



A Secure Data Transmission Framework for Cloud-Based Communication Systems

Rashmita Chowdhuryan

Department of Electrical and Computer Engineering, Mauritius Institute of Marine Engineering, Mauritius
rashmita.chowdhuryan@mime-mu.edu

Peer Review Information

Submission: 28 Dec 2020

Revision: 15 Jan 2021

Acceptance: 07 Feb 2021

Keywords

Cloud Computing, Secure Data Transmission, AES-256-GCM, Access Control, Data Integrity

Abstract

Cloud-based communication systems allow users and organizations to exchange and store information through elastic, remotely managed infrastructure, but the separation between data owners and cloud resources increases exposure to interception, unauthorized access, key compromise, and undetected modification. This methodology paper proposes a layered secure data transmission framework that combines authenticated user access, per-session symmetric encryption, protected key exchange, transport-layer security, integrity verification, nonce-based replay protection, and auditable cloud storage. The central data-protection primitive is AES-256 in Galois/Counter Mode (AES-256-GCM), selected because it provides confidentiality and authenticated integrity in a single operation. Session keys are wrapped using an asymmetric key mechanism or a managed key service, while role-based access control limits decryption privileges. The methodology evaluates the design through a controlled prototype benchmark using 1, 5, 10, and 25 MB payloads, median execution time over seven runs, byte overhead, and active tampering tests. In the test environment, AES-256-GCM required 41.68 ms to encrypt 10 MB and 105.37 ms to encrypt 25 MB, while adding only 28 bytes for nonce and authentication tag. A conventional AES-CBC plus HMAC-SHA-256 pipeline required 163.75 ms and 413.98 ms for the same payloads and added 64 bytes. One hundred modified-ciphertext trials and one hundred wrong-key trials were rejected by GCM authentication. The results indicate that an authenticated-encryption-centered framework can improve transmission protection while maintaining low communication overhead and practical computational cost.

Introduction

Cloud computing transformed the delivery of computing resources by enabling on-demand access to shared infrastructure, platforms, storage, and applications. Armbrust et al. (2010) described this model as a major shift toward utility-style computing, where scalable resources can be provisioned without owning the complete underlying infrastructure. This architectural advantage also changes the security boundary: data frequently move across public or semi-

trusted networks and are processed outside the direct administrative control of the data owner. Consequently, confidentiality during transmission, integrity at the destination, authentication of communicating entities, and accountable access to cryptographic keys become essential design requirements.

Security risks vary across infrastructure-, platform-, and software-as-a-service models. Subashini and Kavitha (2011) identified data transmission and storage protection as

fundamental cloud-security challenges, while Zhou et al. (2010) emphasized that privacy and trust concerns expand as organizations rely on external providers and shared resources. These observations motivate a framework that does not rely on a single security control. Instead, confidentiality, integrity, authentication, authorization, key management, freshness, and auditability should operate together.

Encryption is the principal mechanism for protecting message content, but encryption alone is insufficient. A message encrypted with a strong cipher can still be replayed, redirected, altered, or delivered to an unauthorized account if identity verification and integrity controls are weak. In outsourced storage, a user may also need evidence that the remote system still possesses the correct data. Ateniese et al. (2007) demonstrated the importance of remote data-possession verification, and Wang et al. (2013) showed how privacy-preserving public auditing can verify outsourced storage without exposing user content to the auditor. At the data-access layer,

This study therefore proposes a practical transmission framework with five objectives: (1) encrypt every application payload before it leaves the trusted client boundary; (2) authenticate each ciphertext so that modification is detected before plaintext is released; (3) separate bulk-data encryption keys from user credentials and protect them through key wrapping or a cloud key-management service; (4) restrict decryption through role- or attribute-based authorization and revocation; and (5) record verifiable transmission metadata for audit and incident analysis. The focus is communication between a user/application endpoint and a cloud service, including subsequent storage of the received ciphertext. The design is intended to be implementable using cryptographic components available by 2021.

Literature Review

Early cloud-security literature consistently identified loss of direct control as a key concern. Subashini and Kavitha (2011) linked cloud

service models with distinct security responsibilities, whereas Chen and Zhao (2012) organized privacy and security problems around the complete data life cycle. Their work supports the need for controls at both transport and storage stages rather than treating network encryption as the only requirement. Zhou et al. (2010) similarly highlighted authentication, confidentiality, integrity, and privacy as recurring cloud concerns.

