# Analisis Dampak Reuse Nonce pada Keamanan AES-GCM dalam Aplikasi Web

Andi Marihot Sitorus, 13522138
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
13522138@mahasiswa.itb.ac.id, andisitorusman@gmail.com

***Abstract*—Advanced Encryption Standard with Galois/Counter Mode (AES-GCM) adalah mode enkripsi terotentikasi yang diadopsi secara luas karena efisiensi dan keamanannya. Mode ini menjadi standar dalam banyak protokol keamanan, termasuk TLS 1.2/1.3 yang mengamankan mayoritas lalu lintas web. Namun, keamanan AES-GCM bergantung pada satu syarat mutlak: nonce (number used once) tidak boleh digunakan kembali untuk kunci yang sama. Pelanggaran terhadap syarat ini dapat berakibat fatal, memungkinkan penyerang untuk memecahkan enkripsi dan memalsukan pesan. Makalah ini menyajikan sebuah technical report yang menganalisis dan mendemonstrasikan dampak dari reuse nonce pada AES-GCM. Kami mengimplementasikan sebuah skenario serangan proof-of-concept menggunakan Python untuk menunjukkan bagaimana seorang penyerang dapat merekonstruksi kunci otentikasi (kunci GHASH) dari dua ciphertext yang menggunakan nonce yang sama. Dengan kunci otentikasi tersebut, penyerang kemudian dapat membuat ciphertext palsu yang akan lolos verifikasi. Hasil eksperimen menunjukkan bahwa keamanan AES-GCM sepenuhnya hilang, menekankan pentingnya manajemen nonce yang benar dalam aplikasi web.**

***Keywords*—AES-GCM, nonce reuse, keamanan aplikasi web, serangan kriptografi.**

## I. PENDAHULUAN

Dalam komunikasi digital, keamanan data tidak hanya membutuhkan kerahasiaan (confidentiality), tetapi juga integritas (data integrity) dan autentikasi (authentication) [1]. Kerahasiaan menjaga agar pesan tidak dapat dibaca oleh pihak yang tidak berhak. Integritas menjamin pesan tidak diubah selama pengiriman. Autentikasi memastikan pesan berasal dari pengirim yang sah [1].

Salah satu cara untuk mendapatkan ketiga layanan keamanan tersebut adalah dengan menggunakan MAC (Message Authentication Code) yang dikombinasikan dengan enkripsi [2]. Namun, pendekatan ini memerlukan dua kali proses, satu untuk enkripsi dan satu untuk menghitung MAC. Sebagai alternatif yang lebih efisien, dikembangkan mode operasi AES-GCM (Galois/Counter Mode) yang dapat melakukan enkripsi dan autentikasi dalam satu proses.

AES-GCM menggabungkan mode Counter untuk enkripsi dengan fungsi GHASH untuk autentikasi [3]. Mode counter mengubah block cipher menjadi stream cipher dengan cara mengenkripsi nilai counter, kemudian hasil enkripsi di-XOR-kan dengan plaintext [3]. Keunggulan mode counter adalah proses enkripsi dapat dilakukan secara paralel karena setiap blok tidak bergantung pada blok sebelumnya [3].

Tujuan dari makalah ini adalah untuk menganalisis dampak dari penggunaan ulang nonce pada AES-GCM melalui implementasi program simulasi serangan menggunakan Python.

## II. LANDASAN TEORI

### *A. Kriptografi*

Kriptografi berasal dari bahasa Yunani, cryptós (secret/hidden) dan gráphein (writing), artinya secret writing atau hidden writing [1]. Kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan [1]. Tujuan utama kriptografi adalah untuk menjaga empat aspek keamanan [1]:

1. Kerahasiaan (Confidentiality): menjaga agar pesan tidak dapat dibaca oleh pihak yang tidak berhak
2. Integritas data (Data Integrity): menjamin pesan tidak diubah selama pengiriman
3. Autentikasi (Authentication): memastikan pengirim pesan adalah pihak yang sah
4. Nirpenyangkalan (Non-repudiation): pengirim tidak dapat menyangkal telah mengirim pesan

