INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

ORIGINAL ARTICLE

Check for updates

OPEN ACCESS

Received: 13/10/2026

Accepted: 24/10/2026

Published: 11/11/2026

Citation: Surendhar Kumar R, Manikandasaran SS (2026) Protected-Secret Convergent Encryption with Ciphertext-Token Verification for Secure Cloud Deduplication. Indian Journal of Science and Technology 19(40): 2745-2756. https://doi.org/10.17485/IJST/v19i40.1358

* **Corresponding author**.

surendharkumar.r@gmail.com

Funding: None

Competing Interests: None

Copyright: © 2026 Surendhar Kumar & Manikandasaran. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Published By Indian Society for Education and Environment (iSee)

ISSN
Print: 0974-6846
Electronic: 0974-5645

Protected-Secret Convergent Encryption with Ciphertext-Token Verification for Secure Cloud Deduplication

R Surendhar Kumar[1]*, S S Manikandasaran[1]

1 Research Scholar, PG and Research Department of Computer Science, Adaikalamatha College, Vallam, Thanjavur. Affiliated to Bharathidasan University, Tiruchirappalli, Tamil Nadu, India

Abstract

Objectives: To develop and evaluate an Advanced Convergent Encryption (ACE) method that preserves exact encrypted deduplication while preventing an external observer from deriving the encryption key from a guessed file alone. The method integrates protected-secret key derivation, deterministic authenticated encryption, ciphertext-token verification, and ownership-aware references. **Method:** For plaintext file or block P, ACE computes the SHA-256 content digest h. The protected service derives the 256-bit key K by applying HMAC-SHA-256 to a domain label and h under server secret S. AES-SIV produces deterministic authenticated ciphertext, and SHA-256 of the ciphertext forms the repository token. A reproducible single-host experiment used deterministic 1, 5, 10, and 25 MiB files, 30 measured runs per method and size after three warm-up runs, and content-only MLE and 2048-bit RSA blind-OPRF controls. Mean, sample standard deviation, throughput, storage saving, and two-sided Welch tests were evaluated. **Findings:** Across the four size-level means, ACE required 24.859 ms for end-to-end in-memory upload, compared with 24.004 ms for content-only MLE and 47.485 ms for RSA-OPRF. The 3.56% overhead relative to MLE was not significant at any size ($p \geq 0.071$), whereas ACE reduced upload time by 47.65% relative to RSA-OPRF ($p < 0.001$ at every size). ACE key derivation averaged 6.730 ms, 77.19% below RSA-OPRF. Authorized retrieval averaged 17.559 ms for ACE and 40.726 ms for RSA-OPRF. ACE correctly matched all 120 exact-duplicate conditions and separated all 120 one-byte-modified conditions. In the controlled corpus, eight unique ciphertext objects replaced twelve undeduplicated objects, giving 33.33% ciphertext-storage saving before metadata. **Novelty:** ACE combines a secret-dependent deterministic key, misuse-resistant authenticated encryption, ciphertext-derived duplicate tokens, and authorization-gated ownership pointers in one symmetric-key path. Unlike content-only convergent encryption, a guessed plaintext is insufficient for offline key reconstruction while the protected secret remains unavailable. Unlike the evaluated RSA-OPRF control, ACE avoids public-key operations. The method does not claim oblivious key derivation and explicitly retains equality leakage inherent to exact deduplication.

Keywords: Convergent Encryption; Encrypted Deduplication; Deterministic Authenticated Encryption; Ciphertext Token; Cloud Storage Security; Message-Locked Encryption

1 Introduction

Cloud deduplication removes repeated files or blocks, but conventional randomized encryption prevents the storage service from recognizing equal plaintexts. Secure deduplication therefore requires a controlled equality signal while protecting content, keys, and ownership. Recent work has improved re-encryption efficiency and verifiable duplicate decisions, yet these protections add key-update, proof, or metadata costs [1], [2]. Convergent and message-locked encryption preserve equality by deriving encryption material from content; however, a low-entropy or predictable file can be guessed, hashed, and tested offline when the key is content-only. A recent convergent-encryption study demonstrates the continuing practical interest in this model, while password-protected key constructions show that an additional secret or server contribution can strengthen it [3], [4].

Current schemes address different parts of the problem. Tang et al. combine access control and key management with lightweight deduplication [5]. FuzzyDedup extends protection to similar content and explicitly considers false ownership claims [6]. LSDedup applies confidentiality levels and layered protection [7], while dynamic access-control systems connect duplicate storage with changing authorization [8]. Ownership verification has also advanced through full-file challenge mechanisms and blind bidirectional protocols [9], [10]. These methods strengthen important controls, but they do not remove the need to balance server trust, public-key computation, ownership evidence, and the equality leakage required by exact deduplication.

Key-management research further illustrates this trade-off. Learning-assisted key generation and deduplication can reduce selected performance costs, but it introduces a model lifecycle and additional components [11]. Password-protected and subversion-resistant constructions provide stronger guarantees under key-server compromise models at the cost of more involved protocols [12]. Autoencoder-derived summary tags reduce comparison work in randomized message-locked encryption but require model training and storage [13]. End-edge-cloud collaboration distributes secure deduplication functions across tiers [14]. Recent trusted-key and cloud-optimization models likewise demonstrate the value of external key support and adaptive duplicate identification [15], [16]. Multilayer cryptosystems, integrated cloud-edge systems, encrypted-key deduplication, and multi-key management broaden the design space but add layers, distributed services, or key state [17], [18], [19], [20].

