International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Blockvault: Blockchain-Based Document Verification and Management

Mr. Anil Arjun Astage (Roll No. 23) | **Mr. Sankalp Suhas Patil** (Roll No. 32)
M.Sc. Computer Science, Department of Computer Science
Chikitsak Samuha's S.S. & L.S. Patkar College of Arts & Science &
V.P. Varde College of Commerce & Economics, Mumbai, India
Project Guide: **Mrs. Ahinsa Gaikwad** | Academic Year 2026–2027

Abstract—BlockVault is a web-based system for registering and checking the integrity of digital documents. When a file is uploaded, the system computes its SHA-256 fingerprint and records that fingerprint through a Solidity smart contract, while the file itself and all related application data remain off-chain. The platform is built from a React and TypeScript frontend, an Express and TypeScript backend, a MongoDB database, and an Ethereum-compatible blockchain. Users can register documents, verify either a file or a hash, organize their records, request access to files owned by others, and review verification history. A central design decision is that proving a document is authentic is kept separate from being allowed to download it. This paper presents the design, modules, implementation, security controls, and test coverage described in the project documentation. The documentation reports 51 backend tests and 9 smart-contract tests; since execution logs were not available, pass status is not independently confirmed. No performance benchmarks exist, so the discussion stays within the documented functions of the system.

Index Terms—Blockchain, document verification, SHA-256, smart contract, access control, document management.

I. INTRODUCTION

Files travel easily between people and institutions, and they are just as easily copied, edited, and passed along in altered form. Anyone who receives a document therefore has a reasonable question to ask: is this the same file that was originally issued? The person who owns the document faces a second, quite different question: who should be permitted to obtain the original? BlockVault was designed to address both concerns inside a single application, using a blockchain record as a lasting reference to a document's fingerprint.

At the heart of the system is a SHA-256 digest computed from the exact bytes of an uploaded file. Only this digest is written to the blockchain; the content of the document never enters the smart contract. The files and the application's own records are stored separately from the on-chain proof. To check a document later, a user can upload the file so that it is hashed and compared against the registered entry, or can paste in a hash directly. Each check ends in one of three outcomes: VERIFIED, MODIFIED, or NOT_FOUND.

The project, as described in its blackbook, is a full-stack web application. Its interface uses React 18, TypeScript, Vite, and Tailwind CSS. The server side runs on Node.js with Express and TypeScript, and stores application records in MongoDB through Mongoose. The blockchain portion is written in Solidity, developed with Hardhat, and accessed from the backend through ethers.js. The parts communicate through the application's APIs and a dedicated blockchain service.

The contributions of this paper are limited to what the project materials document. First, it describes how a hash-only on-chain registration can be combined with conventional off-chain storage. Second, it explains how verification and access control are treated as independent workflows. Third, it summarizes the modules, security measures, deployment targets, and test coverage reported for the system, and it states plainly where the available evidence stops. A live instance of the application is referenced in [7].

II. BACKGROUND AND RELATED CONCEPTS

A. Cryptographic Fingerprints

A cryptographic hash function turns an input of any size into a fixed-length output. SHA-256, specified in the Secure Hash Standard [4], produces a 256-bit value. Two properties make it useful here: changing even a single byte of the input yields a completely different digest, and it is computationally impractical to construct a different file that produces a chosen digest. Together these properties let a short value act as a stand-in for an entire document when the question is simply whether two files are identical.

B. The Blockchain as a Registration Layer

A blockchain is an append-only ledger maintained by many participants rather than by one authority [2]. Ethereum extends this idea by allowing programs, called smart contracts, to run on the ledger and to keep their own state [3]. When a record is written through such a contract, it is difficult to alter afterwards, and the time and block of its creation are available to anyone who inspects the chain. These characteristics suit a registry whose main job is to show that a particular fingerprint was recorded by a particular account at a particular time.

C. Why Document Content Stays Off-Chain

Placing complete files on a public ledger would be costly and would expose their contents. BlockVault avoids both problems by storing only a 32-byte hash on-chain and keeping the file in the application environment. The blockchain then answers a narrow question—was this fingerprint registered, and by whom—while the backend and database handle everything else.

