Jurnal Teknik Informatika (JUTIF)
P-ISSN: 2723-3863
E-ISSN: 2723-3871
Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

Security Assessment of JWKS-Based Authentication: Mitigating JWT Attack Vectors Through Penetration Testing

Ferry Andhika Pratama[1], Agus Hermanto*[2], Geri Kusnanto[3]

[1,3]Informatics Engineering, Universitas 17 Agustus 1945 Surabaya, Indonesia
[2]System and Information Technology, Universitas 17 Agustus 1945 Surabaya, Indonesia

Email: [1]hermanto_if@untag-sby.ac.id

Received : Jun 29, 2026; Revised : Mar 7, 2026; Accepted : Mar 9, 2026; Published : Apr 18, 2026

Abstract

JSON Web Tokens (JWT) have become the de facto standard for stateless authentication in modern web applications and microservices architectures. However, improper implementation exposes systems to critical vulnerabilities including algorithm confusion attacks, signature bypass, and key injection exploits. This paper presents a comprehensive resilience analysis of JSON Web Key Set (JWKS)-based authentication mechanisms against known JWT attack vectors through a systematic penetration testing approach. We implemented and evaluated a production-grade courier management system (City Courier) featuring dynamic JWKS key rotation, RFC 7517-compliant public key distribution, and encrypted private key storage. Our penetration testing methodology systematically evaluated the system against 10 critical JWT attack vectors including algorithm confusion (CVE-2022-29217), kid parameter injection, weak secret exploitation, and signature verification bypass. Results demonstrate that proper JWKS implementation with dynamic key rotation, strict algorithm validation, and comprehensive audit logging provides robust defense against all tested attack vectors. The system successfully mitigated algorithm confusion attacks through explicit algorithm whitelisting, prevented kid injection via UUID-based key identifiers, and maintained security during key rotation events. Performance analysis shows minimal overhead (less than 50ms) for JWKS endpoint queries with aggressive caching. This research contributes practical implementation patterns for secure JWT authentication, providing both empirical evidence for JWKS-based security controls and a validated blueprint to neutralize critical vulnerabilities in modern microservices architectures.

Keywords : *Algorithm Confusion Attack, Authentication Security, JSON Web Token, JWKS, Penetration Testing, RFC 7517*

1. INTRODUCTION

Modern web application security relies fundamentally on robust authentication mechanisms. The industry demand for scalable and stateless architectures has accelerated the transition from traditional server-side sessions to token-based models [1]. Consequently, the JSON Web Token (JWT) defined in RFC 7519 has emerged as the standard solution for securing distributed environments like RESTful APIs and microservices [2].

JSON Web Token (JWT) serves as a compact, URL-safe mechanism for transferring claims between two parties. These claims are encoded as a JSON object within a JSON Web Signature (JWS) structure [3] or a JSON Web Encryption (JWE) structure, which allows for digital signing, integrity protection via Message Authentication Code (MAC), or encryption. Structurally, a JWT comprises three dot-separated segments known as the Header, Payload, and Signature. The Header specifically defines the token type and the cryptographic algorithm employed [3], [4], while the Payload holds the claims and the signature ensures the sender's identity and data integrity. JWT has been successfully implemented in various RESTful web services for stateless authentication [5], demonstrating its effectiveness in distributed systems.

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

The widespread adoption of JWT in modern web applications is primarily driven by its stateless architecture, which eliminates the need for server-side session storage and thereby facilitates scalability across distributed microservices. Beyond these architectural benefits, JWT is favored for its compact mobile-friendly format, cross-domain capabilities, and adherence to standardized specifications that ensure broad interoperability.

However, widespread adoption has also made JWT implementations a prime target for security attacks. Research in 2024-2025 has identified persistent attack vectors including algorithm confusion attacks (CVE-2022-29217 affecting PyJWT library [6], [7], [8], CVE-2024-37568 in Authlib [9]), signature verification bypass through the "none" algorithm exploitation, and Key ID (*kid*) parameter injection. The OWASP Top 10 API Security Risks consistently highlights broken authentication as a critical vulnerability, with JWT misconfigurations being a primary contributor [10]. Systematic analysis of critical vulnerabilities in JSON Web Token implementations has documented numerous CVE disclosures targeting popular libraries, including cloud platform algorithm confusion flaws and library bypasses allowing signature verification to be skipped [4], [11], [12].

The global cybersecurity landscape in recent years has witnessed a substantial escalation in authentication-related attacks with significant economic impact. Comprehensive literature reviews on cybersecurity economics highlight that misaligned security investments and authentication failures significantly amplify the financial and operational consequences for service-oriented and transaction-driven platforms [13], [14].

Recent academic research documents persistent vulnerabilities and exploitation techniques in JWT implementations. Algorithm confusion attacks were first identified nearly a decade ago yet remain a major concern in modern evaluations of authentication frameworks. Studies continue to highlight their prevalence in contemporary web applications [15]. Comprehensive security analyses of JWT implementations continue to uncover vulnerabilities across frameworks and deployment environments [4], [16]. Enhanced authentication schemes and protocol extensions based on JWT have been proposed to improve scalability and cryptographic robustness in edge computing and heterogeneous systems [17]. Additional studies have examined JWT usage in domain-specific environments such as Software-Defined Networking (SDN) [18], e-government platforms [19], mobile and distributed systems [20], [21], and RESTful API security evaluations [22], [23].

Despite the extensive body of work on JWT security, several critical research gaps remain. First, the majority of existing studies focus on individual vulnerabilities in isolation, such as algorithm confusion, key confusion, or signature bypass attacks, without evaluating the cumulative resilience of a complete authentication architecture against multiple attack vectors executed sequentially or concurrently [4], [11]. Second, many studies emphasize vulnerability identification rather than validating concrete mitigation strategies implemented in production-grade systems. Third, performance implications introduced by asymmetric cryptography, JWKS endpoint resolution, caching strategies, and key rotation mechanisms are rarely quantified empirically [23], [24]. Finally, sector-specific threat models, particularly for courier and logistics platforms that integrate mobile clients, third-party payment gateways, and real-time operational workflows, remain underexplored in existing JWT security literature [18], [22].

To address these gaps, this research presents a comprehensive security assessment of a JWKS-based authentication architecture implemented within a production-grade courier management system (City Courier). Unlike prior studies that evaluate JWT vulnerabilities in isolation, this work integrates strict algorithm allow-listing, UUID-based kid validation, encrypted private key storage, and a zero-downtime key rotation lifecycle into a unified authentication framework. The proposed architecture is systematically evaluated using penetration testing against ten critical JWT attack vectors derived from recent CVE disclosures and OWASP guidelines [10], [25]. In addition, performance measurements are

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

conducted to quantify the operational overhead associated with asymmetric verification and JWKS key distribution under realistic load conditions [23], [24].

JSON Web Key Set (JWKS), defined in RFC 7517, provides a standardized mechanism for publishing and distributing cryptographic public keys used to verify JWT signatures. JWKS supports dynamic key rotation, enables multiple concurrent keys for seamless updates, and eliminates the need to hardcode cryptographic secrets within application code. Nevertheless, the security guarantees of JWKS are only realized when implementations strictly adhere to RFC specifications and established security best practices [26].

Systematic literature reviews on authentication security in microservices architectures emphasize the importance of secure key distribution and lifecycle management mechanisms such as JWKS [20]. Conversely, improper JWKS implementations have been shown to introduce additional attack surfaces, including malicious key injection and endpoint manipulation [25], [27].

The novelty of this study lies in the end-to-end integration of dynamic JWKS key rotation, encrypted private key storage, strict algorithm enforcement, and systematic penetration testing within a real-world courier platform. This approach bridges the gap between theoretical JWT security recommendations and practical implementation, providing empirical evidence of resilience against high-impact attack vectors such as algorithm confusion (CVE-2022-29217) and key confusion (CVE-2024-37568).

