Available online at website: https://jurnal.iaii.or.id/index.php/RESTI

JURNAL RESTI
(Rekayasa Sistem dan Teknologi Informasi)
Vol. 10 No. 4 (2026) 1099 - 1106 e-ISSN: 2580-0760

Efficient Assured Cloud Data Deletion and Verification Scheme Using AES-XOR Encryption and Merkle Trees

Huda Sharaf Eldin Shakur [1], Baban Ahmed Mahmood[2]
[1]Department of Computer Science, College of Computer Science and Information Technology, Kirkuk University, Kirkuk, Iraq
[2]Department of Information Technology, College of Computer Science and Information Technology, Kirkuk University, Kirkuk, Iraq

[*]*stcm23004@uokirkuk.edu.iq*

Abstract

With the increasing interest of cloud computing users in storing and processing their data, the need for secure and permanent data deletion has grown. This research paper proposes a practical, evaluable, and testable mechanism for verifying data integrity and ensuring its deletion. The proposed system integrates sequential XOR technology with AES-based block-level encryption, thereby achieving strong interconnection between encrypted blocks. This prevents block separation, decryption, or tampering during transmission and storage. The encryption key is divided into independent parts, the purpose of which is to prevent any single party from possessing the complete encryption key. This ensures that the file cannot be recovered if any part of the key is deleted. The encrypted blocks are then arranged with one of the key parts in a Merkel tree structure to guarantee data integrity and verifiability during storage and retrieval. Through experimentation and providing similar levels of confidentiality and data recovery, the experimental results showed that this mechanism achieves competitive computing efficiency with reasonable storage and communication costs compared to basic methods. This solution was implemented on a real cloud platform within the Microsoft Azure environment, proving its feasibility, security and reliability without the need for major modifications to the existing cloud infrastructure.

Keywords: AES–XOR encryption; cloud security; data integrity; key splitting; merkle tree; secure deletion

How to Cite: H. S. E. Shakur and B. A. Mahmood, "Efficient Assured Cloud Data Deletion and Verification Scheme Using AES-XOR Encryption and Merkle Trees," *J. RESTI (Rekayasa Sist. Dan Teknol. Informasi),* vol. 10, no. 4, pp. 1099–1106, 2026.
*Permalink/*DOI: https://doi.org/10.29207/resti.v10i4.7421

Received: December 26, 2025
Accepted: April 19, 2026
Available Online: August 19, 2026

This is an open-access article under the CC BY 4.0 License
Published by Ikatan Ahli Informatika Indonesia

1. Introduction

In the absence of local infrastructure, cloud computing provides flexible and scalable processing and storage options[1]. This issue is critical for both academic research and real-world applications, as the inability to ensure complete data removal raises serious concerns, such as illegal retrieval, privacy breaches, and legal violations [2]. In light of this, one of the most critical issues in modern cloud environments is the concept of guaranteed assured deletion. The term "assured deletion" refers to the ability to ensure that, once deleted, data cannot be recovered under any circumstances and to provide verifiable proof to reassure users and authorities that assured deletion has been successfully performed. Several requirements must be carefully integrated to accomplish this: robust cryptographic guarantees that prevent any possibility of recovery [3], adaptable structures that support dynamic storage scenarios where data is constantly added or removed, and efficient execution to reduce costs and delays. Recent developments in data formats and cryptographic techniques imply that guaranteed assured deletion is now a feasible need for fostering confidence in cloud services rather than just a theoretical goal[4]. Designing systems that guarantee not just safe data management and storage but also irreversible and verifiable assured deletion when necessary has become crucial due to people's and businesses' growing dependence on cloud-based solutions. This emphasizes the importance of continuing research to improve models and methods that balance security[5], efficiency, and public verifiability, opening the door to a cloud ecosystem that is more reliable and secure[6-8].

The originality of this paper lies not only in the introduction of new encryption techniques but also in its integrated approach to combining the AES and XOR

algorithms. This scheme uses XOR-based sequential linking of blocks and AES encryption. The randomly generated encryption key is then split into two complementary parts and stored in independent locations. The purpose is to prevent any single entity from possessing the complete encryption key and to enable secure deletion in cloud storage. The encrypted blocks are arranged with one of the key fragments in a Merkle tree structure, ensuring data integrity during transmission and storage, and verifying data without requiring complete data recovery. This scheme ensures the inseparability of encrypted data during storage, unlike traditional methods that rely solely on decrypting the encryption key or encrypting the file. This guarantees that no single entity possesses sufficient information to decrypt the data. This scheme supports practical and verifiable deletion within a real cloud environment

Increased emphasis on cloud computing has made assured deletion a point of concern where deletion assurance is acute. Cloud providers usually have traditional deletion operations that are not transparent and can not guarantee full deletion. Therefore, researchers have suggested cryptography protocols and system architectures that maximize control, safeguard privacy, and allow verifiable assured deletion. These strategies tend to amalgamate into four categories: (1) symmetric encryption-only approaches, (2) key management and key transformation methods, (3) attribute-based and identity-based access control, and (4) some form of verifiable deletion that makes use of distributed or cryptographic proofs.

