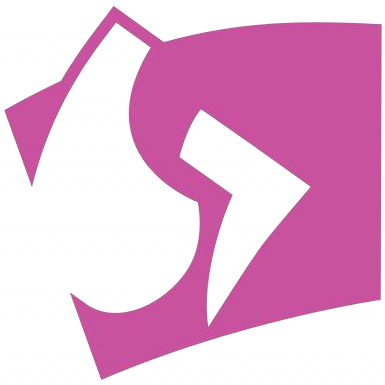
Implementasi Algoritma RSA dan SHA-3 sebagai Digital Signature Hasil Karya Anggota Unit Genshiken ITB

Muhammad Raihan Aulia 18220031 (*Author*)
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): raihanaulia332@gmail.com

***Abstract*—Hasil karya seni yang telah dibuat oleh seseorang merupakan milik pribadi orang tersebut. Mengakui karya seni milik orang lain sebagai miliknya tentu merupakan perilaku yang tidak terpuji. Hal ini juga terjadi pada Unit Genshiken ITB. Tidak sedikit anggota Unit Genshiken ITB yang merasa tidak aman dengan karya seninya. Akibat dari rasa tidak aman tersebut, anggota Unit Genshiken ITB merasa enggan untuk membuat karya seni dan menampilkan karya tersebut. Oleh karena itu, dibutuhkan teknologi yang dapat memastikan karya tersebut milik seseorang. Salah satu teknologi yang dapat digunakan adalah teknologi *Digital Signature*. Untuk mengimplementasikan teknologi ini, dapat digunakan berbagai pasangan algoritma. Salah satu dari pasangan tersebut adalah pasangan Algoritma RSA dan SHA-3.**

Keywords—RSA; SHA-3; Digital Signature; Public Key; Private Key; Encryption

I. PENDAHULUAN



Gambar 1. Logo Genshiken

(Sumber: https://genshiken-itb.org)

Unit Kebudayaan Genshiken ITB merupakan salah satu unit yang terdapat pada ITB. Unit ini memiliki fokus menghasilkan karya kontemporer. Untuk mewadahi tujuannya, Genshiken memiliki beberapa divisi kekaryaan seperti divisi musik yang berfokus menghasilkan karya seni berjenis musik dan divisi *story* yang berfokus untuk menghasilkan karya seni berjenis cerita. Hak mengenai karya-karya yang dihasilkan oleh anggota-anggota Genshiken diserahkan kepada anggota yang menghasilkan karya tersebut. Walaupun hak-hak karya dipegang oleh anggota yang menghasilkan karya tersebut, Genshiken ITB berhak mengarsip karya sebagai bukti karya tersebut dihasilkan oleh anggota Genshiken dan agar karya seni yang dihasilkan tidak luput oleh waktu.

Karya-karya yang dihasilkan oleh Genshiken saat ini sudah cukup banyak sehingga cukup sulit untuk dilacak siapa pembuat dari karya tersebut. Jika terdapat salah satu anggota yang melakukan klaim terhadap karya seni yang dihasilkan oleh anggota lainnya, tidak dapat dilakukan pelacakan selain menggunakan gaya seni yang digunakan dalam proses pembuatan karya tersebut. Ketika tidak ada yang dapat membuktikan karya tersebut, tidak ada yang bisa menolak klaim dari anggota tersebut.

Selain masalah klaim suatu karya seni, terdapat juga anggota yang menghasilkan suatu karya seni tetapi menyangkal sudah membuat karya seni tersebut. Alasan kenapa anggota tersebut menyangkal sudah membuat karya seni tersebut biasanya karena malu sudah membuat karya seni tersebut atau karya seni yang dibuat melanggar aturan yang ada pada Unit Genshiken ITB. Ketika penyangkalan terjadi, pihak badan pengurus Unit Genshiken ITB akan mengalami kesulitan untuk melacak pembuat karya seni tersebut.

II. DASAR TEORI

A. *Kriptografi*

Kriptografi adalah ilmu untuk mengamankan pesan. Dengan kriptografi, orang selain penerima dan pengirim pesan tidak dapat membaca pesan [5]. Kriptografi sudah digunakan sejak zaman dulu untuk memastikan pesan tidak dapat dibaca oleh musuh selama pesan dikirim. Terdapat beberapa aspek yang dijaga oleh kriptografi.

