SHM PUBLISHER
SURYA HIJAU MANFAAT

Journal of Information System Exploration and Research

https://joiser.shmpublisher.com

p-ISSN 2964-1160 | e-ISSN 2963-6361

SentinelKEM: Securing AI Model Distribution Using Hybrid Cryptography

Singgih Naufal[1], Eko Hari Rachmawanto[2*], Mohamed Doheir[3]
[1,2]*Department of Informatics Engineering, Universitas Dian Nuswantoro, Indonesia*
[3]*Department of Technology Management, Universiti Teknikal Malaysia Melaka, Malaysia*

DOI: https://doi.org/10.52465/joiser.v4i3.76
Received 04 June 2026; Accepted 17 July 2026; Available online 21 July 2026

Article Info

Keywords:
AI model protection;
AES-256-GCM;
Kyber/ML-KEM;
Ed25519;
Post-quantum

Abstract

Deep neural network (DNN) models are valuable intellectual property that can be copied or redistributed without authorization during distribution. Existing protection methods mainly rely on watermarking, which verifies ownership but does not secure model files during transfer. This study proposes SentinelKEM, a hybrid cryptographic framework that combines AES-256-GCM for authenticated encryption, Kyber/ML-KEM-768 with HKDF-derived key encryption keys for post-quantum key protection, Ed25519 digital signatures for ownership verification, and scrypt for passphrase-based private-key security. Encrypted models and cryptographic metadata are packaged into a secure ZIP archive, while private keys are distributed separately. During decryption, SHA-256 integrity verification and Ed25519 authentication are performed before model restoration. SentinelKEM was implemented as a Python Streamlit application and evaluated on various AI model formats, including .h5, .pt, .pth, .pkl, and .joblib. Experimental results showed successful encryption and decryption, reliable ownership authentication, effective tamper detection, and a constant cryptographic overhead of approximately 4.5 KB regardless of model size. Unlike watermarking, SentinelKEM protects AI models before recipient access through post-quantum key encapsulation and authenticated encryption, providing practical and robust security for AI model distribution.

1. Introduction

The rapid advancement of artificial intelligence (AI), particularly deep neural network (DNN) models, has transformed numerous application domains over the past decade. DNN-based systems are now widely deployed in critical areas such as computer vision, natural language processing, medical diagnosis, and autonomous vehicles [1], [2], [3]. The development of these models typically requires

* ***Corresponding Author:***

Eko Hari Rachmawanto,
Department of Informatics Engineering,
Universitas Dian Nuswantoro,
Indonesia.
Email: eko.hari.rachmawanto@dsn.dinus.ac.id

substantial investments in large-scale data collection, high-performance computing infrastructure, and specialized expertise in model architecture design [2]. As a result, trained AI models have become valuable intellectual property (IP) assets that represent significant technological and economic value for their developers [1], [4].

The growing value of AI models has also increased their exposure to intellectual property violations. Once a model is distributed through cloud platforms, public APIs, or physical storage media, it becomes vulnerable to unauthorized copying, illegal redistribution, and unlicensed use [1], [4], [3]. Fkirin et al. [4] reported that distributed models can be replicated and redistributed with little difficulty, often without the knowledge of the original owner. Similarly, Liu and Xu [2] highlighted the urgent need for effective copyright protection mechanisms as DNN models become increasingly susceptible to IP infringement.

To address these concerns, researchers have proposed a variety of AI model protection techniques that can generally be categorized into two paradigms: white-box watermarking and black-box watermarking [1], [2]. In white-box approaches, ownership information is embedded directly into the model parameters, and verification requires access to the internal weights of the model [1], [5]. In contrast, black-box approaches establish ownership by analyzing the model's responses to specific trigger inputs without exposing its internal structure [6], [7], [8]. Numerous extensions of these approaches have been developed, including wet paper coding techniques [6], backdoor-based watermarking for speech recognition models [5], public-key infrastructure frameworks such as PK-Judge[9], multi-layer ownership protection schemes [10], [3], and dynamic GAN-based watermarking methods for federated learning environments [7].

Despite their effectiveness in ownership verification, existing watermarking techniques share a common limitation. Protection mechanisms become effective only after the recipient has obtained access to the model, leaving the file distribution stage largely unprotected. Consequently, these approaches cannot prevent unauthorized copying or redistribution of model files before ownership verification takes place. El Hajjar et al. [8] observed that many watermarking methods struggle to achieve both robustness and security simultaneously. Kanakri and King [9] further noted that traditional symmetric approaches remain vulnerable to misuse in ownership disputes. In addition, Li et al. [11] emphasized that large-scale model deployment, particularly in edge computing environments, increases the risk of intellectual property violations. These limitations suggest the need for a complementary protection mechanism that operates at the file-distribution level before the model reaches any recipient.

At the same time, advances in quantum computing have introduced a new security challenge. Conventional public-key cryptographic algorithms, including RSA and Elliptic Curve Cryptography (ECC), are theoretically vulnerable to quantum attacks based on Shor's algorithm [12], [13]. Zhang et al. [12] argued that recent developments, such as Google's Willow processor, indicate that practical quantum cryptanalysis may become feasible within the next five to ten years. Olushola and Meenakshi [14] highlighted the threat of harvest-now-decrypt-later (HNDL) attacks, in which encrypted data collected today may be decrypted once sufficiently powerful quantum computers become available. This concern has accelerated interest in lattice-based key encapsulation mechanisms (KEMs) for a variety of applications, including Internet of Things (IoT) environments [15]. Ngwenya and Ndlovu [16] identified CRYSTALS-Kyber as one of the most promising post-quantum cryptographic schemes currently available. In recognition of its security and practicality, the National Institute of Standards and Technology (NIST) standardized Kyber as a Key Encapsulation Mechanism under FIPS 203, where it is formally designated as ML-KEM [14], [16]

To address both the distribution-layer protection gap and emerging quantum threats, this study proposes SentinelKEM, a hybrid cryptographic framework for securing AI model distribution. The proposed framework integrates four complementary cryptographic layers: (1) AES-256-GCM for authenticated encryption of model files, (2) Kyber/ML-KEM-768 as a post-quantum Key Encapsulation Mechanism for protecting AES keys through HKDF-derived Key Encryption Keys (KEKs), (3) Ed25519 digital signatures for cryptographically binding ownership information to watermark metadata, and (4) scrypt as a memory-hard Key Derivation Function (KDF) for passphrase-protected private key storage. AES-256-GCM was selected because of its authenticated encryption capabilities and its demonstrated performance advantages over ChaCha20-Poly1305 in several comparative studies [17] , [18]. Kyber/ML-KEM-768 was chosen due to its status as the NIST-standardized post-quantum KEM and its security level comparable to AES-192 [12], [14]. Furthermore, Nagy et al. [19] reported that ML-KEM provides performance suitable for practical deployment scenarios. Ed25519 was adopted because of its

deterministic signature generation and compact key size [20], while SHA-256 was employed for integrity verification because no practical collision attacks are currently known and the algorithm remains compliant with both NIST and ISO standards [21].

