Executable Trust: A Formal Model and Architecture for Verifiable Digital Interactions

Geun-Hyung Kim * and Young Kuen Jang

Posted Date: 30 April 2026

doi: 10.20944/preprints202604.2080.v1

Keywords: executable trust; TrustEvidence; formal trust model; digital trust; verifiable credentials; self-sovereign identity; digital wallet

 doi:10.20944/preprints202604.2080.v1

Article

Executable Trust: A Formal Model and Architecture for Verifiable Digital Interactions

Geun-Hyung Kim [1,*] and Young Kuen Jang [2]

[1] Blockchain Technology R&D Laboratory, Department of Game Engineering, Dong-eui University, Busan, South Korea
[2] Department of Digital Media Engineering, Dong-eui University, Busan, South Korea
* Correspondence: geunkim@deu.ac.kr; Tel.: +82-010-3453-6552

Abstract

Digital trust in online interactions is commonly established through mechanisms such as decentralized identifiers (DIDs), verifiable credentials (VCs), and digital wallets. While these technologies ensure the correctness of individual components, they do not guarantee that an interaction as a whole is trustworthy. This limitation arises because real-world interactions consist of sequences of dependent steps, where inconsistencies may occur even when each step is locally valid. In this paper, we introduce the concept of executable trust, which models trust as a verifiable property of execution across interaction steps. We formalize interactions as sequences of *TrustEvidence* objects that capture both step-level validity and cross-step dependencies. Based on this model, we demonstrate that step-level correctness is insufficient to guarantee interaction-level trust, and we derive a minimal and sufficient condition for establishing end-to-end trust through composable verification and consistency constraints. We further present the Executable Trust Architecture (ETA), which operationalizes the proposed model through components for evidence generation, constraint enforcement, secure communication, and auditability. The feasibility and effectiveness of the approach are validated through scenario-based evaluations covering key trust properties, including authenticity, integrity, privacy, and accountability. The proposed approach provides a systematic foundation for verifying trust in complex digital interactions and supports the design of systems in which trust can be explicitly enforced, evaluated, and audited at runtime.

Keywords: executable trust; TrustEvidence; formal trust model; digital trust; verifiable credentials; self-sovereign identity; digital wallet

1. Introduction

Trust is a fundamental requirement in digital systems, particularly in environments where interactions span multiple entities, domains, and execution steps. In digital wallet ecosystems, trust is typically established through mechanisms such as cryptographic verification, credential validation, and access control policies, which ensure the correctness of individual components.

However, ensuring the correctness of individual components does not necessarily guarantee that an entire interaction is trustworthy, especially in dynamic and cross-domain environments involving multiple execution steps. In such settings, interactions span distributed trust boundaries, where dependencies between steps may introduce inconsistencies that are not captured by isolated verification.

This limitation can be understood as a gap between *step-level correctness* and *interaction-level trust*. Even if each step s_i satisfies a validity condition $verify(s_i)$, the overall interaction I may still violate trust due to inconsistencies across steps. Such issues arise in scenarios such as replay attacks, inconsistent authorization contexts, or tampering with execution logs. These observations highlight the need for models that explicitly capture dependencies across execution steps.

To address this limitation, we introduce the concept of *Executable Trust*, in which trust is defined and validated over the entire execution of an interaction. We formalize this idea through a trust model

 doi:10.20944/preprints202604.2080.v1

based on *TrustEvidence*, which captures the outcome of each execution step along with its associated constraints and linkage to previous steps. This model explicitly distinguishes between step-level validity and interaction-level trust, and defines trust in terms of both correctness and cross-step consistency.

Based on this model, we propose the *Executable Trust Architecture (ETA)*, a layered architecture that operationalizes the formal conditions of trust. Each layer of ETA contributes to the generation, validation, and propagation of *TrustEvidence*, ensuring that trust conditions are consistently enforced throughout the interaction lifecycle.

To evaluate the proposed model, we design a set of experimental scenarios that represent both normal and adversarial conditions in digital wallet systems. These scenarios are mapped to the formal conditions of the model, allowing us to assess whether trust properties are correctly enforced in practice. The results confirm that violations of consistency or constraint conditions can propagate across execution steps, leading to the failure of end-to-end trust.

The contributions of this paper are as follows:

- We formalize trust as a property of execution over an interaction sequence and define it using *TrustEvidence* with explicit conditions for validity, constraints, and cross-step consistency.
- We propose the Executable Trust Architecture (ETA), a layered architecture that operationalizes the formal trust model by enforcing these conditions throughout the interaction lifecycle.
- We validate the proposed model through scenario-based evaluation, showing that violations of consistency or constraints propagate across execution steps and lead to the failure of end-to-end trust.

The remainder of this paper is organized as follows. Section 2 reviews related work on digital trust and credential-based systems. Section 3 presents the formal model of trust based on execution. Section 4 describes the Executable Trust Architecture (ETA) and its layered design. Section 5 provides experimental validation of the model. Section 6 discusses implications, and Section 7 concludes the paper and discusses future work.

2. Related Works

2.1. Foundations of Digital Trust

Trust in digital systems refers to the degree of confidence that one party can place in another based on expectations of authenticity, integrity, and reliability. In technical environments, these expectations are realized through mechanisms that enable secure and verifiable interactions.

Digital trust is typically established through a combination of trust principles and trust mechanisms [1]. Trust principles define the desired properties of a system, such as authenticity, integrity, and accountability, while trust mechanisms provide the technical means to enforce these properties. For example, cryptographic protocols ensure data integrity and authenticity, and authentication systems verify the identity of entities participating in an interaction.

Historically, digital trust has been implemented through centralized models, such as Public Key Infrastructure (PKI) [2], where trusted authorities are responsible for identity issuance and validation. More recently, decentralized approaches have emerged to address the limitations of centralized control. Technologies such as Decentralized Identifiers (DIDs) [3] and Verifiable Credentials (VCs) [4] enable entities to create and manage their own identifiers and credentials in a distributed manner.

These developments provide the foundation for modern digital trust systems, where trust can be established through verifiable data and cryptographic proofs.

2.2. Decentralized Identity and Digital Wallets

Decentralized identity [5–7] has emerged as a key approach for enabling trust in distributed digital environments. Unlike traditional identity systems that rely on centralized authorities, decentralized identity allows entities to create, manage, and control their own identifiers and credentials across multiple domains.

 doi:10.20944/preprints202604.2080.v1

Decentralized Identifiers (DIDs) provide a mechanism for generating and resolving globally unique identifiers without requiring centralized registration. A DID is associated with a DID document that contains cryptographic material and service endpoints, enabling secure and verifiable interactions.

Verifiable Credentials (VCs) extend this model by allowing issuers to create digitally signed claims about a subject. These credentials can be independently verified using cryptographic proofs, ensuring their authenticity and integrity without requiring direct communication with the issuer at the time of verification.

Digital wallets [8] act as the primary interface for managing decentralized identity artifacts. They store cryptographic keys, manage credentials, and enable selective disclosure of information during interactions. Through these capabilities, wallets allow users to control how their identity data is presented and shared.

Together, DIDs, VCs, and digital wallets form a foundational layer for decentralized identity systems, supporting secure, verifiable, and user-centric data exchange.

2.3. Digital Trust Architecture

To support trust in complex digital environments, several architectural frameworks have been proposed to organize trust relationships and system components. These architectures define how trust-related functions are distributed across different layers and entities.

The Trust over IP (ToIP) model [9,10] introduces a layered architecture that separates governance, assurance, and technical infrastructure. This structure supports interoperability across domains by clearly defining responsibilities for identity, credential management, and verification processes.

Zero Trust Architecture (ZTA) [11] adopts a different approach by emphasizing continuous verification of entities and resources. Instead of assuming trust based on network location or prior authentication, ZTA requires that each access request be evaluated based on contextual information and security policies.

Other frameworks, such as Digital Trust Ecosystem Frameworks (DTEF) [12] , focus on coordinating trust relationships across organizations by defining shared policies, governance models, and interoperability requirements.

