost thresholds of all policies defined are reached, the default policy, which does not contain a cost threshold, is activated.

When defining cost thresholds, the framework always makes sure that the cost threshold of the previous interval does not exceed the threshold of the following intervals. This is to make sure that no policies with a larger interval are rendered useless, as the framework would always skip the next intervals as soon as the threshold for the smaller one has been reached. Before this parameter is used for the BC selection within the framework, all the other parameters are used for filtering policy compliant BCs first. This is done for performance reasons, as this parameter is computationally the most expensive one. As a consequence, the amount of calculations can be limited on the BCs that fulfill the other parameters. Table 6 lists the different BCs with the respective transaction calculations and their source.

Table 6: Costs of BC Transactions and Calculation Methods

Blockchain	Fee Calculation	Source
Bitcoin	Estimation via APIs (average of 2 sources)	Bitcoinfees [27] Bockcypher [38]
Ethereum	Estimation via APIs (average of 2 sources)	Etherchain [37] Bockcypher [38]
Stellar	Fixed transaction rate: Each operation: 0.00001 XLM	-
EOS	RAM Price calculation via Bancor Formula	EOS Canada API [31]
IOTA	No Transaction Fees	-
Hyperledger	No Transaction Fees	-
Multichain	No Transaction Fees	-

Cost Interval: As mentioned in the previous Section, for defining cost thresholds, the user needs to define for which interval this threshold is valid. The framework supports five different intervals. Namely: *daily*, *weekly*, *monthly*, *yearly*, and *default*. When an API

call to the Transaction Component of the framework is made, the calculated costs are added to each of the intervals for the current user. These accumulate for each interval until the specified amount of time is reached. As soon as the user interacts with the framework and the time for the interval has passed since the last reset, the costs for this interval are set to zero. This is valid for all intervals except for default. The default interval does not accumulate costs, since it is not bound to costs and represents the fallback, in case all the other policies are currently inactive. Every time the user makes an API call to the transaction component, the default policy ensures that there is always an active policy and a BC implementation can be returned. Compared to the other intervals, default does not contain a cost threshold. While policies for the other intervals are optional, every user must have a default policy, which is why the user gets prompted on creation to configure the default policy as a fallback. Only one default Policy per user is allowed, since they do not offer any other distinctive features, besides the BC selection pool, which would make it impossible to differentiate between default policies.

Currency: The currency is used for translating cost calculations to a uniform and known entity. As previously mentioned, the APIs used for calculating costs, always return the cost in the respective BC currencies. To be able to calculate costs in an efficient and for the user understandable way, the user is able to choose a currency in which the calculations are done. Just as with the default policy, on user creation, it is necessary to select a currency. Currently, the framework supports *CHF*, *EUR,* and *USD* as the selectable currencies.

Cost Profile: Cost profiles provide a way to put an emphasis on either performance or costs. This parameter is used in case multiple BC implementations fulfill the criteria defined by the policy. The users can choose between a *performance* and an *economic* cost profile. In case performance is chosen, the framework chooses the most performing implementations from the BC selection pool. Consequently, for economic, the implementation with the lowest transaction costs is chosen from the selection pool. In terms of policy priority, the framework prioritizes performance over economic policies within the same interval. The reasoning behind that is that an economic profile signals a prioritization in costs. Should the emphasis completely be on reducing costs, no performance policy would be configured for this interval. If there is some priority in performance, a policy with a performance cost profile can be defined, while there can also be an economic one within the same interval. As soon as the cost threshold of the policy with the performance cost profile is reached (*i.e.,* the willingness of the user to spend money on performance), the policy would switch to the economic one.

Transaction Split: Since more than one BC implementation can be in the pool of BCs for a certain policy, a parameter exists to define whether the data to be stored should be spread over all available BCs or not. In case the transaction split is not chosen, the cost profile chooses a single implementation for all transactions. If a split is chosen, the transactions are spread out over the pool of valid BCs. The priority is still handled over the cost profile, which means that in case an economic cost profile is chosen, the framework chooses the BC with the lowest transaction cost first and vice versa. Consequently, this also means that a not so performing BC can be chosen in the performance cost profile for a certain transaction, should the amount of transactions needed for storing all the data line-up with the number of BCs available in the selection pool, vice versa for the economic cost profile.

Time Frame: The time frame specifies during which time of the day the policy should be active, given that all the other constraints (*e.g.,* cost thresholds) are fulfilled. This parameter can either be set as valid for the whole day or at a specific time frame of the day (*e.g.,* 08:00 - 17:00). To avoid policy conflicts, for each cost profile within a certain interval, the time frames are not allowed to overlap. For instance, the user has already defined a policy with a daily cost interval and economic cost profile that is valid between 08:00 - 17:00. Additional policies with the same interval and cost profile would only be permitted between 17:01 and 07:59. This parameter allows for putting an emphasis on a certain time frame during the day. For instance, if more performing BCs are needed during working hours, configurations can be made to emphasize performing implementations during that time frame, while on the other hand less performing implementations can be chosen off-peak.

Preferred BC: Finally, a possibility to select preferred BCs is also provided. The user can select a single or multiple implementations. This allows users to let the system only choose between BCs they trust. If the user selects a single implementation, all the other provided parameters are ignored and the policy framework selects the BCs implementation the user wishes to use. If the user selects multiple implementations, the other parameters are applied on the selected pool of BC implementations returning the most suitable one. If no preferences are specified, the parameters of the policy are applied to the whole BC selection pool.

4.2.4 Selection Algorithms

After the filters are applied, based on the *Cost Profile* defined by the user in the policy, PleBeuS executes two algorithms to select the most appropriate BC from BC pool. If the user specified a *Cost Profile* that prioritizes *performance*, PleBeuS executes Algorithm 1. This algorithm selects the BC that presents the highest *TPS*, meaning that the data will be immutable recorded in a BC as fast as possible disregarding the costs. However, if two BCs present the same TPS value, the algorithm prioritizes the one with lower costs.

Algorithm 1 Fastest BC Selection

```
1: procedure SELECTFASTESTBC(bcSet, costs)
2:     fastest ← bcSet[0]
3:     for all bc ∈ bcSet do
4:         if (bc.tps > fastest.tps) then
5:             fastest ← bc
6:         else
7:             if (bc.tps == fastest.tps) and (costs[bc] <
   costs[fastest]) then
8:                 fastest ← bc
9:             end if
10:        end if
11:    end for
12:    return fastest
13: end procedure
```

Algorithm 2 is executed by PleBeuS for policies containing the *economic Cost Profile*. This algorithm selects the BC that presents the lowest transaction costs from the

BC pool. If two BCs present the same transaction costs, then the algorithm prioritizes the BC with the highest *TPS*, as there is no impact on the cost, and the transaction is included faster. It is important to note that the `bcSet` is a pre-processed set based on the data to be stored. For example, if the size of the data to be stored is above the maximum data supported by a BC transaction, then, this BC is not appended in the `bcSet`.

Algorithm 2 Most Economic BC Selection

```
1: procedure SELECTECONOMICBC(bcSet, costs)
2:     economic ← bcSet[0]
3:     for all bc ∈ bcSet do
4:         if (costs[bc] < costs[economic]) then
5:             economic ← bc
6:         else
7:             if (costs[bc] == costs[economic]) and
   (bc.tps > economic.tps) then
8:                 economic ← bc
9:             end if
10:        end if
11:    end for
12:    return economic
13: end procedure
```

4.2.5 Functionality and Data Workflow

This Section focuses on the data workflow of the framework. It provides a detailed explanation on the functionality of the framework and the interaction between the different components introduced in Section 4.2.1. First, the Policy Management Component is examined followed by the Transaction Component.

4.2.5.1 Policy Management Component

Figure 9 summarizes the just explained data workflow. The starting point for the framework represents the Policy Management Component. This component has a GUI build on top of the API that it exposes. When the user enters the index page (1), a username can be entered into the provided form. A submission executes an HTTP GET request to the endpoint `/api/user-not-exist-check/{username}` (2), which is a GET request used to check if a user with the same username already exists. If such a user does exist, the user is prompted with an error message, informing that the username is already taken. If the username is not taken, the user is redirected to a page, which allows configuration of a default policy and the currency of choice (3). Submitting these configurations executes a POST request to the `/api/policies` endpoint containing the username, BC characteristics for the default policy and the currency in the request payload (4). The Request Handler of said endpoint processes the data, validates it and then stores the user and the default policy to the database.

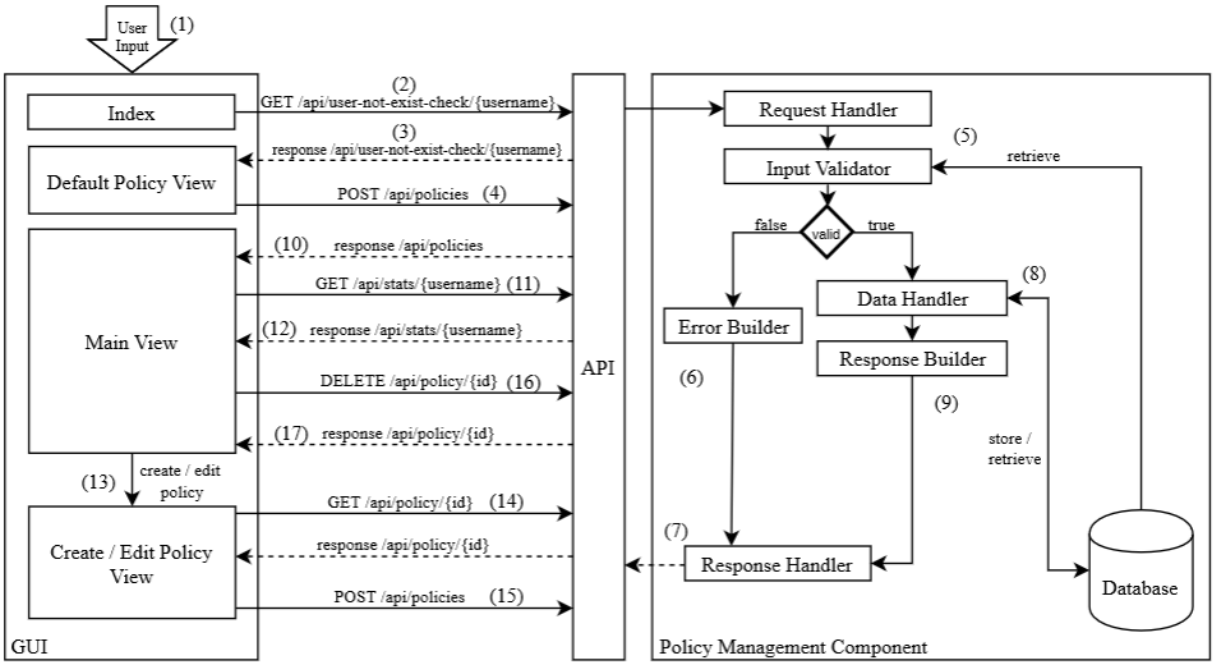

Figure 9: Policy Management Component Workflow

Each API request follows the same workflow in the back end. First, the request data is parsed in the Request Handler and validated by the Input Validator, which retrieves data from the database, if necessary (5). In case the input is invalid, the Error Builder builds the error message and error code, which it forwards to the Response Handler (6) before the response is returned (7). In case the input is valid, the data is passed to the Data Handler, which either stores data in the database or retrieves it, depending on the type of request that is executed (8). Following that, the response of the request is built in the Response Builder and passed to the Response Handler (9), which returns the API response (7).

In case the storage of the default policy is successful, the user is then redirected to the main view (10), containing all defined policies, which can be edited or deleted (as depicted in Figure 10. Before the user is redirected, the back end fetches all policies for the specified user and puts them into the view directly. The main view also allows for the creation of additional policies, it shows statistics about the executed transactions so far, and it shows the currently active policy. The statistics and the policy activity status are fetched via a GET request on the `/api/stats/{username}` endpoint (11). This endpoint is not essential to the operation of the framework, as it simply enables to show statistics and the policy activity in real time, without having to reload the view. The view periodically makes requests to this endpoint to be constantly up to date with the data. When such a request reaches the back end, it assembles the necessary data from the database and formats it in a way which is readable for the front end elements. These are then updated, upon successfully executed request (12).

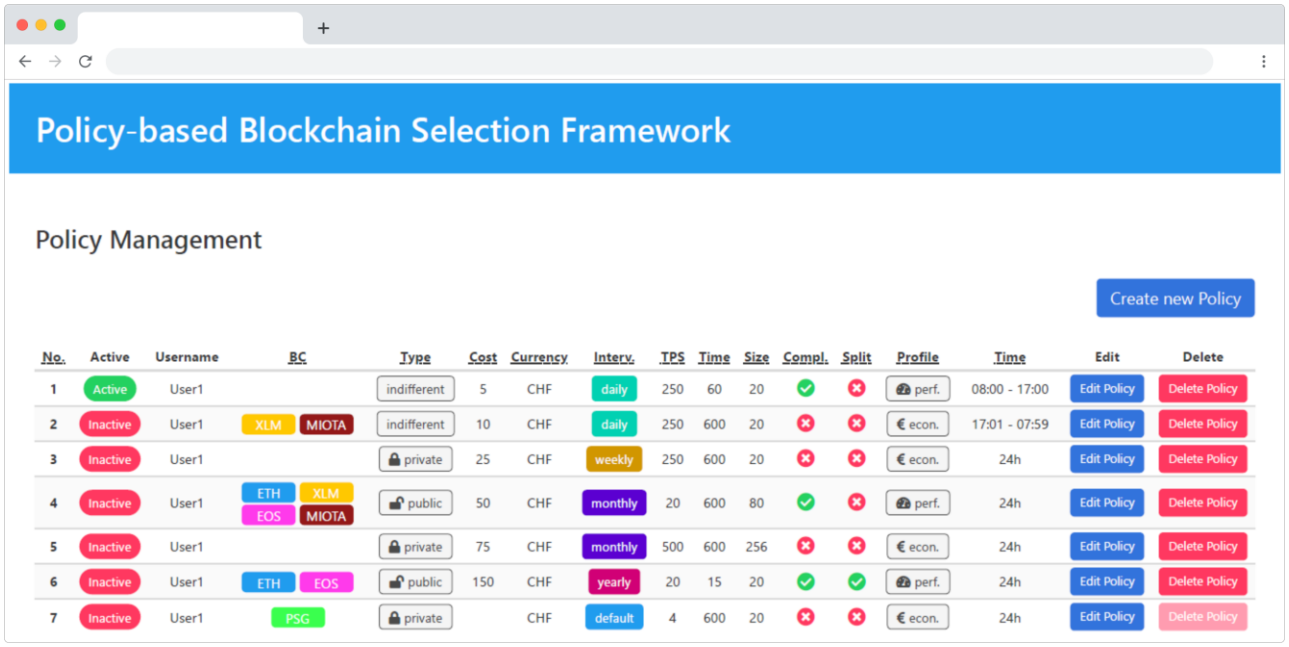

Figure 10: Main view of the Policy Management Component

By clicking on a button to create new policies, the user is redirected to a configuration view which contains a form (13). This form allows for creation and configuration of policies. The same view is used for editing policies, in which case the same endpoint to the view is expanded with an additional query parameter, containing the id of the policy (14). In case a policy is edited, the existing data is already filled into the form. As soon as the parameters have been configured and the form has been submitted, a POST request to `/api/policies` is executed, with the request body containing all information about the policy parameters and the username (15). Before the data is stored though, the back end first checks for any policy conflicts (5). If there are no conflicts, the policy is stored in the database (8) and as soon as the front end is notified (7), the user gets redirected to the main view with the updated polices and statistics (10). In case there are conflicts, the user receives an error message informing about the conflict. Conflicts arise in the following cases:

- No BC fits the chosen parameters (*e.g.,* public BC with a too high TPS threshold).
- A policy with a higher cost interval is not allowed to have a lower cost threshold and vice versa.
- Within the same cost interval and cost profiles, no overlapping time frames are allowed.
- Only one default policy is allowed per user, as it does not contain other distinguishable features, besides the BC selection pool.

Finally, in the main view, the user can also delete policies. In case the user clicks the delete button on the corresponding policy, a DELETE request is sent to `/api/policy/{id}` (16). In the back end, it checks whether the policy to be deleted is the default policy (5). If that is the case, the policy does not get deleted and the user gets an error message. This is to make sure that a user always has an active policy. In case it is not a default policy, it gets deleted and the user gets a refresh on the main view (17).