1.1 Research Gap and Contributions

Table 1 positions the present contribution against representative recent approaches. The identified gap is a reproducible, symmetric-key path that (i) makes the convergent key dependent on a protected service secret rather than content alone, (ii) uses deterministic authenticated encryption, (iii) verifies exact duplicates using a ciphertext-derived token, and (iv) creates an ownership reference only after authorization. The objective is not to replace oblivious or formally subversion-resistant protocols. It is to measure the cost and security boundary of a simpler protected-secret alternative.

Table 1. Research gap relative to recent secure-deduplication approaches

Approach	Principal protection	Duplicate or ownership control	Limitation relevant to this study
Convergent encryption [3]	Content-derived key	Deterministic equality	Predictable content remains guessable when no independent secret contributes to the key
Password-protected key [4], [12]	Password/server-assisted derivation	Secure key recovery	Stronger compromise model, but greater protocol and key-service complexity
Lightweight access and key management [5], [8]	Authorization and managed keys	Dynamic access control	Additional authorization and key-management machinery
FuzzyDedup and layered deduplication [6], [7]	Similarity or confidentiality-aware protection	Fuzzy matching or protection levels	Classification, fuzzy matching, or multilayer processing
Ownership-verification protocols [9], [10]	Interactive proof of possession	Strong duplicate-claim validation	Focuses on ownership proof rather than a complete deterministic encryption-to-token path
Learning-assisted tags or keys [11], [13]	Learned key or summary representation	Tag filtering and deduplication	Training, model storage, and model-governance overhead
Proposed ACE	Protected HMAC key, AES-SIV, ciphertext token	Authorization-gated ownership pointer	Trusts the protected service, is not oblivious, and leaks accepted ciphertext equality

The paper makes four contributions. First, it defines protected-secret convergent key derivation with explicit domain separation. Second, it connects deterministic authenticated ciphertext to a repository token and an authorization-gated ownership pointer. Third, it gives a threat-oriented analysis that distinguishes offline resistance from online oracle abuse, service-secret compromise, and unavoidable equality leakage. Fourth, it reports a reproducible 30-run benchmark with raw observations, variability, statistical tests, retrieval time, throughput, and storage saving.

2 Methodology

2.1 System and Threat Model

The system comprises an authenticated client, protected key service, ciphertext-token repository, cloud ciphertext store, and ownership-metadata repository. The client supplies a file or block. The key service retains secret S and performs deterministic derivation. The token repository maps a ciphertext token to an object identifier. The ownership repository maps authenticated identities to authorized object references and protected content records. The cloud can observe object size, access timing, and equality among accepted tokens, but it does not receive plaintext, S, or a plaintext-derived key.

The adversary may read stored ciphertext, guess predictable files, replay observed tokens, submit a false duplicate claim, manipulate ciphertext, or request an object without authorization. Authenticated and confidential service channels are assumed. The protected key service and ownership repository enforce access policy. Exact deduplication necessarily exposes equality for accepted objects. Denial of service, unrestricted online access to the derivation service, compromise of S, and traffic-analysis resistance are outside the demonstrated guarantee; operational controls for these risks are discussed in Section 3.7.

2.2 System Architecture

Figure 1 shows the overall service boundaries and data flow. The client sends authenticated upload or retrieval requests; cryptographic derivation occurs in the protected key service; duplicate decisions use the token repository; ciphertext and ownership records are stored separately.

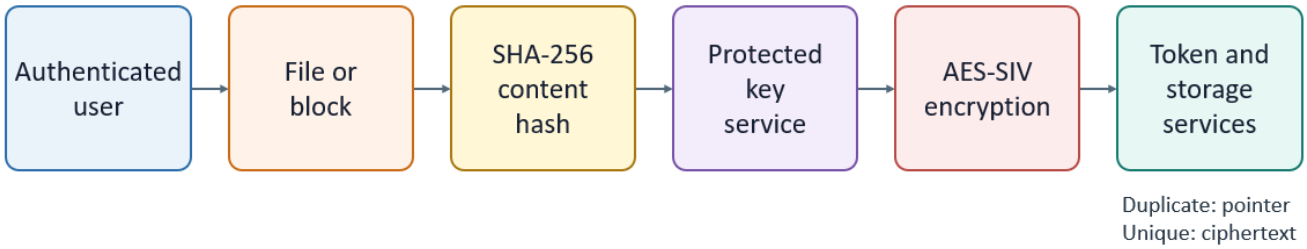


Fig 1. Overall architecture of protected-secret encryption for secure cloud deduplication.

Figure 2 details protected key generation and deterministic authenticated encryption. The protected secret is combined with the content hash through HMAC-SHA-256, and the resulting 32-byte value is used by AES-SIV.

Figure 3 shows ciphertext-token verification. A token match selects the duplicate path only after the duplicate claim is validated; otherwise, the ciphertext is stored as a new object.

Figure 4 presents authorized retrieval. Ownership is verified before the protected record is opened, the key is rederived, and authenticated decryption is attempted.