One of the simplest methods of secure deletion is encrypting and destroying the encryption key during file deletion, initially proposed by Perelman [9], [10].In this type of study, the data owner uses and maintains a randomly generated encryption key, but this method is not always secure because the data is left in the cloud, and the encryption key is either kept with them or sent to a trusted third party. The deletion method in this study involves canceling or deleting the encryption key, thus rendering the file unrecoverable [11], [12]. Examples of this method include policy-based access control and time-based secure deletion in early systems such as FADE.[13]. Although they are inexpensive and their implementation is easy, their verifiability is limited, and they are dependent on trusted key managers, which is a source of a single point of vulnerability [14], [15].

More advanced models are trying to enhance flexibility at the expense of focused updates or distributed storage. As an example, Geambasu et al. [16] presented the idea of a system with threshold secret sharing under which keys split get distributed to a Distributed Hash Table (DHT) in order to be revoked automatically. Equally, Luo scheme [17]is an advance on Rivest AONT-based package Transform [18], where it creates a strongly nonseparable Ciphertext sequence with pseudorandom functions and hash operations, thus giving a reduced encryption overhead. Those techniques make deletion more secure, but act as implementation complexities and raise overheads of communication, especially in distributed systems[19-21].

The ABE and IBE systems and schemes guarantee finer-granularity access control and revocation schemes that allow assured deletion through changing user attributes or revoking identity-based keys. As an example, other schemes update attributes dynamically such that, at the end of the update, it makes the ciphertext inaccessible without re-encryption [22], [23]. Efficient revocation is also experienced in IBE-based methods, but they rely on a trusted party known as a private key generator (PKG). This opens up the possibility of creating bottlenecks or loss of privacy. These schemes suffer from the fact that they must be highly computationally expensive and depend on a trusted third-party infrastructure, despite being flexible [20], [24], [25].

To address these shortcomings of the unverifiability of traditional mechanisms, the recent literature has implemented verifiable deletion mechanisms[26]. A blockchain-based protocol of deletion was proposed by Liu et al. [21]wherein identity verification and deletion evidence are stored, and smart contracts are in control. In other models, there is an override verification tree [7]or the dynamic hash structures to make sure that deletion has been done properly by the cloud provider. The approaches contribute to a high level of user confidence and compliance, although they introduce processing overhead and cost to modify the cloud infrastructure[27-29]. Despite extensive research on data deletion assurance in cloud storage, current systems typically suffer from several limitations, including high computational load, a lack of post-deletion integrity verification capabilities, key recovery, and inadequate resistance to cryptographic attacks. Some security issues may persist because many methods focus solely on key destruction without complete separation of the encrypted data components, potentially leaving residual security risks. Furthermore, data integrity verification is often neglected. To overcome these shortcomings, this paper proposes an efficient cloud data deletion and verification system for strong block separation. It utilizes AES-based encryption with XOR binding, key splitting, and a Merkel tree structure for file verification, resulting in a lightweight system. A comprehensive performance study demonstrates that the proposed scheme reduces computational effort, improves post-deletion security, and provides practical deletion assurance[30].

2. Methods

In this section, the proposed scheme is presented in detail. The assured deletion system has four main components, which are: The data owner (DO), who encrypts the original data and stores it in a trusted third party (TTP) and cloud storage server (CSS), the authorized users (AU), who have access to the

encrypted data, the CSS, who provides data storage services, and the TTP, who holds part of the ciphertext and part of the encryption key, which is supposed to destroy it when required by the applicable policy as shown in Figure 1.

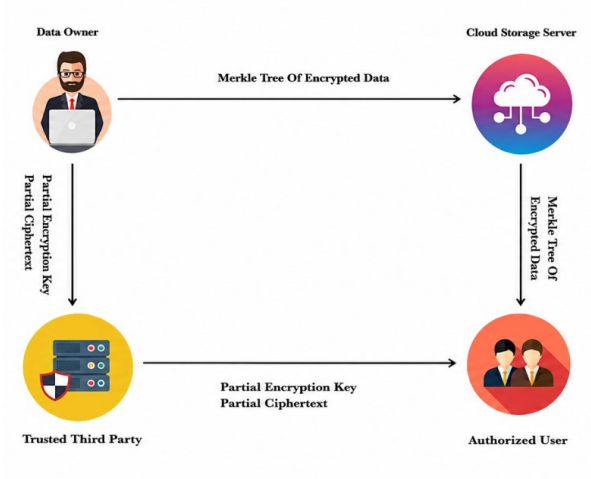


Figure 1. Assured Deletion System Model.

2.1 In the Proposed Scheme

The proposed technique combines block-level AES encryption, XOR binding, and key splitting to ensure both assured deletion and verification. The original text file is divided into blocks, each linked to the preceding block via XOR operations, creating a strong, interconnected string. Each intermediate block, denoted as mi', is encrypted using XOR to generate ciphertext blocks. To facilitate efficient verification, each ciphertext block is structured with selected key components in a Merkle tree, from which a root hash is generated and securely stored.

