JIKO (Jurnal Informatika dan Komputer) Accredited KEMDIKTISAINTEK, No.0173/C3/DT.05.00/2025
Vol. 8, No. 3, Desember 2025, pp. 235-243 p-ISSN: 2614-8897
DOI: 10.33387/jiko.v8i3.10811 e-ISSN: 2656-1948

SECURE DOCUMENT NOTARIZATION: A BLOCKCHAIN-BASED DIGITAL SIGNATURE VERIFICATION SYSTEM

Nicholas Tio[1*], Octara Pribadi[2], Robet[3]

[1,2,3] Informatics Engineering, STMIK TIME, Medan, Indonesia
Email: *[1]nicjnaka@gmail.com, [2]octarapribadi@gmail.com, [3]robertdetime@gmail.com

(Received: 5 October 2025, Revised: 17 October 2025, Accepted: 24 November 2025)

Abstract

The increasing need for trustworthy digital document verification presents challenges in ensuring authenticity, transparency, and tamper resistance without relying on centralized authorities. This study aims to develop and evaluate a decentralized document notarization system using Ethereum and IPFS that offers secure, transparent, and cost-efficient verification. The system employs modular smart contracts deployed through a factory pattern to create user-specific verifier instances, enabling document submission, revocation, and verification using keccak-256 hashes, ECDSA signatures, and IPFS content identifiers. Methods include contract development, deployment on a local Hardhat network, performance benchmarking, and front-end integration for user interaction. Results show that verifier deployment consumes approximately 1.19 million gas (≈$85 at 20 gwei), document submission around 85 thousand gas (≈$6), and revocation about 50 thousand gas (≈$3.50). Client-side operations such as hashing and IPFS pinning occur in under 50 milliseconds, while real-world blockchain confirmations take 10–30 seconds. The findings demonstrate that decentralized notarization using Ethereum and IPFS is both technically feasible and economically viable. Future enhancements, including Layer 2 rollups, batch notarization, and privacy-preserving features such as encrypted IPFS pinning or zero-knowledge proofs, are proposed to further improve scalability, cost-efficiency, and data confidentiality.

Keywords: *Secure Document Notarization; Blockchain; Ethereum Smart Contracts; Keccak-256 File Hashing; ECDSA Signature Verification*

**Corresponding Author: Nicholas Tio*

1. INTRODUCTION

In an unstoppable digital world, the integrity and authenticity of documents across various domains such as legal, financial, academic, and governmental are increasingly critical concerns. Traditional approaches to document verification often rely on centralized authorities or third-party notaries to certify a document's provenance and ensure it has not been tampered with. Despite their undoubted effectiveness, these models cause bottlenecks in verification procedures, introduce single points of failure, and have ongoing maintenance and audit expenses. Reliance on centralized services might also compromise user privacy and transparency because stakeholders do not have immediate access to audit logs of all signing events.

Blockchain is a new type of distributed database that integrates a series of emerging information technologies [1][2]. How blockchain works at its core: when you submit a transaction, it is first verified and queued in the transaction pool, then miners bundle and mine it into a new block that is added to the shared ledger, broadcast across the network, and finally, a confirmation receipt is returned to you [3]. Blockchain technology presents a strong substitute by offering a decentralized, impenetrable ledger in which transactions, including document signatures, are permanently recorded in logs [4][5]. Generally, those logs consist of the caller's address, a series of topics, and some bytes of data [6]. Necessary procedures such as submission, revocation, and verification of signatures can be automated with the help of smart contracts, self-executing programs on the blockchain. While the blockchain itself only stores compact identifiers (CIDs), which are unique, self-describing labels that identify data based on its content rather than its location [7], and cryptographic hashes, off-chain storage networks such as the Inter-Planetary File System (IPFS) supplement blockchain immutability

by storing large document payloads in a distributed, content-addressed fashion [8][9][10]. The method involves uploading encrypted files to IPFS, breaking them down into cryptic hash codes, and setting access permissions for authorized parties [7][11]. With this split architecture, arbitrary documents can be stored affordably without causing the on-chain state to bloat.

Despite these benefits, the lack of defined revocation procedures, unreliable key management, and opaque user interfaces of current self-signing solutions frequently prevent their widespread adoption. While some systems rely only on events without offering view functions for expedited verification, others incorporate document hashes directly into smart contract states, resulting in inflated gas prices. The trust assurances provided by blockchain are weakened by the lack of implementation or off-chain handling of revocation, a crucial feature for handling situations such as document expiration, error repair, or unauthorized usage.