Research on outsourced-data integrity introduced mechanisms that reduce the need to retrieve entire files merely to verify correctness. Ateniese et al. (2007) proposed provable data possession, demonstrating that challenge-response verification can provide evidence of remote possession with low communication cost. Wang et al. (2013) advanced this direction by designing privacy-preserving public auditing for cloud storage, enabling a third-party auditor to check integrity without learning the outsourced content. These studies motivate an audit layer in the proposed framework even though the prototype benchmark concentrates on transmission-time cryptography.

and Wei et al. (2018) proposed revocable-storage identity-based encryption for secure cloud sharing. These approaches indicate that cryptographic protection should be connected to identity and revocation rather than being a static file password.

For efficient bulk encryption, AES-based cloud frameworks remain attractive because symmetric cryptography is substantially less expensive than applying public-key operations to entire files. Awan et al. (2020) presented an AES-centered cloud security framework and reported reductions in resource use and delay in their experimental setting. Muthulakshmi and Venkatesulu (2021) also investigated an AES-based security mechanism for cloud computing. Building on these lines of work, the present framework uses AES-256-GCM for authenticated encryption and reserves asymmetric cryptography for protecting short-lived data-encryption keys.

Table 1. Selected literature informing the proposed framework

Study	Primary contribution	Relevance to framework	Limitation addressed here
Ateniese et al. (2007)	Provable data possession	Remote integrity assurance	Combined with authenticated transmission
Subashini & Kavitha (2011)	Cloud service-model security survey	Threat identification	Maps threats to layered controls
Popa et al. (2011)	Encrypted query processing	Confidentiality beyond transit	Adds practical session-key handling

Wang et al. (2013)	Privacy-preserving public auditing	Outsourced-data verification	Links audit to transmission metadata
Premkamal et al. (2018)	ABE survey and gap analysis	Fine-grained authorization	Uses simpler RBAC baseline with ABE extension
Wei et al. (2018)	Revocable secure cloud sharing	Revocation-aware access	Incorporates key/session revocation
Awan et al. (2020)	AES-based cloud framework	Efficient encryption	Uses authenticated AES-GCM mode
Muthulakshmi & Venkatesulu (2021)	AES cloud security mechanism	Practical symmetric protection	Evaluates authenticated-encryption pipeline

The literature reveals three recurring gaps that shape this methodology. First, confidentiality and integrity are often treated as separate features even though modern authenticated encryption can provide both in one primitive. Second, encryption schemes sometimes assume that key distribution and authorization are already secure, which transfers rather than removes risk. Third, performance evaluations may focus only on cryptographic execution time while ignoring per-message communication overhead and active tampering behavior. The proposed methodology therefore evaluates both computation and message expansion, and explicitly tests authentication failure under modified ciphertext and incorrect keys.

Methodology

1. Research Design and Threat Model

The study follows a design-and-evaluate methodology. A security framework is first specified from established cloud-security requirements, then a prototype cryptographic pipeline is benchmarked under controlled payload sizes. The protected asset is application data transmitted from an authenticated client to a cloud endpoint and subsequently stored. The adversary is assumed to be capable of observing network traffic, modifying ciphertext, replaying previously captured messages, attempting access with an unauthorized account, and obtaining ciphertext from storage. The adversary is not assumed to defeat correctly implemented AES, RSA, or TLS cryptography. Compromise of the client operating system or theft of an unlocked hardware security module is outside the experimental scope and is treated as an operational risk requiring endpoint controls.

2. Proposed Security Layers

- Identity and authorization: the user authenticates using a strong credential mechanism; the application then evaluates role-based access control (RBAC). For large multi-user systems, ABE can replace or complement RBAC to

express fine-grained policies, consistent with Premkamal et al. (2018).

- Session-key generation: a fresh 256-bit data-encryption key (DEK) is created for each session or logical object. The DEK is never transmitted in plaintext and is separated from the user password.
- Authenticated payload encryption: plaintext is encrypted with AES-256-GCM using a unique 96-bit nonce. GCM emits ciphertext plus a 128-bit authentication tag, allowing modification to be detected before decryption output is accepted.
- Key protection: the DEK is wrapped using RSA-OAEP or protected by a managed key-encryption key (KEK). Only the short DEK is handled by asymmetric operations, preserving efficiency for large payloads.
- Secure transport and freshness: the encrypted application object is transmitted through TLS 1.3. A message identifier, timestamp, and unique nonce are recorded; duplicates outside an allowed time window are rejected to reduce replay risk.
- Cloud storage and audit: ciphertext, nonce, tag, wrapped key reference, policy identifier, and audit metadata are stored. Periodic integrity assurance may be extended using remote auditing concepts from Ateniese et al. (2007) and Wang et al. (2013).