2. Literature Review

SentinelKEM is designed based on a layered security architecture, where multiple cryptographic mechanisms work together to provide comprehensive protection for AI model distribution. Rather than relying on a single security primitive, the framework combines complementary cryptographic components to address confidentiality, integrity, ownership verification, and post-quantum resilience. Each component was selected through a review of the relevant literature, considering both its security properties and its suitability for protecting distributed AI models. Table 1 summarizes the cryptographic components adopted specifically in the proposed SentinelKEM framework, together with the rationale for their selection.

Table 1. Cryptographic components and selection rationale

Component	Algorithm	Function	Selection Rationale
Payload Encryption	AES-256-GCM	Authenticated encryption of model files	AEAD scheme with superior performance compared to ChaCha20-Poly1305 [17], [18]
Key Encapsulation	Kyber/ML-KEM-768	Protection of AES keys against quantum threats	NIST FIPS 203 standard based on the MLWE problem [12],[14]
Digital Signature	Ed25519	Ownership verification through watermark authentication	Deterministic signature scheme with compact 32-byte keys [20]
Model Integrity	SHA-256	Model hashing and tamper detection	No practical collision attacks known; NIST-compliant standard [21]
Key Derivation	HKDF-SHA256	Derivation of Key Encryption Keys (KEKs) from KEM shared secrets	Standardized in RFC 5869 [12]
Private Key Protection	scrypt + AES-256-GCM	Passphrase-based encryption of KEM private keys	Memory-hard design resistant to GPU-assisted brute-force attacks [22]
This Study	AES-256-GCM + Kyber/ML-KEM-768 + Ed25519 + scrypt	Multi-layer AI model distribution protection	Hybrid post-quantum framework operating at file-distribution level; evaluated on .h5, .pt, .pth, .pkl, .joblib formats

AES-256-GCM was selected as the primary encryption mechanism because it provides authenticated encryption, combining confidentiality and integrity protection within a single cryptographic operation [17]. In a comparative study involving 5,000 PDF documents, Patria and Andriati [17] reported that AES-GCM consistently outperformed ChaCha20-Poly1305 across multiple performance metrics. Susanti et al. [18] further observed that AES-GCM offers stronger resistance against timing-based attacks. Ilham et al. [23] found that the throughput difference between AES-128-GCM and AES-256-GCM was approximately 12.5%, indicating that the stronger 256-bit configuration can be adopted with only a modest performance overhead. Amirkhanova et al. [24] demonstrated that AES-256-GCM can be efficiently deployed on commodity hardware while maintaining a success rate of 99.72% in cryptographic operations. Additional studies by Yahu et al. [25] and Nasrullah [26] confirmed the suitability of AES-based encryption for secure communication channels and web-based systems, respectively.

Kyber/ML-KEM-768 was selected as the post-quantum Key Encapsulation Mechanism because it has been standardized by NIST under FIPS 203 and provides a security level comparable to AES-192 [12], [13]. Within SentinelKEM, Kyber is employed in a hybrid key-protection workflow. The encapsulation process generates a shared secret and a KEM ciphertext using the public key. The shared secret is

subsequently processed through HKDF-SHA256 with the application-specific context string "sentinelkem-aes-key-wrap-v8" to derive a 256-bit Key Encryption Key (KEK). This KEK is then used to protect the AES encryption key through AES-256-GCM. Such a design follows established recommendations for hybrid cryptographic architectures that combine post-quantum and symmetric cryptography [14]. Zhang et al. [12] reported that Kyber768 can achieve throughput exceeding 550 Mbps with an average latency of 3.14 ms, while Nagy et al. [19] confirmed that ML-KEM satisfies the performance requirements of production environments. Ramakrishna and Shaik [13] also highlighted the practicality of combining Kyber-based key encapsulation with AES encryption to provide efficient and secure data protection.

3. Method

To achieve secure and modular post-quantum file protection, SentinelKEM was designed using a layered architecture that integrates multiple cryptographic primitives and supporting infrastructure. The overall system architecture and workflow are presented in the following subsection.

3.1. System overview and workflow

SentinelKEM was implemented in Python 3 using well-established cryptographic libraries and a modular architecture, where each cryptographic function is encapsulated in a dedicated module under the core/ directory to improve maintainability, simplify auditing, and enable independent updates. Figure 1 presents the layered architecture consisting of the User Interface, Cryptographic, Infrastructure, and Secure Distribution layers. The User Interface supports Encryption and Decryption, which are processed by the Cryptographic Layer using AES-256-GCM for payload confidentiality, Kyber/ML-KEM-768 for post-quantum key protection, Ed25519 for ownership signatures, and SHA-256 for integrity hashing. The Infrastructure Layer packages encrypted artifacts into a secure ZIP archive, records JSONL audit logs, and removes temporary files. The Secure Distribution stage delivers the package together with the separately stored private key. This layered design allows individual cryptographic primitives (e.g., replacing SHA-256) without modifying other layers. Figure 2 illustrates the package contents, including the encrypted AI model, wrapped AES key, ownership watermark, Ed25519 signature, and separately stored private key, reflecting the privilege-separation design described in Section 3.4.

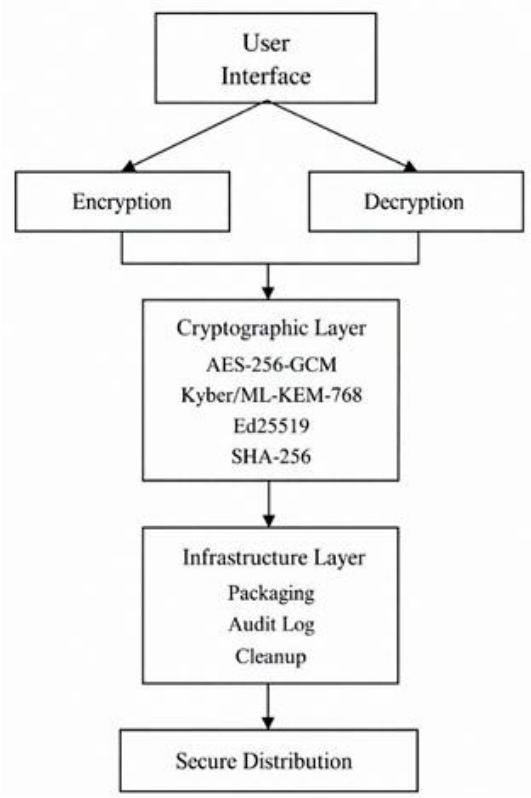

Figure 1. Modular architecture of SentinelKEM

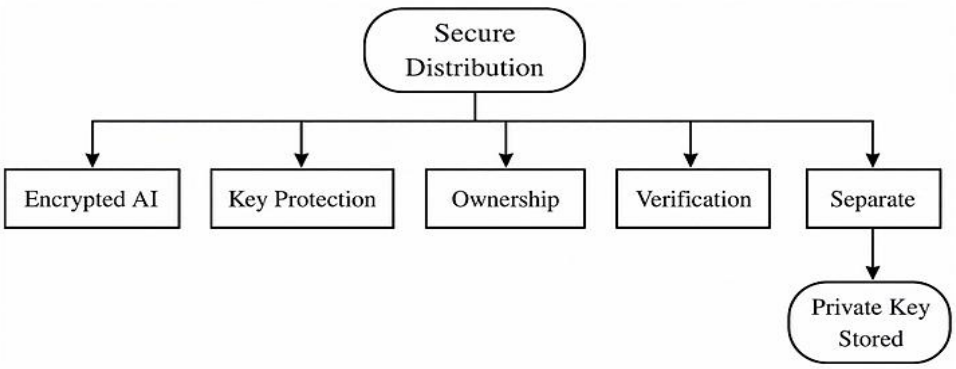

Figure 2. Structure of the secure distribution package and Its cryptographic artifacts

