

Secure Multi-Cloud Data Storage with Encryption: A Review

Yash Sukhdev Jadhav, Prof. Bangar A.P., Prof. Bhosale S.B., Dr. A. A. Khatri

Computer engineering
Jai hind college of engineering, Pune , India

Abstract- Storing data with a single cloud provider creates a single point of failure and ties an organisation to one vendor's security posture. Multi-cloud storage spreads data over several providers, but it also multiplies the places where keys, ciphertexts and integrity evidence must be protected. This paper reviews four recent research contributions that approach the problem from complementary angles: hybrid authenticated encryption with sliced storage (CloudLock), publicly verifiable equality testing over encrypted replicas (PVPKEET), blockchain-coordinated, collusion-resistant consistency verification (MDCV), and password-protected key management that survives provider collusion (PECS). We compare their system models, threat assumptions, cryptographic tools, reported costs and limitations, and we outline open issues such as dynamic data support, trusted middleware, and post-quantum readiness.

Keywords— multi-cloud storage, data encryption, integrity verification, key management, blockchain, collusion resistance, authenticated encryption

I. INTRODUCTION AND RESEARCH OBJECTIVES

Organisations now routinely keep information with more than one cloud service provider (CSP). The motivation is practical: a regional outage at one vendor need not interrupt service, storage prices can be balanced, and regulation may require data to stay in particular jurisdictions [8], [9]. The price of this flexibility is a larger attack surface. Every additional provider is another party that may leak, corrupt, or silently lose a replica, and the keys that protect the data must be handled in a way that does not hand control back to any single party.

Encryption before upload is the accepted baseline, yet it only moves the difficulty. Three questions follow. How should encrypted data be divided and placed across providers? How can an owner learn that all replicas are still identical without downloading and decrypting them? And who holds the keys, so that neither a provider nor the key service can read the data on its own? Recent literature answers these questions separately, and the answers are rarely compared side by side.

This review therefore examines four studies published between 2024 and 2026 [1]–[4]. Section II

gives the background, Section III describes each approach, Section IV compares them, Section V lists open problems and Section VI concludes.

II. BACKGROUND

1. Multi-Cloud Storage Model

A typical deployment has a data owner, several independent CSPs, and some coordinating component. Depending on the design, the coordinator may be a middleware unit, a key management server (KMS), a third-party auditor, or a smart contract. Fig. 1 shows the generic flow shared by the reviewed schemes: data is optionally sliced, encrypted, spread over providers, and later checked or retrieved with the help of a key service and a verifier.

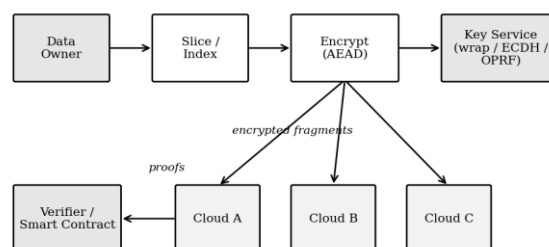


Fig. 1. Generic workflow of secure multi-cloud storage

2. Threats Considered

The studies converge on a few recurring threats: eavesdropping on data in transit, unauthorised reads by outsiders or insiders, silent corruption or deletion of a replica, forged audit evidence, leakage of a data encryption key together with stored ciphertext, and collusion between providers or between a provider and the key service. Most analyses use a semi-honest (honest-but-curious) adversary, meaning that parties follow the protocol but may try to learn more than they should.

3. Cryptographic Building Blocks

- **Authenticated encryption:** Authenticated encryption with associated data (AEAD) protects confidentiality and detects tampering in one primitive. ChaCha20-Poly1305 and AES-GCM are the common instances; both appear in the reviewed work [1], [4].
- **Key agreement and wrapping:** Elliptic-curve Diffie–Hellman (ECDH) lets two parties derive a shared secret over an open channel, and a data key can then be wrapped under that secret or under a key held by a key service.
- **Equality testing:** Public key encryption with equality test (PKEET) allows a designated tester to learn whether two ciphertexts hide the same message without decrypting them [11]. This suits replica comparison.
- **Oblivious PRF:** An oblivious pseudorandom function (OPRF) lets a client obtain a keyed function of its input from a server, while the server learns neither the input nor the output [10]. It can harden a low-entropy password into a strong key.
- **Provable data possession:** Provable data possession (PDP) and proofs of retrievability (PoR) allow integrity checks without full retrieval [5], [6], but they were designed for one provider and do not compare replicas across providers on their own.

