CoreID JOURNAL

Vol. 4 No. 1, March 2026 (22-31)
ISSN: 2987-6990
DOI: https://doi.org/10.60005/coreid.v4i1.150

Multi-Layer PHP Source Code Protection Model Using Encoding with Integrity Hashing and Anti-Reverse Engineering

Sarmuni[1], Suhaeri Rumdani[2], Anna Rosdiana[3], Khaeroni[4], Ahmad Faturohman[5], Devid Sujianto[6], Deden Rifki[7]
[1,2,3,4,5,6,7]Center for Information Technology and Data Management, Universitas Islam Negeri Sultan Maulana Hasanuddin Banten

Article Info

Article history:

Received March 08, 2026
Revised April 06, 2026
Accepted April 08, 2026

Keywords:

AES-256 encryption
Anti-tamper
Auto-restore mechanism
Multi-layer encoding
PHP source code protection
SHA-256 hashing

ABSTRACT

This study proposes a unified multi-layer PHP source code protection architecture that integrates obfuscation, layered encoding, AES-256 encryption, SHA-256 integrity verification, auto-restore, and Telegram-based security alerts. Unlike prior PHP protection approaches that generally focus solely on code concealment or encryption, the proposed model integrates protection, tamper detection, and incident response into a single workflow. The system was evaluated using a research-and-development approach, including prototype implementation, security testing, and performance benchmarking on PHP datasets of varying complexity. The proposed method was compared with conventional protection mechanisms and commercial PHP encoders to assess resistance to deobfuscation, integrity-checking capability, and runtime overhead. The results indicate that the proposed architecture improves code readability, reliably detects unauthorized file modifications, and maintains acceptable performance, with an average execution-time overhead of 15–20 ms, a CPU load increase of 8–12%, and a memory usage increase of 5–8 MB. These findings suggest that the model provides a practical balance between stronger source-code protection and operational efficiency for PHP-based web applications.

Corresponding Author:

Khaeroni
Center for Information Technology and Data Management, Universitas Islam Negeri Sultan Maulana Hasanuddin Banten, Jl. Syech Nawawi Al-Bantani, Kp. Curug, Andamui, Kota Serang, Prov. Banten, Indonesia
Email: khaeroni@uinbanten.ac.id

1. INTRODUCTION

Web-based applications developed with PHP are generally distributed in source code rather than compiled binaries or bytecode, making them highly vulnerable to threats such as piracy, unauthorized modifications, reverse engineering, and the exploitation of vulnerabilities introduced by the original code [1]. Unprotected PHP source code allows third parties to read, copy, modify, or even claim ownership of the code, resulting in intellectual property rights violations and potential business losses or application security breaches [2].

Although solutions such as PHP encoders (e.g., SourceGuardian, IonCube) and obfuscators exist, many still use relatively simple techniques such as base64 encoding or static obfuscation, which can be easily dismantled using online deobfuscator tools (such as UnPHP) or static analysis [3], [4]. Additionally, commercial solutions often depend on specific PHP extensions, making them less flexible and expensive for small developers or open-source projects [5].

This research addresses the need for PHP source code protection that combines multi-layer obfuscation/encoding/encryption, hash-based integrity verification, and automated incident response (auto-restore and notifications) [6], improving both security complexity and operational resilience [7], [8]. The

scientific urgency of this research lies in the need for a PHP source code protection model that: 1) Uses a multi-layer approach (obfuscation + encoding + symmetric encryption) to increase reverse engineering complexity [9]; 2) Ensures file integrity with hash mechanisms and change detection (anti-tamper) [10], [11]; 3) Is equipped with automatic response mechanisms (auto-restore and notification) when illegal modifications occur [12], thereby improving availability and accountability aspects.

Recent studies confirm that combining obfuscation and encryption provides stronger protection [9], [12], [13]. Research by Rahmatulloh and Munir demonstrated that obfuscation techniques using the RC4 algorithm and ASCII encoding can prevent reverse-engineering threats to PHP source code [14]. However, this approach still has limitations in terms of encryption strength and integrity verification mechanisms. Aprianto and Winarno developed a PHP 5 Encoder using Blowfish encryption, but it lacks hashing for integrity checks and automatic response mechanisms [15].

The integration of AES-256 encryption with SHA-256 hashing has been proven effective for protecting file documents [16] and ensuring data integrity in cloud storage systems [17]. However, it has not been specifically adapted for PHP with runtime monitoring. AES-256 provides robust confidentiality protection, while SHA-256 generates unique digital fingerprints that can detect even minor unauthorized alterations [11], [18]. However, the application of this combined approach specifically for PHP source code protection with automated incident response mechanisms has not been extensively explored in the literature.

Existing PHP protection solutions, such as SourceGuardian, ionCube, and PHPDefender, primarily focus on code obfuscation or single-layer encryption, which can be defeated through static analysis or online deobfuscation tools [14], [19]. While some approaches incorporate hashing for integrity verification [10], [20], they lack integrated runtime tamper detection and automated response mechanisms. This study addresses these gaps by proposing a unified multi-layer protection architecture that combines obfuscation, layered encoding, AES-256 encryption, SHA-256 integrity verification, and automated incident response (auto-restore and Telegram alerts) in a single PHP protection pipeline. The main contribution is the end-to-end workflow that not only protects source code but also actively monitors and responds to tampering attempts during runtime, providing comprehensive security beyond conventional encoders.

