Proof-of-Monitoring (PoM): A Novel Consensus Mechanism for Blockchain-based Secure Service Level Agreement Management

Nisita Weerasinghe, *Student Member, IEEE,* Raaj Mishra *Student Member, IEEE,* Pawani Porambage *Senior Member, IEEE,* Madhusanka Liyanage, *Senior Member, IEEE,* and Mika Ylianttila, *Senior Member, IEEE*

***Abstract*—In the current 5th Generation (5G) networking paradigm, the enforcement of Service Level Agreements (SLAs) is a non-trivial measure to ensure the scope and the quality of services and standards between tenants and service providers (SPs). On top of this, Secure Service Level Agreements (SSLA) are introduced to ensure that SPs deliver the most critical and required security-related standards defined in the contract, such as integrity, confidentiality, availability, non-repudiation, and privacy assurance. However, with the tendency for more distributed and multi-stakeholder networking architectures in next-generation networks, the management process of such SSLAs will be challenging due to the diversified security vulnerabilities and complexity of underlying technologies. Although blockchain is emerging as a platform to facilitate such distributed SSLA/SLA management frameworks, its currently available consensus mechanisms are more generic. Still, they need to improve in terms of applying in multi-stakeholder networks. Therefore, this paper presents a novel consensus mechanism called Proof-of-Monitoring (PoM) for a blockchain-based novel SSLA management framework. Moreover, we provide details about the prototype implementation of our proposed consensus algorithm and SSLA management framework. It is proven by comparing our proposal with the other existing solutions that our solution outperforms in many aspects, such as energy consumption, computation cost, and security features.**

***Index Terms*—Blockchain, Secure Service Level Agreements, Consensus Algorithm, Network monitoring, Smart Contracts**

I. Introduction

A service Level Agreement (SLA) is a contractual financial agreement between a service provider and a customer. In the telecommunication context, it defines as a form of assurance that the Communication Service Providers (CSPs) deliver the diversified communication service requirements of Communication Service Customer (CSC), aligning with service compliance standards (3GPP TS 28.530). Hence, it is of utmost importance to any CSC. SLA violations may degrade the expected service quality, where the user has to encounter unexpected experiences and consequences for the paid services, which eventually negatively impact the user experience. Traditionally, end customers must follow manual and time-consuming procedures to prove the evidence of these SLA violations. Hence, a fully automated SLA system is required to support continuous monitoring and boost the user satisfaction level. Additionally, a trustworthy guaranteeing platform must monitor service deliveries and maintain fairness between the customer and SP. Furthermore, SLA includes measurable metrics to ensure Quality of Service (QoS) [1], while there is an urge for security assurance of the service delivery. Hence, researchers are working on the upcoming Secure Service Level Agreements (SSLAs) concept. SSLA specifies the security-oriented requirements required to assure the expected level of security. Essentially, the security features such as confidentiality in different dimensions, the integrity of transaction records, and the availability of persistent services are a few of the critical security requirements anticipated in the SSLAs [2].

The limitations encountered in most state-of-art solutions [3], [4] are SSLA monitoring party functions as a centralized entity, which inherits default challenges such as a single point of failure and is prone to network attacks (e.g., Distributed Denial of Service (DDoS)). Still, not much research has been carried out in the area of SSLA. However, more research has been done in the area of SLA. Therefore, we examined SLA-based state-of-art solutions.

To address the limitations encountered in conventional SLA systems, one of the vital approaches commonly researched is the adaptation of blockchain technology. It can be incorporated to automate monitoring and improve trust across every 5G service delivery. Blockchain can convert the architecture from a centralized to a decentralized one and execute manual agreements via smart contracts. Every transaction is recorded in the distributed ledger to ensure non-repudiation [5]. Furthermore, miners process transaction validation to guarantee immutability within blockchain records and restrict non-authorized parties from tampering with the data within the blockchain [6]. Therefore, we propose to incorporate blockchain technology into our research.

However, state-of-the-art blockchain-based solutions [7], [8] still have challenges, such as high blockchain operational costs and energy data silos. For the deployment of blockchain, the tenant nodes should have the capability to perform extensive computations to reach a consensus result. However, many

Nisita Weerasinghe and Mika Ylianttila are with the Centre for Wireless Communications, University of Oulu, Finland. e-mail: firstname.lastname@oulu.fi

Raaj Mishra is with the Dell Technologies, India. e-mail: raaj2045@gmail.com

Pawani Porambage is with VTT Technical Research Centre, Finland and University of Oulu, Finland. e-mail: pawani.porambage@vtt.fi, pawani.porambage@oulu.fi

Madhusanka Liyanage is with the School of Computer Science, University College Dublin (UCD), Ireland and the Centre for Wireless Communications, University of Oulu, Finland. e-mail:madhusanka@ucd.ie, madhusanka.liyanage@oulu.fi

resource-constrained devices will struggle to achieve that, making it impossible to adopt blockchain technology by such devices. Massive amounts of energy and time will be wasted where comparable results can be obtained with simpler consensus algorithms. In addition to that, the existing consensus algorithms focus on achieving consensus as an independent task, which is entirely irrelevant to the services obtained from the blockchain. Therefore, a customized consensus protocol is advantageous for efficient blockchain integration. Therefore, the room is still available to leverage the concept and develop a blockchain-based system. In our proposal, both the limitations of non-blockchain and blockchain-based SLAs/SSLAs are expected to be rectified.

In particular, the key contributions of our work are as follows:

- Propose blockchain-based SSLA management framework
- Propose novel cost and energy efficient consensus mechanism, called Proof-of-Monitoring (PoM), which is customized for SSLA management applications
- Evaluate the performance of the proposed system in a simulated environment and confirm the viability through a prototype implementation
- Evaluate the correctness of the proposed consensus protocol by performing formal modeling and formal verification

The rest of the paper is outlined as follows: Section II provides background knowledge on SLA/SSLA and different technologies used. III examines existing works while Section IV introduces novel blockchain-based SSLA architecture. Section V proposes a novel consensus algorithm. Section VI carries out numerical simulations to evaluate the performance of the proposed consensus protocol and to compare it with existing systems. Section VII presents the prototypical implementation of the proposed consensus algorithm and SSLA management system. Section VIII discusses the experimental results. Section X compares the proposed model with existing systems and discusses the limitations of the proposed approach. Section IX defines the formal model of the proposed consensus protocol, verifies its properties, and analyzes its results. Finally, Section XI concludes the paper.

II. Background

This section discusses the core concepts of SLA/SSLA, the challenges of both blockchain and non-blockchain-based SSLAs, and the related technologies used throughout our study.

A. Fundamentals of SLA/SSLA

SLA is a legal agreement containing negotiated services between consumers and service providers. Services define based on multiple Service Level Objectives (SLOs). SLOs represent quantifiable metrics with values. The primary expectation of establishing a SLA is to ensure the defined SLOs are met. Furthermore, SLA focuses more on leveraging the QoS, while SSLA concentrates on improving the expected level of security.

The life cycle of the SLA [9] is generally divided into five phases as follows: (1) Identification of capacities of SP and definition of requirements of customer (2) SLA negotiation between customers and SP (3) Resource provision and service activation (4) SLA monitoring, validating, reporting and violation detecting (5) SLA assessment with SP(service quality, key issues) and customer (service experience and management of requirements). Our target is to advance the fourth phase of the SLA's lifecycle.

An example of a security SLO is the Packet Loss Ratio (PLR), defined as the ratio of the number of data packets lost to the total number of packets a network node should have forwarded. Packet losses usually occur due to channel errors or network congestion. This metric is typically associated with QoS considerations, and the amount of tolerable packet losses (e.g., 1% or 5%-10%) depends on the type of data being sent. In the context of network security, packet-dropping or blackhole is a type of denial-of-service (DoS) attack where a network node drops packets that it should not have. A DoS attack can happen at different layers, e.g., the application layer or network layer. If a node repeatedly drops packets, that indicates potential malicious behavior, leading to communication unavailabilities for benign users. Furthermore, the lower the PLR, the higher the reliability and availability of the service against security threats.

B. Potential challenges of SLA/SSLA management systems

The most significant probable challenges related to SLA/SSLA management are described below.

1) Single Point of Failure: The current systems are primarily aligned with centralized deployment architecture, which causes several limitations. For instance, being vulnerable to DDoS attacks, which makes the service unavailable by attacking the centralized services [10]. In addition, centralized storage, such as classical database systems, can expose data to malicious parties.

2) Transparency: There is a potential for resource providers to violate pre-established SSLAs and raise discrepancies [11]. Hence, the lack of transparency in the current system makes traceability harder. These problems make the service delivery unsatisfactory to the customer, resulting in losses to the service providers. Furthermore, the dispute resolution process becomes exhaustive and will incur overheads for all parties.

3) Scalability: The number of tenants and operators connected to the industrial ecosystems will expand with the complexity of future telecommunication applications. Especially the future industrial integration of IoT generates massive demand for 5G service delivery with diversification [12]. Hence, handling exponential load is challenging with state-of-the-art centralized architectures in future scenarios.

4) Lack of Reliable Monitoring Data: Due to obvious reasons, we cannot always be certain that a third-party monitoring solution always provides reliable data. Some reasons are their tendency to suffer from a single point of failure and the possibility to be compromised easily because of being centralized.

5) No Automated Violation Detection: Traditional methods do not support automated SSLA violation detection and compensation methods. Customers must follow a manual and time-consuming verification and resolution process to claim for a violation (e.g., via e-mails).

C. Potential challenges of blockchain-based SLA/SSLA management systems

Existing blockchain-based SLA/SSLA suffers from high latency, high cost, high computational overhead, and resource wastage. Extra delays are caused by the time-consuming execution of the mining process (e.g., Proof of Work (PoW)). Unnecessarily expensive for some because of favoring the wealthier participants (e.g., Proof of Stake(POS)). Excessive computational overhead, high energy consumption, and resource wastage because miners have to perform a computationally intensive mining task (e.g., PoW). Miners must put much effort into non-value-added tasks such as hash calculation (e.g., PoW) or stake management (e.g., PoS). Hence, it is evident that these consensus protocols suffer from different challenges. Furthermore, although the adaptation of private blockchain networks might favor addressing these issues, it is more centralized than public networks.

In a nutshell, the main task of the SLA/SSLA management application is to ensure that the SP meets obliged service standards. A fair, trustworthy, and automated platform is required to monitor whether the expected standards are met. For this, numerous blockchain solutions have been introduced already. However, energy-efficient and fully decentralized blockchain solutions are still yet to be introduced.

Hence, it is optimal to design a consensus protocol with a mining task that complements its application's primary function to avoid the aforementioned issues. As a result, the energy and resources are not wasted on non-value-adding tasks. Instead, they are utilized for the core service of the application.

D. Pervasive Authentication Protocol and Key Establishment (PAuthKey) scheme

PAuthkey protocol is a lightweight authentication and key Establishment mechanism proposed for resource-constrained WSNs in IoT applications [13]. It mainly comprises two phases: registration and authentication phase. In the registration phase, the sensor node must acquire a certificate from its cluster head which they assume to be the Certificate Authority (CA). Initially, the sensor node generates a random number $r_{\mathrm{u}} \epsilon [1, ..., n-1]$ to compute Elliptic Curve (EC) point R_{u} as given in equation 1 and send the value to CA along with certificate request.

$$R_{\mathrm{u}} = r_{\mathrm{u}} G \tag{1}$$

Note that the EC domain parameters defined in this paper follow the standard elliptic curve equation over the finite field F_q is $y^2 = x^3 + ax + b$, where $4a^3 + 27b^2 \neq 0$, variables a and b are coefficients, G is the generator point, and integer n is the order of the curve.

Upon receiving the certificate request. CA generate random number $r_{\mathrm{CA}} \epsilon [1, ..., n-1]$ and calculates implicit certificate $Cert_{\mathrm{u}}, e, s$ respectively as follows, Let d_{CA} be the private key of CA.

$$Cert_{\mathrm{u}} = R_{\mathrm{u}} + r_{\mathrm{CA}} G \tag{2}$$

$$e = H(Cert_{\mathrm{u}}) \tag{3}$$

$$s = e r_{\mathrm{CA}} + d_{\mathrm{CA}} (mod n) \tag{4}$$

CA responds to the node by sending a message including $Cert_{\mathrm{u}}$ and s. Then, the node can generate e using the equation and the same hashing function that CA has used. Thereafter, the node can calculate its private key d_{u} and public key Q_{u} as follows.

$$d_{\mathrm{u}} = e r_{\mathrm{u}} + s (mod n) \tag{5}$$

$$Q_{\mathrm{u}} = d_{\mathrm{u}} G \tag{6}$$

Further, [13] proves that Q_{u} can also be calculated using the below equation.

$$Q_{\mathrm{u}} = e Cert_{\mathrm{u}} + Q_{\mathrm{CA}} \tag{7}$$

E. Shamir's Secret Sharing (SSS) scheme

To develop a novel consensus algorithm, we intend to use the key-sharing scheme proposed by Shamir [14]. It is a process in which a secret key (S) is split into n parts, known as shares. The combination of these shares re-creates the original key. The minimum number of shares required to restore the key is known as threshold k. Therefore, SSS is defined as a (k, n) threshold scheme. This technique is ideal in a trustless environment such as a blockchain where the key can be distributed over many nodes. SSS is built based on polynomial interpolation. Note that, Size of finite field $p \epsilon \mathbb{P} : p > S$, such that $a_{\mathrm{i}} < p$, $a_0 = S$ where we select $k-1$ random elements, a_1, $...a_{\mathrm{k\text{-}1}}$ from a finite field of size p. The polynomial is constructed as follows,

$$f(x) = a_0 + a_1 x + a_2 x^2 + ... + a_{\mathrm{k\text{-}1}} x^{\mathrm{k\text{-}1}} \tag{8}$$

Generate n points $(i, f(i))$ where $i = 1, ...n$, from $f(x)$ and distribute these points among nodes. A node who has captured k of these points are able to rebuild the $f(x)$. In our scenario, we are focused only in finding the S which can be identified as a_0 in the $f(x)$. Hence, reconstruction of entire polynomial is unnecessary to calculate a_0. It can be generated using following formula.

$$f(0) = \sum_{j=0}^{k-1} y^{\mathrm{j}} \prod_{m=0, m \neq j}^{k-1} \frac{x_m}{x_m - x_j} \tag{9}$$

III. Related Works

Up to date, various approaches are evolving to investigate across different avenues to cater to the diversified requirements of the SLA context at present. Out of the phases of the SLA lifecycle, researchers have been most focused on the monitoring phase, then the negotiation phase and least concentrated on violation management and reporting [15]. Philipp et al. [16] proposes a SLA model to support the negotiation phase by permitting dynamic changes to multiple service levels during the deployment phase of an SLA. Sanjay et al. [17] propose a trust model for SLA monitoring to support the assessment phase. Eduardo et al. [4] introduce an architecture comprising multiple software agents to identify SLA breaches and a third-party auditor to avoid conflict of interest between SP, contractor, and client. The proposed approach permits gathering SLA metrics from each stakeholder and analyzing the deviations and demand for solutions. However, this proposal lacks transparency by relying on external services from third parties.

