IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 18 June 2024, accepted 14 August 2024, date of publication 22 August 2024, date of current version 2 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3447785

RESEARCH ARTICLE

Enhancing Privacy and Integrity in Computing Services Provisioning Using Blockchain and Zk-SNARKs

ALBERTO BALLESTEROS-RODRÍGUEZ[1] **(Member, IEEE), SALVADOR SÁNCHEZ-ALONSO**[2], **AND MIGUEL-ÁNGEL SICILIA-URBÁN**[1]
[1]Intelligence, Innovation, Internet and Information (I4) Group, Department of Computer Science, University of Alcalá, 28801 Alcalá de Henares, Spain
[2]Department of Computer Science and Statistics, Rey Juan Carlos University, 28933 Móstoles, Spain

Corresponding author: Alberto Ballesteros-Rodríguez (alberto.ballesterosr@uah.es)

ABSTRACT The widespread integration of on-demand services founded on proprietary algorithms into various software applications has ushered into a new era of advanced service capabilities. However, using these services entails disclosing information by the customer, not only during the payment process but also when using the service, where certain personal information must be shared to obtain a more personalized service. This practice potentially exposes users to increased security risks in case of data security breaches. In this paper, we introduce a novel framework aimed at enhancing client privacy and ensuring service integrity within the context of computing services that rely on proprietary algorithms. A blockchain-based approach is proposed to enhance user privacy throughout service provision, encompassing both the payment process and the verification of the provided service. Our proposal leverages properties of distributed ledger networks to improve user privacy during payment transactions and incorporates a verification system using zero-knowledge proofs on blockchain to validate the integrity of the contracted service. Finally, we analyze the privacy, overhead, and performance aspects of the framework, employing custom proprietary algorithms. We illustrate this through examples of Convolutional Neural Networks with multiple layers, undisclosed to the client. This emphasizes the potential benefits of its applicability for both service providers and clients.

INDEX TERMS Blockchain, proprietary algorithms, service verification, privacy, zk-SNARKs.

I. INTRODUCTION

With the continued proliferation of cloud-based services, the Software as a Service (SaaS) model has proven to be highly successful in providing a diverse array of solutions to both individual users and companies. Furthermore, as organizations increasingly adopt SaaS solutions, providers are expanding their services using not only open-source algorithms but also proprietary ones. This expansion aims to offer a greater variety of options to optimize and differentiate the services provided in the present competitive SaaS market. Given the propagation of such services, subscription-based models are becoming increasingly prevalent as an approach to ensuring consistent revenue streams, ongoing service enhancements, and accessible, cost-effective software solutions.

In this context, it is important to acknowledge that the development of proprietary algorithms, especially in the field of artificial intelligence (AI), often represents a significant investment in terms of research, resources, and expertise. Advanced algorithms are at the core of many SaaS solutions, and their creation requires a substantial financial commitment. To recoup the investment and continue to fund ongoing research and development, subscription-based models have become an essential approach. These models not only ensure the sustainability of SaaS solutions but also support the continuous refinement and expansion of their cutting-edge technologies, which are crucial in providing the diverse and valuable services that users and organizations demand.

The associate editor coordinating the review of this manuscript and approving it for publication was Yan Huo.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

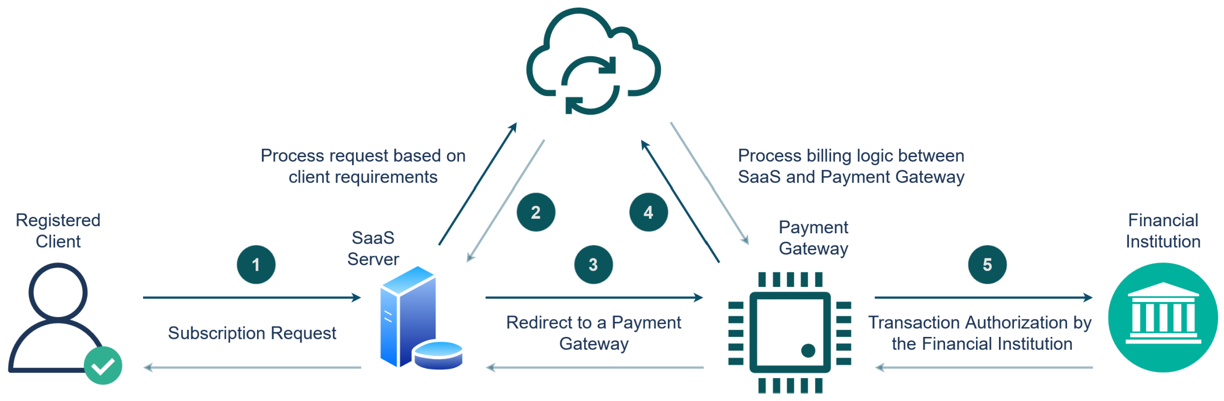

Process request based on client requirements
Process billing logic between SaaS and Payment Gateway
Registered Client
1
Subscription Request
SaaS Server
2
3
Redirect to a Payment Gateway
4
Payment Gateway
5
Transaction Authorization by the Financial Institution
Financial Institution

FIGURE 1. **Simplified Typical Subscription Billing Model in SaaS Services.**

To access these provisions, cloud-based services require user registration on their platform to access all the available solutions. Subsequently, as illustrated in Fig. 1, to enroll in a subscription-based solution, the client must provide a payment method, such as a credit card. This enables the service provider to issue an initial charge, which will be refunded once the functionality of the chosen payment method has been verified. Through this procedure, it becomes possible to connect the client's identity, who initiated the payment, with the provided service. Also, this approach entails significant limitations in terms of privacy, anonymity, and confidentiality. On one hand, payments cannot be anonymous as they are linked to the client's identity within the service provider. At present, most service providers only allow payments associated with the registered client's identity, which means that the client's information is directly tied to the use of a specific service. Moreover, this can lead to privacy and security breaches, as payment systems and user databases become potential points of compromise for sensitive data. On the other hand, occasionally, the client is not an individual user but rather a company acting as a customer. Consequently, the company lacks the capability to conceal its identity from the provider when consuming a service and interacting with it, raising notable confidentiality concerns.

Moreover, concerning the payment aspect, it is entirely plausible that the customer may prefer making the payment only once the service has been fully provided. Currently, this is not the case, as payments are made prior to accessing the service. This becomes even more relevant in the context of proprietary Machine Learning (ML) models, where users are unaware of whether the generated computations come from a specific model or another. Considering the current trend of engaging cloud-based services oriented towards achieving outcomes derived from their inputs, it is noticeable that the field of ML is the most widely exploited. Since legal agreements bind users and companies, the need for technical verification becomes crucial, given the prevalent trend of ML cloud-based services. Nevertheless, in the case of services based on non-public proprietary algorithms, also known as black box algorithms, the user cannot be fully confident that the rendered service aligns with the contracted one. Black box algorithms are computational models or algorithms that function as opaque systems, implying that their internal computations and decision-making processes are not transparent or understandable to external observers.

These proprietary algorithms, developed internally or obtained through strategic acquisitions, are designed to enhance the functionality, performance, and personalization of the services offered. By leveraging the power of data analytics and artificial intelligence, these algorithms can optimize processes, predict trends, and offer personalized experiences to users. The problem that arises from these premises is the lack of privacy surrounding client data when using these services, whose internal functioning is undisclosed. This extends beyond merely data tied to their identity to also include the information customers are required to provide for accessing personalized services.

To address the challenges mentioned above, it is crucial to decouple the client's identity from the service usage, both in the payment process and in terms of the data used for service provision. This should ensure that the service provider remains unaware of the client's identity, thus preventing the client from being linked to their information, thereby enhancing privacy, confidentiality, and anonymity. It should also ensure that the payment is released once the results from the provided service have been verified. Therefore, achieving a scenario where customers can make privacy-preserving payments that can be released to the vendor only once the service has been undoubtedly completed in a secure environment is complex and calls for innovative solutions. Additionally, it is important to consider that such services often operate on subscription models, so payments should only occur after verifying all expected computations. The application of decentralized systems such as blockchain technology, in combination with cryptographic mechanisms like zero-knowledge proofs, provides an approach to resolve the challenges.

Zero-knowledge proof (ZKP) can be defined as an advanced cryptographic technique that allows the demonstration of the possession of certain information without revealing the information itself. This is particularly valuable in blockchain environments where privacy and confidentiality are paramount, requiring the avoidance of disclosing sensitive information to third parties while still being verifiable. However, ZKP in blockchain presents significant challenges, as the computational complexity associated with its implementation directly impacts the network's efficiency and speed, leading to increased latency and higher transaction fees. Furthermore, ZKP deployment might hinder scalability, as the growing number of transactions might overload nodes, reducing their processing capacity. To address these problems, zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge), a type of zero-knowledge proof that possesses properties including reduced proof size, easy verification, and the elimination of the requirement for interactive communication, is utilized. In the context of this research, zk-SNARKs are employed to authenticate the computations conducted by the service provider, without disclosing critical information about the proprietary algorithms employed in the service provision. Because of their intrinsic characteristics, it is feasible to verify the service rendered at different time periods to assess the execution of the subscription-based algorithms. This type of zero-knowledge proof is considered a suitable solution for enabling recurring transactions commonly required to verify different responses derived from subscription-based services. Indeed, zk-SNARKs have been shown to align seamlessly with the inherent performance of blockchain [1].

Leveraging a decentralized system such as blockchain allows users to make payments through native tokens, as well as stablecoins—cryptocurrencies pegged to stable assets—thus enhancing payment process anonymity and reliability. The combination of blockchain and zk-SNARKs provides clients with a high level of assurance and verifiability. Meanwhile, service suppliers can offer a high level of trust by ensuring the privacy and confidentiality of sensitive customer data without incurring additional computational costs.

In this paper, we introduce an innovative model designed to tackle the aforementioned challenges, specifically by leveraging blockchain technology and ZKP. The primary objective of this research is to establish a framework that enhances client privacy preservation throughout service provisioning, covering both the payment process and the service provision, the latter being cryptographically verified when it depends on proprietary algorithms. The proposed framework guarantees that payment is only released after verifying the successful provision of the contracted service. We efficiently utilize zk-SNARKs for service verification by deploying a single smart contract on a general-purpose blockchain instead of creating one for each individual client request. This approach minimizes the overhead associated with contract execution and enhances the overall scalability and performance of the entire proposal. The effectiveness of the framework is evaluated through the analysis of deployment cost, proof sizes, proof generation times, and the cost of each interaction to be performed by each participant.

The rest of this paper is structured as follows. In "Related Work," literature related to the proposed model is reviewed. "Model Overview" presents both the architecture and setup of the proposed framework, as well as its operation. In "Results and Discussion" evaluations conducted to obtain metrics are provided, along with the results derived. Finally, the article offers final conclusions and considerations for future works in "Conclusions and Outlook."

II. RELATED WORK

This section provides a literature review concerning previous works where blockchain serves as the foundation for developing private payment systems across different types of applications. Furthermore, the state of the art is explored regarding the implementation of zero-knowledge proofs for service providers seeking to safeguard the confidentiality of their algorithms.

A. PRIVACY PRESERVING BLOCKCHAIN-BASED PAYMENT SYSTEMS

Blockchain technology has brought significant advancements to the field of electronic payment systems, enabling secure, decentralized, and transparent transactions. However, in this review, the intrinsic properties of blockchain are set aside, and protocols built upon this technology that enable privacy-preserving payment systems are explored. This is why blockchain platforms that prioritize enhancing the privacy and anonymity of native transactions are excluded from this review.

First, a survey from Andola et al. [2] focuses on the anonymity-provisioning mechanisms in blockchain-based e-cash protocols. This article emphasizes the importance of preserving the privacy of identities and transactions in public blockchain frameworks. The paper identifies the limitations of pseudonymity and proposes protocols for achieving stronger anonymity. Additionally, it discusses threats and attacks that aim to deanonymize e-cash protocols. Some of the techniques assessed in the work by Andola et al. for improving privacy in blockchain payment systems include the Non-Interactive Zero-Knowledge proofs from Blum et al. [3].

When it comes to blockchain-based payment systems, the focus on privacy and anonymity of users is the main objective. Addressing privacy concerns in decentralized payment systems, Lin et al. [4] present a Decentralized Conditional Anonymous Payment (DCAP) system. The paper introduces a novel definition of DCAP and outlines the security requirements. The system design incorporates a Conditional Anonymous Payment (CAP) scheme, based on a proposed signature of knowledge, which assumes concepts from the cryptography of zero-knowledge proofs. Comparisons with other privacy-focused systems demonstrate the utility and performance of the proposal. Then, building upon this article as a starting point, Hatefi et al. [5] propose a blockchain-based electronic payment scheme that prioritizes the security and privacy of honest users. The scheme employs fair Blind Digital Signatures, secret sharing, and pseudonyms to achieve anonymity. Notably, this scheme allows tracking and punishment of malicious users without relying on a trusted server, including properties such as traceability, conditional privacy, and revocation.

As has been evidenced, the research reviewed up to this point shows a diverse range of solutions leveraging zero-knowledge proofs to enhance the anonymity and privacy protection of blockchain-based payment systems. Subsequently, we delve into a comprehensive analysis of previous studies centered on the application of zero-knowledge proofs in blockchain. Zero-knowledge proofs have emerged as a fundamental cryptographic technique with significant applications in the field of blockchain technology. These mathematical constructions are considered highly valuable in the context of blockchain, where privacy, security, and efficiency are critical concerns.

As an operational payment system on blockchain that employs zero-knowledge proofs, Boo et al. [6] introduces a framework designed for resource-limited devices to support multiple anonymous payments through a smart contract-based ZKP protocol. The proposal addresses challenges associated with minimizing computational overhead and achieving full anonymity. Moreover, it demands a significantly lower block processing fee compared to a naive ZKP-based scheme. Also using zk-SNARKs, Guo et al. [7] proposes a zk-SNARKs-based anonymous payment channel, which supports an unlimited number of off-chain payments while preserving the privacy of participants. By employing zero-knowledge proofs and commitment schemes, the proposed scheme achieves relational anonymity and amount privacy for both on-chain and off-chain transactions. The performance evaluation demonstrates the effectiveness of the method compared to similar schemes.

B. APPLICABILITY OF BLOCKCHAIN-BASED PAYMENT SOLUTIONS

Apart from payment systems that prioritize anonymization, other works explore blockchain-based payment solutions that operate effectively in resource-limited environments, as well as those specifically designed to address constraints associated with mobile devices.

Focusing on remote, rural regions with intermittent network connectivity, Hu et al. [8] propose a blockchain-based digital payment scheme. The system leverages the distributed verification guarantees of the blockchain for transaction verification and uses smart contracts for secure service management. It utilizes probabilistic modeling to ensure robust operation over unreliable networks. The study validates the feasibility of the proposed system through practical implementations and simulations. In relation to the above and given the emergence of these payment systems and the current rise of financial technology applications enabling easy payment transactions, the work of Xu et al. [9] addresses the challenges of enabling cryptocurrency payments on mobile devices, especially in developing countries with limited financial infrastructure. It proposes two schemes for cryptocurrency mobile payments, one involving a centralized bank and the other being fully decentralized.

