electronics

Article

An Interoperable Blockchain Security Frameworks Based on Microservices and Smart Contract in IoT Environment

Khulud Salem Alshudukhi [1,2,*], Maher Ali Khemakhem [1], Fathy Elbouraey Eassa [1] and Kamal Mansur Jambi [1]

[1] Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Saudi Arabia
[2] Department of Computer Science, College of Computer and Information Science, Jouf University, Sakaka 72388, Saudi Arabia
* Correspondence: ksalshudukhi@stu.kau.edu.sa or ksalshudukhi@ju.edu.sa

Abstract: In the Internet of Things (IoT), technological developments have increased the significance of federated cloud systems with integrated cloud providers for exchange transactions. Monolithic IoT systems implement service-oriented architecture (SOA), which is complex for supporting scalability and communicating transactions in a federated cloud system. One weakness of conventional security methods is that they depend on a centralized party, which means there is a single point of failure for the system. In contrast, blockchain (BC) and microservice (MS) technologies allow services to split for independent tasks. In this research paper, we introduce BC security managers based on MS technology for federated cloud systems in an IoT environment. In addition, we present the design of the Federation Security System Manager (FSSM) MS with interoperability features. This enables the exchange of transactions between permissioned BC managers at different cloud providers, with some constraints. Furthermore, a security framework based on MSs and BCs is implemented to ensure security and protect access control. The security functions are deployed based on a smart contract between the permissioned BC managers to achieve interoperability. Finally, we introduce the development process of the proposed framework, which allows for interoperability and ensures the security and privacy of the participating data for a distributed IoT based on the federated cloud system.

Keywords: blockchain; microservices; security; IoT; interoperability; smart contracts; federated cloud system

Citation: Alshudukhi, K.S.; Khemakhem, M.A.; Eassa, F.E.; Jambi, K.M. An Interoperable Blockchain Security Frameworks Based on Microservices and Smart Contract in IoT Environment. *Electronics* **2023**, *12*, 776. https://doi.org/10.3390/electronics12030776

Academic Editor: Dimitris Apostolou

Received: 3 January 2023
Revised: 29 January 2023
Accepted: 31 January 2023
Published: 3 February 2023

1. Introduction

The meaning of the term "Internet of Things" (IoT) is the provision of services to users through networks, allowing users to cooperate and interoperate with each other and to access these services from any place at any time. [1]. In [2] reported that from 2020 to 2022, IoT connections increased by 140% worldwide, from 8.4 billion to 20.4 billion. They also found that numerous different types of use drove IoT connections between machines from 5.6 billion to 27 billion between 2016 and 2024.

The scaling of IoT devices may be hindered by having a centralized data center, which is the conventional IoT model. Connected devices typically communicate using a central system, which is accountable for collecting data, computing transactions, and sending data and commands to other devices. As the number of IoT smart devices grows to more than tens of billions, the central server model will no longer be viable due to its maintenance costs, among other issues. At that point, the system will no longer be scalable [3].

In the conventional IoT environment, the development of application software is supported by the implementation of a service-oriented architecture (SOA). The standard SOA uses a monolithic system, which is problematic when adapting to new IoT-enabled system requirements. This is because an SOA comprises various platform functions in an

interdependent and shareable database. An SOA database features tightly coupled dependencies between components and functions, which limits cross-platform interoperability, data privacy, and the scalability of services [4].

In contrast to SOAs, microservices (MSs) are developed and deployed individually. Development teams can work in an autonomous way, allowing for independent implementation through various teams. Generally, MSs allow for autonomous, scalable, flexible, and agile development [5]. These important benefits of MSs allow for interoperability within a permissioned, homogeneous blockchain (BC) network.

The interoperability of BCs can be a challenge for numerous reasons, such as the type of BC, consensus protocols, different cryptographic methods or hash methods, and smart contracts [6]. Interoperability achieves flexibility, application portability, and scalability. Some solutions to improve interoperability include notary schemes, sidechains and relays, and hashed time lock contracts (HTLCs) [7]. This research focuses on smart contract interoperability between homogeneous BCs based on MSs. Smart contracts had a significant impact on BCs' architectures, converting BCs from simple ledgers into a dynamic and flexible environment. The main aim of this research is to develop a framework based on BC security management for an IoT environment that includes federated cloud providers. This will allow for interoperability techniques between BC managers at different cloud providers. MS and smart contract techniques will be used. The main contributions of our proposed solution are as follows:

- Building an MS technique to ensure that system allows for interoperability between homogeneous BC security managers during interactions with different cloud providers.
- Developing a security manager for IoT environments with BC managers using MS technology.
- A full architecture of the MS-based IoT environment, which is executed in hierarchical layers of a federated cloud system in an edge, fog, aggregation, and federated layers.
- Building multiple smart contracts, which are developed and tested on a permissioned BC network using MS technology, to ensure secure communication between different cloud providers in the IoT environment.

The remainder of this paper is structured as follows: Section 2 provides background information on the Ethereum BC, smart contracts, and MS technologies and a literature review with regard to interoperability to ensure security requirements in IoT environments. Section 3 presents the design architecture of the proposed framework. Section 4 presents the main aspects of the experiment, the performance evaluation, and the results. Section 5 discusses the security analysis of the proposed framework. Finally, Section 6 illustrates the conclusions and future work.

2. Literature Review

2.1. Background

In this section, we illustrate BC, Ethereum, smart contracts, security and privacy, Access Control, and MS.

2.1.1. Blockchain

At the core of BC technology is a series of blocks, which are chronologically ordered in a link-list data structure with distributed ledger characteristics. The committed blocks cannot be updated, which means that the integrity of the data in the ledger remains intact. This is because every block includes the hash value for the prior block. Additionally, a ledger is repeated among peers in the BC environment. At the core of BCs are various cryptographic primitives that maintain security, for example, public key infrastructure (PKI) protocols, digital signatures, and hashing algorithms. Timestamped transactions are usually bundled together and stored in the form of a Merkle tree. In a BC network, the key activities are generating transactions, validating them, and storing them in the ledger [8]. There are three main types of BC, as described below [9]:

- Public BCs: These are most commonly used for decentralized transactions. They aim to avoid the use of a third party and can be used by any client with access to the network. Examples of public BCs are Bitcoin, IOTA, and Ethereum.
- Private BCs: Also called permission BCs or consortium BCs, these are generally used when a permission-based network can be established amongst known and trusted users. Commonly used by organizations, access to the system is restricted. This can result in centralized control over access to the network to achieve data privacy. Examples of private BCs include those created by the enterprise Ethereum, such as Hyperledger Besu, or other private BC platforms, such as Hyperledger Fabric.
- Combination BCs: These are a hybrid of public and private BCs. Public BCs work on the principle that all participants are part of the network, but the private network is by permission. The benefit of a hybrid system is that there is control over which data are shared on the public ledger and which data are kept private.

There are some key characteristics that distinguish BC technologies [10]:

- Decentralized: The essential feature of BCs is that there is no requirement for a central node. This means that multiple systems can be used to record, store, and update data.
- Transparent: Each node can update data in the BC system, and the location of the node that makes the changes is transparent. This makes the system more reliable.
- Open source: Anyone can check and use the BC system; records can be examined; and users can create the applications they require.
- Autonomous: Trust from a single point in the system becomes trust in the whole system; each node can update data securely. This is because of the basis of consensus, which prevents anyone from intervening and overriding the system.
- Immutable: which means that records will be reserved for always.
- Secure: Transactions are anonymous, which means that trust can be maintained between nodes. The only requirement is an individual's BC address, which is needed to make changes.

2.1.2. Ethereum

The fundamental characteristic of Ethereum is that it is an ecosystem with no central authority. All nodes are capable of processing and replicating data. They also process smart contracts for each participant. Essentially, Ethereum is a programmable BC that allows users to create their own complex operations. This is different from the Bitcoin BC, where user processes are constrained. Ethereum is a more comprehensive system that extends beyond the use of cryptocurrencies such as Bitcoin [11]. The main features of using the Ethereum BC are its improved system, ease of development, and enhanced interoperability. In addition, developers can create an arbitrary consensus for any case or reason in the system. Ethereum is built on a foundational layer, so anyone is permitted to write their own smart contracts in decentralized applications. Additionally, they can create their own arbitrary consensus, transaction, and state transition rules [12]. The code execution level in Ethereum is Ethereum smart contracts. It is a low-level code based on a bytecode language in the Ethereum virtual machine (EVM). The code has a series of bytes, each of which states an operation. These operations can store three types of data: stack, memory, and storage. The execution of the EVM code is very simple, as shown in Figure 1. Firstly, the computational state is determined by the block_state (e.g., transaction, message, code, memory, stack, pc, and gas). As the first argument in the tuple, block_state is the global state for all accounts. It contains balances and storage [12]. Further, a transaction is a signed data package that stores a message sent from an externally owned account. These transactions include a signature that identifies who sent the message, the amount of data sent, and who the recipient of the message is. In addition, there are two values called STARTGAS and GASPRICE [12].