4.2.5.2 Transaction Component

Figure 11 illustrates the workflow of the transaction component, which is the component that creates the transaction and communicates with the BC interoperability API. Once policies have been defined, transactions are sent to the `/api/transactions` endpoint (1). To do that, a POST request containing the username, an Excel file containing sheets with temperature data and the minimum and maximum temperature thresholds need to be passed in the body of the request. Alternatively, instead of the Excel file and the temperature thresholds, a string can be used as input.

As soon as this request reaches the back end, the body of the request is checked for valid parameters (2). First, a check that the user has passed all necessary parameters is conducted. This means the request body contains a username and either the temperature data and the thresholds or the string to be stored. The request cannot contain both a string and an Excel file. This is to not leave any ambiguity about which data should be stored and to make sure to not cause any conflicts. The passed parameters are then further validated. The username must be available in the database and have at least one policy defined and the temperature thresholds must be valid (*i.e.,* minimum temperature lower than maximum temperature). If this is not the case, the API returns an error to the user (3).

If the request conforms to the framework's requirements, the data is further processed. However, in a first step, the system checks whether the accumulated costs over a certain interval have reached said interval (4). This is done by comparing the last update for an interval to the amount of time of that interval (*e.g.,* it checks whether a day has passed for the daily interval, since the last update) (5). If this is the case, the costs for that interval are set to zero.

Following that, the request is processed. In case the Excel file has been sent, sheets are parsed and the violating data is extracted from the sheets (6). Each sheet represents a temperature measurement of a delivery and corresponds to a single transaction. Next, the costs generated by the transactions are fetched (7). For public BCs, exchange rate data is fetched from CoinMarketCap [16], which is then used to convert the costs, fetched from various external sources [37] [31] [27] [38] to the currency selected by the user. Different calculations take place to convert the costs in to costs per byte, depending on the format the external sources return the respective cost. This is done to limit the amount of requests to the external resources, as it would drastically reduce the performance of the framework, if a call for each sheet would be made. Now, the amount of bytes needed can simply be multiplied by the costs. As soon as those calculations are done, each sheet violation data is examined step by step.

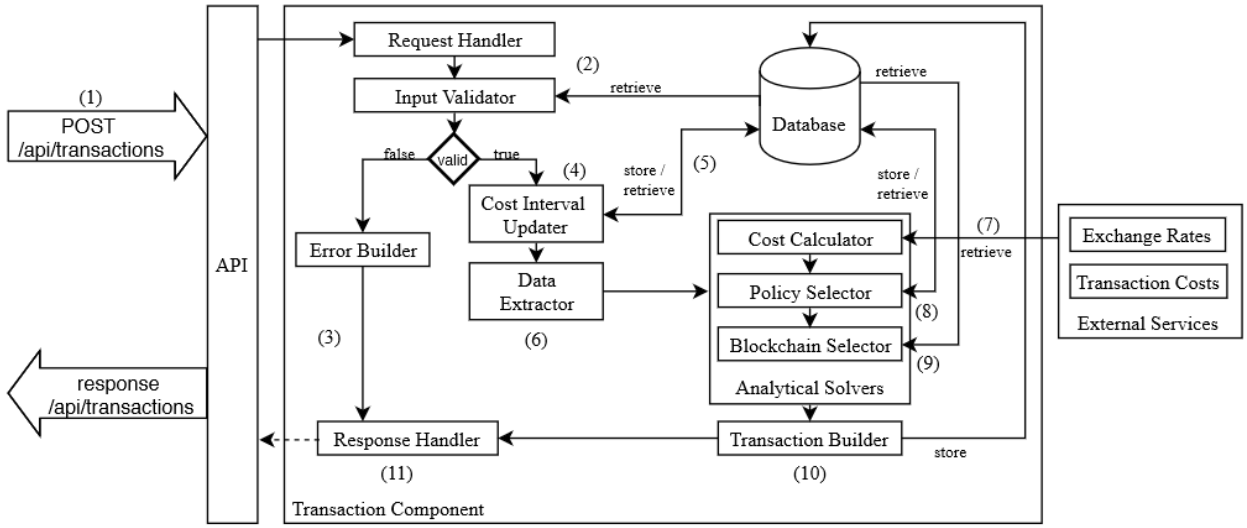

Figure 11: Transaction Component Workflow

First, the policy is selected depending on the cost thresholds (8). The policy with the lowest interval is selected, that is within the time frame and has not reached the max costs yet. From this policy the BC selection pool is determined, selecting the BCs that conform to said policy (9). From the selection pool the most performing or the cheapest one is selected for the transaction, depending on the cost profile the policy exhibits. The calculated costs for the transaction are added to all cost intervals before the next sheet is passed through the same process. In case a transaction split is requested, the framework will split the transactions to send in parallel to multiple BCs, prioritizing them according the respective cost profile. If the user only passes a string for the transaction, pretty much the same process is conducted, with the difference, that the data is not looped through, instead, only one transaction is executed. Finally, the selected BCs are collected for each transaction (10) and returned to the user (11).

4.3 A Modular Blockchain Interoperability API

This section describes the Modular Blockchain Interoperability component, which was named Bifröst, being organized into three subsections. Firstly, it is defined the API goal and the requirements taken into consideration during the development of the API. Secondly, it is described the details of the components and workflow of the API. Lastly, it is presented the implementation of the API, and listed technical details of libraries and technologies.

4.3.1 Design

The main objective of Bifröst is to provide a simple interface to interact with different BCs, *i.e.,* allows users to store, retrieve, and migrate data from BCs. In this paper, a “user” is defined as a developer of a BC application. Thus, this interface allows the developer to create programs that support a variety of BCs without knowing the underlying BC and library implementation. Additionally, this allows further abstractions by potentially letting algorithms decide on which BC to use depending on certain parameters, *e.g.,* transaction costs or BC performance. Figure 13 depicts the interaction with a BC application and Bifröst.

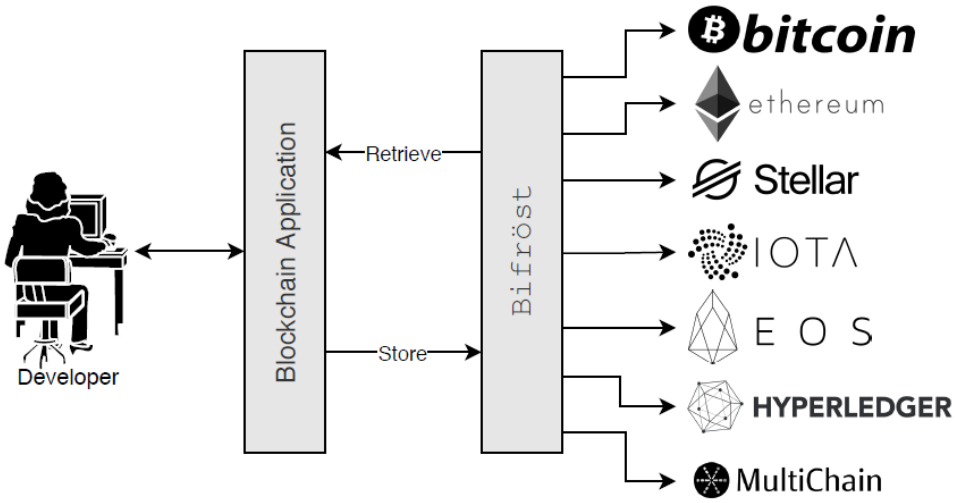

Figure 12: User Interaction with Blockchain Application and Bifröst

Three requirements were taken into consideration for the development of Bifröst: *(i)* flexibility, *(ii)* modularity, and *(iii)* ease of use.

The first, *flexibility*, is given by allowing the user to store a string in the BC, which could represent any arbitrary data, *e.g.,* a Secure Hash Algorithms (SHA)-256 hash. The second, *modularity*, is provided by implementing the adapter of each BC with a standard interface, simplifying adding adapters to new BC. Finally, the third, *ease of use*, is achieved by abstracting technical details from the underlying BC implementation, providing simple API functions (cf. Section 4.3.4), which require only two inputs from the user, being the data to be included and the BC identification. Moreover, using *docker* to run BC Remote Procedure Call (RPC) servers improves *ease of use* as its employment minimizes compatibility problems because the nodes execute in an isolated and replicable environment.

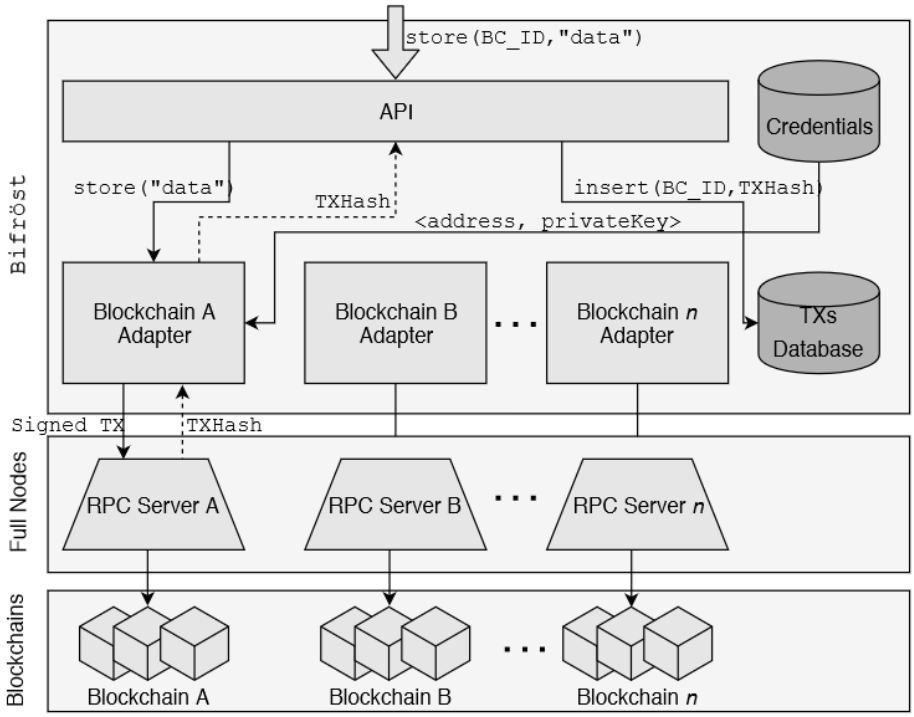

Figure 13: General Bifröst Architecture and `store` Function Flow

Bifröst relies on a notary scheme to interact with multiple BCs. This scheme was selected because it is a straightforward manner to manage data stored on different BCs without changing the underlying BCs implementation or maintaining parallel chains. Bifröst consists of three main components: (1) the *API*, (2) the *BC adapters*, and (3) a *database*. Figure 13 presents an overview of these three parts and the data flow between them using the `store` function as an example.

1. The *API* is the entry point for interacting with Bifröst. It consists of the exposed functions `store`, `retrieve`, and `migrate`. The API is responsible for receiving the user input and communicating with the correct BC adapter.
2. The *Adapters* convert the user input into a transaction which is subsequently transmitted to the BCs nodes. The nodes forward the transactions to the BC network, where miners will process them. In the case of the `retrieve` function, the adapter requests the data from the BC instead of creating a transaction. A new adapter must be implemented to allow support for each new BC.
3. The *Database* stores the necessary credentials for the transactions, and stores the transaction hash after a successful transaction has been included in the BC. This hash can be later used to retrieve the stored data. It is worth mentioning that the data is only stored in the BC and not in the database.

4.3.2 Bifröst Application Programming Interface (API)

Bifröst has three exposed functions in the API, which are used by developers or applications to interact with the available BC implementation. Their implementation is presented in Listing 3 and described in the following paragraphs.

```
1 def store(text, blockchain):
2    adapter = Adapter[blockchain]
3    transaction_hash = adapter.store(text)
4    return transaction_hash
5 def retrieve(transaction_hash):
6    blockchain = database.find_blockchain(transaction_hash)
7    adapter = Adapter[blockchain]
8    text = adapter.retrieve(transaction_hash)
9    return text
10 def migrate(transaction_hash, blockchain):
11   value = retrieve(transaction_hash)
12   new_hash = store(value, blockchain)
13   return new_hash
```

Listing 3: Exposed API Functions

The `store(text, blockchain)` function receives the data in the form of a string and the identification of the BC as input. It then stores the string on the defined BC, waits for the defined transaction confirmation time, and returns the transaction hash.

The `retrieve(transaction_hash)` function receives a transaction hash as a parameter and returns the string previously stored in the BC. The corresponding BC used to retrieve the data is automatically recognized by the API using a query in the database. Therefore, BC identification does not need to be provided.

The `migrate(transaction_hash, blockchain)` function retrieves a stored string from one BC and copies it to another BC. The parameters are the transaction hash of the origin and the name of the target BC. It should be noted that it is not possible to delete data from the BC and management of which transaction hash is valid depends on the user. Thus, this function copies the data to another BC.

4.3.3 BC Confirmation Time

Sending a transaction to a BC is generally no guarantee that the value is stored on the ledger. On the one hand the transaction could be refused by the network *e.g.,* because of a low transaction fee or if the validation failed. On the other hand, even if it was included, block finality needs to be ensured. Skipping this check could lead to transactions located on a fork which is abandoned later. However, finality depends on the consensus mechanism implemented in the BC.

In the prototype, this issue is solved by using a waiting time before writing the transaction to the database. If the transaction is not found after this time, it is considered invalid and will be discarded. Considering Table 7, Bitcoin and Ethereum need to have a longer waiting time of 3600 seconds respectively 105 seconds to have a high probability of finality. Even if block finality is given, transactions do not happen instantly. Especially with PoS consensus, there can be multiple rounds of communication between the nodes

until two thirds of them agree on one solution. As this communication takes some time, a minimum waiting time of 20 seconds was implemented [39].

The prototype was tested using the BC's public testnets or a local private network. Even though the consensus mechanism of those networks often differ from the mainnet, waiting times were set to conform with confirmation times on the mainnet.

Table 7: BCs Conformation Time Overview

Blockchain	Type	Consensus	Finality	Blocktime [s]	Confirmation After
Bitcoin	Public	PoW	No	600	6
Ethereum	Public	PoW	No	15	7
Stellar	Public	SCP	Yes	5	1
EOS	Public	dPoS	Yes	0.5	1
IOTA	Public	IOTA	Yes	60	1
Hyperledger	Private	PoET	Yes	20	1
Multichain	Private	PoA	Yes	15	1

4.3.4 API Documentation

The documentation of the API can be found at [40]. It contains information on the usage of the API, such as how to retrieve transactions, store data, migrate data, and so on. Further, the documentation presents code listings for its integration on several languages, including Python, cURL, jQuery, Node, PHP, and Go. Listing 4 presents the "`store`" function call to the API using cURL, where *String* represents the data to be stored in the transaction, and "*1"* informs the BC ID of the BC to which the transaction will be sent.

```
1 curl --location --request POST "https://bc4cc.ddns.net/api/store" \
2  --form "data=String" \
3 --form "bc_id=1"
```

Listing 4: Store Function Call in cURL

4.4 Proof-of-Concept Prototype Implementation

The prototype of the system was implemented in Node.JS (PleBeuS) and Python (Bifröst). The communication between the components is envisioned to be performed using the APIs provided by the components. This section is organized as follows, Section 4.4.1 overviews the networking structure of the prototype, Section 4.4.2 describes implementation details of PleBeuS, and Section 4.4.3 presents the implementation of Bifröst.

4.4.1 Entry-point and Networking Structure

Figure 14 illustrates the low-level networking structure of the prototype. As the system is, at the moment, a prototype and not production-ready, the deployment of the components, such as the server, and BC nodes, was performed in the CSG management network, with one exposed entry-point to the Internet. Even though the system is a prototype, security measures were implemented, such as a firewall and server isolation, to prevent malicious actions from external users.