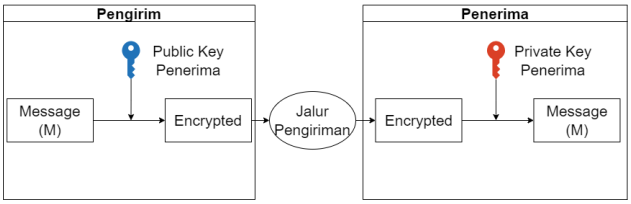
1. Kerahasiaan Pesan,

2. keutuhan Pesan,
3. pengirim pesan adalah asli,
4. pengirim pesan tidak dapat menyangkal telah mengirim pesan.

Untuk menjaga aspek-aspek tersebut, terdapat beberapa teknik yang diterapkan. Teknik-teknik yang digunakan untuk kriptografi dibagi menjadi dua berdasarkan 2 masa dibentuknya teknik tersebut yaitu kriptografi klasik dan kriptografi modern. Untuk implementasi yang akan dilakukan pada makalah ini, teknik kriptografi yang digunakan adalah teknik kriptografi modern.

B. Algoritma RSA

Algoritma RSA (Rivest-Shamir-Adleman) adalah sebuah algoritma kriptografi modern yang menggunakan konsep kunci asimetris. Algoritma ini dibuat oleh tiga orang yaitu Ron Rivest, Adi Shamir, dan Leonard Adleman. Dari ketiga nama pembuat tersebut, Algoritma RSA mendapatkan namanya. Kesulitan pemecahan Algoritma RSA terletak pada kesulitan untuk memfaktorkan hasil perkalian dari dua bilang prima yang besar. Karena kesulitan pemecahannya, Algoritma RSA masih digunakan sampai sekarang pada berbagai hal.



Gambar 2. Skema Kriptografi Kunci Publik

(Sumber: Dokumentasi Penulis)

Pada algoritma RSA, terdapat dua jenis kunci yang digunakan yaitu kunci publik dan kunci privat. Kunci publik adalah kunci yang diketahui oleh banyak orang sehingga sementara kunci privat adalah kunci yang hanya diketahui oleh pemilik kunci tersebut. Kedua kunci tersebut dapat digunakan untuk melakukan enkripsi tetapi untuk melakukan dekripsi pesan, hanya pasangan kunci yang dapat digunakan untuk melakukan dekripsi pesan [1]. Tetapi, kunci yang digunakan untuk melakukan enkripsi pesan adalah kunci publik penerima pesan dan kunci yang digunakan untuk mendekripsi pesan adalah kunci privat penerima pesan.

Berikut merupakan langkah-langkah untuk menghasilkan pasangan kunci publik dan kunci privat yang digunakan untuk melakukan enkripsi dan dekripsi pesan pada Algoritma RSA:

1. ditentukan dua bilangan prima yang besar, p dan q dengan nilai p tidak sama dengan nilai q. Selanjutnya, kedua bilangan prima dikalikan sehingga

$$n = p * q \tag{1}$$

Walaupun nilai n diumumkan kepada orang lain, nilai faktor dari n yaitu p dan q akan disembunyikan karena kesulitan untuk memfaktorkan bilangan tersebut. Karena p dan q dapat disembunyikan, faktor-faktor lainnya yaitu kunci privat d dan kunci publik e dapat disembunyikan.

2. Selanjutnya, dipilih bilangan e yang merupakan bilangan acak dan besar yang merupakan bilangan relatif prima dengan hasil perkalian antara (p-1) dan (q-1) sehingga memenuhi persamaan (2)

$$gcd(e, (p-1)*(q-1)) = 1 \tag{2}$$

dengan "gcd" adalah pembagi persekutuan terbesar.

3. Kunci privat d merupakan *modular multiplicative inverse* dari e sehingga

$$d \equiv e^{-1} mod\ ((p-1)*(q-1)) \tag{3}$$

Dengan ditemukan d dan e, pembuatan pasangan kunci privat dan kunci publik selesai.

