An Evidence-driven Protocol for Trustworthy CI Pipelines

Fernando Castillo[1], Eduardo Brito[2,3], Pille Pullonen-Raudvere[2], Sebastian Werner[1], and Stefan Tai[1]

[1] Information Systems Engineering, TU Berlin, Einsteinufer 17, Germany
{fc,sw,st}@ise.tu-berlin.de
[2] Cybernetica AS, Estonia Mäealuse 2/1, 12618 Tallinn, Estonia
{eduardo.brito, pille.pullonen-raudvere}@cyber.ee
[3] University of Tartu, Narva mnt 18, 51009, Tartu, Estonia

Abstract. Enterprise software supply chains are increasingly vulnerable to infrastructure attacks, resulting in financial and reputational damage. Ensuring the integrity and provenance of software artifacts remains a significant challenge, where re-execution of the build and tests by every consumer to guarantee provenance produces a verification bottleneck and credibility reduction. This paper presents an evidence-driven protocol for trustworthy Continuous Integration (CI) pipelines that combines Deterministic Build Systems (DBS) with Trusted Execution Environments (TEEs). The approach provides cryptographically verifiable guarantees of integrity, authenticity, and attestation for CI artifacts in distributed environments, reducing implicit trust without requiring costly re-execution by consumers. We introduce a protocol that binds deterministic builds with TEE-based attestations, formalizing the evidence life cycle, together with a practical implementation using Nix and Intel TDX. Experimental results show that artifact verification is reduced from redundant computation to lightweight signature and policy checks. These findings demonstrate that evidence-driven CI pipelines establish scalable and verifiable trust in digital infrastructure, effectively amortizing the initial computational overhead introduced by TEEs.

Keywords: Continuous integration security, Trustworthy software, Secure software supply chain, Evidence-driven trust

1 Introduction

Attacks on digital infrastructure, whether within enterprises or in external systems they depend on, can directly and indirectly compromise organizations and cause large-scale operational disruptions. In such scenario, software supply chain security has emerged as a critical area of research, particularly in response to high-profile attacks like SolarWinds [1], Log4Shell [13], or the XZ exploit [24], which demonstrated the vulnerabilities inherent to modern software supply chains. Such incidents have caused losses of hundreds of millions of dollars, frequently exploiting vulnerabilities in the software supply chain of

third-party components [47], and most recently exemplified by the December 2025 Trust Wallet pipeline bypass causing $8.5M USD in losses[4].

As software development becomes increasingly complex and globally distributed, establishing trust in produced software artifacts has become progressively more difficult. For enterprise infrastructure, where code directly governs economic behavior, trustworthy artifacts must provide not only integrity, but also authenticity and attestation guarantees across the entire software life cycle. However, modern DevOps and CI/CD practices, while enabling rapid delivery, significantly hinder the enforcement of these guarantees [34]. Security-sensitive steps such as building and testing are frequently outsourced to shared CI environments, expanding the attack surface of the software supply chain.

A fundamental weakness of current software supply chains is that deployment environments rarely rebuild software from source, instead relying on pre-built binaries or container images obtained from external infrastructure [15]. Ensuring end-to-end trust from source commits to build artifacts and test results therefore remains an open challenge, which is exacerbated by growing codebases, extensive third-party dependencies [18], globally distributed teams, or development outsourcing [31]. While approaches such as DevSecOps [39,6], Software Bills of Materials (SBOMs) [48], and Continuous Compliance [14] improve visibility and security practices, they lack scalable mechanisms to provide verifiable trust guarantees across heterogeneous and distributed execution environments. This reliance on implicit trust fuels a 'credibility deficit' [22], where users cannot confidently adopt artifacts without performing redundant validations.

In this paper, we introduce an evidence-driven protocol for trustworthy Continuous Integration (CI) pipelines that mitigates reliance on implicit trust assumptions. The protocol formalizes declarative, inspectable processes backed by cryptographic guarantees of integrity, authenticity, and attestation throughout the software supply chain. Our main contributions are as follows:

- **Novel Evidence-driven Trustworthy CI Protocol:** A generalized protocol that seamlessly integrates into standard CI pipelines, transforming implicit trust into a verifiable chain of evidence and enabling a lightweight artifact verification model without redundant rebuilds.
- **Implementation and evaluation:** A practical proof of concept using GitLab, NixOS, and Intel TDX-based TEEs, together with an evaluation demonstrating cost savings from avoided rebuilds and tests.

The remainder of this paper is organized as follows: Section 2 introduces background concepts; Section 3 reviews related work; Section 4 defines trust model; Section 5 presents the protocol; Section 6 describes the implementation; Section 7 evaluates the system; Section 8 discusses its implications; and Section 9 concludes.

[4] Trust Wallet v2.68 Incident Update

2 Background

In this section, we first describe how TEEs can produce evidence, and discuss how trust can be established in CI pipelines.

2.1 Evidence from Trusted Execution Environments

Trusted Execution Environments (TEEs) are hardware-supported isolation mechanisms that protect code and data from external tampering and disclosure [28]. Programs executed within a TEE run in isolated or encrypted memory, ensuring computation integrity even in the presence of a potentially untrusted system operator. TEEs support remote attestation, enabling external parties to verify the integrity of the enclave's execution state and the authenticity of its outputs. This mechanism prevents malicious entities from impersonating them. Each TEE is provisioned with a hardware-backed identity key at manufacturing time, verifiable via a Public Key Infrastructure (PKI). This key is used to derive attestable identities for instantiated TEEs, enabling the generation of verifiable evidence of correct execution.

Modern platforms such as Intel TDX (Trusted Domain Extensions) and AMD SEV (Secure Encrypted Virtualization) support VM-based TEEs, where entire virtual machines are protected by hardware-enforced isolation. These platforms provide confidentiality and integrity guarantees for VM memory and execution state, making them well suited for executing security-critical CI tasks in a verifiable manner. By combining secure execution, identity verification, and remote attestation, TEEs enable the production of evidence required to create a verifiable chain of trust across CI phases.

