Jurnal Ilmiah Wahana Pendidikan, Juli 2026, 12(7.D), 136-158
DOI: https://jurnal.peneliti.net/index.php/JIWP/article/view/13632
p-ISSN: 2622-8327 e-ISSN: 2089-5364
Accredited by Directorate General of Strengthening for Research and Development
Available online at https://jurnal.peneliti.net/index.php/JIWP

Implementasi Model Enkripsi Hybrid Adaptif AES dan Android Keystore untuk Pengamanan Data Teks dan Citra Android

Luthfillah Mafazi[1], Dadang Iskandar Mulyana[2]

[1,2] Sekolah Tinggi Ilmu Komputer Cipta Karya Informatika (STIKOMCKI) Jakarta

Received: 19 Januari 2026
Revised: 29 Januari 2026
Accepted: 9 Februari 2026

Abstrak

Keamanan data pribadi pada perangkat mobile menjadi semakin kritis seiring meningkatnya penggunaan aplikasi digital yang menyimpan informasi sensitif. Namun, banyak aplikasi Android belum menerapkan enkripsi yang kuat dan aman sehingga rentan terhadap eksploitasi pihak ketiga. Penelitian ini mengusulkan *Hybrid Model Encryption* berbasis kombinasi algoritma Advanced Encryption Standard (AES) dengan Android Keystore System untuk meningkatkan perlindungan data pengguna. Model yang dikembangkan bersifat adaptif melalui mekanisme Adaptive Mode Selector (AMS), yang secara otomatis memilih mode operasi AES (CBC, GCM, atau CTR) berdasarkan ukuran data dan kondisi perangkat. Implementasi dilakukan melalui prototipe aplikasi Android yang mampu melakukan enkripsi dan dekripsi pada teks, gambar, dan file umum. Hasil pengujian menunjukkan bahwa model hybrid mampu menjaga kerahasiaan dan integritas data, memberikan performa enkripsi yang efisien, serta meningkatkan kenyamanan pengguna melalui proses enkripsi otomatis. Dengan demikian, penelitian ini memberikan kontribusi berupa model enkripsi adaptif yang aman, efisien, dan sesuai dengan kebutuhan perlindungan data pada perangkat mobile modern.

Kata Kunci: enkripsi adaptif, Android Keystore System, AES, keamanan data mobile, hybrid encryption, proteksi data pribadi.

(*) Corresponding Author: luthfimfazi18@gmail.com, mahvin2012@gmail.com

How to Cite: Mafazi, L., & Mulyana, D. (2026). Implementasi Model Enkripsi Hybrid Adaptif AES dan Android Keystore untuk Pengamanan Data Teks dan Citra Android. *Jurnal Ilmiah Wahana Pendidikan*, *12*(7.D), 136-158. Retrieved from https://jurnal.peneliti.net/index.php/JIWP/article/view/13632.

PENDAHULUAN

Perkembangan teknologi informasi dan komunikasi yang pesat telah mengubah cara masyarakat berinteraksi, bertransaksi, dan mengelola data pribadi secara digital. Sistem operasi Android menjadi platform utama dengan lebih dari 70% pangsa pasar perangkat *mobile* di dunia (Blessing et al., 2025). Keunggulan Android yang bersifat *open-source* menjadikannya fleksibel, namun sekaligus meningkatkan risiko terhadap keamanan data dan privasi pengguna.

Berbagai aplikasi berbasis Android saat ini menangani data sensitif seperti identitas pribadi, lokasi, rekam medis, maupun data keuangan. Kelemahan utama yang sering ditemukan pada sistem keamanan aplikasi Android adalah penyimpanan kunci enkripsi (*encryption key*) yang tidak aman. Banyak pengembang menyimpan kunci secara lokal di *shared preferences* atau file sistem, yang mudah diakses melalui teknik *reverse engineering* (Shin et al., 2013). Kondisi

ini mengancam integritas dan kerahasiaan data pengguna, meskipun algoritma enkripsi yang digunakan tergolong kuat.

Algoritma *Advanced Encryption Standard* (AES) merupakan salah satu algoritma simetris paling populer dan digunakan secara luas karena tingkat keamanan tinggi dan efisiensi proses enkripsi-dekripsinya (Kumar & Goel, 2025). Namun, kelemahan AES terletak pada manajemen kunci (*key management*), di mana kunci yang disimpan secara tidak aman dapat menjadi titik lemah sistem.

Sebaliknya, *Android Keystore System* (AKS) merupakan mekanisme keamanan berbasis perangkat keras (*hardware-backed security module*) yang disediakan oleh Google untuk melindungi kunci kriptografi di dalam *Trusted Execution Environment* (TEE). AKS membatasi akses kunci hanya melalui *API* sistem, sehingga mencegah pencurian kunci oleh aplikasi pihak ketiga (Rzayeva et al., 2025).

Dengan mempertimbangkan kekuatan masing-masing metode, penelitian ini mengusulkan sebuah model enkripsi hibrida adaptif (*Hybrid Model Encryption*) yang menggabungkan AES sebagai algoritma enkripsi utama dengan *Android Keystore System* sebagai pengelola kunci kriptografi. Model adaptif ini mampu menyesuaikan mode operasi AES (CBC, GCM, CTR) secara dinamis berdasarkan tingkat sensitivitas data dan kebutuhan performa aplikasi.

Penelitian ini diharapkan menghasilkan solusi keamanan data pribadi pada perangkat Android yang lebih kuat, efisien, dan adaptif, tanpa mengorbankan kinerja aplikasi serta pengalaman pengguna.

METODE

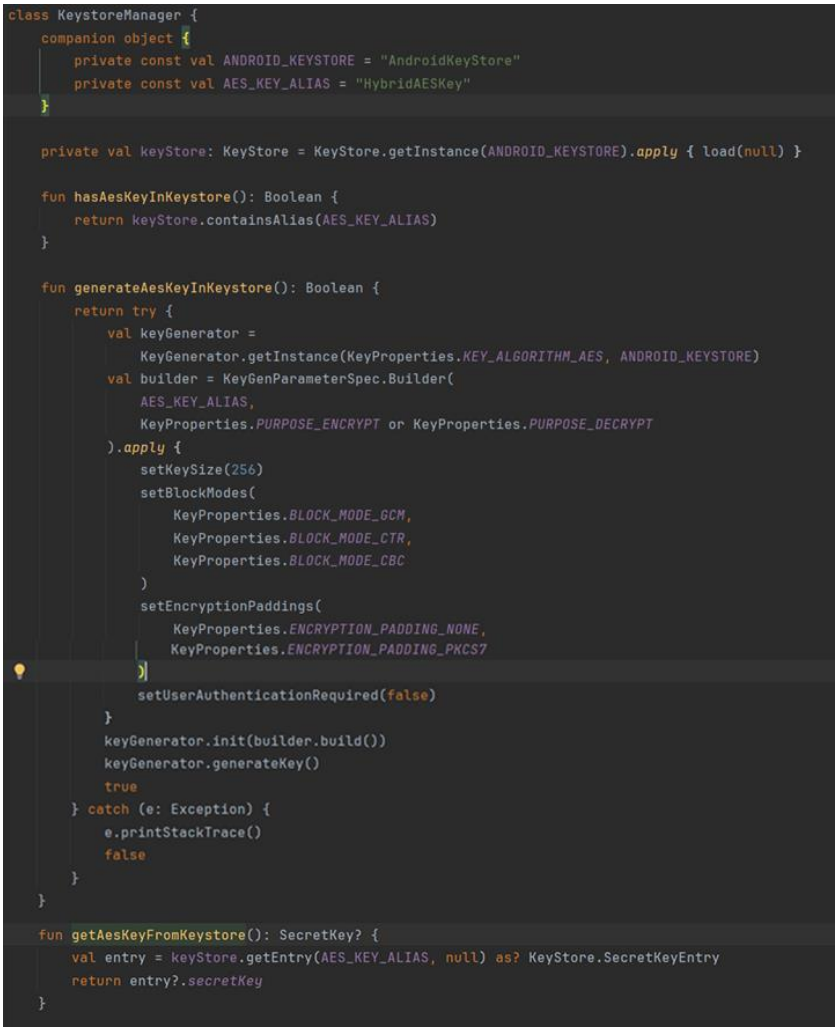
Penelitian ini menggunakan metode eksperimen rekayasa perangkat lunak (*software engineering experiment*) dengan fokus pada pengembangan dan evaluasi model *Adaptive Hybrid Encryption* yang mengintegrasikan *Advanced Encryption Standard* (AES) 256-bit dan *Android Keystore System* untuk perlindungan data pada perangkat Android. Data yang digunakan berupa data uji sintetis (*test data*) berupa teks dan citra (*image*) yang merepresentasikan data pribadi pengguna, tanpa melibatkan dataset publik atau data berlabel (*labeled dataset*), karena penelitian berfokus pada aspek kriptografi dan keamanan data. Tahapan metode meliputi pengumpulan data uji, *preprocessing* (validasi format, pengukuran ukuran, dan penentuan sensitivitas data), penerapan *Adaptive Mode Selector* (AMS) berbasis aturan (*rule-based decision*) untuk memilih mode operasi AES (CBC, GCM, atau CTR) sesuai karakteristik data, implementasi lapisan enkripsi (*encryption layer*) AES dan manajemen kunci melalui *Android Keystore*, serta pengujian sistem. Implementasi dilakukan menggunakan *Android Studio* dengan bahasa pemrograman Kotlin dan *library* kriptografi terkait. Pengujian mencakup validasi enkripsi–dekripsi, keamanan pengelolaan kunci (*keystore test*), adaptivitas pemilihan mode AES, serta evaluasi performa (waktu eksekusi, *throughput*, dan penggunaan memori) guna menilai efektivitas, keamanan, dan efisiensi model enkripsi hybrid yang dikembangkan.