Hiroki et al. [7] proposes a blockchain-based platform to automate SLA contract enforcement. However, the proposal does not consist of a system to analyze the credibility of SLA breaches. Ke et al. [18] proposes a blockchain-based auditing scheme to preserve the privacy of SLA of network slicing and execution of punishments based on auditing results. Rohit et al. [19] introduces a blockchain-based approach to detect SLA breaches and analyze their root causes in a multi-cloud ecosystem. Another interesting piece of work has been done by Huan et al. [8] with the introduction of a decentralized witness model to detect SLA violations. Nevertheless, all these proposed solutions expend high costs and computational overheads for mining tasks which need to be minimized. This is due to the fact that most of these state-of-art SLA blockchain-based solutions suffer from a lack of an application-specific consensus.

Alemany et al. [3] presents a security SLA manager to guarantee the security of end-to-end network slices. Its core component is the SSLA Manager, which functions as a centralized entity. It tends to be compromised easily, leading to the single point of failure, service unavailability. In addition, it cannot guarantee the transparency of the service delivery. However, this is the only state-of-art that has significantly contributed to developing a security-based SLA framework. Nevertheless, there are significant modifications required to improve the entire system.

Only a few works have been carried out to ensure the correctness of novel consensus protocols. For instance, Hamra et al. [20], Wai Yan Maung Maung et al. [21] present formal models built using CSP# and verified using PAT model checker. While they have inspired our formal model, the proposed consensus protocol fundamentally differs from theirs.

Therefore, there is a demand for a proper blockchain-based SSLA management system and, to complement that, a custom-built application-specific consensus algorithm.

IV. Proposed Blockchain-based SSLA Management Framework

The main objective of our proposal is to invent and execute a zero-touch security-oriented SLA framework that guarantees trustworthiness, accountability, and security in delivering communication services at low cost and low energy. Hence, we will require real-time network data from trustworthy and authentic sources to devise a sustainable security approach. Our system uses a decentralized blockchain network architecture where nodes can voluntarily deploy their network sensors in the wireless channel that the client and SP communicate. Then blockchain nodes can capture the data and feed it to the blockchain service layer. Our system introduces a mechanism for blockchain nodes to earn revenue for their resource investment and network monitoring. Further, our approach proposes a method to maximize the profits of nodes that provide correct data continuously and, at the same time, reduce the dependency on nodes who report false information. Moreover, the occurrence of a violation is considered based on the majority vote of the monitoring nodes. Accordingly, the trustworthiness and reliability of the monitoring data are guaranteed. However, to empower an economic model with less computational overhead, our framework proposes a use case-oriented consensus algorithm where the blockchain nodes do not have to perform extra work to participate in the consensus. The high-level architecture of the proposed blockchain-based SSLA management framework integrated with a novel consensus algorithm is in the Fig.1.

The performance features of the SP are monitored and evaluated precisely with feedback from the deployed nodes in the 5G core networks. Furthermore, the consumer end has been enabled to report performance feedback and service level compliance through the blockchain network. The alignment of performance with the SSLA has been logged in the blockchain to ensure non-repudiation. Then no party could upload false data or alter data within the blockchain network.

Blockchain integration ideally facilitates the dynamic and adaptable enforcement of service agreements through smart contracts. Smart contracts' decentralized and dynamic nature enforces the dynamic deployment of contractual agreements across the ecosystem. Essentially, blockchain is decentralized and ensures the decentralized service operation by distributing the load between multiple collaborative nodes. Therefore, it caters to the diverse service requirements of stakeholders with less delay.

Transparency of agreements is enabled by encoding SSLAs in smart contracts. The transparency of smart contracts ensures the terms and conditions of the contract are clear and understandable, as well as not deviate from the pre-agreed terms and conditions. Therefore, non-repudiation is guaranteed.

The decentralized ledger of blockchain incorporated with interlinked cryptographic verification, avoids the possibility for a data in the database to be tampered. Furthermore, it ensures that the execution of sequential events in the SSLA is compliant and in order. Moreover, blockchain's decentralized operational capability eliminates the single point of failure and tolerates the scaled-up transaction volumes.

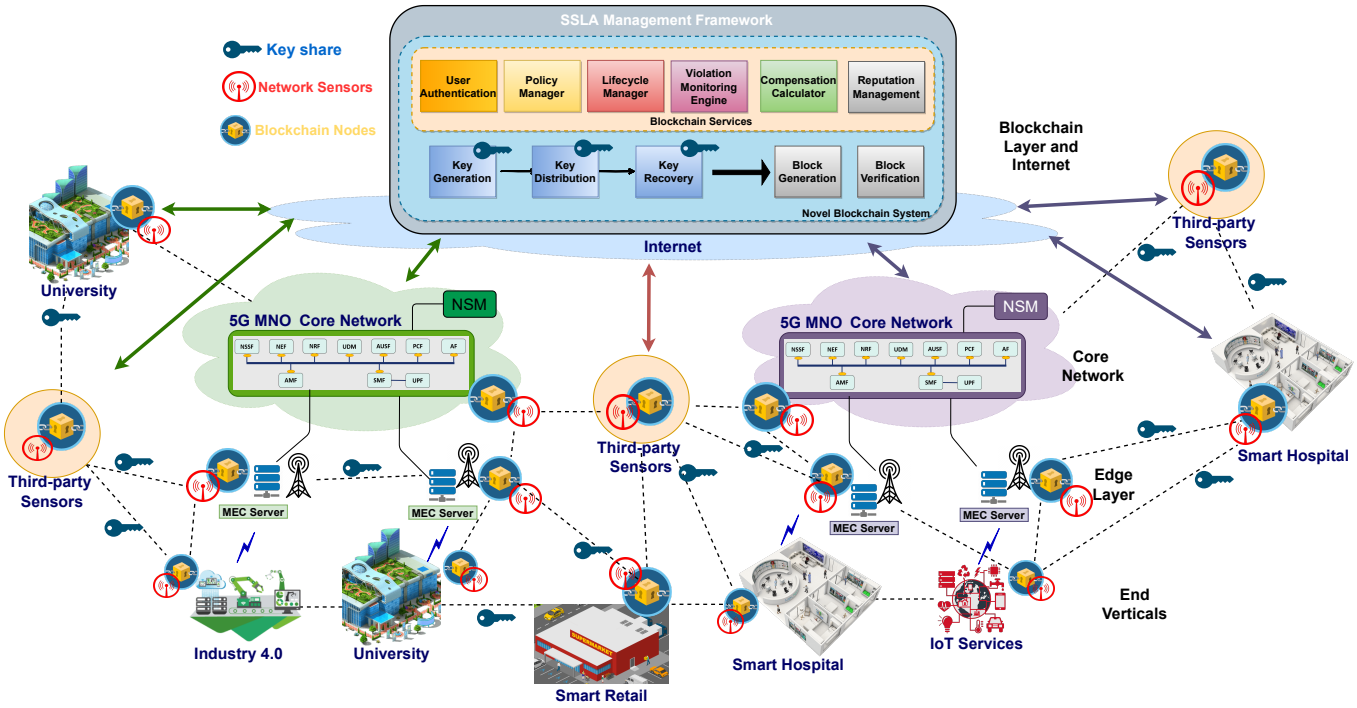


Fig. 1: High-level architecture of the proposed blockchain-based SSLA management framework

A. *Key components of the proposed framework*

We determined the main functional modules of the SSLA framework and their primary services. Fig. 2 represents key service modules of the proposed SSLA management framework and its interaction with the stakeholders. The stakeholders (client, SP) and monitoring nodes invoke back-end (blockchain) services through an API. The primary services of each functional module are discussed below.

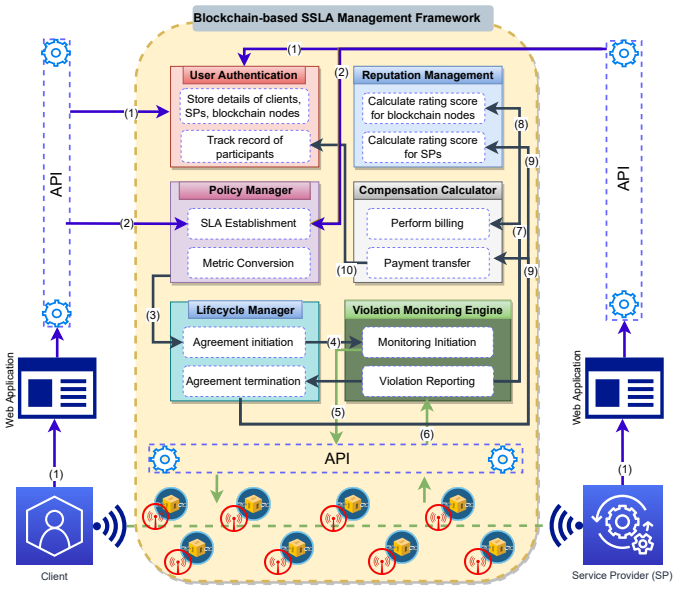


Fig. 2: Fundamental services of the proposed blockchain-based SSLA management framework

User Authentication: This service is responsible for managing accounts and keeping records of SPs, customers, and monitoring nodes. Stakeholders register to the proposed platform to become accountable for receiving services. Likewise, the monitoring nodes must register with the system before initiating the network monitoring. Initially, participants enter their credentials through the web application. The API passes the data to the blockchain and invokes the User Authentication service. It stores the credentials in the distributed ledger and links the wallets of each user to their accounts. Furthermore, this service module invokes whenever a registered stakeholder requires to log into the system, and it will grant access permission by retrieving the user information from the distributed ledger.

Policy Manager: The client can select a SP from our web application and send its desired security requirements to the selected SP. Then, the SP generates a SSLA draft and sends it to the client. The client can moderate selected requirements and request reviews from the SP during the negotiation period. SP can review and revise the SSLA. This process will run recursively until the client is satisfied. The client can approve the agreement by sending the service charge to the blockchain. Then, the blockchain generates an event to notify SP and invokes the Policy Manager service module. The Policy Manager translates high-level metrics into measurable metrics. For example, if the high-level metric is PLR, then translated measurable metrics are the number of packets transmitted and dropped over a period. It stores measurable metrics along with other SSLA parameters in the blockchain and sends a SSLA deployed notification to the Lifecycle Manager. Moreover, it transfers the service charge to the Compensation Calculator, where the fee will be held until the SSLA termination stage.

SSLA agreement includes the following negotiated parame-

ters such as the address of SP, address of the customer, service fee, validity period, monitoring period (= Period at which the monitoring node reports monitored data to the blockchain), SLO metrics, SLO values, allowable violation count per SLOs, priority weightage of SLOs, blockchain fee weightage. Note that the priority weightage is a percentage that can be set to adjust the priority levels among SLOs of SSLA, and it depends on the criticality level of the security metric in question. The blockchain fee weightage is a percentage fee that is distributed among blockchain nodes who correctly monitor the network (e.g., 0.1 %).

Lifecycle Manager: This functional module handles both SSLA initiation and termination processes. It always refers to the contract period of the stored SSLAs. The Lifecycle Manager invokes Violation Monitoring Engine whenever the starting period of any SSLA reaches and terminates the contract when the contract period expires or discovers a SSLA policy infringement. In addition, it is responsible for notifying the SSLA termination to Compensation Calculator and Reputation Management modules.

Violation Monitoring Engine: The main task is to manage network monitoring activities. Peer nodes in the blockchain network monitor the packets transferring between each client and service provider pair via the wireless communication channel. Each node has the freedom to install a preferred network sensor and monitor the measurable metrics of each security metric in SSLAs, during the specified monitoring period in SSLA. Next, they feed the monitored data to the blockchain at the end of the monitoring period. Then, the Violation Monitoring engine calculates the values for each security metric of SSLAs and checks whether the SP meets the SLOs. In our system, we declare a violation if more than 50% of the active monitoring nodes input data, confirming a violation. Subsequently, the Violation Monitoring engine invokes Compensation Calculator and Reputation Management modules. Additionally, It notifies the Lifecycle Manager to terminate the SSLA when the number of violations conducted by a specific SP exceeds a certain pre-defined threshold.

Compensation Calculator: This entity mainly performs billing activities at the end of the monitoring period. It divides the service fee among SP and blockchain nodes based on their service delivery. Blockchain nodes refer to the winning miners and violation reports (=blockchain nodes who have correctly reported violations). A blockchain fee is a reward offered to blockchain nodes for allocating their resources throughout the monitoring period. Further, it calculates the losses incurred by the user for service failures.

The client is liable to deposit the service fee at the establishment of SSLA. In the SSLA termination stage, our system distributes service fee percentages among SP, winner miner, and violation reports. Also, the customer will receive a portion of it as compensation if the violation count exceeds the allowable number of violations.

Initially, the system allocates a portion of the service fee to the blockchain nodes at every $\mathrm{T}_{\text{monitoring}}$ period, and it calculates the blockchain fee $F_{\text{blockchain}}$ as follows. Let service fee be F_{service} and blockchain fee weightage be p.

$$F_{\text{blockchain}} = F_{\text{service}} \times p \times T_{\text{monitoring}}$$

A certain amount of the blockchain fee collects as an incentive for winning miners. The rest divides among the violation reporters based on their reputation score. The one which has the highest reputation score receives the highest revenue. Therefore, it is evident that providing accurate data all the time will maximize their profit.