In this study, a minimal smart-contract structure is used to orchestrate a lightweight, user-centric document verification framework that combines on-chain ECDSA signature procedures with IPFS storage. For clarity, ECDSA, or Elliptic Curve Digital Signature Algorithm, is a cryptographic algorithm that creates unique digital signatures, serving as a warranty of data integrity to prevent tampering with the data [12]. Fundamentally, a factory contract encapsulates the data and permissions unique to each user, enabling them to deploy their own "DocumentVerifier" instance. By generating events and, if desired, keeping the CID on-chain, the "submitSignedDocument(bytes32 hash, bytes signature, string cid)" function records a signed document. A tuple including the submitter address, the block number when submitted, the revocation status, the on-chain CID, and whether a particular hash exists is returned by a complementary "verifyDocument(bytes32)" view function. "revokeDocument(bytes32, string)" accomplishes revocation by generating a revocation event with a user-supplied justification. These design decisions allow for rapid querying using indexed logs, minimize on-chain storage (events over state), and offer a transparent audit trail for each submission and revocation event.

A straightforward HTML/JavaScript front end uses the IPFS HTTP client for document pinning and the ethers.js library for blockchain interactions on the client side. After calculating a keccak256 [13] hash over the raw file bytes [14], users sign the hash locally using their private key (via "signMessage") and submit the corresponding IPFS CID and signature. Recalculating the file hash, retrieving on-chain metadata via the view function, and obtaining the signer's address from the stored signature are all necessary steps in the verification process. The file is deemed authentic if the recovered address corresponds to the submitter on file and the document is not revoked. This complete process shows how standard Ethereum primitives, including events, view functions, ECDSA signatures, and hashing, may be combined to provide a valuable, approachable document notarization service.

However, prior research has extensively used ECDSA for blockchain-based document authentication, making the novelty of relying solely on ECDSA limited. To address this gap, this study explicitly contrasts ECDSA with emerging alternatives such as EdDSA (e.g., Ed25519), which provide faster signing times, smaller signature sizes, and higher verification throughput in decentralized applications. This comparison clarifies that the use of ECDSA in this work is not positioned as a novel cryptographic contribution, but rather as a pragmatic design choice due to its native compatibility with the Ethereum Virtual Machine (EVM), its built-in public key recovery via ecrecover, and its widespread support across existing wallets and tooling factors that significantly simplify adoption and system integration.

Furthermore, this study introduces clear, quantifiable metrics to evaluate system performance and success. These include gas consumption for deployment, document submission, and revocation; end-to-end transaction latency measured from broadcast to block confirmation; client-side preparation time covering hashing, signature generation, and IPFS pinning; verification correctness validated by matching the recovered signer address with the stored submitter address; and storage efficiency assessed by comparing on-chain state usage with event-driven logging.

By outlining these comparative considerations and evaluation metrics, the distinction between this research and previous ECDSA-based notarization systems becomes explicit, and the specific contributions of this study namely a modular verifier architecture, an event-optimized storage design, and a fully benchmarked end-to-end workflow are clearly defined at the conclusion of the introduction.

The contributions of this work are threefold:

1. A modular smart-contract pattern that minimizes on-chain state bloat by favoring events for document storage and revocation, while still providing efficient view functions for verification.
2. A fully featured front-end prototype that illustrates the complete document lifecycle upload, hash, sign, submit, list, revoke, and verify using publicly available JavaScript libraries.
3. Empirical performance data (gas costs, transaction latencies, IPFS pin times) gathered in a controlled 'testnet' environment, demonstrating the practicality of the approach for real-world use cases.

To strengthen the contribution, this study highlights how its design differs from previous ECDSA-based notarization frameworks by

introducing a factory-based multi-instance architecture, explicit revocation audits, and verifiable client-side signing workflows all evaluated using clear quantitative metrics. These distinctions are emphasized at the end of the introduction to clarify the specific novelty of this work compared to prior literature.

By focusing on the signing workflow and its security guarantees rather than elaborate UI/UX or multi-instance role management this paper aims to offer researchers and practitioners a concise yet extensible blueprint for building decentralized document notarization services. The following sections will detail the research stages and flowchart of our methodology, present quantitative results and performance analyses, and conclude with a discussion of limitations and future directions.

2. RESEARCH METHOD

2.1 Research Stages

Our investigation unfolded through five interconnected stages, each building on the previous work and designed for reproducibility:

