IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Journal Européen des Systèmes Automatisés
Vol. 59, No. 9, September, 2026, pp. 2721-2733
Journal homepage: http://iieta.org/journals/jesa

Structurally Bound File Encryption for Secure Surveillance Video Storage Application

Azzam Adnan Mohammed, Mohammed Younis Thanoun, Yazen S. Sheet*, Firas S. Alsharbaty

Department of Communications and Intelligent Digital Systems Engineering, College of Engineering, University of Mosul, Mosul 41002, Iraq

Corresponding Author Email: yazenalnuaimi@uomosul.edu.iq

Copyright: ©2026 The authors. This article is published by IIETA and is licensed under the CC BY 4.0 license (http://creativecommons.org/licenses/by/4.0/).

https://doi.org/10.18280/jesa.590923

Received: 4 July 2026
Revised: 5 September 2026
Accepted: 14 September 2026
Available online: 30 September 2026

Keywords:
authentication, security, authenticated encryption with associated data, X25519, surveillance video

ABSTRACTs

This research paper contributes to the study of structurally bound file encryption by introducing a fragmentation–permutation container architecture with authenticated metadata coupling, with particular relevance to secure surveillance video storage. The proposed design integrates randomized fragmentation, permutation-based block dispersion, and authenticated encryption with associated data (AEAD) to extend conventional message level security toward container level structural authentication in adversarial storage environments. Unlike monolithic file encryption schemes that preserve sequential organization, the construction decouples index protection from payload encryption and cryptographically binds container identifiers, permutation descriptors, and index integrity hashes within the Associated Data (AD) of each encrypted fragment. A formal analysis demonstrates that structural binding reduces to the Integrity of Ciphertext (INT-CTXT) authenticity guarantees of the underlying AEAD scheme, thereby preventing block reordering, cross-container substitution, and metadata manipulation without introducing new cryptographic assumptions. The system is implemented using X25519 key exchange, HKDF-based key derivation, and ChaCha20-Poly1305 AEAD. Experimental evaluation on datasets up to 1.9 GB demonstrates stable encryption throughput in the range of 540–620 MB/s, with structural metadata overhead remaining below 0.03%. Integrity stress testing involving 100 adversarial modifications achieved complete detection with zero unexpected acceptances, confirming practical deployment within established AEAD security models.

1. INTRODUCTION

Data protection at rest has become a critical requirement in modern storage systems, particularly for applications that continuously generate large volumes of sensitive sequential data, such as surveillance video. Authenticated encryption schemes like AES-GCM and ChaCha20-Poly1305 are widely used because they provide strong confidentiality and integrity guarantees under well-established security notions, including indistinguishability under chosen-ciphertext attack (IND-CCA) and ciphertext integrity (INT-CTXT). These schemes have been formally analyzed within the authenticated encryption with associated data (AEAD) framework and are valued for their provable security and practical performance. As a result, AEAD-based encryption has become the de facto standard for protecting files and data at rest. However, standard AEAD schemes treat the underlying file merely as a monolithic stream of opaque bytes without modifying or protecting its internal structural context. Structural information—such as block sequencing, index mapping, and container segmentation—is typically retained outside the cryptographic boundary. While this design preserves ciphertext indistinguishability, it creates subtle attack surfaces in adversarial storage environments where an adversary possesses full read and write access. This vulnerability is particularly acute in surveillance video storage, where the temporal continuity and sequential integrity of footage are paramount, leaving the data susceptible to selective block reordering, fine-grained truncation, and cross-container block substitution [1]. This structural weakness is not adequately handled by current paradigms. Without provable privacy or cryptographic integrity, common file obfuscation strategies just conceal a file's header or presentation. To prevent local compromise, fragmentation-based encryption approaches disperse data pieces across numerous clouds or distributed nodes. They see fragmentation as a storage-level distribution mechanism, not as a cryptographically bound integrity tool, though. Conversely, ordinary AEAD primitives authenticate payload bytes together with additional associated data (AAD), but they do not have an internal process that links structural layout parameters, like permutation matrices and index hierarchies, to the decoding context directly. Therefore, there is a systemic distinction between payload byte-level security and container-level structural binding. This structural gap gives rise to a fundamental question: Can file-level encryption be extended beyond standard payload confidentiality to achieve cryptographic structural binding, particularly for large-scale sequential data such as surveillance video, without introducing new cryptographic assumptions? To address this challenge, we introduce a structurally bound file encryption

framework utilizing a fragmentation–permutation container architecture with authenticated metadata coupling. Unlike conventional schemes, our design elevates structural organization to a primary security element. Index management is decoupled from raw payload encryption, while container identifiers, permutation descriptors, and index integrity hashes are cryptographically bound within the associated data (AD) parameter of each encrypted fragment. This construction extends authentication guarantees beyond traditional payload-level AEAD bounds to the container layout itself. Formal and empirical evaluations on multi-gigabyte surveillance video datasets demonstrate that this structural binding effectively eliminates structural manipulation attacks while maintaining high throughput and minimal computational overhead. The main contributions of this study are summarized as follows:

(1) Architectural Contribution: A fragmentation–permutation container architecture is proposed to enforce cryptographic structural binding between metadata and payload blocks.

(2) Benchmark Contribution: A systematic comparison and classification (Table 1) is presented, clearly distinguishing between the proposed structural anchor framework and existing file obfuscation, fragmentation-based encryption, and standard AEAD schemes.

(3) Security Contribution: A formalization is provided showing how authenticated metadata coupling prevents block reordering, cross-container substitution, and index manipulation under standard AEAD assumptions.

(4) Implementation Contribution: A complete implementation based on X25519, HKDF, and ChaCha20-Poly1305 is presented, demonstrating practical deployment.

(5) Empirical & Conceptual Contribution: Extensive benchmarking across multi-gigabyte surveillance video datasets is conducted, achieving 540–620 MB/s encryption throughput and validating 100% tamper detection, thereby demonstrating that structural confidentiality can be systematically integrated into file encryption containers without abandoning established AEAD security models.

2. RELATED WORKS

2.1 Authenticated encryption and AEAD guarantees

AEAD has become a principal cryptographic paradigm for protecting data in storage and communication systems. AEAD combines confidentiality and integrity in a single construction while allowing selected non-confidential information to be authenticated as AD [1]. This property is important for file-storage applications because parameters such as version numbers, record types, timestamps, or container identifiers may need to remain visible while still being protected against unauthorized modification.

AES-GCM and ChaCha20-Poly1305 are two of the most popular AEAD designs. ChaCha20-Poly1305 is especially alluring in embedded and software situations because it delivers great throughput without requiring specialized AES commands [2, 3]. From the perspectives of implementation and performance, its application in limited IoT applications and systems has been explored [3]. In a multi-user environment, the security of ChaCha20-Poly1305 was also examined. It relates to storage systems that, under ideal operating conditions, may shield many files, containers, or users [4]. Strong validated encryption and significant hardware acceleration advantages are available with AES-GCM. Both confidentiality and dependability can be severely jeopardized by a failure to properly handle single-use data. Misuse-resistant designs like GCM-SIV were consequently put forth to lessen the effects of nonce reuse [5]. Other research has examined committing authenticated encryption, in which ciphertexts are cryptographically linked to a particular underlying key or message environment [6]. Further demonstrating the significance of connecting cryptographic output to context information rather than viewing the ciphertext as an isolated sequence of bytes, more current studies on AEAD with context capture and leak resistance have been published [7].

Despite these advances, conventional AEAD normally authenticates a plaintext message and the explicitly supplied AD. It does not, by itself, define the semantics of a file container, the relationship between fragments, the correctness of an index, or the permitted order of blocks. Consequently, a system may provide strong message-level authenticity while leaving the organization of multiple encrypted fragments to an external storage layer. This distinction is central to the present study: the proposed framework uses AD to bind container identifiers, permutation descriptors, and index-integrity values to each encrypted fragment, thereby extending authentication from individual payloads to the logical structure of the file container.