Building on the aforementioned, recent research [10], [11] explores payment schemes for Electric Vehicles (EVs), including blockchain-based systems for Vehicular Ad-hoc Networks (VANETs) and privacy-preserving energy trading schemes. These systems prioritize customer privacy by concealing charging locations while also ensuring the security and privacy of vehicle accounts.

C. APPLICATION OF ZERO-KNOWLEDGE PROOFS TO PROPRIETARY ALGORITHMS

Regarding the integration of ZKP to ensure the trustworthiness and privacy of systems founded on proprietary algorithms, we will focus on the application of ZKP to machine learning models that, for business reasons, should remain private and cannot be disclosed to the general public in most cases. Additionally, this focus is driven by the significant surge in demand for those services due to their widespread use, making the application of Zero-Knowledge Machine Learning (ZKML) relevant.

There are studies that employ ZKML in various processes related to machine learning models, and following that, we refer to those that implement the use of ZKP in both inference and training processes. Lee et al. [12] present a verifiable convolutional neural network (vCNN) framework based on the zero-knowledge Succinct Non-Interactive Argument of Knowledge construction, aiming to allow clients to verify the correctness of Artificial Intelligence (AI) inference services without revealing sensitive weight values or user input data. This efficient vCNN approach significantly accelerates the proving performance and achieves formal security guarantees. Additionally, Liu et al. [13] introduce a ZKP scheme for convolutional neural networks (zkCNN), enabling the validation of machine learning predictions' integrity without compromising model information. ZkCNN achieves outstanding efficiency demonstrating its applicability in verifying CNN predictions for large models like VGG16 [14]. Ultimately, Sun and Zhang [15] address the challenge of ensuring verifiable computations in deep learning training by proposing zero-knowledge Deep Learning (zkDL). The proposed scheme ensures data and model parameter privacy while reducing both proving time and proof sizes significantly.

D. MOTIVATION AND CONTRIBUTION

The previous literature review shows that there is a growing relationship between blockchain-based payment systems and the utilization of zero-knowledge proofs. Nonetheless, these approaches primarily include theoretical studies that, although applicable, are not associated with any specific service, as they focus on enhancing privacy through different applications or protocols. Whereas some studies consider the potential implementation of such proofs in payment systems, it is not the prevailing approach, and they often neglect associating payments with particular services, remaining exclusively within the domain of theoretical investigations.

Furthermore, if we incorporate ZKML into this analysis, it becomes apparent that the application of this novel method does not extend beyond evaluating different validation components of models, such as training or inference. As a result, integration with payment systems or blockchain-based applications has not been explored. None of the analyzed studies contemplate the combination of zero-knowledge proofs with blockchain technology for payments to be released automatically after the cryptographic verification of service provision. Consequently, we observe that an unmet need arises from the widespread use of third-party services, particularly proprietary machine learning models, where the purpose is to verify the computed results prior to payment release to the service provider, while keeping the client's identity separate from the service usage.

The public release and the expanding use of artificial intelligence models, such as Gemini [16] or OpenAI GPT [17], have highlighted the advantages that proprietary models can offer, such as real-time data or customized dataset access. These models are backed by significant infrastructure investment, often exclusive to large corporations, making replication in open-source models challenging. Additionally, concerns arise when adapting datasets, as exporting the entire model can potentially expose users' personal information. This is where our proposal gains value, as the use of zero-knowledge proofs protects the model's privacy, providing users with confidence in the authenticity of the results obtained.

Based on the aforementioned, when employing black box or proprietary algorithms, users deal with insecurity concerning whether the outcomes are generated using the correct version of the selected algorithm. However, even if users have employed a specific version of the model, they are not entirely confident that the generated responses are based on the selected model, despite the existence of a contractual and legal agreement between the client and the service provider. In addition, subscription models are commonly used for such services, allowing for automatic plan upgrades if a pre-established threshold is exceeded. Payment is typically scheduled based on batches of inferences, time intervals, or token limits, especially for language models.

This work introduces a privacy-preserving framework that allows verification of the correspondence between the received sets of inferences and the expected ones, even when received at different timestamps or corresponding to different inference batches, as long as it is specified before the client engages with the service. For the payment process, we propose that the user makes a deposit through a smart contract, thus avoiding revealing their identity. This smart contract is responsible for releasing the payment once verifying the zero-knowledge proofs associated with the service based on a specific proprietary algorithm. The proofs, directly associated with the inferences, are provided by the vendor, in such a way that the contract will only release the deposit if and only if all the required proofs by the client are successfully verified. In this manner, the client is protected against any malicious behavior from the provider, while the provider, acting honestly, can be confident of receiving the payment. It should be noted that the payment can be subject to a set of inferences, a time interval, or any alternative criteria determined by the vendor.

III. PRELIMINARIES

A. BLOCKCHAIN & SMART CONTRACTS

Blockchain is a distributed ledger technology that underlies cryptocurrencies like Bitcoin [18] and provides a secure, transparent, and tamper-resistant way to record transactions. It operates as a decentralized and distributed ledger maintained by multiple nodes in a Peer-to-Peer (P2P) network through a consensus protocol. A blockchain consists of a chain of blocks, where each block contains validated transactions and is linked to the previous block using cryptographic techniques. This chaining creates an immutable record of transactions, which are verified and validated. Once a block is added to the chain, it becomes confirmed and cannot be altered, ensuring the security and integrity of the data.

The decentralized nature of blockchain promotes trust in the system, as the network is maintained by a distributed set of nodes rather than relying on a central authority. Its transparency enables participants to access stored information, fostering openness and accountability. Moreover, the implementation of cryptographic techniques ensures data protection and enhances user anonymity.

A significant aspect of blockchain technologies is the adoption of smart contracts, which are self-executing software programs with predefined conditions and actions. These contracts are stored on the blockchain and replicated across all network nodes, enabling decentralized execution and tamper-proof data storage. Smart contracts [19] were conceptualized and have attracted significant interest since their introduction. Widely recognized with the development of Ethereum [20], smart contracts represent an innovative approach to digital agreements.

Smart contracts are immutable and resilient against unauthorized alterations. Blockchain characteristics guarantee that smart contracts operate in a trustless environment, removing the need for intermediaries or central authorities. In multiple industries, smart contracts offer numerous advantages, enabling transparent, efficient, and secure execution of agreements while minimizing the risk of fraud or manipulation. By automating contract terms, transaction costs and processing times are reduced, resulting in an overall improvement in efficiency.

B. ZERO-KNOWLEDGE PROOFS: zk-SNARKs

1) ZERO-KNOWLEDGE PROOFS

A Zero-Knowledge Proof (ZKP) [21], [22] is a cryptographic protocol in which the Prover works to convince the Verifier that a specific statement is true or that certain information is possessed, abstaining from disclosing the statement itself to the Verifier. The main idea of this protocol is to prevent the

Verifier from acquiring any knowledge or insight regarding the statement during the process. To achieve this, three fundamental properties must be satisfied by any ZKP.

- *Completeness*: If a true input statement is sent by an honest Prover, the honest Verifier, adhering correctly to the protocol, will be convinced of the truthfulness of the statement.
- *Soundness*: If the Prover attempts to deceive the Verifier by submitting a false statement, the Verifier will be convinced of the statement's truth with negligible probability.
- *Zero-knowledge(ness)*: Regardless of whether the statement sent by the Prover is true or false, the Verifier learns nothing beyond the mere verification of the statement's truth.

In order to prevent the interactive communication between the Prover and the Verifier, resulting in enhanced efficiency and reduced computational costs, the work by [3] and [23] addressed the possibility of removing the interaction between the Prover and the Verifier using a short random string previously shared, naming this new variant as Non-Interactive Zero-Knowledge Proofs (NIZKPs). NIZKPs achieve the same objective but with the additional advantage of requiring a single round of communication, avoiding back-and-forth interactions between the Prover and the Verifier. This one-round interaction significantly reduces the computational overhead and enhances the efficiency of the proof process. NIZKPs prove particularly beneficial in scenarios where reducing the frequency of transactions and minimizing interactions between involved parties are critical, aligning seamlessly with the requirements of blockchain contexts.

2) zk-SNARKs

A more sophisticated method that inherits the definition of ZKP emerged with the use of Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs). [24] came up with zk-SNARKs from the protocol described by [25]. This advanced cryptographic technique enables highly efficient and compact proofs, making it particularly valuable in scenarios with limited computational resources and storage constraints, such as blockchain. By employing zk-SNARK proofs, it becomes possible to use ZKP without requiring interactive communications, leading to even greater computational efficiency and improved privacy protection.

Zk-SNARKs are characterized by their reduced proof size, allowing the Prover to convincingly demonstrate the validity of a statement to the Verifier using a single concise message. This succinctness significantly reduces the communication overhead, making this construction suitable for applications where bandwidth or computational resources are limited. Furthermore, zk-SNARKs have a constant verification cost, meaning that the computational resources required to verify a statement remain constant regardless of the statement's complexity or size. The non-interactive nature of zk-SNARKs allows the Prover to undergo verification without requiring additional interactions. Because of these properties, zk-SNARKs are used in blockchain networks. Additionally, the Verifier's inability to infer any details from the proof ensures privacy, even within a public blockchain ledger, highlighting zk-SNARKs as a well-suited verification scheme for such networks.

C. ZKML: ZERO-KNOWLEDGE MACHINE LEARNING

In a scenario where content generated by artificial intelligence is becoming more prevalent, it is occasionally required to verify the source of a specific content to determine whether it has been generated by one model or another. The integration of zero-knowledge cryptography presents a highly plausible alternative for assessing the source of such content. The purpose is not only to determine whether the content has been generated from one model or another, but also to assert which version of the same model has generated the content. Furthermore, the majority of these models are proprietary, implying that their development depends on their own implementation to provide a variety of services.

Zero-Knowledge Machine Learning (ZKML) allows not only the concealment of parts of the model's input but also parts of the model itself, as shown in the works from [13] and [26]. This has diverse applications depending on the use case. For instance, applying a proprietary algorithm to a client's data could enable users to determine if their data has been processed using a specific model or not. Besides, this implies that the use of ZKML involves generating proofs for the model's inference process rather than for its training phase.

Although zk-SNARKs are a promising technique for ZKML [12], [27], it should be noted that their implementation and scalability for large-scale models still poses significant computational cost challenges due to arithmetic circuits transformations and the associated proof generation process.

The utilization of verifiable proofs, specifically zk-SNARKs, provides a robust mechanism to demonstrate the correctness of computations. Within the domain of machine learning, these proofs assume a critical role in verifying the correctness of model inferences or validating that a specific model produced a particular output based on a given input. This property is particularly valuable in scenarios where trust and privacy are paramount, as the Prover can demonstrate the validity of a computation without revealing sensitive details of the inputs or the model. By leveraging zero-knowledge techniques, these proofs assure that the Prover has the required information to verify the computation's correctness without disclosing any additional information to the Verifier.

IV. MODEL OVERVIEW

A. PRELIMINARY ASSUMPTIONS

In this section, the preliminary assumptions are introduced, which serve as the foundation for understanding our approach. These assumptions establish the conceptual framework needed for a comprehensive grasp of the methodologies and insights presented. The decisions made at this stage have a significant impact on the interpretation and validity of the results.

The outset of our research is defined by a client who intends to engage a service whose implementation is unknown, albeit being aware that it is offered by a specific vendor. Indeed, the service has specific features and outcomes, which is why the client considers it as a potential choice to engage with. The following includes assumptions and trust requirements.

- The client is aware of a service provider that claims to offer a specific service based on a proprietary algorithm.
- There needs to be public evidence of the service being rendered and payment being successfully processed.
- The client trusts the service provider because it is a legally recognized entity exposed to liabilities.
- The service contracting process, as well as the exchange of data by the client, is assumed to allow the service provider to effectively execute the contracted algorithm.

a: PARTICIPATING ENTITIES

In the proposed approach, three entities are expected to interact within the framework: clients, a single service provider, and the smart contract.

- *Clients (U_c):* We refer to clients in the plural form since the framework allows multiple users to make payments and request validations from the contracted algorithm.
- *Service Provider (U_s):* We specifically refer to a single service provider, as the smart contract is designed to verify zero-knowledge proofs for an exclusive proprietary algorithm.
- *Smart Contract (SC):* The proposed system includes two smart contracts. The first one incorporates the verification logic for a particular proprietary algorithm using zk-SNARK proofs, whereas the second inherits this logic to enable verification combined with the payment mechanism, allowing users to pay for the service. Hence, clients and the service provider will only need to interact through this latter smart contract, which is the one we refer to in the following explanations.

b: PAYMENTS VIA BLOCKCHAIN

As the primary technology to carry out our proposal, we propose the usage of blockchain, leveraging its privacy and security features to ensure the confidentiality of client identities. Specifically, a blockchain capable of supporting the execution of arbitrary code through smart contracts. In this instance, we opt for an Ethereum-based network due to its widespread adoption as a smart contract platform. Blockchain technology facilitates payments between services or individuals, with multiple payment system models supported by smart contracts as intermediaries.

The client U_c making the payment, i.e., the deposit D through the smart contract SC, does not need to disclose any identity-related information, as only their blockchain address (addr_c) is used, represented as $\text{deposit}(\text{addr}_{U_c}, D)$.

Our research proposes carrying out payments through an intermediary responsible for coordinating clients and a service provider. This intermediary takes the form of a smart contract that receives deposits from clients and releases payments to the service provider as long as the verification through zero-knowledge proofs of the provided service is completed. Formally, for each client C_i, the deposit D_i is made to the smart contract SC, represented as: $f_{\text{deposit}}(\text{addr}_{U_{c_i}}, D_i)$. The smart contract then verifies the service provided by the service provider U_s using zero-knowledge proofs: $f_{\text{ZKPverify}}(\text{service}_{U_s})$. Upon successful verification, the smart contract releases the payment to the service provider: $f_{\text{releasePayment}}(\text{addr}_{U_s}, D_i)$. Thus, the smart contract not only serves as a mere intermediary for executing payments but also functions as a service verifier. To evaluate the performance of the framework, an Ethereum-based network is employed to deploy the smart contract. Although interactions within the contract are discussed, the deployment process is not illustrated in this work.

c: PROPRIETARY ALGORITHM

In order to operate as a verifier, the smart contract must be linked to a specific service. In this context, we can define a service as the task undertaken by a company for a client based on a black box algorithm, which is developed by the same company and whose internal implementation cannot be disclosed or made public. To achieve this connection between the smart contract and the proprietary algorithm without disclosing sensitive information about it, ZKML is employed. Mathematically, $f_\theta(X_{U_c}) \rightarrow y$ represent the proprietary algorithm, where θ denotes the internal parameters, X_{U_c} the user input data, and y the outputs.

Although this study primarily focuses on verifying results produced by undisclosed proprietary algorithms, the proposed framework undergoes testing and analysis using machine learning models. Specifically, different instances of Convolutional Neural Networks (CNNs) serve as examples of such algorithms. These CNNs are based on models implemented using the Keras [28] library, trained, and evaluated using the MNIST dataset [29] —a widely-used collection of handwritten digit images. Consequently, the outputs produced by these CNNs must align with the model contracted by the client, thus requiring the verification process by the smart contract through zero-knowledge proofs associated with the inferences and CNNs provided by the service provider.

