FMDB *Transactions on Sustainable Computing Systems*

TrustFlow: A Chain-of-Custody Based Traceability Framework for Secure Document Collaboration

S. Rogit[1,*], K. N. Nawaz Sheriff[2], B. Mahesh Bala[3], V. Nivedita[4]

[1,2,3,4]Department of Computer Science and Engineering, SRM Institute of Science and Technology, Ramapuram, Chennai, Tamil Nadu, India.
rs1097@srmist.edu.in[1], nk2496@srmist.edu.in[2], mb3025@srmist.edu.in[3], niveditv1@srmist.edu.in[4]

Abstract: When it comes to secure document collaboration in organisational settings, access control is not enough; it also requires end-to-end traceability of every activity that is performed on a document during its entire lifecycle. Researchers introduce TrustFlow, a full-stack framework that integrates cryptographic chain-of-custody (CoC) tracing, steganographic watermarking, Shamir threshold secret sharing, time-lock encryption, and tamper-evident audit logging into a unified document collaboration platform. TrustFlow is a full-stack framework. TrustFlow constructs a hash-linked provenance tree for each document. Within this tree, each node corresponds to an Ed25519-signed state transition, which may include uploading, sharing, revoking, or deleting. Forensic leak identification is made possible by the utilisation of three complementary steganographic watermarking modes: zero-width Unicode, linguistic synonym replacement, and whitespace encoding. All recipients are guaranteed that their documents will be destroyed in a verifiable manner using a deletion-cascade engine equipped with signed tokens and replay prevention. PostgreSQL serves as the system's database, with a Fast-API backend and a React frontend. Additionally, researchers provide an outline of a staged roadmap to complete decentralisation through the use of abstract backend interfaces, blockchain anchoring, and threshold-gated access controls.

Keywords: Chain-of-Custody; Document Traceability; Steganographic Watermarking; Shamir Secret Sharing; Time-Lock Encryption; Tamper-Evident Logging; Access Control Lists.

Received on: 28/03/2025, **Revised on:** 25/05/2025, **Accepted on:** 02/09/2025, **Published on:** 12/06/2026

Journal Homepage: https://www.fmdbpub.com/user/journals/details/FTSCS

DOI: https://doi.org/10.69888/FTSCS.2026.000684

Cite as: S. Rogit, K. N. N. Sheriff, B. M. Bala, and V. Nivedita, "TrustFlow: A Chain-of-Custody Based Traceability Framework for Secure Document Collaboration," *FMDB Transactions on Sustainable Computing Systems,* vol. 4, no. 2, pp. 78-96, 2026.

1. Introduction

Organisations routinely handle sensitive documents (contracts, intellectual property, compliance records, classified reports) that require strict accountability over who accessed what, when, and what they did with it. The chain-of-custody (CoC) concept, long established in the handling of physical evidence in legal proceedings [1], provides a rigorous model: every transfer or transformation of an artefact must be recorded, signed, and independently verifiable. Digital document systems, however, typically rely on access control lists (ACLs) and server-side audit logs that are trivially forgeable by a privileged administrator [2]. A compromised server can silently read, modify, or leak documents while producing a clean audit trail. Existing solutions

*Corresponding author.

address fragments of this problem: blockchain for immutable provenance [3], digital watermarking for leak detection [4], threshold secret sharing for distributed trust [5], but rarely integrate these mechanisms into a single, deployable platform.

1.1. Problem Statement

Current document collaboration platforms lack end-to-end cryptographic traceability, specifically:

- Provenance records are stored in server-controlled databases that a privileged administrator can silently alter or delete.
- No mechanism exists to attribute leaked documents to the responsible recipient after the fact.
- Document deletion cannot be cryptographically verified across distributed holders.
- Temporal access boundaries and threshold-gated secret recovery are typically absent from document collaboration workflows.
- These gaps mean that organisations cannot establish a tamper-evident, independently verifiable chain of custody for their sensitive documents.

1.2. Research Objectives

This work addresses the above gaps through the following objectives:

- Design and implement a cryptographic chain-of-custody engine that records every document lifecycle event as an Ed25519-signed, hash-linked provenance node.
- Develop a multi-modal steganographic watermarking system that survives partial document modification, enabling forensic leak attribution.
- Integrate Shamir threshold secret sharing with hybrid AES encryption into a group collaboration workflow with proposal, approval, and time-locked reveal semantics.
- Construct a deletion cascade protocol with signed tokens, cryptographic receipts, and replay prevention for verifiable document destruction.
- Provide a tamper-evident audit log with SHA-256 hash chaining for independently verifiable event histories.
- Demonstrate the feasibility of the integrated system through a full-stack prototype with microbenchmark evaluation.

2. Literature Survey

This section reviews the literature that informs TrustFlow's design across six research threads: digital provenance, blockchain-based document systems, text watermarking, cryptographic primitives, secure deletion and access control, and peer-to-peer infrastructure. Ramachandran and Kantarcioglu [1] present a framework for automated provenance capture, validation, and audit using Ethereum smart contracts. The Open Provenance Model (OPM), comprising agents, processes, artefacts, and causal edges, is mapped onto on-chain records managed by Solidity contract functions that enforce insertion rules, validate relationship consistency, and implement access-control checks. The resulting provenance graph is inherently tamper-evident. Gas consumption analysis per operation type (record insertion, edge creation, validation query) quantifies the cost of representative provenance workloads. Ethereum's gas model renders frequent writes economically prohibitive, and on-chain storage constraints require off-chain data with hash references. The prototype does not address the privacy of on-chain provenance records. TrustFlow's provenance model parallels the OPM-to-blockchain mapping; the gas-cost analysis motivates TrustFlow's evaluation of fee-less alternatives (e.g., Nano block-lattice) for on-chain anchoring. Paci et al. [2] survey access-control models for community-centred collaborative systems, establishing a taxonomy that covers RBAC, ABAC, and relationship-based access control (ReBAC/social graph policies). Each paradigm is evaluated against dynamic group membership, fine-grained sharing, delegation chains, provenance-aware policies, and privacy preservation. Design goals, including policy expressiveness, conflict resolution, and administrability, are distilled, with open problems identified in policy composition across overlapping communities.

The literature cutoff predates blockchain-based access control and decentralised identity (DID) developments; few proposals include production-scale evaluations. TrustFlow implements group-based access control with owner, editor, and viewer roles combined with share-level permissions. The RBAC and delegation patterns map directly to TrustFlow's permission model, while the ReBAC discussion informs potential social-graph-aware sharing policies. Liang et al. [3] propose ProvChain, a blockchain-anchored provenance architecture for cloud environments. Provenance collectors embedded in cloud infrastructure capture data operation events; records are hashed and the resulting digests committed to a blockchain as immutable anchors, avoiding on-chain storage bloat while preserving verifiability. Privacy-enhancing mechanisms, including selective disclosure and encryption of sensitive provenance content, protect operational details while maintaining audit transparency. Evaluation on a cloud testbed demonstrates low per-transaction overhead for hash-and-timestamp writes. However, blockchain

confirmation latency and throughput constraints necessitate batching for high-frequency logging, and off-chain record integrity depends on correct encryption outside blockchain guarantees. TrustFlow follows a structurally similar pattern (local hash-chained provenance with a planned blockchain anchoring phase), and ProvChain's off-chain-record/on-chain-digest design directly informs TrustFlow's Phase 4 ledger integration strategy. Kamaruddin et al. [4] present a systematic review of text watermarking techniques, synthesising approximately 94 prior works into a four-category taxonomy: (i) format-based methods (whitespace manipulation, font alteration), (ii) structural methods (layout modification, line/word reordering), (iii) linguistic methods (synonym substitution, paraphrase-based embedding), and (iv) zero-width/invisible character methods (zero-width Unicode insertion, homoglyph substitution).

Each category is evaluated against robustness, capacity, imperceptibility, security, and language dependence. The canonical tradeoff triangle (robustness, imperceptibility, and capacity) is confirmed. Format-based methods offer high capacity but are fragile to reformatting; linguistic methods achieve superior imperceptibility but support low payloads; zero-width schemes are highly imperceptible but brittle to Unicode normalisation. The absence of standardised benchmarks limits cross-method comparison. TrustFlow's steganographic module implements three of the four surveyed categories (zero-width Unicode, linguistic synonym substitution, whitespace encoding), directly instantiating this taxonomy. Chandramouli et al. [5] survey verifiable secret-sharing (VSS) schemes achieving perfect (information-theoretic) security against computationally unbounded adversaries. Standard secret sharing differs from VSS, which augments the protocol with mechanisms to detect inconsistent or malicious shares from a cheating dealer. Constructions are organised across synchronous, asynchronous, and hybrid network models with systematic comparison of round and communication complexity bounds. The survey explicates how VSS serves as a building block for Byzantine agreement and secure multi-party computation. Coverage is restricted to perfectly secure constructions; computational VSS variants and practical engineering considerations receive limited treatment. TrustFlow implements Shamir (t, n)-threshold secret sharing for group key recovery and proposal workflows. At the same time, the current scheme omits share verification. The VSS constructions surveyed here provide potential hardening against malicious share submission. Herschel et al. [6] survey digital provenance research across databases, scientific workflows, and operating-system capture, organising the field around three questions: purpose ("what for"), representation ("what form"), and derivation source ("what from").

