




Toward Data Integrity Architecture for Cloud-Based AI Systems

Elizabeth Nathania Witanto , Yustus Eko Oktian  and Sang-Gon Lee * College of Software Convergence, Dongseo University, Busan 47011, Korea;
d0205114@kowon.dongseo.ac.kr (E.N.W.); d0185099@kowon.dongseo.ac.kr (Y.E.O.)

* Correspondence: nok60@dongseo.ac.kr

Abstract: AI has been implemented in many sectors such as security, health, finance, national defense, etc. However, together with AI's groundbreaking improvement, some people exploit AI to do harmful things. In parallel, there is rapid development in cloud computing technology, introducing a cloud-based AI system. Unfortunately, the vulnerabilities in cloud computing will also affect the security of AI services. We observe that compromising the training data integrity means compromising the results in the AI system itself. From this background, we argue that it is essential to keep the data integrity in AI systems. To achieve our goal, we build a data integrity architecture by following the National Institute of Standards and Technology (NIST) cybersecurity framework guidance. We also utilize blockchain technology and smart contracts as a suitable solution to overcome the integrity issue because of its shared and decentralized ledger. Smart contracts are used to automate policy enforcement, keep track of data integrity, and prevent data forgery. First, we analyze the possible vulnerabilities and attacks in AI and cloud environments. Then we draw out our architecture requirements. The final result is that we present five modules in our proposed architecture that fulfilled NIST framework guidance to ensure continuous data integrity provisioning towards secure AI environments.

Keywords: data integrity; AI systems; cloud computing; blockchain



Citation: Witanto, E.N.; Oktian, Y.E.; Lee, S.-G. Toward Data Integrity Architecture for Cloud-Based AI Systems. *Symmetry* **2022**, *14*, 273. <https://doi.org/10.3390/sym14020273>

Academic Editors: Kuo-Hui Yeh, Chunhua Su and Shi-Cho Cha

Received: 3 December 2021

Accepted: 20 January 2022

Published: 29 January 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Artificial Intelligence (AI) is one of the most disruptive technologies in recent years. AI started in 1956 and has had rapid development since then [1]. In 2016, an AI named AlphaGo defeated the world Go champion over five matches [2]. Then, in 2018, Google launched spin-off Waymo's self-driving taxi service in Phoenix, Arizona [2]. Applications of AI also transform other sectors such as national security, finance, health care, criminal justice, transportation, and smart cities [3]. Meanwhile, we have already seen some people use AI to do harmful things. For example, attacks in self-driving cars with adversarial examples. Changing a few pixels of a stop sign's image might be misclassified as something else by an AI system and could lead to a car accident [4]. Another example is that if an adversary attacking multiple robots controlled by a single AI system on a centralized server could produce fatal failures on a massive scale [4]. AI systems and the knowledge of how to design them can be put toward beneficial and harmful ends [5]. IT systems and applications are susceptible to attacks, and AI is no exception. Therefore, we need to put more effort into securing AI systems as much as we develop them.

The expense of constructing an AI system with excellent precision is not cheap. Training data necessitates a vast number of datasets and a lot of computing power. Fortunately, rapid development in cloud computing technology brings advantages to AI researchers and developers. This advanced improvement introduced a cloud-based AI system. So, instead of building an AI system from scratch, users might employ Cloud Service Provider (CSP) capabilities and resources to train and deploy machine learning applications. They can also access their data anytime from anywhere using their laptop or personal computer. Cloud computing can save users money and time. However, it is more important to trust the system since the vital asset in machine learning (ML) is the data. Cloud-based AI

systems have various drawbacks that users should be aware of. First, data integrity and privacy are in doubt. In cloud computing, users must migrate their assets and resources from their safe zone to the CSP in order to access their services in an unfamiliar security environment. Let assume it is an untrusted environment due to the fact that CSP could be compromised. If this occurs, it may result in data integrity breaches or, worse still, data leakage and loss. Second, cloud computing vulnerabilities will impact the security of AI services, potentially compromising user data or training results. These concerns should be a significant problem that researchers and developers should pay special attention to. Based on the article from [6], they show that there have been instances of data misuse and data loss to hackers, resulting in the user's loss of trust in the CSP. They also provided a survey about the respondents' trust in seven tech giants; the most trusted was Amazon at 28%, followed by Microsoft at 24%, Apple at 22%, Facebook at 19% Google at 13%, Dropbox at 9%, and Instagram at 7%. Even the tech-giants can be compromised, and as a result, their customers' data are at risk. Numerous researchers also conducted a research about trust issue problems and auditing protocol in cloud computing [7–12]. Therefore in this paper, we propose an architecture to tackle the data integrity issues in cloud-based AI systems.

Besides the integrity problem, according to the security triad, a cloud-based AI system is also exposed to vulnerabilities related to confidentiality and availability. First, in terms of confidentiality, the data utilized for training may contain classified or sensitive information. For example, patients' medical data is sensitive information in the health industry. Similarly, the financial industry has much secret information that is not accessible to the general public. If the data are kept in an unencrypted format and stolen by enemies, they can exploit it to their advantage. Secondly, the adversary could poison as much as training data in terms of availability so that the predicting results will be useless. Another case in availability is the expected attack, Denial of Service (DoS), that could make the system unavailable to be accessed. However, those two aspects are out of the scope of our paper. We solely focus on protecting data integrity in cloud-based AI systems by designing the system architecture. We remark that training data is a crucial asset in machine learning to ensure that the AI system acts as intended. When the training data integrity is compromised, the AI outcomes are also compromised. These results might lead to more significant issues, as described before. From this background, we argue that it is essential to keep the data integrity in AI systems. So, we propose an architecture to enhance data integrity and make data falsification more difficult.

In [13], stated that it is vital to build a software architecture that is secure by design. Since the design stage, we must identify the threat model, establish a mitigation strategy, reduce vulnerabilities, and improve security. The National Institute of Standards and Technology (NIST) published the Cybersecurity Framework (CSF) in 2014 to help enterprises construct safe computer systems. It offers a flexible basis that all firms may build upon and shape to meet their own needs [14]. This framework is divided into five parts: identifying capabilities and vulnerabilities, protecting and securing critical infrastructure, detecting security threats as soon as feasible, responding to breaches appropriately, and recovering swiftly and effectively with as little downtime as possible. Until now, there are many companies worldwide that have embraced the use of the framework, including JP Morgan Chase, Microsoft, Boeing, Intel, Bank of England, Nippon Telegraph and Telephone Corporation, and the Ontario Energy Board [15].

The rise of blockchain technology becomes a suitable answer to solve data integrity and trust issues in cloud-based AI systems between users and CSP. Blockchain is a decentralized and distributed ledger system. The history and transaction are recorded permanently. Anyone in the blockchain network can verify and monitor the transactions through a consensus system. In addition, smart contracts in the blockchain allow us to write and run some codes that have deterministic results. Smart contracts make blockchain programmable that could bind the trust between users and CSP. Therefore we utilize blockchain and smart contracts in our proposed system architectures to ensure continuous data integrity provisioning towards secure AI environments.

The contributions of our paper are:

- Proposing a system architecture to ensure continuous data integrity provisioning in cloud-based AI systems based on the NIST cybersecurity framework.
- Providing modules that implement five main points from NIST cybersecurity framework guidance.
- Integrating our architecture based on blockchain and smart contracts to enhance integrity throughout ML pipeline in cloud-based AI system. The identity management and access control module to prevent adversaries from entering the system by impersonating real users. API token management module to prevent unauthorized users calls the API of AI services. Integrity module to keep track of data integrity and prevent data manipulation. Each module is connected to the smart contracts to ensure automation and policy enforcement.

We organize the rest of the paper as follows. Section 2 provides information about existing vulnerabilities and threats in AI and cloud environments. In Section 3, we present other researchers' works that are related to our paper. The proposed architecture described in Section 4 followed by evaluation and discussion in Section 5. Finally, we conclude in Section 6.

2. Background

2.1. AI Environment

There are three potential adversarial interferences in the AI data pipeline [4]:

- Attacks against the data used for training and decision-making.
- Attacks against the classifier in the training environment.
- Attacks against models in the deployment environment.

These three potential attacks might happen inside the machine learning pipeline. Therefore, this section identifies and analyzes possible vulnerabilities and challenges in AI and cloud environments.

First, we will discuss the typical AI's data pipeline and integrity issues in each phase as shown in Figure 1 along with its possible attacks method in Table 1.

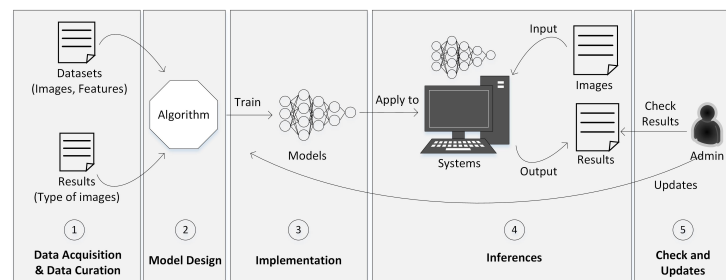


Figure 1. Common AI's data pipelines.

Table 1. ML pipeline's challenges.

Phase	Vulnerabilities and Attacks	Integrity Issues
A1. Data acquisition and curation	Data poisoning [16–19]	Adversary alter the datasets, so the results will be in a way that attacker desires.
A2. Model design	No specific issue related to ML	Generic issues related to choices of hardware/software deployment, or external services.
A3. Implementation	Data Poisoning [16–19], Backdoor attack [20,21]	Such as phase A1, adversary attempts to corrupt the training datasets. Furthermore, this phase could be an entrance to backdoor attack.
A4. Inferences	Adversarial examples [22–26]	Adversary try to manipulating the inputs that cause AI system to misclassify it and behave incorrectly.
A5. Check and updates	Backdoor attack [20,21]	This attack is being triggered if the adversary successfully manipulated the training datasets at training phase.

2.1.1. A1. Data Acquisition and Curation

Large datasets are required before an AI system can learn a model that solves a specific task. In the data acquisition phase, the user prepares the training datasets that are used for expected results. For example, if the system is designed for image classification or specific image detection, their datasets are collections of images. Users can obtain data from numerous resources. It might come from their own or well-known datasets (e.g., CIFAR10, MNIST, etc.). In the data curation phase, the user activities include converting data to specified formats, removing noise, or labeling the data. This step is critical as it determines the accuracy of the results.

Vulnerabilities and attacks—Data poisoning. Occurs when attackers use erroneous and mislabeled data to train an AI model. Pictures of stop signs are labeled as something else so that the algorithm will not recognize them when they appear on the road. These risks show the need for careful control of both training datasets and inputs used to build AI models to ensure the security of AI decision-making processes [4,22,23,27]. Another example is there are various studies of data poisoning in recommender systems [16–19]. The adversary injects various data into the systems, causing the systems to make suggestions based on the adversary's desires.

Issues—Integrity of the training data is taking a crucial part in the ML pipeline as it defines the result's accuracy. Therefore, we need to guarantee that the data is not compromised. For example, in supervised learning, the data must get the correct label to achieve maximum results. Poisoning is a type of threat that can jeopardize data integrity. However, it can be accidentally poisoned when the user inputs the inconsistent data that is unfit for the purpose [28].

2.1.2. A2. Model Design

In this step, users decide what AI algorithms they want to use and set the hyperparameter (e.g., number of nodes and layers, learning rate, bias, activation function).

Issues—There are no specific issues related to ML systems in this phase. However, we should consider the general issues. For instance, it is related to hardware/software deployment or external services choices. Furthermore, there is a possibility of human error that can cause a mistake in algorithms.

2.1.3. A3. Implementation

There are two parts in the implementation phase: training and testing. Training is one of the most critical steps because it establishes the baseline behavior of the systems [28]. In this phase, we train the model with datasets as the input and predict the output with the lowest error as possible [29]. If this stage is compromised, the model will behave incorrectly. The training phase consists of running the algorithms iteratively and adjusting the parameters in each iteration. It will stop when the results converge and give the desired level of accuracy. In the testing part, datasets not used in the training phase are applied to validate the model's performance and ensure the parameters configuration is correct.

Vulnerabilities and attacks—Backdoor attack. This attack was first proposed in [20,21]. The training phase is most likely to become an entrance to backdoor attacks. It relies on data poisoning since the adversary needs to manipulate the training dataset to include examples with the triggers. As a result, it will correlate the trigger with the target class. When receiving the normal images as input during the inference phase, it will behave as expected. However, when it sees the trigger's images, it will return the target class as a result instead of the correct label [28].

Issues—In this phase, the integrity challenge is when the adversary tries to influence, corrupt, or alter the training data itself. It can be done either by inserting adversarial inputs into existing training data or altering training data directly [22]. If successful, it can affect the accuracy results. Furthermore, the training phase can be an entrance gate to the backdoor attack triggered in the last phase.

2.1.4. A4. Inferences

After the model is ready, it will be applied to the applications/systems. The input is in the image format if the system is for image classifications or detection. Then, the systems will classify the image according to the trained model and give the result.

Vulnerabilities and attacks— Adversarial examples are a constant problem and the most well-known attack in the AI environment. Attackers cause the AI system to make mistakes by manipulating inputs. Tiny perturbations to digital images undetectable to the human eye can be sufficient enough to cause AI algorithms to misclassify those images completely [4,22,23]. We present some types of adversarial examples below:

- Fast Gradient Sign Method (FGSM) was introduced by Goodfellow et al. [30] to set up an effective adversarial training. FGSM is fast in generating adversarial examples since it only involves calculating one back-propagation step [31].
- DeepFool. Moosavi-Dezfooli et al. [32] proposed to find minimal L_2 adversarial perturbations on both an affine binary classifier and a general binary differentiable classifier in an iterative manner. Their experiments show that the DeepFool algorithm can generate perturbations more minor than FGSM.
- Jacobian-based Saliency Map Attack (JSMA). Papernot et al. [33] introduce an efficient, targeted attack based on calculating the Jacobian matrix of the score function. It can fool the classifier by restricting the small L_0 perturbations.
- Basic Iterative Method (BIM) was proposed by Kurakin et al. [34] to improve the performance of FGSM by running an iterative optimizer for multiple iterations [35].
- Universal perturbations. All adversarial examples above are working on a specific network. In contrast, universal perturbations can fool the classifier at 'any' image with high probability [23]. Moosavi-Dezfooli et al. [36] successfully find a perturbation that can attack 85.4% of the test samples in the ILSVRC 2012 dataset under a ResNet-152 classifier.

Issues—Integrity challenges here are threatened by various adversarial examples, as explained earlier. The adversary goal is to try to disrupt the model. In this phase, the adversary cannot poison the training data or tamper with the model parameters but still can access the deployed models.

2.1.5. A5. Check and Updates

Admin will check the accuracy and performance of the systems. If necessary, the systems will retrain to update the models.

Vulnerabilities and attacks— Backdoor attack. As mentioned in phase A3, if the adversary successfully plants the trigger, it can be triggered in this phase when the system detects the trigger's images. The system will give the targeted class as the output instead of the correct label.

Issues—The integrity issue can emerge as backdoor attacks that were deployed during training data at the implementation phase being triggered when updating the parameters or the models.

2.2. How to Defend AI System

The aforementioned issues revealed that the AI system is sensitive to data alteration/modification. When poorly classified data is used, the accuracy results will be decreasing, and the system will behave wrongly. Thus, there are various approaches to protect AI systems from such threats. We classify it into two defense mechanisms: AI-algorithm-based and architecture based. The first approach attempts to strengthen the AI system to be more robust against ML attacks. The second approach aims to design an architecture to prevent the adversary from entering the system in the first place. Doing so will reduce the possibility of an adversary modifying the ML datasets.

2.2.1. AI-Algorithm-Based

Many researchers are trying to find a defense mechanism against attacks in AI systems. Hence, the attacker always finds ways to break it. We present several examples of AI-algorithm-based defense mechanisms in Table 2 along with the attacks it is effective against. We divided it into two categories, complete-defense and detection-only. Complete-defense aims to enhance the AI system to classify the label even if the adversary perturbed it. In comparison, detection-only aims to raise a warning on suspicious inputs or even reject them to be processed.

Adversarial training is the most popular defense mechanism. It is effective against adversarial examples in the inference phase. This method works by retraining the model with adversarial examples, but given the correct labels [37]. By doing so, the model will learn to ignore the adversarial examples and increase accuracy results. However, a problem with adversarial training is that the model will only be ‘immune’ to the attacks already trained before [38–40].

Data compression. Image classification datasets are most likely to compress the JPEG images. Authors in [41] show that JPEG compression can work effectively to counter adversarial attacks (e.g., FGSM, DeepFool) and highly reduce their effects. Their technique can remove high-frequency components inside the square block of an image. However, the limitation of this technique is that the larger the compression caused loss of accuracy results, and the smaller the compression insufficiently removes the perturbations.

Randomization. Xie et al. [42] added two-layer of randomization operations to mitigate adversarial effects at the inference phase, random resizing and padding. The first operation resizes the input images to a random size, while the second operation pads zeros around the input images randomly.

Gradient regularizations and adversarial training. The authors [43] show when gradient regularization combined with adversarial training can have good results and robustness against attacks like FGSM and JSMA. However, these methods double a network’s training complexity, which is already prohibitive in many cases.