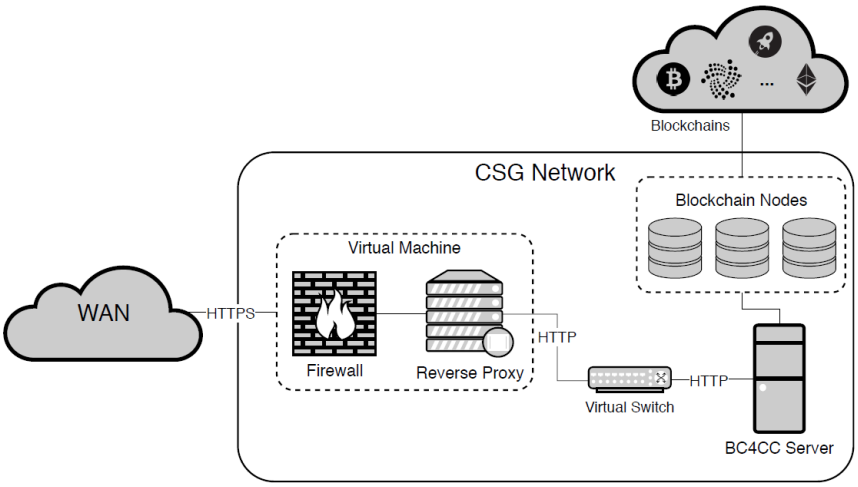

Figure 14: System Networking Structure

The system is composed of the following components:

- **Reverse Proxy.** The entry point to the system is an Apache HTTP server running inside a virtual machine, behind an `iptables` software firewall. The Apache server acts as a reverse proxy, forwarding requests coming from the internet to the Flask application, located on a separate machine. The reverse proxy exposes aside from HTTP and HTTPS also Secure Shell (SSH) for administrative access. It is the only machine exposed directly to the internet, and as such its availability is of critical importance. Transport Layer Security (TLS) traffic terminates here and traffic is further forwarded as HTTP. The Operational System (OS) is Ubuntu 18.04.3 LTS.
- **Virtual Switch.** This virtual switch is a software program which forwards packets from the VM to the Flask application.
- **BC4CC Server.** Physical server hosting the Flask application implementing the business logic of BC4CC and the database. It is a physical machine, a Dell XPS desktop, running Ubuntu 18.04.3 LTS.
- **Blockchain nodes.** RPC servers needed to send and read transactions, necessary for the core functionality of BC4CC. These are full BC nodes connected to their respective testnets.

4.4.2 PleBeuS

The framework at its core is a web application. The reasoning for choosing to build the prototype based on a web application is, that it allows for a combination of the GUI-based components and APIs really well. As mentioned in Section 4.2.2, one of the points of emphasis is to create an application which does provide a user-friendly interface, but on the other hand can also be flexible enough to be used in an automated context. HTTP-based REST APIs allow for a combination of these two worlds really well, which is the main reason, why this architecture had been chosen. This section provides an overview over the technology stack used to provide the web application. PleBeuS source-code can be found at [41].

4.4.2.1 Server and Client

The core of the framework is based on a Node.js server application. Node.js is an open-source, cross-platform JavaScript (JS) run time environment that executes JS code outside of a browser. It is based on the V8 Engine, which is a C++ based high-performance JS and WebAssembly engine. V8 has initially been developed by The Chromium Project for Google Chrome and Chromium web browsers [42]. It compiles the JS code to native machine code, instead of interpreting it as bytecode as is more common for this use case, which is where Node.js gets its speed from [43]. Other tasks it handles is the call stack of JS functions, management of the memory allocation for objects, garbage collection and it is responsible for providing all the necessary data types, operators, objects and functions. To run its asynchronous, I/O operations Node.js relies on a single-threaded event loop, which is provided by the Libuv library. It is a multi-platform support library with a focus on asynchronous I/O written in C [44]. By providing the event loop, Libuv allows Node.js to perform non-blocking I/O operations, despite the fact that JS is single-threaded.

By compiling JS to machine native code, Node.js allows usage of JS on the server side. This makes it possible to write full-stack JS web applications, which are performing, highly responsive as well as easily scalable thanks to the asynchronous nature of Node.js. By being able to write JS on the server side as well as on the client side, a developer has to only rely on one language to create a web application. All these reasons made the choice to use Node.js for this prototype an easy one.

For this prototype, Node.js provides the server of the various components. To handle HTTP requests the prototype uses the often accompanied Express Framework. Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications [45]. Main features of Express include the addition of simple to implement routing capabilities, query string, path string and response body parsing, grace full error handling and support for connect middleware, which allows for the extension of the application with additional plug-ins. These capabilities helped tremendously during the creation of the REST API, as they allowed for the handling of requests and the request data in a simple and extendable way. Express also allows to configure various templating engines for delivering HTML files. For this prototype, Nunjucks has been used to render the HTML used for the GUI. Nunjucks is a templating language specifically made for JS, which is developed by Mozilla [46]. It allows for a dynamic pass of data to HTML files based on the data passed from the rendering function. This helps eliminate duplication of HTML snippets and allows reusability of often used components.

4.4.2.2 Database

The REST API provided by the Node.js/Express server for managing users and policies, is designed to perform CRUD operations throughout the application. CRUD is an acronym the four basic functions of persistent storage, namely Create, Read, Update and Delete. CRUD-based applications apply the paradigm throughout the different layers of the application. At the user interface level, they allow interaction with the app to create, list, edit, and delete data, as it is the case for the policies in this prototype. At the HTTP layer, CRUD operations are mapped via the different HTTP methods, namely POST, GET, POST/PUT/PATCH, DELETE. Finally, at the database layer it used for storing, reading, updating and deleting entries. At the core of these operations stands the database. The

prototype uses MongoDB as its database storage, which is a document-based, NoSQL storage [47]. Compared to conventional relational databases, MongoDB does not use SQL to perform operations on data and data is not stored in relational tables. Additionally, while SQL databases conform to a predefined Schema, NoSQL databases allow dynamic structures of the data, which means that each document can have its own structure [48]. MongoDB stores data in JSON format (as presented in Listing 1 and Listing 2) in a document-based fashion. Each document is identified by a unique ObjectId, which can be used for retrieving a specific document.

The decision to use MongoDB has been made, as the policy and BC data fit really well into a document-based store, since they can be treated as single entities and do not require any joins. As for the policies, retrieving a policy retrieves all information it contains, without the need for joining different tables. The NoSQL approach also allows for scalability and easy adaptability, should the framework evolve in the future. For example, if new characteristics are added to the BC implementations or additional parameters are added to the policy, backwards compatibility is ensured without the need for adapting previous documents. Finally, since MongoDB is storing data in JSON, it integrates well with a JS-based framework like Node.js. By using MongoDB in combination with Node.js applications, serialization and de-serialization are automatically taken care of, since the documents are already stored in JSON format.

4.4.3 Bifröst

The Bifröst prototype was developed using python. Python was chosen because its properties allow for rapid prototyping while keeping the code clean and comprehensible. Furthermore, there is extensive support for libraries used by this project, *e.g.,* for cryptographic operation, HTTP and RPC. In terms of compatibility, the prototype was tested on both Linux Ubuntu 16 and MacOS Mojave using Python 3.6.6. Python's built-in virtual environment manager was used to prevent environment issues. Bifröst source-code can be found at [49].

4.4.3.1 BC Adapters

Internally, the `store` function constructs a raw transaction using the provided string. This transaction is then signed using the stored private key and sent using the RPC or HTTP protocol. After receiving the transaction hash, the function `confirmation_check` validates the existence of the transaction after waiting for a specified period. Thus, it is confirmed that the block was included and finality is given. If the confirmation was successful, the hash is saved to an SQLite database, referencing a BC identifier and including a timestamp.

The `retrieve` function requires the transaction hash as a parameter. The transaction is retrieved from the BC using RPC or HTTP. Afterwards, it extracts the string from the obtained transaction. Listing 5 presents an excerpt from the implementation. The parameter `cls` reflects the chosen BC in the form of an adapter class.

```
1 def store(cls, text):
2    tx = cls.create_tx(text)
3    signed_tx = cls.sign_tx(tx)
4    tx_hash = cls.send_raw_tx(signed_tx)
5         if(cls.confirmation_check(tx_hash)):
6              cls.add_tx_to_database(tx_hash)
7              return tx_hash
8         else:
9              raise LookupError('Tx not confirmed!')
10
11 def retrieve(cls, tx_hash):
12   transaction = cls.get_tx(tx_hash)
13   data = cls.extract_data(tx)
15   return cls.to_text(data)
```

Listing 5: Adapter Implementation

Currently, seven adapters to popular BCs and one for generic database are implemented in the Bifröst prototype. As Bifröst is modular, if there is the need to support a new BC implementation, a new adapter can be implemented (following the standard). The supported BCs and the generic database information are presented in Table 8. The PostgreSQL adapter was implemented to allow the data to be stored in a traditional database instead of a BC. Apart from the BC and library name, Table 8 also presents information about the node type, connection, and network type (*e.g.,* local or public testnet). For example, the Bitcoin implementation connects to a full node with access to the public Bitcoin testnet and uses the RPC protocol to communicate with the adapter.

Table 8: Blockchain Adapter Information

Blockchain	Library Name	Node Type	Connection	Network
Bitcoin	`Python-bitcoinrpc`	Full Node	RPC	Public Testnet
Ethereum	`Web3.py`	Full Node	RPC	Local Testnet
Stellar	`Py-stellar-base`	Remote Full Node	HTTP	Public SDF Testnet
IOTA	`PyOTA`	Remote Full Node	RPC	Public Testnet
EOS	`Eosjs_python`	Remote Full Node	RPC	Public Jungle Testnet
Hyperledger	`Sawtooth_sdk`	Full Node	HTTP	Local Testnet
Multichain	`Python-bitcoinrpc`	Full Node	RPC	Local Tesnet
PostgresSQL	`Psycopg2`	PostgresSQL 9.6.8	PostgreSQL	Local PostgreSQL

4.4.3.2 Database

The database is responsible for storing a list of supported BCs, credentials for the BCs, and transaction hashes. In the prototype, an SQLite database and the Python library `sqlite3` were selected. SQLite was chosen because of the simplicity of storing the whole database without maintaining a dedicated server [50]. Figure 15 depicts the Entity Relationship (ER) model of the Bifröst database. It is important to notice that the stored data is not saved on the database, but only on the selected BC.

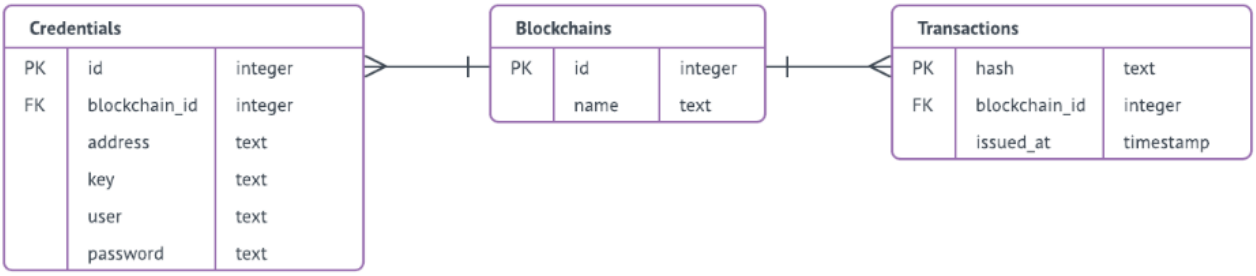

Figure 15: Database Entity-Relationship (ER) Model

The next items describe the tables which compose the database.

- **Blockchains Table**: The BC table has two columns, being a unique `id` and the `name` of the BC. This table is used to link the BCs to the transactions and credentials. An example of row in such a table is the tuple `<id:1, name:'Ethereum'>`.
- **Credentials Table**: The credentials table has six columns. The unique `id` used to map the identity of users to their credentials. The `blockchain_id` connects the

credentials to the corresponding BC. Furthermore, `address` stores the public key and `key` the private key of the BC account. The `user` and `password` provide the credentials for the RPC or HTTP client. The necessary credentials depend on the BC. For example, Stellar does not need `user` and `password`, and IOTA does not need `key`, as zero-value transactions do not require a sender. Thus, no need for a signature.

- **Transactions Table**: After a transaction is completed, its hash is stored in the column `hash`. The BC that the transaction is included is identified by the `blockchain_id` column. Furthermore, the `timestamp` of issuing is included in the column `issued_at`.

5 Case Study – BC4CC Integration

This chapter explains how Bifröst was integrated into the Demo Environment of modum.io – thus, the productive code base of modum.io to demonstrate functionality to customers – and discusses its perceived benefits and limitations. While the straight forward integration shows the important advantages of the system, it is mainly limited by its abstracted functionality.

5.1 Integration of BiFröst and MODSense

BC4CC facilitates data sharing between different actors in the supply chain. Those stakeholders can rely on the trusted data that is enabled due to the BC4CC framework being integrated into MODSense. MODSense is Modum.io's solution for temperature monitoring, designed for sensitive shipments. Customers can hence predefine requirements per shipment and automatically evaluate conformity upon readout of data.

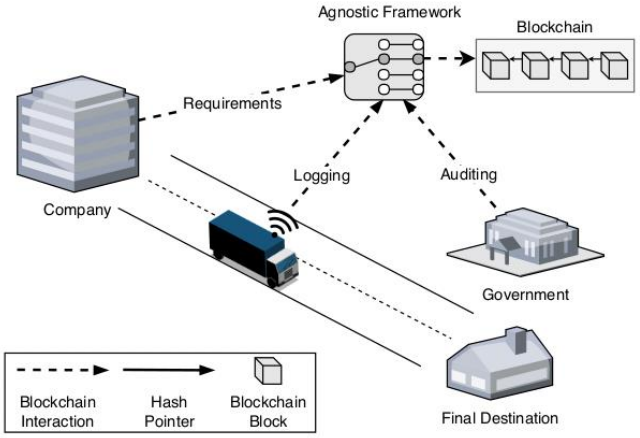

Figure 16: The stakeholders Involved in the System

As introduced in Section 3.2, Figure 16 presents a high-level system overview of the different stakeholders involved in the usage of the BC4CC system. Temperature data is recorded during the transport of medical drugs. Using the framework, the data can be stored in one or multiple BC and audited by the government.

5.1.1 The Modum System

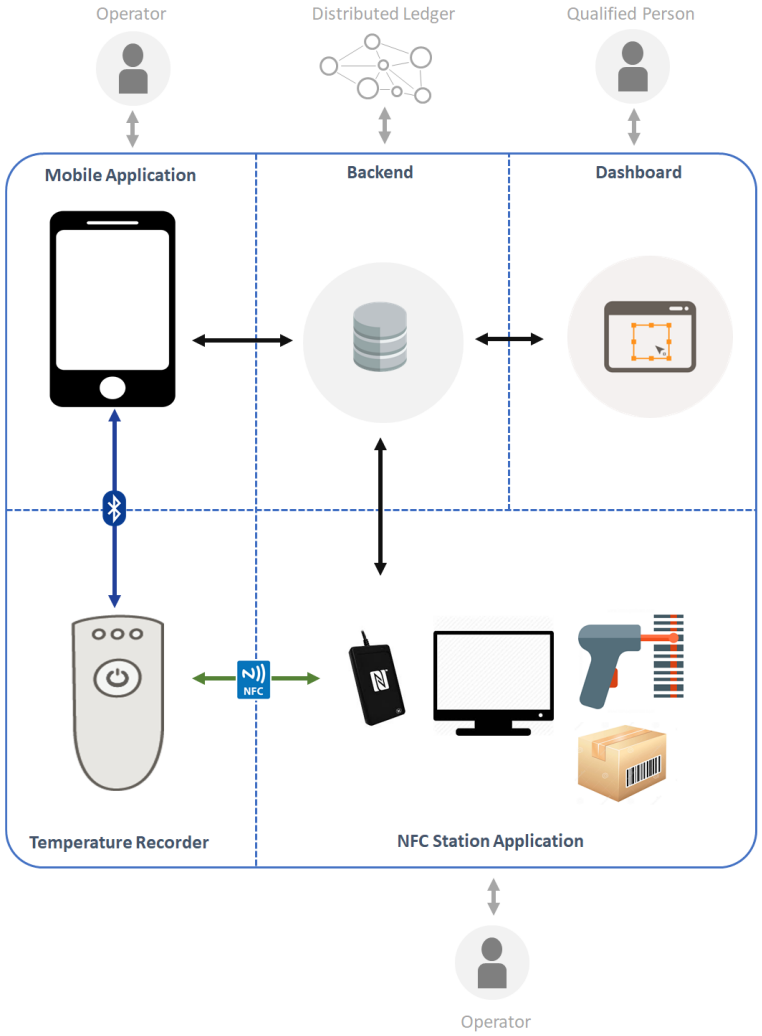

Figure 17: The General Setup of the Modum System

Figure 17 shows the general setup of the Modum system. Besides the web application for shipment management and a mobile application for shipment initialization and readout of data, the MODSense T Temperature logger is an important component of the solution. It evaluates the collected temperature data against the alarm criteria that was defined for each shipment and can be configured over the air using Bluetooth Low Energy (BLE). The device features a tamper-evident casing, a maximum storage capacity of forty-five thousand temperature measurements, Near Field Communication (NFC) support, and traceability via the QR code. Critical for this project is the built-in on-board hardware security module that holds a private key initialized in the factory at manufacturing time yet completely inaccessible to anyone. The private key enables the logger to digitally sign any kind of data. In the remainder of this document we also refer to the MODSense device as the modum.io logger. The modum.io logger firmware runs on the Microcontroller Unit (MCU) which contains a Central Processing Unit (CPU), volatile and non-volatile memory, hardware peripherals for bus communication and other tasks, as well as all necessary hardware for a BLE connection except the antenna. The HSM contains its own processor and can run code independent of the MCU.