III. REVIEW OF EXISTING APPROACHES

1. CloudLock: Hybrid Encryption with Sliced Storage

Bharot et al. [1] place a middleware, the Secure Cloud Storage Enabler (SCSE), between users and

providers. For an upload, the file is divided into parts, each part is indexed in a registry that records which cloud holds which piece, and every part is encrypted with ChaCha20-Poly1305 before being stored on services such as Dropbox, Google Drive and Mega. For retrieval, a key dissemination system runs an ECDH exchange between the SCSE unit and the requesting user, and the file key is wrapped under the derived secret using AES before delivery. The user then fetches the fragments, decrypts them and merges them by index.

Reported results show that encryption time grows with file size and file type, and that the authenticated cipher is slower than plain AES, DES or ChaCha20 in isolation. For a 1 MB text file, encryption took about 0.309 s with the proposed cipher against 0.008 s for AES. Once upload, download and key distribution are included, the end-to-end turnaround for a 1 GB CSV file was about 87.97 s, essentially the same as AES (88.21 s) and ChaCha20 (87.30 s). The main limitation is the assumption that the SCSE unit is fully trustworthy: it handles slicing, encryption and key wrapping, so its compromise would expose everything.

2. PVPKEET: Publicly Verifiable Equality Testing

Li et al. [3] target the replica-consistency question directly. Each copy of a file is encrypted under a different user key but also under the cloud server's public key, so that only the server can run the equality test, which removes the need to hand out trapdoors. The server compares two ciphertexts, and if their plaintexts match it produces a proof. Anyone holding the ciphertexts, the proof and the public keys can check it through a set of pairing equations, without decrypting and without the server's secret. The scheme also extends to homomorphic evaluation across ciphertexts under different keys. Security is argued as IND-CPA under the computational Diffie–Hellman assumption in the random oracle model, together with a contradiction argument showing that a forged proof for unequal messages fails verification. The authors themselves note two weaknesses: only chosen-plaintext security is achieved, and the scheme is less efficient than designs built for a single function. In their evaluation,

communication stayed around 5000 bytes even with 11 cloud servers.

3. MDCV: Decentralised, Collusion-Resistant Verification

Qin et al. [2] remove the trusted coordinator altogether. A data owner submits a verification request as a blockchain transaction. Each CSP derives its challenge from the transaction identifier through a pseudo-random function, computes a proof over its own replica using its private key, and posts it. A smart contract then checks the proofs pairwise with bilinear-pairing equations and records the verdict publicly. Because every proof is bound to the provider's secret, providers that collude cannot fabricate one another's evidence.

Proofs rely on the discrete logarithm and computational Diffie–Hellman assumptions. In experiments on Hyperledger Fabric, proof generation and verification cost roughly 12 ms each and stayed flat as the number of challenged blocks increased, while two baselines grew linearly. On-chain verification took about 0.1 s against roughly 4.8 s for a blockchain PDP baseline, and the contract sustained 100 transactions per second compared with 0.5–2 for that baseline. Costs do grow with the number of CSPs, and proof generation grows with file size. The scheme handles static data only; dynamic updates are left for future work.

4. PECS: Password-Protected Key Management

Qin et al. [4] study where keys live. In a conventional design the provider stores the ciphertext and a wrapped data key, while a KMS holds the key-encryption key. Two failures follow: an attacker who obtains both the data key and the ciphertext can read the data, and a provider and KMS that collude can unwrap everything. PECS addresses both. The provider re-encrypts every upload under its own key so that stolen original ciphertexts are useless by themselves. The client wraps each per-file key through an oblivious key management exchange and then encrypts the wrap under a public key derived from the user's password, hardened through an OPRF run with the provider. The client therefore stores nothing but a memorised password.

The security proof uses a game-hopping argument that reduces to password guessing, PKE-CCA security and AEAD security. As the authors acknowledge, a malicious provider can still mount offline password guessing against stored ciphertexts, so password strength matters. With AES-GCM-256, SHA-256, Cramer–Shoup and Curve25519, they report about 0.66 ms client computation for an upload and 1.19 ms for a download, with total times near 9.76 ms and 37.55 ms once network latency is included. The model assumes semi-honest servers and does not let the client check a malicious KMS.

IV. COMPARATIVE ANALYSIS

Table I summarises the four studies. They are not direct competitors: CloudLock concerns placement and sharing, PVPKEET and MDCV concern consistency evidence, and PECS concerns key custody. A complete system would combine ideas from several of them.