SafetyNet. This method was proposed in [44] based on a hypothesis that adversarial examples work by producing different patterns of ReLU activations in its late-stage than produced by original data. They used a Radial Basis Function SVM classifier to detect adversarial examples on binary or quaternary codes activation patterns. Then compared its code with training samples using the SVM. In their demonstration, this detector effectively detects adversarial examples generated by FGSM, BIM, and DeepFool.

Convolution filter statistics. This approach aims to detect adversarial examples using statistics on convolutional layer output in CNN-based neural networks. Li and Li [45] designed a cascade classifier based on these statistics and shown its capability to detect more than 85% of adversarial examples.

Perturbation Rectifying Network (PRN). This method aims to defend the AI system against adversarial examples generated using universal perturbations. The authors [46] added additional ‘pre-input’ layers, named PRN, to the targeted model without modifying the model. A separate detector is trained on Discrete Cosine Transform by extracting features of input-output difference of the PRN. First, an image is passed through the PRN to be verified. If it detects a perturbation, use the output of the PRN to classify the image.

GAN-based. There are plenty of works that utilize GAN to improve the robustness against adversarial perturbations. In [47], the authors train a generator and classifier. The generator network generates adversarial perturbations to fool the classifier. At the same time, the classifier is trained to classify correctly both original and adversarial examples generated by the generator. Another GAN-based example is APE-GAN [48], the authors’ trained generator network to cleanse the perturbed image. They also claimed that APE-GAN could be combined with other defenses, such as adversarial training.

Denoising/Feature squeezing. Xu et al. [49] observed that feature input spaces are often unnecessarily large. Therefore, they reduce the available options to the adversary by ‘squeezing’ out unnecessary input features. They added two squeezing methods to

reduce the color bit depth of each pixel in the image and utilize spatial smoothing over the image. If the original and squeezed images show extensively different output, the image is considered an adversarial example. In further work [50] shows that feature-squeezing methods proposed in reference [49] effective to mitigate C&W attack.

Table 2. Examples of AI-Algorithm-based defense mechanisms.

Defense Mechanisms	Effective to	Category
Adversarial training	Adversarial examples	Complete-defense
Data compression	FGSM, DeepFool	Complete-defense
Randomization	Adversarial examples	Complete-defense
Gradient regularizations + adversarial training	FGSM, JSMA	Complete-defense
SafetyNet	FGSM, BIM, DeepFool	Detection-only
Convolution filter statistics	Adversarial examples	Detection-only
Perturbation Rectifying Network (PRN)	Universal perturbations	Complete-defense
GAN-based	Adversarial perturbations	Complete-defense
Denosing/Feature squeezing	Adversarial perturbation to an image	Detection-only

Each of these defenses mechanisms has its strength, weakness, and something to trade-off. Nevertheless, it is worth noting that researchers and developers keep investigating these algorithm-based defense mechanisms to achieve better results.

2.2.2. Architecture-Based

Architecture-based works by building an architecture to prevent the adversary from entering the system in the first place to reduce the possibility of an adversary modifying the ML datasets. Some features can be established inside the architecture to enhance the architecture's robustness against data integrity violation.

- Strengthen the authentication mechanism so the fake user cannot impersonate the real one.
- Enhance authorization mechanism by limiting the user's permissions according to their given role. So, it can prevent malicious users from behaving arbitrarily, and only authorized users can interact with the services.
- Keep track of datasets' integrity through the ML lifecycle phase using a hash algorithm. So, if an attacker alters the datasets, we can compare the hash of the datasets. If the result is different, it means the data has been compromised.
- Logging and monitoring data flow and user activities are necessary in order to detect suspicious actions.

As shown in Table 3, we conclude the merit and weakness of these two defense mechanisms by giving a plus (+) sign to show the phase that the mechanism can cover. Otherwise, we give the minus (−) sign. Since AI-algorithm-based works at the algorithm level, we conclude that this method can cover the ML lifecycle from implementation (A3) to check and updates (A5). Unfortunately, this method could not assure data integrity when users collect and curate the training datasets (A1) and configure the algorithm they want to use (A2). Unlike architecture-based, implementing examples mentioned earlier will make this method cover phase A1–A5.

Nevertheless, in our perspective, it is better if cloud-based AI systems have both defenses since they have their strength and weaknesses points that can complement each other. We will explain more about our proposed idea for the architecture-based method in Section 4.

Table 3. AI-Algorithm-based vs. Architecture-based in ML pipeline.

Phase	AI-Algorithm-Based	Architecture-Based
Data acquisition and curation	–	+
Model design	–	+
Implementation	+	+
Inferences	+	+
Check and updates	+	+

2.3. Cloud Environment

Cloud computing technology brings advantages for its customers. For example, it offers simplicity because it is easy and fast to deploy [51]. Instead of building infrastructure with their resources, users can conveniently utilize cloud service providers' services (CSP). However, in cloud computing, customers outsource a third party to manage their systems and data, so they need a high level of trust in the entity with whom they will be partnering [52]. Besides, they need to be aware of vulnerabilities and threats that follow. When users employ cloud services, they must migrate the resources from their secure perimeter to CSP that they do not know how secure it is. Not to mention existing threats when transmitting the data. In the case of cloud-based AI systems, users need to migrate confidential data such as training data, models, parameters, and configurations to CSP. If these data are compromised, it will lead to unintended systems behavior. Therefore, based on the list of risks from the Open Web Application Security Project (OWASP), we identify potential vulnerabilities and threats in the cloud environment. OWASP presents ten possible risks in the cloud computing [53], and we select some risks that are related to the data integrity to follow our paper scope. We present two categories of challenges in the cloud environment, system access and cloud infrastructure. In the first category, risks that follow are about accountability and data ownership, service, and data integration. Both risks are about the credibility and safety of data when users store and transmit it to the CSP. Since users considered using a third party means they add a new layer of risk [54]. In the second category, risks that follow are about multi-tenancy and infrastructure security. A point that distinguishes cloud computing from on-premises systems lies in its infrastructure. In cloud computing, the tenant is shared the cloud resources and services with other tenants. If there is a failure in the multi-tenancy system, there is a potential risk that other users in the same host mistakenly access other users' data. Further details description is shown in Table 4.

Table 4. Cloud Environment's Challenges.

Category	Risks [53]	Vulnerabilities	Issues
System Access	R1. Accountability and Data Ownership, R6. Service and Data Integration	C1. Account and service hijacking	Adversary could gain access to the cloud resources and services
		C2. Malicious insiders	Leaked important data to adversary
		C3. Lack of authentication and authorization mechanisms	Impersonate real user to compromise the data, resources, and services
Cloud Infrastructure	R7. Multi-tenancy, R9. Infrastructure Security	C4. Insecure API gateway	Exposed to unauthorized data access that could lead to a black-box attack
		C5. Security misconfiguration	Breach in API, account and service hijacking
		C6. Multi-tenancy failure	One tenant can access neighbor's data or resources. Adversary could use it to harm data integrity

2.3.1. System Access

C1. Account and service hijacking. This threat occurs due to phishing, fraud, exploitation of software vulnerabilities, and reuse of credentials and passwords. The results are that attackers can steal user credentials and gain access to the service [55–57].

C2. Malicious insiders. This is a well-known threat for most organizations. The repercussion depends on the level of access owing to the higher the level, the looser the policies. People with high-level access can access confidential data and services. Malicious insiders can cause a considerable impact on the organization, such as stealing confidential data, doing brand damage, financial losses, and productivity losses [55–57].

C3. Lack of authentication and authorization mechanisms. These two mechanisms are the front-line defenders before someone enters the cloud environment. Authentication is a mechanism to identify who wants to enter, authenticated user or the adversary. At the same time, authorization aims to control the access of the data. It determines the access level of each authenticated user has to maintain the system's resource control. Authorization is crucial to ensuring that only authorized users can interact with the data. Lack of both mechanisms could lead to compromising the data in many ways, such as unauthorized access to personal information and cloud services, loss of data privacy, data loss, and leakage [55–57].

2.3.2. Cloud Infrastructure

C4. Insecure API gateway. The client uses API to interact with cloud services. If no security function is implemented, it may expose organizations to various threats, such as anonymous access and authorization, reusable password, and non-encrypted data transmission. Furthermore, it will lead to another threat: account and service hijacking, data loss, and leakage, [55–57]. The insecure API gateway can be vulnerable to black-box attacks in cloud-based AI systems since the adversary could utilize insecure API to query the ML model.

C5. Security misconfiguration. The misconfiguration could happen at the framework, web server, application stack, or browser. For example, using a browser with weak security could lead to security misconfiguration. Furthermore, it could lead to a breach in interface API or account and service hijacking issues. It is important to check the security configuration and use a browser or framework that enforces security policy [55].

C6. Multi-tenancy failure. Multi-tenancy in cloud computing is a crucial component. It enables the cloud vendor to share resources between multiple customers/users. However, multi-tenancy failure could jeopardize their system and especially users' data. For instance, one user can wrongly access a neighbor's data [55]. As a result, the adversary could use this vulnerability to tamper with the data integrity.

3. Related Work

Although cloud-based AI system brings advantages, we should be aware of the possible breach of the systems. Several works from researchers and academia give solutions to overcome cloud and AI environments problems.

Ref. [58] developed a cloud-based machine learning service with a focus on healthcare services. However, the focal point of this work is about the optimization selection of Virtual Machines (VMs) to process medical requests based on cloud environment rather than providing a security architecture for machine learning development and specific to healthcare applications. Álvaro et al. in [59] proposed a cloud-based framework based on DEEP-Hybrid-DataCloud [60]. They create tools for effectively sharing of machine learning models, metadata, and knowledge exchange between clients. However, they do not mention the data integrity assurance inside their framework.

Ref. [61] aims to ensure training data's integrity by proposing a verification scheme called DML-DIV. The authors combine a Provable Data Possession (PDP) sampling auditing algorithm and Discrete Logarithm Problem (DLP) in this scheme. Another paper [11] also proposes a data integrity auditing protocol that relies on the stability of the Computational Diffie Hellman Problem (CDHP) in a Random Oracle Model (ROM). The similarity of these two papers is that in their protocol, they use a third-party auditor (TPA) in the process of data integrity verification/auditing. However, both papers assume that TPA is credible and faithful. If adversaries successfully compromise TPA, the data integrity verification process is at the fence. As a result, trust issues arise between the client, CSP, and TPA.

Therefore, blockchain technology could be a reasonable solution for ensuring data integrity and building trust between clients and CSP instead of TPA.

In [62], the authors proposed an architecture to keep the data integrity for cloud storage services. They created an Integrity Management Service (IMS) as a third-party verifier. The hash algorithm verifies the data integrity exchanged between the client application and the cloud storage service. Zhongshu Gu et al. [63] proposed an idea to secure the input data for image classification service using DeepEnclave. They utilize symmetric cryptography algorithms in their system before and after training data to secure the input data. However, they do not ensure the integrity of the datasets in their proposed idea. Different from our proposed architecture. Our paper proposes an architecture that provides a hash algorithm to ensure no data integrity violation and digital signature to verify the sender and provide non-repudiation. Furthermore, to automate and guarantee those processes work as expected, we use smart contracts on the top of the blockchain network as a trusted Service Level Agreement (SLA).

The proposed idea in [64] has the same goal as our proposal. They proposed a blockchain-based method to keep the integrity of AI learning data in IoT service environments. Their integrity verification process works by comparing the data hash stored in the blockchain. This part is similar to our proposal. However, in our research, we also propose a whole architecture based on the NIST cybersecurity framework to preserve the data integrity through the ML lifecycle. There are identity and access control management in our work to prevent adversaries from entering our system, integrity management to protect and detect data integrity violations. On top of that, we bind the trust between users and CSP using smart contracts.

Another work that was developed to preserve data integrity is an immutable database. This kind of database only permitted insert data queries but restricted update and delete methods. This policy aims to prevent adversaries from altering or deleting data stored in a database. Furthermore, like blockchain, this immutable database relies on cryptographic methods to verification mechanisms. Several examples of immutable database are immudb [65], Amazon Quantum Ledger Database (QLDB) [66], Azure append-only ledger [67]. We present the differences between the immutable databases/append-only databases and blockchain technology in Table 5.

Table 5. Comparison of immutable databases and blockchain technology.

	Immutable/Append-Only Databases	Blockchain
System Access	Centralized	Decentralized
Agreement mechanism	Do not have agreement mechanism	Have consensus protocol as an agreement mechanism
Throughput	High throughput	Depends on the consensus protocol

First, immutable databases are centralized. It means there is still a central authority/control behind the database system. So, immutable databases are suitable in an environment with high trust between parties involved (i.e., central authority and the users). In a cloud-based system, we see a trust issue problem exists between the client and CSP as mentioned previously. Blockchain technology comes with a decentralized system that removes the existence of the intermediary and enables peer-to-peer interactions between nodes, thus enhancing trust.

Second, it is shown that the immutable databases trade the role of consensus protocol to achieving high throughput transactions. Hence, immutable databases are suitable for the system that needs high throughput data while preserving data integrity. However, there is no control over who can append the data to the database due to the absence of a consensus protocol. If the adversary successfully impersonates the user, appends fabricated data in a database as a correct value, the effect will threaten the data integrity. Unlike blockchain with a consensus protocol, before the data can be stored in the blockchain, nodes in the network are required to reach a consensus. Doing so can prevent a node from

behaving arbitrarily. However, the consensus protocol becomes a bottleneck for blockchain networks to achieve high throughput. The number of transactions per second (TPS) varies depending on the consensus algorithm used. For example, the widely used blockchain supports smart contract implementation, such as Ethereum, with Proof-of-Work (PoW) as the consensus algorithm has around 15 TPS in the public network. Fortunately, Ethereum developers are in the progress of migrating their consensus protocol to Proof-of-Stake (PoS) that claimed to have higher throughput because there are shard chains that allow Ethereum to create multiple blocks at the same time [68]. Based on those comparison, we conclude that blockchain is a suitable technology for preserving data integrity in our proposed architecture.

4. Proposed Architecture

4.1. Architecture Requirements

Based on our analysis in Section 2, we draw out several requirements that are needed to build a data integrity architecture for cloud-based AI systems as shown in Table 6. Furthermore, we also present our proposed solutions that meet the requirements and describe them as follows.

- **Identity and Access Control Management.** It is important to identify who accesses our system and what role he/she has before entering and using the cloud services. The cause of adversary can mostly compromise the data integrity because of lack of proper authentication and authorization mechanisms. This requirement aims to cover AI pipeline phase A2; also cloud vulnerabilities points C1 and C3–C5. Phase A2 is related to generic issues specifically associated with cloud services vulnerabilities in our paper. This case could lead to C1 and C3–C5, which explain the lack of identity and access control management. However, a small flaw like this will expose data integrity at risk. *Our solutions*—We use a digital signature to verify the user's identity every time they enter the system and use services.
- **Consistency and completeness.** *Consistency* in our context is related to ML datasets. For instance, there are ten categories in image classification, and each category should have the same amount of images to be trained to achieve accurate results. If the adversary poisons the training data by adding or erasing some training images on one category, the datasets become unbalanced. Then, the decisions made by the AI system would be biased and compromised. *Completeness.* The adversary tries to compromise the training data by altering or tampering with the label of the data. So, the system will give the wrong results with high accuracy. Therefore, it is important to preserve the consistency and completeness of the AI system. This requirement aims to cover AI pipeline phases A1 and A5. The goal is to prevent falsifying data integrity as explained in Section 2.1. *Our solutions*—We use a hash function to ensure no violation of data integrity and record it in the blockchain. In the blockchain, there is no central record-keeping. It is decentralized to all nodes. Each block is chaining with the hash of the previous block. So, changes of one data reflect the hash changes and break this chain. Users or CSP will notice this as a signal of data alteration.
- **Non-repudiation.** One of the well-known threats to a system is malicious insiders. We do not expect that people inside the system will do such a thing. When the actual user becomes an imposter and depends on their access level, they can leak the information to the adversary or modify it undetected. Therefore, this requirement aims to cover AI pipeline phase A2 and cloud vulnerabilities point C2. These two vulnerabilities are related. In C2, we explained malicious insiders that could threaten the cloud environment. It means this vulnerability will also be a threat to the ML pipeline, specifically in phase A2. This person could leak essential information such as the algorithm we used, ML datasets, ML models, or even alter the data. *Our solutions*—User needs to sign the data whenever they change it to verify their identity. Furthermore, there is a module for logging and monitoring unusual user activities in the system.

- **Trusted Service Level Agreement (SLA).** SLA is a contract between customers and the CSP. It binds the trust of both parties. However, there is a possibility that one side breaks the agreement and causes a trust issue. State-of-the-art SLA required another third party as an auditor. Nevertheless, it does not fix the problem but raises another trust issue if the third party is compromised. This requirement aims to cover AI pipeline phases A1–A2, A5. Furthermore, cloud vulnerabilities C1–C6 by enhancing policy enforcement as well. By maintaining policy enforcement through trusted SLA, we can minimize exposed vulnerabilities to an unauthorized party. *Our solutions*—We use smart contracts to bind the trust between users and CSP that can automate the SLA process.

