ISSN: 2581-3404 (Online) IF: 5.86 (SJIF)
IJIRTM, Volume-10, Issue-6, November-2026

# Trust X: A Blockchain-Backed, Context-Aware Platform for Identity Access Control and Digital Asset Management – Literature Review and System

**Sagar[1], Harsh Rahul[2], Nisha Sharma[3]**
Student, Department of Computer Science and Engineering (Cyber Security) [1,2]
Panipat Institute of Engineering and Technology, Samalkha, Panipat, India[1,2,3]
Kurukshetra University, Kurukshetra, India[1,2,3]
Assistant Professor[3]

**Abstract.** *Organizations now keep confidential reports, academic records and business files as digital assets, and most guard them with a centralized database, role-based permissions and ordinary application logs. This arrangement is simple, but it struggles when an access decision depends on the sensitivity of the asset, the requested operation, the time or the device, and its audit records can be altered by anyone who controls the logging system. This paper reviews the literature on access control, zero-trust security, blockchain-based authorization and identity, encrypted off-chain storage and tamper-evident auditing, and describes the design of Trust X, a blockchain-backed, context-aware digital asset management platform. Trust X is a web application built with React, Express and Prisma that is intended to run on ordinary student hardware. It decides every protected request in a deterministic policy engine that returns ALLOW, STEP-UP or DENY, encrypts each file with AES-256-GCM before storing it off-chain, checks SHA-256 digests for each asset version, chains audit events by hash, and anchors Merkle roots and version records through a Solidity smart contract, with a local-ledger fallback. Drawing on thirty-four sources covering access-control models, standards, blockchain surveys, cryptographic primitives and smart-contract security, the paper sets out the architecture, the planned evaluation, the development schedule and the safeguards of the platform, and compares Trust X with existing categories of systems. Trust X is presented as an academic prototype, and its performance and security claims remain to be demonstrated through the planned evaluation.*

**Keywords:** Blockchain, Identity Access Control, Digital Asset Management, Zero Trust, Context-Aware Authorization, RBAC, ABAC, Smart Contracts, AES-256-GCM, Data Integrity, Audit Logging.

## Introduction

Digital assets such as confidential reports, academic records and business files are now created, shared and revised almost entirely online. Protecting them takes more than checking a password. A platform must decide who may view, download, edit or share a particular file, keep the content unreadable to everyone else, notice when a file has been changed, and leave a record that an investigator can trust. Standards bodies describe these as separate but related duties: authorization models such as role-based and attribute-based access control [7][8], zero-trust principles [3], cryptographic hashing [10], authenticated encryption [11] and key management [12].

In practice these duties are handled by different components with different owners. Role-based permissions are easy to administer, but broad roles lead to excessive privilege, and a role says nothing about the time, device or network from which a request arrives [7][8]. A compromised but legitimate account can attempt operations that break organizational policy even though its login is valid. Audit logs kept in the same database as the data are only as trustworthy as the administrators and software that can write to them, so a privileged insider or an attacker with database access may be able to change history without detection [18]. A hash comparison shows that a file has changed, yet it cannot say whether the change was a legitimate new version or tampering.

Blockchain is often proposed as a remedy. A distributed ledger gives a shared, tamper-evident history, and smart contracts can automate authorization and lifecycle rules [1][2]. The technology also has limits. A ledger preserves whatever it is given, so it cannot guarantee that a registered hash was correct. It adds transaction delay and cost, replicated metadata can reveal activity patterns, and it neither encrypts files nor prevents an encrypted file from being deleted from its storage [1][5][6]. Large files are unsuitable for on-chain storage, which is why a hybrid of off-chain storage and on-chain references is the usual choice [23].

Two developments make a combined design realistic for a small team. Standardized primitives, namely AES-GCM, SHA-256, PBKDF2 and JSON Web Tokens, are available in mature libraries [10][11][13][14], and smart-contract tooling such as Solidity, Hardhat and ethers.js lets a contract be written, tested and called from an ordinary web backend [30][33][34]. Trust X combines these ideas. It keeps file contents encrypted and off-chain, evaluates every protected request in a deterministic policy engine instead of on-chain, and uses the blockchain only for selected references and integrity checkpoints, where independent verification is worth the cost.