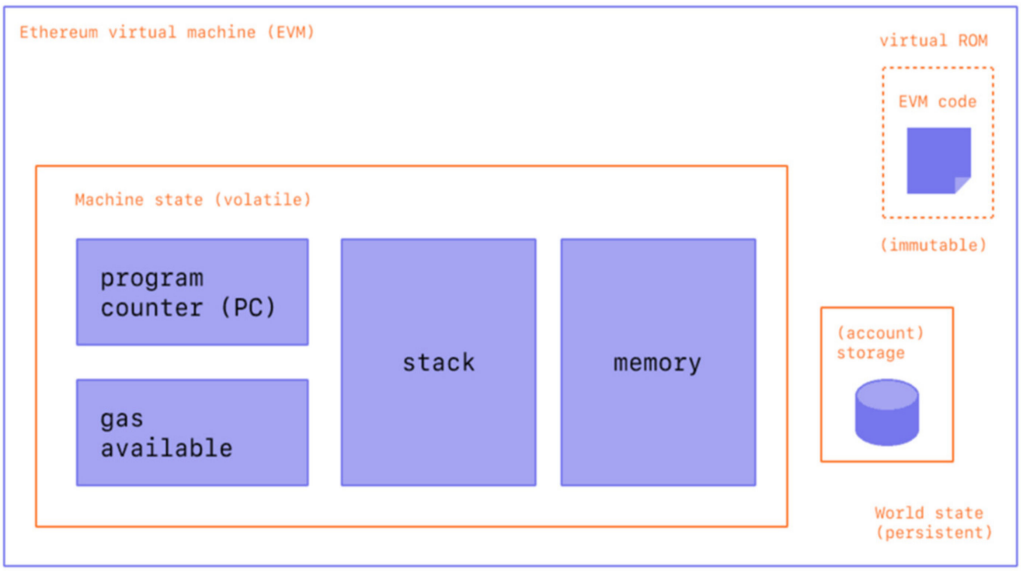

Figure 1. EVM architecture [13].

2.1.3. Smart Contracts

The need for third-party mediation is eliminated by the use of smart contracts, which are programs that define a set of rules that automatically enforce the execution of contract terms or a negotiation. These protocols have been developed with the approval of participants in all nodes; thus, the participants always know the result of a smart contract for any transaction. Smart contracts are run by the BC [14]. Another feature of smart contracts is that they can be used to enable transactions while continuing to meet the contract terms that are written into the code. This programmability allows the node to sign each transaction and send a smart contract with instructions for a particular functionality [3].

2.1.4. Security and Privacy

In order to encourage the continued generation of IoT devices around the world, it is vital that the IoT ecosystem is secure and not vulnerable to privacy concerns, malware, or unreliable communications. Today, billions of devices transfer private personal and business information, which, if unchecked, risks eroding privacy and confidence in the system. As the speed of innovation increases, security is too often ignored, and in its place, functionality becomes the key driving objective. The competition to create the most advanced devices is intense, and securing communication protocols should be of paramount importance [15].

2.1.5. Access Control

In conventional access control systems, a single administrator manages the access control procedures within the IoT devices. This ensures that the resource is protected because only those with express permissions can access and control the devices. BC-based access control systems avoid this single interface by storing the access control policies in a ledger, which can be accessed via a smart contract, thereby automatically granting or denying access to devices. The ability to decentralize access control is further enhanced by BC technology, which ensures data integrity and creates a trustworthy and tamperproof solution [16].

2.1.6. Microservices

MS architecture takes small tasks, independent running, and loosely coupled architecture into consideration. The lowest-level development teams process the separate codebases that include each task. Each function can be remade and redeployed independently. Whereas traditional architecture includes data persistence at every level, in

MS architecture, all services run individually with data persistence. Hence, well-defined application programming interfaces (APIs) are used for the services to communicate with each other, and no one service knows the implementation details of any other service. Moreover, the same technology stack, library, or framework does not need to be shared by the service, i.e., MS supports polyglot programming [17]. For an IoT environment using an MS, the main challenge is that the IoT has a smart domain with different applications and services that use various software and platforms to collect and process data. Numerous MS architectures have been developed based on different IoT domains, such as smart cities, smart commerce, and smart cars. In terms of, the IoT is a distributed system. MS technology supports this kind of architecture through autonomous tasks. Therefore, MSs do not share data with each other, meaning that they can use different languages and databases. Alternatively, they use representational state transfer (REST) protocols to reach and connect with each other [5]. Hence, development managers have embraced the benefits of MSs for the following reasons [18]:

- "Flexibility and agile deployment": MSs form a loosely coupled architecture that offers the developing team flexibility in determining what is necessary for their work and what is not. Deactivating the MSs that are not required saves IoT network resources. This means that MSs offer optimum control and flexibility over the software deployed to run the services as well as management of all of the IoT systems, such as the networks.
- "Resource-efficiency and portability": IoT system resources are more efficient because containerized MSs enable scalability. These lightweight MSs can be used independently of each other and on or off premises to optimize the performance requirements of the IoT environment.
- "Resilient operations and easy updates": As each MS in the IoT system operates independently of any other MS, their failures do not halt all working portions of the system. This makes the system secure and flexible. The entire IoT system's operation can also be maintained and shutdowns avoided during maintenance cycles because each MS has its own release protocol. The challenges related to implementing IoT platforms are consequently minimized due to this fine-grained and loose system, and tasks are contained in the MS.

These MS features improve IoT systems compared with using a monolithic system (e.g., an SOA). We developed our framework using MS technology. The main advantage is that the security functionality of each BC is separated into independent containerized MSs. Furthermore, each BC security manager has its own working container that is designed using a smart contract. This enhances the interoperability of our framework.

2.2. Related Work

In recent years, BC interoperability has become a significant topic among researchers. There are many challenges to ensuring security requirements in IoT environments. MS technology offers numerous advantages, including using loosely coupled, independent, flexible, and agile deployment to increase the interoperability of the IoT environment in a federated cloud. Table 1 describes the related works and their research gaps, methods, limitations, and workloads.

Lafourcade et al. [19] demonstrated that BC interoperability is impractical according to the classical concept of a BC. They proved that the interoperability of two BCs is equal to the design of a "2-in-1 BC," including both ledgers. This excludes the theoretical concerns regarding having interoperable BCs in the first position. They also noted that all interoperable BC frameworks currently operate by sharing already-created tokens between two BCs and not by providing the possibility of one BC exchanging tokens with the other. This leads to a change in the balance of total generated tokens for all BCs. This indicates that it is only possible to provide interoperability by building a two-in-one BC containing all ledgers.

Table 1. Comparative analysis of related works.

Research Paper	Research Gap	Methods	Workload	Limitations
Lafourcade et al. [19]	Interoperability of two blockchains.	Creating a blockchain ″2-in-1″ containing both ledgers. Apply formalization for this issue.	Introduce a theoretical definition of a blockchain and of interoperability.	Need to give a practical implementation to encourage the theoretical.
Malomo et al. [20]	A blockchain based on federated cloud framework to handle secure storage of offsite digital assets.	BFC2: block vault operations; digital asset; smart contract; participant auditors; clients; generator; buffers.	BFC2 system is a permissioned federated blockchain.	On the performance evaluation of the system, a comprehensive evaluation is needed.
Esposito et al. [21]	Cloud secure manager on healthcare system of management and exchange of data.	Cloud secure manager on healthcare systems for management and exchange of data based on microservices.	Microservices deployed in the cloud to manage and exchange healthcare data.	The exanimating on specifics of the scenario.
Cheng et al. [22]	To solve a weak of confidentiality and lack of performance through integrate BC of trustworthy execution environments (TEEs).	"Ekiden" is a merging blockchain within Trusted Execution. Environments (TEEs).	Hybrid system integrating trusted hardware and BC. It has three elements in Ekiden: clients, compute nodes, and consensus nodes.	Need to expand "Ekiden" so it can perform with a stronger threat model. No interoperability management to insure the system meets critical requirements.
Madine et al. [23]	Lack of cross-chain interoperability for different kinds of BC networks.	"AppXchain" framework that allows for different kinds of BC to communicate, sharing data and sending requests. Using CCHDA-cross chain hup DAPP.	Implemented in two hospitals based on Ethereum, including smart contracts. To share an "Electronic Medical Record (EMR)" between different BC networks.	"Increasing cost of deployment." "Limited upgradability." "Cross-industry interoperability."
Punathumkandi et al. [24]	A new framework to solve the interoperability problem based on incorporating EVM Chaincode and Fabric VM.	By a cross-chain protocol which including Notary scheme centralized. The communication is performed by EVM Chaincode and Fabric VM.	Implemented in Ethereum and Hyperledger Fabric BCs. EVMCC performs as a smart contract.	Not generalizing the framework for all platforms.
Xu, Nikouei, et al. [25]	MS architecture for security issues deployed on edge and fog layer nodes.	BlendMAS authorized BC and performed decentralized microservices of SPS.	Implemented on a private Ethereum BC for secure video stream service running on security MS via Docker container.	No attacker defined.
Zhang et al. [26]	By merging an edge computing framework with BC. To handle edge clients within an authentication algorithm.	"Edgex Foundry framework" for improving the business capacity.	MC uses Hyperledger Fabric BC.	Need to focus on lightweight consensus algorithms in BC.