D. Verification Versus Access

Confirming that a file matches a registered fingerprint is not the same as being entitled to receive that file. A public record can support the first activity, but it cannot deliver a document or authorize a download. BlockVault therefore pairs its verification features with a separate set of access-management workflows, a distinction that recurs throughout the rest of this paper.

E. Comparison with Conventional Approaches

Document integrity is often handled today by keeping fingerprints in an ordinary database run by one organization, by attaching digital signatures to files, or by relying on a third-party timestamping service. Each of these works, but each also asks the verifier to trust whoever operates the record. A central database can be edited by its administrators, and a timestamp is only as dependable as the service that issued it. Anchoring the fingerprint on a shared ledger changes where that trust sits: the registration is visible to anyone who can read the chain, and it cannot be quietly rewritten by a single party.

This does not make the other tools obsolete. A signature still tells a recipient who vouched for a file, which a bare hash does not, and a conventional database is still the natural place to keep searchable metadata. BlockVault takes the pragmatic route of using the blockchain for the single job it does well, a tamper-resistant registration, and using ordinary application components for everything else.

III. PROBLEM STATEMENT AND OBJECTIVES

The problem addressed by BlockVault has three connected parts: registering a document's fingerprint, checking a later file against that registration, and controlling who may obtain the stored document. These are related but distinct tasks. A blockchain entry can show that a fingerprint exists; it does not hand over the document or decide who may download it. The application consequently offers verification functions and access-management workflows side by side.

The objectives recorded in the blackbook are secure document registration, tamper detection through hash comparison, blockchain-backed proof, access management, identity and security controls, administrative functions, and support for deployment. In addition, the system keeps a record of relevant activity, such as verifications, downloads, access requests, and notifications.

IV. SYSTEM ARCHITECTURE

BlockVault is arranged in four tiers: frontend, backend, database, and blockchain. The browser displays the React interface, which talks to the Express REST API. The backend is the coordinating layer; it handles authentication, document processing, application records, verification logic, and blockchain operations. Application data lives in MongoDB, and a blockchain service within the backend communicates with the DocumentVerification smart contract. Fig. 1 shows how these pieces relate.

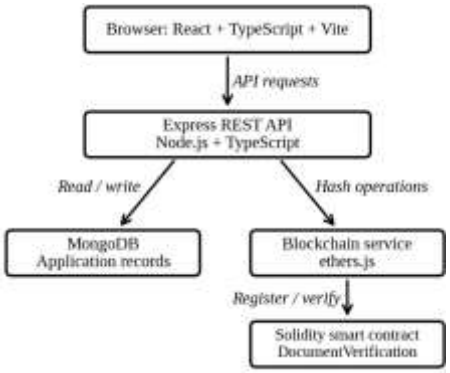


Fig. 1. High-level architecture of BlockVault.

On the contract side, each record holds a bytes32 document hash along with fields described in the blackbook: an existence flag, the owner's wallet address, a timestamp, and a block number. When a document is registered, the contract emits a DocumentRegistered event. The file never leaves the application environment, which allows the blockchain to serve purely as a verification record.

Table I lists the technologies used at each tier, as stated in the project documentation.

TABLE I
TECHNOLOGY STACK OF BLOCKVAULT

Tier	Technologies
Frontend	React 18, TypeScript, Vite, Tailwind CSS, React Router, Axios, Recharts
Backend	Node.js, Express, TypeScript, JWT, Zod, Helmet
Database	MongoDB with Mongoose
Blockchain	Solidity, Hardhat, ethers.js v6, Sepolia test network (chain ID 11155111)
Deployment	Docker Compose, MongoDB Atlas, Render, Vercel

V. METHODOLOGY ADOPTED

Development followed an iterative approach. The application was split into frontend, backend, database, and blockchain modules, and each major subsystem could be built and tested on its own before being integrated with the others. The blackbook lists six stages in the workflow: requirement analysis, design, implementation, integration, testing, and refinement.