For this research, the following CNNs have been implemented:

- *Custom CNN 1:* Neural network designed to process images with dimensions of 28×28 pixels and a single channel as input. It applies a convolutional layer with one filter and a kernel size of 3×3 to extract features from the images. Then, it applies a rectified linear unit activation function to introduce non-linearity.

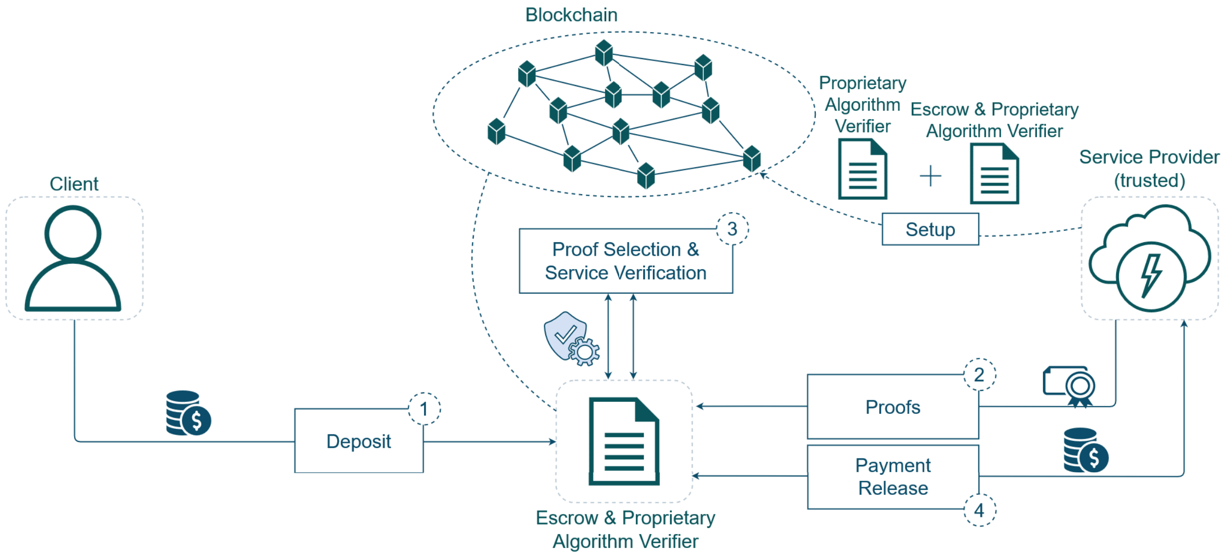

Blockchain
Proprietary Algorithm Verifier
Escrow & Proprietary Algorithm Verifier
Service Provider (trusted)
Client
Setup
Proof Selection & Service Verification
3
Proofs
2
Deposit
1
Payment Release
4
Escrow & Proprietary Algorithm Verifier

FIGURE 2. **Framework processes and workflow. The processes of hiring a service, sending data, receiving results, and other potential procedures related to the service provision itself are omitted in this diagram. Numbers represent the sequence in which processes and actions must be executed.**

The output is then flattened into a one-dimensional vector and passed through a fully connected layer with 10 neurons and no activation function. Finally, a softmax layer is applied to generate probability scores for classification. This CNN is trained and evaluated using the MNIST dataset.

- *Custom CNN 2:* Neural network created for analyzing 28 × 28 pixel images with a single channel as input. It first normalizes the input values, then applies a series of convolutional layers to extract features from the images. After each convolutional layer, it performs a non-linear transformation on the output. The network also includes average pooling layers to reduce the dimensions of the feature maps. Finally, it flattens the output and passes it through a fully connected layer with 10 neurons and no activation function, followed by a softmax layer to generate probability scores for classification. Although this neural network is complex in terms of structure, it has fewer parameters due to the presence of multiple convolutional and dimensionality reduction layers. This CNN is trained and evaluated using the MNIST dataset.
- *Custom CNN 3*: Neural network designed to take a dataset with three features as input, process it through a hidden layer with two neurons using an activation function, and then generate a single output. This CNN does not involve image processing and instead relies on a dataset with three randomly generated features for its input. Compared to the previous ones, this CNN is the simplest of all evaluated in this research.

d: ALGORITHM VERIFICATION

The role of the verifier, as explained, is embedded in the smart contract deployed on the blockchain. The verification process occurs via blockchain, ensuring both transparency and accountability. The use of zero-knowledge proofs (π), generated for the statement that $f_\theta(X_{U_c}) \rightarrow y$, for verifying proprietary algorithms ensures that critical components of the model (θ) or sensitive information about it remain undisclosed. Simultaneously, it enables verification that the computations have originated from a specific algorithm (f_θ). The smart contract SC is linked to f_θ such that $f_{\text{ZKPverify}}(\pi)$. Moreover, this guarantees that no client information is disclosed by the service provider. Taking into account the properties described above, zk-SNARKs are used for algorithm verification.

Thus, the idea is to assert that a result was derived from the model's execution without revealing the details of the inference process. This is accomplished due to the fact that the smart contract is configured to verify computations from a specific algorithm, such as one of the described custom CNN models. In the following sections, we will explain how the contract is developed to be able to process proofs of the CNNs built for the evaluation of the framework.

e: SCOPE AND DELIMITATIONS

We acknowledge the limitations of our proposal with respect to the smart contract, such as the smart contract's capability to only process zero-knowledge proofs concerning a single proprietary algorithm. Moreover, the proprietary algorithm to be verified by the smart contract must be defined in advance, as there exists a set of procedures associated with the processing of zero-knowledge proofs that influence the contract implementation. Using a single algorithm reduces deployment costs, enables modular implementation, facilitates external audits of the proprietary algorithm and zk-SNARK scheme used, and, in case of an error, only one contract would need to be redeployed.

Taking into consideration the limitations, our research aims not only to enhance users' privacy while conducting payments through service engagements but also to ensure that the rendered service corresponds to the contracted one, which is backed by technical evidence beyond the legal contractual terms agreed between the client and the trusted vendor.

Furthermore, as shown in Fig. 2, the proposed framework demands a minimal number of interactions for both the client and the provider, removing the requirement of direct interaction between the two parties. It should be noted that this research focuses on the verification and payment aspects of specific service provisions, excluding any direct communication between the client and the provider. Consequently, this omits the provider's result communication to the client and the service contracting process between the two entities.

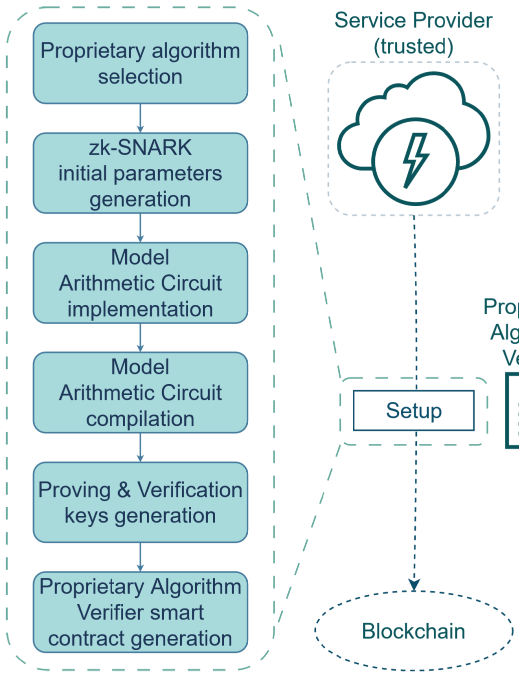

Service Provider (trusted)
Proprietary algorithm selection
zk-SNARK initial parameters generation
Model Arithmetic Circuit implementation
Model Arithmetic Circuit compilation
Proving & Verification keys generation
Proprietary Algorithm Verifier smart contract generation
Setup
Proprietary Algorithm Verifier
Blockchain

FIGURE 3. **Steps involved in generating the verifier smart contract associated with a proprietary machine learning algorithm and a zk-SNARK construction.**

Next section covers the essential aspects of the implementation of the mechanism for verifying the computations produced by a proprietary algorithm through a smart contract that validates zero-knowledge proofs associated with a proprietary model. At first, given that this research is divided into two clearly delineated processes, in the next section, we present the approach of each one separately. The first part is related to the verification of the provided service, whereas the second focuses on the payment mechanism.

We suggest the deployment of the smarts contracts on a public blockchain for higher transparency and accountability, although the framework also supports the implementation of private blockchains. Furthermore, the interactions within the smart contract will be described, along with the logic behind them and the responsible parties for their execution on each step.

B. ALGORITHM VERIFICATION MECHANISM

This section covers all aspects related to the generation of a verifier smart contract for a proprietary algorithm. The visual representation of the sequential steps to be carried out by the service provider, outlining the complete procedure in a general context, can be observed in Fig. 3.

First, and as the foundation of the entire solution, the algorithm that requires verification of the results generated by the service provider must be selected. We do not delve into the criteria for selecting one algorithm over another; it must simply be understood that, once chosen, it cannot be modified or updated, as the verification process will be tied to the model. In this research, the previously defined custom CNN models are opted for.

As the starting point of the proving system, to set up the initial security parameters of the zk-SNARK scheme, a collaborative protocol known as the Ceremony Process is conducted. This process enhances the reliability and security of the scheme by preventing any single entity from controlling all the essential information required for its operation, including the generation of false proofs as valid. In this proposal, where the responsibility of generating both the proving system and the proofs lies with the service provider, to avoid performing the ceremony process, the vendor also has the option to download a set of random values and secrets generated previously from a ceremony process known as Powers of Tau, which is widely employed in cryptographic protocols [30]. This ceremony process results in the generation of the global parameters Z from a security parameter λ, such that $f_{\text{setup}}(\lambda) \rightarrow Z$. These global parameters Z are described as three bilinear groups $\mathbb{G}_2$, $\mathbb{G}_1$, and $\mathbb{G}_T$, each of prime order p, along with their respective generators g_1 and g_2, as well as an efficiently computable bilinear pairing $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.

As mentioned in previous sections, the algorithm execution verification is conducted through zero-knowledge proofs, specifically employing zk-SNARKs. However, for zk-SNARKs to effectively demonstrate computational statements or assertions, it is essential to ensure that these statements are formatted correctly. Particularly, zk-SNARKs require that the computational problem to be demonstrated by modeling it through the utilization of an arithmetic circuit. An arithmetic circuit refers to a mathematical representation of a series of arithmetic operations used to model and express a statement that is intended to be demonstrated without revealing any underlying information. The circuit can be represented as follows.

$$\mathcal{C}(X_{U_c}, \theta) \rightarrow y = \begin{cases} \text{true} & \text{if } y \leftarrow f_\theta(X_{U_c}) \\ \text{false} & \text{if } y \nleftarrow f_{\theta'}(X'_{U_c}) \end{cases}$$

following the notation presented earlier. Due to the complexity of the custom CNN models and the extent of the resulting arithmetic circuits, a representation of these is not provided. However, the code for each model can be consulted in its respective repository [31].

Once the proprietary algorithm is encoded as an arithmetic circuit, the specific construction of the zk-SNARK proof to be employed must be selected. There are several existing zk-SNARK constructions, nevertheless, as our proposal is designed for a distributed environment such as blockchain, efficiency and performance must be prioritized. Furthermore, the consideration of proof sizes should be taken into account, as this could also have economic implications for the usage of the developed framework. The following three zk-SNARK constructions are considered: Groth16 [32], Plonk [33], and Fflonk [34]. The selection of these constructions is based on their positive performance and efficiency on blockchain architectures [1].

TABLE 1. R1CS circuit information for different Convolutional Neural Networks created as examples for the evaluation of the framework. Wires are variables or intermediate elements employed for calculations within the circuit, whereas constraints are mathematical expressions denoting relationships between variables in the circuit.

Circuit	Wires	Constraints	Private Inputs	Public Inputs	Outputs
Custom CNN 1	371,086	364,883	7,564	0	11
Custom CNN 2	70,524	67,403	3,130	0	11
Custom CNN 3	1,050	1,042	11	0	1

After the arithmetic circuit C is implemented and the ceremony process has been completed, the generation of the circuit's R1CS takes place through circuit compilation. The Rank-1 Constraint System (R1CS) [35] is an algebraic approach to systems of equations and constraints used to describe arithmetic circuits that represent computational problems. The equations characterizing the circuit C are referred to as constraints, conceptualized as the conditions that signals within the circuit must adhere to. Essentially, ZKP enables the demonstration of circuit satisfiability $C(X_{U_c}, \theta) \rightarrow$ true; in other words, this implies the ability to prove knowledge of a group of values θ that fulfill the circuit's conditions, effectively demonstrating an understanding of a solution to the R1CS. This set of values is referred to as the witness, which is used for the proof construction and will be discussed subsequently.

Table 1 is intended to clarify the conceptual understanding of circuit design providing R1CS information for the custom CNN models simulating proprietary algorithms composed of multiple layers, which are utilized for subsequent evaluations of the framework. These circuits present numerous private inputs that generate a series of outputs through thousands of wires and constraints. There are no public inputs in these circuits to prevent the disclosure of any parameters associated with the proprietary algorithms. Furthermore, as shown in Table 2, as the number of parameters increases, resulting in a more complex circuit, there is a corresponding rise in the count of inputs, wires, and constraints in the circuits of Table 1.

The next step involves generating a zero-knowledge key that includes both the proving key and the verification key,

TABLE 2. Layers employed in custom Convolutional Neural Network models to evaluate the framework. The Layer Type indicates the type of layer used. Output Shape displays the dimension of the output data after passing through each layer, helping to understand how the data is transformed throughout the network. The None dimension means that it can be any scalar number, allowing this model to be used to infer on an input of arbitrary length. The Parameter Number specifies the number of trainable parameters in each layer, providing insights into the network's complexity and learning capacity.

Model	Layer Type	Output Shape	Parameters
Custom CNN 1	InputLayer	(None, 28, 28, 1)	0
	Conv2D	(None, 26, 26, 1)	10
	ReLU	(None, 26, 26, 1)	0
	Flatten	(None, 676)	0
	Dense	(None, 10)	6,770
	ArgMax	(None, 10)	0
Custom CNN 2	InputLayer	(None, 28, 28, 1)	0
	Conv2D	(None, 26, 26, 4)	40
	AveragePooling2D	(None, 13, 13, 4)	0
	Conv2D	(None, 11, 11, 8)	296
	AveragePooling2D	(None, 5, 5, 8)	0
	Flatten	(None, 200)	0
	Dense	(None, 10)	2,010
	ArgMax	(None, 10)	0
Custom CNN 3	InputLayer	(None, 3)	0
	Dense	(None, 2)	8
	Dense	(None, 1)	3

$ZK_k = \{P_k, V_k\}$. It should be noted that a zero-knowledge key is tied to a specific arithmetic circuit C, i.e., to a specific algorithm f_θ, as well as to a particular zk-SNARK scheme. Thus, based on the previously defined global parameters Z and the arithmetic circuit C corresponding to a specific custom CNN model and to a zk-SNARK construction, we have $f_{ZK_k}(C, Z) \rightarrow ZK_k$. Anyone who has access to the proving key P_k has the ability to generate a proof π using the witness w, a set of secret parameters that the prover demonstrates knowledge of, such that $f_{\text{proof}}(P_k, w) \rightarrow \pi$. On the other hand, to perform the verification algorithm, the proof π, the verification key V_k, and the output y are used, resulting in $r \in [0, 1]$ through the function $f_{\text{verifyProof}}(V_k, y, \pi)$. The value r serves as an indicator of the validity of the proof π, where $r = 1$ denotes a valid proof, while $r = 0$ signifies an invalid one. As clarification, the verification key is embedded in the verifier smart contract generated at the end of this process.