The system calculates the customer compensation relating to i^{th} SLO, $F(i)_{\text{compensation}}$ as follows, Let violation period relating to i^{th} SLO be $T_{\text{violation}}(i)$, priority weightage of SLO be q_{i}, the total number of SLOs per SSLA be n.

$$F_{\text{compensation}}(i) = (F_{\text{service}} - F_{\text{blockchain}}) \times \frac{T_{\text{violation}}(i)}{T_{\text{monitoring}}} \times q(i)$$

Note that $T_{\text{violation}}(i)$ is the total time within the monitoring period during which a violation occurred corresponding to the i^{th} SLO.

Our system computes the total compensation fee per monitoring period as below,

$$F_{\text{compensation}} = \sum_{i=1}^{n} F_{\text{compensation}}(i)$$

Subsequently, $F_{\text{compensation}}$ amount is transferred to the customer's wallet at the SSLA termination stage. The rest of the funds are credited to SP, F_{SP}

$$F_{\text{SP}} = (F_{\text{service}} - F_{\text{blockchain}}) - F_{\text{compensation}}$$

Reputation Management: The primary function is to compute rating scores for the blockchain nodes and SPs at every monitoring period and agreement termination session, respectively. The reputation score of each violation reporter increments while the reputation scores of non-violation reporters decrements if the system confirms a violation. The blockchain node is discarded from the network if its reputation score exceeds a certain threshold. Furthermore, the updated rating scores of every SP displays on the web application at the stage where a customer is selecting a SP. Hence, it eases the customer to choose an optimal SP.

B. Workflow of the proposed SSLA management process

The workflow presents a sequence of steps from the generation of a SSLA to the termination of a SSLA. Fig. 3, Fig. 4, and Fig. 5 illustrate the flow of the initiation phase, monitoring phase, and termination phase of the SSLA management framework, respectively. The vital steps of the three phases are explained below.

1) Initiation Phase: Initially, a client selects a SP based on his/her needs and sends the security requirements to the SP. SP is accountable for generating a SSLA draft considering the user input. Then SP will send the created SSLA to the customer for his/her approval. The customer has the privilege to request modifications, and SP is obliged to review and revise the defined terms in SSLA until he/she is fully satisfied. Hence, the SSLA negotiation process follows a recursive process.

Once the client is pleased with the received SSLA version, he/she can provide his/her consent by depositing the agreed service charge defined in SSLA.

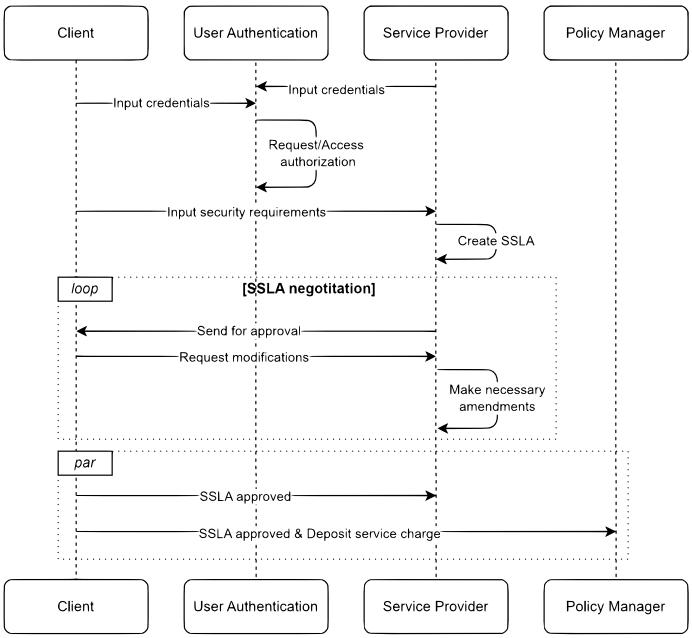


Fig. 3: SSLA initiation phase

2) Monitoring Phase: The Policy Manager converts the SLOs into measurable metrics and stores them along with other SSLA parameters in the blockchain. Then it sends the SSLA deployment notification to the Lifecycle Manager. When the starting period of the contract reaches, the Lifecycle Manager triggers the SSLA initiation process by sending a SSLA initiation notification to the Violation Monitoring Engine. Then, Violation Monitoring Engine commences monitoring the wireless channel, which the client and SP connected, with respect to the measurable metrics. Subsequently, it reports any infringements found to Compensation Calculator and Reputation management in every monitoring cycle. The Compensation Calculator module computes the compensation for the customer, penalty for SP, and service fee blockchain nodes. The Reputation Management calculates reputation scores for blockchain nodes. If the reputation score of a blockchain node reaches a certain threshold, it will be eliminated from the system.

3) Termination Phase: The termination phase mainly includes the termination of SSLA monitoring and settlement of final payments.

SSLA termination happens if any of the following actions are triggered, (1) the Lifecycle Manager receives the SSLA termination request from the Violation Monitoring engine that it has encountered that the SP has exceeded the maximum allowable number of violations (2) the SSLA met the contract expiration date. Subsequently, the Lifecycle Manager sends the SSLA termination notification to Compensation Calculator and Reputation Management modules. The Compensation Calculator transfers the service charges for SP and blockchain nodes if no violation is detected. If not, it compensates the client by transferring the compensation fee (a portion of the service

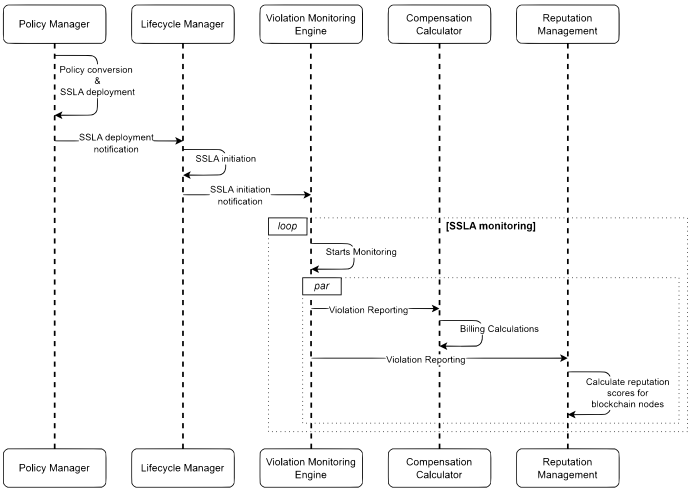


Fig. 4: SSLA monitoring phase

charge). Finally, the Reputation Management calculates and updates the reputation score of the SP based on their violation status.

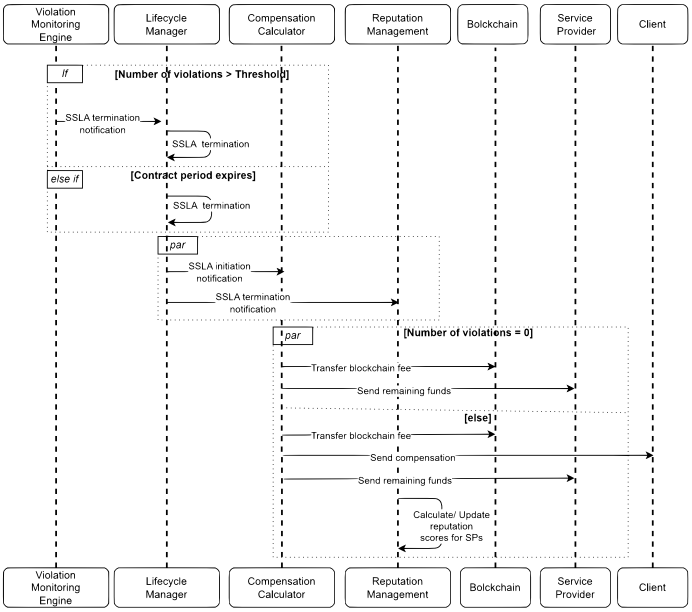


Fig. 5: SSLA termination phase

V. Proposed Proof-of-Monitoring consensus algorithm

This section proposes a novel consensus algorithm named Proof-of-Monitoring, customized to the SSLA management application. Its principal concept and the key phases are explained explicitly in this section.

A. Principal concept

The consensus protocols of state-of-art blockchain solutions pose intensive computational demands which require a large amount of energy for each miner to maximize their chances of becoming a winning miner. Hence, our system proposes

an innovative approach to remove this unnecessary overhead from the system by designing a novel consensus specific to the SSLA management system. The idea is to make the mining task the same as the main work of the SSLA management application, network monitoring.

The novel consensus algorithm utilizes both PAuthkey protocol [13] and Shamir's Secret Sharing scheme [14] technologies. The main phases of the proposed approach are depicted in Fig. 6 and explained explicitly with their corresponding pseudo-codes.

B. Key phases of the PoM mechanism

1) Key Generation: To develop the consensus algorithm, we incorporate the basic operations introduced for the authentication and key establishment phases of the Wireless Sensor Networks (WSNs) in the PAuthKey protocol [13]. We program its computations in a black box of the miner software without exposing them to any blockchain nodes. The internal calculations include the generation of Elliptic Curve (EC) R_{u} point, implicit certificate $Cert_{\text{u}}$ and the private key d_{u}. The algorithm 1 shows the mathematical operations performed to generate them. Note that EC domain parameters such as generator point G, Private and Public key of certificate authority d_{CA}, Q_{CA}, Curve order n are predefined.

Algorithm 1 Key and Certificate Generation

Input: Generator point (G), Private and Public key of certificate authority $(d_{\text{CA}}, Q_{\text{CA}})$, Curve order (n)

Output: Private key (d_{u}), Implicit certificate $(Cert_{\text{u}})$

r_{u} := a random number between 1 and $n-1$
Compute EC point $R_{\text{u}} = r_{\text{u}}G$
r_{CA} := a random number between 1 and $n-1$
Generate $Cert_{\text{u}} = R_{\text{u}} + r_{\text{CA}}G$
Integer $e \leftarrow$ Hash of Cert$_{\text{u}}$
Integer $s = er_{\text{CA}} + d_{\text{CA}}(modn)$
Generate $d_{\text{u}} = er_{\text{u}} + s(modn)$

2) Key Distribution: The proposed consensus protocol concatenates the previously computed d_{u} and $Cert_{\text{u}}$ using the PAuthKey protocol. Let's called the combination of d_{u} and $Cert_{\text{u}}$ as $Secret(S)$. S is divided into n shares using the Shamir Secret Sharing approach. Next, the mining application reveals random key shares to each node, and nodes can broadcast the received share to the wireless channel where SP and client are connected via a known port which they have already declared when joining the network. Note that the system shares the public key Q_{CA} with all the blockchain nodes, which will be necessary for the later calculations in the block generation phase. Algorithm illustrates the pseudo-code related to the key distribution phase.

3) Network Monitoring and Key Recovery: The designated task for the miners is to calculate public key Q_{u} by recovering d_{u} and $Cert_{\text{u}}$. Hence, nodes must attentively listen to the service ports, capture each key share, and reconstruct the desired parameters. The system rewards the one who completes the task with incentives, encouraging the blockchain nodes to find the solution by investing their resources to the maximum. Miners can recover the key by capturing its shares while monitoring the network for SSLA violations. Algorithm 3 shows the pseudo-code related to the key recovery stage.

Algorithm 2 Key Distribution

Input: Private key (d_{u}), Implicit certificate $(Cert_{\text{u}})$, Number of active nodes (m), Number of shares (n), Threshold (k)

Output: n pieces of (d_{u}), n pieces of $(Cert_{\text{u}})$

$S = \{d_{\text{u}}, Cert_{\text{u}}\}$
Require: Size of finite field $p \epsilon \mathbb{P} : p > S$, such that $a_{\text{i}} < p$, $a_0 = S$ where
Sample $k-1$ random numbers $\{a_1, a_2 ...., a_{\text{(k-1)}}\}$ from a finite field of size p
Require: $n \geq 3$
Ensure: $n = 2k-1$
for $i = 1$ `to` $k-1$ **do**
    a_{i}
end for
Generate the polynomial $f(x) = a_0 + a_1x + a_2x^2 + ... + a_{\text{(k-1)}}x^{\text{(k-1)}}$
Generate n points from $f(x)$ and store in an array $\{D_0, D_1 ...., D_{\text{(n-1)}}\}$
for $j = 1$ `to` n **do**
    $D_{\text{j-1}} \leftarrow ((j, f(j))modp)$
    `Array` $Y[j] \leftarrow D_{\text{j-1}}$
end for
Generate a portion of S in each active node
for $q = 1$ `to` m **do**
    $r_{\text{q}} \leftarrow$ `a random number between 1 and` n
    `Reveal n point` $Y[r_{\text{q}}]$
end for
Each active node broadcasts the received portion of S to all other active nodes
Reveal G, Q_{CA} ▷ Required for later computations

Blockchain nodes can store the network monitoring data locally in off-chain storage (InterPlanetary File System (IPFS) and sends the pointer (the hash of the monitored data) to the blockchain. When necessary, these data can be accessed later at the blockchain by retrieving the data from the off-chain database. This approach prevents unnecessary ledger growth [22].

4) Block Generation: The proposed consensus protocol declares the node which discovers Q_{u}, $Cert_{\text{u}}$ and d_{u} first as the winner miner. The winning miner gets the chance to create the next block of the blockchain. Fig. 7 depicts the block structure that the winner miner has to create. The Block header includes the version, timestamp, hash of the previous block, hash of the Merkle root, and RESULT (which is the solution discovered from the mining process). RESULT = (Q_{u} digitally signed by d_{u}) + signature + $Cert_{\text{u}}$. The body of the block header contains the data pointer to the network monitoring data, blockchain service invocations, and service fee payments. Later, the winning miner broadcasts the generated block to the peer nodes for verification.