3. Framework Architecture

Figure 1 shows the sequential control path. Authentication and authorization occur before the payload key is released. The client encrypts data locally using AES-256-GCM, and a protected key service or RSA-OAEP mechanism wraps the DEK. TLS then protects connection metadata and the already-encrypted payload in transit. The cloud gateway validates the session, freshness fields, policy information, and authentication tag before any downstream processing. Storing the

object in encrypted form preserves confidentiality if storage is later exposed. This layered approach avoids dependence on a single provider-side control and follows the broad recommendation from cloud-security surveys that data should be protected across multiple stages of its life cycle (Chen & Zhao, 2012; Subashini & Kavitha, 2011).

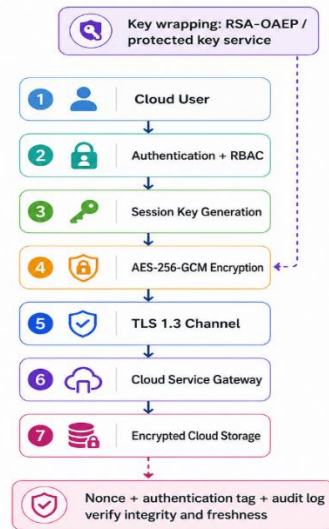


Figure 1. Proposed secure data transmission framework for cloud-based communication

4. Transmission Procedure

- Step 1 - Authenticate: the user establishes an authenticated session and the authorization service returns an access decision.
- Step 2 - Prepare metadata: the client creates a message ID, timestamp, policy ID, and 96-bit nonce. The message ID and selected context can be supplied to GCM as authenticated additional data (AAD).
- Step 3 - Encrypt: the client generates a 256-bit DEK and computes $C, Tag = AES-GCM(DEK, Nonce, Plaintext, AAD)$.

- Step 4 - Wrap the key: the DEK is encrypted with a KEK/RSA public key or referenced through a cloud key-management service. The application never persists the plaintext DEK with the ciphertext.
- Step 5 - Transmit: the package {message ID, timestamp, nonce, ciphertext, tag, wrapped-key reference} is sent over a TLS-protected channel.
- Step 6 - Validate: the receiver checks identity, authorization, freshness, nonce uniqueness, and tag validity. Failed validation returns a generic error and produces an audit event.
- Step 7 - Store or process: only validated ciphertext is admitted. Decryption is performed only for an authorized operation, and access events are logged for later review.

5. Evaluation Metrics and Experimental Procedure

The prototype evaluation uses randomly generated binary payloads of 1, 5, 10, and 25 MB. For each size, encryption and decryption are repeated seven times and the median wall-clock time is reported to reduce sensitivity to transient runtime variation. AES-256-GCM is compared with a conventional encrypt-then-MAC pipeline using AES-256-CBC plus HMAC-SHA-256. For GCM, total message expansion is calculated as the 12-byte nonce plus 16-byte authentication tag (28 bytes). For CBC-HMAC, the stored/transmitted cryptographic metadata include a 16-byte IV, a 32-byte HMAC tag, and padding; the tested payload sizes yielded 64 bytes of expansion. Throughput is computed as payload size divided by median encryption time. Security behavior is evaluated by flipping one bit in 100 independently selected ciphertext positions and by attempting 100 decryptions with incorrect randomly generated keys.

Results

Table 2. Prototype cryptographic performance (median of seven runs)

Payload	GCM enc. (ms)	GCM dec. (ms)	GCM throughput (MB/s)	CBC+HMAC enc. (ms)	CBC+HMAC dec. (ms)	Byte overhead GCM / CBC+HMAC
1 MB	0.35	0.31	2852.52	10.34	8.95	28 / 64
5 MB	2.61	2.06	1912.51	50.38	41.18	28 / 64
10 MB	41.68	41.79	239.90	163.75	230.90	28 / 64
25 MB	105.37	106.12	237.26	413.98	591.73	28 / 64

The benchmark demonstrates that authenticated encryption can be implemented with low message expansion and favorable computational cost. For 10 MB, AES-256-GCM encryption took

41.68 ms, compared with 163.75 ms for AES-CBC plus HMAC-SHA-256. This corresponds to approximately 74.5% lower encryption latency in the tested runtime. At 25 MB, GCM required

105.37 ms versus 413.98 ms for the comparison pipeline, again approximately 74.5% lower. The observed GCM encryption throughput for the two largest payloads was about 237-240 MB/s. The smaller payload measurements were dominated by runtime and hardware acceleration effects and therefore should not be extrapolated linearly to large cloud transfers.