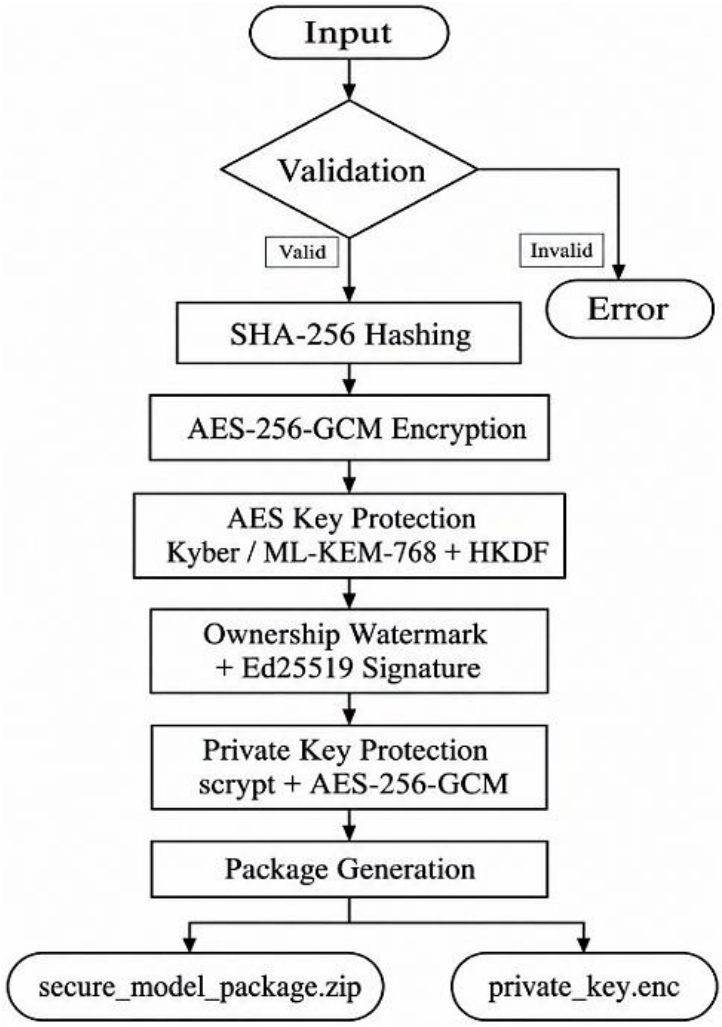

Figure 3. SentinelKEM encryption workflow

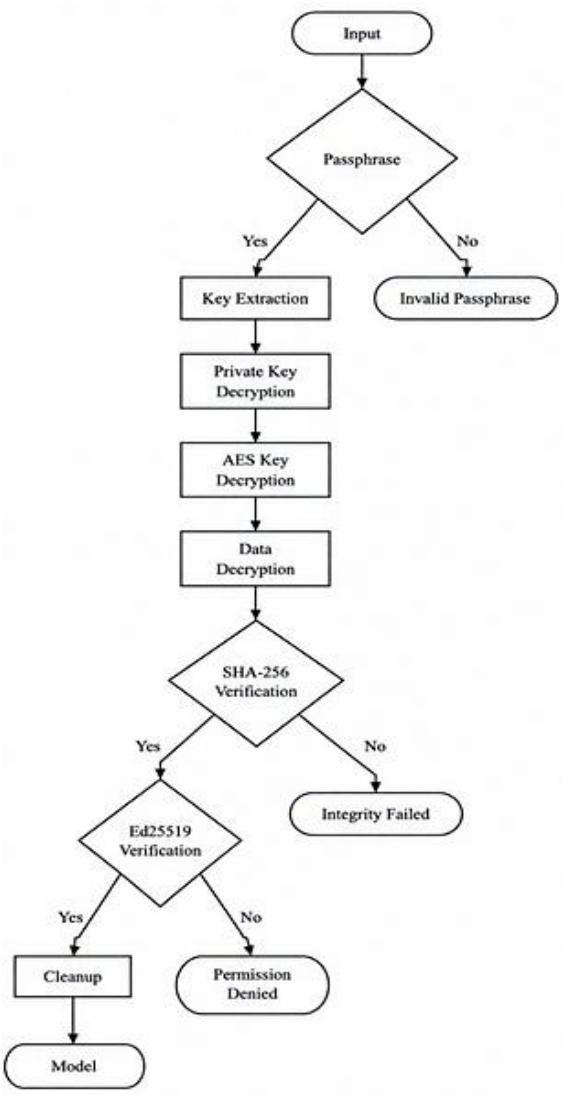

Figure 4. SentinelKEM decryption and verification workflow

Figure 3 illustrates the SentinelKEM encryption workflow. The process begins with input validation, followed by SHA-256 hashing to generate an integrity fingerprint. A random 256-bit AES key is then generated, and the model is encrypted using AES-256-GCM with a unique 12-byte nonce, while the nonce and authentication tag are stored in metadata.json. Next, a Kyber/ML-KEM-768 key pair encapsulates a shared secret, which is processed using HKDF-SHA256 to derive a Key Encryption Key (KEK) for wrapping the AES key. An ownership watermark is created and digitally signed with Ed25519. The KEM private key is protected using a passphrase-derived key generated by scrypt and encrypted with AES-256-GCM. Finally, the encrypted model, wrapped AES key, public keys, watermark, and metadata are packaged into a secure ZIP archive, while the encrypted private key is stored separately as private_key.enc. Figure 4 presents the decryption workflow. Users provide the encrypted package and passphrase; invalid passphrases immediately terminate the process. After restoring the encrypted private key, the system recovers the AES key and decrypts the model. SHA-256 verifies model integrity, followed by Ed25519 signature verification for ownership authentication. If both checks succeed, the model is released to the authorized recipient and all temporary files are automatically removed to reduce residual security risks.

