Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.085633

ARTICLE

SecuAudit: Integrity-Preserving Metadata Compliance Auditing for Secure Data Circulation in MCP-Enabled AI Agents

Yufa Shi[1,#], **Jiaxing Hu**[2,#], **Lipeng Wang**[1,3,*], **Rui Ma**[1,3,*], **Mengyao Wang**[1] and **Zhijuan Jia**[1,3]

[1]School of Computer and Artificial Intelligence, Zhengzhou University, 100 Science Avenue, Zhengzhou, China
[2]Faculty of Engineering and Information Sciences, University of Wollongong, Wollongong, Australia
[3]School of Information Science and Technology, Zhengzhou Normal University, Zhengzhou, China

*Corresponding Authors: Lipeng Wang. Email: wlpscu@zznu.edu.cn; Rui Ma. Email: marui@zznu.edu.cn

[#]These authors contributed equally to this work

Received: 21 May 2026; Accepted: 22 July 2026; Published: 13 August 2026

ABSTRACT: AI agents frequently access external files, databases, and application programming interfaces (APIs) through the Model Context Protocol (MCP). However, these external resources typically lie outside the security boundary of the agent. During data circulation, attackers can not only tamper with the external data but also manipulate critical metadata, such as access permissions, validity periods, and authorization scopes. Even when the underlying data remains intact, such attacks can cause proxies to ingest expired or policy-violating resources, leading to severe privacy breaches and risks of unauthorized execution. To address these challenges, we propose SecuAudit, a privacy-enhancing decentralized data auditing scheme tailored for the MCP architecture. Leveraging Shamir's Secret Sharing mechanism, key management is securely decentralized across multiple servers, effectively mitigating sub-threshold collusion attacks. Furthermore, by introducing a supervised hash mechanism, dynamic access policies and metadata constraints are cryptographically bound to the user's private keys and file tags. Finally, based on a carefully designed randomized challenge-response protocol, third-party auditor (TPA) can efficiently verify both data integrity and metadata compliance without accessing the sensitive raw data. Security analysis demonstrates that SecuAudit can effectively resist data forgery, metadata tampering, and sub-threshold collusion attacks under the defined threat model. Experimental results obtained from our implementation under the evaluated configurations show that: (i) at (t = 30, m = 59), SecuAudit reduces Keygen latency by 54.6% relative to our implementation of the Pedersen-based Threshold Label-Aggregating Remote Data Auditing scheme (Ped-TLARDA), while maintaining comparable online auditing performance; (ii) metadata binding introduces an average additional Signblock overhead of 2.5%; and (iii) the local EVM evaluation requires approximately 770 bytes of on-chain storage per file and 499,021 gas for the evaluated contract-interaction lifecycle; (iv) the theoretical cumulative detection probability for 1% data loss exceeds 99.9% after 23 independent audits, while all six implemented attack cases were detected in the controlled evaluation. These results indicate the feasibility of SecuAudit under the evaluated settings. In conclusion, SecuAudit establishes a feasible framework for secure data circulation under the evaluated deployment assumptions.

KEYWORDS: Privacy enhancing computing; secure data circulation; model context protocol; metadata compliance; decentralized data auditing

1 Introduction

In privacy-sensitive secure data circulation scenarios, data often needs to be shared, accessed, and reused among different entities. At the same time, the exposure of sensitive information should be reduced

as much as possible, and the use of data should comply with corresponding access policies and regulatory requirements. This issue becomes more prominent in AI agent systems. During task execution, AI agents heavily rely on the Model Context Protocol (MCP) as a unified interface to dynamically invoke external resources, such as files, databases, and cloud services, residing outside their local security boundaries in a standardized manner. However, it also brings new security and privacy risks during cross-domain data circulation [1–3]. Since these external data sources lie outside the agent's direct security perimeter, the data stored, transmitted, or retrieved through them may suffer from device failures, malicious attacks, and service anomalies, thereby compromising data integrity and metadata compliance. In addition, the security risks in MCP scenarios are not limited to bit-level tampering of the file content itself. Attackers may also tamper with metadata and policy attributes, such as access permissions, file types, validity periods, and authorization scopes. Such attacks cause the agent to misinterpret the data's meaning and usage boundaries while leaving the original data content unchanged. For example, the attacker can use the external server to tamper with the access semantic rule from "Top Secret" to "Public Reference" without altering the original data. In this scenario, traditional integrity audit schemes may still determine that the data is available, but after parsing the compromised metadata and policy attributes, the AI agent may mistakenly use or export restricted information as public information, thereby leading to privacy breaches.

Recent research indicates that data auditing in the fields of AI still faces threats to data integrity [4–8], and MCP workflows may introduce additional risks such as tool abuse, privilege escalation, and data leaks during cross-domain resource scheduling [9]. However, most existing data auditing schemes [10–12] are designed for traditional static cloud storage environments, primarily focusing on whether outsourced data has suffered physical damage or content tampering, making them inappropriate for the frequent dynamic interactions and changing metadata and access control constraints inherent in MCP scenarios. More importantly, these schemes typically lack the ability to dynamically update and effectively validate access policies, registered compliance semantic rules, and metadata constraints, making it difficult to promptly detect context pollution issues caused by the tampering of metadata and policy attributes. Furthermore, existing schemes generally rely on centralized trusted nodes for key management, which not only introduces a single point of failure but also creates a centralized trust assumption. In MCP environments, AI agents require dynamic authorization based on different users, resource types, and task contexts; traditional centralized key management mechanisms struggle to efficiently support such complex permission changes. Consequently, there remains a significant lack of comprehensive data auditing schemes capable of simultaneously supporting decentralized key management, dynamic permission control, data integrity verification, and metadata compliance checks.

To address the above challenges, we propose SecuAudit, a decentralized security data auditing scheme designed for MCP environments. In contrast to traditional cloud storage auditing schemes designed for static outsourced data, SecuAudit is formulated for MCP scenarios involving dynamic external resource invocation. As illustrated in Fig. 1, SecuAudit seamlessly integrates into the standard MCP resource invocation workflow through a cohesive lifecycle of decentralized key management and challenge-response auditing. During the initial resource registration phase, the data owner utilizes a threshold-based distributed key infrastructure to generate rule-bound authenticators. These authenticators cryptographically bind the raw file blocks to their corresponding access policies and metadata constraints (such as validity periods and authorization scopes), subsequently anchoring the verifiable file tags to an immutable smart contract. Later, when an AI agent (MCP Client) attempts to access these external resources—such as databases or API tools—via the MCP Server, the cross-domain interaction triggers the auditing mechanism. Rather than requiring the full dataset to be downloaded for metadata compliance checking, a Third-Party Auditor (TPA) issues a randomized cryptographic challenge to the MCP Connector, which forwards the challenge to the

corresponding External Server. The External Server computes an aggregated proof based on the stored data blocks and authenticators, and the MCP Connector relays the resulting proof to the TPA. By verifying this proof against the on-chain smart contract state, the TPA simultaneously confirms bit-level data integrity and strict metadata compliance. This strictly enforced workflow ensures that the AI agent only ingests policy-compliant and untampered context, thereby significantly reducing the risks of context pollution and unauthorized execution caused by metadata manipulation without exposing the underlying raw data.

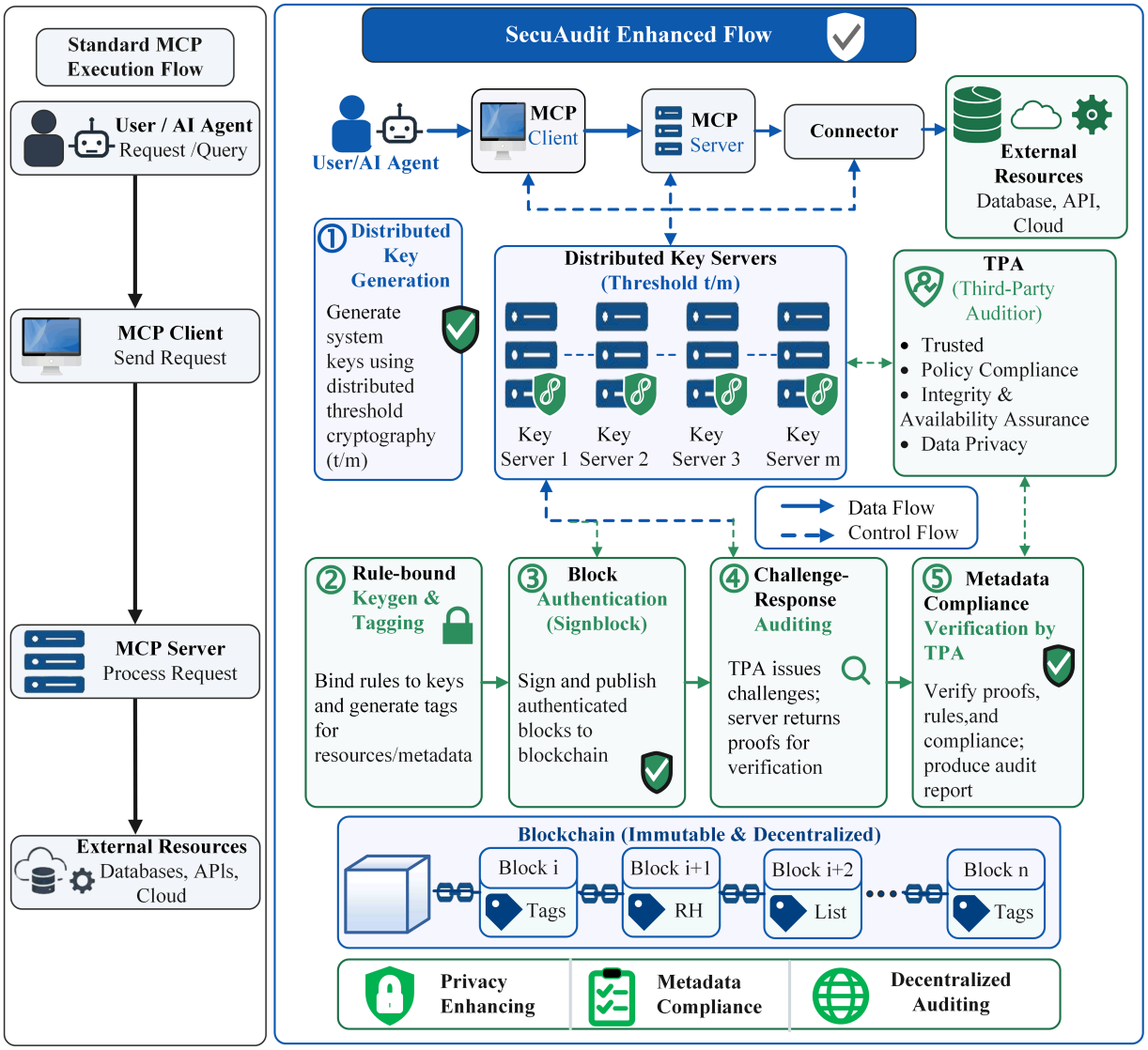

Figure 1: Integration of SecuAudit into the standard MCP resource-invocation workflow. The left panel shows the standard MCP execution flow from a user or AI agent to the MCP client, MCP server, and external resources. The right panel shows the SecuAudit-enhanced workflow: (1) distributed system-key generation by m key servers under a t/m threshold setting; (2) generation of rule-bound user keys and file tags; (3) block authentication and publication of verifiable tags; (4) randomized challenge-response auditing; and (5) joint verification of data integrity and metadata compliance by the third-party auditor (TPA). Solid arrows denote data or resource-invocation flows, whereas dashed arrows denote control and auditing interactions. RH denotes the registered semantic-rule hash.

1.1 Related Work

1.1.1 Threshold-Based Key Distribution

Threshold-based key distribution and secret sharing have been extensively studied as fundamental techniques for eliminating single points of trust and improving the robustness of distributed cryptographic systems. Existing approaches can be broadly classified into two representative paradigms: constructions based on the Chinese Remainder Theorem (CRT) and polynomial-based secret sharing schemes. The modular threshold scheme proposed by Asmuth and Bloom is a typical CRT-based construction, which enables efficient secret reconstruction under non-ideal access structures [13]. The main advantage of CRT-based schemes lies in their computational efficiency, making them attractive for resource-constrained and application-driven distributed systems. For example, CRT-based signature mechanisms have been explored in blockchain voting scenarios to improve practical efficiency [14]. Similarly, Sarkar et al. introduced a CRT-based RSA threshold cryptosystem for mobile ad hoc networks (MANETs), leveraging CRT to support decentralized operations in resource-limited environments [15]. However, the security of CRT-based schemes heavily depends on strict parameter selection, such as pairwise coprime moduli and carefully adjusted size relationships. Improper parameter choices may weaken their robustness in adversarial settings, thereby limiting their general applicability in security-critical scenarios. In contrast, the polynomial-based paradigm introduced by Shamir has become the dominant theoretical foundation for modern secret sharing and threshold key distribution [16]. Shamir's (t, m) threshold scheme employs random polynomials over finite fields and Lagrange interpolation to achieve information-theoretic security, while maintaining conceptual simplicity and structural flexibility. These properties make it particularly suitable as a building block for stronger security guarantees and more expressive cryptographic functionalities. Based on Shamir's construction, Feldman proposed a practical non-interactive verifiable secret sharing (VSS) scheme, which enables participants to verify the correctness of distributed shares and provides protection against malicious dealers [17]. Subsequent studies have further extended Shamir-based schemes to address stronger adversarial models and practical deployment requirements. For instance, recent work has improved local leakage resilience to defend against side-channel attacks [18]. Baghery et al. introduced pre-constructed publicly verifiable secret sharing and explored its applications in advanced cryptographic protocols [19]. In addition, Shamir-based secret sharing has also been adapted to blockchain environments to support complex functionalities such as multi-receiver and multi-message signcryption [20].

1.1.2 Data Security Audit Schemes

Remote data auditing has now been extensively studied for verifying the integrity of outsourced data in cloud storage environments. Existing schemes typically rely on trusted third parties, key management centers, or other centralized entities to manage keys and support the audit verification process. While such designs simplify system implementation, they are prone to introducing single points of failure and centralized trust issues, making them inappropriate for highly dynamic, heterogeneous MCP environments characterized by frequent cross-domain interactions. As shown in Table 1, existing audit schemes have achieved varying degrees of progress in data integrity verification, privacy protection, and audit efficiency, but they still have shortcomings in decentralized key management, dynamic user control, and metadata compliance verification. Shen et al. [21] proposed a certificateless auditing scheme that reduces certificate management overhead in traditional public key infrastructures and improves system availability. However, this scheme still relies on a centralized key generation process, making it difficult to completely eliminate the single-trust assumption. Additionally, the scheme does not fully support dynamic user addition and revocation, nor does it address the correctness verification of semantically relevant attributes such as access policies, validity periods, and usage constraints. Wu et al. [22] and Wang et al. [23] further investigated

security and privacy protection issues in remote data auditing. While these schemes offer improvements in audit security, privacy protection, or verification efficiency, their key management processes typically still rely on a centralized trusted entity. Consequently, if this entity fails or is compromised, the overall security and availability of the system may be compromised. Furthermore, such schemes primarily focus on data content integrity, making it difficult to effectively address dynamic user lifecycle management issues; they also cannot verify whether metadata and policy attributes—such as access permissions, data validity periods, and usage restrictions—have been maliciously tampered with. Chen et al. [24] proposed a Merkle multi-branch hash tree to audit dynamic data in B5G networks, though it lacks metadata compliance verification. Das et al. [25] conducted a comparative review of blockchain-assisted mechanisms for access control, identity management, and data integrity verification in cloud environments, highlighting the potential of decentralization to alleviate centralized trust and single-point-of-failure risks. In contrast, Wang et al. [26] proposed a concrete blockchain-assisted mechanism for publicly verifiable data integrity. Outside the domain of remote cloud data auditing, Simmons and Winograd [27] investigated provenance authentication for broadcast media by combining cryptographically authenticated open-standards metadata with audio and video watermarking. Although their work provides useful insights into metadata authenticity, it targets media provenance rather than outsourced cloud-data auditing and is therefore not included as a technical baseline in Table 1. In recent years, Liu et al. [28] proposed the Ped-TLARDA scheme, achieving further progress in decentralized key generation and distribution. By reducing reliance on a single trusted authority, this scheme enhances system robustness and mitigates the risk of single points of failure. However, current data auditing schemes strictly focus on bit-level data integrity verification, failing to support the joint verification of file content integrity and metadata compliance (e.g., access policies, validity periods, file types, and authorization scopes). Furthermore, even the most advanced decentralized auditing scheme, Ped-TLARDA [28], does not provide mechanisms for dynamic user addition and revocation or for updating registered compliance rules in real time.

Table 1: Feature comparison of related auditing schemes and baselines.