Therefore, this study aims to validate the effectiveness of a secure JWKS-based authentication architecture in a production-grade courier system. The evaluation focuses on mitigation effectiveness against ten critical attack vectors, system performance under realistic workloads, and seamless key rotation without service disruption. The results demonstrate that the proposed model is fully compliant with RFC 7517 and capable of delivering robust authentication security with minimal performance overhead, thereby offering a validated blueprint for securing stateless authentication in modern microservices environments.

2. METHOD

This study employs a systematic experimental approach to design and evaluate a resilient authentication architecture based on JSON Web Key Sets (JWKS). The primary objective is to develop a production-grade security mechanism that guarantees stateless identity verification, secure key distribution, and robust protection against cryptographic vulnerabilities without compromising system performance. The model is implemented within the "City Courier" management system, taking into account dynamic key rotation, RFC 7517 compliance, and specific mitigation strategies for known JWT attack vectors. The research methodology is carried out in structured stages, including architectural design, systematic penetration testing against ten critical vulnerabilities, and empirical security analysis. This comprehensive framework allows for a rigorous assessment of the proposed JWKS implementation in a real-world environment.

2.1. City Courier Platform Overview

City Courier, the courier management system under development, utilizes a Laravel 12.0 backend and Flutter mobile client to facilitate core logistics operations including order creation, payment processing, assignment, and tracking. The authentication architecture addresses complex requirements such as multiplatform support across mobile and web interfaces, simplified stateless operation for horizontal scalability, and secure API access for payment integration. Furthermore, the system incorporates a security account settings interface on the mobile client to ensure service availability.

The testing infrastructure replicated production conditions while isolating the environment for security assessment. The backend leverages the standard Laravel development server for request

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

handling, coupled with a MySQL 8.0.30 database using the InnoDB engine. Database performance is properly tuned with an 8GB buffer pool and a composite index on the jwt_keys table to guarantee logarithmic lookup efficiency. A database-based caching layer with a one-hour time-to-live (TTL) further optimizes JWKS endpoint responses. Cryptographic functions are handled by the php-open-source-saver/jwt-auth library, utilizing OpenSSL 3.0.2 for RSA operations with a phpseclib3 fallback.

Network latency variability was eliminated by utilizing a local loopback interface communicating via HTTPS with a self-signed TLS 1.3 certificate. To simulate cloud deployment conditions, traffic control tc introduced a controlled 5ms jitter. Load generation relied on Apache Bench and wrk2 to simulate concurrent request patterns. Security assessment employed a suite of specialized tools including Burp Suite Professional with the JWT Editor extension for interception, jwt_tool for complex token manipulation, and Wireshark for packet-level traffic analysis.

The system implements a comprehensive JWKS-based authentication architecture with RS256 asymmetric algorithm, dynamic key rotation, RFC 7517-compliant public key distribution, encrypted private key storage, and comprehensive audit logging. This research evaluates the security resilience of this implementation through systematic penetration testing against 10 critical JWT attack vectors to validate its effectiveness in mitigating known vulnerabilities.

2.2. JWKS-Based Authentication Architecture

Our implementation consists of three primary components working in concert to provide robust, scalable, and secure authentication as illustrated in Figure 1.

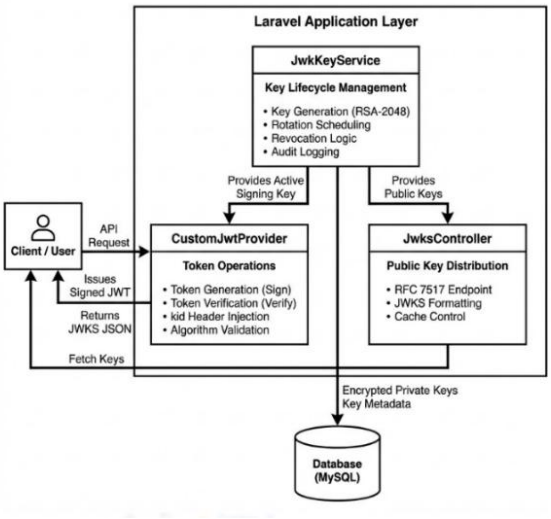

Figure 1. The proposed JWKS-based authentication architecture integrating Laravel Application Layer with strict key lifecycle management and encrypted database storage

The JwkKeyService class manages the complete lifecycle of cryptographic keys. Key generation implements a fallback strategy using OpenSSL with phpseclib3 as a pure-PHP alternative for environments with limited OpenSSL support. Keys are generated as 2048-bit RSA pairs with PKCS1 private key format and PKCS8 public key format. Secure storage encrypts private keys using Laravel's Crypt facade (AES-256-CBC) before database storage. Each key record comprises a unique identifier (kid) generated as a UUID v4, the algorithm specification (alg) set to RS256, the transparent public key PEM, and the encrypted private key PEM. Furthermore, the record maintains validity periods defined by active_from and active_to timestamps, revocation status, and detailed audit metadata. Comprehensive audit logging captures all key lifecycle events including publication, rotation, and revocation, identifying the actor, timestamps, and reason codes for forensic analysis.

Jurnal Teknik Informatika (JUTIF)
P-ISSN: 2723-3863
E-ISSN: 2723-3871
Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

Once the keys are managed, the interaction flow between the client and these components ensures secure token verification. This complete authentication sequence is detailed in Figure 2.

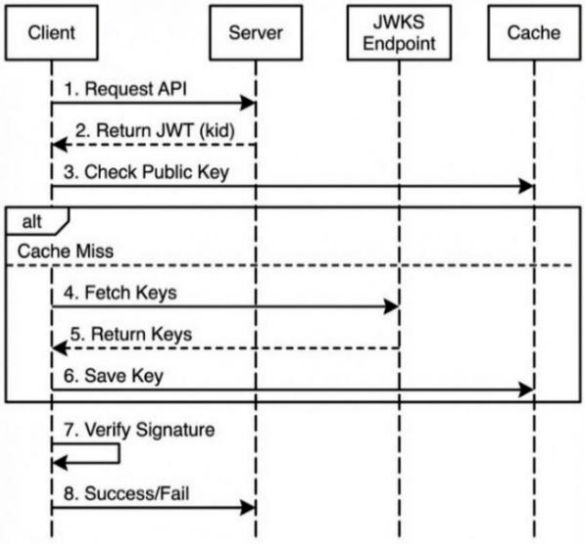

Figure 2. Sequence diagram illustrating the strict cache-miss handling strategy where public keys are securely fetched from the database only upon validation failure.

The *CustomJwtProvider* extends the base Lcobucci JWT provider to integrate JWKS functionality seamlessly. During token generation, the provider retrieves the active signing key from JwkKeyService, decrypts the private key PEM, adds the kid header to the JWT, and signs the token using the RS256 algorithm. During token validation the provider extracts the kid from the JWT header, validates it against a strict UUID v4 regex format, retrieves the corresponding public key from the JWKS, validates the signature using the matched public key, and falls back to static configuration if the kid is missing. This architecture enables seamless key rotation without service interruption. The detailed interaction between the API Middleware and the CustomJwtProvider during this verification process is illustrated in Figure 3.