3.2. Dataset

This study evaluates only the file-level cryptographic security of SentinelKEM and does not assess model accuracy, training performance, or other machine-learning properties. No training dataset is used; instead, AI model files serve as test objects to evaluate encryption, decryption, integrity verification, and ownership authentication. Five commonly used model formats from different AI

ecosystems were selected: TensorFlow/Keras (.h5), PyTorch (.pt and .pth), and Scikit-learn (.pkl and .joblib). All models were processed using the same cryptographic configuration to evaluate format compatibility, encrypted package size, and the correctness of decryption and verification. As shown in Figure 5, a pretrained ResNet18 model in .pth format from the official PyTorch repository increased from 44.66 MB to 44.68 MB after encryption, demonstrating compatibility with PyTorch weight files. In Figure 6, the TorchScript .pt version increased from 44.75 MB to 44.77 MB, confirming compatibility with production deployment models. Figure 7 shows a Scikit-learn Random Forest Classifier serialized as .pkl, which increased from 170.82 KB to 174.89 KB, representing conventional machine-learning classification models.

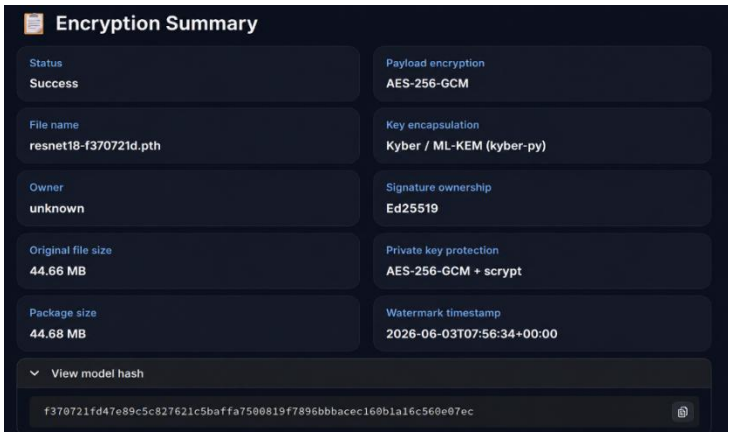

Figure 5. Encryption result of the ResNet18 model (.pth)

The .joblib file originates from the same Random Forest model but is stored using the Joblib serialization library based on Figure 8, which is widely adopted for Scikit-learn models. The original file size is 186.14 KB, while the encrypted package generated by SentinelKEM is 190.22 KB. This experiment was intended to evaluate compatibility with an alternative serialization mechanism frequently used in machine learning workflows. As the primary case study in this research in Figure 9, a TensorFlow/Keras model stored in the .h5 format was employed. This format was selected because it remains one of the most widely used methods for storing deep learning models in both research and industrial applications. The model was used to evaluate the capability of SentinelKEM to protect large model files through the integration of AES-256-GCM, ML-KEM (Kyber), Ed25519, and scrypt. The results demonstrate that SentinelKEM successfully processed all tested model formats regardless of the underlying framework or internal model structure as in Table 2. This observation indicates that the proposed approach operates at the file level rather than the model-parameter level, allowing the framework to be applied to a wide range of AI and machine learning models without requiring modifications to their architecture or learned parameters.

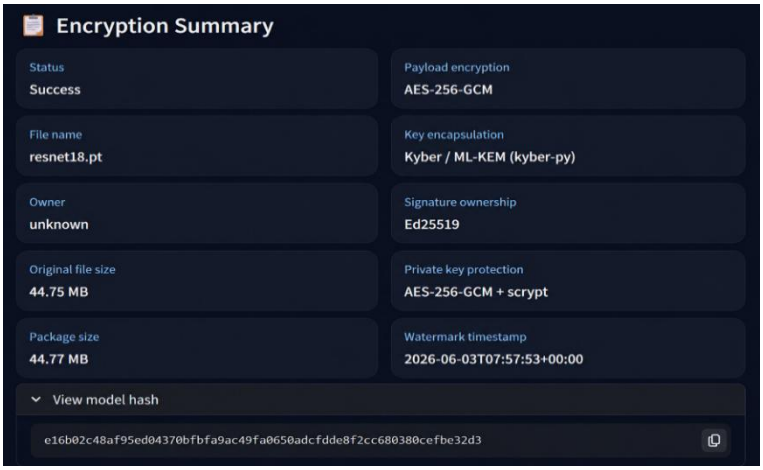

Figure 6. Encryption result of the ResNet18 TorchScript model (.pt)

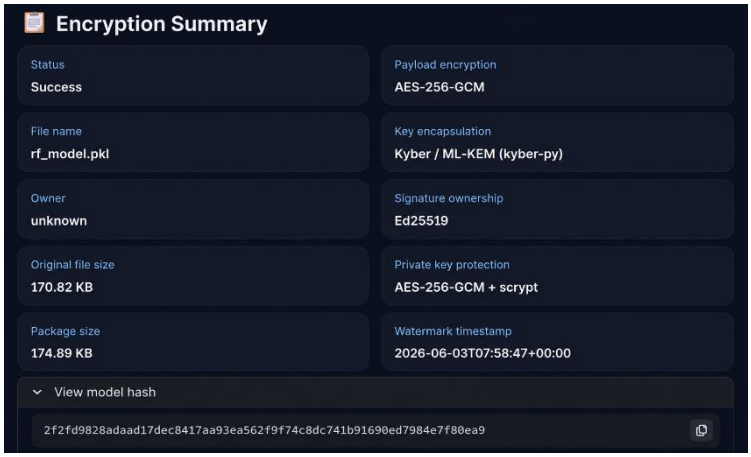

Figure 7. Encryption result of the random forest model (.pkl)

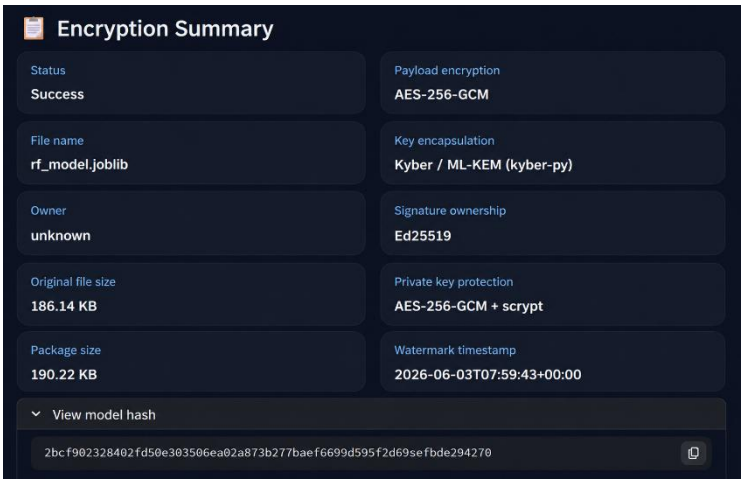

Figure 8. Encryption result of the random forest model (.joblib)