Their taxonomy distinguishes between why-provenance (which input tuples contributed to an output), how-provenance (the computational derivation path), and where-provenance (specific source locations). A lifecycle model spanning capture, storage, querying, and summarisation enables systematic comparison of provenance systems. Standards such as the W3C PROV family and provenance query languages are covered. The survey predates blockchain-based tamper-evidence and offers limited treatment of cryptographic integrity guarantees. TrustFlow's Chain-of-Custody graph implements how-provenance as hash-chained, Ed25519-signed nodes recording creation, sharing, modification, and deletion events, directly instantiating the lifecycle model described here. Nizamuddin et al. [7] present a decentralised document versioning system that couples Ethereum smart contracts with IPFS for immutable version metadata and content-addressed off-chain storage. A Solidity contract implements a version registry recording content identifiers (CIDs), author identities, timestamps, and parent-version linkages. IPFS content addressing guarantees that any modification invalidates the on-chain CID, providing automatic tamper detection. The prototype is evaluated on a local Ethereum testnet, measuring gas costs, version-retrieval latency, and version-chain integrity. Transaction fees and 15–30 s block confirmation latency impose cost and responsiveness constraints unsuitable for high-frequency collaborative editing; IPFS content availability depends on active pinning. TrustFlow's document storage mirrors this pattern (documents stored locally with metadata tracked separately), and the documented Ethereum+IPFS limitations motivate TrustFlow's evaluation of alternative storage and ledger substrates.

Atzei et al. [8] systematise security vulnerabilities in Ethereum smart contracts, producing the first structured taxonomy of contract-level attack vectors at three abstraction layers: Solidity-level (reentrancy, integer overflow, tx. origin misuse), EVM-level (gasless send, call-to-the-unknown, stack-size limits), and blockchain-level (timestamp dependence, transaction-ordering dependence). Twelve vulnerability classes are documented with minimal reproducing code and linked to real-world incidents, most notably the DAO reentrancy attack (∼3.6M ETH in losses). The survey is descriptive rather than prescriptive, serving as a foundation for subsequent tooling research (Mythril, Slither, Securify). For TrustFlow, this SoK informs the threat model for any smart contract integration in the blockchain anchoring roadmap, particularly reentrancy and transaction-ordering risks relevant to provenance-recording contracts. Rizzo et al. [9] propose a text watermarking scheme that exploits Unicode homoglyphs (visually identical characters from distinct Unicode blocks) and zero-width character insertion to embed invisible fingerprints into plain text. Two complementary channels are introduced: homoglyph substitution (replacing Latin characters with visually equivalent Cyrillic or Greek alternatives) and zero-width character insertion (zero-width space, joiner, non-joiner) at word boundaries. Imperceptibility is validated through rendering tests across common display environments; capacity analysis demonstrates bitrates sufficient for short identifiers. The scheme is inherently fragile to Unicode normalisation (NFC/NFD/NFKC/NFKD), which canonicalises homoglyphs and strips zero-width characters. Copy-paste through normalising intermediaries (PDF renderers, word processors) destroys the watermark, and automated detection via codepoint inspection limits security against informed adversaries.

TrustFlow's zero-width watermarking mode directly implements this approach, and the identified normalisation fragility motivates TrustFlow's multi-modal fallback strategy with linguistic and whitespace channels as secondary extraction paths. Kirchenbauer et al. [10] propose a statistical watermarking scheme for large language model (LLM) output operating at decoding time without model retraining. At each generation step, the vocabulary is partitioned into "green" and "red" lists via a pseudorandom hash seeded by preceding tokens; a soft bias toward green tokens produces text with a statistically elevated green-token proportion detectable by a z-test. Evaluation on OPT-family models demonstrates high detection accuracy on passages of several hundred tokens with negligible perplexity degradation. The scheme is vulnerable to paraphrase attacks and requires control over the decoding process, precluding retroactive watermarking. While TrustFlow targets document content rather than LLM output, the statistical detection framework (embedding imperceptible distributional signals and detecting via hypothesis testing) provides a methodological parallel to TrustFlow's multi-modal fingerprint extraction with confidence scoring. Josefsson and Liusvaara [11] specify the Edwards-Curve Digital Signature Algorithm (EdDSA) in RFC 8032, with two instantiations: Ed25519 (128-bit security over edwards25519) and Ed448 (224-bit security over edwards448). The defining innovation is deterministic nonce generation, in which the per-signature scalar is derived from a hash of the private-key prefix and the message, eliminating the catastrophic nonce-reuse vulnerability that has historically afflicted ECDSA. Twisted Edwards curves with complete addition formulas simplify implementation by removing exceptional-case branches, reducing the side-channel attack surface. Signatures are compact (64 bytes) and verification is efficient.

The specification is informational rather than standards-track; constant-time implementation remains an engineering obligation not enforced by the RFC. TrustFlow uses Ed25519 via PyNaCl as its sole digital signature primitive to authenticate Chain-of-Custody nodes, deletion tokens, and audit-log entries. The deterministic nonce property is particularly valuable in server environments where RNG quality may vary. Kavun et al. [12] survey authenticated encryption with associated data (AEAD) schemes from a hardware (ASIC) implementation perspective. A comparative framework evaluates schemes across gate count, throughput, latency, and power consumption, organising constructions into block-cypher-based modes (AES-GCM, OCB), dedicated stream/feedback designs, and sponge/permutation-based schemes. CAESAR competition candidates are analysed, with resource-sharing and side-channel countermeasure strategies evaluated for their impact on area-throughput. The hardware focus excludes software-only profiling and predates NIST lightweight AEAD standardisation. While XChaCha20-Poly1305 was evaluated for its resistance to nonce misuse via extended nonces, TrustFlow selected AES-256-GCM as its authenticated encryption primitive, leveraging hardware acceleration (AES-NI) and NIST standardisation. The survey's comparative methodology informed this design decision and remains relevant for potential future hardware-accelerated deployments in embedded chain-of-custody devices. Barker [12] establishes a comprehensive key-management framework in NIST SP 800-57 Part 1 Rev. 5 that covers the full key lifecycle: generation, distribution, storage, use, rotation, archival, recovery, and destruction. The central contribution is a set of security-strength equivalence tables mapping symmetric key lengths to comparable asymmetric parameters (128-bit symmetric ↔ 256-bit elliptic-curve keys). Cryptoperiods (maximum key-use durations) are formalised by key type and usage context.

The publication mandates the use of HSMs for high-value keys and separation of duties. Oriented toward U.S. federal compliance (FIPS, FedRAMP), the recommendations may not directly translate to non-governmental contexts; implementation-level decisions (such as serialisation formats and cloud KMS integration) fall outside the scope. TrustFlow's key management practices (Ed25519 keypair lifecycle, symmetric key generation, Shamir share distribution) are informed by SP 800-57 guidance on cryptoperiods, key separation, and algorithm selection at the 128-bit security level. Biryukov et al. [14] present Argon2, the winner of the 2015 Password Hashing Competition, designed to impose substantial memory costs on brute-force attackers and neutralise GPU/ASIC-based cracking. Three variants are specified: Argon2d (data-dependent memory access, maximum TMTO resistance, side-channel risk), Argon2i (data-independent addressing, timing-attack resistance), and Argon2id (hybrid, recommended default). Parameters for memory (m), time/passes (t), and parallelism (p) are independently tunable. Internally, Blake2b-based compression over 1024-byte blocks with multiplication-heavy operations increases ASIC circuit depth. Subsequent cryptanalysis by Alwen and Blocki demonstrated low-storage tradeoff attacks against Argon2i, prompting specification revisions (v1.3). Parameter selection remains non-trivial. TrustFlow derives user encryption keys from passwords using Argon2id, in line with OWASP and RFC 9106 recommendations, balancing server-side latency with resistance to offline credential attacks. Boneh et al. [15] formalise verifiable delay functions (VDFs): functions that require a prescribed number of sequential steps to evaluate yet produce efficiently and publicly verifiable outputs. Three security properties are defined: sequentiality (bounded-parallelism adversaries cannot accelerate evaluation), uniqueness (one valid output per input), and efficient verifiability (polylogarithmic verification time).