These architectural approaches provide systematic ways to structure trust in digital systems, enabling scalable and interoperable deployments across multiple domains.

2.4. Trust Verification Methods

A wide range of techniques has been developed to support the verification of trust-related properties in digital systems. These techniques ensure that data, identities, and system states can be validated in a secure and reliable manner.

Cryptographic mechanisms form the foundation of trust verification. Digital signatures are widely used to ensure the authenticity and integrity of data, while public key infrastructures and decentralized key management systems enable the verification of entity identities.

In decentralized identity systems, verification is typically performed through the validation of verifiable credentials, including signature verification, issuer authenticity checks, and revocation status validation.

Privacy-preserving techniques further enhance trust verification by enabling proofs of validity without revealing underlying data. Methods such as zero-knowledge proofs and selective disclosure allow users to demonstrate specific attributes while minimizing information exposure.

Additional mechanisms, including revocation registries, distributed ledgers, and secure communication protocols, contribute to maintaining the integrity and auditability of digital interactions.

2.5. Limitations of Existing Approaches

The reviewed approaches provide complementary capabilities for establishing trust in digital systems. However, they primarily operate at the level of isolated components, focusing on validating individual elements rather than the interaction as a whole.

 doi:10.20944/preprints202604.2080.v1

As a result, they do not provide a unified mechanism for determining whether an entire interaction—spanning identity verification, authorization, execution, and outcome—satisfies trust requirements as a coherent process.

In particular, existing approaches lack mechanisms that (1) link identity and credentials to their usage, (2) capture dependencies across multiple steps, and (3) explain how trust conditions are maintained throughout an interaction.

Consequently, even when all individual components are successfully validated, it remains difficult to ensure that the overall interaction is trustworthy from an end-to-end perspective. These limitations suggest the importance of evaluating trust across complete interaction sequences rather than isolated verification steps.

3. Problem Formulation: Executable Trust

Building on the limitations identified in the previous section, we formalize the problem of evaluating trust across interaction sequences, where trust depends not only on the correctness of individual steps but also on the consistency of their execution.

3.1. From Step-Level Verification to Interaction-Level Trust

Existing approaches to digital trust primarily verify individual components of a system. For example, identities can be authenticated, credentials can be validated, and access decisions can be enforced according to predefined policies. When each component is verified successfully, the system appears to behave correctly.

However, interactions in digital systems do not occur as isolated events. They unfold as sequences of dependent steps, where the outcome of one step affects the conditions under which subsequent steps are executed. As a result, even if each step is locally valid, the interaction as a whole may violate trust requirements due to inconsistencies across steps.

Even when each step is individually verified, the correctness of one step does not guarantee that subsequent steps are executed under consistent conditions. In practice, mismatches between verification context and usage context may lead to violations that are not observable at the step level.

This observation highlights a fundamental gap: step-level correctness does not necessarily imply interaction-level trust. Trust must therefore be evaluated not only at individual verification points, but also across the dependencies that arise during the execution of an interaction.

3.2. Formal Model of Executable Trust

To formally capture interaction-level trust, we model an interaction as a sequence of dependent steps and introduce *TrustEvidence* to represent per-step validity and cross-step dependencies.

Formally, an interaction I can be represented as a sequence of steps:

$$I = (s_1, s_2, \ldots, s_n) \tag{1}$$

where each step s_i corresponds to an operation such as identity verification, credential presentation, or action execution.

In conventional approaches, trust is evaluated at the step level:

$$\forall s_i \in I, \quad \text{verify}(s_i, \mathcal{B}_i) = \mathit{true} \tag{2}$$

However, this condition does not necessarily imply that the interaction as a whole is trustworthy:

$$\bigwedge_{i=1}^{n} \text{verify}(s_i, \mathcal{B}_i) \;\not\Rightarrow\; \text{verify}(I, \mathcal{C}) \tag{3}$$

 doi:10.20944/preprints202604.2080.v1

This limitation arises because certain violations—such as time-of-check-to-time-of-use (TOCTOU) inconsistencies or mismatches between consent scope and actual usage—only become observable when examining the dependencies between steps rather than the steps in isolation.

To address this limitation, we define a sequence-aware representation using *TrustEvidence*. For each step s_i, we define:

$$TE_i = (E_i, H(TE_{i-1}), \mathcal{C}_i) \tag{4}$$

where E_i denotes the evidence generated at step i, $H(TE_{i-1})$ represents the cryptographic linkage to the previous step, and $\mathcal{C}_i$ denotes the set of constraints applicable at that step.

To clarify the relationship between conventional step-level verification and *TrustEvidence*-based verification, we define the following definitions.

Definition 1 (Constraint Model). Let $\mathcal{C}_i$ denote the set of constraints applicable to step s_i. We define:

$$\mathcal{C}_i := (\mathcal{B}_i, \mathcal{X}_i),$$

where $\mathcal{B}_i$ denotes baseline step-level conditions (e.g., cryptographic correctness and credential validity), and $\mathcal{X}_i$ denotes execution-level constraints (e.g., authorization scope, revocation status, temporal validity, and policy constraints).

The global constraint set is:

$$\mathcal{C} := \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}.$$

Definition 2 (Step-Level Validity). We say that

$$\text{verify}(s_i, \mathcal{B}_i) = \textit{true}$$

if step s_i satisfies the baseline conditions $\mathcal{B}_i$.

Step-level verification refers specifically to baseline conditions $\mathcal{B}_i$, excluding execution-level constraints $\mathcal{X}_i$.

Definition 3 (Evidence Validity). Let E_i denote the set of observable outputs at step i, including cryptographic proofs, credential validation results, and verification artifacts.

We say that

$$\text{verify}(E_i) = \textit{true}$$

if E_i is cryptographically valid. This evidence serves as the observable basis for verifying step-level validity under $\mathcal{B}_i$.

Definition 4 (TrustEvidence Validity). For each step s_i, the corresponding *TrustEvidence* object is verified as:

$$\text{verify}(TE_i) := \text{verify}(E_i) \wedge \text{link}(TE_{i-1}, TE_i) \wedge \text{constraint\_ok}(E_i, \mathcal{X}_i). \tag{5}$$

This condition ensures evidence validity, linkage across steps, and enforcement of execution-level constraints.

3.3. Execution State and Cross-Step Consistency

To capture dependencies across steps, we define execution state and preconditions.

Definition 5 (Execution State). We define the execution state derived from a *TrustEvidence* object as:

$$\text{state}(TE_i) := (\textit{status}_i,\ \textit{scope}_i,\ \textit{time}_i,\ \textit{context}_i)$$

where $\textit{status}_i$, $\textit{scope}_i$, $\textit{time}_i$, and $\textit{context}_i$ represent credential status, authorized scope, temporal constraints, and execution context at step i, respectively. We denote by $\mathcal{S}$ the space of possible execution states.

Definition 6 (Precondition). We define the precondition of step s_i as a predicate over execution states:

$$\text{precondition}(TE_i) : \mathcal{S} \to \{\textit{true}, \textit{false}\}$$

 doi:10.20944/preprints202604.2080.v1

The precondition specifies the conditions under which step s_i can be executed, and is evaluated against the execution state derived from TE_{i-1}.

Definition 7 (Cross-Step Consistency). We define cross-step consistency between two consecutive execution steps as:

$$\text{consistent}(TE_{i-1}, TE_i) := state(TE_{i-1}) \models precondition(TE_i). \tag{6}$$

This condition holds if the execution state at step $i-1$ satisfies the precondition of step i.

3.4. Properties of the Model

Proposition 1 (Strict Refinement of Step-Level Verification).

For any execution step s_i, if

$$\text{verify}(TE_i) = true,$$

then

$$\text{verify}(s_i, \mathcal{B}_i) = true.$$

However, the converse does not necessarily hold; that is, there exist cases where

$$\text{verify}(s_i, \mathcal{B}_i) = true \quad \text{but} \quad \text{verify}(TE_i) = false.$$

Proof. The forward implication follows from Definition 4, since $\text{verify}(TE_i)$ requires $\text{verify}(E_i)$, which provides evidence supporting the baseline validity of step s_i under $\mathcal{B}_i$.