Scheme	Key Features			
	Decentralized Key Generation and Distribution	**Resilience to Single Points of Failure**	**Support for Dynamic User Addition and Revocation**	**Metadata Compliance Verification**
Shen et al. [21]	×	✓	×	×
Wu et al. [22]	×	×	×	×
Wang et al. [23]	×	×	×	×
Chen et al. [24]	×	✓	×	×
Wang et al. [26]	✓	✓	×	×
Liu et al. [28]	✓	✓	×	×
Our Scheme	✓	✓	✓	✓

1.1.3 MCP-Enabled AI Agent Systems

With the rapid development of large language models, AI agents are increasingly relying on external tools, services, and data sources to enhance their task execution capabilities. The MCP connects AI agents to external servers through standardized interfaces, thereby improving interoperability in accessing heterogeneous resources, collaborating across systems, and invoking tools. However, this open interaction model also expands the attack surface of AI agent systems. Since external data is typically located outside the agent's

direct security perimeter, MCP workflows may face various security threats, including data tampering, data leakage, metadata manipulation, and tool abuse [9,29–31]. Existing MCP security research has primarily focused on threat modeling, access control, tool permission management, and isolation mechanisms. These studies reveal potential risks during agent-external tool interactions and provide important references for foundational protection mechanisms within the MCP ecosystem. However, existing research typically lacks a unified cryptographic audit mechanism capable of continuously verifying the trustworthiness of external data throughout the entire resource invocation process. In particular, when AI agents retrieve data from external servers via MCP, the system must not only confirm that the data content itself has not been tampered with but also verify that metadata and policy attributes related to data usage remain correct, such as access policies, validity periods, authorization scopes, and usage constraints. If these metadata and policy attributes are maliciously modified, AI agents may still generate policy-violating or semantically incorrect outputs based on erroneous context, even if the underlying data content remains bit-for-bit intact. Therefore, AI agent systems for MCP require an audit mechanism capable of jointly verifying data integrity and metadata compliance, while further supporting decentralized key management and dynamic user operations.

1.1.4 Research Gaps

In summary, while existing studies have made progress in distributed cryptographic systems and remote data auditing, critical research gaps remain in adapting these techniques for dynamic, cross-domain MCP environments.

1. Gap in Key Management and Policy Binding: Existing threshold-based key distribution schemes are not specifically designed for dynamic resource invocation and lack native support for binding user identities, access policies, and metadata constraints to cryptographic keys within a unified framework.
2. Gap in Metadata Compliance Auditing: Current remote data auditing schemes strictly focus on bit-level data integrity verification. They fail to support the joint verification of file content integrity and metadata compliance, such as access policies, validity periods, and file types. This leaves AI agents vulnerable to context pollution and unauthorized execution.
3. Gap in Unified Frameworks for MCP: A significant deficiency remains in current AI agent systems for MCPs, as no existing scheme simultaneously integrates decentralized key generation, dynamic user management, data integrity verification, and metadata compliance auditing into a single comprehensive framework.

1.2 Main Contributions

To address these interconnected gaps and establish a practical paradigm for secure data circulation, we propose SecuAudit. The main contributions are summarized as follows:

1. **Privacy-Enhancing Auditing Framework.** We propose SecuAudit, a privacy-enhancing decentralized auditing framework for secure data circulation in MCP-enabled AI agent environments. It enables reliable verification of external resources during dynamic data invocation. Unlike existing solutions that focus only on bit-level integrity, SecuAudit can jointly verify file content integrity and metadata compliance.
2. **Decentralized Key Management with Dynamic Operations.** We design a threshold-based key management mechanism that supports dynamic user revocation and semantic-rule updates gated by an on-chain threshold-count check over an approval count reported by the off-chain governance process. The asymptotic complexity of KeyGen is reduced from $\mathcal{O}(m \cdot t \cdot T_{\text{exp}})$ to $\mathcal{O}(m \cdot T_{\text{exp}})$. In a controlled, same-environment comparison with our reimplementation of Ped-TLARDA, SecuAudit achieved a 54.6% reduction in KEYGEN latency at the largest evaluated configuration, where $(t, m) = (30, 59)$.

3. **Lightweight Protocol Design and Security Evaluation.** We develop a challenge-response auditing protocol and evaluate its security and computational overhead. Measurements from our implementation show an average additional Signblock overhead of 2.5% relative to our Ped-TLARDA reimplementation and a cost of 499,021 gas for the evaluated contract-interaction lifecycle on local Ganache. Under the defined threat model and cryptographic assumptions, the formal analysis and controlled attack cases indicate resistance to data forgery, metadata tampering, unauthorized rule replacement, and sub-threshold collusion attacks.

Unlike existing approaches that separately address data integrity auditing or access control enforcement, SecuAudit introduces a unified cryptographic binding mechanism that couples file blocks, metadata constraints, and dynamic authorization states into a single auditable object. The novelty lies in the cross-layer binding and verification design rather than the individual cryptographic primitives themselves.

2 Results

2.1 Experimental Setup

The evaluation framework for SecuAudit is deployed on a high-performance workstation equipped with an Intel Xeon W-2235 CPU clocked at 3.80 GHz, 32 GB DDR4 RAM, and an NVIDIA GeForce RTX 3060 GPU, operating under Microsoft Windows 10 Pro (Build 19045). To eliminate execution volatility and ensure rigorous empirical consistency, a standardized Unix-like CLI environment is maintained via Git Bash across all deterministic automation workflows and build pipelines. To facilitate future independent verification, we provide detailed descriptions of the hardware configuration, software dependencies, cryptographic parameters, and evaluation procedures. Although the current evaluation is conducted using our implementation, all reported measurements are obtained through repeated executions under fixed configurations to improve experimental transparency and reproducibility.

The off-chain protocol core and cryptographic primitives are engineered within the Java ecosystem, executing on the OpenJDK Temurin 1.8.0_492 runtime environment. Algebraic operations and bilinear pairing constraints are instantiated using the Java Pairing-Based Cryptography Library (JPBC v2.0.0) under a Type-A pairing configuration (160-bit subgroup order, 512-bit base field), backed by Bouncy Castle v1.78 for underlying multi-precision integer arithmetic and secure hashing. The Model Context Protocol (MCP) multi-agent system and its heterogeneous resource connectors are benchmarked using CPython 3.11.3 and the FastMCP harness, with statistical variance and confidence bounds computed via NumPy and Pandas. The decentralized policy registry and threshold compliance auditing mechanisms are implemented as Solidity (v0.8.36) smart contracts, compiled using the viaIR-based Yul intermediate representation pipeline with optimization enabled and set to 200 runs. The EVM target is configured as Cancun, supporting post-Dencun EVM semantics. Localized Ethereum Virtual Machine (EVM) opcode behavior and precise gas footprints are profile-tested utilizing Ganache 7.9.2 (JavaScript fallback mode) running on Node.js v24.15.0 and npm v9.6.6. Interface binding and asynchronous ledger interactions are facilitated through Web3.js v4.16.0.

2.2 Algorithmic Benchmarks

We investigated the impact of block size on system performance using a fixed 2 MB file with security parameters $m = 8$, $t = 4$, and $c = 8$, while varying the block size from 1 to 128 KB. As shown in Fig. 2, the Signblock time decreases significantly as the block size increases, from 82399.09 ms at 1 KB to 631.47 ms at 128 KB. This is because larger blocks reduce the total number of blocks and therefore decrease the number of authenticators that need to be generated. In contrast, the Genproof and Checkproof phases remain relatively stable across different block sizes, ranging from 66.08 to 85.85 ms and from 247.34 to 293.36 ms, respectively.

The Usrrevo and Updaterule operations incur negligible overhead, remaining below 0.07 and 0.05 ms in all configurations. Although the 128 KB block size achieves the lowest Signblock time, we select 64 KB for subsequent experiments as a balanced setting. At 64 KB, the Signblock time is reduced to 1285.78 ms, while Genproof and Checkproof remain efficient at 70.23 and 247.34 ms, respectively. Compared with excessively large blocks, 64 KB provides a practical trade-off between signing efficiency, block granularity, memory usage, and auditing flexibility, and is therefore adopted in the following experiments.

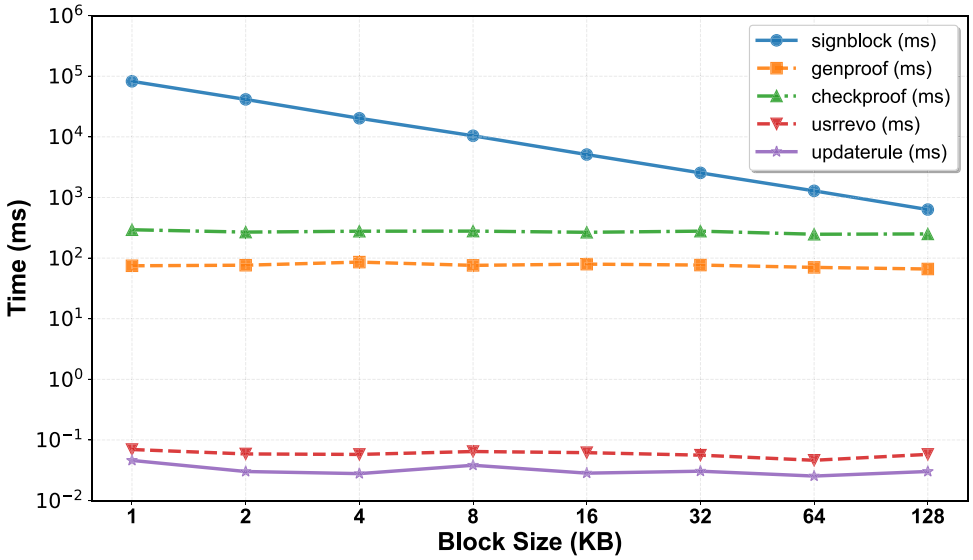

Figure 2: Effect of block size on the runtime of SecuAudit operations. A fixed 2 MB file was divided using block sizes ranging from 1 to 128 KB under $m = 8$, $t = 4$, and $c = 8$, where m denotes the total number of key servers, t denotes the reconstruction threshold, and c denotes the number of challenged blocks. Each marker represents the measured execution time under the corresponding block-size configuration. The y-axis is displayed on a logarithmic scale. As the block size increases, the number of data blocks and authenticators decreases, resulting in a substantial reduction in Signblock latency, whereas Genproof and Checkproof remain relatively stable.

The per-block signing cost remains remarkably constant at approximately 40.03 ms per block (CV = 1.0%) across the 128× range of n, from 16 to 2048 blocks. Linear regression of Signblock time against n yields Signblock = $40.27n - 29.80$ ms with $R^2 = 0.99996$, providing strong quantitative support for the $\mathcal{O}(n)$ Signblock complexity stated in Table 2. By comparison, Genproof remains within 66.08–85.85 ms and exhibits negligible dependence on n ($R^2 = 0.02$), which is consistent with its $\mathcal{O}(c)$ theoretical complexity when the challenge size c is fixed. Checkproof ranges from 247.34 to 293.36 ms and yields $R^2 = 0.45$, indicating a moderate empirical association with n in the evaluated measurements rather than negligible correlation. Therefore, these data alone do not establish that Checkproof is independent of n. Nevertheless, its runtime remains within a relatively narrow range and does not increase proportionally over the 128× variation in n; thus, the observed behavior is compatible with, but does not independently confirm, the $\mathcal{O}(c)$ complexity stated in Table 2.

Then, we systematically evaluated the impact of the threshold value on the performance of SecuAudit, as summarized in Fig. 3. Experiments were conducted with a fixed block size of 64 KB, 10 KS nodes, a file consisting of 1000 data blocks, and 100 challenged blocks per audit. The threshold t was varied from 2 to 7, and each configuration was executed 100 times to obtain average results. The results show that the Setup phase increases from 1376.53 ms at $t = 2$ to 2063.08 ms at $t = 7$. This is because a larger threshold increases the degree of the distributed key generation polynomial and introduces more coefficient commitments and

share verification operations. In contrast, the Keygen phase fluctuates within a relatively narrow range, from 590.78 to 674.20 ms, indicating that its practical overhead is only moderately affected by t under the 10-node setting. The Updaterule operation remains consistently lightweight, ranging from 0.023 to 0.032 ms, which shows that rule update checking is almost insensitive to the threshold value.

Table 2: Time complexity comparison across phases.

Scheme	Keygen	Signblock	Genproof	Checkproof
Liu et al. [28]	$O(m \cdot t \cdot T_{exp})$	$O(n \cdot T_{exp})$	$O(c \cdot (T_{exp} + T_{mul}))$	$O(c \cdot (T_{exp} + T_{mul}) + T_{pair})$
SecuAudit	$O(m \cdot T_{exp})$	$O(n \cdot T_{exp})$	$O(c \cdot (T_{exp} + T_{mul}))$	$O(c \cdot (T_{exp} + T_{mul}) + T_{pair})$

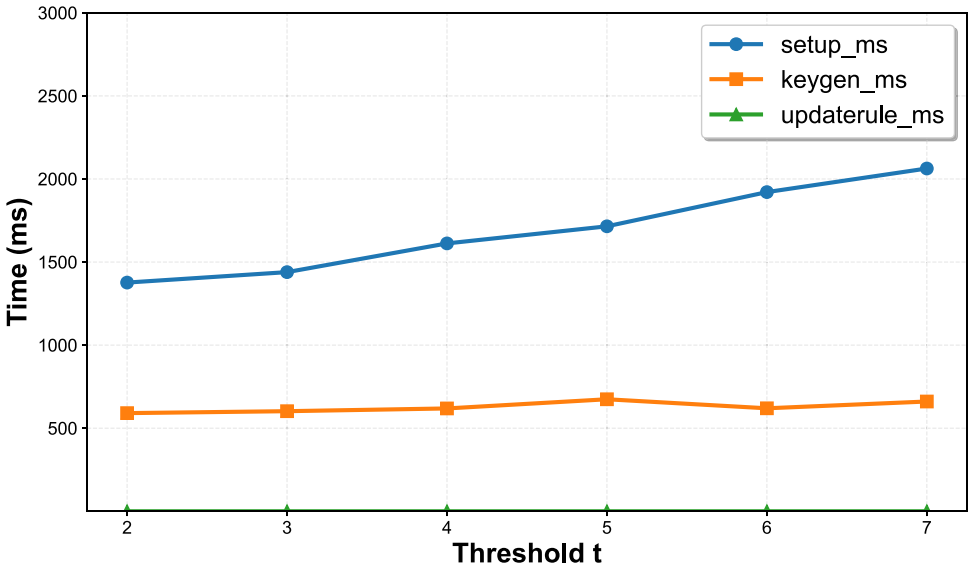

Figure 3: Effect of the threshold value t on the execution time of SecuAudit operations. Experiments were conducted using a block size of 64 KB, m = 10 key servers, n = 1000 data blocks, and c = 100 challenged blocks per audit. The threshold value t was varied from 2 to 7. Each marker represents the mean execution time over 100 independent runs. Setup latency increases with t because a higher threshold requires additional polynomial coefficients, commitments, and share-verification operations. Keygen latency varies moderately, whereas Updaterule remains below 0.04 ms.

Based on the performance profile in Fig. 3, we select t = 5 as a balanced threshold setting for subsequent experiments. Although t = 5 does not achieve the minimum execution time, it provides a practical trade-off between collusion resistance and computational overhead. Under this setting, the Setup and Keygen phases take 1715.11 and 674.20 ms, respectively, while Updaterule remains negligible at 0.030 ms. Compared with lower thresholds, t = 5 improves resistance against colluding KS nodes, since fewer than five KS nodes cannot reconstruct the system master secret. Compared with higher thresholds, it avoids further increasing the Setup cost while still requiring only half of the 10 KS nodes to participate in key reconstruction. Therefore, t = 5 is adopted in the following experiments to balance security, availability, and efficiency.

Next, we further examined the effect of the total number of *KS* nodes on system performance, as illustrated in Fig. 4. The threshold was fixed at $t = 5$, the file contained 1000 data blocks, and 100 blocks were challenged in each audit. As the number of *KS* nodes increases from 5 to 10, the Setup time rises from 594.66 to 1734.03 ms, while the Keygen time increases from 341.10 to 603.31 ms. This is because more *KS* nodes introduce additional polynomial-share generation, public share publication, and response verification operations. In contrast, the Updaterule operation remains nearly constant and negligible, ranging from 0.023 to 0.041 ms. The results indicate that increasing the number of *KS* nodes improves the robustness of distributed key management but also introduces higher initialization and key-generation overhead. Under the setting $m = 10$ and $t = 5$, an adversary must compromise at least five *KS* nodes to reconstruct the system master secret, while any five valid *KS* nodes are sufficient to support user key generation. Therefore, we adopt 10 *KS* nodes as a balanced configuration in the main experiments, providing stronger collusion resistance and acceptable computational overhead. To further investigate scalability, comparative experiments were additionally conducted with configurations up to $m = 59$ *KS* nodes (Fig. 5), demonstrating consistent performance trends as the network size increases. Evaluation on substantially larger distributed deployments is left for future work.