2.2 Structural, metadata, and access-pattern leakage

Encryption does not necessarily hide all information revealed by a storage system. Encrypted databases and searchable cryptographic systems can expose metadata such as access patterns, hit sizes, frequency distributions, timing, and relationships between repeated queries. Quantitative analysis of leaks in searchable encryption shows that privacy should be assessed not only at the level of ciphertext content, but also at the level of information revealed by system behavior and metadata [8]. Similarly, it has been shown that a distributed secure data storage design requires joint consideration of query processing, encrypted indexes, and secure data organization [9].

Sequential surveillance video is subject to the same worry. The interpretation of a surveillance file goes beyond just a collection of separate bytes; it also takes into account continuity, temporal order, frame grouping, indexing, and the connection between segments and recording sessions. The adversary may try to reorder encrypted blocks, eliminate selected segments, swap a fragment from another container, or modify an index while keeping the payload ciphertexts individually unaltered if they are able to read and change storage objects. Standard encryption may hide the visual content, but it does not automatically prove that the recovered parts come from the same container or are in their original structural positions.

File obfuscation and layout randomization can make direct inspection more difficult, but obfuscation alone does not provide a formal authenticity guarantee. The limitation is not simply that metadata may remain visible; rather, the storage system may lack a cryptographic statement describing which metadata values are valid for which ciphertext fragments. The structurally bound approach addresses this limitation by placing structural descriptors within the authenticated context of AEAD. Under this design, a modification to a permutation descriptor, container identifier, or index-integrity value causes

authentication failure during decryption, provided that nonce use and key management follow the requirements of the underlying AEAD scheme [1, 4].

2.3 Remote data auditing, provable data ownership, and proof of retrievability

Provable data ownership (PDP) and proof of retrievability (PoR) are designed to allow clients to verify that an untrusted server still owns stored data without having to download the entire file. Early PDP work created an efficient challenge-and-response technique to verify ownership of external data [10]. On the other hand, later projects improved scalability and reduced the cost of validating large data collections [11]. This area of study was expanded upon by remote data-auditing frameworks, which dealt with real-world auditing needs and a variety of trust models [12].

Further advancements in which files may be updated, inserted, or removed while preserving verifiable integrity have been inspired by dynamic storage. Methods such as oblivious RAM have been included in dynamic PoR constructions to enable updates without exposing a lot of information regarding the modified sites [13]. In other work, dynamic retrievability has been examined using a variety of storage and verification architectures [14]. Since these protocols handle server-side retention, availability, and the ability to find corruption or loss, they are useful for repositories used for long-term surveillance.

Remote auditing and structurally restricted encryption, on the other hand, solve problems that are similar yet distinct. PDP and PoR are mostly used to determine whether a remote service's data matches an audit protocol. The whole internal topology of an encrypted file is not always authenticated with every decryption procedure. An audit can prove that encoded symbols or blocks are still accessible, but a different application-level index is susceptible to cross-container substitution or reordering. Thus, cryptographic binding between the payload fragments and the container metadata that specifies their interpretation cannot be replaced by auditing, which can be utilized to augment the proposed framework.

2.4 Fragmentation, permutation, dispersal, and multi-cloud protection

To mitigate the impact of compromise in distributed storage, fragmentation and dispersion have been used for a long time. Encryption and fragmentation together may stop a single storage location from disclosing the whole data object and scatter sensitive characteristics over several sites [15]. Studies of distributed-storage security identify fragmentation, encryption, replication, and dispersion as complementary methods for enhancing confidentiality and resilience [16]. Moreover, structural fragmentation examines the possibility of dispersing encrypted data across storage systems [17].

Many studies use fragmentation in cloud and multi-cloud settings. Random pattern fragmentation is combined with a distributed database to reduce the risk of exposing all data objects in the same location [18]. To improve privacy and availability, other approaches combine erasure coding, dispersal, and encryption across multiple infrastructures [19]. Large-scale or multi-cloud hybrid encryption systems for data storage have also been developed [20]. Relocation-based fragmentation has been studied to protect digital images in cloud environments, demonstrating real-world applications of relocation and fragmentation to confound direct structural interpretation [21].

These techniques provide crucial protection at the distribution and isolation levels, although their primary purpose is generally to isolate data across nodes, clouds, or databases. Rather than being a verified cryptographic relationship between the fragment, its position, and the file-level metadata, fragmentation is frequently viewed as a storage-management decision. As a result, a system may encrypt and distribute pieces successfully while relying on an external index or reconstruction technique whose integrity is not totally connected to the ciphertext.

The proposed framework differs in its security objective. Randomized fragmentation and permutation are not used only to distribute or obscure the data; they are incorporated into a cryptographically authenticated container architecture. Each fragment is associated with structural information through AEAD AD. This design aims to detect block reordering, selective deletion, index manipulation, and cross-container substitution as authentication failures rather than treating them as ordinary reconstruction errors.

2.5 Secure surveillance-video storage and privacy

Surveillance-video systems require protection mechanisms that preserve confidentiality while supporting high-volume storage, retrieval, and processing. Privacy-preserving cloud-based video-surveillance research has examined adjustable privacy protection so that sensitive visual information can be protected before or during cloud processing [22]. Multi-level encryption systems for surveillance video have proposed layered security mechanisms that combine different protection stages for video content and associated processing operations [23]. Other video-surveillance approaches have investigated wavelet-based encryption and related multimedia-security techniques [24]. Privacy-protection systems for video transmission in edge–fog–cloud environments further illustrate the need to secure video as it moves across heterogeneous storage and computing layers [25].

These findings show that surveillance video cannot be considered as a simple little data unit. It is massive, sequential, regularly generated, and frequently kept over extended durations. The security system must thus strike a compromise between secrecy, integrity, throughput, storage overhead, and the capacity to detect tampering of certain temporal regions. Existing surveillance-security approaches are largely concerned with visual-content secrecy, selective privacy protection, transmission security, and layered encryption. They do not typically specify the cryptographic coupling of a full fragment topology—including container identity, permutation information, and index-integrity data—to each encrypted storage unit.

2.6 Synthesis of the research gap

In addition to that, there are four distinct but insufficient layers of security in the literature. First, AEAD ensures the strict confidentiality and authentication of encrypted payloads and AD provided explicitly [1-7]. Second, from the work on leaks and searchable storage, we know that structural and access-related information may be disclosed even if the payload is encrypted [8, 9]. Third, PDP and PoR provide remote proofing of ownership, integrity and retrievability, mainly in the case of outsourced storage [10-14]. Finally, fragmentation, shuffling, and multi-clouds lower the risks of

having all the data stored in a single place [15-21]. The literature on video surveillance provides further requirements specific for the particular application area [22-25].

However, these lines of work create an explicit gap with respect to the integrity of the payload versus the structure of the container. First, existing AEAD constructions provide no inherent specification of a valid topology for a fragmented file, while remote auditing cannot guarantee that the topology is correct at decryption time. Furthermore, fragmentation-based systems tend to make heavy use of dispersion techniques but are often not bound by integrity constraints on the complete mapping. This work attempts to fill this gap by introducing a fragmentation-permutation-based container design which incorporates authenticated metadata bindings. By tying container identifiers, permutation information, and index-integrity hashes to the authenticated data of each fragment, the system considers structure to be first-class. The security claim of this work is thus not the creation of any new cryptographic primitives, but that structural tampering can be reduced to forging the authenticity of the underlying AEAD construction.

In order to summarize the two paradigms in a systematic manner, and to show how our proposed framework bridges the research gap found, Table 1 provides a comparison between existing security paradigms and our container-based framework based on structural properties.

Table 1. Systematic comparison between existing security paradigms and the proposed framework

Paradigm/Approach	Protection Target	Authentication Scope	Computational Overhead
File Obfuscation [8, 21]	File header/visual layout appearance	None (No formal cryptographic integrity guarantees)	Very Low O (1)
Fragmentation-based Encryption [15, 17, 18]	Data chunks across distributed nodes	Chunk-level or block-level isolation	Moderate to High (Storage/IO)
Standard AEAD (e.g., AES-GCM) [1, 3, 5, 7]	Arbitrary payload byte streams	Payload bytes & static (AAD)	Low (Hardware accelerated)
Proposed Framework	Structurally bound metadata & layout	Full payload + Permutation descriptors & Index topology	Low to Moderate (Overhead < 0.03%)

3. THE PROPOSED THREAT MODEL AND APPLICATION SCOPE

3.1 Application scope and system boundaries

The new design approach is clearly aimed at securing the data at rest level in multi-tier video surveillance systems. In order to set boundaries between system components, we will delineate our scope in terms of three architectural levels:

(1) Endpoint Layer (Edge Cameras & Ingestion Nodes): End-point capture devices are trusted environments. They carry out encoding of videos, creation of metadata, and structure-based encryption prior to data distribution. Compromise of the endpoint camera, infection of root, and key recovery from the endpoint camera itself is outside the scope of consideration.