To show strictness, consider a case where

$$\text{verify}(s_i, \mathcal{B}_i) = true$$

but an execution-level constraint in $\mathcal{X}_i$ is violated (e.g., revocation or scope constraints). In such a case,

$$\text{constraint\_ok}(E_i, \mathcal{X}_i) = false.$$

Therefore,

$$\text{verify}(TE_i) = false.$$

Therefore, *TrustEvidence*-based verification is a strict refinement of step-level verification. □

Proposition 2 (Detection of Hidden Cross-Step Violations).

There exist execution sequences in which all individual steps satisfy step-level verification, yet the interaction violates cross-step consistency. Formally, there exists an interaction

$$I = (s_1, s_2, \ldots, s_n)$$

such that

$$\forall i,\ verify(s_i, \mathcal{B}_i) = true,$$

but

$$\exists i > 1 \text{ such that } \text{consistent}(TE_{i-1}, TE_i) = false.$$

Proof. Consider two consecutive steps s_{i-1} and s_i such that both satisfy their baseline verification conditions:

$$\text{verify}(s_{i-1}, \mathcal{B}_{i-1}) = true, \quad \text{verify}(s_i, \mathcal{B}_i) = true.$$

Consider a case where the execution state produced by TE_{i-1} violates the precondition required for step s_i, i.e.,

$$state(TE_{i-1}) \not\models precondition(TE_i).$$

 doi:10.20944/preprints202604.2080.v1

Such a situation can arise, for example, when a credential is valid at the time of verification but becomes invalid at the time of use, or when the actual usage exceeds the authorized scope. This provides a concrete instance of an execution sequence satisfying the stated conditions.

By Definition 7, this implies:

$$\text{consistent}(TE_{i-1}, TE_i) = \textit{false}.$$

Thus, there exists an execution sequence in which all individual steps satisfy step-level verification, yet cross-step consistency is violated. □

Implications of Propositions 1 and 2.

Together, Propositions 1 and 2 show that step-level verification is insufficient to guarantee interaction-level trust. While local correctness can be established at each step, violations may arise from dependencies across execution steps.

These results highlight the need for a sequence-aware notion of trust that captures both per-step validity and cross-step consistency, which we formalize in the following theorems.

3.5. Interaction-Level Trust and Counterexamples

We define interaction-level trust as:

$$\text{verify}(I, \mathcal{C}) := \forall i \; verify(TE_i) \wedge \forall i > 1 \; \text{consistent}(TE_{i-1}, TE_i) \tag{7}$$

This definition requires both step-level validity and cross-step consistency. We illustrate the necessity of cross-step consistency using two counterexamples.

Counterexample 1: Time-of-Check to Time-of-Use (TOCTOU) Violation.

Let s_1 denote credential verification and s_2 a subsequent resource access. Suppose the credential is valid at s_1 but revoked before s_2 is executed. If s_2 relies on stale state, both steps may satisfy step-level verification.

However, the interaction violates the requirement that the credential remain valid across steps. This is captured as:

$$state(TE_1) \not\models \text{precondition}(TE_2)$$

and thus

$$\text{consistent}(TE_1, TE_2) = \textit{false}.$$

This illustrates a violation arising from cross-step dependency rather than individual step validity.

Counterexample 2: Scope Violation.

Let s_3 denote the issuance of a consent token with scope R_1, and s_4 a selective disclosure step. Both steps may satisfy step-level verification.

If the disclosed attribute set exceeds the authorized scope, i.e.,

$$attr(s_4) \not\subseteq scope(s_3),$$

the interaction violates the cross-step constraint.

This is captured as:

$$state(TE_3) \not\models \text{precondition}(TE_4),$$

and thus

$$\text{consistent}(TE_3, TE_4) = \textit{false}.$$

Together, these examples show that violations arise from dependencies across steps rather than individual step validity.

 doi:10.20944/preprints202604.2080.v1

3.6. Theoretical Results

Theorem 1 (Strict Separation of Step-Level and Interaction-Level Trust).

Step-level verification does not characterize interaction-level trust. Formally, there exists an interaction $I = (s_1, \ldots, s_n)$ such that

$$\forall i,\ \text{verify}(s_i, \mathcal{B}_i) = \textit{true},$$

but

$$\text{verify}(I, \mathcal{C}) = \textit{false}.$$

Proof.

The result follows from the counterexamples presented earlier. In each case, all steps satisfy baseline step-level verification, while a violation of cross-step consistency occurs:

$$\exists i > 1 \text{ such that } \text{consistent}(TE_{i-1}, TE_i) = \textit{false}.$$

By Definition (7), interaction-level trust requires both per-step validity and cross-step consistency. Therefore,

$$\text{verify}(I, \mathcal{C}) = \textit{false}.$$

This establishes that local cross-step violations propagate to the interaction level. □

Significance of Theorem 1.

Theorem 1 establishes a strict separation between step-level verification and interaction-level trust. It shows that local correctness does not compose into global correctness, as violations may arise from dependencies across execution steps.

This result is significant because it demonstrates that interaction-level trust cannot be characterized by independent verification of individual steps. Instead, it requires reasoning about the relationships between steps, which motivates the sequence-aware model developed in this work.

Theorem 2 (Decomposability of Interaction-Level Trust).

For an interaction $I = (s_1, \ldots, s_n)$, interaction-level trust is equivalent to the conjunction of per-step validity and cross-step consistency:

$$\text{verify}(I, \mathcal{C}) = \textit{true} \iff (\forall i,\ \text{verify}(TE_i) = \textit{true} \ \wedge\ \forall i > 1,\ \text{consistent}(TE_{i-1}, TE_i) = \textit{true}).$$

Proof.

($\Rightarrow$) By Definition (7), interaction-level trust requires that both per-step validity and cross-step consistency hold. Therefore,

$$\forall i,\ \text{verify}(TE_i) = \textit{true} \quad \text{and} \quad \forall i > 1,\ \text{consistent}(TE_{i-1}, TE_i) = \textit{true}.$$

($\Leftarrow$) Conversely, if all $\text{verify}(TE_i)$ and all $\text{consistent}(TE_{i-1}, TE_i)$ hold, then by Definition (7),

$$\text{verify}(I, \mathcal{C}) = \textit{true}.$$

Thus, interaction-level trust can be fully characterized by local and pairwise conditions. □

Significance of Theorem 2.

Theorem 2 shows that interaction-level trust, although defined as a global property over an execution sequence, can be decomposed into independently verifiable local and pairwise conditions.

This establishes that interaction-level trust admits a compositional verification structure.

 doi:10.20944/preprints202604.2080.v1

This result is significant because such decomposability is not guaranteed in general for global properties. It enables modular verification of trust across distributed systems, where each step and transition can be validated independently while still ensuring end-to-end correctness.

Together, Theorems 1 and 2 establish both the limitation and the precise characterization of interaction-level trust.

4. Executable Trust Architecture

4.1. Architectural Overview of Executable Trust

The ETA provides a system-level realization of the formal executable trust model defined in Section 3.2. In particular, ETA operationalizes the *TrustEvidence* structure in Eq. (4) and the associated verification conditions defined in Eqs. (5)–(6).

In this model, trust is defined as a property of execution over an interaction sequence (Eq. (1)), where correctness must be preserved not only at each step but also across dependencies between steps. ETA implements this model by ensuring that evidence generation, constraint enforcement, and execution linkage collectively satisfy the formal conditions without redefining them at the architectural level.

Figure 1 illustrates the layered organization of ETA. The architecture consists of five functional layers that collectively enable the realization of executable trust.

