# Perancangan dan Implementasi Skema Kriptografi End-to-End pada Transaksi E-Wallet Menggunakan ECC, SHA-3, dan ECDSA

Mohammad Nugraha Eka Prawira - 13522001[1]
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
[1]13522001@stei.itb.ac.id

***Abstrak*— Pertumbuhan layanan e-wallet di Indonesia meningkatkan kebutuhan akan mekanisme keamanan transaksi yang kuat, efisien, dan mampu memitigasi serangan manipulasi data maupun penyadapan. Pada makalah ini dirancang dan diimplementasikan suatu skema end-to-end cryptographic protocol berbasis Elliptic Curve Cryptography (ECC), fungsi hash SHA-3, dan tanda tangan digital ECDSA. Skema yang diusulkan bertujuan untuk memastikan kerahasiaan, integritas, autentikasi, serta non-repudiation pada proses transaksi e-wallet. Implementasi dilakukan dalam lingkungan simulasi client–server, dengan evaluasi terhadap waktu komputasi, verifikasi tanda tangan, dan resistansi terhadap serangan umum seperti man-in-the-middle, payload tampering, dan replay attack. Hasil pengujian menunjukkan bahwa ECC memberikan kinerja enkripsi yang ringan untuk perangkat mobile, SHA-3 menghasilkan digest yang aman terhadap serangan collision, dan ECDSA mampu mendeteksi modifikasi transaksi secara real-time.**

***Kata Kunci*— ECC, SHA-3, ECDSA, e-wallet, end-to-end security, digital signature.**

## I. PENDAHULUAN

Perkembangan ekonomi digital di Indonesia mendorong penggunaan aplikasi dompet digital (e-wallet) seperti GoPay, OVO, Dana, dan ShopeePay dalam berbagai transaksi sehari-hari, mulai dari pembayaran transportasi, belanja daring, hingga tagihan utilitas. Semakin tingginya ketergantungan masyarakat terhadap e-wallet menjadikan aspek keamanan transaksi sebagai faktor krusial dalam menjaga kepercayaan pengguna dan keberlangsungan ekosistem digital.

Di sisi lain, ekosistem e-wallet tidak lepas dari berbagai ancaman keamanan. Serangan seperti man-in-the-middle (MITM), payload manipulation, pencurian identitas, hingga replay attack dapat terjadi apabila data transaksi dikirimkan tanpa mekanisme perlindungan yang memadai. Pada banyak implementasi, perlindungan keamanan hanya mengandalkan protokol transport seperti TLS. Pendekatan ini memang melindungi data saat berada "di jalur", tetapi tidak selalu menjamin keamanan end-to-end pada level payload. Apabila terjadi kompromi pada server, gateway pembayaran, atau komponen middleware lain, data transaksi tetap berpotensi dimodifikasi tanpa terdeteksi.

Berangkat dari permasalahan tersebut, diperlukan suatu skema kriptografi yang secara eksplisit melindungi payload transaksi sejak dibentuk di sisi client hingga diproses di sisi server. Skema ini idealnya mampu menjamin kerahasiaan pesan, integritas data, autentikasi pengirim, serta sifat non-repudiation sehingga pengguna tidak dapat menyangkal transaksi yang telah ia kirimkan.

Pada makalah ini diusulkan sebuah skema end-to-end cryptographic protocol untuk transaksi e-wallet yang mengombinasikan tiga komponen utama, yaitu Elliptic Curve Cryptography (ECC) untuk enkripsi kunci publik yang efisien, fungsi hash SHA-3 untuk menjamin integritas data, dan Elliptic Curve Digital Signature Algorithm (ECDSA) sebagai mekanisme tanda tangan digital. Ketiga komponen ini dipilih karena memiliki tingkat keamanan yang telah banyak dikaji serta cocok diterapkan pada lingkungan dengan keterbatasan sumber daya, seperti perangkat bergerak.

Kontribusi utama dari makalah ini adalah:

1) Merancang skema kriptografi end-to-end berbasis ECC, SHA-3, dan ECDSA yang dapat diterapkan pada arsitektur transaksi e-wallet.
2) Mengimplementasikan skema tersebut dalam simulasi lingkungan client–server menggunakan bahasa pemrograman Python.
3) Melakukan pengujian terhadap skema yang diusulkan untuk menilai kemampuan deteksi modifikasi payload, pencegahan replay attack, dan mengukur overhead komputasi dari proses hashing, penandatanganan, enkripsi, dan verifikasi.