Candidate constructions use repeated squaring in RSA groups with succinct proof systems (Wesolowski, Pietrzak). Applications include randomness beacons, blockchain leader election, and proof-of-elapsed-time protocols. RSA-group constructions require a trusted setup; no quantum-resistant VDF is known. Trust-Flow's time-lock encryption uses server-managed TTL with AES-256-GCM rather than computational puzzles. Still, the VDF framework informs the theoretical basis for trustless time-lock guarantees in Trust-Flow's planned decentralised deployment. Garg et al. [16] provide the first cryptographic formalisation of data deletion motivated by GDPR's "right to be forgotten." Deletion-compliance is defined via

a simulation-based framework: a system is compliant if its post-deletion state is computationally indistinguishable from an execution in which the data was never submitted. Impossibility results demonstrate that, under standard assumptions (e.g., the existence of one-way functions), universal deletion-compliance cannot be achieved for arbitrary computations. Positive constructions using history-independent data structures, re-randomisation, and differential privacy characterise achievable deletion boundaries. The framework is purely theoretical, assumes honest-but-curious data collectors, and does not address proving deletion to external auditors. TrustFlow's deletion cascade implements Ed25519-signed tokens with receipt-based confirmation. Garg et al.'s impossibility results contextualise fundamental limitations for derived data (cached copies, shared fragments) in distributed deployments. Kumar et al. [17] survey attribute-based encryption (ABE) for cloud storage, comparatively analysing ciphertext-policy ABE (CP-ABE), key-policy ABE (KP-ABE), and multi-authority variants. The gap analysis identifies critical open problems: efficient revocation without re-encryption overhead, multi-authority coordination, policy hiding for attribute privacy, and outsourced decryption with verifiable correctness.

Table 1: Feature comparison with related systems. CP-ABE refers to ciphertext-policy attribute-based encryption [17]

Feature	Prov-Chain	DocChain	CP-ABE	Trust Flow
Hash-linked provenance	✓	✓	–	✓
Blockchain anchoring	✓	✓	–	Planned
Steganographic watermark	–	–	–	✓
Shamir threshold	–	–	–	✓
Time-lock encryption	–	–	–	✓
Deletion cascade	–	–	–	✓
Tamper-evident audit	✓	–	–	✓
Access control	ACL	Contract	ABE	RBAC

Evaluation dimensions include computational cost, ciphertext/key sizes, collusion resistance, and policy expressiveness. The focus on cloud scenarios limits applicability to non-cloud contexts; subsequent advances in lattice-based ABE are not covered. Trust-Flow's current access control is role-based and server-enforced; ABE represents a candidate mechanism for Trust-Flow's Phase 5 roadmap, enabling cryptographic enforcement of attribute-based policies in decentralised deployments where a trusted server may be unavailable. Daniel and Tschorsch [18] survey peer-to-peer data networks using IPFS as a reference architecture, decomposing systems into six building blocks: naming/content addressing, routing (Kademlia DHT variants), storage/replication, incentivisation (token economies, proof-of-storage), privacy/access control, and data permanence. Comparative analysis spans IPFS, Ethereum Swarm, Dat/Hypercore, SAFE Network, Storj, and Arweave, revealing divergent design philosophies (content-addressed Merkle DAGs versus append-only logs versus permanent storage with economic incentives. The qualitative scope excludes quantitative benchmarks and formal security analysis. TrustFlow's architecture roadmap evaluates decentralised storage substrates for document distribution; this survey's framework (particularly IPFS content addressing, Swarm incentivisation, and availability guarantees) directly informs TrustFlow's substrate selection criteria.

Coretti et al. [19] construct a formally verified gossip protocol that provides provable message dissemination against Byzantine adversaries. The protocol is formalised as a modular networking functionality within the Universal Composability (UC) framework, enabling compositional security proofs for higher-level protocols. Application-layer information (verifiable random functions and stake weights) combined with bilateral peer primitives yields a Byzantine-resilient dissemination layer. Novel graph-theoretic results establish that sparse random topologies exhibit sufficient expansion for timely propagation despite adversarial interference. The analysis relies on random graph models and honest-stake-majority assumptions; engineering translation to production systems is ongoing. TrustFlow's Phase 3 roadmap specifies a gossip-based P2P layer for document propagation and the dissemination of deletion tokens; this architecture provides the theoretical foundation for liveness and consistency guarantees under adversarial network conditions. Table 1 summarises TrustFlow's capabilities relative to representative prior systems. To our knowledge, no existing system integrates all six capabilities (CoC provenance, multi-mode watermarking, threshold sharing, time-lock encryption, deletion cascade, and tamper-evident logging) in a single deployable platform.

3. System Architecture

Trust-Flow is organised as a four-layer architecture with a cryptographic engine at its core (Figure 1). Layer labels appear at left; dashed rectangles group subsystem packages; component tabs mark each module. The application layer wraps the CoC framework via a bidirectional Python API; the framework persists state through asyncpg (PostgreSQL) and file I/O.

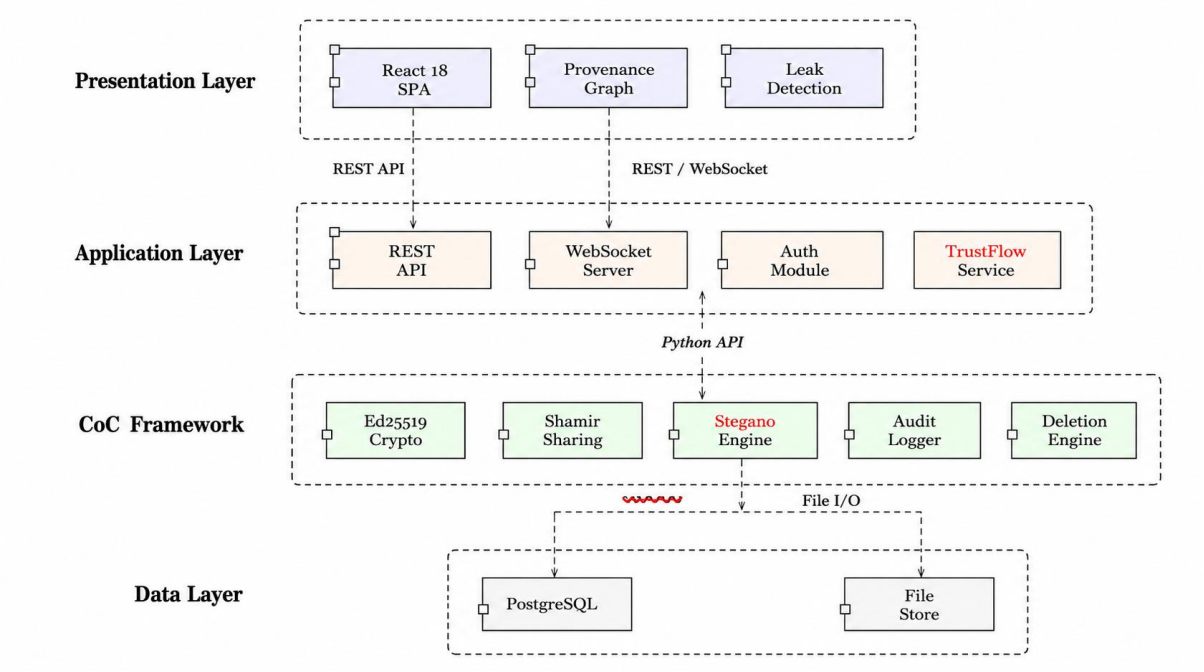

Figure 1: UML component diagram of the trust-flow architecture

3.1. Frontend Layer

The frontend is a React 18 single-page application built with Vite and Tailwind CSS. Key views include:

- **Workspace:** Document grid with upload, share, and inline CoC hash display.
- **Admin Dashboard:** Interactive provenance graph visualisation with pathfinding and chain validation.
- **Group Detail:** Shamir proposal creation, threshold approval, and time-locked secret reveal.
- **Leak Detection:** Suspect file upload, three-mode watermark analysis, and leak history.
- **DM Chat:** WebSocket-based direct messaging with per-document chat rooms.

3.2. Backend Layer

The backend is a FastAPI application that serves REST endpoints and WebSocket connections. Key components:

- **Authentication:** bcrypt password hashing, Ed25519 keypair generation, Argon2id-derived key wrapping, session-based auth with SHA-256 hashed tokens.
- **Document routes:** CRUD, sharing with permission constraints (download, reshare, chat), watermarked copy generation, CoC node creation.
- **Group routes:** group management, Shamir proposal workflow, deletion proposals with threshold approval.
- **Admin routes:** CoC graph queries, leak-detection triggers, leak attribution, and leak history.
- **Trust-Flow service:** bridge between REST endpoints and the coc_framework cryptographic engine.

