Padding Oracle Attack on AES-CBC Encrypted Web Session Cookies and Migration to AES-GCM Authenticated Tokens

Aldoy Fauzan Avanza - 18223113
Department of Information Systems and Technology
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jl. Ganesha 10, Bandung, Indonesia
E-mail: aldoyfa@gmail.com , 18223113@std.stei.itb.ac.id

***Abstract*—Encrypted web session cookies are attacker-controlled ciphertexts and therefore require integrity as well as confidentiality. This work implements a local Flask prototype whose AES-256-CBC cookie exposes distinct padding and payload errors, then develops an adaptive PKCS number 7 padding-oracle attack that recovers the canonical session JSON without the key. Across 35 full-profile trials, plaintext recovery succeeded in every trial. Median oracle queries increased from 4,763.5 for two ciphertext blocks to 31,822 for sixteen blocks, while local in-process recovery increased from 0.162 s to 1.060 s. For four blocks, a modeled five millisecond per-query delay raised median recovery time from 0.284 s to 42.94 s. In 1,000 one-bit mutations per scheme, unauthenticated CBC accepted 12.6 percent, whereas CBC Encrypt-then-MAC and AES-GCM rejected 100 percent before payload processing. AES-GCM also produced shorter tokens than CBC-HMAC and lower measured latency for the tested implementation. The results show that generic errors alone are not a cryptographic repair; versioned migration to authenticated encryption removes the padding oracle and provides explicit tamper detection.**

Keywords—Padding oracle; AES-CBC; PKCS#7; AES-GCM; authenticated encryption; session cookie

I. Introduction

A web session token crosses a trust boundary on every request: the server creates it, but the browser stores and returns it. An attacker can copy, truncate, reorder, or modify its bytes even when the cryptographic key remains secret. Consequently, a token design must provide both confidentiality and integrity. AES-CBC provides confidentiality for fixed-size blocks when used with an unpredictable initialization vector; however, CBC does not inherently authenticate either the initialization vector or the ciphertext [1]. Treating encrypted data as trustworthy solely because it decrypts successfully is therefore a protocol design error.

This error becomes exploitable when CBC is combined with PKCS#7 padding and a server reveals whether the unpadding operation succeeded. Vaudenay demonstrated that such a validity signal transforms a decryption endpoint into a chosen-ciphertext oracle [3]. Subsequent research generalized and optimized the technique [4], while attacks such as Lucky Thirteen showed that even ostensibly uniform protocol responses may still leak information through timing differences [10]. Importantly, the vulnerability does not represent a practical break of AES itself. Rather, it arises from the malleability of CBC and the application's exposure of distinguishable validation outcomes.

This paper addresses these issues by experimentally evaluating the practicality and impact of padding-oracle attacks against AES-CBC-encrypted web session cookies and by comparing authenticated-encryption countermeasures. The specific objectives of this paper are to (1) evaluate whether the complete plaintext of an AES-CBC-encrypted web session cookie can be recovered using only a padding-validity oracle; (2) measure the effect of ciphertext length and per-query delay on attack cost, including oracle-query requirements and recovery time; (3) compare the security properties of unauthenticated CBC, CBC Encrypt-then-MAC, and AES-GCM under ciphertext-modification attempts; and (4) analyze the size and performance implications of migrating from unauthenticated CBC to authenticated-encryption schemes. The experiments are restricted to a local prototype developed for this study, and no third-party application, production system, or real user account is tested.

II. Background and Related Work

A. CBC and PKCS#7 Padding

For plaintext block P_i, ciphertext block C_i, key K, and C_0 equal to the initialization vector (IV), CBC encryption and decryption are defined as

$$C_i = E_K(P_i \oplus C_{i-1}) \tag{1}$$

$$P_i = D_K(C_i) \oplus C_{i-1} \tag{2}$$

NIST SP 800-38A specifies CBC and emphasizes the unpredictability requirement for the IV [1]. The exclusive-OR relation in (2) also means that changing C_{i-1} predictably changes the next decrypted plaintext block, even though $D_K(C_i)$ is unknown.