(2) Communication Layer (Data-in-Transit): It is considered that network transmission from the edge nodes to the local servers and cloud storage is adequately secured via transport layer security protocols (for example, TLS 1.3 or IPsec). Protection against interception and packet injections when data is transmitted is not the main focus of this work.

(3) Storage layer (storage servers and cloud repositories): It is the first line of defense for our architecture. We assume an adversarial setting where some storage node(s), cloud vaults, or local video storage systems are untrusted or compromised.

3.2 Threat assumptions and attacker capabilities

Assume an adversary operating within the compromised storage system with full read and write access to the encrypted container. Such an attacker may intercept, modify, reorder, truncate, or insert ciphertext blocks, as well as manipulate headers and encrypted index metadata. The model explicitly considers cross-container block insertion, replay attacks, and structural manipulation [13, 14, 16].

All underlying primitives are subject to standard cryptographic assumptions. In particular, key agreement based on X25519, key derivation via HKDF, and authenticated encryption using ChaCha20-Poly1305 are assumed to be secure according to the definitions of IND-CCA and INT-CTXT [10, 11, 24, 25]. This scope excludes endpoint compromise, key extraction, and side-channel leakage.

The attacker cannot make use of any secret keys, but he/she can exploit the container's structure and layout. It is achieved through movement of blocks, merging data from other containers, modifying metadata, or truncating data blocks in order to produce an erroneous decryption or avoid integrity checking [12, 18], as shown in Figure 1 below.

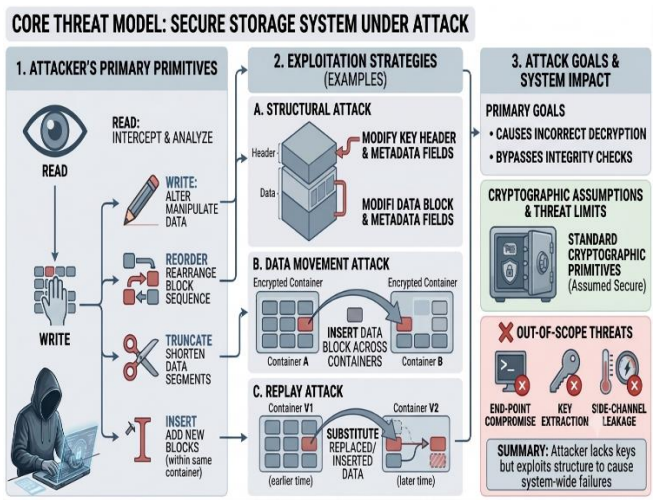

Figure 1. Threat model and operational security boundaries of the proposed system

3.3 Target security properties

Security properties guaranteed by this system include:

(1) Confidential authentication: Ciphertexts are not distinguishable under any adaptive chosen-ciphertext attack,

while all modifications will be detected with overwhelming probability.

(2) Integrity at the container level: Header parameters, index structure, and block order are verified upon each access, and any modification results in failed authentication.

(3) Resistance to reordering and substitution: All ciphertexts that emanate from other containers or are out of sequence are disregarded.

(4) Structure confusion: The structure and division of the content cannot be deduced by the ciphertext arrangement, with the exception of the file size. Therefore, structurally bound encryption strongly couples the container structure with the ciphertext authentication of AEAD [11, 12].

The system aims at ensuring the following security guarantees:

(1) Authenticated confidentiality: The ciphertext cannot be distinguished by means of any adaptive chosen ciphertext attack and modification of the ciphertext can be reliably detected.

(2) Container-level integrity: The integrity of all parameters of the header, index, and block order is verified at every access attempt, resulting in authentication failure in case of any tampering.

(3) Resistance to reordering and substitution attacks: Ciphertext blocks coming from other containers or out of order are considered invalid.

(4) Structural confidentiality: Neither content order nor segmentation of content can be inferred from the ciphertext except for the total size of the file.

Hence, the encryption that is structure bound is strongly correlated with ciphertext authentication in the context of AEAD [11, 12].

4. SYSTEM ARCHITECTURE AND IMPLEMENTATION DETAILS

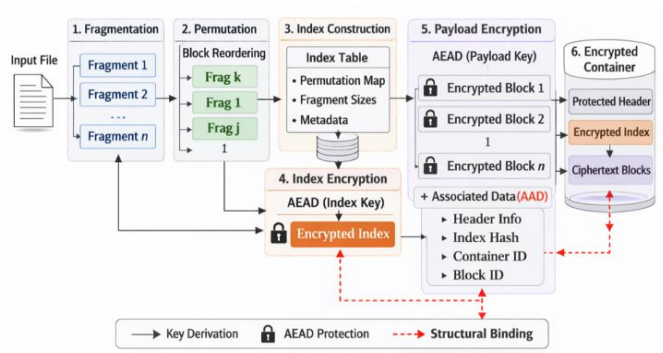

Figure 2. Architectural overview of the proposed scheme

Figure 2 depicts the suggested container design using fragmentation and permutation. Instead of a passive byproduct of file processing, structural organization is treated as an authenticated security element in the system's container-based encryption framework.

(1) permutation-based block reordering.
(2) randomized fragmentation.
(3) index construction.
(4) index encryption.
(5) authenticated payload encryption.
(6) final container assembly.

These phases collaboratively apply structural binding by cryptographically linking container metadata and payload blocks within the authentication realm.

4.1 Parameter specifications and index topology

For precise algorithmic reproducibility and to solve the problem of parameter formalization, the following definitions are provided for the operational variables, randomizers, and layouts:

Fragment Size Generation (S_{frag}): The input bitstream P of length L is partitioned into n fragments $F = \{F_1, F_2, \dots, F_n\}$. In standard storage deployments, a baseline block size S_{frag} (e.g., 64 KB aligned to storage sector boundaries) is utilized. For dynamic surveillance video streams, fragment boundaries are aligned with video Group of Pictures (GOP)/Frame boundaries, resulting in variable sizes $S_{frag}(i)$. PKCS#7 zero-padding is appended to the terminal fragment F_n to ensure exact block alignment.

(1) Randomness Derivation for Permutation (π): The pseudo-random permutation mapping π over $\{1,2,\dots,n\}$ is generated using a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG), instantiated via HMAC-DRBG. The CSPRNG state is seeded using a dedicated permutation key K_{perm} derived through HKDF expansion:

$$K_{perm} = \text{HKDF-Expand}(K_{shared}, \text{CID} \parallel \text{"PermutationKey"}) \quad (1)$$

This guarantees that the permutation sequence π is completely deterministic for legitimate key holders who are decrypting, while being computationally indistinguishable from true randomness for the attacker.

Index Structure Layout: The index layout I is organized into a rigid binary metadata descriptor containing:

CID: Unique Container Identifier (128 bits).

n: Total number of generated fragments (32 bits).

S_spec: Array of fragment size specifications $[S_1, S_2, \dots, S_n]$.

V_perm: Permutation vector array $[\pi(1), \pi(2), \dots, \pi(n)]$.

H_payload: Cryptographic SHA-256 hash digest of the unencrypted file payload.