Table 1. *Cont.*

Research Paper	Research Gap	Methods	Workload	Limitations
Viriyasitavat et al. [27]	Solving the interoperability and trust issues in IoT services.	Merging these technologies: BCT, service-oriented architecture (SoA), and key performance indicators (KPIs).	Implement a blockchain based on smart contracts for registering devices via a voting scheme. PBFT uses a permissioned BC. PKI used for the validator certification.	Need to extend the interoperations on the semantic layer. No algorithm for external users to trust in the smart contract.
Peng et al. [28]	Validated data sharing for privacy. For authenticated data to decentralize and effectively validate any part of a shared data register.	"BlockShare" framework based on blockchain with a new structure for authenticating data. "Dynamic data verification, zero-knowledge proof design."	Implemented a prototype for framework "BlockShare" using JavaScript and Python for data about owner, consumer, source. Blockchain at Solidity. Also used by Circom and Snarkjs.	No more details for rainbow attacks. Need to cover comprehensive defense techniques against malicious behavior.
Peng et al. [29]	Decreases latency via pipelining and increases throughput via asynchronous block creation. Different security methods are created to make systems resistant to malicious attempts.	"Ordering-Free Architecture execute-validate (EV)". "NeuChain" System. Permissioned blockchains use the EV.	They used two benchmarks, "YCSB and Mall Bank."	Need to improve the NeuChain system for authenticated queries in unsecure system.

To secure the storage of the off-site digital position, Malomo et al. [20] introduced a BC-enabled federated cloud framework. This framework identified authentication vulnerabilities and enhanced the early detection of cyber threats. Operational expenses were controlled and monitored by constantly evaluating the object's access control and resource communication. The assessment showed evidence that their model and strategy outperformed conventional approaches; also, their framework design was private and highly scalable, with access control rules.

Esposito et al. [21] ensured both security and privacy for transaction sharing and management in healthcare systems using a new MS technology. They presented a security management model based on the cloud for exchanging and managing data in healthcare systems. The researchers investigated the importance of the cloud system in healthcare. They presented their solution to interoperability, which had numerous advantages and features, such as improving the quality of this type of system. They also created a specialized system focusing on the client's patients. The system reduced operational costs and errors. Additionally, the authors examined privacy and security requirements.

Cheng et al. [22] created Ekiden, a platform that solves the limitations of a lack of confidentiality and weak performance by merging BCs in trustworthy execution environments (TEEs). In Ekiden, they deployed a new framework that divided consensus and execution, supporting effective TEE confidentiality using smart contracts to increase the scale of the system.

Madine et al. [23] created the solution appXchain, which allows for different BC networks to communicate, share data, and send requests. Their approach made use of decentralization for applications to interact with and understand various BCs and to distribute transaction requests and values across organizations. They used the healthcare context to describe the modules' requirements in an interoperable BC network. They

explained that all interactions need to share an electronic medical record between different BC networks, with some algorithms required. The solution was implemented in two hospitals using Ethereum smart contracts. Moreover, they presented a security analysis of the AppXchain solution.

Punathumkandi et al. [24] presented a sustainable system that could solve the interoperability problem in permissioned BCs. Their infrastructure was implemented in Ethereum and Hyperledger BCs with a 100% success rate.

Xu et al. [25] proposed the use of BlendMAS within a permissioned BC network and the use of MSs for security issues to ensure that data access is controlled, allowing for smart public safety. The functionality of security issues was decoupled by MSs based on smart contracts, which were then deployed to edge and fog layer nodes.

Zhang et al. [26] proposed a trusted edge within a BC network to construct an edge security platform to guarantee that clients' data remained private. They designed the platform's architecture based on MSs to make the platform's environment lighter. They designed a security authentication process based on an MS. This MS was built using the Hyperledger Fabric BC to preserve security on the edge platform.

Viriyasitavat et al. [27] discussed the issue of interoperability for IoT services and proposed an architecture solution based on merging a BC and SOA as well as select services. The proposed solution solved interoperability and trust problems in IoT services. The method was validated using smart contracts.

Peng et al. [28] proposed the BlockShare system to share data while respecting privacy. They designed a new architecture based on BCs with a new decentralized platform for authenticating data to effectively validate any part of a shared data register. They improved a zero-knowledge mechanism that allows the user to demonstrate a dynamic state without revealing the particular data attribute, which improves privacy. In addition, they evaluated the performance and efficiency of the system.

Peng et al. [29] provided an ordering-free system that executes deterministically while making ordering implicit. The "NeuChain" system evolves into a permissioned blockchain. They used some optimizations for improving throughput and latency, such as pipelining and asynchronous generation. Additionally, different security methods are created to make their system resistant to malicious attempts at "epoch servers, client proxies, block servers, and user clients." Moreover, "47.2–64.1 $\times$ throughput" is achieved by the "NeuChain" system, which is better than HyperLedger Fabric. Furthermore, "1.6–12.2 $\times$ throughput" is more advanced than that of other high-performance blockchains.

3. Materials and Methods

3.1. Design and Architecture of Proposed Solution

The proposed framework is based on a hierarchical federated system based on BC and MS technologies. This achieves interoperability between homogeneous BC platforms for IoT systems, as shown in Figure 2. Our solution contains hierarchical layers for a federated system in an IoT environment, as shown in Figure 3. The sequential layers used for processing the data, starting from the bottom, are the Edge Security Manager (ESM MS), Fog Security Manager (FSM MS), Aggregation Security Manager (ASM MS), Federation Security System Manager (FSSM), and, finally, Blockchain Security Manager (BCSM MS). In addition to that, all these layers run on MS technology, as shown in Figure 4. We assume that an IoT device in subsystem-1 wants to communicate with and exchange data with another IoT device in subsystem-2. This integration will be carried out by the FSSM MS (interoperability module). Therefore, our interoperability module merges transactions between the two BCSM MS in different cloud providers. The main role of BCSM MS is to manage and authorize the BC platform whenever there is any request from another BC platform at a different cloud provider. These rules occur through the interoperability module, FSSM MS. The following figures illustrate the main components of our framework. As shown in Figure 5, this illustrates the performance of our framework and its components.

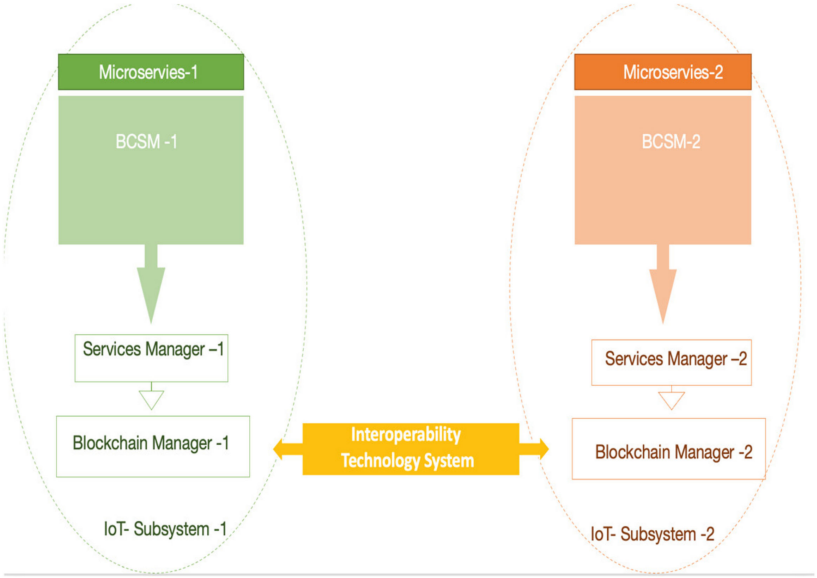

Figure 2. High-level architecture of the proposed solution.