3.3. Data Layer

PostgreSQL stores 17 tables organised into five domains:

- **Identity:** users, sessions
- **Documents:** documents, file_shares, coc_nodes, coc_recipients, deletion_events, tombstones
- **Groups:** groups, group_members, group_proposals, shamir_shares, group_documents
- **Messaging:** dm_channels, dm_channel_closures, messages
- **Audit:** leak_detections

Documents are stored on the filesystem with watermarked copies maintained alongside originals.

3.4. CoC Framework

The coc_framework package implements all cryptographic subsystems independently of the web layer. This separation enables the framework to be used standalone, in unit tests, or wired into different deployment models. The following sections detail each subsystem.

3.5. Chain-of-Custody Engine

3.5.1. CoC Node Structure

Every document operation produces a CoCNode that forms part of a hash-linked provenance tree (Table 2).

Table 2: CoCNode data structure (simplified)

1	Class Co Cnode:			
2	Schema_Version:	int	#	schema v3
3	Node_Hash:	str	#	SHA -256(sig_hex | content_hash)
4	Content_Hash:	str	#	SHA -256 of document content
5	Parent_Hash:	str	#	parent node hash (None for root)
6	Owner_Id:	str	#	acting peer ID
7	Recipient_Ids:	list	#	sorted recipient list
8	Timestamp:	str	#	ISO 8601 UTC
9	Depth:	int	#	tree level
10	Metadata:	dict	#	operation - specific
11	Signature:	bytes	#	Ed 25519 signature

The signing data is the pipe-delimited canonical string:

$$sig\_data = version \parallel content\_hash \parallel parent\_hash \parallel owner\_id \parallel recipients \parallel ts \parallel metadata \quad (1)$$

The signature field contains an Ed25519 signature over sig _data, binding the node to the actor's identity. The node hash is then computed as:

$$h = H(sig\ hex \parallel content\_hash) \quad (2)$$

where H is SHA-256, since the signature covers the full canonical record and the hash is derived from the signature, any modification to node fields invalidates both the signature and the hash chain.

3.5.2. Provenance Tree

Unlike a linear blockchain, TrustFlow's CoC forms a tree: a document upload creates the root node, and each subsequent operation (share, revoke, reshare, delete) creates a child node referencing its parent's hash. This structure naturally represents branching workflows (e.g., a document shared with three recipients produces three child nodes from the same parent). This tree structure implements how-provenance [6], recording the full derivation path from document creation through all downstream operations.

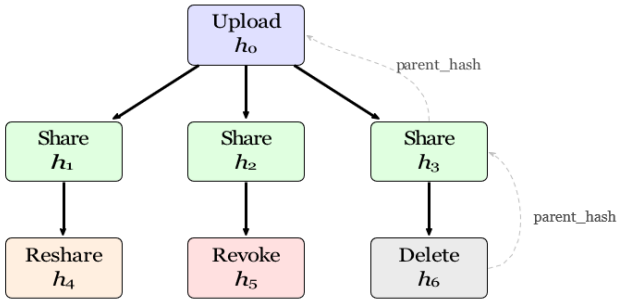

Figure 2: CoC provenance tree

Figure 2 illustrates the provenance tree structure. Each node stores a parent_hash field (dashed arrows) referencing its parent's digest; solid arrows show the parent–child production relationship. A document upload creates the root node, and each subsequent operation (share, revoke, reshare, delete) creates a child node referencing its parent's hash.

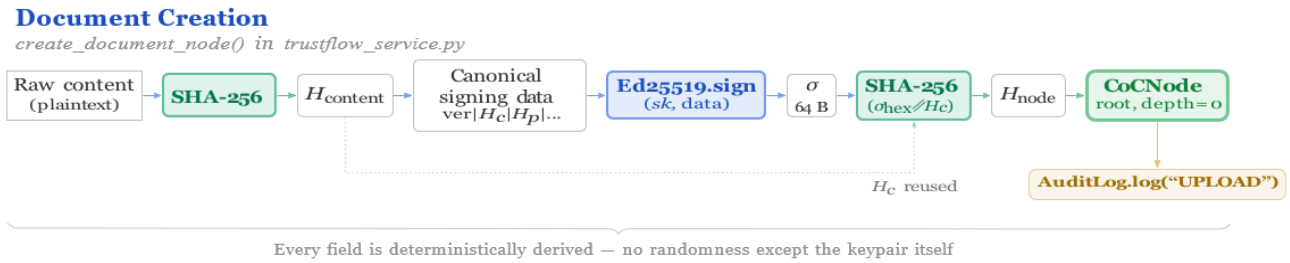

Figure 3: Document creation cryptographic data flow

Figure 3 details the cryptographic data flow during document creation. Raw content is hashed, a canonical signing string is assembled, Ed25519 produces the signature, and the node hash is derived as:

SHA-256(sig hex ‖ content _hash), producing the root CoCNode (3)

3.5.3. Chain Validation

The validation module provides:

- **Hash Verification:** Recompute each node's hash from its fields and verify against the stored node_hash.
- **Signature Verification:** Verify each node's Ed25519 signature using the actor's public key.
- **Chain Integrity:** Walk the tree from leaves to root, verifying that each node's prev_hash matches its parent's node_hash.
- **Temporal Ordering:** Verify that child timestamps are strictly after parent timestamps.

The Admin Dashboard provides an interactive visual representation of the provenance tree (using vis-network) with one-click chain validation and path-finding between arbitrary nodes.

3.6. Cryptographic Subsystems

Steganographic Watermarking: TrustFlow's steganographic engine implements three complementary watermarking modes for text documents [4], each targeting a different embedding channel.

Zero-Width Unicode Characters: Binary data (recipient identifier) is encoded using Unicode zero-width characters: zero-width space (U+200B) for bit 0, zero-width non-joiner (U+200C) for bit 1. These invisible characters are inserted at word boundaries throughout the document. Detection extracts the zero-width sequence and decodes the embedded identifier. Resilience survives copy-paste in most text editors and email clients. Vulnerable to Unicode normalisation or zero-width stripping.

Linguistic Synonym Substitution: Words are selectively replaced with contextual synonyms from a curated mapping (e.g., "important" → "crucial", "begin" → "commence"). The choice of a synonym at each substitution point encodes a bit of the watermark. Detection uses feature-level comparison between the suspect document and the original, analysing which synonym variant was chosen at each substitution point. Resilience survives format conversion, OCR re-digitisation, and partial editing. Difficult to detect or remove without the original synonym mapping.

Whitespace Pattern Encoding: Trailing whitespace (spaces and tabs) is appended to line endings in patterns that encode the watermark. Specifically, the system varies the number of trailing spaces per line to embed binary data. Resilience survives most text editing operations. Vulnerable to automated trailing-whitespace strippers.

Combined Detection: Leak detection runs all three modes in parallel and produces a confidence score. Attribution requires matching at least two of the three modes to positively identify the leak source. The system maintains a watermark_records table

that links each watermarked copy to the recipient and embeds the embedding parameters, and a leak_reports table for forensic history.

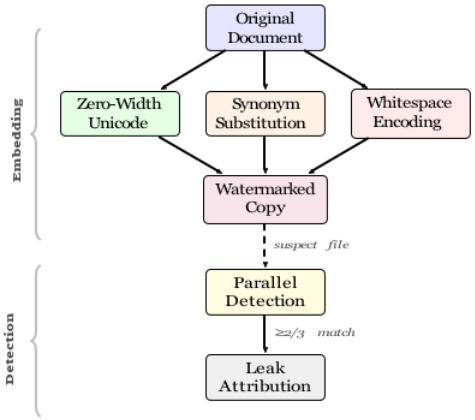

Figure 4: Multi-mode steganographic watermarking pipeline

Figure 4 illustrates the two-phase watermarking pipeline. The embedding phase applies three orthogonal techniques to the original document; the detection phase runs all three extractors in parallel and requires at least two matching modes for positive leak attribution.

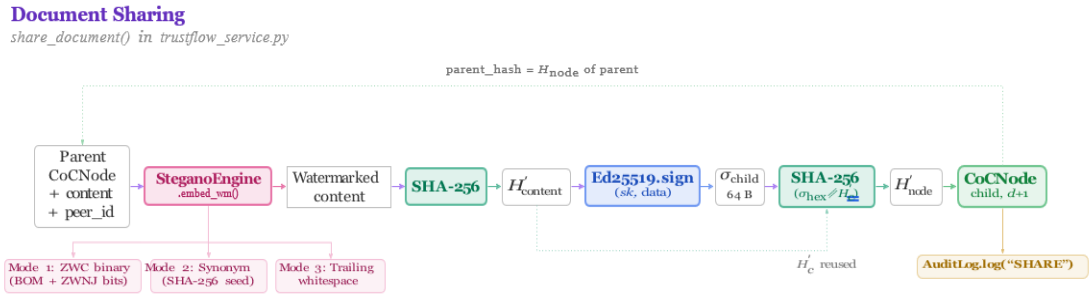

Figure 5: Document sharing cryptographic data flow

Figure 5 illustrates the complete cryptographic data flow for document sharing, which begins with watermark embedding. A parent CoCNode and raw content are passed to the SteganoEngine, which embeds a per-recipient watermark using three modes (ZWC, synonym, whitespace). The watermarked content is then hashed, signed via Ed25519, and linked into the provenance tree as a child CoCNode.