HASIL & PEMBAHASAN

Implementasi Dan Pengujian.

1. Implementasi Enkripsi Hybrid AES dan Android Keystore

Tahap awal implementasi dimulai dengan pembuatan kunci AES yang disimpan secara aman di dalam Android Keystore System. Kunci ini tidak pernah diekspor keluar dari sistem dan hanya dapat digunakan melalui API kriptografi Android.



```
class KeystoreManager {
    companion object {
        private const val ANDROID_KEYSTORE = "AndroidKeyStore"
        private const val AES_KEY_ALIAS = "HybridAESKey"
    }

    private val keyStore: KeyStore = KeyStore.getInstance(ANDROID_KEYSTORE).apply { load(null) }

    fun hasAesKeyInKeystore(): Boolean {
        return keyStore.containsAlias(AES_KEY_ALIAS)
    }

    fun generateAesKeyInKeystore(): Boolean {
        return try {
            val keyGenerator =
                KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES, ANDROID_KEYSTORE)
            val builder = KeyGenParameterSpec.Builder(
                AES_KEY_ALIAS,
                KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
            ).apply {
                setKeySize(256)
                setBlockModes(
                    KeyProperties.BLOCK_MODE_GCM,
                    KeyProperties.BLOCK_MODE_CTR,
                    KeyProperties.BLOCK_MODE_CBC
                )
                setEncryptionPaddings(
                    KeyProperties.ENCRYPTION_PADDING_NONE,
                    KeyProperties.ENCRYPTION_PADDING_PKCS7
                )
                setUserAuthenticationRequired(false)
            }
            keyGenerator.init(builder.build())
            keyGenerator.generateKey()
            true
        } catch (e: Exception) {
            e.printStackTrace()
            false
        }
    }

    fun getAesKeyFromKeystore(): SecretKey? {
        val entry = keyStore.getEntry(AES_KEY_ALIAS, null) as? KeyStore.SecretKeyEntry
        return entry?.secretKey
    }
```

Gambar 1. Generate AES dengan *Android Key Store*

Pada tahap enkripsi, data masukan berupa teks atau citra diproses sebagai array byte. Sistem kemudian menentukan mode operasi AES secara adaptif berdasarkan ukuran data:

a) Data berukuran kecil → AES-CBC
b) Data berukuran menengah → AES-GCM
c) Data berukuran besar → AES-CTR

Sebagai contoh, ketika pengguna memilih sebuah citra berukuran 100 KB, sistem secara otomatis memilih mode AES-GCM. Data kemudian dienkripsi menggunakan kunci dari Keystore, menghasilkan *ciphertext* dan *initialization vector (IV)* yang disimpan bersama metadata dalam berkas JSON.

```
private fun chooseModeForSize(size: Int): String =
    when {
        size < 100 * 1024 -> "CBC"
        size < 5 * 1024 * 1024 -> "GCM"
        else -> "CTR"
    }
```

Gambar 2. Implementasi AES secara adaptif

```
private fun encryptWithKeystoreKey(
    plain: ByteArray,
    key: SecretKey,
    mode: String
): CipherResult {
    return when (mode) {
        "GCM" -> {
            val cipher = Cipher.getInstance("AES/GCM/NoPadding")
            try {
                cipher.init(Cipher.ENCRYPT_MODE, key)
            } catch (ex: InvalidAlgorithmParameterException) {
                val iv = randomBytes(12)
                val spec = GCMParameterSpec(128, iv)
                cipher.init(Cipher.ENCRYPT_MODE, key, spec)
            }
            val ct = cipher.doFinal(plain)
            val iv = cipher.iv // IV produced by Keystore (or by fallback)
            CipherResult(
                Base64.encodeToString(ct, Base64.NO_WRAP),
                Base64.encodeToString(iv, Base64.NO_WRAP),
                "GCM"
            )
        }

        "CBC" -> {
            val cipher = Cipher.getInstance("AES/CBC/PKCS7Padding")
            try {
                cipher.init(Cipher.ENCRYPT_MODE, key)
            } catch (ex: InvalidAlgorithmParameterException) {
                val iv = randomBytes(16)
                cipher.init(Cipher.ENCRYPT_MODE, key, IvParameterSpec(iv))
            }
            val ct = cipher.doFinal(plain)
            val iv =
                cipher.iv ?: /* if provider didn't expose iv, generate/track it */ randomBytes(
                    16
                )
            CipherResult(
                Base64.encodeToString(ct, Base64.NO_WRAP),
                Base64.encodeToString(iv, Base64.NO_WRAP),
                "CBC"
            )
        }

        "CTR" -> {
            val cipher = Cipher.getInstance("AES/CTR/NoPadding")
            try {
                cipher.init(Cipher.ENCRYPT_MODE, key)
            } catch (ex: InvalidAlgorithmParameterException) {
                val iv = randomBytes(16)
                cipher.init(Cipher.ENCRYPT_MODE, key, IvParameterSpec(iv))
            }
            val ct = cipher.doFinal(plain)
            val iv = cipher.iv ?: randomBytes(16)
            CipherResult(
                Base64.encodeToString(ct, Base64.NO_WRAP),
                Base64.encodeToString(iv, Base64.NO_WRAP),
                "CTR"
            )
        }

        else -> throw IllegalArgumentException("Mode not supported")
    }
}
```

Gambar 3. Implementasi Enkripsi AES Berbasis Android Keystore System

2. Implementasi Enkripsi AES Tanpa Android Keystore System (AES-Only)

Pada skema AES-only, kunci enkripsi AES dihasilkan dan dikelola langsung oleh aplikasi menggunakan objek SecretKeySpec. Kunci tersebut disimpan di memori aplikasi selama proses enkripsi dan dekripsi berlangsung. Pendekatan ini merepresentasikan praktik enkripsi konvensional yang masih banyak ditemukan pada aplikasi Android.

```
fun generateRawAesKey(): SecretKey {
    val keyBytes = ByteArray(32)
    SecureRandom().nextBytes(keyBytes)
    return SecretKeySpec(keyBytes, "AES")
}
```

Gambar 4. Generate AES Key

```
fun encryptBytesAESOnly(
    plain: ByteArray,
    mode: String,
    lastRawKey: SecretKey
): CipherResult {

    return when (mode) {

        "GCM" -> {
            val cipher = Cipher.getInstance("AES/GCM/NoPadding")
            val iv = ByteArray(12).also { SecureRandom().nextBytes(it) }
            val spec = GCMParameterSpec(128, iv)

            cipher.init(Cipher.ENCRYPT_MODE, lastRawKey, spec)
            val ct = cipher.doFinal(plain)

            CipherResult(
                Base64.encodeToString(ct, Base64.NO_WRAP),
                Base64.encodeToString(iv, Base64.NO_WRAP),
                "GCM"
            )
        }

        "CBC" -> {
            val cipher = Cipher.getInstance("AES/CBC/PKCS7Padding")
            val iv = ByteArray(16).also { SecureRandom().nextBytes(it) }

            cipher.init(Cipher.ENCRYPT_MODE, lastRawKey, IvParameterSpec(iv))
            val ct = cipher.doFinal(plain)
            CipherResult(
                Base64.encodeToString(ct, Base64.NO_WRAP),
                Base64.encodeToString(iv, Base64.NO_WRAP),
                "CBC"
            )
        }

        "CTR" -> {
            val cipher = Cipher.getInstance("AES/CTR/NoPadding")
            val iv = ByteArray(16).also { SecureRandom().nextBytes(it) }

            cipher.init(Cipher.ENCRYPT_MODE, lastRawKey, IvParameterSpec(iv))
            val ct = cipher.doFinal(plain)

            CipherResult(
                Base64.encodeToString(ct, Base64.NO_WRAP),
                Base64.encodeToString(iv, Base64.NO_WRAP),
                "CTR"
            )
        }

        else -> throw IllegalArgumentException("Unsupported mode")
    }
}
```

Gambar 5. mplementasi Enkripsi AES-Only Menggunakan SecretKeySpec Tanpa Android Keystore System

Secara konseptual, implementasi AES-only memiliki kompleksitas yang lebih rendah dan performa eksekusi yang lebih. Namun, pendekatan ini memiliki kelemahan utama pada aspek keamanan kunci, karena kunci berada di memori aplikasi dan berpotensi diekstraksi melalui teknik reverse engineering atau memory inspection.