The objectives of this paper are: (1) to describe the weaknesses of centralized, role-only digital asset management in terms of authorization, integrity and audit; (2) to review access-control models, zero trust, blockchain-based access control and identity, and blockchain–cloud integration, and identify what they leave unsolved; (3) to present a layered design in which an application-level policy engine decides access while encryption, hashing and a ledger protect content and records; (4) to explain how the three-outcome decision, deny-by-default enforcement, expiry and revocation put least privilege into practice; (5) to show how hash-chained audit records and Merkle-root anchoring make changes to history detectable; (6) to compare Trust X with existing categories of systems and state a plan for measuring it; and (7) to outline the development schedule, risks and future directions.

The rest of the paper is organized as follows. The Literature Review discusses prior work across six areas. The Proposed System section describes the architecture, the access-decision workflow, the cryptographic and audit mechanisms and the development plan. The Findings section compares Trust X with existing paradigms and summarizes the reviewed studies, their gaps and the research opportunities that follow. The Conclusion closes with limitations and future work.

## Literature Review

The sources were grouped into six areas: access-control models and zero trust, identity management, blockchain foundations and blockchain-based access control, blockchain–cloud integration and storage, integrity, encryption and audit logging, and smart-contract security with supporting software. This is a focused narrative review of selected publications and standards. It is not a systematic review: no database search strings, screening counts or inclusion criteria are reported, and relevant work may be missing.

Access-control models and zero trust. Role-based access control assigns permissions to roles and lets users acquire them through membership, which simplifies administration [7]. Its weakness is that a broad role may grant more than a task needs, which conflicts with the principle of least privilege set out by Saltzer and Schroeder [9]. Attribute-based access control evaluates attributes of the user, the resource, the requested action and the environment, which allows fine-grained and contextual decisions, but NIST notes that it depends on well-defined and reliable attributes [8]. NIST SP 800-207 describes zero trust as protecting resources rather than trusting a user because of network location, so each request is judged against identity, resource and policy [3]. None of these documents prescribes a complete asset-management workflow; they supply the vocabulary that Trust X applies. Trust X uses roles for coarse privileges and ABAC-style conditions for context, and treats the two as complementary.

Identity management. Digital identity covers identification, authentication, authorization and lifecycle management, and NIST SP 800-63B sets requirements for authentication and session management [26]. Liu et al. review blockchain-based identity systems and show that decentralization does not remove trusted parties: issuers, recovery services and governance mechanisms remain, and recovery, privacy and revocation are open questions [4]. Mühle et al. identify the components that a self-sovereign identity needs [27]. A blockchain address identifies the signer of a transaction, not a person, so a platform must still bind an authenticated application account to the address that may act for it. Trust X therefore authenticates users in the application, using PBKDF2 password hashing [13] and JSON Web Tokens [14] with refresh-token rotation, and uses the ledger only as a record.

Blockchain foundations and blockchain-based access control. Yaga et al. explain hashing, distributed ledgers, consensus and smart contracts for security applications [1], and Nakamoto and Wood introduced the Bitcoin and Ethereum ledgers on which these ideas rest [15][16]. Hu's NISTIR 8403 examines blockchain-based access-control systems, their components and policy models [2]. Read together with the other work, it shows that a ledger can make policy changes verifiable, but that enforcement still requires every request to be evaluated against current policy and storage to be reachable only through that evaluation, and that a transaction for every read adds latency. Early systems illustrate the pattern. Zyskind et al. use a blockchain as an access-control manager while the data stays off-chain [21]; MedRec uses Ethereum smart contracts to manage permissions to medical records held by providers [20]; and FairAccess applies blockchain-issued tokens to access control in the Internet of Things [22]. Each targets one domain, and none combines file encryption, version history and contextual decisions in one workflow. Punia et al. survey blockchain-based access control in cloud environments, reportedly covering 118 papers [5]. Because that review is broad, the present paper takes the narrower view of how the pieces fit together.

