

International Journal Research Publication Analysis

Page: 01-10

BLOCKCHAIN-DRIVEN DIGITAL ASSET EXCHANGE PLATFORM

***Prajwal R., Pranav V. Bajikar, Prasad R., Rajeshwar S.**

Department of Computer Science and Engineering Jyothy Institute of Technology Bengaluru,
India.

Article Received: 16 September 2026

Article Revised: 06 October 2026

Published on: 26 October 2026

***Corresponding Author: Prajwal R.**

Department of Computer Science and Engineering Jyothy Institute of
Technology Bengaluru, India.

DOI: <https://doi-doi.org/101555/ijrpa.4477>

ABSTRACT

Digital asset custody is split across wallets, multi-signature contracts, and off-chain fraud monitors that do not act on one another, leaving a gap between detecting a suspicious transfer and preventing it. This paper presents the implementation of AI Smart Vault, a unified platform that closes that gap. A Solidity vault contract enforces multi-signature approval, while an off-chain risk engine scores every proposed transaction from amount, frequency, and behavioral features using an Isolation Forest combined with deterministic rules. The score reaches the contract as an EIP-712 signed attestation, and the contract maps it to a risk tier that raises the number of required approvals or imposes a time lock. We describe the architecture, contract design, risk model, data pipeline, dashboard, and security hardening, and we define a reproducible protocol for evaluating gas cost, detection quality, and attack resistance.

KEYWORDS: Blockchain, smart contracts, multi-signature, risk scoring, Isolation Forest, EIP-712, Ethereum, fraud detection, digital asset custody.

I. INTRODUCTION

Self-custody of digital assets shifts the burden of security from institutions to individuals and teams. The survey in [1] examined eleven recent works on blockchain asset management, multi-signature security, and AI-driven monitoring, and concluded that the field is fragmented: wallets execute transfers [2], [3], authentication schemes secure identity [4], multi-signature constructions share control [5], and machine-learning detectors flag suspicious activity [6], yet none of these components governs the others.

Four consequences follow. First, users juggle several tools to reach complete control. Second,

multi-signature approval uses a fixed m -of- n rule that ignores how risky a particular transfer is. Third, fraud detectors observe but cannot block, because they sit outside transaction execution. Fourth, smart-contract defects remain a persistent source of loss [7]–[9], and few platforms give users intuitive, real-time visibility.

This paper turns the architecture proposed in [1] into a working system. Our contributions are:

1. A vault contract whose approval threshold is a function of a transaction-level risk score, so that detection directly controls execution;
2. A trust-minimized bridge between the off-chain risk engine and the contract, based on eip-712 signed attestations [10] with nonce, expiry, and fail-safe defaults;
3. A hybrid risk model that fuses an isolation forest [11] with deterministic rules and returns human-readable reason codes; and
4. An evaluation protocol covering gas cost, scoring latency, detection quality, and adversarial scenarios.

Section II states the design goals, Section III the architecture, Sections IV and V the contract and risk engine, Section VI the supporting services, Section VII the security analysis, Section VIII the evaluation protocol, and Sections IX and X the limitations and conclusions.

II. DESIGN GOALS

Each research gap identified in [1] is translated into a testable requirement:

- 1) Unification.** Custody, approval, risk analysis, and monitoring operate as one system with one user interface.
- 2) Adaptivity.** The number of required approvals varies with transaction risk rather than staying constant.
- 3) Enforceability.** A risk verdict must change what the contract permits; it cannot be advisory.
- 4) Fail-safe behavior.** If the risk service is unavailable, stale, or forged, the vault must degrade to its strictest policy, never its weakest.
- 5) Auditability.** Every proposal, score, approval, and execution is recorded on-chain and mirrored off-chain.
- 6) Usability.** Users see pending approvals, risk levels, and alerts without reading raw transactions.

III. SYSTEM ARCHITECTURE

AI Smart Vault is organized in four layers (Fig. 1). The presentation layer is a React decentralized application that talks to the user's wallet and signs transactions through ethers.js. The application layer contains two services: a Python risk engine that scores proposals, and an indexer that listens to contract events, persists them, and raises alerts. The blockchain layer holds a factory and per-vault contracts written in Solidity 0.8.x on top of audited OpenZeppelin primitives [13]. The data layer is a PostgreSQL database for off-chain metadata, features, and logs, consistent with the hybrid-storage pattern described in [1] and [3].

