



An Efficient Approach for Secure Data Storage in Cloud Computing Environments

Wariya Imamverde

Department of Technology and Innovation Management, Padma Institute of Business and Management, Bangladesh

wariya.imamverde@pibm-bd.org

Peer Review Information

Submission: 18 Jan 2015

Revision: 03 Feb 2015

Acceptance: 19 March 2015

Keywords

cloud computing, secure data storage, encryption, data integrity, access control

Abstract

Cloud computing provides elastic and economical storage, but outsourcing data to infrastructure outside the owner's direct control introduces confidentiality, integrity, access-control, and availability concerns. This methodology paper proposes an efficient secure-storage approach in which sensitive files are protected before upload, encryption keys are separated from cloud-resident ciphertext, access decisions are enforced through authenticated policies, and stored objects are periodically verified through integrity metadata and challenge-based auditing. The design uses authenticated symmetric encryption for bulk data because it provides confidentiality and tamper detection with low client overhead, while hash-based metadata and provable data possession concepts support verification without repeatedly downloading complete files. A lightweight key-management layer isolates encryption keys from the storage provider and supports key rotation and revocation. The methodology is evaluated at the design level against the main threats identified in cloud-storage literature published up to 2015. The resulting architecture provides defense in depth: compromise of the cloud store alone does not disclose plaintext, unauthorized modification is detectable, and access can be revoked without exposing master keys. The approach is suitable for academic and enterprise systems that need practical security without excessive computational or communication cost.

Introduction

Cloud computing has transformed storage from a locally managed resource into an on-demand service that can be scaled according to user requirements. Armbrust et al. [1] described cloud computing as a utility-oriented model whose elasticity and resource sharing can reduce the cost of deploying and operating large systems. The same model, however, changes the security boundary. When data are outsourced, the owner no longer has continuous physical control over the servers, storage media, administrative processes, or backup copies that hold the information. Consequently, secure cloud storage must protect data even when the underlying

storage platform is treated as honest-but-curious or when a storage component is compromised.

Security studies have repeatedly identified data confidentiality, privacy, transmission security, isolation, and third-party trust as central concerns in cloud service delivery [2], [9]. A secure-storage mechanism should therefore do more than encrypt a file once. It must control who can obtain decryption capability, preserve integrity when files are modified, provide a way to verify remotely stored data, and reduce the impact of key compromise. Kamara and Lauter [3] argued that cryptographic protection can be placed above public-cloud infrastructure so that

customers do not need to fully trust the provider. This principle motivates the present work: sensitive information should leave the client only after confidentiality and integrity controls have been applied.

The objective of this paper is to define a practical, layered methodology for secure data storage in a cloud environment using mechanisms that were established in the literature by 2015. The proposed approach combines authenticated encryption, cryptographic hashing, separated key management, policy-based access control, integrity auditing, and controlled recovery. Its emphasis is efficiency: expensive public-key operations are restricted to key or policy handling, while high-volume file protection relies on symmetric cryptography. The methodology also supports dynamic data by updating authentication metadata when file blocks or versions change. The contribution is not a new cryptographic primitive; rather, it is an integrated workflow that organizes established techniques into a deployable storage-security process.

Literature Review

Ateniese et al. (2007) [4] introduced Provable Data Possession (PDP), allowing a client to verify that an untrusted server still possesses outsourced data without downloading the entire file. The work demonstrated that probabilistic challenge-response verification can reduce communication and input/output cost while retaining meaningful assurance about remote data possession. PDP established an important design principle for cloud storage: integrity checking should be remote, selective, and substantially cheaper than full retrieval.

Juels and Kaliski (2007) [5] proposed Proofs of Retrievability (PORs) for large files. Their model sought evidence not only that selected blocks exist, but that the complete file remains recoverable. POR techniques influenced later cloud auditing systems by showing how embedded checks and compact proofs can reduce the cost of remote verification. For a practical methodology, PDP and POR concepts justify maintaining integrity metadata that can be challenged periodically rather than trusting successful upload alone.

Bethencourt, Sahai, and Waters (2007) [6] introduced ciphertext-policy attribute-based encryption (CP-ABE), enabling encrypted content to be associated with expressive access policies. Their work showed how confidentiality and fine-grained authorization can remain coupled even when storage is untrusted.

Although CP-ABE is computationally heavier than symmetric encryption for large files, it is valuable for protecting small data-encryption keys or enforcing policy at the key-distribution layer.

Kamara and Lauter (2010) [3] examined cryptographic architectures for constructing storage services on top of a public cloud that is not completely trusted. They emphasized combining cryptographic techniques rather than relying on the cloud provider as the sole security control. Their architecture-level view supports client-side protection, searchable or auditable metadata where needed, and separation between storage infrastructure and key material.