This root serves as a concise reference for verifying data integrity during decryption. Assured deletion is achieved by removing a portion of the encryption key along with the assembly value $c0'$ obtained in a final cipher, which is then reused in a decryption stored in a trusted third party. In this paper, the trusted third party was implemented using Azure Key Vault. Once these components are destroyed, it becomes impossible to decrypt the ciphertext back into meaningful plaintext, thus ensuring permanent data assured deletion. As a result, this technique addresses both criteria simultaneously: robust, guaranteed deletion and verifiable data integrity within the context of modern cloud computing security.

2.2 Description Of Specific Steps

The algorithms of encryption and decryption of files, and the destruction and integrity verification mechanism, employed in the realization of the discussed scheme, are also provided in this section.

Let us say **Enc** is a symmetric block encryption operation, and Dec is the decryption operation to **Enc**. We will consider that the initial input file could be transformed into a series of plaintext blocks $\mathbf{F = (m_1, m_2, ..., m_n)}$, which could be transformed, encrypted, and securely distributed according to the system design.

2.3 Data Encryption

The data owner (DO) performs the encryption process according to the proposed methodology. Initially, the original file is divided into equal-sized blocks. Then, each block is linked to its predecessor using a sequential XOR operation, employing a randomly generated initialization vector $m0'$. This results in the generation of intermediate blocks mi'. For each mi', the value of $AES_k(m_i')$ is encrypted using a randomly generated symmetric AES key K. The final ciphertext block is calculated as follows: $c_i' = AES_k(m_i') \oplus prev_mi'$. This results in a strong interconnection between the blocks, and security is provided through this two-layer transformation (sequential XOR + AES). After each ciphertext block is created, the assembly value $c0'$ is calculated using an XOR operation between $m0'$ and all ci' values. Then, simple key splitting is applied to divide the key K into two complementary parts, ensuring that the loss or assured deletion of either part makes recovery of the original key impossible.

One of the key parts, with the value $c0'$, forms the Δ (delta) packet, which is securely stored in Azure Key Vault, a trusted third-party service, to guarantee its assured deletion. The ciphertext blocks consist of [$c1'$, $c2'$, …,cn'], and the remaining basic components are arranged like the leaves of a Merkle tree. The tree is uploaded to the cloud as a Σ (sigma) packet, and the Merkle tree's root hash is stored as a compressed integrity token used later to verify data integrity. Access to Σ is restricted to authorized users, and deletion in TTP removes Δ, facilitating recovery of the original text as described in Figure 2.

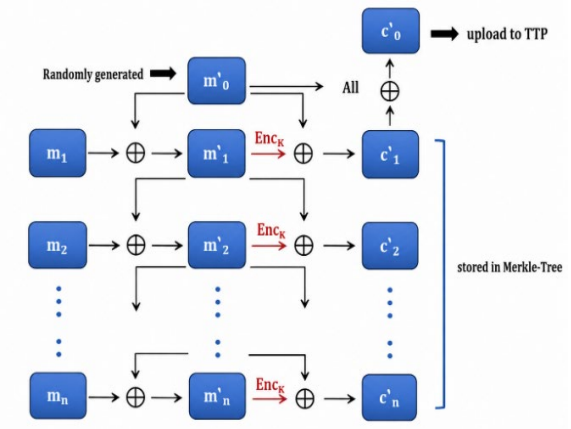


Figure 2. Data Encryption Process using AES-XOR chaining and block dependency.

Furthermore, during the encryption process, confirmation is given that the Merkle tree is sent to the cloud, and the core components are also sent to a trusted third party. Access and storage in the cloud and by the trusted third party are guaranteed. Afterward, the file is deleted from local storage or from the data owner's location.

2.4 Data Decryption

An authorized user performs the decryption process after receiving packets Δ and Σ from a trusted third party (TTP) and a cloud service, respectively. First, the data is verified for integrity by comparing Merkle tree hashes to ensure it hasn't been tampered with. Next, the necessary bits for reconstructing the decryption key are retrieved. This is done by recombining key bits from packet Σ and a second bit from packet Δ. This process yields the complete decryption key Then, the operations used in encryption are reversed for all blocks sequentially, and each decrypted block is placed in the final output in the correct order. As a final step, the file's authenticity is verified using the SHA-256 hash value, as shown in Figure 3.

From a security perspective, an authorized user does not have permission to modify the file's contents, delete any part of it, or even access other files stored in the same cloud storage. These permissions are defined by the data owner and can be revoked by the data owner.

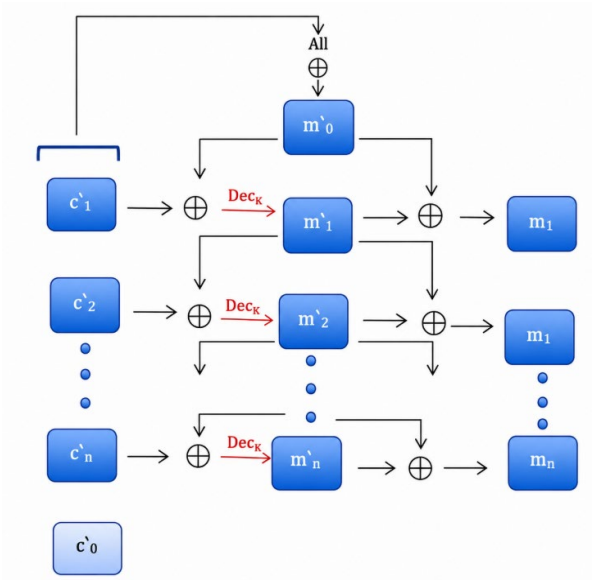


Figure 3. Data Decryption Process using AES-XOR chaining and previous block dependency.

