Analisis Perbandingan Algoritma RSA dengan Algoritma Elliptic Curve Cryptography pada Tanda Tangan Digital

Fikri Muhammad Fahreza / 18220012
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: aafahreza76@gmail.com

***Abstract*—Tanda tangan digital merupakan suatu bentuk tanda tangan elektronik yang digunakan untuk mengamankan dokumen ataupun suatu pesan elektronik dan memverifikasi keaslian pengirim ataupun integritas pesan yang dikirim. Proses pembubuhan tanda tangan digital melibatkan suatu *hash function* tertentu dan suatu algoritma kriptografi kunci publik seperti algoritma RSA dan algoritma ECC. Pada makalah ini akan dibahas mengenai perbandingan penggunaan dari kedua algoritma tersebut.**

Keywords—Tanda tangan digital; RSA; ECC; Hash function

I. PENDAHULUAN

Dalam era digital yang semakin maju, tanda tangan digital memainkan peran penting dalam memastikan keaslian, integritas, dan otentikasi dokumen-dokumen yang ditandatangani secara elektronik. Tanda tangan digital memungkinkan pengguna untuk mengidentifikasi diri mereka sendiri dan memvalidasi dokumen-dokumen tersebut dengan cara yang sama seperti tanda tangan tradisional. Seiring dengan kebutuhan akan keamanan yang semakin meningkat, algoritma kriptografi menjadi unsur vital dalam implementasi tanda tangan digital yang aman.

Dalam konteks ini, dua algoritma kriptografi yang sering digunakan untuk tanda tangan digital adalah algoritma RSA (Rivest-Shamir-Adleman) dan algoritma Elliptic Curve Cryptography (ECC). Algoritma RSA telah menjadi salah satu standar industri yang mapan dan banyak digunakan dalam aplikasi keamanan digital. Namun, seiring dengan berkembangnya teknologi dan peningkatan kebutuhan akan efisiensi dan keamanan yang lebih tinggi, algoritma ECC telah menjadi pilihan yang menarik dalam implementasi tanda tangan digital.

Pada makalah ini akan dibahas lebih lanjut mengenai algoritma kriptografi RSA dan ECC tersebut dalam melakukan tanda tangan digital beserta dengan perbandingannya dengan mengimplementasikannya secara langsung pada dokumen perjanjian antara dua pihak.

II. DASAR TEORI

A. Algoritma RSA

Algoritma RSA merupakan salah satu jenis algoritma kriptografi kunci publik yang paling banyak digunakan. Algoritma ini pertama kali dikembangkan pada tahun 1976 oleh tiga peneliti dari MIT, yaitu Ronald Rivest, Adi Shamir, dan Len Adleman, dan dinamai sesuai dengan inisial nama mereka. Keamanan algoritma RSA didasarkan pada kesulitan faktorisasi bilangan bulat besar menjadi faktor-faktor prima. Dalam penggunaannya untuk memberikan keamanan pada pesan, algoritma RSA memerlukan pembangkitan kunci publik dan kunci privat yang akan digunakan untuk mengenkripsi dan mendekripsi pesan secara aman.

a. Langkah Pembangkitan Kunci

1. Pilih dua bilangan prima, p dan q (sebaiknya $p \neq q$)
2. Hitung nilai n sebagai sebagai hasil perkalian p dan q

$$n = p \times q$$

3. Hitung nilai $\phi(n)$

$$\phi(n) = (p-q)(q-1)$$

4. Pilih sebuah bilangan bulat *e* sebagai kunci publik. nilai *e* harus relatif prima terhadap $\phi(n)$
5. Hitung kunci privat d

$$d \equiv e^{-1} \, mod \, (\phi(n))$$

Hasil dari langkah ini adalah pasangan kunci publik (e, n) dan kunci privat (d, n).