From this zero-knowledge key, the attached verifier smart contract for a specific proprietary model and a zk-SNARK scheme is generated. Also, considering that the employed zk-SNARK construction may vary, the verifier contract will therefore have a different implementation for each construction, but all adhere to the same basic structure. This structure includes variable declarations that encompass the verification key, as well as a verifyProof function that accepts the proof and its corresponding public signals as parameters. Given the proposed framework, it is important to clarify that the service provider is responsible for generating the verifier contract and the proofs. Consequently, both the proving and verification keys will be held by the service provider.

C. PAYMENT AND PROOF SELECTION MECHANISM

In order to complete the proposed framework, the payment mechanism's logic is integrated with the verifier described

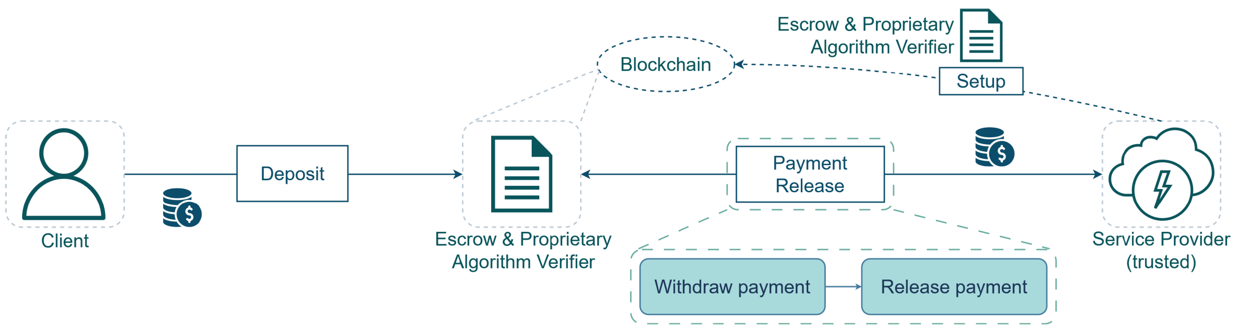

Escrow & Proprietary Algorithm Verifier
Setup
Blockchain
Client
Deposit
Escrow & Proprietary Algorithm Verifier
Payment Release
Service Provider (trusted)
Withdraw payment
Release payment

FIGURE 4. **Steps involved in the payment process. The service provider initiates the payment request, and upon successful verification of the service, the smart contract releases the funds.**

above. Specifically, the function responsible for verifying zero-knowledge proofs from the proprietary algorithm is imported into a new smart contract where all the logic associated with the payment mechanism is implemented.

Regardless of the payment modality, each computation result provided to the client should be subject to verification. However, it may happen that the requested set of inferences is excessively extensive, making it impractical to verify the result of all of them, both in terms of effort and time. That is why a proof selection mechanism is proposed, in which the client selects a minimum number of zero-knowledge proofs required to consider the service provided by the vendor as valid. Based on this quantity, the smart contract uses a pseudo-random algorithm to select the proofs that must be verified to validate the rendered service in order to release the payment. The payment process involves the following steps:

- Initially, the client is required to deposit funds (D_{U_c}) into the smart contract equivalent to the service cost. It falls upon the service provider to determine the amount that the client must pay based on the number of proofs required to verify the supplied service.
- Let $P = \{\pi_1, \pi_2, \ldots, \pi_k\}$ be the set of all proofs generated by the service provider, where k is the total number of proofs associated with the provision of the service. The client selects a minimum number m of zero-knowledge proofs required to consider the service as valid. The smart contract uses a pseudo-random function $f_{\text{selectProofs}}(P, m)$ to select a subset $S \subset P$ of size m.
- Once the proofs have been selected, to complete the payment process, the service provider needs to submit the proofs required by the smart contract for verification. Any submitted proof that does not match the required ones is automatically rejected by the smart contract. Also, the same proof cannot undergo the verification process twice. Let π_i represent the i-th proof in the subset S, then:

$$\forall \pi_i \in S : f_{\text{verify}}(\pi_i) = 1$$
$$\implies f_{\text{releasePayment}}(\text{addr}_{U_s}, D_{U_c})$$

- After all the required proofs are verified by the smart contract, the service provider is able to request the release of the deposit from the customer. If all the requested proofs are verified successfully, then the smart contract can release the deposit from the client upon request. Otherwise, the deposit will remain locked in the smart contract.

This procedure, illustrated in Fig. 4, entails the creation of a different smart contract as part of the setup phase, which combines the payment functionality with the verification mechanism imported from the verifier contract.

1) PROOF SELECTION

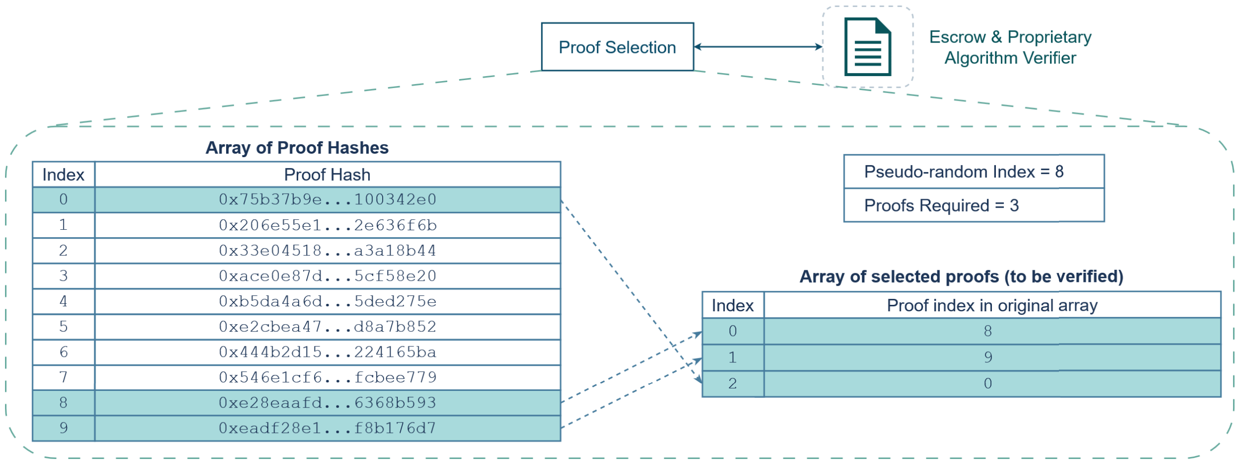
To prevent the submission of all proofs $P = \{\pi_1, \pi_2, \ldots, \pi_k\}$ generated from the service provider's computations to the smart contract, a hash from each one $h(\pi_i)$ is submitted instead. Therefore, when the proofs are subsequently requested, the smart contract verifies not only the match between the provided proof hash and the stored proof hash, but also the validity of the proof itself.

The proposed pseudo-randomization process for proof selection is detailed below. It is important to emphasize that generating random values in a deterministic environment like blockchain poses significant security risks without relying on second-layer applications or oracles, which are off-chain entities operating independently from the blockchain network, providing external data and functionality to smart contracts on the blockchain. This is because blockchains are inherently transparent, making it possible for malicious actors to predict or manipulate random number generation, thus compromising the integrity and fairness of processes reliant on these values. However, we have chosen an on-chain implementation, which means that the entire process operates directly within the blockchain network, ensuring that all transactions are recorded on it. This approach ensures that the generation of pseudo-random values occurs within the blockchain without impacting both client deposits and associated variables within the smart contract, thus safeguarding the integrity of the information stored in it.

The smart contract's proof selection algorithm is primarily based on two parameters supplied by the client: the minimum required number of proofs (P_{req}) and a sealed seed (S_s), along with the number of the blockchain block ($\text{block}_{\text{number}}$) in which the deposit D is made. The sealed seed refers to a

Algorithm 1 Proof Selection Algorithm

```
Input: bytes32: seed                                        ▷ Input parameter: seed of type bytes32
Require: sha256(msg.sender, seed) == sealedSeed             ▷ Function execution condition
1: randomHash ← hash(encode(seed, blockHash))               ▷ Serialization and encoding
2: N ← proofs.length                                        ▷ N representing the total number of proof hashes
3: pseudoRandomIndex ← randomHash % N                       ▷ Modulo operation between randomHash and N
4: for i = 0 ... proofsRequired do                          ▷ Loop iterating from 0 up to proofsRequired
5:     index ← (pseudoRandomIndex + i) % N                  ▷ Array bound checking for circular array iteration
6:     evaluableProofs.push(index)                          ▷ Adding the index to the list of evaluableProofs
7: end for
```



Array of Proof Hashes

Index	Proof Hash
0	0x75b37b9e...100342e0
1	0x206e55e1...2e636f6b
2	0x33e04518...a3a18b44
3	0xace0e87d...5cf58e20
4	0xb5da4a6d...5ded275e
5	0xe2cbea47...d8a7b852
6	0x444b2d15...224165ba
7	0x546e1cf6...fcbee779
8	0xe28eaafd...6368b593
9	0xeadf28e1...f8b176d7

Array of selected proofs (to be verified)

Index	Proof index in original array
0	8
1	9
2	0

FIGURE 5. **Proof selection algorithm. The indexes of the (3) required proofs, starting with the proof hash corresponding to the pseudo-random index (8), followed by the hashes corresponding to subsequent indexes which are 9 and 0, circularly, are copied from the total set of proof hashes to the set of proofs to be verified.**

value provided by the user, obtained by off-chain hashing the combination of their wallet W with a randomly chosen 32-byte value selected by the client, also known as the seed s. Thus, $S_s = h(W, s)$. The seed parameter is hashed upon submission to mitigate the risk of malicious behavior by the service provider, such as potential manipulation of the random number generation process.

Once the service provider has uploaded all the hashes $H_\pi = \{h(\pi_1), h(\pi_2), \ldots, h(\pi_k)\}$ corresponding to the generated proofs, the contract becomes aware of the total number of proofs available from the service provider for a specific client U_c. From this point on, client interaction is required to reveal the seed s so that the smart contract can perform the selection of the zero-knowledge proofs that the service provider must submit. Therefore, the smart contract selects from the entire range of stored proof hashes in a pseudo-random manner. Thus, neither the client nor the provider can know in advance which specific proofs will be selected for the verification process, thereby preventing malicious behavior by the provider.

By providing the seed s, the smart contract confirms that it corresponds to the sealed seed S_s submitted earlier, such that $h(W, s) \equiv S_s$. Then, a pseudo-random index $r \in \{0, 1, \ldots, k-1\}$ is obtained, which indicates from which index in the array of proof hashes uploaded by the vendor the smart contract should begin selecting the proofs. Thus, $P_{\text{sel}} = \{h(\pi_r), h(\pi_{r+1}), \ldots, h(\pi_{r+P_{\text{req}}-1})\}$. The steps involved in the proof selection mechanism executed by the smart contract are outlined in Algorithm 1 to ease comprehension.

The process it undergoes is as follows:

- **Input**: The client passes the seed as a 32-byte type parameter to the function.
- **Require**: The function verifies if the stored sealed seed corresponds to the SHA-256 hash of the serialized combination of the seed and the client's wallet.
- **Line 1**: The blockhash, referring to the hash of the block where the client made the deposit, is serialized and encoded together with the seed. Subsequently, the SHA-256 hash is computed for the serialized and encoded result. Then, the preceding result is converted into decimal format, specifically, as a 256-bit unsigned integer.
- **Line 2**: The variable N is declared as the length of the array containing the proof hashes presented by the service provider.
- **Line 3**: A modular arithmetic operation is performed between the previously obtained random hash and the total count of proof hashes submitted by the service provider. Then, the result is cast to an 8-bit unsigned integer. This measure is taken to prevent the ranges of proofs to be verified from becoming excessively large and overloading the smart contract.
- **Lines 4 - 7**: Finally, the function iterates over the number of proofs required by the user. When the proof hash

stored at the previously obtained pseudo-random index is found, its index is added to an array of evaluable proofs. If more than one proof is required, the others will be those corresponding to the immediately subsequent indices to the one calculated before, and so on until reaching the total requested number of proofs.

For a better understanding of the above, Fig. 5 serves as clarification.

Using this pseudo-random index, which is the result of all previous operations, and the number of proofs to be verified, the smart contract determines which zero-knowledge proofs must be submitted for verification. This is done by counting from the proof hash corresponding to the pseudo-random index within the circular array of proof hashes to the proof hash whose index corresponds to the sum of the pseudo-random index and the number of proofs required by the client to be verified.

D. AGENT ACTIONS WITHIN THE FRAMEWORK

This section describes the set of actions executed by the client and the service provider through the smart contract after its deployment. Taking into consideration the importance of the proof generation process within the scope of this research, it is described in its own section, whereas the remaining framework interactions are explained in a unified section.

1) PROOF GENERATION

Once the framework is set up, the main task for the service provider (U_s) is to generate the zk-SNARK proofs (π) concerning the job carried out for a particular client U_c. These proofs allow determining whether outcomes (y) are consistent with the rendered service, thereby ensuring the integrity and transparency of the system. Fig. 6 presents a step-by-step progression detailing the proof generation process, as well as the required files for its creation.

In order for the proposal to cover a general procedure that can be adopted by different schemes, it is worth noting that the processes outlined for both verifier generation and proof construction, as illustrated in Fig. 3 and Fig. 6, respectively, are agnostic to the specific zk-SNARK scheme. For the analysis of results and their accurate evaluation, as mentioned earlier, the zk-SNARK constructions Groth16, Plonk, and Fflonk are employed.

Similar to other components within the established framework, zk-SNARK proofs are also generated by the service provider. It is essential to point out that, as the algorithms are proprietary, only the vendor is aware of their implementation. This is why the provider, who acts as the prover, should always know all the inputs of the circuit, which are $\{X_{U_c}, \theta\}$. The generation of proofs is based on the knowledge of the circuit inputs that serve as a solution to the proposed computational problem.