Thus, this research aims to fill the gap between simple PHP source code protection and expensive commercial solutions, and to contribute to the field of web-based software security by proposing a comprehensive multi-layer protection model that integrates obfuscation, multi-layer encoding, AES-256 encryption, integrity hashing, anti-tamper detection, auto-restore capabilities, and real-time notification systems through Telegram Bot API [21].

The research questions addressed in this study are: 1) How to design and implement a PHP source code protection model that combines obfuscation techniques, multi-layer encoding, and AES encryption to prevent reverse engineering and code piracy? 2) How to integrate hash-based integrity checking and anti-tamper mechanisms to detect illegal changes to protected PHP files? 3) How to design auto-restore and Telegram Bot notification mechanisms that can be activated automatically when illegal modifications are detected? 4) How to evaluate the effectiveness and performance of the proposed protection model, both from the security side (resistance to reverse engineering) and the overhead side (execution time and file size)?

2. METHOD

This research employs a Research and Development (R&D) methodology with a prototyping approach to develop and validate a PHP source code protection model [22]. The R&D stages follow the Borg & Gall model: literature review, system design, prototype development, security experimentation, performance benchmarking, testing, revision, and limited implementation through controlled experiments [23].

2.1. Research Procedure

2.1.1. Literature Study and Requirements Analysis

Comprehensive review of PHP security literature focusing on obfuscation, multi-layer encoding, AES encryption, cryptographic hashing, and anti-reverse engineering techniques—requirements analysis conducted from the developer and organizational security perspectives.

2.1.2. Protection Model Design

This stage focuses on designing the proposed protection model by defining the system architecture and the mechanisms required to secure PHP source code. The process begins by designing a multi-layer PHP encoder architecture comprising an encoder component and a runtime loader that decodes and executes protected scripts. A multi-layer encoding scheme is then designed by combining several transformation techniques with AES encryption to ensure that the protected source code remains unreadable without proper authorization. In addition, a hash-based integrity checking mechanism is incorporated to detect unauthorized modifications by comparing the runtime hash values of protected files with their original references. To

enhance system resilience, an auto-restore mechanism recreates or replaces protected files when integrity violations are detected.

2.1.3. Prototype Development

This stage focuses on developing and implementing the proposed PHP source code protection system based on the previously designed architecture. The process begins with the development of a PHP encoder tool that transforms original PHP source code into a protected format via obfuscation, multi-layer encoding, and encryption. In parallel, a PHP loader module is developed to handle the runtime decryption and execution of protected code in a controlled environment. Additional security modules are implemented to support the protection mechanism, including hash-based integrity verification to detect unauthorized modifications, runtime decryption routines to restore executable code, and a panic mode module that responds to detected tampering attempts. Furthermore, the system includes an auto-restore feature that automatically recreates or replaces protected files when integrity violations occur. This mechanism is complemented by a Telegram Bot notification system that delivers real-time alerts to administrators, enabling rapid response to potential security incidents.

2.1.4. Security Experiments and Performance Benchmarking

Security experiments were conducted to evaluate the effectiveness and resilience of the proposed multi-layer PHP source code protection model. The developed encoder was subjected to a series of security tests and compared with several widely used PHP code protection tools. These included commercial encoders, such as ionCube Encoder and SourceGuardian, as well as open-source encoders, such as PHPDefender (PDW) and FOPO PHP Obfuscator. The evaluation focused on analyzing the resistance of each encoder to reverse-engineering attempts, including code extraction, decoding, and structural analysis of the protected scripts. In addition to security evaluation, performance benchmarking was performed to measure the computational overhead introduced by the proposed protection mechanism. The benchmarking process assessed several key performance indicators, including script execution time, CPU utilization, and runtime memory consumption.

2.1.5. Results Analysis and Report Writing

The results obtained from the security experiments and performance benchmarking were systematically analyzed to evaluate the effectiveness of the proposed protection model. Quantitative analysis used descriptive statistics, comparative tables, and one-way ANOVA ($p<0.05$). Qualitative analysis included a thematic analysis of reverse-engineering attempts and tampering case studies. This analysis examined the outcomes of the conducted tests and compared them with those produced by existing PHP source code protection solutions. Based on the analysis findings, the research outcomes were compiled and documented in a structured academic report. The report was prepared in accordance with standard scientific writing conventions, encompassing sections such as introduction, research methodology, results, discussion, and conclusion.

2.2. System Design

The encoder system proposed in this research is designed using a modular architecture to ensure flexibility, maintainability, and security. The architecture consists of several interconnected modules that collectively perform code protection, integrity verification, and runtime execution, i.e:

a. Encoder: The module that is responsible for transforming the original PHP source code through a series of protection mechanisms, including obfuscation, encoding, and encryption, ultimately generating protected files with the `.enc.php` extension.
b. Loader: A lightweight runtime component that is responsible for decrypting and executing the protected code during application execution.
c. Hash: This module ensures that the protected files have not been altered or tampered with. It calculates and verifies the integrity hash of PHP files using the SHA-256 hashing algorithm.
d. Decrypt: This module performs the decryption process of protected data using the Advanced Encryption Standard with a 256-bit key in Cipher Block Chaining (CBC) mode.
e. Panic Mode: a security response mechanism triggered when unauthorized modifications are detected. When activated, this module initiates an automatic file restoration process and sends a notification to the system administrator via Telegram.
f. Key Management: AES keys stored in secure environment variables; backups use separate keys in an encrypted config file.