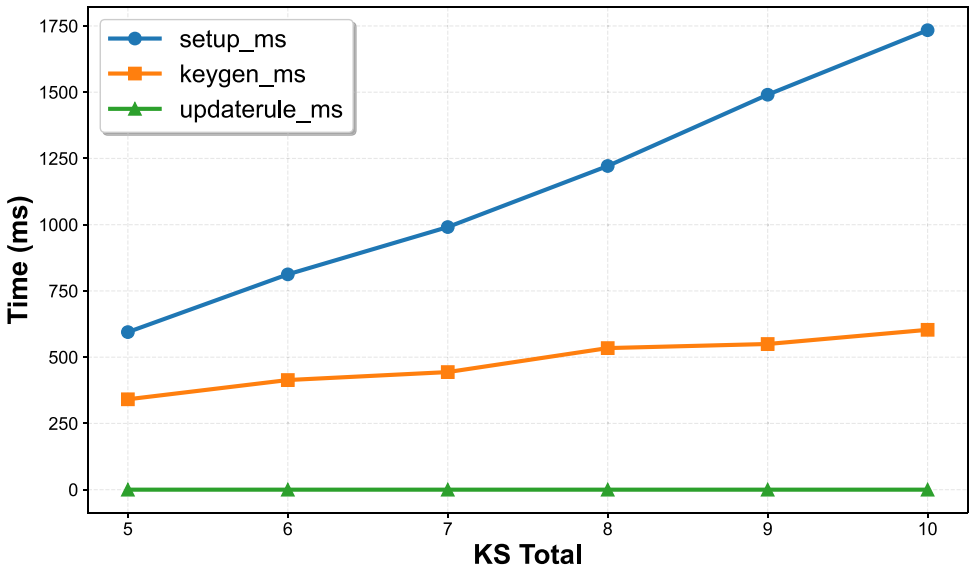

Figure 4: Effect of the total number of key servers m on the execution time of SecuAudit operations. The threshold was fixed at $t = 5$, the file contained $n = 1000$ data blocks, and $c = 100$ blocks were challenged in each audit. The total number of key servers m was varied from 5 to 10. Each marker represents the measured execution time under the corresponding configuration. Increasing m raises Setup and Keygen latency because more polynomial shares, public shares, and authenticated responses must be generated and verified, whereas Updaterule remains nearly constant and below 0.05 ms.

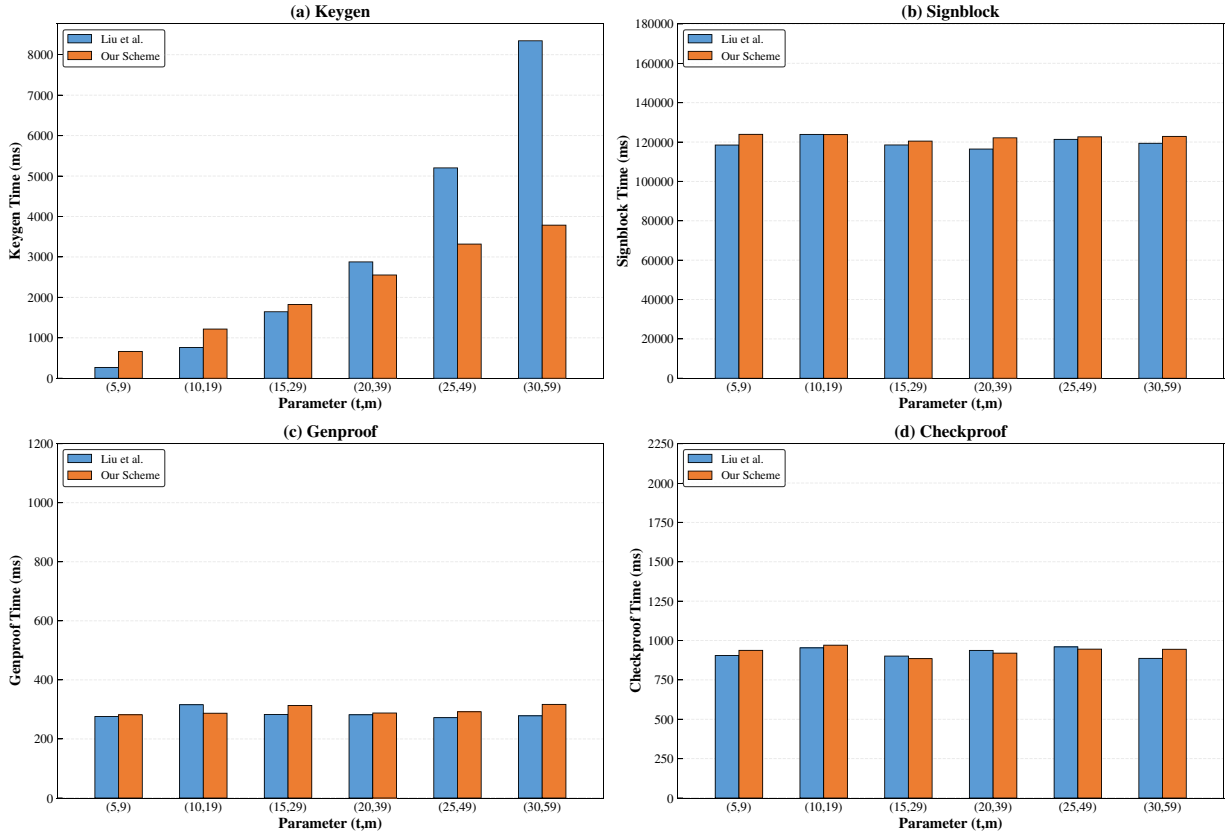

Figure 5: Runtime comparison between SecuAudit and Ped-TLARDA [28]. The panels compare the execution times of (**a**) Keygen, (**b**) Signblock, (**c**) Genproof, and (**d**) Checkproof under six threshold configurations: $(t, m) = (5, 9), (10, 19), (15, 29), (20, 39), (25, 49)$, and $(30, 59)$, where t denotes the reconstruction threshold and m denotes the total number of key servers. All experiments were conducted using a Type-A pairing configuration, $n = 3000$ data blocks, a block size of 64 KB, and $c = 30$ challenged blocks per audit. Each bar represents the mean execution time over 100 independent runs. SecuAudit exhibits higher Keygen latency under small parameter settings but achieves better scalability as t and m increase. It introduces a slightly higher Signblock cost because of the additional block-level and metadata-level bindings, while its Genproof and Checkproof performance remains comparable to that of Ped-TLARDA.

As shown in Fig. 6, the runtime distribution of SecuAudit is dominated by the Signblock phase, which is expected because this phase generates authenticators for all outsourced blocks. This overhead is a one-time preprocessing cost and is naturally parallelizable across file blocks. Once the data have been signed and outsourced, the online auditing process remains efficient. In particular, both Genproof and Checkproof complete within several seconds, showing that the proposed scheme can verify outsourced data without downloading the entire file. The Keygen phase also stays within a practical range, completing in less than 10 s under the tested setting. This cost is acceptable because key generation is performed only when a user joins the system or when rule-bound keys need to be refreshed. The Setup phase is an initialization cost and does not affect routine auditing. In addition, user revocation and rule update checks incur only negligible local overhead. Overall, the results indicate that SecuAudit shifts most computation to offline preprocessing while keeping the online audit procedure lightweight. The reported measurements only capture local cryptographic computation and simulated contract logic; blockchain consensus latency, gas cost, transaction confirmation time, and network propagation delay are not included.

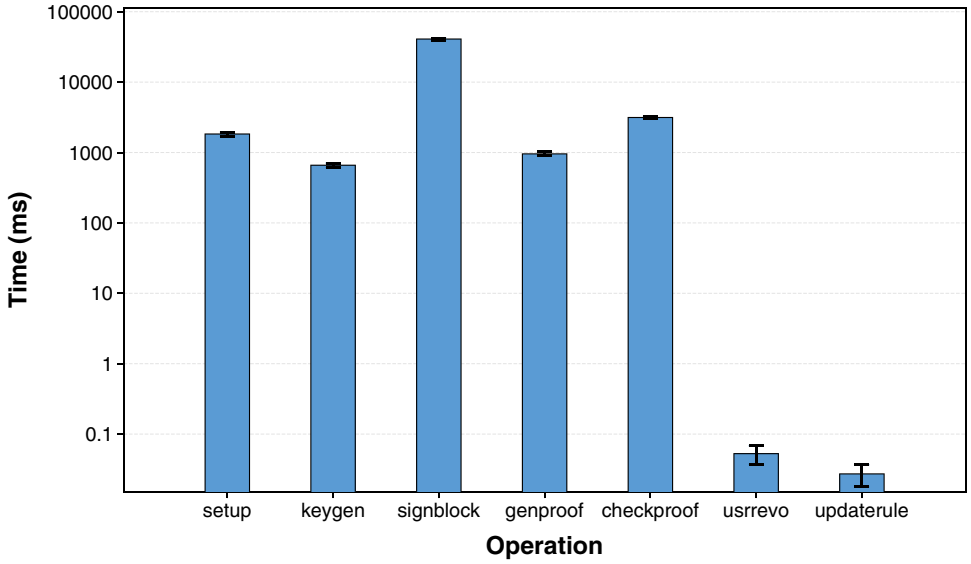

Figure 6: Execution time of the main SecuAudit operations. The experiments were conducted using a block size of 64 KB, $n = 1000$ data blocks, $c = 100$ challenged blocks per audit, $m = 10$ key servers, and a reconstruction threshold of $t = 5$. Each bar represents the mean execution time over 100 independent runs, and the error bars indicate the standard deviation. The y-axis is shown on a logarithmic scale because the execution times span several orders of magnitude. Signblock incurs the highest runtime because it generates authenticators for all outsourced data blocks and is performed as a one-time offline preprocessing operation. Genproof and Checkproof represent the recurring online auditing operations, whereas Usrrevo and Updaterule incur negligible computational overhead. The reported times include local cryptographic computation and simulated smart-contract logic, but exclude blockchain consensus latency, transaction-confirmation delay, network-propagation delay, and gas cost.

To characterize run-to-run variability within our implementation, each configuration was executed 100 times with a block size of 64 KB and parameters $n = 1000$, $c = 100$, $m = 10$, and $t = 5$. Table 3 reports the mean, standard deviation (SD), standard error of the mean (SEM), and 95% confidence interval (CI) for each operational phase. These statistics quantify repeatability within the authors' experimental environment; they do not assess reproducibility across independent implementations, hardware platforms, or evaluators.

Table 3: Performance metrics of system operations.

Operation	Mean (ms)	SD (ms)	SEM (ms)	95% Confidence Interval (CI) [Lower Bound, Upper Bound]
Setup	1831.98	118.18	11.82	[1808.82, 1855.15]
Keygen	659.77	47.88	4.79	[650.39, 669.16]
Signblock	40,930.55	1359.02	135.90	[40,664.19, 41,196.92]
Genproof	956.27	61.87	6.19	[944.15, 968.40]
Checkproof	3143.51	148.26	14.83	[3114.45, 3172.57]
Usrrevo	0.053	0.015	0.002	[0.050, 0.056]
Updaterule	0.027	0.009	0.001	[0.025, 0.029]

To further validate the scalability of SecuAudit under large-scale workloads, we conducted additional experiments with actual executions of the Signblock phase, rather than relying solely on regression-based estimation. The dataset size was increased from $n = 10^2$ to $n = 10^6$ blocks while keeping other parameters

fixed ($c = 100$, $m = 10$, $t = 5$). The detailed results are summarized in Table 4. The experimental results show that the Signblock latency increases approximately linearly with the number of blocks. Specifically, the measured execution times for $n = 10^4$ and $n = 10^6$ are 458.34 and 45904.78 s, respectively. Compared with the previous regression model, the deviations are within 14%, demonstrating that the theoretical $O(n)$ complexity reflects the demonstrates linear scalability trend behavior even at the million-block scale. Since Genproof and Checkproof depend mainly on the challenge size c rather than the total number of data blocks n, the large-scale validation focuses on Signblock, which is the only phase with linear dependency on the dataset size.

Table 4: Large-scale experimental validation of Signblock scalability.

Number of Blocks n	Experimental Signblock Time (ms)
10^2	4824.664
10^3	47,187.969
10^4	458,344.456
10^5	4,536,490.821
10^6	45,904,777.516

2.3 Comparative Experiment

To ensure a fair and consistent comparison, both the SecuAudit and Ped-TLARDA [28] were evaluated under the same pairing environment using a Type A pairing-friendly curve. The comparison focuses on the relative computational overhead of protocol components rather than curve-specific optimizations. We conducted a series of comparative experiments between SecuAudit and Ped-TLARDA under identical parameters. The threshold t and the total number of KS nodes m were set to $(5, 9)$, $(10, 19)$, $(15, 29)$, $(20, 39)$, $(25, 49)$, and $(30, 59)$. The dataset consisted of 3000 file blocks, with 30 blocks challenged in each audit, and each block was set to 64 KB. Each configuration was executed 100 times, and the average execution time was reported in milliseconds for the Keygen, Signblock, Genproof, and Checkproof phases.

As illustrated in Fig. 5, the four phases exhibit different performance trends. In the Keygen phase, SecuAudit incurs higher latency under small parameter settings because each KS response must be authenticated and verified before key reconstruction. However, as t and m increase, its growth rate is lower than that of Ped-TLARDA, and SecuAudit achieves better scalability under larger parameter settings. This observation is consistent with Table 2, where the Keygen complexity of SecuAudit is reduced from $O(m \cdot t \cdot T_{exp})$ to $O(m \cdot T_{exp})$.

This qualitative observation is quantitatively validated by linear regression of the data in Table 5: SecuAudit Keygen scales as Keygen $= 64.69m + 27.73$ ms with $R^2 = 0.996$ (per-m cost $= 64.9$ ms, CV $= 5.5\%$), while Ped-TLARDA Keygen scales as $4.65\,(m \cdot t) - 257.95$ ms with $R^2 = 0.988$ (per-$(m \cdot t)$ cost $= 4.4$ ms, CV $= 17.6\%$). The near-perfect linear fit of both models, with the per-unit cost remaining approximately constant across the full parameter sweep, directly validates the asymptotic complexity reduction from $O(m \cdot t \cdot T_{\text{exp}})$ to $O(m \cdot T_{\text{exp}})$. In the Signblock phase, SecuAudit shows slightly higher latency than Ped-TLARDA. Although both schemes have the same asymptotic complexity of $O(n \cdot T_{exp})$, SecuAudit introduces stronger block-level and metadata-level bindings during authenticator generation. Specifically, each authenticator is associated with the user identity, public key component, file identifier, and block index, and the implementation also processes block hashes before mapping data into Z_r. These operations increase the constant overhead in practice. Nevertheless, Signblock is a preprocessing operation performed before outsourcing and can be parallelized over file blocks.

Table 5: Quantitative performance comparison between SecuAudit and Ped-TLARDA (Liu et al.) across six (t, m) parameter configurations. Each configuration was executed 100 times; reported values are arithmetic means. Δ% = (SecuAudit − Ped-TLARDA)/Ped-TLARDA × 100%. Negative values indicate SecuAudit is faster.

(t, m)	Keygen (ms)		Signblock (ms)		Genproof (ms)		Checkproof (ms)	
	Liu	Ours (Δ%)	Liu	Ours (Δ%)	Liu	Ours (Δ%)	Liu	Ours (Δ%)
(5, 9)	268	664 (+147.4%)	118,476	123,910 (+4.6%)	276	282 (+2.1%)	904	938 (+3.7%)
(10, 19)	763	1,217 (+59.6%)	123,876	123,770 (−0.1%)	316	287 (−9.1%)	953	970 (+1.8%)
(15, 29)	1645	1825 (+10.9%)	118,514	120,474 (+1.7%)	283	313 (+10.8%)	901	885 (−1.8%)
(20, 39)	2878	2554 (−11.3%)	116,439	122,139 (+4.9%)	282	288 (+2.1%)	937	920 (−1.9%)
(25, 49)	5202	3318 (−36.2%)	121,360	122,649 (+1.1%)	272	292 (+7.3%)	959	945 (−1.5%)
(30, 59)	8344	3786 (−54.6%)	119,370	122,866 (+2.9%)	278	317 (+13.9%)	887	944 (+6.5%)
Average |Δ%|		53.3%		2.5%		7.6%		2.9%

For the Genproof and Checkproof phases, the two schemes show comparable performance. This result is also consistent with Table 2, since both schemes have the same asymptotic complexity in these two phases, namely $O(c \cdot (T_{exp} + T_{mul}))$ for Genproof and $O(c \cdot (T_{exp} + T_{mul}) + T_{pair})$ for Checkproof. The small fluctuations in Fig. 5 mainly come from implementation-level differences and constant factors rather than asymptotic complexity differences.

To provide a complete quantitative data, Table 5 summarizes the per-phase execution times and percentage differences between SecuAudit and Ped-TLARDA across all six (t, m) configurations.