Untuk penggunaan kunci publik pada Algoritma RSA, berikut langkah-langkah yang harus dilakukan:

1. Pada enkripsi pesan M, dilakukan dengan mengubah M menjadi blok-blok pesan yang lebih kecil m_1, m_2, m_3, dan seterusnya dengan aturan

$$0 \leq m < n \tag{4}$$

2. Selanjutnya tiap blok-blok pesan diproses dengan menggunakan kunci publik dengan persamaan

$$c_i = e^{mi}\ mod\ \ n \tag{5}$$

Untuk melakukan dekripsi, berikut langkah-langkah yang harus dilakukan:

1. Enkripsi pesan dipecah-pecah menjadi blok-blok c_1, c_2, c_3, dan seterusnya.
2. Blok-blok pesan m didapatkan dengan menggunakan persamaan

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

3. Pesan M didapatkan dengan merangkai pesan kembali menggunakan blok-blok pesan yang dihasilkan dari persamaan sebelumnya.

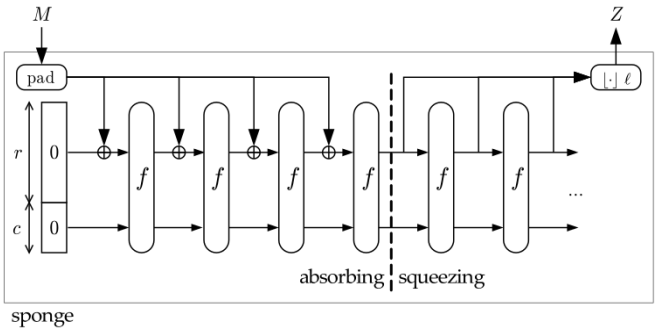
C. Algoritma Hash SHA-3 (Keccak)

Secure Hash Algorithm 3, disingkat SHA-3, adalah standar algoritma hash yang paling baru dalam keluarga standar *secure hash algorithm*. Algoritma *Hash* SHA-3 sering disebut sebagai Algoritma Keccak yang merupakan algoritma pemenang pada kompetisi SHA-3. Algoritma Keccak sendiri dibuat oleh Michael Peeters, Guido Bertoni, Joan Daemen, Ronny Van Keer, dan Gilles Van Assche. Algoritma ini pertama kali dipublikasikan pada tahun 2016.

Algoritma Keccak termasuk ke dalam kelompok fungsi hash yang artinya fungsi ini bersifat *irreversible*. Nilai *hash*

yang dihasilkan dari fungsi ini bersifat unik yang artinya jika dimasukkan pesan yang berbeda, akan menghasilkan nilai hash yang berbeda juga. Tidak seperti finalis SHA3 lainnya yang menggunakan fungsi kompresi, Algoritma ini menggunakan konstruksi 'spons' sebagai dasar teorinya [2]. Proses *hashing* dilakukan dengan cara 'menyerap' pesan dan kemudian memeras '*digest'*. Kelebihan dari algoritma dibandingkan dengan algoritma lainnya adalah kebebasan untuk menentukan panjang hasil *hash* yang diinginkan.

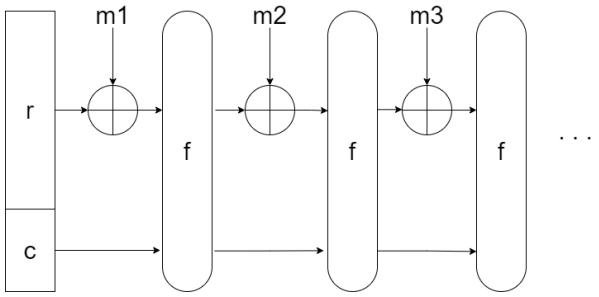
Dalam proses *hashing*, pesan M dipersiapkan terlebih dahulu agar dapat diproses. Persiapan dilakukan dengan menambahkan beberapa bit pengganjal pada pesan sehingga M habis dibagi panjang proses r dengan r + c adalah *State* S yang digunakan untuk menangkap pesan dan dipersiapkan untuk diproses pada tahap-tahap selanjutnya.. Proses ini disebut juga sebagai *padding*. Tahap ini terjadi ketika awal proses *hashing* dilakukan, sesuai dengan gambar di bawah.