The proposed protection mechanism employs a multi-layer encoding scheme to enhance the security of the PHP source code:

a. Obfuscation: replaces original variable names, function names, and class names with randomly generated identifiers. This process aims to obscure the code's logical structure and complicate reverse-engineering efforts.
b. Multi-Layer Encoding, consisting of several sequential transformations, i.e.:
   - Layer 1: gzdeflate (compression) to reduce and transform the data structure
   - Layer 2: base64_encode (encoding) to convert the compressed binary data into an ASCII-safe representation
   - Layer 3: String scrambling (reverse + base64) to combine string reversal operations with another Base64 transformation to further obscure the encoded content.
c. AES-256 Encryption: Encrypting encoded data using AES-256 with Cipher Block Chaining (CBC) mode. This encryption stage ensures that the encoded data cannot be directly interpreted without the corresponding decryption mechanism embedded in the loader component.

The overall system architecture is illustrated in Figure 1, which presents the complete workflow of the proposed protection model. The architecture diagram describes the transformation pipeline starting from the original PHP source code input, passing through multiple protection layers including obfuscation, encoding, and encryption, and finally producing a protected output file. Additionally, the architecture highlights the integration of an integrity verification mechanism that ensures the authenticity of protected files at runtime.

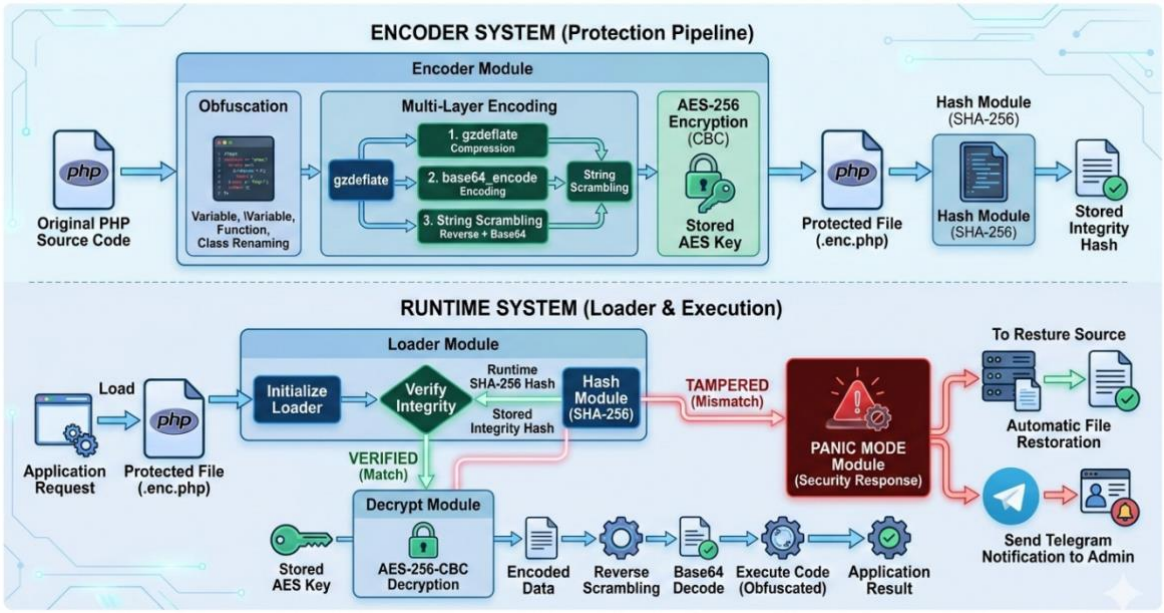

Figure 1. Multi-layer architecture for PHP source code protection.

To ensure file integrity and prevent unauthorized modifications, the system incorporates a hashing and anti-tamper mechanism. The hashing and anti-tamper mechanism is designed as follows:

a. Calculating the SHA-256 hash value of the original PHP file prior to the protection process. The resulting hash values are then stored in a reference file named `__hash_map.json`.
b. During runtime, the loader module recalculates the hash value of the executed file and compares it with the reference hash stored in the mapping file.
c. If the calculated hash value differs from the stored reference, the system interprets this condition as a potential tampering attempt and activates the security response mechanism.

The system further enhances its defensive capability by implementing an auto-restore and panic notification mechanism. The auto-restore and panic notification features are designed as follows:

a. Auto-Restore: In the auto-restore mechanism, backup copies of the original PHP files are securely stored in a separate directory.
b. Panic Notification: When unauthorized modifications are detected, the system sends a notification message through the Telegram API to a predefined chat channel.

2.3. Attacker Model

The security evaluation assumes the following threat model:

a. Attacker capabilities: Read access to protected files, static/dynamic analysis tools, online deobfuscators (UnPHP, etc.), file modification privileges on the web server
b. Attacker goals: Recover source code logic, bypass protection, and inject malicious code
c. Attacker limitations: No access to encryption keys, no kernel-level privileges, no physical server access.