b. Langkah Enkripsi / *Signing*

1. Bagi pesan menjadi blok-blok plainteks yang lebih kecil: m_1, m_2, … (syarat $0 \leq m_i < n$)

2. Hitung cipherteks c_i untuk blok plainteks m_i menggunakan kunci public e

$$c_i = {m_i}^e \ mod \ n$$

c. Langkah Dekripsi / *Verification*

1. Untuk setiap blok cipherteks c_i, hitung nilai m_i menggunakan kunci privat d

$$m_i = {c_i}^d \ mod \ n$$

B. Algoritma ECC

Algoritma *Elliptic Curve Cryptography* (ECC) merupakan jenis algoritma kriptografi kunci publik yang juga sering digunakan. Algoritma ini didasarkan pada operasi matematika yang melibatkan titik-titik pada kurva eliptik. Keamanan ECC didasarkan *Elliptic Curve Discrete Logarithm Problem*.

a. Langkah Pembangkitan Kunci

1. Pilih sebuah kurva eliptik yang sesuai, misalnya kurva eliptik dengan persamaan

$$y^2 = x^3 + ax + b \ mod \ p$$

Dimana p adalah bilangan prima yang lebih besar dari nilai maksimum x dan y.

2. Pilih titik basis (*base point*), $B(x_B, y_B)$ yang berada pada kurva eliptik yang dipilih.
3. Pilih bilangan bulat d_A yang berada pada selang $[1, p-1]$
4. Hitung hasil kali antara bilangan bulat k dengan titik basis B

$$Q_A = d_A \times B$$

Hasil dari langkah ini adalah pasangan kunci publik titik Q_A dan kunci privat d_A.

b. Langkah Enkripsi / *Signing*

1. Hitung *hash* pesan *M* menggunakan fungsi *hash*

$$h = hash(M)$$

2. Pilih nilai k yang berada pada selang $[1, \ n-1]$. n adalah urutan bilangan bulat dari B, yang berarti $n \times G = O$, dengan O adalah elemen identitas
3. Hitung titik kurva

$$k \times B = (x_1, y_1)$$

4. Hitung nilai r

$$r = x_1 \ mod \ n$$

Jika $r = 0$, kembali ke langkah 2

5. Hitung nilai s

$$s = k^{-1}(h + r \cdot d_A) \ mod \ n$$

Jika $s = 0$, kembali ke langkah 2

Tanda tangan yang dihasilkan adalah pasangan nilai (r, s).

c. Langkah Dekripsi / *Verification*

1. Hitung hash pesan *M* menggunakan fungsi *hash* yang sama pada saat proses *signing*

$$h = hash(M)$$

2. Hitung *modular inverse s*

$$u_1 = h \cdot s^{-1} \ mod \ n$$

$$u_2 = r \cdot s^{-1} \ mod \ n$$

3. Hitung titik kurva

$$(x_1, y_1) = u_1 \times B + u_2 \times Q_A$$

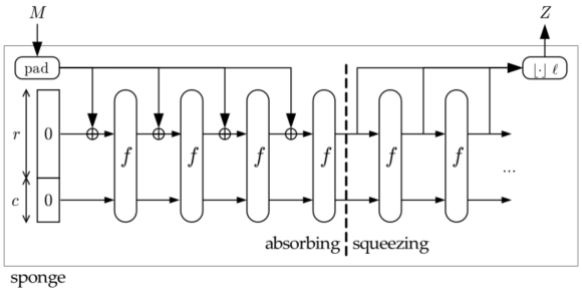
Tanda tangan otentik jika $r \equiv x_1 \ mod \ n$.

C. Fungsi Hash SHA-3 (Keccak)

Fungsi hash SHA-3 (Secure Hash Algorithm 3), juga dikenal sebagai Keccak, adalah fungsi hash kriptografik yang digunakan untuk menghasilkan nilai hash yang unik dari suatu data atau pesan. Fungsi hash SHA-3 didasarkan pada permutasi Keccak, yang merupakan hasil dari kompetisi fungsi hash yang diadakan oleh Institut Teknologi NIST (National Institute of Standards and Technology) pada tahun 2007.