Yu et al. (2010) [7] addressed secure, scalable, and fine-grained access control for outsourced cloud data. Their approach showed that encryption can enforce authorization policies while reducing dependence on a fully trusted storage server. This is particularly relevant to multi-user storage systems, where different users may need distinct rights and where access should be revoked without exposing the data owner's master secret.

Wang et al. (2011) [8] studied public auditability and data dynamics for cloud storage. Their work combined verification mechanisms with support for data modification and illustrated that integrity auditing must remain practical when cloud-resident files are not static. In the proposed methodology, the integrity layer is therefore bound to file versioning: each authorized modification triggers regeneration of the corresponding verification metadata.

Wang et al. (2012) [10] developed mechanisms for secure and dependable cloud storage using distributed verification and erasure-coded data. The study reinforced the importance of lightweight auditing, error localization, and resilience across distributed storage resources. These concepts motivate the separation of confidentiality, integrity, and availability controls rather than treating encryption as a complete storage-security solution.

Xiao and Xiao (2013) [9] surveyed cloud security and privacy issues across confidentiality, integrity, availability, accountability, and privacy. Their analysis highlights that each security attribute requires different controls and that security mechanisms can introduce performance and management trade-offs. For this reason, the methodology in this paper uses symmetric encryption for bulk data, independent integrity metadata for verification, and a separate authorization layer for key release.

Erway et al. (2015) [11] extended PDP to Dynamic Provable Data Possession (DPDP), supporting updates to outsourced data while preserving verifiability. Their work is especially important for real cloud applications, because files are frequently inserted, modified, or deleted.

The proposed design adopts this dynamic perspective by treating integrity metadata as version-dependent and by invalidating stale verification records after each authorized update.

Table 1. Security mechanisms selected for the proposed approach

Security requirement	Selected mechanism	Reason for inclusion
Confidentiality	Client-side AES-256 authenticated encryption	Efficient for large files; plaintext is not exposed to the storage provider.
Integrity	SHA-256 metadata + authenticated encryption tag	Detects unauthorized modification and binds ciphertext to verification data.
Remote verification	PDP/POR-inspired challenge checks	Avoids full file download for routine integrity assurance [4], [5].
Fine-grained access	Policy-controlled key release / optional CP-ABE wrapping	Separates authorization from bulk encryption [6], [7].
Key protection	Independent key vault, rotation and revocation	Cloud compromise does not automatically disclose data-encryption keys.
Dynamic files	Versioned metadata and re-verification after updates	Maintains assurance when files change [8], [11].

Methodology

1. Threat Model and Design Goals

The methodology assumes that the cloud storage provider offers normal storage functionality but cannot be fully trusted with plaintext. An attacker may obtain a copy of stored objects, alter or delete data, steal user credentials, replay an older file version, or gain limited administrative access to storage. The model also considers accidental corruption and provider-side failure. The design does not attempt to protect a client device that is already fully compromised while plaintext and keys are in active use; instead, it minimizes the time and location in which sensitive keys and plaintext exist.

Five design goals are defined. First, confidentiality must hold even if ciphertext is copied from cloud storage. Second, integrity violations must be detectable before decrypted data are accepted. Third, authorization must determine who may obtain a usable data key. Fourth, routine verification should not require downloading complete files. Fifth, changes to stored objects must produce new integrity metadata so that stale or substituted versions can be recognized. These goals reflect the multi-property security view identified by Subashini and Kavitha [2] and Xiao and Xiao [9].

2. Proposed Secure Storage Architecture

The workflow begins at the data owner or trusted client application. A file is assigned a unique object identifier and version number. The client generates a fresh random data-encryption key for the object and encrypts the content using AES-256 in an authenticated mode such as GCM. Authenticated encryption is selected because it produces ciphertext and an authentication tag in one operation; therefore, an attacker who changes protected content cannot normally produce a valid tag without the key. A SHA-256 digest of the protected object and its metadata is also recorded to support indexing, audit logging, and independent verification.

The data-encryption key is never stored next to the ciphertext in ordinary cloud object storage. Instead, the key is kept in a separate key vault or wrapped under a key-encryption key. Where fine-grained sharing is required, the wrapped key can be associated with a role or attribute policy rather than encrypting the entire file with public-key cryptography. This hybrid organization follows the efficiency rationale of cryptographic cloud-storage architectures [3] while retaining the policy flexibility demonstrated by CP-ABE and related access-control work [6], [7].