2.3 Protected Convergent-Key and Ciphertext Model

Let P be a plaintext file or block, S a uniformly generated service secret retained in the protected key service, AD stable non-personal associated data containing the scheme identifier and version, K a 256-bit AES-SIV key, C the authenticated ciphertext, and T the duplicate-verification token. Concatenation is denoted by $\|$. The content fingerprint is

$$h = \text{SHA-256}(P) \tag{1}$$

The protected convergent key is derived with domain separation:

$$K = \text{HMAC-SHA-256}\,(S, \text{"ACE-Key"} \,\|\, h) \tag{2}$$

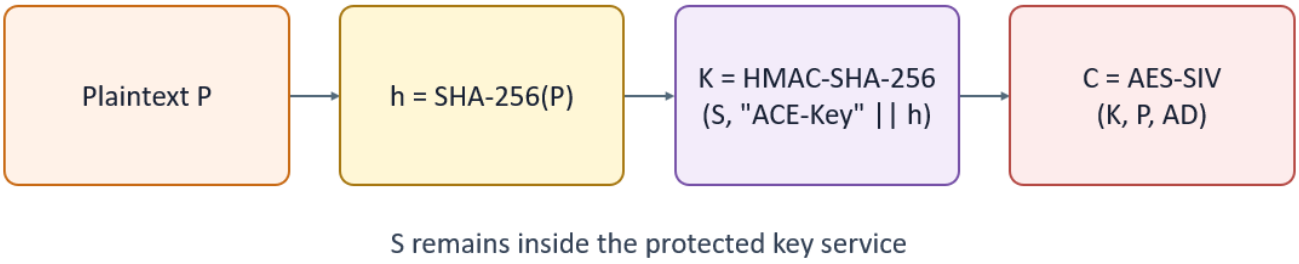


Fig 2. Protected convergent-key generation and encryption process.

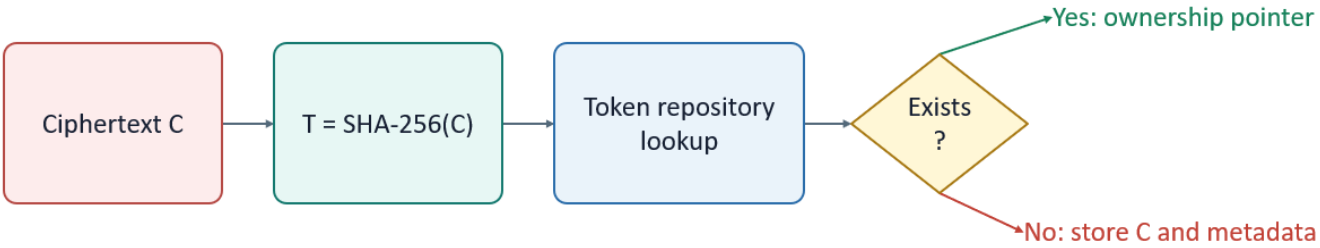


Fig 3. Ciphertext-token duplicate-verification process.

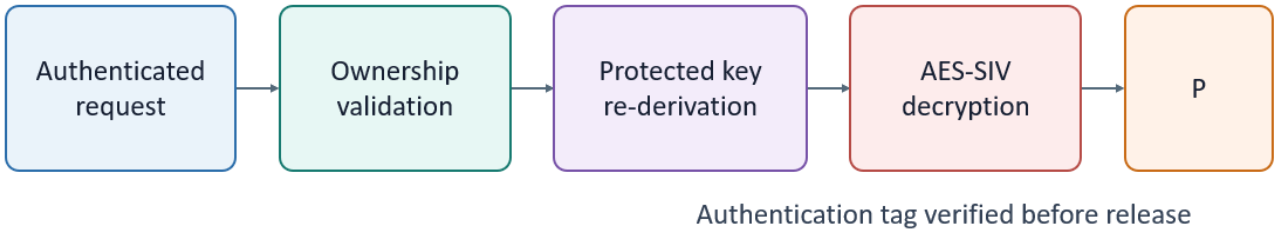


Fig 4. Authorized retrieval and decryption process.

AES-SIV is invoked with stable associated data:

$$C = \text{AES-SIV-Enc}\,(K, P, \text{AD}) \tag{3}$$

The repository token is computed from the complete authenticated ciphertext:

$$T = \text{SHA-256}(C) \tag{4}$$

For fixed S and AD, equal plaintext produces equal h, K, C, and T, enabling exact deduplication. A change to P changes the content hash and, except with negligible collision probability, changes the derived key, ciphertext, and token. User identity is excluded from AD because including it would deliberately prevent cross-user equality. Authorization is instead enforced by ownership metadata.

The ciphertext-storage saving for a workload is

$$\text{Saving}(\%) = \frac{S_{base} - S_{dedup}}{S_{base}} \times 100 \tag{5}$$

where S_{base} is ciphertext storage without deduplication and S_{dedup} is storage after duplicate ciphertexts are replaced by references. Metadata is reported separately because its size is deployment-dependent.

2.4 Algorithms

Algorithm 1. Protected convergent-key generation and encryption
Input: authenticated identity u, plaintext P, protected secret S, stable associated data AD
Output: authenticated ciphertext C and protected content record r

1. if AuthorizeUpload(u) = false, return ACCESS_DENIED
2. h <- SHA-256(P)
3. K <- HMAC-SHA-256(S, ”ACE-Key” || h)
4. C <- AES-SIV.Enc(K, P, AD)
5. r <- ProtectContentRecord(u, h, AD)
6. erase K from working memory
7. return (C, r)

Algorithm 2. Ciphertext-token generation and duplicate verification
Input: authenticated identity u, ciphertext C, protected record r
Output: storage decision and object identifier oid

1. T <- SHA-256(C)
2. oid <- TokenRepository.Lookup(T)
3. if oid exists then
4. if ValidateDuplicateClaim(u, T) = false, return CLAIM_REJECTED
5. OwnershipRepository.AddPointer(u, oid, r)
6. return DUPLICATE_POINTER_CREATED
7. else
8. oid <- CiphertextStore.Put(C)
9. TokenRepository.Insert(T, oid)
10. OwnershipRepository.Create(u, oid, r)
11. return UNIQUE_CIPHERTEXT_STORED

Algorithm 3. Authorized retrieval and decryption
Input: authenticated identity u and object identifier oid
Output: plaintext P or failure