As it can be seen, the Modum.io Backend-Services communicate with a BC. Modum.io has built a connector from their system to the Bifröst API to communicate with this additional service. Thus, the integration of Bifröst is easy and seamless. To evaluate the effectiveness of the system, Bifröst was integrated into the “Demonstration” Environment of the Modum.io Application. An integration into the productive system is not possible, as Modum.io operates in a regulated environment, and the components would have had to be developed according to regulated frameworks and standards. In addition, contracts with all different users would have been required.

5.1.2 Bifröst Integration

Figure 8 shows the architecture of the BC-agnostic framework and its components, which consists of an externally visible API (*i.e.,* Bifröst). The framework also includes a Policy Decision Point (PDP) that implements the selection logic and external services that include cost monitoring and exchange rates.

Having a closer look at the Open API, which is depicted in Figure 13, it shows that the API offers a storing method that accepts the Identification (ID) of the BC in which the data should be stored. The BC correspondent adapter then creates a new raw transaction and signs it using the credentials stored in the system. For later management the transactions hashes are stored in a database. The adapter forwards the transaction to the respective RPC interface of the local node running the BC to which the transaction is broadcasted. The RPC server returns the transaction hash once the transaction has been included into the mining pool. This hash is used for retrieval of the data stored in the transaction. A simple integration of BC4CC and the modum.io system could reuse the same Smart Contract that is currently used in the modum.io system, which facilitates the integration even more.

5.2 Advantages and Limitations of the Approach

The BC4CC approach has many tangible benefits due to the integration of the Bifröst framework. The open API enables a broader market penetration, as more supply chain service providers could benefit from decentralized systems for their products, money and data. They can reduce their spending and streamline their processes by means of the self-executing Smart Contracts, data immutability and auditability, which Modum.io’s system provides. These service providers also do not need to have any deeper knowledge about the technology itself if they want to implement it, which makes it much more accessible and again broadens the spectrum of potential users. In addition, since the Modum.io logger is not directly connected to the Internet, it cannot interface the BC platform directly and obtain a trusted copy of the external values. BC4CC as an intermediate system, however, can provide these values signed digitally and, thus, vouch for their correctness. This would prevent modification of this data along the chain via the mobile application. By building an open API, the integration into other systems is greatly facilitated, making it easier to collaborate with different partners and customers. The data that is stored on the BC is accessible to every stakeholder at all times and hence immutable, which improves trust and transparency massively.

Moreover, the easy integration is crucial for the interoperability that sets BC4CC apart from single public BC integrations and prevents technology lock-ins. Even when choosing a BC that might not be widely used by other players, the company can still

integrate their system. Making this possible, BC4CC tackles one of the most significant issues of why BC has not yet been implemented in supply chain logistics. By developing Bifröst, the integration and hence accessibility of BC solutions have been increased massively, paving the way for new cold chain applications.

Despite the many advantages of the integration, it also has its limitations. Bifröst operates on an abstraction layer which abstracts its functionality and thus its application is limited to simple use cases such as data integrity. As soon as it can be integrated into a productive environment, the architecture can be tested in more specific use cases. Moreover, while the BC-agnostic approach is allowing users to choose the BC that matches their individual requirements, it does not enable different networks to operate within the same BC. Lastly, the API is part of a centralized architecture and by implementing Bifröst, a single point of failure is introduced into the system, which defeats the purpose of BC technology itself to some degree.

6 System's Evaluation and Discussion

Based on the prototyping as well as integration of these components as described above, the feasibility of the policy-based management is evaluated and discussed. In addition, a preliminary security analysis is presented.

6.1 Discussion on the Role of BCs in the Industry

During the course of the KTI/Innosuisse project, multiple discussions and talks with potential users in the pharmaceutical, perishables and logistics industry, with other logger companies and related BC startups have shown that the industry still struggles with the adoption of BC technology. Especially in the final months of the project, feedback and further insights into the state of adoption have been gathered. In the few cases where the technology is being adopted, the companies typically revert to either a private BC or one that was established by a consortium. The reason for this being that it is a much safer approach to the technology, which does not hold as much risk. Furthermore, solving the data integrity issue itself does not realize the full potential of a BC. Adding an abstraction layer to the BC may simplify the adoption, however, it also prevents users from unlocking the full potential of the technology.

Further, there is a general resistance against using a public BC which is mainly based on security concerns. Users are afraid of unwanted customer data and internal information to be published when using a public BC. Although this issue could be solved by encrypting the data before sending it to the BC, it also generates more complexity and might introduce additional costs, which presents another disadvantage. Another solution against security concerns is choosing a private BC that is designed for a use case where nodes are run only by a small number of companies or individuals. Thus, access to the data stored on the BC is possible only for the participants of the private BC. This is enticing for companies who fear that putting data on a public BC could be associated with privacy concerns. Many companies are still in the process of evaluating or monitoring BC adoption but are waiting for a push from their own customer base to do so. For many, it is still too early to adopt at this point in time, but they are waiting for larger organizations to come up with networks and consortia (*i.e.,* private permissionless and private permissioned) which then can be joined. Two examples of consortia that are developing their own BC frameworks are PharmaLedger [51] and MediLedger [52], which both strive

to enable BC-based healthcare solutions. In conclusion, the interest in the technology is still high, and once public BC adoptions become more necessary, the BC4CC deliverables have a greater impact.

6.2 Technical and Operational Impact

This section analyses the presented prototype with regard to performance, management and security. Firstly, an analysis of the performance of the prototype, including introduced overhead and data size limitations is described. Secondly, major management aspects are discussed.

6.2.1 Performance Analysis

A performance analysis was performed to measure both the overhead and the stability of the application. The measurements were performed on a MacBook Pro 2017, Dual-Core i5 @ 2.3 GHz, 8 GB RAM, MacOS Mojave 10.14.1 with Python 3.6.6. At total, 1000 measurements were gathered for each BC, except for Bitcoin. In case the of Bitcoin, 100 samples were taken because of the limit of 25 unconfirmed transactions on the Bitcoin RPC server. As some remote nodes impose restrictions on how many transactions can be sent over a specific time, the measurements were completed in batches of 25 transactions each. PostgreSQL was included in the analysis to compare BC performance with a regular database. Even though all the 1000 transactions per BC were completed without issues, a stability and scalability evaluation must be conducted to consider the prototype stable.

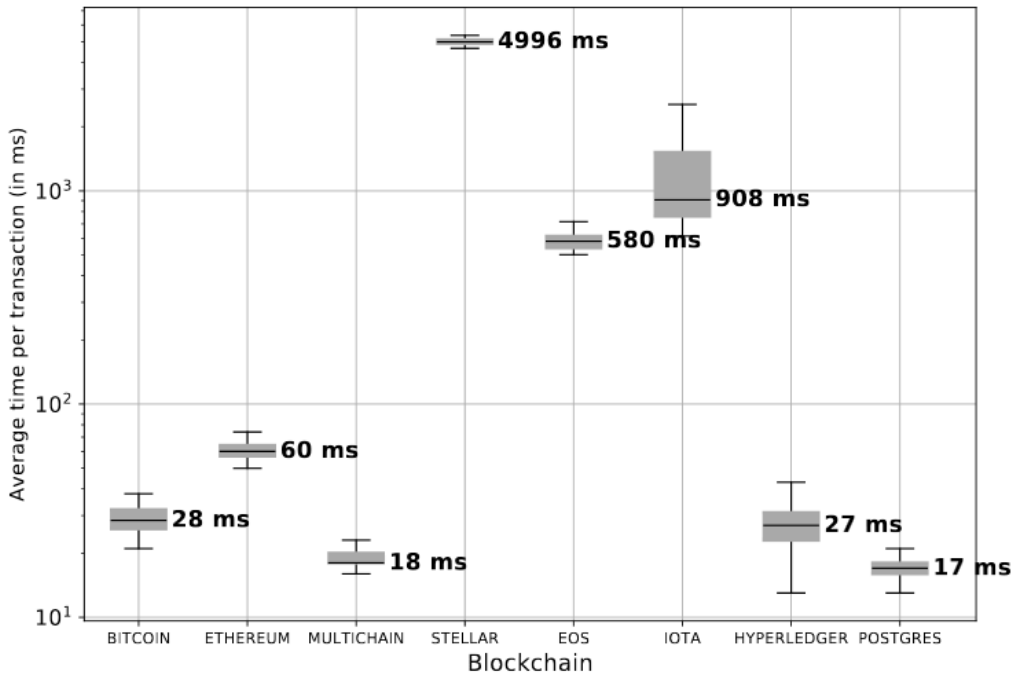

Figure 18: Performance Measurements on Supported Blockchains

Figure 18 depicts the outcome of the performance measurements. The *x*-axis represents the different BC adapters, and the *y-axis* represents the average time per transaction (in milliseconds). It can be seen that there is a performance difference between using local nodes and remote nodes. Remote nodes were used by Stellar, IOTA and EOS

as presented in Table 8. Noticeably, the Bitcoin client using a full local node with a public testnet was faster than other BCs (*e.g.,* Ethereum) which were using a local node with a private testnet. Multichain presented a similar performance as the PostgreSQL database due to the fact that Multichain is a private BC focused on data streams. However, Multichain requires more operations (*e.g.,* raw transaction creation and cryptographic signature) that PostgreSQL does not. It has to be noted that with some adapters (*e.g.,* Hyperledger), multiple transactions could be put into one batch. Thus, only one batch would need to be transmitted for multiple transactions, which would result in higher throughput.

Transaction-focused BCs, *e.g.,* Bitcoin and Stellar, are not designed to store arbitrary data. Thus, there is a limit on the amount of data included in a transaction. Table 9 summarizes how much data is possible to store in the different BCs supported by the system. Note that on some BCs, *e.g.,* Ethereum, this restriction could be circumvented by using SC. In an SC deployed in Ethereum, the storage of data can be divided over more than one block, bypassing the maximum data size limitation by the block *gas* utilization.

Table 9: Maximum Data Size in Different Blockchains

Blockchain	**Maximum String Size**
Bitcoin	80 Byte
Ethereum	46 kByte
Stellar	28 Byte
EOS	256 Byte
IOTA	1300 Byte
Hyperledger	20 Byte
Multichain	80 Byte

Due to the presented data limitation and the fact that the BC technology was not conceived as a database, but rather only a distributed ledger, holding only transaction-related data, it can be seen that it is not cost-efficient to store large amounts of data on BCs. In this sense, a (decentralized) database can be used to store the “raw” data, and a hash of this data is stored in the BC. Thus, the validity of the data can be verified at any time while the data is stored efficiently.

6.2.2 Management Discussion

In the following sections, relevant management and security aspects are discussed. First, it is discussed the security of storing the private keys in a central server. Then, the security of local and remote nodes is described. Finally, the centralization issue is addressed.

6.2.2.1 Private Keys – Single Address

A major concern is to secure the private keys from attackers. A successful attack in this vector could have two implications. First, an attacker could spend the funds linked to the account. As this solution is made for storing data on the BC, the funds would only need

to cover the transaction costs. Keeping only the necessary funds for the transactions would; therefore, mitigate this risk. Second, an attacker could change the data by storing some arbitrary data which would be hard to distinguish from the genuine transactions. Depending on the use case, this could be an issue to consider. In most use cases, the motive for such actions is questionable.

The presented prototype is a Proof-of-Concept (PoC), implemented to assess the system's feasibility. Thus, the private key is stored in plain text on an SQLite database. In this sense, if an attacker would get access to the server, it would be trivial to access the private keys. One solution to circumvent this problem is to encrypt the private key with a symmetric key which is based on a password set by the user, thus, only allowing for a transaction to occur, if the user temporarily decrypts the private key. Another possible solution consists of a *multisignature* transaction scheme, where more than one key is necessary to sign transactions, and they are stored in another, more secure location.

6.2.2.2 Local and Remote Nodes

Another risk is the exposure of the RPC or HTTP port by the BC node. If the node additionally holds the private keys in a local wallet, transactions could be initiated remotely. However, all of the BC adapters in this prototype sign the transactions locally, meaning there is no need for the node to hold the private keys. Nevertheless, the ports could be used to transmit unrelated, malicious transactions and Distributed Denial-of-Service (DDoS) attacks targeting the node could be executed. In cases where public remote nodes are used, the owner of those nodes could potentially block a transaction from being processed. Furthermore, the availability of the whole system could become an issue.

Nonetheless, these security issues could be circumvented by using full local nodes and allowing RPC connections only from the local machine (*i.e.,* localhost) or restricting connections to trusted IP addresses. Distributed Denial of Service (DDoS) attacks on the system can be mitigated by implementing a request limiter, throttling the processing of requests by IP or request type.

6.2.2.3 Centralized Prototype Deployment

One of the main benefits of BCs is the removal of trust by making use of decentralization and cryptographic properties. Thus, it would be advantageous if this property holds for this solution as well. However, using a notary scheme implies that a user needs to trust the notary *i.e.,* the host of the application. First of all, the notary has access to the private keys which makes him vulnerable to the issues discussed in Section 6.2.2.1. Furthermore, the notary controls the application and the node and therefore is able to arbitrarily alter the original transactions *e.g.,* change the sender or the data or censor certain transactions.

Nevertheless, BC applications are rarely trust-free, and there are always layers which require a certain amount of trust, such as monitoring applications in Internet-of-Things (IoT) applications. Taking this system as an example, the first layer of trust is the BC underlying code and cryptographic properties. The second layer is the RPC server and the RPC client. The notary is the third layer of trust. After that, other layers must be trusted, *e.g.,* the user's hardware and software. From this perspective, the notary scheme adds a layer which requires trust, but is still only one out of many. However, the amount of trust needed can be minimized by running the approach in a trusted computing environment, such as the Intel's Software Guard Extensions (SGX) [53].

6.3 Policy-based System Evaluation

To verify the system's functionality regarding the policy switching mechanism, and the BC selection algorithm, a prototype was implemented and evaluated in three scenarios with different sets of defined policies. These scenarios are defined in Section 6.3.1 and the results from each scenario are presented and discussed in Section 6.3.2.

6.3.1 Evaluation Scenarios

Each scenario combined both public and private BCs, and varied in parameters, such as the BC Set (*i.e.,* BCs available for selection), costs thresholds intervals (*e.g.,* daily, weekly, monthly, and yearly), which trigger a policy switch, and the profile (*e.g.,* performance or economic). The time frame parameter was not considered because the switch of policies is triggered by the cost interval. The evaluation scenarios and the defined policies are described in the next sections.

6.3.1.1 Scenario #1

For the first scenario, seven policies were defined (see Table 10) with the BC Set containing only private BC implementations (*e.g.,* Hyperledger (HYP), Multichain (MLC) and PostgreSQL (PSG)). PSG was defined as the BC for the default policy. Even though private BCs do not require transactions fees as public BCs do, they incur other costs, such as hardware and server maintenance, and support. Thus, an arbitrary cost of CHF 0.01 per transaction was considered, triggering the switch of policies.

Table 10: Policies Defined in Evaluation Scenario #1

No	BC Set	Interval	BC Type	Cost [CHF]	Profile
1	All Private BCs	Daily	Private	5	Performance
2	Hyperledger, Multichain	Daily	Private	8	Economic
3	Multichain, PostgreSQL	Weekly	Private	20	Performance
4	Hyperledger, Multichain	Monthly	Private	50	Performance
5	All Private BCs	Yearly	Private	100	Economic
6	PostgreSQL	Default	-	-	-

6.3.1.2 Scenario #2

In the second evaluation scenario, only public BC (*e.g.,* Bitcoin (BC), Ethereum (ETH), Stellar (XLM), EOS (EOS), and IOTA (MIOTA)) were selected to compose the BC Set. Nine policies were defined in this scenario (see Table 11), in which the profile alternated between Performance and Economic to evaluate both parameters in a balanced manner. Ethereum was arbitrary selected as the BC for the default policy.

Table 11: Policies Defined in Evaluation Scenario #2

No	BC Set	Interval	BC Type	Cost [CHF]	Profile
1	All Public BCs	Daily	Public	2	Performance
2	Bitcoin, Ethereum, EOS	Daily	Public	10	Economic
3	EOS, IOTA, Stellar	Weekly	Public	15	Performance
4	Bitcoin, EOS	Weekly	Public	30	Economic
5	Ethereum, EOS, IOTA	Monthly	Public	40	Performance
6	Bitcoin, EOS, Stellar	Monthly	Public	60	Economic
7	Ethereum, EOS	Yearly	Public	80	Performance
8	All Public BCs	Yearly	Public	100	Economic
9	Ethereum	default	-	-	-

6.3.1.3 Scenario #3

The last scenario combined both public and private BCs to evaluate the behavior of PleBeuS in heterogeneous environments. Seven policies were defined (presented in Table 12), with PostgreSQL being the selected BC for the default policy. Moreover, to increase performance, the split parameter was set to true, meaning that the data will be stored in the BC Set in a round-robin scheme. Similarly, to Scenario #1, an arbitrary cost of CHF 0.01 per transaction was considered for private BCs.