As summarized in Table 5, the four phases exhibit quantitatively distinct behaviors. In the Keygen phase, at $(t = 5, m = 9)$, SecuAudit is 147.4% slower than Ped-TLARDA owing to additional per-response authentication verification required for security. However, as t and m increase, this overhead rapidly diminishes and eventually reverses: at $(t = 20, m = 39)$, SecuAudit becomes 11.3% faster, and at $(t = 30, m = 59)$, it achieves 54.6% lower Keygen latency. This crossover and subsequent advantage are consistent with the asymptotic complexity reduction from $O(m \cdot t \cdot T_{\text{exp}})$ to $O(m \cdot T_{\text{exp}})$ shown in Table 2. In the Signblock phase, SecuAudit introduces an average of only 2.5% additional overhead across all six parameter groups (range: −0.1% to +4.9%). This modest one-time offline cost enables the metadata-level binding that Ped-TLARDA does not provide. For Genproof and Checkproof—the two online auditing phases that dominate recurring costs—the two schemes exhibit comparable performance: Genproof differences range from −9.1% to +13.9%, and Checkproof differences range from −1.9% to +6.5%. Overall, SecuAudit achieves its extended functionality (decentralized key management, dynamic user revocation, and metadata compliance verification) at negligible online performance cost, with average absolute differences of 53.3%, 2.5%, 7.6%, and 2.9% for Keygen, Signblock, Genproof, and Checkproof, respectively. The Keygen average is dominated by small-parameter authentication overhead; at scale ($t \geq 20$), SecuAudit is consistently faster.

Furthermore, the comparative measurements are consistent with the expected Keygen scaling of the two schemes within the tested parameter range. SecuAudit scales approximately linearly with m, whereas our Ped-TLARDA reimplementation exhibits growth associated with $m \cdot t$. These results indicate more favorable Keygen scaling for SecuAudit under the larger evaluated configurations, but they do not establish implementation-independent superiority across other platforms or implementations.

2.4 End-to-End Performance and Concurrency Scalability

Building upon the foundational experimental setup described in Section 2.1, we further extended our evaluation into a comprehensive discrete-event simulation framework to analyze the system's end-to-end (E2E) performance. While the local cryptographic computation follows the exact benchmarks detailed previously, the on-blockchain transaction overheads were verified via Solidity smart contracts deployed on a local Ganache Ethereum emulator. Crucially, to capture the empirical stochasticity of cross-domain resource access within the Model Context Protocol (MCP) ecosystem, the network propagation delays of REST connectors and database I/O latency of DB connectors were statistically modeled using Gaussian (Normal) distributions configured with realistic Round-Trip Times (RTT). As detailed in Table 6, we tested four deployment scenarios spanning from a local area network to a wide area network (300 ms delay), under distinct blockchain confirmation profiles including Layer-2 Polygon (2.0 s) and the Ethereum Mainnet (12.0 s). Even under the most rigorous scenario (WAN + Remote DB + Mainnet), the cryptographic computation remains the dominant performance bottleneck, accounting for 79.2% of the total E2E latency, while the MCP communication and blockchain consensus overheads are minimal. Furthermore, to investigate the system's resilience under volatile multi-agent workloads, we simulated high-concurrency request patterns utilizing a Poisson M/G/k queuing model combined with extensive Monte Carlo trials. As shown in Table 7, the system scales robustly with the number of available CPU cores, achieving a peak throughput of 0.33 TPS under a 16-core configuration, while the Coefficient of Variation (CV) drops from 14.1% to 0.22% as concurrency increases.

Table 6: Performance evaluation across different deployment scenarios.

Scenario	Network Latency	DB I/O	Blockchain Confirmation	Cryptography (Mean ± SD)	E2E Time (Mean ± SD)	P95 (s)	P99 (s)	Cryptography Ratio
Local (No Latency)	0 ms	0 ms	0 s	47.33 ± 1.37 s	47.33 ± 1.37 s	49.14	49.87	100.0%
LAN + L2 (Polygon 2s)	50 ± 10 ms	10 ± 3 ms	2.0 s	47.35 ± 1.54 s	49.40 ± 1.54 s	51.74	52.17	95.8%
WAN + Mainnet (12s)	200 ± 50 ms	50 ± 10 ms	12.0 s	47.57 ± 1.35 s	59.83 ± 1.36 s	61.70	62.20	79.5%
WAN + Remote DB + Mainnet	300 ± 80 ms	100 ± 30 ms	12.0 s	47.21 ± 1.43 s	59.60 ± 1.44 s	61.63	62.51	79.2%

Table 7: Performance evaluation (wall time, CV, and TPS) across different CPU cores.

Agent	2 Cores		4 Cores		8 Cores		16 Cores		
Count	Wall (s)	CV (%)	Wall (s)	CV (%)	Wall (s)	CV (%)	Wall (s)	CV (%)	TPS
1	56.8	18.40	56.1	15.00	59.4	27.00	56.5	14.10	0.018
10	240.7	1.40	143.6	1.60	97.1	1.70	57.9	5.60	0.173
50	1192	0.52	616.5	0.46	329.8	0.68	189.2	0.90	0.264
100	2384	0.35	1197	0.59	617.1	0.52	331.1	0.52	0.302
500	11,891	0.14	5955	0.16	2992	0.17	1515	0.22	0.330

Overall, these results demonstrate that SecuAudit scales robustly across key dimensions within the MCP ecosystem. First, regarding multi-agent concurrency, the system handles increasing concurrent requests (from 1 to 500) under a Poisson M/G/k queuing distribution with the completion time Coefficient of Variation (CV) decreasing monotonically to 0.22% under 16-core parallelism. Second, regarding heterogeneous connector latency across REST API and database connectors with varied network delays (50–300 ms RTT), cryptographic computation consistently accounts for the majority (79.2%) of end-to-end latency, proving

that diverse external resource types do not introduce disproportionate communication bottlenecks. Finally, node and governance operations scale efficiently; key generation scales linearly with $\mathcal{O}(m \cdot T_{exp})$, while rule governance operations rely on fixed $\mathcal{O}(1)$ on-chain lookups independent of the total user population. In addition to the block-scale experiment described above, we conducted a connector-level evaluation using five heterogeneous MCP payloads. This experiment was designed to assess the applicability of SECUAUDIT to representative REST API, vector-store, and relational-database resources, rather than to evaluate large-dataset scalability, because each individual payload was smaller than the 64 KB block size. Specifically, we acquired three live REST payloads from two public APIs: a single post record (HTTP 200, 282 bytes) and a multi-record post list (1308 bytes) from the JSONPlaceholder API, representing API responses and remote documents, as well as real-time forecast data (HTTP 200, 537 bytes) from the Open-Meteo weather API, representing dynamic tool outputs. We additionally constructed two representative business payloads: a 768-dimensional embedding-vector retrieval result (40,328 bytes, including similarity scores) following a representative Pinecone/Weaviate schema, and a PostgreSQL compliance-audit record (637 bytes, containing 12 metadata fields). Injecting these five heterogeneous payloads into the complete SecuAudit pipeline (KEYGEN → METADATA BINDING → SIGNBLOCK → GENPROOF → CHECKPROOF), configured with $m = 10$, $t = 5$, and $c = 30$, yielded the following results. First, the SIGNBLOCK latency for each payload remained approximately 10–11 ms because all payloads occupied only one 64 KB block. Second, the GENPROOF latency (approximately 310 ms, CV = 0.49%) and CHECKPROOF latency (approximately 22 ms, CV = 1.68%) remained stable across the different payload types. This result indicates that these phases are primarily determined by the challenge size c, rather than by the internal structure of the external resource. Third, the total end-to-end latency ranged from 1037 to 2544 ms. Cryptographic processing contributed approximately 1019.1 ms on average, whereas network retrieval contributed approximately 628.8 ms on average. Although cryptographic processing represented the larger average component, the latency variation among the payloads was mainly caused by network-fetch fluctuations, which reached 1562.8 ms for the weather API. All complete audits finished within approximately 2.6 s. Finally, all five payloads successfully passed the metadata compliance verification, $M_{\text{meta}} \vDash \text{rule}_{\text{cur}}$. These results demonstrate the compatibility of SecuAudit with heterogeneous real-world MCP resource types, while the large-dataset scalability of the SignBlock phase is evaluated separately in Table 4.

2.5 On-Blockchain Overhead and Attack-Driven Safety Mitigations

The economic feasibility of the blockchain component is critical for practical deployment. We deployed the SecuAudit smart contract on a local Ganache emulator to measure precise transaction-level Gas consumption. The reported Gas results focus on operational costs after deployment, including semantic rule updates, tag uploads, and user revocation, while contract deployment cost is excluded because it is a one-time initialization overhead. As presented in Table 8, a complete contract interaction lifecycle, including semantic rule update, tag uploading, and potential user revocation operations, consumes a total of 499,021 gas. At a standard rate of 20 Gwei/Gas and an ETH price of $2000, this translates to approximately $19.96 on the Ethereum Mainnet

Under the evaluated implementation, the contract state associated with each file requires approximately 770 bytes, including the file tag, rule-registry entry, and revocation information. Transaction-confirmation latency was not directly measured on a public blockchain. Using an assumed Layer-2 confirmation interval of 2.0 s, the simulation estimates that the cumulative confirmation latency of the evaluated contract interactions is within 15 s. Actual confirmation latency and transaction cost may vary depending on the selected blockchain network, network congestion, fee policy, and confirmation requirements.

Table 8: Gas consumption and description of smart contract operations.

Contract Operation	Gas Used	Function/Description
Semantic rule update	138,962	$R[ID] \leftarrow \langle \text{rule}', RH' \rangle$ after KS threshold voting
Tag upload	290,867	$T[fid] \leftarrow \langle ID, A, \text{name}, RH, M_{\text{meta}}, Z, hTag \rangle$ after quadruple verification
User revocation	69,192	$\zeta_{\text{deny}} \cup \{ID, A\}$; $\zeta_{\text{allow}} \backslash \{ID, A\}$
Total	**499,021**	—

To examine the behavior of the prototype under the threat model defined in Section 4.1.2, we implemented six attack cases covering data-block tampering, expired-rule replay, revoked-user bypass, metadata modification, file-type substitution, and old-rule rollback. As summarized in Table 9, all six implemented attack cases were detected or rejected in the controlled evaluation. Data-block tampering caused the bilinear pairing verification to fail, whereas the remaining cases were rejected through the corresponding smart-contract state and rule-consistency checks. In addition, one legitimate control case completed without a false-positive decision. This observation confirms the expected behavior of the implemented control case but does not establish a general zero false-positive rate.

Table 9: Security analysis and attack evaluation of the proposed scheme.

Attack Name	Attack Type	Attack Method	Detection Mechanism	Detection Result	Status
Data Block Tampering	Integrity	Clear Block 3; Add 1 to Block 7	Pairing verify: $e(T, g) \neq e(U \cdot C, Z)$	Tampering detected	SECURE
Expired Semantic Rule Replay	Privilege Escalation	`valid_until:2020`; Replay expired RH	RH comparison: $H_3(\text{exp}) \neq H_3(\text{cur})$	Keygen rejected	SECURE
Revoked User Bypass	Revocation Bypass	User $A \in$ denySet initiates signature	denySet.contains(A) via $O(1)$ lookup	Access blocked	SECURE
Metadata Modification	Tampering	Rename `file1` →`file1_hacked`	Tag hash: $H(\text{user}\|\|A\|\|\text{name}\|\|Z)$	Tag mismatch	SECURE
File Type Substitution	Policy Violation	Modify extension `.json` →`.bin`	Semantic rule check: allowedRH ≠ forbiddenRH	RH mismatch	SECURE
Old Rule Rollback	Rollback	Replay old RH (`junior 2025`)	Version check: $H_3(\text{current}) \neq H_3(\text{old})$	Version mismatch	SECURE
Legitimate Operation	False Positive	Correct data + valid rules	All checks passed	All passed	SECURE

Taken together, the controlled measurements characterize the implementation-specific overhead of the extended metadata-compliance functions. Relative to our Ped-TLARDA reimplementation, metadata binding introduces an average additional Signblock overhead of 2.5%, while the differences in the online auditing phases remain within 14% across the evaluated configurations. The local contract implementation consumes 499,021 gas for the evaluated interaction lifecycle and requires approximately 770 bytes of contract state per file.

2.6 Protocol Sizing and Deployment Trade-Offs

Based on the quantitative evaluation, SecuAudit provides flexible trade-offs for varied MCP deployment scales. We recommend a default block size of 64 KB, which balances signing efficiency (1285.78 ms per 2 MB) and fine-grained recovery granularity. For the audit parametrization, let n denote the total number of file blocks, s the number of missing blocks, and c the number of blocks randomly challenged in each audit. The probability that a single audit detects at least one missing block is $P_{\text{detect}}^{(1)} = 1 - \frac{\binom{n-s}{c}}{\binom{n}{c}}$. Assuming statistically independent challenge sampling across repeated audit rounds, the cumulative detection probability after r rounds is $P_{\text{detect}}^{(r)} = 1 - \left(\frac{\binom{n-s}{c}}{\binom{n}{c}}\right)^r = 1 - \left(1 - P_{\text{detect}}^{(1)}\right)^r$. In the evaluated configuration, $n = 3000$, $s = 30$, and $c = 30$, corresponding to a block-loss rate of 1%. Therefore, $P_{\text{detect}}^{(1)} = 1 - \frac{\binom{2970}{30}}{\binom{3000}{30}} \approx 0.2614$. After $r = 23$ statistically independent audit rounds, the cumulative detection probability is $P_{\text{detect}}^{(23)} = 1 - (1 - 0.2614)^{23} \approx 0.999059$, which exceeds 99.9%. This theoretical result assumes uniform random sampling, statistically independent challenge sets across audit rounds, and a fixed set of missing blocks.

All underlying cryptographic operations are securely instantiated using a Type-A pairing-friendly curve (160-bit subgroup, approximately 80-bit symmetric security), mapping protocol elements to highly compact sizes (e.g., 128 bytes per $\mathbb{G}_1$ authenticator).

3 Discussion

Demonstrated Results. Empirical results demonstrate that SecuAudit integrates decentralized key management, metadata-compliance verification, and dynamic user revocation into an MCP-oriented prototype. Relative to our reimplementation of Ped-TLARDA, SecuAudit exhibited 54.6% lower KeyGen latency at the largest evaluated configuration, where $(t, m) = (30, 59)$, while introducing an average additional SignBlock overhead of 2.5%. The blockchain-related gas and storage measurements were obtained using a local Ganache environment, whereas the deployment and high-concurrency results were generated through simulation-based analysis. In addition, all six implemented attack cases were detected or rejected under the defined threat model.

Theoretical Claims. Beyond the empirical performance metrics, the security of SecuAudit is founded on provable cryptographic guarantees. It is theoretically proven that no probabilistic polynomial-time adversary can forge a valid proof for corrupted data or bypass the metadata compliance checks, assuming the hardness of the Computational Diffie-Hellman (CDH) and Discrete Logarithm (DL) problems. Additionally, the protocol achieves strict collusion resistance; leveraging the information-theoretic properties of Shamir's threshold secret sharing, we mathematically guarantee that any coalition of fewer than t key servers cannot reconstruct the system master secret or forge valid user credentials.

Assumptions. These security and operational guarantees inherently rely on several foundational assumptions within our threat model. We assume a partially honest distributed key infrastructure, where at least t out of m key servers remain uncompromised and execute the protocol correctly. Furthermore, we rely on the standard assumption of blockchain immutability; the underlying ledger is presumed to provide deterministic smart contract execution and eventual consistency against long-range forks. From a network perspective, we assume that the communication channels between users and key servers are authenticated to prevent man-in-the-middle impersonations.

Future Work. While SecuAudit establishes a secure paradigm for data circulation, certain architectural enhancements are explicitly deferred to future work. First, although the system exhibits extreme cost-efficiency on a local Ganache emulator, future deployments will evaluate real mempool dynamics and gas

volatility on live public blockchain mainnets. Second, while the current prototype has been experimentally evaluated under distributed key-server configurations of up to 59 *KS* nodes, future work will investigate larger-scale deployments on production MCP clusters with substantially more participating nodes and real-world AI-agent burst workloads to further validate the scalability and concurrency of the proposed framework. Finally, while the current design significantly reduces data exposure, incorporating advanced privacy-enhancing extensions—such as zero-knowledge proofs (zk-SNARKs) for confidential policy compliance, random masking for audit-response privacy, and Private Information Retrieval (PIR) for access-pattern obfuscation—remains a critical future direction to achieve full user unlinkability.

4 Methods

4.1 System Model

4.1.1 System Architecture

As illustrated in Fig. 7, to clearly demonstrate the overall architecture of SecuAudit and the interaction workflow among entities, this section defines and describes the key components within the MCP ecosystem:

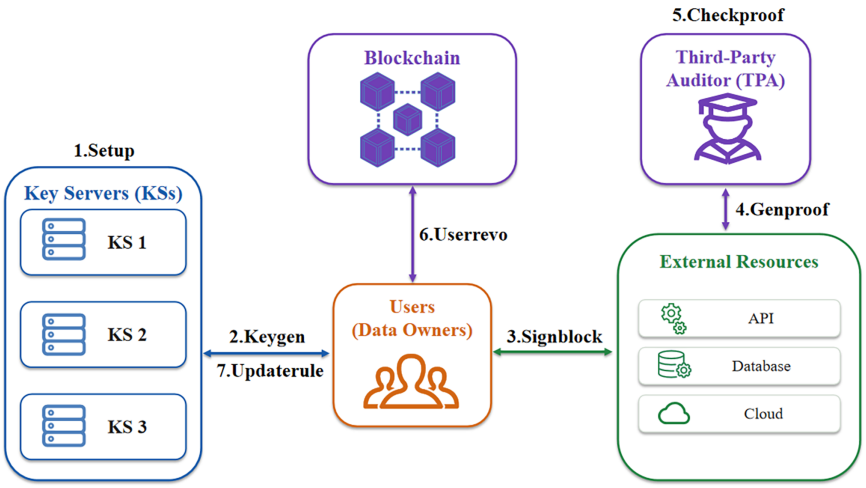

Figure 7: The Cryptographic system model and protocol interactions of SecuAudit. The system consists of distributed key servers (KSs), data owners or users, external resources, a third-party auditor (TPA), and a blockchain-based policy registry. The numbered arrows denote the seven protocol operations: (1) Setup initializes public parameters and distributed key shares; (2) Keygen reconstructs a rule-bound user key from at least t authenticated KS responses; (3) Signblock generates authenticators and outsources the authenticated data blocks; (4) Genproof produces an aggregated proof in response to a randomized audit challenge; (5) Checkproof verifies data integrity and metadata compliance against the on-chain state; (6) User revocation updates the authorization and denial sets; and (7) Rule update replaces a registered semantic rule after threshold approval. Blockchain interactions record file tags, rule hashes, and revocation information, whereas raw data remain off-chain.