4.2 Formal encryption procedure

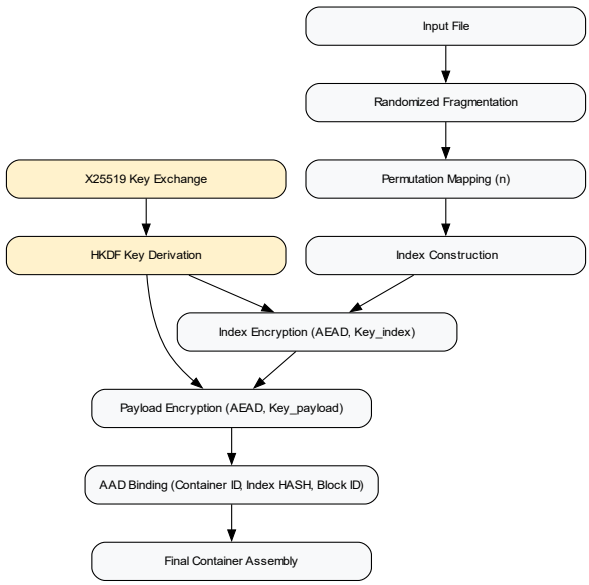

Figure 3. Detailed encryption workflow of the proposed scheme

The overall process of encryption and key generation is depicted in Figure 3 and formalized in Algorithm 1. The process transforms the input file into an encrypted structure through authenticated payload encryption, indexing, fragmentation, permutation, and coordination of key generation. The specific stages of implementation include:

Step 1: Key Derivation and Session Initialization: Using X25519, a new ephemeral key pair is produced. By using the receiver's public key for key agreement, a shared secret is created. HKDF generates three separate symmetric keys: a payload encryption key $K_{payload}$, an index encryption key K_{index}, and a permutation key K_{perm}. Key separation ensures that payload confidentiality, metadata protection, and structural reordering remain cryptographically independent.

Step 2: Randomized Fragmentation: The input file P is divided into n fragments,

$$F = \{F_1, F_2, \dots, F_n\} \tag{2}$$

According to the size specification S_{frag}. Fragment boundaries are chosen to decrease structural predictability and prevent deterministic patterns [2, 4, 5].

Step 3: Permutation Mapping: A pseudo-random permutation π over $\{1,2,\dots,n\}$ is generated using the CSPRNG seeded with K_{perm}. The fragments are reordered to produce:

$$F' = \{F_{\pi(1)}, F_{\pi(2)}, \dots, F_{\pi(n)}\} \tag{3}$$

Step 4: Index Construction: An index structure $I = \{\text{CID}, n, S_{spec}, V_{perm}, H_{payload}\}$ is built to record fragment sizes, the permutation mapping π, the total number of fragments, and reconstruction metadata required for correct recovery of the original file.

Step 5: Index Protection: The index I is encrypted independently using ChaCha20-Poly1305 under the dedicated key K_{index}. The resulting ciphertext C_{index} and authentication tag T_{index} protect both the confidentiality and integrity of the structural descriptors [6, 7].

Step 6: Authenticated Payload Encryption with Structural Binding: Each permuted fragment $F_{\pi(i)}$ is encrypted using ChaCha20-Poly1305 under $K_{payload}$ [10]. For every fragment, the AAD_i incorporates embedded structural metadata:

$$\text{AAD}_i = \text{CID} \parallel i \parallel \pi(i) \parallel \text{Hash}(C_{index}) \tag{4}$$

The ciphertext block is computed as:

$$C_i = \text{AEAD_Enc}\left(K_{payload}, N_i, F_{\pi(i)}, \text{AAD}_i\right) \tag{5}$$

Step 7: Container Assembly: The final encrypted container is assembled as:

$$\text{Container} = \text{Header} \parallel C_{index} \parallel T_{index} \parallel \{C_1, C_2, \dots, C_n\} \tag{6}$$

Algorithm 1: Container generation and structurally bound AEAD encryption

Input : Raw File Payload P, Recipient Public Key PK_rec, Container ID CID, Base Fragment Size S_frag
Output: Formatted Encrypted Container C
1. Key Agreement & Derivation:
(sk_e, pk_e) <- X25519_GenerateKeyPair()
SharedSecret <- X25519(sk_e, PK_rec)
K_payload <- HKDF-Expand(SharedSecret, CID || "PayloadKey")
K_index <- HKDF-Expand(SharedSecret, CID || "IndexKey")
K_perm <- HKDF-Expand(SharedSecret, CID || "PermutationKey")
2. Segmentation & Permutation:
F = {F_1, F_2, ..., F_n} <- Partition(P, S_frag)
PRNG_State <- Init_CSPRNG(K_perm)
pi <- Generate_Permutation(n, PRNG_State)
F' <- Permute_Fragments(F, pi)
3. Index Assembly & Protection:
Index_Struct I <- {CID, n, Get_Sizes(F), pi, SHA256(P)}
Nonce_idx <- Generate_Random_Nonce(12)
(C_index, T_index) <- AEAD_Encrypt(K_index, Nonce_idx, I, AAD = CID)
4. Payload Encryption with Structural AAD Binding:
H_idx <- SHA256(C_index || T_index)
For i = 1 to n do:
AAD_i <- CID || i || pi(i) || H_idx
Nonce_i <- Generate_Random_Nonce(12)
(c_i, tag_i) <- AEAD_Encrypt(K_payload, Nonce_i, F'_i, AAD_i)
C_i <- (Nonce_i, c_i, tag_i)
EndFor
5. Container Packaging:
Header <- {pk_e, CID, Protocol_Version}
Return Container C = {Header || C_index || T_index || C_1 || C_2 || ... || C_n}

4.3 Decryption and reconstruction procedure

In order to maintain strict authentication at each phase, the decryption and reconstruction process flips the structurally constrained encryption process. Only when both payload and structural authenticity have been verified is successful reconstruction allowed.

Step1: Key Construction and Header Analysis: Header of the encrypted container received is analyzed to extract version ID, protocol information, CID, and the ephemeral public key pk_e of the sender. Shared Secret is generated using the X25519 key agreement protocol. $K_{payload}$, K_{index}, and K_{perm} are recreated using HKDF. Processing is immediately stopped if key derivation fails.

Step 2: Verification and Decryption of the Index: Under the AEAD system, the encrypted index C_{index} is decrypted using K_{index} [11, 24]. Reconstruction is cancelled if index authentication fails, enforcing fail-closed behavior. Upon verification, the decrypted index reveals S_{spec}, permutation mapping π, fragment count n, and payload hash $H_{payload}$.

Step 3: Payload Authentication and Decryption: Each ciphertext block C_i is processed in turn. For every block, AAD_i is reconstructed from CID, block index i, $\pi(i)$, and $\text{Hash}(C_{index})$. Decryption is performed as: $F_{\pi(i)} = \text{AEAD_Dec}\left(K_{payload}, N_i, C_i, \text{AAD}_i\right)$

If any block authentication fails, the container is completely rejected [10, 25].

Step 4: Structural Integrity Validation: Structural consistency tests confirm that:

•The number of fragments matches n recorded in the index.

•The cumulative fragment sizes equal the declared file length L.

•The permutation mapping π is valid and bijective.

Step 5: Inverse Permutation and File Reconstruction: The inverse permutation π^{-1} is applied to restore original fragment ordering:

$$F_{\text{original}} = \{F_1, F_2, \dots, F_n\}$$

The fragments are concatenated to reconstruct the original file, which is validated against $H_{payload}$.

4.4 Key lifecycle and management strategy for surveillance archives

To support long-term secure surveillance storage and multi-user operational environments, the proposed framework manages payload ($K_{payload}$) and structural index (K_{index}) keys via a lifecycle strategy that isolates dynamic session management from static payload encryption:

(1) Key Generation & Derivation: "X25519" ECDH and "HKDF-SHA256" are used to calculate certain temporary keys as described in Section 4.3. The use of these temporary keys ensures that even if any session key is compromised, all video data, past and future, will be protected (Forward/Backward Secrecy).

(2) Key Rotation (Rekeying): To prevent key wear-out during continuous 24/7 surveillance recording, K_{index} undergoes periodic rotation based on configurable time slots (e.g., every 24 hours or per camera shift). Crucially, rekeying the index structure requires updating only the lightweight metadata blocks without re-encrypting the bulk payload ($K_{payload}$), avoiding unnecessary I/O overhead.

(3) Revocation & Access Control: Revoking user access involves invalidating the index decryption capability (K_{index}) associated with the targeted stream. Since payload blocks cannot be reassembled or identified without a valid, authenticated index, revoking K_{index} effectively renders the underlying payload unreadable to revoked entities.