Table 12: Policies Defined in Evaluation Scenario #3

No	BC Set	Interval	BC Type	Cost [CHF]	Profile	Split
1	All Private BCs	Daily	Private	15	Performance	✓
2	All Private BCs	Daily	Private	30	Economic	✓
3	All BCs	Weekly	Indifferent	1500	Performance	✓
4	All Public BCs	Monthly	Public	4000	Performance	✓
5	All Public BCs	Monthly	Public	8000	Economic	✓
6	All BCs	Yearly	Indifferent	15000	Economic	✓
7	PostgreSQL	Default	-	-	-	

6.3.2 Results and Discussion

For each scenario described above, 10000 `storeData` requests were sent to PleBeuS API, which generated a total of 10000 BC transactions. Each request represented one collected random data point from a generic sensor, *e.g.,* temperature or humidity. The performance (*i.e.,* TPS) of the public BCs followed the lower spectrum of the values from Table 7. For the private BCs, the values from the performance evaluation conducted in [26] were selected. The results of the conducted evaluation in each scenario are depicted in Figure 19.

The policy switching mechanism is influenced by two parameters, *(i)* cost threshold, and *(ii)* time frame. As the time frame was not considered in the evaluation, the switch in the active policy was determined by the cost threshold. Figure 19a, Figure 19d, and Figure 19g represent the number of transactions executed per policy in the scenarios. It can be seen in Figure 19a that the default policy (No. 6) is never activated in Scenario #1 because the 10000 transactions were executed within the yearly cost interval (*i.e.,* 100 CHF) with policy No. 5 active for half of the total transactions, *i.e.,* 5000 transactions. Figure 19b confirms this behavior, depicting the costs thresholds for each policy, where the cost for policy No. 6 is exactly 50 CHF. Even though the same yearly cost was defined in Scenario #2, the switching occurred in a different pattern (see Figure 19d) due to the higher costs of interacting with public BCs. As soon as the accumulated cost reaches the yearly threshold in the last policy (No. 8), the default one is activated, and the accumulated cost reaches 400 CHF.

The results of the evaluation of the BC selection algorithms (cf. Section 4.2.4), are illustrated in Figure 19c, Figure 19f, and Figure 19i. The first two figures (Figure 19c and Figure 19f) show a clear distinction among the BCs selected due to the selected deployment type private BCs and public BCs. Moreover, the Bitcoin BC is not selected due to its high transaction costs because of its high prices (10000 USD as of September 22,

2019). However, in the third scenario (Figure 19i), with the split parameter selected, the transactions were sent to all available BCs, with PostgreSQL being selected 3500 times because it presents the fastest TPS and cheapest costs because it is a regular database. Despite the inclusion of PostgreSQL in the BC set, the split parameter guaranteed that others BC were selected with a tendency to private BCs being selected for the *Economic* profile. This tendency is confirmed because more than half of the 10000 transactions, approximately 6700 transactions, were included in Hyperledger, Multichain, and PostgreSQL.

The results of the evaluation showed that if the BC set contains the same deployment type, the selection process is able to correctly minimize costs (*e.g.,* not selecting Bitcoin) or maximize performance (*e.g.,* selecting a generic database). However, if the BC set contains both private and public BCs, the priority is always given to private BCs because they present a higher TPS in comparison to public BCs. Moreover, the prices of interacting with public BCs must be taken into consideration when selecting this deployment type. Sending a transaction to a public BC for each sensor measurement is unfeasible, as the cryptocurrencies prices are higher compared to private BCs. Thus, measurements should be combined and only an average from the last *n* measurements included in a transaction to minimize costs.

Even though private BCs are prioritized in the selection mechanism, one must take into consideration the degree of centralization that this type of deployment introduces. As described in Section 3.3.1, private BCs are controlled by a single entity or a group of trusted entities. Thus, are not decentralized, and the property of immutability might be broken if these entities collude or the central entity behaves maliciously.

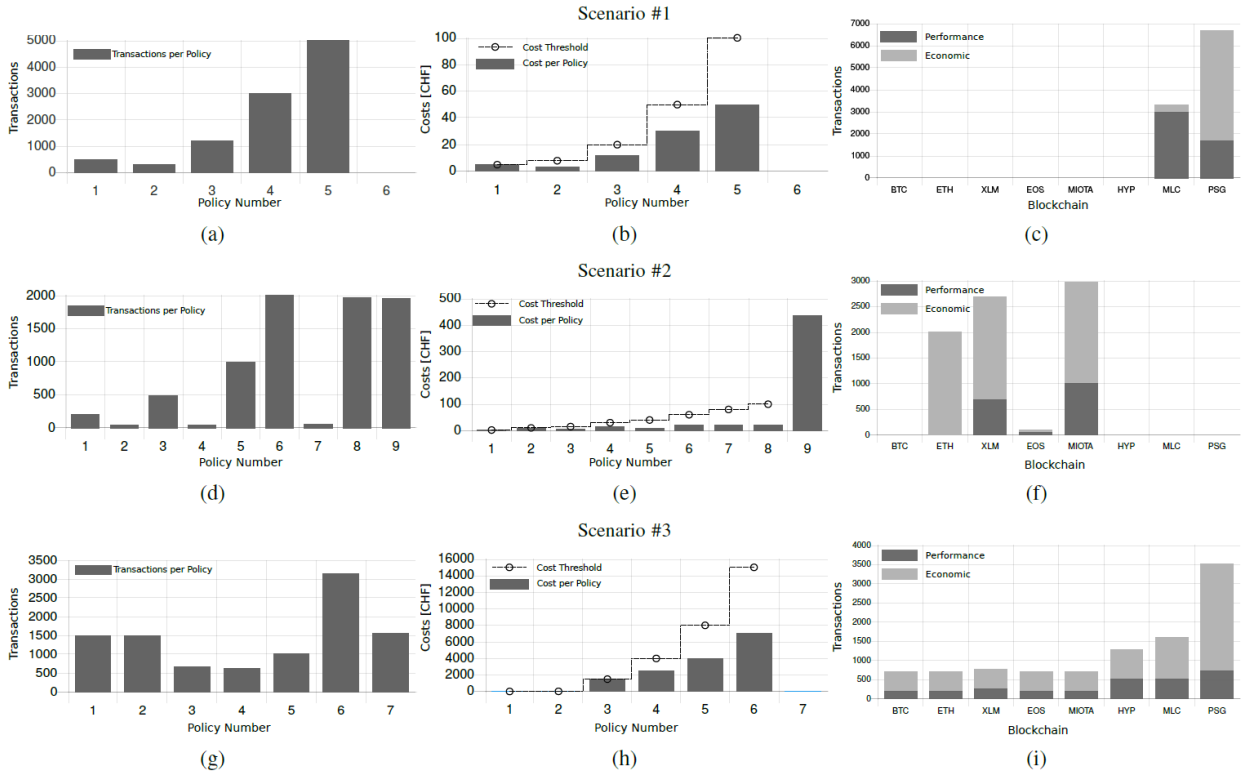

Figure 19: Comparison of the Results in the Evaluation Scenarios, Where (a), (d) and (g) Represent Transactions Executed per Policy, (b), (e) and (h) Represent Costs per Policy, and (c), (f) and (i) Represent the Transactions Stored per BC

6.4 Security Evaluation

The overall framework deployment (described in Section 4.4.1) follows a centralized approach, *i.e.,* a central server hosts the BC4CC application and the BC nodes. Further, it maintains the private keys of the BC addresses. Thus, being a point of interest to attacks, in this sense, a risk analysis and security of the deployment was conducted, which is described in Section 6.4.1. Further, the security of the device that monitors the temperature of the goods was evaluated in Section 6.4.2.

6.4.1 Risk Analysis and Security Audit

This section describes the most critical attack vectors for standard web application. In order to conduct the survey, the following sources were researched for possible attack vectors: Open Web Application Security Project (OWASP) Top 10 Web Application Security Risks [54], NIST Server Hardening guide [55], and Flask Security Guideline [56]. Further, it presents the conducted risk analysis and security audit.

6.4.1.1 Attack Vectors

The Open Web Application Security Project (OWASP) is an online community and non-profit organization founded in 2001 with the goal of producing freely-available content on the topic of web application security. Since its inception it has become the de-facto standard in the field, with other reputable entities, for example, the National Institute of Standards and Technology (NIST) or Payment Card Industry (PCI) Security Standards

Council [57] regularly referencing OWASP's work as an integral step to mitigating web application security risks.

The OWASP Top 10 focuses on identifying the most serious web application risks in broad terms, but each organization is unique. As such, it is important to develop a risk analysis to accurately determine the level of risk of a system. In general, OWASP presents the most comprehensive survey on attack vectors, in particular when considering the previous versions as well. The latest version of the Top 10 was released in 2017 and contains the following items:

1. **Injection**. An injection occurs when non-trusted data is sent to an interpreter as part of a command or query. This data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
2. **Broken Authentication**. Authentication mechanisms are often very complex. Mistakes in the implementation or configurations may expose the application to the risk of attackers compromising passwords, key, or session tokens, or impersonating other users.
3. **Sensitive Data Exposure.** Insufficient controls may often result in attackers obtaining sensitive data, such as financial, healthcare, and Personally Identifiable Information (PII).
4. **XML External Entities (XXE).** XML documents may refer to external entities such as file system locations, external services, etc. A poorly configured XML processor may attempt to evaluate these entities which can then be used to disclose sensitive information, to cause a Denial-of-Service (DoS), or perform a Remote Code Execution (RCE) attack.
5. **Broken Access Control.** Closely related to *Broken Authentication*, implementation and configuration errors might make it possible for attackers to access unauthorized functionality or data. Examples include viewing and/or changing another user's data and sensitive files, and change access rights.
6. **Security Misconfiguration.** The most common risk, the result of insecure configurations might often end up exposing sensitive information, *e.g.,* exposing an unencrypted database service with a weak password. Further, software at all levels should regularly be patched and upgraded.
7. **Cross-Site Scripting (XSS).** An XSS flaw is a type of injection attack in which non-trusted code is injected in a web page without validation or escaping. This allows attackers to execute scripts in the victim's browser which can result in session hijacking, defacement, or redirecting users to malicious sites.
8. **Insecure De-serialization.** In many programming languages and frameworks, incoming request payloads (often JSON or XML) are transformed from byte streams or strings into an object. This process is called de-serialization. Through implementation errors or misconfiguration, it may be possible to abuse this de-serialization to run the attackers code or cause a DoS.
9. **Using Components with Known Vulnerabilities**. Often exploits for commonly used libraries and frameworks become readily available. Using software with known vulnerabilities may facilitate compromise and reduce the effectiveness of other controls.

10. **Insufficient Logging and Monitoring.** Insufficient logging and monitoring make it easier for attackers to attack a system, maintain persistence, pivot to more system, and tamper, extract or destroy data. Most breaches studies show time to detect a breach is over 200 days, typically detected by external parties.

Notable mentions from previous OWASP Top 10 releases are **Cross-Site Request Forgery (CSRF)** and **Unvalidated Redirects and Forwards.** A CSRF attack forces the victim's browser to send an authenticated request on behalf of the victim without the victim's knowledge to the target application. CSRF attacks leverage the fact that browsers automatically include cookies and authorization headers when sending a request to an application it has previously visited. **Unvalidated Redirects and Forwards**, in which applications often redirect users to other locations. Often this happens based on requests parameter (*e.g.,* OAuth2's `redirect_uri` parameter). When these redirect parameters are not validated, an attacker can trick victims in browsing web pages controlled by the adversary. The fact that these redirect links come from the application itself helps in giving credibility to the malicious pages.

Most recommendations on Flask's security Considerations page and NIST's Server Hardening guide, refer to secure configuration aspects. These would fall under OWASP's Top 10 Security Misconfiguration presented previously.

6.4.1.2 Security Analysis

Threat modeling is a process by which we attempt to draw a profile of probable attackers, the most likely attack vectors, and the assets most desired by an attacker [58]. Risk Analysis is then the process of identifying and estimating risks to the identified assets, given profiles of attackers [59].

Cybersecurity is by nature asymmetric. Considering as an example an email service in which only legitimate users can access only their own mailboxes: even such a system can be composed of various subsystems, such as a front-end, database, access control components, and email reading and sending components. An adversary has numerous possibilities for attacking the system. Any subcomponent could be compromised independently from each other. An attacker for example might attack the front-end, injecting code, which when executed in the context of a legitimate user's browser, leaks information, or the attacker might exploit a vulnerability in the operating system.

In contrast, engineers developing and implementing security measures must consider the security of the entire system. Covering all possible attack scenarios is simply not feasible. Thus, to discuss attack surface and attack vectors, first it is necessary to define, which are the components to protect, and the motivation and skill level of possible attackers, in order to assess the probability and impact of an incident happening.

Firstly, it is necessary to define the scope of the system that is going to be protected, which determines the scope of the following risk analysis. The scope of the system is defined in Section 4.4.1.

Secondly, it is necessary to define the stakeholders. A stakeholder is any individual or group which is influenced by a platform's successes or failures. A stakeholder while being invested in the success of a system might also be a threat source. The following stakeholder were identified:

- **Normal Users.** A normal user can use the application through the front-end or the API and only see their own transactions or a transaction in case they have the transaction id. A normal user can be an employee of a logistics company or authorities auditing the cold chain.
- **Admin User**. In addition to the permissions of a normal user, can create new users and view any transaction on the platform.
- **System administrator.** System administrators of the platform do not have access to the application directly, but manages the infrastructure.
- **Communication Systems Group (CSG).** The CSG developed the prototype and is interested in its smooth and secure operation.

Thirdly, the *assets* present in the system and their relevance to security are discussed. The ISO 27000 standard defines an *asset* simply as *"any item that has value to an organization"* [60]. This does not simply include information, as one might be inclined to think at first thought, but can take many forms such as: information, software, physical assets (*e.g.,* computers), services, people, and intangible assets (*e.g.,* reputation).

Usually, assets are subdivided in different categories such as physical, logical, persons, and intangible goods [61]. Inside the same categories we often find the same type of vulnerabilities; thus, these are grouped accordingly. Assets are then, associated to a certain state, depending on whether it complies with security requirements.

Logical assets can be divided into two categories, *(a)* software, and *(b)* information [61]. Software includes operating systems and application software. All software can either be up-to-date; thus, patched with all available patches for known vulnerabilities or outdated. Information includes all data that is of value to stakeholders. The state of information can be determined by the security property associated with the information. For example, if a required property is availability, its state can be negatively affected if a legitimate user cannot access the information. Examples of information *assets* are the following:

- **Username & Passwords for the application or access tokens.** While all transactions are not confidential (assuming they happen on public BCs), access to credentials allows to act on the behalf of the user, and as such should remain confidential.
- **Username & Passwords or access keys to the system components.** Should only be known to the system administrator and should also remain confidential.
- **Transaction information.** Integrity is guaranteed by the BCs. Regarding confidentiality, we should distinguish between transactions happening on public BCs and private ledgers:
    - **Public BC.** Information has to be available to anyone per as per BC definition. The availability is guaranteed by the decentralization of the BC itself, in case the BC4CC application went offline.
    - **Private ledger.** Information has to be available only to authorized users. Here the application has to apply access control to make sure, unauthorized users cannot access transaction information. The confidentiality guarantees of the private ledger itself are beyond the scope of this audit. Again

availability would be guaranteed, however, to legitimate users by checking the ledger itself, in case the application was offline.

- **BCs private keys.** Private keys are even more critical than user credentials, as they allow to impersonate a user beyond the confines of the BC4CC application directly on the BC. As such, their confidentiality is of utmost importance. Additionally, compromised credentials can be changed, compromised private keys cannot.
- **Backups.** In an incident, backups are necessary to avoid losing state and resuming operations as quickly as possible. Backups should be stored encrypted and only be accessible by the system administrator. Considering the system stores essentially private keys to various cryptocurrencies, a loss or leak of backups, would have catastrophic consequences to the platform and its image, with the effect of a loss of customers.

Fourthly, the personnel involved in the development and operations of BC4CC must be identified, they are:

- **System administrators.** Maintain the servers and VMs. Administrators have access to all critical data on the system. They keep the whole system up to date and running and are responsible for the secure configuration of the platform.
- **Developers.** They have the task of continued development of the application and are inherently responsible for application security.