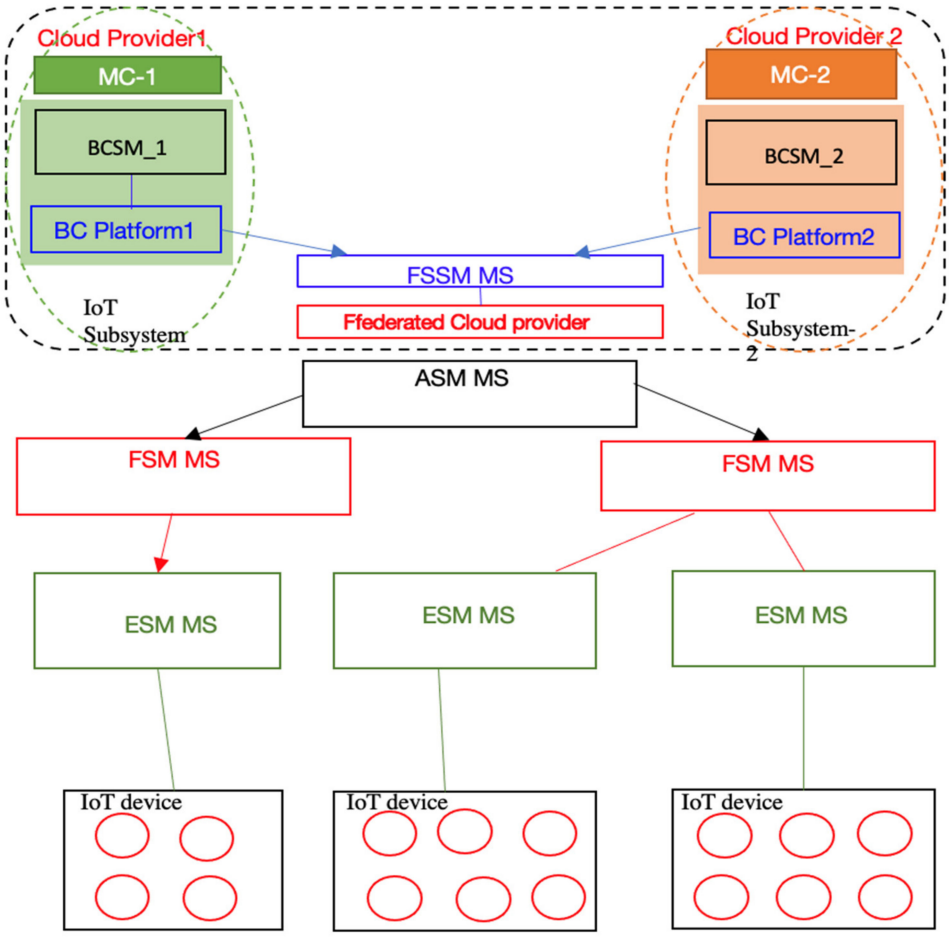

Figure 3. The architecture design of the proposed framework.

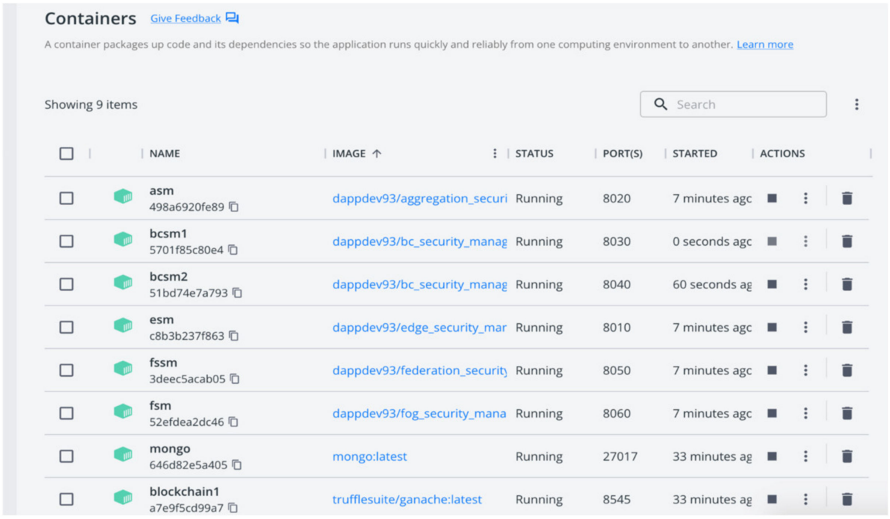

Figure 4. Screenshot of running the MS Docker containers for all the components of our framework.

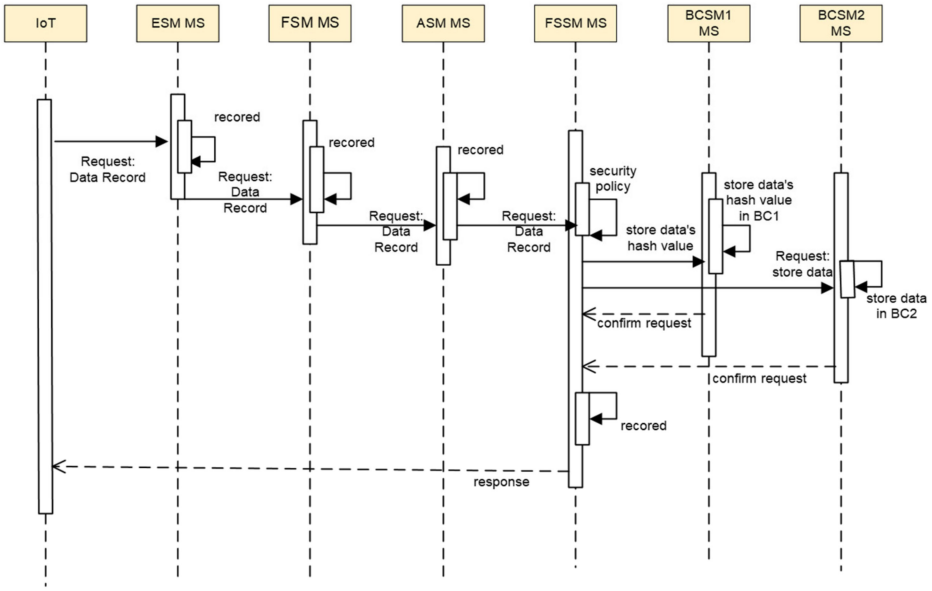

Figure 5. Sequential diagram of the execution of our framework.

3.1.1. Edge Security Manager MS (ESM MS)

The ESM MS locally manages and records IoT devices within each IoT sub-system. As it deals with various IoT devices, each has a different protocol and a different way to work. We implemented this on MS technology.

3.1.2. Fog Security Manager MS (FSM MS)

The FSM MS locally manages and records the data within each IoT sub-system. FSM is generated based on MS technology. MS supports independent deployment, which makes the data security services applicable and autonomous.

3.1.3. Aggregation Security Manager MS (AMSSM MS)

The ASM MS manages and records data exchange between different sub-systems on different cloud providers. ASM MS deals with FSSM MS, so any requirements regarding service security from the downward layers are deployed by this manager, while FSSM MS executes security policies. Our system is a distributed framework, so we deployed this manager using MS technology with flexibility and agility. MS in this layer manages the

exchange of security data functions between various cloud providers to transport them to the FSSM MS layer.

3.1.4. Federation Security System Manager MS (FSSM MS)

FSSM MS merges transactions between two BCSM MSs with different cloud providers. This manager, implemented using MS technology, functions as a gateway to manage communication and secure data exchange between different subsystems in each cloud provider. Our MS features in FSSM are loose coupling for different layers and interactions with two managers of BC platforms. Consequently, the MS technology independently scales the security services between the two BCSM MSs. Furthermore, this manager authorizes any requests from different IoT subsystems through the BC manager. The smart contract policies in this layer resolve security and privacy issues. Further, the FSSM MS uses the HAS-256 algorithm due to its high security and interoperability between two BC managers in a permissioned network. In addition, the permissioned network means that all systems will be under the policy restrictions of this manager to guarantee the security and privacy requirements. FSSM MS is in charge of determining whether the data is sensitive or not and then sending it to BC platforms to be stored in the metadata. Sensitive data can use BC1 for more protection and security. Otherwise, store it in BC2. In addition, FSSM MS stores and updates all kinds of data in its storage database.

3.1.5. Blockchain Security Managers MS (BCSM MS)

The BCSM MS manages and authorizes any request sent to the BC platform from another BC platform based on the FSSM MS. Hence, we have two BCSM MSs for each IoT subsystem that can interact with each other. Our framework implements two important technologies: BC and MS. Our solution uses a BC permission network, which provides security and privacy policies. Our federated system presents peer-to-peer interaction between different cloud providers. Therefore, this communication, which takes place under access control policies, uses smart contracts to authorize any requests with another BC network and carry out a secure exchange. We use MS technology as well as BS technology for this component to promote scalability and efficiency in our framework and support interoperability with other BCSM MSs at different cloud providers. In terms of guaranteeing the user's system's security, our proposed framework implements a smart contract using access control policies through the FSSM MS to limit the user's access. Algorithm 1 presents some of the rules implemented in BC1 of BCSM1MS based on FSSM MS in one subsystem of our framework; hence, we have another BC2 in BCSM2 MS that has its own contract registration management with the same manager as FSSM MS.