Gambar 3. Model Spons pada Algoritma Keccak

(Sumber: https://keccak.team/sponge_duplex.html)

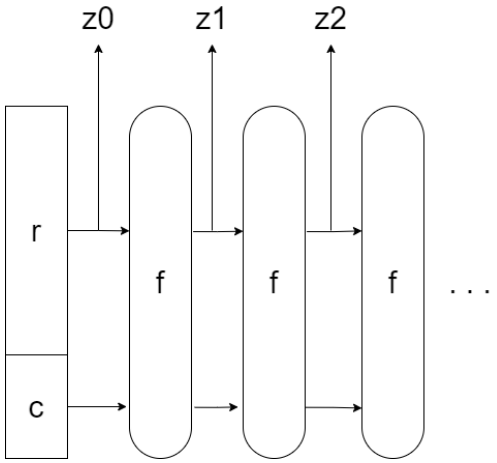
Setelah M diproses dengan melakukan *padding*, M dibagi-bagi menjadi blok-blok pesan m_1, m_2, m_3, dan seterusnya sampai M habis. Panjang blok harus sama besar dengan panjang r pada *State* S. Setelah pembagian dilakukan, blok pertama pesan dilakukan operasi XOR dengan r bit pertama dari S. Setelah itu, S digunakan sebagai masukan dari fungsi permutasi sehingga menghasilkan *State* baru S'. Setelah terbentuknya S', blok pesan kedua yaitu m_2 dilakukan operasi XOR dengan r bit pertama dari S' kemudian digunakan sebagai masukan untuk fungsi permutasi selanjutnya sehingga menghasilkan *State* baru. Tahap ini disebut dengan tahap penyerapan atau *absorbing* dan berlangsung sampai semua blok pesan selesai diserap oleh spons.



Gambar 4. Tahap Penyerapan pada Keccak

(Sumber: Dokumentasi Penulis)

Setelah semua blok pesan selesai diserap dan dilakukan permutasi, diambil r bit pertama dari *State* hasil permutasi sehingga didapatkan blok *hash* z_0. Selanjutnya, *State* diproses lagi dengan menggunakan fungsi permutasi sehingga didapatkan *State* baru. blok z_0 kemudian ditambahkan ke dalam hasil *hash*. Proses ini dilakukan secara terus menerus sampai panjang hasil *hash* sama dengan panjang yang diinginkan. Tahap ini disebut juga tahap pemerasan atau *squeezing*.



Gambar 5. Tahap Penyerapan pada Keccak

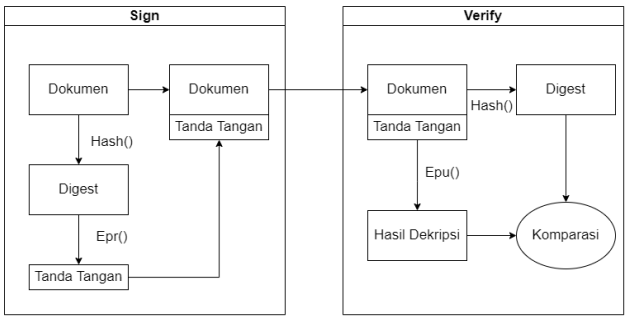
(Sumber: Dokumentasi Penulis)

D. Digital Signature

Digital Signature atau tanda tangan digital adalah suatu metode yang digunakan untuk membuktikan keaslian suatu pesan. Dengan tanda tangan digital, dapat dipastikan pembuat pesan tersebut adalah orang yang dikenal dan pesan tidak diubah setelah dokumen ditandatangani [3]. Metode ini menggunakan kriptografi kunci publik untuk implementasinya.

Pada algoritma kriptografi kunci publik, terdapat kunci privat yang hanya diketahui oleh pemilik kunci. Ketika pemilik membuat pesan dan pemilik kunci ingin melakukan enkripsi menggunakan kunci privat, dibutuhkan kunci publik dari pelaku enkripsi untuk mendapatkan pesan kembali. Kunci publik merupakan kunci yang sudah diketahui oleh banyak orang sehingga kunci ini tidak rahasia lagi. Dengan begitu, pembuat pesan tidak bisa membantah sudah menandatangani pesan tersebut dan dapat dibuktikan dengan menggunakan kunci publik pembuat pesan yang sudah diketahui oleh orang lain [4].

