

# Trusted Compute Units: A Framework for Chained Verifiable Computations

Fernando Castillo, Jonathan Heiss, Sebastian Werner, Stefan Tai

*Information Systems Engineering*

*Technische Universität Berlin*

Berlin, Germany

{fc,jh,sw,st}@ise.tu-berlin.de

**Abstract**—Blockchain and distributed ledger technologies (DLTs) facilitate decentralized computations across trust boundaries. However, ensuring complex computations with low gas fees and confidentiality remains challenging. Recent advances in Confidential Computing—leveraging hardware-based Trusted Execution Environments (TEEs)—and Proof-carrying Data—employing cryptographic Zero-Knowledge Virtual Machines (zkVMs)—hold promise for secure, privacy-preserving off-chain and layer-2 computations. On the other side, a homogeneous reliance on a single technology, such as TEEs or zkVMs, is impractical for decentralized environments with heterogeneous computational requirements.

This paper introduces the Trusted Compute Unit (TCU), a unifying framework that enables composable and interoperable verifiable computations across heterogeneous technologies. Our approach allows decentralized applications (dApps) to flexibly offload complex computations to TCUs, obtaining proof of correctness. These proofs can be anchored on-chain for automated dApps interactions, while ensuring confidentiality of input data, and integrity of output data.

We demonstrate how TCUs can support a prominent blockchain use case, such as federated learning. By enabling secure, off-chain interactions without incurring on-chain confirmation delays or gas fees, TCUs significantly improve system performance and scalability. Experimental insights and performance evaluations confirm the feasibility and practicality of this unified approach, advancing the state of the art in verifiable off-chain services for the blockchain ecosystem.

**Index Terms**—Service Workflow, Data Sharing, Verifiable Computation, Trusted Execution, Zero-knowledge, Blockchain.

## I. INTRODUCTION

In recent years, decentralized applications (dApps) and blockchain-based solutions have begun proposing off-chain computation mechanisms to address the high costs and privacy challenges of fully on-chain execution. For instance, a hospital might outsource AI-based diagnostic analytics to a specialized service that collaborates with multiple healthcare providers—similar to a federated learning (FL) setup. In this scenario, the hospital must trust that the external computation is performed correctly and confidentially, without exposing sensitive patient data on the blockchain or to unauthorized parties.

Such cross-organizational workflows often extend beyond a single “provider–consumer” pair, forming a directed acyclic graph (DAG) of dependent computations, thereby forming a

cross-organizational workflow where one organization’s output can become another’s input. In these contexts, data integrity and the correctness of the underlying service computations are critical. Not only do organizations rely on accurate external data for sound decision-making, but they must also comply with stakeholder and regulatory demands to demonstrate the correctness of their claims. For example, the EU AI Act requires organizations to verify that their machine-learning applications conform to relevant regulations [1].

However, achieving verifiability in computations across organizations is inherently challenging [2] due to the need to protect sensitive inputs while proving the correctness of computations. A naïve approach, such as re-executing computations with original inputs, introduces confidentiality risks, and verifying every computation on-chain translates to more gas fees and confirmation delays. Sensitive internal inputs, such as proprietary business data or personally identifiable information, cannot be exposed without violating confidentiality and security requirements [3]. Moreover, prior research on verifiable computation<sup>1</sup> has shown that no single technique or cryptographic construct can fully meet all desired properties of functionality and efficiency [4]–[6]. Instead, combining heterogeneous methods—such as zero-knowledge proofs and trusted hardware attestation [7]—remains a challenging yet promising direction, particularly in real-world scenarios where each organization maintains its own distinct technological stack.

Addressing these problems, we propose the *Trusted Compute Unit (TCU)* as a key component to enable interoperable and composable trusted computations across organizations. At its core, TCU builds upon containers that wrap around services and execute data processing tasks within verifiable computation environments like zero-knowledge virtual machines (zkVM), or trusted execution environments (TEE), to enable trusted chained computations. A Blockchain-based Program Registry is applied to enable decentralized and cross-organizational management and traceability of the TCU’s in- and outputs.

<sup>1</sup>Some authors reserve ‘verifiable computation’ strictly for cryptographic proof systems. In this paper, we include TEEs under the broader category of trusted or verifiable techniques, while noting that TEEs rely on hardware trust assumptions.

In this paper, we make the following individual contributions:

- 1) We present the TCU as a unifying framework that enriches chained off-chain computations with verifiable computation and blockchain technology. TCUs are modular and combinable components encapsulating service logic for verifiability and a foundational infrastructure. A blockchain-based registry enables cross-organizational traceability of workflow executions thereby also allowing for ex-post auditability.
- 2) We demonstrate the technical realization of TCU using two fundamentally different technologies, i.e., TEE and zkVMs, and evaluate the impact achieving trustworthiness has on the system’s performance through initial experiments in a federated machine learning scenario, highlighting performance trade-offs between technology choice.

In the remainder of this paper, we introduce the Model in section 2 to introduce the system model, threat and requirements. We present the TCU Design in Section 3 and TCU Technical Realization in Section 4. In Section 5, we describe the experiment-driven evaluation. We present related work in Section 6, and finally, conclude in Section 7.

## II. MODEL AND REQUIREMENTS

In this section, we characterize how cross-organizational service workflows operate in a *decentralized* manner, and the threat model, define the core notions of *computational* and *workflow* integrity, discuss the confidentiality and verifiability challenges that arise, and derive the requirements our framework must fulfill.

### A. System Model: A Decentralized Service Workflow

We define cross-organizational service workflows as connected services, where each is executed by a different organization ( $Org_i$ , with  $i$  the id for the location on the workflow). By *decentralized*, we stress that no single coordinator or authority orchestrates the entire process. Instead, every organization controls and deploys its own service. Each service can consume:

- **External inputs:**  $EIn_i = \{EIn_i^1, EIn_i^2, \dots\}$ , typically a *set* of outputs from one or more predecessor services (e.g.,  $Out_i = \{Out_{(i-1)}^1, Out_{(i-1)}^2, \dots\}$ , as different models to aggregate in a global model for FL).
- **Internal inputs**  $IIn_i$ , kept under the exclusive control of  $Org_i$  and deemed confidential (e.g., proprietary weighting schemes for models or personally identifiable information).