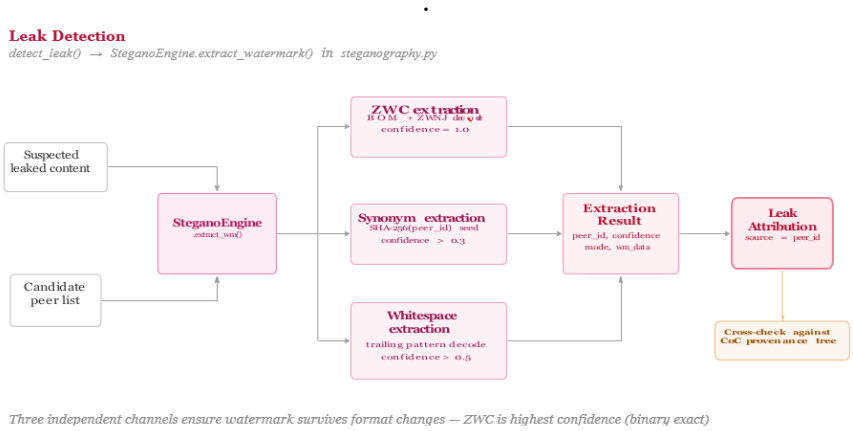

Figure 6: Leak detection cryptographic data flow

Figure 6 depicts the inverse operation: extracting a watermark from a suspected leaked document. A suspect document and candidate peer list are entered into the SteganoEngine, which attempts ZWC, linguistic, and whitespace extraction in priority order, producing an Extraction Result with a confidence score and an attributed peer_id.

3.7. Shamir Threshold Secret Sharing

The secret sharing engine implements a (k, n) threshold scheme with several practical extensions.

3.7.1. Core Protocol

A secret S is split into n shares such that any k shares can reconstruct S via Lagrange interpolation over GF(p), where p is a large prime. Each share is HMAC-authenticated (HMAC-SHA256) to prevent tampering (Table 3).

Table 3: Share structure (simplified)

1	Class Share:			
2	Index:	int	#	share index (1... n)
3	Value:	byte	#	share data
4	Hmac:	byte	#	HMAC - SHA 256 over value
5	Commitment:	byte	#	Feldman commitment

3.7.2. Hybrid Encryption for Large Payloads

For secrets larger than the finite field element size, the engine uses hybrid encryption: the secret is encrypted with a random AES-256-GCM key (Data Encryption Key, DEK), and the DEK is then Shamir-split. Shares contain the encrypted ciphertext alongside the DEK, enabling reconstruction without a separate ciphertext-delivery channel.

3.7.3. Group Workflow Integration

In the web application, Shamir shares a group proposal system:

- A group member creates a proposal, splitting the secret content into n shares (one per group member) with threshold k.
- Each member receives their share and submits an approval (revealing their share to the system).
- Once k approvals are collected, any group member can trigger reconstruction.
- If a time-lock is attached, the secret remains encrypted until the TTL expires, adding temporal access control to the threshold requirement.

This workflow is fully integrated into the web application's group management interface.

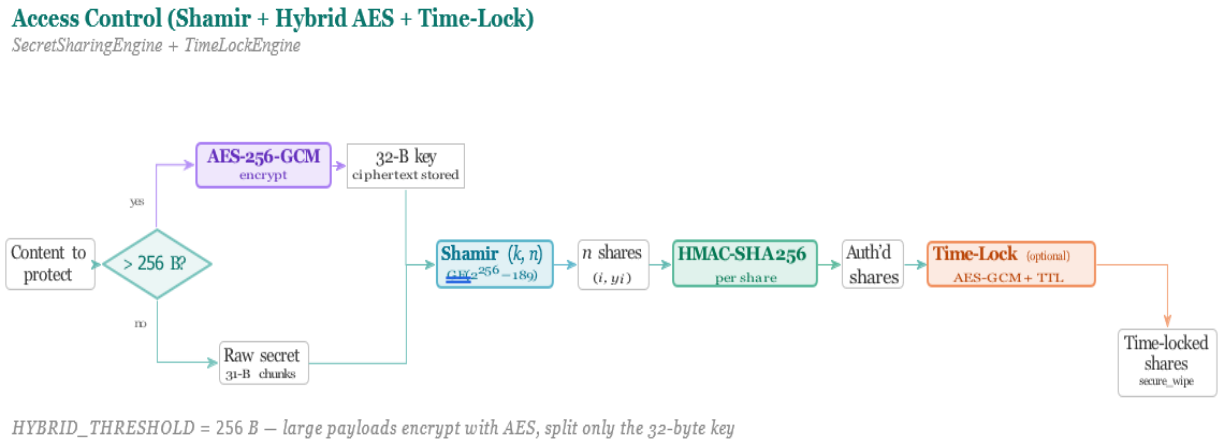

Figure 7: Access control cryptographic data flow

Figure 7 illustrates the complete data flow through the access-control pipeline. Content exceeding 256 bytes is AES-256-GCM encrypted (hybrid mode); smaller secrets are chunked directly. Both paths converge on Shamir (k, n) splitting over GF(2256−189), followed by HMAC-SHA256 per-share authentication and time-lock encryption.

3.8. Time-Lock Encryption

The time-lock engine provides temporal access control through AES-256-GCM encryption with managed TTL [12]; [12].

Encryption and Storage: It uses a random 256 bit AES key and a random 96 bit nonce to encrypt the data in GCM mode. The ciphertext is then authenticated, providing confidentiality and integrity. The system also stores the encrypted ciphertext together with the GCM authentication tag to ensure data authenticity. Also, an expiry timestamp (expires_at) is maintained to indicate the validity period of the encrypted data. The engine also monitors the lifecycle state of the data, categorising it as active, expired or destroyed based on its condition. Also, an optional callback identifier to allow for expiry notifications when the data reaches the specified expiry time.

Lifecycle Management: A background cleanup daemon periodically scans for expired entries and transitions them from active to expired. Once expired, the decryption key is no longer available, rendering the ciphertext unrecoverable via the API. The explicit destroy() method permanently deletes the key material.

Integration with Shamir Proposals: When a group proposal is created with a TTL, the revealed secret (after threshold reconstruction) is time-lock encrypted. The proposal enters a "locked" state until the TTL expires, at which point the underlying secret becomes accessible. This combines two independent access control dimensions: who (threshold) and when (time-lock).

3.9. Deletion Cascade Engine

Document deletion in a system with sharing is non-trivial; Garg et al. [16] formalise this as requiring post-deletion state indistinguishability, and simply deleting the owner's copy leaves watermarked copies with recipients. TrustFlow's deletion engine implements a cryptographically verifiable cascade.

3.9.1. Deletion Protocol

- Token generation: The document owner generates a Deletion Token containing the document ID, the owner's peer ID, timestamp, and an Ed25519 signature.
- Propagation: The token is sent to all recipients (identified from file_shares).
- Local deletion: Each recipient verifies the token signature, deletes their local copy (the original and the watermarked copy), and generates a Deletion Receipt (an Ed25519-signed acknowledgement).
- Receipt collection: The initiator collects receipts and records deletion_events with status (confirmed/pending/failed) per recipient.
- Tombstone: A CoC tombstone node is created, marking the document as deleted in the provenance tree.

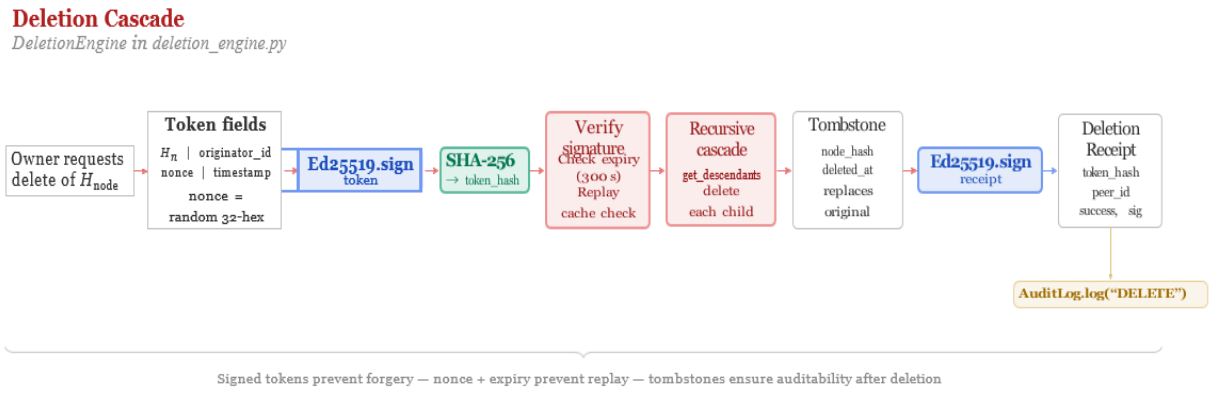

Figure 8: Deletion cascade cryptographic data flow

Figure 8 illustrates the cryptographic data flow of the deletion cascade. The owner constructs a signed Deletion Token, which is SHA-256 hashed and verified (signature, 300s expiry, replay cache). A recursive cascade via get_descendants () removes all children, creates a Content Tombstone, and returns a signed Deletion Receipt.