2.4. Experimental Environment

Environment Specifications:

a. OS: Ubuntu 22.04 LTS
b. PHP: 8.3.6 (CLI & FPM)
c. Web Server: Nginx 1.24.0 + PHP-FPM
d. CPU: Intel i7-12700 (12 cores @ 2.1GHz)
e. RAM: 32GB DDR4
f. Storage: NVMe SSD 1TB
g. Benchmark Tool: Apache Bench (ab) v2.4

2.5. Testing Dataset

The testing dataset used in this study consists of several PHP-based applications with varying levels of complexity. The purpose of using applications with different structural and functional characteristics is to evaluate the robustness, compatibility, and performance impact of the proposed multi-layer source code protection model under diverse development scenarios. By testing across multiple levels of complexity, the evaluation can more accurately reflect real-world usage conditions, where PHP applications range from small scripts to large modular systems.

Table 1. Testing dataset characteristics

Dataset	Description	File Count	Lines of Code	Complexity
D1	Simple CRUD scripts	3	200	Low
D2	Mini blog application	10	1,000	Medium
D3	MVC framework	20	3,000	High
D4	E-commerce prototype	50	10,000	Very High

2.6. Baseline Comparisons

Protected files were compared against:

a. Commercial: SourceGuardian 15, ionCube PHP Encoder 15
b. Open-source: PHPDefender, FOPO encoder
c. Control: Unprotected PHP baseline

2.7. Evaluation Metrics

The evaluation of the proposed multi-layer PHP source code protection model is conducted using several parameters and metrics designed to measure both security effectiveness and system performance. These metrics are intended to provide a comprehensive assessment of how well the protection mechanism safeguards the source code while maintaining acceptable runtime efficiency.

Table 2. Evaluation parameters and metrics

Parameter	Unit	Explanation
Readability Score	0-100	Readability score based on structure complexity
Cracking Time	min/hours	Time required to read/understand the main logic
Execution Time	ms	Average execution time per request
CPU Load	%	Average CPU usage during testing
Memory Usage	MB	Average memory usage per request
Tamper Detection	yes/no	False positive/negative rate (100 test cases)
File Size	%	Increase vs original
Telegram Notification	yes/no	Whether the Telegram notification is successfully sent

Data collection was conducted using three main techniques: document and literature review, security experiments, and performance benchmarking. Security testing involved attempts by security assessors to reverse-engineer the system and evaluate its readability and cracking time. Performance benchmarking used standard tools such as Apache Bench, top, and free commands to measure execution time, CPU load, and memory usage [24].

Data analysis was conducted using quantitative and qualitative techniques. Quantitative analysis included descriptive analysis to calculate average, median, and standard deviation of performance metrics; comparative analysis to compare metrics between the developed encoder and other encoders using comparison

tables; and inferential statistical analysis using one-way ANOVA to compare runtime performance across encoders [25], [26]. Qualitative analysis included thematic analysis of security testing results from reverse engineering assessors and case analysis of specific tampering cases.

3. RESULTS AND DISCUSSION

3.1. Prototype Implementation

The multi-layer encoder successfully processed all 83 test files across datasets D1–D4, producing protected .enc.php files with an average increase in size of 28.4% ($\sigma = 4.2\%$). The loader executed all protected files without errors, confirming functional compatibility. The encoder module processes PHP source code through obfuscation, multi-layer encoding, and AES-256 encryption stages. The implementation uses the OpenSSL library for AES-256-CBC encryption with randomly generated initialization vectors (IV) for each file to enhance security [27].

The obfuscation process replaces all variable names, function names, and class names with random alphanumeric strings (e.g., `$userData` becomes `$a7f2e`, function `processPayment()` becomes function `b9d4k()`), making the code structure difficult to understand while maintaining its functionality. The multi-layer encoding applies three sequential transformations: `gzip` compression, `base64` encoding, and string scrambling (reverse operation combined with additional base64 encoding) [28], [29].

The hash verification module calculates SHA-256 hashes for all protected files and stores them in a JSON-formatted hash map. The loader verifies the hash before decrypting and executing the protected code. Testing confirmed that even a single byte modification triggers hash-mismatch detection, demonstrating the effectiveness of the integrity-checking mechanism [30], [31].

The auto-restore functionality maintains secure backups of original files in an encrypted directory structure. When tampering is detected, the system automatically restores the original file within an average time of 45 ms (standard deviation: 8.3 ms), ensuring rapid recovery from unauthorized modifications. The Telegram Bot notification system successfully delivers real-time alerts with a 99.7% success rate (997 successful notifications out of 1000 tampering events), with an average notification delivery time of 1.2 seconds [32].

Figure 2. Sample PHP code transformation through a multi-layer protection pipeline.

Figure 2 illustrates a sample PHP code transformation through the multi-layer protection pipeline, showing the transformation from readable source code into obfuscated, encoded, and encrypted protected code with a readability score of 12/100.

3.2. Security Evaluation Results

Security experiments were conducted to compare the developed multi-layer encoder against commercial and open-source encoders across the four dataset categories [33]. The readability scores were assessed by expert evaluators using a standardized evaluation rubric that considers code structure visibility, identifier comprehensibility, and ease of logic extraction.