Table 6. Requirements for data integrity architecture for Cloud-based AI Systems.

Requirement	Covered AI Pipeline	Covered Cloud Vulnerabilities	Description	Our Proposed Solutions
Identity and Access Control Management	A2	C1, C3, C4, C5	Prevent adversary to impersonate the real user to login to cloud environment and gain control over the data.	We use a digital signature to verify the user’s identity every time they enter the system and use services.
Consistency and Completeness	A1, A5		Prevent adversary to alter the training data, mislabelled the training data to another value, and disrupt the consistency in datasets.	We use a hash function to keep track to ensure no violation of data integrity and record it in the blockchain.
Non-repudiation	A2	C2	Prevent malicious insiders that has direct access to services and data to leak the information to the adversary or even modify it undetected.	Whenever users changes the data, they need to sign it to verify their identity.
Trusted SLA	A1, A2, A5	C1–C6	Trust issues between user and CSP.	We use smart contracts to bind the trust between users and CSP that can automate the SLA process.

4.2. Architecture Details

We present our proposed architecture that meets the four requirements mentioned above in Figure 2.

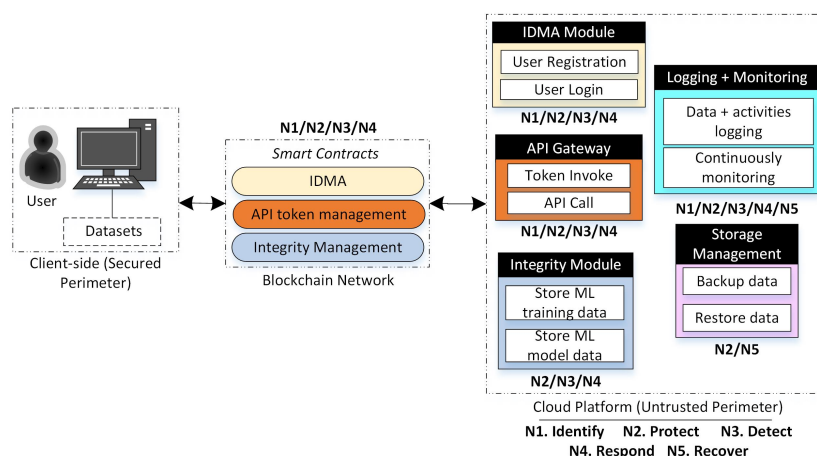


Figure 2. Proposed data integrity architecture in cloud environment.

There is a user on the client-side, smart contracts in the blockchain network, and the cloud platform/CSP (we assume that CSP is not trustable). Users will interact with the cloud platform, such as uploading ML datasets from their secured environment, training the ML system on the cloud platform, and calling the cloud’s API to use AI services. To ensure there is no data integrity violation throughout the ML lifecycle in the cloud

environment, we propose an architecture based on the NIST cybersecurity framework by developing several modules that will collaborate with the blockchain network. Mapping of our modules to NIST framework guidance is depicted in Figure 2 and the analysis of the mapping is discussed in Section 5. There are five modules, Identity Management and Access Control (IDMA), API Gateway, Integrity Module (IM), Logging Monitoring, and Storage Management. In addition, we design six protocols for our first three modules, IDMA, API gateway, and IM.

- The IDMA module is responsible for managing the user registration (Protocol I) and login (Protocol II) process. Each user will be assigned a role limiting their activities and preventing arbitrary behavior. Furthermore, the IDMA module will provide the user with a credential that will be a token for authentication. Every time users want to log in to the cloud system, it is required. In addition, this module collaborates with smart contracts to validate information sent from both parties.
- API Gateway is designed to handle API requests from users wanting to use cloud services. Before utilizing cloud services, users must invoke a token from the API gateway (Protocol III). The API gateway also cooperates with smart contracts to record the user's request for an API token and the token value. Both user and cloud systems can verify the token's legitimacy to smart contracts. The token value is required every time user want to call an API for cloud services to the API gateway (Protocol IV).
- Integrity Module aims to preserve the integrity of ML training data (Protocol V) and model (Protocol VI) by leveraging blockchain. So, the user and cloud system can verify the data integrity to the smart contracts.
- Logging and Monitoring aims to record data flows and user activities. Furthermore, it will continuously monitor the system and raise a warning if there is malicious behavior.
- Storage Management aims to maintain the systems backup data process regularly. The data will be stored in an encrypted form to keep the confidentiality.

4.2.1. Notations

We present the notations that will be used in the proposed architecture details.

1. $Addr_x$ is refers to x 's blockchain address.
2. $SK_x, Addr_x$ are a pair of secret key and public key of x .
3. $Sign_{SK_x}(Y)$ generates a digital signature for data Y using secret key of x .
4. $PKVer_{Addr_x}(M, N)$ is a function to verifies whether blockchain address $Addr_x$ signs data N by generates digital signature M .
5. $E_{PK_x}(Y)$ is a asymmetric encryption of data Y using public key of x .
6. $D_{SK_x}(Y)$ is a asymmetric decryption of data Y using secret key of x .
7. $H(Y)$ is generates a hash of data Y .
8. SC_z is the smart contract for module z . It is resides in the blockchain network and act as trusted SLA to bind the trust between users and CSP.
9. $JWTVer(O, SK_x)$ is a function to verifies the access token O using secret key of x .

4.2.2. Identity Management and Access Control Module (IDMA)

This module will manage authentication and access control by defining who (identity) has what access (role) for which resources [69]. Before accessing the cloud services, the user must prove that they are the authentic user registered in the cloud system. It is used to prevent adversaries from impersonating the actual user. When new users sign up to the cloud system, this module will also ensure they get the roles and permissions. We utilize Role-Based Access Control (RBAC) as a policy enforcement mechanism that limits the system access. In RBAC, we assign a role to every user registered in the cloud system. This role will determine the user privileges about what cloud services they can access. We define three roles that are suitable with the scope of our system architecture as presented in Table 7.

1. *General user*. This role authorizes the users to access only their personal data, such as their training data and model. In addition, they can access the cloud services (e.g., AI

- service) through API calls, but users need to invoke an additional token to the API gateway.
2. *Log admin*. This role authorizes the users to access only the logging data because this role is specified for the user who is in charge of analyzing the system (e.g., data flow, network traffic, etc.).
 3. *System admin*. This role has higher privilege than the previous two. It authorizes a user to access user information and logging data. A system admin will manage the user's role and have the authority to revoke the user's role if the user is compromised.

Table 7. User's role and permission in the proposed protocol.

Role	Personal Data (Training Data & Model)	Logging Data	Detail User Info	Cloud Services
General-user	✓	×	×	△
Log-admin	×	✓	×	×
System-admin	×	✓	✓	×

✓: permitted ×: prohibited △: Additional token required.

The details of the user's role and permission in JSON format are depicted in Figure 3. In the *user-role* there are three values, *role*, *scope* and *access*. The enumeration is as follows.

- Role has three options, 1 is for a general user, 2 for log admin, and 3 for system admin.
- Scope limits what data users can access. This parameter also has three options, 1 for personal data (e.g., training data and model), 2 for logging data, and 3 for user information details.
- Access defines what the user can do to the data. It is filled with 3 bits. If the last digit is one (001 = 1), the user can edit the data. If the second digit is one (010 = 2), the user can delete the data. Lastly, users can view the data if the first digit is one (100 = 4). So, if all three digits are one (111 = 7), users can edit, delete, and view data.

```

{
  "user-info": {
    "user-id": "0xDC25EF3F5B8A186998338A2ADA83795FBA2D695E"
    "user-name": "Alice",
    "password": "98d234db7e91f5ba026a25d0d6f17bc5ee0a347ea2216b0c9de06d43536d49f4",
    "email": "alice@gmail.com",
    "timestamp": "20211005-16.00",
    "user-role": {
      "role": 1,
      "scope": 1,
      "access": 7
    }
  },
  "user-credential": "13532B050D3E5CEB06D8BE17F3280C76ADD84520844E20F553370DD4C4E5A7EA"
}

```

Figure 3. User information details.

We present two use cases protocol for IDMA module design, the user registration and user login process as shown in Figure A1 (Appendix A) and Figure A2 (Appendix B), respectively.

- **Protocol I: User Registration**
Before using the cloud services, new users need to go through the registration process. However, users cannot directly register themselves to the IDMA module. They need to send a transaction that contains their registration request to the smart contract (Steps 1–2). After their transaction is recorded to the blockchain, they can send registration data to the IDMA module (Steps 5–6). If the IDMA module cannot find the user's data in the smart contract when the user sends a registration request, it will reject the request. The IDMA module can easily verify the corresponding user registration request by comparing the hash of registration data sent by user and the blockchain (Step 7). Subsequently, it also verify the message signature. If the user's request is verified, the IDMA module will assign the user's role and generate the user credential (U_{cred}) in Steps 8–9. U_{cred} is required every time users want to log in to the system. IDMA module will also record the U_{cred} to the SC_{idma} (Step 10). Then, the

IDMA module will send U_{cred} and the user's role value to the user together with its signature (Step 13). Users can verify the integrity of U_{cred} from the IDMA module by comparing it with the value from the blockchain. The user will also verify the message signature to ensure the sender is the IDMA module and not an adversary impersonating the IDMA module. By the end of this process, users will know their role and have a user credential in their local storage. Details description of each step is provided in the Appendix A.

- Protocol II: User Login

After a new user successfully registered to the cloud system, he owned a user credential (U_{cred}) in their local storage. Later, he will use this credential every time he log in to the cloud system.

First, the user prepares the login data and send to the IDMA module as shown in Steps 1–2. Upon receiving the login request, the IDMA module begins the authentication and verification process (Step 3). First authentication step is by checking the email, password, and user credential to see whether it matches the recorded data in the local DB. After that, additional authentication process in IDMA module is by comparing U_{cred} value from user and the one existed in blockchain. This second authentication step is required to ensure that the sender is the authentic user. So, IDMA module will query the user credential from SC_{idma} (Steps 4–5). Then, the IDMA module compares U_{cred} value from the user and the blockchain (Step 6). Subsequently, the IDMA module checks the user's role and verify the message signature. This signature verification step is crucial to prove that the sender of R'_5 is the same user who has sent a transaction to the smart contract. Furthermore, it will prevent the adversary from impersonating the user. If return true, send a signed message to the user that login is successful (Steps 7–8). In the last step, the user verifies the reply message to ensure that the sender is IDMA module and not altered on the transmission process. Details description of each step is provided in the Appendix B.

4.2.3. API Gateway

The user uses API in order to call cloud services through a web browser. The API's lack of security will expose the system to unauthorized resource access. Therefore we provide an API key/token generated by the API gateway. It acted as a guard to limit whoever wanted to access cloud resources and services. The user must attach the token as a parameter every time calling the API. However, if CSP is compromised, the adversary could steal the token to access our services undetected. Thus, we build a trustable API gateway by leveraging smart contracts. Furthermore, there is a token expiration time in our system. If a token is expired, the user needs to invoke a new token to the API gateway. For a token generation, we utilize JWT (JSON Web Token). JWT is a group of JSON objects that are base64url-encoded and combined as a string separated by a dot [70]. We present two use cases protocol for API token management, invoke token and API call process as depicted in Figure A3 (Appendix C) and Figure A6 (Appendix D), respectively.

- Protocol III: Invoke Token

Before requesting a token to the API gateway, we assume that the user has already login to the cloud system.

First, the user needs to know a list of services that they can use. So, Steps 1–11 are when users request a list of services to the API gateway. Every time user or API gateway receives a request, they will verify the message first to ensure the message is not from the adversary and the message is not altered in the transmission. Before API gateway can give the list of services, it will check the user's role beforehand to the IDMA module. At the end of Step 11, users receive the list of services they can access from the API gateway and choose what services they want to use. In Steps 12–26, there are some interactions with API Token smart contract (SC_{api}). Similar to Protocol I, the user needs to send a transaction to the SC_{api} that contains a list of services that the user wants to access. If the user sends a services request to the API gateway without

sending a transaction to the smart contract beforehand, the API gateway will reject this request. The services request must be recorded in the smart contract so the API gateway can verify the integrity of the request. After the API gateway verifies the user's service request, it will generate an API token (U_{token}) based on the user's list of services (O) that is shown in Step 18. U_{token} is a required parameter to be sent every time users call the corresponding API services. API gateway also stores this value to the smart contract and will be used later in the token verification process. By the end of Step 26, the user will possess the API token and store it in the local storage. Details description of each step is provided in the Appendix C.

- Protocol IV: API Call

The user needs an API token to access cloud services through the API gateway. We assume that the user already invoked the token and has it in their local storage for this process.

A user needs to prepare the request data to call the API service, as shown in Step 1. The process of API gateway verifies the user's API call is presented in Steps 3–6. API gateway will verify this request by checking the signature (Step 3). Subsequently, it will get the U_{token} value stored in SC_{api} (Step 4). Then, SC_{api} will send a hash of U_{token} in a variable called H_{10} as mentioned in the previous protocol (Step 5). The next step is necessary to prove the data integrity stored in the local DB and the value sent by the user. First, the API gateway gets the expiration time of the token ($Time_{exp}$) from the local DB. Then, computes U_{token} by hashing $AccessToken'$ and $Time_{exp}$. After that, it will compare the value from the smart contract and the hashing computation result. If equal, continue next process; otherwise, reject. It means either $Time_{exp}$ value from DB or $AccessToken'$ from the user was altered. Next, it checks whether the Q' value is a member of O or not. If the token is not expired, and the previous condition is true, API gateway will validate the token using $JWTVer$ function with $AccessToken'$ and SK_{gw} as the inputs (Step 6). If this step is true, the API gateway will relay the request to the corresponding service in the system. Details process of the service is present in Protocol VI. At the end of this protocol, the user receives the corresponding service from the API gateway. Details description of each step is provided in the Appendix D.

- Token revoke. In our system, a token can be revoked because (i) Token expired. (ii) System detects a malicious activity such as user verification failed several times when using the Token.

4.2.4. Integrity Module (IM)

The Integrity Module aims to keep the integrity of training data and the models in cloud-based AI systems. Specifically before stored in cloud storage, before AI service use the data, and when the data-at-rest in the cloud database. Our IM will collaborate with a smart contract as a trusted SLA. Before storing the data in the cloud, the user will store the data's hash to the smart contract. It makes the IM easily verify the data integrity because of the transparency nature of the blockchain. Besides, there are two components in the Integrity Module:

1. The hash algorithm, such as SHA256, ensures no data integrity violation.
2. Digital signature to verify the sender and provide non-repudiation.

We know that it is expensive to record an enormous amount of data in the blockchain in terms of storing data. Training data obviously consume a large amount of storage. The alternatives is utilize a storage that based on distributed hash tables (DHTs). There are several instances, IPFS (InterPlanetary File System) [71], Ethereum Swarm [72], Storj [73], and MaidSafe [74]. In our proposed protocol, we choose IPFS to store the ML training data and model. IPFS offers a decentralized system for storing and accessing files, websites, applications, and data [71]. When someone stores data in IPFS, it will generate a hash of data that points to the file location in return. So, instead of storing an extensive amount of data, the user will store this file location value to the blockchain. The details of our integrity

management protocol when the user wants to store the training data is shown in Figure A7 (Appendix E).

- Protocol V: Store training data
First, the user will store the training data, metadata, and its hash to the IPFS (Steps 1–2). After IPFS receive and store the data, it will return the path of the stored data in the form of hash (H_{ipfs}) in Step 4. Subsequently, the user prepares the hash of these values, user blockchain address, user credential, IPFS hash, and current timestamp. So, the user will store this hash to the IM smart contract (SC_{IM}). After SC_{IM} successfully stores the user's transaction in the blockchain, the user will store the dataset saved in variable R_{17} to the IM module (Step 10). IM module will verify the integrity of the data to the smart contract first before storing it to the local DB. This process is shown in Steps 11–16. IM module compares the hash received from the user and the blockchain. This part is crucial to ensure that the adversary has not corrupted data integrity. If the data is not equal, there is a chance that the adversary has altered data. Then, the IM module will reject the data; otherwise, data integrity is assured, and the process continues. Furthermore, it also checks the user's role in the IDMA module to assure whether the user has permission to store data or not. The IM module will stop the process if the user does not have permission. This part aims to strengthen access control in the cloud system and prevent users' arbitrary behavior that can threaten data integrity. At the end of Step 16, data will be stored in the cloud storage. Furthermore, the IM module will notify the user that the data is stored (Steps 17–19). Details description of each step is provided in the Appendix E.
- Protocol VI: Store ML model
The second protocol of our integrity module is depicted in Figure A8 (Appendix F). It is the case when the user wants to store the ML model after the training process. We assume that:
 - User already past the previous protocol process then uses the AI service to start ML training step and store the model.
 - User receives *param* variable that relayed from API gateway (see Figure A6). Specific to this protocol, *param* contains parameters for training the data in the cloud system, such as the IPFS hash (H_{ipfs}) that shows the location of training data and hyperparameters for training the AI system (e.g., number of nodes and layers, learning rate, bias, weight, activation function).