Fifthly, the intangible goods must be defined. Intangible assets are typically of qualitative nature and reflect the intended public image of the service provider.

- **Customer confidence.** Since the customer and auditors use BC4CC to ensure that no Service Level Agreement (SLA) violations took place, the integrity of the information provided is crucial to maintain customer confidence, which is a prerequisite for a successful business relationship.
- **Timeliness.** Data needs to be available when the customers require it, as there are often time restrictions in place.

Finally, the most relevant threat sources in the context of the prototype should be identified. They are the following:

- **Employees**. System administrators, developers and anyone who has physical access to the machines running BC4CC have to be taken into account (this includes for example also the cleaning and the maintenance team). Employees might have malicious intents (for example stealing the funds managed by the system), but also well-meaning employees may cause unintentional damage.
- **Amateur Hackers.** These adversaries usually only possess basic computer knowledge and use mainly known vulnerabilities for which exploits are publicly available. The reverse proxy is connected directly to the internet and is thus exposed to attacks by even amateur hackers. The web page and API are also indirectly connected. They may be motivated by the private keys to get to the cryptocurrencies stored or simply the challenge.
- **Skilled Hackers.** A skilled hacker has expert knowledge in applications and networking. They will usually carry out more complex attacks and might even use

unpublished vulnerabilities (zero-days). They may be hired by a competitor or act on their own. They may also act out of their own desire for glory and challenge.

- **Competitors.** Usually these adversaries do not carry out their own attacks but hire someone else to do them. They are usually interested in espionage, although damaging the image of the competition to sway potential customers might also be a goal.
- **Malware.** Although malware such as viruses or worms may be used by the other categories, there is still the possibility of undirected attacks, which is why it is also listed as its own source. The goal for undirected malware attacks might be to control as many machines as possible to create a botnet, which can later be used to launch distributed denial of service attacks. Or ransomware for monetary gain.

For the given system other possible threat sources might be *(a)* Organized crime or *(b)* Governmental Agencies.

The former *(a),* is typically motivated by monetary gain. However, unless BC4CC will hold millions in cryptocurrencies (or rather the private keys to the funds), it seems unlikely that organized crime would be interested. Assuming organized crime is using BC4CC to transport medical drugs and give legitimacy to their business, another possibility is that, they might be interested in stealing private key to fake their cold-chain data to lower transportation costs in order to undercut competitors.

While the latter *(b)* are usually motivated by espionage, surveillance, and criminal prosecution, BC4CC is a Business-to-Business (B2B) platform and as such it probably would not warrant much interest from the government, unless there is a suspicion of money laundering or tax fraud. Knowing this both actors would unlikely pose a threat to the system, not out of a lack of skills (it is rather the opposite, they would probably be the most skilled) but rather out of a lack of incentive.

6.4.1.3 Risk Analysis

The NIST defines risk as a function of the likelihood of a threat event happening, and the impact, the adverse effect, such an event has on the organization [59]. Thus, measures for both impact and likelihood, and the function by which to compute the resulting *risk* must be defined. Given the difficulty in assigning an absolute value to these measures, it was preferred to use a five-step qualitative scale as presented in Table 13 and Table 14.

In order to estimate the risk associated with an event, first, it must be defined, which the **impact** of this event is in case that it occurs. Table 13 presents the five steps of the impact severity.

Table 13: NIST Impact Definitions

Severity	Description
Very High	The event would have multiple severe or catastrophic adverse effects, in such a way that recovery might not be possible.
High	The event would have a severe or catastrophic adverse effect, in such a way *(1)* to cause a severe degradation or loss in mission capability; *(2)* cause major damage to assets and/or financial loss; or *(3)* result in human death or injury.
Moderate	The event would have a serious adverse effect, in such a way *(1)* to cause degradation in mission capability but its extent and duration would still allow an organization to perform its primary functions; *(2)* result in significant damage to assets and/or financial loss; or *(3)* result in significant human injury
Low	The event would have a limited adverse effect, in such a way *(1)* to cause degradation in mission capability but its extent and duration would still allow an organization to perform its primary functions *(2)* result in minor damage to assets and/or financial loss; or *(3)* result in minor harm to individuals.
Very low	The event would have negligible adverse effect.

Further, following [58], the **likelihood** of a threat event be initiated from an adversarial standpoint is defined in Table 14.

Table 14: NIST Likelihood Definitions

Severity	Description
Very High	The threat source is highly motivated and sufficiently capable and is almost certain to initiate a threat event. The controls put in place are ineffective.
High	The threat source is highly motivated and sufficiently capable and is highly likely to initiate a threat event. The controls put in place are ineffective
Moderate	The threat source is motivated and capable. The controls put in place might impede the adversary.
Low	The threat source lacks the motivation or is not capable of initiating a threat event. The controls put in place might severely impede the adversary.
Very low	The threat source is neither motivated nor capable of initiating a threat event. The controls put in place are effective.

Having defined all necessary parameters, a risk evaluation, following the examples found in [61], is conducted on the assets defined in Section 6.4.1.2. In total, 35 Threats (T) were identified, the complete list can be found in Annex 1. For each one of these threats, an **impact**, a **likelihood**, and a **risk**, was assigned, resulting in the risk matrix presented in Table 15. The matrix shows, that most risks are in the moderate-high region. These need to be addressed, either by accepting the risks or by introducing measures, before release. Considering the sensitivity of the data stored on the application, a high-level risk is to be expected.

Table 15: Risk Matrix

Likelihood/Impact	Very Low	Low	Moderate	High	Very High
Very High					
High	T18	T19, T22	T2, T3	T8	
Moderate		T20	T1, T16, T35	T7	T5, T6, T14, T15, T28, T29, T30, T31
Low		T25, T-26, T32	T17, T23, T34	T21, T23	T4,
Very Low				T24	T9, T10, T11

Table 16: Risk Matrix Legend

Color	Legend
	Very High - Not Acceptable. Risk reduction required.
	High - Risk reduction necessary or risk acceptance after further analysis by stakeholders.
	Moderate - Acceptable using ALARP. Consider further risk reduction.
	Low – Acceptable.
	Very Low – Acceptable.

6.4.1.4 Security Audit

This sections presents the security audit of the prototype deployed, which was carried out from September 2019 to December 2019.

The general architecture of the system is well suited for the requirements. The design is simple and usable and only the firewall being connected to the internet is a good way to minimize exposure. Simplicity is an important security property, often overlooked [62]. Simple systems are easier to analyze and are less likely to contain flaws than complex ones. The choice of having only a reverse proxy exposed to the internet would also simplify DoS protection. All traffic goes through the reverse proxy; thus, the proxy

could be extended to rate limit requests which do not contain a valid, session cookie, and freely let through the rest of the requests. If a logged-in user is misbehaving, this can be handled on the application layer by rate limiting user requests as well. If the load is such that even with the countermeasures in place, the reverse proxy cannot handle the traffic, additional proxies can be added for load balancing.

There are some improvement points possible. Communication inside the perimeter happens over HTTP, a channel which does not provide confidentiality: an adversary which sits behind the firewall, would be able to listen in on any traffic to the application, including user credentials and transaction information for private ledgers which should not be made public. Given that the university is a public institution, any number of people could enter the premise and depending on the internal UZH network configuration, access the CSG network from behind the firewall. Aside from passive traffic sniffing, also active MITM is possible, since HTTP is not an authenticated channel. A defense-in-depth approach would be to encrypt traffic even behind the perimeter to mitigate the risk.

Eventually, when the application is deployed to production, special care should be taken to protect the connection between the BC4CC server and the RPC nodes. For instance, as an active interference of this connection the application being unusable would render due to not being able to fulfill its functional requirements anymore. In this sense, a single connection to a BC node is a single point of failure in the system. A possibility would be to introduce multiple connections to various remote nodes, to avoid a possible separation attack between BC4CC and the various BCs. At the moment the nodes are running in docker containers with a direct interface to the BC4CC application, thus the connection is secure, in the sense that it cannot be interrupted or interfered with.

The storage of private keys could be further improved. At the moment all keys are stored as plain-text in the database. This is necessary in order to compute a transaction signature before sending the transaction to the BC. This does have the disadvantage that if an adversary were able to access the BC4CC server, all private keys would be compromised. A simple, yet effective, improvement is to delegate the signing operation to a further system which exposes an API to compute signatures and/or to use a tamper-proof hardware module to store the private keys that handles transaction signing. This approach can be seen in Figure 20. This brings two advantages, *(1)* a compromise of the BC4CC server does not compromise the private keys, and *(2)* the business logic becomes simpler, which means that it is easier to reason about, which in turn increases the security of both the BC4CC and Private Keys Server.

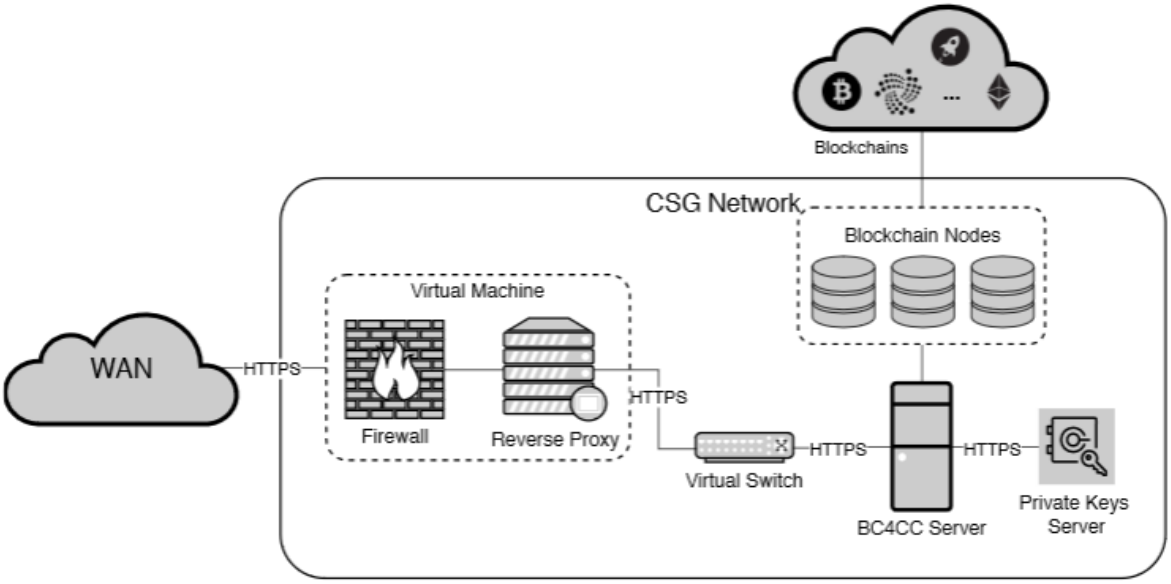

Figure 20: Separation of the Signing Component from the BC4CC Server

Additionally, BC4CC could employ a hot-wallet, cold-wallet approach. Two distinct sets of private keys are needed. The first set would be further stored as plain-text and directly accessible to BC4CC. The second set would be stored offline, in a hardware wallet. The majority of the cryptocurrencies would be stored in the cold-wallet. As the funds in the hot-wallet are used, more can be transferred from the cold-wallet in a controlled manner. Both approaches can be used in parallel in order to further reduce risk of compromise.

The complete analysis of vulnerabilities found on the reverse proxy and the BC4CC server can be found in Annex 1. Each finding is assigned a score, which represents a subjective opinion on the severity of the finding as well as the cost-effectiveness of implementing the measure. However, two major ones are described in the next paragraphs.

The first **Cross-Site Request Forgery (CSRF)** is an attack in which an adversary tries to carry out a legal operation in the name of another user [54]. An example is effecting a payment or changing a password. These are operations a user is authorized to do. If an adversary is able in changing the password of a user, he has effectively taken over the account. For the attack to be successful the following conditions need to be fulfilled:

- The user needs to have a valid session/be logged in at the time of the attack.
- The attacked needs to lure the user to a specially crafted website.

As soon as the user lands on the attacker's website, the "*unintended*" request is sent to the server. As the user already has a valid session, the browser sends the session token, stored in the cookie, to the server as well and the request is authorized.

```
fetch('https://bc4cc.ddns.net/api/createclient', {
    credentials: 'include',
    method: 'POST',
    body: 'username=csrf&password=test' +
    '&name=test&email=test%40test.com&phone=test',
    headers: {
          'Accept': 'text/html',
          'Content-Type': 'application/x-www-form-urlencoded'
    }
});
```

Listing 6: CSRF Exploit Code

In the case of BC4CC, if an authenticated admin user goes to a website containing the JavaScript code contained in Listing 6, a new user account will be created without the knowledge of the user. Thus, CSRF protection has to make sure that an operation is carried by the user on purpose. This may be achieved if for every request of the browser in the response of the server, an additional token is returned. This token must not be stored as cookie, otherwise it will always be included by default as the session cookie. This token needs to be reused with the next request. The server verifies whether the token in the new request matches with the token in the previous response. This procedure makes sure that the user triggers the operation on purpose.

The second key concern is when using passwords for authentication, is how to verify passwords. The naive solution is to simply store passwords in clear text and check if they match on login requests. This has the disadvantage that if the user database is ever leaked, all credentials are immediately compromised. A better approach is to compute a cryptographic hash function of a password and store only the hash in the database. Knowledge of the hashing algorithm and clear text password, provided by the user on login (which is immediately discarded after verification), is enough to authenticate the user. Additionally, a *salt,* a per-password fixed-length random value, should be appended to the password when computing the hash. This has two goals:

1. Two identical passwords, do not appear the same once hashed.
2. Increase entropy with the aim of making pre-computed lookup attacks not feasible.

The hashing function used should not be too fast to compute, thus enabling a brute-force attack, while still being quick enough to be usable for legitimate purposes: if a hashing functions is too fast, an adversary can pre-compute lookup tables using word-lists and then use these to derive the user credentials; if it is too slow the application can become unusable and in extreme case enable DoS attacks. Special purpose-built adaptive algorithms like BCrypt and Argon2 are designed to be memory- and time-hard in order to make pre-computation attacks intractable [63].

In the BC4CC prototype, passwords are stored hashed using SHA-512 without a salt. SHA-512 while being a cryptographic hash, was designed to be fast for signature

verification, which makes it a poor candidate for password storage, indeed, SHA-512 is 6 orders of magnitude faster than BCrypt [64].

6.4.1.5 Risk Analysis and Security Audit Summary

The security audit has produced medium results within the scope of the project. The architecture is well-thought but there is margin for improvement. Handling of the private keys that give access to the cryptocurrencies needs to be hardened. A leak of the keys, has the potential for financial damage and DoS, which would have severe consequences on the customer's confidence in the platform, directly influencing the possibility of success of the project. This document presents possible improvements on the existing architecture which would greatly impede an adversary that attempts to access the keys or limit the damage an incident could cause. Hardening measures aside, there are no fundamental errors in the design of the BC4CC application.

The implemented application however does present some issues. The most critical issue is the possibility of executing Cross-Site Request Forgery (CSRF). Through CSRF it is possible to create unverified user accounts, as well as performing transactions on behalf of the user. Creating unverified user accounts, removes any traceability of the operations occurred, and makes it impossible to recover any funds the user might have consumed. By generating enough CSRF requests the adversary has the possibility of locking a user out of the application by triggering the rate limiter. The CSRF vulnerability hurts the availability and non-repudiation security properties, makes any form of accounting impossible, as well as causing severe financial damage. All complete findings as well as the architecture review can be found in Annex 1.

6.4.2 Security Whitepaper Summary

The security whitepaper for IoT-Enabled Hardware Temperature Loggers (cf. Annex 2) has shown that the IoT device and Smart Contracts are the critical components of the system. Edge-to-Chain security can only be achieved by having digital signatures on the device itself. While certain risks remain, almost all attacks can be mitigated. Due to the link to the real-world, there are security concerns and risks that cannot be tackled at a technical level alone.

One of the solutions to overcome those risks is auditing the Modum.io system and Smart Contracts. This allows for trust that the data on the chain cannot be mutated nor be originating from a faked device. There are however attack vectors on the user facing applications, such as the GUI based front-end application or reports generated for other systems. These limitations can only be overcome by verifying data against data stored in the chain.

6.4.2.1 Logger Device

Each logger is equipped with a Hardware Security Module (HSM). The HSM is designed to generate and store secret keys and to resist tampering, decapping and probing so that these keys cannot be retrieved. This tamper-resistance is assumed to be unbreakable in this work and, thus, the secret keys irretrievable.