Keccak memiliki perbedaan dalam metode konstruksi dibandingkan dengan finalis SHA-3 lainnya, yaitu menggunakan konstruksi "spons" (*sponge construction*). Sementara desain SHA-3 lainnya bergantung pada "fungsi kompresi", Keccak menggunakan fungsi yang tidak memampatkan (non-kompresi) untuk proses "penyerapan" (*absorbing*) dan kemudian "pemerasan" (*squeezing*) nilai *hash*. Berikut merupakan langkah fungsi *hash* Keccak dalam menghasilkan *message digest*.

a. Pra-proses

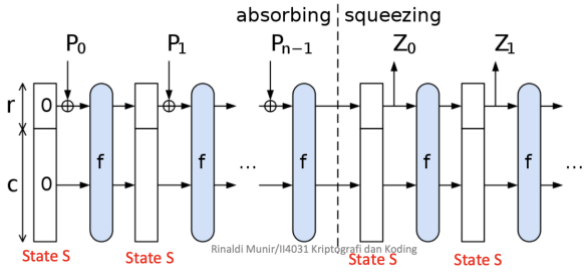


Gambar 1 Konstruksi Spons (Sumber: Slide Kuliah II4031 Kriptografi dan Koding - Rinaldi Munir)

Pada tahap ini, pesan *M* diubah menjadi sebuah *string* P dengan menambahkan bit-bit pengganjal (*padding*), sehingga panjang *string* P dapat dibagi habis oleh nilai r atau n = panjang(P)/r. Setelah itu, *string* P dibagi

menjadi blok-blok P_i dengan panjang r-bit. Setelah itu, sejumlah b-bit dari peubah status (*state*) *S* diinsialisasi dengan nol.

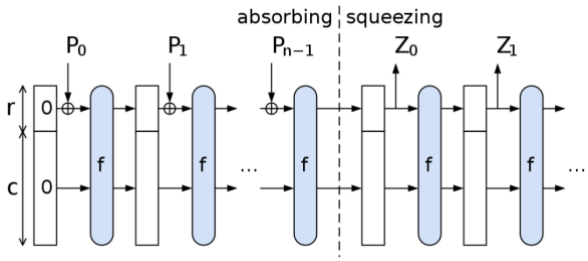
b. Fase Penyerapan (*Absorbing*)



Gambar 2 Fase Penyerapan (Sumber: Slide Kuliah II4031 Kriptografi dan Koding - Rinaldi Munir)

Pada fase ini, untuk setiap blok masukan P_i dengan ukuran r-bit, dilakukan operasi XOR dengan r-bit pertama dari *state S*. Hasilnya kemudian dimasukkan ke dalam fungsi permutasi *f*, yang menghasilkan *state S* baru. Setelah semua blok masukan diproses, konstruksi spons beralih ke fase berikutnya yakni fase pemerasan (*squeezing*).

c. Fase Pemerasan (*Squeezing*)



Gambar 3 Fase Pemerasan (Sumber: Slide Kuliah II4031 Kriptografi dan Koding - Rinaldi Munir)

Pada fase ini, nilai *hash* pesan akan disimpan dalam variabel *Z*. Langkah awal adalah menginisialisasi *Z* dengan sebuah *string* kosong. Selanjutnya, selama panjang *Z* masih belum mencapai nilai yang ditentukan, r-bit pertama dari *state S* akan digabungkan (*append*) ke *Z*. Jika panjang *Z* masih belum mencapai nilai yang ditentukan, blok masukan akan dimasukkan ke dalam fungsi permutasi *f* untuk menghasilkan *state S* baru.

D. Tanda Tangan Digital

Tanda tangan digital pada makalah ini dilakukan dengan menggunakan kombinasi kriptografi kunci-publik dan fungsi hash merupakan salah satu metode yang umum digunakan untuk memastikan keaslian dan integritas pesan digital.