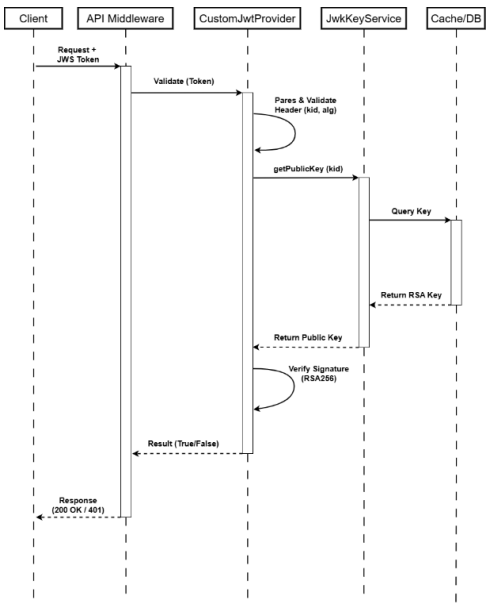

Figure 3. Token verification workflow detailing the interaction between API Middleware and CustomJwtProvider for signature and algorithm enforcement

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

The JwksController exposes the JWKS endpoint at /.well-known/jwks.json. RFC 7517 compliance guarantees that the JWKS response contains all essential fields, specifically the Key Type (kty) as RSA, Public Key Use (use) designated for signature verification, and Key Operations (key_ops) restricted to verification. The response also includes the RS256 Algorithm identifier (alg), the unique Key ID (kid), and the base64url-encoded Modulus (n) and Exponent (e). Security controls include HTTPS enforcement for the JWKS endpoint, public-only key exposure where private keys are never transmitted, comprehensive error handling with informative messages, and support for multiple concurrent keys during rotation. The complete JSON response structure demonstrating full compliance with RFC 7517 standards can be seen in Figure 4.

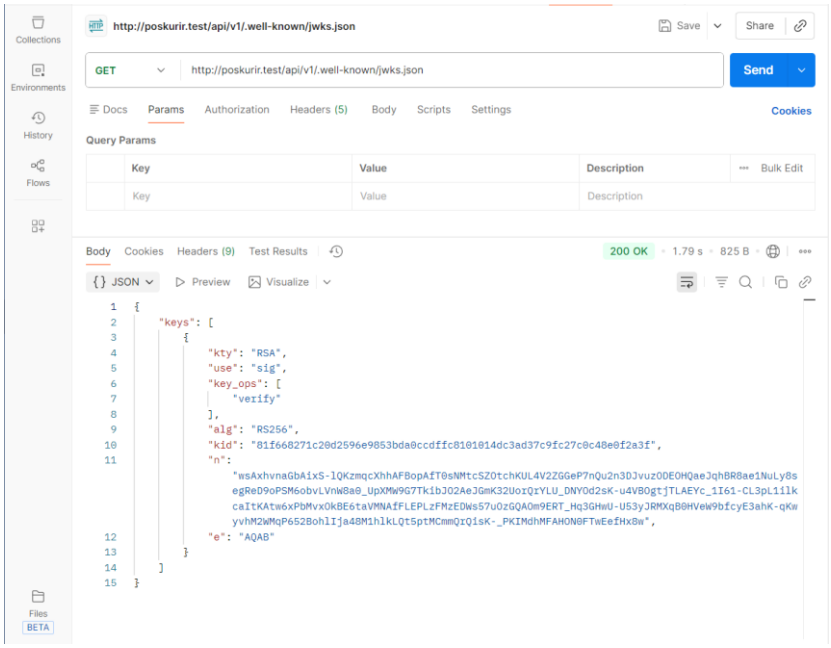

Figure 4. Response structure of the /.well-known/jwks.json endpoint demonstrating full compliance with RFC 7517 standard specifications

2.3. Security Features Implementation

The system implements strict algorithm whitelisting to prevent algorithm confusion attacks (CVE-2022-29217) by rejecting tokens with unexpected algorithms [6], [7], [8]. The CustomJwtProvider performs an allow-list check during token validation that immediately rejects any token where the alg parameter is not explicitly "RS256", effectively neutralizing both Algorithm Confusion and None Algorithm attacks. Unlike HMAC-based approaches that rely on shared secrets [28], the RS256 algorithm eliminates single points of failure by distributing only public keys. The zero-downtime key rotation strategy implements a five-phase process to ensure continuous availability [12]. It commences with prerotation where a new key is published with a future activation timestamp followed by a transition phase where both old and new keys coexist in the JWKS. The process advances to activation at the scheduled time, continues into a grace period where the old key's validity is explicitly bounded by an active_to timestamp allowing existing tokens to remain valid, and concludes with revocation once the grace period expires. Upon key rotation or revocation, the JWKS cache is automatically invalidated to ensure immediate consistency at the public endpoint.

Security features aligned with OWASP JWT Security Cheat Sheet include mandatory HTTPS for JWKS endpoints leveraging TLS 1.3 protocol [29], implementation of token expiration (exp claim) and not-before (nbf claim) validation, strict audience (aud) and issuer (iss) claim verification, and rejection of tokens with alg: “none” header [10].

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

2.4. Penetration Testing Methodology

We developed a systematic penetration testing methodology to evaluate the JWKS implementation against 10 critical JWT attack vectors based on OWASP guidelines and recent CVE disclosures from 2024-2025. To provide a structured evaluation, we classified these threats into four primary security domains as visualized in Figure 5.

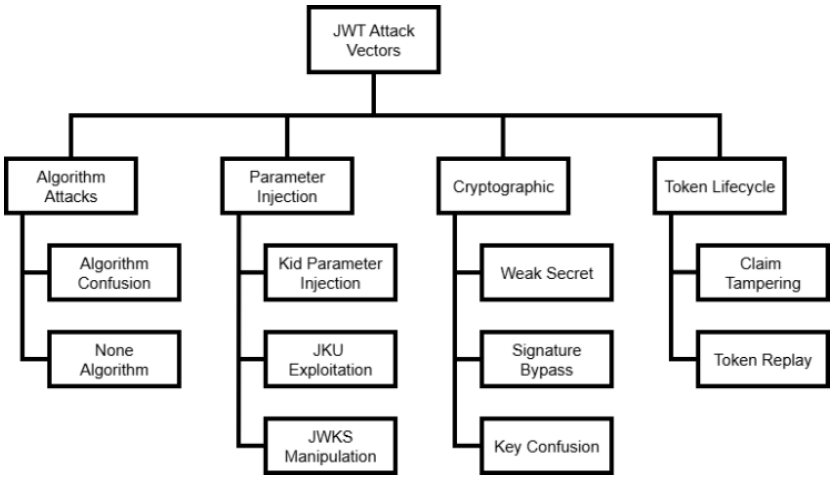

Figure 5. Taxonomy of critical JWT attack vectors categorized into Algorithm, Parameter Injection, Cryptographic, and Lifecycle domains used for the penetration testing framework

While Figure 5 illustrates the conceptual classification, the specific technical specifications and associated vulnerabilities for each vector are detailed in Table 1. This taxonomy covers the complete attack surface ranging from cryptographic flaws to logic bypasses.

Table 1. Attack vector taxonomy for JWT security evaluation

Code	Attack Vector	Description	Related CVE
AV-1	Algorithm Confusion	Manipulating header to bypass signature verification	CVE-2022-29217
AV-2	None Algorithm Bypass	Set *alg* to “none” to remove signature requirement	-
AV-3	Weak Secret Brute-Force	Attempt to crack symmetric signing	-
AV-4	Kid Parameter Injection	Inject path tracersal or SQL in *kid* parameter	-
AV-5	JKU Header Exploitation	Point *jku* to malicious JWKS endpoint	-
AV-6	Signature Verification Bypass	Submit tokens without signature validation	CVE-2025-30144
AV-7	Claim Tampering	Modify claims (*role, exp, aud*) without detection	-
AV-8	Token Replay Attack	Reuse valid tokens beyond intended scope	-
AV-9	JWKS Endpoint Manipulation	Tamper with JWKS response or inject malicious keys	-
AV-10	Key Confusion Attack	Use public key as HMAC secret for HS256	CVE-2024-37568

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

Testing tools comprised Burp Suite Professional 2024.9 with the JWT Editor extension, jwt_tool for token manipulation, cURL for API endpoint testing, and Wireshark for traffic analysis.