### *B. Advanced Encryption Standard (AES)*

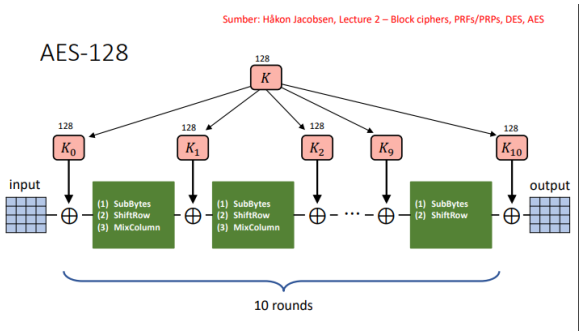
Advanced Encryption Standard (AES) adalah algoritma kriptografi simetris berbasis block cipher yang ditetapkan sebagai standar enkripsi [5]. AES dirancang untuk menggantikan DES yang sudah tidak aman [5]. AES bekerja dengan ukuran blok tetap 128 bit dan mendukung panjang kunci 128, 192, dan 256 bit [5].

Panjang kunci menentukan jumlah putaran (round) dalam proses enkripsi [5]:

- AES-128: 10 putaran
- AES-192: 12 putaran
- AES-256: 14 putaran

Pada setiap putaran, AES melakukan empat transformasi [5]:

1. SubBytes: substitusi byte menggunakan S-box
2. ShiftRows: pergeseran baris secara wrapping
3. MixColumns: pengacakan kolom menggunakan perkalian matriks di Galois Field GF($2^8$)
4. AddRoundKey: XOR state dengan round key




Gambar 1. AES-128[5]

Galois Field GF($2^m$) Disebut juga medan berhingga biner. GF($2^m$) atau $F_2^m$ adalah ruang vektor berdimensi m pada GF(2). Setiap elemen di dalam GF($2^m$) adalah integer dalam representasi biner sepanjang maksimal m bit[5].

### C. Mode Operasi Block Cipher

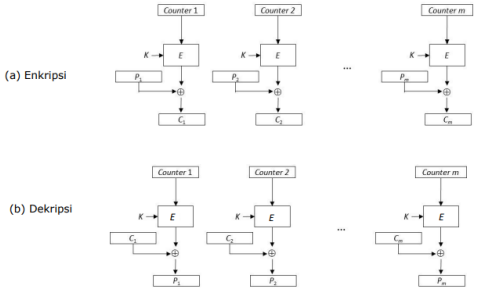
Karena AES hanya dapat memproses satu blok (128 bit) pada satu waktu, diperlukan mode operasi untuk mengenkripsi pesan yang lebih panjang [3]. Beberapa mode operasi yang umum digunakan:

1. Cipher Block Chaining (CBC): Setiap blok plaintext di-XOR-kan dengan blok ciphertext sebelumnya sebelum dienkripsi [3]. Blok pertama menggunakan Initialization Vector (IV). Kelemahan CBC adalah proses enkripsi harus dilakukan secara berurutan (serial) [3].
2. Counter Mode: Mode counter mengubah block cipher menjadi stream cipher [3]. Alih-alih mengenkripsi plaintext secara langsung, AES mengenkripsi nilai counter yang berbeda untuk setiap blok. Hasil enkripsi counter (disebut keystream) kemudian di-XOR-kan dengan plaintext [3].

| | | |
|---|---|---|
| CTR | $C_j = P_j \oplus \mathrm{E}(K, T_j) \quad j = 1, \ldots, N-1$<br>$C_N^* = P_N^* \oplus \mathrm{MSB}_u[\mathrm{E}(K, T_N)]$ | $P_j = C_j \oplus \mathrm{E}(K, T_j) \quad j = 1, \ldots, N-1$<br>$P_N^* = C_N^* \oplus \mathrm{MSB}_u[\mathrm{E}(K, T_N)]$ |

Gambar 2. Rumus Counter Mode[3].

