Analisis Perbandingan Algoritma RSA dengan ElGamal pada Tanda Tangan Digital

Jacelyn Felisha / 18219097
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
18219097@gmail.com

***Abstract*—Perkembangan teknologi menyebabkan digitalisasi terhadap banyak kegiatan. Diantaranya yaitu proses tanda tangan menjadi digital. Tanda tangan digital dapat menggunakan beberapa algoritma kriptografi seperti RSA dan ElGamal dengan tambahan fungsi hash SHA-1. Pada makalah ini, akan dibahas mengenai perbandingan penggunaan dari algoritma tersebut.**

Keywords—tanda tangan digital; RSA; ElGamal; SHA-1;

I. PENDAHULUAN

Seiring dengan berkembangnya teknologi, masyarakat menjadi semakin sering berkomunikasi secara *online* dengan saling bertukar informasi melalui jaringan internet terutama sejak masa pandemi COVID-19. Kegiatan-kegiatan yang biasanya dibatasi oleh waktu dan tempat menjadi dimudahkan dengan teknologi dan perangkat digital yang dapat terhubung dengan internet di hampir seluruh tempat di dunia. Salah satu kegiatan yang terdigitalisasi yaitu tanda tangan pada suatu pesan, surat, dan kontrak perjanjian antara dua atau lebih pihak. Namun, tanda tangan merupakan hal yang penting untuk dijaga karena merupakan salah satu identitas dari seseorang atau lembaga yang perlu dipertanggungjawabkan. Kriptografi menghadirkan keamanan pada dokumen yang perlu ditandatangani melalui tanda tangan digital sedemikian sehingga dokumen tersebut dapat terotentikasi dan dijaga keaslian dari isi pesannya.

Pada makalah ini akan dianalisis lebih lanjut mengenai algoritma-algoritma kriptografi yang dapat digunakan untuk tanda tangan digital yaitu algoritma ElGamal dan algoritma RSA beserta dengan perbandingannya dengan mengimplementasikannya secara langsung pada dokumen perjanjian antara dua pihak.

II. DASAR TEORI

A. Algoritma ElGamal

Algoritma ElGamal merupakan salah satu algoritma kriptografi kunci publik yang didasarkan pada pertukaran kunci Diffie-Hellman. Algoritma ElGamal diciptakan oleh Taher Elgamal pada tahun 1985 beserta dengan skema tanda tangan ElGamal. Keamanannya didasarkan pada sulitnya melakukan komputasi pada logaritma diskrit.

Penggunaan algoritma ElGamal dalam memberikan keamanan pada pesan diawali dengan pembangkitan kunci publik dan kunci privat yang akan digunakan untuk mengenkripsi dan mendekripsi pesan. Berikut ini adalah langkah-langkah untuk pembangkitan kuncinya.

1. Memilih sebuah bilangan prima p secara sembarang.
2. Memilih dua buah bilangan secara acak yaitu g dan x, dengan syarat sebagai berikut.
$$g < p \, dan \, 1 \leq x \leq p-2$$
3. Menghitung nilai y.
$$y = g^x \, mod \, p$$
4. Akan dihasilkan nilai dari kunci privat dan kunci publiknya sebagai berikut.
$$Kunci \, publik = tripel \, (y, g, p)$$
$$Kunci \, privat = pasangan \, (x, p)$$

Selanjutnya dilakukan langkah-langkah untuk mengenkripsi pesan sebagai berikut.

1. Menyusun plainteks menjadi beberapa blok $m1, m2, \ldots$ dengan nilai masing-masing blok berada pada interval $[0, p-1]$.
2. Memilih bilangan acak k dengan syarat sebagai berikut.
$$1 \leq k \leq p-2$$
3. Melakukan enkripsi terhadap setiap blok m dengan rumus berikut ini.
$$a = g^k \, mod \, p$$
$$b = y^k m \, mod \, p$$
Hasilnya merupakan pasangan (a, b) cipherteks untuk blok pesan m.

Kemudian cipherteks yang telah dihasilkan pada tahap enkripsi dapat didekripsi melalui langkah berikut ini.