Key Servers (*KS*): The *KS* nodes establish the distributed trust foundation of the system and collaboratively execute a threshold key generation protocol to produce shares of the system master private key. Since no single node or small coalition can reconstruct the complete master key, this design mitigates the risks of single-point failure and insider collusion. The *KS* nodes also support collaborative user private-key generation, participation in the off-chain policy-governance process, and user revocation.

Users: Users are the original owners and uploaders of data in the MCP environment, and may be either human users or AI agents acting on their behalf. Each user interacts with at least *t* *KS* nodes through

a two-round protocol to generate its private key collaboratively. This process cryptographically binds the user identity to the access semantic rule, thereby enabling subsequent access control, behavior tracing, and accountability. The user then computes authenticators for the outsourced data blocks and records the associated tags and provenance information on the blockchain to ensure traceability and auditability.

Blockchain: The blockchain serves as an immutable and trusted public infrastructure. It maintains system parameters, data tags, user revocation lists, and policy update records. Smart contracts deployed on the blockchain automatically enforce access control, policy updates, and user revocation in a trustless manner, ensuring transparency and auditability.

External Resources: External Resources host and manage heterogeneous resources accessed through MCP, such as databases, API servers, cloud servers. In SecuAudit, the external servers store outsourced data blocks and their corresponding authenticators, whereas connectors handle resource access forwarding, tag binding, challenge routing, and proof collection. When challenged by the TPA, the external server generates an integrity proof from the stored data to demonstrate that the target data remain correct and intact.

TPA: The TPA is an optional independent verification entity that issues lightweight random challenges to external servers on behalf of data owners or system participants and further checks that it meets current access policies and metadata and access control constraints. By validating the returned proofs, TPA can assess the integrity of remotely stored data without accessing the original data, reducing the computational burden on the user side and improving audit efficiency.

4.1.2 Threat Model

In this section, we consider a multi-party interactive environment involving malicious entities and define the threat model of SecuAudit based on the framework of provable security. We assume that system participants may deviate from the prescribed protocol due to self-interest or external compromise. The semantic rule hash RH is public and recorded on blockchain as a policy handle. The corresponding semantic rule $rule$ is maintained by an on blockchain or off-chain policy registry and is accessible to authorized verifiers during auditing. Communication between users and KS nodes is authenticated, each KS response is accompanied by a verifiable KS node signature. These signatures authenticate user–KS communication and Keygen responses; they are not submitted to or verified by the current semanticRuleUpdate smart contract. Specifically, we define five primary classes of probabilistic polynomial-time (PPT) adversaries, denoted as A:

Malicious External Server (A_1):

Adversary A_1 compromises an External Server that hosts an outsourced resource. The External Server stores the outsourced file blocks, their corresponding authenticators, and the source-side semantic metadata associated with the resource. It is distinct from the MCP Connector, which only mediates resource access and protocol-message forwarding. The objective of A_1 is to deceive an authorized verifier, such as an optional TPA, during the audit phase. Its attack surface includes two main forms: (1) *data-storage corruption*, where A_1 deletes, modifies, or incompletely stores the outsourced file blocks or their authenticators and attempts to generate a valid proof $\langle fid, \widehat{F}, T, M_{\text{meta}} \rangle$ without correctly maintaining the challenged resource state; and (2) *source-side metadata substitution*, where A_1 replaces the semantic metadata stored at the resource origin with tampered metadata M^*_{meta} and attempts to produce a proof for a resource state that is inconsistent with the metadata committed in the on-chain file tag or with the latest semantic rule $rule_{\text{cur}}$. Adversary A_1 does not control connector-level challenge routing, response forwarding, stale-response replay, or selective message suppression.

Colluding KS Nodes and Users (A_2):

Adversary A_2 attempts to undermine the distributed key infrastructure and the semantic rule binding mechanism. A_2 controls fewer than tKS nodes, namely at most $t-1$ nodes, and may collude with malicious data owners. Its attack objectives include: (1) reconstructing the system master private key σ or deriving a legitimate user's private key b without authorization; (2) forging a valid public verification component Z without obtaining enough authenticated KS responses; (3) injecting a malicious or outdated semantic rule hash RH^* during Keygen or Signblock; and (4) bypassing the smart contract threshold approval process during semantic rule updates.

Malicious MCP Client and Prompt Injection (A_3):

Adversary A_3 models an AI agent (MCP Client) that has been compromised through prompt injection, model poisoning, or credential theft. The adversary's capabilities include: (1) prompt-injection-based tool misuse, where A_3 crafts adversarial prompts that cause the MCP Client to invoke audit-related MCP tools with attacker-chosen parameters (e.g., attempting to register resources under forged metadata M^*_{meta} or triggering challenge-response for resources the client should not access); (2) unauthorized resource access, where A_3 attempts to access external resources bound to policies that do not authorize the client's identity; (3) forged metadata injection, where A_3 submits M^*_{meta} with falsified file types, validity periods, or authorization scopes during resource registration; and (4) stale rule hash replay, where A_3 submits an outdated RH_{old} during Keygen to obtain a key bound to a weaker, previously valid policy.

Compromised MCP Connector (A_4):

Adversary A_4 compromises an MCP Connector that mediates communication among the MCP Server, the TPA, and an External Server. In this adversarial case, the underlying External Server and its stored file blocks, authenticators, and source-side metadata are assumed to remain uncompromised. Consequently, A_4 cannot independently generate a fresh valid proof because it does not possess or control the authenticated resource state maintained by the External Server. The attack surface of A_4 includes: (1) *stale-response replay*, where A_4 replays a previously valid proof or metadata response containing $M^{\text{old}}_{\text{meta}}$ that is inconsistent with the metadata currently committed on-chain or with the latest semantic rule $rule_{\text{cur}}$; (2) *challenge redirection and confused-deputy attacks*, where A_4 attempts to redirect a challenge intended for one resource or requester to another resource, thereby misusing the TPA's delegated auditing authority; (3) *selective message suppression*, where A_4 drops or delays challenge-response messages and creates a denial-of-audit condition; and (4) *in-transit metadata or provenance manipulation*, where A_4 modifies the file identifier, resource version, provenance fields, or M_{meta} contained in a response while forwarding it between the External Server and the verifier. Unlike A_1, adversary A_4 operates only at the connector and message-forwarding layer and does not modify the resource state stored at the External Server.

Malicious or Semi-Honest TPA (A_5):

Adversary A_5 models a TPA that deviates from the prescribed verification protocol. In the semi-honest (honest-but-curious) variant, A_5 correctly executes the challenge-response protocol but attempts to infer sensitive information from the aggregated responses. In the fully malicious variant, A_5 may: (1) attempt data exfiltration by issuing an excessive number of challenges with overlapping or carefully crafted coefficient sets to solve for individual data blocks from the system of linear equations $\hat{F}^{(k)} = \sum_{j\in J^{(k)}} v_j^{(k)} m_j$; (2) engage in false audit reporting, claiming that a valid proof is invalid (false positive) or that an invalid proof is valid (false negative) to manipulate the system's compliance status; and (3) execute a denial-of-audit by issuing challenges at an excessive frequency to degrade the External Server's performance.

Security Assumptions:

The security proof of our scheme relies on the following infrastructural premises and cryptographic assumptions:

1. **CDH Assumption:** Given $\langle g, g^x, g^y \rangle \in G^3$, the success probability of any PPT adversary in computing g^{xy} is bounded by $\mathrm{Adv}^{\mathrm{CDH}} \lesssim 2^{-80}$ under the 160-bit subgroup order $q \approx 2^{160}$ used in our Type-A pairing configuration (corresponding to approximately 80-bit symmetric security).
2. **Discrete Logarithm (DL) Assumption:** Given $\langle g, g^x \rangle \in G^2$, the probability of any PPT adversary computing $x \in Z_q$ is bounded by $\mathrm{Adv}^{\mathrm{DL}} \lesssim 2^{-80}$ under the same 160-bit subgroup.

The quantitative threat-model parameters, their default values, and the corresponding derivations are summarized in Table 10.

Table 10: Quantitative threat model parameters.

Threat Parameter	Symbol	Default Value	Derivation
Total KS nodes	m	10 (up to 59)	Evaluated across 6 (t, m) groups
Reconstruction threshold	t	5 ($m = 10$)	Security–availability trade-off
Max. compromised KS fraction	$(t-1)/m$	40% ($m = 10, t = 5$)	Shamir threshold bound
Master secret recovery prob.	$1/q$	$\approx 2^{-160}$	Information-theoretic (Thm. 2)
CDH/DL symmetric security	κ	$\approx$80 bits	Type-A pairing (160-bit subgr.)
Hash collision bound	$q_H^2/2^{l+1}$	$\ll 2^{-128}$	$l = 256$ (SHA-256), birthday
Block-loss detection prob.	P_{detect}	$1 - \frac{\binom{n-s}{c}}{\binom{n}{c}}$	Game 3, Theorem A1
Rule-update count threshold	$a_{\mathrm{KS}} \geq t$	5 of 10	On-chain threshold-count check
On-chain KS-approval verification	—	Not implemented	No on-chain KS signature verification

4.1.3 Design Goals

To securely support AI agents dynamically invoking heterogeneous resources in the MCP environment, SecuAudit is designed to achieve the following security and functional goals:

1. **Robust Data Integrity.** The system should ensure that any modification or deletion of outsourced data by a malicious external server can be detected with a probability determined by the challenge size. Under the Computational Diffie-Hellman (CDH) assumption, no PPT adversary can forge a valid proof that passes the verification.
2. **Metadata Compliance.** SecuAudit should ensure that outsourced data satisfies the latest access policies and metadata constraints during auditing. Any unauthorized change to metadata, such as file type, validity period, authorization scope, or access rules, should be detected even if the file content remains unchanged.
3. **Collusion Resistance.** The system must withstand collusion attacks. Specifically, a malicious external server colluding with revoked users, or an adversary controlling up to $t-1$ KS nodes, must not be able to forge valid file tags, bypass access control, or reconstruct the system master private key σ.
4. **Rule Dynamic Updates.** The scheme should support policy updates and user revocation without requiring the data owner to re-download or re-upload the outsourced file content. In the strict rule-bound mode, file tags and authenticators may need to be refreshed to bind existing outsourced data to the updated semantic rule.

4.1.4 Metadata Compliance Model

To provide a mathematically rigorous foundation for verification, we formalize the policy language, metadata schema, satisfaction relations, and dynamic lifecycle management below.

Definition 1 (Compliance Syntax). A semantic rule is a 5-tuple $rule = \langle \mathcal{T}, \mathcal{V}, \mathcal{S}, \mathcal{A}, v \rangle$, where $\mathcal{T} \subseteq \mathbb{M}$ is a finite set of permitted IANA media types; $\mathcal{V} = \langle \tau_{\text{lb}}, \tau_{\text{ub}} \rangle \in (\mathbb{T} \cup \{\perp\})^2$ is the validity interval ($\perp$ denotes an unbounded endpoint); $\mathcal{S} \subseteq \mathbb{S}$ is the set of authorized scopes; $\mathcal{A} \subseteq \mathbb{A}$ represents access permission levels; and $v \in \mathbb{N}$ is the monotonically increasing policy version. The corresponding outsourced-file metadata is defined as the seven-tuple $M_{\text{meta}} = \langle \tau, \tau_{\text{from}}, \tau_{\text{until}}, \sigma, \alpha, \omega, \upsilon \rangle \in \mathcal{M} \times \mathcal{T} \times (\mathcal{T} \cup \{\perp\}) \times \mathcal{S} \times \mathcal{A} \times \{0,1\}^* \times \{0,1\}^*$. Here, τ denotes the file's IANA media type; τ_{from} denotes the beginning of the file's active validity interval; τ_{until} denotes the end of the validity interval, where $\perp$ indicates that no upper expiration bound is specified; σ denotes the declared authorization scope; α denotes the required access-permission level; ω denotes the identity of the resource owner; and υ denotes the MCP URI identifying the external resource. The symbol υ used for the MCP URI is distinct from the policy-version variable v in the semantic rule. The first five components are evaluated by the semantic-satisfaction predicate in Definition 2. The owner identity ω and MCP URI υ serve as provenance-binding fields; they are cryptographically included in the registered metadata and file tag and are verified for integrity rather than through the policy-membership conditions in Eq. (1).

Definition 2 (Semantic Satisfaction). Let $\tau_{\text{now}} \in \mathbb{T}$ be the current audit timestamp. The semantic rule is temporally active iff $\text{Active}(rule, \tau_{\text{now}}) \triangleq (\tau_{\text{lb}} = \perp \vee \tau_{\text{lb}} \leq \tau_{\text{now}}) \wedge (\tau_{\text{ub}} = \perp \vee \tau_{\text{now}} \leq \tau_{\text{ub}})$. The scope vocabulary $\mathbb{S}$ is equipped with a partial order $\preceq$ defining restrictiveness (e.g., restricted $\preceq$ internal $\preceq$ partner $\preceq$ public). Metadata M_{meta} satisfies $rule$, denoted as $M_{meta} \vDash rule$, iff $\text{Active}(rule, \tau_{\text{now}})$ holds and:

$$M_{meta} \vDash rule \iff (\tau \in \mathcal{T}) \wedge (\tau_{\text{lb}} \neq \perp \Rightarrow \tau_{\text{lb}} \leq \tau_{\text{from}}) \wedge (\tau_{\text{ub}} \neq \perp \Rightarrow \tau_{\text{until}} \leq \tau_{\text{ub}}) \wedge (\exists \sigma' \in \mathcal{S} : \sigma \preceq \sigma') \wedge (\alpha \in \mathcal{A}). \quad (1)$$

All sub-conditions reduce to decidable finite-set membership or partial-order tests, ensuring deterministic, polynomial-time execution during the *Checkproof* phase.

Definition 3 (Semantic Rule Update State Machine). The dynamic lifecycle of a policy is governed by a deterministic finite state machine with states $\mathcal{Q} = \{\text{Active}, \text{Pending}, \text{Revoked}\}$ and transitions triggered via smart contracts:

1. **Update:** $\text{Active} \xrightarrow{\text{UpdateRule}(rule')} \text{Pending} \xrightarrow{\text{Record}(rule', RH')} \text{Active}'$. Triggered when the smart contract receives a reported approval count a_{KS} satisfying $a_{\text{KS}} \geq t$, together with a proposed rule $rule'$ whose version satisfies $v' = v + 1$. In the current prototype, the collection and authentication of individual KS-node approvals are abstracted as an off-chain governance process. The smart contract does not verify individual KS signatures or an aggregated threshold proof; it only applies the threshold-count condition before committing the rule hash $RH' = H_3(rule')$ to the on-chain registry $R[ID]$.
2. **Revoke:** $\text{Active} \xrightarrow{\text{Revoke}(ID, A)} \text{Revoked}$. Executed via an authorized transaction that inserts the user identity ID and its associated public-key component A into the denial set ζ_{deny} and removes the corresponding authorization record from the authorization set ζ_{allow}.

4.2 Proposed Scheme

Description of SecuAudit

The system master private key is collaboratively generated by m KS nodes in a distributed manner. No single KS node or small coalition of compromised nodes can reconstruct the complete key. SecuAudit is designed for MCP-enabled environments, where AI agents dynamically access heterogeneous external resources beyond their local security boundary. Each user's private key is jointly derived by the user and

at least t KS nodes through a two-round interactive protocol, where $t \le m$. During key generation, the user's identity is bound to an on blockchain registered semantic rule hash RH. The hash RH is a public policy handle. The corresponding semantic rule, denoted by $rule$, specifies access-control requirements and metadata constraints, such as permitted file types, validity periods, authorization scopes, and access permissions. The data owner uses the derived private key to generate authenticators for file blocks and outsources the file together with the authenticators to an external server. In the audit phase, an authorized verifier, such as an optional TPA, verifies data integrity through a lightweight random challenge and bilinear pairing operations without downloading the entire file. The verifier also checks the on blockchain tag, the returned semantic metadata, the latest semantic rule hash, and the corresponding semantic rule to determine whether the outsourced data satisfies the current metadata and access control constraints. SecuAudit maintains user revocation information through smart contracts and supports secure key updates and semantic rule updates.