2.5 Data Delete Mechanism

Both **Σ** (from the cloud service provider) and **Δ** (from the trusted third party) are deleted when an assured deletion request is made; alternatively, assured deletion is assured once **Δ** is permanently deleted by the TTP. The loss of either **Δ or Σ** renders the original file unavailable and the ciphertext undecipherable due to the irreversible link formed by the Merkle tree and XOR operations.

Furthermore, when only Δ is deleted by the trusted third party while Σ remains in the cloud, guaranteed deletion is also ensured. This is because the absence of essential components used for decryption makes key retrieval impossible, thus rendering the original file unrecoverable.

2.6 Threat Model

The system takes into account that the proposed system is a practical model for cloud threats and countering various malicious activities. It assumes that the cloud service provider is curious, meaning it performs its procedures correctly but also aspires to examine the text stored in the cloud and its metadata while correctly implementing the protocol. However, the proposed system protects the data from unauthorized disclosure or modification because the encryption key to decrypt the file and read its contents is not available. This is because the encryption key is not stored fully in the cloud with the file, but rather divided into two parts. One part is embedded in the Merkle tree, while the other part is stored with a trusted third party. This ensures that the file's contents cannot be decrypted or read simply by accessing or intercepting the cloud's contents, as the data stored in the cloud is insufficient for decryption

Moreover, the proposed system considers an external attacker who possesses encrypted data or a Merkle tree, or who may have partial information stored in the cloud, or who obtained data before deletion. However, the attacker cannot access or decrypt the file because once the core components stored with a trusted third party are deleted, the file or these components cannot be recovered, as the remaining information is useless and unusable. Thus, the proposed system assumes resistance to potential cryptographic attacks from external parties.

In addition, this study employs a Merkle tree architecture with a collision-resistant hashing function. This step is crucial for detecting any unauthorized modification of the encrypted or metadata by unauthorized individuals or entities. Furthermore, a third-party key management service operates in a secure and isolated environment. This ensures that the core data (C0 and a key fragment) remains secure. The Merkle tree is used to store the encrypted data fragments and a second part of the encryption key, helping to prevent tampering with sensitive information during storage. Secure channels are also used during data transfer between the user and the cloud computing environment. The study combines XOR data binding, AES encryption, key splitting, and the Merkle tree.'

2.7 Security Analysis

The proposed plan is designed to achieve three key security objectives: verifying data integrity, maintaining confidentiality, and guaranteeing assured deletion. Advanced Encryption Standard (AES)-based encryption and sequential XOR linking provide confidentiality, while data integrity is verified using a Merkle tree structure. The plan assumes secure key management by a trusted service, AES secrecy strength, and collision resistance for the hashing function used. After deleting the relevant metadata and the stored key component, even if some of the storage contents are exposed, the remaining ciphertext stored in the cloud is insufficient to recover the original data.

Additionally, the proposed method is designed to be highly secure so that even if all the essential components and the Merkle tree are exposed to an attacker, they cannot recover the original file. This is because the initial value m_0`, which is crucial for file decryption (since all blocks rely on $m_0$` to decrypt them), is unavailable.

2. Results and Discussions

This section addresses a theoretical analysis of the study's effectiveness from three axes: storage costs, communication costs, and computation costs. The table illustrates that the proposed scheme achieves balanced overhead performance while maintaining strong non-separability and confidentiality compared to existing schemes. Results of encryption only: Xie et al. scheme[2], Luo's scheme[1] and the proposed scheme.

3.1 Storage Overhead

In the proposed methodology, file F, after being encrypted and divided into n equal-sized blocks, encrypts each block to obtain cipher blocks $c' = (c_1', c_2', ..., c_n')$. Then, an additional block (the base block c_0') is added. This block ensures that the user cannot decrypt blocks separately, which is impossible. Thus, we obtain (n + 1) blocks that can be encrypted in total. Furthermore, after dividing the blocks, these blocks are arranged in a Merkle tree along with a portion of the encryption key that has been divided and distributed. One portion is arranged in the Merkle tree with the encrypted blocks, while the other portion of the encryption key is stored in a third party trusted with the base block c_0'. The purpose of using a Merkle tree is to verify data integrity during transmission or storage in the cloud. While a Merkle tree increases storage size because it requires approximately $(2n - 1)$ fragmentation, this increase is relatively negligible as the individual file size increases relative to the overall file size. However, this design provides scalable verification of the integrity of stored blocks and detection of any unwanted tampering. It achieves a practical balance between a slight increase in storage and computation costs and a significant improvement in data security and integrity, after assured deletion features lacking in many traditional trusted assured deletion systems. This makes it suitable for real-world cloud computing applications.