2.2 Trust in Continuous Integration Pipelines

As software pipelines grow in scale and complexity, ensuring trust and transparency has become a central concern. Frameworks such as DevSecOps [39] and VeriDevOps [12] integrate security and verification mechanisms throughout the software life cycle, while TrustOps [6] emphasizes continuous evidence collection and validation across development and operation phases. CI pipelines automate the integration of code changes, enabling frequent builds, testing, and deployment while maintaining a consistent deployable state. However, as CI infrastructures expand to support distributed teams and multi-cloud environments, they face increasing challenges related to security, integrity, and traceability [2].

Deterministic build systems (DBS) establish a crucial foundation for software supply chain security by enabling consistent, reproducible pipelines [7]. Nevertheless, reproducibility alone does not guarantee the security or correctness of the resulting artifacts [19], necessitating the explicit declaration and enforcement of security policy checks throughout the CI pipeline. Consequently, combining the determinism of a DBS with the attestation of a TEE presents a promising avenue for producing the verifiable evidence needed to secure CI pipelines.

3 Related Work

In this section, we position our work with respect to existing approaches. Despite substantial progress, current solutions still lack comprehensive, verifiable evidence spanning the full software development and deployment life cycle. Our work addresses these gaps with an evidence-driven protocol.

Software Supply Chain Security Challenges: Software supply chain security has become a major research focus following high-profile incidents such as ByBit [27], SolarWinds [1], and the XZ exploit [24], which exposed systemic vulnerabilities in modern supply chains and caused losses exceeding hundreds of millions of dollars. Tools such as Trivy and other SBOM-based solutions aim to improve transparency and traceability of software components [32], with recent work extending dependency analysis [25] and policy auditing or usage justification [45]. However, important gaps remain, including the lack of unified threat modeling across integration, CI, and CD phases [35], as well as inconsistencies between SBOM outputs of different tools [50], which undermine reproducibility and trust.

Limitations of DevSecOps: DevSecOps promotes embedding security controls throughout the software life cycle, including testing and vulnerability analysis. However, it does not provide a systematic approach for managing and verifying the evidence produced by these activities [6], nor does it fully address trust guarantees required by frameworks such as SLSA [43]. For example, in [26], TEEs are employed within CI pipelines, but without producing attestable guarantees for final artifacts, leaving trust assumptions confined to the execution environment rather than the end-to-end pipeline.

Limitations of Existing Systems and Frameworks: Several systems improve specific aspects of supply chain security but fall short of comprehensive, verifiable evidence generation. VeriDevOps integrates automated policy verification into DevOps workflows [38], but does not produce independently verifiable evidence. in-toto [46] focuses on securing SBOM integrity across the supply chain, while related work on distributed builds emphasizes SBOM reproducibility using Merkle trees [23]. However, evaluations show that many SBOM tools do not meet minimum requirements defined by security authorities such as the U.S. National Telecommunications and Information Administration [16]. Sigstore [29] provides artifact signing through developer authentication, ephemeral keys, and transparency logs, but it does not address execution environment attestation or deterministic build guarantees. Its reliance on append-only logs and ephemeral keys also introduces exposure to denial-of-service and key-reuse attacks [29]. Ultimately, without cryptographically verifiable evidence to enforce these security claims, existing frameworks fail to resolve the credibility deficit in modern software ecosystems [22].

In summary, these limitations highlight the need for an integrated approach that (i) enables continuous and verifiable evidence collection across the software life cycle, (ii) integrates deterministic execution guarantees directly into the CI pipeline, and (iii) provides guarantees of authenticity, integrity, and attesta-

tion in distributed software supply chains to replace redundant rebuilds with constant-time verification checks.

4 Trust Model and Trust Mechanisms

In this section, we first describe the Trust Model we use as reference, based on known security practices and standards [9,44,43], including the actors/entities involved and the trustworthiness requirements. Later, we introduce the mechanisms to address those requirements with the proposed evidence-driven protocol.

4.1 Trust Model

We first define the entities (**E**) in the baseline CI pipeline:

E1 - Repository Owner: holds the primary responsibility for defining security policies, including permissions, access controls, and workflow triggers and setting the rules governing the CI pipeline.

E2 - Developers: write and sign commits in the repository.

E3 - CI Administrators: manage the CI infrastructure.

E4 - Version Control System (VCS): stores and versions the source code in a repository.

E5 - Artifact Registry: stores the artifacts.

E6 - CI Pipeline Infrastructure: where the code is built and build artifacts are tested.

E7 - Deployment Environment: The final destination for the artifacts, e.g., package managers, production environment.

We use the following scenarios (**S**) for possible threats, based on existing characterizations of CI pipeline security properties [20,9]:

S1 - Compromised Integrity of Pipeline Processes: Malicious modifications or unintended alterations in the build and test environments can lead to compromised artifacts, for example, from CI administrators or developers potentially introducing vulnerabilities or backdoors, through code or workflow modifications, e.g., the Bybit attack.

S2 - Dependency of Third-Party Components: The use of third-party dependencies in the source repository introduces the risk of vulnerabilities or malicious code within these components, which could affect the final software product, e.g. the ReactJS 2025 vulnerability (CVE-2025-55182)[5].

S3 - Insufficiency of Continuous Evidence and Audit Trails: Insufficient evidence collection and traceability from the CI pipeline can compromise the ability to detect, investigate, or establish accountability if an incident occurs.

To mitigate these threats and establish operational credibility, we derive the following trustworthiness requirements (**TR**) for artifacts produced in CI pipelines:

[5] https://www.cve.org/CVERecord?id=CVE-2025-55182

TR1 - Source Integrity and Authenticity: Ensure that source code and version control data originate from authorized and verified sources.

TR2 - Environment Integrity: Guarantee that build and test environments are isolated, immutable, and operate in a known, verifiable state.