## II. TEORI DASAR

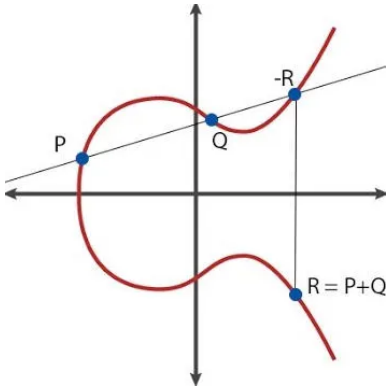
### A. *Elliptic Curve Cryptography (ECC)*

Elliptic Curve Cryptography (ECC) merupakan skema kriptografi kunci publik yang mendasarkan keamanannya pada permasalahan Elliptic Curve Discrete Logarithm

Problem (ECDLP), yaitu sulitnya menemukan bilangan skalar $k$ dari hubungan $Q = kP$ , dengan $P$ dan $Q$ adalah titik pada kurva eliptik. Tidak seperti RSA yang berbasis faktorisasi bilangan bulat besar, ECC bekerja di atas struktur grup aditif dari titik-titik kurva eliptik yang didefinisikan pada medan hingga.

Kelebihan utama ECC adalah tingkat keamanan tinggi dengan ukuran kunci relatif kecil. Sebagai contoh, kunci ECC 256 bit umumnya dianggap setara dengan RSA 3072 bit dalam hal tingkat keamanan. Hal ini membuat ECC sangat menarik untuk implementasi pada perangkat dengan keterbatasan komputasi dan bandwidth, seperti smartphone dan perangkat IoT. Pada konteks e-wallet, sifat ini penting karena proses enkripsi dan dekripsi perlu dilakukan berulang kali untuk setiap transaksi, namun tidak boleh menghasilkan latensi yang signifikan bagi pengguna.

Dalam skema yang diusulkan pada makalah ini, ECC digunakan sebagai dasar untuk membangun mekanisme pertukaran kunci rahasia antara client dan server (melalui ECDH) yang selanjutnya digunakan untuk membentuk kunci simetris sementara guna mengenkripsi payload transaksi. Pendekatan ini serupa dengan konsep ECIES (Elliptic Curve Integrated Encryption Scheme), di mana keamanan bergantung pada kerahasiaan kunci privat dan kekuatan ECDLP.




**Gambar 2.1.** Ilustrasi kurva ECC
Sumber:https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627

## B. *Secure Hash Algorithm 3 (SHA-3)*

SHA-3 adalah keluarga fungsi hash standar yang ditetapkan oleh NIST sebagai penerus generasi SHA-2. Berbeda dengan SHA-2 yang mengandalkan struktur Merkle–Damgård, SHA-3 dibangun di atas fungsi permutasi Keccak dengan pendekatan sponge construction. Hal ini menjadikan SHA-3 lebih tahan terhadap kelas serangan tertentu yang relevan dengan desain lama, sekaligus menawarkan fleksibilitas panjang keluaran.

Fungsi hash kriptografis seperti SHA-3 memiliki beberapa sifat yang penting untuk keamanan:

1) One-wayness: secara komputasional sulit untuk merekonstruksi input dari nilai hash.
2) Collision resistance: sulit menemukan dua input berbeda yang menghasilkan hash sama.
3) Second preimage resistance: sulit menemukan input baru yang memiliki hash sama dengan input tertentu.

Pada skema yang diusulkan, SHA-3 digunakan untuk menghitung digest dari payload transaksi sebelum dilakukan proses penandatanganan digital menggunakan ECDSA. Dengan demikian, tanda tangan dibuat bukan atas seluruh pesan mentah, tetapi atas nilai hash-nya. Pendekatan ini sejalan dengan praktik umum pada tanda tangan digital modern karena lebih efisien dan memanfaatkan sifat keamanan dari fungsi hash.