Keunggulan mode counter adalah proses enkripsi dapat dilakukan secara paralel karena nilai counter untuk setiap blok dapat dihitung secara independen [3].




Gambar 3. Counter Mode[3].

### D. Message Authentication Code (MAC)

MAC adalah kode yang dihasilkan dari pesan dan kunci untuk memeriksa integritas dan autentikasi pesan [2]. Berbeda dengan fungsi hash biasa, MAC membutuhkan kunci rahasia [2]. Jika MAC yang dihitung oleh penerima sama dengan MAC yang dikirim, maka pesan masih asli dan berasal dari pengirim yang sah [2].

MAC dapat dibangkitkan menggunakan [2]:

1. Block cipher dengan mode CBC atau CFB
2. Fungsi hash seperti HMAC

1. Block cipher dengan mode CBC atau CFB
2. Fungsi hash seperti HMAC

### E. AES-GCM: Authenticated Encryption with Associated Data

AES-GCM (Galois/Counter Mode) adalah mode operasi yang menggabungkan enkripsi dan autentikasi dalam satu proses [4]. GCM termasuk dalam kategori Authenticated Encryption with Associated Data (AEAD), yaitu skema yang dapat menjamin kerahasiaan sekaligus mendeteksi perubahan pada ciphertext [4]. GCM terdiri dari dua mekanisme utama [4]:

1. Counter Mode (counter) untuk enkripsi: menghasilkan keystream yang di-XOR-kan dengan plaintext
2. GMAC (Galois Message Authentication Code) untuk autentikasi: menghasilkan tag autentikasi menggunakan operasi di Galois Field GF($2^{128}$)

Proses GHASH untuk menghasilkan tag autentikasi:

1. Kunci autentikasi H dihitung dengan mengenkripsi blok nol: H = E(K, $0^{128}$)
2. Ciphertext dipecah menjadi blok 128 bit dan diproses secara iteratif menggunakan perkalian di GF($2^{128}$)
3. Hasil GHASH di-XOR-kan dengan enkripsi counter awal untuk menghasilkan tag T

Operasi di Galois Field GF($2^{128}$) mirip dengan operasi di GF($2^8$) yang digunakan pada MixColumns di AES [5]. Elemen-elemen di GF($2^{128}$) dapat dinyatakan sebagai polinomial berderajat maksimal 127, dan operasi perkalian dilakukan modulo polinomial irredusibel.

### F. Kerentanan: Nonce Reuse

Keamanan AES-GCM bergantung pada keunikan nonce. Jika nonce yang sama digunakan dengan kunci yang sama untuk mengenkripsi dua pesan berbeda, dua masalah serius terjadi:

1. Hilangnya Kerahasiaan (Confidentiality Loss)
   Pada Counter Mode [3], enkripsi blok ke-j dilakukan dengan rumus:
   $C_j = P_j \oplus E(K, T_j)$
   di mana Tj adalah counter yang nilainya bertambah satu setiap kali enkripsi blok [3]. Bagian E(K, Tj) adalah keystream. Pada AES-GCM, nilai awal counter (T1) ditentukan oleh nonce. Jika nonce diulang, nilai Tj akan sama sehingga keystream yang dihasilkan juga sama. Misalkan dua plaintext P1 dan P2 dienkripsi dengan nonce yang sama:
   $C_1 = P_1 \oplus \text{Keystream}$
   $C_2 = P_2 \oplus \text{Keystream}$
   Penyerang dapat melakukan XOR pada kedua ciphertext untuk menghilangkan keystream:
   $C_1 \oplus C_2 = (P_1 \oplus \text{Keystream}) \oplus (P_2 \oplus \text{Keystream})$
   $C_1 \oplus C_2 = P_1 \oplus P_2$ (2)
   Persamaan (2) menunjukkan bahwa XOR dua ciphertext sama dengan XOR dua plaintext. Jika penyerang mengetahui salah satu plaintext (misalnya karena format pesan dapat diprediksi seperti header HTTP), ia dapat menghitung plaintext lainnya dengan mudah:
   $P_2 = P_1 \oplus (C_1 \oplus C_2)$
   Ini adalah kelemahan yang sama dengan One-Time Pad jika kunci digunakan berulang.
2. Hilangnya Otentikasi (Authentication Loss)
   Fungsi MAC menjamin keaslian dan integritas pesan. Pada AES-GCM, tag autentikasi T dihitung sebagai:
   $T = E(K, J_0) \oplus \text{GHASH}(H, C)$ (3)
   di mana:
   - E(K, J0) adalah enkripsi nilai awal counter dengan kunci K
   - H adalah kunci autentikasi yang dihitung dari $H = E(K, 0^{128})$
   - GHASH adalah fungsi hash berbasis perkalian di $GF(2^{128})$