After the AI service receives H_{ipfs} from the previous step, it will check whether the same hash value recorded in the IM smart contract SC_{IM} (Steps 1–2). Then, it also gets the IPFS hash from the local DB. Finally, it compares three values: IPFS hash relayed from API gateway, smart contract, and local DB. If the comparison does not match, there is a possibility that either data relayed from the API gateway or local DB has been altered. Therefore, it will reject the request. Otherwise, it means there is no alteration of the data, and the data integrity is assured. AI service continues to get the training data from IPFS and start the training process (Steps 4–7). At the end of the training process, there will be an ML model that needs to be stored and preserve the model integrity. Then, the AI service will store the ML model to the IPFS and get the path of the stored ML model in hash form (H_{Mipfs}) (Steps 9–11). To guarantee the data integrity, the AI service will also record ML model IPFS hash to the SC_{IM} (Steps 13–14). Then, when the AI service stores the ML model to the IM module, the IM module can verify the data by comparing the hash from the AI service and the blockchain (Steps 18–19). If the hash is not matched, there is a chance that the data received was altered; then, the IM module will reject the request. Otherwise, the IM module stores data received to local DB. Lastly, the IM module sends a message notification to the AI service that data is successfully stored (Steps 20–22). Details description of each step is provided in the Appendix F.

4.2.5. Logging and Monitoring

This module will record activities from three previous modules, IDMA, API gateway, and IM module. We enforce an access control policy from the IDMA module that only the user with the admin role can access the log.

- First, from the IDMA module, it will monitor the registration and login activities. For example, if a user has a high number of failed authentication attempts in a row, it can be a sign of malicious activity. Then, this module will notify the IDMA to suspend the corresponding user from login for several amounts of time. If the same case happens again from the same user, this module will notify the IDMA to revoke the user credential.
- Second, this module will monitor the invoke token and API call activities from the API gateway. Use case example, if one API token has been called from a different location simultaneously, it is considered suspicious activity. So, this module will notify the API gateway to suspend/revoke the corresponding API token.
- Lastly, from the IM module, it will also monitor the storing training data and ML model activities. Use case example, when storing the ML training data/model, the IM module needs to verify the message signature. However, if there is a high number of failed verification, there is potential that someone alters the message. Then, the logging and monitoring module will notify the IM module to stop the corresponding process.

4.2.6. Storage Management

Storage Management maintains the system to back up the data periodically. In addition, the data will be stored in encrypted form to keep confidentiality. The user needs to configure it once to set the automation backup system. People often underestimate this step, even though backup data is vital to enabling the recovery process after an attack. The data will be restored to replace tainted or lost data with the original one.

5. Evaluation and Discussion

5.1. Off-Chain Computational Complexity

This part presents an evaluation of off-chain computational complexity for six protocols, User Registration, User Login, Invoke Token, API Call, Store ML Training Data, and Store ML Model protocol. We calculate the total cryptographic algorithm and estimation average processing time in each protocol performed by user and cloud modules on Table 8 and Table 9, respectively. In Table 9, we combine protocol for API call and Store ML model because the latter process is the continuation from the former protocol.

Table 8. Number of cryptographic algorithm execution and estimation average time for five protocols performed by user. U = User, AT = Average Time.

Cryptographic Algorithm	AT (ms)	User Registration		User Login		Invoke Token		API Call		Store ML Training Data	
		U	U × AT (ms)	U	U × AT (ms)	U	U × AT (ms)	U	U × AT (ms)	U	U × AT (ms)
Asymmetric-Enc	0.027	1	0.03	1	0.03	2	0.05	1	0.03	1	0.03
Asymmetric-Dec	0.823	0	0.00	0	0.00	1	0.82	0	0.00	1	0.82
Signature	0.041	1	0.04	1	0.04	2	0.08	1	0.04	2	0.08
Verify	0.117	1	0.12	1	0.12	2	0.23	0	0.00	1	0.12
Total		3	0.19	3	0.19	7	1.19	2	0.07	5	1.05

Table 9. Number of cryptographic algorithm execution and estimation average time for six protocols performed by cloud modules. CM = Cloud Module, AT = Average Time.

Cryptographic Algorithm	AT (ms)	User Registration		User Login		Invoke Token		API Call + Store ML Model		Store ML Training Data	
		CM	CM × AT (ms)	CM	CM × AT (ms)	CM	CM × AT (ms)	CM	CM × AT (ms)	CM	CM × AT (ms)
Asymmetric-Enc	0.027	0	0.00	0	0.00	2.00	0.05	1	0.03	1	0.03
Asymmetric-Dec	0.823	1	0.82	1	0.82	3	2.47	2	1.65	1	0.82
Signature	0.041	1	0.04	1	0.04	4	0.16	3	0.12	3	0.12
Verify	0.117	1	0.12	1	0.12	4	0.47	5	0.59	3	0.35
Total		3	0.98	3	0.98	13	3.16	11	2.38	8	1.32

The associated cryptographic algorithms consist of asymmetric encryption, asymmetric decryption, signature generation, and signature verification. To get the average processing time for each cryptographic algorithm, we run a cryptographic benchmarking tool called wolfCrypt [75]. Benchmarking process is run on Ubuntu Linux 16.04 under the Oracle VM VirtualBox environment. The virtual environment was installed on a Windows 10 host machine with Intel Core i5-7200U @2.50 GHz CPU and 4 GB memory. We measure the average processing time of RSA-2048 for encryption and decryption and ECDSA-256 for signing and verification.

Then, to get the estimation average processing time of each protocol, we multiply this number by the total value of each associated algorithm. Finally, we can get the estimation request per second that cloud modules can process. For a bigger picture, we present the estimation numbers of requests per day that cloud modules can process for each protocol in Figure 4. These results show that the highest request is achieved by user registration and user login protocols, with around 88 million requests per day. This value is 3.2 times higher than Invoke API Token, with 27 million requests per day. The other protocols produce low requests per second due to the number of cryptographic algorithms needed to perform. For example, the number of cryptographic algorithms in the Invoke API token protocol is three times higher than user registration and login. However, even the number of requests per day of this protocol is lower, the performance is still acceptable because users only need to invoke API tokens once and last for a period of time.

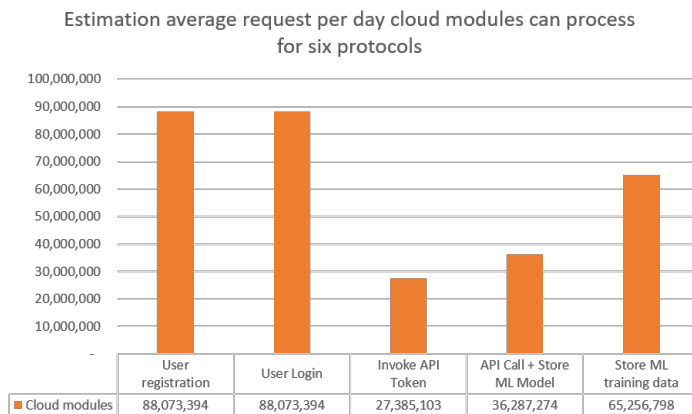


Figure 4. Estimation average request per second that can performed by cloud modules for six protocols.

Figure 5 shows the percentage of the total heavy cryptographic algorithm performed by user and cloud modules. Cloud modules perform the most cryptographic algorithms in our protocol with a value of 66%. By these amount of percentage, the cloud system should be aware of the burden of computational cost since it will computed centrally, while users only perform 34%. Overall, it shows that our protocols serve low computational requirements for the users. Lastly, it is worth noting that our evaluation does not include pre-processing data preparation and the transmission time, only for performing cryptographic algorithms.

Percentage comparison of total cryptographic algorithms performed by user and cloud modules

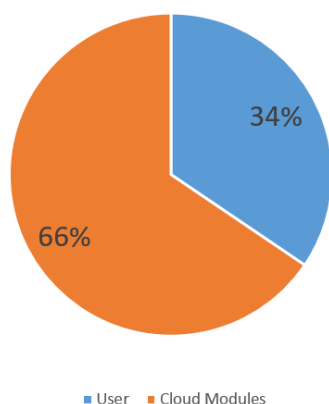


Figure 5. Percentage comparison of total cryptographic algorithms in six protocols performed by user and cloud modules.

5.2. Smart Contracts’ Complexity

This part presents the evaluation of our smart contracts’ complexity by calculating the gas consumption of each function in our smart contracts. We choose Ethereum as our blockchain platform because it supports smart contracts which have been widely used. In Ethereum, gas is a unit of measurement for the amount of computing effort necessary to perform specified functions [76]. Based on this gas, we can calculate the transaction fee (*Tx* fee) that users need to pay every time they call a smart contract function that will change the storage state. The user needs to be aware that the *Tx* fee depends on the smart contract code complexity. The more complex the code, *Tx* fee will become higher. The gas price in Ethereum is measured in Gwei. Then, to calculate the *Tx* fee, we must multiply the gas used by the current gas price. The result in Gwei can be converted to ether (ETH), so we can get the value in USD fiat money. The other important thing is, in Ethereum, the total amount of gas in one block is restraint by the block gas limit [76]. In other words, it determines the number of transactions fitting into a block. Therefore, smart contracts’ complexity evaluation is essential to measure the implementation of the feasibility of our proposed protocol.

Our smart contracts were developed on the same specification as mentioned in Section 5.1. The blockchain network is based on Ganache. Ganache is a rapid Ethereum, and Corda distributed application development [77]. Solidity is used as the programming language to write a smart contract. Furthermore, we use Truffle for the smart contracts development environment. It supports an easy compilation, management, and deployment of the smart contracts [78].

We present our measurement of the gas used for each function in our smart contracts and the *Tx* fee in Table 10.

Table 10. Gas used of writable functions in our smart contracts. * Data calculated from ETH Gas Station [79] on 15 January 2022. The average gas price at that time is 111 Gwei.

Function	Description	Caller	Gas Used	Tx Fee (USD) *
StoreRegData ¹	Store hash of registration data from user	User	45,811	16.72
StoreUcred ¹	Store user credential from IDMA	Cloud	45,833	16.73
StorelistServices ²	Store hash of list services requested by user	User	45,811	16.72
StoreUtoken ²	Store user token from API gateway	Cloud	45,789	16.72
StoreTdataHash ³	Store training data store message hash + IPFS hash of training data	User	79,260	28.94
StoreMdataHash ³	Store model data store message hash + IPFS hash of model	Cloud	79,260	28.94

¹: IDMA smart contract, ²: API token management smart contract, ³: IM smart contract.

We only put functions that change the smart contract’s state because the read/call function runs without gas. Our proposed architecture has three smart contracts: IDMA, API token management, and Integrity Management smart contract. The first smart contract has two functions, StoreRegData and StoreUcred. StoreRegData aims to store payload hash of registration data from user, while StoreUcred is to store user credential value generated by IDMA. Our second smart contract has two functions, StorelistServices and StoreUtoken. The former is a function to store payload hash that contains the list of services requested by the user, and the latter is a function to store user API token value generated by API gateway. The last smart contract also has two functions, StoreTdataHash and StoreMdataHash. The first function aims to store training data payload hash and the IPFS hash of ML training data, while the second has the same function, the data stored is the payload hash of the ML model and its IPFS. From all functions, the highest gas used is 79,260 each for two functions storing the payload hash of store training and model data request, respectively. These functions cost higher because they store two parameters to the smart contract, while the other only store one parameter. The gas consumption of the rest functions is similar, around 45,800 gas. We also present the comparison of gas consumption by user and cloud module for every protocol in Figure 6. It shows that both user and cloud module have the same total gas consumption. The absence of two protocols is because in those protocols do not involve smart contract call function that change the storage state.

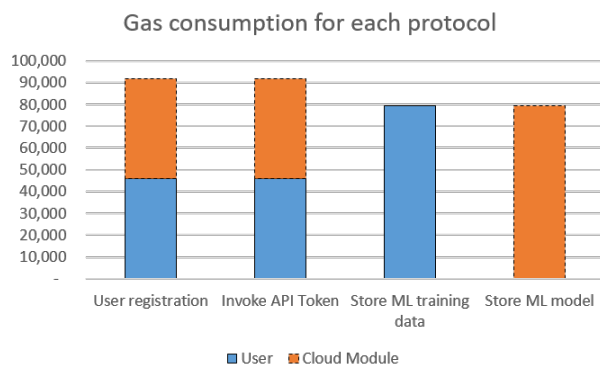


Figure 6. Comparison gas consumption for each protocol.

Additional measurement of blockchain implementation is the throughput. It shows the number of transactions per second (TPS) processed. In Ethereum, total transactions per block are restrained by the block gas limit, as mentioned earlier. So, a higher block gas limit means higher throughput. Another factor that affects the throughput is block creation time. It determines how long one block is produced. So, a longer time means lower throughput. Therefore, based on gas used values, block gas limit, and block creation time, we create two scenarios to measure the estimation throughput for each function. Scenario A is for private Ethereum, and scenario B is for public Ethereum. In scenario A, we refer to truffle development configuration [80], the default block gas limit is 6,721,975, and block creation time is every 1 s. In scenario B, based on this reference [76] block gas limit in public Ethereum is 15 million, while average block creation time is every 15 s [81]. So, we present the estimation TPS of these two scenarios in Figure 7. In the estimation, the first four functions in scenario A can achieve around 147 TPS that 6.7 times faster than scenario B that achieves around 22 TPS. The former scenario can achieve around 85 TPS for the last two functions and the latter 13 TPS. So, if 10,000 users want to run our smart contract function with the highest TPS, it takes 1 min with scenario A and 7 min with scenario B. For the smart contract function with the lowest TPS, the former scenario takes 2 min, and the latter takes 13 min.

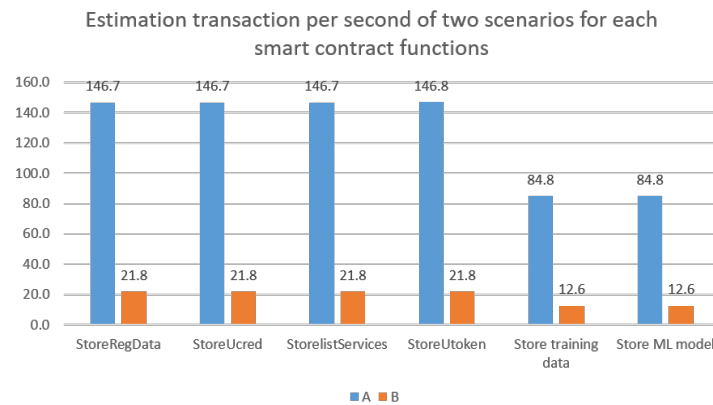


Figure 7. Estimation transaction per second of two scenarios for each smart contract function.

In scenario B, because it is public Ethereum, the callers of the smart contract functions also have to pay a Tx fee that is presented in Table 10. Furthermore, the gas price may vary over time depending on the market situation. Furthermore, when we see Figure 6 at a glance, the total gas consumption for both user and cloud are the same. However, the Tx fees are a different case. Cloud systems should be aware of Tx fees because the more users they have, the more they need to call the smart contract function frequently. In other words, it means higher Tx fees. Fortunately, the frequency of three smart contracts that the cloud needs to call is low. For example, user registration will only be called once every user. Nevertheless, cloud systems should be aware of this problem. However, there is a benefit about the Tx fees. If users/cloud systems successfully validate a transaction in Ethereum, they will also get the Tx fees paid by the caller as an incentives reward [76].

The public Ethereum uses Tx fees systems to secure the network from bad actors spamming the network and to avoid accidental infinite loops or another computational wasting in code [76]. Besides, Ethereum developers keep improving their systems to decrease Tx fees. They introduce Layer 2 Rollup that claimed can decrease the user Tx fees. Rollup performs transaction execution outside the main Ethereum chain but posts transaction data on layer one [82]. Thus, the Tx fee will be cheaper because decreasing the burden of gas consumption for transaction execution. Another alternative is implementing our protocol in a private blockchain where the Tx fee does not exist. However, there are also no incentives given to the transaction validator. Subsequently, there is a chance for bad actors spamming the network because of the absence of Tx fees. Fortunately, Ethereum is now working on developing Ethereum 2.0. At present, Ethereum uses Proof-of-Work as their consensus protocol that becomes a bottleneck for achieving high TPS. However, in Ethereum 2.0, they will migrate their consensus protocol to Proof-of-Stake (PoS) that claims to have higher throughput because there are shard chains that allow Ethereum to create multiple blocks at the same time [68]. Finally, based on this evaluation, our proposed protocols are feasible to be implemented in both private and public Ethereum, with all pros and cons discussed.

5.3. NIST Framework Mapping

NIST created guidance to consider when an organization wants to build a secure architecture to overcome a cyber attack. There are five main points: Identify, Protect, Detect, Respond, and Recover. These points are not a checklist of actions to perform. Nevertheless, it helps organizations manage their cybersecurity risks. It is flexible so that organizations can adjust it based on their need. In our scope, we adopt it to build our proposed architecture that ensures continuous data integrity provisioning in cloud-based AI systems. We present the mapping of our proposed architecture to these five main points based on the expected outcomes of several sub-points from reference [83] and the NIST related sub category [84] in Table 11. We modified some sub-points to fit our proposed architecture.

Table 11. Mapping our proposed architecture to the NIST framework that associated with data integrity.