Requirement analysis established the need for document integrity, verification, and controlled access. The design stage defined the layered architecture, the data models, the API routes, and what the smart contract would be responsible for. During implementation, the React frontend, the Express backend, and the Solidity contract were developed. Integration then joined the frontend to the backend APIs and connected application operations to MongoDB and the blockchain services.

Testing spanned authentication, document operations, verification, access control, blockchain registration, and supporting services. Refinement was the stage at which validation, security, integration problems, and the user workflow were revisited. Fig. 2 gives a simplified picture of the document integrity workflow that these stages produced.



Fig. 2. Simplified document integrity workflow.

VI. SYSTEM MODULES

A. Identity and Authentication

The identity module covers registration, email verification, login, and the multi-factor authentication options documented in the project. Role checks separate USER permissions from ADMIN permissions, so that administrative operations are not available to ordinary accounts.

B. Document Management

Registered records can be uploaded and managed through the document-management module. Around this core, the project provides document versioning, folders, tags, certificates, search, and a history of verifications, which together let users keep a growing collection of records in order.

C. Registration and Verification

Registration ties the SHA-256 digest of a file to a record in the smart contract. Verification accepts either a document file or a hash and returns one of the defined outcomes. VERIFIED indicates a match with a registered record, MODIFIED indicates that the supplied content differs from the registered one, and NOT_FOUND indicates that no corresponding registration exists.

D. Sharing, Notifications, and Administration

Access and sharing functions let a user request access to a document and let the owner approve or deny the request, after which delivery is controlled. The documented module list also includes notifications, analytics, activity records, blockchain exploration, and administration. Table II summarizes the modules.

TABLE II
SUMMARY OF DOCUMENTED MODULES

Module	Documented functions
Identity	Registration, email verification, login, multi-factor options, USER/ADMIN roles
Documents	Upload, versioning, folders, tags, certificates, search, history
Blockchain	Hash registration, receipt and event checks, blockchain exploration
Verification	File or hash input; VERIFIED, MODIFIED, NOT_FOUND
Access	Access requests, approval or denial, controlled downloads
Operations	Notifications, analytics, activity records, administration

The blackbook documents an upload limit of 10 MB. If no wallet is connected at the time of registration, the record can be left in a PENDING state and completed afterwards, either individually or through bulk registration. These are presented as documented features of the system and not as claims about measured user experience.

VII. IMPLEMENTATION DETAILS

A. Frontend

The user interface is written in React 18 and TypeScript and is built with Vite and styled with Tailwind CSS. React Router handles navigation among public pages, authenticated pages, and administrator pages. Axios is listed among the frontend libraries for communicating with the API, and Recharts supplies charts for analytics views. The frontend's role is to present actions and results; the decisions about validation and authorization are made on the server.

B. Backend and Database

The backend uses Node.js, Express, and TypeScript, and is organized into routes, controllers, services, models, and middleware. This separation keeps API entry points distinct from business operations and from persisted data. MongoDB and Mongoose hold the application records. The backend processes documents, coordinates verification, and applies validation and authorization checks to every protected operation.

C. Blockchain Layer

The contract is called DocumentVerification and is written in Solidity [6]. Contract development and testing use Hardhat, and the backend reaches the chain through ethers.js version 6. The documented operations include registerDocument(bytes32 documentHash) and verifyDocument(bytes32 documentHash), and the implementation also describes a getDocument function. The contract logic prevents the same hash from being registered twice.

As part of validating a registration, the backend inspects the blockchain transaction receipt and looks for the DocumentRegistered event. The documentation

names Sepolia as the target test network, with chain ID 11155111. It should be noted that having configuration for a network is not evidence that a deployment is live or continuously available.