1. Menghitung nilai berikut ini menggunakan kunci privat x yang telah dihasilkan sebelumnya.
$$(a^x)^{-1} = a^{p-1-x} \, mod \, p$$

2. Menghitung plainteks m.

$$m = \frac{b}{a^x} mod\ p = b\ (a^x)^{-1}\ mod\ p$$

B. Algoritma RSA

Algoritma RSA merupakan algoritma kriptografi kunci publik yang paling sering digunakan. Algoritma ini diciptakan oleh tiga peneliti dari MIT yaitu Ronald Rivest, Adi Shamir, dan Len Adleman, pada tahun 1976. Nama Algoritma RSA sendiri diambil dari nama mereka yaitu Rivest, Shamir, dan Adleman. Keamanan algoritma ini didasarkan pada sulitnya memfaktorkan bilangan bulat yang besar menjadi faktor-faktor prima.

Penggunaan algoritma RSA dalam memberikan keamanan pada pesan juga diawali dengan pembangkitan kunci publik dan kunci privat yang akan digunakan untuk mengenkripsi dan mendekripsi pesan. Berikut ini adalah langkah-langkah untuk pembangkitan kuncinya.

1. Memilih dua bilangan prima yaitu p dan q.
2. Menghitung nilai n sebagai berikut.

$$n = p\ x\ q$$

3. Menghitung nilai $\phi(n)$ sebagai berikut.

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

4. Memilih sebuah bilangan bulat e sebagai kunci publik, dengan syarat nilai e harus relatif prima terhadap $\phi(n)$.
5. Menghitung kunci privat d dari e.

$$d \equiv e - 1\ mod\ (\phi(n))$$

Hasilnya merupakan pasangan kunci publik (e, n) dan kunci privat (d, n).

Selanjutnya dilakukan langkah-langkah untuk mengenkripsi pesan sebagai berikut.

1. Membagi pesan menjadi blok-blok plainteks $m1, m2, ..$ dengan syarat $0 \leq m_i < n - 1$.
2. Menghitung nilai blok cipherteks c_i untuk blok plainteks m_i menggunakan kunci public e sebagai berikut.

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

Persamaan di atas diturunkan dari Teorema Euler.

Kemudian cipherteks yang telah dihasilkan pada tahap enkripsi dapat didekripsi melalui langkah berikut ini.

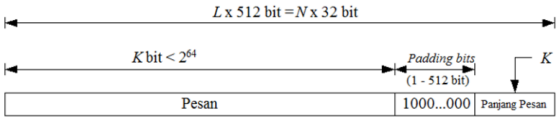
1. Untuk setiap blok cipherteks $c_1, c_2, c_3, ..$, dihitung nilai m_i menggunakan kunci privat d sebagai berikut.

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

C. Fungsi *Hash* SHA-1

SHA (*Secure Hash Algorithm*) merupakan fungsi *hash* satu-arah yang dipublikasikan oleh NIST (*National Institute of Standards and Technology*) dan digunakan bersama DSS (*Digital Signature Standard*). Oleh NSA (*National Security Agency*), SHA dinyatakan sebagai standard fungsi *hash* satu-arah. SHA memiliki beberapa varian algoritma yaitu SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, serta SHA-3 (Keccak). Pada makalah ini, SHA yang digunakan adalah SHA-1. SHA-1 menerima pesan dengan panjang kurang dari 2^{64} bit dan menghasilkan *message digest* dengan panjang 160 bit. Secara umum, langkah-langkah dalam menghasilkan *message digest* melalui 4 tahapan yaitu sebagai berikut.

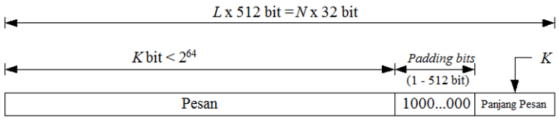
1. Penambahan bit-bit pengganjal (*padding bits*).



Gambar 1. Sumber: Slide Kuliah II4031 Kriptografi dan Koding - Rinaldi Munir

Pada tahap ini, pesan semula yang akan di-*hash* ditambahkan dengan sejumlah bit pengganjal hingga panjang pesan menjadi bernilai 448 bit jika dimodulokan dengan 512. Panjang bit-bit pengganjalnya berkisar di antara 1 hingga 512 bit dan terdiri atas sebuah bit 1 dan diikuti oleh bit-bit 0.

2. Penambahan nilai panjang pesan semula.



Gambar 2. Sumber: Slide Kuliah II4031 Kriptografi dan Koding - Rinaldi Munir

Selanjutnya, pesan tersebut ditambahkan dengan nilai K yaitu panjang pesan semula yang bernilai 64 bit sehingga panjang pesan sekarang menjadi kelipatan 512 bit.