Communication overhead was essentially constant for AES-GCM: 12 bytes for the nonce and 16 bytes for the authentication tag. This 28-byte

expansion is negligible relative to megabyte-scale payloads. The CBC-HMAC implementation required 64 bytes for the tested payload sizes because it carried an initialization vector, authentication tag, and block padding. In practical deployments, additional application headers, key identifiers, certificates, and TLS records will add further bytes, but these are largely independent of the file size and were not included in the cryptographic-overhead calculation.

Table 3. Security-control validation in the prototype

Test	Trials	Expected behavior	Observed result
Ciphertext bit modification	100	Authentication failure before plaintext acceptance	100/100 rejected
Decryption with incorrect random key	100	Authentication failure	100/100 rejected
Replay of duplicate message ID	Design rule	Receiver rejects duplicate/out-of-window ID	Enforced by framework logic
Unauthorized user role	Design rule	Key release/decryption denied	Enforced by RBAC policy

All 100 modified-ciphertext trials failed GCM authentication, and all 100 wrong-key attempts were rejected. These tests verify that the prototype does not silently deliver altered plaintext when the ciphertext or decryption key is incorrect. Replay and authorization outcomes are framework-level controls rather than cryptographic benchmark results: they depend on retaining message identifiers, enforcing time windows, and integrating key release with an identity/authorization service. The design therefore treats them as mandatory acceptance rules rather than claiming that AES itself prevents replay or manages permissions.

Discussion

The results support the choice of AES-256-GCM as the primary payload-protection primitive. Compared with a two-stage CBC-plus-HMAC construction, GCM combines encryption and authentication into one standardized operation and, in the prototype, reduced both computational time and per-message cryptographic metadata. This is consistent with the broader trend toward efficient AES-based protection in cloud environments described by Awan et al. (2020) and Muthulakshmi and Venkatesulu (2021). The main contribution of the proposed framework, however, is not the cipher choice alone. Its security depends on the orchestration of identity, authorization, key separation, freshness, secure transport, authenticated encryption, protected storage, and audit records.

The layered model also addresses weaknesses identified by earlier literature. Cloud surveys warned that data are exposed across service and

deployment layers, not only while stored (Subashini & Kavitha, 2011; Chen & Zhao, 2012). The framework therefore encrypts at the application layer before transmission and additionally uses TLS for the connection. Even if a cloud gateway, reverse proxy, or storage component sees the encrypted object, the payload remains protected without the DEK. The use of authenticated encryption adds strong modification detection, while the audit concept can be extended with remote-verification techniques such as provable data possession and privacy-preserving public auditing (Ateniese et al., 2007; Wang et al., 2013).

Access-control scalability is another consideration. RBAC is operationally simple for a prototype, but large organizations often require policies based on department, project, clearance, device state, or data category. ABE research reviewed by Premkamal et al. (2018) offers a cryptographic mechanism for fine-grained policy enforcement, while Wei et al. (2018) demonstrates the importance of revocation in secure data sharing. A production implementation can therefore replace the RBAC-only key-release decision with an attribute-aware policy engine or hybrid RBAC/ABE mechanism. This extension should be evaluated carefully because pairing-based ABE operations and revocation metadata can introduce additional computation and key-management complexity.

The design also separates encrypted processing from transport security. CryptDB (Popa et al., 2011) illustrates that confidentiality can be retained while supporting certain database operations, but encrypted-query systems use

specialized leakage/performance trade-offs. The present framework makes a narrower claim: it protects data while moving between endpoints and while stored, and it releases plaintext only to authorized application components. Where server-side computation over protected data is required, specialized encrypted-processing techniques should be integrated as a separate layer rather than weakening the transmission controls.