1. if OwnershipRepository.Authorize(u, oid) = false, return ACCESS_DENIED
2. (C, r) <- LoadCiphertextAndRecord(oid, u)
3. (h, AD) <- ProtectedKeyService.OpenRecord(r)
4. K <- HMAC-SHA-256(S, ”ACE-Key” || h)
5. P <- AES-SIV.Dec(K, C, AD)
6. if authenticated decryption fails, return INTEGRITY_FAILURE
7. if SHA-256(P) != h, return CONTENT_MISMATCH
8. erase K from working memory
9. return P

2.5 Baselines and Experimental Environment

The evaluation compares three controlled implementations. The content-only MLE baseline derives K as SHA-256 of the domain label “MLE-Key” concatenated with h and therefore exposes candidate-key reconstruction for guessed content. The server-assisted control uses a 2048-bit RSA blind-OPRF sequence: the client maps h to an integer, blinds it with a fresh invertible factor, the server applies the private exponent, the client unblinds the result, and SHA-256 produces the 32-byte AES-SIV key. This control represents the public-key work of server-assisted derivation discussed in recent secure-deduplication

literature [4], [5]; it is not claimed to reproduce the network protocol, security proof, or deployment of a specific published system. All three methods use identical AES-SIV encryption, associated data, SHA-256 ciphertext tokens, and in-memory repository logic so the derivation cost is isolated (Table 2).

Table 2. Reproducible experimental configuration

Item	Configuration
Host	KVM virtual machine; AMD EPYC 9V74 processor; 9 logical CPUs exposed; 9.73 GiB RAM
CPU control	Single process pinned to one logical CPU
Operating system	Linux 6.18.44 x86_64
Software	Python 3.12.14; cryptography 50.0.2
Primitives	SHA-256; HMAC-SHA-256; AES-SIV with 32-byte key; 2048-bit RSA blind-OPRF control
Associated data	Constant byte string ACE-v1
Input source	Deterministic SHAKE-256 output from fixed seeds ACE-IJST-{size}-MiB
File sizes	1, 5, 10, and 25 MiB
Correctness cases	Original, exact duplicate, and one-byte-modified copy at each size
Repetitions	3 warm-up runs and 30 measured runs per method and size
Scheduling	Method order randomized with fixed seed; garbage collection disabled only during timed trials
Timer	time.perf_counter_ns()
Repository	In-memory token dictionary; network and persistent-storage latency excluded

The benchmark records key generation, encryption, token generation, repository lookup, end-to-end upload, authorized retrieval, and throughput. End-to-end upload begins before content hashing and ends after token lookup or insertion. Retrieval includes protected key rederivation, AES-SIV decryption, authentication, and plaintext-hash verification. Each correctness condition is executed during every measured run. Raw observations are summarized as mean $\pm$ sample standard deviation. Two-sided Welch tests compare ACE upload time with each baseline independently at each file size, using $\alpha = 0.05$. Percentage reduction for a lower-is-better metric is

$$\text{Reduction}(\%) = \frac{B - A}{B} \times 100 \tag{6}$$

where B is the baseline value and A is the ACE value. Throughput improvement uses $(A - B)/B \times 100$.

2.6 Reproducibility and Data Integrity

The supplementary package contains the complete Python benchmark, all 360 per-run timing records, the 12-row summary table, and the recorded environment. The deterministic input generator avoids undisclosed external data. The script creates a fresh 2048-bit RSA key for the control at program start, performs three untimed warm-up cycles per method and size, and checks that decrypted plaintext equals the original input. All result tables and Figure 5 are generated from the supplied CSV observations rather than transcribed from estimated values.

3 Results and Discussion

3.1 Component-Level Performance

Table 3 reports the three cryptographic components. Key generation separates the methods most clearly. Across the four size-level means, ACE required 6.730 ms, content-only MLE required 6.466 ms, and RSA-OPRF required 29.506 ms. ACE therefore incurred a 4.08% descriptive overhead relative to content-only derivation but reduced mean key-generation time by 77.19% relative to RSA-OPRF. The difference reflects the modular exponentiation and blinding work in the public-key control.

Encryption and token costs are close because all methods use the same primitives. Their growth is approximately proportional to input length, as expected for full-file AES-SIV processing and SHA-256. The observed differences should not be attributed to the derivation design; they fall within the measured run-to-run variation. Figure 5(a-c) makes this separation visible: RSA-OPRF dominates at small sizes during derivation, while the symmetric bulk operations converge.

Table 3. Component timing by file size, mean ± sample SD over 30 runs

Size (MiB)	Method	Key generation (ms)	AES-SIV encryption (ms)	Ciphertext token (ms)
1	Content-only MLE	0.635 ± 0.073	1.017 ± 0.145	0.679 ± 0.245
1	RSA-OPRF	24.198 ± 2.446	1.263 ± 0.575	0.892 ± 1.299
1	Proposed ACE	0.738 ± 0.377	1.093 ± 0.189	0.668 ± 0.149
5	Content-only MLE	3.250 ± 0.442	5.472 ± 1.014	3.187 ± 0.256
5	RSA-OPRF	27.084 ± 7.625	5.700 ± 0.873	3.347 ± 0.538
5	Proposed ACE	3.184 ± 0.203	5.913 ± 1.185	3.285 ± 0.355
10	Content-only MLE	6.341 ± 0.646	10.805 ± 1.969	6.263 ± 0.429
10	RSA-OPRF	28.555 ± 1.962	10.694 ± 1.566	6.290 ± 0.363
10	Proposed ACE	7.496 ± 5.433	11.674 ± 1.788	6.393 ± 0.590
25	Content-only MLE	15.640 ± 0.985	26.916 ± 6.517	15.795 ± 1.382
25	RSA-OPRF	38.189 ± 3.403	27.756 ± 3.604	15.954 ± 1.525
25	Proposed ACE	15.503 ± 1.013	27.794 ± 3.662	15.680 ± 1.224