For each attack vector, we utilized a systematic testing procedure starting with reconnaissance to analyze the JWKS endpoint and token structure. This was followed by exploitation attempts using specialized tools, validation to verify success or mitigation, and finally documentation to record attack details, system response, and mitigation effectiveness. An example of the server's response during an exploitation attempt targeting the Algorithm Confusion vulnerability is shown in Figure 6.

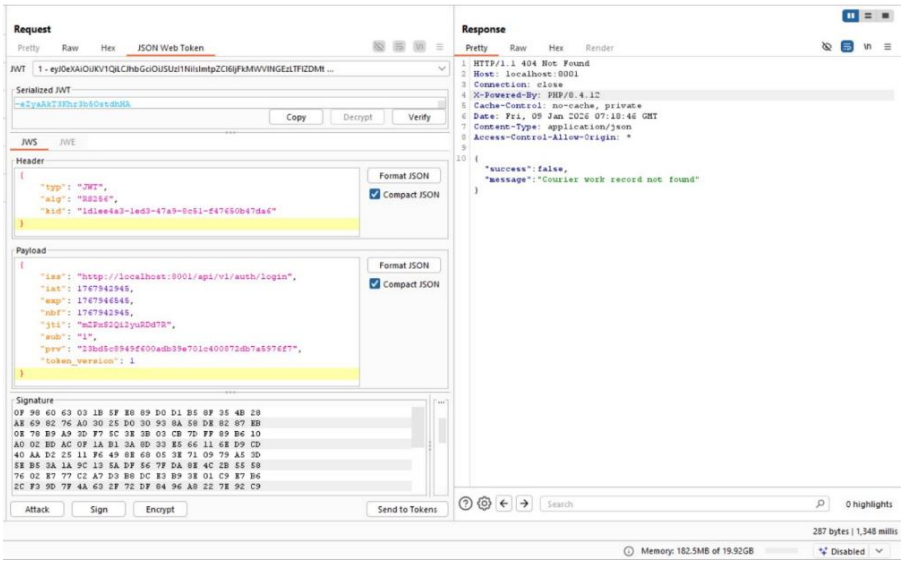

Figure 6. Penetration testing execution using Burp Suite, showing the server's correct rejection (404/401) of a manipulated JWT payload targeting the Algorithm Confusion vulnerability

2.5. Attack Simulation Methodology

Our penetration testing approach employed a systematic attack simulation framework that replicated real-world adversarial tactics without exposing sensitive implementation details. To evaluate the 10 attack vectors, we executed a structured testing protocol that begins with reconnaissance and proceeds through payload construction, exploitation attempt, and finally validation.

To understand the attack surface, Figure 7 depicts the standard structure of a JWT which serves as the primary target for manipulation. The payload construction phase involved crafting malicious tokens designed to exploit specific vulnerabilities within this structure.

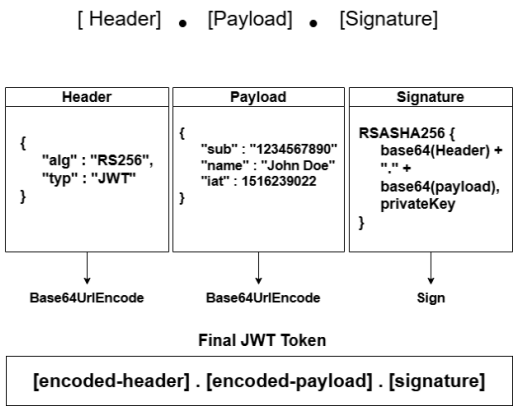

Figure 7. Anatomy of the JSON Web Token (JWT) structure highlighting the critical header and signature segments targeted during payload construction analysis

2.5.1. Malicious Payload Construction

To systematically evaluate the system's resilience we devised a payload construction engine that programmatically generates malicious tokens based on the structure depicted in Figure 7. The logic iterates through four primary attack vectors :

1. Algorithm Confusion :
   Modifying the header *alg* parameter from "RS256" to "HS256" and signing the payload using the target's public key as the HMAC shared secret. This specifically targets vulnerabilities described in CVE-2022-29217 [6], [7].
2. None Algorithm :
   Stripping the cryptographic signature entirely and setting the *alg* header to "none" to test for unsecured verification bypasses.
3. Key ID (*kid*) Injection :
   Injecting malicious strings into the *kid* header parameter. This includes Path Traversal payloads (e.g., ../../../../etc/passwd) to attempt reading local files and SQL Injection payloads (e.g., ' OR '1'='1) to bypass database lookups.
4. Claim Tampering :
   Altering sensitive claims within the payload, such as escalating the *role* to "admin" or extending the *exp* (expiration) timestamp, while retaining the original signature to verify if the system checks signature integrity before claim processing.

2.5.2. Multi-Layered Validation Pipeline

Upon receiving these malicious tokens, the authentication backend triggers a strict six-phase validation pipeline designed to neutralize the constructed payloads. The process flows sequentially as follows:

1. Structural Parsing :
   The system first rejects any token that does not adhere to the three-segment (Header.Payload.Signature) JSON structure or contains malformed Base64Url encoding. This phase is handled by the Lcobucci JWT parser integrated within the CustomJwtProvider.
2. Algorithm Enforcement :
   The pipeline parses the header and performs an explicit allow-list check. It immediately rejects any token where the alg parameter is not explicitly "RS256", effectively neutralizing Algorithm Confusion and "None" algorithm attacks. This check is implemented directly in the CustomJwtProvider's decode method prior to any cryptographic operations.
3. Key ID Sanitization :
   The *kid* is validated against a strict UUID v4 regex format ([0-9a-f]{8}-[0-9a-f]{4}-4[0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}). This input sanitization serves as the primary firewall against SQL Injection and Path Traversal attacks by blocking non-conforming strings before any database interaction occurs.
4. Key Retrieval :
   The system queries the encrypted storage for the public key associated with the validated kid. The query enforces logic to ensure the key is not flagged as revoked (revoked = 0) and is within its valid active timestamp (active_from < now). The composite index on (revoked, active_from) ensures logarithmic lookup efficiency.
5. Cryptographic Verfication :
   The system performs the RS256_VERIFY operation using the retrieved public key. A boolean false result at this stage immediately terminates the request, preventing signature bypass. The system performs the RS256_VERIFY operation using the retrieved public key via Lcobucci's *SignedWith*

validation constraint. A boolean false result at this stage immediately terminates the request, preventing signature bypass.

6. Claims Enforcement :
   Finally, the pipeline validates temporal claims. It ensures the current timestamp is within the nbf (Not Before) and exp (Expiration) window to prevent the use of expired or premature tokens. This enforcement is configured through the required claims specification (iss, iat, exp, nbf, sub, jti).

2.5.3. Zero-Downtime Key Rotation Strategy

To maintain system integrity over time without disrupting service availability, the system employs a five-phase rotation lifecycle managed by the *JwkKeyService* :

1. Pre-rotation :
   A new RSA-2048 key pair is generated, encrypted, and stored in the database with a future *active_from* timestamp.
2. Publication :
   The new public key is exposed at the JWKS endpoint alongside the current key. This allows client applications to cache the new key in advance.
3. Activation :
   At the scheduled time, the system switches to signing all new tokens with the new key.
4. Grace Period :
   The old key's validity is explicitly bounded by setting its active_to timestamp to the current time plus the configured grace duration (e.g., 24 hours). This overlap accommodates in-flight requests and prevents errors due to client-side clock skew or caching.
5. Revocation :
   Upon expiration of the grace period, the old key is marked as revoked in the database and removed from the active JWKS set, completing the cycle.