Sub Points	NIST Expected Outcomes	NIST Related Sub Category	Our Proposed Architecture
N1-1	Document information flows		Logging and monitoring
N1-2	Maintain inventory access	ID.AM-4, ID.GV-1, ID-RA-1, ID-RA-3, ID-RM-2	IDMA, API gateway + smart contracts
N1-3	Establish policies for cybersecurity that include roles and responsibilities		Blockchain smart contracts as trusted SLA
N2-1	Manage access to assets and information	PR-AC-1, PR-AC-4, PR-AC-6-7, PR-DS-1-2, PR-DS-5-6, PR-IP-3-4, R-IP-9, PR-MA-1, PR-PT-1, PR-PT-3	IDMA, API gateway + smart contracts
N2-2	Conduct regular backups		Storage Management
N2-3	Data Integrity Protection		IM + smart contracts
N3-1	Maintain and monitor logs	DE.AE-3, DE-AE-5, DE-CM-3, DE-CM-7, DE-DP-1, DE-DP-4	Logging and monitoring, IDMA, IM, API gateway + smart contracts
N4-1	Maintain Response plan	RS-RP-1, RS-CO-2, RS-AN-1-2, RS-MI-1-2	IDMA, IM, API gateway, Logging and monitoring
N5-1	Maintain recovery plan	RC-RP-1	Logging and monitoring, Storage management

- N1. Identify:** We need to identify and manage resources that enable the organization to focus and prioritize its efforts to finish the goal. In our case, the goal is to ensure no violation of data integrity in cloud-based AI systems. In order to do that, we have identified threats and vulnerabilities in Section 2 along with the integrity issues in AI and cloud environments. We use these data to identify which assets and resources need protection to achieve our goal.

N1-1. Document information flows. It is necessary to understand the information stored in the cloud storage and its flows. Therefore, our proposed architecture's logging and monitoring module will continuously record information, including the data flow.

N1-2. Maintain system access. It is crucial to define and monitor who can access the system because it is most likely the adversary's entrance. In our proposed architecture, there are Identity Management and Access control (IDMA) and API gateway modules connected to two smart contracts for each module, respectively, to guard the system access.

N1-3. Establish policies for cybersecurity that include roles and responsibilities. There are two or more parties that have interaction in a cloud-based AI system. They do not know whether they can trust each other or not. Therefore policies are needed to increase the trust between parties and prevent arbitrary actions. Our proposed architecture utilizes blockchain smart contracts to enforce the policy as a trusted Service Level Agreement (SLA).
- N2. Protect:** Based on our analysis in Section 2, the system access and cloud infrastructure category need protection from the existing vulnerabilities in the cloud environment. Besides, we need to protect the data transmission from the user to the cloud storage to ensure the data integrity of training data in the ML pipeline.

N2-1. Manage access to assets and information. IDMA module and its smart contracts will manage authentication and authorization for each user to log in to the right account and get the corresponding access control. At the same time, the API gateway and its smart contract verify the users' permission when they call queries to request the AI service.

N2-2. Conduct regular backups. Backup data has a significant impact, yet users often neglect maintaining it regularly because they underestimate its purpose. Therefore, in our proposed architecture, there is storage management to automate regular backups in a secure form.

N2-3. Data integrity protection. It is the vital part since it is the goal of our paper,

manage data integrity. Therefore, we append an Integrity Management (IM) module to achieve it in our proposed architecture. This module collaborates with smart contracts to increase the difficulty of falsifying the data and further enhances data integrity. By recording the data in the blockchain, both parties, users and CSP, can verify the received data integrity. It is difficult and expensive for adversaries to alter blockchain data since it is required to overcome more than 50% computing power.

- N3. Detect: In order to detect malicious activities, continuous logging and monitoring are needed. So, for instance, if there are unusual activities in login, writing, or executing the file, the system can send a warning to the user.

N3-1. Maintain and monitor logs. There are four modules in our proposed architecture to ensure the success of anomaly behavior detection, IDMA, IM, API gateway, Logging and monitoring. Furthermore, three smart contracts will cooperate with the first three modules as a trusted SLA. The data flow from the first three modules will be recorded in the logging and monitoring module. Consequently, this module will continuously monitor the user activities whether there is malicious behavior or not. Besides recording and monitoring the data flow, this module also monitors system errors, network traffic, and statistics. By analyzing those data, the system can warn the user of suspicious behavior.

- N4. Respond: The response plan should be prepared prior to the attack happens. For instance, user and system can have their responsibility to mitigate the attack, respectively.

N4-1. Maintain response plan. There are four modules included in the response plan, IDMA, IM, API gateway, Logging and monitoring.

- IDMA secures the entrance point of the system. It authenticates the user that wants to log in to the system. If failed prove their identity several times, IDMA will reject the user from entering the system. This activity will be recorded by logging and monitoring module.
- IM preserves the integrity of training data and model by leveraging smart contracts as a trusted SLA. If training/model data has been compromised in the process, IM will cancel the store data process. This activity will be recorded by logging and monitoring module.
- API gateway guards the system against an unauthorized API call. Each user has a unique token to access a specific cloud's services/resources. If the user accessed the system with the wrong token several times, the system will reject the request and revoke the token used. This activity will be recorded by logging and monitoring module.
- The logging and monitoring module will analyze IDMA, API gateway, and IM data. If it finds suspicious behavior, it will warn the user and limit access to the system. Details explained in Section 4.2.
- N5. Recover: The recovery plan also should be prepared previously and communicated to the members inside organizations. For instance, maintain backup regularly, so if an incident happens, the user already has the last data before it is compromised. *N5-1. Maintain a recovery plan.* For this point to work, sub-points N2-2 needed to be done beforehand. If the backup data exists, the Storage management module could conduct a recovery plan to restore the last backup data after attacks.

In summary, the collaboration of our modules enables the system to identify and protect the data integrity when someone attempts to log in, access files, access API services, and use AI services. In addition, all those activities will be recorded, and the data is used to detect anomaly behavior and execute the response plan. Furthermore, after the incident happens, the system will run a recovery plan by restoring the last backup data from the secured storage. However, our proposed architecture is only complete NIST points that are related to preserving data integrity architecture designing point of view. There are NIST points that are related to such as the organizational procedure, communication, training, and also their stakeholders. These parts are outside our proposed architecture relevance.

5.4. Architecture Analysis

In our proposed architecture, blockchain is our strong point but there are also other technologies that support our architecture. First, we design our architecture by following NIST framework guidance. Second, we utilize cryptographic algorithms such as digital signature to prove the sender identity and hash function to verify the data integrity. Lastly, blockchain and smart contracts are used to record our data, such as user registration and login, invoking API token, ML training data, and model. We record that data in the blockchain because it is a shared and immutable ledger. Once data enters the blockchain, it cannot be erased. It is difficult to tamper with data that has been recorded in the blockchain. In addition, all nodes that join blockchain networks have the same data and can monitor the data. So, a CSP or user can easily ensure data integrity by comparing data hashes with the one that has been stored in the blockchain. Although, there are threats to blockchain-based technology. For example, if adversaries successfully gain more than 50% computing power in a blockchain network, they could have control of the blockchain network itself. However, for now, gaining such computing power is very difficult and costly to achieve. Thus, blockchain is a promising technology for tracing and ensuring continuous data integrity provisioning in the ML lifecycle. We do not use other technology because there is a chance that an adversary can compromise it. Finally, we argue that the synergy between the three points above that built our proposed architecture can ensure data integrity in cloud-based AI systems.

The next part of this section is that we will analyze our proposed architecture by comparing it with three other works [11,59,61,62,64] as shown in Table 12.

- For our first requirement, identity and access control management, only our proposed idea covers this part; the other five are not. Sometimes we diminish the importance of authentication and authorization, even though we know it is essential for the system. The lack of these two functions could expose our data integrity to the edge. Our proposed architecture enhanced our identity management and access control module by utilizing the smart contract.
- For our second requirement, consistency and completeness, four out of five satisfy this requirement. So, most of other works fulfilled this requirement. Although, there are different approach from ours that we already explained in Section 3.

Table 12. Proposed architecture analysis.

Reference	Identity and Access Control Management	Consistency and Completeness	Non-Repudiation	Trusted SLA
[59]	×	×	×	×
[61]	×	✓	✓	×
[11]	×	✓	✓	×
[62]	×	✓	✓	×
[64]	×	✓	×	✓
Ours	✓	✓	✓	✓

- For our third requirement, non-repudiation, three out of five cover this part. Non-repudiation is required to prove who is in charge of some actions inside the systems. Furthermore, it could prevent malicious insiders from leaking or modifying the data because we could track their signatures.
- For the last requirement, trusted SLA, one out of five ideas meet this requirement. In [64], they also use blockchain to bind the trust between client and server. However, they only implement it to cover the second requirement since our trusted SLA will cover the first to third requirements.

Besides the above analysis, our proposed idea has one merit point not implemented in three other works. We built our architecture based on the NIST cybersecurity framework. It means our architecture will be able to identify and detect a threat and respond to protect

the system. Lastly, if an incident happens, our system will be able to maintain a recovery arrangement.

6. Conclusions

We propose an architecture-based defense mechanism that aims to ensure continuous data integrity provisioning towards secure AI environments that follow NIST Cybersecurity Framework guidance. After analyzing the vulnerabilities throughout the AI pipeline and the cloud environment, we drew out several requirements. There are identity and access control management, consistency and completeness, non-repudiation, and trusted SLA. We designed six protocols for three modules, Identity Management and Access Control, API token management, and Integrity management module. Then, we connect those modules to the smart contracts as a trusted SLA to automate and guarantee the processes work as expected. Doing so will increase the difficulty of falsifying the data and further enhance data integrity. Furthermore, there are two other modules, a logging module will log all activities in the system. These records will help to detect unusual behavior. Lastly, the recovery system provides a storage management module that maintains backup data periodically in encrypted form. However, the architecture-based mechanism is designed to prevent adversaries enter the system in the first place before they corrupt the data. So, if an adversary has already entered the system and corrupted the ML datasets, it is not easy to mitigate. Therefore, our future work will combine architecture-based and algorithm-based defense mechanisms to overcome this problem. We also evaluate and analyze our protocols' feasibility by evaluating off-chain and on-chain code. We get the estimated average requests processed by cloud modules for every six protocols from the off-chain analysis. On-chain evaluation measures our smart contracts' complexity by calculating the gas used, transaction fees, and transaction per second. We calculate the throughput/transaction per second (TPS) with two scenarios, private and public Ethereum blockchain, based on gas consumption, block gas limit, and block creation interval. Furthermore, we presented both scenarios' pros and cons.

Author Contributions: Conceptualization, E.N.W. and Y.E.O.; data curation, E.N.W. and Y.E.O.; formal analysis, E.N.W. and Y.E.O.; funding acquisition, S.-G.L.; investigation, E.N.W. and Y.E.O.; methodology, E.N.W. and Y.E.O.; project administration, S.-G.L.; resources, S.-G.L.; software, E.N.W. and Y.E.O.; supervision, S.-G.L.; validation, E.N.W. and Y.E.O.; visualization, E.N.W.; Writing—original draft, E.N.W.; writing—review and editing, E.N.W., Y.E.O. and S.-G.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (grant number: 2018R1D1A1B07047601).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: NIST Cybersecurity Framework guidance publicly available on this link: <http://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf> (accessed on 2 December 2021).

Acknowledgments: We would like to thank the anonymous reviewers for their valuable comments and suggestions that helped us improve the paper.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Appendix A. Protocol I: User Registration

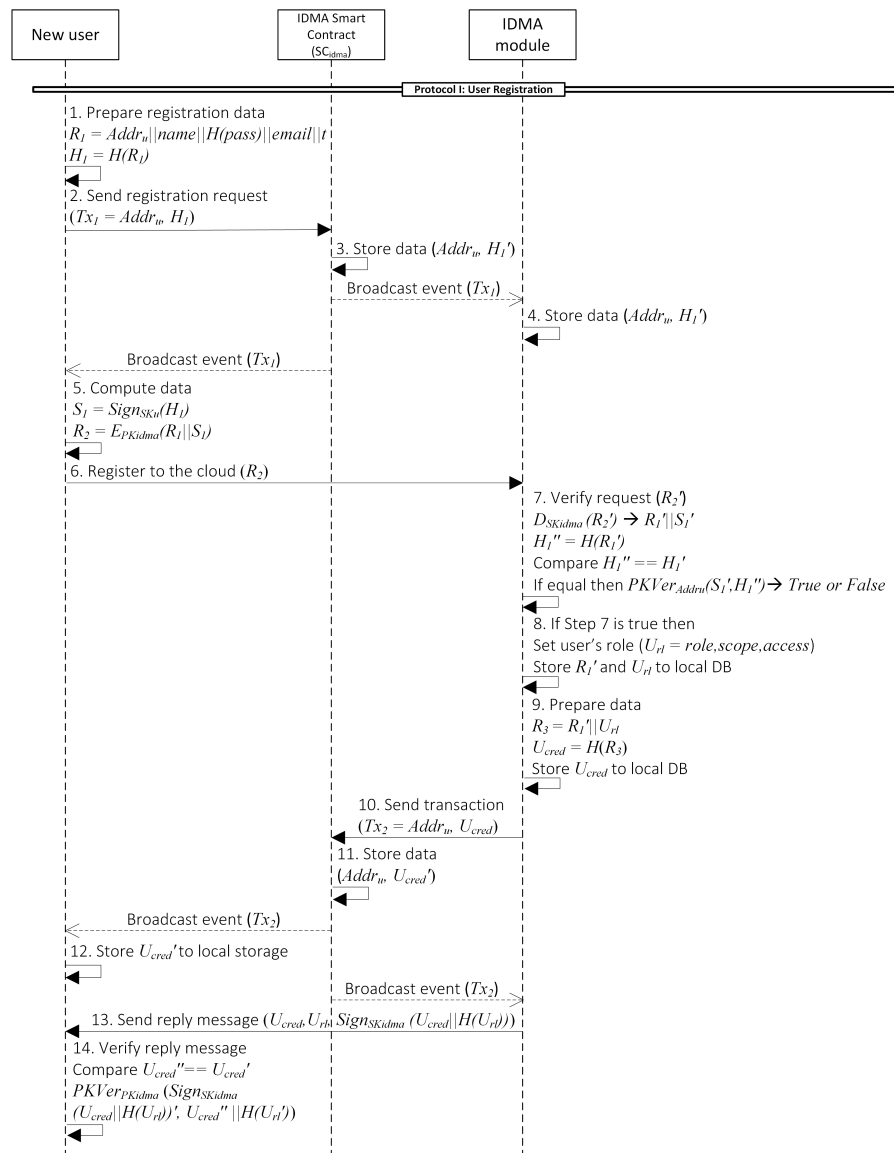


Figure A1. Proposed Protocol for User Registration.

- New user prepares data:
 - t , current timestamp
 - $pass$, user's password
 - $R_1 = Addr_u || name || H(pass) || email || t$
 - $H_1 = H(R_1)$

R_1 contains new users' information that will be registered to the cloud system. Whilst H_1 is the corresponding hash. It plays a vital role in the authentication process.

- New user send a registration request Tx_1 that contains their blockchain address ($Addr_u$) and H_1 to the SC_{idma} .
- SC_{idma} then will record the request to the blockchain with $Addr_u$ as the key and H_1' as the value. We use the variable H_1' instead of H_1 to differentiate the original value from the sender and the current value that was received. There is also one parameter, U_{cred} , that is set as $NULL$ by default in this smart contract. This value stores the authorization token for registered users and will be needed every time they log in to the system. After the transaction is stored in the blockchain, SC_{idma} will broadcast

this event to blockchain network, so every node in the same network will get this broadcast, including the IDMA module and user.

4. After receiving the broadcast event, IDMA module stores the corresponding information ($Addr_u$ and H'_1) to its local database.
5. A new user also receives the broadcast event of request Tx_1 from Step 3, it indicates that their request was successfully inserted into the blockchain. Then, he will compute values as follows.
 - $S_1 = Sign_{SK_u}(H_1)$
 - $R_2 = E_{PK_{idma}}(R_1 || S_1)$

R_2 consists of encrypted form of R_1 and its signature (S_1) and will be sent off-chain.
6. User send R_2 to the IDMA module.
7. After receiving R'_2 , IDMA module begins the verification process.
 - (a) First, it will decrypt R'_2 to get $R'_1 || S'_1$.
 - (b) Compute H''_1 from R'_1 .
 - (c) Compare value H''_1 and H'_1 to ensure no alteration during the transmission process. If equal, then continue to the signature validation process; otherwise, reject it.
 - (d) To validate the signature, IDMA module use function $PKVer_{Addr_u}$ with S'_1 and H''_1 as the input values. This step is crucial to prove that the sender of R'_2 is the same user who has sent a registration request to the smart contract. Furthermore, it will prevent the adversary from impersonating the user.
8. If the verification result is true, the IDMA module will set the role (U_{rl}) for the new user. U_{rl} filled with *role, scope, access* which each value of those parameters has explained before. U_{rl} will determine the user's privileges in the cloud system to prevent arbitrary behavior. Then, it stores both R_1 and U_{rl} to the local database.
9. Next, IDMA module will prepares user credential (U_{cred}) by compute hash of R'_1 and U_{rl} . Subsequently, store U_{cred} to the local database.
 - $R_3 = (R_1 || U_{rl})$
 - $U_{cred} = H(R_3)$
10. IDMA module sending Tx_2 that contains $Addr_u$ and U_{cred} to the SC_{idma} .
11. Upon receiving Tx_2 , SC_{idma} stores U'_{cred} . Then, it will broadcast an event to the blockchain network.
12. New user receiving a broadcast event from Step 11 contains U'_{cred} value. Then, he stores it in the local storage.
13. IDMA module also receives Tx_2 it means the U_{cred} value has successfully been stored in the blockchain. Next, the IDMA module will send U_{cred} and U_{rl} to the user along with its digital signature signed by IDMA module secret key (SK_{idma}).
14. New user will verify the value received. First, compare the U'_{cred} value from IDMA module and from the SC_{idma} . If the value does not equal, there is a chance that someone alters the value when it is on transmission. Then the user will reject this value; otherwise, the user will continue the signature verification process. He validates the signature using $PKVer_{PK_{idma}}$ function with the digital signature and hash of the U'_{cred} and U'_{rl} as the inputs. If the function's output is true, it means the actual sender is the IDMA module, so the user keep the U'_{cred} value received from the smart contract earlier. He will use it every time he wants to log in to the cloud system.