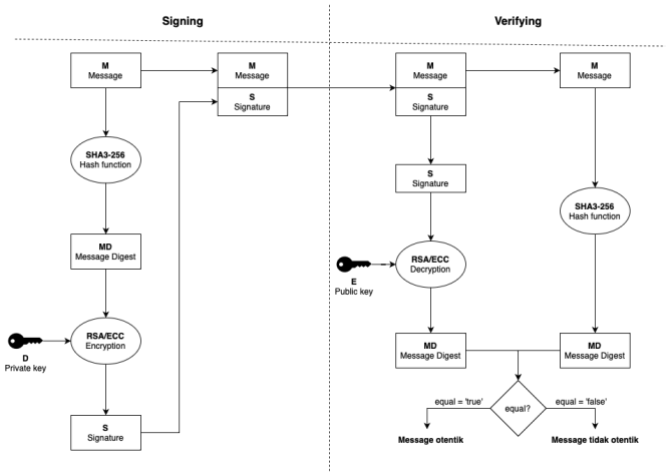
Kriptografi kunci-publik digunakan untuk menghasilkan pasangan kunci, yaitu kunci publik dan kunci privat. Kunci publik dapat dibagikan kepada semua pihak yang ingin memverifikasi tanda tangan, sementara kunci privat harus dijaga dengan baik oleh pemiliknya. Kunci publik digunakan untuk memverifikasi keaslian tanda tangan, sedangkan kunci privat digunakan untuk menghasilkan tanda tangan yang unik.

Fungsi *hash* digunakan untuk menghasilkan nilai hash dari pesan yang akan ditandatangani. Fungsi *hash* mengubah pesan menjadi serangkaian angka unik dengan panjang tetap. Nilai hash ini merepresentasikan pesan asli dan akan digunakan sebagai "ringkasan" pesan yang akan ditandatangani.

Proses penandatanganan dimulai dengan menghasilkan nilai hash dari pesan menggunakan fungsi *hash* yang dipilih. Selanjutnya, nilai *hash* tersebut dienkripsi menggunakan kunci privat yang dimiliki oleh penandatangan. Hasil enkripsi ini merupakan tanda tangan digital yang akan dikirimkan dengan pesan tersebut.

Untuk memverifikasi tanda tangan, penerima pesan akan memisahkan pesan dan tanda tangan digital. Pesan akan di *hash* kembali dengan jenis fungsi hash yang sama, sedangkan tanda tangan digital akan di dekripsi dengan menggunakan kunci publik pengirim. Penerima kemudian akan membandingkan *output* dari kedua proses tersebut. Jika sama, maka pesan tersebut dinyatakan otentik.

Berikut merupakan diagram proses pembubuhan tanda dan verifikasi tangan digital.



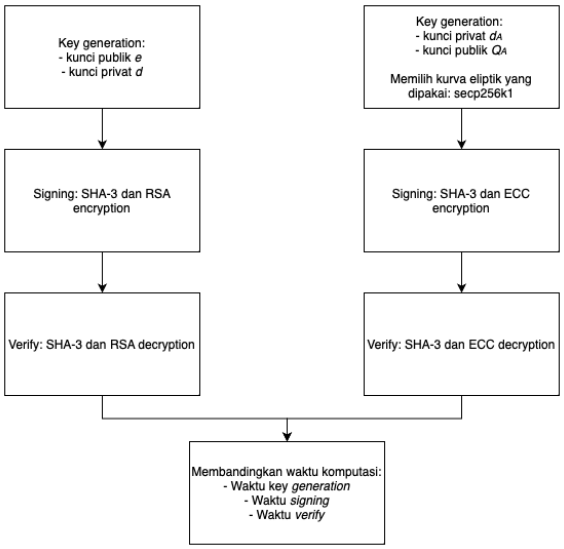
Gambar 4 Proses Tanda Tangan Digital (Sumber: pribadi)

III. RANCANGAN

Secara umum, kedua algoritma tersebut akan melakukan pembangkitan kunci (*key generation*). Pesan yang akan ditandatangani di-*hash* terlebih dahulu dengan menggunakan fungsi *hash* SHA-3 (Keccak), kemudian masuk ke proses enkripsi (*signing)* menggunakan kunci privat pengirim. Terakhir, pesan yang telah ditandatangani akan didekripsi (*verify*) menggunakan kunci publik yang berpadanan.