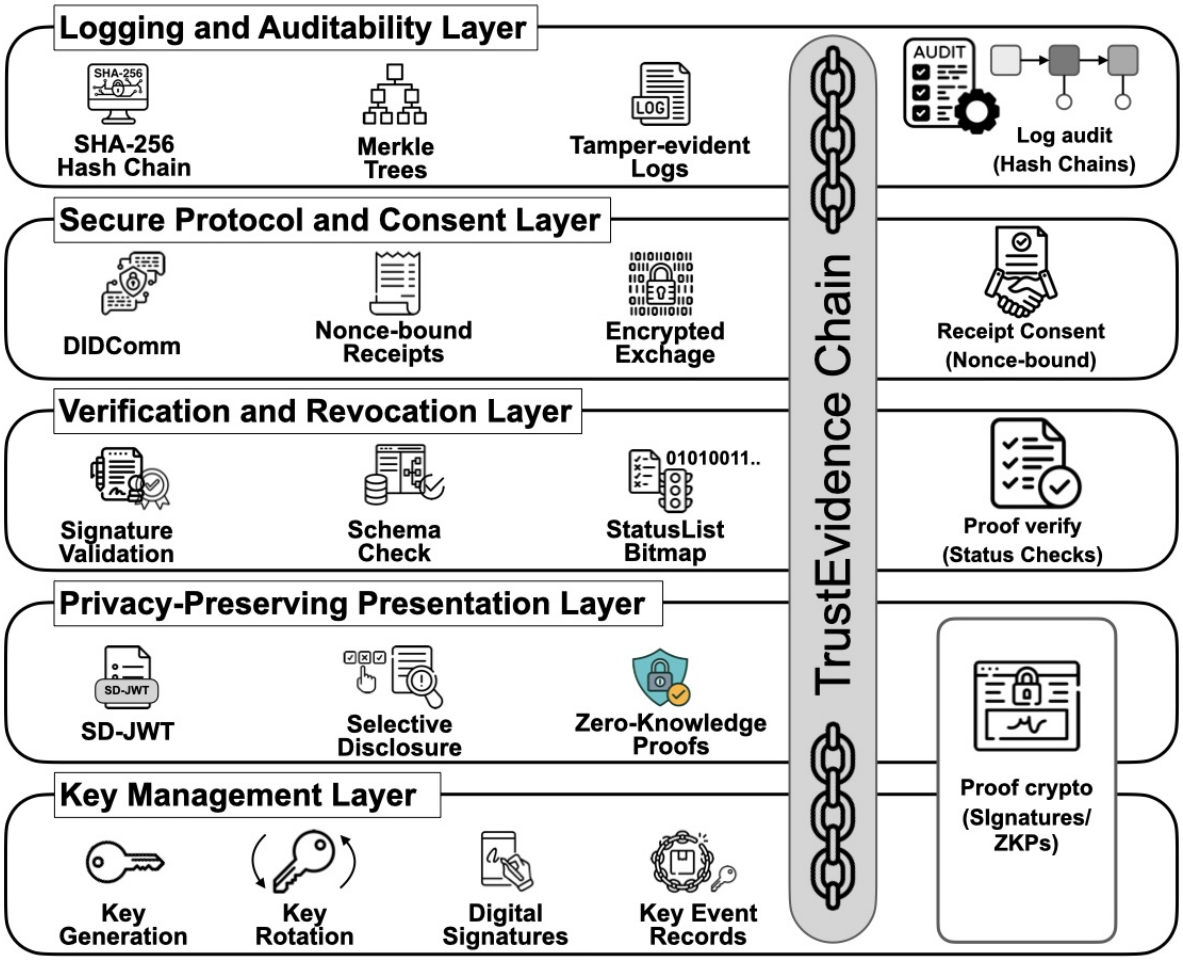

Figure 1. Executable Trust Architecture (ETA) aligned with the formal executable trust model. The architecture supports the generation and validation of *TrustEvidence* across execution steps, ensuring that the verification conditions in Eqs. (5)–(6) are satisfied throughout the interaction lifecycle.

At a high level, the lower layers are responsible for generating the evidence component E_i, while the middle layers enforce constraints $\mathcal{C}_i$ and validate correctness according to Eq. (5). The upper layer

 doi:10.20944/preprints202604.2080.v1

preserves execution continuity by maintaining the linkage $H(TE_{i-1})$, supporting inter-step consistency as defined in Eq. (6).

Specifically, the **Key Management Layer** and the **Privacy-Preserving Presentation Layer** generate cryptographic material and verifiable credentials, forming the evidence component E_i in Eq. (4).

The **Verification and Revocation Layer** enforces constraint validation through $constraint\_ok(E_i, \mathcal{X}_i)$ in Eq. (5), ensuring that credentials satisfy validity conditions.

The **Secure Protocol and Consent Layer** governs communication and enforces authorization conditions, ensuring that each step satisfies the preconditions required for subsequent steps, thus supporting the consistency condition in Eq. (6).

Finally, the **Logging and Auditability Layer** maintains a hash-linked sequence of execution records, implementing the linkage term $H(TE_{i-1})$ in Eq. (4) and enabling traceability across steps.

Through this layered structure, ETA ensures that trust is not evaluated as a collection of independent checks, but as a property of the entire execution sequence, consistent with the formal definition of executable trust in Eq. (7). The architecture is organized into five functional layers. Rather than redefining these layers, we align each layer with a specific component of the formal model, thereby bridging abstract trust definitions and concrete system mechanisms.

4.2. Trust Execution Model and TrustEvidence

Unlike conventional trust architectures that verify individual components in isolation, the ETA defines trust as a property of system execution. In this model, trust is established only when all steps within an interaction are executed correctly and remain consistent with predefined constraints.

Formally, an interaction is represented as a sequence of execution steps (Eq. (1)), and trust is evaluated based on the conditions defined in Eqs. (5)–(6). Rather than treating verification outputs as independent artifacts, ETA models them as structured *TrustEvidence* that captures both per-step validity and inter-step dependencies.

For each execution step s_i, a corresponding *TrustEvidence* object TE_i is generated as defined in Eq. (4). The evidence component E_i encapsulates the observable and verifiable outputs of the system, including:

- **Cryptographic Proof** ($Proof_{\text{crypto}}$): digital signatures and selective disclosure proofs ensuring authenticity and integrity;
- **Consent Receipt** ($Receipt_{\text{consent}}$): cryptographically bound user consent associated with a specific transaction;
- **Verification Result** ($Proof_{\text{verify}}$): validation outcomes including schema compliance, issuer authenticity, and revocation status;
- **Audit Log Entry** (Log_{audit}): tamper-evident execution records maintained via hash chains.

These components do not exist as independent entities; rather, they are integrated within the structured evidence representation E_i and evaluated through the step-level verification condition in Eq. (5). This formulation ensures that each step is validated not only in isolation but also in relation to its execution context.

The execution state $state(TE_i)$ is materialized through the aggregated evidence E_i and system metadata, including credential status, authorization scope, temporal validity, and execution context.

Crucially, ETA extends beyond step-level validation by incorporating cross-step consistency. As defined in Eq. (6), each execution step must satisfy the preconditions derived from the state of the previous step. This enables the detection of violations that cannot be observed at the level of individual steps, such as time-of-check-to-time-of-use inconsistencies or mismatches between consent scope and actual usage.

The overall trust of an interaction is therefore defined as in Eq. (7), where both per-step verification and inter-step consistency must hold. This distinguishes ETA from conventional models, where trust is often inferred from independent verification results without considering their sequential dependencies.

 doi:10.20944/preprints202604.2080.v1

Trust base and scope of the model.

The formal model assumes that the execution platform on which *TrustEvidence* is generated operates correctly. Specifically, ETA adopts the following **trust base**:

- The cryptographic library (e.g., ECDSA P-256 [18], SHA-256 [19], AES-128-GCM [20]) produces correct outputs, assuming that the host environment is not compromised;
- The key storage mechanism (software keystore or hardware-backed secure enclave) protects private keys from extraction;
- The system clock used for expiration and nonce validation is not subject to adversarial manipulation.

Attacks that violate these assumptions (e.g., hardware compromise, OS-level attacks, or physical key extraction) are considered out of scope. ETA is designed to complement a hardware Root of Trust (RoT) rather than replace it. The Key Management Layer interfaces with platform-specific key storage, while the Logging Layer provides tamper-evident records that support higher-level anomaly detection. Integration with hardware attestation mechanisms is considered future work.