Blockchain–cloud integration and storage. Zou et al. survey how blockchain can provide security services in cloud computing, including protection of data integrity, and also consider the effect of cloud computing on blockchain performance [6]. Because validation and consensus add overhead, large files are not stored on-chain. The taxonomy of Xu et al. lists on-chain versus off-chain data storage as a basic architectural decision [23]. Trust X follows the hybrid choice: encrypted content stays off-chain, and only references and digests are recorded on the ledger.

Integrity, encryption and audit logging. SHA-256 is standardized in FIPS 180-4 [10], and AES-GCM is standardized in NIST SP 800-38D, whose secure use requires correct nonce handling and verification of the authentication tag [11]. NIST SP 800-57 gives key-management guidance [12], and the OWASP Cryptographic Storage Cheat Sheet offers practical advice on storing encrypted data and its keys [29].

Hashing does not conceal content and does not show who changed a file. For audit records, Schneier and Kelsey describe cryptographically protected logs in which entries are bound to earlier entries so that alteration can be detected [18], and Merkle trees let a single root hash commit to a large set of records [17]. Trust X applies both ideas: a hash-chained audit trail whose Merkle roots are anchored on the ledger.

Smart-contract security and supporting software. A smart contract is a security boundary of its own. Luu et al. show that deployed Ethereum contracts can contain exploitable flaws [25], and Atzei et al. catalogue classes of attacks against them [24]. OpenZeppelin provides reusable contract components [31]. For unusual access patterns, the Isolation Forest method isolates outliers by random partitioning [19], which is the idea behind the alerts in Trust X. OWASP's Application Security Verification Standard requires access control to be enforced on a trusted server-side layer, so hiding a button in the interface is never sufficient [28]. The implementation relies on Prisma for data access [32], ethers.js for contract calls [33] and Hardhat for compiling, deploying and testing contracts [34].

Taken together, the literature shows strong individual components but few designs that join contextual authorization, encrypted off-chain storage, per-version integrity checks and independently verifiable audit records in one workflow. That is the gap Trust X targets.

## Proposed System

Architecture overview. Users reach Trust X through a React web interface. Requests pass to an Express API that authenticates the session using a JSON Web Token and a device token. Before any protected operation runs, the API asks a zero-trust policy engine for a decision. Asset records, users and permissions live in a Prisma-managed SQLite database. File contents are held as AES-256-GCM ciphertext in local storage that is reachable only through the API after an ALLOW decision. Security events flow into a SHA-256 hash-chained audit log with anomaly alerts, and Merkle roots of that log, together with asset version records, are anchored through the TrustXRegistry smart contract on an EVM-compatible chain or, if no node is available, in a local ledger.

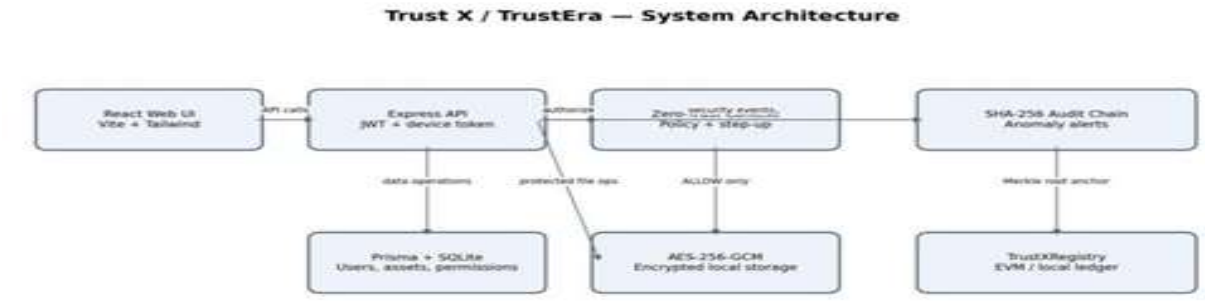



Figure 1: High-level architecture of Trust X.