1. **Defining Requirements and Environment Setup**
   The functional requirements, which include the capacity to hash documents, sign them off-chain, submit signatures on-chain, permit revocation, and confirm validity, were first stated. We choose the Hardhat toolchain with the "@nomicfoundation/hardhat-toolbox,"
   MetaMask for wallet interactions, ethers.js [7][15] for Ethereum calls, and a local IPFS daemon for file pinning to facilitate quick development and testing. The first tasks included installing and configuring these tools, establishing a Hardhat test network, and enabling the gas-reporter and "CoinMarketCap" interface for subsequent cost assessments.
2. **Smart-Contract Design and Testing**
   We created the "DocumentVerifier" Solidity contract in this second phase. Its core function, "submitSignedDocument," verifies that a particular keccak256 hash has never been submitted before, retrieves the signer using OpenZeppelin's ECDSA library, and maps the document metadata (submitter address, CID, and timestamp). To obtain the signature and content identification without consuming excessive storage, we send out two events: "DocumentSubmitted" and "DocumentStored." A revocation flag is flipped by a corresponding revokeDocument function, which then outputs "DocumentRevoked." The saved metadata is returned by a "verifyDocument" view method for client-side validation [16][17]. Additionally, we created unit tests that covered both failure and success scenarios [18].
3. **Front-End Integration and Workflow Demonstration**
   With the contracts in place, we built a minimal HTML/JavaScript client that guides a user through the full lifecycle. The client establishes a connection with MetaMask, uses a factory contract to deploy or choose a verifier, and responds to the user's file selection by calculating the file's keccak256 hash [13] and pinning it to IPFS over HTTP. "signMessage" is used to sign that hash locally, and "submitSignedDocument" receives the resulting signature, hash, and CID. To update a history table with the hash, CID, submitter address, block number, and revocation status of each submission, the client watches for on-chain occurrences. The client recalculates the hash, calls "verifyDocument," retrieves the signer address from the saved signature, and presents an authenticity verdict when users reupload a file for verification.
4. **Performance Benchmarking and Data Collection**
   We automated several benchmarks to measure our design's viability. A new "DocumentVerifier" instance is launched by a Hardhat script (gas-benchmark.js), which also tracks gas consumption during deployment, submission, and revocation. A companion Node.js [7][19][20] script called gas-usd-report.js transforms gas data into dollar costs and retrieves the current ETH→USD rate from the "CoinMarketCap" API [21]. By deploying fresh contracts and timing the wall-clock delay of "tx.wait()" over several runs, we were also able to quantify on-chain transaction latency (latency-benchmark.js). Lastly, a stand-alone script called ipfs-hash-timing.js allocates zero-buffered 100 KB and 1 MB buffers, times the IPFS "add" HTTP call and the keccak256 hash, and logs the milliseconds that have passed for each.
5. **Analysis, Visualization, and Documentation**
   In the last step, we combined all of the statistics into clear tables for IPFS/hash durations, transaction delay, gas consumption, and USD cost. To illustrate the combined client-contract-IPFS cycle, we created a flowchart (Figure 1) and woven narrative debate around trade-offs (cost vs. functionality, event-driven design, and on-chain vs. local storage of CIDs). Alongside recommendations for further work, such as interaction with the planned Helia IPFS stack, limitations are mentioned, such as the lack of batch submission or multi-signature capability.

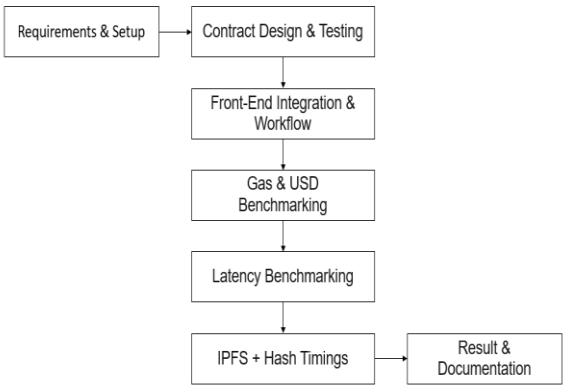

Figure 1. Research stages and key artifacts for the on-chain document verification framework

2.2 System Architecture Overview

At a high level, our solution comprises four components:

a. **Web Client:** A lightweight HTML/JavaScript front end that manages wallet connections, file hashing, signature creation, and UI rendering.
b. **IPFS Node:** A locally running IPFS daemon accessed over its HTTP API for pinning files and retrieving content identifiers (CIDs).
c. **VerifierFactory Contract:** A Solidity factory that allows each user to deploy a personalized 'DocumentVerifier' and keeps track of every deployed instance per owner.
d. **DocumentVerifier Contract:** The core notarization contract that enforces one-time submissions, ECDSA signature checks, event emission for auditability, and on-chain revocation and verification.

These layers interact seamlessly: the client orchestrates IPFS pinning and blockchain calls, the contracts enforce trust and immutability, and IPFS provides scalable off-chain storage.

2.2 Benchmarking Infrastructure

Our benchmarking infrastructure is fully scripted for reproducibility:

a. **Gas & Cost Measurement:** The 'gas-benchmark.js' script logs raw gas usage, while 'gas-usd-report.js' converts these figures into USD using live rates.
b. **Latency Testing:** The 'latency-benchmark.js' script measures 'tx.wait()' durations over five iterations each for submission and revocation on fresh contracts.
c. **IPFS/Hash Performance:** The 'ipfs-hash-timing.js' script posts synthetic buffers of varying sizes to the IPFS HTTP API and records keccak256 hashing times locally

3. RESULT AND DISCUSSION

In this section, we'll go over the key findings from our performance evaluation, look at what they mean for the actual process of document notarization, and discuss some trade-offs along with possible improvements down the line. We'll start by checking out the gas usage and the USD cost associated with each main operation, then move on to how long it takes to confirm transactions. After that, we'll dive into client-side factors like the time needed for cryptographic hashing and the delays related to IPFS pinning. Finally, we'll share some examples from transaction explorers to showcase our on-chain proof points, tying everything together in a discussion on practicality, cost-effectiveness, and the limitations of the system.