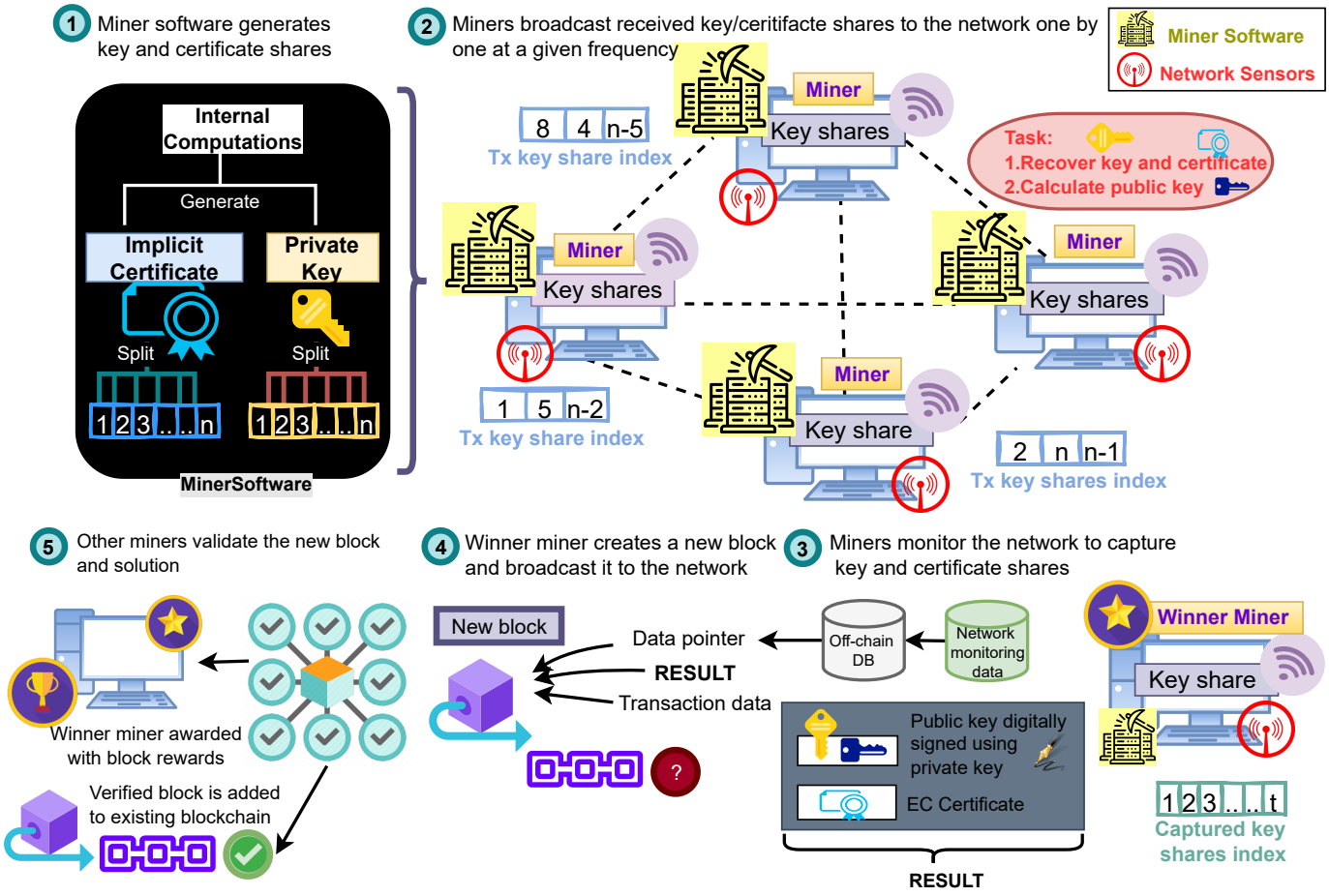


Fig. 6: The workflow of the proposed PoM consensus protocol

Algorithm 3 Key Recovery

Input: k pieces of $S\{(x_0, y_0), (x_1, y_1)...., (x_k, y_k)\ \}$
Output: Recovered d_u, $Cert_u$, Computed Q_u

Require: $l = 1, f(0)$
for $j = 0$ to $k - 1$ **do**
  for $m = 0$ to $k - 1$ **do**
    if $m \neq j$ **then**
      $l \leftarrow l * (x_m)/(x_m - x_j)$
    end if
  end for
  $f(0) \leftarrow f(0) + (l * y_k)$
  $p = 1$
end for
$f(0) \leftarrow S=\{d_u, Cert_u\}$
Compute public key using d_u,$Cert_u$
$Q_u = d_u G$
or $Q_u = e * Cert_u + Q_{CA}$

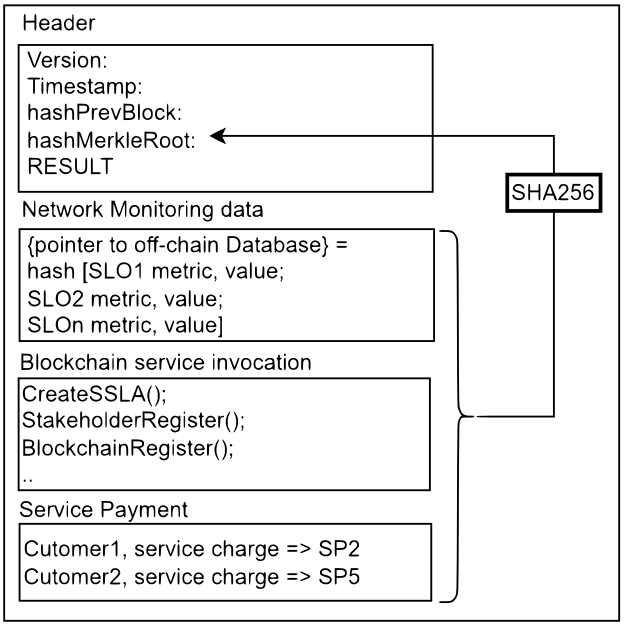


Fig. 7: Block structure of the proposed PoM

5) Block Verification: When the rest of the miners receive the new block, they can find the RESULT solution containing a digitally signed Q_u, the signature, and $Cert_u$ in it. Next, they can calculate the public key Q_u using $Cert_u$ to verify the signature sent by the winner miner. If the verification is successful, the rest of the peer nodes add the verified block to the blockchain, and the winning miner receives incentives.

The proposed system allocates a portion of the service charge as incentives (refer IV-A). Algorithm 4 shows the pseudo-code of the key verification approach.

Algorithm 4 Block Verification

Input: Digitally signed Q_{u}, signature, $Cert_{\mathrm{u}}$
Output: successful or unsuccessful

Calculate Q_{u} using $Cert_{\mathrm{u}}$
$Q_{\mathrm{u}} = e * Cert_{\mathrm{u}} + Q_{\mathrm{CA}}$
ECDSA signature verification
$Q_{\mathrm{u}}{}' \leftarrow$ `decoded message using` Q_{u}
if $Q_{\mathrm{u}}{}' == Q_{\mathrm{u}}$ **then**
    Verification is successful
else
    Verification is unsuccessful
end if

In conclusion, our consensus protocol is a cost-effective and energy-efficient approach since it does not includes a computationally intensive task to solve to achieve consensus.

C. Mining difficulty of the PoM mechanism

Every work-based consensus algorithm consists of a mechanism to adjust the difficulty levels of its defined mining task, allowing control of block creation time, transaction throughput, and competition between miners. In our case, the difficult aspect is mainly regulated based on cryptography, blockchain, and network parameters. Table I lists an estimation of how different factors affect the difficulty of the mining task.

TABLE I: Analysis of mining difficulty of the proposed PoM

Category	Parameter	Impact on difficulty
Cryptography	Key length of EC key/certificate • Depends curve size – Order of curve(n′) ∗ Curve co-factor (h) ∗ Order of subgroup (r) – Finite field Fq (where q is prime) – Field size	Increase
	Threshold (k)	Increase
	Number of shares (n)	Decrease
	Key sharing frequency	Decrease
	Number of miners (m)	Decrease
	Key revealing probability	Decrease
Mining resources	Computational power	Negligibly decrease
Network	Network latency	Increase
	Network bandwidth	Decrease

The difficulty of recovering the key depends mainly on the key size, number of shares, threshold, key sharing frequency, and number of nodes. Obviously, with the increase in key size, the number of key shares to be captured increases. Similarly, setting the threshold to a higher value still increases the number of key shares a miner node has to capture to reconstruct the key. However, more key shares will be available in the network to recover by increasing the number of shares (which determines the number that the key is divided into) while keeping the key size and threshold fixed. Therefore, the availability of many shares in the network lowers the recovery difficulty, while key size and threshold increase the difficulty. In addition, with a lower key sharing frequency (the duration a blockchain node must wait to send its next key share), the completion of the mining task will be delayed. Hence, the difficulty increases with the key-sharing frequency. Nevertheless, having a more significant number of miner nodes (=key distributors) grows the number of key shares in the network, increasing the key recovery probability. Similarly, the difficulty lowers by increasing the key revealing probability (willingness to broadcast its key share to other peer nodes). This fact is elaborated quantitatively in a later section VI-C. Furthermore, the time taken to solve the mining task will be minimal when the computational power of a miner node increases. However, the effect of computational power is negligible on the difficulty since the mining task is not computationally intensive. Moreover, the increase in network latency and decrease in network bandwidth interrupt and delay the process of miners distributing their keys and capturing keys to become winner miners. As a result, the entire mining task operation will get delayed. Therefore, high latency and low bandwidth affect the difficulty aspect negatively.

VI. Numerical Analysis

We carried out multiple numerical simulations to evaluate the performance of the proposed consensus PoM. This section discusses the simulation models developed to assess the internal performance factors of PoM and compare PoM with other traditional consensus protocols. The simulation tool used for all the tests discussed in this section is MATLAB [23].

Simulation model 1 represents a wireless blockchain network with N number of nodes. We evaluated the blockchain performance metrics by varying N from 1 to 1000. Simulation model 2 illustrates an estimation of energy consumption for different blockchain architectures. Simulation model 3 exemplifies a blockchain network connected with m number of miner nodes at the key distribution phase of the PoM, where miner nodes broadcast and capture key shares. We instantiated different instances of this model when m = 4,10,20, and 50. It analyzes key recovering probability by varying key revealing probability from 0 to 1.

A. Simulation Model 1: Inter-parameter comparison between PoM and other conventional consensus mechanisms [24]

This subsection presents a quantitative analysis of the communication impacts on blockchain performance metrics in a wireless blockchain ecosystem. We compared our proposed PoM consensus protocol with widely used consensus algorithms in a SLA/SSLA ecosystem. Namely, PoW [25], [26], [8], Raft [27], [28] and PBFT [29]. The comparison is made with respect to several significant performance metrics such as communication complexity [24], spectrum requirement [24], security bound [24], and transaction throughput [24].

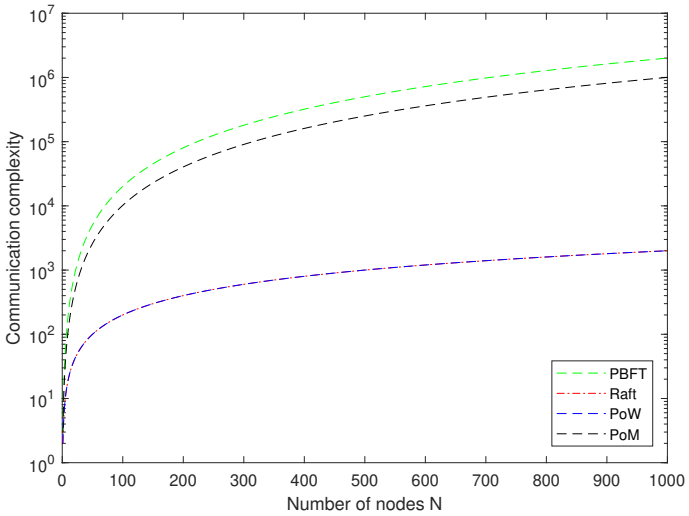


Fig. 8: Comparison of the communication complexity of different consensus mechanisms [24]

1) Communication Complexity: Communication complexity defines the number of communications between transmitter and receiver [24]. Fig.8 exemplifies the communication complexity of commonly used consensus protocols in a wireless network. It is clearly visible that PBFT requires the highest degree of communication $2N+N^2$. It is because all connected nodes must communicate among others at their primary stages of prepare, prepare and commit. On the other hand, Raft and PoW depict the same pattern of requiring up to $2N$ communications. In Raft, the communication complexity is the summation of communications happening between followers to the leader (uplink) and leader to followers (downlink), which adds up to $2N$. However, in PoW and PoM, $2N$ number of communications are required to broadcast the client-transaction request and winner miner's solution to all other connected nodes. In addition, PoM requires communications at the stage where each node (i.e., N) broadcasts its key shares among peer nodes. Hence, at that point, N^2 number of communications are required. However, PoM triggers key re-transmission if it finds no winner miners at a defined interval. Therefore, the communication complexity of PoM is $rN^2 + 2N$, where r is the number of re-transmissions. However, at the ideal stage, $r = 1$ makes the communication complexity of PoM equal to N^2+2N. Note that we considered the ideal stages of each consensus protocol throughout this comparison [24].

2) Spectrum Requirement: The required number of communication spectrum resources or the number of transmitter processes in a wireless blockchain network refers to the spectrum requirement [24]. A comparison is made among consensus algorithms with respect to this regard and demonstrated its results in Fig.9. The PoW model requires two broadcast communications to transmit the transaction request and the winner miner's result. Hence, the overall spectrum requirement of PoW is only 2, which is a constant value and does not depend on the number of nodes. Similarly, PoM protocol includes the transaction broadcasting and broadcasting the winner node's result to all peer nodes at the verification stage (i.e., 2). In addition, each node transmits its key shares among other nodes (i.e., N). Hence, the spectrum requirement of PoM is $N+2$. The Raft model requires broadcasting transmission in downlink (i.e., 1), and each follower node conducts a one-way communication to the leader (i.e., N). Therefore, the spectrum requirement of Raft is equal to $N + 1$. In PBFT, spectrum resources are allocated for broadcasting transactions at the pre-prepare stage (i.e., 1) and the communication between nodes at prepare and commit stages (i.e., $2N$). Hence, the spectrum requirement of PBFT sums up to $2N+1$, which is the highest of all.