3.2. FSSM MS Algorithm

The details of the FSSM MS manager are shown in Figure 6 and Algorithm 2. If any external request to interact with or conduct a transaction between two cloud providers is made, this interoperability request is carried out by this manager. If IoT devices request data from the BCSM MS in another subsystem, the BCSM MS sends this request to the FSSM MS, which confirms the interoperability between the two BC platforms by applying the access control policies based on the MS and smart contract. However, the main goal of this manager is to support interoperability based on the BC platforms encapsulated in the MS container. Moreover, two BCSM MSs can encapsulate the smart contracts in the MS technology, which allows for a flexible and agilely executed cross-chain bridge between the two platforms. We organized our data ledgers into two BC platforms' metadata to guarantee security and privacy and to support interoperability. The first one is located in BCSM MS_1 and is for sensitive data, and the other is in BCSM MS_2 and is for non-sensitive data. At this point, the FSSM MS authorizes the requester from the list of users with a private key by applying the HAS 265 algorithm between the two BC platforms to confirm the communication. The FSSM MS checks the type of data. If the data are sensitive, then the FSSM MS sends a request regarding the data hash value with the data_ID from

the BCSM MS_1 platform. It retrieves the data hash value from the BC_1 metadata. BCSM MS_1 returns the data hash value to the FSSM MS. If the data are not sensitive, the FSSM MS sends the requested data with the data_ID from BCSM MS_2, which retrieves the data from BC_2. BCSM MS_2 then returns the data to the FSSM MS. Finally, the FSSM MS returns the data and the hash value to IoT_Subsystem_2, which validates the data.

Algorithm 1 Contract Registration in BC of BCSM MS

require: action in FSSM MS, user[_address], null
outcome: update in FSSM MS the user_event, return the userlist and status for BC, BCSM MS
in BC of BCSM MS if mapping (address => action) for public user
else private user of userList in BC
if action == authorized in BC of BCSM MS then require security policy of FSSM MS with (address user, action, uint256)
in BC of BCSM MS if user[_address] == action.null or user[_address] == action.unauthorized user then user is already authorized in BC of BCSM
else
user[_address] = action. Authorized in BC of BCSM MS then require security policy of FSSM MS then userList.push this (_address)
outcome: update in BC of BCSM MS and FSSM MS user_event has address, action. authorized
else
In BC of BCSM MS if action== unauthorize user in BC then user[_address] == action. authorized in BC of BCSM MS then user is already unauthorized
else
user[_address] = action. unauthorized in BC of BCSM
outcome: update in BC of BCSM MS and FSSM MS user_event has address, action. unauthorized

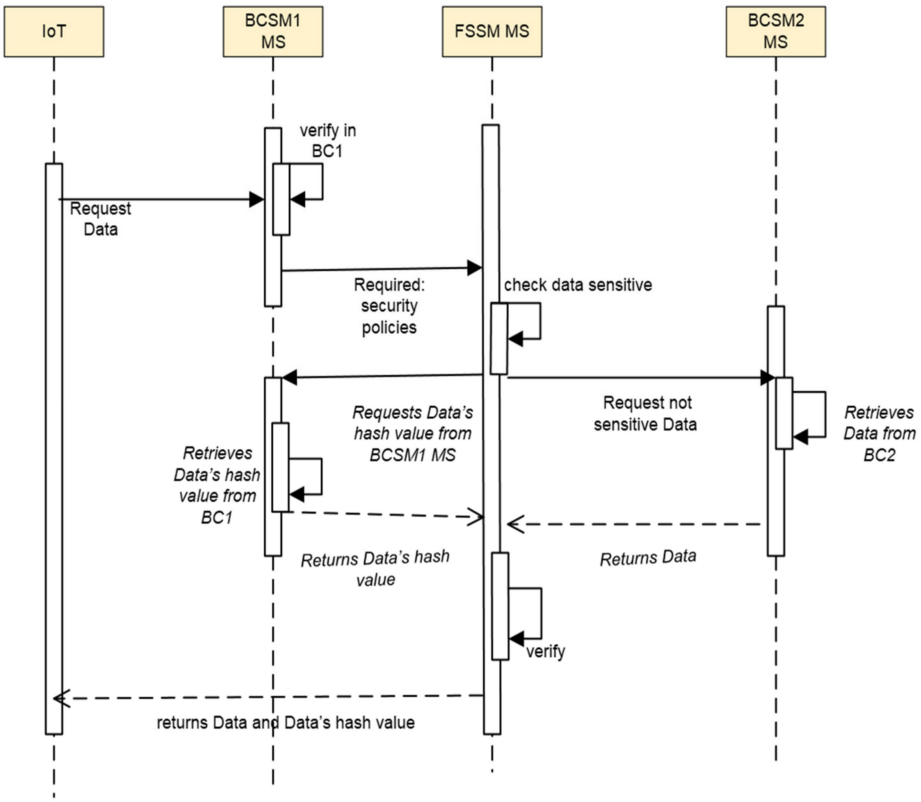

Figure 6. Sequential diagram of FSSM MS.

HAS-256

A one-way algorithm called a cryptographic hash mechanism produces a fixed-sized hash result from a changeable-length plaintext input. The hash algorithm guarantees that the original text has not been tampered with by comparing the decoded hash value to the one that was sent [30]. The Secure Hash Algorithm (SHA) is a cryptographic algorithm that ensures more security than others. The SHA-256 algorithm, when given a string, this hash function generates a 256-bit hash result. This function has four parts. Firstly, the standard length of this algorithm is 512 bits, and then it will add padding bits. Second, insert length bits for the remaining 64 bits. Third, initialize the buffer with an 8-hash value and 63 keys. In the 256-Hash Algorithm, the 64 keys are applied in 64 rounds. Finally, compression steps are taken in which all messages are separated into "N" 512-bit chunks and then used in compression function. Then, in 64 rounds, 512 bits were used. Furthermore, every round has a 32-bit word and key as input ([31], pp. 643–655). To ensure the integrity of the authentication system, we used SHA-256 for hashing data on the blockchain based on smart contracts and MS. In our framework, hashing data values is performed via FSSM MS, and then the hash value is sent to BC to be stored. The advantages of using BC are autonomy and immutability. Hence, when based on smart contracts, their programmability improves, making them better than regular ledgers in BC. Furthermore, the FSSM MS node will be established in a container MS, which will improve the hash algorithm's deployment agility.

Algorithm 2 FSSM MS

```
if IoT_Subsystem requests Data from BCSM1 MS
BCSM1 MS send the requests to FSSM MS
        if Data is sensitive then
        FSSM MS send requests to Data's hash value from BCSM1 MS
        BCSM1 MS retrieves Data's hash value from BC1 MS
        BCSM1 MS returns Data's hash value to FSSM MS
                    else
                    FSSM MS send requests Data from BCSM 2
                    BCSM2 MS retrieves Data from BC2 MS
                    BCSM2 MS returns Data to FSSM MS
        End if
FSSM MS returns Data and Data's hash value to IoT_Subsystem
IoT_Subsystem receive Data
End if
```

4. Results

4.1. Implementation

This section reviews the key aspects of deployment and testing for our framework. A system generally has a three-tier architecture, with a front end, middle end, and back end. The first tier is the front end; this is the user interface presentation tier. The middle tier is used for managing business, logic, and execution. The last tier, which is the back end, usually handles database management, as shown in Figure 7. Let us now turn to our framework. Table 2 illustrates the main tools implemented in our tiers. We tested and evaluated two tiers: the middle end and the back end. The first part is the middle-end tier, which holds the BC networks. MS technologies include smart contract functionality to resolve security and privacy issues in the interoperability framework. However, we deployed and implanted our framework in two Ethereum BCs with smart contracts, which are permissioned networks based on a full architecture of Docker containers for MS. The smart contracts' written language was given solidity through the use of Remix IDE tools. The Truffle framework was used to test and deploy smart contracts through permissioned BCs. The back-end tier is a Mongo Database (MongoDB) for off-chain storage. This NoSQL database is scalable and flexible. It deals with the web3.js library to connect the off-chain module with the middle-tier BC network to achieve consistency in the data.

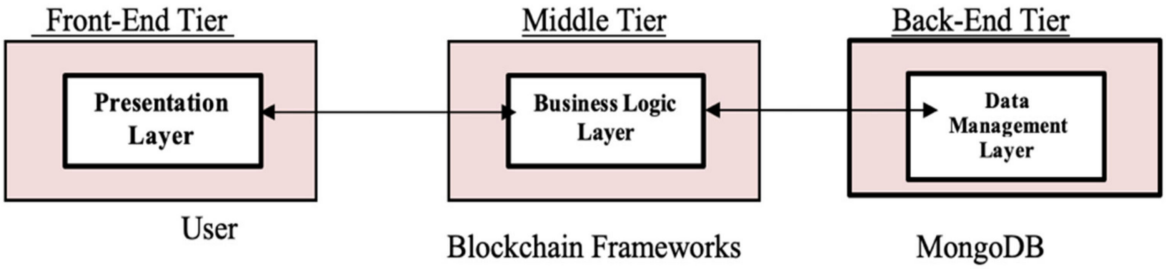

Figure 7. Three-tiered architecture.

Table 2. Development Tools Required.

Tier	Tools	Implement
Middle tier	Remix IDE	To write smart solidity contracts [32].
	Truffle Framework	To test and deploy smart contracts [33].
	Docker	To run MS [34].
	Ganache	To create a local Ethereum blockchain for development and testing [35].
	Hyperledger Besu	To build private BC [36].
	Azure cloud	To host a back-end server and provide the VMs required by the private BC [37].
Back-end tier	Node.js	To develop a back-end server and off-chain components [38]
	MongoDB	For off-chain storage. This is a database platform and NoSQL document; it is also a scalable, flexible DB [39].