## C. *Elliptic Curve Digital Signature Algorithm (ECDSA)*

ECDSA adalah varian algoritma tanda tangan digital yang didefinisikan di atas kurva eliptik. Secara konseptual, ECDSA memiliki peran yang sama dengan DSA atau RSA signature, namun memanfaatkan struktur matematis ECC sehingga ukuran tanda tangan dan kunci publik menjadi lebih ringkas.

Secara garis besar, proses penandatanganan menggunakan ECDSA melibatkan:

1) Menghitung nilai hash $h$ = H (m) dari pesan m.
2) Menghasilkan nilai acak sementara (nonce) k pada kurva eliptik.
3) Menggunakan kunci privat pengguna dan nilai k untuk menghasilkan pasangan tanda tangan (r,s).

Verifikasi tanda tangan dilakukan oleh penerima menggunakan kunci publik pengirim. Jika tanda tangan valid, penerima mendapat jaminan bahwa pesan benar-benar berasal dari pemilik kunci privat (autentikasi), pesan tidak diubah (integritas), dan pengirim tidak dapat menyangkal keterlibatannya (non-repudiation). Pada konteks e-wallet, ECDSA digunakan untuk menandatangani payload transaksi sehingga setiap perubahan terhadap data seperti nominal, tujuan, atau timestamp akan menyebabkan verifikasi gagal.

## D. *Model Ancaman pada Ekosistem E-Wallet*

Ekosistem e-wallet melibatkan berbagai komponen seperti aplikasi mobile client, backend server, payment gateway, dan pihak ketiga seperti bank atau penyedia layanan top-up. Kompleksitas ini membuka permukaan serangan yang luas. Beberapa vektor ancaman yang relevan antara lain:

- Payload tampering: penyerang yang berhasil mengintersepsi atau mengakses jalur komunikasi dapat mencoba mengubah isi transaksi, misalnya mengubah nominal atau ID tujuan.
- MITM interception: pihak ketiga menyusup di antara pengguna dan server untuk menyadap atau memodifikasi pesan tanpa sepengetahuan kedua pihak.
- Replay attack: penyerang menyimpan pesan transaksi yang valid kemudian mengirimkannya kembali pada waktu berbeda untuk mencoba menduplikasi transaksi.

- Kompromi database server: jika server atau basis data tersusupi, penyerang dapat mengubah status atau data transaksi secara langsung.

Tanpa adanya mekanisme kriptografi yang secara eksplisit mengikat payload transaksi dengan identitas pengirim dan waktu pengiriman, serangan di atas berpotensi tidak terdeteksi. Oleh karena itu, diperlukan skema end-to-end yang menggabungkan enkripsi, hashing, dan tanda tangan digital untuk mengurangi risiko tersebut.

## III. Analisis Permasalahan dan Desain

### A. *Permasalahan Utama*

Pada sistem e-wallet yang hanya bergantung pada TLS, payload transaksi yang tiba di backend biasanya diproses sebagai data biasa. Selama komunikasi terenkripsi, sistem mengasumsikan bahwa data tidak dimodifikasi di jalur. Namun, asumsi ini tidak selalu aman apabila:

- terjadi kompromi pada application server,
- ada reverse proxy atau layanan pihak ketiga yang disisipi malware,
- atau terjadi kesalahan konfigurasi keamanan di dalam jaringan internal.

Payload transaksi e-wallet umumnya mengandung informasi penting seperti:

json:
```
{
 "sender": "user123",
 "receiver": "merchantXYZ",
 "amount": 50000,
 "timestamp": 1734203100
}
```

Tanpa adanya mekanisme kriptografi tambahan, perubahan kecil pada field amount atau receiver sulit dideteksi, apalagi jika manipulasi terjadi setelah data meninggalkan perangkat pengguna. Selain itu, ketiadaan timestamp atau nonce yang diverifikasi secara kriptografis membuka peluang terjadinya replay attack.

### B. *Design Skema End-to-End Cryptography*

Untuk menjawab permasalahan di atas, dirancang sebuah skema end-to-end yang mengikat payload transaksi dengan identitas pengirim dan waktu pengiriman. Skema ini menggunakan ECC untuk pertukaran kunci rahasia, SHA-3 untuk fungsi hash, dan ECDSA untuk tanda tangan digital.