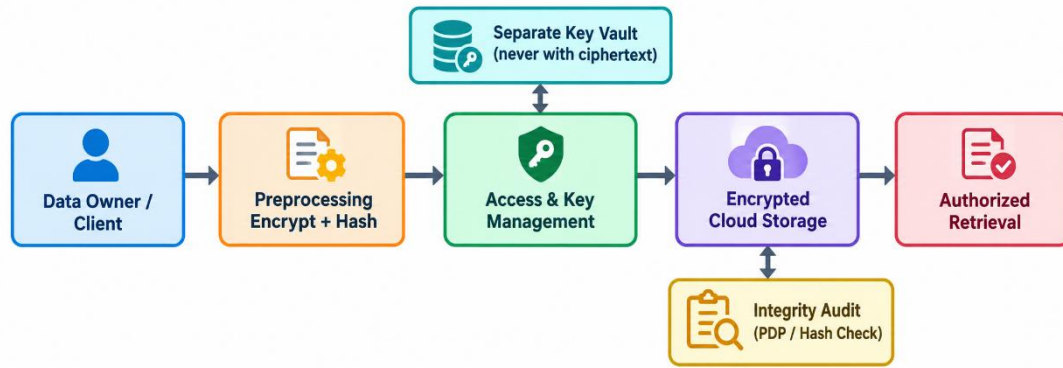


Figure 1. Layered workflow for secure cloud data storage.

Step-by-Step Storage Procedure

Step 1 - Classification and preparation. The client identifies whether the file contains sensitive information, creates an object identifier, assigns the initial version, and removes unnecessary temporary plaintext copies.

Step 2 - Key generation. A cryptographically secure random 256-bit data key and a unique nonce are generated for the object. Reusing a nonce with the same authenticated-encryption key is prohibited.

Step 3 - Client-side protection. The file is encrypted locally. Essential metadata such as object identifier, version, owner identifier, and content type may be supplied as authenticated associated data so that tampering with metadata is also detected.

Step 4 - Integrity record creation. The client stores the authenticated-encryption tag and computes a SHA-256 digest for audit and indexing. For large or long-lived objects, PDP/POR-style block verification metadata can additionally be generated to support remote challenge checks [4], [5].

Step 5 - Key separation and policy binding. The data key is transferred to a logically separate key-management service, encrypted under a master or user-specific key, and associated with an access policy. Revocation changes the ability to recover the data key rather than requiring immediate re-encryption of every stored byte.

Step 6 - Cloud upload. Only ciphertext, non-secret metadata, the nonce, and required verification material are uploaded to cloud storage. The plaintext data key and master key are excluded from this storage location.

Step 7 - Audit and monitoring. Periodic verification challenges, digest checks, access logs, and version checks are used to detect corruption, unauthorized modification, replay, or

unexplained deletion. Public or delegated auditing can be adopted where appropriate, as established by Wang et al. [8].

Step 8 - Authorized retrieval. A requesting user authenticates, the authorization service evaluates policy, and the wrapped data key is released only when access conditions are satisfied. The client verifies integrity before or during decryption and rejects data when verification fails.

Step 9 - Update and deletion. Authorized modification creates a new version, new nonce, new authentication tag, and new verification metadata. Dynamic proof concepts are applicable when outsourced data change frequently [11]. Secure deletion is implemented by deleting or revoking the relevant key material in addition to requesting object deletion from the provider.

3. Verification Criteria

The design is evaluated against five criteria: exposure of plaintext after storage compromise, detectability of unauthorized modification, ability to enforce revocation, cost of routine integrity verification, and support for dynamic files. A successful design should keep stored content unreadable without a separate key, reject modified ciphertext through authentication failure, allow administrators to disable key release for revoked users, and permit audit checks whose transferred data are substantially smaller than the protected file. Dynamic updates must also invalidate old verification metadata. These criteria map directly to the confidentiality, integrity, access-control, and audit problems addressed in earlier cloud-storage research [4], [7], [8], [10], [11].

Results

A design-level evaluation shows that the proposed workflow covers the main storage-security requirements with limited coupling between controls. If an attacker obtains only the

cloud-resident object, the attacker receives ciphertext, a nonce, and non-secret verification information; the data-encryption key remains in a separate trust domain. This reduces the consequence of a storage-only breach because readable content is not stored with the object. Unauthorized ciphertext modification is detected by the authenticated-encryption tag, while versioned hashes and optional PDP/POR verification provide an additional mechanism for identifying corruption, deletion, or replay.

The architecture also improves administrative control over shared data. Authorization is

concentrated at key release rather than at the physical storage layer. As a result, a user can be revoked by preventing recovery of the wrapped data key, while the encrypted object can remain in the cloud. For large files, this avoids the cost of re-encrypting every byte after a routine access-policy change. The use of symmetric encryption for the data path further limits computation compared with applying public-key or attribute-based encryption to complete files. More expensive cryptographic operations are therefore reserved for small keys and policy objects.