3. Inisialisasi penyangga MD.

Kemudian dilakukan inisialisasi 5 buah penyangga dengan panjang masing-masing 32 bit. Kelima penyangga tersebut diinisialisasi dengan nilai dalam bilangan HEX sebagai berikut.

$$A = 0x67452301$$

$$B = 0xEFCDAB89$$

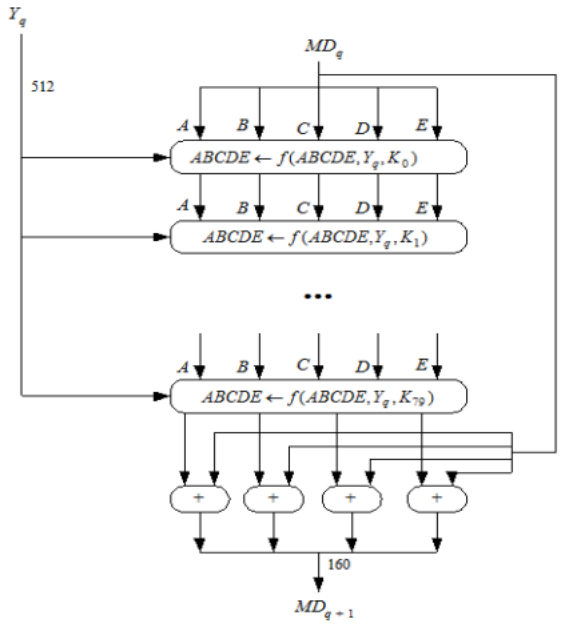
$$C = 0x98BADCFE$$

$$D = 0x10325476$$

$$E = 0xC3D2E1F0$$

4. Pengolahan pesan dalam blok berukuran 512.

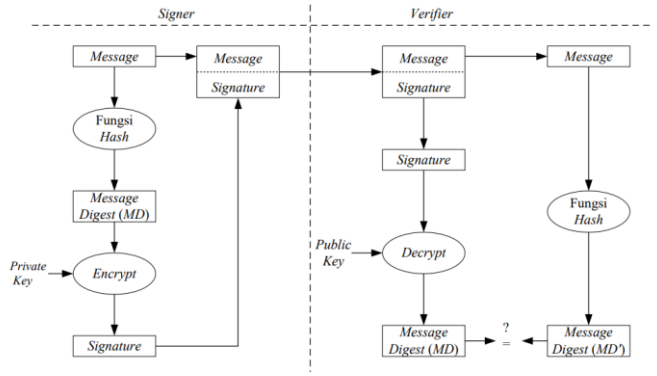
Terakhir, dilakukan pemrosesan pesan yang terdiri atas 80 putaran.



Gambar 3. Sumber: Slide Kuliah II4031 Kriptografi dan Koding - Rinaldi Munir

D. Tanda Tangan Digital

Skema tanda tangan digital pertama kali diideasikan oleh Whitfield Diffie and Martin Hellman pada tahun 1976. Tanda tangan digital berbeda dengan tanda tangan asli yang dipindai menjadi foto. Tanda tangan digital digunakan untuk mengotentikasi suatu dokumen digital. Untuk memastikan keamanannya, tanda tangan digital pada setiap dokumen pasti berbeda dan dihasilkan dari isi pesan dan sebuah kunci. Terdapat 2 cara untuk menandatangani suatu pesan yaitu dengan mengenkripsi pesan atau menggunakan kombinasi algoritma kriptografi kunci publik dengan fungsi *hash*. Pada makalah ini, akan dibahas mengenai tanda tangan digital menggunakan kombinasi algoritma RSA/ElGamal dengan fungsi *hash* SHA-1. Berikut ini adalah gambaran skemanya secara umum.



Gambar 4. Sumber: Slide Kuliah II4031 Kriptografi dan Koding - Rinaldi Munir

Langkah-langkah penandatangan digital menggunakan algoritma RSA dan fungsi hash SHA-1 dibagi menjadi 3 tahapan sebagai berikut.

1. Membangkitkan kunci publik PK dan kunci privat SK milik pengirim pesan menggunakan algoritma RSA yang tertera pada bagian sebelumnya.
2. Pengirim memberikan tanda tangan digital pada pesan dengan langkah sebagai berikut.
    1. Menghitung nilai *hash* dari pesan M.
    $$h = H(M)$$
    2. Mengenkripsi pesan M menggunakan kunci privat SK pengirim untuk menghasilkan *signature* S.
    $$S = h^{SK} \bmod n$$
    3. Menambahkan *signature* S pada pesan M lalu mengirimkannya kepada penerima pesan.
3. Penerima melakukan verifikasi tanda tangan digital pada pesan dengan langkah sebagai berikut.
    1. Memisahkan pesan M dari *signature* S.
    2. Menghitung nilai *hash* dari pesan M.
    $$h = H(M)$$
    3. Mendekripsi *signature* S menggunakan kunci publik PK pengirim.
    $$h' = S^{PK} \bmod n$$
    4. Membandingkan nilai h dengan h'. Jika sama maka tanda tangan berhasil terverifikasi dengan benar.