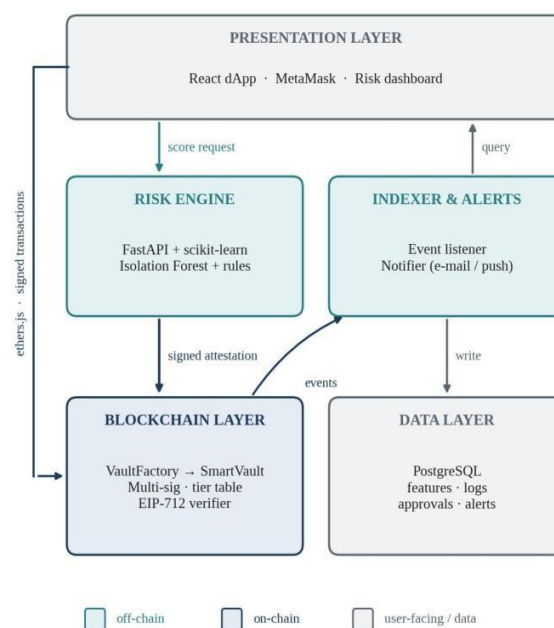


Fig. 1. Layered architecture of AI Smart Vault. Teal components run off-chain; navy components run on-chain.

The seven modules of the survey architecture [1] map onto this implementation as shown in **Table I. MAPPING OF SURVEY MODULES TO IMPLEMENTATION.**

Module [1]	Implementation
Authentication and access control	ECDSA wallet signatures; <i>Owner</i> and <i>Signer</i> roles stored in the vault contract
Vault management	<i>VaultFactory</i> deploys one <i>SmartVault</i> per vault; signers and <i>m</i> are updatable only through a vault proposal
Transaction management	propose, approve, execute, cancel; full event log
AI-based risk analysis	FastAPI risk engine; Isolation Forest plus rules
Approval and multi-signature	On-chain tier table maps score <i>R</i> to required approvals
Smart-contract execution	Guarded transfer with reentrancy protection and time lock
Database and monitoring	PostgreSQL, event indexer, e-mail and push alerts, dashboard

IV. SMART CONTRACT IMPLEMENTATION

A. Vault Factory and Roles

VaultFactory deploys a minimal SmartVault for each group of owners, initialized with the signer set S ($|S| = n$), a base threshold m with $2 \leq m \leq n$, and the address of the risk oracle. Role checks use a signer mapping rather than a single privileged owner, so no individual key can alter policy. Changes to S , m , or the oracle address are themselves transactions that must reach the full threshold and a time lock, which prevents a single compromised signer from quietly weakening the vault.

B. Transaction Lifecycle

A transaction passes through four states: *Proposed*, *Scored*, *Approved*, and *Executed* (or *Cancelled*). Fig. 2 shows the end-to-end flow. When a signer calls `propose(to, value, data)`, the contract assigns an identifier and emits `Proposed`. The indexer forwards it to the risk engine, which returns a signed score. Only after a valid attestation is stored can approvals count toward execution score R to required approvals. Guarded transfer with reentrancy protection and time lock PostgreSQL, event indexer, e-mail and push alerts, dashboard