Table 2. Design-level result of the proposed secure-storage controls

Evaluation criterion	Without layered controls	Proposed approach
Storage breach	Plaintext or provider-managed keys may expose data	Ciphertext remains unreadable without the separated data key
Unauthorized modification	May remain unnoticed until application use	Authenticated-encryption tag and hash verification detect tampering
User revocation	Often depends only on storage permissions	Key release can be disabled independently of stored ciphertext
Routine integrity checking	Full retrieval can be expensive	Challenge/digest verification reduces unnecessary transfer
Dynamic updates	Old metadata may become inconsistent	New versions regenerate nonce, tag, digest, and audit metadata
Key compromise impact	One long-term key may expose many objects	Per-object keys and rotation reduce the scope of compromise

These results are architectural rather than a claim of provider-independent throughput. Actual performance depends on file size, network bandwidth, hardware acceleration, cloud API latency, key-management implementation, and audit frequency. Nevertheless, the expected cost distribution is favorable: bulk encryption is linear in file size and uses efficient symmetric cryptography, while key wrapping and policy checks operate on small values. Remote integrity mechanisms proposed in prior work specifically aim to avoid the bandwidth cost of downloading complete outsourced data for every verification operation [4], [8], [10].

Conclusion and Discussion

This paper presented an efficient methodology for secure data storage in cloud computing environments using a layered combination of client-side authenticated encryption, independent key management, fine-grained authorization, integrity metadata, and remote auditing. The central security decision is to prevent the cloud storage service from becoming the only trust anchor: encrypted data are stored in one domain, while usable keys and policy decisions are maintained separately. This structure limits information exposure if the storage layer is copied or compromised and

provides a clear location for revocation, rotation, and access logging. The literature up to 2015 shows that no single mechanism addresses every cloud-storage risk. PDP and POR support efficient verification of outsourced data [4], [5]; cryptographic cloud-storage architectures reduce dependence on provider trust [3]; attribute- and policy-based techniques enable fine-grained authorization [6], [7]; public auditing and dependable storage mechanisms improve integrity assurance [8], [10]; and DPDP extends verification to changing files [11]. The proposed methodology integrates these ideas while reserving computationally heavier techniques for keys and policies rather than bulk data. Several limitations remain. Secure key recovery must be carefully governed because loss of all valid key material makes correctly encrypted data inaccessible. Metadata leakage is also possible even when file content is encrypted, so filenames, object labels, and access logs should be minimized or protected where confidentiality requirements are high. Availability still depends partly on provider resilience and backup strategy; encryption cannot recover a file that no longer exists. Finally, client endpoints must be secured because encryption cannot protect plaintext after an authorized client has decrypted it. Future implementations can add hardware-

backed key storage, stronger multi-factor authentication, automated key rotation, distributed backups, and formal measurement of encryption, upload, audit, and recovery overhead under realistic cloud workloads.

References

M. Armbrust et al., "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010. DOI: 10.1145/1721654.1721672.

S. Subashini and V. Kavitha, "A Survey on Security Issues in Service Delivery Models of Cloud Computing," *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 1–11, 2011. DOI: 10.1016/j.jnca.2010.07.006.

S. Kamara and K. Lauter, "Cryptographic Cloud Storage," in *Financial Cryptography and Data Security Workshops*, LNCS 6054, pp. 136–149, 2010. DOI: 10.1007/978-3-642-14992-4_13.

G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pp. 598–609, 2007. DOI: 10.1145/1315245.1315318.

A. Juels and B. S. Kaliski Jr., "PORs: Proofs of Retrievability for Large Files," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pp. 584–597, 2007. DOI: 10.1145/1315245.1315317.

J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-Policy Attribute-Based Encryption," in *2007 IEEE Symposium on Security and Privacy*, pp. 321–334, 2007. DOI: 10.1109/SP.2007.11.

S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing," in *IEEE INFOCOM 2010*, pp. 534–542, 2010. DOI: 10.1109/INFCOM.2010.5462174.

Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 5, pp. 847–859, 2011. DOI: 10.1109/TPDS.2010.183.

Z. Xiao and Y. Xiao, "Security and Privacy in Cloud Computing," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 843–859, 2013. DOI: 10.1109/SURV.2012.060912.00182.

C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, "Toward Secure and Dependable Storage Services in Cloud Computing," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 220–232, 2012. DOI: 10.1109/TSC.2011.24.

C. C. Erway, A. K p  , C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession," *ACM Transactions on Information and System Security*, vol. 17, no. 4, Article 15, pp. 1–29, 2015. DOI: 10.1145/2699909.