3.9.2. Replay Prevention

A Processed Token Cache maintains a set of processed token hashes to prevent replay attacks. Each incoming deletion token is checked against the cache before processing.

3.9.3. Retry and Monitoring

The system tracks deletion propagation status per recipient and supports retry for failed deletions. The get_deletion_status() method provides a summary of confirmed, pending, and failed deletions enriched with recipient metadata.

3.10. Tamper-Evident Audit Log

The audit log maintains a hash-chained log of all system events.

3.10.1. Hash Chain Structure

Each log entry contains:

- Entry_hash: SHA-256 of the entry's content
- Prev_hash: Hash of the previous entry (genesis entry uses a sentinel value)
- Timestamp: UTC timestamp
- Event_type: Category of event
- Actor: Identifier of the entity that acted
- Details: Structured data about the event

The chain structure ensures that any modification to a historical entry invalidates all subsequent hashes, providing tamper evidence.

3.10.2. Verification and Export

The audit log supports:

- Integrity verification: Walk the chain and verify each entry's hash against its predecessor
- Filtering: Query by event type, actor, and time range
- Export: Serialise the full or filtered log for external audit tools

3.11. Identity and Access Control

TrustFlow uses Ed25519 key pairs by Josefsson and Liusvaara [11] as the fundamental identity primitive. The identity module provides certificate utilities and a lightweight peer registry: Certificate Authority for issue/validate/revoke workflows, and Peer Info records that bind peer public keys to endpoints and trust metadata. On user registration, the server generates an Ed25519 keypair and protects the signing key with authenticated symmetric encryption. The private key is wrapped with AES-256-GCM using a wrapping key derived from the user's password via Argon2id. Argon2id parameters are chosen to be robust yet practical: a 64 MB memory cost and parallelism of 2. Persisted artefacts are limited to the encrypted blob (stored as salt (16) || nonce (12) || ciphertext), the bcrypt password hash, and the verification (public) key hex. The plaintext signing key never touches persistent storage; it exists only transiently in memory during registration or explicit decryption for authorised signing operations. Session management issues cryptographic session tokens at login. The backend generates a 32-byte token via secrets.token_hex(32); only the SHA-256 digest of this token is stored in the sessions table, while the raw token is delivered to the client as an HTTP-only cookie with SameSite=Lax. Validation re-hashes the cookie value and looks up the digest, verifying expiry before granting access.

Logout removes all session records for the user and instructs the client to delete the cookie; expired sessions are reclaimed lazily during validation checks. Sharing semantics are expressed as a per-share permission record with three boolean flags (can_download, can_reshare, can_chat) and an optional expires_at timestamp. The system default at share creation is

can_download=true, can_reshare=false, can_chat=true, and expires_at=null. When a resharing occurs, permission flags are constrained by intersection: each boolean is computed as the logical AND of the parent's flag and the requested flag, preventing privilege escalation. The effective lifetime is constrained to the earlier of the parent's and the requested expires_at. Expiry is enforced lazily: shares whose expires_at precedes the current time are treated as expired during validation, and access is denied. Groups expose two effective roles: creator and member. Creator status is conferred either by an owner_id match or by an explicit role='creator' value; creators may add or remove members, delete the group, and initiate deletion proposals. Members participate in Shamir secret-sharing proposals and vote on deletion proposals, but lack administrative powers. The group workflow enforces a single active deletion proposal per group to avoid conflicting quorum semantics and simplify resolution.

3.12. Implementation and Evaluation

3.12.1. Implementation Summary

TrustFlow is implemented as a full-stack web application comprising three principal components. The CoC Framework is a standalone Python library that encapsulates all cryptographic primitives (Ed25519 signatures, Shamir secret sharing, steganographic watermarking, time-lock encryption, and hash-chained audit logging). The backend is built with FastAPI and backed by PostgreSQL (with an in-memory fallback for development), exposing RESTful endpoints for document management, group collaboration, and administrative operations. The frontend is a React 18 single-page application styled with Tailwind CSS that communicates with the backend via an Axios HTTP client with cookie-based session authentication. All cryptographic operations are delegated to the CoC Framework, ensuring a clean separation between application logic and security primitives.

3.12.2. Document Lifecycle and Data Flow

A document's journey through TrustFlow proceeds through four cryptographic checkpoints. On upload, the backend receives the raw file, computes its SHA-256 content hash, persists the file to disk, and invokes the CoC Framework's create_document_node() to construct the root provenance node. This node records the content hash, the owner's peer identity, a UTC timestamp, and an Ed25519 signature over the canonical representation. The resulting node hash, computed as SHA-256(signaturehex ‖ content_hash), serves as the document's immutable identifier in the provenance tree. On share, the backend first checks the caller's access permissions via a centralised authorisation layer, then invokes share_document(), which performs two operations atomically: (i) the steganographic engine embeds a Watermark Data payload into the document content, encoding the recipient's peer ID, the current timestamp, the tree depth, the content hash, and a deterministic fingerprint seed derived as the integer value of SHA-256(peer_id)[: 8]; and (ii) a child CoC node is created with its parent_hash set to the parent node's hash, its depth incremented, and the recipient's identity recorded in the recipient_ids field.

The watermarked file persisted separately to allow forensic attribution without exposing the original content. On revocation, the share record's status is set to revoked, removing the recipient's access. The CoC node recording the share event remains in the provenance tree to preserve audit completeness: revocation appends a new signed event rather than erasing history. On deletion, the deletion cascade engine issues a signed deletion token, propagates it to all holders, collects signed receipts, and removes the CoC subtree. This section details the token protocol. The implementation-level flow proceeds as follows: tree traversal via remove_node_tree(), revocation sharing across all recipients, tombstone creation for unreferenced content (with a configurable 300-second grace period), watermarked file cleanup from disk, and deletion_events insertion for audit.

3.13. Cryptographic Pipeline

3.13.1. CoC Node Construction

Each CoCNode (schema version 3) contains eight fields: content_hash (SHA-256 of document content), parent_hash (linking to the parent node or None for roots), owner_id (peer identity), recipient_ids (sorted recipient list), timestamp (ISO 8601 UTC), depth (tree level), metadata (extensible dictionary), and signature (Ed25519 over the canonical signing data). The signing data is constructed as the pipe-delimited concatenation:

version |content_hash |parent_hash |owner_id |recipients|timestamp |metadata (4)

The node hash is then computed per Equation 2 as H = SHA-256(sighex ‖ content_hash). Chain integrity verification walks the tree and recomputes each node's signature and hash, detecting any modification to node fields or to the chain's linkage.

3.13.2. Three-Mode Watermarking

Fingerprint extraction follows a priority cascade: zero-width character (ZWC) extraction is attempted first (confidence 1.0 on exact match), followed by linguistic fingerprint matching, and finally whitespace pattern analysis (confidence threshold > 0.5). In ZWC mode, the JSON-serialised Watermark Data payload is encoded bit-by-bit using U+200B (zero-width space) for 0 and U+200C (zero-width non-joiner) for 1, delimited by U+FEFF (start) and U+2060 (end). Linguistic mode uses a curated synonym mapping (e.g., "important" ↔ "significant"), selecting the variant via (seed + MD5(word)) mod |synonyms|. Whitespace mode appends (seed + line_index) mod 3 trailing spaces per line. The multi-modal strategy ensures that an adversary must simultaneously defeat all three channels to remove attribution.

3.13.3. Hybrid Shamir Secret Sharing

Shamir polynomial evaluation operates over GF(p), where $p = 2256 - 189$, with secrets split into 31-byte segments that fit within the field. For payloads exceeding 256 bytes, the system transparently switches to hybrid mode: a random 256-bit AES key is generated, the payload is encrypted with AES-256-GCM (12-byte nonce), and only the 32-byte AES key is split via Shamir. Each share stores the encrypted ciphertext and nonce alongside the polynomial evaluation, the content hash (SHA-256 of the original secret), the threshold and total-shares parameters, and an HMAC-SHA256 integrity tag computed over index|value|content_hash. Reconstruction applies Lagrange interpolation over the finite field to recover the AES key, then decrypts the ciphertext.