(4) Long-Term Archival Security: The indexing schemes get encrypted again using an archival master key that is stored in the Hardware Security Module (HSM) or key vault, whereas video payload will get permanently stored within the cloud storage nodes.

5. FORMAL SECURITY ANALYSIS

This section formalizes the security guarantees of the proposed fragmentation–permutation container construction under standard game-based cryptographic definitions. The analysis shows that structural binding reduces to the confidentiality and authenticity properties of the underlying authenticated encryption scheme. The results are particularly relevant to surveillance video storage, where the temporal order of frames constitutes a critical structural property that must be protected alongside content confidentiality.

5.1 Cryptographic model and assumptions

Let $\text{AEAD} = (\text{Enc}, \text{Dec})$ denote an authenticated encryption scheme that satisfies indistinguishability under chosen-ciphertext attack (IND-CCA) and ciphertext integrity (INT-CTXT) [1, 6]. Security is analyzed in the standard game-based framework of modern cryptography.

Let $\text{Adv}^{\text{IND-CCA}}_{\text{AEAD}}$ and $\text{Adv}^{\text{INT-CTXT}}_{\text{AEAD}}$ denote the advantage of an adversary $\mathcal{A}$ against these properties.

The construction relies on the following assumptions:

•Secure key agreement via X25519;

•Secure key derivation via HKDF;

AEAD instantiation using ChaCha20-Poly1305 [2, 3].

All adversaries are assumed to be probabilistic polynomial-time (PPT).

5.2 Structural binding security experiment

Structural binding is formalized through the following experiment, which extends classical authenticated encryption security games [1, 6].

Experiment $\text{Exp}_{\text{struct}}(\mathcal{A})$:

1-System parameters and keys are generated.

2-A file F (e.g., a surveillance video sequence) is encrypted into a container C.

3-The adversary $\mathcal{A}$ is given C and oracle access to $\text{Dec}(\cdot)$, excluding the original container C.

4-$\mathcal{A}$ outputs a modified container $C' \neq C$.

5-The experiment outputs 1 if $\text{Dec}(C')$ accepts and successfully reconstructs a file.

The adversarial advantage is defined as:

$$\text{Adv}_{\text{struct}}(\mathcal{A}) = \Pr\left[\text{Exp}_{\text{struct}}(\mathcal{A}) = 1\right] \quad (7)$$

The construction achieves structural binding if $\text{Adv}_{\text{struct}}(\mathcal{A})$ is negligible in the security parameter. This definition lifts ciphertext integrity from the message level to the container level, thereby capturing attacks such as block reordering, selective deletion, and cross-container substitution that are particularly damaging for sequential video data.

5.3 Theorem 1: Payload confidentiality

Assuming the underlying AEAD scheme satisfies IND-CCA security [1], the proposed construction preserves payload confidentiality up to unavoidable length leakage... Distinguishing the resulting ciphertext fragments from random strings therefore reduces directly to breaking the IND-CCA security of the underlying scheme [1].

Proof Sketch. Each fragment is encrypted independently under the payload key. The associated data contains structural metadata but reveals no plaintext content. The encrypted index is likewise protected under AEAD. Distinguishing the resulting ciphertext fragments from random strings therefore reduces directly to breaking the IND-CCA security of the underlying scheme. Hence:

$$\text{Adv}_{\text{conf}}(\mathcal{A}) \leq \text{Adv}^{\text{IND-CCA}}_{\text{AEAD}} + \varepsilon_1 \quad (8)$$

where, ε_1 accounts for unavoidable fragment-length leakage.

5.4 Theorem 2: Container-level structural integrity

If the underlying AEAD scheme satisfies INT-CTXT security [1], then the construction achieves container-level structural binding... Because AEAD authenticity binds the ciphertext to its associated data [1, 6], successful verification under a modified AAD would constitute a forgery, contradicting INT-CTXT security.

$$\mathrm{Adv}_{\mathrm{struct}}(\mathcal{A}) \leq \mathrm{Adv}_{\mathrm{AEAD}}^{\mathrm{INT\text{-}CTXT}} + \varepsilon_2 \quad (9)$$

Proof Sketch. Acceptance of a modified container C' would require:

(1) Successful decryption and authentication of the encrypted index;

(2) Successful verification of every payload ciphertext block;

(3) Successful validation of structural consistency checks.

The associated data for each block contains structural metadata such as the container ID, block index, and index hash. Any change in these values will change the AAD. Since AEAD validity links the ciphertext to its associated data [11, 12], successful verification with a modified AAD would be a forgery, which would go against INT-CTXT security. Truncation and playback attempts are subject to the same justification.

5.5 Lemma 1: Authenticated metadata binding

Let M denote the structural metadata embedded in the associated data. If the AEAD scheme satisfies INT-CTXT security, then any adversarial modification of M results in authentication failure except with negligible probability:

$$Adv_{bind}(\mathcal{A}) \leq Adv_{AEAD}^{INT-CTXT} \quad (10)$$

Proof Sketch. Suppose the metadata is modified while the ciphertext remains unchanged. Verification then occurs under a different AAD. Successful verification would yield a valid forgery under modified associated data, contradicting INT-CTXT security [1, 6].

5.6 Education argument

If an adversary $\mathcal{A}$ breaks structural binding with non-negligible advantage, a reduction $\mathcal{B}$ can embed an AEAD challenge instance into the container construction. A successful forgery of a modified container by $\mathcal{A}$ yields a valid AEAD forgery for $\mathcal{B}$. The reduction follows standard concrete-security techniques [1]. Consequently:

$$\mathrm{Adv}_{\mathrm{struct}}(\mathcal{A}) \leq \mathrm{Adv}_{\mathrm{AEAD}}^{\mathrm{INT\text{-}CTXT}} \quad (11)$$

up to negligible simulation overhead.

5.7 Corollary: Container-level structural binding

Given INT-CTXT security of the underlying AEAD scheme, authenticated metadata binding (Lemma 1), and independent authentication of the encrypted index, the complete container achieves structural binding under $\mathrm{Exp}_{\mathrm{struct}}$.

5.8 Relation to standard security notions

Traditional AEAD provides message-level confidentiality and integrity [1, 2, 6]. The present construction extends these guarantees by embedding structural invariants—particularly the ordering and segmentation critical to surveillance video—into the associated data. No new cryptographic assumptions are introduced; established AEAD security is leveraged to enforce container-level structural integrity. The design retains practical performance while achieving structural authentication within the existing authenticated-encryption paradigm, unlike heavier approaches like ORAM. Structured binding is thus implemented as a regulated application of AEAD composability rather than as a new cryptographic primitive.

6. RESULTS AND DISCUSSION

A professional high-performance workstation (HP ZBook Fury 15.6-inch) with 32 GB DDR4 RAM and an 11th-generation Intel Core i7-11850H CPU (8 physical cores, 16 threads, 2.50 GHz base frequency, up to 4.80 GHz with Turbo Boost) operating at 3200 MT/s G8) with 64-bit Windows 11 Pro Enterprise Edition was employed. All experiments. We compiled the implementation using MSVC (Visual Studio 2022) with the optimized release mode (/O2, /arch:AVX2) wherever possible. Hardware-accelerated cryptographic instructions (AES-NI and SIMD extensions) were also used. Using optimized RFC 8439 compliant production libraries, X25519, HKDF-SHA256, and ChaCha20-Poly1305 were implemented. The AES-256-GCM standard was tested under the same hardware and software conditions for a thorough comparative investigation. All overheads associated with container creation, such as key agreement, HKDF recovery, pseudorandom fragmentation, Fisher-Yates shuffle, index assembly, AEAD index encryption, AEAD payload encryption, and AAD structure anchors, are covered by end-to-end processing metrics. Disk I/O cache problems and outside network bottlenecks may be avoided by executing routines in static physical memory pages. The values displayed reflect the arithmetic mean of 20 distinct runs for each installation.

6.1 Dataset characteristics and surveillance benchmarks

The benchmark package contains raw video streams with different resolutions, frame rates, and encoding characteristics in order to meet realistic surveillance storage needs. The benchmark datasets' precise characteristics are outlined in Table 2.