In terms of overhead, we implemented our framework in two parts. First, data size refers to the amount of data sent for each of the following: when data size sent to ESM MS = 0.13 KB and the data size sent to FSM MS = 0.27 KB, ASM MS= 0.13 KB, FSSM MS = 0.13 KB, BCSM MS_1 = 0.18 KB, and BCSM MS_2 = 0.13 KB. As a result, the size of reading all the data from BC 1 = 0.18 KB and BC 2 = 0.15 KB. The second part is time processing, which refers to completing some processing in our system. For complete processing of the hashing algorithm, it takes 1.348618984222412 ms. In addition, when all the data from BC 1 is read, the process is complete in 52.099485009908676 ms, and for BC2, it is complete in 44.0608149766922 ms.

4.2. Performance Evaluation of Our Framework

In the proposed framework, a performance evaluation was conducted to assess our BC network by utilizing Hyperledger Caliper v0.5.0, which is an open-source benchmarking tool [40]. We tested the performance of our proposed framework using the configuration settings shown in Table 3. We implemented four VMs running Ubuntu 20.04 in Microsoft Azure cloud. Each VM has 4 Intel CPUs and 16 GB of memory. We used Hyperledger Besu v22.7 to build a private BC on one validator node with three peer nodes. Further, we used the Clique Consensus Protocol. The programming language Solidity was used in the smart contracts, and our BC network was assessed using the Hyperledger Caliper v0.5.0 benchmarking tool.

Table 3. Testing environment for performance.

Requirements	Settings
Nodes	Four VMs running Ubuntu 20.04 on Microsoft Azure cloud. Each VM has 4 Intel CPU and 16 GB memory
Blockchain	Hyperledger Beus v22.7 1 validator node 3 peer nodes
Consensus Protocol	Clique
Smart Contracts Programming Language	Solidity
Benchmarking Tool	Hyperledger Caliper v0.5.0

The popular metrics used to evaluate our framework are throughput and latency. Two metrics can be split into two types related to the operations that deal with these metrics. The two categories were read operation and write, or transaction, operation. As shown in the following equations [41]:

$$\text{Read throughput} = \text{Total read operations/total time in seconds} \quad (1)$$

$$\text{Read latency} = \text{Time when response received} - \text{submit time} \quad (2)$$

$$\text{Transaction throughput} = \text{Total valid transactions/total time in seconds} \quad (3)$$

$$\text{Transaction latency} = (\text{confirmation time} * \text{network threshold}) - \text{submit time} \quad (4)$$

As shown in Equation (1), we used it to read the throughput, which refers to the quantity of read operations that are executed in a certain amount of time. Read latency, as shown in Equation (2), refers to the total amount of time from the point at which the read operation is transmitted to the point at which a reply to the request is received. The write/transaction throughput, as shown in Equation (3), refers to the rate at which accepted transactions are spread through the BC network over a certain period of time, represented as transactions per second (TPS). Transaction latency, as shown in Equation (4), refers to the time required for a transaction to become applicable throughout the network [41].

Performance Estimate Affected by Read and Write Operations Based on Various Send Rate Values

In our experiment, the number of test rounds ranged from 1 to 9, with 1000 transactions for each round. We also executed the send rates with different values: 110 tps, 210 tps, 310 tps, 410 tps, 510 tps, 610 tps, 710 tps, 810 tps, and 910 tps. Therefore, there are impacts on the send rate values to test our system, especially in read and write operations. A send rate [24] = total number of transactions sent/total time.

All 1000 transactions were successful for the two operations. Therefore, we achieved good results in each case for latency and throughput. Figure 8 depicts how send rate values affect throughput and latency rates in a read operation. It ranges from 110 to 910 TPS. Overall, as the send rate increases, the throughput rate increases. However, at the first level of the send rate, which was 110 TPS, growth was slow. Then, the bar chart dramatically increased, from 210 to 610 TPS. The throughput rate fell slightly, from 510 to 610 TPS. Then, the bar chart increased from 710 to 910 TPS. Finally, the maximum amount of throughput was 138.48 TPS, the minimum amount was 93.78 TPS, and the average throughput was 127.70 TPS. In terms of transaction latency, its rate has steadily increased. While the send rates increased, the transaction latency also dramatically increased, from 110 to 310 TPS. However, the transaction latency gradually rose to a 910 latency rate, then fell slightly. Finally, the maximum amount of latency was 4.10 milliseconds (ms), and the minimum amount was 0.62 ms. The average latency was 3.08 ms.

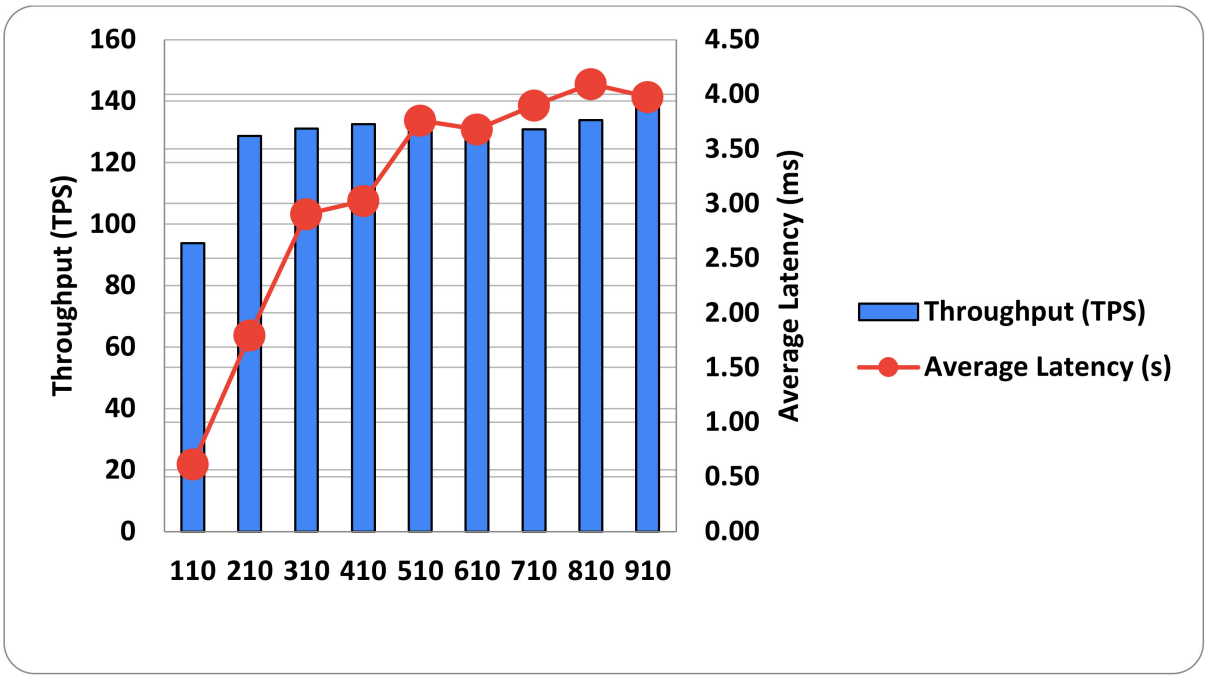

Figure 8. Various send rates for read latency and read throughput.

Figure 9 presents the experimental values in terms of write throughput and write latency. We used different send transaction rates ranging from 110 to 910 TPS. Overall, the bar chart illustrates that the transaction throughput was a sequence of wavering moving above, then down. The average throughput for write operations was 102.4 TPS, with a minimum of 93.93 TPS and a maximum of 108.13 TPS. The transaction latency rate increased at send rates of 110, 210, and 310 TPS. Then, latency fell slightly at the fourth point (send rate of 410 TPS). After that, the send rate latency grew exponentially to 510, 610, and 710 TPS. Then, the latency fell to the eighth point/send rate (at 810 TPS). The average latency was 4.02 ms, with a minimum of 2.11 ms and a maximum of 5.66 ms.