1) Tahap pada Client

Di sisi client (aplikasi e-wallet pengguna), langkah-langkah berikut dilakukan sebelum transaksi dikirim:

1. Payload transaksi disusun dalam format terstruktur (misalnya JSON) dan diserialisasi menjadi byte array.
2. Fungsi hash SHA3-256 diterapkan pada byte array untuk menghasilkan nilai hash h.
3. Pengguna menggunakan kunci privat ECDSA miliknya untuk menandatangani h, menghasilkan tanda tangan digital σ.
4. Payload asli beserta tanda tangan σ kemudian dienkripsi menggunakan skema berbasis ECC. Dalam implementasi, dilakukan pertukaran kunci rahasia sementara menggunakan ECDH (Elliptic Curve Diffie–Hellman), yang selanjutnya diturunkan (derived) menjadi kunci simetris untuk enkripsi menggunakan AES-GCM.
5. Paket akhir yang dikirim ke server berisi ciphertext, kunci publik pengirim (untuk keperluan verifikasi ECDSA), kunci publik sementara (ephemeral public key) untuk proses ECDH, serta timestamp transaksi.


**Gambar 3.1.** Implementasi Enkripsi untuk Server Side

2) Tahap pada Server

Di sisi server, langkah berikut dilakukan ketika paket diterima:

1. Server menggunakan kunci privat ECC miliknya dan ephemeral public key dari client untuk menghitung kunci rahasia bersama melalui ECDH.
2. Kunci rahasia ini kemudian diturunkan menjadi kunci simetris dan digunakan untuk mendekripsi ciphertext menggunakan AES-GCM. Apabila terjadi perubahan pada ciphertext, proses dekripsi akan gagal karena authentication tag tidak valid.
3. Dari hasil dekripsi, server memperoleh kembali payload transaksi dan tanda tangan σ. Server menghitung ulang nilai hash SHA3-256 dari payload.
4. Menggunakan kunci publik ECDSA milik pengguna, server memverifikasi σ terhadap hash

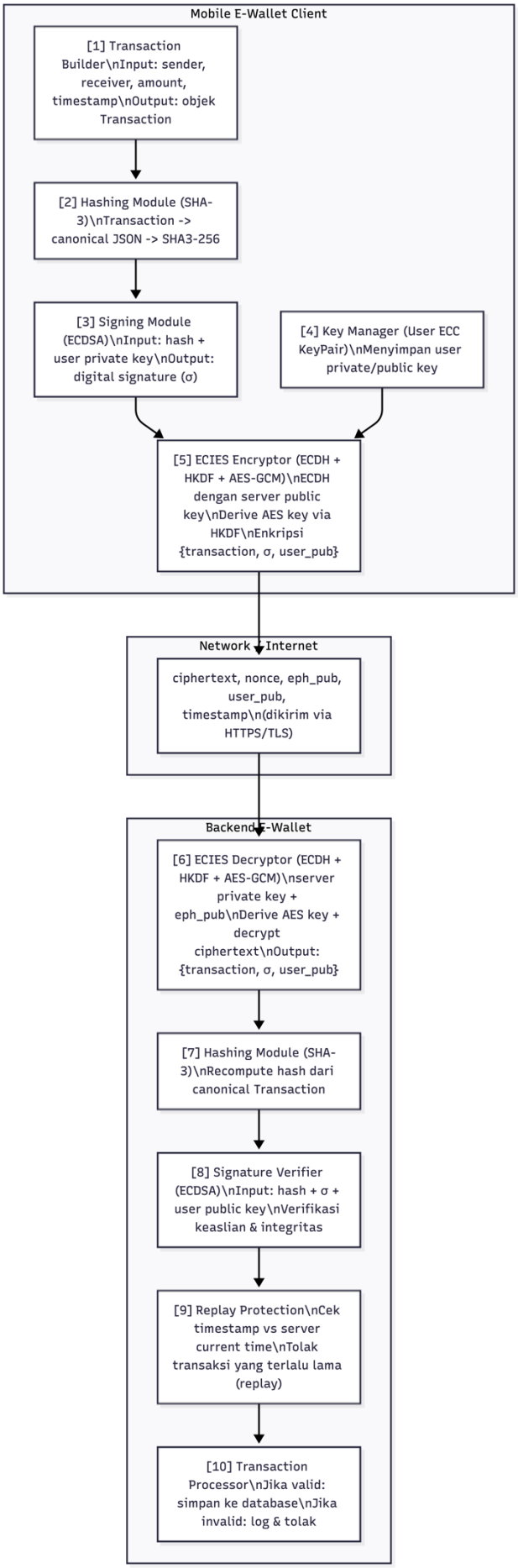
yang baru dihitung. Jika verifikasi gagal, transaksi ditolak.