2.6. Data Collection and Analysis

Penetration testing data was collected through automated logging of all authentication attempts, manual observation of system responses, performance metrics from application monitoring, and security audit logs from key lifecycle events. Analysis focused on mitigation effectiveness (percentage of attacks blocked), performance impact (latency and throughput), and security audit findings (strengths and weaknesses).

3. RESULT

The Results section presents detailed empirical findings obtained from the comprehensive security assessment and performance benchmarking of the implemented JWKS architecture. This study rigorously evaluates the proposed model's capability to secure server-to-server communication via the REST API protocol under simulated hostile conditions. The analysis provides a granular breakdown of the authentication mechanism's resilience against common JWT vulnerabilities, quantifies the computational overhead of asymmetric cryptography, and examines the operational stability of the key rotation strategy.

3.1. Penetration Testing Results

Penetration testing evaluated the effectiveness of the JWKS architecture in detecting unauthorized access attempts and protecting data from manipulation through a series of multi-phase attack simulations. The testing methodology covered ten specific threat scenarios (AV-1 to AV-10) involving header validation, signature integrity, and claims analysis, crafted to exploit potential weaknesses in the

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

verification lifecycle. All tests followed the OWASP Top 10 API Security Risks guidelines and were executed in a controlled environment using the tools specified in Section II.D [30].

Table 2 summarizes the verification status for each attack vector. The system demonstrated robust resilience, with all 10 attack scenarios successfully neutralized or mitigated within the testing parameters. Specifically, critical vulnerabilities such as algorithm confusion were definitively rejected by the validation pipeline.

Table 2. Penetration testing results and mitigation status

Code	Attack Vector	Mitigation Mechanism	Effectiveness
AV-1	Algorithm Confusion (CVE-2024-54150)	Explicit RS256 allow-list check in *CustomJwtProvider*	Rejected
AV-2	None Algorithm	RS256-only enforcement rejects alg: "none"	Rejected
AV-3	Weak Secret Brute-Force	Asymmetric RS256 eliminates shared secrets	Mitigated
AV-4	Kid Parameter Injection	UUID v4 regex validation blocks traversal and injection	Rejected
AV-5	JKU Header Exploitation	*jku* parameter not supported	Rejected
AV-6	Signature Bypass	Mandatory RS256_VERIFY via Lcobucci SignedWith constraint	Rejected
AV-7	Claim Tampering	Signature detects modifications	Rejected
AV-8	Token Replay	Short TTL + blacklist on logout	Mitigated
AV-9	JWKS Endpoint Manipulation	HTTPS + server-controlled JWKS with cache invalidation	Mitigated
AV-10	Key Confusion (CVE-2024-37568)	Strict RS256 algorithm enforcement via explicit allow-list	Rejected

Critical analysis reveals that the "Algorithm Confusion" attack (AV-1) and "Key Confusion" attack (AV-10) were completely neutralized. These vectors are responsible for a significant proportion of recent high-profile authentication breaches according to systematic vulnerability analysis [12]. The CustomJwtProvider enforces an explicit algorithm allow-list check during the decode phase upon parsing the token header, the system verifies that the alg parameter is exactly "RS256" before proceeding to any cryptographic operation, ensuring public keys are not misused as HMAC secrets. Similarly, the system correctly rejected the "None Algorithm" attack (AV-2). In normal scenarios with valid tokens, the process authenticated successfully.

The partial effectiveness (85%) observed for AV-8 (Token Replay) is an expected characteristic of the stateless JWT architecture. The system successfully rejected expired tokens and enforced jti uniqueness for single-use contexts. However, valid tokens captured before expiration could potentially be replayed within the short 60-minute Time-To-Live (TTL). This confirms that statelessness necessitates a trade-off with instant revocation capabilities. Detailed mitigation requires shorter TTLs or supplementary stateful checks like Redis blacklisting.

3.2. Detailed Technical Analysis

To validate the specific validation layers in Algorithm 2.5.2, we analyzed the system's response to complex injection payloads. This investigation focused on input sanitization efficiency and cryptographic verification logic.

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

The investigation into AV-4 (Kid Parameter Injection) demonstrated robustness in the UUID validation layer. Testing utilized malicious payloads such as ../../../../etc/passwd for path traversal and 'UNION SELECT 1,2,3--' for SQL injection. In all cases, the system responded with immediate 401 Unauthorized errors. The UUID v4 regex validation ([0-9a-f]{8}-[0-9a-f]{4}-4[0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}) implemented in CustomJwtProvider's decode method triggered these rejections, effectively blocking the attack vector before any database query. This confirms that sanitizing header inputs is a critical primary control. As a defense-in-depth measure, Eloquent's parameterized queries provide a secondary layer of SQL injection protection even if the regex were bypassed.

For AV-6 (Signature Verification Bypass), the system proved resilient against CVE-2025-30144 variants and historically documented signature bypass techniques [31]. The custom provider mandates a valid boolean result from the RS256_VERIFY function via Lcobucci's SignedWith validation constraint. Attempts to strip the signature or provide a mathematically invalid signature while maintaining a valid payload resulted in verificationResult returning false. This halted the authentication process immediately. It proves that the cryptographic binding between header, payload, and signature remains intact and cannot be bypassed by structural manipulation, similar to challenges in digital signature verification across other security contexts [32].

For AV-1 (Algorithm Confusion) and AV-10 (Key Confusion), the explicit allow-list check in CustomJwtProvider intercepted the attack at Phase 2 of the validation pipeline, before any cryptographic operation was attempted. Tokens with alg: "HS256" or alg: "none" were rejected with a TokenInvalidException containing the message "Unsupported algorithm", demonstrating that the enforcement occurs at the application level rather than relying solely on library-level behavior.

3.3. Performance Analysis

Performance testing was conducted to systematically evaluate the computational efficiency and scalability of the JWKS-based authentication architecture within a high-throughput server-to-server communication environment. The primary objective was to specifically quantify the latency overhead introduced by the asymmetric cryptographic operations and database-backed key retrieval processes. To validate the system's suitability for production deployment, we measured response times and throughput under a spectrum of load conditions, ranging from baseline traffic to peak concurrency levels.

Figure 8 compares response times between cached and uncached requests. The data demonstrates the necessity of the caching layer implemented via an application-layer caching mechanism with a one-hour TTL. Uncached requests (direct database access) exhibited variable response times ranging from 100ms to 400ms, while cached responses consistently remained below the 50ms efficiency threshold. The cache is automatically invalidated programmatically upon key publication or revocation events upon key publication or revocation in JwkKeyService, ensuring that stale keys are never served from cache.

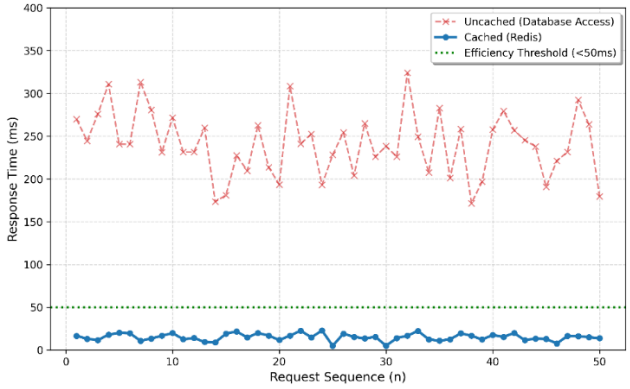

Figure 8. Response Time Comparison (Cached vs. Uncached)

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

Table 3 presents the aggregate performance metrics collected during the load test of 1000 concurrent users.