Perbedaan utama antara implementasi AES-only dan AES berbasis Android Keystore System terletak pada mekanisme pengelolaan kunci. Pada AES-only, kunci disimpan di memori aplikasi dan diinisialisasi secara langsung, sehingga memiliki performa lebih tinggi namun rentan terhadap ekstraksi. Sebaliknya, pada implementasi Android Keystore System, kunci disimpan dalam lingkungan aman berbasis perangkat keras (TEE) dan tidak dapat diekspor, sehingga memberikan peningkatan signifikan pada aspek keamanan dengan konsekuensi overhead performa yang masih dapat diterima.

3. Implementasi Proses Dekripsi dengan Android Keystore System

Proses dekripsi dilakukan dengan membaca metadata dari berkas JSON, yang mencakup mode AES, IV, dan ciphertext. Sistem kemudian mengambil kembali kunci AES dari Android Keystore System dan melakukan proses dekripsi sesuai dengan mode yang tersimpan.

Sebagai contoh, ketika file terenkripsi dengan mode AES-GCM dipilih, sistem akan melakukan dekripsi menggunakan parameter GCM yang sesuai. Hasil dekripsi kemudian diverifikasi dengan menampilkan kembali citra pada komponen ImageView sebagai validasi visual keberhasilan proses dekripsi.

```
fun decryptBytesAES(cipherBase64: String, ivBase64: String, mode: String): ByteArray {
    val secretKey = keystoreManager.getAesKeyFromKeystore()
    if (secretKey != null) {
        return decryptWithKeystoreKey(cipherBase64, ivBase64, secretKey, mode)
    } else {
        throw IllegalStateException("No keystore AES key available for decryption")
    }
}

private fun decryptWithKeystoreKey(
    cipherBase64: String,
    ivBase64: String,
    key: SecretKey,
    mode: String
): ByteArray {
    val ct = Base64.decode(cipherBase64, Base64.NO_WRAP)
    val iv = Base64.decode(ivBase64, Base64.NO_WRAP)
    return when (mode) {
        "GCM" -> {
            val cipher = Cipher.getInstance("AES/GCM/NoPadding")
            val spec = GCMParameterSpec(128, iv)
            cipher.init(Cipher.DECRYPT_MODE, key, spec)
            cipher.doFinal(ct)
        }

        "CBC" -> {
            val cipher = Cipher.getInstance("AES/CBC/PKCS7Padding")
            cipher.init(Cipher.DECRYPT_MODE, key, IvParameterSpec(iv))
            cipher.doFinal(ct)
        }

        "CTR" -> {
            val cipher = Cipher.getInstance("AES/CTR/NoPadding")
            cipher.init(Cipher.DECRYPT_MODE, key, IvParameterSpec(iv))
            cipher.doFinal(ct)
        }

        else -> throw IllegalArgumentException("Mode not supported")
    }
}
```

Gambar 6. Implementasi Proses Dekripsi Data Menggunakan Android Keystore System

4. Implementasi Proses Dekripsi AES-Only

Pada skema AES-only, kunci AES tidak disimpan di Android Keystore System, melainkan dihasilkan dan disimpan sementara di memori aplikasi

menggunakan SecretKeySpec. Proses dekripsi dilakukan sepenuhnya di sisi aplikasi dan implementasi ini bertujuan untuk mengevaluasi perbedaan mekanisme pengelolaan kunci.

```
fun decryptWithRawKey(
    cipherBase64: String,
    ivBase64: String,
    key: SecretKey,
    mode: String
): ByteArray {

    val ct = Base64.decode(cipherBase64, Base64.NO_WRAP)
    val iv = Base64.decode(ivBase64, Base64.NO_WRAP)

    return when (mode) {
        "GCM" -> {
            val cipher = Cipher.getInstance("AES/GCM/NoPadding")
            val spec = GCMParameterSpec(128, iv)
            cipher.init(Cipher.DECRYPT_MODE, key, spec)
            cipher.doFinal(ct)
        }

        "CBC" -> {
            val cipher = Cipher.getInstance("AES/CBC/PKCS7Padding")
            cipher.init(Cipher.DECRYPT_MODE, key, IvParameterSpec(iv))
            cipher.doFinal(ct)
        }

        "CTR" -> {
            val cipher = Cipher.getInstance("AES/CTR/NoPadding")
            cipher.init(Cipher.DECRYPT_MODE, key, IvParameterSpec(iv))
            cipher.doFinal(ct)
        }

        else -> throw IllegalArgumentException("Unsupported AES mode")
    }
}
```

Gambar 7. Implementasi Proses Dekripsi Data Menggunakan AES-Only Tanpa Android Keystore System

```
enum class DecryptTestMode {
    NORMAL,                 // decrypt normal
    WRONG_IV,               // IV dimodifikasi
    CORRUPTED_CIPHERTEXT,   // ciphertext dimodifikasi
    WRONG_MODE,             // mode AES salah
    MISSING_KEY             // key dihapus
}
```

Gambar 8. Implementasi Pengujian Error Handling

Untuk menguji ketahanan sistem terhadap kesalahan dan manipulasi data, dilakukan pengujian error handling menggunakan enumerasi DecryptTestMode. Setiap mode pengujian mensimulasikan kondisi kesalahan tertentu, antara lain:

1. **WRONG_IV**: satu byte IV dimodifikasi

```
DecryptTestMode.WRONG_IV -> {
    val iv = Base64.decode(ivBase64, Base64.NO_WRAP)
    iv[0] = (iv[0].toInt() xor 0xFF).toByte()
    ivBase64 = Base64.encodeToString(iv, Base64.NO_WRAP)
}
```

Gambar 9. satu byte IV dimodifikasi

2. **CORRUPTED_CIPHERTEXT**: satu byte ciphertext dimodifikasi

```
DecryptTestMode.CORRUPTED_CIPHERTEXT -> {
    val cipherBytes = Base64.decode(cipherBase64, Base64.NO_WRAP)

    cipherBytes[0] = (cipherBytes[0].toInt() xor 0xFF).toByte()

    cipherBase64 = Base64.encodeToString(cipherBytes, Base64.NO_WRAP)
}
```

Gambar 10. satu byte ciphertext dimodifikasi

3. **WRONG_MODE**: mode AES diubah menjadi tidak sesuai

```
DecryptTestMode.WRONG_MODE -> {
    mode = when (cipherToUse.mode) {
        "GCM" -> "CBC"
        "CBC" -> "GCM"
        else -> "GCM"
    }
}
```

Gambar 11. mode AES diubah menjadi tidak sesuai

4. **MISSING_KEY**: kunci AES dihapus dari Keystore

```
DecryptTestMode.MISSING_KEY -> {
    val ks = KeyStore.getInstance("AndroidKeyStore").apply { load(null) }
    ks.deleteEntry("HybridAESKey")
}
```

Gambar 12. kunci AES dihapus dari Keystore

Sebagai contoh pada pengujian WRONG_IV, sistem menggunakan IV yang telah dimodifikasi satu byte dari IV asli. Proses dekripsi kemudian dijalankan dan sistem diharapkan menolak hasil dekripsi karena kegagalan autentikasi atau padding.

Hasil Akhir Pengujian

1. Hasil Pengujian Fungsional

Hasil pengujian fungsional menunjukkan bahwa sistem Hybrid Model Encryption yang diimplementasikan mampu melakukan proses enkripsi dan dekripsi data teks dan citra dengan baik. Pengujian dilakukan pada beberapa skenario penggunaan dengan variasi jenis data dan mode operasi AES yang dipilih secara adaptif oleh sistem. Ringkasan hasil pengujian fungsional ditunjukkan pada tabel dibawah.