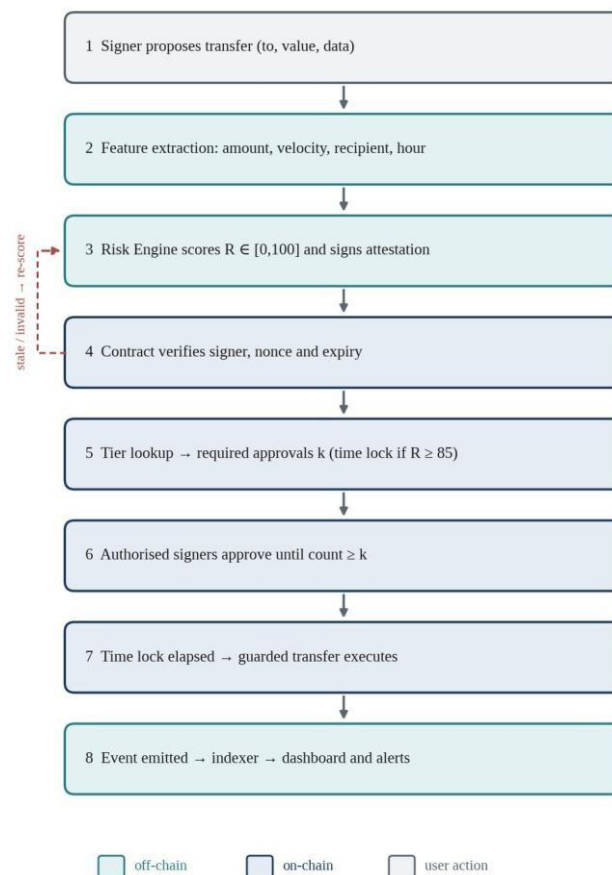


Fig. 2. Transaction lifecycle from proposal to execution. Invalid or stale attestations return the transaction to re-scoring.

C. Risk Attestations

The risk engine signs a typed structure containing the transaction identifier, the score, a per-transaction nonce, and an expiry timestamp. The contract recomputes the EIP-712 digest and recovers the signer with ECDSA, accepting the attestation

only if it was produced by the registered oracle, has not expired, and carries the expected nonce. Binding the attestation to a single transaction identifier and nonce prevents replay across transactions and vaults, and the expiry bounds the window in which a stale score can be used. Listing 1 shows the core logic.

```

struct RiskAttestation {
    uint256 txId; uint8 score;
    uint256 nonce; uint64 expiry;
}

function requiredApprovals(uint8 s)
    public view returns (uint8) {
    if (s < 30) return m;
    if (s < 60) return m + 1 <= n ? m + 1 : n;
    return n; // s >= 60
}

function submitAttestation(
    RiskAttestation calldata a,
    bytes calldata sig) external {
    require(block.timestamp <= a.expiry);
    require(a.nonce == nonces[a.txId]++);
    address r = _hashTypedDataV4(
        keccak256(abi.encode(TYPEHASH, a.txId,
            a.score, a.nonce, a.expiry)))_recover(sig);
    require(r == riskOracle);
    txs[a.txId].score = a.score;
    if (a.score >= 85)
        txs[a.txId].executeAfter = now + 24 hours;
}

```

Listing 1. Attestation check and tier lookup (abridged).

D. Risk-Adaptive Thresholds

The contract converts the score into a policy using the tier table (Table II). Because the table lives on-chain, a compromised off-chain service can influence only the score it reports; it cannot lower the approvals a given tier requires, and the strictest tier always requires every signer.

TABLE II: RISK TIERS AND ENFORCED POLICY

Tier	Score R	Approvals required	Extra control
Low	0–29	m	None
Medium	30–59	$\min(m+1, n)$	Alert to all signers Alert, reason codes
High	60–84	n (all signers)	shown
Critical	85–100	n (all signers)	24 h time lock; any signer may cancel

E. Security Hardening

Transfers follow the checks-effects-interactions pattern and are protected by a reentrancy guard. Transaction status is updated before any external call. Arithmetic uses Solidity 0.8 checked operations. Without a valid attestation a transaction is treated as $R = 100$, so an oracle outage delays transfers but never weakens approval. An optional emergency path allows execution without an attestation only with all signers and the 24 h time lock. These choices respond to the attack classes catalogued in [9] and the vulnerability themes surveyed in [7] and [8].

V. AI RISK ENGINE

A. Feature Engineering

The engine computes the features in Table III from the vault's transaction history and the proposed transfer. They correspond to the amount, frequency, and behavior signals identified in [1] as the basis for adaptive control.

TABLE III: FEATURES USED FOR RISK SCORING

ID	Definition
f_1	Amount ratio: transfer value divided by current vault balance
f_2	Amount deviation: z-score against the rolling 30-day mean of outgoing transfers
f_3	Velocity: number of proposals in the last 1 h and 24 h
f_4	Cumulative 24 h outflow divided by balance
f_5	New-recipient flag: 1 if the address has never received from this vault
f_6	Hour deviation: circular distance from the proposer's typical hours
f_7	Recipient watch-list flag from a maintained blocklist

B. Hybrid Scoring Model

An Isolation Forest [11] isolates anomalies by random partitioning; points that are isolated in few splits are anomalous.