Parameter yang akan dibandingkan adalah:
- Waktu pembangkitan kunci (*key generation*)
- Waktu pembubuhan tanda tangan (*signing*)
- Waktu verifikasi pesan (*verify*)

Berikut merupakan diagram alur dari program yang akan diimplementasikan.



Gambar 5 Rancangan program (Sumber: pribadi)

IV. Implementasi dan Pengujian

A. Implentasi

Implementasi program tanda tangan digital dilakukan dengan menggunakan bahasa Python. Berikut merupakan kode program tanda tangan digital dengan menggunakan algoritma kriptografi kunci publik RSA dan tanda tangan digital dengan menggunakan algoritma kriptografi kunci publik ECC.

Program RSA Digital Signature

```
   import random
   from typing import Tuple
   import hashlib
   import time

   # Fungsi untuk mengecek apakah a dan b bilangan yang relatif prima
   def is_relative_prime(a: int, b: int) -> bool:
       while b:
           a, b = b, a % b

       return a == 1

   with open('prime_list_1536.txt', 'r') as f:
       primes = list(map(int, f.read().split()))

   # Fungsi untuk memilih nilai p, q, e secara random
   def get_random_key() -> Tuple[int, int, int]:
       p, q, e = random.choices(primes, k=3)
       totient_n = (p - 1)*(q - 1)

       while not is_relative_prime(totient_n, e):
           p, q, e = random.choices(primes, k=3)
       return p, q, e

   # Fungsi untuk menghitung invers modulo
   def find_mod_inverse(e: int, totient_n: int) -> int:
       u1, u2, u3 = 1, 0, e
       v1, v2, v3 = 0, 1, totient_n
       while v3 != 0:
           q = u3 // v3
           v1, v2, v3, u1, u2, u3 = (u1 - q * v1), (u2 - q * v2), (u3 - q * v3), v1, v2, v3

       return u1 % totient_n

   # Fungsi untuk menghitung invers modulo
   def generate_public_key(p: int, q: int, e: int) -> Tuple[int, int]:
       return e, p * q

   # Fungsi untuk menghitung dan membangkitkan private key
   def generate_private_key(p: int, q: int, e: int) -> Tuple[int, int]:
       totient_n = (p - 1)*(q - 1)
       d = find_mod_inverse(e, totient_n)

       return d, p * q

   # Fungsi untuk menghitung hash pesan menggunakan SHA-256
   def hash_message(message: str) -> str:
       return hashlib.sha3_256(message.encode()).hexdigest()

   # Fungsi untuk mengenkripsi pesan
   def encrypt(message: str, d: int, n: int) -> str:
       return ''.join(hex(pow(ord(m), d, n)) for m in message)

   # Fungsi untuk mendekripsi pesan
   def decrypt(cipher: str, e: int, n: int) -> str:
       cipher = [int(x, 16) for x in cipher.split('0x')[1:]]

       return ''.join(chr(pow(c, e, n)) for c in cipher)

   # Fungsi untuk menghasilkan tanda tangan digital
   def create_signature(text: str, private_key: Tuple[int, int]) -> str:
       d, n = private_key

       return encrypt(hash_message(text), d, n)

   if __name__ == '__main__':
       print("RSA Digital Signature\n")

       p, q, e = get_random_key()

       start_time = time.perf_counter()
```

```
        e, n = generate_public_key(p, q, e)
        d, n = generate_private_key(p, q, e)
        end_time = time.perf_counter()

        key_gen_time = (end_time-start_time)*1000

        print(f'Private key: {(d, n)}')
        print(f'Public key: {(e, n)}')
        print(f'Waktu       key       generation:
{key_gen_time}ms\n')

        message = "II4031 Kriptografi dan Koding"
        print(f'Message: {message}')

        start_time = time.perf_counter()
        signature = create_signature(message, (d,
n))
        end_time = time.perf_counter()

        signing_time = (end_time-start_time)*1000

        # print(f'Signature: {signature}')
        print(f'Waktu                      signing:
{signing_time}ms\n')

        start_time = time.perf_counter()
        decrypted = decrypt(signature, e, n)
        valid = hash_message(message) == decrypted
        end_time = time.perf_counter()

        validation_time        =        (end_time-
start_time)*1000

        print(f'Tanda tangan{"" if valid else "
TIDAK"} VALID')
        print(f'Waktu                     validasi:
{validation_time}ms\n')

        print(f'Total                        waktu:
{key_gen_time+signing_time+validation_time}ms')
```