Keeping the above in mind, the process of generating a zk-SNARK proof associated with a proprietary machine learning model, not only requires understanding the circuit

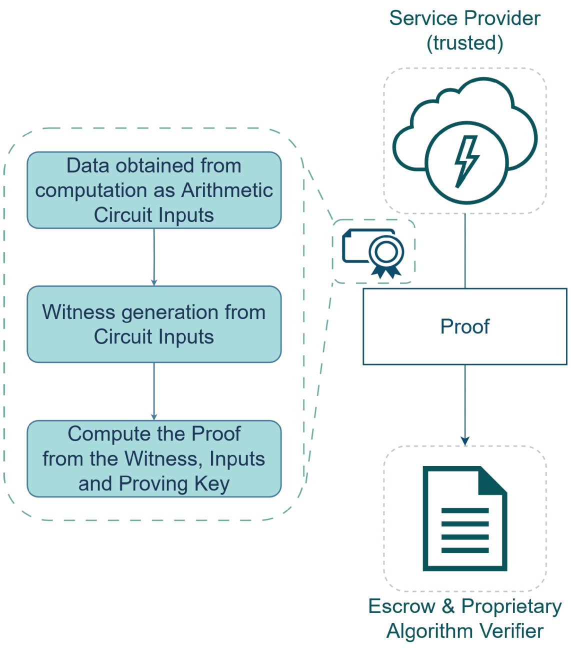

Service Provider (trusted)
Data obtained from computation as Arithmetic Circuit Inputs
Witness generation from Circuit Inputs
Compute the Proof from the Witness, Inputs and Proving Key
Proof
Escrow & Proprietary Algorithm Verifier

FIGURE 6. Steps involved in generating the zk-SNARK proof to be evaluated by the escrow and proprietary algorithm zero-knowledge verifier.

inputs to be processed, but also the weights and biases, which are considered critical parameters of such algorithms. This does not necessarily imply that these elements are always essential. There may be layers within a CNN model that only require inputs without biases or weights, such as normalization, pooling, or flatten layers, as shown in Table 2. Following the identification of these elements, it should be emphasized that, as part of the circuit input set, both public inputs and private inputs may exist. Although the former can be disclosed, the latter are designed to remain undisclosed and not deduced from the R1CS. In this scenario, given that the inputs represent values pertinent to the model's implementation, it is imperative to keep them private to avoid public disclosure.

From these inputs, calculations are performed for both intermediate and output signals, effectively fulfilling the required signals that satisfy the arithmetic circuit. This process is equivalent to acknowledging a solution to the R1CS. The set of all signals that satisfy the constraints of the circuit is known as the witness, which is $w = \{X_{U_c}, \theta\}$. Zk-SNARK proofs enable computations on circuits without disclosing any information about the involved signals, except for the public inputs and outputs. Consequently, it is possible to demonstrate knowledge of private inputs without disclosing their values, ensuring that the circuit's conditions are met, such that $C(X_{U_c}, \theta) \rightarrow$ True. A valid witness must match all the circuit constraints without disclosing any inputs other than those publicly known. On the contrary, an invalid witness fails to meet the circuit's constraints, where $C(X'_{U_c}, \theta') \rightarrow$ False.

To calculate the proof π, both the witness w and the specific proving key P_k, generated during the verifier setup, are used. Although the private inputs are not explicitly included in the proof, their information is indirectly integrated into the proof parameters, enabling the verification of the computation performed by the prover without revealing the underlying values. This procedure results in the creation of the proof itself, such that $\pi = f_{\text{proof}}(P_k, \{X_{U_c}, \theta\})$. In the specific context of this research, as depicted in Table 1, the circuits have a fixed number of outputs and lack any public inputs. Consequently, the outputs y constitute the only parameters presented alongside the proof π in the verification process, that is, $f_{\text{verifyProof}}(V_k, y, \pi)$. A valid proof not only demonstrates the knowledge of a set of signals that fulfill the circuit constraints but also verifies the consistency between the public inputs and outputs presented alongside the proof and those used to generate it.

2) FRAMEWORK INTERACTIONS

Having comprehensively outlined the foundational elements of the framework and its setup, this section explores the interactions between the client U_c and the service provider U_s through the developed escrow and verifier smart contract SC. Additionally, the specific procedures carried out by the smart contract are detailed below.

Initially, we propose maintaining the conventional approach where users obtain services from service providers, accessing them through their respective websites or platforms. However, in this study we do not cover the mechanisms governing client-provider interactions for service delivery; rather, we focus on the payment process and the subsequent service verification. Thus, the only client-related information accessible to the vendor is the address $addr_{U_c}$ used by the client to make the deposit D_{U_c}. Furthermore, the submission and presentation of computations stemming from service provision to clients lie outside the scope of this paper.

For ease of readability, Table 3 presents the notations used in the proposed framework, i.e., in the smart contract. Besides, Algorithm 2 displays the state variables used within the smart contract, including the smart contract incorporating the logic for verifying proofs associated with a custom CNN model and a specific zk-SNARK construction. Additionally, it includes the struct containing information related to the proofs, the minimum number of proofs the service provider must submit, and mappings to associate both the proofs and balances with clients.

Algorithm 3 and Algorithm 4 describe the possible actions within the smart contract, as well as the required parameters. For brevity, the getter functions have been omitted. As previously discussed, the smart contract acts as an intermediary, serving the dual role of an escrow mechanism and a proof verifier facilitated by the service provider regarding the proprietary algorithm. Although the current implementation of the contract supports multiple payments linked to a single proprietary algorithm owned by the supplier, thus enabling several clients to engage with a specific algorithm, this description focuses only on the interaction with a single client.

TABLE 3. Notations within the proposed system for the smart contract.

Notation	Description
proofVerifier	Imported verifier smart contract associated with a custom CNN model and a zk-SNARK construction
serviceVerification	Structure defining service verification
sealedSeed	Sealed seed
storedBlockNumber	Block number of the blockchain
proofs	Submitted proof hashes
evaluableProofs	Index corresponding to evaluable proofs (candidates for verification)
proofsRequired	Number of proofs required by the client. Equal to or less than total_proofs
proofExistence	Mapping to determine if a proof has been submitted
proofToEval	Mapping to determine if a proof has been selected to be verified
totalProofs	Minimum number of proof hashes the service provider must submit. Amount fixed by service provider.
clientServices	Mapping associating service verification with the client
balanceOf	Mapping associating accounts and their balance
clientAddr	Client's address
proofHash	Hash generated for a proof following the algorithm of Fig. 8
msg.sender	Sender's address of the transaction
msg.value	Amount of Ether sent along with a transaction in the smart contract
publicSignals	Proof's public signals
proof	Zk-SNARK proof
toAddr	Recipient's payment address
getProofHash	Internal function that computes the proof hash based on the proof and the public signals

Algorithm 2 Escrow ZKML Verifier State Variables

```
 1: zkSNARKVerifier                  ▷ zkSNARK Verifier Contract
 2: Struct serviceVerification:                          ▷ Struct
 3:     sealedSeed;                                     ▷ bytes32
 4:     storedBlockNumber;                              ▷ uint256
 5:     proofs;                                ▷ array of bytes32
 6:     evaluableProofs;                       ▷ array of uint256
 7:     proofsRequired;                                 ▷ uint256
 8:     proofExistence;                ▷ mapping(bytes32 => bool)
 9:     proofToEval;                   ▷ mapping(bytes32 => bool)
10: totalProofs;                                        ▷ uint256
11: clientServices;              ▷ mapping(address => proof_data)
12: balanceOf;                      ▷ mapping(address => uint256)
```

It is important to highlight that the proposed solution does not account for adversarial scenarios, such as instances where the service provider declines to submit proofs or uploads a quantity of proof hashes below the specified amount. Given the assumption that the vendor is a trusted entity, the smart contract lacks mechanisms to address such situations. Also, to clarify the interactions outlined in the proposed framework, Fig. 7 displays a timeline of the actions performed among the main components of the model.

In order to validate the service provided by the vendor and subsequently release the payment, the proposed framework requires a set of inputs that must be attached to the smart

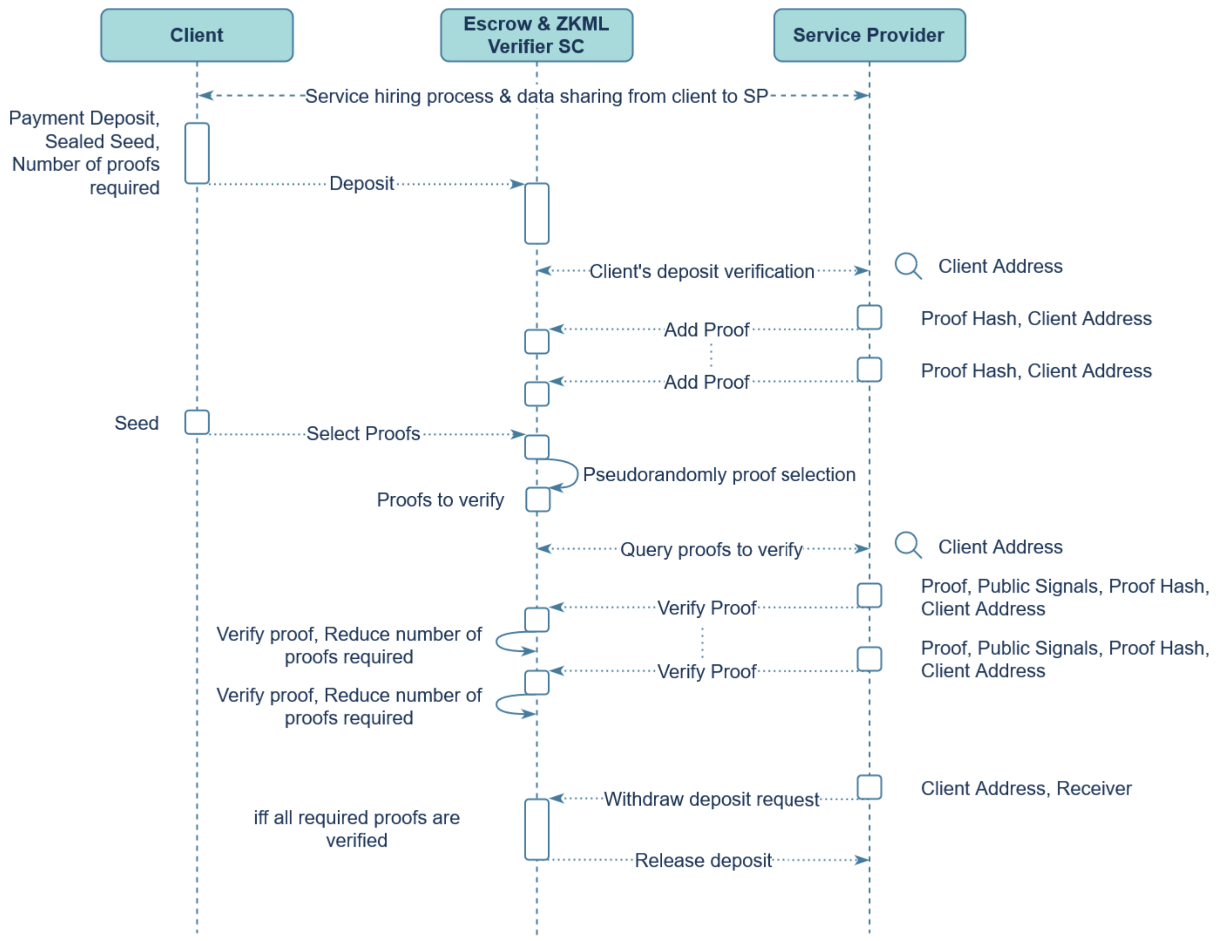

Client
Escrow & ZKML Verifier SC
Service Provider
Service hiring process & data sharing from client to SP
Payment Deposit, Sealed Seed, Number of proofs required
Deposit
Client's deposit verification
Client Address
Add Proof
Proof Hash, Client Address
Add Proof
Proof Hash, Client Address
Seed
Select Proofs
Pseudorandomly proof selection
Proofs to verify
Query proofs to verify
Client Address
Verify Proof
Proof, Public Signals, Proof Hash, Client Address
Verify proof, Reduce number of proofs required
Verify Proof
Proof, Public Signals, Proof Hash, Client Address
Verify proof, Reduce number of proofs required
Withdraw deposit request
Client Address, Receiver
iff all required proofs are verified
Release deposit

FIGURE 7. Agents' Interactions within the Escrow and Verifier Smart Contract. The diagram excludes the processes of hiring a service and data sharing for the provision of the service between the client and the service provider, as these are beyond the scope of this study.

Algorithm 3 Escrow ZKML Verifier Functions (Client)

```
1:  function deposit($\_sealedSeed$, $\_proofsRequired$)
2:      require(balanceOf[$msg.sender$] == 0);
3:      require($msg.value > 0$);
4:      clientServices[$msg.sender$] ← serviceVerification {
5:          sealedSeed: $\_sealedSeed$,
6:          storedBlockNumber: $block_{number} + 1$,
7:          proofsRequired: $\_proofsRequired$
8:      }
9:      balanceOf[$msg.sender$] ← $msg.value$;
10: end function
11:
12: function selectProofs($seed$)
13:     require(len($proofs[]_{clientAddr}$) == totalProofs);
14:     require(len($evaluableProofs[]_{clientAddr}$) == 0);
15:     require($storedBlockNumber_{clientAddr} < block_{number}$);
16:     proofSelectionAlgorithm($seed$); ▷ See Algorithm 1
17:     return true;
18: end function
```

contract during the interactions, leading to the recording of transactions on the blockchain.

In conjunction with the requirement of depositing an asset with a predetermined value on the blockchain, the client must also specify the quantity of proofs (P_{req}) that the smart contract must verify for the service to be considered valid. Additionally, a sealed seed S_s from the client, which will later be disclosed for the proof selection algorithm, must be attached. The deposit D_{U_c} is a sum set by the service provider and, as outlined in the proposal, represents a specific quantity of cryptocurrency assets. These assets could be a native token inherent to the blockchain used or a stablecoin, which is a type of cryptocurrency designed to maintain a stable value by pegging it to an external asset such as fiat currency or commodities. Whereas the former option aligns with our current proposal, the latter presents a practical alternative for implementation.

From the provider's perspective, there exists the possibility of checking whether a deposit has been made specifically for the proprietary service associated with the smart contract, by invoking balanceOf(addr$_{U_c}$). This interaction incurs no cost, and the only parameter the vendor needs to include in the blockchain transaction is the client's address. Prior knowledge of this client address is beyond the scope of this manuscript but may result from previous direct communication between the client and the service provider.

Algorithm 4 Escrow ZKML Verifier Functions (Service Provider)

```
1:  function constructor(verifier)
2:      proofVerifier ← zkSNARKVerifier(verifier)
3:  end function
4:
5:  function addProofs(clientAddr, proofHashes[])
6:      require(balanceOf[clientAddr] > 0);
7:      for each hash h in proofHashes[] do
8:          if proofExistence[h] == false then
9:              proofs[].push(h)
10:             proofExistence[h] ← true
11:         end if
12:     end for
13: end function
14:
15: function verifyProof(clientAddr, proofHash, proof[], publicSigals[])
16:     require(proofExistence[proofHash] == true);
17:     require(proofToEval[proofHash] == true);
18:     require($proofsRequired_{clientAddr}$ > 0);
19:     result ← getProofHash(proof[], publicSigals[]);
20:     require(result ≡ proofHash);
21:     verify ← call(
22:         proofVerifier.verifyProof(proof[], publicSigals[]
23:     );
24:     require(verify);
25:     proofToEval[proofHash] ← false;
26:     $proofsRequired_{clientAddr}$−−;
27:     return true
28: end function
29:
30: function withdraw(clientAddr, toAddr)
31:     require(len($proofs[]_{clientAddr}$) == totalProofs);
32:     require($proofsRequired_{clientAddr}$ == 0);
33:     result ← toAddr.transfer(balanceOf[clientAddr]);
34:     require(result);
35: end function
```

It would also be possible to obtain this client address by monitoring the deposits made on the smart contract. Moreover, an increased requirement of proofs by the client correlates with a greater number of blockchain transactions that the service provider must submit, potentially impacting the final service cost for the client. Hence, it is important to consider that the provider may adjust the service price based on the number of transactions to be recorded.