Langkah-langkah penandatangan digital menggunakan algoritma ElGamal dan fungsi hash SHA-1 juga dibagi menjadi 3 tahapan sebagai berikut.

1. Pengirim membangkitkan kunci privat dan kunci publik dengan langkah sebagai berikut.
    1. Memilih panjang kunci N.
    2. Memilih bilangan prima p dengan panjang N bit secara acak.
    3. Memilih bilangan generator g dengan syarat $g < p$ dan nilai g relatif prima terhadap p.
    4. Dihasilkan parameter (p, g) yang dapat dibagikan kepada orang lain.
    5. Memilih bilang bulat x sebagai kunci privat secara acak dengan syarat $1 \leq x \leq p-2$.
    6. Menghitung nilai y sebagai kunci publik.
    $$y = g^x \bmod p$$
    7. Didapatkan nilai kunci privat x dan kunci publik y.
2. Pengirim memberikan tanda tangan digital pada pesan dengan langkah sebagai berikut.
    1. Memilih bilangan bulat k secara acak dengan syarat $1 < k < p-1$ dan nilai k relatif prima terhadap $p-1$.
    2. Menghitung nilai $r = g^k \bmod p$

3. Menghitung nilai $s = (H(m) - xr)k^{-1} \, mod \, (p-1)$.
4. Jika nilai $s = 0$, maka hitung ulang dengan memilih nilai k lain.
5. Didapatkan sepasang nilai *signature* yaitu (r, s).

3. Penerima melakukan verifikasi tanda tangan digital pada pesan dengan langkah sebagai berikut.
   1. Memastikan bahwa nilai r dan s memenuhi persamaan di bawah ini.

$$0 < r < p$$

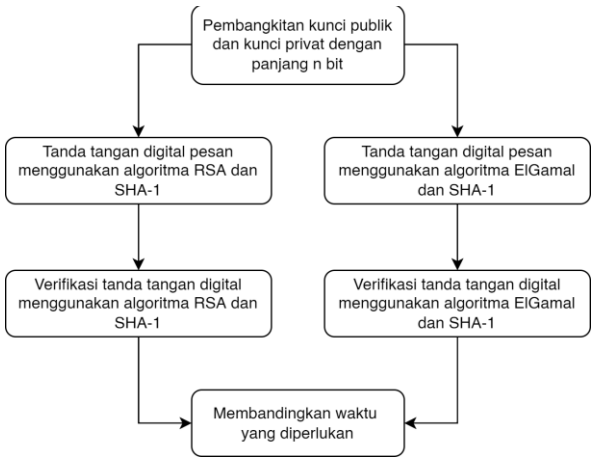
$$0 < s < p - 1$$

   2. Tanda tangan *valid* jika memenuhi persamaan berikut.

$$g^{H(m)} \equiv y^r \, r^s \, (mod \, p)$$

III. RANCANGAN

Analisis perbandingan efisiensi algoritma RSA dibandingkan dengan algoritma ElGamal pada tanda tangan digital akan diimplementasikan menggunakan panjang kunci yang berbeda. Berikut ini adalah gambaran alur dari program yang akan diimplementasikan.



Gambar 5. *Flowchart* Rancangan Alur Program

IV. IMPLEMENTASI DAN PENGUJIAN

A. Implementasi

Program tanda tangan digital menggunakan algoritma RSA, Algoritma ElGamal dan SHA-1 diimplementasikan menggunakan bahasa pemrograman Python. Berikut ini adalah kode programnya.