3.2 End-to-End Time and Statistical Comparison

Table 4 adds repository work and reports the complete in-memory upload and authorized-retrieval paths. The arithmetic mean of the four size-level upload means was 24.859 ms for ACE, 24.004 ms for content-only MLE, and 47.485 ms for RSA-OPRF. ACE was 3.56% slower than MLE descriptively, but the two-sided Welch tests found no significant difference at 1, 5, 10, or 25 MiB ($p = 0.072$, 0.160, 0.080, and 0.714, respectively). Consequently, the evidence supports comparable performance, not superiority over content-only MLE.

Table 4. End-to-end upload and authorized-retrieval time, mean ± sample SD over 30 runs

Size (MiB)	Method	Upload total (ms)	Retrieval total (ms)
1	Content-only MLE	2.333 ± 0.273	1.794 ± 0.565
1	RSA-OPRF	26.355 ± 3.008	25.871 ± 4.584
1	Proposed ACE	2.501 ± 0.419	1.889 ± 0.592
5	Content-only MLE	11.913 ± 1.123	8.637 ± 0.801
5	RSA-OPRF	36.136 ± 7.618	31.478 ± 2.539
5	Proposed ACE	12.386 ± 1.429	8.920 ± 0.927
10	Content-only MLE	23.413 ± 2.804	17.702 ± 2.268
10	RSA-OPRF	45.546 ± 2.835	40.009 ± 3.553
10	Proposed ACE	25.569 ± 5.961	17.217 ± 2.171
25	Content-only MLE	58.356 ± 8.005	44.195 ± 4.617
25	RSA-OPRF	81.903 ± 5.830	65.545 ± 7.814
25	Proposed ACE	58.981 ± 4.700	42.211 ± 4.169

Relative to RSA-OPRF, ACE reduced the average upload time by 47.65%; the difference was significant at every size ($p < 0.001$). The benefit decreases as file size grows because bulk hashing and AES-SIV consume a larger share of the path, but the public-key derivation remains visible at 25 MiB. Retrieval shows the same pattern: the four-size mean was 17.559 ms for ACE, 18.082 ms for MLE, and 40.726 ms for RSA-OPRF. Thus ACE reduced retrieval time by 56.89% relative to the RSA control while remaining close to content-only MLE.

3.3 Throughput

Table 5 reports upload throughput. The four-size mean was 411.527 MiB/s for ACE, 430.581 MiB/s for MLE, and 176.826 MiB/s for RSA-OPRF. ACE was 4.43% below content-only MLE and 132.73% above RSA-OPRF on this host. Throughput rose with size for RSA-OPRF because its largely fixed public-key cost was amortized over more data. ACE and MLE were dominated by linear hashing and encryption and consequently remained within a narrower range.

Table 5. End-to-end upload throughput, mean ± sample SD over 30 runs

Size (MiB)	Content-only MLE (MiB/s)	RSA-OPRF (MiB/s)	Proposed ACE (MiB/s)
1	433.540 ± 44.027	38.364 ± 3.839	408.553 ± 55.756
5	422.980 ± 36.021	141.944 ± 18.058	408.619 ± 44.516
10	431.638 ± 39.999	220.334 ± 12.938	402.556 ± 53.903
25	434.165 ± 44.637	306.663 ± 20.815	426.381 ± 32.827

Figure 5 summarizes component time, end-to-end time, retrieval, and throughput with sample-standard-deviation error bars. The figure confirms that the proposed HMAC derivation does not introduce a public-key-scale penalty, but it also shows why small descriptive differences between ACE and MLE should not be generalized beyond this host.

3.4 Duplicate Accuracy and Storage Saving

Every ACE run reproduced the same token for the exact duplicate and a different token for the one-byte-modified copy. Across 30 runs and four sizes, this produced 120/120 exact matches and 120/120 modified-file separations. These tests establish deterministic exact-match correctness for the controlled inputs; they do not measure fuzzy or semantic duplicate detection, which is addressed by different classes of schemes[6],[13],[16].

The corpus contained three instances at each size: an original, an exact duplicate, and a modified copy. Without deduplication, twelve ciphertext objects represented 123 MiB of plaintext. ACE stored the original and modified objects at each size, giving eight ciphertext objects and 82 MiB of plaintext-equivalent payload. Including the 16-byte AES-SIV tag per stored object, the ciphertext saving was 42,991,680 bytes, or 33.33%, before token and ownership metadata. The same ciphertext saving is expected from any exact deterministic baseline on this corpus; ACE's contribution is the protected-secret and authorization path, not a higher duplicate ratio (Table 6).

Table 6. Functional correctness and ciphertext-storage outcome for the proposed method

Measure	Result	Interpretation
Exact-duplicate token matches	120/120 (100%)	Equal input, secret, and associated data produced equal ciphertext tokens
Modified-file separations	120/120 (100%)	One-byte changes produced distinct tokens in all measured conditions
Input instances	12	Original, duplicate, and modified copy for each of four sizes
Unique ciphertext objects stored	8	One duplicate object per size was replaced by an ownership reference
Plaintext-equivalent input volume	123 MiB	Undeduplicated workload volume
Plaintext-equivalent stored payload	82 MiB	Two unique objects per file size
Ciphertext-storage saving	33.33%	Excludes deployment-specific token and ownership metadata

Repository lookup was not a bottleneck: the ACE mean of the four size-level lookup means was 0.0036 ms. This value describes an in-memory dictionary and must not be interpreted as cloud-database latency.