CBC requires a complete final block. CMS padding appends k bytes, each containing the integer k, where $1 \leq k \leq 16$ for

AES [5]. A full block of padding is added when the input is already block-aligned. During decryption, an implementation normally validates every padding byte before parsing the application payload. If the observable response differs between invalid padding and later errors, the server reveals a padding-validity signal about a chosen ciphertext. For a modified previous ciphertext block C'_{i-1}, the resulting plaintext block is

$$P'_i = D_K(C_i) \oplus C'_{i-1} \tag{3}$$

Equation (3) shows that modifying the previous ciphertext block predictably modifies the next decrypted plaintext block, even though the attacker does not know $D_K(C_i)$.

B. Padding Oracles

To recover a byte, the attacker makes the target ciphertext block the final block and modifies its predecessor. For padding value 0x01, at most 256 candidates are tested for the final predecessor byte. A valid response reveals the corresponding intermediate byte $D_K(C_i)$. Already recovered suffix bytes are then rewritten to request 0x02 0x02, 0x03 0x03 0x03, and so forth. XOR with the original predecessor yields the original plaintext. The procedure repeats independently for each block.

A false positive can occur for the last byte when a candidate preserves an existing multi-byte padding suffix. The implementation confirms a 0x01 candidate by flipping the adjacent byte: genuine one-byte padding remains valid, while an accidental longer suffix is broken. This additional query is included in all reported totals. The theoretical worst case is close to 256 queries per recovered byte, although the expected count is lower because the successful guess is typically encountered midway through the search.

C. Authentication and AEAD

Encrypt-then-MAC computes a MAC over the version, IV, and ciphertext and verifies it before CBC decryption. With independent encryption and MAC keys, a modified token is rejected without reaching unpadding. Authenticated encryption with associated data integrates confidentiality and integrity in one interface [6], [8]. GCM combines counter-mode encryption with a polynomial authenticator; NIST recommends a 96-bit IV for efficient interoperable processing and requires IV uniqueness for a fixed key [2]. The version string is associated data in this study, so changing the token format identifier also invalidates the tag.

III. Threat Model and System Design

A. Attacker Capabilities and Scope

The attacker can obtain a valid token for a synthetic account, modify arbitrary token bytes, and submit unlimited requests to the local validation endpoint. The attacker observes HTTP status and error class in the leaky configuration but cannot read server memory, obtain keys, or influence the random-number generator. TLS, rate limiting, distributed noise, browser isolation, and key compromise are outside the primary experiment. These exclusions isolate the cryptographic protocol behavior rather than model a production breach.

B. Token Formats

```
v1.cbc.Base64URL(IV          || ciphertext)

v1.etm.Base64URL(IV          || ciphertext || HMAC-SHA256)

v2.gcm.Base64URL(nonce || ciphertext || tag)
```

All encryption keys are 256 bits. CBC uses a fresh 16-byte IV and PKCS#7 padding. The Encrypt-then-MAC control uses a separate 256-bit HMAC key and authenticates the ASCII version prefix together with IV and ciphertext. GCM uses a fresh 12-byte nonce, a 16-byte tag, and the version identifier v2.gcm as associated data. Payloads are canonical UTF-8 JSON with sorted keys and no insignificant whitespace. The public research keys are deterministic only to make the experiment reproducible.

C. Validation Behavior

TABLE I. Server-Side Validation Observations

Mode	Modified Tokens	Externally Visible Result
CBC-leaky	Unpadding fails	403 invalid padding
CBC-leaky	Padding valid, JSON fails	422 invalid_payload
CBC-unified	Any validation failure	401 invalid_token
CBC+HMAC /GCM	Authentication fails	401 invalid_token

The CBC-unified variant is a negative control, not the recommended repair. It maps validation failures to one status/body and therefore removes the explicit oracle used by the attack script, but the token remains unauthenticated. Different code paths may also retain timing differences, as prior protocol attacks caution [10]. Only the authenticated variants establish a cryptographic rejection boundary before plaintext use.

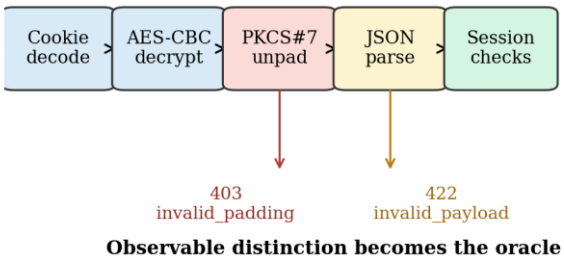

Fig. 1. Vulnerable CBC validation exposes the stage at which processing failed.