5. Server memeriksa timestamp untuk memastikan bahwa transaksi berada dalam jendela waktu yang diizinkan (misalnya kurang dari beberapa menit). Transaksi dengan waktu terlalu lama dideteksi sebagai kandidat replay attack dan ditolak.


**Gambar 3.2.** Implementasi Proses Paket pada Server

## C. *Arsitektur Sistem*

Secara arsitektural, skema ini dapat direpresentasikan sebagai rangkaian komponen yang menghubungkan mobile client dan backend server. Di sisi client, modul utama mencakup pengelola kunci (key manager), modul hashing SHA-3, modul ECDSA untuk penandatanganan, dan modul enkripsi berbasis ECC + AES-GCM. Di sisi server, modul yang bersesuaian mencakup key manager server, modul dekripsi, modul hashing SHA-3, modul verifikasi ECDSA, serta transaction processor yang berinteraksi dengan basis data.



**Gambar 3.3** Arsitektur End-to-End Cryptographic Protocol

Diagram tersebut dapat menampilkan alur:
***Client App → SHA-3 → ECDSA Sign → ECC+AES-GCM Encrypt → Network → ECC+AES-GCM Decrypt → SHA-3 → ECDSA Verify → Transaction Processor.***

## IV. IMPLEMENTASI

Implementasi skema dilakukan menggunakan bahasa pemrograman Python 3 dengan memanfaatkan pustaka cryptography untuk operasi ECC, ECDSA, SHA-3, dan AES-GCM. Lingkungan pengujian menggunakan sistem operasi MacOS dengan interpreter Python.

Untuk memudahkan pemeliharaan kode, implementasi dibagi menjadi beberapa fungsi utama:

- fungsi pembangkitan kunci pengguna dan server,
- fungsi serialisasi transaksi ke bentuk byte,
- fungsi hashing menggunakan SHA3-256,
- fungsi penandatanganan dan verifikasi ECDSA,
- fungsi enkripsi dan dekripsi berbasis ECDH + AES-GCM,
- serta fungsi pengujian skenario serangan (modifikasi payload, replay attack, dan kesalahan kunci).

### *A. Pembangkitan Kunci*

Kunci privat dan publik ECC untuk pengguna dan server dihasilkan menggunakan kurva SECP256R1. Kunci privat disimpan secara lokal (dalam konteks simulasi, sebagai objek di memori), sementara kunci publik digunakan untuk verifikasi tanda tangan dan pertukaran kunci rahasia melalui ECDH.

makalah_1 - main.py

```python
def generate_ec_keypair() -> KeyPair:
    private_key = ec.generate_private_key(ec.SECP256R1())
    public_key = private_key.public_key()
    return KeyPair(private_key=private_key, public_key=public_key)

def derive_aes_key(shared_secret: bytes) -> bytes:
    """
    Derive 256-bit AES key from ECDH shared secret using HKDF-SHA256.
    """
    hkdf = HKDF(
        algorithm=hashes.SHA256(),
        length=32,
        salt=None,
        info=b"e-wallet-end-to-end-crypto-demo",
    )
    return hkdf.derive(shared_secret)

def sha3_256(data: bytes) -> bytes:
    """
    Compute SHA3-256 digest of input data.
    """
    digest = hashes.Hash(hashes.SHA3_256())
    digest.update(data)
    return digest.finalize()
```

**Gambar 4.1.** Implementasi Pembangkitan Kunci

- generate_ec_keypair() membuat pasangan kunci ECC SECP256R1 untuk user maupun server.
- sha3_256 mengimplementasikan fungsi hash SHA3-256 untuk seluruh payload transaksi.
- derive_aes_key menggunakan HKDF-SHA256 untuk menurunkan shared secret hasil ECDH menjadi kunci simetris AES-256, sebagaimana dijelaskan pada bagian ECIES-like.