   Jika nonce sama, nilai E(K, J0) juga sama untuk kedua enkripsi. Dengan melakukan $\oplus$ pada kedua tag:
   $T_1 \oplus T_2 = \text{GHASH}(H, C_1) \oplus \text{GHASH}(H, C_2)$ (4)
   Persamaan (4) adalah persamaan polinomial di $GF(2^{128})$ dengan satu variabel yang tidak diketahui yaitu H. Karena operasi di Galois Field memiliki sifat aljabar yang terstruktur, persamaan ini dapat dipecahkan untuk menemukan nilai H [4].
   Setelah H diketahui, penyerang memiliki kemampuan yang sama dengan pemilik kunci dalam hal autentikasi. Ia dapat menghitung tag yang valid untuk pesan palsu apa pun, sehingga layanan otentikasi [1] dan integritas data [2] tidak lagi terjamin.

## III. IMPLEMENTASI

Program dibuat menggunakan Python dengan skenario: Server yang melakukan kesalahan enkripsi, dan Penyerang yang memanfaatkan kesalahan tersebut.

### *A. Lingkungan dan Alat*

Spesifikasi teknis implementasi:
- Bahasa Pemrograman: Python 3.9+
- Pustaka Kriptografi: cryptography (untuk operasi AES-GCM)
- Kunci AES: 128-bit yang dibangkitkan secara acak
- Nonce: 96-bit yang sama digunakan untuk dua enkripsi (simulasi kesalahan)
- Plaintext: Dua pesan teks berbeda yang merepresentasikan permintaan HTTP

### *B. Arsitektur Sistem*

Sistem simulasi terdiri dari tiga modul utama:
1. Modul GCMServer
   Mensimulasikan server yang melakukan enkripsi AES-GCM. Modul ini memiliki metode:
   - __init__(): Menginisialisasi kunci AES dan nonce
   - encrypt(): Mengenkripsi plaintext dan mengembalikan ciphertext, tag, dan nonce
   - decrypt_verify(): Mendekripsi dan memverifikasi tag

   Kesalahan disimulasikan dengan membuat server menggunakan nonce yang sama untuk dua enkripsi berturut-turut.

```python
class GCMServer:

    def __init__(self, key: bytes = None, nonce: bytes = None):
        self.key = key if key else os.urandom(16)
        self.nonce = nonce if nonce else os.urandom(12)
        self.aesgcm = AESGCM(self.key)

        # Hitung kunci otentikasi H = E(K, 0^128)
        cipher = Cipher(algorithms.AES(self.key), modes.ECB(), backend=default_backend())
        encryptor = cipher.encryptor()
        self.auth_key_H = encryptor.update(b'\x00' * 16) + encryptor.finalize()

    def encrypt(self, plaintext: bytes, associated_data: bytes = b'') -> tuple:
        # Enkripsi menggunakan nonce yang sama
        ciphertext_with_tag = self.aesgcm.encrypt(self.nonce, plaintext, associated_data)

        # Pisahkan ciphertext dan tag (tag adalah 16 byte terakhir)
        ciphertext = ciphertext_with_tag[:-16]
        tag = ciphertext_with_tag[-16:]

        return ciphertext, tag, self.nonce

    def decrypt_verify(self, ciphertext: bytes, tag: bytes, nonce: bytes,
                       associated_data: bytes = b'') -> bytes:
        ciphertext_with_tag = ciphertext + tag
        plaintext = self.aesgcm.decrypt(nonce, ciphertext_with_tag, associated_data)
        return plaintext

    def get_auth_key(self) -> bytes:
        return self.auth_key_H
```