Figure 1 makes the information flow explicit. The attacker does not need the decrypted JSON response. It is enough to distinguish the branch leaving the unpadding stage from any branch reached after padding succeeds. The response may be a status code, body string, connection behavior, cache effect, or sufficiently separable latency distribution. This study intentionally uses status and JSON error labels so the causal

mechanism is observable and testable rather than hidden inside measurement noise.

IV. Attack Implementation

The attack engine receives only a token and a Boolean oracle. It splits the IV and ciphertext into 16-byte blocks. For every target C_i, it submits a two-block envelope in which the crafted IV acts as C'_{i-1} and C_i is the sole ciphertext block. This truncation makes the target block subject to final-block padding validation regardless of its original position. After recovering all intermediate bytes, the engine XORs them with the original IV or ciphertext predecessor and removes the original final padding.

```
for each target block C_i:
  for pad = 1 .. 16:
    rewrite known suffix bytes to pad
    for guess = 0 .. 255:
      if oracle(crafted_predecessor || C_i):
        confirm pad=1; record intermediate byte;
break
plaintext_i = intermediate_i XOR
original_predecessor
```

Queries include confirmation probes, and failure to find a consistent candidate raises an explicit AttackFailure rather than returning partial plaintext. Unit tests cover one-, multi-, and full-block padding behavior indirectly through payloads of different lengths. A separate test supplies an oracle that always returns false and confirms that the attack terminates as a failure.

A. Byte-Recovery Invariant

At padding depth p, positions $16 - p + 1$ through 15 of the crafted predecessor are fixed to $I_j \oplus p$, where I_j is the recovered intermediate byte. The candidate at position $16 - p$ is accepted when $D_K(C_i)[16 - p]$ XOR candidate equals p. Therefore I_{16-p} equals candidate XOR p. This invariant is independent of the original predecessor; the original block is used only after all intermediate bytes have been recovered. Keeping intermediate-state recovery separate from plaintext reconstruction simplifies testing and prevents accidental use of modified bytes in the final XOR.

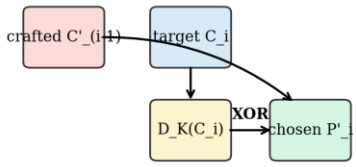

Fig. 2. A crafted predecessor controls the target block's decrypted plaintext suffix.

For b ciphertext blocks, the attack recovers 16b bytes and makes at most approximately 4096b candidate probes plus confirmation probes. The actual count depends on candidate order and intermediate-byte values. Because blocks are attacked independently after truncation, errors do not accumulate across blocks. The final PKCS#7 removal is performed only once, after concatenating all recovered plaintext blocks.

B. Worked Four-Block Example

```
Original :
{"f":"xxxxxxxxxxxxxxxxxxxxxx","r":"user","u":1}
Recovered:
{"f":"xxxxxxxxxxxxxxxxxxxxxx","r":"user","u":1}
Comparison: exact byte-for-byte match
```

The filler is not secret or semantically important; it makes the padded length deterministic. Exact comparison is performed on serialized bytes rather than parsed objects, so success requires recovering braces, quotes, key order, separators, values, and final padding correctly. This is stricter than comparing decoded dictionaries, which could conceal differences in serialization.

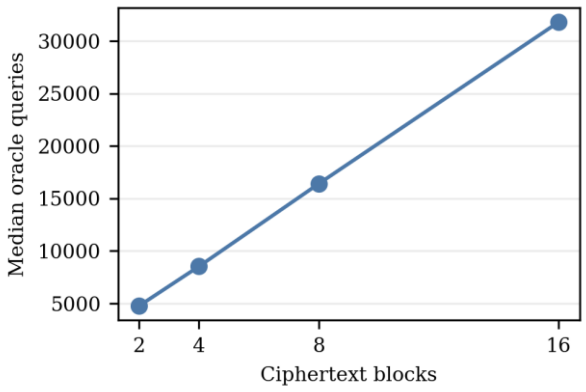

Fig. 3. Median oracle queries increase approximately linearly with ciphertext blocks.

V. Secure Migration Design

A. CBC Encrypt-then-MAC

The control computes HMAC-SHA256 over v1.etm || IV || ciphertext. Verification uses a constant-time comparison and occurs before CBC decryption, unpadding, or JSON parsing. Encryption and MAC keys are independent. This construction demonstrates that CBC can be protected when composed correctly; it also separates the effect of integrity from the effect of changing the block-cipher mode. Its cost is an additional 32-byte tag and a second cryptographic operation.