Appendix B. Protocol II: User Login

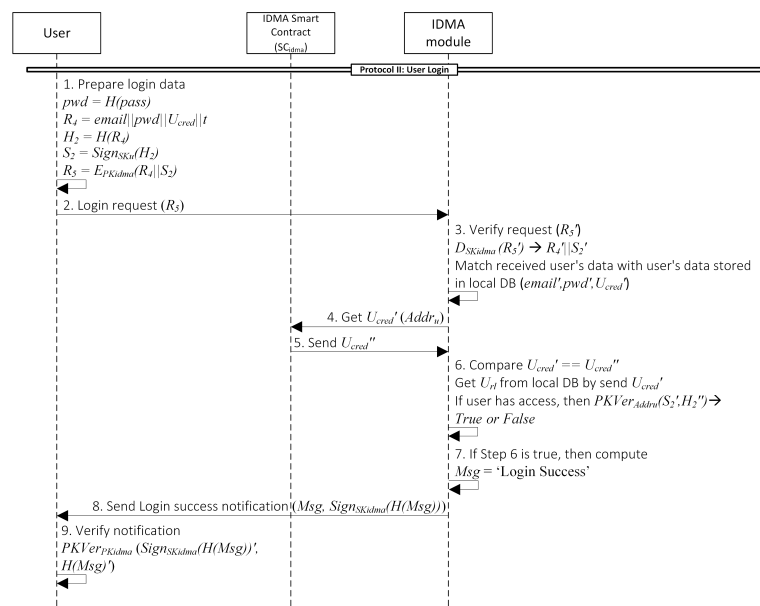


Figure A2. Proposed Protocol for User Login.

1. User prepares data for login:

- $pwd = H(pass)$
- $R_4 = email || pwd || U_{cred} || t$
- $H_2 = H(R_4)$
- $S_2 = Sign_{SK_u}(H_2)$
- $R_5 = E_{PK_{idma}}(R_4 || S_2)$

R_4 contains data for user login along with its corresponding hash H_2 , while R_5 contains encrypted R_4 and S_2 using IDMA module public key (PK_{idma}).

2. Then user submits a login request to the cloud through the IDMA module by sending R_5 to the IDMA module.
3. IDMA module receive R_5' and begins the verification process. First, decrypt R_5' with its secret key (SK_{idma}) to get R_4' and S_2' . Then, it will check two things:
 - Whether the user information already registered in database or not.
 - Compare the user information (email, password, user credential) from the user and from database match or not.

If one of those two conditions is false, reject the user login request; otherwise, continue the process.

4. Additional authentication process in IDMA module is by comparing U_{cred} value from user and the one existed in blockchain. This multi-authentication is required to ensure that the sender is the authentic user. So, IDMA module will query U_{cred} value by user blockchain address $Addr_u$ to the SC_{idma} .
5. SC_{idma} reply the query by sending U_{cred}'' .
6. Then IDMA module will compare U_{cred} value from the user and from the blockchain. We want to ensure that U_{cred} in local DB is not compromised by adversary. If the value is equal, we can assured that the data in local DB is authentic and unaltered. Next, ensure the user's role and privilege in DB. If user has permission, continue to signature validation process; otherwise, reject it. To validate the signature, the IDMA module use function $PKVer_{Addr_u}$ with S_2' and H_2' as the input values. This step is crucial to prove that the sender of R_5' is the same user who has sent a transaction to the smart contract. Furthermore, it will prevent the adversary from impersonating the user.

7. If Step 6 is true, the IDMA module computes a message containing a ‘Login Success’ notification.
8. IDMA module sends a signed message (Msg) to the user to notify that he successfully logged in to the cloud system.
9. User verifies this message by checking the digital signature with the $PKVer_{PKidma}$ function.

Appendix C. Protocol III: Invoke Token

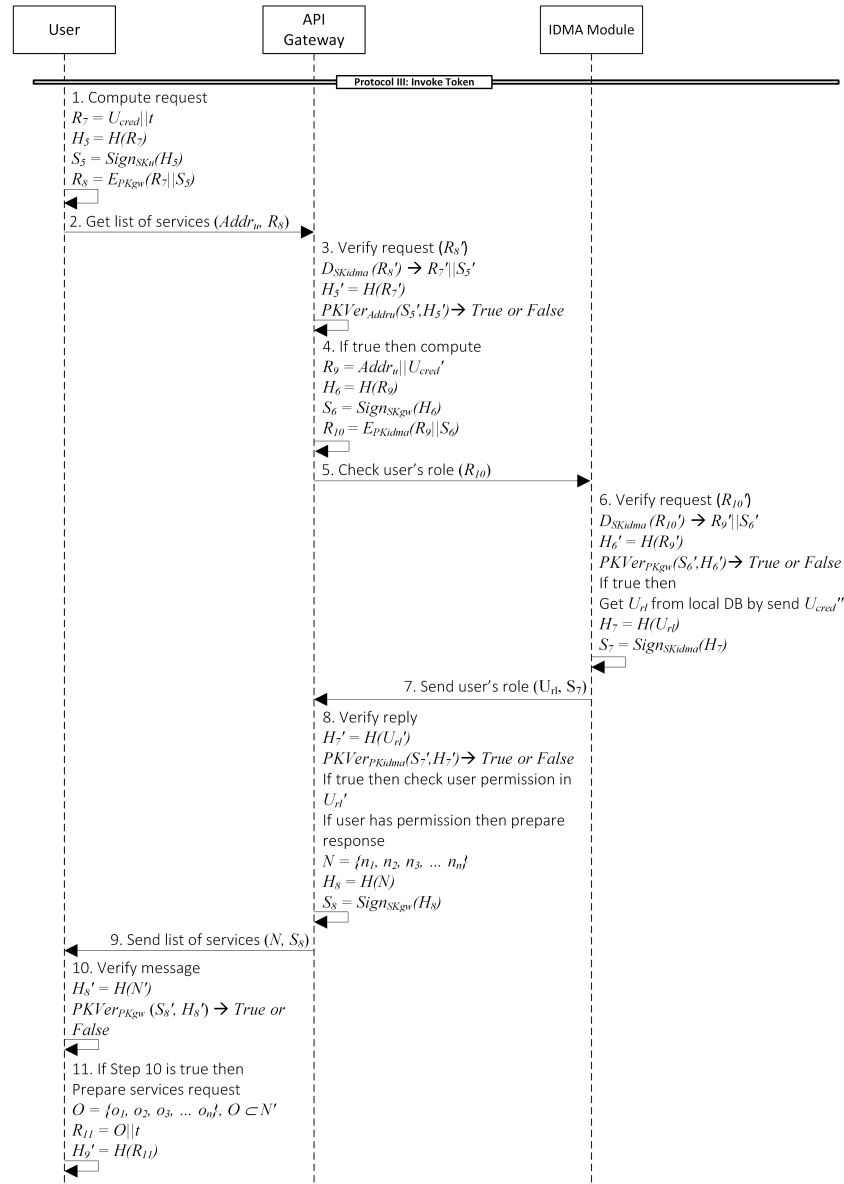


Figure A3. User invoke API token to the API gateway.

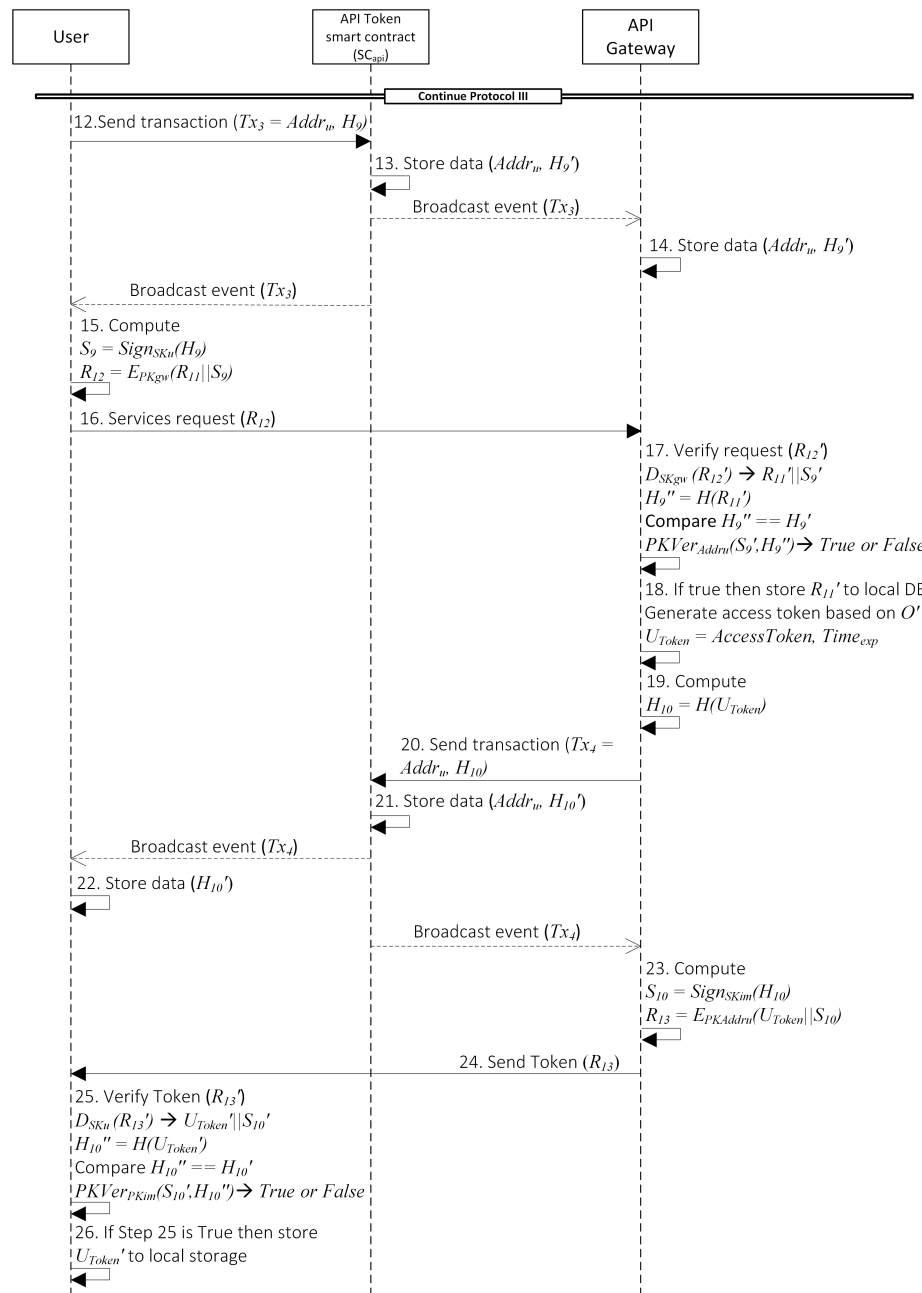


Figure A4. Continue: User invoke API token to the API gateway.

1. User prepares data:

- $R_7 = U_{cred} || t$
- $H_5 = H(R_7)$
- $S_5 = \text{Sign}_{SK_u}(H_5)$
- $R_8 = E_{PK_{gw}}(R_7 || S_5)$

The user needs to include their credential to prove that he has the privilege to request a token to the API gateway. So, the user prepares a message containing U_{cred} , current timestamp (t), and its signature to prove the sender's authenticity. Then, encrypt it because the user will send it off-chain.

2. User gets the list of services they can use by sending $Addr_u$ and R_8 to the API gateway.
3. Upon receiving R_8' , the API gateway begins the verification process.
 - (a) First, it will decrypt R_8' to get R_7' and S_5' .
 - (b) Compute H_5' from R_7' .

- (c) To validate the signature, API gateway use $PKVer_{Addr_u}$ function with S'_5 and H'_5 as the inputs. This step is crucial to prove that the sender of R'_8 is not an adversary impersonating the authentic user.
4. If the previous step is true, then the API gateway computes some values to check the user's permission to the IDMA module.
 - $R_9 = Addr_u || U'_{cred}$
 - $H_6 = H(R_9)$
 - $S_6 = Sign_{SK_{gw}}(H_6)$
 - $R_{10} = E_{PK_{idma}}(R_9 || S_6)$
 5. API gateway sends R_{10} to the IDMA module off-chain.
 6. IDMA module receive R_{10} and begin the verification process.
 - (a) First, it will decrypt R'_{10} to get R'_9 and S'_6 .
 - (b) Compute H'_6 from R'_9 .
 - (c) To validate the signature, the IDMA module use $PKVer_{PK_{gw}}$ function with S'_6 and H'_6 as the input values.

If the signature is valid, then the IDMA module will get the user's role (U_{rl}) from DB by sending U'_{cred} . After that, it will sign the hash U_{rl} with IDMA module's secret key (SK_{idma}).

7. IDMA module reply the user's role request by sending U_{rl} and the signature.
8. Upon receiving the reply message, the API gateway begins to verify it.
 - (a) Compute hash of user's role (H'_7).
 - (b) To validate the signature, API gateway use $PKVer_{PK_{idma}}$ function with S'_7 and H'_7 as the inputs.

If verification success, check the user permission in U'_{rl} . If the user has a permission to request an API token, then API gateway will computes:

- $N = \{n_1, n_2, n_3, \dots, n_n\}$
 - $H_8 = H(N)$
 - $S_8 = Sign_{SK_{gw}}(H_8)$
9. API gateway reply user's request by sending the list of services (N) and its signature (S_8).
 10. After receiving the reply message from the API gateway, the user starts the verification process.
 - (a) Compute the hash of the list of services (H'_8).
 - (b) To validate the signature, API gateway uses $PKVer_{PK_{gw}}$ function with S'_8 and H'_8 as the inputs.
 11. If the previous step is true, then the user prepares the list of services he wants to access.
 - $O = \{o_1, o_2, o_3, \dots, o_n\}$, where $O \subset N'$
 - $R_{11} = O || t$
 - $H_9 = H(R_{11})$
 12. User send Tx_3 that contains $Addr_u$ and H_9 to the API token smart contract (SC_{api}).
 13. SC_{api} then stores H'_9 to the blockchain with $Addr_u$ as the key. Then, it will broadcast an event to the blockchain network.
 14. API gateway receiving the broadcast event of Tx_3 from Step 13 then stores $Addr_u$ and H'_9 to the local database.
 15. User also receiving the broadcast event of Tx_3 from Step 13, indicates that their request was successfully stored in the blockchain. Then, he prepares values:
 - $S_9 = Sign_{SK_u}(H_9)$
 - $R_{12} = E_{PK_{gw}}(R_{11} || S_9)$
- R_{12} is encrypted message that contains list of cloud services that user want to access and its signature.

16. User sends a services request (R_{12}) to the API gateway off-chain.
17. Upon receiving R'_{12} from the user, the API gateway starts the verification process.
 - (a) First, it will decrypt R'_{12} to get $R'_{11}||S'_9$.
 - (b) Compute H''_9 from R'_{11} .
 - (c) Compare value H''_9 and H'_9 to ensure no alteration during the transmission process. If equal, continue to signature validation process; otherwise, reject it.
 - (d) To validate the signature, API gateway use function $PKVer_{Addr_u}$ with S'_9 and H''_9 as the inputs. This step is crucial to prove the sender of R'_{12} is the same user who has sent the payload to the smart contract. Furthermore, it will prevent the adversary from impersonating the user.
18. If the previous step is true, then API gateway stores R'_{11} to the local database. Next, it will generate an *AccessToken* with JWT format based on the list of services requested by the user (O'). This value is combined with $Time_{exp}$ into U_{token} variable. $Time_{exp}$ shows the time that token will be valid. The user cannot use a token that is expired. We present the example of U_{token} in Figure A5.

```

{"U_Token": {
  "access-token":
  "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyLWlkjoiMHhEQzIIRUYzRjVCOEExODY5OTgzMzhBMkFEQTgzNzk1Rk1BMkQ2OTVFlwidHlwZSI6ImFpLXNlcnZpY2UiQ.4drEJLZ7tWQ-leucUIBhEXpImOZFLCU3Hi6vpqgl2e0",
  "token-exp": "20211212-21:21"}}

```

Figure A5. User's access token details.