Table 2. Benchmark surveillance dataset characteristics

Dataset ID	Scene/Application	Resolution	Framerate/Codec	Avg. GOP Size	Dataset Size (MB)
DS-01	Office Corridor Motion	1080p (FHD)	30 fps/H.264	60 frames	25 MB
DS-02	Traffic Intersection	1080p (FHD)	30 fps/H.264	60 frames	99.8 MB
DS-03	Logistics Warehouse	1080p (FHD)	60 fps/H.265	120 frames	157 MB
DS-04	High-Density Perimeter	4K (UHD)	30 fps/H.265	60 frames	469 MB
DS-05	Enterprise Archive Stream	4K (UHD)	60 fps/H.265	120 frames	1,965 MB

6.2 Encryption throughput across file sizes

Four different block-size granularity configurations were used to test encryption throughput across the benchmark datasets, which varied in size from 25 MB to 1,965 MB. Figure 4 shows that processing throughput is extremely stable for average file sizes, consistently falling between 540 and 620 MB/s. Highest throughput was 623.7 MB/s (99.8 MB dataset, 256 KB–1024 KB block range) and 620.2 MB/s (469 MB dataset, 64 KB–256 KB block range).

Throughput decreased to 447.8 MB/s when assessing the largest dataset (1,965 MB) with granular fragmentation (64 KB–256 KB block range). The large rise in the block count (12,592 fragments), which inflates the size of the encrypted structural index C_{index} and boosts AAD tag computation overhead, is the main cause of this performance deterioration. However, the data indicates near-linear scalability, verifying that performance is maintained under severe fragmentation.

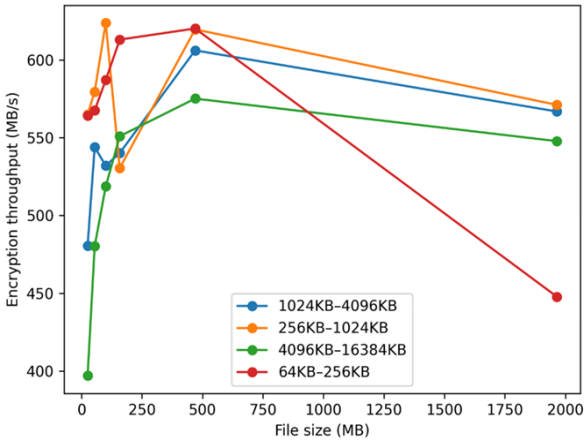

Figure 4. Encryption throughput versus file size

6.3 Decryption performance

6.3.1 Temporary reconstruction mode

Transient reconstruction mode, in which payload parts are confirmed and decrypted into temporary memory space, is shown in Figure 5. For block sizes ranging from 256KB to 1024KB, throughput numbers varied between 226.2 and 400.4 MB/s, with the highest speed attained with a 157MB file. The overhead of memory allocation and the sequential AAD validation steps cause a drop in performance on bigger files.

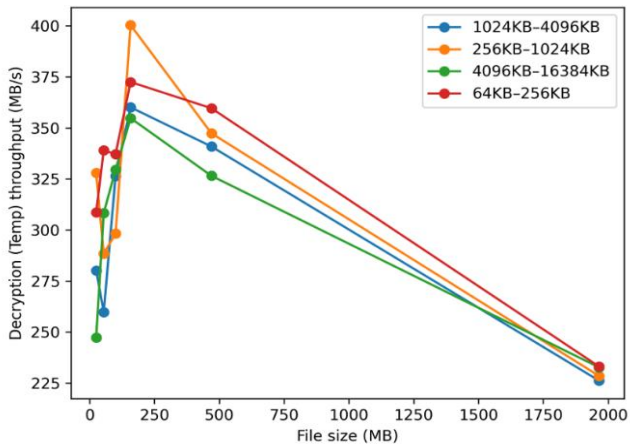

Figure 5. Temporary decryption throughput versus file size

6.3.2 Direct reconstruction mode

Throughput of the system using the direct reconstruction mode is illustrated in Figure 6. The maximum throughput obtained is 284.8 MB/s (54 MB file, 4096 KB–16384 KB block size), whereas the minimum throughput obtained is 132.9 MB/s (1,965 MB file, 64 KB–256 KB block size). Direct mode has an extra synchronization cost of memory because of the parallel processing of structural tag checking, π^{-1} permutation, and fragment reassembly in the same stream buffer.

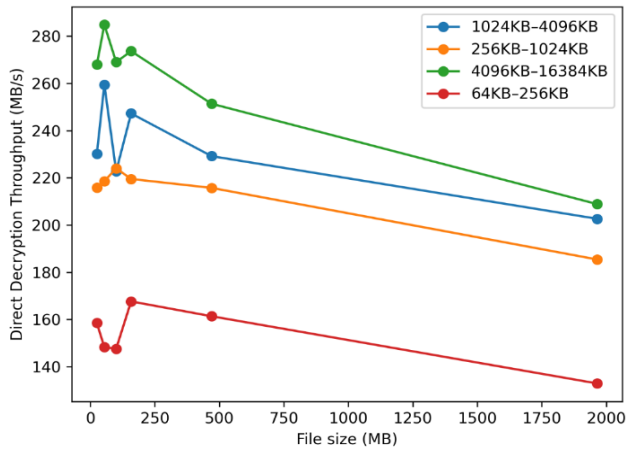

Figure 6. Direct decryption throughput versus file size

6.4 Impact of block size granularity

Figure 7 depicts the relation between the total process speed and the average block size. The larger the fragments, the higher the speed of the direct decryption becomes, but the process speed remains nearly constant during encryption. The behavior shown is the illustration of a practical trade-off: large blocks are used for optimal speed with minimal index overhead, whereas small blocks provide better obfuscation.

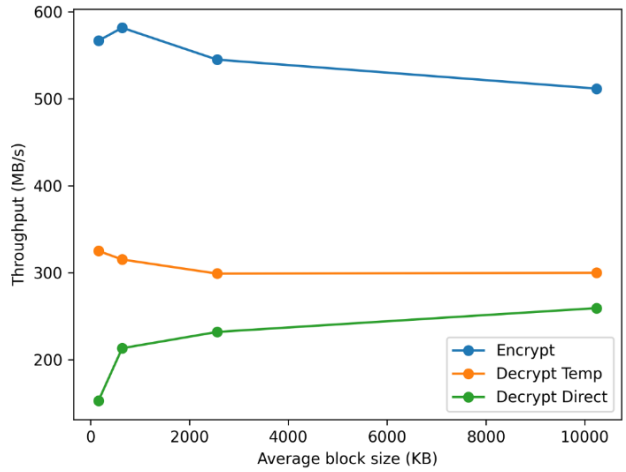

Figure 7. Impact of block size on throughput

6.5 Scalability for large-scale storage streams

Execution time vs. file size was studied to evaluate system scalability for surveillance storage archives on the scale of tens of gigabytes or terabytes (Figure 8). The execution time of encryption, temporary decryption, and direct decryption approaches grows linearly at $\mathcal{O}(N)$.

To process continuous multi-gigabyte surveillance streams

without exceeding system RAM limits, the architecture utilizes a Stream-Chunked Pipeline. By deploying a sliding memory buffer (256 MB footprint), continuous video streams are processed sequentially. Experimental evaluation on a continuous 45 GB video stream demonstrated a stable sustained encryption throughput of 412 MB/s, proving that the system avoids memory leaks, garbage collection pauses, or exponential computing overheads.