Program ECC Digital Signature

```
    import hashlib
    import random
    import time

    # Definisikan parameter kurva eliptik
(secp256k1)
    p =
0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f
    a =
0x0000000000000000000000000000000000000000000000000000000000000000
    b =
0x0000000000000000000000000000000000000000000000000000000000000007
    Gx =
0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d
959f2815b16f81798
    Gy =
0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a685541
99c47d08ffb10d4b8
    n =
0xfffffffffffffffffffffffffffffffebaaedce6af48a03
bbfd25e8cd0364141

    # Fungsi untuk menghitung invers modulo
    def mod_inverse(a, m):
        if a < 0 or m <= a:
            a = a % m
        c, d = a, m
        uc, vc, ud, vd = 1, 0, 0, 1
        while c != 0:
            q, c, d = divmod(d, c) + (c,)
            uc, vc, ud, vd = ud - q * uc, vd - q *
vc, uc, vc
        if ud > 0:
            return ud
        else:
            return ud + m

    # Fungsi untuk menghitung titik hasil
perkalian skalar pada kurva eliptik
    def point_multiplication(k, P):
        if k == 0:
            return None
        elif k == 1:
            return P
        else:
            Q = point_multiplication(k // 2, P)
            Q = point_addition(Q, Q)
            if k % 2 == 1:
                Q = point_addition(Q, P)
            return Q

    # Fungsi untuk menghitung hasil penjumlahan
dua titik pada kurva eliptik
    def point_addition(P, Q):
        if P is None:
            return Q
        elif Q is None:
            return P
        else:
            if P[0] == Q[0] and P[1] == Q[1]:
                lam = (3 * P[0] * P[0] + a) *
mod_inverse(2 * P[1], p)
            else:
                lam = (Q[1] - P[1]) *
mod_inverse(Q[0] - P[0], p)
            x = (lam * lam - P[0] - Q[0]) % p
            y = (lam * (P[0] - x) - P[1]) % p
            return (x, y)

    # Fungsi untuk membangkitkan private key
secara random
    def generate_private_key():
        return random.randint(1, n)

    # Fungsi untuk membangkitkan public key
berdasarkan private key
    def generate_public_key(private_key):
        return point_multiplication(private_key,
(Gx, Gy))
```

```
   # Fungsi untuk menghitung hash pesan
menggunakan SHA-256
   def hash_message(message):
       sha3 = hashlib.sha3_256()
       sha3.update(message.encode('utf-8'))
       return int(sha3.hexdigest(), 16)

   # Fungsi untuk menghasilkan tanda tangan
digital
   def sign_message(private_key, message):
       k = random.randint(1, n)
       R = point_multiplication(k, (Gx, Gy))
       r = R[0] % n
       if r == 0:
           return sign_message(private_key,
message)
       else:
           s = (mod_inverse(k, n) *
(hash_message(message) + private_key * r)) % n
           if s == 0:
               return sign_message(private_key,
message)
           else:
               return (r, s)

   # Fungsi untuk memverifikasi tanda tangan
digital
   def verify_signature(public_key, message,
signature):
       r, s = signature
       if r < 1 or r > n - 1 or s < 1 or s > n -
1:
           return False
       else:
           w = mod_inverse(s, n)
           u1 = (hash_message(message) * w) % n
           u2 = (r * w) % n
           X =
point_addition(point_multiplication(u1, (Gx,
Gy)), point_multiplication(u2, public_key))
           if X is None:
               return False
           else:
               return r == X[0] % n

   if __name__ == '__main__':
       print("ECC Digital Signature\n")
       start_time = time.perf_counter()
       private_key = generate_private_key()
       public_key =
generate_public_key(private_key)
       end_time = time.perf_counter()

       key_gen_time = (end_time-start_time)*1000

       print(f'Private key: {private_key}')
       print(f'Public key: {public_key}')
       print(f'Waktu key generation:
{key_gen_time}ms\n')

       message = 'II4031 Kriptografi dan Koding'

       start_time = time.perf_counter()
       signature = sign_message(private_key,
message)
       end_time = time.perf_counter()

       print(f'Message: {message}')

       signing_time = (end_time-start_time)*1000

       print("r =", signature[0])
       print("s =", signature[1])
       print(f'Waktu tanda tangan:
{signing_time}ms\n')

       start_time = time.perf_counter()

       valid = verify_signature(public_key,
message, signature)
       end_time = time.perf_counter()

       validation_time = (end_time-
start_time)*1000

       print(f'Tanda tangan{"" if valid else "
TIDAK"} VALID')
       print(f'Waktu validasi:
{validation_time}ms\n')

    print(f'Total waktu yang diperlukan:
{key_gen_time+signing_time+validation_time}ms')
```