Internal inputs are considered confidential, i.e., they must not be shared with external parties, while external inputs can be obtained by any organization.

In a typical workflow cycle, each *required* service in the DAG is invoked at least once, so that its output can serve as external input ( $EIn$ ) for any successors. Once a service  $Out$  is not shared as  $EIn$ , we consider the workflow instance,

conforming the particular DAG, complete. An example of such workflows can be seen in the context of decentralized AI application, as described in 1:

Federated learning [8]–[10] comprises worker nodes and an aggregator node, each of which can be represented by a different organization. In each learning cycle, the worker nodes execute a machine learning task on local data representing confidential  $IIn_i$  and collective learning results of the previous cycle representing  $EIn_i/Out_{i-1}$ . The resulting local models, representing  $Out_i$ , are sent to the aggregator node where they are combined into a global model which is returned to the worker nodes as  $EIn_{i+1}$  for the next cycle. Additionally, the inference from a global model can be used to make recommendations for other patients.

Although a workflow may seem “complete” when terminal services produce their outputs, decentralized environments naturally allow new computations to *extend* the DAG. For example in the FL scenario, a workflow execution represents a learning cycle, and a DAG composing all the computations could be recreated from the first local training until the last model aggregation after many learning cycles. Any output can serve as a fresh external input for a newly introduced service, underscoring the ongoing need for integrity guarantees across an evolving cross-organizational workflow.

### B. Threat Model

We characterize the threat model by classifying each organization  $Org_i$  as:

- **Honest-but-negligent:** Executes its service but may inadvertently misconfigure or misuse the code, generating incorrect outputs ( $Out_i$ ) or failing to implement the intended logic  $P$  accurately.
- **Malicious:** Intentionally alters its logic  $P$ , data, or outputs to gain advantage (e.g., skipping computation in FL, not using the indicated ML model for inference).

We assume no organization can forcibly read another’s internal inputs ( $IIn_{j \neq i}$ ); thus, data theft is not the primary threat. Instead, we focus on *detecting* incorrect computations—whether caused by negligence or malice.

We do not consider side-channel or fault-injection attacks on hardware, nor do we assume any trusted setup. Other well-known security practices such as network security (e.g., TLS) to prevent eavesdropping in transit are assumed to be in place.

### C. Computational and Workflow Integrity

Building on prior approaches for integrity and verifiability (e.g., [6], [11]), we require that a program  $P$  produce a proof  $\pi$  demonstrating computational correctness under confidentiality constraints. Formally, given a *proving key*  $PK$ , the execution is  $P(EIn, IIn, PK) \rightarrow (Out, \pi)$ . Any verifier holding the corresponding *verification key*  $VK$  can then check the correctness via  $verify(Out, \pi, VK) \rightarrow \{0, 1\}$ , without needing access to the confidential inputs  $IIn$ .

**Computational Integrity (Single Service):** To define *computational integrity*, we adopt the model presented in [11] for trustworthy pre-processing off-chain data. A service program

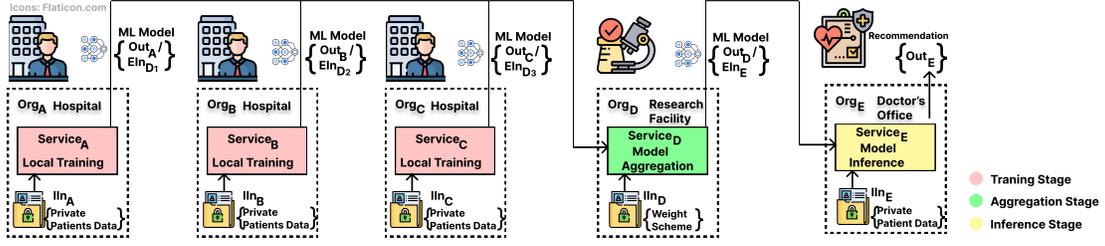


Fig. 1. Illustration of a federated learning workflow. Organizations A, B, and C (Hospitals) perform local model training on private datasets ( $IIn_{A,B,C}$ ), then send model updates ( $Out_{A,B,C}/EIn_{D_{1-3}}$ ) to Organization D (Research Facility), which aggregates them into a global model ( $Out_D/EIn_E$ ) with a private weight scheme ( $IIn_D$ ). Finally, Organization E (Doctor's Office) applies this aggregated model for making patient treatment recommendations ( $Out_E$ ).

$P$  is executed on external input  $EIn$  and some internal input  $IIn$  and returns output  $Out$  such that  $P(EIn, IIn) \rightarrow Out$ . A malicious organization may benefit from corrupting either the program  $P$  or the inputs. In the former case, an organization creates a manipulated program  $P'$  such that  $P'(EIn, IIn) \rightarrow Out' | Out' \neq Out$ . Furthermore, an organization may manipulate the  $EIn$  such that  $P(Ein', IIn) \rightarrow Out' | Out' \neq Out$  or manipulate  $IIn$  such that  $P(Ein, IIn') \rightarrow Out' | Out' \neq Out$ .

**Workflow Integrity (Chained Services):** Because the output  $Out_{i-1}$  from one service can become the external input  $EIn_i$  of another, the *entire* DAG preserves computational integrity, as long as  $EIn_i$  is verified as part of  $P_i$ , step by step ensuring end-to-end *workflow integrity*. Consequently, a final consumer can verify only the *last* output without re-verifying all preceding services. In other words, they need *not* check that *every* intermediate output ( $Out_{j<i}$ ) was produced by the intended program ( $P_{j<i}$ ) on the correct inputs, even when ( $IIn_{j<i}$ ) remain private—this guarantee is provided transitively through chained verifiable computations proofs, as each step verifies its input and computation, ensuring correctness propagates through the chain.

#### D. Confidentiality and Verifiability Challenges

Enforcing both *computational* and *workflow* integrity in a decentralized environment is non-trivial for several reasons:

- **Preserving Internal Inputs:** Organizations cannot expose proprietary or personal data ( $IIn_i$ ) to external re-execution or naive on-chain logging or verification.
- **Preventing Undetected Code Changes:** If verification keys depended on hidden parameters or a trusted setup, an organization could surreptitiously modify the intended  $P$ , e.g., if the building process is not reproducible.
- **Heterogeneous Execution Technologies:** Some organizations might use trusted execution (TEEs), while others might use cryptographic zero-knowledge systems. A unifying approach must handle both while preserving verifiability and low on-chain overhead.