Several features of this contract design are worth explaining. Because a record stores an existence flag, a lookup for an unregistered hash can be answered without ambiguity, which is what makes the NOT_FOUND outcome possible. Storing the owner's wallet address, the timestamp, and the block number alongside the hash means that each registration carries its own context: who submitted it and when it entered the chain. The event emitted at registration gives the backend a second, independent place to confirm that a transaction did what it was supposed to do. Finally, rejecting duplicate hashes means a fingerprint has exactly one registrant, so a later submission of the same file cannot overwrite or reassign the original entry.

D. Registration and Verification Workflow

A registration begins when a user uploads a file. The backend hashes the file contents with SHA-256, the blockchain service submits the digest to the contract, and the resulting transaction is checked before the record is finalized. If a wallet is not connected, the entry waits in the PENDING state until registration is completed.

Verification can start from either a file or a hash. A file is hashed first; a pasted hash is used as it is. The digest is then looked up against the registered records, and the application reports VERIFIED, MODIFIED, or NOT_FOUND. Verification activity is recorded so that users can later review it.

The three outcomes should be read with care. VERIFIED means the digest matches a registered entry, so the file is byte-for-byte identical to what was registered. MODIFIED means a record exists but the supplied content does not match it, which is the signal that a file may have been changed. NOT_FOUND means no registration corresponds to the supplied digest. None of the three says anything about whether the document's contents are true; they speak only to whether the file is the same one that was registered.

E. Technology Choices

The technologies named in Table I are common, widely documented choices, and the reasons for using them are largely practical. TypeScript on both the client and the server lets the two sides share a typed vocabulary of data structures, which reduces mismatches at the API boundary. Express keeps the REST layer lightweight, and MongoDB suits records such as folders, tags, and notifications whose fields may vary. Hardhat provides a local environment for compiling and testing Solidity contracts before they are pointed at a public test network, and ethers.js gives the backend a compact way to submit transactions and read receipts. Using one language family across the stack also keeps the project approachable for a small team.

VIII. ILLUSTRATIVE USAGE SCENARIOS

The following scenarios are offered to show how the documented features fit together. They are illustrative examples and are not case studies or reported deployments.

In the first scenario, an institution registers a document, such as a certificate, when it is issued. The SHA-256 digest is recorded on-chain and the file is kept in the application. A third party who later receives a copy can upload it, or submit its hash, and see whether the result is VERIFIED. If a single character of the file was altered, the result would instead be MODIFIED, and a file that was never registered would return NOT_FOUND. At no point does the third party need permission to download the original.

In the second scenario, a user wants to share a registered document with a colleague. The colleague submits an access request, the owner approves it, and the system issues a one-time link that expires after 15 minutes. Because the link is consumed when it is used, a forwarded copy of the link does not give a second person the same access. The owner's approval controls the off-chain file, while the blockchain record stays unchanged.

In the third scenario, a team works on a document that changes over time. Each revision can be stored through the versioning feature and organized in folders and tags, and the verification history shows when checks were performed. This gives the team a documented trail of what was registered and checked, without placing the working files on a public ledger.

IX. SECURITY AND ACCESS CONTROL

A. Documented Security Measures

The project documents several protective measures: JWT-based authentication [5], password hashing, role checks for USER and ADMIN, Zod validation of inputs, request rate limiting, Helmet security middleware, validation of uploaded files, and authorization checks on protected downloads. The backend configuration requires a JWT secret in production. These controls are reported here as implementation features recorded in the blackbook; the paper does not claim that an independent security audit was carried out.

B. Controlled Access to Off-Chain Files

Having access to a document is not the same as being able to confirm its authenticity. Owners and administrators may download registered files directly. Other users must either request access or use the controlled sharing mechanisms. According to the blackbook, approved one-time access links last 15 minutes, download codes rotate every 20 minutes, and access tokens are consumed atomically. All of these controls are intended to govern access to the off-chain file. Table III collects the documented parameters.

TABLE III
DOCUMENTED OPERATIONAL PARAMETERS

Parameter	Documented value
Upload size limit	10 MB
One-time access link lifetime	15 minutes
Download code rotation	Every 20 minutes
Access token consumption	Atomic
Verification outcomes	3 (VERIFIED, MODIFIED, NOT_FOUND)

C. Threat Considerations

It helps to be clear about what the design protects against and what it does not. The hash comparison detects any change to a file after it was registered, since a modified file produces a different digest. The one-time links, rotating codes, and role checks are aimed at unauthorized retrieval of the off-chain file, and rate limiting and input validation are aimed at misuse of the API.