For a feature vector x and sample size N , its anomaly score is

$$s(x, N) = 2^{-E[h(x)] / c(N)} \quad (1)$$

where $E[h(x)]$ is the mean path length across trees and $c(N)$ is the average path length of an unsuccessful search in a binary tree, used for normalization. We take $s_{iso} = s(x, N) \in (0, 1)$.

A rule score $s_{rule} \in [0, 1]$ is the clipped weighted sum of binary triggers (for example f_1 above a ratio limit, a velocity burst, or a new recipient combined with a large amount). The final score is

$$R = \text{round}[100 (0.6 s_{\text{iso}} + 0.4 s_{\text{rule}})] \quad (2)$$

with an override of $R = 100$ whenever $f_7 = 1$. Mixing a learned score with explicit rules keeps behavior predictable for auditors while still capturing patterns that fixed rules miss, which is the advantage over rule-only monitoring reported in [6]. The model is implemented with scikit-learn [12].

C. Cold Start, Retraining, and Explanations

A new vault has no history, so for its first transactions the engine relies on rules and a global prior fitted on pooled, Transfers follow the checks-effects-interactions pattern and are protected by a reentrancy guard. Transaction status is updated before any external call. Arithmetic uses Solidity 0.8 checked operations. Without a valid attestation a transaction is treated as $R = 100$, so an oracle outage delays transfers but never weakens approval. An optional emergency path allows execution without an attestation only with all signers and the 24 h time lock. These choices respond to the attack classes catalogued in [9] and the vulnerability themes surveyed in [7] and [8]. anonymized features; per-vault models take over after a minimum history is reached and are retrained on a schedule to follow behavioral drift. Each response includes reason codes naming the features that contributed most (for instance, *large amount versus 30-day mean* or *first-time recipient*), which the dashboard displays beside the tier.

VI. SERVICES, DATA, AND DASHBOARD

The risk engine exposes a single authenticated endpoint that accepts a proposal identifier and returns the score, reason codes, and the signed attestation; the signing key is held in a managed

key store and is never present in application code. The indexer subscribes to Proposed, Scored, Approved, Executed, and Cancelled events through JSON-RPC, writes them idempotently keyed by transaction hash and log index, and recomputes per-vault aggregates used as model features.

PostgreSQL stores vaults, signers, proposals, scores with Precision, recall, F1 False-positive rate Held-out labeled data; injected anomalies bReplay of benign vault histories Replay, stale, forged, outage, Exceed rule-only baseline Low enough to avoid alert fatigue

reason codes, approvals, and alerts. The React dashboard addresses the user-centric monitoring gap in [1] with four views: pending approvals with the required count k , a risk gauge with reason codes, a chronological audit trail, and alert preferences. Wallet interaction uses ethers.js, so private keys never leave the user's wallet.

We assume an adversary who may control fewer than m signer keys, may compromise the risk service, and can observe all on-chain data. Table IV summarizes the main threats.

TABLE IV THREATS AND MITIGATIONS.

Threat	Mitigation
Attestation replay	Per-transaction nonce and chain-bound EIP-712 domain separator
Stale score reuse	Expiry timestamp checked on-chain; re-score on expiry
Forged or compromised oracle	Tier table fixed on-chain; oracle can only report scores; key in managed store; optional 2-of- k oracle quorum
Oracle outage	Missing attestation treated as $R = 100$
Reentrancy	~~Reentrancy~~ guard; state updated before transfer
Policy downgrade by a signer	Signer, threshold, and oracle changes require all signers and a time lock
Score evasion by slow drift	Cumulative 24 h outflow feature f_i and periodic retraining

VIII. EVALUATION PROTOCOL

We specify the evaluation so results are reproducible. Contracts are tested with Hardhat and Chai, supplemented by fuzz and property-based tests for invariants such as executed value never exceeds balance and no execution without k valid approvals. Static analysis and manual review precede deployment to the Sepolia test network. The risk model is benchmarked on a labeled public transaction dataset in the style of [6] and on synthetic attack injections into replayed vault histories. Table V lists the metric metrics and the design targets against which measured values will be compared.