#### E. Requirements

From the above setting and challenges, we derive the following key requirements for a cross-organizational trusted computation framework:

**R1 - Confidential Internal Inputs:** Each organization must retain confidentiality over its internal inputs ( $IIn_i$ ); no

party should be forced to disclose or re-execute these sensitive data externally.

**R2 - Verifiable Correctness of Outputs:** Any output ( $Out_i$ ) must be provably correct with respect to the intended program  $P$  on ( $EIn_i, IIn_i$ ), without revealing private inputs. This includes *deterministic reference to the code of  $P$  during verification* so that any modification to  $P$  invalidates existing proofs and no hidden parameters can be introduced.

**R3 - End-to-End Chainability:** Because the output of one service can become the input of another, proofs of correctness must *chain* through the workflow. A final consumer should not need to re-verify every step individually to ensure end-to-end correctness.

**R4 - Cross-Organizational Traceability:** A transparent record (e.g., on a blockchain registry) must indicate *which service* produced *which outputs*, under *which code version*. This enables ex-post audits and dispute resolution without exposing proprietary data.

**R5 - Low On-Chain Overhead and Heterogeneity:** Minimizing on-chain interactions avoids high gas fees and prevents data leakage on public ledgers. Moreover, the framework must handle multiple verifiable computation technologies (e.g., TEEs or ZKPs) so each organization can choose the technology best suited to its needs without compromising overall verifiability.

In the following section, we present the *TCU* framework that addresses these requirements by combining off-chain verifiable computation technologies (TEEs or zkVMs) with a blockchain-based registry for deterministic code references and proof traceability.

### III. TRUSTED COMPUTE UNIT DESIGN

To fulfill the requirements described in the previous section, we introduce our framework for Trusted Compute Units (TCUs) by describing their key architectural components and the framework procedures.

#### A. TCU Architecture

As depicted in Figure 2, the proposed framework advances cross-organizational workflows through two major architectural elements, the TCU and the Program Registry (PR).

TCUs are service units assumed to execute a program  $P$  using a verifiable computation technology that allows a  $TCU_i$

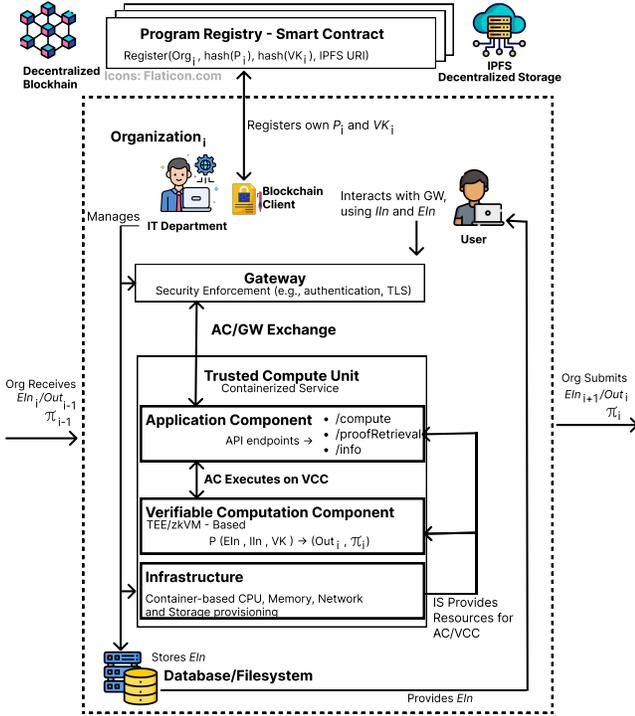


Fig. 2. TCU Framework Diagram

to create proofs  $\pi$  with a TCU-specific  $PK$  and the successor  $TCU_{i+1}$  to verify  $\pi$  with the corresponding  $VK$ .

The TCU can be modeled through 2 main components, the Application Component and Verifiable Computation Component, and a foundational Infrastructure for provisioning them.

1) *Application Component (AC)*: The AC provides a lightweight RPC interface for submitting computation tasks, retrieving results, and obtaining Program  $P$  specifications. We assume that incoming requests have already passed through gateway-based authentication and authorization filters. By delegating these policies to external components, the TCU retains a simple boundary and focuses on core compute functionality. The minimal endpoints are a `/compute` call for scheduling tasks, and a `/proofRetrieval` route for obtaining proofs. A `/info` endpoint reveals metadata such as TCU Program  $P$  location for validation and used technology for verification. In practice, the TCU only trusts requests that have already passed through the gateway’s authentication and authorization filters, keeping the component minimal and easily composable with other (micro)services. As illustrated in Figure 2, its primary responsibility is to expose a minimal set of endpoints that interact with the TCU’s Verifiable Computation Component.

2) *Verifiable Computation Component (VCC)*: The VCC can be modeled through three operations that are all executed (via the AC `/compute` call) within the same verifiable computation technology and a key pair consisting of the proving key (PK) used to create proofs  $\pi$  by the  $TCU_i$  and the verification key (VK) used to verify  $\pi$  by the successor  $TCU_{i+1}$ .

The (1) *External Input Verification (EIV)* takes the proofs from its predecessor TCUs as input and verifies them using the respective verification keys. With that, the computational integrity of the previous TCU operations can be confirmed. On successful EIV, the (2) *service is executed*. It takes the  $IIn$  and the verified  $EIn$  and returns the computational output. Finally, the TCU (3) *commits to the internal inputs* which is necessary to bind the inputs to the service execution. The TCU execution returns the computational output to the AC (so it’s retrieved in the `/proofRetrieval` endpoint), the commitment to the internal inputs, and the proof of computational correctness. The  $VK$  of a TCU is used for EIV by the successor TCU.

3) *Infrastructure (IS)*: The IS includes provisioning CPU, memory, and storage resources, orchestrating container deployments, and managing lifecycle events such as rolling updates. Container orchestration frameworks (e.g., Kubernetes) or virtualization managers typically provide these capabilities.