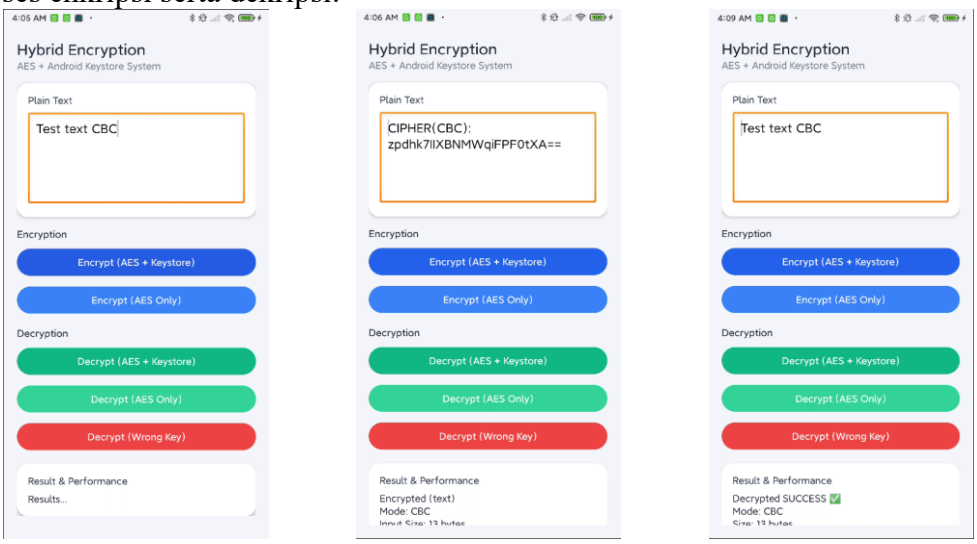
Tabel 1. Hasil pengujian fungsional

Jenis Data	Mode AES	Enkripsi	Dekripsi	Keterangan
Teks	CBC	Berhasil	Berhasil	Sesuai
Teks	GCM	Berhasil	Berhasil	Sesuai
Citra	GCM	Berhasil	Berhasil	Ditampilkan
Citra	CTR	Berhasil	Berhasil	Ditampilkan

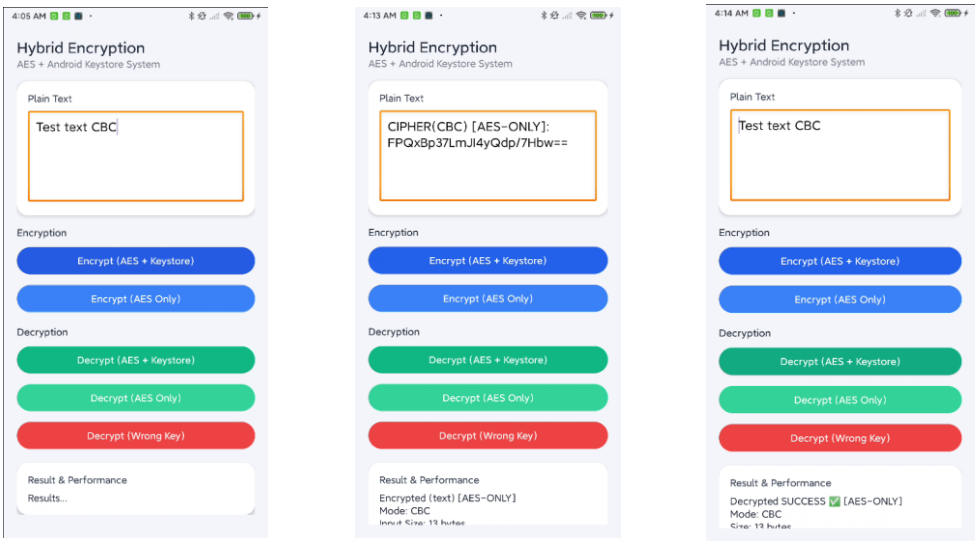
Untuk memastikan bahwa data pada Tabel 1 benar-benar berasal dari hasil implementasi sistem, setiap skenario pengujian didukung dengan bukti visual berupa tangkapan layar (*screenshot*) dari aplikasi yang dikembangkan.

a. Pengujian Data Teks dengan Mode AES-CBC

Pengujian ini dilakukan dengan memasukkan data teks berukuran kecil ke dalam aplikasi. Sistem secara otomatis memilih mode **AES-CBC** dan melakukan proses enkripsi serta dekripsi.



Gambar 13. Menggunakan AES dengan Android Key Store

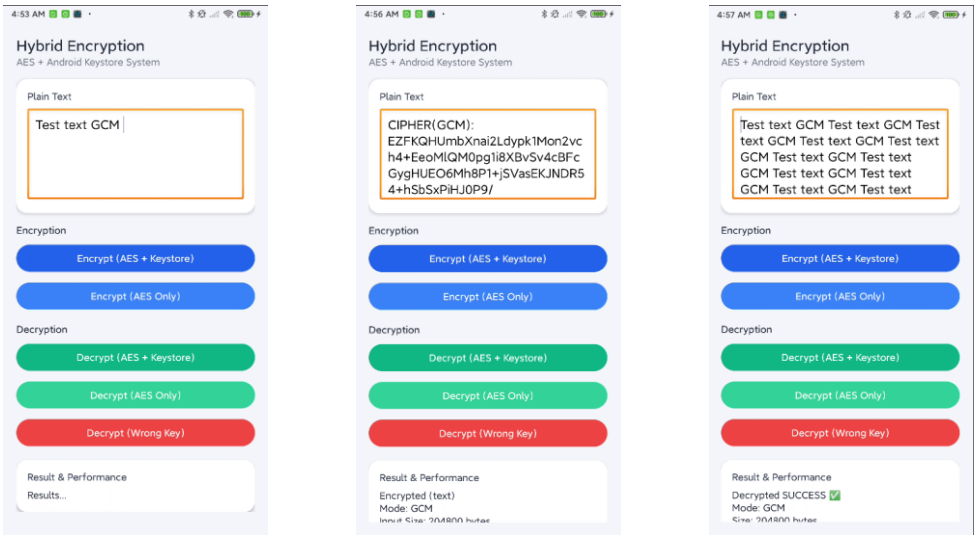


Gambar 14. Menggunakan AES Tanpa Android Key Store

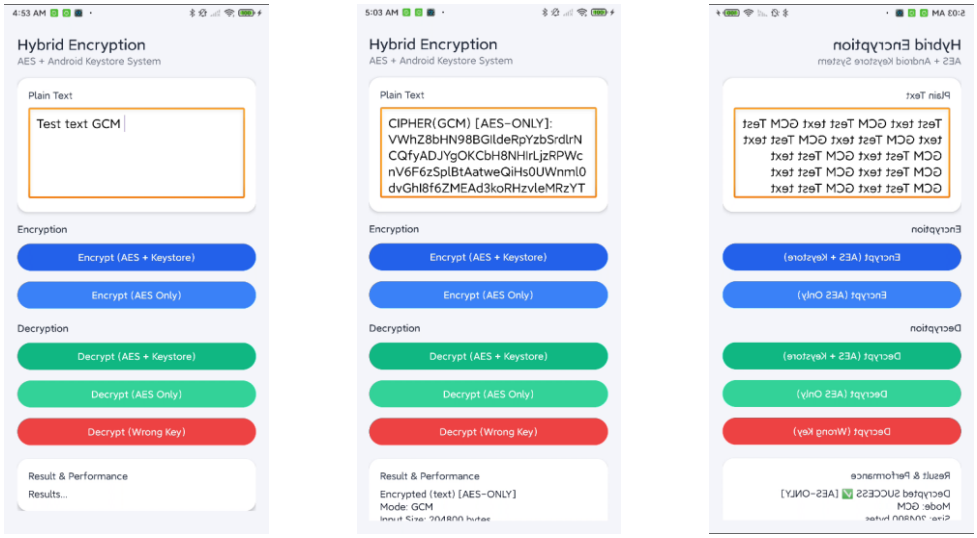
Berdasarkan hasil pengujian, data teks berhasil dienkripsi dan didekripsi kembali tanpa perubahan isi, sehingga hasil pengujian dinyatakan sesuai dengan perancangan sistem.

b. Pengujian Data Teks dengan Mode AES-GCM

Pada pengujian ini digunakan data teks berukuran 200Kb dan dalam pengujian ini ditambahkan script untuk membuat text menjadi 200 Kb. Sistem memilih mode AES-GCM untuk memberikan keseimbangan antara keamanan dan performa.