19. API gateway compute hash of the U_{token} value.
20. Then, API gateway store it to the SC_{api} by sending $Tx_4 = Addr_u, H_{10}$.
21. After SC_{api} receiving this request, it will store H'_{10} to the blockchain with $Addr_u$ as the key. Then, it will broadcast an event to the blockchain network.
22. After user receiving a broadcast event of Tx_4 , he store H'_{10} value to their local storage.
23. API gateway also receiving the broadcast event of Tx_4 . After that, it will computes:
 - $S_{10} = Sign_{SK_{im}}(H_{10})$
 - $R_{13} = E_{Addr_u}(U_{token}||S_{10})$
24. API gateway sends R_{13} that contains the encrypted U_{token} and its signature to the user off-chain.
25. Upon receiving R_{13} from API gateway, user begin the verification process.
 - (a) First, it will decrypt R'_{13} to get $U'_{token}||S'_{10}$.
 - (b) Compute H''_{10} from U'_{token} .
 - (c) Compare value H''_{10} and H'_{10} to ensure that there is no alteration during the transmission process. If equal, continue to signature validation process; otherwise, reject it.
 - (d) To validate the signature, API gateway use function $PKVer_{PK_{im}}$ with S'_{10} and H''_{10} as the input values.
26. If previous step true, then user stores U'_{token} to their local storage.

Appendix D. Protocol IV: API Call

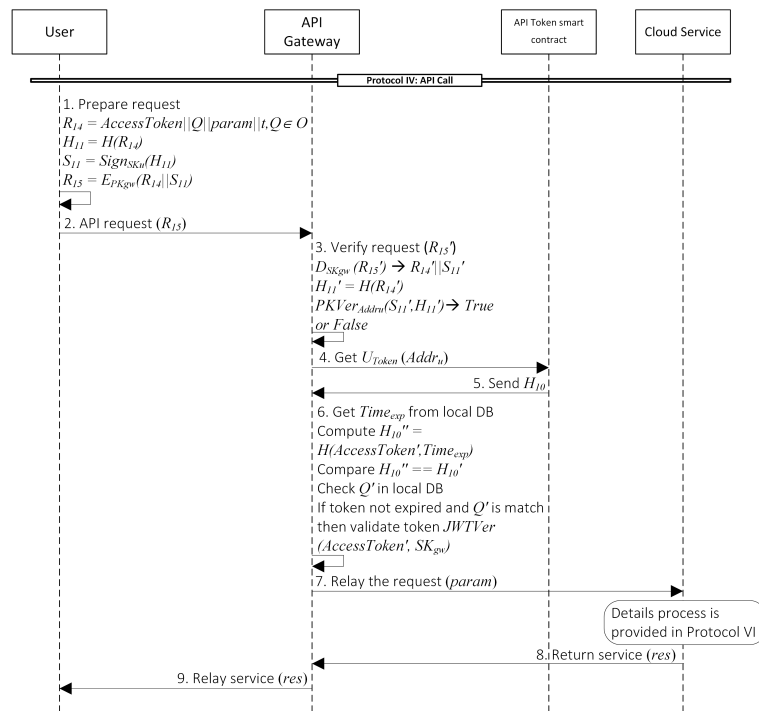


Figure A6. Proposed protocol for API calling.

1. User will prepares data before call the API as follows:

- $R_{14} = AccessToken || Q || param || t$, where $Q \in O$
- $H_{11} = H(R_{14})$
- $S_{11} = Sign_{SK_u}(H_{11})$
- $R_{15} = E_{PK_{gw}}(R_{14} || S_{11})$

R_{15} is an encrypted message that consists of the user's access token ($AccessToken$), the cloud service (Q), specific parameters for the corresponding cloud service ($param$), current timestamp (t), and also the signature (S_{11}).

2. User sends an API request to the API gateway by sending R_{15} .

3. After receiving R_{15} , API gateway begins the verification process.

(a) Decrypt R'_{15} to get $R'_{14} || S'_{11}$.

(b) Compute H'_{11} from R'_{14} .

(c) To validate the signature, API gateway use function $PKVer_{Addr_u}$ with the input S'_{11} and H'_{11} .

4. If the previous step is true, then API gateway will query U_{token} value from smart contract by sending $Addr_u$ as parameter.

5. SC_{api} will send the hash of U_{token} which is H_{10} .

6. API gateway get the expiration time of the token ($Time_{exp}$) first to the local DB. Then, computes U_{token} by hashing $AccessToken'$ and $Time_{exp}$. After that, it will compare the value from the smart contract and from the result of hashing computation. If equal, continue next process; otherwise, reject. It means either $Time_{exp}$ value from DB or $AccessToken'$ from the user was altered. Next, it checks whether the Q' value is a member of O or not. If token is not expired and previous condition is true, API gateway will validate the token using $JWTVer$ function with $AccessToken'$ and SK_{gw} as the inputs.

7. If the verification result is true, then API gateway will relay the user's request to the corresponding cloud service along with the $param$. Detail process in the cloud service is explained in Protocol VI that depicted in Figure A8.

8. The related cloud service will return the result (res) to the API gateway.
9. Lastly, API gateway relay the service back to the user (res).

Appendix E. Protocol V: Store Training Data

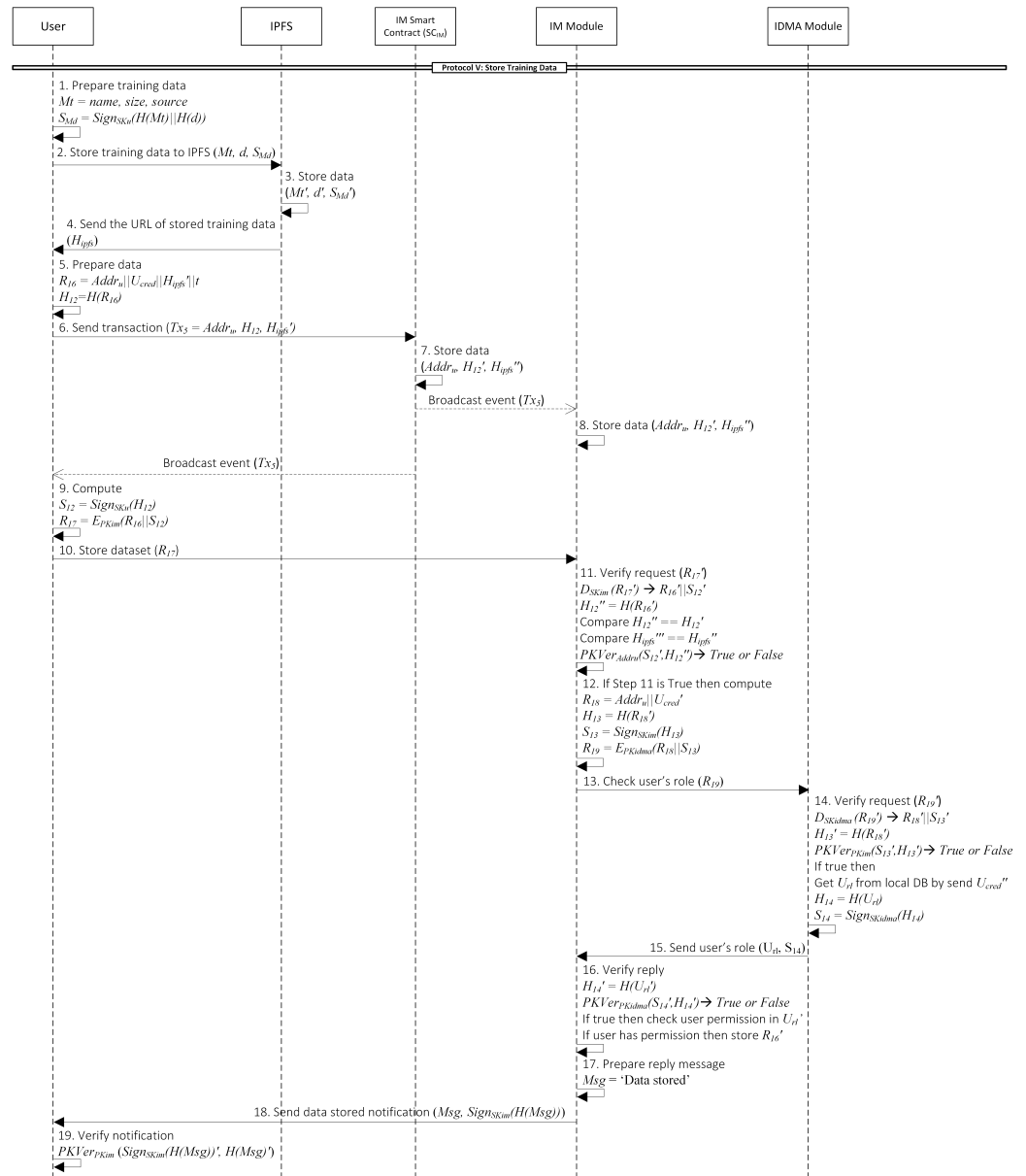


Figure A7. Data integrity management when storing the ML training data.

1. User prepares three data, metadata of training data (Mt), the training data (d), and the signature of two previous data (S_{Md}). Mt contains the name of the dataset, size, and source URL to download the original dataset.
2. User stores Mt, d, S_d to the IPFS.
3. IPFS then stores Mt', d' and S_d' to its storage. Subsequently, it computes a unique fingerprint called content identifier (CID). This CID act as a permanent record and a URL of the files.
4. IPFS sends the CID, that denoted as H_{ipfs} in our protocol, to the user.
5. After that, user computes R_{16} that contains $Addr_u, U_{cred}, H'_{ipfs}, t$ and the hash (H_{12}).
6. Continuously, user sends transaction Tx_5 that contains $Addr_u, H_{12}, H'_{ipfs}$ to the IM smart contract (SC_{IM}).

7. SC_{IM} then store those three values ($Addr_u, H_{12}, H'_{ipfs}$) to the blockchain. After that, it broadcast an event to the blockchain network.
8. Upon receiving a broadcast event of Tx_5 , the IM module stores three values in the local DB.
9. User also receives a broadcast event Tx_5 , which means their request is successfully stored in the blockchain. So, he computes R_{17} that is an encrypted message of R_{16} and its signature.
10. User submits a request to the IM module to store the training data information by sending R_{17} .
11. Next, the IM module will verify the message.
 - (a) Decrypt R'_{17} to get $R'_{16}||S'_{12}$.
 - (b) Compute H''_{12} from R'_{16} .
 - (c) Compare value H''_{12} and H'_{12} to ensure that there is no alteration during the transmission process. If equal, continue to the next process; otherwise, reject it.
 - (d) In this case, the IM module also compares the IPFS hash (H''_{ipfs}) from the user and the one that received from blockchain (H'_{ipfs}). If equal, continue to signature validation process; otherwise, reject it.
 - (e) To validate the signature, API gateway use function $PKVer_{Addr_u}$ with the input S'_{12} and H'_{12} .
12. If the previous step is true, the IM module needs to check the user's role to the IDMA module whether he has the permission or not. Therefore, it will compute R_{19} that is an encrypted message consists of $Addr_u, U'_{cred}$ and its signature.
13. IM module send R_{19} to the IDMA module.
14. IDMA module begins to verify R'_{19} .
 - (a) First, it will decrypt R'_{19} to get R'_{18} and S'_{13} .
 - (b) Compute H'_{13} from R'_{18} .
 - (c) To validate the signature, IDMA module use $PKVer_{PKim}$ function with S'_{13} and H'_{13} as the inputs.

If the signature is valid, then, IDMA module will get the user's role (U_{rl}) from DB by sending U''_{cred} . After that, it will sign the hash U_{rl} with IDMA module's secret key (SK_{idma}).
15. IDMA reply the user's request by sending U_{rl}, S_{14} .
16. Upon receiving the reply message, IM module begin to verify it.
 - (a) Compute hash of user's role (H'_{14}).
 - (b) To validate the signature, API gateway use $PKVer_{PKidma}$ function with S'_{14} and H'_{14} as the inputs.

If signature validation is true, IM checks the user's permission in U'_{rl} . If the user has permission, stores R'_{16} to the local database.
17. After the training data is successfully stored, the IM module computes a message (Msg) that contains a notification to the user about the status of their request.
18. IM module gives notification to the user by sending Msg and its signature.
19. Upon receiving the reply from the IM module, the user verifies the message first. He computes $PKVer_{PKim}$ function with the signature and hash of Msg as the inputs.

Appendix F. Protocol VI: Store ML Model

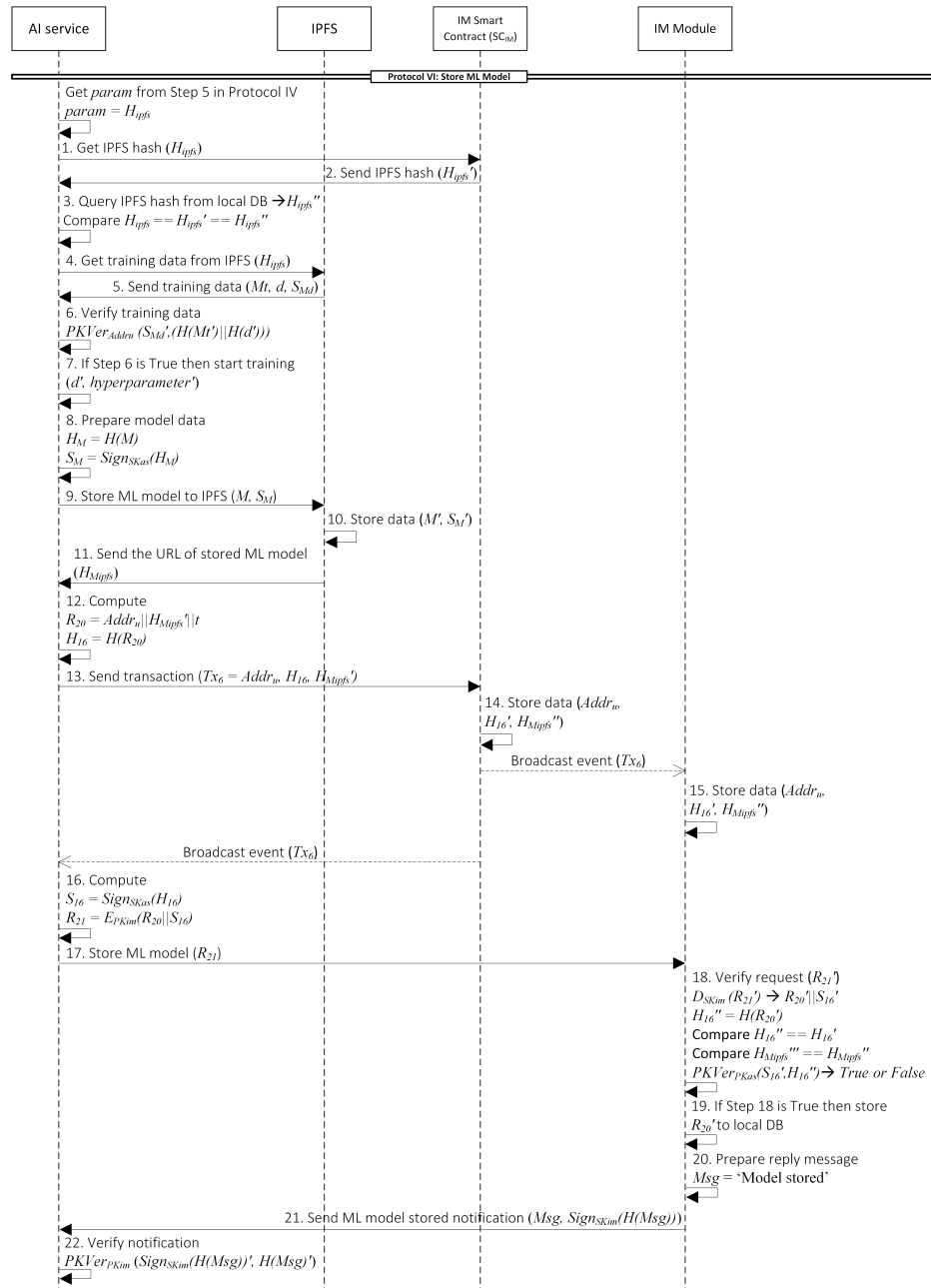


Figure A8. Data integrity management when storing the ML model.

1. AI service receives H_{ipfs} from the user through the API gateway and wants to verify this value. First, it will get the IPFS hash value from SC_{IM} .
2. If the value exist, SC_{IM} sends the corresponding IPFS hash to the AI service.
3. Then, AI service will check is it recorded in DB or not. If recorded, then compare the IPFS hashes values from the user (H_{ipfs}), from the smart contract (H'_{ipfs}), and from the DB (H''_{ipfs}).
4. If the hashes comparison matches, then the AI service gets the training data to IPFS by sending H_{ipfs} as a parameter.
5. IPFS will give the corresponding training data, metadata, and signature to the AI service (Mt, d, S_d).
6. AI service will verify the training data by using $PKVer_{Addru}$ function with S'_d and $H(Mt') || H(d')$ as the inputs.