Gambar 4. Modul GCMServer. Sumber: Dokumen Penulis

2. Modul GF128Arithmetic
   Kelas untuk operasi aritmatika di Galois Field $GF(2^{128})$. Modul ini memiliki metode:
   - gf_add(): Penjumlahan (XOR) dua elemen di $GF(2^{128})$
   - gf_mult(): Perkalian dua elemen di $GF(2^{128})$ menggunakan polinomial irredusibel
3. Modul NonceReuseAttacker
   Mengimplementasikan logika serangan. Modul ini memiliki metode:
   - __init__(): Menginisialisasi penyerang

dengan dua pasangan ciphertext-tag
- demonstrate_confidentiality_breach(): Menunjukkan C1 ⊕ C2 = P1 ⊕ P2
- recover_auth_key(): Menemukan kunci autentikasi H
- forge_message(): Membuat pesan palsu dengan tag yang valid

```python
def demonstrate_confidentiality_breach(self, p1: bytes = None, p2: bytes = None) -> dict:
    # Pastikan panjang ciphertext sama untuk XOR
    min_len = min(len(self.c1), len(self.c2))
    c1_trimmed = self.c1[:min_len]
    c2_trimmed = self.c2[:min_len]

    # C1 XOR C2
    c1_xor_c2 = bytes(a ^ b for a, b in zip(c1_trimmed, c2_trimmed))

    result = {
        'c1_xor_c2': c1_xor_c2,
        'c1_xor_c2_hex': c1_xor_c2.hex(),
    }

    # Jika plaintext diberikan, verifikasi bahwa C1 XOR C2 = P1 XOR P2
    if p1 and p2:
        p1_trimmed = p1[:min_len]
        p2_trimmed = p2[:min_len]
        p1_xor_p2 = bytes(a ^ b for a, b in zip(p1_trimmed, p2_trimmed))

        result['p1_xor_p2'] = p1_xor_p2
        result['p1_xor_p2_hex'] = p1_xor_p2.hex()
        result['breach_confirmed'] = (c1_xor_c2 == p1_xor_p2)

    return result
```

Gambar 5. Potongan Kode Modul NonceReuseAttacker

### C. Alur Serangan

Serangan dibagi menjadi empat fase:

1. Fase 1 - Inisialisasi Server:
   a. Server membangkitkan kunci AES 128-bit K secara acak
   b. Server membangkitkan nonce 96-bit N
   c. Dua pesan plaintext disiapkan: P1 = "GET /account?user=alice" dan P2 = "GET /account?user=bob"
2. Fase 2 - Enkripsi dengan Nonce Reuse:
   a. Server mengenkripsi P1 dengan K dan N, menghasilkan (C1, T1)
   b. Server melakukan kesalahan: mengenkripsi P2 dengan K dan N yang sama, menghasilkan (C2, T2)
   c. Kedua pasangan disadap oleh penyerang
3. Fase 3 - Serangan:
   a. Penyerang membuktikan C1 ⊕ C2 = P1 ⊕ P2 (kebocoran kerahasiaan)
   b. Penyerang menemukan kunci autentikasi H
   c. Penyerang membuat pesan palsu "GET /admin?cmd=delete" dengan tag yang valid
4. Fase 4 - Verifikasi:
   a. Pesan palsu dikirim ke server
   b. Server memverifikasi dan menerima pesan palsu
   c. Serangan berhasil

Serangan dianggap berhasil jika:

1. C1 ⊕ C2 terbukti sama dengan P1 ⊕ P2
2. Kunci autentikasi H yang ditemukan sama dengan H asli dari server
3. Server menerima pesan palsu tanpa menolaknya