The IS supports features like automatic scaling, which starts additional TCU instances in response to rising workloads, or rolling upgrades to deploy patches with minimal downtime. It also enforces isolation boundaries between routine and verifiable computations, ensuring the TCU remains shielded from potential interference. Deterministic builds and cryptographic checksums can verify that the deployed code matches the audited binaries (matching the `/info` endpoint), mitigating supply-chain attacks [12] and preserving trust over time.

In practice, the Infrastructure is typically administered by the organization’s internal IT Department or DevOps teams, whether running on-premises or via a cloud provider. While the Figure 2 shows this infrastructure ‘inside’ the organization, its physical location can vary according to each team’s operational policies without affecting the TCU’s verifiability or confidentiality guarantees.

4) *Program Registry*: We introduce a smart contract-based program registry (PR), on a decentralized blockchain, that stores the specifications of the program  $P$ . By anchoring program  $P$ ’s logic, configuration, and cryptographic references on an immutable ledger, external verifiers can confirm exactly which code and parameters generated any given proof—even after the TCU has gone offline or been replaced. This approach decouples ephemeral TCU computations from the enduring record of the application-specific program, allowing auditors or external services to validate that a proof indeed corresponds to the original, trusted version of  $P$  (matching the `/info` endpoint). By maintaining a tamper-proof ledger entry, we ensure that verifiability persists long-term, preserving integrity and trust in the computed results.

## B. TCU Life Cycle

As depicted in Figure 2, the service lifecycle consists of a one-time setup and recurring `/compute` calls. As a prerequisite, we assume that the Program Registry is deployed to a decentralized public infrastructure as provided by smart contract-enabled blockchains.

**Setup:** During the setup, the TCU is instantiated and the program  $P$  registered in the registry. The *instantiation* involves

the specification of the logic as  $P$ , the compilation into an executable and provable format. This ensures that the AC has a well-defined program  $P$  to execute within the VCC. Once integrated, the TCU-specific proving and verification keys are generated for the VCC.

For *registration*, the organization registers  $P$  into the Program Registry, the TCU’s identifier, and the  $VK$ .

**Operation:** Once the setup is complete, the TCU can operate in a service workflow. On receiving the “/compute” call, with  $El_{i-1}$  from a predecessor TCU, the AC checks the validity of the predecessor  $TCU_{i-1}$ , from the corresponding record on the PR. Then, the AC calls the TCU  $i$ ’s VCC to execute the program  $P$  with the operations on the internal and external inputs, with the successful EIV as a prerequisite for the service execution and the conclusive internal input commitment. The execution returns a proof of computational correctness  $\pi$  with the output and the internal input commitment  $C(IIn)$ . Then an authorized User submits  $\pi$  and  $C(IIn)$  to the successor Organization’s TCU, using the “/proofRetrieval” endpoint through the AC.

In our framework, we use the term “Trusted Compute Unit” (TCU), reflecting how trust emerges from satisfying the framework’s requirements within a self-contained, manageable unit-like service interface. Besides the  $VK$ , each TCU offers a deterministically generated reference to  $P$  for verification (**R2: Verifiable Correctness of Outputs**) and anchors that reference and the  $VK$  on a Program Registry (**R4: Cross-Organizational Traceability**), ensuring any code changes become self-revealing. The AC presents a uniform API so heterogeneous verifiable computation technologies—TEEs or zkVMs—can be integrated without modifying service endpoints (**R5: Low On-Chain Overhead and Heterogeneity**). Meanwhile, the VCC preserves confidential inputs (**R1: Confidential Internal Inputs**) and chains correctness proofs across computations (**R3: End-to-End Chainability**). Lastly, because only sink computations need on-chain verification (if required by a dApp), the on-chain overhead remains minimal (**R5** again), while the IS orchestrates off-chain deployments seamlessly. This design unifies code provenance, proof generation, and cross-technology composability in a portable, tamper-evident framework for verifiable off-chain services.

#### IV. TECHNICAL REALIZATIONS

In this section, we describe how TCUs can be realized with current Container Orchestration Frameworks for the IS, with Trusted Execution Environments (TEEs) and Zero-knowledge Virtual Machines (zkVMs) for the VCC, and lightweight web frameworks for the AC. While we consider these technologies as candidates, we would like to underline the generality of TCU which may be used as a blueprint for further technologies, in particular for other Verifiable Computation Technologies, e.g., FHE and SMPC [4]–[6]. Finally, we outline how the PR can be realized with blockchain smart contracts.

##### A. Infrastructure with Orchestration Framework

In practice, this layer leverages container orchestration frameworks (e.g., Kubernetes or Argo Workflows) to provision and lifecycle-manage TCU instances. For TEE-based TCUs (e.g., Intel TDX or AWS Nitro Enclaves), the corresponding container images include both the application logic and enclave drivers, deployed only on nodes that support enclave-capable hardware. For zkVM-based TCUs (e.g., Risc0), the container images embed the zkVM runtime and deterministic ELF binaries. The IS ensures correct scheduling (e.g., via node labels), secure distribution of proving and attestation keys, and rolling updates to patch or replace TCU containers with minimal disruption. In the case of TEEs, it also facilitates remote attestation workflows, where the container attests to a trusted authority (e.g., Intel TA or AWS Nitro) prior to accepting inputs. For zkVMs, container-level integrity checks (e.g., Docker Image signing) ensure the correct zkVM runtime is loaded. Finally, the IS integrates with DevOps pipelines (e.g., reproducible builds, image signing) to maintain a consistent, verified environment for TCU operations.

##### B. Verification Computation Component with Trusted Execution Environments

Trusted Execution Environments (TEEs) are secure hardware components that safeguard data and code from external tampering and disclosure [13]. Programs executed within TEEs operate within isolated and/or encrypted memory regions, shielding the content even from the hardware owner and ensuring the integrity of computations conducted within. While TEEs also enable confidential computation, i.e., protecting *data in use* from the executor, we leverage its ability to make internally executed programs externally verifiable.

*Remote Attestation* enables external parties to verify the integrity of a Trusted Execution Environment (TEE)’s internal state and the authenticity of messages from within it. TEE-enabled machines have a machine identity key embedded into the hardware during manufacturing. Using this key, each TEE instance generates an identity certificate that can be externally verified through a Public Key Infrastructure (PKI). These keys are used to sign measurements that represent a complete snapshot of the TEE’s internal state at boot. When a remote attestation request is made, the TEE returns signed measurements that can be reconstructed outside the TEE to verify the integrity of its internal state. These measurements can include a reference to a specific container, whose correctness can be validated with a deterministic build system [12], thereby enabling verification of a trustworthy computation of the program  $P$ . This general model of TEEs represents the essential concepts of confidential VMs, or containers [14], like AWS Nitro [15] which we use for the TCU realization.