Table 1: Trust X modules and their roles

| Module | Technology | Role |
|---|---|---|
| Web interface | React, Vite, Tailwind CSS | Role-specific screens, access requests, administrator console with a policy simulator |
| API and session layer | Node.js, Express, JWT, Helmet, express-rate-limit | Authenticates requests, rotates refresh tokens, applies security headers and rate limits |
| Policy engine | Deterministic rules in TypeScript | Evaluates role, permission, expiry, quota, IP range and device trust; returns ALLOW, STEP-UP or DENY |
| Data layer | Prisma ORM, SQLite (PostgreSQL as an option) | Stores users, assets, versions, permissions and access requests |
| Encrypted storage | AES-256-GCM, local file storage | Holds assets off-chain under per-asset data keys |
| Integrity verifier | SHA-256 | Compares a retrieved file with the registered digest of the correct version |
| Audit chain | SHA-256 hash chain | Records security events in a tamper-evident sequence |
| Anomaly detector | Isolation Forest-style model | Flags unusual access patterns for review |
| Blockchain registry | Solidity (TrustXRegistry), Hardhat, ethers.js | Registers asset versions and anchors audit Merkle roots; local-ledger mode as fallback |

Access-decision workflow. Every protected operation follows the same path, shown in Figure 2. The user authenticates, and if policy requires it, completes an additional factor. The user then selects an asset and requests an operation such as viewing, downloading, editing or sharing. The policy engine checks the user's role and asset-specific permissions, ownership, expiry, download quota, IP range, device trust, asset sensitivity and time restrictions, and returns one of three outcomes. ALLOW means the request satisfies policy and the operation proceeds. STEP-UP means the operation stays blocked until extra verification, such as multi-factor authentication, a device check or re-authentication, succeeds; if it fails, access is denied. DENY means the request violates policy or fails a mandatory check, and the resource is not released. Protected resources are denied by default, and the backend enforces permissions for every operation regardless of what the interface displays [28]. Whatever the outcome, a security event is recorded.

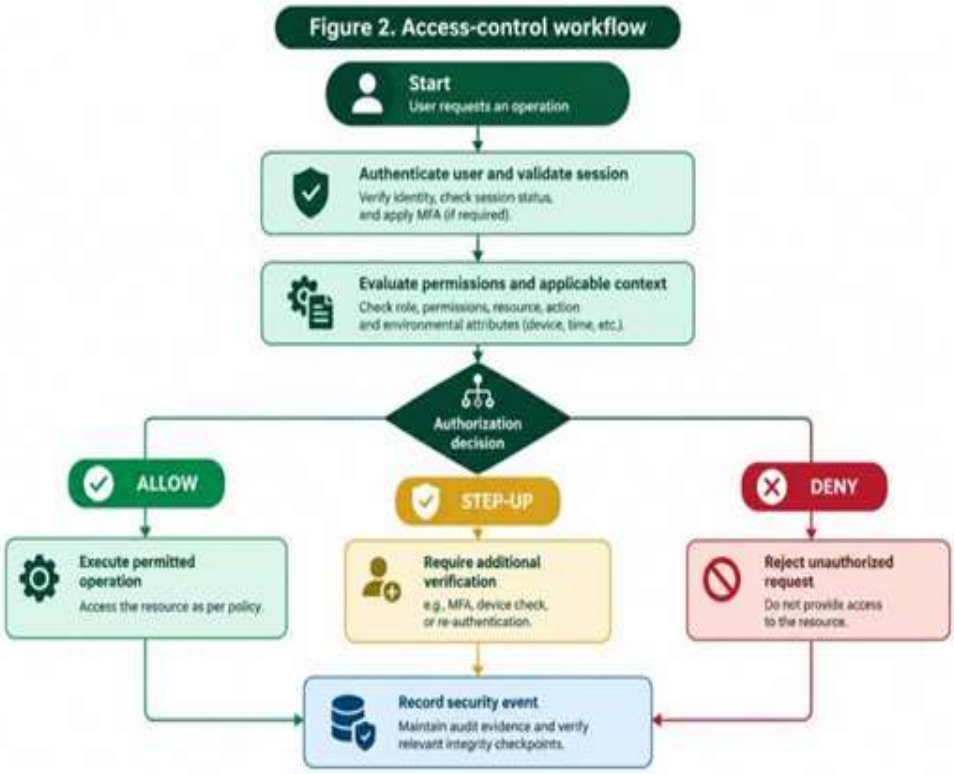



Figure 2: Access-control workflow with ALLOW, STEP-UP and DENY outcomes.