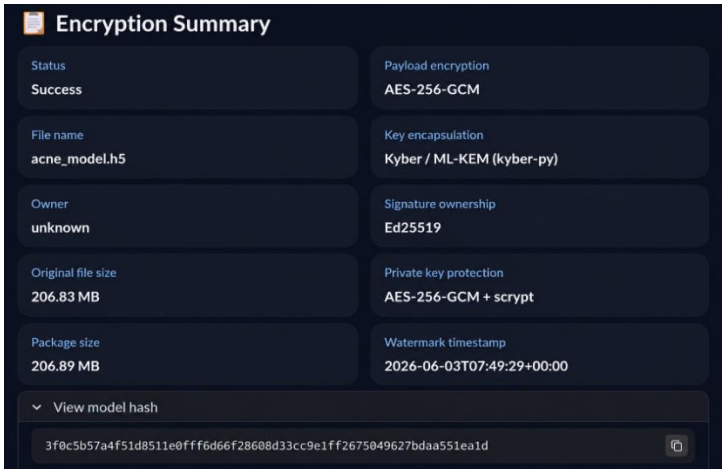

Figure 9. Encryption result of the tensorFlow/keras model (.h5)

Table 2. Test objects used in sentinelKEM evaluation

Format	Framework	Model	Original Size	Package Size
.h5	TensorFlow/Keras	AI Model	206.83 MB	206.89 MB
.pth	PyTorch	ResNet18 Pretrained Weights	44.66 MB	44.68 MB
.pt	PyTorch	ResNet18 TorchScript	44.75 MB	44.77 MB
.pkl	Scikit-learn	Random Forest Classifier	170.82 KB	174.89 KB
.joblib	Scikit-learn	Random Forest Classifier	186.14 KB	190.22 KB

3.3. Proposed scheme

SentinelKEM was implemented in Python 3 using widely adopted and well-established cryptographic libraries. The implementation follows a modular architecture in which each cryptographic function is encapsulated within a dedicated module located under the core/ directory. This design improves maintainability, simplifies auditing, and allows individual cryptographic components to be updated independently. Table 3 summarizes the libraries and dependencies used throughout the system.

Table 3. Cryptographic libraries and system dependencies

Module	Library	Version	Function
AES-256-GCM	cryptography	≥ 41.0	Model encryption/decryption and AES key wrapping
scrypt	cryptography	≥ 41.0	Private key protection KDF ($N=2^{14}$, r=8, p=1)
Kyber/ML-KEM	kyber-py	FIPS 203	KEM key generation, encapsulation, and decapsulation
Ed25519, HKDF	cryptography	≥ 41.0	Key generation, signing, verification, and KEK derivation
SHA-256	hashlib	Standard Library	Incremental model hashing using 64 KB blocks
Audit Logging	audit.py + json	Standard Library	JSONL-based operation logging [27]
User Interface	streamlit	≥ 1.28	Interactive web application

The encryption process is implemented in encryptor.py and consists of seven sequential stages. First, the original model file is hashed incrementally using SHA-256 with a block size of 64 KB (CHUNK_SIZE = 65,536 bytes), producing a 256-bit integrity fingerprint. Second, a random 256-bit AES key is generated using os.urandom(32), which relies on the operating system's entropy source. Third, the model file is encrypted using AES-256-GCM with a single 12-byte nonce generated by os.urandom(12). The file is processed in 64 KB chunks to reduce memory consumption. The encrypted output is stored as encrypted_model.enc, while the nonce and 16-byte authentication tag are recorded in metadata.json. Fourth, a Kyber/ML-KEM-768 key pair is generated. The encapsulation procedure produces a shared secret and a KEM ciphertext, after which HKDF-SHA256 derives a 32-byte Key Encryption Key (KEK). The resulting KEK is then used to protect the AES encryption key through AES-256-GCM. Fifth, an ownership watermark is generated in JSON format containing the owner name, original filename, model hash, ISO 8601 timestamp, cryptographic metadata, system identifier (SENTINELKEM-AI-MODEL-PROTECT-V8), and ownership claim. This watermark is digitally signed using Ed25519, producing watermark.sig and owner_verify_key.pub. Sixth, the KEM private key is protected using scrypt with a randomly generated 16-byte salt (N=2^14, r=8, p=1) and encrypted with AES-256-GCM to produce private_key.enc. Finally, the generated artifacts are assembled into a secure ZIP distribution package, while private_key.enc is retained separately by the model owner.

The decryption workflow is implemented in decryptor.py and performs model recovery together with two sequential cryptographic verification stages. Integrity verification is performed first by comparing the computed SHA-256 hash of the recovered model against the reference value stored in the package. If a mismatch is detected, the process is terminated immediately and ownership verification is not performed. A matching hash confirms that the recovered file remains identical to the original model [21]. After successful integrity verification, the system validates the Ed25519 digital signature contained in watermark.sig using the corresponding public verification key. Any signature verification failure results in immediate termination of the recovery process [9], [20]. When both verification stages succeed, the recovered model becomes available to the authorized recipient, and all temporary files generated during processing are automatically removed by cleanup.py.

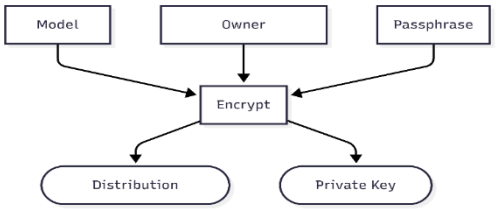

Figure 10. SentinelKEM encryption interface

Figure 10 shows the encryption view of the Streamlit interface. The user supplies three inputs - the Model file, the Owner name, and a Passphrase - which the interface passes to encryptor.py. On completion, two downloadable artifacts are produced side by side: the Distribution package (the ZIP archive containing the encrypted model and its cryptographic metadata) and the Private Key file, which, per the privilege-separation design in Section 3.4, is never bundled inside the distribution package itself. As illustrated in Figure 11, the file private_key.enc is consistently generated as a separate artifact rather than being included within the distribution package, thereby enforcing the principle of privilege separation. Figure 11 further shows that users must supply the distribution package, private_key.enc, and the correct passphrase before the system performs integrity verification (SHA-256) and ownership verification (Ed25519) and subsequently provides access to the recovered model.

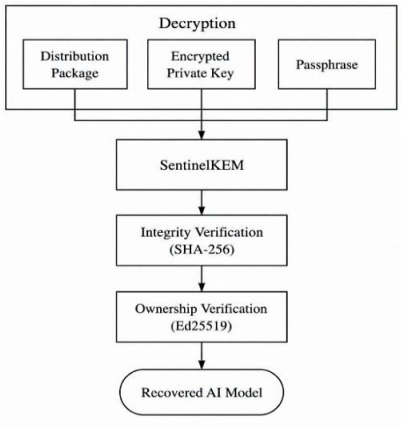

Figure 11. SentinelKEM decryption interface

3.4. Privacy by design

SentinelKEM incorporates privacy-by-design principles through two security mechanisms embedded directly within the implementation. The first mechanism is automatic temporary-file removal. Within the finally blocks of both encryptor.py and decryptor.py, the cleanup.py module invokes remove_path(work_dir) to remove all temporary files stored in the working directory after each operation, regardless of its outcome. By design, this mechanism is intended to ensure that plaintext model files are not left behind on the server after processing; however, this behavior has been verified only through code inspection and functional testing of the cleanup routine, not through an independent runtime audit or forensic disk analysis, and this is noted as a limitation in Section 4.5. The second mechanism is the separation of the private key from the distribution package. The file private_key.enc is not included in the secure distribution package and is instead returned separately to the model owner. As a result, possession of the distribution package alone is insufficient to recover the protected model. Successful decryption additionally requires both the encrypted private key and the correct passphrase. This design follows the principle of privilege separation discussed by Kanakri and King [9] in the context of AI model intellectual-property protection.