Fungsi *Generate Key* pada Algoritma RSA

```
def generate_key():
    # Memilih nilai p dan q secara random
    p = num.getPrime(9)
    q = num.getPrime(9)

    # Menghitung nilai n dan totient
    n = p * q
    totient = (p-1) * (q-1)

    # Menghitung kunci publik
    e = randint(3, 2**16+1)
    while(fpb(totient, e) != 1):
        e = randint(3, 2**16+1)

    # Menghitung kunci private
    k = 1
    d = 0.1
    while(d%1 != 0):
        d = (1 + k * totient)/e
        k += 1
    d = int(d)

    # Menyimpan kunci publik dan private
    f = open("key.pub", "w")
    f.write(str(e) + "," + str(n))
    f.close()

    f = open("key.pri", "w")
    f.write(str(d) + "," + str(n))
    f.close()

  return n
```

Fungsi *Sign* pada Algoritma RSA

```
def sign(n):
    # Membaca pesan awal dari file
    f = open("text.txt","r")
    message = f.read()
    f.close()

    # Membaca kunci private dari file
    f = open("key.pri","r")
    file_pkey = f.read()
```

```
        private_key = int(file_pkey.split(",")[0])
        private_n = int(file_pkey.split(",")[1])
        f.close()

        # Menghitung hash dari pesan awal
        hash = sha1(message.encode('utf-8'))
        digest = hash.hexdigest()

        # Menghitung sign dari hash
        s    =    HexToDec(digest)**private_key    %
private_n
        s_hex = DecToHex(s)

        # Menambahkan sign ke file
        f = open("text.txt", "a")
        text = "\n<ds>" + str(s_hex) + "</ds>"
        f.write(text)
    f.close()
```

Fungsi *Verify* pada Algoritma RSA

```
    def verif(n):
        # Membaca pesan dan sign dari file
        f = open("text.txt", "r")
        file_text = f.read()
        sign_message = file_text.split("\n<ds>")
        verif_message = sign_message[0]
        verif_sign                                  =
sign_message[1].split("</ds>")[0]
        f.close()

        # Membaca public key dari file
        f = open("key.pub","r")
        file_key = f.read()
        public_key = int(file_key.split(",")[0])
        public_n = int(file_key.split(",")[1])
        f.close()

        # Menghitung hash dari pesan dan signature
        hash_message                                =
HexToDec(sha1(verif_message.encode('utf-
8')).hexdigest()) % n
        hash_sign                                   =
HexToDec(verif_sign)**public_key % n

        if (hash_message == hash_sign):
            print("Tanda tangan digital otentik")
        else:
         print("Tanda    tangan    digital    tidak
otentik")
```

Fungsi *Generate Key* pada Algoritma ElGamal

```
    def generate_key():
        # Memilih nilai p secara random
        p = num.getPrime(32)

        # Menghitung generator
        g = randint(2, p-1)
        while(True):
            g = randint(2, p-1)
            if((p-1)%g != 1):
                break

        # Menyimpan nilai p dan g
        f = open("parameter.txt", "w")
        f.write(str(p) + "," + str(g))
        f.close()

        # Menghitung kunci
        x = randint(1, p-2)
        y = (g**x)%p

        # Menyimpan kunci publik dan private
        f = open("elkey.pub", "w")
        f.write(str(y))
        f.close()

        f = open("elkey.pri", "w")
        f.write(str(x))
     f.close()
```

Fungsi *Sign* pada Algoritma ElGamal

```
    def sign():
```

```
        # Membaca pesan awal dari file
        f = open("text.txt","r")
        message = f.read()
        f.close()

        # Membaca nilai p dan g
        f = open("parameter.txt","r")
        file_pkey = f.read()
        p = int(file_pkey.split(",")[0])
        g = int(file_pkey.split(",")[1])
        f.close()

        # Membaca kunci private dari file
        f = open("elkey.pri")
        file_pkey = f.read()
        private_key = int(file_pkey)
        f.close()

        s = 0

        while(s == 0):
            # Memilih nilai k
            k = randint(1, p-2)
            while(fpb(k, p-1) != 1):
                k = randint(1, p-2)

            # Menghitung nilai r
            r = (g**k)%p

            # Menghitung hash dari pesan awal
            hash = sha1(message.encode('utf-8'))
            digest = hash.hexdigest()

            # Menghitung sign dari hash
            s = ((HexToDec(digest)-
(private_key*r))/k)%(p-1)
            s_hex = DecToHex(s)

        # Menambahkan sign ke file
        f = open("text.txt", "a")
        text = "\n<ds>" + str(r) + "," + str(s_hex)
+ "</ds>"
        f.write(text)
    f.close()
```