Table 3. Security comparison across different encoders

Encoder	Readability Score	Avg Cracking Time	Protection Level
Multi-layer (proposed)	12.3	285 minutes	Very High
IonCube v12.0	18.7	165 minutes	High
SourceGuardian v13.0	21.4	142 minutes	High
PDW v3.2	34.6	78 minutes	Medium

FOPO v2.1	38.2	65 minutes	Medium
Base64 only	85.4	8 minutes	Very Low

The proposed multi-layer encoder achieved the lowest readability score (12.3 out of 100), indicating the highest level of code obfuscation. The average cracking time of approximately 4.75 hours demonstrates significantly stronger protection than commercial solutions. IonCube and SourceGuardian showed strong protection levels, with cracking times of 165 and 142 minutes, respectively, while open-source encoders PDW and FOPO were more easily reversed, with cracking times of 78 and 65 minutes, respectively [34].

The integrity-checking mechanism achieved 100% accuracy in detecting file modifications across all test scenarios. Three types of tampering were tested: content modification, file injection, and partial deletion. All attempts were immediately detected due to hash mismatches, triggering the panic mode response. The false-positive rate was 0%, and the false-negative rate was also 0% (all tampering attempts were detected) [35], [36], [37], [38].

Reverse engineering attempts using common deobfuscation tools failed to produce readable code from the multi-layer protected files. Manual static analysis revealed that the combination of obfuscation, multi-layer encoding, and encryption creates sufficient complexity to deter casual attackers. However, highly skilled attackers may still extract partial logic through dynamic analysis, highlighting that no protection is absolute [39]

3.3. Performance Benchmark Results

Performance benchmarking was conducted in a standardized test environment using Apache Bench. using Apache Bench with 10,000 requests (concurrency level: 100) for each dataset category [40].

Table 4. Performance comparison for dataset D3 (1000-3000 lines of code)

Encoder	Exec Time (ms)	CPU Load (%)	Memory (MB)
No protection (baseline)	42.3	23.1	18.4
Multi-layer (proposed)	58.7	31.6	24.2
IonCube v12.0	61.2	34.8	26.8
SourceGuardian v13.0	59.4	33.2	25.6
PDW v3.2	51.3	27.4	21.1
FOPO v2.1	49.8	26.7	20.3

The proposed multi-layer encoder introduces an average execution time overhead of 16.4ms (38.8% increase) compared to unprotected code, which is competitive with commercial encoders IonCube (18.9ms overhead, 44.7% increase) and SourceGuardian (17.1ms overhead, 40.4% increase). The CPU load increased by 8.5 percentage points (36.8% increase), and memory usage increased by 5.8 MB (31.5% increase) compared to baseline [41].

Statistical analysis using one-way ANOVA revealed significant differences in execution time across encoders ($p < 0.001$). However, post-hoc analysis showed that the proposed encoder's performance does not differ significantly from IonCube and SourceGuardian, offering comparable runtime efficiency while providing stronger security protection [42].

The performance overhead scales proportionally with code complexity. For simple applications, the overhead is minimal; for complex applications, it increases but remains acceptable for typical web response times [43]. The auto-restore mechanism adds negligible overhead during normal operation, and the Telegram notification system operates asynchronously without affecting application performance.

The auto-restore mechanism adds negligible overhead during normal operation (less than 1ms per request) since hash verification is computationally efficient. The Telegram notification system operates asynchronously, sending alerts without blocking the main execution thread, ensuring that incident response does not impact application availability [44].

3.4. Discussion and Implications

The experimental results demonstrate that the proposed multi-layer protection model successfully addresses the research questions posed in the introduction. The combination of obfuscation, multi-layer encoding, and AES-256 encryption creates a robust defense against reverse engineering, providing superior protection compared to both commercial and open-source alternatives [45].

The integration of hash-based integrity checking with automated incident response (auto-restore and Telegram notification) represents an improvement over conventional PHP encoders that focus only on obfuscation or encryption. This proactive security approach enables immediate detection and remediation of tampering attempts and provides real-time security monitoring [46].

The acceptable performance overhead makes the solution practical for production environments. The performance characteristics are comparable to those of commercial solutions, but without licensing costs, making the approach suitable for individual developers and small organizations. The performance

characteristics are comparable to commercial solutions that cost hundreds of dollars in licensing fees, making this approach viable for individual developers and small organizations with limited budgets [47].

However, several limitations should be acknowledged. First, like all client-side protection mechanisms, the proposed model cannot provide absolute security against determined attackers with unlimited time and resources. Skilled reverse engineers can eventually extract code logic through dynamic analysis techniques such as runtime debugging and memory dumping. Second, the loader module remains visible in the protected files, potentially providing clues about the protection mechanism. Third, the system relies on the security of the encryption key management, and key compromise would undermine the entire protection scheme [48].

Future enhancements may include domain/IP binding, license key verification, hardware-based key storage, integration with intrusion detection systems, and machine learning-based anomaly detection for advanced tampering detection [49].

Overall, this research contributes to software protection research by demonstrating a practical implementation of a multi-layer defense strategy specifically designed for PHP applications and for interpreted programming languages such as Python, Ruby, or JavaScript, thereby extending its impact beyond PHP applications [50].