TR3 - Process Integrity: Ensure that pipeline processes execute as defined, without unauthorized modifications or deviations.

TR4 - Artifact and Evidence Integrity and Authenticity: Ensure that CI artifacts and evidence generated during the pipeline are authentic, untampered, retaining their integrity throughout the CI processes.

4.2 Trust Mechanisms

The following mechanisms **(M)** are intended to fulfill the trustworthiness requirements of the trust model by together producing the necessary evidence of the executed CI pipeline, illustrated in Figure 1.

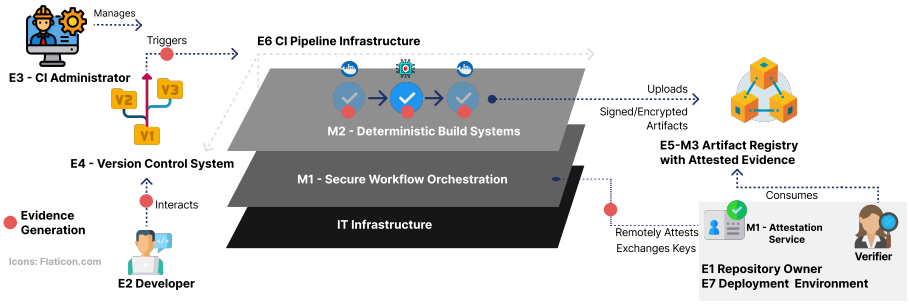

Fig. 1. Diagram of Trust Mechanisms in a CI Pipeline

M1 - Secure Workflow and Orchestration Engines: Modern software development and deployment pipelines rely on complex workflows composed of interdependent tasks that execute in a predefined sequence. Workflow engines programmatically control these processes using a directed acyclic graph (DAG) structure, where nodes represent tasks and edges define dependencies. Each task produces and consumes evidence in the form of inputs and outputs, ensuring that its execution can be validated based on the successful completion of preceding tasks. Pipeline orchestration specifications define the tasks, their relationships, and metadata, including environment configurations, triggers, and runtime requirements, offering a high-level abstraction for creating and managing pipelines. For trustworthy CI/CD pipelines, the orchestration layer plays a role in ensuring authenticity, integrity, and attestation. To support this, pipeline specifications must clearly identify and configure tasks that run within a TEE.

To maintain the integrity, authenticity, and confidentiality of processes running within TEEs, a KMS is essential in the CI pipeline pipeline. The KMS

manages cryptographic keys used for verifying the TEE's identity, enabling remote attestation, and securely signing or encrypting CI outputs, such as build artifacts, test results, or audit logs. Proper key management ensures that only authorized keys are used, allowing the production of verifiable and, when necessary, confidential outputs as evidence that external verifiers can attest or decrypt. Before any CI task can be executed in a TEE, the environment must undergo remote attestation to confirm its identity and integrity.

M2 - Deterministic Build Systems: In a reproducible build, running the same process in the same environment should consistently produce the same results, regardless of when or where it is executed [41]. This reproducibility is especially crucial in scenarios involving TEEs, where independently verifiable build processes are essential to establish trust in the integrity of the system [19]. A fully specified, reproducible build process enables authentication and distribution of the build specification as authenticated evidence. The specification can then be tracked, attested, and used within TEEs. In practice, a declarative build specification could be stored in a trusted registry as part of the CI pipeline. Whenever a build is triggered, the orchestration engine would use the attested specification to recreate the environment within the TEE, ensuring that the build is reproducible, isolated, and verifiable through its secure boot, isolation and attestation mechanisms [40].

M3 - Artifact Registry with Attested Evidence: Once the CI pipeline completes, the produced artifacts are pushed into a secure artifact registry that ensures their integrity, authenticity, and traceability. This registry supports signed and encrypted artifacts, holding and providing the cryptographic proofs of their provenance. Blockchain commitments add an extra layer of security by providing immutable and tamper-proof, publicly auditable records of CI pipeline evidence. By storing cryptographic commitments (e.g., hashes of artifacts or test reports) in a distributed ledger, organizations can ensure the integrity and traceability of their pipelines. Nonetheless, the artifact registry represents the final, crucial step in a secure CI pipeline. Only artifacts accompanied by attested evidence, such as test or security audit reports, should be allowed to be deployed or processed further once verified against a defined policy, safeguarding the integrity of the software supply chain.

Together, these mechanisms enhance the security, traceability, and verifiability of the CI pipeline, establishing trustworthy evidence for the software development environment.

5 Evidence-driven Protocol for Trustworthy CI Pipelines

The following section explains the evidence life cycle and how the protocol processes this data to produce a single, verifiable attestation for the entire CI pipeline.

5.1 Trustworthiness through Evidence

To ensure the trustworthiness requirements are met, we formalize the collection of evidence throughout the CI pipeline. In the context of CI pipelines, evidence refers to the collection of data and artifacts that document each phase of the software development process to evaluate the trustworthiness of the produced software [49]. The use of the word evidence is to have a common abstraction for an argument of a claim with different technologies [4,8], rather than having a unique perfect mathematical proof concept [36].

The evidence life cycle consists of four main stages: Raw, Authenticated, Attested, and Actioned as described in TrustOps [6]. The life cycle of evidence within a CI pipeline is critical to ensuring the trustworthiness of the software life cycle, which has a continuous evolution and is key to providing verifiable and tamper-resistant trustworthy software artifacts. Each action taken using evidence generates new evidence that feeds back into the cycle to continuously enhance trust. Based on [6], we specify the evidence life cycle as follows:

Raw Evidence (E_{Raw}): This stage involves the initial collection of data and logs generated throughout the phases in the CI pipeline. This includes code commits, configuration changes, build logs, test outputs, and metadata such as timestamps, contributor identities, and system environment states.

Authenticated Evidence (e_{auth}): Once raw evidence is collected, it is authenticated by adding a verification mechanism to its source. Authentication involves mechanisms such as developer-signed commits or cryptographically signed build artifacts. This allows commits to be signed by developers, or to ensure that build artifacts are generated in secure environments such as TEEs.