TABLE V: EVALUATION METRICS AND DESIGN TARGETS.

Metric	Method	Design target
Gas per transaction	Hardhat gas reporter, 2-of-3 to 5-of-7 vaults	Report per operation
Scoring latency	End-to-end proposal to attestation	Under 2 s at p95

Metric	Method	Design target
Precision, recall, F1	Held-out labeled data; injected anomalies	Exceed rule-only baseline
False-positive rate	Replay of benign vault histories Replay, stale, forged, outage,	Low enough to avoid alert fatigue
Attack resistance	and reentrancy tests	All attacks rejected

Measured values are to be inserted here once the testnet deployment is complete; this paper deliberately reports the protocol rather than unverified numbers.

IX. LIMITATIONS AND FUTURE WORK

The scheme concentrates trust in a score producer, even though the damage it can cause is bounded by the tier table. A threshold quorum of independent oracles would reduce this further. An Isolation Forest works on tabular behavior features and does not exploit transaction-graph structure; graph convolutional networks, as used in [6], are a natural next step. Gas cost grows with signer count, which layer-2 deployment can mitigate, and sharded designs such as [14] point to higher throughput. Formal verification of the approval invariants and AI-assisted contract auditing remain open directions, as does cross-chain interoperability.

X. CONCLUSION

We implemented AI Smart Vault, which makes a risk verdict binding on the transaction it describes. A signed, nonce-bound attestation carries an off-chain anomaly score to a contract that raises approvals or imposes a time lock according to an on-chain tier table, with a fail-safe default when the score is missing. The design closes the integration, adaptivity, and enforcement gaps identified in the survey [1], and the evaluation protocol provides a path to quantifying its cost and benefit.

REFERENCES

1. Dr. Ramya B. N., Prajwal R., Pranav V. Bajikar, Prasad R., and Rajeshwar S., “A comprehensive survey of blockchain-driven digital asset exchange platform,” *Int. J. Res. Publication Anal.*, vol. 2, no. 5, pp. 1–12, May 2026.
2. Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, “An overview of blockchain technology: Architecture, consensus, and future trends,” in *Proc. IEEE 6th Int. Congr. Big Data*, 2017.
3. V. Zakhary, M. J. Amiri, S. Maiyya, D. Agrawal, and A. El Abbadi, “Towards global

- asset management in blockchain systems,” *arXiv:1905.09359*, 2019.
4. C. McCabe, A. I. C. Mohideen, and R. Singh, “A blockchain-based authentication mechanism for enhanced security,” *Sensors*, vol. 24, no. 17, p. 5830, 2024.
 5. Y. Tan, Y. Cheng, L. Ding, and Y. Zhao, “Multi-signature authentication and simulation extractability security optimization for account-based blockchain anonymous systems,” in *Proc. SPIE 13562, Int. Conf. Comput. Appl. Inf. Security (ICCAIS)*, 2024.
 6. M. Weber *et al.*, “Anti-money laundering in Bitcoin: Experimenting with graph convolutional networks for financial forensics,” in *KDD '19*
 7. M. Alharby and A. van Moorsel, “Blockchain-based smart contracts: A systematic mapping study,” *Comput. Sci. Inf. Technol. (CS & IT)*, pp. 125–140, 2017.
 8. T. M. Hewa, Y. Hu, M. Liyanage, S. S. Kanhare, and M. Ylianttila, “Survey on blockchain-based smart contracts: Technical aspects and future research,” *IEEE Access*, vol. 9, pp. 87643–87662, 2021.
 9. N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on Ethereum smart contracts (SoK),” in *Proc. Int. Conf. Principles Security Trust (POST)*, 2017, pp. 164–186.
 10. R. Bloemen, L. Logvinov, and J. Evans, “EIP-712: Typed structured data hashing and signing,” *Ethereum Improvement Proposals*, no. 712, 2017.
 11. F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest,” in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, 2008, pp. 413–422.
 12. F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
 13. OpenZeppelin, “OpenZeppelin Contracts documentation.” [Online].
 14. Available: <https://docs.openzeppelin.com/contracts>
 15. E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, “OmniLedger: A secure, scale-out, decentralized ledger via sharding,” in *Proc. IEEE Symp. Security Privacy (SP)*, 2018.