Authentication and session security. Passwords are stored using PBKDF2 with SHA-512 [13]. Sessions use signed JSON Web Tokens [14] with refresh-token rotation, so a stolen refresh token has a limited useful life, and users can review and end their devices and sessions. Helmet and express-rate-limit add security headers and slow repeated attempts. These measures follow the authentication and session guidance of NIST SP 800-63B [26].

Encrypted off-chain storage. Each asset is encrypted with AES-256-GCM before it is written to storage. A per-asset data key protects the file and is itself protected by a separate key, an envelope approach in line with key-management practice [12][29]. Authenticated encryption detects modification of the ciphertext, provided nonces are never reused and the tag is always checked [11]. Decryption takes place only inside the backend after an ALLOW decision, and the file is delivered through that protected path. Key management is the weakest point in any such design, so keys are kept apart from the stored files and are never recorded in logs.

Integrity checking and version tracking. When an asset is registered, the platform computes its SHA-256 digest [10] and stores it as the trusted reference. Before content is released, the digest of the retrieved file is compared with the reference for the requested version. A legitimate edit is not overwritten in place: it creates a new version with its own digest, linked to the authorized action that produced it, while earlier versions remain on record. A digest mismatch is therefore judged against the version history rather than assumed to be an attack. The registry stores digests and version identifiers, never file contents.

Audit chain and blockchain anchoring. Each audit record holds the asset identifier, the operation, the outcome and a timestamp, and avoids unnecessary sensitive data. Every record also includes the hash of the one before it, so altering or removing an old entry breaks the chain from that point onward [18]. At intervals the system computes the Merkle root of a batch of records [17] and anchors it, together with asset version registrations, through the TrustXRegistry contract [30][31]. An auditor can later recompute the chain and compare the root with the anchored value. This detects tampering but does not prevent it, and it does not prove that an event was accurately described when it was first recorded [1]. In local-ledger mode the anchor is held in the application database, which does not give the same independent verification, and this limit is stated openly in the project scope.

Anomaly detection and administration. An Isolation Forest-style model [19] scores access patterns, for example unusual volumes or timing, and raises alerts to administrators; it assists review and does not make access decisions, which remain deterministic and explainable. The administrator console includes a policy simulator so that the effect of a rule can be inspected before it is applied. Four user types are supported: administrators who manage users, roles and policies; asset owners who upload files, approve requests and manage versions; authorized users who access files within their grants; and auditors who review records and verify integrity evidence.

Planned evaluation. Trust X will be compared with a centralized baseline that has equivalent authentication, permissions, encryption and protected logging, so that the blockchain design is not measured against an insecure system. Functional tests will cover valid requests and the denial of revoked, expired, over-quota and unauthorized requests, and the detection of audit tampering. Security tests will cover attempts to bypass the policy engine, reuse of expired permissions, direct access to storage, invalid version transitions and smart-contract authorization flaws [25][24]. Performance will be measured as authorization latency, transaction confirmation time, revocation propagation time, integrity verification time, throughput, resource consumption and audit verification time, using several file sizes and repeated runs.

Development schedule. The project is planned for 12 weeks in eight phases.

Table 2: Development phases

| Phase | Focus | Duration | Main deliverables |
|---|---|---|---|
| 1 | Problem identification and requirement analysis | Week 1 | Problem definition |
| 2 | Literature review and technology study | Week 2 | Literature review |
| 3 | System architecture and database design | Weeks 3–4 | Architecture and ER design |
| 4 | Authentication, user roles, backend APIs and database integration | Weeks 5–6 | Core application |
| 5 | Asset management, encryption, integrity verification and policy enforcement | Weeks 7–8 | Secure asset workflows |
| 6 | Blockchain integration, audit verification and feature integration | Weeks 9–10 | Integrated application |

| Phase | Focus | Duration | Main deliverables |
|---|---|---|---|
| 7 | Functional, integration and security testing; issue resolution | Week 11 | Test results and corrections |
| 8 | Documentation, presentation preparation and final demonstration | Week 12 | Final report and presentation |

Harsh is responsible for requirement analysis, system architecture, authentication, access-control design and backend integration. Sagar is responsible for blockchain integration, integrity verification, audit mechanisms and security testing. Both authors share integration, debugging, testing, documentation, the final report and the presentation.