Table 1. Comparison of Reviewed Schemes

Aspect	CloudLock [1]	PVPKEET [3]	MDCV [2]	PECS [4]
Main goal	Sliced, encrypted storage and sharing	Replica check on ciphertext	Decentralised consistency audit	Key custody and leakage resilience
Core tools	ChaCha20-Poly1305, ECDH, AES	PKEET, pairings	Blockchain, PRF, pairings	AEAD, OPRF, PKE
Trust anchor	SCSE middleware	Cloud server tests	Smart contract	Client password
Public check	No	Yes	Yes	No
Collusion handling	Not covered	Not covered	Proofs bound to keys	Provider + KMS
Proof basis	Standard AEAD bounds	CPA, CDH (ROM)	DL, CDH	CCA PKE, AEAD, ROM
Key weakness	Trusted SCSE	CPA only	Static data only	Offline guessing

Cost figures cannot be ranked across the papers because the testbeds differ: a Python prototype using three consumer storage services, a pairing library on a C++ platform, a Go and Fabric setup on an Apple M2 machine, and a browser client talking to cloud servers over HTTPS. The numbers are better read as evidence that each design is practical in its own setting than as a leaderboard.

Lessons: First, authenticated encryption should be the default, since confidentiality alone does not detect tampering. Second, proof size and cost that are independent of the challenged block count, as in MDCV, make frequent audits affordable. Third, moving trust from a single coordinator to a blockchain or to the user's own secret removes single points of failure but adds latency or usability costs. Fourth, key and ciphertext custody should be separated so that one leak is not enough.

Open Challenges

- **Dynamic data:** MDCV is limited to static files, and PVPKEET targets stored copies. Efficient proofs for insertion, deletion and modification across providers remain open.
- **Untrusted middleware:** CloudLock assumes a secure SCSE. Replacing it with distributed or threshold components, or verifying it, would strengthen the model.
- **Stronger adversaries:** Most proofs assume semi-honest providers. Fully malicious servers, including a KMS that returns wrong keys, are largely unaddressed.
- **Stronger security notions:** PVPKEET offers only IND-CPA. A CCA-secure variant would suit real deployments better.
- **Post-quantum readiness:** ECDH, pairings and discrete-logarithm assumptions are vulnerable to large quantum computers, so migration paths to lattice-based or hash-based constructions are needed.
- **Fair benchmarking:** A common dataset, hardware profile and provider mix would allow honest comparison between schemes.

VI. CONCLUSION

Securing multi-cloud storage requires protection at three levels: how data is encrypted and placed, how its continued integrity and consistency are shown, and how keys are held. The four studies reviewed here each advance one level and show reasonable cost in their evaluations. None covers all three together under a strong adversary, which points to the main direction for future work: an integrated design combining authenticated encryption, publicly verifiable and collusion-resistant audits, and password- or threshold-based key custody, extended to dynamic data and quantum-resistant primitives.

Acknowledgment

The authors thank their institution for support and the reviewers of the cited works for the publicly available research on which this review is based.

REFERENCES

1. N. Bharot, N. Mehta, J. G. Breslin and P. Verma, "CloudLock: Secure data sharing using a hybrid cryptosystem in multicloud data storage," *Cluster Computing*, vol. 28, art. 464, 2025.
2. X. Qin, K. Zhu and Y. Huang, "MDCV: A decentralized and collusion-resistant consistency verification scheme for multi-cloud storage," *Cybersecurity*, vol. 9, art. 125, 2026.
3. W. Li, W. Susilo, C. Xia, L. Huang, F. Guo and T. Wang, "Secure data integrity check based on verified public key encryption with equality test for multi-cloud storage," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 6, pp. 5359–5373, 2024.
4. S. Qin, R. Zhang and H. Ma, "Securing leakage and collusion attacks on encrypted cloud storage using memorizable passwords," *Cybersecurity*, vol. 9, art. 52, 2026.
5. G. Ateniese et al., "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Computer and Communications Security*, 2007, pp. 598–609.
6. H. Shacham and B. Waters, "Compact proofs of retrievability," *J. Cryptology*, vol. 26, no. 3, pp. 442–483, 2013.

7. A. Bessani, M. Correia, B. Quaresma, F. André and P. Sousa, "DepSky: Dependable and secure storage in a cloud-of-clouds," *ACM Trans. Storage*, vol. 9, no. 4, 2013.
8. M. A. AlZain, E. Pardede, B. Soh and J. A. Thom, "Cloud computing security: From single to multi-clouds," in *Proc. 45th Hawaii Int. Conf. System Sciences*, 2012, pp. 5490–5499.
9. J.-M. Bohli, N. Gruschka, M. Jensen, L. L. Iacono and N. Marnau, "Security and privacy-enhancing multicloud architectures," *IEEE Trans. Dependable Secure Comput.*, vol. 10, no. 4, pp. 212–224, 2013.
10. S. Jarecki, H. Krawczyk and J. Resch, "Updatable oblivious key management for storage systems," in *Proc. ACM SIGSAC Conf. Computer and Communications Security*, 2019, pp. 379–393.
11. G. Yang, C. H. Tan, Q. Huang and D. S. Wong, "Probabilistic public key encryption with equality test," in *Proc. CT-RSA*, Springer, 2010, pp. 119–131.
12. A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.