Attested Evidence (e_{att}): After authentication, evidence is subjected to attestation, where its validity is verified in a broader context. Attested evidence builds on authenticated evidence by proving that specific requirements or standards have been met. For example, it means attesting that a build artifact was produced using verified source code and that the test results were generated in a secure, isolated environment. Attestation ensures that evidence can be trusted not only locally, but also by external verifiers, creating a chain of trust throughout the software life cycle.

Actioned Evidence (e_{act}): Finally, actioned evidence refers to evidence that directly influenced decision-making processes or triggered automated actions within the CI pipeline. For instance, only attested test results might trigger the deployment of the software to a production environment or revoke access if a security policy is violated. In this context, actioned evidence represents evidence that was used to ensure that only trusted and verified artifacts progressed, enabling continuous trust and accountability in distributed CI pipelines.

5.2 Protocol using Trustworthiness Mechanisms

The protocol's trustworthiness mechanisms (M1, M2, M3) operate in a strict choreography where each task iteration is executed within a TEE via DBS (D). This ensures that even the policy evaluation (f_{eval}) remains tamper-proof and

reproducible. By "actioning" the prior state e_{att}^{i-1} and binding outputs through a cryptographic hash ($\| \mathcal{H}(e_{act}^{i-1})$), the protocol produces an immutable DAG. This enables post-facto auditing of the end-to-end provenance by tracing blockchain-anchored commitments (c_{ledger}) back to the initial trigger. The process follows Algorithm 1:

Algorithm 1 Evidence-driven Protocol for Trustworthy CI Pipeline

Require: Trigger E_{Raw}^0, Tasks $T_{1..n}$, Keys K, Ref. Environment Env_{ref}, Policies Π.
Ensure: Final commitment c_{ledger}^n and actioned evidence e_{act}^n, or $\perp$.

Init: $e_{auth}^0 \leftarrow f_{auth}(E_{Raw}^0, K_{actor})$; **if** $e_{auth}^0 == \perp$ **return** $\perp$
  $e_{att}^0 \leftarrow e_{auth}^0$; $c_{ledger}^0 \leftarrow f_{commit}(e_{att}^0)$ {Baseline origin state}
for $i = 1 \dots n$ **executing within TEE via DBS (D) do**
  Action: $(a^i, e_{act}^{i-1}) \leftarrow f_{eval}(e_{att}^{i-1}, \Pi)$ {Policy evaluation}
    if $a^i \in \{\text{REJECT}, \text{REVOKE}\}$ **return** (ABORT: Stage i Policy, $\perp$)
  Raw: $E_{Raw}^i \leftarrow D(t_i, e_{act}^{i-1})$ {Reproducible task execution}
  Auth: $e_{auth}^i \leftarrow f_{auth}(E_{Raw}^i \| \mathcal{H}(e_{act}^{i-1}), K_{system})$ {DAG Binding}
    if $e_{auth}^i == \perp$ **return** $\perp$
  Attest: $e_{att}^i \leftarrow f_{attest}(e_{auth}^i, Env_{ref})$; **if** $e_{att}^i == \perp$ **return** $\perp$
  Ledger: $c_{ledger}^i \leftarrow f_{commit}(e_{att}^i)$ {Anchor step to blockchain}
end for
Deploy: $(a^{final}, e_{act}^n) \leftarrow f_{eval}(e_{att}^n, \Pi)$ {Final policy check}
  if $a^{final} \notin \{\text{REJECT}, \text{REVOKE}\}$ **then** $f_{feedback}(c_{ledger}^n, e_{act}^n)$
return $(c_{ledger}^n, e_{act}^n)$

By following the state transitions defined in Algorithm 1, the protocol ensures that every artifact is backed by a verifiable chain of custody. This formal progression directly addresses the identified threat scenarios by preventing the advancement of any task that fails integrity checks.

To mitigate compromised pipeline processes (**S1**), the protocol secures the execution environment and source origins. During execution, secure workflow orchestration (**M1**) isolates tasks within hardware-protected TEEs, verifying the authenticity of commits and source code modifications (**TR1**), and ensures that only verified identities can sign or attest to the outputs, protecting the integrity of the processes and environments (**TR2, TR4**).

To address risks introduced by third-party dependencies (**S2**), the system enforces strict validation and reproducible environments. Secure workflow orchestration (**M1**) requires validation reports for external components before the pipeline can progress, isolating these checks within TEEs. Deterministic build systems (**M2**) further neutralize dependency risks by ensuring builds are reproducible. This combination guarantees that artifacts remain consistent and verifiable (**TR4**), even when incorporating external code.

To counter the insufficiency of continuous evidence and audit trails (**S3**), the protocol establishes a verifiable chain of custody. Secure workflow orchestration (**M1**) structures and schedules tasks to ensure evidence is systematically generated at each pipeline step. The artifact registry (**M3**) then binds this data,

creating a tamper-proof chain of evidence that securely links final artifacts back to their source code and build processes. This fulfills process and artifact integrity requirements (**TR3, TR4**) by ensuring provenance is maintained and fully auditable throughout the continuous integration pipeline.

Consequently, this verifiable chain of evidence, materializing the TrustOps evidence lifecycle [6], enables lightweight artifact verification. Because the entire evidence DAG is cryptographically bound (with evidence produced by TEEs running DBS tasks and attested during the pipeline's execution phase), stakeholders are no longer required to perform redundant, independent source rebuilds to establish trust. Instead, artifact verification is reduced to constant-time ($O(1)$) cryptographic signature and policy checks against the final attested evidence.

6 Protocol Proof of Concept Implementation

To demonstrate practical feasibility, we prototyped an example of a Trustworthy CI pipeline implementation, available in our public repository[6]. Our implementation is a first instantiation of the TrustOps methodology for CI environments [6]. It follows the presented protocol and leverages a combination of technologies to enforce security, verifiability, and reproducibility throughout the software development life cycle. Figure 2 illustrates the implementation.