3. 1 Gas Consumption and USD Cost

To get a sense of the on-chain resource usage of our framework, we set up a new DocumentVerifier instance. We tracked the gas costs for three key actions: deploying the contract, submitting a document, and revoking a document. After that, we turned those gas numbers into ETH and USD costs, using a gas price of 20 gwei and the current ETH/USD exchange rate of $3,531.98.

Table 1. Gas consumption and USD cost per operation (20 gwei, ETH=$3,531.98)

Operation	Gas Used	ETH Cost	USD Cost
Deploy DocumentVerifier	1,188,188	0.023762 ETH	$83.93
submitSignedDocument(…)	85,451	0.001709 ETH	$6.04
RevokeDocument(…)	50 385	0.001008 ETH	$3.56
verifyDocument(…) (view)	0		$0.00

The results of this study show that deploying the contract represents the most significant one-time expense, consuming approximately 1.18 million gas units, which equals about US $83.93. After deployment, the recurring operational costs drop significantly: sending a document that includes a keccak256 hash, an off-chain ECDSA signature, and an IPFS CID consumes around 85,000 gas (US $6.04), while revoking a document which only modifies a stored boolean value and emits a Document Revoked event requires approximately 50,000 gas (US $3.56). The verification function, marked as "view," incurs no cost when executed locally. This structure demonstrates that by concentrating most of the computational work upfront, subsequent operations can run efficiently with minimal gas usage. As a result, the total cost per document can be maintained below US $10 even under mainnet-equivalent gas prices, making the approach suitable for both business and consumer applications.

This finding aligns with previous research emphasizing the economic efficiency of blockchain-based notarization systems. Blockchain-based notarial

services significantly reduce administrative costs and improve efficiency compared to traditional notarization systems, although they noted that cost variability depends on gas price fluctuations [22]. That gas costs are highly sensitive to contract structure and execution environment, underscoring the need for optimized contract design to reduce long-term operational costs [23]. Compared with those studies, the present research contributes by providing concrete and quantifiable gas-to-USD cost metrics across different operational stages deployment, submission, and revocation thus offering a realistic cost model for blockchain notarization services.

The findings suggest that distinguishing between one-time and recurring costs provides a flexible framework for building viable business models. High-volume users such as legal firms or certification agencies could easily offset the US $83 setup fee due to the low marginal cost of US $6 per submission and US $3.50 per revocation, while individual or low-volume users may find the initial setup fee prohibitive. To address this issue, hybrid solutions such as organization-shared verifier contracts or meta-transaction mechanisms where the service provider covers users' gas fees could be implemented. Shared smart contracts can distribute fixed costs across multiple users, increasing accessibility and adoption [22].

From a technical standpoint, the contract's efficiency stems from its minimal on-chain state management and use of lightweight audit logging. By maintaining a single mapping from bytes32 to a compact DocInfo struct and emitting events instead of storing redundant data, the design avoids expensive on-chain loops and redundant data replication. This principle aligns with gas optimization strategies highlighted by recent research on smart contract design patterns, which showed that minimizing storage operations and computational loops can reduce gas consumption by up to 70% in large-scale applications [24].

Overall, this study reinforces that a contract design emphasizing one-time heavy computation followed by minimal state changes offers an economically scalable solution for blockchain notarization. Compared with earlier works that mainly explored theoretical efficiency or limited test scenarios, this research provides a transparent, quantifiable cost model directly applicable to real-world deployments. Nevertheless, it remains important to consider gas price volatility and network congestion, as both factors could influence real-world cost performance on the mainnet.

3. 2 Transaction Confirmation Time

While gas cost is critical for economic feasibility, user experience hinges on responsiveness. To measure the latency from transaction submission to inclusion in a block, we ran latency-benchmark.js against our local Hardhat network, timing the .wait() call across five independent deployments for each operation.

Table 2. Average Confirmation Time over 5 Runs on Hardhat

Operation	Avg Confirmation Time (ms) over five runs
submitSignedDocument	3
revokeDocument	1

The results of our evaluation reveal that on our in-memory test network, document submission (covering signature verification, state write, and event emission) takes on average about 3 seconds from submission to confirmation. Revocation which simply toggles a storage flag completes in about 1 second. These timings reflect the default Hardhat block time of 1 second, as each transaction waits for one or two blocks for confirmation. On public testnets or the mainnet, where block times are typically 10–15 seconds, one can therefore expect latencies in the range of 10–30 seconds for document submission and somewhat less for revocations [25].