During the **setup**, the TCU’s program logic with the verification keys of the predecessor TCUs are specified and compiled during the initialization of the TEE. On initialization, the measurements of the internal state, i.e., the TCU’s binaries, are created and signed through the TEE-specific key for remote attestation. This attestation report and the TEE’s public key

and specific container are registered at the Program Registry. The attestation represents a commitment to the TCU allowing other workflow parties to check the integrity of the TEE’s internal state whereas the TEE-specific public pair represents the verification key.

In the case of the workflow **operation**, the  $EIn$  and  $IIn$  are provided through the AC, the host of the TEE, where the TCU logic is executed. After executing the three TCU VCC operations, the computational output  $Out$  and the commitment to the internal inputs  $com(IIn)$  are signed with the TEE’s private key. The signature representing the proof of computational integrity can be verified using the corresponding TEE’s public key.

### C. Verification Computation Component with Zero-Knowledge Virtual Machines

Zero-Knowledge Virtual Machines (zkVMs) leverage non-interactive zero-knowledge proof (ZKP) protocols to make the computational integrity of programs executed inside the zkVMs independently verifiable. ZKPs can encode program logic in mathematical constraint systems called circuits. Computational integrity can be asserted if a valid, input-specific variable assignment of the constraint system can be found. To enable external verifiability without disclosing computational inputs, elliptic curve cryptography applies where proofs are constructed and verified with a circuit-specific key pair. zk-STARKs, as described in [16], define a class of such protocols, which is typically used in zkVMs, characterized by fast proof generation and a transparent, deterministic setup free of trust assumptions, enabling verification of  $P$ ’s computation.

Different from application-specific circuits, zkVMs encode the instruction set of virtual machines in circuits. This allows for executing programs in a format compliant with the zkVM’s instruction set. Risc0 [17], for example, is a zkVM built upon the RiscV instructions and, hence, allows executing programs deterministically compiled to general Executable and Linking Format (ELF) binaries. The execution of an ELF binary file returns a cryptographic proof that can be verified in any other Risc0 zkVM through a reference to the binary, called the ImageID. For the following, we assume Risc0-enabled TCUs.

During the **setup**, the TCU logic is specified in a high-level language and compiled into an ELF binary file. From that, the ImageID can be created which is a cryptographic (hash-based) representation of the initial zkVM memory state produced when the ELF binaries are loaded. The ImageID allows the zkVM of the successor organization to verify that the computational proof has been generated by the expected ELF binary. The ImageID corresponding to the verification key and the ELF binary representing the TCU commitment are made available on the Program Registry.

In the case of **operation**, the AC provides the  $IIn$ ,  $EIn$ , and ELF binary to the zkVM’s executor which runs the ELF binary and records the session as complete snapshots of the state of the zkVM throughout the execution. Based on that, the Receipt is created which serves as proof of computational integrity. The Receipt contains the computational outputs, the

execution’s imageID, and the seal, a cryptographic artifact that attests to the validity of the outputs and imageID. The Receipt can be verified with the original ImageID in the VCC of the successor organization.

### D. Application Component with Containers

For a real-world deployment, the AC is realized as a lightweight containerized service exposing RPC endpoints (e.g., `/compute` and `/proofRetrieval`). An external gateway (such as Envoy or Istio) is delegated with the handling of advanced features like authentication tokens, TLS termination, and rate limiting before requests reach the TCU.

Inside the TCU container or enclave, a minimal web framework (e.g., a Flask service) receives validated requests and forwards them to the VCC. When the `/compute` endpoint is called, the AC packages  $EIn$  along with any local  $IIn$  and invokes the verifiable code inside the TEE or zkVM runtime. Once the computation finishes, on the VCC, it returns the resulting output with a verifiable proof.

For chained trusted computations, the AC also records a proof reference or TCU identifier on the Program Registry (also available at the `/info` endpoint), enabling cross-organization independent verification. Similarly, the `/proofRetrieval` endpoint serves as a lightweight retrieval mechanism for downstream consumers to obtain existing proofs on demand. Because business logic and advanced security checks reside outside the TCU, this component remains easy to adapt and integrate into existing DevOps pipelines or microservice meshes, ensuring each TCU instance can be deployed with minimal friction.

### E. Program Registry with Blockchains

Blockchains are decentralized systems designed to resolve trust issues among collaborating parties without depending on trusted third parties (TTPs). Submitted transactions are redundantly processed through a collectively executed consensus protocol, and agreed-upon transactions are securely recorded in an immutable, append-only transaction history.

We use smart contracts to technically realize the Program Registry, e.g., those based on Ethereum Virtual Machine [18] or WebAssembly [19]. Each organization is assumed to have its own blockchain account represented through a public-private key pair. As transactions are by default authenticated with the organization’s account keys, transactions can be associated with the organization’s blockchain account. While this hides the organization’s identity behind the account representation, it allows to associate TCUs if the mapping of the account keys and the organization’s identity are known.

As blockchains suffer from storage limitations, we propose keeping large artifacts off-chain, in particular  $P$ , in distributed file storage like IPFS [20] and only storing a hash-based reference on the blockchain to preserve the artifacts’ integrity and transparent validation. Such off-chain storage patterns are well-known and, for example, described in [21].

## V. EVALUATION

In this section, we evaluate TCU by assessing trustworthiness, technical and performance concerns in the context of a FL scenario, and discuss open security-related issues.

### A. Experiment-driven Evaluation

To technically evaluate the TCU framework, we implement the TCU’s VCC using Risc0<sup>2</sup> as a zkVM, and AWS Nitro Enclave<sup>3</sup> as a VM-based TEE, and the PR in EVM-based [18] for smart contracts execution. The AC is a light web server API on Flask<sup>4</sup> containerized on a Dockerfile<sup>5</sup>, and the IS is managed with AWS Nitro Hypervisor. The experiments were executed in an AWS EC2 instance of type c5.4xlarge, with 16 vCPUs and 32 GB of memory.