However, these security enhancements are minimal and acceptable when file sizes increase to large sizes, considering the efficiency and layers of security this system provides. The key size remains constant even when the file size increases, and in some very large files, the key size is negligible compared to the file size. Additionally, the Merkle tree grows relatively slowly compared to the file size. Therefore, the system achieves a low storage load.

3.2 Calculation Overhead

In the proposed scheme, the data owner (DO) performs the following operations, as shown in Equation 1.

$$3n \cdot XOR + n \cdot ENC + 1PRF \quad (1)$$

At the start of the process, a single hash function (SHA-256) is used before encryption, and therefore it is excluded from the computational cost analysis because it is not part of the encryption and decryption cycle. Subsequently, the encryption key K and the initial value m_0' are randomly generated using a PRF operation via os.random(). During decryption, the decryption process is performed by the authorized user (AU) to obtain the following operation: XOR + n · DEC. In comparison, the Luo study requires the following operation: XOR + n · HASH + (n + 1) · ENC + n · PRF. Thus, it can be observed that the proposed scheme incurs a higher computational cost. In terms of better efficiency and lightweight performance, it reduces the use of the PRF to one run and eliminates the hashing operations within the encryption while providing enhanced security guarantees without imposing an excessive computational burden

Additionally, this study surpasses the previous Luo study. Although it contains security layers, these layers are not included in the computational cost, as key splitting falls under key management, while the Merkle tree is outside file encryption and falls under the methods of storing files in cloud computing. Therefore, this method is considered effective compared to the previous study.

3.3 Communication Overhead

To calculate the connection cost in this scheme, after the encryption algorithm is calculated and the Merkle tree is arranged with the first portion of the encryption key, the Merkle tree is sent to the cloud service provider for storage. The second portion, containing the second part of the encryption key and the c0′ encryption sequence, is sent to a trusted third party such as Azure Key Vault. This equation provides an accurate quantitative description of the connection cost in the proposed model. This cost is directly proportional to the number of data fragments and remains practically manageable, significantly improving post-deletion security. Thus, the proposed system achieves a reasonable balance between increased connection costs and enhanced data integrity and permanent deletion assurance.

However, the proposed system achieves a reasonable balance between increased communication costs and enhanced data integrity through the Merkle tree, while maintaining data confidentiality by splitting the encryption key and storing it in different repositories, ensuring the secure and unrecoverable permanent deletion of data.

3.4 Comparative Analysis

The comparative results in Table 1, after comparing the proposed study with previous studies, show that the study maintains the additional cost balance between storage, computing, and connectivity costs. Compared to the schemes of Luo's and Shi et al., Luo's system

adds additional hashing operations to ensure data integrity, while the proposed study uses XOR-based data binding with AES encryption to maintain strong non-splicing, with moderate additions such as Merkle tree verification and cryptographic key splitting. This maintains the workflow and supports guaranteed deletion in a cloud environment.

Table 1. Performance Analysis of Schemes

Scheme	Storage Overhead	Calculation Overhead	Communication Overhead	Strong Non-separability	Confidentiality
Luo's Scheme [1]	(n + 1)β	2nXOR + nHASH + (n + 1)ENC + nPRF	(n + 1)β	Yes	Yes
Xie et al. Scheme [2]	(n + 1)β+k	3nXOR + nENC + 1PRF	(n + 1)β+k	Yes	Yes
Proposed Scheme	(n + 1)β + 2k + (2n – 1)h	3nXOR + nENC + 1PRF	(n + 1)β + 2k + (2n – 1)h	Yes	Yes

Furthermore, despite the added security features of this system, such as the Merkle tree for verifying the integrity of its storage fragments and the encryption key partitioning, the additional storage costs decrease with increasing file size because the key size remains constant. The size of the Merkle tree increases only slightly compared to the file size. Therefore, this system achieves a balance between efficiency and security.

3.5 Adversarial and Stress Analysis

The proposed system was evaluated under multiple stress and challenge scenarios to assess its robustness. Any modifications made to the ciphers or Merkle tree metadata in tampering scenarios resulted in immediate verification failure due to hash inconsistencies, enabling effective tamper detection. Partial assured deletion, achieved by removing the protected key component, a portion of the ciphertext, or both from the trusted key management service, becomes computationally impossible to reconstruct the remaining ciphertext, even if the data remains stored in the cloud.

Furthermore, XOR-based sequencing and block-level AES encryption enable the system to operate in independent segments, providing scalable processing under concurrent or large-scale workloads without compromising integrity or confidentiality. This mechanism also demonstrates high resistance to partial assured deletion attempts, as the incomplete data prevents successful recovery of the original file.

3.6 Parameters and Environment

The encryption and decryption operations and algorithms of the proposed scheme and the scheme developed by Shi et al. [2] were compared and validated using Python for both methods. The computational efficiency, storage efficiency, and connectivity efficiency were evaluated.

Absent the use of sophisticated hardware, every experiment was conducted on a personal computer in a typical internet computing environment. A 1KB block size was chosen as the main test model after testing a range of file sizes. This size strikes a suitable balance between computing efficiency and representing realistic cloud storage scenarios. The proposed system is also scalable, as it relies on block-level data processing. All examples employed a 128-bit encryption key (k = 16 bytes), and the file size (β) was varied as 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 50 KB, and 100 KB. To guarantee consistency, each experiment was conducted several times, and the arithmetic mean of the findings was presented. All schemes were tested with the same parameter values, optimized within their design, and implemented in a similar programming language to provide a fair comparison.