The advantages of using the HSM over implementing cryptography in software on the MCU are as follows:

- Prohibit cloning of loggers. I.e. if cryptography is implemented in software on the MCU, stealing a logger's secret keys allows producing an arbitrary number of fake loggers that produce authentic signatures using the same secret keys. If a HSM is used, even if all other logger components can be subverted and the HSM can be made to sign arbitrary temperature measurements, only one logger device is affected. No clones can be manufactured since the secret keys cannot be retrieved from the HSM.
- Cryptographic computations may be more power efficient since they are accelerated by dedicated hardware.
- Cryptographic computations may be faster since they are accelerated by dedicated hardware.
- The burden of correctly implementing cryptographic algorithms does not lie with modum.io.

The MCU has to authenticate itself to the HSM for every use of the HSM before it can be used (this lasts until the MCU cuts power to the HSM or terminates the secure channel session). The HSM also authenticates itself toward the MCU (mutual authentication) to prevent man-in-the-middle attacks. A pre-shared symmetric key and challenge-response are used for authentication. The pre-shared-key is different for each logger and HSM. The key is only kept in RAM. Thus, a logger reset kills the key, yielding reverse-engineering efforts harder. This risk analysis covers the risks of an operator or attacker attempting to remove unfavorable measurements from the temperature measurements log or attempting to modify the logger device to not record unfavorable measurements. Table 17shows a number of Logger Manipulation Risks that could occur, while Table 18 shows the different mitigation steps that could be taken against them.

Table 17: Risks of Manipulation

No.	Description	Mitigation Steps
1	Change temperature data before upload to Smart Contract	1
2	Upload temperature data from a different logger	2
3	Replay old temperature data from the same logger	3,4
4	Separate logger from shipment	
5	Switching loggers	2,3
6	Restarting logger (with different parameters)	3,4,5
7	Replaying previous BLE communication	4,6
8	Critical bug in logger firmware	14
9	Resoldering HSM onto different logger (or fake logger)	(7),(13),(16),(17),(18)
10	Reverse-engineering firmware dump	(7),(16)
11	Reflashing a logger with new firmware	(17),(18)
12	Cloning a logger	10,17,18
13	Modifying flash memory of a logger	(17),(18)
14	Obtaining a firmware binary dump / full flash dump	(15),(17),(18)
15	Manipulation of measurements in flash	12,(17),(18)
16	Readout of data in RAM	(17),(18)
17	Manipulation of data in RAM	(17),(18)
18	Replacement of components	9,(17),(18)
19	Manipulation of data on the bus	9,13,(17),(18)
20	Tampering with the quartz	(17),(18)
21	Replacing the modum.io master key certificate	11,(17),(18)
22	Manipulate measurements of a single shipment	12,(17),(18)
23	Craft a modified logger from the given hardware where all safeguards are disabled and measurements for each shipment can be modified easily (for internal use or with intent to sell it to interested logistics companies)	(10),(16),(17),(18)
24	Craft a large number of non-genuine loggers that look and function like the modum.io logger, but where all safeguards are disabled and measurements for each shipment can be modified easily	10,(17),(18)
25	Reuse the logger hardware for a completely different purpose (e.g. home automation)	(17),(18)

Table 18: Mitigation Steps for Logger Risks

No.	Description	Mitigates Risks No.
1	Signing data on the logger	1
2	Each logger has an individual key pair	2,5
3	The data signed on the logger for a shipment includes all shipment parameters	3,5,6
4	The logger generates a random, unique shipment identifier every time a shipment parameter changes, as well as when a shipment is started	3,5,6,7
5	Shipment parameters can only be changed when the logger is not recording	6
6	Time syncing is performed through a protocol resistant to replay attacks	7
7	A freshly rebooted logger cannot be used for shipment tracking	(9),(10)
8	The recording enable characteristic write is optionally authenticated	
9	Implausible measurements are cross-checked with other sensors	18,19
10	The ECC key pair private key is stored on a HSM	12,(23),24
11	The modum.io master key certificate is stored on the HSM	21
12	Measurements are hashed in real time on the HSM	15,22
13	Bus communication between the MCU and the HSM is authenticated and encrypted	(9),19
14	Independent code review for logger firmware	8
15	All cryptographic operations are computed on the HSM	(14)
16	The key used to authenticate the MCU toward the HSM is different for each logger	(9),(10),(23)
17	Tamper-evident casing	(9),(11),(12),(13),(14),(15),(16),(17),(18),(19),(20),(21),(22),(23),(24),(25)
18	Trace loggers until recycling	(9),(11),(12),(13),(14),(15),(16),(17),(18),(19),(20),(21),(22),(23),(24),(25)

6.4.2.2 Smart Contract

In the modum.io system one Smart Contract is created per shipment. The same contract code is always used; which auditors should regularly check by comparing against their own compiled binary. The public modum.io master key is hard-coded in the Smart Contract and the contract code can verify the authenticity of the uploaded data along the chain of trust back to the master key.

During construction, the Smart Contract is supplied a hash of all shipment parameters that are known at the beginning of the shipment, which is stored immutably in

the contract state. This includes tracking number, interval, alarm criteria, nonce, start timestamp, fingerprint of the time stamping server certificate, and the starter operator ID. At the end of a shipment a compliance decision is computed via a call of a constant method.

Then, via non-constant method call, the Smart Contract is supplied a hash of all shipment data, the previously computed compliance decision and a signature of the hash (and timestamp). All of these are stored immutably in the contract state, *i.e.*, this function call is only allowed once per Smart Contract. A constant method call allows verification of the stored hashes against the confidential shipment data, the certificate of the logger that signed the data and the provisioning certificate completing the chain-of-trust back to the master key. If the hashes do not match the shipment parameters or shipment data, if the signature isn't valid for the temperature report hash, if the certificate cannot be verified along the chain to the master key, or if the stored compliance decision does not match the shipment data, an exception is thrown and the auditor should investigate the issue.

Table 19: Risks of Manipulation of the Smart Contract

No.	Description	Mitigation Steps
28	Loss of Ethereum availability	(19),21
29	Loss of confidentiality through Smart Contract data	22
30	Compromise of the owner key of the Smart Contract	19,20,23
31	Hack / buggy contract code	21,23,24,26,30
32	Temperature data can be overwritten	25
33	Modification of the master key	23
34	Duplicates of the modum.io contract are deployed	20,23
35	Replay of old data	27,29
36	Hard fork	19,28

Table 20: Mitigation Steps for Smart Contract Risks

No.	Description	Mitigates Risks No.
19	Shipment data is stored immutably in the Smart Contract	(28),30,36
20	The Smart Contract checks the signature of uploaded data and validity of the logger certificate	30,34
21	Modum.io system is Blockchain agnostic	28,31
22	No non-hashed data stored in the Smart Contract	29
23	Regular checking of Smart Contract instances	30,31,33,34
24	Monitoring Ethereum / Blockchain news channels	31
25	Spot testing Smart Contract data	32
26	Local evaluation of shipment data	31
27	Checking for hash collisions	35
28	Stakeholder mentoring	36
29	Checking for orphaned Smart Contracts	35
30	Smart Contract code is publically auditable	31

Table 19 shows the manipulation risks in connection to the Smart Contract and Table 20 shows the Mitigation Steps for those risks. A more detailed description of the risks and mitigation steps as shown in Section 6.4.2.1 and Section 6.4.2.2 can be found in Annex 2.

6.5 Business Economic Impact Evaluation

In order to evaluate the business economic impact, the total transactions per day is estimated to be around 2000 and the data size to be included in the BC is 68 Byte. With the help of the transaction fee function derived for each of the BC4CC public BC, the fee per transaction is calculated and the total fee for the 2000 transactions is estimated and shown in Figure 21 as an overall comparison among the BC4CC public BCs. Private BCs were not compared because they do not present any transaction costs, besides the maintenance of the BC node hardware, which is outside of the scope of this evaluation.

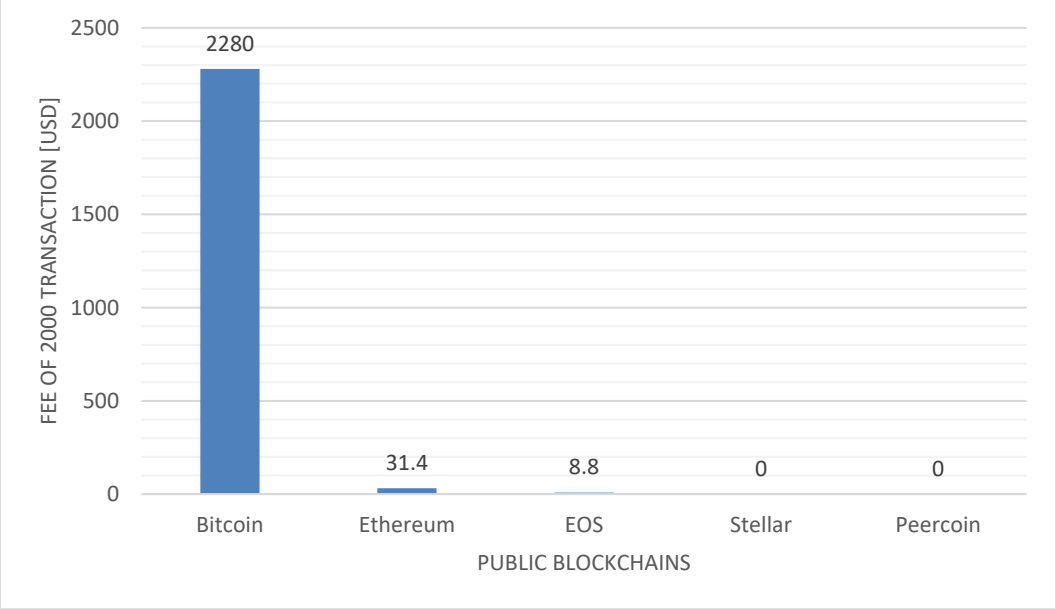

Figure 21: Transaction Fee Economic Impact

6.5.1 Bitcoin

The value of Bitcoin per byte on July 5, 2019 was 0.00000058 BTC/Byte. This value is used to calculate the fee per transaction. Based on the equations below, the estimation of the fee for 2000 transactions was 2280 USD.

$$Transaction\ size = default\ transaction\ size + size\ of\ data$$
$$Transaction\ size = 106 + 68 = 174\ Byte$$
$$Fee\ per\ transaction\ in\ Bitcoin = (174 \times 0.00000058) = 0.00010092\ BTC$$

$$Fee\ per\ transaction\ in\ Dollar = 1.14\ USD\ as\ of\ 5th\ July\ 2019$$

$$Fee\ for\ 2000\ transactions = 2000 \times 1.14 = 2280\ USD$$

6.5.2 Ethereum

In Ethereum, 68 gas is paid for every byte of data for a transaction [15]. Since the data to be included in the BC is 68 Byte, the gas consumed equals 4624 gas. 21000 unit of gas is consumed as a base fee for any transaction.

$$Gas\ consumed\ in\ a\ transaction = base\ gas + (68 \times 68)$$

$$Gas\ consumed\ in\ a\ transaction = 21000 + 4624 = 25624\ gas$$

The average gas price of 2.1 G*wei* (*i.e.,* 2100000000 *wei*) is used to calculate the transaction cost. As of July 5, 2019 the value of 53810.4 G*wei* equals 0.0000538104 Ether. Based on the equations below, the estimation of the fee for 2000 transactions was 31.4 USD.

$$Fee\ per\ transaction\ Gwei = 25624 \times 2.1 = 53810.4\ Gwei$$

$$Fee\ per\ transaction\ in\ Ether = 0.0000538104$$

$$Fee\ per\ transaction\ in\ Dollar = 0.0157\ USD$$

$$Fee\ for\ 2000\ transactions = 31.4\ USD$$

6.5.3 EOS

The value of current network price July 9, 2019 was 0.00036709 EOS/KiB/Day. This value was used to calculate the net usage fee. The value of the CPU price on July 9, 2019 was 0.00199012 EOS/ms/Day. Based on the equations below, the estimation of the fee for 2000 transactions was 8.8 USD.

$$Net\ usage\ fee\ per\ transaction = ((128 + size\ of\ data) \times current\ network\ price) \div 1024$$

$$CPU\ usage\ fee\ per\ transaction = ((350\ \mu s) \times current\ cpu\ price) \div 1000$$

$$CPU\ usage\ fee\ per\ transaction = (350\ \mu s \times 0.00199012) \div 1000 = 0.0006965$$

$$Total\ fee\ per\ transaction\ in\ EOS = Net\ usage\ fee\ per\ transaction + CPU\ usage\ fee\ per\ transaction$$

$$Total\ fee\ per\ transaction\ in\ EOS = 0.00007026 + 0.0006965 = 0.00076676\ EOS$$

$$Fee\ per\ transaction\ in\ Dollar = 0.0044\ USD$$

$$Fee\ for\ 2000\ transactions = 8.8\ USD$$

6.5.4 Stellar

As in Stellar the calculation for the fees are based on operations, and the store of data in a transaction requires one operation, which is the “payment” operation, the base fee is neglected. However, there is the need to maintain a minimum balance. Thus, based on the equations below, the estimation of the fee for 2000 transactions was 0.1 USD.

$$Transaction\ fee\ per\ transaction = number\ of\ operations \times base\ fee$$

$$Number\ of\ operations = 1$$

$$Transaction\ fee\ per\ transaction = 1 \times 0.00001\ XLM = 0.00001\ XLM$$

$$Transaction\ fee\ per\ transaction\ in\ Dollar = 0\ USD$$

$$Minimum\ balance = (2 + number\ of\ entries) \times base\ reserve$$

$$number\ of\ entries = 0$$

$$Minimum\ balance = 2 \times 0.5\ XLM = 1\ XLM = 0.10\ USD$$

6.5.5 Peercoin

Peercoin follows a calculation based on the transaction size and the size of data in the transaction, which is similar to Bitcoin. However, the values are lower than Bitcoin. Thus, based on the equations below, the estimation of the fee for 2000 transactions was 0.0 USD.

$$Transaction\ size = default\ transaction\ size + size\ of\ data$$

$$Transaction\ size = 196 + 68$$

$$Fee\ per\ transaction\ in\ PPC = (Transaction\ size \times 0.01) \div 1024$$

$$Fee\ per\ transaction\ in\ PPC = (264 \times 0.01) \div 1024 = 0.00257$$

$$Fee\ per\ transaction\ in\ Dollar = 0\ USD$$

$$Fee\ for\ 2000\ transactions = 0\ USD$$

6.5.6 Summary of the Business Evaluation

This business evaluation presents the solution to estimate the transaction fee of BC4CC public BCs. Various factors affecting the transaction fee are identified and also the methods and functions used to estimate the fee per transaction for the BC4CC public BCs are shown. There are various methods to calculate the fee per transaction. However, in this report the functions provided by the respective API of the BC adapter are used to estimate the transaction fee. By analyzing the various factors affecting the transaction fee, the private BC tends to be the most fee-effective way as there is no transaction fee for the transaction validation. Suggestions to minimize the transaction fee would be the initiator of

the transaction should monitor the average BC transaction fee in the respective network, if the transaction is not time sensitive, then the initiator can wait until the average fee drops. The sender can check the average fee on block explorers or websites.

Thus, the evaluation underlines the reason why many companies choose private BCs over public ones. While the transaction costs of public BCs can amount to quite a large sum, as seen with Bitcoin, private BCs do not have any transaction costs. This is certainly much more attractive to potential users, which do not want to or cannot wait for fees to drop. Moreover, despite the fact that the lesser known BCs are much less expensive, new users are more likely to consider the BCs they know, such as Bitcoin or Ethereum, and be discouraged by the costs.

7 Summary and Conclusions

The BC-agnostic BC4CC system as developed, prototyped, and evaluated proves to be much more effective than a single public BC integration, since this solution does not only offer interoperability, but the ideal BC can be chosen automatically depending on the needs specified by customers in advance. However, the current demand request for BC-based data sharing is lower at the time of writing. Many organizations still struggle to adopt public BC technology; may it be due to technical concerns or the lack of perceived benefits. Instead of using public BC technology, there is a trend in the industry to build consortia or even enterprise approaches. Within the space of BC4CC, there are two consortia-based approaches relevant: PharmaLedger and MediLedger. Both strive to provide a BC-based framework in which widely trusted, BC-enabled healthcare solutions can be built. These networks are not open – in contrast to the BC4CC Open API – and can only be joined by known entities, therefore, providing a different approach to solve a comparable set of problems BC4CC was addressing successfully inits way. To summarize from an Industry perspective, the BC4CC approach allows for a simpler adoption of the BC and provides a measurable benefit compared to an individual public BC integration. This comes at the cost of a challenge with respect to *(1)* the status quo and *(2)* non-public, consortia-based approaches for specific applications.