Table 1 summarizes the structure of *TrustEvidence* and its relationship to system components and trust properties.

Table 1. Structure of *TrustEvidence* and its Role in the Formal Model.

Component	Symbol	Role in E_i	Generated By	Formal Condition
Cryptographic Proof	$Proof_{\text{crypto}}$	Ensures authenticity and integrity	Key Mgmt., Presentation	$verify(E_i)$
Consent Receipt	$Receipt_{\text{consent}}$	Binds authorization scope	Secure Protocol	$consistent(\cdot)$
Verification Result	$Proof_{\text{verify}}$	Validates schema and status	Verification Layer	$constraint\_ok(E_i, \mathcal{X}_i)$
Audit Log Entry	Log_{audit}	Provides traceability	Logging Layer	$link(\cdot)$
Aggregated Evidence	E_i	Integrated step-level evidence	All Layers	Eq. (5)

4.3. Layered Execution of Trust Mechanisms

The Trust Execution Model defined in Section 4.2 establishes that trust is a property of execution over an interaction sequence, evaluated based on the generation and validation of *TrustEvidence* as defined in Eqs. (4)–(6). To realize this model in practice, the ETA adopts a layered execution structure in which each layer contributes to specific components of the formal model.

Unlike conventional architectures where layers operate as independent functional modules, the layers in ETA form a coordinated execution pipeline. In this pipeline, *TrustEvidence* TE_i is progressively generated, validated, and linked across layers, ensuring that both step-level verification (Eq. (5)) and cross-step consistency (Eq. (6)) are satisfied throughout the interaction lifecycle.

In particular, the layered execution can be understood as a structured process in which: (i) the evidence component E_i is generated, (ii) constraint conditions $\mathcal{C}_i$ are enforced, and (iii) inter-step linkage and consistency are preserved. Through this process, the architecture ensures that the end-to-end trust condition (Eq. (7)) holds.

The following subsections describe how each layer contributes to the execution of the formal trust model.

4.3.1. Key Management Layer

The Key Management Layer provides the cryptographic foundation for the generation of valid evidence E_i within the *TrustEvidence* structure defined in Eq. (4). It ensures that all cryptographic operations required for evidence generation and verification are securely executed.

This layer supports the evaluation of $verify(E_i)$ in Eq. (5) by guaranteeing the correctness and integrity of digital signatures and cryptographic proofs. Secure key generation, storage, and usage

 doi:10.20944/preprints202604.2080.v1

mechanisms—such as hardware-backed keystores and secure enclaves—ensure that private keys remain protected and that cryptographic operations are resistant to unauthorized access.

From the perspective of the formal model, this layer establishes the authenticity and integrity of the evidence component E_i, forming the basis for step-level verification. Without these guarantees, the condition $verify(TE_i)$ in Eq. (5) cannot be reliably satisfied.

4.3.2. Privacy-Preserving Presentation Layer

The Privacy-Preserving Presentation Layer is responsible for generating verifiable credential presentations while enforcing privacy constraints within the *TrustEvidence* structure defined in Eq. (4). Specifically, this layer contributes to the construction of the evidence component E_i and supports the satisfaction of both the validity and constraint conditions in Eq. (5).

To achieve this, the layer employs privacy-preserving cryptographic techniques such as Zero-Knowledge Proofs (ZKPs) [17], BBS+ signatures [16], and Selective Disclosure JWTs (SD-JWT) [14]. These mechanisms enable the holder to disclose only a subset of credential attributes while preserving the ability of the verifier to validate their authenticity and integrity.

From the perspective of the formal model, these techniques serve two key roles. First, they ensure that the generated evidence satisfies the condition $verify(E_i)$ by producing cryptographically verifiable proofs bound to the issuer and the original credential. Second, they enforce privacy-related constraints embedded in $\mathcal{C}_i$, such as data minimization and disclosure policies, thereby contributing to the condition $constraint\_ok(E_i, \mathcal{X}_i)$.

Importantly, privacy preservation is not treated as an independent objective but as an integral component of trust validation. In particular, selective disclosure ensures that only authorized attributes are revealed in accordance with consent and policy constraints, preventing over-disclosure while maintaining verifiability.

As a result, this layer ensures that *TrustEvidence* remains both verifiable and privacy-compliant, enabling the system to satisfy the step-level verification condition in Eq. (5) even under strict data minimization requirements.

4.3.3. Verification and Revocation Layer

The Verification and Revocation Layer enforces constraint validation by implementing the condition $constraint\_ok(E_i, \mathcal{X}_i)$ in Eq. (5). It evaluates whether the evidence generated at each execution step satisfies all required validity conditions.

This includes verification of digital signatures, schema compliance, issuer authenticity, credential expiration, and revocation status using mechanisms such as Status List 2021 and decentralized registries. These checks ensure that only valid and compliant credentials are accepted.

Within the formal model, this layer is responsible for enforcing the constraint component $\mathcal{C}_i$ associated with each *TrustEvidence* instance. By doing so, it ensures that step-level verification does not rely solely on cryptographic correctness but also incorporates policy and lifecycle constraints.

As a result, this layer guarantees that $constraint\_ok(E_i, \mathcal{X}_i)$ holds, which is a necessary condition for satisfying $verify(TE_i)$ in Eq. (5).

4.3.4. Secure Protocol and Consent Layer

The Secure Protocol and Consent Layer ensures that execution steps satisfy the preconditions required for subsequent steps, thereby supporting the cross-step consistency condition defined in Eq. (6).

This layer governs secure communication between entities using protocols such as DIDComm v2 and mutual TLS, ensuring confidentiality and integrity of transmitted data. In addition, it enforces authorization and consent constraints by binding user consent to specific transactions through cryptographically protected tokens.

From the perspective of the formal model, this layer ensures that the state produced at step s_{i-1} is valid for step s_i, enabling the evaluation of $consistent(TE_{i-1}, TE_i)$. The precondition $\text{precondition}(TE_i)$

 doi:10.20944/preprints202604.2080.v1

is evaluated based on the execution state derived from TE_{i-1} and the constraint set $\mathcal{X}_i$. Violations such as consent scope mismatches, unauthorized reuse of credentials, or replayed interactions correspond to failures in satisfying Eq. (6) and are detected at this layer.

Thus, this layer plays a critical role in ensuring that trust is preserved across execution steps, rather than being evaluated solely at isolated points.

4.3.5. Logging and Auditability Layer

The Logging and Auditability Layer maintains the linkage between *TrustEvidence* objects by implementing the hash reference $H(TE_{i-1})$ in Eq. (4) and enabling the evaluation of $link(TE_{i-1}, TE_i)$ in Eq. (5).

Each execution step is recorded as a hash-linked entry, forming an immutable sequence of evidence. This structure ensures that any modification, omission, or reordering of execution steps can be detected, thereby preserving the integrity of the interaction sequence.

In addition, this layer enables reconstruction of the full execution trace, supporting post-hoc validation of the end-to-end trust condition defined in Eq. (7). By providing tamper-evident logs and traceability, it ensures that both step-level verification and cross-step consistency can be independently audited.

From the perspective of the formal model, this layer is essential for enforcing inter-step linkage and ensuring that the evaluation of trust extends beyond individual steps to the entire execution sequence.

4.3.6. Cross-Layer Execution Perspective

While each layer in ETA is responsible for a specific aspect of the trust model, trust is not established within any single layer. Instead, it emerges from the coordinated execution across all layers.

In particular, *TrustEvidence* TE_i is not generated or validated in isolation. Rather, it is progressively constructed as it traverses the layered execution pipeline, where each layer contributes to the evaluation of different components of Eqs. (4)–(6).

The evidence component E_i is produced and refined in the lower layers, constraint conditions $\mathcal{C}_i$ are enforced in the middle layers, and inter-step linkage is maintained in the upper layers. This layered dependency ensures that the output of each layer becomes the input for subsequent layers.

As a result, failures in any layer propagate across the execution sequence, affecting both step-level verification (Eq. (5)) and cross-step consistency (Eq. (6)). This property enables the system to detect violations that cannot be identified within isolated components.