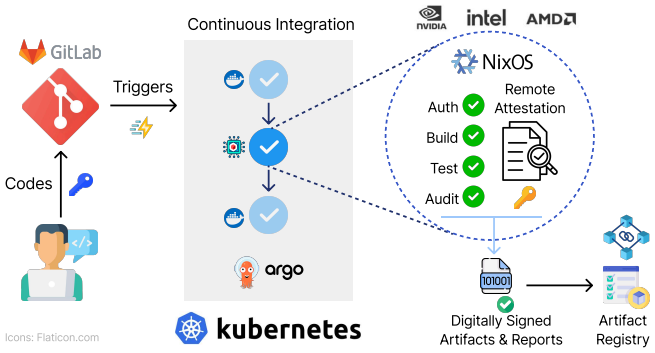

Fig. 2. Evidence-driven Trustworthy CI Pipeline Implementation.

6.1 Workflow Orchestration

The pipeline is built around GitLab, which serves as the central VCS. The pipeline execution is triggered by events like commit pushes or merge requests. Every commit is authenticated through developer-signed commits, ensuring that only trusted identities can introduce changes. Argo Workflows is employed for

[6] Repository: https://github.com/trustops/trustworthy-ci-pipelines/

orchestrating the CI tasks, operationalizing the CI pipeline in sequence according to dependency graphs defined in pipeline configurations. It runs on top of a Kubernetes cluster, which provides scalability and resource management. Tasks, defined by the developers in the pipeline specification, are executed in TEEs, and the Argo engine handles the task orchestration within the Kubernetes cluster. Before execution, each TEE should perform remote attestation to verify it is operating on trusted hardware with an untampered software stack, and exchanges trusted signing keys to establish a secure environment. These tasks have specific labels that trigger their placement and verifiable execution. An illustration of the labeling and pipeline orchestration is shown in Figure 3.

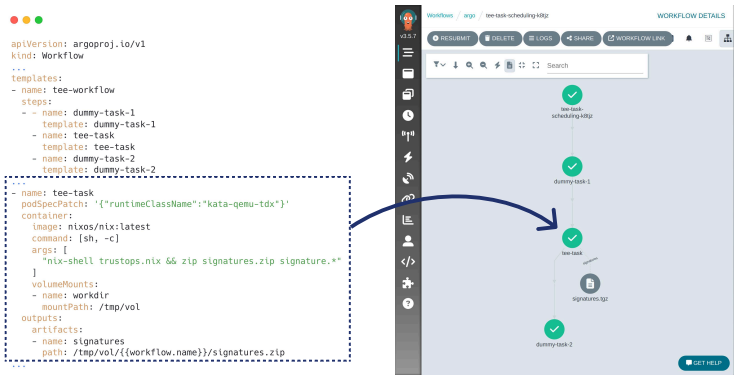

Fig. 3. Argo Workflow example pipeline, as YAML file on the left, comprising one TEE-based task, labeled via the *runtimeClassName* attribute as running inside an Intel TDX VM, and its execution and visualization on the right.

6.2 Declarative and Reproducible Builds

The build process relies on NixOS and Nix, which enforce a declarative, reproducible, and deterministic build environment. It allows to capture the full dependency tree and build instructions in a declarative format, ensuring that every build is reproducible, enabling any stakeholder to independently verify the results. The Nix script running inside the TEE VM authenticates the source code and performs builds and tests in an isolated, reproducible, and verifiable environment. This ensures that the same inputs (e.g., code, dependencies) always produce the same outputs, regardless of when or where the build is performed.

6.3 Testing, Auditing, and Output Signing

Although flexible, tests can also run inside the same reproducible, isolated environment to ensure the integrity of the testing process. Audit logs are generated

and signed, during both build and testing phases, or as part of additional audit processes, such as security audits, creating a transparent and traceable record of authenticated evidence for each task. Upon successful execution, the build artifacts, test results, and audit logs are digitally signed using the TEE's trusted cryptographic keys. These evidence outputs are then uploaded to a secure artifact storage system, ensuring their integrity and providing verifiable proof that they were generated by trusted processes.

6.4 Artifact Storage and Verifiable Evidence

The final step in the pipeline is the secure storage of artifacts in an artifact registry. The artifacts and reports, now digitally signed, are uploaded with cryptographic proof of their provenance. These attestations are then committed into a blockchain, via smart contracts, using the Ethereum Attestation Service[7] standard for efficient off-chain verifications [3], enabling the creation of a tamper-proof chain of evidence, where all actions within the CI pipeline are linked through cryptographic proofs. This approach enables post facto verification of specific policies, allowing retrospective confirmation at each phase in the CI pipeline. This allows external auditors and stakeholders to independently verify the integrity, provenance, and assurance of any artifact produced by the CI pipeline.

7 Protocol Evaluation

To evaluate the properties our system, we modeled a scenario involving trust establishment between a Provider and a Consumer within a software supply chain. The Provider represents or controls **E1** to **E6**, while the Consumer controls **E7**. This scenario can be generalized to various contexts, including intra-organization dependencies, business-to-business software supply chains, and open-source software. We modeled this scenario following our implementation of the Evidence-driven Trustworthy CI pipeline protocol. For testing, to represent the 3 threat scenarios (**S**), we used a frontend interface[8], a backend service[9], and an oracle application[10], each with a suite of tests and known vulnerabilities, detectable by tools such as trivy[11]. We compared two variations of the scenario, using the same pipeline, one using the mechanisms for trustworthiness and one without them.

[7] https://docs.attest.org
[8] https://github.com/aave/interface
[9] https://github.com/ChainSafe/ChainBridge/
[10] https://github.com/smartcontractkit/chainlink
[11] https://github.com/aquasecurity/trivy

7.1 Results and Threat Mitigation

Our results, summarized in Table 1, demonstrate that the system introduces a manageable overhead while significantly enhancing evidence quality and reducing verification complexity for the Consumer.