### *B. Hashing dengan SHA-3*

Fungsi hash diimplementasikan menggunakan *hashes.SHA3_256()* dari pustaka cryptography. Seluruh payload transaksi yang telah diserialisasi menjadi byte diproses untuk menghasilkan digest sepanjang 256 bit.

### *C. Digital Signature menggunakan ECDSA*

Proses penandatanganan dilakukan dengan memanggil private_key.sign(..., ec.ECDSA(hashes.SHA3_256())), sementara verifikasi dilakukan menggunakan metode public_key.verify(...). Dengan demikian, secara internal algoritma tanda tangan juga menggunakan SHA-3, selaras dengan fungsi hash eksplisit yang digunakan.

makalah_1 - main.py

```python
class EndToEndCrypto:
    def __init__(self, server_keys: KeyPair, replay_window_seconds: int = 60) -> None:
        self._server_keys = server_keys
        self._replay_window_seconds = replay_window_seconds

    # ---------- Client side operations ----------

    @staticmethod
    def sign_transaction(
        transaction: Transaction,
        user_keys: KeyPair,
    ) -> Tuple[bytes, bytes]:
        """
        Buat hash SHA3-256 dari transaction, lalu tanda tangani dengan ECDSA.
        Return (digest, signature).
        """
        tx_bytes = transaction.to_canonical_bytes()
        digest = sha3_256(tx_bytes)

        signature = user_keys.private_key.sign(
            digest,
            ec.ECDSA(hashes.SHA3_256()),
        )
        return digest, signature
```

**Gambar 4.2.** Implementasi Penandatanganan Paket

- Transaksi diserialisasi ke bentuk kanonik, lalu di-hash dengan SHA3-256.
- Nilai digest ditandatangani dengan user_keys.private_key menggunakan ECDSA-SHA3-256.
- Inilah implementasi langkah 2–3 pada urutan client di bagian desain (hash → sign).

### *D. Enkripsi Payload Menggunakan ECC (ECIES-like)*

Untuk membangun enkripsi end-to-end, digunakan pendekatan mirip ECIES. Client membangkitkan kunci privat ECC sementara (ephemeral), kemudian melakukan ephemeral_key.exchange(ec.ECDH(), server_public_key) untuk memperoleh shared secret. Nilai ini lalu masuk ke fungsi HKDF untuk diturunkan menjadi kunci simetris AES-GCM 256 bit. Payload (berisi transaksi dan tanda tangan) dienkripsi menggunakan AES-GCM dengan nonce acak sepanjang 96 bit.


**Gambar 4.3.** Implementasi Enkripsi untuk Server Side

- Transaction merepresentasikan payload logis (sender, receiver, amount, timestamp) yang akan di-hash dan ditandatangani.
- to_canonical_bytes() memastikan serialisasi JSON kanonik (kunci terurut, separator rapat) sehingga hashing dan tanda tangan deterministik.
- NetworkPacket adalah format paket jaringan yang dikirim: seluruh nilai biner (kunci publik sementara, nonce, ciphertext) di-encode base64 agar mudah ditransmisikan dan di-log.

### *E. Format Paket Transaksi*

Format paket transaksi yang dikirimkan melalui jaringan dirancang dalam bentuk struktur JSON yang berisi:

- ciphertext: hasil enkripsi AES-GCM dalam bentuk byte yang di-encode (misal base64/hex),
- nonce: nonce AES-GCM,
- tag: tag autentikasi AES-GCM (bila tidak digabung di ciphertext),
- ephemeral_pub: representasi kunci publik sementara dalam format PEM,
- user_pub: kunci publik ECDSA pengguna (untuk verifikasi),
- timestamp: waktu transaksi dikirim.

Struktur ini memudahkan server melakukan dekripsi dan verifikasi tanpa memerlukan informasi tambahan di luar paket.

## V. PENGUJIAN

Pengujian dilakukan untuk mengevaluasi:

1. Kemampuan deteksi modifikasi payload
   Menguji apakah perubahan terhadap isi transaksi dapat terdeteksi melalui kombinasi AES-GCM dan ECDSA.
2. Ketahanan terhadap replay attack
   Menguji apakah paket transaksi lama yang dikirim kembali akan ditolak berdasarkan pemeriksaan timestamp.
3. Validasi terhadap kesalahan kunci.
   Menguji apakah penggunaan kunci publik yang salah akan menyebabkan proses verifikasi tanda tangan gagal.