This cross-layer execution perspective highlights that trust in ETA is a property of the entire execution process, rather than of individual system components.

4.4. Mapping ETA Layers to Experimental Validation

To validate the Trust Execution Model defined in Section 4.2, it is necessary to establish a clear correspondence between the formal conditions and the experimental scenarios. In ETA, each layer contributes to specific components of the formal model, and the experimental scenarios are designed to validate these conditions in practice. Table 2 summarizes the mapping between ETA layers, formal conditions, and experimental scenarios.

 doi:10.20944/preprints202604.2080.v1

Table 2. Mapping of ETA Layers, Formal Conditions, and Experimental Scenarios

Layer	Formal Condition	Scenario	Purpose
Key Management	$verify(E_i)$	S2	Validate cryptographic correctness
Privacy-Preserving Presentation	$verify(E_i)$, $constraint\_ok(E_i, \mathcal{X}_i)$	S1	Validate selective disclosure and privacy constraints
Verification & Revocation	$constraint\_ok(E_i, \mathcal{X}_i)$	S3	Validate credential validity and revocation
Secure Protocol & Consent	$consistent(TE_{i-1}, TE_i)$	S5	Validate cross-step consistency and authorization
Logging & Auditability	$link(TE_{i-1}, TE_i)$	S6	Validate integrity of execution trace
Integrated Execution	$verify(I, \mathcal{C})$	S9	Validate end-to-end trust

As shown in Table 2, each experimental scenario is explicitly associated with one or more formal conditions defined in Eqs. (5)–(7). This mapping ensures that the evaluation is not merely functional, but directly validates the correctness of the formal trust model.

In particular, scenarios such as S5 and S9 play a critical role by validating cross-step consistency (Eq. (6)) and end-to-end trust (Eq. (7)), respectively, which cannot be captured by step-level verification alone.

4.5. End-to-End Trust Execution Flow

ETA realizes end-to-end trust as a continuous execution process in which identity, credentials, authorization, and actions are linked across an interaction sequence. Unlike traditional systems, where verification occurs at isolated stages, ETA ensures that trust is preserved and evaluated across the entire interaction lifecycle according to the formal model defined in Eqs. (4)–(7).

Figure 2 illustrates the dynamic execution flow of ETA. Each interaction step in the sequence corresponds to the generation and validation of a *TrustEvidence* instance TE_i, forming a chain of dependent execution states.

A typical interaction flow in ETA consists of the following six stages:

1. **Credential Issuance (Key Management Layer):** An issuer generates and signs a credential using cryptographic keys, producing the initial evidence component E_i associated with the issuance step.
2. **Secure Storage (Digital Wallet):** The credential is stored in the holder's digital wallet, where secure key management ensures the integrity and confidentiality of the stored evidence.
3. **Selective Disclosure (Privacy-Preserving Presentation Layer):** The holder selectively discloses required attributes, generating verifiable proofs that contribute to the construction of E_i while enforcing privacy constraints in $\mathcal{C}_i$.
4. **Consent-Bound Communication (Secure Protocol Layer):** The credential and associated proofs are transmitted through secure protocols, with explicit user consent cryptographically bound to the interaction, ensuring that preconditions for subsequent steps are satisfied.
5. **Verification and Validation (Verification Layer):** The verifier evaluates the validity of the presented evidence by checking signatures, schema compliance, and revocation status, enforcing $constraint\_ok(E_i, \mathcal{X}_i)$ as defined in Eq. (5).
6. **Audit Logging (Logging Layer):** The interaction is recorded in tamper-evident logs, establishing the hash-linked structure $H(TE_{i-1})$ and enabling traceability across execution steps.

 doi:10.20944/preprints202604.2080.v1

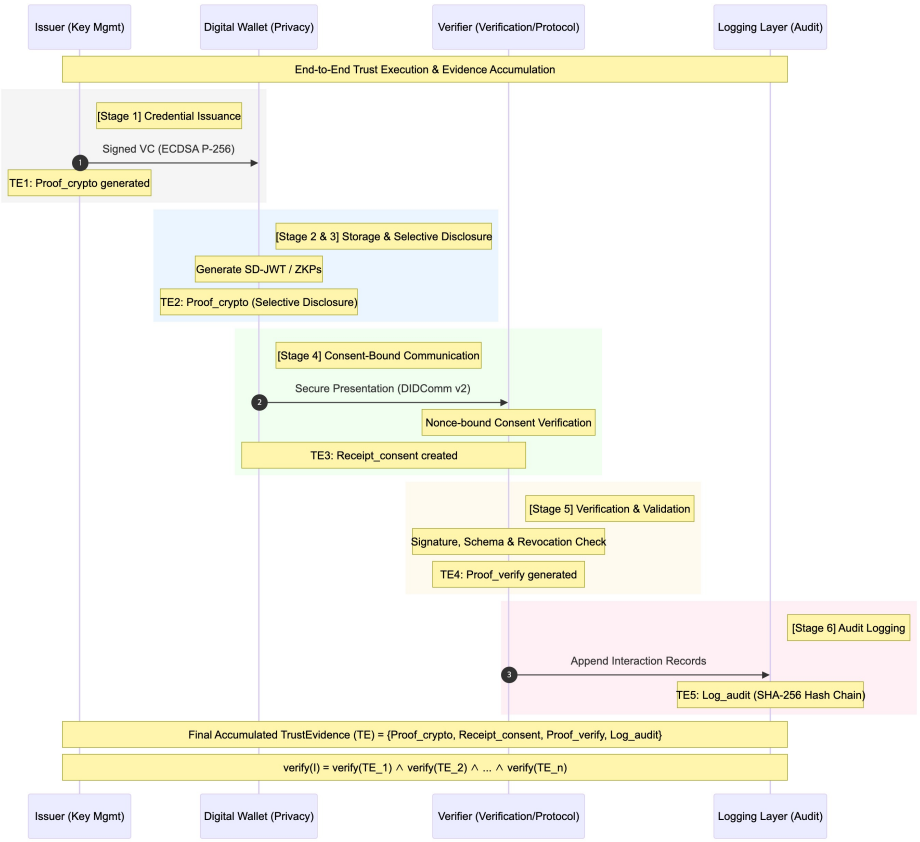

Figure 2. Sequence diagram of the end-to-end trust execution flow. Each interaction step produces a *TrustEvidence* instance TE_i, which is validated and linked across the sequence to ensure step-level correctness and cross-step consistency.

As shown in Figure 2, each stage in the interaction contributes to the generation of a *TrustEvidence* instance as defined in Eq. (4).

At each step, the system evaluates the step-level verification condition $verify(TE_i)$ as defined in Eq. (5), ensuring that the generated evidence is valid and satisfies all associated constraints.

However, trust cannot be established solely through independent verification of steps. ETA therefore enforces cross-step consistency by evaluating $consistent(TE_{i-1}, TE_i)$ as defined in Eq. (6). This ensures that the state produced at each step remains valid for subsequent steps, preventing inconsistencies such as unauthorized reuse of credentials or violations of consent scope.

Through this execution flow, *TrustEvidence* instances are not treated as isolated artifacts, but as elements of a hash-linked execution chain. Any modification or failure at a given step propagates through the sequence via $H(TE_{i-1})$, ultimately affecting the evaluation of the entire interaction.

The overall trust of the interaction is therefore determined by the end-to-end condition $verify(I, \mathcal{C})$ in Eq. (7), which requires both step-level correctness and cross-step consistency. This execution model demonstrates that trust in ETA is not a static property of individual verification steps, but an emergent property of the entire execution sequence.

4.6. Discussion: From Mechanism to Verifiable Trust

The proposed ETA shifts the perspective of trust from a design-time property to an execution-time outcome. Traditional approaches often assume that trust is guaranteed once appropriate mechanisms are implemented, relying on the correctness of individual components. However, as highlighted in Eq. (3), step-level correctness does not imply end-to-end trust in dynamic and cross-domain environments.