Table 1. Comparison Results of 10 runs across Three Use Cases

Metric	Oracle Core		Frontend Interface		Backend Service	
	Without	With	Without	With	Without	With
Total Workflow Time (min)	3.32 ± 0.13	6.00 ± 0.24	4.98 ± 0.20	9.00 ± 0.36	9.13 ± 0.37	16.50 ± 0.66
CPU Consumption (s / CPU)	21.60 ± 0.86	39.20 ± 1.57	32.40 ± 1.30	58.80 ± 2.35	59.40 ± 2.38	107.80 ± 4.31
Memory Consumption (s / 100 MiB)	4.80 ± 0.19	9.60 ± 0.38	7.20 ± 0.29	14.40 ± 0.58	13.20 ± 0.53	26.40 ± 1.06
Runtime of Tasks (s)	52.64 ± 2.11	113.20 ± 4.53	78.96 ± 3.16	169.80 ± 6.79	144.76 ± 5.79	311.30 ± 12.45
Evidence Size (B / signature)	N/A	294 ± 0	N/A	294 ± 0	N/A	294 ± 0
Cryptographic Operation Time (ms / Op)	N/A	80 ± 3	N/A	80 ± 2	N/A	80 ± 2
Blockchain Commit Cost (gas)	N/A	393k±11k	N/A	393k±11k	N/A	393k±11k
Blockchain Revocation Cost (gas)	N/A	6.4k	N/A	88k	N/A	88k

These numbers also depend on factors such as the size of the codebase, the complexity of the build process, and the volume of generated artifacts. Despite this, the results show that the performance remains within the same order of magnitude, for the comparison between the two workflows, with and without mechanisms across the three codebases. The TEE-based workflow offers considerable potential for optimization in areas such as orchestration, image management, and networking overhead, especially with better integration into the workflow engine. Additionally, TEE hardware is known to introduce another layer of overhead [21].

By cryptographically binding CI outputs to TEE-attested execution, Consumers can efficiently verify artifact integrity, authenticity, and policy compliance using lightweight checks. To quantify this effect, we simulated a software release process in which the number of Consumers grows by a factor of 10 per month[12], while each release remains approximately constant in size and complexity, representing development patterns such as iterative bug fixes, feature adjustments, or controlled modifications with balanced additions and deletions [11]. In the untrusted CI model, every Consumer independently rebuilds, tests, and audits each release. In contrast, under our Trustworthy CI Protocol, CI execution is performed once per release, and each Consumer performs only cryptographic verification. As shown in Figure 4, the cumulative execution cost in the untrusted model grows rapidly with the number of Consumers, whereas the marginal overhead at the Producer's pipeline in our approach remains nearly constant. This

[12] Many open-source projects may experience even larger user growth rates and more dynamic release cycles [5]

demonstrates that verification costs are effectively amortized across Consumers, enabling scalable trust in highly collaborative software supply chains where artifacts are shared across multiple stakeholders.

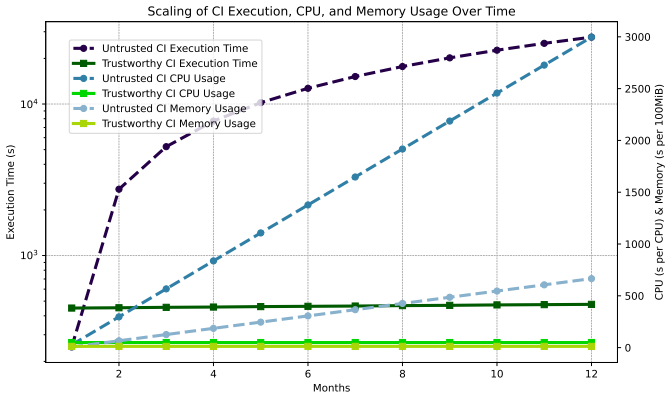

Fig. 4. Scaling of CI execution, CPU, and memory usage over time, for a growth rate of 10 consumers per month and a new release every month. The nature and size of the code base remained approximately constant. The Execution times have their scale on the left axis, while CPU and Memory usage have their scale on the right axis.

To validate threat mitigation, we simulated **S1** by attempting to modify a build script after TEE attestation, resulting in a failed artifact signature verification, and **S2** vulnerabilities from third-party dependencies detected with the SBOM generation. For **S3**, we altered a committed hash in the artifact registry; the registry verification detected the inconsistency within $\leq$100ms.

8 Discussion

There are several points to consider when using a trustworthy CI protocol. A major weakness of TEEs is the reliance on the hardware manufacturer which controls the public key infrastructure (PKI) for the TEE identity keys. Furthermore, several attack scenarios for TEE exist, including memory corruption, as demonstrated in [10]. An important factor of the security and practical effectiveness of TEEs and blockchain commitments relies heavily on the accuracy and integrity of their formal verification to guarantee their desired properties [37]. Additionally, it is important to consider that evidence reproducibility can be affected by non-stable inputs, such as timestamps or environment differences, which prevent full determinism in the build process [33]. Incorrectly defined constraints could lead to evidence that is technically valid but fail to represent the intended computation accurately. Undetected vulnerabilities or malicious code

could compromise the computation's integrity, resulting in incorrect or insecure outcomes, even within the secure confines of a TEE.

One additional strength of our CI protocol is its seamless integration into existing CI/CD pipeline tools. We show that it is possible to adapt and leverage widely adopted platforms such as GitLab, Argo Workflows, and Kubernetes to ensure minimal disruption to current development practices [42]. This approach eliminates the need for extensive custom development or reconfiguration, making the adoption process straightforward and accessible, reducing barriers to entry, while maintaining scalability and performance. Despite challenges like computational overhead and complexity, blockchain remains a practical solution for improving trust in the evidence verification and validation process in software supply chains [17].