3.5 Comparative Security and Functional Analysis

Table 7 separates the tested properties. Content-only MLE has the lowest conceptual complexity but allows a candidate plaintext to determine the candidate key. RSA-OPRF hides the unblinded derivation from the server-side evaluation step and offers a natural rate-limiting point, but public-key operations increased latency in the controlled implementation. ACE prevents content-only offline key reconstruction while S remains secret and retains a symmetric operation path. It does not provide OPRF obliviousness: the protected service handles h when opening an authorized record. Recent password-protected and subversion-resistant designs offer stronger compromise models than ACE[4],[12], while recent ownership protocols provide more complete proof-of-ownership mechanisms than the validation hook evaluated here[9],[10].

3.6 Security Analysis

Deterministic correctness. For fixed S and AD, equality of P_1 and P_2 implies equality of their SHA-256 digests, HMAC outputs, AES-SIV ciphertexts, and ciphertext tokens. Conversely, a changed plaintext is expected to produce a distinct token except with the combined negligible collision or forgery probability of the cryptographic primitives. The 120 modified-file conditions empirically checked this property but do not replace the underlying cryptographic assumptions.

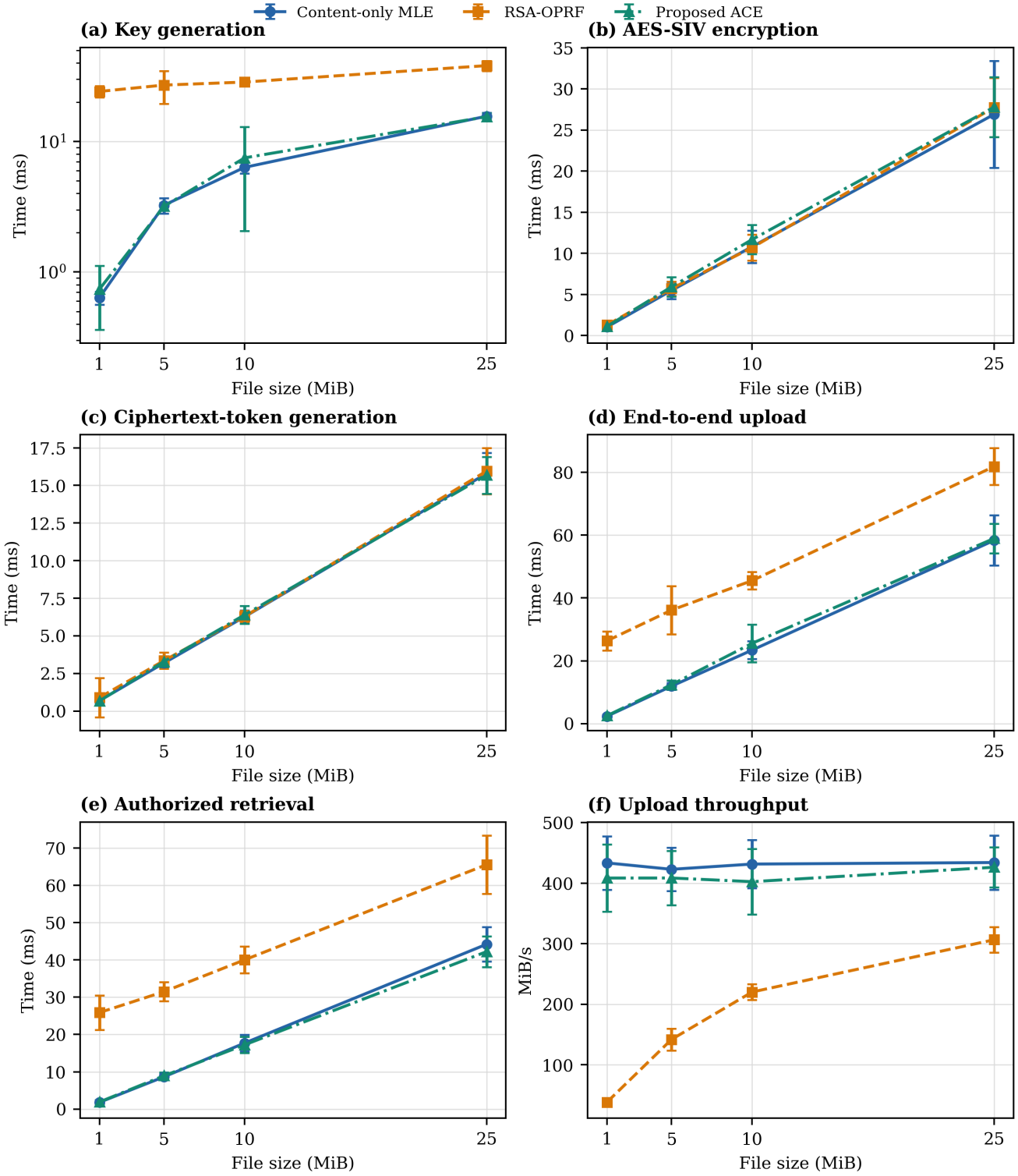


Fig 5. Performance comparison across file sizes; error bars indicate sample standard deviation over 30 runs.