Setup

The Setup phase initializes the public parameters and the distributed key infrastructure. The system defines a cyclic group G of large prime order q, a target group G_T, a symmetric bilinear pairing $e : G \times G \to G_T$, and a generator $g \in G$. The system also derives an independent public element $u \in G$, whose discrete logarithm with respect to g is unknown. It specifies four hash functions: $H_1 : \{0,1\}^* \times G \times \{0,1\}^* \to Z_q$, $H_2 : \{0,1\}^* \to Z_q$, $H_3 : \{0,1\}^* \to \{0,1\}^*$, and $H_G : \{0,1\}^* \to G$. The functions H_1, H_2, and H_G are modeled as random oracles. The function H_3 is used to derive semantic rule hashes. Each KS node i selects a random polynomial $f_i(x)$ of degree $t-1$ over Z_q and securely sends the share $f_i(SD_j)$ to each KS node j, where SD_j is a distinct nonzero node identifier. To prevent malicious KS nodes from distributing inconsistent or invalid shares, SecuAudit adopts a Feldman-type verifiable secret sharing mechanism during distributed key generation. Let $f_i(x) = c_{i,0} + c_{i,1}x + \cdots + c_{i,t-1}x^{t-1}$. Each KS node i publishes coefficient commitments $Com_{i,\ell} = g^{c_{i,\ell}}, \quad \ell = 0, \ldots, t-1$. After receiving the share $f_i(SD_j)$ from KS node i, node j verifies

$$g^{f_i(SD_j)} \stackrel{?}{=} \prod_{\ell=0}^{t-1} Com_{i,\ell}^{SD_j^{\ell}}. \tag{2}$$

If the equation does not hold, node j rejects the share and reports the invalid contribution. Only verified shares from non-disqualified KS nodes are used to compute the local private key share. After receiving all shares, KS node i computes its local private key share $\gamma_i = \sum_{j=1}^{m} f_j(SD_i) \bmod q$ and publishes $P_i = g^{\gamma_i}$. The system master private key is implicitly defined as $\sigma = \sum_{i=1}^{m} c_{i,0} \bmod q$. The system public key is $D = g^{\sigma}$. The Setup phase outputs $Params = \langle e, G, G_T, q, g, u, D, \{P_i\}_{i=1}^{m}, \{Com_{i,\ell}\}_{i\in[m],\ell\in[0,t-1]}, H_1, H_2, H_3, H_G \rangle$.

Keygen

The Keygen phase derives a user private key with the help of at least t KS nodes. The key is bound to the semantic rule registered on blockchain. A semantic rule is denoted by $rule = \langle type, validity, scope, access, version \rangle$. Here, $type$ specifies permitted file types, $validity$ defines the valid time interval, $scope$ denotes the authorized usage scope, $access$ describes access permissions, and $version$ records the rule version. The semantic rule hash is defined as $RH = H_3(rule)$. The user inputs its identity ID, obtains the current rule $rule_{cur}$ from the semantic rule registry, computes $RH = H_3(rule_{cur})$, selects $a \in Z_q$, computes $A = g^a$, and sends $\langle ID, A, RH \rangle$ to the KS nodes. Each KS node i retrieves the latest registered rule hash RH_{cur} associated with ID. If $RH \neq RH_{cur}$, or if ID or A appears in the denial registry, the request is rejected. Otherwise, KS node i selects $r_i \in Z_q$, computes $B_i = g^{r_i}$, derives $X_i = r_i + \gamma_i \cdot H_1(ID, A, RH) + H_2(0 \| ID \| A^{\gamma_i}) \bmod q$, and returns $\langle B_i, X_i \rangle$ to the user through an authenticated channel or with a KS node signature. The user first verifies the source of each KS response. The user then verifies the algebraic correctness of each response by checking

$$g^{X_i} \stackrel{?}{=} B_i \cdot P_i^{H_1(ID,A,RH)} \cdot g^{H_2(0\|ID\|P_i^a)}. \tag{3}$$

Invalid responses are discarded. For each valid response, the user computes $b_i = X_i - H_2(0\|ID\|P_i^a) \bmod q$. After collecting at least t valid shares from a KS node set S, the user computes $\rho_\xi = \prod_{j\in S, j\neq\xi} SD_j/(SD_j - SD_\xi) \bmod q$ and reconstructs $b = \sum_{\xi\in S} b_\xi \rho_\xi + a \bmod q$. The public verification component is $Z = (\prod_{\xi\in S} B_\xi^{\rho_\xi}) \cdot D^{H_1(ID,A,RH)} \cdot A$. The user obtains the private key b and the public information $\langle ID, A, Z, RH\rangle$.

Signblock

The Signblock phase generates authenticators and creates a rule-bound file tag before outsourcing. The data owner assigns a unique identifier fid to file F and divides the file into n blocks $(m_1, m_2, \ldots, m_n)$, where $m_j \in Z_q$. For each block index j, the data owner computes $U_j = H_G(2\|ID\|A\|fid\|j)$ and generates the authenticator $T_j = (U_j \cdot u^{m_j})^b$. The authenticator set is $\Phi = \{T_1, T_2, \ldots, T_n\}$. This construction binds each authenticator to the user identity, public key component, file identifier, and block index. The independent public element u prevents a server from converting an existing authenticator into a valid authenticator for a modified block by using only the public value $Z = g^b$. The data owner extracts semantic metadata M_{meta}, such as file type, validity period, authorization scope, and access attributes. The file tag is $tag = ID\|A\|fid\|name\|RH\|M_{meta}\|Z\|h_{tag}$, where $h_{tag} = H_2(1\|ID\|A\|fid\|name\|RH\|M_{meta}\|Z)$. The data owner signs the transaction through its blockchain account and uploads the tag to the smart contract. The smart contract checks the sender identity, the revocation status of A, the consistency between RH and the latest registered semantic rule hash, and the correctness of h_{tag}. If all checks pass, the tag is recorded on blockchain. The data owner sends $\langle F, \Phi\rangle$ to the external server.

Genproof

The Genproof protocol is executed by an authorized verifier and the external server. The verifier may be an optional TPA or another entity with access to the required public verification information. Given a challenge parameter c, the verifier randomly selects a subset of block indices $J \subseteq \{1, 2, \ldots, n\}$ with $|J| = c$. It also chooses a random coefficient $v_j \in Z_q$ for each $j \in J$. The challenge is $chal = \{(j, v_j)\}_{j\in J}$. The external server computes $\hat{F} = \sum_{j\in J} v_j m_j \bmod q$ and $T = \prod_{j\in J} T_j^{v_j}$. It also returns the stored semantic metadata M_{meta}. The proof is $proof = \langle fid, \hat{F}, T, M_{meta}\rangle$.It should be noted that the current construction focuses on public verifiability, integrity assurance, and semantic metadata compliance. The verifier only receives an aggregated challenge response $\hat{F}$ over randomly selected blocks rather than the complete outsourced file. Therefore, the auditing process avoids full data disclosure during verification and reduces unnecessary data exposure in secure data circulation scenarios. More advanced auditor-side data privacy, such as hiding the aggregated response from the verifier, can be further supported by standard random masking techniques and is left as an extensible enhancement of the proposed framework.

Checkproof

The Checkproof phase verifies data integrity and metadata compliance. After receiving $proof = \langle fid, \hat{F}, T, M_{meta}\rangle$, the verifier retrieves the on blockchain tag associated with fid. It checks whether A has been revoked. It recomputes $h_{tag} = H_2(1\|ID\|A\|fid\|name\|RH\|M_{meta}^{chain}\|Z)$ and compares it with the recorded tag hash. It also checks $M_{meta} = M_{meta}^{chain}$. For each challenged block index $j \in J$, the verifier computes $U_j = H_G(2\|ID\|A\|fid\|j)$. It then computes $U = \prod_{j\in J} U_j^{v_j}$ and $C = u^{\hat{F}}$. The integrity proof is accepted only if

$$e(T, g) \stackrel{?}{=} e(U \cdot C, Z). \tag{4}$$

The verifier retrieves the latest semantic rule $rule_{cur}$ and its hash RH_{cur} from the semantic rule registry. It checks $RH_{cur} = H_3(rule_{cur})$, $RH = RH_{cur}$, and $M_{meta}^{chain} \vDash rule_{cur}$. The proof is accepted only when the integrity equation holds, the metadata is consistent with the on blockchain tag, and the metadata satisfies the latest semantic rule.

Usrrevo

The Usrrevo protocol revokes a user's access privilege through the smart contract. Given a user identity ID and its public key component A, the smart contract checks whether the revocation request is issued by an authorized entity. If the request is valid, the contract inserts ID and A into the denial set ζ_{deny} and removes the corresponding authorization record from the authorization set ζ_{allow}. Subsequent Keygen, tag upload, and audit verification requests associated with the revoked identity or public key component are rejected.

Updaterule

The Updaterule protocol updates registered compliance rules using an approval count produced by an off-chain governance process. Given a new semantic rule $rule'$, the system computes $RH' = H_3(rule')$. In the current prototype, rule-update approvals are collected through an off-chain governance process, which reports the number a_{KS} of approving KS nodes to the smart contract. The contract performs a threshold-count check and accepts the update only if $a_{\mathrm{KS}} \geq t$. It does not verify individual KS-node signatures, signer uniqueness, or an aggregated threshold signature on-chain. If the reported count satisfies the threshold condition, the contract records $rule'$ and updates the current semantic rule hash to RH'. Therefore, the authenticity of the reported approval count is an explicit off-chain trust assumption in the current implementation. For existing outsourced files, SecuAudit adopts a strict rule-bound mode. If the rule hash recorded in an old tag differs from the latest rule hash RH_{cur}, the old tag is rejected during auditing. The data owner must refresh the file tag and regenerate the corresponding authenticators under the updated rule-bound key. This process does not require re-downloading or re-uploading the file content, but it does introduce a re-authentication cost for existing files.

The notations is summarized in Appendix A, while the detailed smart contract architecture and execution mechanics, correctness analysis and security analyses are provided in Appendices B–D, respectively.

5 Conclusion

In this paper, we proposed SecuAudit, a privacy-enhancing and decentralized data auditing framework strictly tailored to secure cross-domain data circulation within the MCP-enabled AI agent ecosystem. Addressing the inherent limitations of traditional bit-level integrity checkers, SecuAudit introduces a joint verification mechanism that cryptographically binds dynamic metadata, access control policies, and user identities directly to the underlying data content. By integrating a threshold-based distributed key management architecture with a lightweight challenge-response protocol and immutable smart contract registries, the system effectively mitigates single points of failure while supporting seamless semantic rule updates and dynamic user revocation.

Under the stated cryptographic assumptions and threat model, the formal analysis establishes resistance to the modeled data-forgery, metadata-tampering, and sub-threshold collusion attacks. Within our controlled implementation, SecuAudit exhibited approximately linear Keygen scaling with the number of key servers m and achieved lower Keygen latency than our reimplementation of Ped-TLARDA under the largest evaluated configurations. The local Ganache measurements and simulation-based analyses further indicate bounded blockchain and concurrency overhead under the selected parameters. These findings support the feasibility of SecuAudit within the evaluated settings, but they do not establish implementation-independent

superiority, production readiness, or independently reproduced performance. SecuAudit therefore provides a foundation for further investigation of metadata-compliance auditing in MCP-enabled AI-agent systems.

Although SecuAudit demonstrates effective metadata compliance auditing under the evaluated MCP scenarios, several limitations remain. First, the current implementation is evaluated mainly in a local Ethereum-compatible environment rather than on a production blockchain network. Therefore, the reported latency, gas consumption, and system stability may not fully reflect blockchain congestion, transaction confirmation delays, and network fluctuations in real-world deployments. Second, although the experiments evaluate configurations with up to 59 key servers, large-scale and geographically distributed deployments involving hundreds of key servers require further investigation, particularly with respect to communication overhead, node churn, fault tolerance, and availability. Third, the security guarantees rely on the defined threat model, especially the assumption that fewer than the threshold number of key servers collude. Attacks such as endpoint compromise, denial-of-service attacks, and side-channel leakage are not explicitly addressed in the current framework. Fourth, the current challenge-response protocol returns an unmasked linear aggregate of the challenged data blocks. Although a single audit does not directly disclose the individual blocks, an adaptive TPA may issue repeated correlated challenges and accumulate additional information about the challenged data. The current implementation therefore does not provide a formal confidentiality guarantee against unrestricted adaptive-query attacks. Incorporating verifier-independent challenge randomness, challenge-rate control, and randomized proof masking is left for future work. Fifth, the randomized challenge-response mechanism provides probabilistic rather than deterministic detection, meaning that a single audit may not detect extremely sparse data corruption. Finally, the current framework focuses on integrity and metadata compliance verification, while privacy-preserving rule evaluation over encrypted metadata remains an open direction for future research. In addition, the current semanticRule-Update prototype does not cryptographically verify individual KS-node approvals on-chain. It relies on an off-chain governance process to authenticate the approvers and report the approval count correctly. Implementing on-chain verification of unique KS signatures, together with nonce- and version-based replay protection, remains future work.

Acknowledgement: Not applicable.

Funding Statement: This work was supported in part by Natural Science Foundation of Henan (252300421879, 252300420987), in part by Key Technologies R&D Program of Henan Province (252102210213), in part by outstanding young science and technology talent project of Zhengzhou (42).

Author Contributions: The authors confirm contribution to the paper as follows: conceptualization, Yufa Shi, Jiaxing Hu and Lipeng Wang; methodology, Yufa Shi and Jiaxing Hu; software, Yufa Shi; validation, Yufa Shi, Jiaxing Hu and Mengyao Wang; formal analysis, Yufa Shi and Jiaxing Hu; investigation, Yufa Shi and Mengyao Wang; resources, Lipeng Wang, Rui Ma and Zhijuan Jia; data curation, Yufa Shi and Mengyao Wang; writing—original draft preparation, Yufa Shi; writing—review and editing, Jiaxing Hu, Lipeng Wang, Rui Ma, Zhijuan Jia and Mengyao Wang; visualization, Yufa Shi; supervision, Lipeng Wang, Rui Ma and Zhijuan Jia; project administration, Lipeng Wang and Rui Ma; funding acquisition, Lipeng Wang, Rui Ma and Zhijuan Jia. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the Corresponding Author, Lipeng Wang, upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A Notations

The main mathematical symbols and system parameters used throughout this paper are summarized in Table A1.

Table A1: Notations.

Symbol	Description
m	Number of KS nodes
t	Threshold value
Z	Public verification component
M_{meta}	Semantic metadata of outsourced data
fid	File identifier
n	Number of file blocks
m_j	The j-th file block
U_j	Index-binding public element for block j
T_j	Authenticator of block j
Φ	Set of authenticators
tag	File tag recorded on blockchain
c	Number of challenged blocks
J	Set of challenged block indices
v_j	Random challenge coefficient
$chal$	Challenge message
$\hat{F}$	Aggregated block value
T	Aggregated authenticator
C	Commitment to the aggregated block value
U	Aggregated index-binding element
ζ_{allow}	Authorization set
ζ_{deny}	Denial set
T_{exp}	Time for group exponentiation
T_{mul}	Time for group multiplication
T_{pair}	Time for bilinear pairing

Appendix B Smart Contract Architecture and Execution Mechanics

Algorithm A1: Smart contract execution for SecuAudit

Input: Transaction request Req, authorization set ζ_{allow}, denial registry ζ_{deny}, semantic rule registry R, and tag registry T.
Output: $true$ for success, $false$ otherwise.

1: // Semantic rule update.
2: **if** Req is a semantic rule update containing $\langle ID, rule', a_{\text{KS}} \rangle$ **then**
3: **if** $a_{\text{KS}} < t$ **then**
4: **return** false

(Continued)

Algorithm A1 (continued)

end if
$RH' \leftarrow H_3(rule')$
$R[ID] \leftarrow \langle rule', RH' \rangle$
Record the updated rule and rule hash on the blockchain
return true
end if
// Tag upload.
if Req is a tag upload containing $\langle ID, A, fid, name, RH, M_{meta}, Z, h_{tag} \rangle$ **then**
  if sender is not bound to ID **then return** $false$
  end if
  if $ID \in \zeta_{deny}$ or $A \in \zeta_{deny}$ **then return** $false$
  end if
  Retrieve $\langle rule_{cur}, RH_{cur} \rangle \leftarrow R[ID]$
  if $RH \neq RH_{cur}$ **then return** $false$
  end if
  if $h_{tag} \neq H_2(1 \| ID \| A \| fid \| name \| RH \| M_{meta} \| Z)$ **then return** $false$
  end if
  if M_{meta} does not satisfy $rule_{cur}$ **then return** $false$
  end if
  $T[fid] \leftarrow \langle ID, A, name, RH, M_{meta}, Z, h_{tag} \rangle$
  Record the tag on blockchain
  return $true$
end if
// User revocation.
if Req is a revocation request containing $\langle ID, A \rangle$ **then**
  if sender is not authorized to revoke ID or A **then return** $false$
  end if
  $\zeta_{deny} \leftarrow \zeta_{deny} \cup \{ID, A\}$
  $\zeta_{allow} \leftarrow \zeta_{allow} \backslash \{ID, A\}$
  Record the updated access control sets on blockchain
  return $true$
end if
return $false$

Note: Algorithm A1 abstracts the collection and authentication of KS-node approvals as an off-chain process. In the current implementation, a_{KS} is an externally reported approval count. The smart contract checks only whether this count reaches the threshold t and does not perform on-chain verification of individual KS signatures or an aggregated threshold proof.