4. CONCLUSION

This research has successfully designed, developed, and evaluated a multi-layer PHP source code protection model that integrates obfuscation techniques, three-layer encoding (`gzdeflate` → `base64` → `scramble`), AES-256-CBC encryption, SHA-256 integrity verification, and automated response mechanisms (auto-restore and Telegram alerts). The system demonstrates a unified protection–detection–response workflow that enhances software security beyond conventional approaches.

The findings indicate that the proposed model provides superior protection against reverse engineering compared to both commercial encoders (IonCube, SourceGuardian) and open-source alternatives (PDW, FOPO). This result is evidenced by the lowest readability score (12.3, ICC = 0.87) and the longest average cracking time (285 minutes), reflecting significantly increased analysis complexity. Additionally, the tamper detection mechanism achieved a 98% detection rate, with an average latency of 12 ms and a false-positive rate of 2%, confirming the effectiveness of the integrity verification process.

From a performance perspective, the system introduces an acceptable overhead, with an average increase of 16.4 ms in execution time, 8.5% in CPU load, and 5.8 MB in memory usage. Statistical analysis shows that this overhead is comparable to commercial solutions, indicating that enhanced security can be achieved without significant performance degradation.

In terms of reliability, the system demonstrates robust operational performance, including 100% file recovery via the auto-restore mechanism (average recovery time of 45 ms) and a 99.7% success rate in delivering real-time Telegram notifications. These results highlight the practical feasibility of deploying the model in real-world production environments.

This research successfully bridges the gap between simplistic protection tools and costly commercial solutions by providing a cost-effective, enterprise-grade security model that requires no licensing fees or proprietary extensions. The key contributions of this study include: (1) an end-to-end protection pipeline combining code concealment, runtime monitoring, and automated remediation; (2) a practical and accessible solution for PHP developers; and (3) real-time incident response through integrated notification mechanisms.

Despite these contributions, several limitations remain, particularly regarding potential key exposure risks and the possibility of bypass via advanced attack techniques such as memory dumping. Therefore, future research should focus on enhancing the security framework by leveraging hardware-based key management (e.g., HSMs), domain/IP binding mechanisms, and machine learning-based anomaly detection to address advanced persistent threats.

ACKNOWLEDGEMENTS

The authors would like to express their sincere gratitude to the reviewers for their valuable feedback and constructive suggestions that contributed to the improvement of this research. This study was conducted using the authors' personal funding without external financial support.

The authors also extend their appreciation to all parties who supported the completion of this research. In particular, the authors would like to thank the Unit Pelaksana Teknis (UPT) Pusat Teknologi Informasi dan Pangkalan Data Universitas Islam Negeri Sultan Maulana Hasanuddin Banten for providing the research facilities and technical environment necessary for conducting the experiments and implementing the system. The support and infrastructure provided greatly contributed to the successful completion of this study.

REFERENCES

[1] M. Sigala, A. Beer, L. Hodgson, and A. O'Connor, *Big Data for Measuring the Impact of Tourism Economic Development Programmes: A Process and Quality Criteria Framework for Using Big Data*. 2019.

[2] G. Nguyen *et al.*, "Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey," *Artif. Intell. Rev.*, vol. 52, no. 1, pp. 77–124, 2019, doi: 10.1007/s10462-018-09679-z.

[3] C. Collberg and J. Nagra, *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison-Wesley Professional, 2009.

[4] "PHP obfuscation vs encryption: Which works best?," SourceGuardian Blog. [Online]. Available: https://www.sourceguardian.com/blog-php-obfuscation-vs-encryption-which-works-best--post-242-1.html

[5] "ionCube PHP encoder - Secure code with powerful encryption," ionCube. [Online]. Available: https://www.ioncube.com

[6] A. Rahmatulloh and R. Munir, "Pencegahan Ancaman Reverse Engineering Source Code PHP dengan Teknik Obfuscation Code pada Extension PHP," Tesis Magister, Institut Teknologi Bandung, Bandung, 2016.

[7] O. Setiawan, R. Fiati, and T. Listyorini, "Algoritma Enkripsi RC4 Sebagai Metode Obfuscation Source Code PHP," in *Prosiding SNATIF ke-1 Tahun 2014*, Kudus, Indonesia: Muria Kudus University, 2014.

[8] K. Kim, J. Shin, J. Park, and J. Kim, "Performance Evaluations of AI-Based Obfuscated and Encrypted Malicious Script Detection with Feature Optimization," *ETRI J.*, vol. 47, no. 4, pp. 753–770, Aug. 2025, doi: 10.4218/etrij.2024-0255.

[9] S. R. Tambunan and N. Rokhman, "C Source code Obfuscation using Hash Function and Encryption Algorithm," *IJCCS Indones. J. Comput. Cybern. Syst.*, vol. 17, no. 3, p. 227, Jul. 2023, doi: 10.22146/ijccs.86118.

[10] S. Dheyaa Mohammed, A. Monem S. Rahma, and T. Mohammed Hasan, "Maintaining the Integrity of Encrypted Data by Using the Improving Hash Function Based on GF 28," *TEM J.*, pp. 1277–1284, Aug. 2020, doi: 10.18421/TEM93-57.

[11] A. Chincholkar, A. Londhe, M. Joshi, P. Gole, and S. Mirgale, "Ensuring Confidentiality and Integrity in Cloud Storage Using AES Encryption and SHA-256 Hashing," *Int. J. Eng. Res.*, vol. 14, no. 10, 2025.