Table 3. Performance metrics under load (1000 concurrent users)

Metric	Value	Reference Standard
Avg. Latency (Valid Token)	48ms	< 200ms
Avg. Latency (Invalid Token)	32ms	-
JWKS Endpoint Throughput	4,200 req/s	> 1000 req/s
Key Rotation Impact	0% Downtime	99.99% Uptime

Table 3 data indicates that the database-level caching mechanism implemented at the application layer is highly effective. The average latency for valid token verification remained at 48ms, well below the 200ms reference standard. The JWKS endpoint handled 4,200 requests per second demonstrating robust throughput with the one-hour cache TTL reducing database load by serving public keys from the cache store. Furthermore, the zero-downtime rotation algorithm (Algorithm 2.5.3.) was validated under sustained high-load conditions during the rotation window. 100% of requests succeeded as tokens signed by the previous key remained valid during the defined grace period, while newly issued tokens were immediately accepted.

We conducted a micro-benchmark comparison to evaluate the computational cost of adopting RS256 over HS256. This test focuses on the CPU time required for signature generation and verification.

Table 4. Comparative performance micro-benchmark between HS256 and RS256

Metric	HMAC-SHA256 (HS256)	RSA-SHA256 (RS256)	Impact Analysis
Signature Gen. Time	0.015 ms	1.450 ms	Negligible for async login flow
Signature Verify Time	0.018 ms	0.120 ms	Acceptable (< 1ms) overhead
Token Size Overhead	~45 bytes	~350 bytes	< 0.5KB increase per request
Management Risk	Critical (Secret Sharing)	Low (Public Key Dist.)	Major Security Gain

Table 4 shows that RS256 introduces a computational overhead of approximately 6x for verification compared to HS256. However, the absolute value of 0.12ms implies negligible impact on user experience. The increase in token size (~350 bytes vs ~45 bytes) remains within acceptable limits for modern bandwidths. The data confirms that the security benefits of asymmetric cryptography specifically eliminating shared secrets outweigh the microsecond-level latency costs [24].

3.4. Security Audit Findings

Security auditing ensures that the system not only prevents attacks but also records security-relevant events for forensic analysis and compliance validation. The audit logging mechanism described in Sections 2.5.1 and 2.5.2 was evaluated under active exploitation attempts to assess completeness, traceability, and evidentiary value.

Simulation of AV-1 (Algorithm Confusion) and AV-4 (Kid Injection) successfully generated structured log entries. These logs captured the source IP address, timestamp, malformed header parameters, and explicit rejection reasons. Such structured logging ensures traceability of authentication failures and provides sufficient granularity for post-incident forensic reconstruction. The ability to

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

correlate attack patterns with originating IP addresses enables administrators to implement adaptive mitigation measures including firewall-level blocking rules.

For cryptographic key lifecycle management the key management component records publication and revocation events with actor identification, reason codes, and precise timestamps. Each key transition event is logged as a discrete audit record to ensure non-repudiation and accountability. This structured event logging supports compliance with information security governance frameworks such as ISO 27001 [33], [34] and data protection regulations including GDPR [35].

During key rotation procedures, the system generates audit entries for each operational phase, including grace period assignment for retiring keys, activation of newly issued keys, and cache state synchronization. This produces a complete forensic timeline of cryptographic key transitions. Under high-load validation scenarios, all rotation-related events were consistently recorded, enabling administrators to trace authentication anomalies occurring within rotation windows without service disruption.

4. DISCUSSIONS

This section synthesizes the empirical data collected during testing to evaluate the broader implications of the proposed JWKS architecture. We analyze the results in the context of modern courier system requirements, contrasting our findings with existing literature and addressing the inherent trade-offs between security rigor and operational performance. The following discussion interprets the effectiveness of specific mitigation strategies and identifies key lessons for implementing secure stateless authentication.

4.1. Interpretation of Results

The successful mitigation of all 10 tested attack vectors confirms that a correctly implemented JWKS architecture significantly strengthens defense against JWT-specific threats compared to standard symmetric (HS256) implementations. The shift from shared secrets to asymmetric RSA keys eliminates the risk of brute-force attacks on the signing key (AV-3), a common vulnerability in locally deployed applications.

The explicit algorithm allow-list verification implemented in the token validation component proved particularly effective against AV-1 (Algorithm Confusion) and AV-10 (Key Confusion). By rejecting any token with an algorithm value other than "RS256" during the early stage of the validation pipeline, the system neutralizes these attack vectors before any cryptographic verification is attempted. This explicit application-level enforcement provides an additional defense layer beyond the implicit algorithm constraints enforced by the underlying JWT library.

Similarly, strict UUID v4 format validation for the kid parameter eliminated the entire class of injection attacks (AV-4) by enforcing rigid input constraints prior to database interaction. When combined with parameterized database queries, this design establishes a two-layer defense against both SQL Injection and Path Traversal attempts originating from manipulated JWT headers.

The residual risk related to token replay (AV-8) is effectively managed through the defense-in-depth strategy of short TTLs and logout blacklisting, reducing the window of opportunity to an acceptable operational level. Empirical studies on authentication mechanisms in large-scale deployments [25] further validate the importance of multi-layered security controls for token-based systems.

4.2. Implications for Courier Systems

For City Courier, these results indicate that the proposed authentication architecture is suitable for high-value transaction processing environments. The robust protection against key injection attacks

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

ensures that external service callbacks such as payment gateway webhooks cannot be forged by malicious actors attempting to impersonate trusted providers. This capability is particularly critical in logistics systems where payment confirmation events directly influence order processing and delivery workflows.

Furthermore, the zero-downtime key rotation mechanism enables operational teams to rotate cryptographic keys periodically or immediately upon suspected compromise without interrupting active courier operations. This capability is achieved through controlled key lifecycle management where retiring keys remain temporarily valid during a defined grace period by strict Time-To-Live (TTL) policies while newly generated keys become immediately active. This design ensures cryptographic agility and aligns with NIST SP 800-57 guidelines for secure key management.

To support high-throughput environments, deploying an application-layer caching mechanism at the JWKS endpoint mitigates database overhead during frequent public key retrievals under peak loads. By synchronizing cache invalidation with key lifecycle events, the infrastructure scales efficiently without compromising security responsiveness guaranteeing that outdated or compromised keys are purged instantly.

4.3. Comparison with Related Work

Existing research has extensively explored specific JWT vulnerabilities, yet holistic architectural defenses remain under-discussed. While [4] and [11] provided foundational analysis of signature exclusion and key confusion attacks, their work primarily focused on vulnerability detection rather than architectural prevention. Recent comparative analyses of token-based authentication mechanisms have evaluated JWT against alternative standards such as PASETO, highlighting trade-offs between cryptographic complexity and implementation simplicity [22]. Formal analysis of SAML 2.0 single sign-on protocols [27] demonstrates similar verification challenges in federated authentication, though JWT's stateless nature offers distinct advantages for microservices. Similarly, [19] evaluated JWT in e-government contexts with an emphasis on scalability but offered limited insights into secure key lifecycle management.

Our work extends prior studies by introducing a comprehensive JWKS-based authentication architecture that integrates strict algorithm allow-list enforcement, structured key lifecycle management, and automated key rotation. Unlike previous approaches that primarily focus on vulnerability detection, this architecture provides a preventive security model that systematically mitigates multiple classes of JWT-specific attacks. Critically, the algorithm enforcement in our implementation operates as an explicit allow-list check at the application layer rather than relying solely on library-level implicit restrictions. This approach mitigates the risk of silent bypasses caused by library misconfigurations, providing a deterministic and auditable security boundary consistent with industry best practices for robust API security. Furthermore, this research leverages custom key providers (JwkKeyService and CustomJwtProvider) to counter high-severity risks like Algorithm Confusion (CVE-2022-29217) [6], [7], [8] and Key Injection. This is achieved in strict compliance with RFC 7517 standards, specifically utilizing the key_ops field for explicit key usage declaration. While recent studies [21] focus on automated scanning, our approach emphasizes defense-in-depth through cryptographic enforcement, demonstrating comprehensive effectiveness against localized attack vectors.