7. If the previous step is true, then the AI service starts the training process with training data (d') and *hyperparameter* as the inputs.
8. After the training process is finished, the AI service will prepare ML model to be stored (M). ML model also will be stored in IPFS. Therefore, the AI service computes a hash of the model (H_M) and its signature (S_M).
9. AI service sends the ML model (M) and the signature (S_M) to the IPFS.
10. IPFS then store these two values (M' and S'_M).
11. Subsequently, IPFS generates the IPFS hash of the ML model denoted as H_{Mipfs} and sends it to the AI service.
12. Next, AI service will prepare data to be stored in the SC_{IM} which are R_{20} and the corresponding hash H_{16} .
13. AI service sends transaction Tx_6 to the SC_{IM} that contains $Addr_u, H_{16}, H'_{Mipfs}$.
14. Upon receiving Tx_6 from AI service, SC_{IM} then stores those values to the blockchain. Then, it will broadcast an event to the blockchain network.
15. IM module that receiving broadcast event of Tx_6 will also store $Addr_u, H_{16}, H'_{Mipfs}$ to the local DB. These values will later be used for the verification process.
16. AI service also receives a broadcast event of Tx_6 , which means their request is successfully stored in the blockchain. So, it will compute an encrypted message of R_{20} and the corresponding signature (S_{16}).
17. Next, the AI service sends R_{21} to the IM module in order to store the ML model.
18. Then, the IM module will begin to verify R'_{21} .
 - (a) Decrypt R'_{21} to get $R'_{20} || S'_{16}$.
 - (b) Compute H''_{16} from R'_{20} .
 - (c) Compare value H''_{16} and H'_{16} to ensure that there is no alteration during the transmission process. If equal, continue to the next process; otherwise, reject it.
 - (d) Continuously, the IM module also compares the IPFS hashes of the ML model from the AI service (H'''_{Mipfs}) and from the blockchain (H''_{Mipfs}). If equal, continue to the signature validation process; otherwise, reject it.
 - (e) To validate the signature, API gateway use function $PKVer_{PKAs}$ with the input S'_{16} and H'_{16} .
19. If the previous step is true, then the IM module stores the model (R'_{20}) to the local database.
20. Subsequently, it prepares a reply message (Msg) to notify the AI service that the model is successfully stored.
21. IM module sends Msg and its signature to the AI service.
22. Upon receiving the reply from the IM module, the AI service verify the message first. It computes $PKVer_{PKim}$ function with the signature and hash of Msg as the inputs.

References

1. Anyoha, R. The History of Artificial Intelligence. Available online: <https://bit.ly/3x2jid7> (accessed on 3 March 2021).
2. Marr, B. The Most Amazing Artificial Intelligence Milestones So Far. Available online: <https://bit.ly/3oLq2s3> (accessed on 1 December 2020).
3. West, D.M.; Allen, J.R. How Artificial Intelligence Transforming the World. Available online: <https://brook.gs/3CyGQrp> (accessed on 1 December 2020).
4. Herpig, D.S. Securing Artificial Intelligence. 2019; p. 48. Available online: <https://bit.ly/3nMt2F9> (accessed on 15 September 2020).
5. Brundage, M.; Avin, S.; Clark, J.; Toner, H.; Eckersley, P.; Garfinkel, B.; Dafoe, A.; Scharre, P.; Zeitzoff, T.; Filar, B.; et al. The malicious use of artificial intelligence: Forecasting, prevention, and mitigation. *arXiv* **2018**, arXiv:1802.07228.
6. Tom Foremski. Survey: Trust in Tech Giants Is 'Broken'. Available online: <https://zd.net/3mCATVe> (accessed on 18 May 2021).
7. Wang, Y.; Wen, J.; Zhou, W.; Luo, F. A novel dynamic cloud service trust evaluation model in cloud computing. In Proceedings of the 2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science Furthermore, Engineering (TrustCom/BigDataSE), New York, NY, USA, 1–3 August 2018; pp. 10–15.

8. Dave, M.; Saxena, A.B. Loss of trust at IAAS level: Causing factors & mitigation techniques. In Proceedings of the 2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN), Gurgaon, India, 12–14 October 2017; pp. 137–143.
9. Huang, J.; Nicol, D.M. Trust mechanisms for cloud computing. *J. Cloud Comput. Adv. Syst. Appl.* **2013**, *2*, 9. [CrossRef]
10. Yang, Y.; Chen, Y.; Chen, F. A Compressive Integrity Auditing Protocol for Secure Cloud Storage. *IEEE/ACM Trans. Netw.* **2021**, *29*, 1197–1209. [CrossRef]
11. Garg, N.; Bawa, S.; Kumar, N. An efficient data integrity auditing protocol for cloud computing. *Future Gener. Comput. Syst.* **2020**, *109*, 306–316. [CrossRef]
12. Yu, Y.; Xue, L.; Au, M.H.; Susilo, W.; Ni, J.; Zhang, Y.; Vasilakos, A.V.; Shen, J. Cloud data integrity checking with an identity-based auditing mechanism from RSA. *Future Gener. Comput. Syst.* **2016**, *62*, 85–91. [CrossRef]
13. Fonseca, J.; Vieira, M. A Survey on Secure Software Development Lifecycles. Available online: <https://bit.ly/3fOyumr> (accessed on 1 December 2020).
14. 5 Key Changes Made to the NIST Cybersecurity Framework V1.1. Available online: <https://bit.ly/3FtfWD4> (accessed on 1 December 2020).
15. Cybersecurity Framework. Available online: <https://bit.ly/3oM7XKH> (accessed on 1 December 2020).
16. Mobasher, B.; Burke, R.; Bhaumik, R.; Williams, C. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Trans. Internet Technol. (TOIT)* **2007**, *7*, 23-es. [CrossRef]
17. Fang, M.; Yang, G.; Gong, N.Z.; Liu, J. Poisoning attacks to graph-based recommender systems. In Proceedings of the 34th Annual Computer Security Applications Conference, San Juan, PR, USA, 3–7 December 2018; pp. 381–392.
18. Fang, M.; Gong, N.Z.; Liu, J. Influence function based data poisoning attacks to top-n recommender systems. In Proceedings of the Web Conference 2020, Taipei, Taiwan, 20–24 April 2020; pp. 3019–3025.
19. Huang, H.; Mu, J.; Gong, N.Z.; Li, Q.; Liu, B.; Xu, M. Data Poisoning Attacks to Deep Learning Based Recommender Systems. *arXiv* **2021**, arXiv:2101.02644.
20. Gu, T.; Dolan-Gavitt, B.; Garg, S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv* **2017**, arXiv:1708.06733.
21. Gu, T.; Liu, K.; Dolan-Gavitt, B.; Garg, S. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access* **2019**, *7*, 47230–47244. [CrossRef]
22. Papernot, N.; McDaniel, P.; Sinha, A.; Wellman, M.P. Sok: Security and privacy in machine learning. In Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P), London, UK, 24–26 April 2018; pp. 399–414.
23. Akhtar, N.; Mian, A. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access* **2018**, *6*, 14410–14430. [CrossRef]
24. Lu, J.; Sibai, H.; Fabry, E.; Forsyth, D. No need to worry about adversarial examples in object detection in autonomous vehicles. *arXiv* **2017**, arXiv:1707.03501.
25. Brown, T.B.; Mané, D.; Roy, A.; Abadi, M.; Gilmer, J. Adversarial patch. *arXiv* **2017**, arXiv:1712.09665.
26. Iter, D.; Huang, J.; Jermann, M. *Generating Adversarial Examples for Speech Recognition*; Available online: <https://bit.ly/3IMpSJJ> (accessed on 2 February 2021).
27. Ketith Manville. Adversarial Machine Learning 101. Available online: <https://bit.ly/3oFMzXr> (accessed on 18 February 2021).
28. ETSI. Securing AI: Problem Statement. Available online: <https://bit.ly/3cuv1rG> (accessed on 1 April 2021).
29. Hapke, H.; Nelson, C. *Building Machine Learning Pipeline*; O'Reilly Media Inc.: Sebastopol, CA, USA, 2020.
30. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and harnessing adversarial examples. *arXiv* **2014**, arXiv:1412.657.
31. Xu, H.; Ma, Y.; Liu, H.C.; Deb, D.; Liu, H.; Tang, J.L.; Jain, A.K. Adversarial attacks and defenses in images, graphs and text: A review. *Int. J. Autom. Comput.* **2020**, *17*, 151–178. [CrossRef]
32. Moosavi-Dezfooli, S.M.; Fawzi, A.; Frossard, P. Deepfool: A simple and accurate method to fool deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 2574–2582.
33. Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z.B.; Swami, A. The limitations of deep learning in adversarial settings. In Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P), Saarbruecken, Germany, 21–24 March 2016; pp. 372–387.
34. Kurakin, A.; Goodfellow, I.; Bengio, S. Adversarial Examples in the Physical World. 2017. Available online: <https://bit.ly/3qSB0yf> (accessed on 17 March 2021).
35. Ren, K.; Zheng, T.; Qin, Z.; Liu, X. Adversarial attacks and defenses in deep learning. *Engineering* **2020**, *6*, 346–360. [CrossRef]
36. Moosavi-Dezfooli, S.M.; Fawzi, A.; Fawzi, O.; Frossard, P. Universal adversarial perturbations. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1765–1773.
37. Tramèr, F.; Kurakin, A.; Papernot, N.; Goodfellow, I.; Boneh, D.; McDaniel, P. Ensemble adversarial training: Attacks and defenses. *arXiv* **2017**, arXiv:1705.07204.
38. Na, T.; Ko, J.H.; Mukhopadhyay, S. Cascade adversarial machine learning regularized with a unified embedding. *arXiv* **2017**, arXiv:1708.02582.
39. Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv* **2017**, arXiv:1706.06083.
40. Farnia, F.; Zhang, J.M.; Tse, D. Generalizable adversarial training via spectral normalization. *arXiv* **2018**, arXiv:1811.07457.

41. Das, N.; Shanbhogue, M.; Chen, S.T.; Hohman, F.; Chen, L.; Kounavis, M.E.; Chau, D.H. Keeping the bad guys out: Protecting and vaccinating deep learning with jpeg compression. *arXiv* **2017**, arXiv:1705.02900.
42. Xie, C.; Wang, J.; Zhang, Z.; Ren, Z.; Yuille, A. Mitigating adversarial effects through randomization. *arXiv* **2017**, arXiv:1711.01991.
43. Ross, A.S.; Doshi-Velez, F. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018. Available online: <https://bit.ly/341U9oq> (accessed on 17 March 2021).
44. Lu, J.; Issaranoon, T.; Forsyth, D. Safetynet: Detecting and rejecting adversarial examples robustly. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 446–454.
45. Li, X.; Li, F. Adversarial examples detection in deep networks with convolutional filter statistics. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 5764–5772.
46. Akhtar, N.; Liu, J.; Mian, A. Defense against universal adversarial perturbations. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 3389–3398.
47. Lee, H.; Han, S.; Lee, J. Generative adversarial trainer: Defense to adversarial perturbations with gan. *arXiv* **2017**, arXiv:1705.03387.
48. Shen, S.; Jin, G.; Gao, K.; Zhang, Y. Ape-gan: Adversarial perturbation elimination with gan. *arXiv* **2017**, arXiv:1707.05474.
49. Xu, W.; Evans, D.; Qi, Y. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv* **2017**, arXiv:1704.01155.
50. Xu, W.; Evans, D.; Qi, Y. Feature squeezing mitigates and detects carlini/wagner adversarial examples. *arXiv* **2017**, arXiv:1705.10686.
51. Alex Stamos. Cloud Computing Security. Available online: <https://bit.ly/3Euu5Px> (accessed on 23 December 2021).
52. Ezhil Arasan Babaraj. Cloud Security—An Overview. Available online: <https://bit.ly/3EvjPX9> (accessed on 23 December 2021).
53. Ludovic Petit. OWASP Top 10 Cloud Security Risks. Available online: <https://bit.ly/3mVx1in> (accessed on 23 December 2021).
54. Hitachi Systems Security. The Top 10 OWASP Cloud Security Risks. Available online: <https://bit.ly/3qkkNR6> (accessed on 23 December 2021).
55. Khalil, I.M.; Khreishah, A.; Azeem, M. Cloud computing security: A survey. *Computers* **2014**, *3*, 1–35. [[CrossRef](#)]
56. Munir, K.; Palaniappan, S. Framework for secure cloud computing. *Adv. Int. J. Cloud Comput. Serv. Archit. (IJCCSA)* **2013**, *3*, 21–35. [[CrossRef](#)]
57. Pandey, S.; Farik, M. Cloud computing security: Latest issues & countermeasures. *Int. J. Sci.* **2015**, *4*, 3–6.
58. Abdelaziz, A.; Elhoseny, M.; Salama, A.S.; Riad, A. A machine learning model for improving healthcare services on cloud computing environment. *Measurement* **2018**, *119*, 117–128. [[CrossRef](#)]
59. García, Á.L.; De Lucas, J.M.; Antonacci, M.; Zu Castell, W.; David, M.; Hardt, M.; Iglesias, L.L.; Moltó, G.; Plociennik, M.; Tran, V.; et al. A cloud-based framework for machine learning workloads and applications. *IEEE Access* **2020**, *8*, 18681–18692. [[CrossRef](#)]
60. DEEP Hybrid Data Cloud. Available online: <https://bit.ly/3FsfQM3> (accessed on 1 March 2021).
61. Zhao, X.P.; Jiang, R. Distributed Machine Learning Oriented Data Integrity Verification Scheme in Cloud Computing Environment. *IEEE Access* **2020**, *8*, 26372–26384. [[CrossRef](#)]
62. Nepal, S.; Chen, S.; Yao, J.; Thilakanathan, D. DIaaS: Data integrity as a service in the cloud. In Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing, Washington, DC, USA, 4–9 July 2011; pp. 308–315.
63. Gu, Z.; Huang, H.; Zhang, J.; Su, D.; Lamba, A.; Pendarakis, D.; Molloy, I. Securing input data of deep learning inference systems via partitioned enclave execution. *arXiv* **2018**, arXiv:1807.00969.
64. Kim, J.; Park, N. Blockchain-Based Data-Preserving AI Learning Environment Model for AI Cybersecurity Systems in IoT Service Environments. *Appl. Sci.* **2020**, *10*, 4718. [[CrossRef](#)]
65. CodeNotary. Immudb Introduction. Available online: <https://bit.ly/3JYdzuW> (accessed on 4 January 2022).
66. AWS. Amazon Quantum Ledger Database (QLDB). Available online: <https://go.aws/3f9sFzw> (accessed on 5 January 2022).
67. Microsoft. Azure SQL Database Append-Only Ledger Tables. Available online: <https://bit.ly/3r7WfKV> (accessed on 5 January 2022).
68. Ethereum. Ethereum Proof-of-Stake. Available online: <https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/> (accessed on 9 January 2022).
69. How IAM Works. Available online: <https://bit.ly/3HDYm1d> (accessed on 1 April 2021).
70. Introduction to JSON Web Tokens. Available online: <https://bit.ly/30BJCiN> (accessed on 1 October 2021).
71. What Is IPFS. Available online: <https://bit.ly/3HEPAQj> (accessed on 1 October 2021).
72. Swarm. Ethereum Swarm. Available online: <https://bit.ly/3f9BZDw> (accessed on 6 January 2022).
73. Storj. Available online: <https://bit.ly/3zIR7i0> (accessed on 6 January 2022).
74. MaidSafe. Intro to MaidSafe. Available online: <https://bit.ly/33lSvOe> (accessed on 6 January 2022).
75. wolfSSL. wolfCrypt Benchmarking. Available online: <https://www.wolfssl.com/docs/benchmarks/> (accessed on 14 January 2022).
76. Paul Wackerow. Gas and Fees. Available online: <https://bit.ly/3zCzS4D> (accessed on 6 January 2022).
77. Ganache Overview. Available online: <https://www.trufflesuite.com/docs/ganache/overview> (accessed on 20 February 2020).
78. Truffle Overview. Available online: <https://bit.ly/3q03i9O> (accessed on 7 November 2021).
79. Concourse Open Community. ETH Gas Station. Available online: <https://bit.ly/3zySXVm> (accessed on 15 January 2022).
80. Truffle Suite. Truffle Suite Configuration. Available online: <https://bit.ly/3qRHsWr> (accessed on 15 January 2022).
81. Ethereum. Blocks Ethereum. Available online: <https://ethereum.org/en/developers/docs/blocks/> (accessed on 9 January 2022).
82. Paul Wackerow. Layer 2 Rollup. Available online: <https://bit.ly/3HNXOFF> (accessed on 6 January 2022).

-
83. NIST. Getting Started with the NIST Cybersecurity Framework: A Quick Start Guide. Available online: <https://bit.ly/3oLeh50> (accessed on 30 July 2021).
 84. National Institute of Standards and Technology. *Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1*; Technical Report NIST CSWP 04162018; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2018. [[CrossRef](#)]