3.13.4. Deletion Token Protocol

A deletion token contains five fields: node_hash (target), originator_id (issuer), nonce (16-byte random hex), timestamp (ISO 8601 UTC), and an Ed25519 signature over the pipe-delimited concatenation of the preceding four fields. Tokens expire after 300 seconds. On receipt, a peer verifies the signature against the originator's public key, checks a ProcessedTokenCache for replay prevention (hash-based deduplication), confirms the token has not expired, removes the target node and any unreferenced content, and creates a Content Tombstone with a configurable grace period. The peer then issues a signed DeletionReceipt covering token_hash|peer_id|timestamp|success|node_hash, which the originator collects via a DeletionTracker to confirm the cascade's completion.

3.14. Key Management

Ed25519 key pairs are generated via PyNaCl's SigningKey.generate(), producing a 32-byte seed that deterministically derives the 64-byte expanded private key and the 32-byte public (verification) key. For persistent storage, the signing key is encrypted using a password-derived wrapping key generated by Argon2id with parameters t=2, m=65,536 (64 MB), p=2, and a 32-byte output, applied to a 16-byte random salt. The signing key bytes are then encrypted with AES-256-GCM using a 12-byte random nonce; the persisted blob is the concatenation salt16 ‖ nonce12 ‖ ciphertext. Decryption reverses the process: the salt and nonce are parsed from the blob prefix, Argon2id re-derives the wrapping key, and AES-256-GCM recovers the signing key. User passwords are separately hashed with bcrypt (cost 12) for authentication; session tokens are SHA-256-hashed server-side. Time-lock encryption uses AES-256-GCM with a fresh random 32-byte key stored in a KeyStore with an associated TTL. An automatic cleanup daemon (1-second polling interval) transitions keys through three states: active → expired → destroyed. Once destroyed, the key material is irrecoverably zeroed, rendering the ciphertext irrecoverable without it.

3.15. Cryptographic Primitives Summary

Table 4 lists the concrete cryptographic parameters used throughout the system.

Table 4: Cryptographic parameters

Primitive	Parameters
Digital signatures	Ed25519 (256-bit keys)
Symmetric encryption	AES-256-GCM (96-bit nonce)
Password hashing	bcrypt (cost 12)
Key derivation	Argon2id (t=2, m=64 MB, p=2)
Hash function	SHA-256 (CoC nodes, audit chain)
Secret sharing	Shamir over GF (2256−189), HMAC-SHA256
Watermarking	3-mode: ZWC + linguistic + whitespace

3.16. Performance Evaluation

To quantify the runtime overhead of TrustFlow's cryptographic operations, researchers benchmarked each core primitive on a commodity workstation (Windows 11, Python 3.12, PyNaCl 1.5, cryptography 44.x). Each operation was executed 100 times (50 for the audit-log test); Table 5 reports the mean and standard deviation.

Table 5: Microbenchmark results for core cryptographic operations (n=100 iterations; audit: n=50)

Operation	Input	Mean (ms)	σ(ms)
Ed25519 keygen	-	0.03	0.01
Ed25519 sign + verify	1 KB	0.11	0.02
AES-256-GCM enc + dec	10 KB	0.04	0.26
Shamir split (3-of-5)	32 B	0.05	0.06
Shamir reconstruct (3 shares)	32 B	0.05	0.03
Watermark embed (ZWC)	1 KB	0.09	0.01
Watermark extract (ZWC)	1 KB	0.23	0.04
Audit log (100 append + verify)	100 entries	1.10	0.19

All individual cryptographic operations complete in under 0.25 ms, confirming negligible latency for interactive use. The most expensive single operation (zero-width watermark extraction) averages 0.23 ms, dominated by Unicode scanning of the watermarked text. The audit-log benchmark, which appends 100 hash-chained entries and then walks the entire chain for verification, completes in approximately 1.1 ms, demonstrating that integrity verification scales linearly and remains practical even for substantial event histories.

3.17. Threat Model and Security Analysis

3.17.1. Adversary Model

Researchers consider three classes of adversary, ordered by increasing capability:

- A1, External attacker. A network-level adversary who can eavesdrop on, intercept, or replay messages but holds no valid credentials. Researchers assume TLS 1.3 protects the transport channel; A1 therefore targets stored data and leaked documents.
- A2, Malicious insider. A legitimate user who attempts to leak documents, escalate permissions, forge provenance entries, or deny participation in a sharing chain.
- A3, Compromised server. An adversary with full read/write access to the server's database and runtime memory. This is the strongest adversary considered; full mitigation requires the decentralised extensions.

Assumptions: (i) Ed25519 Josefsson and Liusvaara [11] and AES-256-GCM Kavun et al. [12] remain computationally secure; (ii) TLS 1.3 provides a confidential and authenticated transport; (iii) for Shamir threshold operations, the dealing phase is honest (i.e., the server correctly distributes shares).

3.17.2. Security Properties

Table 6 maps each security goal to the mechanism that provides it and the adversary classes it defends against.

Table 6: Security properties and their enforcement mechanisms

Property	Mechanism	Adv.
Provenance integrity	Hash-chained CoC + Ed25519 signatures	*A*1,2
Confidentiality	AES-256-GCM + Shamir threshold	*A*1*
Non-repudiation	Ed25519-signed CoC nodes	*A*2
Leak traceability	3-mode steganographic watermarking	*A*2
Secure deletion	Cascade tokens + signed receipts	*A*2
Temporal access control	Time-lock encryption (AES-GCM + key TTL)	*A*1,2
Audit tamper evidence	SHA-256 hash-chained log	*A*2*

**Partial A3 resistance; full defence requires Phase 5 decentralisation.*

- **Provenance integrity:** Each CoC node includes a SHA-256 hash of its predecessor and an Ed25519 signature from the acting user. An external attacker cannot forge entries without a valid signing key; a malicious insider's entries are non-repudiably bound to their identity.
- **Confidentiality:** Sensitive material (signing keys, time-locked payloads, and Shamir-split secrets) is protected by AES-256-GCM; signing keys are wrapped with Argon2id-derived keys so that only the ciphertext is stored at rest. Shamir's secret sharing ensures that no single share reveals the encryption key; reconstruction requires k of n shares.
- **Leak traceability:** The three-mode watermarking pipeline embeds a unique per-recipient fingerprint. Zero-width characters provide high-confidence attribution; linguistic and whitespace modes offer redundancy if the zero-width layer is stripped.
- **Secure deletion:** The deletion cascade protocol uses Ed25519-signed tokens and requires signed receipts from every recipient. A processed-token cache prevents replay attacks. Incomplete cascades are tracked for retry.

4. Conclusion

Researchers have presented TrustFlow, a Chain-of-Custody-Based Traceability Framework that integrates six cryptographic subsystems, namely hash-linked provenance trees, multi-mode steganographic watermarking, Shamir threshold secret sharing, time-lock encryption, deletion cascade with signed tokens, and tamper-evident audit logging, into a working, full-stack document collaboration platform. The system further includes a complete P2P gossip networking stack, currently validated through simulation tests and designed for integration in future decentralisation phases. Each subsystem enforces a distinct security property. Ed25519-signed, hash-chained provenance trees guarantee non-repudiable audit trails. Three orthogonal steganographic modes (zero-width Unicode, linguistic synonym substitution, whitespace encoding) provide per-recipient fingerprinting with redundant attribution. Shamir (k, n)-threshold secret sharing protects group secrets and proposals without exposing the plaintext to any single party. Time-lock encryption adds a temporal dimension to access control, and the deletion cascade protocol enforces verifiable removal across all recipients via signed tokens and receipts. A SHA-256 hash-chained audit log records every operation, producing tamper-evident forensic history. From an engineering standpoint, the framework demonstrates practical viability: all individual cryptographic operations complete in under 0.25 ms, the full test suite passes over 600 tests, and the architecture cleanly separates the standalone CoC framework library from the Fast API backend and the React frontend.

The threat model addresses three adversary classes (external, insider, compromised server), with the current deployment providing strong defences against the first two and architectural provisions for the third through a five-phase decentralisation roadmap. Looking ahead, Phase 1 of this roadmap (the deletion cascade) is already complete, and Phase 2 (the abstract interface layer) defines abstractions for Ledger Backend, Content Store, and Key Manager that decouple the application from specific storage and verification backends. Phases 3–5 progressively integrate the P2P gossip network, anchor CoC events to a public blockchain, and introduce threshold-gated document encryption keys, collectively shifting the trust model from server-dependent to cryptographically self-sovereign. Trust-Flow demonstrates that comprehensive document traceability, combining provenance integrity, leak attribution, threshold access control, temporal constraints, verifiable deletion, and tamper-evident logging, is achievable within a practical web application. The architectural foundation for fully decentralised trust has been laid incrementally, ensuring that each phase delivers immediate value while laying the path toward a system in which security guarantees are rooted in mathematics rather than institutional trust.