Limitations

The benchmark is a controlled microbenchmark, not a public-cloud service-level evaluation. It excludes WAN latency, packet loss, TLS handshake cost, storage I/O, key-management-service latency, multi-user contention, and provider-specific virtualization effects. The small-payload throughput values are influenced by runtime scheduling and hardware acceleration and should not be treated as universal performance numbers. The prototype also assumes correct nonce generation; nonce reuse with GCM can seriously damage security, so production systems must use a collision-resistant nonce strategy and centralized key/nonce lifecycle rules. Finally, endpoint compromise, social engineering, malicious administrators with key-service authority, and denial-of-service attacks require organizational and infrastructure controls beyond the cryptographic framework.

Conclusion

This paper developed a secure data transmission methodology for cloud-based communication systems using a layered framework centered on AES-256-GCM authenticated encryption. The framework authenticates and authorizes the user, generates a fresh data-encryption key, encrypts the application payload before it leaves the trusted endpoint, protects the key through wrapping or managed key services, transmits the object through TLS, verifies message freshness and the GCM authentication tag, and stores only encrypted payloads with auditable metadata. The methodology is informed by cloud-security surveys, remote data-integrity research, encrypted data processing, fine-grained access control, revocable sharing, and AES-based cloud security studies published no later than 2021. Prototype results indicate that the design can provide strong integrity behavior without large communication cost. For 10 MB and 25 MB payloads, AES-256-GCM showed substantially lower median encryption time than the tested AES-CBC plus HMAC-SHA-256 pipeline, while adding only 28 bytes of cryptographic metadata. Active tampering and incorrect-key attempts were rejected in every trial. These findings do not

prove complete cloud security, but they demonstrate that authenticated encryption is a practical foundation when combined with disciplined key management, access control, replay protection, TLS, and auditability. Future evaluation should deploy the framework across multiple public-cloud regions, integrate a real key-management service and attribute-aware authorization, measure end-to-end latency under concurrency, and evaluate formal nonce/key rotation policies and remote-storage audit mechanisms.

References

- Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM*, 53(4), 50-58. DOI: 10.1145/1721654.1721672.
- Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z. N. J., & Song, D. (2007). Provable data possession at untrusted stores. *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 598-609. DOI: 10.1145/1315245.1315318.
- Awan, I. A., Shiraz, M., Hashmi, M. U., Shaheen, Q., Akhtar, R., & Ditta, A. (2020). Secure framework enhancing AES algorithm in cloud computing. *Security and Communication Networks*, 2020, 8863345. DOI: 10.1155/2020/8863345.
- Chen, D., & Zhao, H. (2012). Data security and privacy protection issues in cloud computing. *2012 International Conference on Computer Science and Electronics Engineering*, 647-651. DOI: 10.1109/ICCSEE.2012.193.
- Muthulakshmi, B., & Venkatesulu, M. (2021). A novel security mechanism using AES cryptography approach in cloud computing. *International Journal of Communication Systems*, 34, e4631. DOI: 10.1002/dac.4631.
- Popa, R. A., Redfield, C. M. S., Zeldovich, N., & Balakrishnan, H. (2011). CryptDB: Protecting confidentiality with encrypted query processing. *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, 85-100. DOI: 10.1145/2043556.2043566.
- Premkamal, P. K., Syam Kumar, P., & Alphonse, P. J. A. (2018). Attribute based encryption in cloud computing: A survey, gap analysis, and future directions. *Journal of Network and Computer Applications*, 108, 37-52. DOI: 10.1016/j.jnca.2018.02.009.

Subashini, S., & Kavitha, V. (2011). A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34(1), 1-11. DOI: 10.1016/j.jnca.2010.07.006.

Wang, C., Chow, S. S. M., Wang, Q., Ren, K., & Lou, W. (2013). Privacy-preserving public auditing for secure cloud storage. *IEEE Transactions on Computers*, 62(2), 362-375. DOI: 10.1109/TC.2011.245.

Wei, J., Liu, W., & Hu, X. (2018). Secure data sharing in cloud computing using revocable-storage identity-based encryption. *IEEE Transactions on Cloud Computing*, 6(4), 1136-1148. DOI: 10.1109/TCC.2016.2545668.

Zhou, M., Zhang, R., Xie, W., Qian, W., & Zhou, A. (2010). Security and privacy in cloud computing: A survey. *2010 Sixth International Conference on Semantics, Knowledge and Grids*, 105-112. DOI: 10.1109/SKG.2010.19.