Scope and risks. Trust X is an academic prototype and is not certified for production use. Its security depends on correct authentication, authorization, encryption and key management. Blockchain verification needs a correctly configured node and a deployed contract, and local-ledger mode is weaker. Contextual signals such as IP range and device tokens can be wrong or spoofed, so their source, freshness and failure behavior must be defined, and a familiar network is never treated as safe. Smart contracts can contain bugs, which is why reviewed components and contract tests are planned [31][25]. Anomaly detection can produce false alerts, so it is kept advisory. Finally, a legitimate modification must be registered as a new version, or the integrity history will no longer match the stored file.

Findings

The first comparison places Trust X against three categories of existing approaches. The table reflects the design goals of Trust X, not measured results, and its claims will be tested in the evaluation phase.

Table 3: Comparison of Trust X with existing paradigms

| Criterion | Traditional file management | Role-based asset management | Blockchain-only access control | Trust X |
|---|---|---|---|---|
| Contextual authorization | Limited | Roles only | Depends on contract design | Yes, role plus context with three outcomes |
| File confidentiality | Depends on storage | Depends on storage | Not provided; data is off-chain | Yes, AES-256-GCM envelope encryption |
| Integrity verification | Rare | Rare | Hash records on ledger | Yes, SHA-256 per version |
| Version history | Manual | Partial | Via ledger records | Yes, each version has its own digest |
| Audit trail | Mutable logs | Mutable logs | On-chain events, at a cost | Hash-chained log with anchored Merkle roots |
| Independent verification | No | No | Yes | Yes with a blockchain node; weaker in local-ledger mode |

| Criterion | Traditional file management | Role-based asset management | Blockchain-only access control | Trust X |
|---|---|---|---|---|
| Per-access latency | Low | Low | Higher if each access needs a transaction | Policy engine is off-chain; chain used for checkpoints |
| Expiry and revocation | Manual | Role removal | Contract update and confirmation delay | Enforced by the policy engine at request time |
| Operational complexity | Low | Low to moderate | High | Moderate to high (keys, contract, node) |

The second table summarizes the main sources reviewed and the gap each one leaves open. Where a source does not report a numerical result, this is marked N/A.

Table 4: Summary of reviewed work and remaining gaps

| Year | Source | Focus and key point | Reported metrics | Gap addressed by Trust X |
|---|---|---|---|---|
| 1996 | Sandhu et al. [7] | Formal role-based access control models | N/A | Roles alone do not express context or asset sensitivity |
| 1999 | Schneier and Kelsey [18] | Cryptographically protected audit logs | N/A | Needs an independent anchor for the log |
| 2014 | NIST SP 800-162 [8] | Definition and considerations for ABAC | N/A (guideline) | Needs reliable attributes and a concrete workflow |
| 2015 | Zyskind et al. [21] | Blockchain as access manager for personal data | N/A | No file versioning or contextual decisions |
| 2016 | MedRec [20] | Smart contracts for medical-record permissions | N/A | Domain-specific; no per-version file integrity |
| 2016 | FairAccess [22] | Blockchain tokens for IoT access control | N/A | Domain-specific; no file encryption or version lifecycle |
| 2017 | Xu et al. [23] | Taxonomy of blockchain system design | N/A | Design choices left to the implementer |
| 2018 | NISTIR 8202 [1] | Blockchain components and security properties | N/A | General; not an asset-management design |

| Year | Source | Focus and key point | Reported metrics | Gap addressed by Trust X |
|---|---|---|---|---|
| 2020 | Liu et al. [4] | Review of blockchain identity management | N/A | Recovery, issuer trust and interoperability unresolved |
| 2020 | NIST SP 800-207 [3] | Zero-trust architecture principles | N/A | Not blockchain-specific |
| 2021 | Zou et al. [6] | Survey of blockchain–cloud integration | N/A | Integration-level; performance trade-offs remain |
| 2022 | NISTIR 8403 [2] | Blockchain-based access control systems | N/A | Guidance, not an evaluated asset platform |
| 2024 | Punia et al. [5] | Systematic review of blockchain access control in the cloud | Reported 118 papers | Broad scope; little on combined asset lifecycle needs |

The review also points to seven research gaps that arise when the components are combined, listed in Table 5 together with the way Trust X responds.