Table 7. Security and functional comparison of the controlled methods

Property	Content-only MLE	RSA-OPRF control	Proposed ACE
Deterministic ciphertext for equal plaintext	Yes	Yes	Yes
Independent protected secret contributes to key	No	RSA service key	HMAC service secret
Candidate key computable from guessed file alone	Yes	No, without service interaction	No, while S remains secret
Oblivious server-side derivation	No	Yes in blind-OPRF step	No
Public-key operation in derivation	No	Yes	No
Deterministic authenticated encryption in experiment	AES-SIV	AES-SIV	AES-SIV
Ciphertext-token duplicate verification	Included in control	Included in control	Integrated
Authorization before pointer creation	Not inherent	Not inherent	Required by Algorithm 2
Protected ownership record in retrieval	Not inherent	Not inherent	Required by Algorithm 3
Equality leakage after accepted upload	Yes	Yes	Yes
Key-service rate limiting	Not applicable	Supported operationally	Required to limit online guesses

Offline dictionary and confirmation attacks. In content-only convergent encryption, an adversary can derive a candidate key from a guessed file and test the resulting ciphertext or token. Under the pseudorandom-function assumption for HMAC-SHA-256, an observer without S cannot compute the ACE key from h alone. Ciphertext and token exposure therefore do not enable the same content-only offline test. This protection fails if S is compromised, and an authenticated adversary with unrestricted derivation queries can mount an online guess attack. Rate limiting, audit logging, anomaly detection, and preferably hardware-backed secret storage are therefore deployment requirements. Stronger password-protected and subversion-resistant models remain appropriate where key-service compromise is in scope[4],[12].

Unauthorized duplicate claims. A token match is treated only as evidence that an object exists; it is not evidence that the requester owns the underlying plaintext. Algorithm 2 calls ValidateDuplicateClaim before adding an ownership pointer. A production deployment should instantiate this hook with an appropriate proof-of-ownership protocol, such as recent full-file or blind bidirectional mechanisms[9],[10]. Without such an instantiation, ACE prevents direct pointer creation by an unauthenticated request but does not claim the complete guarantees of those protocols.

Ciphertext manipulation and integrity. AES-SIV authenticated decryption rejects modified ciphertext or mismatched associated data. Algorithm 3 also recomputes SHA-256 of the recovered plaintext and compares it with the protected record. The ciphertext token can reveal repository inconsistency before retrieval, while the authenticated-encryption tag enforces integrity during decryption.

Unauthorized retrieval. The object identifier alone is insufficient. The ownership repository first checks the user-object relation; only then can the protected record be opened and the key rederived. Separation of token, ciphertext, and ownership repositories limits the information released by any single ordinary interface, although it is not a substitute for host isolation or least-privilege administration.

Metadata and equality leakage. Exact deduplication reveals that accepted uploads share a token and typically reveals object length and access frequency. ACE does not remove this leakage. Restricting token queries to authenticated upload sessions, encrypting sensitive metadata, limiting logs, and separating service roles reduce exposure. Schemes that hide more equality information require a different leakage-performance trade-off[7],[13].

Secret compromise and rotation. Knowledge of S together with a candidate file permits key reconstruction, and a single long-lived secret creates a concentration of risk. Practical deployment should place S in a hardware security module or trusted execution boundary, derive epoch-specific subkeys, maintain versioned AD, and support controlled re-encryption. Secret rotation was not implemented in this experiment and remains a limitation.

3.7 Practical Significance and Limitations

The main empirical result is not that ACE is faster than all alternatives. It is that protected-secret HMAC derivation remained statistically comparable to the content-only MLE control while avoiding the public-key cost measured in the RSA-OPRF control.

This is relevant to backup, object-storage, and cross-user exact-deduplication paths in which the protected service is trusted to see content fingerprints. The method also turns a token match into an authorization decision rather than an automatic ownership grant, aligning it with current work on secure ownership verification [9], [10].

The evaluation remains bounded. It uses deterministic synthetic files, four sizes, one virtualized host, one logical CPU, and an in-memory repository. It excludes network delay, persistent cloud storage, concurrent clients, token-database growth, key rotation, hardware-protected secrets, and service failure. The RSA-OPRF control measures a defined cryptographic sequence rather than a full deployed server-assisted protocol. The Welch tests evaluate timing on this host and are not portable hardware claims. The security analysis is threat-oriented with proof sketches, not a reduction-based proof. Finally, ACE performs exact deduplication only; near-duplicate and semantic approaches solve a different problem [6], [13], [16]. These boundaries prevent generalization beyond the demonstrated conditions and define the next validation steps.

4 Conclusion

This study presented a protected-secret convergent-encryption method for exact cloud deduplication. ACE derives a deterministic 256-bit key from the plaintext hash and a service-held HMAC secret, encrypts with AES-SIV, hashes the authenticated ciphertext for duplicate verification, and creates an ownership pointer only after claim validation. The design prevents an external observer from reconstructing the key from a guessed file alone while the service secret remains confidential, but it does not provide oblivious derivation and cannot eliminate equality leakage.

The revised 30-run experiment produced 360 timed observations. ACE's four-size mean upload time was 24.859 ms, statistically indistinguishable from content-only MLE at each size, and 47.65% below the 47.485 ms RSA-OPRF control. ACE key generation was 77.19% lower than RSA-OPRF, and authorized retrieval was 56.89% lower. All 120 exact duplicates were detected, all 120 one-byte modifications were separated, and the controlled corpus achieved 33.33% ciphertext-storage saving before metadata. These results support ACE as a computationally lightweight protected-secret alternative under its stated trust model, not as a replacement for oblivious or formally subversion-resistant protocols.

Future work will evaluate networked and persistent repositories, heterogeneous real-world datasets, concurrent clients, larger token indexes, hardware-backed secret storage, rate-limited derivation, epoch-based key rotation, and a complete proof-of-ownership instantiation. A formal leakage model and reduction-based analysis are also required before stronger security claims can be made.