From an academic point of view, the BC4CC project presented several research challenges during the 26 months in which the project was executed. Thus, it supported the development of novel and applicable solutions, pushing the state-of-the-art in a myriad of areas, ranging from the BCs, via Internet-of-Things (IoT), to network management. Such results have been verified on its own and via respective metrics – as outlined in this report – and they are visible by the number and the excellent quality of the student thesis developed in the context of the project, which comprised two Master thesis (Timo Hegnauer and Daniel Lakic), one Master Basis Module developed by Patrick Wider, two Independent Studies from Alexander Hofmann and Andreas Knecht, and a Master Project developed by Andreas Knecht and Tim Strasser; all defined, supervised, and evaluated by UZH, the research partner. Thus, even an educational perspective of this BC4CC project became measurable. Moreover, the research dimension of the BC4CC project is also partially embedded in the PhD thesis of Eder John Scheid.

The overall excellent quality of the project outcomes was proven on an independent basis, too, especially via the accepted, presented, and published papers in several conferences with a high reputation and visibility, *i.e.,* the 16th IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019), the 44th IEEE Conference on

Local Computer Networks (LCN 2019), and the IFIP/IEEE Network Operations and Management Symposium (NOMS 2020).

7.1 Project Outcomes

The research of the BC4CC was disclosed in different well-known peer-reviewed conferences, carrying (either and both) the support of Institute of Electrical and Electronics Engineers (IEEE) and the International Federation for Information Processing (IFIP), in the form scientific publications (*i.e.,* papers). All the papers produced with this project's results targeted conferences in the networking area, focusing on network management and distributed systems, which are areas that this research can further contribute to the state-of-the-art.

7.1.1 IM 19 – Accepted and Published

The 16th IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019) was held in April 8-12 2019 in Washington DC, USA. Held in odd-numbered years since 1989, IM 2019 carries forward the 30+ years' tradition of NOMS and IM as the primary IEEE Communications Society's forum for technical exchange on management of information and communication technology, focusing on research, development, integration, standards, operational experiences, and user communities. IM had an acceptance rate of 28.6% in 2017, in which 154 papers were submitted and 44 accepted.

The paper entitled "Toward a Policy-Based Blockchain Agnostic Framework" was presented as a poster by Eder John Scheid. The paper described an overview of the whole BC4CC project and provided directions towards the policy-based BC selection and the interoperability API.

7.1.2 LCN 19 – Accepted and Published

The 44th IEEE Conference on Local Computer Networks (LCN 2019) was held in October 14-17, 2019 in Osnabrück, Germany. The IEEE LCN conference is the premier conference on the leading edge of theoretical and practical aspects of computer networking. LCN is a highly interactive conference that enables an effective interchange of results and ideas among researchers, users, and product developers. For the past 43 years, major developments from high-speed networks to the global Internet to specialized sensor networks have been reported at this conference. LCN 2019 had an acceptance rate of 28.9%, in which 135 papers were submitted and 39 accepted.

The paper entitled "Bifröst: a Modular Blockchain Interoperability API" was presented by Eder John Scheid. The paper disclosed the results of the research on the interoperability API, along with design and implementation aspects, and a discussion on the performance and security.

7.1.3 NOMS 2020 – Accepted and to be Published

The 2020 IEEE/IFIP Network Operations and Management Symposium (NOMS 2020) will be held in 20-24 April 2020 in Budapest, Hungary. NOMS has been held in every even-numbered year since 1988. This is the 32th anniversary of NOMS; and it is marked by the finalization of Horizon 2020 efforts - celebrating European innovation.

NOMS had an acceptance rate of 29.6% in 2018, in which 189 papers were submitted and 56 accepted.

The paper entitled “PleBeuS: a Policy-based Blockchain Selection Framework” was accepted for publication and will be presented by one of the project participants. The paper disclosed the results of the research on Policy-based BC selection, along with design and implementation aspects of the framework, and a discussion its feasibility.

7.1.4 Security-related White Paper

Andreas Knecht’s Whitepaper “Security Whitepaper for IoT-Enabled Hardware Temperature Logger” provides a blueprint on how data integrity can be achieved Edge-to-Chain. It shows the importance and limitations of considering the weakest link in the system, the device that captures the data, and how to overcome the limitations. This whitepaper was developed in the context of an Independent Study (IS) at the CSG.

7.1.5 Measurable modum.io Improvement

During the project, the Modum.io system migrated from a public BC based approach to a consortia-based approach, while only using the public BC technology for demonstration purposes. The benefits of the BC4CC have been shown on the demonstration system, but the impact on the productive system is limited at this point in time. BC4CC is, however, future-proof and enables Modum.io and other providers to quickly and easily migrate towards public networks if demanded by the industry.

7.2 Future Work

To build up on the existing work, the *PleBeuS* Framework can be extended by more criteria or more complex rules can be added. This would allow faster and more precise decision making when choosing the best suited BC technology. Further, in the *Bifröst* context, security improvements could be made, transaction-related error handling and communication to Smart Contracts can be implemented, and research toward developing a decentralized version of the solution, while standardizing the BC interaction is envisioned.

With the BC area still being young and fast-paced, changes in underlying technologies are unavoidable, therefore, maintaining different integrations is key to ensure the future success of the BC4CC approach. Other BC technologies, or support of more specific events, can be added, to further improve the functionality scope of Bifröst.

8 References

[1] E. J. Scheid, B. Rodrigues and B. Stiller, "Toward a Policy-based Blockchain Agnostic Framework," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019)*, Washington - DC, USA, 2019.

[2] E. Schiller, S. R. Niya, T. Surbeck and B. Stiller, "Scalable Transport Mechanisms for Blockchain IoT Applications," in *IEEE 44th LCN Symposium on Emerging Topics in Networking (LCN 2019 Symposium)*, 2019.

[3] B. Rodrigues, T. Bocek, A. Lareida, D. Hausheer, S. Rafati and B. Stiller, "A Blockchain-based Architecture for Collaborative DDoS Mitigation with Smart Contracts," in *IFIP International Conference on Autonomous Infrastructure, Management and Security (AIMS 2017)*, 2017.

[4] T. Bocek and B. Stiller, "Smart Contracts--Blockchains in the Wings," in *Digital Marketplaces Unleashed*, Springer, 2018, pp. 169-184.

[5] *modum.io Website.*

[6] Everledger, *Everledger - A Digital Global Ledger,* 2018.

[7] Provenance, *Provenance - Every Product has a Story,* 2018.

[8] IBM, *Blockchain for Supply Chain,* 2018.

[9] Microsoft, *How Blockchain will Transform the Modern Supply Chain,* 2018.

[10] U. Koester, *How Blockchain Can Transform Your Supply Chain Ecosystem,* 2018.

[11] E. J. Scheid, D. Lacik, B. Rodrigues and B. Stiller, "PleBeuS: a Policy-based Blockchain Selection Framework," in *IEEE/IFIP Network Operations and Management Symposium (NOMS 2020)*, Budapest, 2020.

[12] B. Rodrigues, T. Bocek and B. Stiller, "The Use of Blockchains: Application-Driven Analysis of Applicability," in *Advances in Computers*, Springer, 2018, pp. 163-198.

[13] V. Butarin, *Chain Interoperability,* 2016.

[14] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System,* 2009.

[15] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," *Ethereum project yellow paper,* vol. 151, pp. 1-32, 2014.

[16] CoinMarketCap, *CoinMarketCap Market Capitalizations,* 2019.

[17] D. C. Verma, "Simplifying Network Administration Using Policy-Based Management," in *IEEE Network*, 2002.

[18] J. Strassner, Policy-Based Network Management: Solutions for the Next Generation (The Morgan Kaufmann Series in Networking), San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[19] B. Moore, E. Ellesson, J. Strassner and A. Westerinen, *RFC 3060 - Policy Core Information Model.*

[20] J. S. L. N. C. K. S. S. D. N. A. M. Muhammad Saad, "Exploring the Attack Surface of Blockchain: A Systematic Overview," 6 April 2019. [Online]. Available: https://arxiv.org/abs/1904.03487. [Accessed 14 December 2019].

[21] B. R. B. S. M. Franco, "MENTOR: The Design and Evaluation of a Protection Services Recommender System," in *International Conference on Network and Service Management (CNSM 2019)*, Halifax, Canada, 2019.

[22] T. Bocek, B. B. Rodrigues, T. Strasser and B. Stiller, "Blockchains Everywhere - a Use-Case of Blockchains in the Pharma Supply-Chain," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2017)*, Lisbon, 2017.

[23] D. D. Team, *Transferring Data Through an Ethereum Blockchain Using Transactions,* 2018.

[24] D. Easley, M. O'Hara and S. Basu, "From Mining to Markets: The Evolution of Bitcoin Transaction Fees," *Journal of Financial Economics,* pp. 1-41, 2019.

[25] Wirex, *My blockchain fee is too high. Why? What should I do?,* 2019.

[26] E. J. Scheid, T. Hegnauer, B. Rodrigues and B. Stiller, "Bifröst: a Modular Blockchain Interoperability API," in *IEEE Conference on Local Computer Networks (LCN 2019)*, Osnabrück, Germany, 2019.

[27] Bitcoinfees, *Bitcoin Transaction Fees,* 2019.

[28] Bitcoinfees, *Bitcoinfees: Predicting Bitcoin fees for transactions,* 2019.

[29] C. Query, *Bitcoin API: estimatesmartfee,* 2019.

[30] E. Foundation, *Web3: web3.eth - web3.js 1.2.0 documentation,* 2016.

[31] E. O. S. Canada., *EOS Canada: Leading Block Producer,* 2019.

[32] E. O. S. N. York, *EOS Resource Planner,* 2019.

[33] S. D. Foundation, *Transactions | Stellar Developers,* 2019.

[34] S. King and S. Nadal, *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake,* 2012.

[35] Peercoin, *Peercoin Docs,* 2019.

[36] W3C, *Relationship to the World Wide Web and REST Architectures,* 2004.

[37] Etherchain, *Etherchain - The Ethereum Blockchain Explorer,* 2019.

[38] B. Inc., *Blockcypher: Blockchain Web Services,* 2018.

[39] A. Grigorean, *Latency and Finality in Different Cryptocurrencies,* 2018.

[40] "BC4CC API Documentation," Postman, 2019. [Online]. Available: https://documenter.getpostman.com/view/6990415/S17oyqaR?version=latest#ef5839a2-6c10-41d0-a57d-bc4867f449ea. [Accessed 5 12 2019].

[41] "PleBeuS: a Policy-based Blockchain Selection Framework," IFI Gitlab, 2019. [Online]. Available: https://gitlab.ifi.uzh.ch/scheid/plebeus. [Accessed 5 12 2019].

[42] N. Foundation, *A JavaScript runtime built on Chrome's V8 JavaScript engine,* 2019.

[43] D. Kahn, *What you should know to really understand the Node.js Event Loop,* 2017.

[44] L. *Cross-platform asynchronous I/O,* 2019.

[45] StrongLoop, IBM and other expressjs.com contributors, *Express - Node.js web application framework,* 2017.

[46] Mozilla, *Nunjucks - A rich and powerful templating language for JavaScript,* 2019.

[47] M. Inc., *The most popular database for modern apps,* 2019.

[48] Xplenty., *The SQL vs NoSQL Difference: MySQL vs MongoDB,* 2017.

[49] "Bifrost Code," IFI GitLab, 2019. [Online]. Available: https://gitlab.ifi.uzh.ch/scheid/bifrost. [Accessed 5 12 2019].

[50] SQLite, *SQLite Is Serverless,* 2018.

[51] PharmaLedger, "PharmaLedger - Blockchain Enabled Healthcare," [Online]. Available: https://pharmaledger.eu/. [Accessed 3 March 2020].

[52] MediLedger, "The MediLedger Project - An Open and Decentralized Network for the Pharmaceutical Supply Chain," [Online]. Available: https://www.mediledger.com/. [Accessed 3 March 2020].

[53] I. Corporation, *Intel SGX Homepage,* 2018.

[54] O. W. A. S. P. Foundation, *OWASP Top 10 Web Application Security Risks,* 2010.

[55] J. F. (NIST), G. H. (NIST) and K. B. (NIST), "Mobile Device Security: Corporate-Owned Personally-Enabled (COPE)," *NIST Special Publication,* vol. 1800, 2019.

[56] *Flask - Security Considerations.*

[57] P. C. I. S. S. C. Emerging Technologies, "PCI Mobile Payment Acceptance Security Guidelines for Developers," 2017.

[58] S. Myagmar, A. J. Lee and W. Yurcik, "Threat modeling as a basis for security requirements," in *Symposium on requirements engineering for information security (SREIS)*, 2005.

[59] J. T. F. T. Initiative and others, "Guide for conducting risk assessments," 2012.

[60] "Information technology - IT asset management - Part 1: IT asset management systems," Geneva, 2017.

[61] D. Basin, P. Schaller and M. Schläpfer, Applied information security: a hands-on approach, Springer Science & Business Media, 2011.

[62] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE,* vol. 63, pp. 1278-1308, 1975.

[63] P. A. Grassi, M. E. Garcia and J. L. Fenton, "Digital identity guidelines," *NIST special publication,* vol. 800, pp. 63-3, 2017.

[64] A. Aggarwal, P. Chaphekar and R. Mandrekar, "Cryptanalysis of bcrypt and SHA-512 using Distributed Processing over the Cloud," *International Journal of Computer Applications,* vol. 128, 2015.

[65] M. Iansiti and K. R. Lakhani, "The Truth About Blockchain," *Harvard Business Review,* vol. 95, pp. 118-127, 2017.

[66] I. Anati, S. Gueron, S. P. Johnson and V. R. Scarlata, *Innovative Technology for CPU Based Attestation and Sealing,* 2013.

9 Abbreviations

Short Form	Full Form
API	Application Programming Interface
BC	Blockchain
BLU	Bluetooth Low Energy
CRUD	Create-Read-Update-Delete
CPU	Central Processing Unit
CSG	Communication Systems Group
CSRF	Cross-Site Request Forgery
dApp	Distributed Application
DB	Database
DoS	Denial-of-Service
DDoS	Distributed Denial-of-Service
DLT	Distributed Ledger Technology
ECA	Event-Condition-Action
ER	Entity Relationship
EU	Europäische Union
GUI	Graphical User Interface
HSMS	Hardware Security Module
HTML	Hypertext Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocols
ID	Identification
IP	Internet Protocol
IS	Independent Study
I/O	Input/Output
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IFIP	International Federation for Information Processing
IPFS	InterPlanetary File System
ISO	International Organization for Standardization
IoT	Internet-of-Things
JIT	Just-in-time
JS	JavaScript
JSON	JavaScript Object Notation
LL	Linked List
MCU	Microcontroller Unit
NIST	National Institute of Standards and Technology
NFC	Near Field Communication
OS	Operational System
OWASP	Open Web Application Security Project
P2P	Peer-to-peer
PCI	Payment Card Industry

PDP	Policy Decision Point
PEP	Policy Enforcement Point
PMS	Policy Management System
PMT	Policy Management Tool
PBNM	Policy-Based Network Management
PBM	Policy-based Management
PoC	Proof-of-Concept
PoS	Proof-of-Stake
PoW	Proof-of-Work
QoS	Quality-of-Service
RAM	Random Access Memory
REST	Representational State Transfer
RPC	Remote Procedure Call
SHA	Secure Hash Algorithms
SLA	Service Level Agreement
SC	Smart Contract
SGX	Software Guard Extensions
SQL	Structured Query Language
SSL	Secure Sockets Layer
SSH	Secure Shell
TLS	Transport Layer Security
TPS	Transactions per Second
TTP	Trusted Third Party
UI	User Interface
URL	Uniform Resource Locator
USB	Universal Serial Bus
UZH	Universität Zürich

10 List of Figures

11 List of Tables

12 Acknowledgements

This final report was made possible due to all project partners' support, besides University of Zürich UZH and the modem.io AG, especially with their extensive willingness to discuss and provide commercial inputs Stefan Weber (formally modum.io), Simon Dösegger (CEO modum.io), and Marc Degen (Board Member, modum.io). Additionally, Thomas Bocek (formally UZH, now HSR Rapperswil and Board Member, modum.io) is gratefully acknowledged due to his support in initially writing the project proposal in 2017.

Additionally, thanks are addressed to Innosuisse, namely Dannie Jost, Scientific Officer, ICT, for her kind handling of formal administrative steps very reliably and in-time.

Finally, thanks are addressed to all students and persons who were directly involved with the progress and success of the project described here-in.

13 Annex 1 – Security Analysis of the Blockchain Agnostic Framework Prototype Report (PDF)

14 Annex 2 – Security Whitepaper for IoT-Enabled Hardware Temperature Logger (PDF)