3.7 Data Encryption Test

File encryption was tested for both studies, and the encryption time for a scheme was calculated on random samples. Next, a kilobyte of encrypted data is generated and uploaded to the CSP. The remaining parts (part of the encryption key and part of the encrypted file) are sent to a trusted third party (TTP). Figure 4 illustrates the encryption time cost for both studies. To demonstrate the similarity of the tools used in both studies, at small file sizes (1 to 16 kilobytes), the difference in encryption time is small compared to large sizes (50 to 100 kilobytes), where the difference increases significantly. This increase is acceptable and smoother, indicating that the system has greater scalability as file sizes increase and is more stable than the comparative study. indicates that the proposed scheme generates less computational load than the Xie et al. scheme. [2] In cases where the number of blocks is reduced at large file sizes.

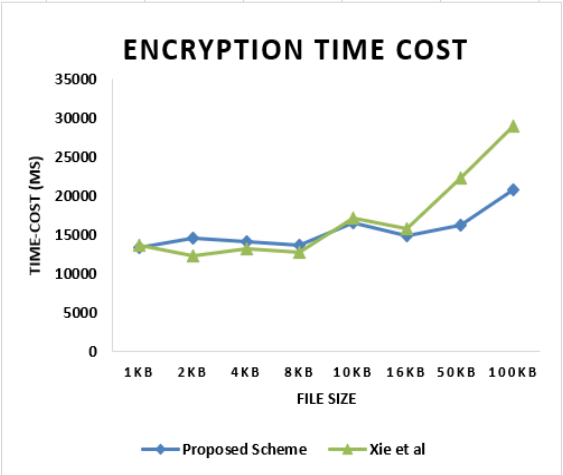


Figure 4. Comparison of Encryption Time Cost between the Proposed Scheme and Xie et al.

Additionally, these file sizes were used because the security enhancements (Merkle tree with encryption key splitting) are prominent in these file sizes, as these enhancements are approximately the size of one file or smaller. Therefore, these sizes were used to highlight the encryption time and demonstrate the system's efficiency.

3.8 Data Decryption Test

In both studies, the file was decrypted by an authorized user, and hashing was performed for both studies. The decryption time was nearly identical between the two studies at the small 1 KB and 16 KB sizes, while the proposed method took longer at 50 KB and 100 KB sizes, as shown in Figure 5. This is an acceptable increase compared to the security levels offered by this study and the file verification methods used during decryption.

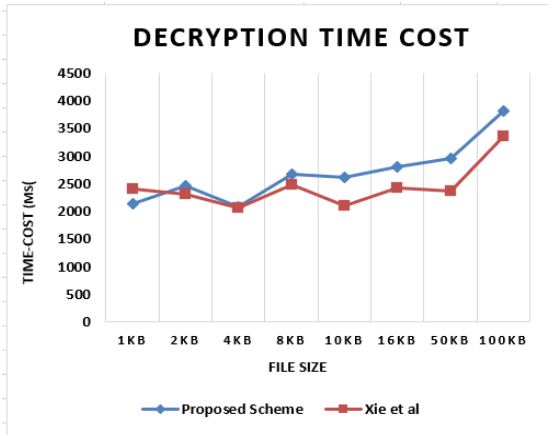


Figure 5. Comparison of Decryption Time Cost between the Proposed Scheme and Xie et al.

Furthermore, while the decryption time shows a slight increase compared to the encryption time, this increase is acceptable considering the added security features of this system, such as the Merkle Tree, which verifies that stored data has not been tampered with, and the encryption key splitting, which enhances the system's encryption strength.

3.9 Data Deletion Test

The deletion process is implemented in both studies, and the execution time is calculated. The results verify that the file cannot be decrypted or recovered after deletion, including the deletion of key elements and the decrypted file itself. The deletion time for both methods is shown, with relatively similar time costs for small file sizes (1 KB to 16 KB) in both studies. However, for large file sizes (100 KB), the Xie et al. method shows a significant increase in deletion time costs, as illustrated in Figure 6.

The proposed study, on the other hand, shows a relatively small and acceptable increase compared to the other study. Therefore, we conclude that the proposed method achieves deletion by preventing the separation of blocks, demonstrating the system's effectiveness at large file sizes.

Furthermore, the deletion time demonstrates that the system outperforms the Xie et al. system. Despite the added security measures, these additions did not affect the deletion time, indicating an effective response to increasing file size**.**

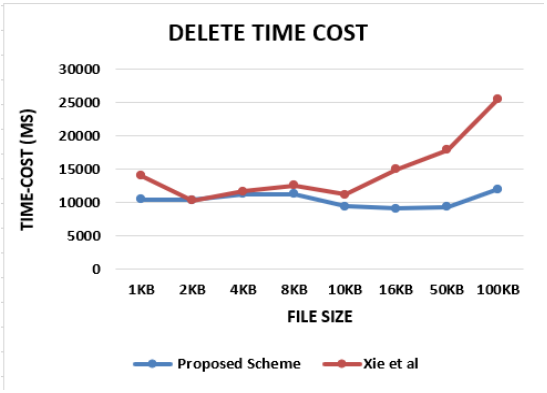


Figure 6. Comparison of Data Deletion Time Cost between the Proposed Scheme and Xie et al.