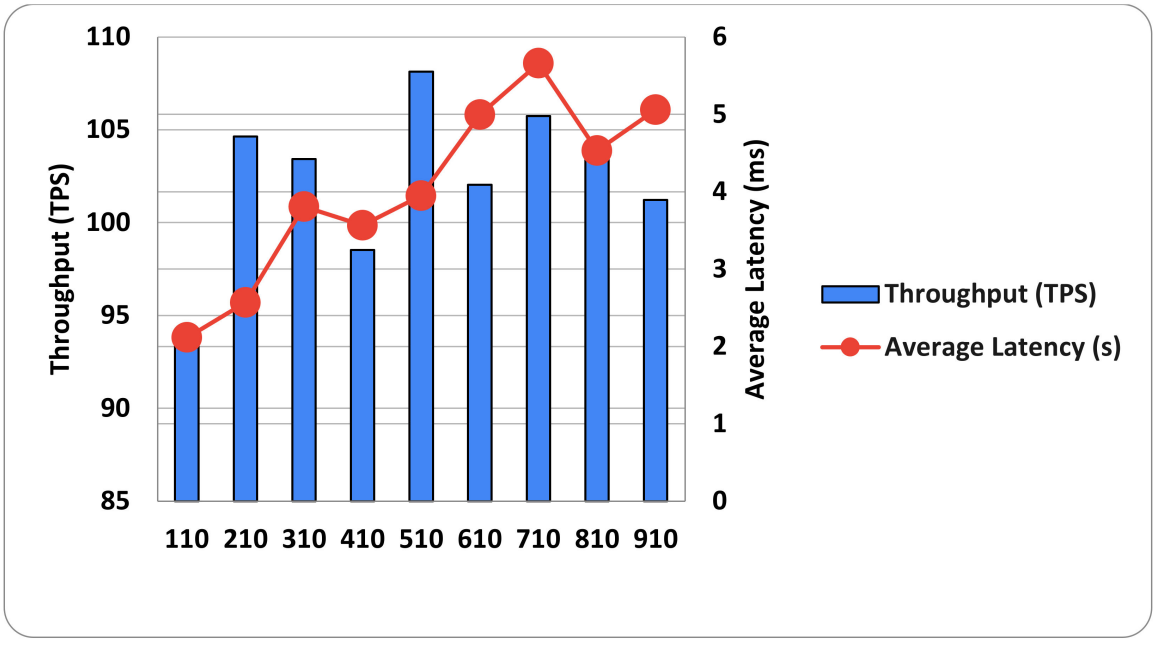

Figure 9. Various send rates for write latency and write throughput.

5. Discussions

In this section, we indicate our framework's security requirements. In addition, we use our security framework to protect against some attacks and compare the proposed solution with the existing work.

5.1. Security Analysis of the Proposed Framework

The main security requirements can be classified into confidentiality, integrity, and availability. Confidentiality implies authorized users' access to data. Integrity protects the transactions or data from any modification by unauthorized users. Availability means the data are available for authenticated objects [16]. Our framework solution fulfills these security and privacy requirements.

Authorization and Confidentiality: The smart contract manages authorization for users and data. The lists of private keys are stored in permissioned BC1 and BC2, while the access policies for the smart contract are implemented based on the FSSM MS. This manages any unauthorized members.

Availability: In our solution, all layers are created by individual containers in the Docker. Hence, every service is autonomous and independent, which means that the system is fast and available for any other requests. Our system is based on the cross-chain bridge between the permissioned BCs; all entities have private keys stored in metadata in BC1 and BC2 based on a smart contract, which drops and manages any unauthorized members. Our system is distributed using a permissioned BC, which is a decentralized module, making it more readily available than a centralized system.

Integrity: The HAS 256 hashing algorithm is used to manage transactions between the two permissioned BCs in BCSM1 MS and BCSM2 MS based on the FSSM MS container to ensure integrity.

5.2. Effectiveness of Our Security Solution

We analyzed the effectiveness of our solution to critical security attacks such as denial of service (DOS) and distributed denial of service (DDOS). The characteristics of these types of attacks are that the targeted servers are flooded with huge numbers of unwanted requests by the attacker. A DDOS attack occurs when a targeted server is flooded by requests from multiple sources. The attacks are possible due to the heterogeneity and complexity of the IoT networks; this means that the network layer of the IoT is at risk from such attacks [2]. To avoid these DDOS and DOS attacks, all traffic in both the BCSM1 and BCSM2 networks must be authorized by the FSSM MS manager implementing the smart contracts in the MS container. This protects the system from any malware behaviors. In other words, we built our system using a permissioned BC network with more restrictions.

5.3. The Difference between our Proposed Framework and Related Works

Table 4 summarizes the main points of comparison with existing solutions. The main focus when comparing our proposed system with related works is the interoperability of the permissioned BC based on the MS when handling security issues in a federated cloud.

Our proposed framework uses MS technology to promote scalability. It can be scaled independently while using MS technology through Docker. There are other advantages to using MSs in the IoT environment. Because the IoT has different devices and sensors, every object will have different protocols and functionalities. We developed our framework using independent containers for each manager at different layers in federated clouds, starting with the edge and fog layers, then the aggregation layer, and lastly the federated layers. This confirms the scalability factor.

Table 4. Comparison of our framework with existing solutions.

Paper Citation	MC-Independently	Interoperability-BC	Integrity	Access Control –Smart Contact	Federated System	Scalability
[23]	No	Yes, heterogeneous BC	Yes	Yes	No	Yes
[25]	Yes	No	Yes	Yes	Yes edge, Fog	Yes
[24]	No	Yes, heterogeneous BC	Yes	Yes	No	Yes
[26]	Yes	No	No	Yes	Yes edge	Yes
[27]	No	No	Yes	Yes	Yes edge	Yes
Proposed framework	YesFF	Yes, homogeneous BC-based on MS and smart contracts	Yes	Yes	Yes edge, Fog, aggregation, and federation	Yes

Our solution confirms the main objective, the interoperability of BC frameworks, by implementing a smart contract handle using MS containers. Therefore, our framework does not require trust in a third party to confirm transactions between the BCSM1 MS and BCSM2 MS in each subsystem. It applies an interoperability framework using smart contracts to every BC cluster. They can be classified, decentralized, and made transparent.

In terms of security and privacy issues, our proposed solution uses a permissioned BC to handle all types of data and confirm the privacy and security requirements for each subsystem. In addition, it uses the HAS 256 hashing algorithm to resolve integrity issues based on FSSM MS containers while managing the interoperability transaction between the two permissioned BCs (BCSM1 MS and BCSM2 MS). In order to guarantee the system's security, our proposed framework implements a smart contract using access control policies through FSSM MS to limit data access between each subsystem. Figure 10 shows some of the rules between BCSM1 and BCSM2 implemented in the FSSM MS.

```
51
52        const dataHashVaule = await HashData(data.data);
53
54        const response_1 = await ToBCSM_1(fileId, dataHashVaule, fileName);
55        const response_2 = await ToBCSM_2(fileId, data, fileName);
56
57        if (response_1 && response_2) {
58          res
59            .status(200)
60            .send(
61              "[Federation Security Manager] Data has been sent to FEDERATION SECURITY SYSTEM MANAGER"
62            );
63        } else {
64          res.status(500).send("error");
65        }
66      }
67    } catch (err) {
68      res.status(500).send(err);
69    }
70  });
```

Figure 10. Screenshot of some restricted functionality in FSSM MS.

The MS contains a bound context so that each security issue implements its own functionality through smart contracts. The registration function in each BCSM MS framework is performed without complexity and without being merged with other security services, which reduces the overhead execution time for each transaction. Some other benefits of interoperable BC are reduced costs for networks. As we have two subsystems, each with its own BC network, each subsystem has its own functions for sending and receiving transactions in its cluster, which reduces the costs of the networks of our entire system.

Moreover, our framework reduced overhead management, which is achieved in the FSSM MS manager based on MS functionality in a loose architecture.

6. Conclusions

The key challenges in SOAs and conventional security strategies for IoT systems are a lack of security and privacy, overly complex systems, a single point of failure, and the lack of a federated cloud framework. In this research paper, we built a framework to ensure the security and privacy of the IoT environment. Our solution is an interoperable system between homogeneous BC managers for security and scalability. We used decentralized BCSM-based MS containers to manage and authorize the BC platforms for any request. Hence, we achieved interoperability between two BCSM MSs in different cloud providers using the FSSM MS manager, which merged the transactions. Moreover, we built multiple smart contracts based on BCs and MSs to ensure secure communication between the different cloud providers in the IoT environment. The proposed framework was deployed on a permissioned, homogeneous BC (Ethereum platform) based on Docker containers to deploy the MS technology. Hyperledger Caliper was used to test the performance of our solution regarding the special metrics of throughput and latency, as well as read and write operations based on smart contract functionality. The performance evaluation results were affected by the different send rate values. Overall, all the transactions in each test round were successful and efficient, without any failures. In future work, we will develop our solution for a heterogeneous BC network; in addition, we will extend our framework by combining artificial intelligence and high-performance computing to improve the latency and throughput of the BC.