We evaluate the TCU in a federated learning (FL) scenario as introduced in Section II. In FL, the TCU helps to mitigate *trustworthiness concerns* derived from aggregation attacks and model poisoning [22] through intervened proofs of computational integrity [9], [10].

For the local learning node, we implement a simple neural network with two hidden layers, using stochastic gradient descent as the optimization method, as the  $P$  logic of a TCU, while doing dataset authentication with a signature from the node and verifying that the base global model to use comes from a TCU aggregator. In the case of aggregation, we implement federated averaging [23] inside the TCU which takes the average of local model updates of the worker nodes and additionally verifies that the local models come from a TCU. Without compromising the generality of our approach, the FL scenario allows us to address the previous concerns about trustworthiness, while evaluating performance in the following experiments E1-E4<sup>6</sup>:

- E1 We set up and deploy TCUs with different VCCs, TEE and zkVM-based, for a worker and aggregator node and measure execution times and transaction costs.
- E2 We run the same compute job, i.e., learning and aggregation, in a TEE and a zkVM-based VCC and alternate TCU  $i-1$  to verify proofs from TEEs or zkVMs.
- E3 We increase the dataset volume, in the range of 20 to 10240 rows, to measure how it affects the authentication of the dataset and learning phase.
- E4 We increase the number of learners, in the range of 2 to 800, to measure how the proof verification time grow inside of the aggregator VCC.

For E1 to E4 we switch between a zkVM-based and TEE-based aggregator and learners respectively.

**Results and Evaluation:** The practical implementation of the TCU, which addresses the *trustworthiness concerns*. By chaining proofs of computational integrity together, such guarantees extend across multiple heterogeneous service executions along the workflow (**R3**, **R4**, **R5**),  $IIn$  can not be

tampered unnoticeably either as Data for the Local Model or the Weighting Scheme, nor  $EIn$  as a Local or Global Model. At the same time the TCU information is maintained on the blockchain-based PR alongside the storage of  $EIn$  on each organization’s side (**R2**).

The TCU protects against input tampering attacks, while guaranteeing input confidentiality, (**R1**, **R2**) inherently by using verifiable computation and against formal incorrectness by storing commitments to TCU’s  $P$  in the PR, so the  $P$  reference to  $VK$  can be created on the verifier side.

Cheating organizations can be held accountable if irregularities in shared data are detected, e.g., through plausibility checks. The integrity of  $IIn$ ,  $EIn$ , and TCU program  $P$  can then be verified by inspecting  $P$  using the PR. This helps in dispute cases or audits to solve conflicts ex-post.

TABLE I  
CONTAINER IMAGE SIZE FOR EACH SCENARIO.

Node Task	VCC <sub>i</sub> (Computing)	$EIn_{i-1}$ Verified	Image Size
Learner	zkVM	TEE	~6.4 GB
Learner	zkVM	zkVM	~6.4 GB
Learner	TEE	TEE	~1.8 GB
Learner	TEE	zkVM	~6.4 GB
Aggregator	zkVM	TEE	~6.4 GB
Aggregator	zkVM	zkVM	~6.4 GB
Aggregator	TEE	TEE	~1.8 GB
Aggregator	TEE	zkVM	~6.4 GB

Across all scenarios, on-chain PR deployment consumes about 979k gas, and registering a TCU costs roughly 265k gas, regardless of dataset or model sizes. Off-chain TEE attestation remains negligible at  $\sim 0.01$ s. However, Table I indicates container image size is inflated by the requirement of the Risc0 framework library to be included in the image (for proving or verifying with the zkVM), reflecting the difference between bundling the specific binaries for a computation versus adding the specific cryptographic runtime.

For local training on  $n$  data samples, Figure 3 shows TEE verification can exceed the training cost for smaller  $n$  (up to  $\sim 2400$ ), while zkVM overhead escalates faster beyond  $n = 320$ . Model aggregation scales linearly with the number of local models ( Figure 4), each requiring a single verification plus an averaging step. For both types of nodes, TEEs run the same compiled instructions as regular binaries, whereas zero-knowledge systems must encode logic into cryptographic operations—leading to *multiple orders of magnitude* difference of execution time.

### B. Discussion

Complementing the previous performance evaluation, we now revisit key aspects of TCU’s for its real world application.

A significant finding for both scenarios, learning and aggregation, is that the verification of a TCU from a previous TCU is more efficient when both the learner TCU and the aggregation TCU share the same technology type. These results can help organizations manage resource allocation more effectively as workflows expand. For example, load distribution strategies can be optimized throughout the workflow, given that TEEs

<sup>2</sup><https://dev.risczero.com/api/zkvm>

<sup>3</sup><https://aws.amazon.com/de/ec2/nitro/>

<sup>4</sup><https://flask.palletsprojects.com/en/stable/>

<sup>5</sup><https://docs.docker.com/reference/dockerfile/>

<sup>6</sup>Repository for experiments: <https://github.com/ferjcast/TCU>

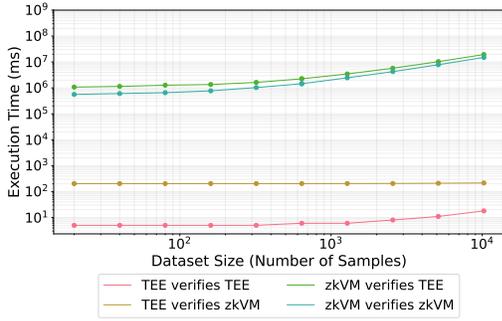


Fig. 3. Impact of data volume ( $IIn_i$  size) in Local Training scenarios.  $VCC_i$ -type verifies  $EIn_{i-1}$ -type.