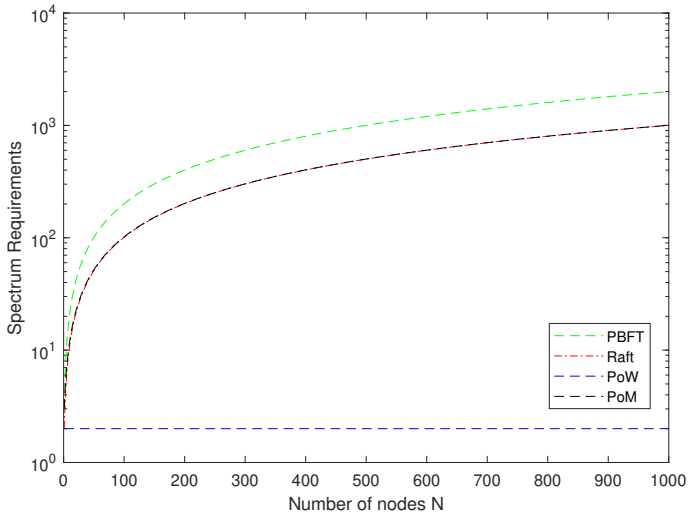


Fig. 9: Comparison of the spectrum requirement of different consensus mechanisms [24]

3) Security Bound: It determines the maximum number of faulty nodes (f) a consensus protocol can withstand [24]. For PoW, the security bound is $2f + 1$ due to the possibility that more than 50% of the computing power within the blockchain network can be acquired by a user, then the whole blockchain jeopardizes. In PoM, more than 50% nodes should verify the winner miner's solution to generate a valid block. Hence, the security bound of PoM becomes $2f + 1$. In contrast, the voting-biased consensus protocols consider faulty nodes as "inactive" or "malicious" nodes transmitting false data within the blockchain network. PBFT permits $1/3$ of nodes to be faulty, making security bound $3f+1$. While, Raft can tolerate 50% faulty nodes, which results in security bound to be $2f+1$.

4) Transaction Throughput: The transactions per second metric measures the transaction throughput. Typically, proof-based consensus algorithms such as PoW suffers a great deal of low throughput compared to other typical consensus protocols. Due to the time taken to conduct intensive computations to achieve consensus is high. Although the mining task of PoM instructs to carry out computations, it does not require plenty of computational power to solve the puzzle. However, in PoM, keys are transmitted at a given frequency, which delays the mining process a bit. Therefore, we can consider the security bound of PoM as "medium". In contrast, the transaction throughput of voting-based consensus algorithms

is excessively high [24].

B. Simulation Model 2: Comparison of per transaction energy utilization [30]

Energy consumption per transaction of typical blockchain architectures is roughly estimated in the paper [30]. To bring PoM into this comparison, we estimated the value by measuring the energy consumption for 100 transactions per block and setting the difficulty level to 4 in our developed simple blockchain network (refer VIII-B). Based on the experimental results, we arrived at a magnitude of 18J per transaction. Fig. 10 illustrates a comparison between different architectures with regard to energy consumption. It is apparent that the PoM can operate transactions at a low energy consumption compared to Public Blockchain systems. Furthermore, in PoM, a portion of the total consumption of Energy (18J) spends on monitoring the network, which is the main functionality of our SSLA management framework. To elaborate more on that, PoM consensus can be achieved only by monitoring the network, which is the primary function of the proposed application. Therefore, it is clear that 18J of Energy (total energy consumption of PoM per transaction) is insignificant compared to other blockchain networks, which consume Energy solely on the consensus. In our case, achieving consensus is a byproduct of network monitoring.

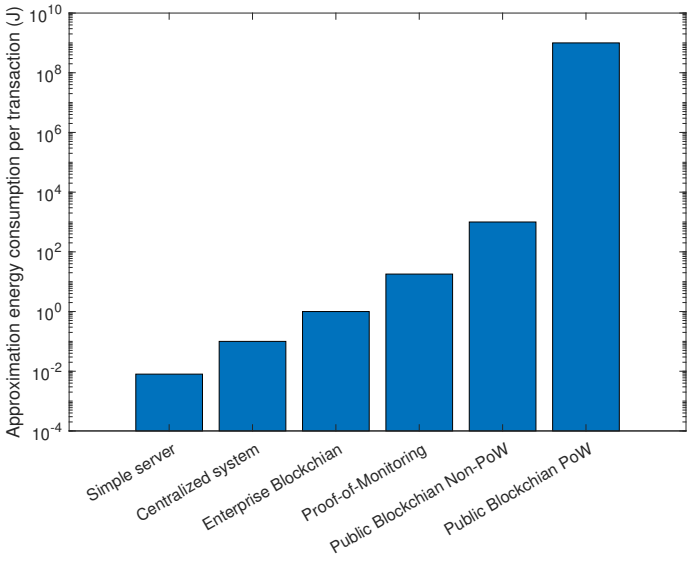


Fig. 10: Comparison of the energy consumption per transaction for different architectures [30]

C. Simulation 3: Intra-parameter comparison of PoM

We analyzed the impact on key recovery by varying one of the difficulty factors listed in Table I, which is the key revealing probability. It is the probability that the miner nodes willingly reveal their key shares with other peer nodes by broadcasting the received key shares to the network. In Fig. 11, we plot the key recovering probability against the key revealing probability, varying the number of miner nodes m in the blockchain network while keeping the threshold k constant. We can formulate the probability of successfully recovering the key [31] as follows. Note that P is the probability of revealing a key share, and $1-P$ is the probability of not revealing a key share to others.

$$P_{\text{KeyRecovery}} = \sum_{i=k}^{m-1} p^{(i+1)}(1-p)^{(m-(i+1))} \tag{10}$$

Based on Fig.11, it is visible that with the increment of the number of nodes, the potential to recover a key dramatically increases. Due to the presence of many cooperative nodes who share their received key shares. Another primary observation is that when the key revealing probability rises with the number of miner nodes, the key recovery rate escalates. That is because of the existence of many nodes that are extremely willing to distribute their key share among others. Hence, every node gets the privilege of receiving a large number of key shares, which ultimately helps with the key reconstruction. As a result, it lowers the difficulty of the mining task.

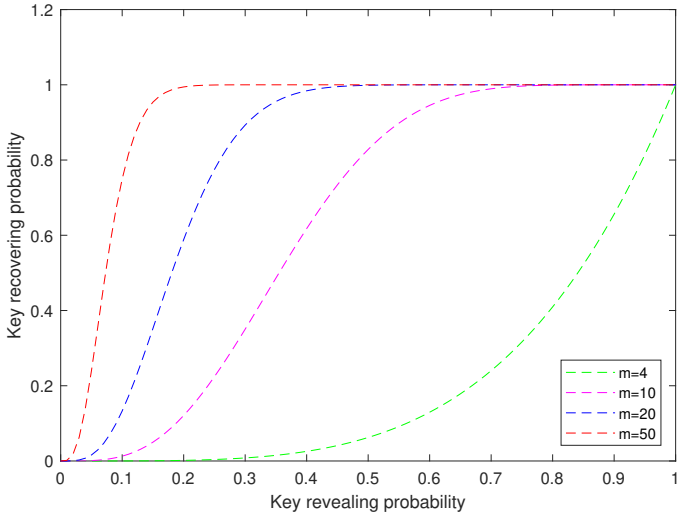


Fig. 11: The impact of key share revealing probability on key recovery probability

VII. Prototypical Implementation

We developed prototypes separately for the two primary modules proposed in this paper. Namely, (1) consensus protocol and (2) SSLA management framework. We have built our own fully functional blockchain prototype from scratch and integrated the proposed consensus protocol. Further, we have executed the proposed SSLA management services on top of the Ethereum test network. These two main implementations are discussed explicitly in this section.

A. PoM

The prototype of the proposed system consists of a newly developed blockchain program using NodeJS, which simulates the behavior of the participating nodes. Fig. 12 shows the experimental setup of the blockchain system. The program has RESTful APIs, which interact with various blockchain operations by the node, i.e., joining the blockchain network,

submitting transactions, fetching block information, mining transactions, and submitting random key and certificate shares in the network. The blockchain program is run on different ports in the same system to simulate multiple nodes. The nodes can join the common blockchain network by sending an API request, which synchronizes the blockchain among the participating nodes by broadcasting the chain.

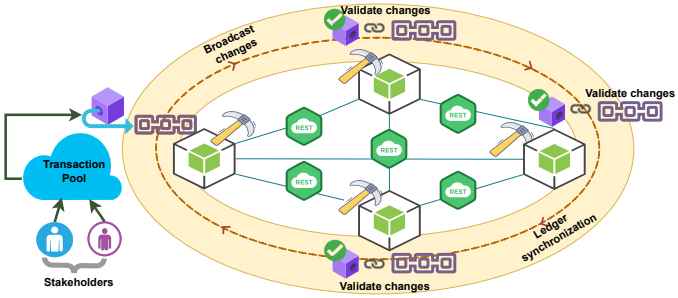


Fig. 12: Prototype setup for the PoM-based blockchain system

A background job is scheduled when a new node is added to the network, which gets triggered every 20 ms. The job ensures that the private key d_{u} and the certificate $Cert_{\mathrm{u}}$ are generated using the previous block timestamp and are divided into parts using the @zippie/secrets.js package, which uses Shamir's threshold secret sharing scheme in JavaScript. During the job, a random part of both generated d_{u} and $Cert_{\mathrm{u}}$ is broadcasted by each node across the network using the broadcast API. The remaining nodes receive the broadcasted shares in a frequency of 10 seconds and store them locally. The difficulty of the system is set by increasing the number of shares to be generated and the minimum number of shares needed to combine them to successfully recover the d_{u} and $Cert_{\mathrm{u}}$. The system's difficulty can increase by increasing the threshold of shares required by the miner to retrieve the random key and certificate successfully.

During the mining process, a mining job has been created that checks if the threshold is reached to combine the shares every 200 milliseconds. The nodes keep listening for more shares if the threshold has not been reached yet to combine the shares. If the threshold is reached, then $Cert_{\mathrm{u}}$ and d_{u} are calculated using the shares. The mining node then calculates the Public Key Q_{u} and signs it using the private key d_{u} using the Elliptic Curve Digital Signature Algorithm (ECDSA). The miner then creates a new block that contains the timestamp, transactions, the signature, Q_{u} and $Cert_{\mathrm{u}}$ and broadcasts information to all the other nodes for verification using the receive-block API, and deletes the mining job. The nodes receiving the block verify the signature using the public key Q_{u} and add the block to the existing blockchain. If the signature has a mismatch, the block gets rejected.

B. SSLA management system

1) Prototype: We developed SSLA architecture on top of a blockchain using Distributed Ledger Technology (DLT), and we selected one of such open-source platforms like Ethereum. We used the Rinkeby test network, an alternative to the main blockchain, as the blockchain service in our

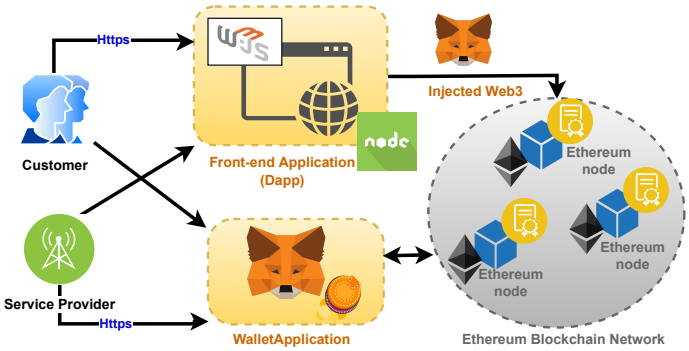


Fig. 13: Prototype setup for the smart contract based SSLA management framework

implementation. The functionalities of each module in the blockchain layer of the proposed architecture (as shown in Fig. 2) are programmed and executed via smart contracts. Fig. 13 depicts the implemented proposed system model. Each customer and SP is connected to a crypto wallet application (In our implementation, we used Metamask) to perform cryptocurrency transactions and interact with the Ethereum blockchain network. The foundation for the client application is an HTTP server deployed in the local host using Node.js. Decentralized Applications (DApps) run on a web browser configured with the MetaMask plugin, which uses Injected web3. It essentially connects the client application, and the Ethereum blockchain ecosystem [32]. The proposed system performs DApp and blockchain communications over a Remote Procedure Call (RPC) protocol. RPC transactions are made possible by the Web3.js library, which translates smart contract scripts to RPC protocol. The client application has access to all the smart contracts deployed on the blockchain, allowing it to perform function calls to manage SSLA management functionalities.

2) Deployment of smart contracts: We executed the developed prototype using Ethereum-based smart contracts and coded the smart contracts using solidity language. We built and deployed Ethereum contracts through a browser-based compiler known as Remix IDE.

The key functions of each smart contract are summarized as follows:

- User Authentication Contract: Registers stakeholders and monitoring nodes. Further, it validates the access permission requests sent by stakeholders.
- Policy Manager Contract: Deploys and stores SSLAs in blockchain
- Violation Monitoring Engine Contract: Stores network monitored data and decides whether a violation happened
- Compensation Calculator Contract: Calculates the compensation fee and the amount to be transferred for the service provider and blockchain for their service deliveries
- Reputation Management Contract: Calculates the reputation scores for service providers and blockchain nodes

VIII. Experimental Results

We evaluated the performance of the proposed PoM and SSLA framework by conducting tests on the custom-built blockchain and the Ethereum-based prototype, respectively. In the custom-built blockchain, we ran tests to compare PoM with PoW with respect to average block creation time and the total energy consumption per block by varying the number of transactions per block and the difficulty levels of the consensus. In the Ethereum-based prototype, we carried out a cost analysis and measured the end-to-end latency of the system. To obtain results for proof of work, we replace the PoM consensus with a locally developed proof of work algorithm, which calculates the correct nonce to get the valid block hash based on different difficulties. For the tests, the nodes in the network are set to 5, broadcasting the shares randomly among each other. The maximum number of shares that can generate is the $difficultyfactor \times numberofnodes$, and the threshold to combine the shares is $maximumshares - (numberofnodes/2)$. One of the nodes sends different sets of transactions ranging from 100 up to 10000, and one of them mines the transactions using RESTful API calls and records the block creation time.