Gambar 15. Menggunakan AES dengan Android Key Store

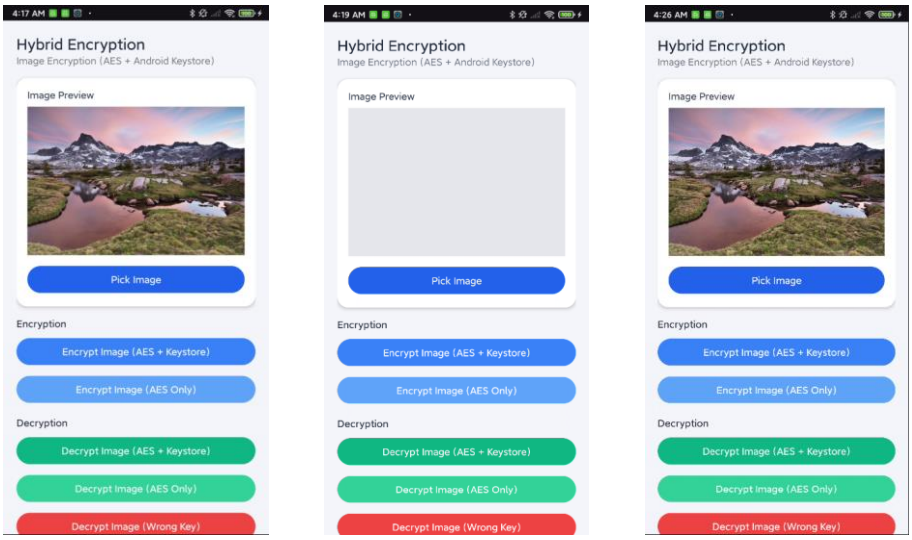


Gambar 16. Menggunakan AES Tanpa Android Key Store

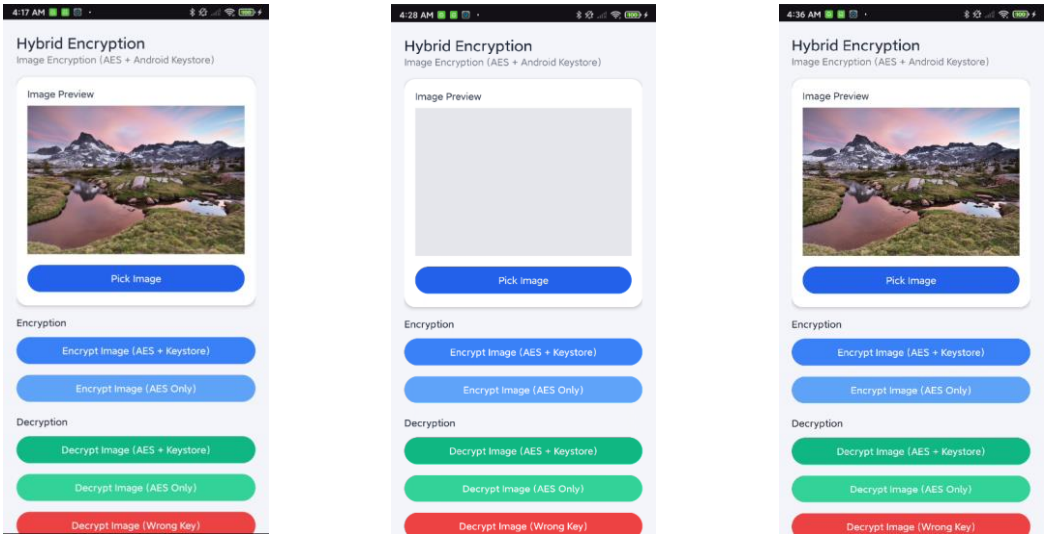
Hasil pengujian menunjukkan bahwa proses enkripsi dan dekripsi berjalan dengan baik serta data kembali identik dengan data awal.

c. Pengujian Data Citra dengan Mode AES-GCM

Pengujian ini dilakukan dengan menggunakan data citra berformat PNG berukuran menengah. Setelah proses enkripsi dilakukan, citra tidak dapat ditampilkan hingga dilakukan proses dekripsi.



Gambar 17. Menggunakan AES dengan Android Key Store

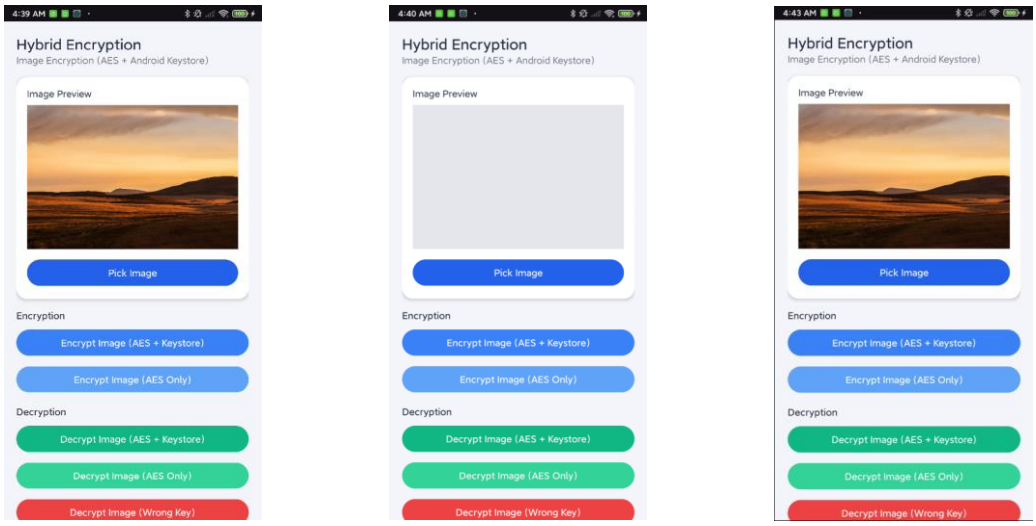


Gambar 18. Menggunakan AES Tanpa Android Key Store

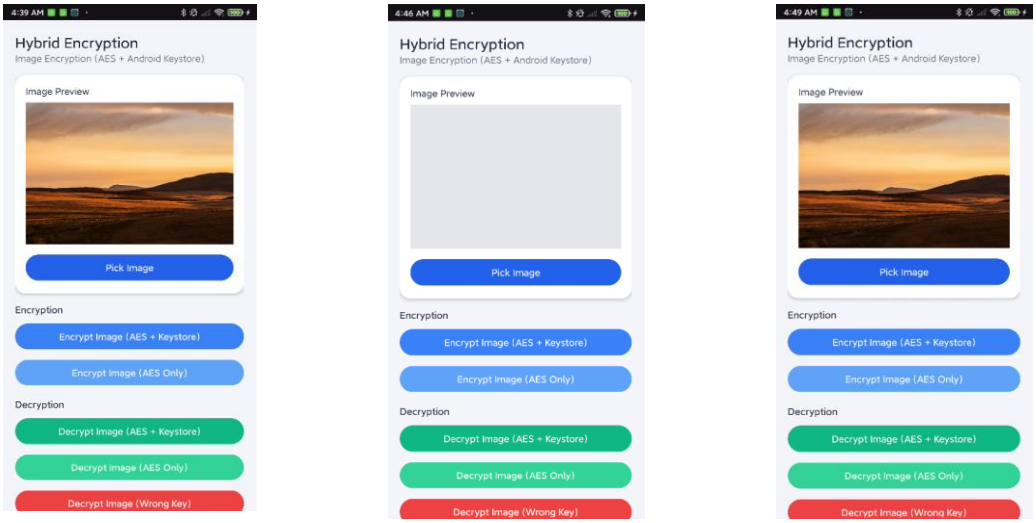
Citra hasil dekripsi berhasil ditampilkan kembali pada *ImageView* dan memiliki tampilan yang sama dengan citra asli sebelum proses enkripsi.

d. Pengujian Data Citra dengan Mode AES-CTR

Pengujian terakhir dilakukan menggunakan data citra berukuran besar. Sistem memilih mode AES-CTR untuk meningkatkan performa proses enkripsi dan dekripsi.



Gambar 19. Menggunakan AES dengan Android Key Store



Gambar 20. Menggunakan AES Tanpa Android Key Store

Hasil pengujian menunjukkan bahwa mode AES-CTR mampu menangani data berukuran besar dengan baik dan citra dapat ditampilkan kembali tanpa mengalami kerusakan.