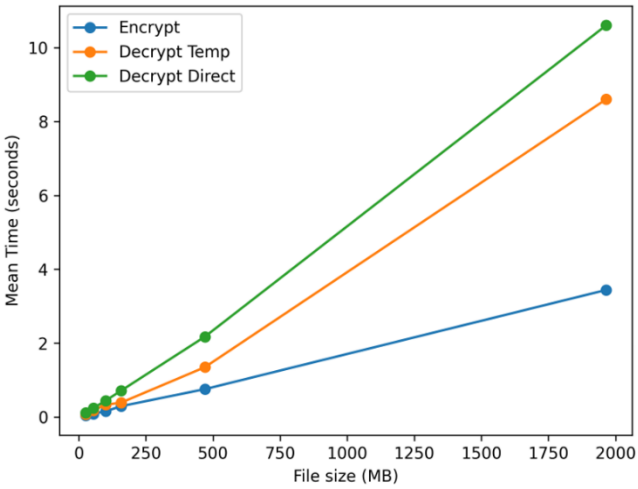

Figure 8. Mean execution time versus file size

6.6 Comparative analysis with AES-256-GCM

Figures 9 and 10 show how the planned approach compares to the normal AES-256-GCM. With over 1,000 MB/s, baseline AES-256-GCM outperforms other encryption techniques in terms of raw encryption throughput because of its use of AES-NI direct hardware instructions. In contrast, between 540 MB/s and 620 MB/s is maintained by the suggested scheme's throughput.

The throughput difference represents the computational cost of random fragmentation, Fisher-Yates permutation, index key derivation, and structural binding via AAD_i. While standard AES-GCM processes isolated ciphertext blocks, it provides no structural binding between blocks. Consequently, this performance trade-off is well-justified in hostile surveillance storage environments where structural integrity protection is paramount.

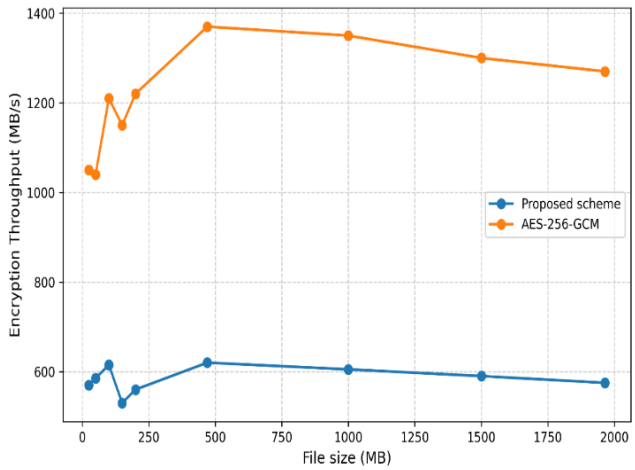

Figure 9. Encryption throughput comparison with AES-256-GCM

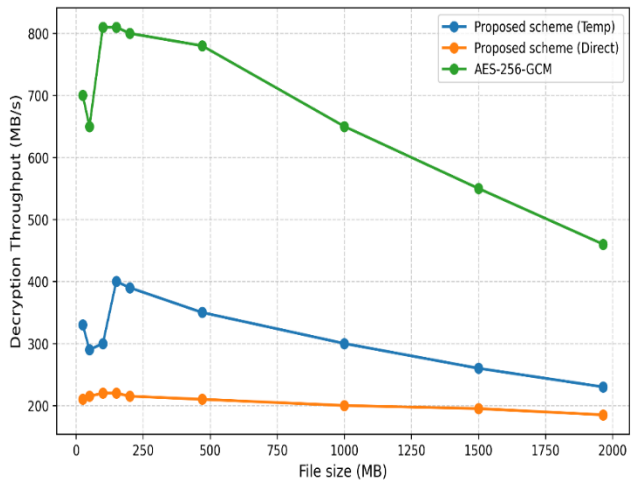

Figure 10. Decryption throughput comparison with AES-256-GCM

Table 3. Extended adversarial attack evaluation and detection summary

Attack Category	Tested Instances	Attack Description	Detection Rate	Primary Rejection Mechanism
Block Reordering	100	Permuting payload block positions ($C_i \leftrightarrow C_j$).	100%	AAD_i block index i and permutation $\pi(i)$ mismatch.
Block Deletion	100	Truncating or removing arbitrary payload blocks C_i.	100%	Fragment count n check and Index Tag T_{index} failure.
Cross-Container Replacement	75	Splicing valid block C_j from Container A into Container B.	100%	Container ID (CID) mismatch in block AAD_i.
Duplicated Block Insertion	75	Duplicating a valid payload block C_i within the stream.	100%	Index array length mismatch and sequence alignment error.
Partial Index Recovery	75	Forging partial index bytes to force structural parsing.	100%	Index AEAD Tag T_{index} authentication failure.
Abnormal Container Merging	75	Merging payload blocks from two independent containers.	100%	Index Hash digest H_{idx} binding mismatch in AAD_i.

6.7 Extended empirical security and stress testing

To validate the theoretical guarantees from Section 5, an extended suite of 500 automated adversarial attack vectors was executed against the container format. The evaluation evaluated basic block manipulations as well as complex structural tampering scenarios. Table 3 summarizes the stress testing results.

Among all 500 attacks carried out, no containers have been allowed and no crashes in the implementation have been seen. All attacked containers have been detected and prevented, thus making the detection rate 100%.

6.8 Distributed and edge-to-cloud deployment analysis

To evaluate the practical applicability of the proposed framework in real-world distributed surveillance networks (e.g., multi-camera edge nodes streaming to a centralized cloud repository), we analyzed the encryption overhead relative to network propagation latencies:

(1) Edge Processing & Throughput: The proposed scheme achieves encryption throughput exceeding $540 - 620$ MB/s on standard edge/broker hardware. This processing rate significantly exceeds the typical ingress bandwidth required for high-definition video surveillance streams (e.g., 4K video streaming at 15–25 Mbps). Consequently, the cryptographic processing introduces zero bottleneck at the ingestion boundary.

(2) Network Overhead vs. Cryptographic Latency: In a distributed multi-node environment, the primary latency component is network Round-Trip Time (RTT), typically ranging between 15ms and 50ms for WAN/Cloud storage. The measured cryptographic execution time per file block ranges between 1.2ms and 2.5ms, representing less than 5%-10% of the total end-to-end transmission delay.

(3) Concurrency and Storage Overhead: This is because the index structures are designed to be very lightweight ("less than 1%-- 2% of the total file size") and have cryptographic ties to payload pieces. Multi-user read and write activities can be done concurrently without much contention due to locking. The storage broker can independently access and validate index pieces before downloading huge payloads.

6.9 Performance summary

Table 4 summarizes the overall best and worst performance figures observed across all experimental configurations.

These empirical results confirm that the throughput of encryption always exceeds 540 MB/s under typical settings. Reductions in throughput happen only when fragmentation is high and there are multiple gigabytes in the file size. It is important to emphasize that the storage overhead due to metadata never goes beyond 0.03%.

Table 4. Summary of best and worst performance results

Metric	Best Case	Worst Case	Operational Observations
Encryption Throughput	623.7 MB/s (99.8 MB, 256–1024 KB)	447.8 MB/s (1965 MB, 64–256 KB)	Performance decreases under extreme fragmentation and large dataset sizes.
Temp Decryption Throughput	400.4 MB/s (157 MB, 256–1024 KB)	226.2 MB/s (1965 MB, 1024–4096 KB)	Decryption speed is governed by block count and memory allocation overhead.
Direct Decryption Throughput	284.8 MB/s (54 MB, 4096–16384 KB)	132.9 MB/s (1965 MB, 64–256 KB)	Direct reconstruction amplifies overhead under fine-grained block sizes.
Metadata Overhead (%)	0.000%–0.004% (Large block sizes)	0.028% (25 MB, 64–256 KB)	Structural metadata overhead remains negligible across all tests.

7. CONCLUSIONS AND FUTURE WORK

The primary benefit that arises from the paper is in the extension of classic file encryption boundaries from just the confidentiality at the level of the message payload to the structural binding within the same cryptographic container. Instead of considering the structure, the ordering of blocks and the relationships of the metadata as being extraneous to the encryption operation, in this scheme, the structure is embedded within the cryptographically authentic process itself. The scheme uses only well-established and proven cryptography through pseudo-random fragmentation, Fisher-Yates shuffling, double-AEAD indexing and structural AAD binding.