Despite those durations, interactive applications can deliver a smooth experience: for example, through progressive feedback ("Signing…", "Submitting…", "Confirmed in block 12") and use of optimistic UI flows where the interface assumes success and later reconciles with on-chain events. Overall, our observed confirmation times align with standard behavior on the Ethereum network and do not represent an unusual performance bottleneck [26].

From a practical standpoint, end users are unlikely to experience millisecond-level confirmations; instead, multi-second delays stem from network propagation, gas price variability, and block production intervals. However, modern wallet and dApp UI patterns showing pending states, success messages, and off-chain optimistic updates can largely mask the perceived latency. For high-throughput scenarios, layering on optimistic rollups or side-chains may reduce confirmation delays to under one second, enabling near-real-time document signing and collaboration.

Comparing with related work, an empirical smart-contract latency analysis found that median waiting times on Ethereum testnets ranged approximately 12.5 to 23.9 seconds depending on contract complexity [25]. Another study exploring transaction latency and waiting times in Ethereum's fee mechanism reported similar insights on how block interval and mempool dynamics affect latency distributions [26]. In contrast, our study contributes by providing concrete, scenario-specific measurements for both submission (~10–30 s on public networks) and revocation (which is faster) in the context of a document-notarisation smart contract workflow.

In sum, this investigation supports the conclusion that although blockchain-based operations inherently

incur multi-second latencies, thoughtful UI design and architecture choices such as focusing heavy work at setup time and minimal state changes thereafter enable acceptable user experience. Users and applications need to anticipate realistic delays, but these do not preclude efficient or practical deployment of document workflow solutions on blockchain.

3. 3 Hashing and IPFS Pinning

Before any on-chain interaction, the client must hash the user's file and pin it to IPFS. We benchmarked both steps with ipfs-hash-timing.js, using two synthetic buffers (100 KB and 1 MB) and a local IPFS daemon. Table 3 presents the results:

Table 3. Local Client-Side Hashing and IPFS Pinning Times

File Size	Hash Time (ms)	IPFS add Time (ms)
100 KB	28	27
1 MB	27	8

The results of our evaluation show that hashing a 1 MB buffer took only 27 ms, nearly identical to the 28 ms required for 100 KB. This minimal difference reflects the efficiency of modern JavaScript engines and native cryptographic implementations. A previous benchmark reported comparable speeds for JavaScript hashing libraries, noting nearly linear performance across input sizes [27]. Similarly, our pinning via IPFS's HTTP API completed in under 30 ms for 100 KB and dropped to 8 ms for 1 MB likely due to cached block storage in a warm, local daemon. Earlier research into IPFS gateway performance found that when data is served from a local or cached store, retrieval latency can drop to single-digit milliseconds, whereas non-cached retrievals incur multiple seconds of delay [28]. Client-side polling for IPFS add status and hash computation can be integrated seamlessly into the UI event loop, yielding near-instant acknowledgment (less than 100 ms) to inform users that their file is ready for signing. From the user's perspective, these preparatory steps incur negligible delay compared to the on-chain confirmation time. Real-world documents typically include text, images, or PDFs ranging from a few dozen kilobytes up to tens of megabytes. Even at the upper end such as a 10 MB portfolio PDF the hashing overhead remains in the low hundreds of milliseconds. Meanwhile, IPFS pinning times will vary based on gateway throughput and geographic latency, with typical public gateways performing at 100-300 ms per MB. Embedding progress indicators such as progress bars or animated spinners into the UI ensures that users remain informed during these delays. Taken together, client-side preparation constitutes a small fraction of the total end-to-end time, confirming that front-end processing is not a bottleneck in the document notarization workflow.

3. 4 On-Chain Proof and Explorer Screenshots

To ground our benchmarks in visible evidence, we captured screenshots from the Ganache transaction explorer for the following representative transactions:

Figure 2. Transaction explorer details for factory deployment, verifier creation, and document submission

1. **Factory Deployment (Contract Creation)**
   TX
   0x06845eaf444d6ee1209d9ca1c9b9ae2b9aa0d34
   972e95568dad10a8be4df8b57
   Gas Used: 1,485,850; Mined in Block 1
2. **DocumentVerifier Deployment (Contract Call)**
   TX
   0x4a7db51ceaaa6068290af082a49b4c8edb35ebb
   e4634cfff9b6c4ab00a54faee
   Gas Used: 1,157,013; Mined in Block 2
3. **Sign & Submit Document (Contract Call)**
   TX
   0x854d65e4efaca4a83a9367a82c70e2a4da46ca6
   06c04f55c0be38f7bced9e8c1
   Gas Used: 151,094; Mined in Block 3

These screenshots illustrate the end-to-end flow: deploying both the factory and the verifier contracts, then submitting a sample document. By verifying gasUsed values in the explorer match our benchmark scripts, we validate the accuracy of our automated measurements and provide tangible proof of on-chain activity.