Pada praktik umumnya, tanda tangan digital dilakukan dengan menggunakan fungsi *hash*. Fungsi *hash* digunakan untuk mendapatkan *digest* dari pesan yang ingin dikirimkan sebagai perwakilan dari konten pesan. Fungsi ini efektif karena jika pesan diubah, akan menghasilkan hasil *hash* yang juga berbeda. Selain itu, fungsi *hash* menghasilkan *digest* yang umumnya lebih kecil dibandingkan dengan pesan awal sehingga proses enkripsi menggunakan algoritma kunci publik menjadi lebih cepat.



Gambar 6. Diagram Proses Tanda Tangan Digital

(Sumber: Dokumentasi Penulis)

Untuk menandatangani dokumen, proses dilakukan dengan cara melakukan *hashing* pada dokumen yang ingin dikirimkan. Setelah *hashing* dilakukan, hasil *hash* yang berupa *digest* di enkripsi menggunakan kunci privat milik pembuat dokumen menghasilkan tanda tangan digital dari dokumen yang akan dikirimkan. Selanjutnya pesan dikirim bersama dengan tanda tangan digital.

Untuk memeriksa atau memverifikasi tanda tangan digital, proses dilakukan dengan melakukan *hashing* pada dokumen. Secara bersamaan, penerima dokumen melakukan dekripsi tanda tangan digital dengan menggunakan kunci publik pembuat dokumen. Setelah proses dekripsi selesai dilakukan, hasil dekripsi dan hasil *hash* dokumen dibandingkan. Jika hasil perbandingan menunjukkan hasil *hash* dokumen sama dengan hasil dekripsi tanda tangan, dokumen terbukti dibuat oleh pembuat dokumen dan terbukti tidak terjadi perubahan setelah dokumen ditandatangani.

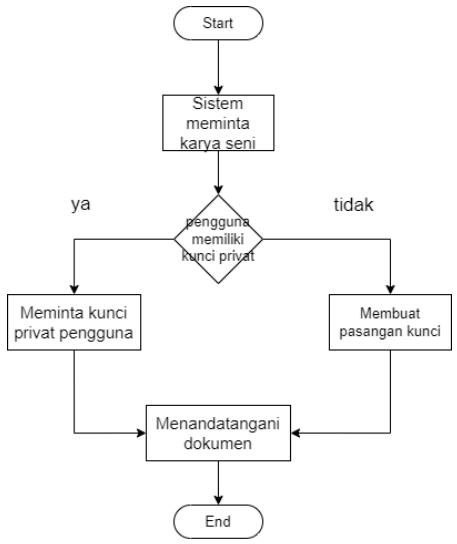
III. Rancangan Solusi dan Implementasi

Masalah yang diangkat pada makalah ini adalah anggota unit yang mengakui suatu karya seni diciptakan oleh anggota tersebut. Selain itu, terdapat anggota yang tidak mengakui suatu karya seni diciptakan oleh anggota tersebut. Untuk itu, dibutuhkan solusi yang dapat menyelesaikan permasalahan tersebut. Berikut rancangan dan implementasi solusi untuk permasalahan tersebut.

A. Rancangan Solusi

Solusi direncanakan berupa sistem yang dapat membuat dan memeriksa tanda tangan digital karya seni yang dihasilkan oleh anggota unit. Algoritma yang digunakan pada sistem adalah Algoritma RSA sebagai algoritma kriptografi kunci publik. Untuk melakukan proses *hashing*, digunakan SHA-3.

Untuk melakukan tanda tangan, Sistem menerima karya seni dan menandatangani karya seni tersebut sesuai dengan kunci privat yang diberikan oleh pengguna. Setelah proses penandatanganan selesai, sistem akan memberikan dokumen khusus yang berisi tanda tangan terkait karya seni.