2. Hasil Pengujian Performa

Pengujian performa pada penelitian ini difokuskan pada pengukuran waktu eksekusi dan throughput sebagai indikator efisiensi implementasi, bukan sebagai analisis ketahanan kriptografi algoritma AES yang telah distandarisasi secara global

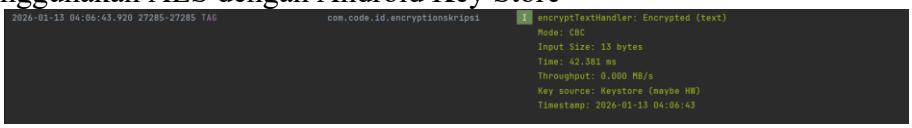
Pengujian performa dilakukan dengan mengukur waktu eksekusi dan throughput sistem. Berdasarkan distribusi waktu eksekusi aktual, klasifikasi performa ditetapkan sebagai berikut:

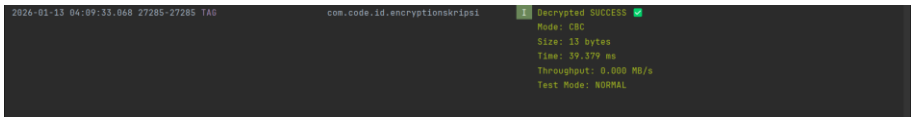
Tabel 2. Hasil Pengujian Performa

Kategori	Rentang Waktu Eksekusi
Rendah	< 100 ms
Sedang	100 – 1000 ms
Tinggi	> 1000 ms

1. TEXT CBC

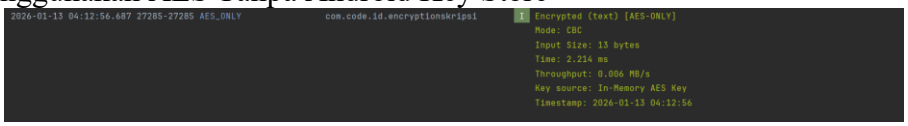
a. Menggunakan AES dengan Android Key Store





Encrypt time: **42,381 ms → RENDAH**
Decrypt time: **39,379 ms → RENDAH**
Throughput: **0,000 MB/s → sangat kecil (ukuran data sangat kecil: 13 bytes)**

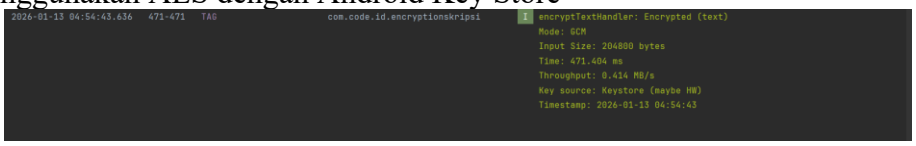
b. Menggunakan AES Tanpa Android Key Store

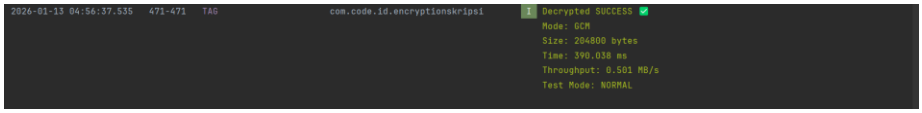


Encrypt time: **2,214 ms → RENDAH**
Decrypt time: **2,352 ms → RENDAH**
Throughput: **0,006 MB/s → kecil (ukuran data sangat kecil)**

2. TEXT GCM

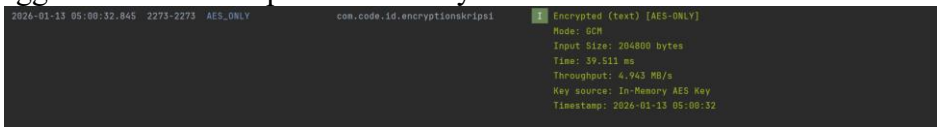
a. Menggunakan AES dengan Android Key Store





Encrypt time: **471,404 ms → SEDANG**
Decrypt time: **390,038 ms → SEDANG**
Throughput: **0,414 – 0,501 MB/s → sedang**

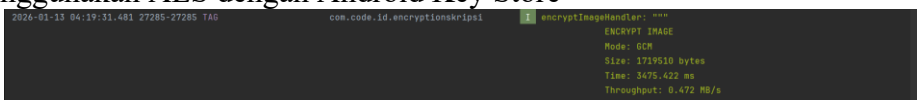
b. Menggunakan AES Tanpa Android Key Store

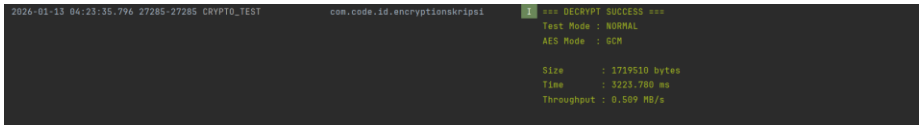


Encrypt time: **39,511 ms → RENDAH**
Decrypt time: **20,366 ms → RENDAH**
Throughput: **4,943 – 9,590 MB/s → tinggi**

3. CITRA GCM

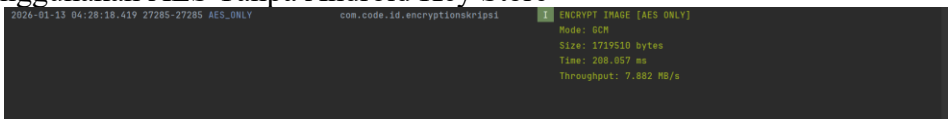
a. Menggunakan AES dengan Android Key Store

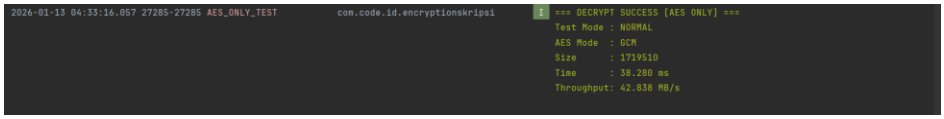




Encrypt time: **3475,422 ms → TINGGI**
Decrypt time: **3223,780 ms → TINGGI**
Throughput: **0,472 MB/s → rendah–sedang**

b. Menggunakan AES Tanpa Android Key Store

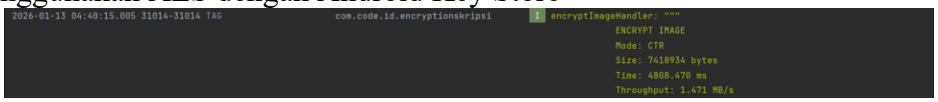


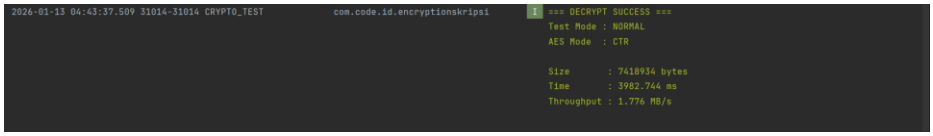


Encrypt time: **208,057 ms → SEDANG**
Decrypt time: **38,280 ms → RENDAH**
Throughput: **7,882 – 42,838 MB/s → tinggi**

4. CITRA CTR

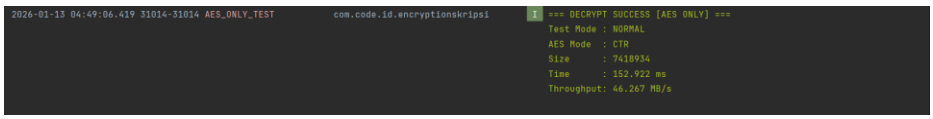
a. Menggunakan AES dengan Android Key Store





Encrypt time: **4808,470 ms → TINGGI**
Decrypt time: **3982,744 ms → TINGGI**
Throughput: **1,471 – 1,776 MB/s → sedang**

b. Menggunakan AES Tanpa Android Key Store



Encrypt time: **778,746 ms → SEDANG**
Decrypt time: **152,922 ms → SEDANG**
Throughput: **9,085 – 46,267 MB/s → tinggi**

Tabel 3. hasil pengujian performa

Jenis Data	Ukuran	Mode AES	Waktu (ms)	Throughput (MB/s)
Jenis Data	**Mode AES**	**Encrypt Time (ms)**	**Decrypt Time (ms)**	**Kategori Waktu**
Teks	CBC	39,209	44,341	Rendah
Teks	GCM	440,118	401,830	Sedang
Citra	GCM	2558,180	2294,020	Tinggi

Berdasarkan hasil pengujian performa, waktu eksekusi proses enkripsi dan dekripsi diklasifikasikan ke dalam tiga kategori, yaitu rendah (<100 ms), sedang (100–1000 ms), dan tinggi (>1000 ms). Klasifikasi ini ditentukan berdasarkan distribusi nilai waktu eksekusi yang diperoleh dari hasil pengujian aktual pada aplikasi.

Hasil pengujian menunjukkan bahwa AES-CBC pada data teks memiliki waktu eksekusi paling rendah karena ukuran data yang kecil. Sebaliknya, AES-GCM menunjukkan peningkatan waktu eksekusi karena integritas data.

Pada data citra berukuran besar, AES-GCM menghasilkan waktu eksekusi yang tinggi dengan throughput sedang, sedangkan AES-CTR memberikan throughput tertinggi meskipun waktu eksekusinya juga berada pada kategori tinggi. Hal ini menunjukkan bahwa mode CTR lebih sesuai untuk data berukuran besar yang membutuhkan performa tinggi.