### *A. Skenario 1: Transaksi Normal*

Pada skenario ini, dilakukan percobaan dimana transaksi pada normalnya dilakukan tanpa ada gangguan ataupun.

Berikut implementasi kode yang dilakukan:


**Gambar 5.1.** Implementasi Skenario Normal

Berikut Output yang dihasilkan:

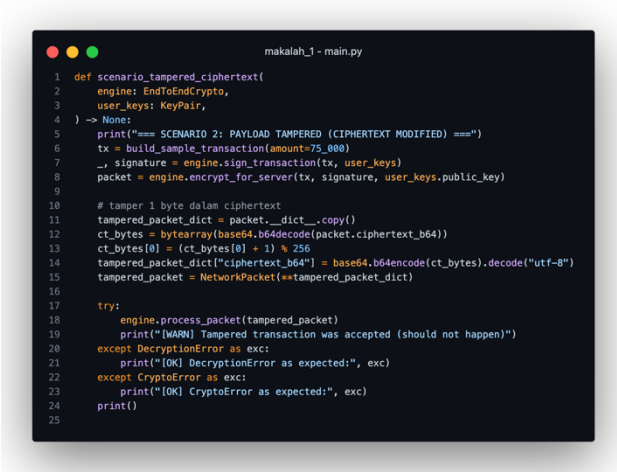

**Gambar 5.2.** Output Pengujian Skenario 1

### *B. Skenario 2: Konten Dimodifikasi*

Pada skenario ini, transaksi asli terlebih dahulu dikirim secara normal dan diverifikasi berhasil oleh server. Selanjutnya, penyerang disimulasikan dengan memodifikasi ciphertext atau, dalam bentuk yang lebih sederhana, mencoba mengubah payload setelah dekripsi namun sebelum verifikasi tanda tangan. Hasil yang diharapkan adalah:

- Jika ciphertext diubah → dekripsi AES-GCM gagal karena tag autentikasi tidak cocok.
- Jika payload diubah tetapi tag AES-GCM valid (misalnya serangan di layer aplikasi) → verifikasi ECDSA gagal karena nilai hash tidak lagi sesuai.

Berikut implementasi kode yang dilakukan:




**Gambar 5.3.** Implementasi Skenario Modifikasi Konten

Pada skenario ini tamper yang dilakukan hanya 1 byte pada ciphertext yang dihasilkan.

Berikut Output yang dihasilkan:

```
=== SCENARIO 2: PAYLOAD TAMPERED (CIPHERTEXT MODIFIED) ===
[OK] DecryptionError as expected: Failed to decrypt payload
```

**Gambar 5.4.** Output Pengujian Skenario 2

### C. *Skenario 3: Replay Attack*

Untuk menyimulasikan replay attack, paket transaksi yang valid disimpan, kemudian dikirim ulang setelah melewati batas waktu tertentu (misalnya lebih dari 60 detik). Server memeriksa selisih antara timestamp di dalam paket dan waktu saat ini. Jika selisih melebihi ambang batas, transaksi ditolak meskipun tanda tangan dan ciphertext masih valid.

Berikut implementasi kode yang dilakukan:


**Gambar 5.5.** Implementasi Skenario Replay Attack

Pada skenario ini tamper yang dilakukan pada age offset dimana dibuat menjadi 1 jam sebelumnya pada sample transaksi yang akan digunakan.

Berikut Output yang dihasilkan:

```
=== SCENARIO 3: REPLAY ATTACK (OLD TIMESTAMP) ===
[OK] ReplayAttackError as expected: Replay attack detected: timestamp too old
```