B. AES-GCM Token

The migration target calls a standard AES-GCM AEAD interface rather than implementing GHASH or counter management directly. Encryption returns ciphertext concatenated with a 128-bit tag. Decryption authenticates the tag before returning plaintext. The 96-bit nonce is generated from

the operating system CSPRNG, and a test encrypts the same payload 500 times to detect accidental reuse. This test is not a proof of uniqueness; production deployments must enforce a nonce strategy appropriate to key lifetime and issuance volume.

A deployed migration should introduce v2.gcm issuance first, retain bounded read-only validation for legacy v1 tokens, and rotate to a new GCM key. Legacy acceptance should have a fixed sunset shorter than the maximum session lifetime. The parser must dispatch by an authenticated or otherwise strictly parsed version, reject unknown versions, and never silently fall back from a failed v2 authentication attempt to v1 processing. Operational logs should record only a generic validation failure category visible to clients and avoid exposing internal padding or tag stages.

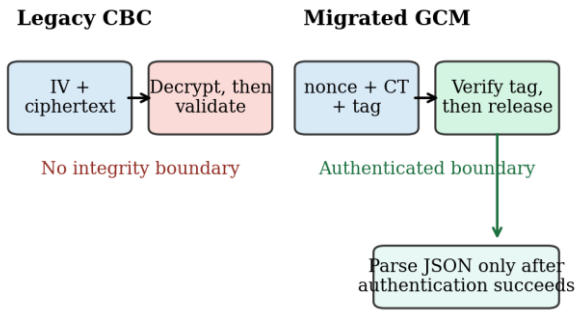

Fig. 4. Migration moves the trust boundary before plaintext parsing.

C. Nonce and Key Management

GCM security depends critically on nonce uniqueness under a fixed key. Reusing a nonce repeats the counter-mode keystream and creates algebraic relations in the authentication function, so a strong tag does not compensate for reuse [2]. The prototype calls the operating-system random source for every 96-bit nonce. At production scale, the issuer must account for the birthday bound, concurrency, rollback, cloning, and multi-node coordination. A counter partitioned by issuer, a persisted monotonic allocator, or a library-supported construction with an appropriate misuse model may be preferable to relying only on random sampling.

Migration keys must be purpose-specific. The legacy CBC encryption key, CBC-HMAC encryption key, HMAC key, and GCM key are logically distinct even if a key-management service derives them from one protected root. Version and purpose labels should be included in derivation context. Rotation must preserve only the minimum key set needed to validate unexpired tokens, and telemetry should identify token versions without logging token contents.

D. Security Properties and Residual Risks

Confidentiality means that an observer without the key should not learn the session plaintext from tokens. CBC can provide this property only under an appropriate chosen-plaintext model and does not by itself address active modification. Ciphertext integrity means that an attacker should not create a new ciphertext accepted by the receiver. The vulnerable token does not satisfy ciphertext integrity because neither the IV nor ciphertext is authenticated; the 12.6% mutation acceptance measured later is an application-level manifestation of that design gap.

The padding oracle is stronger than ordinary malleability because it converts receiver behavior into a decryption capability. The attacker does not forge a valid token first. Instead, many invalid candidates are intentionally submitted, and the validation branch identifies useful candidates. Thus input validation after decryption cannot repair the construction: the parser's rejection is itself the second distinguishable outcome required by the oracle. Authentication must occur before any padding or syntax decision becomes externally observable.

Encrypt-then-MAC provides the required ordering when the MAC is unforgeable, keys are independent, and all encrypted framing fields are covered. Omitting the IV, algorithm identifier, or version from the MAC can reintroduce unauthenticated control data. Comparing tags with ordinary early-exit equality may create a separate timing side channel, so the implementation uses constant-time comparison. These requirements make CBC-HMAC secure in principle but easier to compose incorrectly than a single, standard AEAD API.

AES-GCM authenticates ciphertext and associated data, but it does not solve replay or endpoint authorization. A copied, unexpired token with a valid tag remains valid unless the application includes and checks a server-side revocation identifier, issuance epoch, audience, or one-time state. Likewise, authenticated session claims are only as correct as the authorization rules that create and consume them. The migration therefore removes the padding oracle and tampering class studied here, not every web-session threat.