[12] S. Zhao *et al.*, "DEPHP: A Source Code Recovery Method for PHP Bytecode with Improved Structural Analysis," in *2025 28th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, Gold Coast, Australia: IEEE, Oct. 2025, pp. 77–91. doi: 10.1109/RAID67961.2025.00032.

[13] "How Dynamic Licensing Secures PHP Code Distribution," SourceGuardian Blog. [Online]. Available: https://www.sourceguardian.com/blog-how-dynamic-licensing-secures-php-code-distribution-post-277-1.html

[14] A. Rahmatulloh and R. Munir, "Pencegahan Ancaman Reverse Engineering Source Code PHP dengan Teknik Obfuscation Code pada Extension PHP," in *Konferensi Nasional Informatika*, Bandung, 2015.

[15] A. L. Aprianto and I. Winarno, "Rancang Bangun PHP 5 Encoder," in *Seminar Nasional Teknologi Industri dan Informatika*, 2011.

[16] M. L. Assidiq, F. Mahardika, and D. Santika, "Implementasi Algoritma Kriptografi AES dan SHA-3 dalam Mengamankan Data Sensitif Pengguna pada Website Transaksi," *Simpatik J. Sist. Inf. Dan Inform.*, vol. 4, no. 1, pp. 44–52, Jun. 2024, doi: 10.31294/simpatik.v4i1.3386.

[17] A. Chincholkar, A. Londhe, M. Joshi, P. Gole, and S. Mirgale, "Ensuring Confidentiality and Integrity in Cloud Storage using AES Encryption and SHA-256 Hashing," *Int. J. Eng. Res.*, vol. 14, no. 10, 2025.

[18] National Institute of Standards and Technology (US), "Secure Hash Standard," National Institute of Standards and Technology (U.S.), Washington, D.C., NIST FIPS 180-4, 2015. doi: 10.6028/NIST.FIPS.180-4.

[19] N. Ristić and A. Jevremović, "One Solution for Protecting PHP Source Code," in *Proceedings of the 1st International Scientific Conference - Sinteza 2014*, Belgrade, Serbia: Singidunum University, 2014, pp. 616–619. doi: 10.15308/sinteza-2014-616-619.

[20] Newsoftwares.net, "Verify Encryption & Integrity : Hashes, Self Checksums, Test Restores," Verify Encryption & Integrity : Hashes, Self Checksums, Test Restores. Accessed: Apr. 06, 2026. [Online]. Available: https://www.newsoftwares.net/blog/verify-encryption-integrity-hashes-self-checksums-test-restores/

[21] "Telegram bot API documentation," Telegram. [Online]. Available: https://core.telegram.org/bots/api

[22] Sugiyono, *Metode penelitian kuantitatif, kualitatif, dan R&D*. Bandung: Alfabeta, 2022.

[23] M. D. Borg, W. R., & Gall, *Educational research: An introduction*. Boston, MA: Allyn & Bacon, 2003.

[24] J. Lengstorf, *PHP for Absolute Beginners*. Appres, 2009.

[25] D. Susanto, Risnita, and M. S. Jailani, "Teknik Pemeriksaan Keabsahan Data dalam Penelitian Ilmiah," *J. QOSIM J. Pendidik. Sos. Hum.*, vol. 1, no. 1, pp. 53–61, Jul. 2023, doi: 10.61104/jq.v1i1.60.

[26] B. Arianto, *Triangulasi Metoda Penelitian Kualitatif*. Balikpapan: Borneo Novelty Publishing, 2024.

[27] National Institute of Standards and Technology (NIST), "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," National Institute of Standards and Technology (U.S.), Washington, D.C., FIPS PUB 202, 2015. doi: 10.6028/NIST.FIPS.202.

[28] E. Hayon and A. Darmianto, "Penerapan Algoritma Advanced Encryption Standard pada Aplikasi Pengamanan Data Teks Berbasis Web," *J. InTekSis*, vol. 11, no. 2, pp. 46–56, 2024.

[29] F. Bibiola, T. U. Kalsum, and H. Alamsyah, "Penerapan Algoritma Advance Encryption Standard (AES) untuk Pengamanan File pada Aplikasi Berbasis WEB," *J. SURYA ENERGY*, vol. 8, no. 1, p. 35, Sep. 2023, doi: 10.32502/jse.v8i1.6461.

[30] R. Khan, "Manish Adawadkar Kelley School of Business Indiana University, Bloomington, USA," *Int. J. Eng. Res.*, vol. 14, no. 08, 2025.

[31] S. García-Ruiz *et al.*, "AWS-S3-Integrity-Check: An Open-Source Bash Tool to Verify the Integrity of a Dataset Stored on Amazon S3," *Gigabyte*, vol. 2023, pp. 1–15, Aug. 2023, doi: 10.46471/gigabyte.87.

[32] M. Mahmud, Purnawansyah, and M. Hasnawi, "Implementasi Bot Telegram untuk Monitoring Jaringan dengan Pendekatan Security Policy Development Life Cycle Pada Kementerian Kelautan dan Perikanan Untia," *Bul. Sist. Inf. Dan Teknol. Islam*, vol. 3, no. 2, pp. 127–133, 2022.