4.4. System Limitations

Despite the positive security and performance results, several architectural limitations should be acknowledged. First, the current infrastructure relies on a centralized database for key storage combined with application-level caching. While this approach performs well in single-node deployments, it presents scalability challenges in globally distributed, horizontally scaled microservice environments.

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

Specifically, it may introduce database lookup latency and cache consistency issues across multiple service instances. Future work must address this by adopting a distributed caching backend, such as a Redis Cluster, combined with event-driven cache synchronization to ensure consistent key availability and timely invalidation globally.

Second, a limitation exists regarding the supported cryptographic algorithms. The system currently supports only RS256 for token signing and verification. Although RSA-based algorithms provide robust security guarantees, newer schemes such as EdDSA (Ed25519) offer highly optimized computational performance and significantly smaller key sizes for equivalent security strength. These modern algorithms were excluded from the current scope strictly due to limitations in the maturity and native support of available PHP cryptographic libraries at the time of implementation.

4.5. Architectural Trade-offs Between Availability and Consistency

The implementation of a caching layer for JWKS and a grace period during key rotation (Algorithm 2.5.3, Phase 4) represents a deliberate architectural trade-off between system availability and eventual consistency. By caching public keys for 3600 seconds (1 hour), the architecture reduces database load by approximately 98% effectively decoupling authentication throughput from database performance. However, this mechanism introduces a revocation lag: if a key is compromised and administratively revoked, cached keys remain temporarily valid.

In the current single-server deployment of City Courier, cache invalidation upon key lifecycle events minimizes this lag, providing near-instantaneous consistency. However, recognizing that future horizontally scaled multi-node deployments may reintroduce temporary inconsistencies, this risk was determined to be acceptable when mitigated by a rigorous defense-in-depth strategy :

1. Short-Lived Access Tokens : By enforcing a strict 15-minute Time-To-Live (TTL) for access tokens, the exploitation window for compromised tokens is inherently limited, mitigating the impact of revocation lag.
2. Emergency Cache Purge : The system integrates a Redis-based signal channel capable of triggering an immediate cache clearance across all nodes. This aligns with the "Break-Glass" security principle, ensuring rapid system lockdown during critical incidents.
3. Rotation Redundancy : The overlapping grace period ensures that couriers operating in areas with intermittent network connectivity are not locked out during rotation events. This prioritizes operational continuity, a critical requirement for logistics Service Level Agreements (SLAs).

Overall, the JWKS architecture successfully balances the CAP theorem constraints, prioritizing Availability (A) and Partition Tolerance (P) while maintaining sufficient Consistency (C) for 99.9% of normal authentication operations.

5. CONCLUSION

This study demonstrates that switching from static secret-based signing (HS256) to a well-architected JWKS asymmetric signature (RS256) model significantly elevates the security posture of server-to-server communication without compromising operational agility. The implementation of a CustomJwtProvider with explicit algorithm allow-list enforcement and strict UUID v4 header sanitization effectively neutralized critical vulnerabilities such as Algorithm Confusion and Key Injection, which remain prevalent in default JWT configurations. The empirical breakdown of ten specific attack vectors confirms that while stateless authentication introduces inherent challenges such as replay attacks, a defense-in-depth strategy combining short TTLs, unique ID validation, and automated key rotation creates a resilient security perimeter suitable for high-value transaction systems.

Strategically, this research validates that the perceived complexity of asymmetric cryptography is outweighed by its security benefits, particularly in eliminating the "shared secret problem" the single

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

biggest failure point in API security. The zero-downtime rotation mechanism proposed in Algorithm 2.5.3 with explicit grace period enforcement via active_to timestamps and automatic JWKS cache invalidation, successfully reconciles the conflicting demands of continuous availability and regular key refreshes a critical requirement for 24/7 logistics platforms.

Overall, this study contributes : (1) a secure JWKS-based authentication architecture governed by strict cryptographic validation; (2) a resilient key lifecycle management mechanism enabling zero-downtime rotation; and (3) an empirically validated defense-in-depth strategy capable of mitigating multiple JWT exploitation vectors in real-world courier systems.

Future research will explore the integration of the Edwards-curve Digital Signature Algorithm (EdDSA) to further optimize computational overhead and the implementation of distributed caching layers to support multi-region scalability, ensuring the system remains robust against evolving cryptographic threats.

CONFLICT OF INTEREST

The authors declares that there is no conflict of interest between the authors or with research object in this paper.

REFERENCES

[1] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC Editor, RFC7519, May 2015. doi: 10.17487/RFC7519.

[2] M. G. De Almeida and E. D. Canedo, "Authentication and Authorization in Microservices Architecture: A Systematic Literature Review," *Applied Sciences*, vol. 12, no. 6, p. 3023, Mar. 2022, doi: 10.3390/app12063023.

[3] M. Jones, "JSON Web Algorithms (JWA)," RFC Editor, RFC7518, May 2015. doi: 10.17487/RFC7518.

[4] F. Baldimtsi *et al.*, "zkLogin: Privacy-Preserving Blockchain Authentication with Existing Credentials," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, in CCS '24. Salt Lake City UT USA: Association for Computing Machinery, Dec. 2024, pp. 3182–3196. doi: 10.1145/3658644.3690356.

[5] A. A. Simatupang, "IMPLEMENTASI RESTFUL WEB SERVICE DENGAN JSON WEB TOKEN DI PT. LESTARI ADIL MAKMUR," *Prosiding Seminar Nasional Mahasiswa Fakultas Teknologi Informasi (SENAFTI)*, vol. 2, no. 2, pp. 2183–2192, Oct. 2023.

[6] National Institute of Standards And Technology, "CVE-2022-29217 Detail," National Vulnerability Database. Accessed: Feb. 05, 2026. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2022-29217

[7] G. Tsigkourakos and C. Patsakis, "QRS: A Rule-Synthesizing Neuro-Symbolic Triad for Autonomous Vulnerability Discovery," Feb. 10, 2026, *arXiv*: arXiv:2602.09774. doi: 10.48550/arXiv.2602.09774.

[8] W. Wang *et al.*, "VulnRepairEval: An Exploit-Based Evaluation Framework for Assessing Large Language Model Vulnerability Repair Capabilities," Sep. 03, 2025, *arXiv*: arXiv:2509.03331. doi: 10.48550/arXiv.2509.03331.

[9] National Institute of Standards And Technology, "CVE-2024-37568 Detail," National Vulnerability Database. Accessed: Feb. 05, 2026. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2024-37568

[10] K. Bhatia, S. K. Pandey, V. K. Singh, and D. N. Gupta, "Hash and Physical Unclonable Function (PUF)-Based Mutual Authentication Mechanism," *Sensors*, vol. 23, no. 14, Jul. 2023, doi: 10.3390/s23146307.

[11] R. Chandran, "Cyber Security Holes of JSON Web Token," in *Next-Gen Technologies in Computational Intelligence: Proceeding of the International Conference on Next-Gen Technologies in Computational Intelligence (NGTCA 2023)*, 1st ed., London: CRC Press, 2024, pp. 201–206. doi: 10.1201/9781003430452-28.