Our evaluation showed that our mechanisms can help establish trust in software supply chains and the results demonstrate the viability of this approach. While our implementation introduces modest overhead during release preparation, with no impact in day-to-day development, it significantly reduces verification complexity for Consumers, especially in collaborative environments. However, at organizational scale, additional factors warrant consideration: (1) concurrent pipeline scheduling may create contention for TEE-enabled nodes, requiring capacity planning or queuing strategies; (2) blockchain commitment costs could be reduced through batching attestations across multiple pipeline runs; (3) attestation storage will grow over time, potentially benefiting from off-chain storage with on-chain anchors. Although the modular design of our protocol supports horizontal scaling of the container orchestration layer, a systematic evaluation of large-scale deployment patterns is left to future work. An interesting direction for future work is the integration of build-time integrity guarantees with mechanisms for verifying software composition after artifact generation. Combining TEE-based attestation with confidentiality-preserving SBOM verification [30] could enable end-to-end verifiable supply chains spanning artifact creation through deployment.

9 Conclusion

In this paper, we introduced an evidence-driven protocol that secures continuous integration pipelines by combining DBS with TEEs. This framework generates immutable, blockchain-backed attestations to verify artifact authenticity, integrity, and provenance throughout the software life cycle. Our evaluation across three software use cases demonstrates that this approach mitigates specific supply chain threats with a measurable and manageable computational overhead. By shifting the burden of trust establishment to the pipeline's execution phase, the protocol reduces consumer-side validation to lightweight signature checks. This eliminates the need for redundant, independent rebuilds by consumers, effectively amortizing the initial TEE overhead in multi-consumer scenarios. These results indicate that the proposed framework offers a viable and verifiable trust model for software infrastructure.

References

1. Alkhadra, R., Abuzaid, J., AlShammari, M., Mohammad, N.: Solar winds hack: In-depth analysis and countermeasures. In: 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT). pp. 1–7. IEEE (2021)
2. Bajpai, P., Lewis, A.: Secure development workflows in ci/cd pipelines. In: 2022 IEEE Secure Development Conference (SecDev). pp. 65–66. IEEE (2022)
3. Boi, B., Esposito, C., Seo, J.T.: Ethereum attestation service as a solution for the revocation of hardware-based password-less mechanisms. In: Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing. pp. 553–559 (2024)
4. Bontekoe, T., Karastoyanova, D., Turkmen, F.: Verifiable privacy-preserving computing. arXiv preprint arXiv:2309.08248 (2023)
5. Borges, H., Valente, M.T.: What's in a github star? understanding repository starring practices in a social coding platform. Journal of Systems and Software **146**, 112–129 (2018)
6. Brito, E., Castillo, F., Pullonen-Raudvere, P., Werner, S.: Trustops: Continuously building trustworthy software. In: International Conference on Enterprise Design, Operations, and Computing. pp. 53–67. Springer (2024)
7. Burr, C., Clemencic, M., Couturier, B.: Software packaging and distribution for lhcb using nix. In: EPJ Web of Conferences. vol. 214, p. 05005. EDP Sciences (2019)
8. Castillo, F., Heiss, J., Werner, S., Tai, S.: Trusted compute units: a framework for chained verifiable computations. In: 2025 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). pp. 1–9. IEEE (2025)
9. Chandramouli, R., Kautz, F., Torres-Arias, S.: Strategies for the integration of software supply chain security in devsecops ci/cd pipelines (2024)
10. Chuang, J., Seto, A., Berrios, N., van Schaik, S., Garman, C., Genkin, D.: Tee.fail: Breaking trusted execution environments via ddr5 memory bus interposition. In: 47th IEEE Symposium on Security and Privacy (IEEE S&P '26). IEEE Computer Society (2026), https://tee.fail, to appear
11. Deshpande, A., Riehle, D.: The total growth of open source. In: Ifip international conference on open source systems. pp. 197–209. Springer (2008)
12. Enoiu, E.P., Truscan, D., Sadovykh, A., Mallouli, W.: Veridevops software methodology: Security verification and validation for devops practices. In: Proceedings of the 18th International Conference on Availability, Reliability and Security. pp. 1–9 (2023)
13. Everson, D., Cheng, L., Zhang, Z.: Log4shell: Redefining the web attack surface. In: Proc. Workshop Meas., Attacks, Defenses Web (MADWeb). pp. 1–8 (2022)
14. Fitzgerald, B., Stol, K.J.: Continuous software engineering: A roadmap and agenda. Journal of Systems and Software **123**, 176–189 (2017)
15. Fleischer, F., Busch, M., Kuhrt, P.: Memory corruption attacks within android tees: A case study based on op-tee. In: Proceedings of the 15th International Conference on Availability, Reliability and Security. pp. 1–9 (2020)
16. Halbritter, A., Merli, D.: Accuracy evaluation of sbom tools for web applications and system-level software. In: Proceedings of the 19th International Conference on Availability, Reliability and Security. pp. 1–9 (2024)
17. Helliar, C.V., Crawford, L., Rocca, L., Teodori, C., Veneziani, M.: Permissionless and permissioned blockchain diffusion. International Journal of Information Management **54**, 102136 (2020)