Table 5: Research gaps and the response of Trust X

| No. | Gap | Trust X response (design objective) |
|---|---|---|
| G1 | Identity binding: an application account, a real-world identity and a blockchain address are distinct | Authenticate in the application and treat the ledger as a record, protecting the mapping from unauthorized change |
| G2 | Context-aware enforcement is hard to apply consistently | Hybrid roles and context in one deterministic engine with ALLOW, STEP-UP and DENY |
| G3 | A recorded revocation may not stop a cached decision or an issued link | Check expiry and revocation on every request; measure propagation time |
| G4 | A hash mismatch cannot separate a legitimate update from tampering | Version-aware registry with one digest per authorized version |
| G5 | Identifiers and transaction patterns can leak information | Keep contents off-chain and record only digests, versions and Merkle roots |
| G6 | A transaction per request raises latency and cost | Decide routine requests off-chain; anchor checkpoints in batches |

| No. | Gap | Trust X response (design objective) |
|---|---|---|
| G7 | Contracts are only part of the attack surface | Test the whole system against a threat model, not the contract alone |

Three observations follow from the review. First, every component Trust X needs already exists and works, so the contribution lies in how they are combined and constrained, not in a new primitive. Second, the blockchain access-control literature concentrates on recording permissions, while the operational questions of revocation consistency, version history and key management receive less attention. Third, a blockchain improves verifiability only when independent checking is actually needed. If one trusted organization controls all assets and needs no outside verification, a well-built centralized system may be the better choice, and Trust X is most defensible where the anchoring solves a real auditability problem.

## Conclusion

This paper has presented Trust X as a practical way to combine access control, encryption, integrity checking and independently verifiable audit records in one digital asset management platform. By deciding access in a deterministic policy engine, keeping content encrypted and off-chain, and using the blockchain only for digests, version records and Merkle-root checkpoints, the design aims to deliver least-privilege access and detectable tampering without paying the cost of putting every operation on a ledger. Hash-chained logging, per-version digests, expiry and revocation, and an explainable policy simulator together aim to make the platform both safe to use and easy to audit.

The work has limits. This paper is a design and a focused literature review, and the source selection was not systematic. Claims about performance, revocation delay, anomaly-detection quality and resistance to attack are design objectives that must still be demonstrated through the planned functional, security and performance tests against a centralized baseline. The blockchain records the correctness of nothing it is not given, local-ledger mode offers weaker verification, and the platform remains an academic prototype that depends on sound key management.

Future work includes integration with OpenID Connector SAML for enterprise identity, secure cloud or object-storage back ends for larger files and more users, stronger key management with rotation and hardware-backed protection, formal verification of the smart contract, privacy-preserving identity and audit disclosure, richer anomaly detection, and a systematic literature review with a documented search strategy. Evaluation should also report measured results for authorization latency, transaction confirmation time and revocation propagation, so that the benefit of the blockchain component can be judged against its overhead.

## References

[1] D. Yaga, P. Mell, N. Roby and K. Scarfone, Blockchain Technology Overview, NISTIR 8202, National Institute of Standards and Technology, 2018.

[2] V. C. Hu, Blockchain for Access Control Systems, NISTIR 8403, National Institute of Standards and Technology, 2022.

[3] S. Rose, O. Borchert, S. Mitchell and S. Connelly, Zero Trust Architecture, NIST SP 800-207, 2020.