ETA addresses this limitation by defining trust as a property of system execution, formalized through the conditions in Eqs. (5)–(7). In this model, trust is established only when both per-step

 doi:10.20944/preprints202604.2080.v1

validity and cross-step consistency are satisfied. This shifts trust from an implicit assumption to an explicitly verifiable condition.

A key contribution of this approach is the introduction of *TrustEvidence* as a unifying abstraction that captures execution outcomes in a structured form (Eq. (4)). By linking cryptographic proofs, consent records, verification results, and audit logs into a single representation, ETA enables trust to be observed, measured, and independently verified.

Furthermore, the layered execution model provides a concrete realization of the formal model. Each layer contributes to specific components of *TrustEvidence*, and their coordinated execution ensures that the conditions for step-level verification (Eq. (5)) and cross-step consistency (Eq. (6)) are satisfied. As a result, trust is not confined to individual components but extends across the entire interaction lifecycle.

This is particularly important in decentralized environments, where interactions span multiple domains and trust boundaries. In such settings, failures often arise not from incorrect individual steps but from inconsistencies across step transitions. By explicitly enforcing the consistency condition, ETA enables the detection of violations that cannot be captured by traditional verification mechanisms.

The experimental results further support this model by demonstrating that violations introduced at any stage propagate through the *TrustEvidence* chain, ultimately affecting the evaluation of end-to-end trust (Eq. (7)). This confirms that ETA operationalizes trust as a measurable and verifiable property under realistic conditions.

Overall, ETA provides a formal-to-system mapping that bridges abstract trust definitions and concrete implementations. This enables the construction of trust-aware digital systems in which trust is not assumed, but continuously generated, validated, and verified through execution.

5. Evaluation and Validation

The objective of this section is to validate the formal trust model introduced in Section 3.2 and realized through the ETA presented in Section 4.

Unlike conventional system evaluations that focus on functional correctness, our evaluation is designed to validate whether the formal conditions defined in Eqs. (5)–(7) hold under both normal and adversarial execution scenarios. Each experiment is therefore interpreted as a test of a specific logical condition rather than a standalone system behavior.

5.1. Evaluation Framework

The evaluation framework is structured according to the formal definition of trust. Specifically, the validation is organized into four categories:

- Step-level correctness: validation of $verify(TE_i)$
- Constraint enforcement: validation of $constraint\_ok(E_i, \mathcal{X}_i)$
- Cross-step consistency: validation of $consistent(TE_{i-1}, TE_i)$
- End-to-end trust: validation of $verify(I, \mathcal{C})$

These conditions define the criteria under which an interaction is considered trustworthy. The evaluation therefore aims to verify whether each condition is satisfied or violated under controlled execution scenarios.

Scenario-to-Model Mapping.

To transition from the formal model to empirical validation, we explicitly map each experimental scenario to the corresponding formal condition it validates. This mapping serves as a bridge between the theoretical definitions and their observable behavior in the system.

Table 3 provides a structured correspondence between the formal conditions and the experimental scenarios. Each scenario is constructed to either satisfy or violate a specific condition, enabling direct validation of the proposed trust model.

 doi:10.20944/preprints202604.2080.v1

Table 3. Mapping of Experimental Scenarios to Formal Trust Conditions

Scenario	Formal Condition	Validation Target	Outcome
S1	$verify(E_i),\ constraint\_ok(E_i, \mathcal{X}_i)$	Selective disclosure	Pass
S2	$verify(E_i)$	Cryptographic proof	Pass
S3	$constraint\_ok(E_i, \mathcal{X}_i)$	Revocation enforcement	Pass
S4	$verify(TE_i)$	Offline verification	Pass
S5	$consistent(TE_{i-1}, TE_i)$	Replay attack	Violation detected
S6	$link(TE_{i-1}, TE_i)$	Log integrity	Violation detected
S7	$constraint\_ok(E_i, \mathcal{X}_i)$	Cache consistency	Violation detected
S8	$verify(TE_i),\ consistent(...)$	Cross-domain interaction	Pass
S9	$verify(I, \mathcal{C})$	End-to-end trust	Violation propagated

5.2. Validation of Step-Level Correctness

This section evaluates whether each execution step produces valid and verifiable evidence according to $verify(TE_i)$.

Normal cases: Scenarios S1–S4 demonstrate that valid evidence satisfies $verify(TE_i)$. Selective disclosure (S1) preserves proof validity while enforcing constraints. Cryptographic verification (S2) ensures authenticity of evidence. Revocation handling (S3) rejects invalid credentials. Offline verification (S4) confirms that correctness is maintained without network access.

Adversarial cases: Tampered inputs violate $verify(E_i)$, resulting in $verify(TE_i) = false$.

Formal interpretation:

$$verify(E_i) = false \Rightarrow verify(TE_i) = false$$

These results confirm that invalid evidence cannot satisfy step-level correctness.

5.3. Validation of Constraint Enforcement

This section evaluates whether constraint conditions are correctly enforced through $constraint\_ok(E_i, \mathcal{X}_i)$.

Normal cases: S1 ensures that only authorized attributes are disclosed.

Adversarial cases: S3 (revoked credentials) and S7 (stale cached data) violate constraint conditions.

Formal interpretation:

$$constraint\_ok(E_i, \mathcal{X}_i) = false \Rightarrow verify(TE_i) = false$$

This confirms that policy violations prevent evidence from being accepted.

5.4. Validation of Cross-Step Consistency

This section evaluates whether execution steps remain consistent across the interaction using $consistent(TE_{i-1}, TE_i)$.

Normal cases: Valid execution sequences maintain consistent transitions between steps.

Adversarial cases: S5 introduces replay attacks, and S6 introduces log tampering, both of which violate cross-step consistency.

Formal interpretation:

$$\exists i \text{ such that } consistent(TE_{i-1}, TE_i) = false \Rightarrow verify(I, \mathcal{C}) = false$$

These results demonstrate that cross-step violations invalidate the entire interaction.

5.5. Validation of End-to-End Trust

This section evaluates whether trust holds across the entire interaction sequence using $verify(I, \mathcal{C})$.

Normal cases: S8 demonstrates that multi-issuer interactions maintain trust conditions.

 doi:10.20944/preprints202604.2080.v1

Adversarial cases: S9 introduces faults that propagate across the execution chain.
Formal interpretation:

$$verify(I, \mathcal{C}) = \bigwedge_i verify(TE_i) \wedge \bigwedge_{i>1} consistent(TE_{i-1}, TE_i)$$

Violations in any step propagate through the *TrustEvidence* chain, leading to failure of end-to-end trust.

5.6. Performance Evaluation

To assess practical feasibility, we measure the computational overhead associated with enforcing the formal trust conditions.

The results show that the average end-to-end latency is approximately 1.5 ms. All adversarial scenarios were detected with 100% accuracy, while no false positives were observed for valid execution cases.

This indicates that the validation mechanism is both sound and efficient.