At this point, for each batch of inferences or results generated by the vendor regarding the rendered service, the proof hashes of all produced proofs $H_\pi = \{h(\pi_1), h(\pi_2), \ldots, h(\pi_k)\}$ for each of the obtained results must be submitted, along with the corresponding client address $addr_{U_c}$, to the smart contract as evidence of job completion, such that addProofs($\text{addr}_{U_c}, H_\pi[\,]$). The set of proof hashes is structured as an array, with each hash consisting

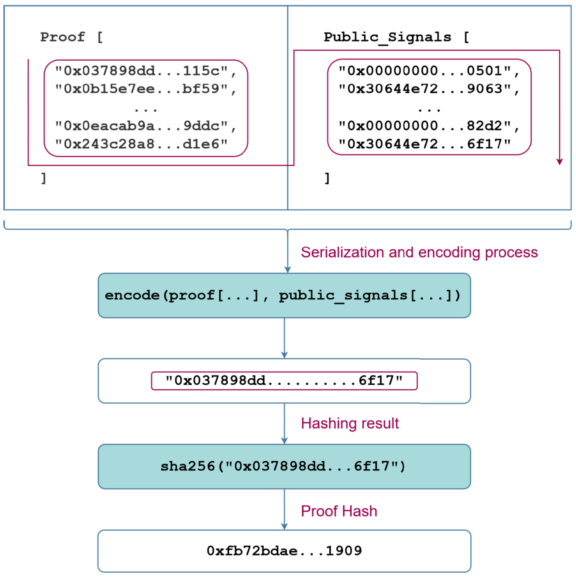


FIGURE 8. Proof hash generation process from the data of a zk-SNARK proof. The proof and public signals are serialized, encoded, and then hashed to produce the result. The process is agnostic to the zk-SNARK construction used.

of 32 bytes. These hashes are computed by serializing and encoding both the proof and the public signals, and then hashing the result using the SHA-256 hash function. Thus, $h(\pi_i) = H_{\text{SHA256}}\big(\text{encode}(\text{proof}_{\pi_i}[\,], \text{publicSignals}_{\pi_i}[\,])\big)$. Fig. 8 aims to provide clarity on this process. This enables the provider to prove possession of proofs that may be subsequently requested by the smart contract for verification. Then, when the provider submits the proof, the contract validates that the submitted proof corresponds with the previously registered proof hash, verifying $h(\pi_i) \equiv \text{checkProof}(\text{proof}_{\pi_i}[\,], \text{publicSignals}_{\pi_i}[\,])$. It is important to note that the vendor is unaware of which proofs will be requested, necessitating the recording of all these attestations in the contract, even if some proofs may remain unrequested.

In the proposed implementation, the smart contract includes an internal parameter specifying the minimum number of proof hashes that the service provider must submit (totalProofs), according to the service provided to a specific client. This number, defined by the vendor, is public; hence, the client must consider it when selecting the quantity of required proofs P_{req} to ensure it is never exceeded, such that $P_{\text{req}} \leq$ totalProofs. However, it is still possible for the service provider to include additional proofs beyond the specified amount of required proofs by the client, although this scenario is considered unlikely due to the potential increase in economic costs for the vendor.

Then, the client initiates a second interaction with the smart contract. During this interaction, the client discloses the seed s parameter that enables the smart contract to pseudorandomly select the proofs that must be later submitted by the provider for verification, known as P_{toEval}. This selection is based on the mechanism presented in Algorithm 1, which employs the seed provided by the client along with an on-chain parameter from the blockchain network. This on-chain parameter, known as the blockhash, represents the hash of the block where the deposit occurred, and it is concatenated with the seed s to generate a new hash. This new hash is then used to calculate a pseudo-random index that enables the proof selection.

When proof selection concludes, the smart contract awaits the submission of the corresponding proofs for verification. Subsequently, the service provider must query the specific proofs required for verification by specifying the client's address. Similar to the transaction designated to verify if a specific client has made a deposit, this interaction also incurs no economic cost to the service provider.

Once the provider identifies the proofs required for verification, each proof must be submitted individually for verification. This entails attaching not only the proof itself but also the corresponding proof hash previously submitted to the contract, along with the client's address and the proof public signals. Thus, invoking $\text{verifyProof}(\text{addr}_{U_c}, h(\pi_i), \text{proof}_{\pi_i}[\quad], \text{publicSignals}_{\pi_i}[\quad])$. Including the proof hash again serves to assert that the provided proof matches the expected one, such that $\left(h(\pi_i) \in P_{\text{toEval}}\right) \wedge \left(h(\pi_i) \equiv \text{checkProof}(\text{proof}_{\pi_i}[\ ], \text{publicSignals}_{\pi_i}[\ ])\right)$. It is important to note that the format of the proof and the public signals varies depending on the zk-SNARK construction used, resulting in different sizes for the corresponding proof data submitted for each scheme. Afterwards, the smart contract verifies each proof as it is submitted by the provider, recording the corresponding transaction on the blockchain. With each successful verification, the number of proofs required by the client diminishes until reaching zero, meaning that no further proofs need verification and payment release can proceed.

Ultimately, when no further proofs require verification, the service provider can initiate the payment release process. This is done through a new interaction where the vendor specifies the client's address and the payment recipient. Usually, the payment recipient will be the owner of the algorithm, but the provider retains the ability to designate an address (toAddr) of their choice as the recipient. If all required proofs are successfully verified, the payment release process is executed. However, if any proof fail verification, the payment cannot be released.

$$\begin{aligned}\forall \pi_i \in P_{\text{toEval}} &: f_{\text{verify}}(\pi_i) = 1 \\ &\implies f_{\text{releasePayment}}(\text{toAddr}, D_{U_c})\end{aligned}$$

3) SECURITY CONSIDERATIONS AND LIMITATIONS

In this section, the constraints of the proposed framework are discussed, covering both functional and security aspects.

To start with, the code of the developed framework is publicly available. Therefore, all participants have access to it, though it falls upon the service provider to deploy it on the blockchain. Conversely, not all methods included in the source code can be invoked by anyone. Instead, a series of restrictions, detailed below, are enforced through internal mechanisms implemented within the smart contract:

Theorem 1: Authorization constraint. Only the client can invoke the deposit and proof selection methods, and only the service provider can invoke the methods for adding, verifying proofs, and withdrawing payments.

Let's denote:

- M_c: Methods that can only be invoked by the client (deposit, proof selection).
- M_s: Methods that can only be invoked by the service provider (add proof, verify proof, withdraw payment).
- Each participant in the system is either a client or a service provider, i.e., $U_c \cap U_s = \emptyset$.

Then, the invocation restriction can be formulated as:

$$\text{Invoke}(u, m) \implies (m \in M_c \wedge u \in U_c) \vee (m \in M_s \wedge u \in U_s)$$

Proof: Since the smart contract is public and the invocation checks are performed within the contract:

- If a user $u \notin U_c$ attempts to invoke $m \in M_c$, the contract will reject the invocation.
- Similarly, if $u \notin U_s$ attempts to invoke $m \in M_s$, the contract will reject the invocation.

The proposed framework establishes a client's association with a proprietary algorithm through the smart contract. Accordingly, the service provider links the zk-SNARK proof, along with its hash, to a specific client.

Theorem 2: Client-specific proof verification: The verification of one client's proof does not affect the proofs of another client.

Proof: Let C_i represent client i and π_{ij} represent the j-th proof associated with client C_i.

- *Proof association:* Each proof π_{ij} is associated with a client C_i through the smart contract SC. This is achieved by linking the proof π_{ij} and its hash to C_i:

  $$\text{mapping}(C_i, \pi_{ij}) = \text{True}$$

- *Independence:* For any two clients C_i and C_k $(i \neq k)$, their proofs are independent:

  $$\text{mapping}(C_k, \pi_{ij}) \vee \text{mapping}(C_i, \pi_{kl}) = \text{False}$$

  where π_{kl} represent the l-th proof associated with client C_k.

Similarly, the required number of proofs for each client is independent of others. Moreover, the remaining number of proofs to be verified depends on the situation of each client. Notably, the minimum number of proofs required by a client is determined individually, unrelated to the total number of required proof hashes established by the provider. The latter remains a constant parameter for all clients of the proprietary algorithm specified in the smart contract. In addition, it should be noted that the current implementation only supports the verification of one proof at a time, rather than in batches.

Regarding the payment amount for the service, as discussed earlier, it is only feasible through an asset with a predetermined value within the blockchain. Therefore, to implement the proposed framework, the blockchain must support smart contracts and possess an inherent token for the deposit. As a result, a native token from another blockchain network cannot be used. Additionally, traditional payment methods through fiat currency are not viable in the current implementation, as they would compromise the privacy objectives of the proposed framework.

In the current proposal, it is assumed that the service provider is trusted, which implies a certain level of trust in the system regarding service provision. This means that the possibility of the provider deceiving the client about the version of the model used is beyond the scope of this research. Likewise, if the framework were to be implemented in a trustless environment, a possible implementation could be through an additional mechanism using publicly available model metadata. In that approach, the service provider would register the metadata of the proprietary algorithm and its hash on the blockchain. When verifying the service employing zk-SNARKs, the corresponding hash of the model registered on the blockchain would additionally be included in the proof.

V. RESULTS AND DISCUSSION

To assess the applicability and effectiveness of the proposed framework, this section exposes the results related to the implementation cost in terms of resource consumption and setup time. The computational cost of interactions through the framework on a general-purpose blockchain, which supports smart contracts and payments via cryptocurrency assets, is also examined. All the evaluations conducted are based on the three zk-SNARK constructions selected for this research.

A. PERFORMANCE EVALUATION

In order to obtain metrics for evaluating the applicability of the framework, we propose using multiple custom CNN proprietary algorithms, as depicted in Table 2, which are based on different layers to configure different scenarios. From these CNN models we evaluate the framework setup, the sizes of the zk-SNARK proofs, zero-knowledge keys and public signals, as well as their corresponding generation times.

The proposed framework operates on a device featuring an AMD Ryzen 5 5600G processor with a clock frequency of 3,900 MHz and 48 GB of RAM, running Ubuntu 22.04.4 LTS. A local instance of an Ethereum-based blockchain network is used, with smart contracts compiled and executed in Ethereum Remix. For representing arithmetic circuits, Circom [36] is used, with circuit designs based on templates from libraries like CircomLib [37] and Circomlib-ml [38]. The snarkjs [39] library is employed for constructing both the proving system and the zk-SNARK proofs.

For the purpose of addressing a use case as close to a real-world environment as possible, the ceremony process conducted is the Powers of Tau. However, to avoid the contribution of individual participants in generating the security parameters of the zk-SNARK construction in this study, a precomputed *ptau* file from a former Powers of Tau ceremony process is used. Although not recommended for production environments, this practice remains a viable option within testing scenarios.

Considering that three custom CNN models simulating proprietary models are used, Table 4 illustrates the input features required to generate the witness for each one. Only layers that contain trainable parameters and contribute to the optimization of the CNN are considered.

Replicating the processes outlined in the Algorithm Verification Mechanism section, the resulting zero-knowledge key comprises both the proving key and the verification key for each circuit and zk-SNARK construction. The variation in the size of these keys, as seen in Table 5, arises from specific computation parameters and other structures within the proving key, fundamental for generating the proofs. Moreover, while the verification key is employed in the verifier generation process, the zero-knowledge key is used to generate the verifier smart contract. This process results in a smart contract encoded in Solidity, a high-level programming language used for developing smart contracts. Hence, considering three distinct CNN models across three different zk-SNARK constructions, a total of nine smart contracts need to be computed.

For verification to take place, it is imperative to generate the corresponding zk-SNARK proofs. With three different zk-SNARK constructions and three unique custom CNNs serving as proprietary algorithms, a total of nine different types of proofs should be generated. These proofs are derived from both the proving key and the witness. However, although the witness is tied to the model, it remains consistent across all zk-SNARK constructions. This implies that the format and quantity of parameters composing the proofs differ uniquely for each zk-SNARK scheme. Additionally, besides the proofs, the public signals must be provided for the verification to proceed. As depicted in Table 5, the components involved in verifying the job executed by the service provider depend on both the specific proprietary algorithm and the zk-SNARK construction employed.

Analyzing the findings shown in Table 5, the size of the witnesses tends to be larger as the model complexity increases, regardless of the zk-SNARK scheme. This trend is similar in the case of public signals due to the higher number of inputs in CNN models with more layers. Notably, there is a difference of 97% in witness size and 98% in public signals between the most complex and simplest custom CNN models.

Regarding the smart contract, no significant differences are noticed as the model becomes more complex. As a case where the difference is most apparent, for the simplest model, there is a reduction in contract size of up to 87.6% for verification using Groth16 instead of Fflonk, while the reduction remains at 54.4% between Plonk and Fflonk.

In relation to zero-knowledge keys, Groth16 emerges as the most efficient scheme across all proposed CNNs, suggesting that efficiency in terms of key size can be achieved regardless of model complexity. Notably, there is an improvement of

TABLE 4. Input data required for generating the witness for each custom Convolutional Neural Network model used in this manuscript. 'Input Features' denotes the number of input characteristics employed by each model. 'Layer' provides the layers within the CNN architecture. 'Weights' and 'Bias' indicate the respective numbers of trainable parameters in each layer. 'Input Size' specifies the dimensions of the input data required for processing by each CNN model.

Model	Input Features	Layer	Weights	Bias	Input Size
Custom CNN 1	784 (28 ×28)	conv2d dense	9 6,760	1 10	76,5 KB
Custom CNN 2	784 (28 ×28)	conv2d conv2d_1 dense	36 288 2,000	4 8 10	14,4 KB
Custom CNN 3	3	dense dense_1	6 2	None None	108 B

TABLE 5. Sizes of the components involved in the verification process determined by specific proprietary CNN models and zk-SNARK constructions.

Model	Witness	zk-SNARK Construction	Smart Contract	ZKnowledge Key	Verification Key	Public Signals	Proof
Custom CNN 1	11.9 MB	Groth16 Plonk Fflonk	11.0 KB 45.2 KB 70.6 KB	234.4 MB 6.61 GB 8.22 GB	4,750 B 2,041 B 1,108 B	828 B	808 B 2,250 B 2,259 B
Custom CNN 2	2.3 MB	Groth16 Plonk Fflonk	11 KB 45 KB 70.4 KB	37.5 MB 821.2 MB 1,022.5 MB	4,750 B 2,042 B 1,109 B	614 B	803 B 2,244 B 2,251 B
Custom CNN 3	33.7 KB	Groth16 Plonk Fflonk	7.2 KB 27.6 KB 60.5 KB	708.3 KB 12.6 MB 18.9 MB	2,923 B 2,038 B 1,106 B	16 B	804 B 2,252 B 2,255 B

TABLE 6. Generation times of the components involved in the verification process, all based on particular proprietary custom CNN models and zk-SNARK constructions.