3. Hasil Pengujian Error Handling

Pengujian error handling dilakukan untuk mengevaluasi ketahanan sistem terhadap kesalahan parameter dan manipulasi data terenkripsi. Pengujian ini bertujuan memastikan bahwa sistem mampu mendeteksi kondisi dekripsi yang tidak valid serta menolak proses dekripsi tanpa menyebabkan kegagalan aplikasi (*application crash*).

Tabel 4. Hasil Pengujian Error Handling

Mode Uji	Hasil Sistem	Keterangan
NORMAL	Berhasil	Data didekripsi normal
WRONG_IV	Gagal	Autentikasi gagal
CORRUPTED_CIPHERTEXT	Gagal	Integritas data rusak
WRONG_MODE	Gagal	Mode tidak sesuai
MISSING_KEY	Gagal	Kunci tidak ditemukan

1. Pengujian NORMAL (Dekripsi Valid)

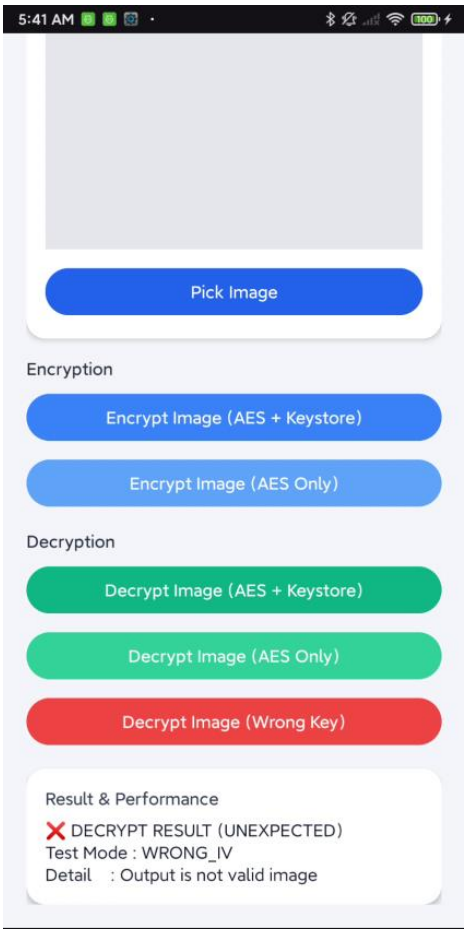
Pada pengujian NORMAL, parameter dekripsi seperti mode AES, IV, ciphertext, dan kunci berada dalam kondisi valid. Sistem berhasil melakukan proses dekripsi dan menampilkan kembali data citra pada *ImageView*.

Gambar 21. Pengujian NORMAL

Pengujian ini membuktikan bahwa sistem berfungsi dengan benar ketika seluruh parameter sesuai.

2. Pengujian WRONG_IV

Pengujian WRONG_IV dilakukan dengan memodifikasi satu byte pada *Initialization Vector (IV)* sebelum proses dekripsi dijalankan. Akibat perubahan tersebut, proses autentikasi gagal dan sistem menolak hasil dekripsi.

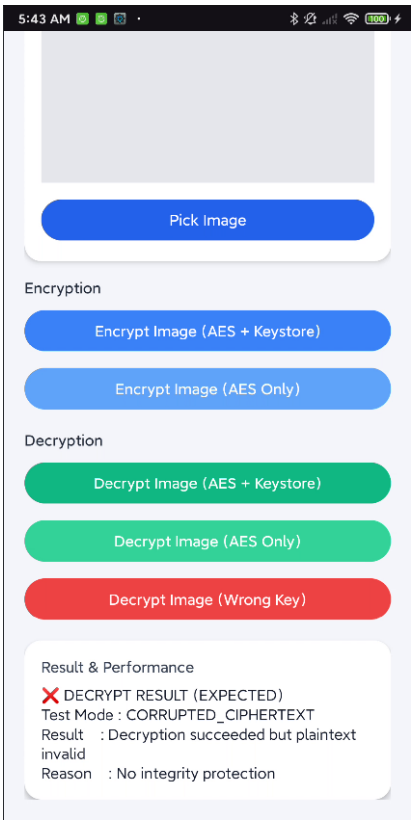


Gambar 22. Pengujian WRONG_IV

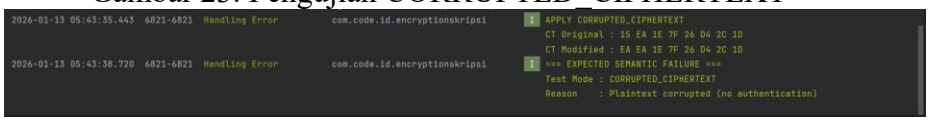
Hasil ini menunjukkan bahwa sistem mampu mendeteksi perubahan IV dan menjaga integritas data.

3. Pengujian CORRUPTED_CIPHERTEXT

Pada pengujian ini, sebagian ciphertext dimodifikasi untuk mensimulasikan kerusakan data terenkripsi. Sistem gagal melakukan dekripsi karena integritas data tidak dapat diverifikasi.



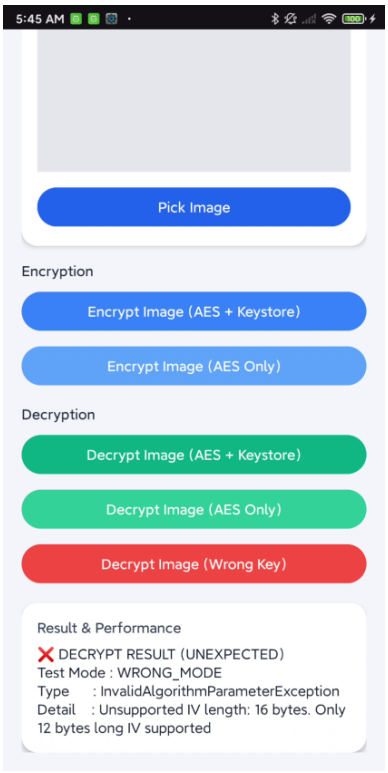
Gambar 23. Pengujian CORRUPTED_CIPHERTEXT



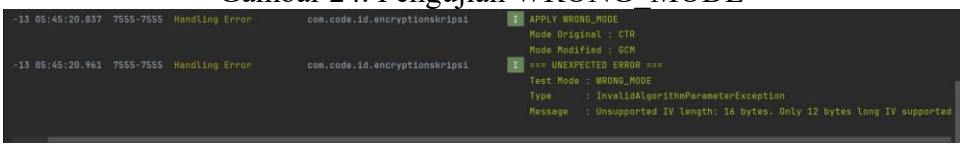
Pengujian ini membuktikan bahwa sistem tidak mengizinkan data terenkripsi yang telah dimanipulasi untuk didekripsi.

4. Pengujian WRONG_MODE

Pengujian WRONG_MODE dilakukan dengan menggunakan mode AES yang berbeda dari metadata enkripsi. Sistem menolak proses dekripsi karena parameter algoritma tidak sesuai.



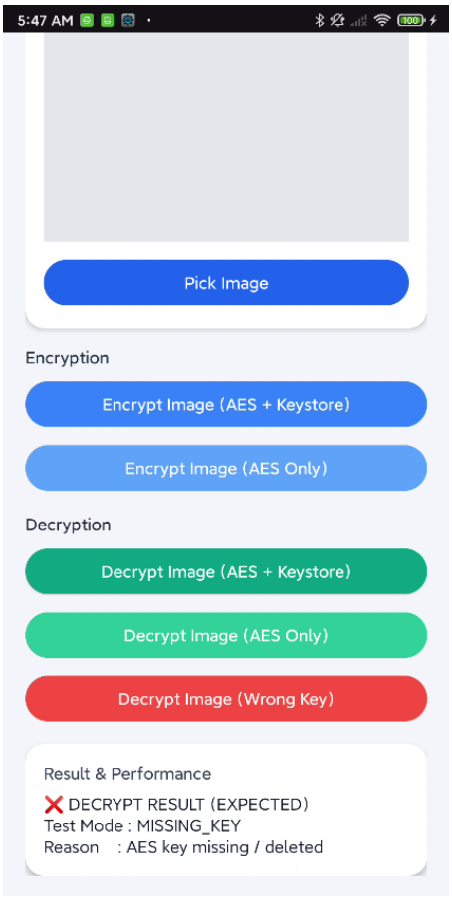
Gambar 24. Pengujian WRONG_MODE



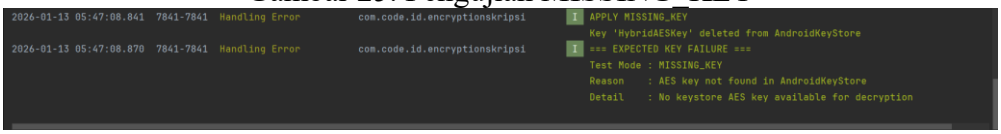
Hasil ini menunjukkan bahwa sistem hanya dapat mendekripsi data menggunakan mode AES yang sesuai dengan proses enkripsi awal.