TABLE II. DEPLOYMENT MIGRATION CHECKPOINTS

Phase	Required action	Failure prevented
Prepare	Create v2 key and strict parser	Key/version ambiguity
Issue	Mint only v2.gcm tokens	New legacy exposure
Overlap	Read v1 for bounded lifetime	Forced mass logout
Observe	Count versions; log no token data	Secret/session leakage
Retire	Disable v1 and destroy old keys	Indefinite downgrade path

The overlap phase must not become permanent compatibility. A migration completion condition should be expressed as both time and observed traffic: for example, after the maximum legacy session lifetime has elapsed and residual v1 validation is negligible, v1 support is disabled. Failed v2 authentication is never retried as v1. If a rollback is necessary, issuance can be paused while existing v2 validation remains available; rolling back to unauthenticated issuance would recreate the original vulnerability.

VI. EXPERIMENTAL METHODOLOGY

Experiments ran on Windows 11, Python 3.14.2, cryptography 41.0.7, and an AMD64 Family 25 processor. Seed 20260619 determines synthetic IVs and mutation locations.

Canonical payloads contain a user identifier, role, and filler selected so that CBC produces exactly 2, 4, 8, or 16 ciphertext blocks.

Attack trials total 35: ten each for 2, 4, and 8 blocks and five for 16 blocks. Metrics are exact-match success, oracle queries, recovered bytes, and elapsed in-process time. The 5 ms network condition is modeled, not slept, as measured CPU time plus 0.005 seconds per query; this preserves the observed query count while making the latency assumption explicit.

Mutation testing applies 1,000 deterministic one-bit changes to each scheme and records acceptance or the validation class. Performance testing performs 1,000 warm-up calls, followed by five rounds of 10,000 encryption and decryption operations per scheme and payload size. Results report mean, median, sample standard deviation, minimum, maximum, and linearly interpolated 95th percentile. Timing is a comparative microbenchmark, not a constant-time audit.

A. Controls and Reproducibility

Deterministic keys and seeds control the experiment, while encryption APIs still use their production nonce/IV generation paths except where a trial intentionally supplies a reproducible IV. The full runner writes raw rows before aggregation. Summary values can therefore be recalculated independently, and figures read CSV rather than embedded constants. A quick profile with reduced iterations exists only for smoke testing; every value in this paper comes from the full profile recorded in metadata.json.

The test suite independently validates round trips, a deterministic CBC known-answer token, malformed Base64URL, invalid lengths, distinguishable padding and JSON failures, authentication-tag mutations, expiration, 500 sampled GCM nonces, exact oracle recovery, oracle failure without a signal, and all Flask modes. This separation reduces the risk that experiment success merely reflects two components sharing the same defect.

VII. Results and Discussion

A. Plaintext Recovery and Scaling

TABLE III. Padding-Oracle Recovery Results

Blocks	Trials	Median queries	Median time (s)	Exact
2	10	4,763.5	0.162	100%
4	10	8,532.5	0.284	100%
8	10	16,419.0	0.578	100%
16	5	31,822.0	1.060	100%

TABLE IV. Query Distribution across Attack Trials

Blocks	Mean	Std. dev.	P95	Min-max
2	4,733.1	899.8	5,731.9	3,150-5,763
4	8,456.6	780.2	9,524.4	7,298-9,717
8	16,397.8	1,585.0	18,372.8	13,393-18,605
16	31,936.8	1,352.2	33,574.6	30,345-33,806

All 35 trials recovered byte-for-byte identical canonical JSON. Median queries rose from 4,763.5 for two blocks to 31,822 for sixteen blocks. Median local time rose from 0.162 s to 1.060 s. The near-linear relationship follows the fixed 16 recovered bytes per additional block and the bounded 256-candidate search per byte. Variation arises from the position of each successful candidate in the deterministic guess order.

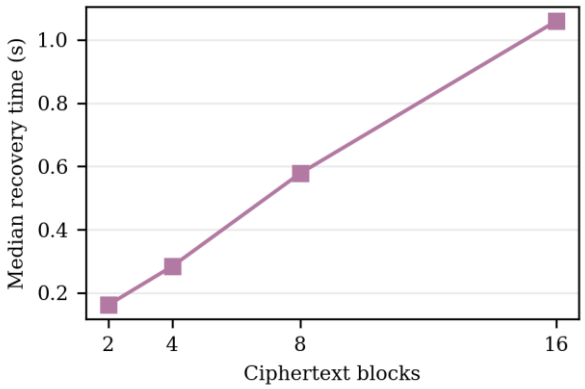

Fig. 5. Median in-process plaintext-recovery time by ciphertext length.