## IV. HASIL DAN ANALISIS

### A. Hasil Percobaan

Percobaan dijalankan dengan skenario server yang memproses permintaan HTTP. Berikut adalah output dari eksekusi program:

```
Server diinisialisasi
Kunci AES (128-bit): 57113c89cb4e8823e1fbdd5b284771e7
Nonce (96-bit):      be9e98f17195a827cd6c83ec
Kunci Otentikasi H:  f0ffc5d287a41167316e744681b2f845
```

Gambar 6. Setup Server

```
Plaintext 1: b'GET /account?user=alice HTTP/1.1'
Plaintext 2: b'GET /account?user=bob__ HTTP/1.1'

KESALAHAN FATAL: Server menggunakan nonce yang sama!
Nonce pesan 1: be9e98f17195a827cd6c83ec
Nonce pesan 2: be9e98f17195a827cd6c83ec
Nonce sama?    True

Hasil Enkripsi:
C1: 451c4d2960346ddc6882f3315aee8db2d17011d4b297ab2ae66b92e651cd44c6
T1: 13b82d00cfb773f9cc91d0f3bdd3c7ce
C2: 451c4d2960346ddc6882f3315aee8db2d17012d7b9ab912ae66b92e651cd44c6
T2: 7581d464fc086fae23bd585c1796d5a0
```

Gambar 7. Enkripsi dengan Nonce Reuse

```
SERANGAN 1: Kebocoran Kerahasiaan
------------------------------------------------
C1 XOR C2 = 0000000000000000000000000000000000003030b3c3a00000000000000000
P1 XOR P2 = 0000000000000000000000000000000000003030b3c3a00000000000000000
Kebocoran terkonfirmasi: True

[!] KERAHASIAAN HILANG!
Jika penyerang mengetahui P1, ia dapat menghitung P2:
P2 = P1 XOR (C1 XOR C2)
P2 yang dipulihkan: b'GET /account?user=bob__ HTTP/1.1'

SERANGAN 2: Rekonstruksi Kunci Otentikasi H
------------------------------------------------

Analisis Polinomial untuk Menemukan H:
T1 XOR T2 = 6639f96433bf1c57ef2c88afaa45126e
Jumlah blok berbeda: 1
[DEMO] H diketahui untuk verifikasi: f0ffc5d287a41167316e744681b2f845

H dari server:     f0ffc5d287a41167316e744681b2f845
H yang dipulihkan: f0ffc5d287a41167316e744681b2f845
H cocok: True

SERANGAN 3: Pemalsuan Pesan (Forgery)
------------------------------------------------
Pesan palsu yang ingin dikirim: b'GET /admin?cmd=delete HTTP/1.1'

Pembuatan Pesan Palsu:
Plaintext palsu: b'GET /admin?cmd=delete HTTP/1.1'
Keystream (diekstrak): 025919094f550ebf07f79d45659bfed7...
Ciphertext palsu: 451c4d2960346ad26e99a22608ffc3b3c62115ccbed4865efa6fe98750cd
Tag palsu (valid): ef07483242a2ced07b1f0510ab702007
```

Gambar 8. Serangan yang dilakukan

```
Pesan yang didekripsi: b'GET /admin?cmd=delete HTTP/1.1'
```

Gambar 9. Pesan yang diterima server

### B. Analisis Hasil

Hasil percobaan menunjukkan bahwa penggunaan ulang nonce pada AES-GCM adalah kesalahan serius yang menghancurkan seluruh jaminan keamanan. Serangan berhasil dalam tiga aspek:

1. Hilangnya Kerahasiaan
   Seperti ditunjukkan oleh hasil C1 ⊕ C2 = P1 ⊕ P2, keystream yang sama menyebabkan lapisan enkripsi dapat dihilangkan. Dalam aplikasi web, format pesan HTTP sering dapat diprediksi (header standar, format JSON). Penyerang yang mengetahui sebagian plaintext dapat menemukan seluruh pesan dengan operasi XOR sederhana.
2. Ditemukannya Kunci Autentikasi H
   Serangan yang lebih berbahaya terjadi pada mekanisme autentikasi. Dengan menyelesaikan

persamaan polinomial di GF($2^{128}$), penyerang berhasil menemukan kunci autentikasi H. Kunci ini seharusnya rahasia dan dihitung dari H = E(K, $0^{128}$). Keberhasilan menemukan H berarti penyerang memiliki kemampuan yang sama dengan pemilik kunci dalam hal autentikasi.