Model	Circuit Compilation	Witness	zk-SNARK Construction	Zero-Knowledge Key	Proof & Public Signals	Smart Contract
Custom CNN 1	8.55 s	0.449 s	Groth16 Plonk Fflonk	10 min 22 s 6 min 18 s 8 min 29 s	7.8 s 14 min 40 s 29 min 50 s	0.386 s 0.388 s 0.391 s
Custom CNN 2	6.83 s	0.103 s	Groth16 Plonk Fflonk	2 min 15 s 27.35 s 31.12 s	1.97 s 1 min 48 s 2 min 50 s	0.298 s 0.312 s 0.380 s
Custom CNN 3	0.113 s	0.030 s	Groth16 Plonk Fflonk	8.07 s 1.19 s 1.29 s	0.387 s 3.81 s 4.87 s	0.301 s 0.306 s 0.308 s

up to 97.1% using Groth16 instead of Fflonk for the Custom CNN 1 algorithm. The situation is reversed for the size of the verification key, which is up to 328.70% larger in Groth16 compared to Fflonk for the same CNN. This implies that the trend observed in the zero-knowledge key would likely be mirrored in the proving key, meaning it would be smaller for Groth16 than for the other schemes.

Upon analyzing the size of the proofs, while the smallest size is found using the Groth16 construction, it can be concluded that regardless of the complexity of the algorithm to be verified, the size of the proofs remains consistent and is only determined by the specific zk-SNARK construction. In fact, there is a reduction of approximately 64% between Groth16 and the other two schemes employed, whereas differences in proof size between these two models do not exceed 1%.

Regardless of size variations, concerning on-chain verification of the proofs, all sizes are sufficiently suitable for both the proof and public signal sizes. Thus far, it can be asserted that Groth16 emerges as the most efficient algorithm in terms of size of the zero-knowledge keys, smart contract size, and proof size. Nonetheless, this conclusion is challenged when examining the results of Table 6.

Similar to the analysis presented in Table 5, an evaluation is performed for each component involved in the verification process within the framework to determine the time required for the service provider to set it up. Across all scenarios, irrespective of the zk-SNARK construction employed, there is a relevant increase in framework configuration times as model complexity escalates.

In this analysis, both the generation times for witnesses and smart contracts yield closely comparable results, irrespective of the model or zk-SNARK scheme. Additionally, the times for circuit compilation remain relatively low across all cases. However, a substantial reduction of 98.6% can be observed between the most complex and simplest model.

About the zero-knowledge keys, there is an escalating trend in the generation times of these keys as the model complexity increases. Furthermore, for each custom CNN model, a significant contrast is observed comparing the Groth16 scheme to Plonk and Fflonk, which have similar generation times, with Groth16 taking up to 6 times longer

TABLE 7. Gas cost for deploying each contract required to enable both the payment process and the verification of proprietary algorithms based on custom CNN models.

Model	Smart Contract Function	Transaction Issuer	zk-SNARK Construction		
			Groth16 (Gas)	Plonk (Gas)	Fflonk (Gas)
Custom CNN 1	Verifier Deployment	Service Provider	550,541	1,675,778	3,702,995
	Escrow & Verifier Deployment	Service Provider	977,758	945,303	945,303
Custom CNN 2	Verifier Deployment	Service Provider	550,253	1,662,993	3,690,603
	Escrow & Verifier Deployment	Service Provider	977,758	945,291	945,291
Custom CNN 3	Verifier Deployment	Service Provider	347,970	1,188,996	3,033,241
	Escrow & Verifier Deployment	Service Provider	977,314	945,531	944,871

than Plonk for the third CNN. Nevertheless, this discrepancy diminishes as the model complexity increases, although Groth16 still exhibits the highest key generation times. The opposite is observed for the creation of the proof and the public signals, which occur during the same process, thus the generation time covers both components. Significant differences are noticeable across each CNN when contrasting Groth16 with the other two zk-SNARK constructions, as Plonk and Fflonk are notably slower in creating both the proofs and the public signals.

It is important to note that the zero-knowledge keys are produced once, whereas the witness, proofs, and public signals need to be computed for each job performed by the service provider, which could entail a considerable time commitment as the number of proofs increases. This becomes even more critical if proofs are not selected for verification, leading to a loss of resources for the service provider. Nonetheless, depending on the zk-SNARK scheme employed, the consumption of computational resources may be lower in certain cases, and therefore considered acceptable, given the generation times outlined in Table 6.

Comparing the proposed framework with the work of Song et al. [40], which presents a traceable and private data exchange scheme based on NFT and zero-knowledge proofs, it is observed that, although the ML models and circuit constraints vary, the proof sizes in our research are slightly smaller and the proof generation times are longer. Both works use Plonk as a zk-SNARK scheme. However, the setup time is not evaluated. Compared to other schemes analyzed in this work, such as Groth16, the proof generation times are better, again without considering the setup time. In terms of gas cost, the deployment of the verifier contract is the same using Plonk. Other actions are not compared as they fall outside the same scope.

B. COST ANALYSIS

To complement the results section, we also examined the computational cost associated with each interaction within the framework on a general-purpose blockchain. For testing purposes, a local instance of an Ethereum-based network is employed, and the computational cost is evaluated using Gas, which denotes the measure of computational resources required to execute a transaction on such networks. Table 7 illustrates the Gas costs of the smart contracts deployment, while Fig. 9 illustrates those associated with all potential interactions within the framework for each proprietary algorithm proposed in this study. Alternatively, the graph in Fig. 9 is provided for easier interpretation, illustrating the transactions cost in comparison to the average Gas cost per transaction on the Ethereum main network throughout the year 2024. Considering the accumulated Gas cost for each custom CNN model and a total of 30 million Gas units available per block in the blockchain network, using the framework represents 0.6% of the total block consumption for a client. Meanwhile, for the service provider, verifying five proofs ranges from 9.3% to 20.7% of the total Gas available per block when using the zk-SNARK constructions Groth16 and Fflonk for custom CNN model 3 and custom CNN model 1, respectively.

The metrics obtained from the simulation process will enable us to evaluate the economic cost implications of this proposal, especially considering that the number of proofs could modify such cost. Based on the graph shown in Fig. 9, it can be observed that certain interactions display negligible differences among various custom CNN models or between different zk-SNARK constructions. Nonetheless, discrepancies are noticeable in contract deployments and verification-related interactions.

The deployment of the contracts associated with the framework, comprising the verification contract and the escrow and proprietary algorithm zero-knowledge verifier contract, exhibits a noteworthy disparity in the former contract, which integrates the implementation of the verification function itself. Upon analysis of the results obtained, using Groth16 represents a significant cost saving compared to Plonk and Fflonk, which incur the highest deployment cost. This relationship remains unchanged for custom model CNN 1 and CNN 2, whereas in the simplest model the difference becomes more pronounced when employing Groth16. In contrast, for the escrow and proprietary verifier contract it can be observed that Groth16 has a slightly higher cost compared to Plonk and Fflonk, although the difference is negligible.

To plan the potential cost associated with the usage of the framework for participating agents, Table 8 presents the accumulated cost in U.S. dollars for each participant, contingent upon the model and zk-SNARK construction employed. The costs are calculated by multiplying the Ether's market price by the Gas price. The Ether's market price is based on the average price throughout the year 2024, which is USD 2,959.67. Additionally, the average Gas price throughout the year 2024 is 37.76 Gwei, a denomination of

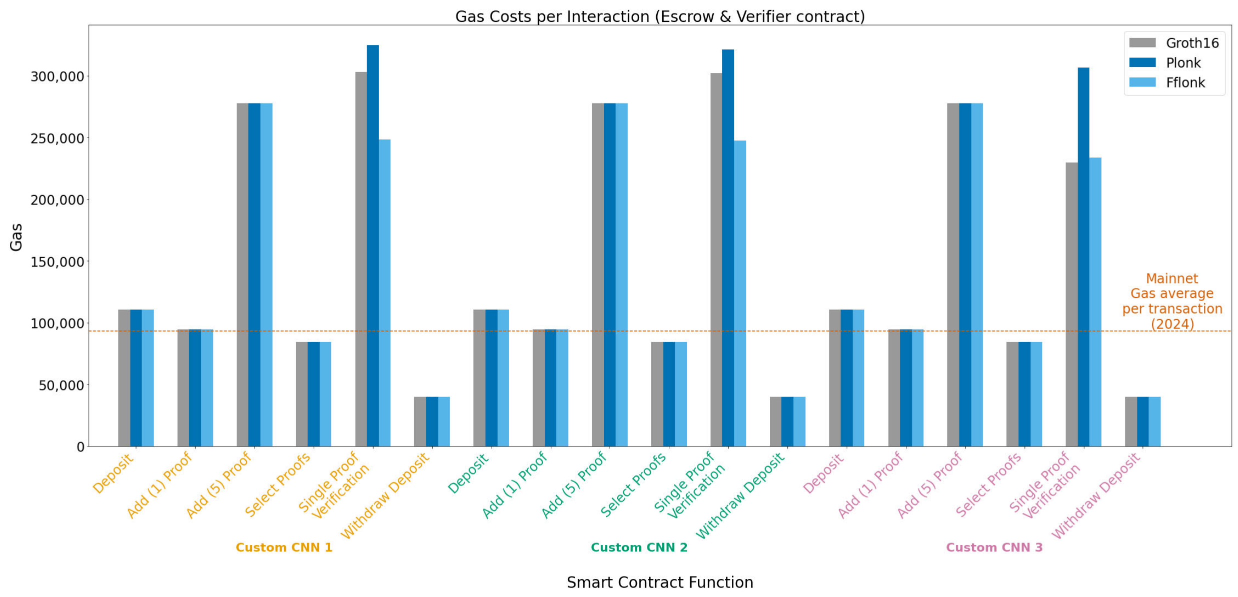


FIGURE 9. Gas cost of each interaction within the framework on the Escrow & Verifier contract. Smart contract deployments are excluded from the graph due to their high values, ensuring clarity in the presentation.

TABLE 8. Accumulated cost in U.S. dollars of all interactions for each agent within the framework using the Escrow & Verifier contract.

Model	Transaction Issuer	Proofs	zk-SNARK Construction		
			Groth16	Plonk	FFlonk
Custom CNN 1	Client	N/A	USD 21.82	USD 21.82	USD 21.82
	Service Provider (Deployment)	N/A	USD 170.8	USD 292.94	USD 519.5
	Service Provider (Verification Process)	1	USD 48.91	USD 51.34	USD 42.81
		5	USD 204.92	USD 217.08	USD 174.43
Custom CNN 2	Client	N/A	USD 21.82	USD 21.82	USD 21.82
	Service Provider (Deployment)	N/A	USD 170.77	USD 291.51	USD 518.11
	Service Provider (Verification Process)	1	USD 48.79	USD 50.96	USD 42.67
		5	USD 204.33	USD 215.2	USD 173.75
Custom CNN 3	Client	N/A	USD 21.82	USD 21.82	USD 21.82
	Service Provider (Deployment)	N/A	USD 148.12	USD 238.56	USD 444.6
	Service Provider (Verification Process)	1	USD 40.72	USD 49.29	USD 41.13
		5	USD 163.97	USD 206.81	USD 166.06

the cryptocurrency Ethereum representing a fraction of one Ether, often used to measure Gas prices on the Ethereum network. By using these values along with the accumulated cost of the actions executed by each agent, the economic cost for each one can be calculated.

As depicted, the cost for a client remains virtually consistent regardless of the model or zk-SNARK scheme utilized. Conversely, for the service provider, it is crucial to scrutinize the cost implications of selecting a specific zk-SNARK construction, as there is a notable disparity between them, particularly depending on the number of proofs requiring verification. The variable number of proofs depends on the client's request, which may in turn be linked to sets of inferences, for instance.

This consideration is paramount, as the cost of verifying a single proof using one construction may be comparable to verifying multiple proofs using another zk-SNARK scheme. This can be observed regarding the accumulated cost, for instance, in the scenario of verifying five proofs using Groth16 with a total cost of USD 375.72, and verifying a single proof using Plonk with an accumulated cost of USD 344.28, both processes for the custom CNN 1 model. Therefore, the service provider must determine the construction to use based on the complexity of the model requiring verification. In making this decision, it is imperative to consider not only the total cost of the entire process, impacting the fee charged to the user for the service provided, but also the generation times of the framework setup and the proofs.

Finally, considering the work of Luong and Park [41], which introduces a healthcare system based on IoT devices, blockchain, and zk-SNARK as an authenticator to protect user privacy and prevent unauthorized access, a comparative analysis is presented that evaluates gas costs on an Ethereum-type network. Compared to our proposal, this work has higher deployment costs for the verifier contract and proof verification; however, the zk-SNARK scheme used is different from those analyzed in our work. Other actions are not compared as they fall outside the same scope.

C. PRIVACY ANALYSIS

In the context of this research, we adhere to the broad definition of personal data outlined by the General Data Protection Regulation (EU GDPR) as any information related to an identified or identifiable natural person. This include identifiers such as a name, identification numbers, location data, an online identifier, or one of several special characteristics that convey physical, physiological, genetic, mental, economic, cultural, or social identity. All of these are considered within the framework of this research as data that could be used by a service provider.

According to Recital 30 of the GDPR, blockchain identifiers, better known as addresses, that can be linked to a natural person fall under the scope of the GDPR. Moreover, these identifiers have the potential to identify an individual user by analyzing transaction patterns associated with publicly known addresses. On the contrary, it is worth mentioning that sending a cryptocurrency asset from A to B is not considered personal data unless combined with additional information [42]. Therefore, to protect user privacy within the proposed framework, the use of one-time addresses is recommended for those who prefer not to be identified and linked to their identity.

Also, whereas encrypted and hashed data serve as pseudonymization techniques, neither inherently anonymizes personal data. Thus, this proposal advocates for the incorporation of zk-SNARK proofs for enhanced privacy protection, as the utilization of such advanced cryptographic techniques ensures compliance with data protection regulations [43]. Hence, data records containing identifiers or transaction traces within a DLT, such as in a use case applied to this proposal, resulting from consented, informed, and voluntary interactions of natural persons, are considered protected data when employing zero-knowledge proofs like zk-SNARKs.

Based on the foregoing, it can be affirmed that the privacy of users is adequately protected within the proposed framework both during the payment process and when receiving a service based on information derived from their personal data. This approach ensures that users can interact securely and confidentially, decoupling their identity from the use of the service and the data involved in its provision at all times.

VI. CONCLUSION AND OUTLOOK

With the expanding availability of services leveraging advanced algorithms, there is a growing demand for the utilization of such AI-powered solutions. Chatbots and content generation platforms currently stand as prominent examples among the extensively utilized solutions within this field. These services are often built on proprietary algorithms, whose implementation remains undisclosed to clients. This lack of transparency serves a dual purpose: protecting intellectual property and maintaining a competitive edge within the industry.