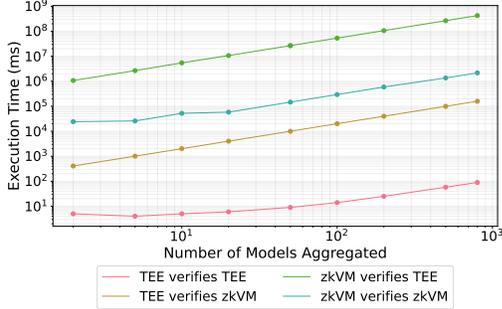


Fig. 4. Impact of total number of models ( $EIn_i$  set size) in Aggregation scenarios.  $VCC_i$ -type verifies  $EIn_{i-1}$ -type.

generally exhibit better performance compared to zkVMs. Understanding these factors and developing tailored strategies to address specific use case constraints will significantly enhance the practical implementation of TCU. In practice, whether a TCU-based service can tolerate higher proof-generation overhead or deferred execution depends on its operational requirements. For instance, model training may only occur weekly or monthly, making it acceptable to use a more computation-intensive verifiable environment if the update cadence is low. By contrast, inference services demand quicker turnaround. This flexibility allows each organization to pick the most suitable verifiable technology in line with the service’s real-time or batch-processing needs, without compromising the overall trust guarantees. This initial experimentation can guide organizations in efficiently deploying TCUs across heterogeneous VCCs.

## VI. RELATED WORK

In this section, we review the related work to TCUs. We include data provenance models, blockchain-based approaches, and existing cooperative services.

Models for provenance [24] and the PROV<sup>7</sup> specification from W3C have been designed with only a focus on data, not on the computations and processes producing the data. Hence, systems in support of data provenance [25]–[27] exist, but all only consider the data and not the computations. Some approaches leverage the blockchains to enhance the trust in

data provenance [28], or also provide service composition information as part of the data provenance record [29], [30]. Additionally, some recent efforts like ZkTLS [31] focus on applying zero-knowledge to the TLS session, allowing clients to verify certain properties of the session without revealing sensitive details. However, these approaches do not have verifiability for the computations generating the data. Hence, approaches like the TCU would add and enhance data provenance.

Similar approaches for verifiable workflows already exist, for instance, some using only TEEs [32], [33]. Authors in [34], also use commitment in the blockchain of a hashed proof, represented as  $EIn$  in the TCU, and leveraging ZKP-based computational correctness guarantees, but their approach has higher gas costs and only uses zkSNARKS. Hence, existing approaches do not consider the adaptability to other types of VCCs, and thus, do not enable organizations to choose between the trade-offs of each verifiable computing technology, unlike TCUs. Other noteworthy ZKP-based workflow systems do not focus on the confidentiality aspect, for example, using blockchains as the verification layer [35] or store public proofs on-chain, thus introducing loss of confidentiality [11], [36].

Lastly, some approaches also tackle the reproducibility problem without naïve re-execution using secure network provenance, both without and with the requirement of trusted hardware components [37]–[39].

While these approaches offer valuable insights into various aspects of secure and verifiable cooperative services, they often focus on only one of those specific elements such as confidentiality, verifiability, or traceability. Unlike previous approaches, that are technology dependent, TCU is a flexible framework that can operate technology agnostic with its abstraction of the TCU with the possibility of chaining verifiable computations.

## VII. CONCLUSION

We introduced the TCU, a framework for a verifiable service workflow system for trustworthy cross-organizational data sharing. Using TCUs allows organizations to verify and demonstrate the computational correctness of a service workflow. The containerized design of TCUs integrates seamlessly with existing services and orchestration pipelines, facilitating ephemeral deployments that are reproducible and auditable.

Our experiments confirm the TCU’s interoperability and composability across TEEs and zkVMs, offering flexible trust solutions for cross-organizational workflows. Thus, the TCU emerges as a pioneering approach to next-generation verifiable off-chain computations. Future research will broaden TCU’s scope to incorporate additional TEE variants (e.g., Intel TDX) and other zkVMs (e.g., Nexus), as well as alternative verifiable computation paradigms such as Fully Homomorphic Encryption and Secure Multi-Party Computation, further reinforcing TCU’s vision of an agnostic, privacy-preserving architecture for diverse organizational contexts.

## ACKNOWLEDGEMENTS

Funded by the European Union (TEADAL, 101070186). Views and opinions expressed are, however, those of the