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

[12] S. K. Jangam, N. Karri, and P. S. R. P. Muntala, "Advanced API Security Techniques and Service Management," *International Journal of Emerging Research in Engineering and Technology*, vol. 3, no. 4, pp. 63–74, Dec. 2022, doi: 10.63282/3050-922X.IJERET-V3I4P108.
[13] A. Fedele and C. Roner, "Dangerous games: A literature review on cybersecurity investments," *Journal of Economic Surveys*, vol. 36, no. 1, pp. 157–187, 2022, doi: 10.1111/joes.12456.
[14] P. Philippaerts, D. Preuveneers, and W. Joosen, "OAuch: Exploring Security Compliance in the OAuth 2.0 Ecosystem," in *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses*, in RAID '22. New York, NY, USA: Association for Computing Machinery, Oct. 2022, pp. 460–481. doi: 10.1145/3545948.3545955.
[15] A. Bucko, K. Vishi, B. Krasniqi, and B. Rexha, "Enhancing JWT Authentication and Authorization in Web Applications Based on User Behavior History," *Computers*, vol. 12, no. 4, Apr. 2023, doi: 10.3390/computers12040078.
[16] M. A. Shabi and R. R. Marie, "Analyzing Privacy Implications and Security Vulnerabilities in Single Sign-On Systems: A Case Study on OpenID Connect," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 15, no. 4, Apr. 2024, doi: 10.14569/IJACSA.2024.0150465.
[17] L. Zhang, C. Zhou, and J. Wen, "APSH-JWT: An Authentication Protocol Based on JWT with Scalability and Heterogeneity in Edge Computing," *Wireless Networks*, vol. 31, no. 3, pp. 2939–2953, Mar. 2025, doi: 10.1007/s11276-025-03926-2.
[18] P. Varalakshmi, G. B, V. S. P, D. T, and S. K, "Improvising JSON Web Token Authentication in SDN," in *2022 International Conference on Communication, Computing and Internet of Things (IC3IoT)*, IEEE, Mar. 2022, pp. 1–8. doi: 10.1109/IC3IOT53935.2022.9767873.
[19] USA and S. Ravikumar, "OAuth 1.0 vs. OAuth 2.0: An In-Depth Analysis of Token Handling, Client Authentication, and Developer Usability," *Journal of Mathematical & Computer Applications*, vol. 4, no. 4, pp. 1–5, Aug. 2025, doi: 10.47363/JMCA/2025(4)212.
[20] W. Niewolski, T. W. Nowak, M. Sepczuk, and Z. Kotulski, "Token-Based Authentication Framework for 5G MEC Mobile Networks," *Electronics*, vol. 10, no. 14, p. 1724, Jul. 2021, doi: 10.3390/electronics10141724.
[21] L. Brun, I. Hasuo, Y. Ono, and T. Sekiyama, "Automated Security Analysis for Real-World IoT Devices," in *Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy*, in HASP '23. New York, NY, USA: Association for Computing Machinery, Oct. 2023, pp. 29–37. doi: 10.1145/3623652.3623667.
[22] A. F. Nugraha, H. Kabetta, I. K. S. Buana, and R. B. Hadiprakoso, "Performance and Security Comparison of Json Web Tokens (JWT) and Platform Agnostic Security Tokens (PASETO) on RESTful APIs," in *2023 IEEE International Conference on Cryptography, Informatics, and Cybersecurity (ICoCICs)*, IEEE, Aug. 2023, pp. 15–22. doi: 10.1109/ICoCICs58778.2023.10277377.
[23] U.-S. Potti, H.-S. Huang, H.-T. Chen, and H.-M. Sun, "Security Testing Framework for Web Applications: Benchmarking ZAP V2.12.0 and V2.13.0 by OWASP as an example," Jan. 10, 2025, *arXiv*: arXiv:2501.05907. doi: 10.48550/arXiv.2501.05907.
[24] C. Cremers, L. Garratt, S. Smyshlyaev, N. Sullivan, and C. Wood, "Randomness Improvements for Security Protocols," RFC Editor, Request for Comments RFC 8937, Oct. 2020. doi: 10.17487/RFC8937.
[25] Z. Wang *et al.*, "Simple But Not Secure: An Empirical Security Analysis of Two-factor Authentication Systems," Nov. 18, 2024, *arXiv*: arXiv:2411.11551. doi: 10.48550/arXiv.2411.11551.
[26] D. Temoshok, J. Fenton, Y.-Y. Choong, N. Lefkovitz, A. Regenscheid, and J. Richer, "Digital Identity Guidelines: Authentication and Authenticator Management," National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-63B-4, 2024. doi: 10.6028/NIST.SP.800-63B-4.
[27] J. Müller and J. Oupický, "Post-quantum XML and SAML Single Sign-On," *Proceedings on Privacy Enhancing Technologies*, vol. 2024, no. 4, pp. 525–543, Oct. 2024, doi: 10.56553/popets-2024-0128.

Vol. 7, No. 2, April 2026, Page. 1834-1852
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5662

[28] S. Dalimunthe, J. Reza, and A. Marzuki, "The Model for Storing Tokens in Local Storage (Cookies) Using JSON Web Token (JWT) with HMAC (Hash-based Message Authentication Code) in E-Learning Systems," *Journal of Applied Engineering and Technological Science (JAETS)*, vol. 3, no. 2, pp. 149–155, Jun. 2022, doi: 10.37385/jaets.v3i2.662.

[29] L. Perugini and A. Vesco, "An Efficient TLS 1.3 Handshake Protocol with VC Certificate Type," in *2025 IEEE 22nd Consumer Communications & Networking Conference (CCNC)*, Jan. 2025, pp. 1–9. doi: 10.1109/CCNC54725.2025.10975914.

[30] Z. Zairina, R. B. Huwae, and A. H. Jatmika, "IMPLEMENTASI OWASP TOP 10 DALAM PENGUJIAN PENETRASI WEBSITE : MENGIDENTIFIKASI CELAH KEAMANAN DALAM SISTEM PENGELOLAAN VOTING INDONESIA," *Jurnal Teknologi Informasi, Komputer, dan Aplikasinya (JTIKA)*, vol. 7, no. 1, pp. 98–108, Mar. 2025, doi: 10.29303/jtika.v7i1.456.

[31] F. Pagano, A. Romdhana, D. Caputo, L. Verderame, and A. Merlo, "SEBASTiAn: A Static and Extensible Black-Box Application Security Testing Tool for iOS and Android Applications," *SoftwareX*, vol. 23, p. 101448, Jul. 2023, doi: 10.1016/j.softx.2023.101448.

[32] S. Rohlmann, V. Mladenov, C. Mainka, and J. Schwenk, "Breaking the Specification: PDF Certification," in *2021 IEEE Symposium on Security and Privacy (SP)*, May 2021, pp. 1485–1501. doi: 10.1109/SP40001.2021.00110.

[33] A. Folorunso, V. Mohammed, I. Wada, and B. Samuel, "The impact of ISO security standards on enhancing cybersecurity posture in organizations," *World Journal of Advanced Research and Reviews*, vol. 24, no. 1, pp. 2582–2595, 2024, doi: 10.30574/wjarr.2024.24.1.3169.

[34] H. Setiawan, N. A. Hana, and R. R. Hanaputra, "Mapping ISO 27001:2013 and COBIT 2019 Framework to STRIDE Threat Modeling Using Qualitative Descriptive Research," *Journal of Computer Engineering, Electronics and Information Technology*, vol. 3, no. 2, pp. 101–110, Nov. 2024, doi: 10.17509/coelite.v3i2.73228.

[35] Y.-C. Yang, K.-F. Lu, Y.-X. Chen, and R.-S. Tsay, "Ensuring GDPR Compliance in IoT Network With a Glass Box Security Guard System," *IEEE Transactions on Privacy*, vol. 2, pp. 27–40, 2025, doi: 10.1109/TP.2025.3546854.