3.5. Evaluation scenarios

The evaluation was conducted using model files in .h5, .pt, .pth, .pkl, and .joblib formats to validate cross-framework compatibility and the effectiveness of the proposed protection mechanism. The .h5 format was selected as the primary demonstration case because it remains one of the most widely

adopted storage formats for TensorFlow/Keras models. Four evaluation scenarios were designed to assess the functionality and security properties of SentinelKEM.

1. Functional Encryption Evaluation – Verifies the completeness of the generated distribution package, including the presence of all required cryptographic artifacts, package size, cryptographic overhead, and the separate generation of private_key.enc.
2. Decryption and Integrity Verification Evaluation – Assesses successful model recovery and verifies that the SHA-256 hash of the recovered model matches the reference value stored within the package.
3. Ownership Verification Evaluation – Evaluates Ed25519 signature verification under two conditions: verification using the correct public key and verification using a manipulated or invalid public key.
4. Tamper-Resistance Evaluation – Examines the system's response to three attack scenarios: modification of a single byte within the ciphertext, alteration of watermark.json without regenerating the corresponding signature, and decryption attempts using an incorrect passphrase.

4. Results and Discussion

The results and discussion section evaluates the effectiveness of SentinelKEM in securing machine learning models. The experiments examine the functionality of the proposed encryption scheme, the generated cryptographic artifacts, and the system performance across various model formats.

4.1 Functional encryption results

For all tested model formats (.h5, .pt, .pth, .pkl, and .joblib), the encryption process successfully generated a complete secure distribution package containing all required cryptographic artifacts. Table 4 presents detailed results obtained from .h5 models of different sizes, which represent the primary use case of the proposed framework.

Table 4. Encryption results for .h5 models: artifacts and cryptographic overhead

.h5 Model Size	encrypted_model.encnc	ZIP Overhead	ZIP Size	private_key.enc	Artifacts
10 MB (10.485.760 B)	10.485.760 B (100%)	4.569 B (~4,5 KB)	~10,00 MB	~4,9 KB	7/7 ✓
50 MB (52.428.800 B)	52.428.800 B (100%)	4.569 B (~4,5 KB)	~50,00 MB	~4,9 KB	7/7 ✓
100 MB (104.857.600 B)	104.857.600 B (100%)	4.569 B (~4,5 KB)	~100,0 0 MB	~4,9 KB	7/7 ✓
500 MB (524.288.000 B)	524.288.000 B (100%)	4.569 B (~4,5 KB)	~500,0 0 MB	~4,9 KB	7/7 ✓

Table 5. Cryptographic artifacts contained in the distribution package

rtifact	Size	Description
encrypted_model.enc	Equal to original model size	AES-256-GCM encrypted model
encrypted_aes_key.key	~2,449 bytes	JSON structure containing KEM ciphertext, nonce, and wrapped AES key
public_key.key	1,184 bytes	Kyber-768 public key
watermark.json	~553 bytes	Ownership metadata and cryptographic information
watermark.sig	64 bytes	Ed25519 digital signature
owner_verify_key.pub	32 bytes	Ed25519 public verification key
metadata.json	~287 bytes	AES-GCM nonce, authentication tag, filename, and job information
Total Package Overhead	~4,569 bytes	Constant overhead independent of model size
private_key.enc (separate file)	~4,989 bytes	Encrypted Kyber private key protected by scrypt and AES-256-GCM

As shown in Table 4, the size of encrypted_model.enc remains identical to the size of the original plaintext model across all tested file sizes. This behavior is expected because AES-GCM employs counter-mode encryption, where each plaintext block is transformed into a ciphertext block of equal length, resulting in no expansion of the encrypted model file [17], [24]. The cryptographic overhead introduced by SentinelKEM remains constant at approximately 4,569 bytes regardless of model as described in Table 4. Consequently, the relative overhead decreases as the model size increases, reaching approximately 0.044% for a 10 MB model and only 0.001% for a 500 MB model. Similarly, the size of private_key.enc remains approximately 4.9 KB in all experiments because it depends solely on the encrypted Kyber private key and associated metadata rather than the size of the protected model. These results indicate that the proposed framework introduces only a negligible storage overhead while maintaining a complete set of cryptographic protection mechanisms

4.2 Decryption and integrity verification results

All decryption experiments successfully restored the original model when valid credentials and unmodified package contents were provided. Integrity validation was performed by recomputing the SHA-256 hash of the recovered model and comparing it with the reference hash stored during the encryption stage. In every valid test case, the generated hash matched the stored value exactly, confirming that no modifications occurred during encryption, storage, or decryption. These findings are consistent with the observations reported by Habbal et al. [21], who highlighted the strong integrity-verification properties of SHA-256 and the absence of practical collision attacks. An important implementation detail is that integrity verification is always performed before ownership verification. Therefore, when integrity validation fails, the recovery process is terminated immediately and no ownership information is disclosed as shown in Table 6.

Table 6. Decryption and integrity verification desults for .h5 models

Test Condition	Decryption Result	SHA-256 Hash Comparison	Integrity Status	Ownership Status
Correct passphrase, valid package	Model recovered successfully	Identical	✓ Valid	✓ Valid
Incorrect passphrase	Recovery failed	Not evaluated	—	—
Modified encrypted_model.enc	Rejected by AES-GCM	Not evaluated	✗ Detected	—
Modified watermark.json	Model recovered	Identical	✓ Valid	✗ Detected
Manipulated public key	Model recovered	Identical	✓ Valid	✗ Detected

4.3 Ownership verification results

Ownership verification was evaluated using both valid and intentionally manipulated verification keys. Under normal conditions, all Ed25519 signatures were verified successfully. The verification procedure relies on a canonical JSON representation generated through key ordering to ensure consistency between the signing and verification stages. When a modified public key was supplied, the signature verification process failed and the system rejected the ownership claim. These results are consistent with the security properties of Ed25519 reported by Wahyuni et al. [20], including its efficiency, reliability, and ease of deployment. Unlike symmetric ownership-verification approaches, public-key-based verification allows ownership claims to be independently validated by any party while ensuring that only the holder of the corresponding private key can generate a valid signature.