Some risks sit outside these controls. A blockchain entry shows that a fingerprint was registered by a given wallet; it does not prove that the registrant was the original author of the document. A document that was already altered before it was registered will verify successfully against its own altered fingerprint. And the confidentiality of stored files depends on the backend and storage environment, not on the chain. The future-scope items on key management and content inspection, discussed later, address some of these points. They are limits of the approach and are stated here to avoid overstating what verification means.

D. Privacy Considerations

Writing only a hash to the chain is a sensible privacy measure, because the digest does not reveal the document it was computed from. Even so, a public chain exposes other information: the wallet address that registered the entry, the timestamp, and the block number. Anyone who already holds a candidate file can also hash it and check whether it was registered. For this reason, the protection of the file itself is placed with the application's access controls, and the chain is used only as a registry of fingerprints.

X. TESTING AND DOCUMENTED RESULTS

The blackbook reports 51 backend tests, spread across the access, folder, notification, versioning, and authentication suites, together with 9 Hardhat tests for the smart contract. The areas covered include authentication, document operations, verification, access control, blockchain registration, and supporting services. The archive provided for this paper did not contain test-run logs, so it cannot be established that every test passed. For this reason the counts in Table IV are treated as documented coverage and not as independently verified outcomes.

TABLE IV DOCUMENTED TEST COVERAGE

Component	Tests	Status
Backend suites	51	Logs not supplied
Smart contract (Hardhat)	9	Logs not supplied

The functional outcomes documented for verification are VERIFIED, MODIFIED, and NOT_FOUND. Documented system functions include file registration, hash-based verification, access requests, approval or denial of those requests, controlled downloads, notifications, document organization, versioning, and administrative operations.

No benchmark data for latency, throughput, transaction cost, or comparative performance was provided with the project archive. As a result, this paper makes no quantitative claims about speed, scalability, or accuracy.

Proposed Evaluation Plan

Because measured results are not available, a sensible next step is to define what should be measured. The following plan is a proposal and does not describe work that has been done. First, the test suites should be executed in a clean environment and their logs preserved, so that the 51 backend and 9 contract tests can be reported as actual pass or fail results. Second, the time taken for registration and for verification should be recorded for files of different sizes up to the 10 MB limit. Third, the transaction cost of registering a hash on the Sepolia test network should be noted. Fourth, behavior under several simultaneous uploads and verifications should be observed. Finally, a set of deliberately altered files should be checked to confirm that each is reported as MODIFIED. Together these measurements would turn the documented coverage into evidence that readers can assess for themselves.

XI. DEPLOYMENT

The deployment documentation describes a Docker Compose setup in which MongoDB, the backend, and the frontend run as separate services. It also describes a hosted configuration that uses MongoDB Atlas, Render, and Vercel, while the blockchain configuration points to Sepolia. These are the technologies and targets named in the project materials. This paper does not independently test the availability, security posture, or production readiness of any deployed instance.

XII. DISCUSSION

Several design choices in BlockVault are worth drawing attention to. Recording only a hash on the blockchain keeps the contract small and keeps document contents out of the public ledger. Keeping files and application records off-chain allows ordinary tools, namely MongoDB and a REST API, to manage

organization, search, and sharing, which a contract alone would handle poorly.

Treating verification and access as separate workflows has practical value. A third party can be given a hash or a file to check without being given any right to download the original, and an owner can approve or deny a download without changing what the blockchain says about the document. The time-limited links and rotating codes narrow the window in which an approved download can be misused.

The PENDING state for registrations made without a connected wallet also reflects a practical concern: a user may be ready to upload a document before they are ready to sign a blockchain transaction. Allowing registration to be completed later, individually or in bulk, makes the workflow more forgiving. These observations describe design intent as documented; they are not conclusions drawn from user studies or measurements.