In this research, we have introduced a framework designed to enhance user privacy and ensure the cryptographic validation of contracted services. This privacy is achieved through the inherent properties of blockchain, where, without any specific additional action, the identity of the client can only be linked to their corresponding address used for payment. Our framework demonstrates the feasibility of privately paying for services built on black box algorithms while verifying the use of the correct algorithm version. Here, the framework only releases user payments when the service provider proves, via zero-knowledge proofs, that the service has been delivered. By leveraging blockchain and zk-SNARKs, our proposal offers a promising solution to privacy and security challenges found in SaaS-based services. The transparency and security features of blockchain enable anonymous and secure transactions without relying on centralized intermediaries. Moreover, periodic verification of computations by service providers in decentralized environments, supported by zk-SNARKs, effectively addresses concerns regarding uncertainties in rendered services.

Ongoing research and implementation efforts suggest that zero-knowledge proofs hold the potential to revolutionize the development of protocols and applications focused on privacy, even in environments like blockchain. Due to the intrinsic blockchain features, only the parties involved in the service provision are aware of the payment process's status. However, applying zero-knowledge proofs to verify proprietary algorithms presents several challenges. First and foremost, creating a smart contract capable of verifying specific algorithms is essential. Using smart contracts for on-chain verification and immediate payment release ensures a transparent, private, and secure process. Nonetheless, current size limitations prevent the smart contract from supporting the simultaneous verification of multiple algorithms. Regarding the verification of the provided service, metrics indicate that whereas zk-SNARKs proofs may require lengthy setup times for some constructions, the size of these proofs and public signals is suitable for on-chain verification. Furthermore, although our proposal is founded on zk-SNARKs, it has the flexibility to incorporate other zero-knowledge proofs, such as Bulletproofs [44] or zk-STARKs (zero-knowledge Scalable Transparent Arguments of Knowledge) [45], both of which demonstrate promising performance on blockchains, as long as the necessary resources for implementation are available.

Nevertheless, there are limitations found in the proposed framework, such as the increased cost it may entail for the service provider. As shown in the metrics from Table 8, the deployment of contracts, submission of proofs, and withdrawal of payment entail significant costs for the vendor, which can directly impact the client's charge. Even so, the smart contract's capability to verify a single proprietary algorithm for multiple clients simplifies the service provider's recovery of expenses associated with contract deployment as the client base grows. Another possibility is to implement the system on other blockchain networks with lower or even nonexistent fees. That is why this limitation is considered acceptable since it depends on the specific implementation used and may not always be the case. Also, concerning proof verification, the process is restricted to one proof at a time, rather than handling multiple proofs in batches. Alternatively, the framework's implementation is limited to blockchain networks supporting advanced smart contracts capable of enabling arbitrary coding for verification and payment mechanisms. Additionally, payment is currently restricted to native tokens or stablecoins developed on the underlying blockchain network, excluding other types of assets that may contribute to the privacy of the model.

Within the blockchain context, taking into account the total Gas consumed by all interactions of the framework, widespread adoption could lead to scalability issues. Specifically, when verifying complex models, the entire process represents an important consumption of the total Gas available in the block. Besides, generating a zk-SNARK proof demands a considerable computational workload to perform the required cryptographic operations. These calculations can be resource-intensive and time-consuming, resulting in substantial computational overhead for the service provider. Furthermore, alongside the computational tasks, the storage requirements, which depend on the zk-SNARK construction, must be taken into account when implementing this framework.

In terms of security, potential risks arise, such as the potential exposure of a client's identity linked to their address. To protect user privacy, it is recommended to employ one-time addresses within the framework. Also, it is worth mentioning that the provider is a trusted entity motivated to deliver the best results to attract a larger user base through the provision of multiple proprietary models. Therefore, it is unlikely that the service provider would act maliciously, as doing so would entail deceiving all their clients by generating smart contracts for model versions that perform worse than advertised.

In conclusion, future research efforts should focus on conducting comprehensive empirical evaluations and case studies to demonstrate the efficacy and scalability of the proposed framework in real-world settings. Although the CNN models presented for analysis in this work are theoretical and do not align with commercial models, the proposed approach could be suitable for evaluating more complex models. However, longer setup and proof generation times must be assumed depending on the zk-SNARK construction. Even though blockchain is not considered in their approaches, there are works that implement verification of realistic large ML models using zk-SNARKs, such as the study by Liu et al. [13] and the proposal by Chen et al. [46], where the latter leaves open the possibility of integrating blockchain, and where our proposed method could be applied. Additionally, integrating the model with other zero-knowledge proof implementations, such as zk-STARKs or Bulletproofs, should be explored further as zero-knowledge libraries mature. Moreover, exploring diverse distributed networks that enable the adoption of this framework in both private and interoperable blockchain environments stands as an imperative and central endeavor research objective for future investigations.

REFERENCES

[1] A. M. Pinto, "An introduction to the use of Zk-SNARKs in blockchains," in *Proceedings in Bus. and Economics*. Cham, Switzerland: Springer, 1007, pp. 233–249.

[2] N. Andola, V. K. Yadav, S. Venkatesan, and S. Verma, "Anonymity on blockchain based e-cash protocols—A survey," *Comput. Sci. Rev.*, vol. 40, May 2021, Art. no. 100394.

[3] M. Blum, A. De Santis, S. Micali, and G. Persiano, "Noninteractive zero-knowledge," *SIAM J. Comput.*, vol. 20, no. 6, pp. 1084–1118, Dec. 1991.

[4] C. Lin, D. He, X. Huang, M. K. Khan, and K. R. Choo, "DCAP: A secure and efficient decentralized conditional anonymous payment system based on blockchain," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 2440–2452, 2020.

[5] Z. Hatefi, M. Bayat, M. R. Alaghband, N. Hamian, and S. M. Pournaghi, "A conditional privacy-preserving fair electronic payment scheme based on blockchain without trusted third party," *J. Ambient Intell. Humanized Comput.*, vol. 14, no. 8, pp. 10089–10102, Aug. 2023.

[6] E. Boo, J. Kim, and J. Ko, "LiteZKP: Lightening zero-knowledge proof-based blockchains for IoT and edge platforms," *IEEE Syst. J.*, vol. 16, no. 1, pp. 112–123, Mar. 2022.

[7] Y. Guo, H. Liang, L. Zhu, and K. Gai, "Zk-SNARKs-based anonymous payment channel in blockchain," *Blockchains*, vol. 2, no. 1, pp. 20–39, Feb. 2024.

[8] Y. Hu, A. Manzoor, P. Ekparinya, M. Liyanage, K. Thilakarathna, G. Jourjon, and A. Seneviratne, "A delay-tolerant payment scheme based on the ethereum blockchain," *IEEE Access*, vol. 7, pp. 33159–33172, 2019.

[9] L. Xu, L. Chen, Z. Gao, L. Carranco, X. Fan, N. Shah, N. Diallo, and W. Shi, "Supporting blockchain-based cryptocurrency mobile payment with smart devices," *IEEE Consum. Electron. Mag.*, vol. 9, no. 2, pp. 26–33, Mar. 2020.

[10] X. Deng and T. Gao, "Electronic payment schemes based on blockchain in VANETs," *IEEE Access*, vol. 8, pp. 38296–38303, 2020.

[11] M. Baza, R. Amer, A. Rasheed, G. Srivastava, M. Mahmoud, and W. Alasmary, "A blockchain-based energy trading scheme for electric vehicles," in *Proc. IEEE 18th Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2021, pp. 1–7.

[12] S. Lee, H. Ko, J. Kim, and H. Oh, "VCNN: Verifiable convolutional neural network based on zk-SNARKs," *IEEE Trans. Dependable Secur. Comput.*, vol. 21, no. 4, pp. 4254–4270, Jul. 2024.

[13] T. Liu, X. Xie, and Y. Zhang, "ZkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 2968–2985.

[14] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015, *arXiv:1409.1556*.

[15] H. Sun and H. Zhang. (2023). *ZKDL: Efficient Zero-Knowledge Proofs of Deep Learning Training*. [Online]. Available: https://eprint.iacr.org/2023/1174

[16] G. Team, "Gemini: A family of highly capable multimodal models," Tech. Rep., 2023.

[17] T. B. Brown, "Language models are few-shot learners," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 1–24.

[18] S. Squarepants, "Bitcoin: A peer-to-peer electronic cash system," *SSRN Electron. J.*, vol. 1, no. 1, pp. 1–25, 2008.

[19] N. Szabo, "Smart contracts: Building blocks for digital markets," *J. Transhumanist Thought*, vol. 18, no. 2, p. 28, 1996.

[20] V. Buterin, "A next-generation smart contract and decentralized application platform," *White Paper*, vol. 3, no. 37, pp. 2–1, 2014.

[21] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Proc. 17th Annu. ACM Symp. Theory Comput.*, 1985, pp. 291–304, doi: 10.1145/22145.22178.

[22] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. Comput.*, vol. 18, no. 1, pp. 186–208, Feb. 1989, doi: 10.1137/0218012.

[23] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," in *Proc. 20th Annu. ACM Symp. Theory Comput.*, 1988, pp. 103–112.

[24] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Proc. 3rd Innov. Theor. Comput. Sci. Conf.*, Jan. 2012, pp. 326–349, doi: 10.1145/2090236.2090263.

[25] G. Crescenzo and H. Lipmaa, "Succinct NP proofs from an extractability assumption," in *Proc. 4th Conf. Computability Europe*, 2008, pp. 175–185.

[26] B. Feng, L. Qin, Z. Zhang, Y. Ding, and S. Chu. (2021). *ZEN: An Optimizing Compiler for Verifiable, Zero-Knowledge Neural Network Inferences*. [Online]. Available: https://eprint.iacr.org/2021/087

[27] D. Kang, T. Hashimoto, I. Stoica, and Y. Sun, "Scaling up trustless DNN inference with zero-knowledge proofs," 2022, *arXiv:2210.08674*.

[28] F. Chollet. (2015). *Keras*. [Online]. Available: https://keras.io

[29] L. Deng, "The MNIST database of handwritten digit images for machine learning research [Best of the Web]," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.

[30] V. Nikolaenko, S. Ragsdale, J. Bonneau, and D. Boneh, "Powers-of-Tau to the people: Decentralizing setup ceremonies," in *Applied Cryptography and Network Security*. Springer, 2024, pp. 105–134.

[31] ballesterosbr. (2024). *ballesterosbr/privacy_integrity_cs_zkSNARKs_blockchain: 1.0.0 (1.0.0)*. Zenodo, doi: 10.5281/zenodo.10966472. [Online]. Available: https://docs.github.com/en/repositories/archiving-a-github-repository/referencing-and-citing-content

[32] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology–EUROCRYPT*. Berlin, Germany: Springer, 2021, pp. 305–326.

[33] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. (2019). *Permutations Over Lagrange-bases for Oecumenical Noninteractive Arguments of Knowledge*. [Online]. Available: https://eprint.iacr.org/2019/953.pdf

[34] A. Gabizon and Z. J. Williamson. (2021). *FFlonk: A Fast-Fourier Inspired Verifier Efficient Version of PlonK*. [Online]. Available: https://eprint.iacr.org/2021/1167

[35] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: Verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology–CRYPTO*. Berlin, Germany: Springer, 2013, pp. 90–108.

[36] M. Bellés-Muñoz, M. Isabel, J. L. Muñoz-Tapia, A. Rubio, and J. Baylina, "Circom: A circuit description language for building zero-knowledge applications," *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 1–18, Jun. 2022.

[37] (2022). *CircomLib: Library of Basic Circuits for Circom*. [Online]. Available: https://github.com/iden3/circomlib

[38] C. So. (2023). *Circomlib-ML: Circom Circuits Library for Machine Learning*. [Online]. Available: https://github.com/socathie/circomlib-ml

[39] (2023). *Snarkjs: ZkSNARK Implementation in JavaScript & WASM*. [Online]. Available: https://github.com/iden3/snarkjs

[40] R. Song, S. Gao, Y. Song, and B. Xiao, ": A traceable and privacy-preserving data exchange scheme based on non-fungible token and zero-knowledge," in *Proc. IEEE 42nd Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2022, pp. 224–234.

[41] D. A. Luong and J. H. Park, "Privacy-preserving blockchain-based healthcare system for IoT devices using zk-SNARK," *IEEE Access*, vol. 10, pp. 55739–55752, 2022.

[42] M. Finck, "Blockchain and the general data protection regulation: Can distributed ledgers be squared with European data protection law?" Eur. Parliamentary Res. Service (EPRS), Belgium, Europe, Tech. Rep. PE 634.445, 2019. [Online]. Available: https://op.europa.eu/en/publication-detail/-/publication/9b759744-be40-11e9-9d01-01aa75ed71a1

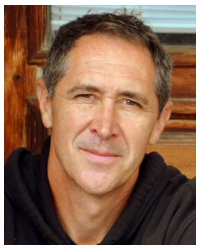
[43] *Blockchain: A Forward-Looking Trade Policy*, document 2018/2085(INI), Committee Int. Trade, 2018.

[44] B. Bunz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 315–334.

[45] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. (2018). *Scalable, Transparent, and Post-quantum Secure Computational Integrity*. [Online]. Available: https://eprint.iacr.org/2018/046

[46] B.-J. Chen, S. Waiwitlikhit, I. Stoica, and D. Kang, "ZKML: An optimizing system for ML inference in zero-knowledge proofs," in *Proc. 19th Eur. Conf. Comput. Syst.*, Apr. 2024, pp. 560–574.

ALBERTO BALLESTEROS-RODRÍGUEZ (Member, IEEE) received the B.S. degree in telecommunication technologies and services engineering from the Polytechnic University of Madrid and the inter-university M.S. degree in information and communication technology security from the Open University of Catalonia, the Autonomous University of Barcelona, and Rovira i Virgili University. He is currently pursuing the Ph.D. degree in information and knowledge engineering with the University of Alcalá. His main research interests include distributed ledger technologies, information security, data protection, cryptography, and decentralized self-sovereign identities.



SALVADOR SÁNCHEZ-ALONSO is currently a Professor with the Department of Computer Science and Statistics, Rey Juan Carlos University, Spain. He has worked and collaborated with several universities, both in undergraduate and graduate degrees and as a Software Engineer at a software solutions company. With a good number of high-impact factor publications in the last ten years, his current research interests include social network analysis applications, web science, and blockchain technologies.

MIGUEL-ÁNGEL SICILIA-URBÁN received the degree in computer science from the Pontifical University of Salamanca and the Ph.D. degree from the Carlos III University of Madrid. Before joining academia, he was an E-Commerce Architect. He is currently a Professor with the Department of Computer Science, University of Alcalá, and the Director of the Postgraduate Program in Blockchain Technology, University of Alcalá. He has developed his research activity in artificial intelligence applied to different areas. He is involved currently in various blockchain projects bridging analytics and decentralization. He is the Editor-in-Chief of *Data Technologies and Applications* (Emerald).