4. Conclusions

This paper presents a secure and efficient framework by integrating AES-XOR encryption and Merkle tree verification for secure and verifiable data deletion in cloud storage, enhancing user and administrator confidence. Thanks to lightweight XOR operations and structured key management, experimental results demonstrate improvements in both security and computing efficiency. Therefore, the proposed system offers a practical and scalable solution for real-world cloud computing environments.

Limitations

Despite the advantages of the proposed scheme, some limitations should be noted. This approach relies on a trusted key management service to securely store and delete key components, creating a quasi-trusted dependency. Furthermore, the system relies on the use of secure communication channels between entities. Additionally, the pilot evaluation was conducted on a limited dataset, and more comprehensive testing may be necessary to confirm performance in actual cloud computing systems.

Acknowledgements

The author declares no conflict of interest. This research has not received any specific funding from any donor, whether public, commercial, or non-profit.

References

[1] Y. C. Luo, “Research on Public Auditing for Shared Data and Assured Deletion in Cloud Storage,” M.S. thesis, National University of Defense Technology, Changsha, China, 2015. [Online].Available:https://globethesis.com/?t=2348330536967717

[2] Z. Xie, W. Fu, J. Xu, and T. Zhu, “Assured Deletion: A Scheme Based on Strong Nonseparability,” *J. Sens.*, vol. 2022, pp. 1–8, Mar. 2022, doi: 10.1155/2022/9691724.

[3] S. M. Khudaier and B. A. Mahmood, "A Review of Assured Data Deletion Security Techniques in Cloud Storage," *Iraqi J. Sci.*, vol. 64, no. 5, pp. 2492–2511, May 2023, doi: 10.24996/ijs.2023.64.5.33.

[4] J. Dave, A. Dutta, P. Faruki, V. Laxmi, and M. S. Gaur, "Secure Proof of Ownership Using Merkle Tree for Deduplicated Storage," Autom. Control Comput. Sci., vol. 54, no. 4, pp. 358–370, Jul. 2020, doi: 10.3103/s0146411620040033.

[5] I. Khalil, A. Khreishah, and M. Azeem, "Cloud Computing Security: A Survey," *Computers*, vol. 3, no. 1, pp. 1–35, Feb. 2014, doi: 10.3390/computers3010001.

[6] Q. Wang, F. Zhou, J. Xu, and Z. Xu, "Efficient verifiable databases with additional insertion and deletion operations in cloud computing," *Future Gener. Comput. Syst.*, vol. 115, pp. 553–567, Feb. 2021, doi: 10.1016/j.future.2020.09.028.

[7] L. Xue, Y. Yu, Y. Li, M. H. Au, X. Du, and B. Yang, "Efficient attribute-based encryption with attribute revocation for assured data deletion," *Inf. Sci.*, vol. 479, pp. 640–650, Apr. 2019, doi: 10.1016/j.ins.2018.02.015.

[8] A. Bentajer, M. Hedabou, K. Abouelmehdi, Z. Igarramen, and S. El Fezazi, "An IBE-based design for assured deletion in cloud storage," *Cryptologia*, vol. 43, no. 3, pp. 254–265, May 2019, doi: 10.1080/01611194.2018.1549123.

[9] S. Bauskar, "Advanced Encryption Techniques for Enhancing Data Security in Cloud Computing Environment," Int. Res. J. Mod. Eng. Technol. Sci., vol. 5, no. 10, pp. 3328–3339, Oct. 2023, doi: 10.56726/IRJMETS45283.

[10] T. Soroceanu, N. Buchmann, and M. Margraf, "On Multiple Encryption for Public-Key Cryptography," *Cryptography*, vol. 7, no. 4, pp. 49, Oct. 2023, doi: 10.3390/cryptography7040049.

[11] K. Qambar Aziz and B. Ahmed Mahmood, "Assured data deletion in cloud computing: security analysis and requirements," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 28, no. 2, pp. 1174, Nov. 2022, doi: 10.11591/ijeecs.v28.i2.pp1174-1183.

[12] S. M. Ahmed and B. A. Mahmood, "Cloud Computing Security: Assured Deletion," *Informatica*, vol. 48, no. 3, pp. 485–496, Sep. 2024, doi: 10.31449/inf.v48i3.6245.

[13] G. Wang and Y. Luo, "A Review on Assured Deletion of Cloud Data Based on Cryptography," *Procedia Comput. Sci.*, vol. 187, pp. 580–585, 2021, doi: 10.1016/j.procs.2021.04.111.

[14] C. Yang, Y. Liu, and X. Tao, "Assure deletion supporting dynamic insertion for outsourced data in cloud computing," *Int. J. Distrib. Sens. Netw.*, vol. 16, no. 9, pp. 1–14, Sep. 2020, doi: 10.1177/1550147720958294.