Fungsi *Verify* pada Algoritma ElGamal

```
    def verif():
        # Membaca pesan dan sign dari file
        f = open("text.txt", "r")
        file_text = f.read()
        sign_message = file_text.split("\n<ds>")
        verif_message = sign_message[0]
        verif_sign                                  =
sign_message[1].split("</ds>")[0].split(",")
        r = int(verif_sign[0])
        s = HexToDec(verif_sign[1])
        f.close()

        # Membaca nilai p dan g
        f = open("parameter.txt","r")
        file_pkey = f.read()
        p = int(file_pkey.split(",")[0])
        g = int(file_pkey.split(",")[1])
        f.close()

        # Membaca public key dari file
        f = open("elkey.pub","r")
        file_key = f.read()
        public_key = int(file_key)
        f.close()

        # Menghitung hash dari pesan dan signature
        if (r > 0 and r < p and s > 0 and s < p-1):
            hash_message                            =
HexToDec(sha1(verif_message.encode('utf-
8')).hexdigest())
            if    ((g**hash_message)    %    p    ==
(public_key**r)*(r**s)):
                print("Tanda    tangan    digital
otentik")
            else:
                print("Tanda tangan digital tidak
otentik")
        else:
```

```
        print("Tanda tangan digital tidak
otentik")
```

Kode Pengujian Waktu

```
    start_time = time.time()

    generate_key()

    end_time = time.time()

    print("waktu generate key: " + str(end_time-
start_time) + " detik")


    start_time = time.time()

    sign()

    end_time = time.time()

    print("waktu sign: " + str(end_time-start_time)
+ " detik")


    start_time = time.time()

    verify()

    end_time = time.time()

print("waktu verifikasi: " + str(end_time-
start_time) + " detik")
```

B. Pengujian

Dilakukan percobaan pemberian tanda tangan digital terhadap sebuah pesan file text.txt dengan isi pesan sebagai berikut.

```
halo, ini adalah teks berisi pesan
```

Panjang kunci diinisialiasi dengan nilai n = 9 bit. Berikut ini adalah output waktu yang diperlukan untuk membangkitkan kunci, memberikan tanda tangan, dan memverifikasi tanda tangan.

Hasil Waktu *Digital Signature* dengan Algoritma RSA

```
p: 359
q: 263
waktu generate key: 0.00499415397644043 detik
waktu sign: 1.9489984512329102 detik
Tanda tangan digital otentik
waktu verifikasi: 0.03600025177001953 detik
```

Gambar 6. Hasil Pengujian 1

Hasil Waktu *Digital Signature* dengan Algoritma ElGamal

```
p: 269
g: 113
x: 261
y: 252
waktu generate key: 0.005013942718505859 detik
waktu sign: 0.04903531074523926 detik
```

Gambar 6. Hasil Pengujian 2

Dari hasil di atas, proses pembangkitan kunci memakan waktu yang relatif sama pada algoritma RSA dan ElGamal yaitu sekitar 0.004 – 0.005 detik. Proses pemberian tanda tangan pada algoritma RSA memakan waktu yang lebih lama daripada ElGamal karena operasi perpangkatan kunci. Proses verifikasi pada algoritma ElGamal membutuhkan waktu yang sangat lama sehingga tidak kunjung selesai karena operasi perpangkatan hasil *hash* yang nilainya sangat besar.

V. KESIMPULAN

Tanda tangan digital merupakan hal yang perlu dijaga keamanannya. Tanda tangan digital dapat diterapkan menggunakan beberapa algoritma kriptografi seperti RSA dan ElGamal. Dari hasil percobaan, ditentukan bahwa algoritma RSA lebih efisien daripada ElGamal karena waktu verifikasinya yang lebih cepat.

REFERENCES

[1] Munir, R. Slide Kuliah Algoritma Elgamal.

[2] Munir, R. Slide Kuliah Algoritma RSA.

[3] Munir, R. Slide Kuliah Secure Hash Algorithm (SHA).

[4] Munir, R. Slide Kuliah Tanda Tangan Digital.

[5] Taher ElGamal (1985). "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". IEEE Transactions on Information Theory. 31 (4): 469–472. CiteSeerX 10.1.1.476.4791. doi:10.1109/TIT.1985.1057074

[6] Mike Rosulek (2008-12-13). "Elgamal encryption scheme". University of Illinois at Urbana-Champaign. Archived from the original on 2016-07-22.

[7] Tsiounis, Yiannis; Yung, Moti (2006-05-24). "On the security of ElGamal based encryption". Public Key Cryptography 1998. Lecture Notes in Computer Science. 1431: 117–134. doi:10.1007/BFb0054019. ISBN 978-3-540-69105-1.

[8] https://www.signix.com/blog/bid/108804/infographic-the-history-of-digital-signature-technology

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, 25 Mei 2022

Jacelyn Felisha - 18219097