A. Block time evaluation

Fig. 14 depicts the comparison of PoW with PoM in terms of the block creation time for various transactions ranging from 100 to 10000, keeping the difficulty constant for both systems. In the case of PoM, the block creation time is high initially, but as the number of transactions increases, the time decreases and stays almost the same. This behavior is because fewer transactions get processed faster, and the miner still does not have the minimum shares to mine the block. Therefore the miner waits for more shares. As the number of transactions increases, the miner has the required shares by the time all the transactions reach the transaction pool to get mined. Hence the block creation time is less, and it increases slowly with the increment of transactions, which then extends the processing time. For PoW, as the number of transactions increases, we see an increase in the block creation time because generating the valid hash becomes more difficult.

We made a further comparison of both algorithms by calculating the block creation time of 10000 transactions by increasing the difficulty from 1 to 6, as shown in Fig. 15. The block creation time increases gradually, with the threshold shares required by the miner increasing with difficulty. The waiting period for the miner to get the threshold amount of shares contributes to the rise in time. As the broadcast frequency is pretty low, the block creation time increases at a plodding pace.

B. Energy consumption evaluation

We carried out tests to compare the energy consumption between PoM and PoW consensus algorithms. We referred to the equation 11 for calculating the energy consumption. Let P_{system} be the power usage of the system, which is 0.55 kW for all the tests, and T_{block} be the block time derived from previous tests.

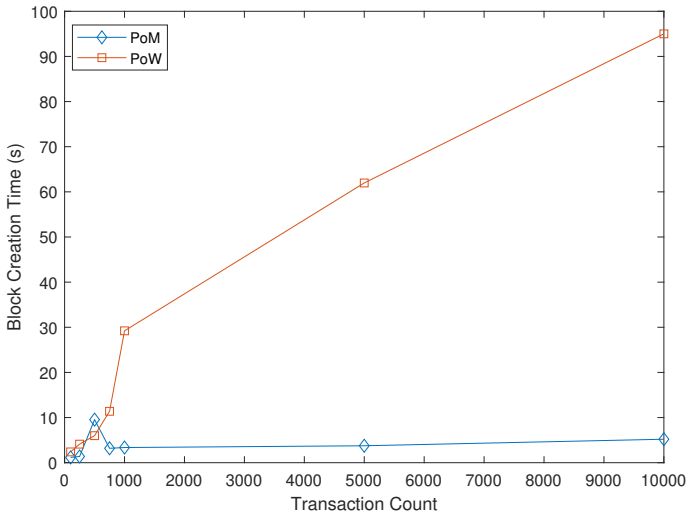


Fig. 14: Average block time against number of transactions per block

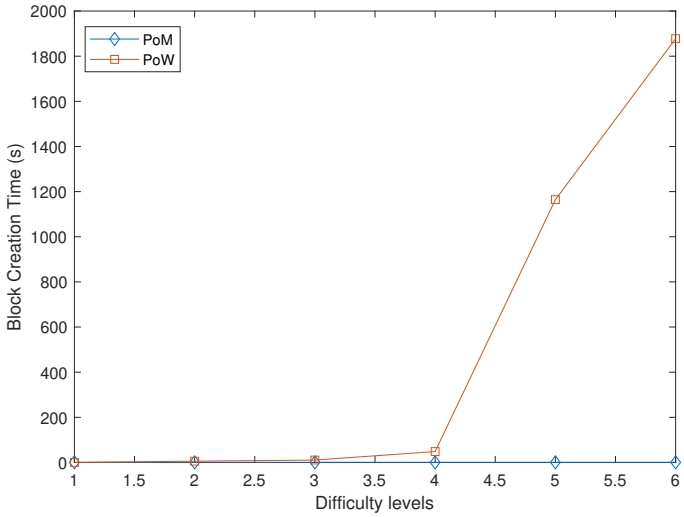


Fig. 15: Average block time versus different difficulty levels

$$Energy = P_{\text{system}} \times \frac{T_{\text{block}}}{1000} \tag{11}$$

We plotted the energy consumption with respect to the number of transactions per block in Fig.16. It is clearly evident that the energy consumption of PoW dramatically increases with the increase in the number of transactions. In contrast, PoM shows less energy consumption even with the variation in the number of transactions per block. Fig. 17 depicts the energy consumption for 1000 transactions for different difficulty levels. The energy consumption of both consensuses remains almost similar. It increases slowly till difficulty 4, after which the energy increases exponentially in the case of PoW, whereas it increases linearly for PoM. The reason is that until a difficulty of 4, the system calculates the valid hash relatively quickly. However, when it increases further, the calculation of the valid hash takes a high amount of time, thus increasing energy consumption. In the case of the PoM, difficulty increase leads to an increase in the threshold shares, which keep getting

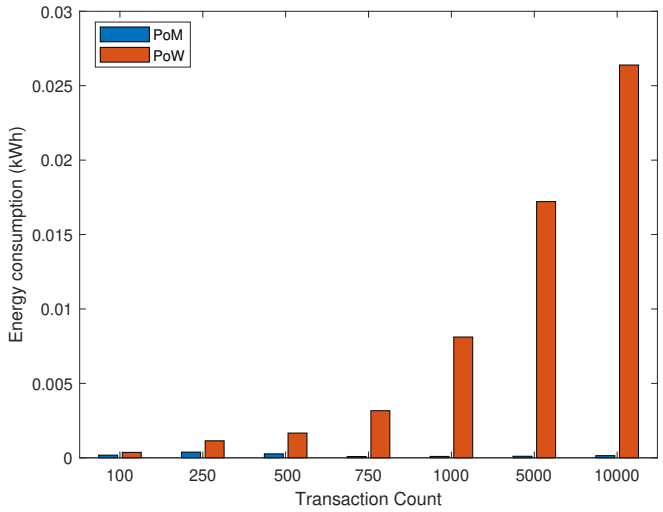


Fig. 16: Total energy consumption per block versus number of transactions per block

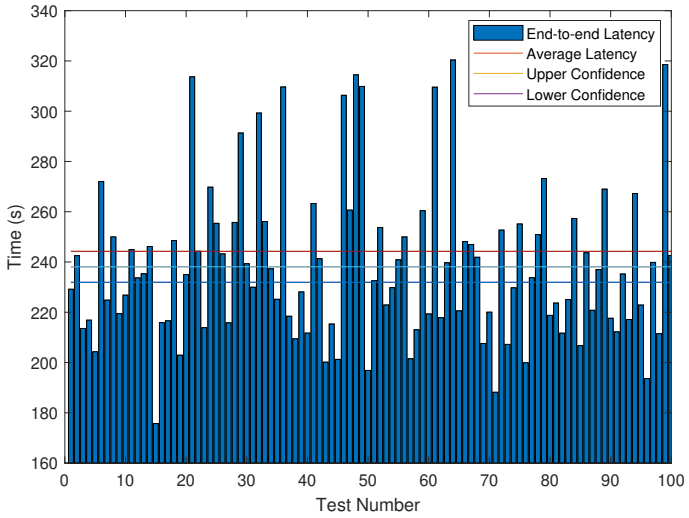


Fig. 18: End-to-end latency from violation reporting to compensating a customer

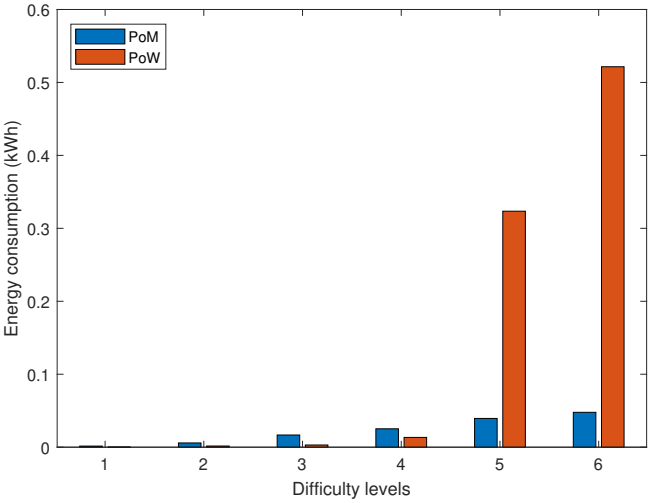


Fig. 17: Total energy consumption per block against different difficulty levels

broadcasted by nodes in fixed intervals. Hence the waiting time for the miner to achieve the threshold increases linearly, resulting in a linear increase in energy consumption.

C. End-to-end latency

We carried out a test to measure the time taken to execute the main functionality of the proposed system. That is, the combination of three main functionalities of the proposed approach: (1) blockchain nodes monitor the network and submit sensed data to the proposed framework (2) Following that, the system decides whether a violation occurred or not (3) finally, imposing penalty (if any violations) on SP, compensating the customer and transferring fee to violation reporters. We coded each of these functionalities in Ethereum-based smart contracts and deployed them in the Rinkeby network (refer VII-B). We invoked each smart contract from the client application (DApp) 100 times, measured the end-to-end latency to run each of the three processes, and summed them together. Each measured end-to-end delay consists of the average block creation time of 15s [33], which is the block creation time of the Rinkeby Testnet. We substituted it with the estimated block verification time of PoM, which we derived by testing. Thereafter, we plotted the previously calculated total end-to-end delay with a 95% confidence interval and demonstrated in Fig. 18. Based on the plotted results, we measured the average end-to-end time taken for the entire process, from data reporting to violation detection and payment settlements, as approximately 4 minutes (238s).

D. Cost analysis

This subsection analyses the costs incurred in smart contract deployment and execution. In the implementation of smart contracts, a payment of a gas fee is required in the Ethereum blockchain to conduct transactions. TableII lists the estimation of the gas consumption for the deployment of contracts and execution of defined functions in the contract. Using the Remix IDE web browser, we found these estimated gas values. Based on the tabulated results, the total cost incurred to deploy all the smart contracts is approximately 14 Euros, a one-time payment. For the execution of all other blockchain service functionalities, it costs around a total of 2 Euros. Based on the results, we can conclude that the cost incurred to implement the whole system is considerable when implemented on the Ethereum blockchain. However, we can significantly reduce these costs by implementing the proposed SSLA management framework on top of our novel blockchain system. Most state-of-art-approaches have implemented their SLA/SSLA services over the Ethereum platform, meaning they have run their services on a PoW-based blockchain platform. In our case, we will be utilizing the proposed PoM-based blockchain, which is custom-built for the management of SSLAs.

TABLE II: Gas consumption analysis of Ethereum-based contracts utilized in SSLA management services

Contract/Functions	Cost	
	Gwei	EUR[a]
User Authentication Contract	713139	2.04
Create User Function	118619	0.334
User Verification Function	24613	0.07
Policy Manager Contract	1515118	4.335
Create SSLA Function	91533	0.026
Customer Approval Function	47703	0.136
Violation Monitoring Engine Contract	1310897	3.75
Security Metric Storage Function	25420	0.0727
Violation Decision Function	23789	0.068
Compensation Calculator Contract	931808	2.666
Stakeholder Compensation Function	23809	0.068
Reputation Management Contract	419667	1.2
Reputation Calculation Function	31117	0.089

1 Ether = 10^9 Gwei,[a]1 ether = EUR 2861.21 on 20.04.2022

IX. Security and Formal Analysis

This section defines and evaluates the general and security properties of the PoM-based blockchain system. Further, we present a threat model with potential attacks and defense strategies. In the end, we present the formal model and formal verification of the PoM consensus protocol analyzing its results.

A. Definitions

Definition 1: Chain Quality [34] Let $\mu \in (0, 1]$ be the chain quality coefficient. The chain quality Q_{CQ} expresses that if $l(l \in \mathbb{N})$ number of successive blocks of a chain C acquired by an honest party, then a portion of μ blocks mined by attackers. Then the expected lower bound on the proportion of honest blocks of chain C can be stated as $Q_{\text{CQ}} = 1 - \mu$.

Definition 2: Chain Growth [34] Let $\tau \in (0, 1]$ be the chain speed coefficient. The chain growth Q_{CG} expresses that if the chains C_1, C_2 held by honest parties at rounds r_1, r_2 with chain lengths l_1, l_2 respectively. r_2 is ahead of r_1 by $s(s \in \mathbb{N})$ number of rounds. Then, it satisfies the $r_2 - r_1 \geq \tau.s$

Definition 3: Fork Probability The fork probability Q_{FP} expresses the potential for blockchain nodes to receive more than one block simultaneously.

Definition 4: Chain Consistency [35] The chain consistency Q_{CC} assures that all honest parties will produce the same order of blocks in round r, allowing only the last T number of blocks to be changed.

Definition 5: Blockchain Fork Blockchain fork may trigger when many perspectives on the state of the blockchain exist. That is when several nodes mine simultaneously and may find conflicting blocks which split the main chain into multiple chains.

B. Property analysis of the PoM-based blockchain

This subsection presents the analysis of the primary blockchain properties of the proposed consensus protocol. Namely, fork probability, chain growth, chain consistency, and chain quality. We developed different models to assess each of the properties. We designed a probabilistic model to evaluate fork probability and chain growth. Whilst a Markov model to analyze chain consistency. In addition to that, we refer to the simulation model discussed in Section VI-C to assess the chain quality.

1) Fork Probability: We divide secret S into n number of unique key shares. For a successful reconstruction of S (or finding the mining solution), a node must recover at least $k(< n)$ number of unique key pieces. Let us assume, as an average, that a single node receives keys at a rate of γ key shares per second. Therefore, the number of key pieces received by the node after q seconds is γq. A node acquires keys from itself and its neighboring nodes. Each key share received draws from n total key shares. The same key share can be received multiple times from other nodes even though unique key shares contribute to achieving the consensus. Therefore, we assume receiving keys as events of simple random sampling with replacement. Let the probability of finding a mining solution at q seconds is Y_{q}, which is the same as the probability of having minimum k unique key shares within $\gamma q(\geq k)$ key shares. Let U_{k} be the probability of having at least k unique keys after q seconds, and N be the number of unique key shares a node captured after q seconds,

$$U_{\text{k}} = Pr(N \geq k) \tag{12}$$

It is obvious that $U_1 = 1$. U_2 is derived from U_1 with probability $1 - \frac{1}{\gamma q}$, or from U_2 itself with probability $\frac{2}{\gamma q}$. That is because there are γq items in total. Hence, we can deduce U_2 as follows,

$$U_2 = (1 - \frac{1}{\gamma q}) * U_1 + (\frac{2}{\gamma q}) * U_2 \tag{13}$$

Therefore, we formulate the general equation as below,

$$U_{\text{k}} = \frac{[1 - \frac{k-1}{\gamma q}] * U_{\text{k-1}}}{[1 - \frac{k}{\gamma q}]} \tag{14}$$

By solving the system of equations, we obtain the probability of having at least k unique key shares after q seconds, which we denoted as $Y_{\text{q}}(= U_{\text{k}})$.