For the four-block payload, the measured median was 0.284 s for 8,532.5 queries. Adding a modeled 5 ms round-trip per query increased the median to 42.94 s, approximately 151 times the local value. Network latency slows the exploit but does not change its information content or success. Rate limiting can increase cost, yet it is not a substitute for eliminating the oracle.

B. Tamper Detection

TABLE V. One-Bit Mutation Results

Scheme	Mutations	Accepted	Rejected	Accept rate
CBC	1000	126	874	12.6%
CBC+HMAC	1000	0	1000	0.0%
GCM	1000	0	1000	0.0%

Unauthenticated CBC accepted 126 of 1,000 mutations. Many accepted changes affected the IV and produced syntactically valid JSON, while other mutations were divided between padding and payload failures. Acceptance does not imply a useful privilege escalation, but it directly demonstrates missing integrity. In contrast, both authenticated schemes rejected every mutation at the authentication boundary before returning session JSON.

C. Performance and Size

TABLE VI. EIGHT-BLOCK MEDIAN MICROBENCHMARK

Scheme	Encrypt (us/op)	Decrypt (us/op)
CBC	35.71	45.34
CBC+HMAC	40.29	53.22
GCM	17.38	25.93

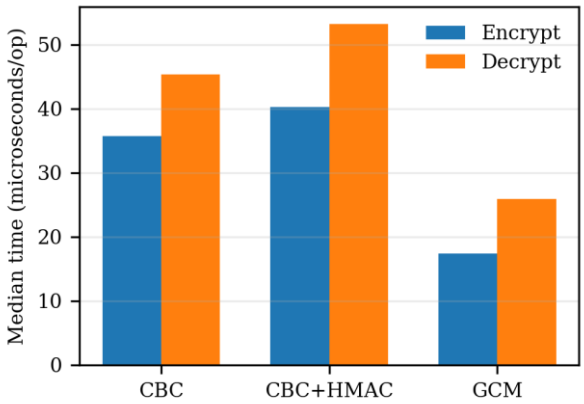

Fig. 6. Eight-block cryptographic operation time; lower is better.

For the eight-block payload, GCM encryption and decryption medians were 17.38 and 25.93 microseconds per operation. CBC measured 35.71 and 45.34 microseconds, while CBC-HMAC measured 40.29 and 53.22 microseconds. These values are implementation- and platform-specific; they show that authenticated encryption did not impose a performance penalty in this environment and should not be generalized as a universal ranking.

TABLE VII. REPRESENTATIVE TOKEN SIZES

Scheme	Blocks	Plaintext B	Raw token B	Encoded chars
CBC	2	25	48	71
CBC+HMAC	2	25	80	114
GCM	2	25	53	78
CBC	8	112	144	199
CBC+HMAC	8	112	176	242
GCM	8	112	140	194
CBC	16	240	272	370
CBC+HMAC	16	240	304	413
GCM	16	240	268	365

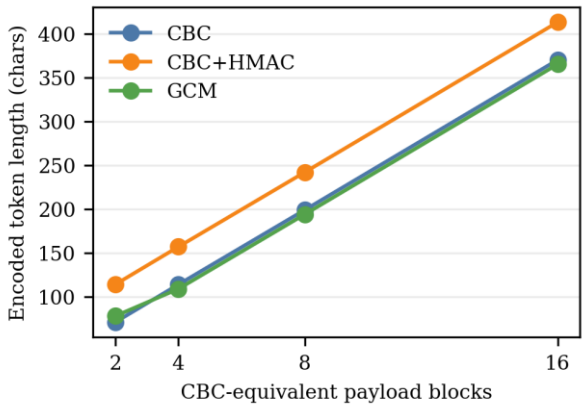

Fig. 7. Encoded token length across payload sizes.

CBC-HMAC is consistently 32 raw bytes larger than CBC because of its SHA-256 tag. GCM avoids PKCS#7 expansion and uses a 12-byte nonce plus 16-byte tag. For the sixteen-block-equivalent payload, the encoded lengths were 370 characters for CBC, 413 for CBC-HMAC, and 365 for GCM. Version prefixes are included in the encoded-token character count, while raw-token bytes count only the decoded body.