18. Ishgair, E.A., Melara, M.S., Torres-Arias, S.: Sok: A defense-oriented evaluation of software supply chain security. arXiv preprint arXiv:2405.14993 (2024)
19. Jämthagen, C., Lantz, P., Hell, M.: Exploiting trust in deterministic builds. In: Computer Safety, Reliability, and Security: 35th International Conference, SAFECOMP 2016, Trondheim, Norway, September 21-23, 2016, Proceedings 35. pp. 238–249. Springer (2016)
20. Koishybayev, I., Nahapetyan, A., Zachariah, R., Muralee, S., Reaves, B., Kapravelos, A., Machiry, A.: Characterizing the security of github {CI} workflows. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 2747–2763 (2022)
21. Kumar, R., Thangaraju, B.: Performance analysis between runc and kata container runtime. In: 2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT). pp. 1–4. IEEE (2020)
22. Leblanc, A., Robin, J., Ben Rabah, N., Huang, Z., Le Grand, B.: Rethinking cybersecurity ontology classification and evaluation: Towards a credibility-centered framework. In: International Conference on Enterprise Design, Operations, and Computing. pp. 284–299. Springer (2025)
23. Lew, K., Sarker, A., Wuthier, S., Kim, J., Kim, J., Chang, S.Y.: Distributed software build assurance for software supply chain integrity. Applied Sciences **14**(20), 9262 (2024)
24. Lins, M., Mayrhofer, R., Roland, M., Hofer, D., Schwaighofer, M.: On the critical path to implant backdoors and the effectiveness of potential mitigation techniques: Early learnings from xz. arXiv preprint arXiv:2404.08987 (2024)
25. Liu, R., Bobadilla, S., Baudry, B., Monperrus, M.: Dirty-waters: Detecting software supply chain smells. In: Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering. pp. 1045–1049 (2025)
26. Mahboob, J., Coffman, J.: A kubernetes ci/cd pipeline with asylo as a trusted execution environment abstraction framework. In: 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC). pp. 0529–0535. IEEE (2021)
27. Monperrus, M.: Software supply chain security of web3. arXiv preprint arXiv:2511.12274 (2025)
28. Muñoz, A., Rios, R., Román, R., López, J.: A survey on the (in) security of trusted execution environments. Computers & Security **129**, 103180 (2023)
29. Newman, Z., Meyers, J.S., Torres-Arias, S.: Sigstore: Software signing for everybody. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2353–2367 (2022)
30. Nguyen, Van Thanh, e.a.: Trustbom: A scalable architecture for confidentiality-preserving sboms across organizations. In: International Conference on Enterprise Design, Operations, and Computing (2026), submitted for review
31. Niazi, M., Mahmood, S., Alshayeb, M., Riaz, M.R., Faisal, K., Cerpa, N., Khan, S.U., Richardson, I.: Challenges of project management in global software development: A client-vendor analysis. Information and Software Technology **80**, 1–19 (2016)
32. O'Donoghue, E., Reinhold, A.M., Izurieta, C.: Assessing security risks of software supply chains using software bill of materials. In: 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering-Companion (SANER-C). pp. 134–140. IEEE (2024)
33. Pöll, M., Roland, M.: Automating the quantitative analysis of reproducibility for build artifacts derived from the android open source project. In: Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks. pp. 6–19 (2022)

34. Rajapakse, R.N., Zahedi, M., Babar, M.A., Shen, H.: Challenges and solutions when adopting devsecops: A systematic review. Information and software technology **141**, 106700 (2022)
35. Reichert, B.M., Obelheiro, R.R.: Software supply chain security: a systematic literature review. International Journal of Computers and Applications pp. 1–15 (2024)
36. Russinovich, M., Fournet, C., Zaverucha, G., Benaloh, J., Murdoch, B., Costa, M.: Confidential computing proofs: An alternative to cryptographic zero-knowledge. Queue **22**(4), 73–100 (2024)
37. Sabt, M., Achemlal, M., Bouabdallah, A.: Trusted execution environment: What it is, and what it is not. In: 2015 IEEE Trustcom/BigDataSE/Ispa. vol. 1, pp. 57–64. IEEE (2015)
38. Sadovykh, A., Widforss, G., Truscan, D., Enoiu, E.P., Mallouli, W., Iglesias, R., Bagnto, A., Hendel, O.: Veridevops: Automated protection and prevention to meet security requirements in devops. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1330–1333. IEEE (2021)
39. Sánchez-Gordón, M., Colomo-Palacios, R.: Security as culture: a systematic literature review of devsecops. In: Proceedings of the IEEE/ACM 42nd international conference on software engineering workshops. pp. 266–269 (2020)
40. Shepherd, C., Markantonakis, K.: Trusted execution environments (2024)
41. Shi, Y., Wen, M., Cogo, F.R., Chen, B., Jiang, Z.M.: An experience report on producing verifiable builds for large-scale commercial systems. IEEE Transactions on Software Engineering **48**(9), 3361–3377 (2021)
42. Singh, N., Singh, A., Rawat, V.: Deploying jenkins, ansible and kubernetes to automate continuous integration and continuous deployment pipeline. In: 2022 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI). pp. 1–5. IEEE (2022)
43. SLSA, S.: Supply-chain levels for software artifacts (2024)
44. Souppaya, M., Scarfone, K., Dodson, D.: Secure software development framework (ssdf) version 1.1. NIST Special Publication **800**(218), 800–218 (2022)
45. Stengele, O., Droll, J., Hartenstein, H.: Supply-chain-aligned software auditing and usage justification via distributed ledgers. In: 2025 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). pp. 1–5. IEEE (2025)
46. Torres-Arias, S., Afzali, H., Kuppusamy, T.K., Curtmola, R., Cappos, J.: in-toto: Providing farm-to-table guarantees for bits and bytes. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 1393–1410 (2019)
47. Williams, L., Benedetti, G., Hamer, S., Paramitha, R., Rahman, I., Tamanna, M., Tystahl, G., Zahan, N., Morrison, P., Acar, Y., et al.: Research directions in software supply chain security. ACM Transactions on Software Engineering and Methodology **34**(5), 1–38 (2025)
48. Xia, B., Bi, T., Xing, Z., Lu, Q., Zhu, L.: An empirical study on software bill of materials: Where we stand and the road ahead. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). pp. 2630–2642. IEEE (2023)
49. Xiaoyan, W., Shufen, L., Tie, B.: An evidence-driven framework for trustworthiness evaluation of software based on rules. Chinese Journal of Electronics **21**(4), 589–593 (2012)
50. Yu, S., Song, W., Hu, X., Yin, H.: On the correctness of metadata-based sbom generation: A differential analysis approach. In: 2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 29–36. IEEE (2024)