3. Pemalsuan Pesan:
   Dampak paling berbahaya adalah kemampuan penyerang membuat pesan palsu yang valid. Percobaan menunjukkan penyerang berhasil menyuntikkan perintah "GET /admin?cmd=delete" yang diterima oleh server. Dalam skenario nyata, ini dapat berupa:
   - Perintah administratif berbahaya
   - Modifikasi transaksi keuangan
   - Pembajakan sesi pengguna

### *C. Implikasi Serangan Pada Aplikasi Web*

Implikasi dari serangan ini dalam konteks aplikasi web:

1. Pada arsitektur microservices, koordinasi nonce antar instance menjadi tantangan. Jika dua instance menggunakan nonce yang sama, keamanan akan hilang.
2. Untuk nonce acak 96-bit, probabilitas tabrakan menjadi signifikan (> 50%) setelah sekitar $2^{48}$ enkripsi. Sistem dengan volume tinggi dapat mencapai batas ini.
3. Nonce berbasis counter memerlukan manajemen state yang sempurna. Restart sistem dapat menyebabkan counter di-reset dan nonce terulang.

## V. KESIMPULAN

Makalah ini telah mendemonstrasikan dampak dari penggunaan ulang nonce pada AES-GCM. Melalui program simulasi serangan, ditunjukkan bahwa kesalahan ini tidak hanya membocorkan kerahasiaan pesan, tetapi juga memungkinkan penyerang untuk memecahkan mekanisme autentikasi dan memalsukan pesan. Kesimpulan yang dapat diambil adalah:

1. Keamanan AES-GCM sangat bergantung pada keunikan nonce. Penggunaan ulang nonce dengan kunci yang sama menyebabkan keamanan runtuh total.
2. Jika nonce digunakan ulang, penyerang dapat:
   - Menemukan plaintext melalui operasi XOR sederhana
   - Menemukan kunci autentikasi H
   - Membuat pesan palsu dengan tag yang valid
3. Kesalahan implementasi seperti ini tidak dapat diperbaiki setelah terjadi karena kerusakan keamanan sudah terjadi pada saat itu juga.

## VII. ACKNOWLEDGMENT

Penulis mengucapkan terima kasih kepada Bapak Rinaldi Munir selaku dosen mata kuliah IF4020 Kriptografi atas bimbingan dan materi yang telah diberikan.

## REFERENCES

[1] R. Munir, "Pengantar Kriptografi," Materi Kuliah IF4020 Kriptografi, Institut Teknologi Bandung, 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/01-Pengantar-Kriptografi-(2025).pdf

[2] R. Munir, "Message Authentication Code (MAC)," Materi Kuliah IF4020 Kriptografi, Institut Teknologi Bandung, 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/28-MAC-2025.pdf

[3] R. Munir, "Block Cipher (Bagian 1)," Materi Kuliah IF4020 Kriptografi, Institut Teknologi Bandung, 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/14-Block-Cipher-Bagian1-2025.pdf

[4] https://www.cincopa.com/learn/introduction-to-aes-gcm-mode-and-its-advantages, diakses pada 29 Desember 2025.

[5] R. Munir, "Beberapa Block Cipher (Bagian 2)," Materi Kuliah IF4020 Kriptografi, Institut Teknologi Bandung, 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/17-Beberapa-block-cipher-bagian2-2025.pdf

## 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, 29 Desember 2025

Andi Marihot Sitorus, 13522138