4.4 Tamper-resistance results

The tamper-resistance evaluation examined three attack scenarios: ciphertext modification, watermark manipulation, and decryption attempts using an incorrect passphrase. In all cases, SentinelKEM successfully detected the attack and prevented unauthorized model recovery. Modifying a single byte within encrypted_model.enc caused AES-GCM authentication-tag verification to fail, resulting in immediate rejection of the ciphertext. This behavior reflects the authenticated-encryption properties of AES-GCM, where even minor modifications invalidate the authentication tag [17], [24]. Similarly, altering the contents of watermark.json without generating a new signature caused ownership verification to fail because the signed data no longer matched the original watermark

content. Finally, the use of an incorrect passphrase prevented successful recovery of the encrypted private key, thereby stopping the decryption process before model recovery could occur as shown Table 7. Within these three tested attack scenarios, these results demonstrate that SentinelKEM provides layered protection against unauthorized modification and access attempts; broader classes of attack (e.g., side-channel or multi-step adversarial tampering) were not evaluated and should not be inferred from this result.

Table 7. Summary of functional and security evaluation results

Evaluation Scenario	Primary Metric	Result	Time (s)	Status
Functional Encryption (10–500 MB)	Completeness of ZIP artifacts	7/7 artifacts generated	Depends on file size	✓ Valid
Decryption and Integrity Verification	SHA-256 hash consistency	Identical hash values	Depends on file size	✓ Valid
Ownership Verification (valid key)	Ed25519 signature verification	Ownership confirmed	< 0.01	✓ Valid
Ownership Verification (manipulated key)	Detection of invalid verification key	Verification rejected	< 0.01	✓ Detected
Ciphertext Manipulation (1-byte modification)	AES-GCM tag validation	Decryption rejected	< 0.01	✓ Detected
Watermark Manipulation	Signature verification	Verification rejected	< 0.01	✓ Detected
Incorrect Passphrase	Private key recovery	Decryption not performed	< 0.01	✓ Detected

4.5 System limitations

Several limitations of the current implementation should be acknowledged. First, a comprehensive quantitative performance evaluation, including throughput and latency measurements across all model sizes, has not yet been conducted through repeated benchmarking experiments. The execution times reported in Table 7 primarily reflect lightweight cryptographic operations such as signature verification and key recovery rather than full-file encryption and decryption, which depend on file size and storage performance. Second, the framework currently assumes a single recipient per distribution package and does not implement key-management mechanisms for multi-recipient distribution scenarios. Supporting multiple recipients would require either re-encryption or an extended key-sharing architecture. Third, resistance against physical side-channel attacks has not been evaluated. As noted by Ramakrishna and Shaik [13], secure deployment of post-quantum cryptographic systems may require additional protections against implementation-level attacks, particularly in embedded or resource-constrained hardware environments. Fourth, the claim that temporary plaintext files are removed after processing (Section 3.4) is based on code-level design and functional testing of the cleanup routine, not on an independent runtime or forensic audit of the server's file system; such an audit is recommended before the framework is relied upon in production deployments.

5. Conclusion

This study presented SentinelKEM, a hybrid cryptographic framework designed to protect the distribution of AI models through the integration of AES-256-GCM, Kyber/ML-KEM-768, Ed25519, and scrypt. Unlike conventional watermarking approaches that operate after a model becomes accessible to a recipient, SentinelKEM provides protection at the distribution stage by securing the model file before access is granted. This approach complements existing watermarking techniques by extending protection to the file-distribution layer. The evaluation results demonstrated that all tested model formats were successfully encrypted and decrypted while maintaining correct SHA-256 integrity verification. The encrypted model file preserved the same size as the original plaintext model, and the cryptographic overhead introduced by the distribution package remained approximately 4.5 KB regardless of model size. Integrity verification and ownership verification were performed automatically during the recovery process, ensuring that only authentic and unmodified models could be released to authorized users. The tamper-resistance evaluation further showed that SentinelKEM consistently detected ciphertext modification, watermark manipulation, and decryption attempts using

an incorrect passphrase within the three tested attack scenarios. These findings indicate that the proposed framework provides multiple layers of protection against unauthorized access and modification for the scenarios evaluated. In addition, the adoption of Kyber/ML-KEM-768, standardized by NIST under FIPS 203, enables the framework to incorporate post-quantum key protection within the model-distribution workflow. The main contributions of this work are fourfold. First, it introduces a multi-layer cryptographic framework for AI model distribution that combines post-quantum key encapsulation, authenticated encryption, digital signatures, and passphrase-based private-key protection within a single system. Second, it implements privacy-by-design principles through automatic temporary-file removal and the separation of private-key material from the distribution package. Third, it provides an automated verification workflow that combines integrity and ownership validation during model recovery. Fourth, it demonstrates a practical implementation through a Streamlit-based application that can be used without requiring advanced cryptographic expertise.

Based on the limitations identified in Section 4.5, three concrete gaps remain open for future research. (1) Performance characterization gap - no repeated, statistically controlled throughput/latency benchmarking exists across the full range of model sizes and hardware environments; future work should conduct such benchmarking to characterize real-world deployment cost. (2) Multi-recipient key-management gap - the current single-recipient design cannot scale to collaborative or multi-party model distribution without re-encryption; future work should design and evaluate a scalable key-sharing or re-encryption architecture. (3) Implementation-level security assurance gap - resistance to physical side-channel attacks and the no-plaintext-retained claim have not been verified by an independent runtime or forensic audit; future work should include formal side-channel testing and third-party audit of the cleanup mechanism before production deployment. Addressing these three gaps would strengthen the practical readiness of SentinelKEM for real-world AI model distribution.

Credit Authorship Contribution Statement

Singgih Naufal : Conceptualization, Methodology, Software, Project administration, Editing. **Eko Hari Rachmawanto :** Project administration, Editing, Evaluation, Validation, Supervision. **Mohamed Doheir** : Validation, Supervision.

Declaration of Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data Availability

Data will be made available on request.

Use of Artificial Intelligence

Authors are requested to identify any parts or aspects of their manuscript that benefited from artificial intelligence technology.

References

[1] F. Regazzoni, P. Palmieri, F. Smailbegovic, R. Cammarota, and I. Polian, “Protecting artificial intelligence IPs: a survey of watermarking and fingerprinting for machine learning,” Jun. 01, 2021, *John Wiley and Sons Inc*. doi: 10.1049/cit2.12029

[2] X. Liu and R. Xu, “A comprehensive survey of watermarking techniques for copyright protection and integrity verification on DNNs and generative models,” May 01, 2026, *Springer Nature*. doi: 10.1007/s42452-026-08576-3

[3] A. Fkirin, A. S. Moursi, G. Attiya, A. El-Sayed, and M. A. Shouman, “Hybrid two-level protection system for preserving pre-trained DNN models ownership,” *Neural Comput. Appl.*, vol. 36, no. 34, pp. 21415–21449, Dec. 2024, doi: 10.1007/s00521-024-10304-0