5 Acknowledgements

Nil

Author details: R. Surendhar Kumar is a Research Scholar; Dr. S. S. Manikandasaran serves as an Assistant Professor cum Associate Director.

References

1) Yuan H, Chen X, Li J, Jiang T, Wang J, Deng RH. Secure cloud data deduplication with efficient re-encryption. *IEEE Transactions on Services Computing*. 2022;15(1):442–456. https://doi.org/10.1109/TSC.2019.2948007.
2) Yu X, Bai H, Yan Z, Zhang R. VeriDedup: A verifiable cloud data deduplication scheme with integrity and duplication proof. *IEEE Transactions on Dependable and Secure Computing*. 2023;20(1):680–694. https://doi.org/10.1109/TDSC.2022.3141521.
3) Ahmad S, Arif M, Ahmad J, Nazim M, Mehfuz S. Convergent encryption enabled secure data deduplication algorithm for cloud environment. *Concurrency and Computation: Practice and Experience*. 2024;36(21):e8205. https://doi.org/10.1002/cpe.8205.
4) Zhang Y, Xu C, Cheng N, Shen XS. Secure password-protected encryption key for deduplicated cloud storage systems. *IEEE Transactions on Dependable and Secure Computing*. 2022;19(4):2789–2806. https://doi.org/10.1109/TDSC.2021.3074146.
5) Tang X, Guo C, Choo KKR, Jiang X, Liu Y. A secure and lightweight cloud data deduplication scheme with efficient access control and key management. *Computer Communications*. 2024;222:209–219. https://doi.org/10.1016/j.comcom.2024.05.003.
6) Jiang T, Yuan X, Chen Y, Cheng K, Wang L, Chen X, et al. FuzzyDedup: Secure fuzzy deduplication for cloud storage. *IEEE Transactions on Dependable and Secure Computing*. 2023;20(3):2466–2483. https://doi.org/10.1109/TDSC.2022.3185313.
7) Song M, Hua Z, Zheng Y, Huang H, Jia X. LSDedup: Layered secure deduplication for cloud storage. *IEEE Transactions on Computers*. 2024;73(2):422–435. https://doi.org/10.1109/TC.2023.3331953.
8) Qi S, Wei W, Wang J, Sun S, Rutkowski L, Huang T, et al. Secure data deduplication with dynamic access control for mobile cloud storage. *IEEE Transactions on Mobile Computing*. 2024;23(4):2566–2582. https://doi.org/10.1109/TMC.2023.3263901.
9) Dave J, Ram KR, Patil P, Patil H, Borole S, Panda S. Secure and efficient ownership verification for deduplicated cloud computing systems. *Journal of Cloud Computing*. 2025;14:20. https://doi.org/10.1186/s13677-025-00743-y.
10) Dave J, Ram KR, Patil P, Patil H, Borole S, Chinni VK, et al. Blind and bidirectional ownership verification for deduplicated cloud computing systems. *Future Generation Computer Systems*. 2026;175:108082. https://doi.org/10.1016/j.future.2025.108082.

11) Leelavathy L, Sugumar R. DNA Cloud Data Security for Numerical Data using Cryptography Methods. *Solid State Technology*. 2020;63(3):4160–4165. Available from: https://solidstatetechnology.us/index.php/JSST/article/view/6154.
12) Li S, Ma M, Han Y, Xu C. Towards subversion-resistant password-protected encryption for deduplicated cloud storage. *Journal of Information Security and Applications*. 2025;94:104233. https://doi.org/10.1016/j.jisa.2025.104233.
13) Wang C, Zhang G, Qi H, Chen B. A CAE model-based secure deduplication method. *Scientific Reports*. 2025;15:24605. https://doi.org/10.1038/s41598-025-09788-0.
14) Yao X, Li C, Guo J, Huang K, Yao T, Zhao M. TrustDedup: Secure data deduplication for IoT based on end-edge-cloud collaboration. *Journal of Systems Architecture*. 2025;168:103541. https://doi.org/10.1016/j.sysarc.2025.103541.
15) Uma G. A novel trusted key support model for synchronized data protection in duplicate-aware cloud storage. *Indian Journal of Science and Technology*. 2026;19(19):1260–1266. https://doi.org/10.17485/IJST/v19i19.742.
16) Tamilselvi JJ, Rani VU, John R, Thomas SS. Improved cloud storage performance using the data deduplication-based efficient cloud optimization (D2-ECO) model. *Indian Journal of Science and Technology*. 2026;19(18):1157–1168. https://doi.org/10.17485/IJST/v19i18.246.
17) Mageshkumar NA, Swapna J, Pandiaraj A, Rajakumar R, Krichen M, Ravi V. Hybrid cloud storage system with enhanced multilayer cryptosystem for secure deduplication in the cloud. *International Journal of Intelligent Networks*. 2023;4:301–309. https://doi.org/10.1016/j.ijin.2023.11.001.
18) Shynu PG, Nadesh RK, Menon VG, Venu P, Abbasi M, Khosravi MR. A secure data deduplication system for integrated cloud-edge networks. *Journal of Cloud Computing*. 2020;9:61. https://doi.org/10.1186/s13677-020-00214-6.
19) Chouhan V, Peddoju SK, Buyya R. dualDup: A secure and reliable cloud storage framework to deduplicate the encrypted data and key. *Journal of Information Security and Applications*. 2022;69:103265. https://doi.org/10.1016/j.jisa.2022.103265.
20) Vignesh R, Preethi J. Secure data deduplication system with efficient and reliable multi-key management in cloud storage. *Journal of Internet Technology*. 2022;23(4):811–825. https://doi.org/10.53106/160792642022072304016.