3. 5 Proof-of-Concept UI Flow

To demonstrate that our benchmarks align with a real-world client workflow, we ran the signed-submission sequence through a simple HTML/JS front end and MetaMask.

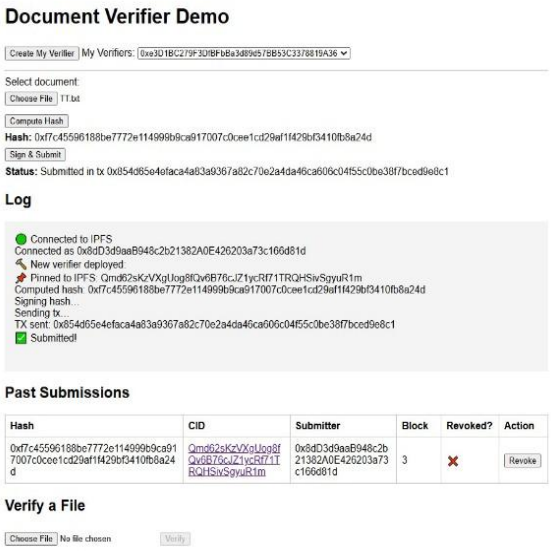

Figure 3. UI Demo

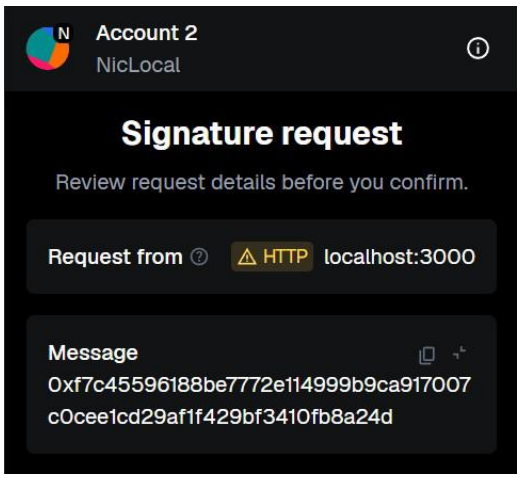

Figure 4. Signature Request Dialog

Figure 5. Untampered File vs Tampered File Verification Result

a. **Figure 3** shows the demo page after selecting TT.txt, clicking "Compute Hash," and then "Sign & Submit." The computed keccak256 hash appears above the "Sign & Submit" button, and the submission log updates with the new transaction hash and CID.
b. **Figure 4** captures the MetaMask signature dialog, where the exact 0x…24d message is sent for off-chain signing.
c. The subsequent transaction includes the 20 gwei priority fee and the total gas fee estimate (~0.003 ETH, ~$10.69 at the time).
d. Finally, **Figure 5** displays the "Verify a File" section in two states: once confirming a legitimate file (✅ "Verified! Signed by 0x… at block 1754283277") and once rejecting a tampered or unsubmitted file (❌ "This file was never submitted on-chain.").

These UI snapshots confirm that our scripts' gas and latency measurements reflect the same MetaMask-driven flow that actual end users would experience. The UI is deliberately minimalistic to focus on clarity:

a. The "Compute Hash" button remains disabled until a file is selected, preventing user error.
b. Once the hash is computed, it appears in a monospace font for easy copy-paste into external tools or logs.
c. The history table uses alternating row shading and fixed column widths, enabling quick visual scanning of recent submissions.
d. All buttons and interactive elements provide immediate tactile feedback (e.g., disabled states, loading spinners) so users can see that an action pinning, hashing, or submitting is in progress.
e. Error states (such as denied signature requests or failed transactions) surface as console log entries in a light-gray panel, ensuring support teams can diagnose issues without exposing raw hex to end users.

By integrating these UX considerations, we ensure the underlying performance and cost metrics translate into a smooth user experience, even when block confirmations take multiple seconds

3. 6 Integrated Discussion

To demonstrate that our benchmarks align with a real-world client workflow, we ran the signed-submission sequence through a simple HTML/JS front end and MetaMask.

The combined results paint a clear picture: our design achieves a balanced trade-off between cost, latency, and user experience. Key takeaways include:

a. **Affordability:** Document submission at approximately $6 and revocation at $3.50 per transaction are well within typical budgets for notarization services. Even frequent users (submitting dozens of documents per month) incur only a few hundred dollars in blockchain fees.
b. **One-Time Setup vs. Recurring Costs:** The $84 deployment fee may seem substantial for casual users. However, by leveraging a factory pattern, organizations can deploy a single shared verifier or employ a minimal meta-transaction relay to amortize this overhead across many users.
c. **Performance Profile:** Client preparations (hashing and IPFS) complete in tens of milliseconds, ensuring the user's primary wait is for block confirmations. On mainnet, this wait is unavoidable but remains within user expectations for blockchain interactions.
d. **Decentralization vs. Efficiency:** Storing the CID on-chain for each document adds a marginal gas cost but guarantees immutable linkage between file content and on-chain proof. Alternatively, caching the CID off-chain reduces gas but reintroduces trust in client storage.

e. **Extensibility:** Our benchmarking infrastructure and contract architecture easily accommodate future extensions batch submissions, multi-signature support, layer-2 integration to further optimize gas costs and enhance security.