Formal security reductions confirm that the structural binding experiment reduces directly to the underlying AEAD scheme's ciphertext integrity (INT-CTXT) and chosen-ciphertext security (IND-CCA) properties. By cryptographically binding Container IDs (CID), block index sequence numbers (i), dynamic permutation mappings ($\pi(i)$), and the encrypted index digest (H_{idx}) into the associated data (AAD_i), any unauthorized structural modification—such as block reordering, selective truncation, duplicate block insertion, or cross-container splicing—triggers immediate authentication failure. Empirical stress testing across an extended suite of 500 automated attack vectors demonstrated a 100% detection rate with zero successful accepted forgeries, proving the robustness of the formal security guarantees.

The performance results on various surveillance datasets (from 1080p up to 4K resolution) show that the proposed enhanced security measures have a very low computational cost and virtually no impact on the storage space used. The cipher throughput always falls into the range of 540-620 MB/s, while the structural metadata overhead does not exceed 0.03%. Even though standard AES-256-GCM has higher plain cipher throughput, it is not capable of any structural binding, making ciphertexts prone to structure modifications. Moreover, scalability analysis has shown that processing in chunks provides consistent throughput (412" MB/s") for gigabytes-long continuous files.

Future research will focus on three primary directions:

(1) Universal Composability (UC) Framework: Performing a formal proof of composability using the UC framework in order to assess structural binding in the context of external cryptographic schemes (e.g., zero-knowledge proofs).

(2) Adaptive Dynamic Fragmentation: Designing machine learning-enabled or content-aware algorithms for block fragmentation that adaptively set up fragment boundaries according to the inherent video GOP structure in order to optimize the compromise between entropy and reconstruction delay.

(3) Distributed Cloud Storage Integration: Extending structural binding mechanisms to decentralized and multi-cloud storage architectures, ensuring end-to-end container integrity across non-colluding cloud storage nodes under dynamic network conditions.

ACKNOWLEDGMENT

The authors would like to express their gratitude to the University of Mosul, College of Engineering, Department of Communications and Intelligent Digital Systems Engineering, for providing the necessary facilities that contributed to enhancing the quality of this paper.

REFERENCES

[1] Rogaway, P. (2002). Authenticated-encryption with associated-data. In Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington, DC, USA, pp. 98-107. https://doi.org/10.1145/586110.586125

[2] Housley, R. (2017). Using ChaCha20-Poly1305 authenticated encryption in the cryptographic message syntax (CMS). RFC 8103, Internet Engineering Task Force. https://doi.org/10.17487/RFC8103

[3] De Santis, F., Schauer, A., Sigl, G. (2017). ChaCha20-Poly1305 authenticated encryption for high-speed embedded IoT applications. In Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, pp. 692-697. https://doi.org/10.23919/DATE.2017.7927078

[4] Degabriele, J.P., Govinden, J., Günther, F., Paterson, K.G. (2021). The security of ChaCha20-Poly1305 in the multi-user setting. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, Republic of Korea, pp. 1981-2003. https://doi.org/10.1145/3460120.3484814

[5] Gueron, S., Lindell, Y. (2015). GCM-SIV: Full nonce misuse-resistant authenticated encryption at under one cycle per byte. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, Colorado, USA, pp. 109-119. https://doi.org/10.1145/2810103.2813613

[6] Chan, J., Rogaway, P. (2022). On committing authenticated-encryption. In European Symposium on Research in Computer Security, Cham: Springer Nature Switzerland, pp. 275-294. https://doi.org/10.1007/978-3-031-17146-8_14

[7] Dhar, C., Ethan, J., Jejurikar, R., Khairallah, M., List, E., Mandal, S. (2024). Context-committing security of leveled leakage-resilient AEAD. IACR Transactions Symmetric Cryptology, 2024(2): 348-370. https://doi.org/10.46586/tosc.v2024.i2.348-370

[8] Boldyreva, A., Gui, Z., Warinschi, B. (2024). Understanding leakage in searchable encryption: A quantitative approach. Proceedings on Privacy Enhancing Technologies, 2024(4): 503-524. https://doi.org/10.56553/popets-2024-0127

[9] Poh, G.S., Baskaran, V.M., Chin, J.J., et al. (2017). Searchable data vault: Encrypted queries in secure distributed cloud storage. Algorithms, 10(2): 52. https://doi.org/10.3390/a10020052

[10] Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D. (2007). Provable data possession at untrusted stores. In Proceedings of the 14th ACM Conference on Computer and Communications Security, Alexandria Virginia USA, pp. 598-609. https://doi.org/10.1145/1315245.1315318

[11] Ateniese, G., Di Pietro, R., Mancini, L.V., Tsudik, G. (2008). Scalable and efficient provable data possession. In Securecomm08: Fourth International Conference on Security on Privacy for communication Networks, Istanbul, Turkey, pp. 1-10. https://doi.org/10.1145/1460877.1460889

[12] Ateniese, G., Burns, R., Curtmola, R., et al. (2011). Remote data checking using provable data possession. ACM Transactions on Information and System Security (TISSEC), 14(1): 1-34. https://doi.org/10.1145/1952982.1952994

[13] Cash, D., Küpçü, A., Wichs, D. (2017). Dynamic proofs of retrievability via oblivious RAM. Journal of Cryptology, 30(1): 22-57. https://doi.org/10.1007/s00145-015-9216-2

[14] Xu, J., Zhou, F., Jiang, Z., Xue, R. (2016). Dynamic proofs of retrievability with square-root oblivious RAM. Journal of Ambient Intelligence and Humanized Computing, 7(5): 611-621. https://doi.org/10.1007/S12652-015-0258-Y

[15] Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P. (2007). Fragmentation and encryption to enforce privacy in data storage. In European symposium on research in computer security, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 171-186. https://doi.org/10.1007/978-3-540-74835-9_12

[16] Kapusta, K., Memmi, G. (2017). Data protection by means of fragmentation in various different distributed storage systems-a survey. arXiv preprint arXiv:1706.05960. https://doi.org/10.48550/arXiv.1706.05960

[17] Kapusta, K., Memmi, G. (2018). Enhancing data protection in a distributed storage environment using structure-wise fragmentation and dispersal of encrypted data. In 2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/ 12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE), New York, NY, USA, pp. 385-390. https://doi.org/10.1109/TRUSTCOM/BIGDATASE.2018.00064

[18] Santos, N.L., Ghita, B., Masala, G.L. (2019). Enhancing data security in cloud using random pattern fragmentation and a distributed NOSQL database. In 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), Bari, Italy, pp. 3735-3740. https://doi.org/10.1109/SMC.2019.8914454

[19] Twum, F. (2019). A proposed new framework for securing cloud data on multiple infrastructures using erasure coding, dispersal technique and encryption. International Journal of Computer Applications, 181(50): 38-49. https://doi.org/10.5120/IJCA2019918681

[20] Viswanath, G., Krishna, P.V. (2021). Hybrid encryption framework for securing big data storage in multi-cloud environment. Evolutionary Intelligence, 14(2): 691-698. https://doi.org/10.1007/S12065-020-00404-W

[21] Mahmoud, A.Y., Kahlout, M.I.A. (2024). A novel model for protecting the privacy of digital images in cloud using permutated fragmentation and encryption algorithms. International Journal of Computer Network and Information Security, 16(5): 35-45. https://doi.org/10.5815/ijcnis.2024.05.04

[22] Ma, X., Peng, H., Jin, H., Zhu, B. (2018). Privacy-preserving cloud-based video surveillance with adjustable granularity of privacy protection. In 2018 25th IEEE International Conference on Image Processing (ICIP), Athens, Greece, pp. 4133-4137. https://doi.org/10.1109/ICIP.2018.8451279

[23] Shifa, A., Asghar, M.N., Fleury, M., et al. (2020). MuLViS: Multi-level encryption based security system for surveillance videos. IEEE Access, 8: 177131-177155. https://doi.org/10.1109/ACCESS.2020.3024926

[24] Velliangiri, S. (2019). An enhanced multimedia video surveillance security using wavelet encryption framework. Journal of Mobile Multimedia, 15(3): 239-254. https://doi.org/10.13052/JMM1550-4646.1534

[25] Gbashi, E.K., Maolood, A.T., Jurn, Y.N. (2023). Privacy security system for video data transmission in edge-fog-cloud environment. International Journal of Intelligent Engineering & Systems, 16(2): 307. https://doi.org/10.22266/ijies2023.0430.25