To facilitate transparent and immutable enforcement within the MCP ecosystem, the SecuAudit smart contract is explicitly architected to optimize gas economics, transaction latency, and state consistency. The design rationale addressing the core operational workflows and failure handling is formalized below:

1. **Contract Design & Revocation Mechanism:** The architecture is built upon three core state mappings: the semantic rule registry $R[ID]$, the tag registry $T[fid]$, and the dual access control sets $\zeta_{allow}/\zeta_{deny}$. The user revocation mechanism leverages $\mathcal{O}(1)$ EVM `SLOAD` operations. Once an identity (ID) or

public key component (A) is flagged into ζ_{deny}, the Ethereum Virtual Machine's strict sequential execution intrinsically and atomically blocks any subsequent tag uploads or Keygen authorizations, ensuring zero race conditions during credential compromises.

2. **Rule-Update Workflow:** The current semanticRuleUpdate implementation applies an on-chain threshold-count guard to rule updates. The contract receives an approval count a_{KS} reported by the off-chain governance process and commits the new rule hash RH' to $R[ID]$ only when $a_{\mathrm{KS}} \geq t$. This mechanism is a deterministic integer comparison rather than an on-chain cryptographic verification of individual KS approvals or a mathematical consensus proof. Accordingly, the current prototype assumes that the off-chain governance process correctly authenticates and counts the approving KS nodes.
3. **Failure Handling and Guard Ordering:** The contract evaluates authorization and revocation conditions before rule-consistency checks and before state-mutating `SSTORE` operations. Consequently, a request that fails an earlier validation guard is reverted without executing the subsequent contract logic or performing state updates. This ordering is intended to avoid unnecessary execution for invalid requests. However, the present evaluation does not measure the Gas consumption of reverted transactions or compare it with a separate baseline implementation; therefore, no quantitative Gas-saving claim is made for rejected transactions.
4. **Transaction Latency & Auditability:** To minimize read-latency for the off-chain TPA, the contract relies on an event-driven design. By emitting structured Solidity events (`RuleUpdated`, `TagRegistered`, `UserRevoked`), the MCP Server and TPA can asynchronously track the compliance state via rapid RPC node queries, circumventing the latency of direct on-chain state reads during real-time data circulation. Gas consumption for the evaluated successful contract operations is reported in Section 2.5. Rejected-transaction Gas costs and confirmation latency on a public blockchain were not directly measured.

Appendix C Correctness Analysis

We first show the correctness of the verifiable share distribution in Setup. For each KS node i, the polynomial is $f_i(x) = \sum_{\ell=0}^{t-1} c_{i,\ell} x^{\ell}$. Since $Com_{i,\ell} = g^{c_{i,\ell}}$, for any valid share $f_i(SD_j)$, we have

$$\begin{aligned} g^{f_i(SD_j)} &= g^{\sum_{\ell=0}^{t-1} c_{i,\ell} SD_j^{\ell}} \\ &= \prod_{\ell=0}^{t-1} g^{c_{i,\ell} SD_j^{\ell}} \\ &= \prod_{\ell=0}^{t-1} Com_{i,\ell}^{SD_j^{\ell}}. \end{aligned} \tag{A1}$$

Thus, any honestly generated share passes the verification Eq. (2), while invalid shares can be detected before being used in the aggregate key generation process.

Then, we define the system-wide aggregate polynomial as $G(x) = f_1(x) + f_2(x) + \cdots + f_m(x)$, where $f_i(x)$ is the random polynomial selected by the i-th KS node. According to the **Setup** step, the system master private key is the sum of the constant terms of all KS nodes' polynomials, that is, $\sigma = \sum_{i=1}^{m} c_{i,0} \bmod q = G(0)$. The share held by each KS node ξ is the evaluation of this aggregate polynomial at its node identifier SD_ξ, that is, $\gamma_\xi = \sum_{j=1}^{m} f_j(SD_\xi) \bmod q = G(SD_\xi)$.

Since $G(x)$ is a polynomial of degree $t-1$, the Lagrange interpolation theorem shows that any t valid point pairs are sufficient to reconstruct its value at $x = 0$. For a valid KS node set S with $|S| = t$, the value of $G(0)$ can be written as $G(0) = \sum_{\xi \in S} G(SD_\xi) \cdot \rho_\xi \bmod q$, where $\rho_\xi = \prod_{j \in S, j \neq \xi} SD_j / (SD_j - SD_\xi) \bmod q$ is the Lagrange coefficient at $x = 0$. Substituting $\gamma_\xi = G(SD_\xi)$ into the equation gives $\sum_{\xi \in S} \gamma_\xi \cdot \rho_\xi = G(0) = \sigma$.

This equality shows that the system master private key σ can be reconstructed through a linear combination of at least t valid KS shares and their corresponding Lagrange coefficients.

Next, we verify the correctness of Eq. (3) in the **Keygen** step. Let $h = H_1(ID, A, RH)$. Since $X_i = r_i + \gamma_i \cdot h + H_2(0\|ID\|A^{\gamma_i}) \bmod q$, $B_i = g^{r_i}$, $P_i = g^{\gamma_i}$, and $A = g^a$, we have $A^{\gamma_i} = P_i^a$. It follows that:

$$\begin{aligned} g^{X_i} &= g^{r_i+\gamma_i \cdot H_1(ID,A,RH)+H_2(0\|ID\|A^{\gamma_i})} \\ &= g^{r_i+\gamma_i \cdot H_1(ID,A,RH)+H_2(0\|ID\|P_i^a)} \\ &= g^{r_i} \cdot g^{\gamma_i \cdot H_1(ID,A,RH)} \cdot g^{H_2(0\|ID\|P_i^a)} \\ &= B_i \cdot P_i^{H_1(ID,A,RH)} \cdot g^{H_2(0\|ID\|P_i^a)}. \end{aligned} \tag{A2}$$

Thus, Eq. (3) holds for every honest KS node.

We then show that the public verification component Z is consistent with the reconstructed private key b. For each valid response, the user computes $b_i = X_i - H_2(0\|ID\|P_i^a) \bmod q$. By substituting the definition of X_i, we obtain $b_i = r_i + \gamma_i H_1(ID, A, RH)$. Therefore:

$$\begin{aligned} b &= \sum_{\xi \in S} b_\xi \cdot \rho_\xi + a \\ &= \sum_{\xi \in S} \left(r_\xi + \gamma_\xi H_1(ID, A, RH)\right) \rho_\xi + a \\ &= \sum_{\xi \in S} r_\xi \rho_\xi + \left(\sum_{\xi \in S} \gamma_\xi \rho_\xi\right) H_1(ID, A, RH) + a \\ &= \sum_{\xi \in S} r_\xi \rho_\xi + \sigma H_1(ID, A, RH) + a. \end{aligned} \tag{A3}$$

Since $B_\xi = g^{r_\xi}$, $D = g^\sigma$, and $A = g^a$, we have:

$$\begin{aligned} Z &= \left(\prod_{\xi \in S} B_\xi^{\rho_\xi}\right) \cdot D^{H_1(ID,A,RH)} \cdot A \\ &= g^{\sum_{\xi \in S} r_\xi \rho_\xi} \cdot g^{\sigma H_1(ID,A,RH)} \cdot g^a \\ &= g^b. \end{aligned} \tag{A4}$$

Finally, we verify the correctness of Eq. (4) in the **Checkproof** step. For each challenged block index $j \in J$, the authenticator is $T_j = (U_j \cdot u^{m_j})^b$, where $U_j = H_G(2\|ID\|A\|fid\|j)$. The external server computes $T = \prod_{j \in J} T_j^{v_j}$ and $\hat{F} = \sum_{j \in J} v_j m_j \bmod q$. The verifier computes $U = \prod_{j \in J} U_j^{v_j}$ and $C = u^{\hat{F}}$. It follows that:

$$\begin{aligned} T &= \prod_{j \in J} T_j^{v_j} \\ &= \prod_{j \in J} (U_j \cdot u^{m_j})^{b v_j} \\ &= \left(\prod_{j \in J} U_j^{v_j} \cdot u^{\sum_{j \in J} v_j m_j}\right)^b \\ &= (U \cdot C)^b. \end{aligned} \tag{A5}$$

Since $Z = g^b$ and e is a bilinear pairing, we have:

$$\begin{aligned} e(T, g) &= e((U \cdot C)^b, g) \\ &= e(U \cdot C, g^b) \\ &= e(U \cdot C, Z). \end{aligned} \tag{A6}$$

Thus, Eq. (4) holds for honestly generated proofs. The metadata compliance check is independent of the pairing equation. It ensures that the verified data also satisfies the current metadata and access control constraints.

Appendix D Security Analyses

Theorem A1 (Auditing Soundness): *In SecuAudit, when the outsourced resource stored on the external server is corrupted or inconsistent with the designated metadata and the latest semantic rule, the external server is unable to produce a legitimate proof that satisfies the verifier's validation process, except with negligible probability.*

Proof: Assume that the CDH assumption holds in group G, the hash functions are collision-resistant, H_G is modeled as a random oracle into G, and the smart contract correctly maintains the tag registry, revocation list, and semantic rule registry. Then SecuAudit achieves auditing soundness and metadata compliance against any PPT adversary A_1 controlling the external server. We prove the theorem through a sequence of games. Let $\mathrm{Adv}_{A_1}^{\mathrm{Game}\,i}$ denote the advantage of A_1 in making the verifier accept an invalid proof in Game i. Let q_H be the maximum number of hash queries and l be the hash output length. □

Game 0: This game is the real attack game. The challenger C honestly executes $Setup$, $Keygen$, and $Signblock$. The adversary A_1 controls the external server. After receiving a challenge $chal = \{(j, v_j)\}_{j \in J}$, it returns a proof $proof^* = \langle fid, \hat{F}^*, T^*, M^*_{meta} \rangle$. The verifier computes $C^* = u^{\hat{F}^*}$ and $U = \prod_{j \in J} U_j^{v_j}$, where $U_j = H_G(2 \| ID \| A \| fid \| j)$.

Analysis: The verifier accepts only if the following condition holds: $Accept = 1 \Longleftrightarrow [e(T^*, g) = e(U \cdot C^*, Z)] \wedge \left[M^*_{meta} = M^{chain}_{meta}\right] \wedge [RH = RH_{cur}] \wedge [RH_{cur} = H_3(rule_{cur})] \wedge \left[M^{chain}_{meta} \vDash rule_{cur}\right]$. This game defines the real-world advantage as $\mathrm{Adv}_{A_1}^{\mathrm{Game}\ 0} = \varepsilon_0$.

Game 1: This game is identical to Game 0, except that the challenger maintains query lists for H_2, H_3, and H_G. If A_1 produces a collision related to h_{tag}, RH, or a block-binding value U_j, the challenger aborts.

Analysis: By the birthday bound, the probability of such a collision is bounded by $\left|\mathrm{Adv}_{A_1}^{\mathrm{Game}\ 1} - \mathrm{Adv}_{A_1}^{\mathrm{Game}\ 0}\right| \leq \frac{q_H^2}{2^{l+1}}$.

Game 2: This game is identical to Game 1, except that the challenger aborts if A_1 outputs a valid authenticator for a new block-index base that was not honestly generated in the $Signblock$ phase.

Analysis: For each block j, the authenticator is $T_j = (U_j \cdot u^{m_j})^b$, where $U_j = H_G(2 \| ID \| A \| fid \| j)$ and $Z = g^b$. Since U_j is modeled as a random group element and u is independent of g, forging a valid authenticator for a new base is equivalent to producing a BLS-type signature under public key Z. If A_1 can forge such an authenticator with non-negligible probability, then a simulator B can solve a CDH instance $\langle g, g^x, g^y \rangle$ by setting $Z = g^x$ and programming the random oracle so that the target base equals g^y. A successful forgery gives $(g^y)^x = g^{xy}$, which solves CDH. Therefore, $\left|\mathrm{Adv}_{A_1}^{\mathrm{Game}\ 2} - \mathrm{Adv}_{A_1}^{\mathrm{Game}\ 1}\right| \leq \mathrm{Adv}_B^{CDH}$.

Game 3: This game is identical to Game 2, except that we consider the case where A_1 does not correctly store some challenged file blocks but still attempts to pass the integrity verification in Eq. (4).

Analysis: For an honest proof, the aggregated authenticator satisfies $T = (U \cdot C)^b$, where $U = \prod_{j\in J} U_j^{v_j}$ and $C = u^{\hat{F}}$ with $\hat{F} = \sum_{j\in J} v_j m_j \bmod q$. If A_1 returns an incorrect aggregate $\hat{F}^* \neq \hat{F}$, then $C^* = u^{\hat{F}^*}$ and the verifier accepts only if $e(T^*, g) = e(U \cdot C^*, Z)$. This means that A_1 has produced a valid authenticator for the modified aggregate base $U \cdot C^*$. By Game 2, this event is bounded by the CDH-based unforgeability of the authenticator. If the server has lost s out of n blocks, the probability that a random challenge of size c avoids all missing blocks is $\frac{\binom{n-s}{c}}{\binom{n}{c}}$. If the challenge hits at least one missing block, the value $\hat{F}$ contains at least one unknown random coefficient and cannot be computed except with probability at most $1/q$. Hence, $\Pr[A_1 \text{ passes Game 3}] \leq \frac{\binom{n-s}{c}}{\binom{n}{c}} + \left(1 - \frac{\binom{n-s}{c}}{\binom{n}{c}}\right)\frac{1}{q} + \mathrm{Adv}_B^{CDH}$. Thus, $\left|\mathrm{Adv}_{A_1}^{\text{Game 3}} - \mathrm{Adv}_{A_1}^{\text{Game 2}}\right| \leq \frac{\binom{n-s}{c}}{\binom{n}{c}} + \frac{1}{q} + \mathrm{Adv}_B^{CDH}$.

Game 4: This game is identical to Game 3, except that we consider the case where A_1 tampers with the returned semantic metadata or attempts to pass the metadata compliance check under an outdated or invalid semantic rule.

Analysis: During $Signblock$, the on blockchain tag contains M_{meta}^{chain} and the tag hash $h_{tag}^{chain} = H_2(1\|ID\|A\|fid\|name\|RH\|M_{meta}^{chain}\|Z)$. During $Checkproof$, the verifier checks $M_{meta}^* = M_{meta}^{chain}$, $RH = RH_{cur}$, $RH_{cur} = H_3(rule_{cur})$, $M_{meta}^{chain} \vDash rule_{cur}$. If A_1 returns $M_{meta}^* \neq M_{meta}^{chain}$ and still passes the tag verification, then it must either find a collision in H_2 or modify the on blockchain tag state. If A_1 tries to make outdated metadata pass the semantic rule check, then it must either forge the latest rule hash RH_{cur}, find a collision in H_3, or bypass the rule registry maintained by the smart contract. Thus, $\Pr[M_{meta}^* \neq M_{meta}^{chain} \wedge VerifyTag = 1] \leq \varepsilon_{hash} + \varepsilon_{sc}$, where ε_{hash} denotes the probability of breaking hash collision resistance and ε_{sc} denotes the probability of bypassing the smart contract state. Therefore, $\left|\mathrm{Adv}_{A_1}^{\text{Game 4}} - \mathrm{Adv}_{A_1}^{\text{Game 3}}\right| \leq \varepsilon_{hash} + \varepsilon_{sc}$.

Thus, Combining the above games, the real-world advantage of A_1 is bounded by

$$\varepsilon_0 \leq \frac{q_H^2}{2^{l+1}} + 2 \cdot \mathrm{Adv}_B^{CDH} + \frac{\binom{n-s}{c}}{\binom{n}{c}} + \frac{1}{q} + \varepsilon_{hash} + \varepsilon_{sc}. \tag{A7}$$

If the challenge size c is properly selected, the term $\binom{n-s}{c}/\binom{n}{c}$ becomes small for any non-negligible data loss. All other terms are negligible under the stated assumptions. Therefore, SecuAudit achieves auditing soundness and metadata compliance against A_1. Theorem A1 is proven.

Theorem A2 (Collusion Resistance): *In SecuAudit, when fewer than t KS nodes collude with malicious users, they cannot reconstruct the system master private key σ, derive an unauthorized user private key b, forge a valid verification component Z, or bind a file tag to a semantic rule hash different from the currently registered on-chain rule hash, except with negligible probability.*

Proof: Let A_2 be an adversary that controls $k \leq t-1 KS$ nodes and colludes with a malicious user. Let $\mathrm{Adv}_{A_2}^{\text{Game } i}$ denote its advantage in recovering secret values, generating unauthorized keys, or bypassing semantic rule binding. Assume that the underlying (t, m) Feldman-type verifiable threshold polynomial scheme is secure under the Discrete Logarithm assumption, that the DL assumption holds in group G, that KS responses are properly authenticated, that the verifiable share distribution is correctly enforced, and that the smart contract correctly enforces the registered rule and file tag consistency checks. Then SecuAudit is resistant to collusion attacks launched by A_2. □

Game 0: The challenger C simulates the complete protocol. The adversary A_2 attempts to recover the system master private key σ, derive an unauthorized user private key b, output a valid public verification

component Z^* without enough valid KS responses, or bind a key or file tag to a forged semantic rule hash RH^*.