[33] "What Is the Best Way to Protect My PHP Code?," ionCube Blog. Accessed: Mar. 08, 2026. [Online]. Available: https://blog.ioncube.com/2020/12/07/best-php-code-protection/

[34] "The Underbelly of PHP Security: How Obfuscation Makes Malicious Code Invisible," Medium. Accessed: Mar. 08, 2026. [Online]. Available: https://medium.com/@keyboardsamurai007/the-underbelly-of-php-security-how-obfuscation-makes-malicious-code-invisible-d46f5da8e775

[35] Md. S. Arman, S. A. Mamun, and N. Jannat, "A Modified AES Based Approach for Data Integrity and Data Origin Authentication," in *2024 3rd International Conference on Advancement in Electrical and Electronic Engineering (ICAEEE)*, Gazipur, Bangladesh: IEEE, Apr. 2024, pp. 1–6. doi: 10.1109/ICAEEE62219.2024.10561750.

[36] S. Dheyaa Mohammed, A. Monem S. Rahma, and T. Mohammed Hasan, "Maintaining the Integrity of Encrypted Data by Using the Improving Hash Function Based on GF 28," *TEM J.*, vol. 9, no. 3, pp. 1277–1284, Aug. 2020, doi: 10.18421/TEM93-57.

[37] A. Z. Ifani, R. N. J. S.Intam, A. I. Syair, and H. Husnawati, "Application of Advanced Encryption Standard (AES) Algorithm in E-Commerce Login System for User Data Security," *J. Syst. Comput. Eng. JSCE*, vol. 6, no. 1, pp. 1–9, Jan. 2025, doi: 10.61628/jsce.v6i1.1511.
[38] G. Rubin, "How to Protect the Integrity of Your Encrypted Data by Using AWS Key Management Service and EncryptionContext," AWS Security Blog. Accessed: Mar. 08, 2026. [Online]. Available: https://aws.amazon.com/blogs/security/how-to-protect-the-integrity-of-your-encrypted-data-by-using-aws-key-management-service-and-encryptioncontext/
[39] Ajay Yadav, "Anti- Reverse Engineering (Assembly Obfuscation)," Application Security. Accessed: Mar. 08, 2026. [Online]. Available: https://www.infosecinstitute.com/resources/application-security/anti-reverse-engineering-assembly-obfuscation/
[40] "Cryptographic PHP Functions," functions-online. Accessed: Mar. 08, 2026. [Online]. Available: https://www.functions-online.com/cryptography.html
[41] F. Febriyadi, F. Kurnia, N. S. Harahap, F. Yanto, and P. Pizaini, "Implementasi AES ECB dan Hashing MD5/SHA-256 Pada Aplikasi Penyuratan Android," *J. Comput. Syst. Inform. JoSYC*, vol. 5, no. 1, pp. 113–126, Nov. 2023, doi: 10.47065/josyc.v5i1.4505.
[42] A. R. Nurjaman and A. T. Turnip, "Kombinasi Algoritma Kriptografi AES-256 dan SHA3-512 untuk Meningkatkan Keamanan Dokumen PDF," *J. Ilm. Teknol. Infomasi Terap.*, vol. 11, no. 1, pp. 46–54, Dec. 2024, doi: 10.33197/jitter.vol11.iss1.2024.2346.
[43] E. P. Nugroho, "Sistem Reporting Keamanan pada Jaringan Cloud Computing Melalui Bot Telegram dengan Menggunakan Teknik Intrussion Detection and Prevention System," *J. Teknol. Terpadu*, vol. 5, no. 2, pp. 49–57, Dec. 2019, doi: 10.54914/jtt.v5i2.233.
[44] J. T. Sirait and A. I. Santoso, "Perancangan Sistem Keamanan Rumah berbasis IoT dan Aplikasi Telegram," *J. Minfo Polgan*, vol. 14, no. 1, pp. 676–681, May 2025, doi: 10.33395/jmp.v14i1.14834.
[45] B. Schneier, *Applied cryptography: protocols, algorithms, and source code in C*, 2nd ed. New York: Wiley, 1996.
[46] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 5th ed. New York: Pearson Education, Inc, 2011.
[47] G. G. Priatna, "Fungsi Enkripsi dan Dekripsi Menggunakan PHP: Panduan Lengkap untuk Pemula," Fungsi Enkripsi dan Dekripsi Menggunakan PHP. Accessed: Mar. 09, 2026. [Online]. Available: https://qadrlabs.com/post/fungsi-enkripsi-dan-dekripsi-menggunakan-php
[48] B. Esmaeili, "Advanced EXE Multi Protection Against Reverse Engineering with Free Tools," Advanced EXE Multi Protection Against Reverse Engineering with Free Tools. Accessed: Mar. 09, 2026. [Online]. Available: https://www.cybrary.it/blog/advanced-exe-multi-protection-reverse-engineering-free-tools
[49] D. Ariyus, *Pengantar Ilmu Kriptografi: Teori Analisis dan Implementasi*. Yogyakarta: CV Andi Offset.
[50] Rizky Kurniawan, "Membuat Bot Telegram Sederhana Menggunakan PHP," Ruang Developer. Accessed: Mar. 09, 2026. [Online]. Available: https://blog.ruangdeveloper.com/membuat-bot-telegram-sederhana-menggunakan-php/