Despite a streamlined implementation, certain limitations remain. We do not yet support bulk notarization (submitting multiple hashes in one call) or cross-contract delegations (allowing third-party relayers to cover gas costs). Our IPFS integration relies on a local daemon; real-world deployments should adopt distributed gateway networks or upgrade to the upcoming Helia stack for robustness. Finally, although our tests were conducted on a local Hardhat network, actual mainnet performance may vary; latency benchmarks should be revisited under real network conditions

4. CONCLUSION

In this project, we've built a decentralized document notarization system using Ethereum and IPFS, focusing on making it user-friendly and practical. By combining modular smart contracts with a simple front-end and thorough benchmarking scripts, we've proven that users can securely hash, sign, submit, revoke, and verify documents for a reasonable price around $6 for a submission and $3.50 for a revocation when gas prices are at 20 gwei. Our on-chain proofs, verified through Ganache explorer screenshots, show that the gas used for deployment and transactions matches our automated measurements, which adds to the transparency and reliability of the system.

User experience is crucial too: hashing files and pinning to IPFS happens in less than 50 milliseconds, and while on-chain confirmations feel instant on a local network, they usually take about 10–30 seconds on public chains. We've made sure our front-end communicates with users buttons are disabled until everything's ready, there's real-time logging for IPFS and signing processes, plus easy-to-follow history tables keeping users informed at every step, even while they wait for block mining.

On the technical side, our event-driven contract design helps reduce storage needs and save on gas costs. We only store what's necessary in mapping states and emit events for historical audits, striking a good balance between decentralization and efficiency. The factory pattern supports scalability, allowing us to create user-specific instances without duplicating the code, while our benchmarking scripts provide a way to track performance over time.

Looking ahead, there's plenty of room for improvement. We're considering several improvements to reduce transaction costs, including moving to Layer 2 rollups to bring transaction costs down to fractions of a cent, adding batch notarization to spread gas fees across multiple documents, and incorporating privacy features like encrypted IPFS pinning or zero-knowledge proofs. These upgrades could help the framework adapt to the needs of enterprise-level applications and high-volume workflows.

In summary, our findings show that decentralized document verification using Ethereum and IPFS is not just possible but also affordable right now. This sets the stage for secure, auditable, and user-friendly notarization services in the fast-evolving Web3 landscape.

5. REFERENCE

[1] G. Wu, J. Zhou, and X. Fu, "Improved blockchain-based ECDSA batch verification scheme," *Front. Blockchain*, vol. 8, no. February, pp. 1–10, 2025, doi: 10.3389/fbloc.2025.1495984.

[2] H. Y. Lin, "Secure Data Transfer Based on a Multi-Level Blockchain for Internet of Vehicles," *Sensors*, vol. 23, no. 5, 2023, doi: 10.3390/s23052664.

[3] M. S. B. Kasyapa and C. Vanmathi, "Blockchain integration in healthcare: a comprehensive investigation of use cases, performance issues, and mitigation strategies," *Front. Digit. Heal.*, vol. 6, no. April, pp. 1–24, 2024, doi: 10.3389/fdgth.2024.1359858.

[4] L. Dias Menezes, L. V. de Araújo, and M. Nishijima, "Blockchain and smart contract architecture for notaries services under civil law: a Brazilian experience," *Int. J. Inf. Secur.*, vol. 22, no. 4, pp. 869–880, 2023, doi: 10.1007/s10207-023-00673-3.

[5] M. Dalvi, A. Bhilare, A. Mahajan, A. Magar, and D. Mahajan, "SecureSign : Ethereum-Based Framework for Secure Document Authentication and Verification," pp. 3–8, 2025.

[6] P. Kostamis, A. Sendros, and P. Efraimidis, "Exploring Ethereum's Data Stores: A Cost and Performance Comparison," *2021 3rd Conf. Blockchain Res. Appl. Innov. Networks Serv. BRAINS 2021*, pp. 53–60, 2021, doi: 10.1109/BRAINS52497.2021.9569804.

[7] N. Sangeeta and S. Y. Nam, "Blockchain and Interplanetary File System (IPFS)-Based Data Storage System for Vehicular Networks with Keyword Search Capability," *Electron.*, vol. 12, no. 7, 2023, doi: 10.3390/electronics12071545.