[4] Y. Liu, D. He, M. S. Obaidat, N. Kumar, M. K. Khan and K.-K. R. Choo, Blockchain-based identity management systems: A review, Journal of Network and Computer Applications, vol. 166, Art. 102731, 2020.
[5] A. Punia, P. Gulia, N. S. Gill, E. Ibeke, C. Iwendi and P. K. Shukla, A systematic review on blockchain-based access control systems in cloud environment, Journal of Cloud Computing, vol. 13, Art. 146, 2024.
[6] J. Zou, D. He, S. Zeadally, N. Kumar, H. Wang and K.-K. R. Choo, Integrated blockchain and cloud computing systems: A systematic survey, solutions, and challenges, ACM Computing Surveys, vol. 54, no. 8, Art. 160, 2021.
[7] R. S. Sandhu, E. J. Coyne, H. L. Feinstein and C. E. Youman, Role-based access control models, Computer, vol. 29, no. 2, pp. 38–47, 1996.
[8] V. C. Hu et al., Guide to Attribute Based Access Control (ABAC) Definition and Considerations, NIST SP 800-162, 2014.
[9] J. H. Saltzer and M. D. Schroeder, The protection of information in computer systems, Proceedings of the IEEE, vol. 63, no. 9, pp. 1278–1308, 1975.
[10] National Institute of Standards and Technology, Secure Hash Standard (SHS), FIPS PUB 180-4, 2015.
[11] M. Dworkin, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST SP 800-38D, 2007.
[12] E. Barker, Recommendation for Key Management: Part 1 – General, NIST SP 800-57 Part 1 Rev. 5, 2020.
[13] K. Moriarty, B. Kaliski and A. Rusch, PKCS #5: Password-Based Cryptography Specification Version 2.1, RFC 8018, IETF, 2017.
[14] M. Jones, J. Bradley and N. Sakimura, JSON Web Token (JWT), RFC 7519, IETF, 2015.
[15] S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.
[16] G. Wood, Ethereum: A Secure Decentralised Generalised Transaction Ledger, Ethereum Yellow Paper, 2014.
[17] R. C. Merkle, A digital signature based on a conventional encryption function, in Advances in Cryptology – CRYPTO ’87, Springer, 1988.
[18] B. Schneier and J. Kelsey, Secure audit logs to support computer forensics, ACM Transactions on Information and System Security, vol. 2, no. 2, pp. 159–176, 1999.
[19] F. T. Liu, K. M. Ting and Z.-H. Zhou, Isolation forest, in Proc. IEEE International Conference on Data Mining (ICDM), 2008.
[20] A. Azaria, A. Ekblaw, T. Vieira and A. Lippman, MedRec: Using blockchain for medical data access and permission management, in Proc. International Conference on Open and Big Data (OBD), 2016.
[21] G. Zyskind, O. Nathan and A. Pentland, Decentralizing privacy: Using blockchain to protect personal data, in Proc. IEEE Security and Privacy Workshops, 2015.
[22] A. Ouaddah, A. Abou Elkalam and A. Ait Ouahman, FairAccess: A new blockchain-based access control framework for the Internet of Things, Security and Communication Networks, vol. 9, no. 18, 2016.
[23] X. Xu et al., A taxonomy of blockchain-based systems for architecture design, in Proc. IEEE International Conference on Software Architecture (ICSA), 2017.

[24] N. Atzei, M. Bartoletti and T. Cimoli, A survey of attacks on Ethereum smart contracts (SoK), in Principles of Security and Trust (POST), 2017.
[25] L. Luu, D.-H. Chu, H. Olickel, P. Saxena and A. Hobor, Making smart contracts smarter, in Proc. ACM CCS, 2016.
[26] P. A. Grassi et al., Digital Identity Guidelines: Authentication and Lifecycle Management, NIST SP 800-63B, 2017.
[27] A. Mühle, A. Grüner, T. Gayvoronskaya and C. Meinel, A survey on essential components of a self-sovereign identity, Computer Science Review, vol. 30, pp. 80–86, 2018.
[28] OWASP Foundation, Application Security Verification Standard (ASVS).
[29] OWASP Foundation, Cryptographic Storage Cheat Sheet, OWASP Cheat Sheet Series.
[30] Solidity Team, Solidity Documentation.
[31] OpenZeppelin, OpenZeppelin Contracts Documentation.
[32] Prisma, Prisma ORM Documentation.
[33] R. Moore, ethers.js Documentation.
[34] Nomic Foundation, Hardhat Documentation.