Blockchain fork occurs due to the propagation delay in the network. Let α be the mean propagation delay. Then, the probability of finding a solution during α period can be defined as the fork probability Q_{FP}, which can be given as follows,

$$Q_{\text{FP}} = Y_{(\text{q}+\alpha)} - Y_{(\text{q})} \tag{15}$$

2) Chain Growth: Maintaining consistent chain growth is always desirable throughout the evolution of the blockchain. However, with the presence of adversary miners, the chain growth can be negatively influenced to make the chain stagnant. In our case, an optimal adversary strategy would be to reduce the chain growth by not sharing their key shares with neighboring nodes. Therefore, the expected number of key shares received by a node per second (γ) drops proportionally to the fraction of adversary nodes. Then the key receiving rate

with the presence of adversaries (γ_{Z}) can be derived as shown in the equation 16,

Let the fraction of adversary nodes in the network be β, then the fraction of honest nodes will be $1-\beta$.

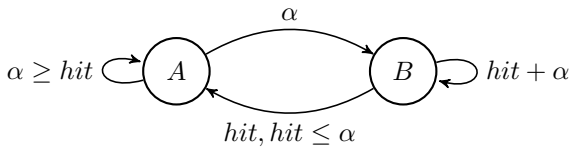
$$\gamma_{\mathrm{Z}} = (1-\beta)\gamma \tag{16}$$

Therefore, the probability of finding a mining solution at any given q second with the presence of adversaries (denoted by $Y_{\mathrm{q}}{}'$) can be deduced by substituting γ with γ_{Z} in equation 14, which gives the following equation 17.

$$Y_{\mathrm{q}}{}' = \frac{[1-\frac{k-1}{(1-\beta)\gamma q}] * U_{\text{k-1}}}{[1-\frac{k}{(1-\beta)\gamma q}]} \tag{17}$$

3) Chain Consistency: We consider the Markov Chain approach [35] in analyzing chain consistency. We rely on convergence opportunities to construct the relationship between when a new block is mined and all nodes synced with the new block. It is a three-step event, (1) No honest player has mined in α rounds. Hence, by the end of α rounds, all honest players learned about all the existing blocks, and they all agreed on the length of the longest chain. (2) an honest player finds the mining solution and mines a block, which will be the last block of the longest chain. (3) Following another α round where no honest player mines. Hence, every honest player learns about the new block and agrees on the longest chain. Note that we define a *round* as an attempt made to mine a block when a node receives a new key.

Based on [35], in order to prove that PoM achieves consistency, we first need to demonstrate the possibilities of how an adversary can break aforementioned convergence opportunities at any given time. Otherwise, honest players have the opportunity of converging on the same chain. For the analysis, we need to count the expected number of convergence opportunities at a given time and compare it with the expected number of blocks that the adversary can mine, which is the minimum amount of work that the adversary must commit to prevent the convergence of honest players. An adversary can break convergence opportunities by inserting one of their blocks in the quiet period of steps (1) and (3). Hence, multiple blocks will be at the same level, creating a fork for honest players to disagree. The Markov model comes in handy in counting events that occur at random intervals. In our situation, blocks can be found close together and far apart during a given time. Therefore we need to count precisely how long these quiet periods are. The simple Markov chain of convergence opportunity [35] corresponding to our PoM protocol can be modeled as follows,



We present two states in the Markov model, $state\ A$: honest mined blocks are found close together, where there is no guarantee of convergence opportunity since the time between the blocks is always less than α. With the existence of α rounds where no player mines, the transition to $state\ B$ occurs. At $state\ B$, a convergence opportunity can occur if an honest block is created and a α period of silence follows that. Otherwise, the system state changes back to $state\ A$ if two honest blocks are found together at less than α time.

Let P_α be the probability of α rounds being unsuccessful (silent), and then we can state P_α as shown in the equation 18,

$$P_\alpha = (1-Y_{\mathrm{q}})^\alpha \tag{18}$$

Let e_{ab} be the edge from $state\ A$ to $state\ B$. Hence, we can compute the stationary distribution for states and edges of the Markov model as follows,

$$\begin{gathered} P[e_{00}] = P[e_{10}] = 1-P_\alpha \\ P[e_{01}] = P[e_{11}] = P_\alpha \\ \pi_0 = P[A] = (1-P_\alpha)\pi_0 + (1-P_\alpha)\pi_1 \\ \pi_1 = P[B] = P_\alpha\pi_1 + P_\alpha\pi_0 \end{gathered} \tag{19}$$

Based on the research work presented in [35], we can derive that our model satisfies consistency if the number of convergence opportunities that can occur in T rounds is greater than or equal to the maximum number of blocks mined by the adversary during the same T rounds. Hence, we can state the equation 20. Note that $\mathrm{l_{ij}}$ is the expected time spent on each edge, and $\delta(>0)$ is the tuning parameter.

$$T * \frac{{P_\alpha}^2}{\sum_{i,j} P[e_{\text{ij}}]\pi_{\text{ij}} l_{\text{ij}}} \geq T*(1+\delta)\beta \tag{20}$$

4) Chain Quality: In our blockchain network, malicious nodes may try to mine blocks selfishly without sharing or revealing their portion of key shares to other nodes, which reduces the chances of reconstructing the key during the desired time interval. This type of scenario is simulated under Section VI-C and based on its results (presented in Fig.11), we can conclude that key reconstruction probability is high with the existence of a large number of nodes in the blockchain network including both honest and malicious nodes. Hence, such malicious behaviors hardly threaten the overall mining activity.

C. Threat model

A malicious miner can model different threats to interfere with the functionality of the consensus process. Blockchain fork triggers when many perspectives on the state of the blockchain are present. That is when several nodes mine simultaneously and may find conflicting blocks, which splits the main chain into multiple chains. In addition to that, there is a potential to have multiple winner miners at the following stages; Key revealing stage: at round r, if a majority of nodes require only a few key shares to find the mining solution, then at the next round $r+1$ there is a high probability of finding the mining solution by more than one node. Key distribution stage: If nodes can find the desired key when they start distributing their key shares. To overcome these situations, we have implemented effective tie-breaking criteria such as longest chain and heaviest chain.

Malicious miners may try to mine blocks secretly without sharing their portion of key pieces with other nodes while receiving key pieces from others. To avoid this behavior, nodes continuously monitor the broadcasts of neighboring nodes to identify nodes with poor sharing patterns. If an uncooperative node is detected, using a tit-for-tat like method, honest parties refrain from sharing key pieces with such nodes.

Furthermore, selfish mining is another potential malicious behavior that can appear in the network, where selfish miners disclose mined blocks to the public blockchain network when they possess a longer chain. However, in our system, the probability of introducing a new block to the network by doing selfish mining is extremely low. This is because selfish miners do not receive keys from the majority of the network. This results in extremely high block creation times. Therefore, the length of the chain created by honest nodes will be very long compared to that of selfish miners' chain, making our model tolerant of selfish mining.

D. Formal analysis of PoM

We developed our proposed consensus protocol using CSP# [36] in the Process Analysis Toolkit (PAT) [37] model checker. CSP# is a sub-language of Communicating Sequential Processes (CSP), and it also supports C# libraries to be imported where we can define required custom data structures and functions. Our model made use of this property substantially to implement fundamental features and functions of our blockchain system. We ran the verification via PAT to validate the general and security properties of the designed formal model by defining several assertion rules. The complete CSP model and the C# library for the PoM consensus are available in a GitHub repository. [1]

1) Formal Modeling: We modeled a blockchain network with N number of nodes, where each node is connected with a few other nodes. We allowed inter-node communication to happen through channels we built between each connected node. Each node runs a series of processes concurrently to achieve consensus. These processes are defined as sequences of events and represent basic functionalities of the proposed consensus protocol, which is required to find a winning node to create the block. According to CSP# [36] modeling language, we can denote our blockchain system as a sequence of fundamental processes that complement the main phases of our PoM protocol as follows,

$BlockChain() = Initialize(); StartBroadcast();$

$StartBroadcast() = (||x : \{1..N\}@(Reveal(x);$
$(Send(x)||Receive(x)); IsMined())); StartBroadcast();$

Note that $P; Q$ denotes a process P followed by process Q and $P||Q$ denotes when P and Q processes run in parallel. The $initialize()$ process generates a master key M, a random sequence of n keys. It assigns each node a master key by shuffling key shares of M.

The $Reveal(x)$ process, when called, reveals node x a key at a time and adds revealed keys to a list (called $Captured\ Keys$). The $Send(x)$ process distributes keys only to their neighbor nodes through channels connecting neighbor nodes. $Receive(x)$ process captures keys distributed by their neighbor nodes and adds to its $Captured\ Keys$ list if and only if the received key is not in its list of collected keys. $IsMined(x)$ process inspects whether any node has captured k number of unique keys, where k is the threshold, the number of keys required to find the winning solution. If a winning miner has been found, the $IsMined(x)$ process sets a flag to indicate it, and the verification program can verify it. $StartBroadcast()$ process allows N number nodes to execute sequence of processes including $Reveal(x)$, $Receive(x)$ and $IsMined(x)$ in parallel, which is considered as one cycle. Inclusion of $StartBroadcast()$ at the end of $StartBroadcast()$ permits the algorithm to run continuously.

2) Formal Verification : We validated our formal model of the proposed consensus protocol using the PAT model checker by considering four aspects for verification. We simulated these four aspects by varying the number of malicious nodes in the system. Scenario 1 (S1): when none of the nodes are malicious, Scenario 2 (S2): when one-third of total nodes are malicious, Scenario 2 (S3): when half of the total nodes are malicious and Scenario 3 (S4): when two-third of nodes are malicious. We consider nodes malicious when they do not cooperate in distributing their collected keys within the network. We performed formal verification by setting the following model parameters in the designed formal model of the PoM: N=6 number of nodes, (n,k) = (20,18) number of keys. We validated our system under the aforementioned scenarios against the following set of properties and tabulated its results in the table III,

- **Deadlock-free (A1)** assures that there does not exist any interference continuing any process and does not halt any activities of the nodes unexpectedly and indefinitely.
- **Consensus (A2)** examines whether any node in the network is successful in acquiring k number of unique keys to become the winning miner. If a node manages to become the winner, we consider that our system has reached a consensus.
- **No Blockchain Fork (A3)** guarantees a network with no more than one winning node within the same cycle. Here, a fork can be created during key revealing or key receiving stages.

TABLE III: Verification results against the properties of the formal model

Assertion Rules	Deadlock free (A1)	Consensus (A2)	No Blockchain Fork (A3)
#assert Blockchain() with no malicious nodes (S1)	✓	✓	✓
#assert Blockchain() with minority malicious nodes (S2)	✓	✓	✓
#assert Blockchain() with half malicious nodes (S3)	✓	✓	✓
#assert Blockchain() with majority malicious nodes (S4)	✓	✓	✓

3) Results Analysis: Based on table III, we can come to the following conclusions corresponding to each assertion rule,

[1] https://github.com/nisitaw94/PoM.git

- **A1:** We can deduce that the system is deadlock-free despite the model variations.
- **A2:** We can observe that regardless of malicious activities present in the network, the system achieves consensus. However, in order for the time to achieve consensus to be in a reasonable range, the number of malicious nodes should be as low as possible.
- **A3:** We can conclude that our model guarantees that no blockchain fork occurs under all four scenarios from the start until the winner of the first block is found.

One of the drawbacks of PAT we experienced during our study is that the time it takes to run a verification grows exponentially when the number of nodes increases. It is because of the nature of the model-checking technique PAT uses, which allows for a state explosion to occur with the increasing complexity of the model [38]. Hence, we restricted our model to six nodes.

X. Discussion

A. Comparison with existing systems

TABLE IV: Feature comparison with related works

Features	[3]	[25]	[26]	[8]	Ours
SLA/SSLA	SSLA	SLA	SLA	SLA	SSLA
Blockchain-based	✗	✓	✓	✓	✓
Consensus Protocol	-	PoET	PoW	PoW	PoM
SSLA-oriented Consensus Protocol	-	✗	✗	✗	✓
Automated System	✓	✓	✓	✓	✓
Network Sensing	✓	✗	✗	✗	✓
Violation Detection	✓	✓	✓	✓	✓
Customer Compensation	✗	✓	✓	✓	✓
SP Penalty Scheme	✗	✗	✓	✗	✓
Reputation Management	✗	✗	✗	✓	✓
Off-chain Storage	✗	✗	✗	✗	✓
Computational Complexity	-	Low	High	High	Low

✓→ Yes, ✗→ No, - → Not Applicable

Table IV compares the proposed SSLA management system with the existing state of the work. There is still no novel blockchain-based SSLA management system introduced. However, there is plenty of research work on SLAs now run on the blockchain. Hence, we compared our work with both SLA and SSLA research studies. Blockchain systems are way better than non-blockchain system since it permits automating the whole system by running services via smart contracts. Nevertheless, there are still bottlenecks in Blockchain-based systems, such as high energy consumption, high computational complexity, and high cost. Our solution rectifies these challenges by introducing a SSLA application-biased consensus protocol.

B. Challenges

During the development of the prototype few design challenges were identified, i.e., to generate the elliptic curve for key computation, a random number had to be used to get the generator point on the curve. However, for each miner to generate the same master key, The generator point had to be the same for all the miners. It is practically not achievable. Hence, we used a combination of the last block's timestamp and its creator's address in place of the random number. In addition, during the block verification phase, the winner miner was supposed to broadcast the message consisting of the public key Q_{u} by encrypting it with the private key d_{u}. Since the block verification has to be done by every node in the network while syncing their local blockchain, instead of using an asymmetric cryptographic algorithm ECDSA, we decided to incorporate a symmetric cryptographic algorithm. So that the broadcasting miner can sign the winning message using the common key Q_{u} and the nodes can verify the signature using their version of Q_{u} and verify the message for a successfully mined block.