**Gambar 5.6.** Output Pengujian Skenario 3

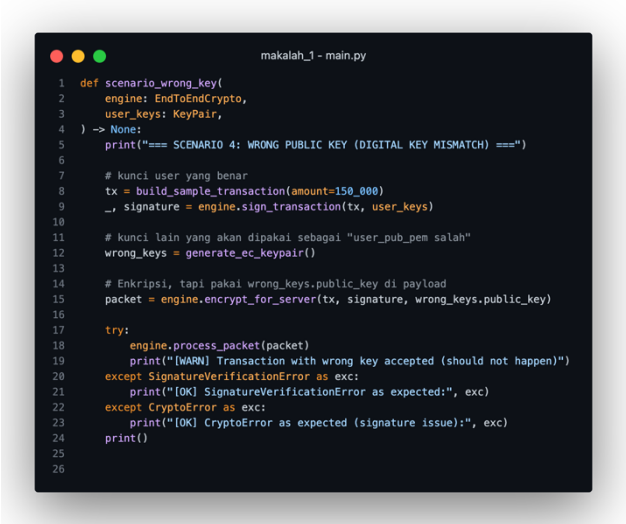
### D. *Skenario 4: Kunci Digital Salah*

Pada skenario ini, server mencoba memverifikasi tanda tangan transaksi menggunakan kunci publik pengguna yang salah atau telah diganti. Harapannya, proses verifikasi akan gagal dan transaksi tidak akan diproses:

Output:

| vbnet |
|---|
| [Wrong key] VerificationError: invalid public key or signature |

Berikut implementasi kode yang dilakukan:




**Gambar 5.7.** Implementasi Skenario Kunci Digital Salah

Pada skenario ini dibuatkan dua kunci (satu kunci user yang benar dan satu lagu kunci yang baru dibuat), kemudian kunci yang baru digunakan untuk menggantikan kunci user saat melakukan enkripsi paket.

Berikut Output yang dihasilkan:

```
=== SCENARIO 4: WRONG PUBLIC KEY (DIGITAL KEY MISMATCH) ===
[OK] SignatureVerificationError as expected: Invalid ECDSA signature
```

**Gambar 5.8.** Output Pengujian Skenario 4

## VI. KESIMPULAN

Makalah ini mengusulkan dan mengimplementasikan skema kriptografi end-to-end untuk transaksi e-wallet dengan mengombinasikan ECC, SHA-3, dan ECDSA. Berdasarkan hasil implementasi dan pengujian, dapat disimpulkan bahwa:

1. Skema yang diusulkan mampu menjamin kerahasiaan, integritas, autentikasi, dan non-repudiation untuk payload transaksi melalui integrasi enkripsi berbasis ECC + AES-GCM, hashing SHA-3, dan tanda tangan digital ECDSA.
2. Mekanisme tanda tangan digital berhasil mendeteksi modifikasi pesan (payload tampering), sementara kombinasi timestamp dan pemeriksaan jendela waktu dapat digunakan untuk mengurangi risiko replay attack.
3. Overhead komputasi dari operasi hashing, penandatanganan, verifikasi, enkripsi, dan

dekripsi terbukti masih dapat ditoleransi untuk skenario transaksi real-time pada sistem prototipe, sehingga berpotensi diterapkan sebagai lapisan keamanan tambahan di atas TLS pada arsitektur e-wallet nyata.

Ke depan, skema ini dapat dikembangkan lebih lanjut dengan melakukan pengujian pada dataset transaksi yang lebih besar, integrasi dengan sistem autentikasi multi-faktor, serta eksplorasi kurva eliptik dan parameter kriptografi lain yang lebih future-proof terhadap ancaman komputasi kuantum.

## VII. Ucapan Terima Kasih

Terima kasih juga penulis sampaikan kepada Bapak Dr. Ir. Rinaldi, M.T. sebagai dosen pengampu dalam mata kuliah IF4020 Kriptografi atas ilmu yang telah diberikan kepada penulis sehingga penulis dapat menyelesaikan makalah ini. Tidak lupa terima kasih juga penulis sampaikan kepada diri sendiri yang senantiasa memberikan dukungan dan motivasi kepada penulis.

## References

[1] W. Stallings, Cryptography and Network Security, Pearson, 2020.
[2] NIST, “SHA-3 Standard,” 2015.
[3] D. Johnson, A. Menezes, and S. Vanstone, “Elliptic Curve Cryptography,” Journal of Cryptology, 2001.
[4] Menezes, van Oorschot, Vanstone, Handbook of Applied Cryptography.

## 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, 24 Desember 2025

Mohammad Nugraha Eka Prawira
13522001