B. Pengujian

Pengujian program dilakukan dengan pembubuhan tanda tangan dan pengecekan validitas tanda tangan dengan pesan "II4031 Kriptografi dan Koding". Panjang kunci yang digunakan pada algoritma RSA adalah 3072-bit dan panjang kunci yang digunakan pada algoritma ECC adalah 256-bit. Berikut merupakan hasil pengujian program.

Program RSA Digital Signature

Gambar 6 Hasil pengujian program RSA Digital Signature
(Sumber: pribadi)

Program ECC Digital Signature

Gambar 7 Hasil pengujian program ECC Digital Signature
(Sumber: pribadi)

Dari hasil pengujian diatas, dapat terlihat bahwa proses pembangkitan kunci pada algoritma RSA lebih cepat dari

algoritma ECC, sedangkan pada proses *signing* dan *verify* algoritma ECC jauh lebih cepat dibandingkan algoritma RSA.

V. KESIMPULAN

Algoritma ECC jauh lebih cepat daripada algoritma RSA dalam waktu pembubuhan tanda tangan dan verifikasi. Hal ini juga dapat dijelaskan oleh sifat matematis yang berbeda antara kedua algoritma ini. RSA menggunakan operasi aritmetika yang rumit dan berbasis faktorisasi bilangan bulat yang besar, sehingga memerlukan waktu yang lebih lama untuk dieksekusi, terutama ketika digunakan dengan kunci yang besar.

Di sisi lain, algoritma ECC menggunakan kurva eliptik sebagai dasar perhitungannya sehingga ukuran kunci yang dibutuhkan untuk mencapai tingkat keamanan yang sama dengan RSA pun jauh lebih kecil. Hal ini mengakibatkan operasi matematika yang lebih sederhana dan waktu yang lebih cepat dalam proses pembubuhan tanda tangan dan verifikasi tanda tangan digital.

LINK REPOSITORY

https://github.com/cinnamonboyz/makalah-kriptaur

REFERENCES

[1] HYPR. *Elliptic Curve Digital Signature Algorithm*. Retrivied 20 May 2023, from https://www.hypr.com/security-encyclopedia/elliptic-curve-digital-signature-algorithm

[2] Munir, Rinaldi. 2023. Slide Kuliah II4031 Kriptografi dan Koding: Algoritma RSA

[3] Munir, Rinaldi. 2023. Slide Kuliah II4031 Kriptografi dan Koding: Elliptic Curve Cryptography (ECC)

[4] Munir, Rinaldi. 2023. Slide Kuliah II4031 Kriptografi dan Koding: Tanda Tangan Digital

[5] Nakov, Svetlin. 2018. *Practical Cryptography for Developers*

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, 22 Mei 2023

Fikri Muhammad Fahreza (18220012)