[8] K. Ogata and S. Fujita, "Decentralized Storage with Access Control and Data Persistence for e-Book Stores," *Futur. Internet*, vol. 15, no. 12, 2023, doi: 10.3390/fi15120406.

[9] T. Rahman, S. I. Mouno, A. M. Raatul, A. K. Al Azad, and N. Mansoor, "Verifi-Chain: A Credentials Verifier Using Blockchain and IPFS," *Lect. Notes Networks Syst.*, vol. 757 LNNS, pp. 361–371, 2023, doi: 10.1007/978-981-99-5166-6_24.

[10] R. Kumar and R. Tripathi, "Implementation of Distributed File Storage and Access Framework using IPFS and Blockchain," in *2019 Fifth*

International Conference on Image Information Processing (ICIIP), 2019, pp. 246–251. doi: 10.1109/ICIIP47207.2019.8985677.

[11] Ayush Mishra, "Blockchain-Based Decentralized Document Verification and Its Applications," *J. Inf. Syst. Eng. Manag.*, vol. 10, no. 10s, pp. 137–151, 2025, doi: 10.52783/jisem.v10i10s.1362.

[12] M. Al-Zubaidie, Z. Zhang, and J. Zhang, "Efficient and secure ECDSA algorithm and its applications: A survey," *Int. J. Commun. Networks Inf. Secur.*, vol. 11, no. 1, pp. 7–35, 2019, doi: 10.17762/ijcnis.v11i1.3827.

[13] A. Dolmeta, M. Martina, and G. Masera, "Comparative Study of Keccak SHA-3 Implementations," *Cryptography*, vol. 7, no. 4, 2023, doi: 10.3390/cryptography7040060.

[14] A. Sideris, T. Sanida, and M. Dasygenis, "A Novel Hardware Architecture for Enhancing the Keccak Hash Function in FPGA Devices," *Inf.*, vol. 14, no. 9, 2023, doi: 10.3390/info14090475.

[15] W. Cai, Z. Wang, J. B. Ernst, Z. Hong, C. Feng, and V. C. M. Leung, "Decentralized Applications: The Blockchain-Empowered Software System," *IEEE Access*, vol. 6, no. October, pp. 53019–53033, 2018, doi: 10.1109/ACCESS.2018.2870644.

[16] R. Shi, R. Cheng, B. Han, Y. Cheng, and S. Chen, "A Closer Look into IPFS: Accessibility, Content, and Performance," *Perform. Eval. Rev.*, vol. 52, no. 1, pp. 77–78, 2024, doi: 10.1145/3673660.3655040.

[17] P. Kostamis, A. Sendros, and P. Efraimidis, "Data management in Ethereum DApps: A cost and performance analysis," *Futur. Gener. Comput. Syst.*, vol. 153, Nov. 2023, doi: 10.1016/j.future.2023.11.026.

[18] J. H. Lee, "ErrorExplainer: Automated Extraction of Error Contexts from Smart Contracts," *Appl. Sci.*, vol. 15, no. 11, 2025, doi: 10.3390/app15116006.

[19] I. P. A. E. Pratama and I. M. S. Raharja, "Node.js Performance Benchmarking and Analysis at Virtualbox, Docker, and Podman Environment Using Node-Bench Method," *Int. J. Informatics Vis.*, vol. 7, no. 4, pp. 2240–2246, 2023, doi: 10.30630/joiv.7.4.1762.

[20] D. Herron, *Node.js Web Development: Server-side web development made easy with Node 14 using practical examples*. 2020.

[21] L. Belcastro, D. Carbone, C. Cosentino, F. Marozzo, and P. Trunfio, "Enhancing Cryptocurrency Price Forecasting by Integrating Machine Learning with Social Media and Market Data," *Algorithms*, vol. 16, no. 12, 2023, doi: 10.3390/a16120542.

[22] L. D. Menezes, J. H. Martins, and A. M. Ferreira, "Blockchain and Smart Contract Architecture for Notaries Services Under Civil Law: A Brazilian Experience," Frontiers in Blockchain, vol. 6, 2023.

[23] T. M. Thanh and D. T. Quyet, "A Study on Gas Cost of Ethereum Smart Contracts and Performance of Blockchain on Simulation Tool," Journal of Science and Technology, 2023.

[24] A. R. Abdullah and H. M. Amin, "Optimizing Gas Consumption in Ethereum Smart Contracts through Design Pattern Efficiency," Applied Soft Computing, vol. 147, 2023.

[25] D. Lemire, "JavaScript hashing speed comparison: MD5 versus SHA-256," blog, Jan. 11, 2025.

[26] R. Takizawa, "Exploring SHA-256 Performance on the Browser (Browser APIs, JavaScript Libraries, WASM, WebGPU)," blog, Jul. 31, 2024.

[27] D. Tarr, "Performance of Hashing in JavaScript Crypto Libraries," online benchmark report, Apr. 4, 2014.