[15] A. K. Ranjan, V. Kumar, and M. Hussain, "Security Analysis of Cloud Storage with Access Control and File Assured Deletion (FADE)," in *2015 Second International Conference on Advances in Computing and Communication Engineering*, Dehradun, India: IEEE, May 2015, pp. 453–458, doi: 10.1109/icacce.2015.10.

[16] R. Guo, C. Zhuang, H. Shi, Y. Zhang, and D. Zheng, "A lightweight verifiable outsourced decryption of attribute-based encryption scheme for blockchain-enabled wireless body area network in fog computing," *Int. J. Distrib. Sens. Netw.*, vol. 16, no. 2, pp. 13, Feb. 2020, doi: 10.1177/1550147720906796.

[17] J. Ma, M. Wang, J. Xiong, and Y. Hu, "CP-ABE-Based Secure and Verifiable Data Deletion in Cloud," *Secur. Commun. Netw.*, vol. 2021, pp. 1–14, Mar. 2021, doi: 10.1155/2021/8855341.

[18] R. L. Rivest, "All-or-Nothing Encryption and the Package Transform," in Fast Software Encryption, Lecture Notes in Computer Science, vol. 1267, Springer, Berlin, Heidelberg, 1997, pp. 210–218, doi: 10.1007/BFb0052348.

[19] A. Rahumed, H. C. H. Chen, Y. Tang, P. P. C. Lee, and J. C. S. Lui, "A Secure Cloud Backup System with Assured Deletion and Version Control," in Proc. 2011 40th Int. Conf. Parallel Process. Workshops (ICPPW), Taipei City, Taiwan, Sep. 2011, pp. 160–167, doi: 10.1109/ICPPW.2011.17.

[20] Y. Zhang, C. Xu, J. Ni, H. Li, and X. Shen, "Blockchain-Assisted Secure Data Sharing with Auditable and Assured Deletion in Cloud Computing," *IEEE Trans. Serv. Comput.*, vol. 15, no. 6, pp. 3298–3311, 2022. doi: 10.1109/TSC.2021.3052989.

[21] Z. Mo, Y. Qiao, and S. Chen, "Two-Party Fine-Grained Assured Deletion of Outsourced Data in Cloud Systems," in Proc. IEEE 34th Int. Conf. Distrib. Comput. Syst. (ICDCS), Madrid, Spain, Jun. 2014, pp. 308–317, doi: 10.1109/ICDCS.2014.39.

[22] C. Yang, X. Tao, and F. Zhao, "Publicly verifiable data transfer and deletion scheme for cloud storage," *Int. J. Distrib. Sens. Netw.*, vol. 15, no. 10, Oct. 2019, doi: 10.1177/1550147719878999.

[23] Y. Ren *et al.*, "Multiple cloud storage mechanism based on blockchain in smart homes," *Future Gener. Comput. Syst.*, vol. 115, pp. 304–313, Feb. 2021, doi: 10.1016/j.future.2020.09.019.

[24] X.-Y. Zheng and C. Kuang, "Arbitration quantum signature protocol based on XOR encryption," *Int. J. Quantum Inf.*, vol. 18, no. 05, p. 10, Aug. 2020, doi: 10.1142/S0219749920500252.

[25] O. G. Abood and S. K. Guirguis, "A Survey on Cryptography Algorithms," *Int. J. Sci. Res. Publ.* (IJSRP), vol. 8, no. 7, pp. 495–516, Jul. 2018, doi: 10.29322/IJSRP.8.7.2018.p7978.

[26] Y. Liu, X. A. Wang, Y. Cao, D. Tang, and X. Yang, "Improved Provable Data Transfer from Provable Data Possession and Deletion in Cloud Storage," in *Lecture Notes on Data Engineering and Communications Technologies*, vol. 23, Cham, Switzerland: Springer, 2019, pp. 445–452, doi: 10.1007/978-3-319-98557-2_40.

[27] L. Wang, B. Wang, W. Song, and Z. Zhang, "A key-sharing based secure deduplication scheme in cloud storage," *Inf. Sci.*, vol. 504, pp. 48–60, Dec. 2019, doi: 10.1016/j.ins.2019.07.058.

[28] J. Tian, R. Bai, and T. Zhang, "Supporting Dynamic Update Scheme for Assured Deletion Based-Multi-Copy Association Tree," *Research Square* , vol. 27, no. 2, pp. 1983–2001 Jun. 29, 2022, doi: 10.21203/rs.3.rs-1548271/v2.

[29] Y. Meng, B. Wang, Q. Xing, X. Wang, J. Liu, and X. Xu, "BBAD: Blockchain-based data assured deletion and access control system for IoT," *Peer-to-Peer Netw. Appl.*, vol. 18, no. 2, Mar. 2025, doi: 10.1007/s12083-024-01881-x.

[30] M. M. Hoobi and I. A. Abdulmunem, "A Strong AES Algorithm Using Hash Functions for Key Generation," Iraqi J. Sci., vol. 66, no. 8, pp. 3422–3440, Aug. 2025, doi: 10.24996/ijs.2025.66.8.31.