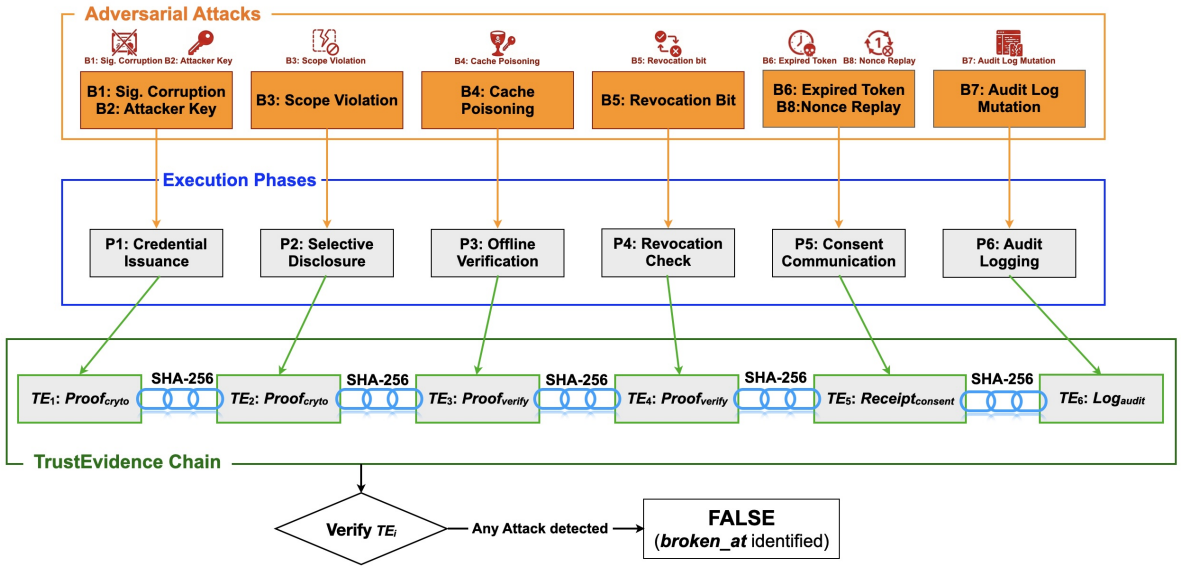

Figure 3. End-to-end trust validation in ETA. Each execution step generates a *TrustEvidence* instance $TE_i = (E_i, H(TE_{i-1}), \mathcal{C}_i)$, validated through $verify(TE_i)$ and consistency checks. The final trust decision is determined by $verify(I, \mathcal{C})$.

Validation Summary.

- $\forall i,\ verify(TE_i)$ holds for valid execution
- $\forall i,\ constraint\_ok(E_i, \mathcal{X}_i)$ is enforced
- Violations of $consistent(TE_{i-1}, TE_i)$ are detected
- $verify(I, \mathcal{C})$ holds if and only if all conditions are satisfied

These results provide empirical support for the correctness of the formal trust model defined in Section 3.2.

6. Discussion

The proposed ETA shifts the notion of trust from a design-time assumption to a property that must be evaluated throughout system execution. This shift has important implications for the design and analysis of digital systems.

First, the results highlight a structural limitation of existing trust mechanisms. While conventional approaches ensure correctness at the level of individual components, they do not account for dependencies across execution steps. As formalized in Section 3 and validated in Section 5, trust violations

 doi:10.20944/preprints202604.2080.v1

may arise from inconsistencies between steps even when all local verifications succeed. This suggests that trust cannot be fully characterized by isolated verification alone.

Second, the formal model provides a general framework for reasoning about interaction-level trust. In particular, the compositional structure established in Theorem 2 shows that a global trust property can be evaluated through local and pairwise conditions. This enables modular verification in distributed and cross-domain environments, where centralized validation is often infeasible.

Third, the notion of *TrustEvidence* offers a unified abstraction for capturing execution outcomes. By integrating evidence, constraints, and linkage, the model allows trust to be explicitly evaluated and traced across the entire interaction lifecycle. This unification provides a foundation for systematic and auditable trust management.

Despite these contributions, several limitations remain. The current model assumes a trusted execution environment and does not explicitly address adversarial conditions such as compromised hosts or malicious verifiers. In addition, scalability issues related to maintaining and verifying long execution chains require further study.

Overall, the proposed approach suggests that trust should be treated as a continuous and composable property of execution, rather than a one-time verification outcome.

7. Conclusion

This paper introduced the ETA, a framework for defining and validating trust as a property of system execution. We showed that step-level verification is insufficient to guarantee interaction-level trust, and formalized trust in terms of per-step validity and cross-step consistency.

Based on this model, we presented a layered architecture that operationalizes these conditions through the generation and validation of *TrustEvidence*. Scenario-based evaluation demonstrated that the system correctly enforces trust conditions and detects violations across execution steps.

These results establish that trust can be treated as a verifiable and composable property of execution, providing a foundation for trust-aware digital systems.

Future work will focus on extending the model to stronger threat models and supporting scalable and automated verification mechanisms.

References

1. Saveljeva, J., and Volkova, T. (2025). A Survey on Digital Trust: Towards a Validated Definition. *Digital*, 5(2), 14.
2. International Telecommunication Union (2019). *Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks* (ITU-T Recommendation X.509). Available online: https://www.itu.int/rec/T-REC-X.509/en (accessed on 10 April 2026).
3. World Wide Web Consortium (W3C) Recommendation (2022). Decentralized Identifiers (DIDs) v1.0: Core architecture, data model, and representations.
4. World Wide Web Consortium (W3C) Recommendation (2025). Verifiable Credentials Data Model v2.0.
5. Allen, C. (2016). *The Path to Self-Sovereign Identity*. Life with Alacrity. Available online: https://www.lifewithalacrity.com/article/the-path-to-self-sovereign-identity/ (accessed on 23 April 2026).
6. Krul, E., Paik, H., Ruj, S., and Kanhere, S. S. (2024). SoK: Trusting Self-Sovereign Identity. Proceedings on Privacy Enhancing Technologies, 2024(3), 297–313.
7. World Economic Forum. (2023). *Reimagining Digital ID: New Models for a Digital World*. Insight Report, June 2023. Available online: https://www.weforum.org/reports/reimagining-digital-id-new-models-for-a-digital-world/ (accessed on 23 April 2026).
8. European Commission. (2026). *European Digital Identity Wallet Architecture and Reference Framework v2.8.0*. Technical Report.
9. Trust Over IP Foundation. (2022). ToIP technology architecture specification V1.0 (Public Review Draft 1). Linux Foundation / Joint Development Foundation Projects. Available online: https://trustoverip.github.io/TechArch/ (accessed on 20 April 2026)

 doi:10.20944/preprints202604.2080.v1

10. Trust Over IP Foundation. (2021). ToIP governance architecture specification V1.0. Linux Foundation / Joint Development Foundation Projects. Available online: https://trustoverip.org/wp-content/uploads/ToIP-Governance-Architecture-Specification-V1.0-2022-12-21.pdf (accessed on 20 April 2026)
11. Rose, S., Borchert, O., Mitchell, S., and Connelly, S. (2020). Zero trust architecture (NIST Special Publication 800-207). National Institute of Standards and Technology. Available online: https://doi.org/10.6028/NIST.SP.800-207 (accessed on 20 April 2026)
12. ISACA. (2024). Digital trust ecosystem framework (DTEF). ISACA. Available online: https://www.isaca.org/digital-trust (accessed on 20 April 2026)
13. Curren, S., Lundkvist, C., and Hardman, D. (2022). *DIDComm messaging specification v2.0*. Decentralized Identity Foundation (DIF). Available online: https://identity.foundation/didcomm-messaging/spec/v2.0/ (accessed on 20 April 2026)
14. Fett, D., Yasuda, K., and Campbell, B. (2025). *Selective disclosure for JSON Web Tokens (SD-JWT)* (RFC 9901). Internet Engineering Task Force. Available online: https://doi.org/10.17487/RFC9901 (accessed on 20 April 2026)
15. Sporny, M., Longley, D., Prorock, M., and Alkhraishi, M. (2025). *Bitstring status list v1.0*. W3C Recommendation. World Wide Web Consortium. Available online: https://www.w3.org/TR/vc-bitstring-status-list/ (accessed on 20 April 2026)
16. Looker, T., Kalos, V., Whitehead, A., and Lodder, M. (2024). *The BBS signature scheme* (Internet Draft draft-irtf-cfrg-bbs-signatures). Internet Research Task Force. Available online: https://datatracker.ietf.org/doc/draft-irtf-cfrg-bbs-signatures/ (accessed on 20 April 2026)
17. Fiat, A., and Shamir, A. (1987). How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko (Ed.), *Advances in cryptology—CRYPTO* 1986 (Lecture Notes in Computer Science, Vol. 263, pp. 186–194).
18. National Institute of Standards and Technology. (2023). *Digital signature standard (DSS)* (FIPS Publication 186-5). U.S. Department of Commerce.
19. National Institute of Standards and Technology. (2015). *Secure hash standard (SHS)* (FIPS Publication 180-4). U.S. Department of Commerce.
20. National Institute of Standards and Technology. (2023). *Advanced encryption standard (AES)* (FIPS Publication 197). U.S. Department of Commerce.