[4] A. Fkirin, G. Attiya, A. El-Sayed, and M. A. Shouman, “Copyright protection of deep neural network models using digital watermarking: a comparative study,” *Multimed. Tools Appl.*, vol. 81, no. 11, pp. 15961–15975, May 2022, doi: 10.1007/s11042-022-12566-z
[5] J. Liao, L. Yi, W. Shi, W. Yang, Y. Fang, and X. Yang, “Imperceptible backdoor watermarks for speech recognition model copyright protection,” *Visual Intelligence*, vol. 2, no. 1, Dec. 2024, doi: 10.1007/s44267-024-00055-w
[6] X. Wang, Y. Lu, X. Yan, and L. Yu, “Wet Paper Coding-Based Deep Neural Network Watermarking,” *Sensors*, vol. 22, no. 9, May 2022, doi: 10.3390/s22093489
[7] Y. Liao, R. Jiang, and B. Zhou, “Dynamic Black-Box Model Watermarking for Heterogeneous Federated Learning,” *Electronics (Switzerland)*, vol. 13, no. 21, Nov. 2024, doi: 10.3390/electronics13214306
[8] T. El Hajjar, M. Lansari, R. Bellafqira, G. Coatrieux, K. Kapusta, and K. Kallas, “RoSe-Mix: Robust and Secure Deep Neural Network Watermarking in Black-Box Settings via Image Mixup,” *Mach. Learn. Knowl. Extr.*, vol. 7, no. 2, Jun. 2025, doi: 10.3390/make7020032
[9] W. Kanakri and B. King, “PK-Judge: Enhancing IP Protection of Neural Network Models Using an Asymmetric Approach,” *Big Data and Cognitive Computing*, vol. 9, no. 3, Mar. 2025, doi: 10.3390/bdcc9030066
[10] M. Li, Z. Wang, and X. Zhang, “An Effective Framework for Intellectual Property Protection of NLG Models,” *Symmetry (Basel).*, vol. 15, no. 6, Jun. 2023, doi: 10.3390/sym15061287
[11] Q. Li, G. Xu, and X. Qi, “FingerMarks: Robust Multi-Bit Watermarking for Remote Deep Neural Networks,” *Electronics (Switzerland)*, vol. 14, no. 24, Dec. 2025, doi: 10.3390/electronics14244818
[12] K. Zhang, M. Yang, Z. Yuan, Y. Zhang, and W. Liu, “Optimized Quantum-Resistant Cryptosystem: Integrating Kyber-KEM with Hardware TRNG on Zynq Platform,” *Electronics (Switzerland)*, vol. 14, no. 13, Jul. 2025, doi: 10.3390/electronics14132591
[13] D. Ramakrishna and M. A. Shaik, “PSESV: A hybrid post-quantum encryption Framework with real-time thermal and EM side-channel attack detection,” *Ain Shams Engineering Journal*, vol. 16, no. 12, Dec. 2025, doi: 10.1016/j.asej.2025.103776
[14] A. Olushola and S. P. Meenakshi, “Design and implementation of an authenticated post-quantum session protocol using ML-KEM (Kyber), ML-DSA (Dilithium), and AES-256-GCM,” *Front. Phys.*, vol. 13, Jan. 2026, doi: 10.3389/fphy.2025.1723966
[15] M. A. Ehsan, W. Alayed, A. U. Rehman, W. ul Hassan, and A. Zeeshan, “Post-Quantum KEMs for IoT: A Study of Kyber and NTRU,” *Symmetry (Basel).*, vol. 17, no. 6, Jun. 2025, doi: 10.3390/sym17060881
[16] T. Ngwenya and B. Ndlovu, “A Systematic Review of Post-Quantum Cryptography for Healthcare Data Protection: Performance, Readiness, and Deployment Challenges,” 2026. [Online]. Available: http://jurnal.polibatam.ac.id/index.php/JAIC
[17] M. Patria and D. A. Andriati, “Analisis Komparatif Performa AES-GCM dan ChaCha20-Poly1305 dalam Enkripsi Dokumen PDF Berbasis AEAD,” *Arcitech: Journal of Computer Science and Artificial Intelligence*, vol. 5, no. 1, pp. 49–69, Jun. 2025, doi: 10.29240/arcitech.v5i1.13645
[18] A. Susanti, B. A. Prasetiya, O. D. Pangesti, L. D. Suryawati, and I. A. Saputro, “Perbandingan Kinerja dan Keamanan Algoritma Kriptografi Modern AES-GCM dengan CHACHA20-POLY1305,” *Infomatek*, vol. 26, no. 2, pp. 253–264, Dec. 2024, doi: 10.23969/infomatek.v26i2.19255
[19] N. Nagy *et al.*, “Module-Lattice-Based Key-Encapsulation Mechanism Performance Measurements,” Sep. 01, 2025, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi: 10.3390/sci7030091
[20] A. P. Wahyuni, A. Bramantoro, R. Rizal, and S. Elmeftahi, “Digitograph: A Mobile Digital Signatures Application for PDF file Using ED25519 and Asymmetric Encryption,” 2025.
[21] A. Habbal, “Comparative Analysis of MD5 and SHA-256 Hash Algorithms for Fingerprint File Integrity Verification,” 2026. [Online]. Available: http://jurnal.polibatam.ac.id/index.php/JAIC
[22] S. J. Choi, D. C. Kim, and S. C. Seo, “Parallel Implementation of Scrypt: A Study on GPU Acceleration for Password-Based Key Derivation Function,” *Journal of Information and Communication Convergence Engineering*, vol. 22, no. 2, pp. 98–108, 2024, doi: 10.56977/jicce.2024.22.2.98
[23] Muhammad Arifin Ilham, Dody Herdiana, M.Agreindra Helmiawan, and Asep Saeppani, “Analisi Perbandingan Kinerja Algoritma AES-128 GCM dan AES-256 GCM Pada Protokol TLS 1.2 Di Server Ngnix,” *Router : Jurnal Teknik Informatika dan Terapan*, vol. 3, no. 4, pp. 87–95, Dec. 2025, doi: 10.62951/router.v3i4.743

[24] G. Amirkhanova, S. Ismailov, A. Amirkhanov, S. Adilzhanova, M. Zhasuzakova, and S. Chen, “A Lightweight, End-to-End Encrypted Data Pipeline for IIoT: An AES-GCM Implementation for ESP32, MQTT, and Raspberry Pi,” *Information*, vol. 17, no. 1, p. 33, Jan. 2026, doi: 10.3390/info17010033
[25] M. Neta Yahu, D. Marvelino Septian, and E. Daud, “Simulation of End-to-End Secure Channels in Digital Payment Systems Using TLS 1.3 Combined with Client-Side AES-GCM Encryption,” *Install: Information System and Technology Journal*, vol. 2, no. 3, pp. 40–47, 2025, doi: 10.33859/install.v2i3.1002
[26] A. H. Nasrullah, “Secure Web-Based File Encryption Using AES-128,” *Journal of Embedded Systems, Security and Intelligent Systems*, pp. 146–155, Jun. 2025, doi: 10.59562/jessi.v6i2.8436
[27] B. Mario and T. Wiradinata, “Enhancing Web Security and Performance with Hybrid Stateless Authentication,” 2025. [Online]. Available: http://jurnal.polibatam.ac.id/index.php/JAIC