D. Security Interpretation

The experiments support two separate conclusions. First, the explicit error distinction is sufficient for complete plaintext recovery without the key. Second, removing that distinction is insufficient to claim token integrity: CBC remains malleable and the study does not establish timing indistinguishability. CBC-HMAC and GCM change the acceptance rule from 'decrypts and parses' to 'authenticates, then may be processed.' This ordering is the substantive mitigation.

VIII. LIMITATIONS

The oracle is intentionally deterministic and in-process, so real HTTP scheduling, TLS, proxies, jitter, and defensive throttling are absent. The 5 ms result is a transparent linear model rather than a wall-clock network experiment. The attack script uses an explicit Boolean padding signal and does not implement statistical timing classification. Consequently, CBC-unified is shown only to defeat this script's observable classification, not to be side-channel free.

Mutation acceptance measures syntactic/session acceptance under the prototype schema, not business impact. Benchmarks cover one library and processor and do not control CPU frequency or hardware acceleration. Nonce uniqueness is sampled rather than formally guaranteed, key rotation is designed but not deployed across multiple servers, and the prototype does not evaluate denial-of-service, replay, browser theft, or authorization logic. These limitations do not affect the demonstrated plaintext recovery or authenticated rejection results.

IX. CONCLUSIONS

A padding-validity response was enough to recover every tested AES-CBC session plaintext without learning the key.

Attack work scaled approximately with ciphertext blocks, and per-query delay dominated execution time without removing the vulnerability. A uniform error response removed the explicit signal used by the script but did not add integrity. CBC Encrypt-then-MAC and AES-GCM rejected all tested modifications before payload processing; GCM additionally avoided padding and produced smaller tokens than CBC-HMAC in every tested size.

The practical recommendation is a versioned migration to a standard AEAD implementation, with unique 96-bit GCM nonces, strict version dispatch, new keys, bounded legacy validation, and generic external errors. Systems that must retain CBC should authenticate version, IV, and ciphertext with an independent MAC and verify it before decryption. The broader lesson is that encrypted client-controlled state must never be treated as authentic merely because decryption succeeds.

SOURCE CODE REPOSITORY AT GITHUB

https://github.com/aldoyfa/padding-oracle-migration

ACKNOWLEDGMENT

The author expresses sincere gratitude to Dr. Ir. Rinaldi Munir, M.T. for his invaluable guidance, insightful lectures, and continuous support throughout this work. His expertise and dedication to teaching have greatly contributed to the author's understanding of cryptography and the development of this paper.

REFERENCES

[1] M. Dworkin, Recommendation for Block Cipher Modes of Operation: Methods and Techniques, NIST SP 800-38A, Dec. 2001. doi: 10.6028/NIST.SP.800-38A.

[2] M. Dworkin, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST SP 800-38D, Nov. 2007. doi: 10.6028/NIST.SP.800-38D.

[3] S. Vaudenay, Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS..., in EUROCRYPT 2002, LNCS 2332, pp. 534-546, 2002. doi: 10.1007/3-540-46035-7_35.

[4] J. Black and H. Urtubia, Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption, in 11th USENIX Security Symposium, pp. 327-338, 2002.

[5] R. Housley, Cryptographic Message Syntax (CMS), RFC 5652, Sep. 2009. doi: 10.17487/RFC5652.

[6] D. McGrew, An Interface and Algorithms for Authenticated Encryption, RFC 5116, Jan. 2008. doi: 10.17487/RFC5116.

[7] A. Barth, HTTP State Management Mechanism, RFC 6265, Apr. 2011. doi: 10.17487/RFC6265.

[8] P. Rogaway, Authenticated-Encryption with Associated-Data, in Proc. ACM CCS, pp. 98-107, 2002. doi: 10.1145/586110.586125.

[9] The cryptography developers, Authenticated Encryption documentation, cryptography.io, accessed Jun. 19, 2026. https://cryptography.io/en/latest/hazmat/primitives/aead/.

[10] N. J. AlFardan and K. G. Paterson, Lucky Thirteen: Breaking the TLS and DTLS Record Protocols, in 2013 IEEE Symposium on Security and Privacy, pp. 526-540, 2013. doi: 10.1109/SP.2013.42.

STATEMENT

I hereby declare that this paper I have written is my own original work; it is not an adaptation or translation of another person's work, nor is it plagiarism.

Bandung, 18 Juni 2026

Aldoy Fauzan Avanza
1822311