Author Contributions: Conceptualization, K.S.A., M.A.K. and F.E.E.; methodology, K.S.A., M.A.K., F.E.E. and K.M.J.; software, K.S.A.; validation, M.A.K., F.E.E. and K.M.J.; writing—original draft preparation, K.S.A.; writing—review and editing, K.S.A., M.A.K. and F.E.E.; supervision, M.A.K., F.E.E. and K.M.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sen, A.A.A.; Basahel, A.M. A comparative study between security and privacy. In Proceedings of the 2019 6th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 13–15 March 2019; pp. 1282–1286.
2. Hassija, V.; Chamola, V.; Saxena, V.; Jain, D.; Goyal, P.; Sikdar, B. A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures. *IEEE Access* **2019**, *7*, 82721–82743. [CrossRef]
3. Jiang, Y.; Wang, C.; Wang, Y.; Gao, L. A Survey on IoT Security: Application Areas, Security Threats, and Solution ArchitecturesA cross-chain solution to integrating multiple blockchains for IoT data management. *Sensors* **2019**, *19*, 2042. [CrossRef] [PubMed]
4. Xu, R.; Ramachandran, G.S.; Chen, Y.; Krishnamachari, B. BlendSM-DDM: BLockchain-ENabled secure microservices for decentralized data marketplaces. In Proceedings of the 2019 IEEE International Smart Cities Conference (ISC2), Casablanca, Morocco, 14–17 October 2019. [CrossRef]
5. Vural, H.; Koyuncu, M.; Guney, S. A Systematic Literature Review on Microservices. In *Computational Science and Its Applications–ICCSA 2017: 17th International Conference, Trieste, Italy, 3–6 July 2017, Proceedings, Part VI 17*; Springer International Publishing: Berlin/Heidelberg, Germany, 2017; pp. 203–217. [CrossRef]
6. Khan, S.; Amin, M.B.; Azar, A.T.; Aslam, S. Towards Interoperable Blockchains: A Survey on the Role of Smart Contracts in Blockchain Interoperability. *IEEE Access* **2021**, *9*, 116672–116691. [CrossRef]
7. Pang, Y. A New Consensus Protocol for Blockchain Interoperability Architecture. *IEEE Access* **2020**, *8*, 153719–153730. [CrossRef]
8. Lal, C.; Marijan, D. Blockchain Testing: Challenges, Techniques, and Research Directions. 2021. Available online: http://arxiv.org/abs/2103.10074 (accessed on 18 March 2021).
9. Attaran, M.; Gunasekaran, A. (Eds.) *Blockchain Principles, Qualities, and Business Applications BT—Applications of Blockchain Technology in Business: Challenges and Opportunities*; Springer International Publishing: Cham, Switzerland, 2019; pp. 13–20. [CrossRef]
10. Niranjanamurthy, M.; Nithya, B.N.; Jagannatha, S. Analysis of Blockchain technology: Pros, cons and SWOT. *Clust. Comput.* **2019**, *22*, 14743–14757. [CrossRef]

11. Panarello, A.; Tapas, N.; Merlino, G.; Longo, F.; Puliafito, A. Blockchain and iot integration: A systematic survey. *Sensors* **2018**, *18*, 2575. [CrossRef] [PubMed]
12. Buterin and Vitalik, Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform. Etherum, January 2014. pp. 1–36. Available online: https://github.com/ethereum/wiki/wiki/White-Paper (accessed on 2 January 2023).
13. Ethereum Virtual Machine (EVM) | Ethereum.org. Available online: https://ethereum.org/en/developers/docs/evm/ (accessed on 2 January 2023).
14. Sahay, R.; Geethakumari, G.; Mitra, B. A novel blockchain based framework to secure IoT-LLNs against routing attacks. *Computing* **2020**, *102*, 2445–2470. [CrossRef]
15. Patel, C.; Vyas, S.; Kalariya, D.; Parmar, N.; Saikia, P.; Patel, S. A Futuristic Survey on Learning Techniques for Internet of Things (IoT) Security: Developments, Applications, and Challenges. TechRxiv. Preprint. 2022. Available online: https://www.techrxiv.org/articles/preprint/A_Futuristic_Survey_on_Learning_Techniques_for_Internet_of_Things_IoT_Security_Developments_Applications_and_Challenges/19642977/1 (accessed on 2 January 2023).
16. Abdi, A.I.; Eassa, F.E.; Jambi, K.; Almarhabi, K.; Khemakhem, M.; Basuhail, A.; Yamin, M. Hierarchical Blockchain-Based Multi-Chaincode Access Control for Securing IoT Systems. *Electron* **2022**, *11*, 711. [CrossRef]
17. Microservice Architecture Style—Azure Architecture Center | Microsoft Learn. Available online: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices (accessed on 9 November 2022).
18. Driss, M.; Hasan, D.; Boulila, W.; Ahmad, J. Microservices in IoT security: Current solutions, research challenges, and future directions. *Procedia Comput. Sci.* **2021**, *192*, 2385–2395. [CrossRef]
19. Lafourcade, P.; Lombard-Platet, M. About blockchain interoperability. *Inf. Process. Lett.* **2020**, *161*, 105976. [CrossRef]
20. Malomo, O.; Rawat, D.; Garuba, M. Security through block vault in a blockchain enabled federated cloud framework. *Appl. Netw. Sci.* **2020**, *5*, 16. [CrossRef]
21. Esposito, C.; Castiglione, A.; Tudorica, C.A.; Pop, F. Security and Privacy for Cloud-Based Data Management in the Health Network Service Chain: A Microservice Approach. *IEEE Commun. Mag.* **2017**, *55*, 102–108. [CrossRef]
22. Cheng, R.; Zhang, F.; Kos, J.; He, W.; Hynes, N.; Johnson, N.; Juels, A.; Miller, A.; Song, D. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P), Stockholm, Sweden, 17–19 June 2019; pp. 185–200. [CrossRef]
23. Madine, M.; Salah, K.; Jayaraman, R.; Al-Hammadi, Y.; Arshad, J.; Yaqoob, I. AppxChain: Application-level interoperability for blockchain networks. *IEEE Access* **2021**, *9*, 87777–87791. [CrossRef]
24. Punathumkandi, S.; Sundaram, V.M.; Panneer, P. Interoperable permissioned-blockchain with sustainable performance. *Sustainability* **2021**, *13*, 11132. [CrossRef]
25. Xu, R.; Nikouei, S.Y.; Chen, Y.; Blasch, E.; Aved, A. BlendMAS: A blockchain-enabled decentralized microservices architecture for smart public safety. In Proceedings of the 2019 IEEE International Conference on Blockchain (Blockchain), Atlanta, GA, USA, 14–17 July 2019; pp. 564–571. [CrossRef]
26. Zhang, J.; Lu, C.; Cheng, G.; Guo, T.; Kang, J.; Zhang, X.; Yuan, X.; Yan, X. A blockchain-based trusted edge platform in edge computing environment. *Sensors* **2021**, *21*, 2126. [CrossRef] [PubMed]
27. Viriyasitavat, W.; Da Xu, L.; Bi, Z.; Sapsomboon, A. New blockchain-based architecture for service interoperations in internet of things. *IEEE Trans. Comput. Soc. Syst.* **2019**, *6*, 739–748. [CrossRef]
28. Peng, Z.; Xu, J.; Hu, H.; Chen, L.; Kong, H. BlockShare: A Blockchain Empowered System for Privacy-Preserving Verifiable Data Sharing. *Bull. IEEE Comput. Soc. Tech. Comm. Data Eng.* **2022**, *1*, 14–24.
29. Peng, Z.; Zhang, Y.; Xu, Q.; Liu, H.; Gao, Y.; Li, X.; Yu, G. NeuChain: A Fast Permissioned Blockchain System with Deterministic Ordering. *Proc. VLDB Endow.* **2022**, *15*, 2585–2598. [CrossRef]
30. Hambouz, A.; Shaheen, Y.; Manna, A.; Al-Fayoumi, M.; Tedmori, S. Achieving Data Integrity and Confidentiality Using Image Steganography and Hashing Techniques. In Proceedings of the 2019 2nd International Conference on New Trends in Computing Sciences (ICTCS), Amman, Jordan, 9–11 October 2019; pp. 1–6. [CrossRef]
31. Reddy, G.P.; Narayana, A.; Keerthan, P.K.; Vineetha, B.; Honnavalli, P. Multiple hashing using SHA-256 and MD5. In *Advances in Computing and Network Communications: Proceedings of CoCoNet 2020*; Springer: Singapore, 2021; Volume 1, pp. 643–655.
32. Remix—Ethereum IDE. Available online: https://remix-ide.readthedocs.io/en/latest/ (accessed on 15 November 2022).
33. Truffle—Truffle Suite. Available online: https://trufflesuite.com/truffle/ (accessed on 15 November 2022).
34. Home—Docker. Available online: https://www.docker.com/ (accessed on 15 November 2022).
35. Ganache. Available online: https://trufflesuite.com/ganache/ (accessed on 15 November 2022).
36. Hyperledger Besu. Available online: https://www.hyperledger.org/use/besu (accessed on 15 November 2022).
37. Cloud Computing Services | Microsoft Azure. Available online: https://azure.microsoft.com/en-us/ (accessed on 15 November 2022).
38. Node.js. Available online: https://nodejs.org/en/ (accessed on 15 November 2022).
39. MongoDB. Available online: https://www.mongodb.com/features (accessed on 15 November 2022).

40. Hyperledger Caliper. Available online: https://hyperledger.github.io/caliper/ (accessed on 15 November 2022).
41. Hyperledger Blockchain Performance Metrics White Paper. Available online: https://www.hyperledger.org/learn/publications/blockchain-performance-metrics (accessed on 15 November 2022).