4.1. Limitations

The current architecture is designed around a centralised trust model, with full document-level encryption (Shamir-gated DEKs) architecturally planned for Phase 5 of the decentralisation roadmap. Until that phase is active, resilience against A3 depends on standard server-hardening practices. The following constraints apply to the present deployment:

- The server acts as the honest dealer for Shamir share distribution; verifiable secret sharing (VSS) is a planned extension that would detect malformed shares from a compromised dealer.
- The audit log is stored on the server; A3 could truncate or rewrite the chain (though the hash structure makes undetectable modification infeasible if any client retains a copy of a prior hash). Phase 4 blockchain anchoring eliminates this vector.
- As with all text-based steganographic techniques, watermark persistence degrades under heavy paraphrasing; redundancy across three independent embedding modes mitigates partial removal.

The five-phase decentralisation roadmap addresses these constraints incrementally: client-side key management (Phase 2), blockchain-anchored audit integrity (Phase 4), and threshold-gated document encryption keys (Phase 5) collectively shift the

trust model from “trust the server” to “trust the math.” Formal protocol verification (e.g., ProVerif, Tamarin) is planned as part of the security hardening effort accompanying these phases.

4.2. Future Work

Peer-to-Peer Network Framework: While the current web application operates in a client-server model, TrustFlow includes a fully implemented P2P networking stack (located in coc_framework/network/) designed for future decentralised deployment. This stack is validated through simulation tests but is not yet wired into the web application.

Protocol Layer: The protocol module defines a MessageType enum with 13 message types (including SHARE, DELETE, REVOKE, VERIFY, KEY_EXCHANGE, HEARTBEAT, GOSSIP_PULL/PUSH, and ANTI_ENTROPY request/response). All messages are wrapped in Signed Envelope structures containing an Ed25519 signature, sender ID, timestamp, and nonce.

Gossip Protocol: The GossipPeer class implements epidemic-style gossip dissemination (Coretti et al. [19]) using ZeroMQ ROUTER/DEALER sockets. Features include configurable fanout, pull-based anti-entropy synchronisation, a Message Cache with TTL-based expiry, message deduplication via unique IDs, and protocol versioning.

Network Coordination: The Network Coordinator orchestrates multi-process peer networks, spawning Network Peer processes that each run a Gossip-Peer instance and handle message dispatch. This infrastructure is validated through network simulation tests.

4.3. Decentralisation Roadmap

Researchers have designed a five-phase roadmap to transition Trust-Flow from its current centralised model to a fully decentralised P2P system. Phase 1 (deletion cascade) is complete; Phases 2–5 are planned.

4.3.1. Phase 2: Abstract Interface Layer

Three abstract base classes decouple the application from specific storage and verification backends:

- Ledger Backend: abstracts where CoC events are recorded (local PostgreSQL or public blockchain)
- Content Store: abstracts where document blobs are stored (local filesystem or IPFS/Arweave)
- Key Manager: abstracts where cryptographic keys live (server-side database or client-side browser storage)

Local default implementations ensure zero behaviour change until a decentralised backend is configured [18].

4.3.2. Phase 3: Service Integration

Wire the P2P gossip network into the web application’s TrustFlowService. REST endpoints delegate event propagation to the gossip layer; incoming gossip messages update local state. The server becomes one peer among many rather than the sole authority.

4.3.3. Phase 4: Blockchain Anchoring

Anchor CoC events to a public blockchain (Ethereum [1, 7, 8] / Polygon for production). A Solidity smart contract stores document hashes, event types, and timestamps. Batch anchoring amortises gas costs by collecting multiple events into Merkle trees and anchoring only the root hash on-chain.

4.3.4. Phase 5: Threshold-Gated Access Control

Extend Shamir secret sharing to protect document encryption keys (DEKs) and proposals. Access to a group document requires k of n members to cooperate in DEK reconstruction. Combined with time-lock encryption, this provides both who and when dimensions of access control without any trusted central authority.

4.3.5. Trust Model Shift

The cumulative effect of Phases 2–5 is a fundamental shift in the trust model: from “trust the server” to “trust the math.” In the current system, a compromised server exposes all documents, forges all history, and bypasses all access control. In the target

system, a compromised server can deny service. Still, it cannot read encrypted documents (no DEK), forge CoC entries (no private keys), tamper with anchored history (blockchain immutability), or bypass threshold access (insufficient Shamir shares).

Acknowledgement: The authors express their sincere gratitude to SRM Institute of Science and Technology at Ramapuram for providing the facilities, resources, and academic environment that supported the successful completion of this research.

Data Availability Statement: The data supporting the findings of this study are available from the corresponding author upon reasonable request.

Funding Statement: No financial assistance, grants, or external funding were received for the preparation of this manuscript and the conduct of this research.

Conflicts of Interest Statement: The authors declare that there are no conflicts of interest concerning the publication of this research article.

Ethics and Consent Statement: This study was conducted in accordance with applicable ethical guidelines, and informed consent was obtained from all participants.

References

1. A. Ramachandran and M. Kantarcioglu, "Using blockchain and smart contracts for secure data provenance management," *arXiv preprint*, 2017. [Accessed by 23/01/2025]
2. F. Paci, A. Squicciarini, and N. Zannone, "Survey on access control for community-centered collaborative systems," *ACM Computing Surveys*, vol. 51, no. 1, pp. 1–38, 2018.
3. X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla, "Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability," in 17th IEEE/ACM International Symposium on Cluster, *Cloud and Grid Computing*, Madrid, Spain, 2017.
4. N. S. Kamaruddin, A. Kamsin, L. Y. Por, and H. Rahman, "A review of text watermarking: Theory, methods, and applications," *IEEE Access*, vol. 6, no. 1, pp. 8011–8028, 2018.
5. A. Chandramouli, A. Choudhury, and A. Patra, "A survey on perfectly secure verifiable secret-sharing," *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1–36, 2022.
6. M. Herschel, R. Diestelkämper, and H. Ben Lahmar, "A survey on provenance: What for? what form? what from?" *The VLDB Journal*, vol. 26, no. 6, pp. 881–906, 2017.
7. N. Nizamuddin, K. Salah, M. Ajmal Azad, J. Arshad, and M. H. Rehman, "Decentralized document version control using ethereum blockchain and IPFS," *Computers and Electrical Engineering*, vol. 76, no. 6, pp. 183–197, 2019.
8. N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on Ethereum smart contracts (SoK)," in *International Conference on Principles of Security and Trust (POST)*, Berlin, Heidelberg, 2017.
9. S. G. Rizzo, F. Bertini, and D. Montesi, "Content-preserving text watermarking through unicode homoglyph substitution," in *Proceedings of the 20th International Database Engineering & Applications Symposium (IDEAS)*, Quebec, Canada, 2016.
10. J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein, "A watermark for large language models," i*n Proceedings of the 40th International Conference on Machine Learning (ICML)*, Hawaii, United States of America, 2023.
11. S. Josefsson and I. Liusvaara, "Edwards-curve digital signature algorithm (EdDSA)," *RFC 8032*, 2017. [Accessed by 23/01/2025]
12. E. B. Kavun, H. Mihajloska, and T. Yalçin, "A survey on authenticated encryption—ASIC designer's perspective," *ACM Computing Surveys*, vol. 50, no. 6, pp. 1–21, 2017.
13. E. Barker, "Recommendation for key management: Part 1 – general," *NIST,* Tech. Rep. SP 800-57 Part 1 Rev. 5, 2020. [Accessed by 25/01/2025]
14. A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: New generation of memory-hard functions for password hashing and other applications," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, Saarbruecken, Germany, 2016.
15. D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," *in Proc. 38th Annu. Int. Cryptology Conf. (CRYPTO 2018), Santa Barbara*, California, United States of America, 2018.
16. S. Garg, S. Goldwasser, and P. N. Vasudevan, "Formalizing data deletion in the context of the right to be forgotten," in *Advances in Cryptology – EUROCRYPT 2020*, Zagreb, Croatia, 2020.
17. P. P. Kumar, S. P. Kumar, and P. J. A. Alphonse, "Attribute based encryption in cloud computing: A survey, gap analysis, and future directions," *Journal of Network and Computer Applications,* vol. 108, no. 4, pp. 37–52, 2018.

18. E. Daniel and F. Tschorsch, "Ipfs and friends: A qualitative comparison of next generation peer-to-peer data networks," *IEEE Communications Surveys and Tutorials*, vol. 24, no. 1, pp. 31–52, 2022.
19. S. Coretti, A. Kiayias, C. Moore, and A. Russell, "The generals' scuttlebutt: Byzantine-resilient gossip protocols," in *ACM Conference on Computer and Communications Security (CCS)*, California, United States of America, 2022.