Analysis: This game defines the real collusion attack. The initial advantage is denoted as $\mathrm{Adv}_{A_2}^{\mathrm{Game}\ 0}$.

Game 1: This game is identical to Game 0, except that A_2 statically compromises k KS nodes and obtains their private key shares.

Analysis: Let the aggregate polynomial be $G(x) = \sum_{i=1}^{m} f_i(x) \bmod q$. The system master private key is $\sigma = G(0)$. Each compromised KS node gives one evaluation point $G(SD_i)$. Since $G(x)$ has degree $t-1$, any set of fewer than t shares gives no information about $G(0)$. Thus, $H(\sigma \mid \{G(SD_i)\}_{i=1}^{k}) = H(\sigma), \quad k < t$. Equivalently, the best probability of reconstructing σ from fewer than t shares is $\Pr[A_2 \text{ reconstructs } \sigma \mid k < t] = \frac{1}{q}$. Hence, $\mathrm{Adv}_{A_2}^{\mathrm{Game}\ 1} = \mathrm{Adv}_{A_2}^{\mathrm{Game}\ 0}$. Moreover, due to the verifiable share distribution mechanism in Setup, malformed shares submitted by malicious KS nodes are detected and excluded before the aggregate shares are formed. Therefore, a coalition of fewer than t KS nodes cannot bias the reconstructed master key or provide enough valid shares for unauthorized key generation.

Game 2: This game is identical to Game 1, except that we consider whether A_2 can derive secret scalars from public group elements, such as $D = g^{\sigma}$ and $P_i = g^{\gamma_i}$.

Analysis: Recovering σ from $D = g^{\sigma}$ or recovering γ_i from $P_i = g^{\gamma_i}$ is equivalent to solving the DL problem in group G. Thus, $\left|\mathrm{Adv}_{A_2}^{\mathrm{Game}\ 2} - \mathrm{Adv}_{A_2}^{\mathrm{Game}\ 1}\right| \le \mathrm{Adv}_{B}^{DL}$.

Game 3: This game is identical to Game 2, except that we examine whether A_2 can obtain a rule-bound key for a forged or outdated semantic rule hash RH^*.

Analysis: In $Keygen$, each honest KS node retrieves the latest registered semantic rule hash RH_{cur} and rejects the request if $RH^* \neq RH_{cur}$. Since A_2 controls at most $t-1$ KS nodes, it cannot collect t authenticated valid responses for an unregistered RH^*. If it tries to forge a response for an honest KS node, it must produce a response accepted as coming from that KS node and satisfying Eq. (3). Under authenticated KS channels or KS signatures, impersonating an honest KS node succeeds only with probability ε_{auth}. If it tries to recover the honest KS node's private share γ_i from $P_i = g^{\gamma_i}$, its success is bounded by the DL assumption. Therefore, $\Pr[A_2 \text{ obtains } t \text{ valid responses for } RH^* \neq RH_{cur}] \le \mathrm{Adv}_{B}^{DL} + \varepsilon_{auth} + \varepsilon_{sc}$.

Game 4: This game is identical to Game 3, except that we examine whether $\mathcal{A}_2$ can bind a file tag to a semantic rule other than the currently registered rule while still passing the Signblock verification.

Analysis: During Signblock, the smart contract retrieves the current on-chain semantic rule hash RH_{cur} and accepts the file tag only if the submitted rule hash satisfies $RH^* = RH_{\mathrm{cur}}$. If $\mathcal{A}_2$ submits a different rule hash $RH^* \neq RH_{\mathrm{cur}}$, acceptance requires bypassing the smart-contract consistency check. Alternatively, if $\mathcal{A}_2$ attempts to use a different semantic rule $rule^* \neq rule_{\mathrm{cur}}$ while preserving the same registered hash, it must find a collision such that $H_3(rule^*) = H_3(rule_{\mathrm{cur}})$. Therefore, $\Pr[\mathcal{A}_2 \text{ binds a tag to an unregistered semantic rule}] \le \varepsilon_{\mathrm{sc}} + \varepsilon_{\mathrm{hash}}$. This game proves only the consistency between the file tag and the currently registered on-chain semantic rule hash. It does not prove the authenticity of the off-chain approval process used to update the rule registry. In the current prototype, the smart contract only checks whether the externally reported approval count satisfies $a_{\mathrm{KS}} \ge t$ and does not verify individual or aggregate KS-node signatures. Therefore, the correctness of the reported approval count is treated as an explicit off-chain trust assumption.

Thus, Combining the above games, the advantage of A_2 is bounded by

$$\mathrm{Adv}_{A_2} \le \frac{1}{q} + 2 \cdot \mathrm{Adv}_{B}^{DL} + \epsilon_{auth} + \epsilon_{hash} + 2\varepsilon_{sc}. \tag{A8}$$

The term $1/q$ is negligible for large prime order q. The DL advantage, hash collision probability, authentication failure probability, and smart contract bypass probability are negligible under the stated assumptions. Therefore, SecuAudit resists collusion by up to $t-1$ malicious KS nodes and a malicious user with respect to master-key reconstruction, unauthorized key generation, and forged file-tag binding. The authenticity of the externally reported rule-update approval count is outside the cryptographic guarantee of this theorem.

MCP-Specific Threat Analysis.

Beyond the formal security guarantees established in Theorems A1 and A2 against adversaries A_1 and A_2, we provide a complementary analysis of the MCP-specific adversaries A_3–A_5 defined in Section 4.1.2 Although full game-based security proofs for these adversaries would require modeling MCP protocol semantics at a level of detail beyond the scope of this paper, we provide rigorous security arguments grounded in the cryptographic and system-level mechanisms of SecuAudit:

1. **Against Malicious MCP Client (A_3):** The prompt-injection attack surface is constrained by three independent security boundaries. First, the KS nodes verify RH against the on-chain registry during Keygen, and each KS response is authenticated with a verifiable signature. The MCP Client cannot forge valid KS responses because doing so would require either breaking the DL assumption (to recover γ_i from P_i), or compromising at least t KS nodes. Second, the smart contract enforces a strict validation sequence (Algorithm A1) that includes sender identity, revocation status, RH consistency, and h_{tag} correctness. This process relies on the deterministic execution of the EVM and cannot be bypassed through client-side prompt manipulation. Third, even if a compromised client successfully registers a resource, subsequent audits by an honest TPA will detect any metadata inconsistency through the Checkproof phase. Therefore, the maximum damage achievable by A_3 is limited to wasting computational resources through spurious registration attempts, which are inherently rate-limited by gas costs on the blockchain.
   Second, the smart contract enforces a strict validation sequence (Algorithm A1) that includes sender identity, revocation status, RH consistency, and h_{tag} correctness. This process relies on the deterministic execution of the EVM and cannot be bypassed through client-side prompt manipulation.
2. **Against Compromised MCP Connector (A_4):** Under the A_4 threat model, the External Server and its stored resource state remain uncompromised, while the adversary controls only the MCP Connector and the messages forwarded through it. Therefore, A_4 cannot independently construct a fresh valid proof because proof generation requires the authenticated data blocks and authenticators stored by the External Server. For a stale-response replay attack, the verifier retrieves the currently registered file tag and semantic rule from the blockchain. A replayed response containing $M_{\text{meta}}^{\text{old}}$ is rejected if $M_{\text{meta}}^{\text{old}} \neq M_{\text{meta}}^{\text{chain}}$ or if $M_{\text{meta}}^{\text{old}} \neq rule_{\text{cur}}$. Thus, replayed metadata that is inconsistent with the current registered resource state or semantic rule cannot be accepted. For in-transit metadata or provenance manipulation, the verifier recomputes $h_{\text{tag}} = H_2(1 \parallel ID \parallel A \parallel fid \parallel name \parallel RH \parallel M_{\text{meta}}^{\text{chain}} \parallel Z)$ and checks the returned metadata against $M_{\text{meta}}^{\text{chain}}$. Any unauthorized modification of fid, $name$, RH, or M_{meta} therefore causes either a tag-hash mismatch or a metadata-equality-check failure. Challenge redirection is prevented because each block authenticator is bound to the user identity, public-key component, file identifier, and block index through $U_j = H_G(2 \parallel ID \parallel A \parallel fid \parallel j)$. Redirecting a challenge to another resource consequently invalidates the pairing verification. Selective message suppression cannot cause an invalid proof to be accepted. Instead, it results in an observable timeout or missing audit response and is treated as a denial-of-audit condition. When audit requests and results are recorded in the blockchain audit log, such suppression also produces a visible gap in the audit sequence.

3. **Against Malicious TPA ($\mathcal{A}_5$):** For the data-exfiltration threat, each audit reveals one aggregated value $\hat{F} = \sum_{j \in J} v_j m_j \in \mathbb{Z}_q$ over the challenged blocks, rather than directly revealing individual data blocks or individual authenticators. Nevertheless, because the returned value is a linear aggregate, an adaptive TPA that is allowed to issue repeated and correlated challenges over the same block set may accumulate multiple linear equations about the challenged data. Therefore, the current prototype does not claim a formal confidentiality guarantee against unrestricted adaptive-query attacks. Verifier-independent challenge randomness, challenge-rate control, and randomized proof-masking mechanisms are left for future work. For the false-reporting threat, the public reproducibility of the pairing verification ensures that any entity with access to the on-chain tag and the published proof can independently verify the TPA's reported result. A discrepancy between the TPA's report and the publicly recomputable verification result provides evidence of TPA misbehavior.

Together with Theorems A1 and A2 and the attack-driven evaluation in Section 2.5, this analysis indicates that SecuAudit provides a multi-layered defense against the modeled threats, subject to the stated assumptions and the adaptive-TPA limitation discussed above. The six attack cases are designed to cover representative attack categories corresponding to the defined threat model, including integrity violation, metadata manipulation, replay, unauthorized update, and collusion attacks. They are not intended to exhaustively enumerate all possible adaptive adversarial strategies. A formal evaluation against fully adaptive adversaries is beyond the scope of this work and will be investigated in future research.

References

1. Hou X, Zhao Y, Wang S, Wang H. Model context protocol (MCP): landscape, security threats, and future research directions. ACM Trans Softw Eng Methodol. 2026;3901–15. doi:10.1145/3796519.
2. Anbiaee Z, Rabbani M, Mirani M, Piya G, Opushnyev I, Ghorbani A, et al. Security threat modeling for emerging AI-agent protocols: a comparative analysis of MCP, A2A, Agora, and AN. arXiv:2602.11327. 2026.
3. Zhao W, Liu J, Ruan B, Li S, Liang Z. When MCP servers attack: taxonomy, feasibility, and mitigation. arXiv:2509.24272. 2025.
4. de Witt CS, Krawiecka K, Krawczuk I, Hagag B, Anderson WL, Belcak P, et al. Open challenges in multi-agent security: towards secure systems of interacting AI agents. arXiv:2505.02077. 2025.
5. dos Santos Filho EB. ESAA-Security: an event-sourced, verifiable architecture for agent-assisted security audits of AI-generated code. arXiv:2603.06365. 2026.
6. Wang L, Hu M, Yang LT, Sun X, Chen Z. AI-auditor: a data auditing framework for enhancing the trustworthiness of AI models. IEEE Trans Ind Inform. 2025;21(12):9208–16.
7. Han G, Li H. Sec-auditor: a blockchain-based data auditing solution for ensuring integrity and semantic correctness. Comput Mater Contin. 2024;80(2):2121–37.
8. Lei H, Wang XA, Liu W, Wu L, Zhang C, Jiang W, et al. An improved blockchain-based cloud auditing scheme using dynamic aggregate signatures. Comput Mater Contin. 2026;86(2):1–32. doi:10.32604/cmc.2025.070030.
9. Radosevich B, Halloran J. MCP safety audit: LLMs with the model context protocol allow major security exploits. arXiv:2504.03767. 2025.
10. Zhao M, Chen H. Identity-based provable data possession with designated verifier from lattices for cloud computing. Entropy. 2025;27(7):753–73. doi:10.3390/e27070753.
11. Miao Y, Huang Q, Xiao M, Susilo W. Blockchain assisted multi-copy provable data possession with faults localization in multi-cloud storage. IEEE Trans Inf Forensics Secur. 2022;17:3663–76. doi:10.1109/tifs.2022.3211642.
12. Alharby M. Preserving data secrecy and integrity for cloud storage using smart contracts and cryptographic primitives. Comput Mater Contin. 2024;79(2):2449. doi:10.32604/cmc.2024.050425.
13. Asmuth C, Bloom J. A modular approach to key safeguarding. IEEE Trans Inf Theory. 1983;29(2):208–10. doi:10.1109/tit.1983.1056651.

14. Wang L, Hu M, Jia Z, Gong B, Zhang J. A chinese remainder theorem-based signature scheme for blockchain voting scenarios. J Comput Appl Res. 2020;37(2):538–43. (In Chinese).
15. Sarkar S, Kisku B, Misra S, Obaidat MS. Chinese remainder theorem-based RSA-threshold cryptography in MANET using verifiable secret sharing scheme. In: Proceedings of the 2009 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications; 2009 Oct 12–14; Marrakech, Morocco. p. 258–62.
16. Shamir A. How to share a secret. Commun ACM. 1979;22(11):612–3. doi:10.1145/359168.359176.
17. Feldman P. A practical scheme for non-interactive verifiable secret sharing. In: Proceedings of the 28th Annual Symposium on Foundations of Computer Science (sfcs 1987); 1987 Oct 12–14; Los Angeles, CA, USA. p. 427–38.
18. Kasser D. An improvement upon the bounds for the local leakage resilience of shamir's secret sharing scheme. In: Theory of Cryptography Conference. Berlin/Heidelberg, Germany: Springer; 2025. p. 395–422.
19. Baghery K, Knapen N, Nicolas G, Rahimi M. Pre-constructed publicly verifiable secret sharing and applications. In: International Conference on Applied Cryptography and Network Security. Berlin/Heidelberg, Germany: Springer; 2025. p. 89–119.
20. Wang L, Gao J, Li Q, Chen Z. A multi-receiver multi-message signcryption scheme based on blockchain. J Softw. 2021;32(11):3606–27. (In Chinese). doi:10.3934/mbe.2023806.
21. Shen J, Zeng P, Choo KKR, Li C. A certificateless provable data possession scheme for cloud-based EHRs. IEEE Trans Inf Forensics Secur. 2023;18:1156–68. doi:10.1109/tifs.2023.3236451.
22. Wu Y, Tan X, Xie Q. Certificateless provable data possession scheme for cloud-based electronic health records system. Mathematics. 2024;12(24):3883–906. doi:10.3390/math12243883.
23. Wang M, Yu J, Shen W, Hao R. Privacy-preserving time-based auditing for secure cloud storage. IEEE Trans Inf Forensics Secur. 2024;19:7866–78. doi:10.1109/tifs.2024.3449095.
24. Chen H, Tao Z, Wang Z, Liu X. Merkle multi-branch hash tree-based dynamic data integrity auditing for B5G network cloud storage. J Inf Secur Appl. 2025;89(1):103981. doi:10.1016/j.jisa.2025.103981.
25. Das S, Priyadarshini R, Mishra M, Barik RK. Leveraging towards access control, identity management, and data integrity verification mechanisms in blockchain-assisted cloud environments: a comparative study. J Cybersecur Priv. 2024;4(4):1018–43. doi:10.3390/jcp4040047.
26. Wang K, Wu Q, Han T, Luo D, Deng H, Qin B, et al. A blockchain-based publicly verifiable data access control scheme without pairing. Comput Electr Eng. 2024;120(2):109724. doi:10.1016/j.compeleceng.2024.109724.
27. Simmons JC, Winograd JM. Interoperable provenance authentication of broadcast media using open standards-based metadata, watermarking and cryptography. arXiv:2405.12336. 2024.
28. Liu K, Ning J, Wu P, Xu S, Chen R. TLARDA: threshold label-aggregating remote data auditing in decentralized environment. IEEE Trans Inf Forensics Secur. 2025;20:3146–60.
29. Liu Y, Jia Y, Geng R, Jia J, Gong NZ. Formalizing and benchmarking prompt injection attacks and defenses. In: Proceedings of the 33rd USENIX Security Symposium (USENIX Security 24); 2024 Aug 14–16; Philadelphia, PA, USA. p. 1831–47.
30. Debenedetti E, Zhang J, Balunovic M, Beurer-Kellner L, Fischer M, Tramèr F. AgentDojo: a dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. Adv Neural Inf Process Syst. 2024;37:82895–920.
31. Hu Y, Fan C, Samyoun S, Du J. Log-To-Leak: prompt injection attacks on tool-using LLM agents via model context protocol. 2026 [cited 2026 May 1]. Available from: https://openreview.net/forum?id=UVgbFuXPaO.