The layered organization of the backend into routes, controllers, services, models, and middleware also supports maintenance. Because blockchain interaction is confined to its own service, the contract and the network it targets can in principle be changed with limited effect on the rest of the application, which is consistent with the multi-chain direction listed as future work. Likewise, the separation between the frontend and the REST API means the interface can evolve without altering the verification logic.

Every design involves trade-offs, and this one is no exception. Writing to a public chain introduces a delay and a transaction cost that a conventional database write does not have, which is why the project targets a test network and why a PENDING state exists for registrations that cannot be completed immediately. A public network also makes registrations visible to outsiders, whereas a private or permissioned chain would restrict visibility but would reintroduce a smaller circle of parties that must be trusted. The documentation chooses an Ethereum-compatible network and a test deployment target, and the paper takes no position on whether a different arrangement would suit a particular institution better.

A further trade-off concerns ownership. Since the contract rejects duplicate hashes, the first account to register a fingerprint becomes its registrant. This is a clean rule, but it also means that a user who registers someone else's file first is recorded as its registrant. Policies for resolving such disputes, and ways of linking a registration to a verified institutional identity, belong to the institutional workflows listed as future work. They are best treated as open design questions and not as problems the current system claims to have solved.

XIII. LIMITATIONS AND FUTURE SCOPE

The project materials do not include measured performance results, detailed test execution logs, or findings from an independent audit. This paper is therefore a description of the documented system and its stated test coverage, and it should not be read as a controlled experimental evaluation.

The project lists several directions for future work: decentralized storage, stronger key management, institutional workflows, and advanced analytics. Further security enhancements are also named, including content inspection and malware scanning, as are centralized audit and retention features, multi-chain support, and automated CI/CD. These are possible future work and are not presented as completed features.

Some of these items can be understood in terms of the limits discussed above. Decentralized storage would reduce reliance on a single storage environment for the files themselves, and stronger key management would protect the credentials involved in signing and in accessing protected resources. Institutional workflows would allow organizations to model their own approval and issuing procedures, and content inspection and malware scanning would add a check on what is uploaded. Centralized audit and retention features would help organizations that must keep records for fixed periods, while multi-chain support would reduce dependence on a single network. Automated CI/CD would make it easier to run the existing tests consistently on every change, and publishing their logs would also address the gap in test evidence noted earlier.

Table V summarizes how far the available evidence extends for each aspect of the system discussed in this paper.

TABLE V
SCOPE OF AVAILABLE EVIDENCE

Aspect	Evidence in project materials
Design and modules	Documented in the blackbook
Security controls	Documented; no independent audit
Test coverage	Counts documented; logs not supplied
Performance	No benchmark data provided
Deployment availability	Configuration described; not tested

XIV. CONCLUSION

BlockVault pairs off-chain storage of documents and application records with on-chain registration of SHA-256 fingerprints. Its documented modules bring registration, verification, document management, controlled access, notifications, and administration together in one web application. An important feature of the workflow is that confirming a document's integrity and granting permission to access the original are handled separately. The blackbook records the security controls and the test coverage, while the supplied archive does not establish that the tests passed or how the system performs. The future work identified by the project includes decentralized storage, stronger key management,institutional workflows, advanced analytics, multi-chain support, and automated CI/CD.

REFERENCES

[1] BlockVault Project Team, "BlockVault: Blockchain-Based Document Verification and Management System," M.Sc. Computer Science project blackbook, Chikitsak Samuha's Patkar Varde College, Mumbai, India, 2026–2027.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[3] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Project Yellow Paper, 2014.

[4] National Institute of Standards and Technology, "Secure Hash Standard (SHS)," Federal Information Processing Standards Publication 180-4, Aug. 2015.

[5] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, Internet Engineering Task Force, May 2015.

[6] Ethereum Foundation, "Solidity Documentation." [Online].
Available: https://docs.soliditylang.org

[7] BlockVault web application. [Online]. Available: https://blockvault-smoky.vercel.app/