5. Pengujian MISSING_KEY

Pengujian MISSING_KEY dilakukan dengan menghapus kunci AES dari Android Keystore System sebelum proses dekripsi. Sistem gagal melakukan dekripsi karena kunci tidak ditemukan.



Gambar 25. Pengujian MISSING_KEY



Pengujian ini membuktikan bahwa sistem sepenuhnya bergantung pada Android Keystore System untuk pengelolaan kunci dan tidak menyimpan kunci secara tidak aman.

KESIMPULAN

Berdasarkan hasil penerapan dan pengujian yang telah dilakukan pada penelitian dengan judul "Implementasi Model Enkripsi Hybrid Adaptif AES dan Android Keystore untuk Pengamanan Data Teks dan Citra Android", dapat disimpulkan bahwa model enkripsi hybrid adaptif berbasis algoritma *Advanced Encryption Standard* (AES) dan *Android Keystore System* dapat diterapkan secara efektif untuk pengamanan data teks dan citra pada perangkat Android. Penelitian ini berfokus pada pengujian perilaku dan performa metode enkripsi yang diterapkan, bukan pada pengembangan sistem baru. Hasil eksperimen menunjukkan bahwa penggunaan *Android Keystore System* sebagai media penyimpanan kunci enkripsi mampu meningkatkan keamanan pengelolaan kunci, karena kunci tidak dapat diakses atau diekstraksi secara langsung dari aplikasi.

Dengan demikian, penerapan metode ini terbukti mampu mengurangi risiko kebocoran kunci enkripsi pada lingkungan aplikasi *mobile* Android. Selain itu, hasil

pengujian performa memperlihatkan bahwa mekanisme AES adaptif yang memilih mode operasi CBC, GCM, atau CTR berdasarkan karakteristik data mampu memberikan keseimbangan antara keamanan dan efisiensi. Mode AES-CBC menunjukkan waktu eksekusi paling rendah pada data teks berukuran kecil, sedangkan AES-GCM memberikan tingkat keamanan yang lebih tinggi dengan konsekuensi waktu proses yang sedang. Untuk data citra berukuran besar, AES-CTR menghasilkan *throughput* tertinggi sehingga lebih efisien dari sisi performa. Hal ini membuktikan bahwa pendekatan adaptif lebih optimal dibandingkan penggunaan satu mode AES secara statis dalam berbagai kondisi data. Hasil pengujian *error handling* juga menunjukkan bahwa metode enkripsi yang diterapkan memiliki ketahanan yang baik terhadap berbagai skenario kesalahan dan manipulasi data terenkripsi, seperti kesalahan *initialization vector* (IV), kerusakan *ciphertext*, ketidaksesuaian mode AES, serta ketiadaan kunci enkripsi. Seluruh skenario tersebut berhasil terdeteksi dan ditangani oleh sistem tanpa menyebabkan kegagalan aplikasi.

DAFTAR PUSTAKA

Ahn, N.-Y., & Lee, D. H. (2017). Countermeasure against side-channel attack in shared memory of TrustZone. *PLOS One / arXiv preprint*. https://doi.org/10.48550/arXiv.1705.08279

Altuwaijri, H., & Ghouzali, S. (2018). Android data storage security: A review. *Journal of King Saud University – Computer and Information Sciences, 30*(4). https://doi.org/10.1016/j.jksuci.2018.07.004

Assidiq, M. L., Mahardika, F., & Santika, D. (2024). Implementasi algoritma kriptografi AES dan SHA-3 dalam mengamankan data sensitif pengguna pada website transaksi. *Simpatik: Jurnal Sistem Informasi dan Informatika, 4*(1).

Bader, A. S., & Sagheer, A. M. (2018). Modification on AES-GCM to increment ciphertext randomness. *International Journal of Mathematical Sciences and Computing, 4*(4), 34–40. https://doi.org/10.5815/ijmsc.2018.04.03

Blessing, J., Anderson, R. J., & Beresford, A. R. (2025). KeyDroid: A large-scale analysis of secure key storage in Android apps. *arXiv preprint* (arXiv:2507.07927v1).

Buhari, B. A., et al. (2025). Performance and security analysis of symmetric data encryption algorithms: AES, 3DES and Blowfish. *International Journal of Advanced Networking and Applications, 16*(04), 6473–6486. https://doi.org/10.35444/IJANA.2024.16404

Chauhan, G. S., et al. (2025). Securing data transmission and storage in cloud computing using hybrid AES-256 and RSA encryption. *International Journal of Science and Engineering Applications, 14*(03). https://doi.org/10.7753/IJSEA1403.1013

Durge, R. S., & Deshmukh, V. M. (2025). Securing cloud data: A hybrid encryption approach with RSA and AES for enhanced security and performance. *Journal of Integrated Science and Technology, 13*(3). https://doi.org/10.62110/sciencein.jist.2025.v13.1060

Ebadollahyvahed, P. (2024). Benchmarking performance of hashing (SHA), symmetric (AES-GCM), and asymmetric (ECC) encryption algorithms….

ResearchGate Technical Publication. https://doi.org/10.13140/RG.2.2.25927.82083

Fajar, M., Kambodji, A. B., & Musdar, I. A. (2023). Implementasi algoritma advanced encryption standard untuk pengamanan data pengguna aplikasi media sosial VirCle. *Jurnal Algoritma, 20*(2), 398–409.

Hamad, N. J., Abdulhameed, A. A., & Ali, M. H. (2024). Implementing a secure mobile application for cardless transactions using QR code and hybrid AES–ECC encryption. *Arab Journal for Scientific Publishing* (Issue 69).

Kumar, P. R., & Goel, S. (2025). A secure and efficient encryption system based on adaptive and machine learning for securing data in fog computing. *Scientific Reports, 15*, 11654. https://doi.org/10.1038/s41598-025-92245-9

Manikandaprabhu, P., & Samreetha, M. (2024). A review of encryption and decryption of text using the AES algorithm. *International Journal of Scientific Research & Engineering Trends, 10*(2).

Muchamad, R. M., Asriyanik, & Pambudi, A. (2023). Implementasi algoritma AES untuk mengenkripsi DataStore pada aplikasi berbasis Android. *Jurnal MNEMONIC, 6*(1), 55–64.

Muzakir, A. (2015). Implementasi teknik steganografi dengan kriptografi kunci private AES untuk keamanan file gambar berbasis Android. *SNATiM 2015 Conference*. ISSN 2302-3805.

Nurrochim, A. (2025). Implementasi kriptografi AES dan steganografi untuk keamanan data customer dan transaksi di PT Guna Bangun Jaya. *Syntax Literate, 10*(10). https://doi.org/10.2541-0849/2701

Nwatuzie, G., & Kwok, W. J. (2019). Android device file encryption and decryption system. *International Journal of Information Systems and Engineering, 7*(1), 13–30. https://doi.org/10.24924/ijise/2019.04/v7.iss1/13.30

Priandani, N. D., Tolle, H., & Ridok, A. (2015). Pengembangan mobile chatting Facebook terenkripsi berbasis Android dengan menggunakan metode AES. *ResearchGate Publication*. https://doi.org/10.13140/RG.2.1.1589.0004

Rachmat, N., & Samsuryadi. (2019). Performance analysis of 256-bit AES encryption algorithm on Android smartphone. *Journal of Physics: Conference Series, 1196*, 012049. https://doi.org/10.1088/1742-6596/1196/1/012049

Rzayeva, L., Imanberdi, A., Opirskyy, I., Harasymchuk, O., & Abitova, G. (2025). Analysis of technical features of data encryption implementation on SD cards in the Android system. *Scientific Journal of Astana IT University, 21*. https://doi.org/10.37943/21LMQF2486

Saripa. (2023). Implementasi sistem keamanan file menggunakan algoritma AES untuk mengamankan file pribadi. *PISCES: Jurnal Pendidikan Teknik Informatika dan Komputer, 1*(2).

Shin, J., Kim, Y., Park, W., & Park, C. (2013). A method for data access control and key management in mobile cloud storage services. *IEMEK Journal of Embedded Systems and Applications, 8*(6), 303–309. https://doi.org/10.14372/IEMEK.2013.8.6.303

Wahyudi, R., & Romli, M. A. (2023). Android-based patient medical record data security application using AES and RSA. *International Journal of Computer Applications, 185*(40). https://doi.org/10.5120/ijca2023923205

Yang, H. (2022). Application of hybrid encryption algorithm in hardware encryption interface card. *Security and Communication Networks*, Article ID 7794209. https://doi.org/10.1155/2022/7794209