There are plenty of modular blockchains available that claim to be fully customizable. However, it is inevitable that the customized blockchain shares most of its fundamental characteristics with its counterpart in most cases. As a result, they still need to be more flexible to adapt to a custom blockchain with a work-based consensus algorithm. For example, the Stratis platform consists of PoW/PoS/ Proof-of-Authority (PoA) consensus options, while Hyperledger supports KAFKA/SOLO. Even though they are customizable, in order to be a feasible model, it is expected from adapters that their new algorithms inherit from the existing ones, which essentially limits the customisation only up to a certain extent. In contrast, we implemented our proposed consensus algorithm in a local blockchain developed using NodeJS from scratch.

XI. Conclusion

In this research, we thoroughly evaluated the potential challenges of conventional SLA/SSLA management systems. We introduced an automated SSLA management framework with an accompanying custom blockchain to mitigate them. Based on the experimental results of the implemented PoM consensus, we concluded that the system consumes less time, energy, and cost compared to PoW, the most commonly used consensus algorithm in state-of-art blockchain-based SLA systems. The calculations indicate satisfactory end-to-end latency levels even though such SLA management systems are not time-critical. More importantly, the overall performance of our solution in terms of available security features proves to outperform most other platforms available. With the utilization of off-chain databases to securely store monitoring data, we prevented excessive growth of the ledger, thereby improving scalability and adaptability. As the very first blockchain-based SSLA management framework, we identified potential application scenarios for the given system in current and future networking paradigms. In the future, we plan to further expand the implementation of SSLA management services and the proposed blockchain system by leveraging AI-based solutions to predict SSLA violations beforehand.

Acknowledgment

This research has been supported by the Academy of Finland, 6G Flagship program under Grant 346208 and the Science Foundation Ireland under Connect Center (13 RC/ 2077_P2). The research leading to these results partly received

funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 871808 (5G PPP project INSPIRE-5Gplus). The paper reflects only the authors' views. The Commission is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] E. Marilly, O. Martinot, S. Betge-Brezetz, and G. Delegue, "Requirements for service level agreement management," in *IEEE Workshop on IP Operations and Management*, 2002, pp. 57–62.

[2] C. Lee, K. M. Kavi, R. A. Paul, and M. Gomathisankaran, "Ontology of Secure Service Level Agreement," in *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*, 2015, pp. 166–172.

[3] P. Alemany, D. Ayed, R. Vilalta, R. Muñoz, P. Bisson, R. Casellas, and R. Martínez, "Transport network slices with security service level agreements," in *2020 22nd International Conference on Transparent Optical Networks (ICTON)*. IEEE, 2020, pp. 1–4.

[4] E. Viegas, A. Santin, J. Bachtold, D. Segalin, M. Stihler, A. Marcon, and C. Maziero, "Enhancing service maintainability by monitoring and auditing SLA in cloud computing," *Cluster Computing*, vol. 24, no. 3, pp. 1659–1674, 2021.

[5] H. Natarajan, S. Krause, and H. Gradstein, *Distributed Ledger Technology and Blockchain*. World Bank, 2017.

[6] Z. Tian, M. Li, M. Qiu, Y. Sun, and S. Su, "Block-DEF: A Secure Digital Evidence Framework Using Blockchain," *Information Sciences*, vol. 491, pp. 151–165, 2019.

[7] H. Nakashima and M. Aoyama, "An automation method of sla contract of web apis and its platform based on blockchain concept," in *2017 IEEE International Conference on Cognitive Computing (ICCC)*. IEEE, 2017, pp. 32–39.

[8] H. Zhou, X. Ouyang, Z. Ren, J. Su, C. de Laat, and Z. Zhao, "A blockchain based witness model for trustworthy cloud service level agreement enforcement," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1567–1575.

[9] E. Marilly, O. Martinot, S. Betge-Brezetz, and G. Delegue, "Requirements for service level agreement management," in *IEEE Workshop on IP Operations and Management*, 2002, pp. 57–62.

[10] J. S. Duela and P. U. Maheswari, "Mitigation of DDoS Threat to Service Attainability in Cloud Premises," *International Journal of Reasoning-based Intelligent Systems*, vol. 10, no. 3-4, pp. 337–346, 2018.

[11] C. A. B. De Carvalho, R. M. de Castro Andrade, M. F. de Castro, E. F. Coutinho, and N. Agoulmine, "State of the Art and Challenges of Security SLA for Cloud Computing," *Computers & Electrical Engineering*, vol. 59, pp. 141–152, 2017.

[12] K. Shafique, B. A. Khawaja, F. Sabir, S. Qazi, and M. Mustaqim, "Internet of Things (IoT) for Next-Generation Smart Systems: a Review of Current Challenges, Future Trends and Prospects for Emerging 5G-IoT Scenarios," *IEEE Access*, vol. 8, pp. 23 022–23 040, 2020.

[13] P. Porambage, C. Schmitt, P. Kumar, A. Gurtov, and M. Ylianttila, "PAuthKey: A pervasive authentication protocol and key establishment scheme for wireless sensor networks in distributed IoT applications," *International Journal of Distributed Sensor Networks*, vol. 10, no. 7, p. 357430, 2014.

[14] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[15] F. Faniyi and R. Bahsoon, "A systematic review of service level management in the cloud," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, pp. 1–27, 2015.

[16] P. Grubitzsch, I. Braun, H. Fichtl, T. Springer, T. Hara, and A. Schill, "ML-SLA: Multi-Level Service Level Agreements for Highly Flexible IoT Services," in *2017 IEEE International Congress on Internet of Things (ICIOT)*, 2017, pp. 113–120.

[17] S. Hm and G. Prakash, "Trust Modeling and Service Level Agreement Monitoring in Federated Cloud," 2021.

[18] K. Xiao, Z. Geng, Y. He, G. Xu, C. Wang, and Y. Tian, "A blockchain-based privacy-preserving 5G network slicing service level agreement audit scheme," *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, no. 1, pp. 1–16, 2021.

[19] R. Ranchal and O. Choudhury, "SLAM: A Framework for SLA Management in Multicloud ecosystem using Blockchain," in *2020 IEEE Cloud Summit*, 2020, pp. 33–38.

[20] H. Afzaal, M. Imran, M. U. Janjua, and S. P. Gochhayat, "Formal Modeling and Verification of a Blockchain-Based Crowdsourcing Consensus Protocol," *IEEE Access*, vol. 10, pp. 8163–8183, 2022.

[21] W. Y. M. M. Thin, N. Dong, G. Bai, and J. S. Dong, "Formal analysis of a proof-of-stake blockchain," in *2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 2018, pp. 197–200.

[22] J. Rupasena, T. Rewa, K. T. Hemachandra, and M. Liyanage, "Scalable Storage Scheme for Blockchain-Enabled IoT Equipped Food Supply Chains," in *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, 2021, pp. 300–305.

[23] "MATLAB," last accessed 15 April 2022. [Online]. Available: https://www.mathworks.com

[24] L. Zhang, H. Xu, O. Onireti, M. A. Imran, and B. Cao, "How Much Communication Resource is Needed to Run a Wireless Blockchain Network?" *arXiv preprint arXiv:2101.10852*, 2021.

[25] R. B. Uriarte, H. Zhou, K. Kritikos, Z. Shi, Z. Zhao, and R. De Nicola, "Distributed service-level agreement management with smart contracts and blockchain," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 14, p. e5800, 2021.

[26] C. Schweizer, "SLAMer: a blockchain-based SLA Management System," 2019.

[27] P. Abhishek, A. Chobari, and D. Narayan, "SLA Violation Detection in Multi-Cloud Environment using Hyperledger Fabric Blockchain," in *2021 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*. IEEE, 2021, pp. 107–112.

[28] A. Alzubaidi, K. Mitra, P. Patel, and E. Solaiman, "A blockchain-based approach for assessing compliance with sla-guaranteed iot services," in *2020 IEEE International Conference on Smart Internet of Things (SmartIoT)*. IEEE, 2020, pp. 213–220.

[29] A. T. Wonjiga, S. Peisert, L. Rilling, and C. Morin, "Blockchain as a trusted component in cloud SLA verification," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, 2019, pp. 93–100.

[30] J. Sedlmeir, H. U. Buhl, G. Fridgen, and R. Keller, "The energy consumption of blockchain technology: beyond myth," *Business & Information Systems Engineering*, vol. 62, no. 6, pp. 599–608, 2020.

[31] P. Porambage, Y. Miche, A. Kalliola, M. Liyanage, and M. Ylianttila, "Secure keying scheme for network slicing in 5G architecture," in *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 2019, pp. 1–6.

[32] W.-M. Lee, "Using the Metamask Chrome Extension," in *Beginning Ethereum Smart Contracts Programming*. Springer, 2019, pp. 93–126.

[33] "Ethereum Testnet," last accessed 22 April 2022. [Online]. Available: https://www.rinkeby.io/

[34] A. Kiayias and G. Panagiotakos, "Speed-security tradeoffs in blockchain protocols," *Cryptology ePrint Archive*, 2015.

[35] L. Kiffer, R. Rajaraman, and A. Shelat, "A better method to analyze blockchain consistency," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 729–744.

[36] J. Sun, Y. Liu, J. S. Dong, and C. Chen, "Integrating specification and programs for system modeling and verification," in *2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering*. IEEE, 2009, pp. 127–135.

[37] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "PAT: Towards flexible verification under fairness," in *International conference on computer aided verification*. Springer, 2009, pp. 709–714.

[38] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, "Model checking and the state explosion problem," in *LASER Summer School on Software Engineering*. Springer, 2011, pp. 1–30.

Nisita Weerasinghe is a Doctoral student of NetSEC (Network security, trust and privacy) research group at Center for Wireless Communications (CWC), University of Oulu, Finland. She received her B.Sc degree in Electrical and Electronic Engineering from Sri Lanka Institute of Information Technology (SLIIT), Malabe, Sri Lanka, in 2018 and Master of Science in Wireless Communications Engineering from the University of Oulu, Finland, in 2020. She has over two-year research experience in CWC, University of Oulu in the area of Blockchain, Local 5G networks and Network Security. Her research interests include Blockchain, 5G, Local 5G Operators, Network Security, Network Slicing.

Raaj Mishra is a Product Lead at Dell Technologies, Bangalore, India. He finished his B.Tech at the Department of Computer and Communication Engineering, School of Computing and Information Technology, Manipal University Jaipur, India in 2020. Raaj has experience in fullstack software and blockchain-based decentralized application (DApp) development. He has also worked on two Blockchain based projects and has published papers at IEEE CCNC 2020 and IEEE 5G World Forum 2020. He also has conducted three workshops: "Blockchain for Cyber-Physical Systems and IoT" at ANTS 2019 and "Blockchain Technology, Its Technical Challenges and Role of Formal Methods" at ETCI 2021, "Emerging Applications of Blockchain Technologies in India" at ISPDA 2022. His research interests include blockchain, decentralized Applications, microfrontends and network security.

Pawani Porambage is a senior researcher at VTT Technical Research Centre of Finland and an Adjunct Professor at University of Oulu, Finland. She was a researcher at the Centre for Wireless Communications, University of Oulu, Finland over ten years. She obtained her B.Sc. Degree in Electronics and Telecommunication Engineering from University of Moratuwa, Sri Lanka in 2010, MSc. Degree in Ubiquitous Networking and Computer Networking from University of Nice Sophia-Anipolis, France in 2012, and Doctor of Technology in communication engineering from University of Oulu, Finland in 2018. She has over nine years of experience in network security domain and co-authored more than 50 publications. Her main research interests are network slicing, blockchain, lightweight security protocols, security and privacy on IoT, and AI/ML for security and privacy.

Madhusanka Liyanage (Senior Member, IEEE) is an Assistant Professor/Ad Astra Fellow and Director of Graduate Research at the School of Computer Science, University College Dublin, Ireland. He is also acting as a Docent/Adjunct Professor at University of Oulu, Finland, University of Ruhuna, Sri Lanka and University of Sri Jayawardenepura, Sri Lanka. He received his Doctor of Technology degree in communication engineering from the University of Oulu, Oulu, Finland, in 2016. From 2011 to 2012, he worked as a Research Scientist at the I3S Laboratory and Inria, Sophia Antipolis, France. He was also a recipient of the prestigious Marie Skłodowska-Curie Actions Individual Fellowship and Government of Ireland Postdoctoral Fellowship during 2018-2020. During 2015-2018, he has been a Visiting Research Fellow at the CSIRO, Australia, the Infolabs21, Lancaster University, U.K., Computer Science and Engineering, The University of New South Wales, Australia, School of IT, University of Sydney, Australia, LIP6, Sorbonne University, France and Computer Science and Engineering, The University of Oxford, U.K. He is also a senior member of IEEE. In 2020, he received the "2020 IEEE ComSoc Outstanding Young Researcher" award by IEEE ComSoc EMEA. In 2021, he was ranked among the World's Top 2% Scientists (2020) in the List prepared by Elsevier BV, Stanford University, USA. Also, he was awarded an Irish Research Council (IRC) Research Ally Prize as part of the IRC Researcher of the Year 2021 awards for the positive impact he has made as a supervisor. Dr. Liyanage's research interests are 5G/6G, SDN, IoT, Blockchain, MEC, mobile, and virtual network security. More info: www.madhusanka.com

Mika Ylianttila (M. Sc, Dr.Sc, eMBA) is a full-time associate professor (tenure track) at the Centre for Wireless Communications - Networks and Systems research unit, at the Faculty of Information Technology and Electrical Engineering (ITEE), University of Oulu, Finland. He is the director of Communications Engineering Doctoral Degree Program and he leads NSOFT (Network security and softwarization) research group which studies and develops secure, scalable and resource-efficient techniques for 5G and beyond 5G and IoT systems. He has co-authored more than 200 international peer-reviewed articles. He is a Senior Member of IEEE and associate editor in IEEE Transactions on Information Forensics and Security.