<sup>7</sup><https://www.w3.org/TR/prov-overview/>

author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

## REFERENCES

- [1] J. Laux, S. Wachter, and B. Mittelstadt, "Three pathways for standardisation and ethical disclosure by default under the european union artificial intelligence act," *Computer Law & Security Review*, vol. 53, p. 105957, 2024.
- [2] C. Liu, Q. Zeng, L. Cheng, H. Duan, M. Zhou, and J. Cheng, "Privacy-preserving behavioral correctness verification of cross-organizational workflow with task synchronization patterns," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 3, pp. 1037–1048, 2020.
- [3] C. Liu, H. Duan, Q. Zeng, M. Zhou, F. Lu, and J. Cheng, "Towards comprehensive support for privacy preservation cross-organization business process mining," *IEEE Transactions on Services Computing*, vol. 12, no. 4, pp. 639–653, 2016.
- [4] X. Yu, Z. Yan, and A. V. Vasilakos, "A survey of verifiable computation," *Mobile Networks and Applications*, vol. 22, pp. 438–453, 2017.
- [5] N. Smart, "Computing on encrypted data," *IEEE Security & Privacy*, vol. 21, no. 4, pp. 94–98, 2023.
- [6] T. Bontekoe, D. Karastoyanova, and F. Turkmen, "Verifiable privacy-preserving computing," *arXiv preprint arXiv:2309.08248*, 2023.
- [7] M. Russinovich, C. Fournet, G. Zaverucha, J. Benaloh, B. Murdoch, and M. Costa, "Confidential computing proofs: An alternative to cryptographic zero-knowledge," *Queue*, vol. 22, no. 4, pp. 73–100, 2024.
- [8] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106775, 2021.
- [9] J. Heiss, E. Grünwald, S. Tai, N. Haimlerl, and S. Schulte, "Advancing blockchain-based federated learning through verifiable off-chain computations," in *2022 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2022, pp. 194–201.
- [10] C. Lee, J. Heiss, S. Tai, and J. W.-K. Hong, "End-to-end verifiable decentralized federated learning," in *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2024, pp. 434–442.
- [11] J. Heiss, A. Busse, and S. Tai, "Trustworthy pre-processing of sensor data in data on-chaining workflows for blockchain-based iot applications," in *Service-Oriented Computing: 19th International Conference, ICSOC 2021, Virtual Event, November 22–25, 2021, Proceedings 19*. Springer, 2021, pp. 133–149.
- [12] C. Lamb and S. Zacchiroli, "Reproducible builds: Increasing the integrity of software supply chains," *IEEE Software*, vol. 39, no. 2, pp. 62–70, 2021.
- [13] A. Muñoz, R. Rios, R. Román, and J. López, "A survey on the (in) security of trusted execution environments," *Computers & Security*, vol. 129, p. 103180, 2023.
- [14] C. C. Consortium *et al.*, "Common terminology for confidential computing," *Online*, (December 2022). (Available from: <https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/Common-Terminology-for-Confidential-Computing.pdf>), 2022.
- [15] M. Brossard, G. Bryant, B. El Gaabouri, X. Fan, A. Ferreira, E. G. Evans, C. Haster, E. Johnson, D. Miller, F. Mo *et al.*, "Private delegated computations using strong isolation," *IEEE Transactions on Emerging Topics in Computing*, vol. 12, no. 1, pp. 386–398, 2023.
- [16] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable zero knowledge with no trusted setup," in *Advances in Cryptology—CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*. Springer, 2019, pp. 701–732.
- [17] J. Bruestle and P. Gafni, "Risc zero zkvm: scalable, transparent arguments of risc-v integrity," *Draft. July*, vol. 29, 2023.
- [18] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.
- [19] A. Tara, K. Ivkushkin, A. Butean, and H. Turesson, "The evolution of blockchain virtual machine architecture towards an enterprise usage perspective," in *Software Engineering Methods in Intelligent Algorithms: Proceedings of 8th Computer Science On-line Conference 2019, Vol. 1 8*. Springer, 2019, pp. 370–379.
- [20] J. Benet, "IPFS - content addressed, versioned, P2P file system," *CoRR*, 2014.
- [21] J. Eberhardt and J. Heiss, "Off-chaining models and approaches to off-chain computations," in *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, 2018, pp. 7–12.
- [22] G. Xia, J. Chen, C. Yu, and J. Ma, "Poisoning attacks in federated learning: A survey," *IEEE Access*, vol. 11, pp. 10 708–10 722, 2023.
- [23] A. Nilsson, S. Smith, G. Ulm, E. Gustavsson, and M. Jirstrand, "A performance evaluation of federated learning algorithms," in *Proceedings of the second workshop on distributed infrastructures for deep learning*, 2018, pp. 1–8.
- [24] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers *et al.*, "The open provenance model core specification (v1. 1)," *Future generation computer systems*, vol. 27, no. 6, pp. 743–756, 2011.
- [25] A. Gehani and D. Tariq, "Spade: Support for provenance auditing in distributed environments," in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 2012, pp. 101–120.
- [26] N. Baracaldo, L. A. D. Bathen, R. O. Ozugha, R. Engel, S. Tata, and H. Ludwig, "Securing data provenance in internet of things (iot) systems," in *Service-Oriented Computing—ICSOC 2016 Workshops: ASOCA, ISyCC, BSCI, and Satellite Events, Banff, AB, Canada, October 10–13, 2016, Revised Selected Papers 14*. Springer, 2017, pp. 92–98.
- [27] S. Malik, S. S. Kanhere, and R. Jurdak, "Productchain: Scalable blockchain framework to support provenance in supply chains," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2018, pp. 1–10.
- [28] D. Mertens, J. Kim, J. Xu, E. Kim, and C. Lee, "Smart flow: a provenance-supported smart contract workflow architecture," *Cluster Computing*, pp. 1–15, 2024.
- [29] C. A. Ardagna, M. Anisetti, B. Carminati, E. Damiani, E. Ferrari, and C. Rondanini, "A blockchain-based trustworthy certification process for composite services," in *2020 IEEE International Conference on Services Computing (SCC)*. IEEE, 2020, pp. 422–429.
- [30] F. Corradini, A. Marcelletti, A. Morichetta, A. Polini, B. Re, F. Tiezzi *et al.*, "Chorchain: A model-driven framework for choreography-based systems using blockchain," in *ITBPM@ BPM*, 2021, pp. 26–32.
- [31] M. Kalka and M. Kirejczyk, "A comprehensive review of tlsnotary protocol," *arXiv preprint arXiv:2409.17670*, 2024.
- [32] A. Delignat-Lavaud, C. Fournet, K. Vaswani, S. Clebsch, M. Riechert, M. Costa, and M. Russinovich, "Why should i trust your code? confidential computing enables users to authenticate code running in tees, but users also need evidence this code is trustworthy," *Queue*, vol. 21, no. 4, pp. 94–122, 2023.
- [33] H. Howard, F. Alder, E. Ashton, A. Chamayou, S. Clebsch, M. Costa, A. Delignat-Lavaud, C. Fournet, A. Jeffery, M. Kerner *et al.*, "Confidential consortium framework: Secure multiparty applications with confidentiality, integrity, and high availability," *arXiv preprint arXiv:2310.11559*, 2023.
- [34] B. Á. Toldi and I. Kocsis, "Blockchain-based, confidentiality-preserving orchestration of collaborative workflows," *arXiv preprint arXiv:2303.10500*, 2023.
- [35] G. Ramezan and E. Meamari, "zk-iot: Securing the internet of things with zero-knowledge proofs on blockchain platforms," *arXiv preprint arXiv:2402.08322*, 2024.
- [36] J. Heiss, T. Oegel, M. Shakeri, and S. Tai, "Verifiable carbon accounting in supply chains," *IEEE Transactions on Services Computing*, 2023.
- [37] W. Zhou, Q. Fei, A. Narayan, A. Haerberlen, B. T. Loo, and M. Sherr, "Secure network provenance," in *Proceedings of the twenty-third ACM symposium on operating systems principles*, 2011, pp. 295–310.
- [38] X. Zhou, A. Nehme, V. Jesus, Y. Wang, M. Josephs, K. Mahbub, and A. Abdallah, "Audiwflow: Confidential, collusion-resistant auditing of distributed workflows," *Blockchain: Research and Applications*, vol. 3, no. 3, p. 100073, 2022.
- [39] M. M. B. Taha, S. Chaisiri, and R. K. Ko, "Trusted tamper-evident data provenance," in *2015 IEEE Trustcom/bigdatase/ispa*, vol. 1. IEEE, 2015, pp. 646–653.