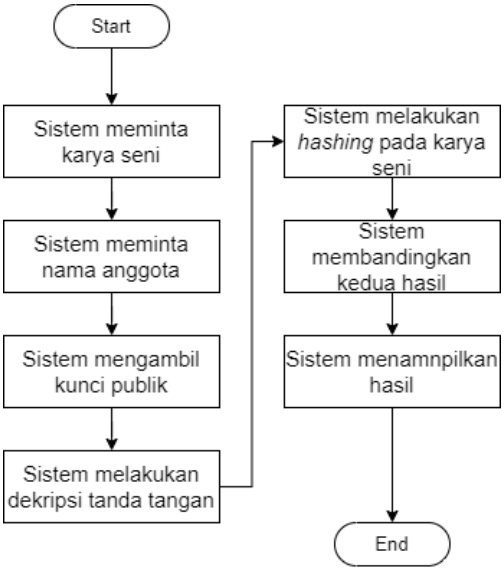


Gambar 7. Skema Tanda Tangan Solusi

(Sumber: Dokumentasi Penulis)

Untuk memverifikasi karya seni, sistem akan melakukan *hashing* pada dokumen digital karya seni yang diberikan pengguna. Setelah itu, sistem akan meminta pengguna memasukkan dokumen tanda tangan. Kemudian sistem akan mencoba mengambil data kunci publik sesuai dengan nama pembuat karya seni yang disimpan di dalam sistem dan mencoba melakukan dekripsi tanda tangan digital. Selanjutnya, sistem akan membandingkan hasil *hash* karya seni dan hasil dekripsi tanda tangan digital. Hasil dari komparasi akan diberikan kepada pengguna.

Jika hasil komparasi menunjukkan hasil *hash* karya seni sama dengan hasil dekripsi tanda tangan digital, dapat dibuktikan pembuat dokumen tersebut adalah anggota tersebut. Sementara jika hasil komparasi menunjukkan hasil *hash* karya seni berbeda dengan hasil dekripsi tanda tangan digital, belum bisa ditentukan apakah dokumen tersebut dibuat oleh anggota tersebut atau tidak. Hal ini terjadi karena terdapat kemungkinan dokumen digital diubah setelah tanda tangan digital diubah.



Gambar 8. Skema Tanda Tangan Solusi

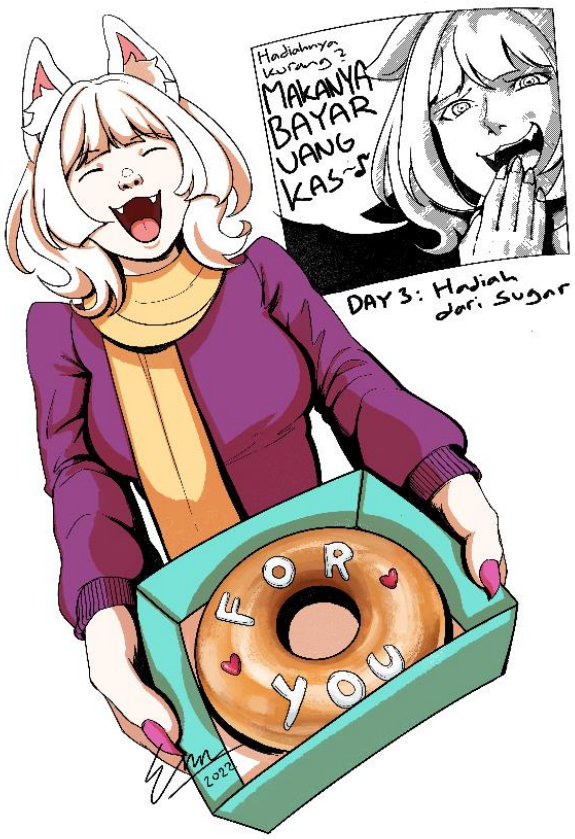
(Sumber: Dokumentasi Penulis)

B. Implementasi

Sistem sebagai solusi dijalankan diimplementasikan menggunakan bahasa pemrograman Python. Untuk basis data sistem, digunakan SQLite yang merupakan salah satu bentuk dari sistem manajemen basis data relasional. Berikut adalah kode program hasil implementasi solusi. Dibuat juga beberapa modul-modul untuk memenuhi fungsi-fungsi dari sistem. Berikut adalah modul-modul sistem dalam bahasa Python. Hasil berupa kode program dilampirkan pada bagian lampiran.

C. Uji Coba Implementasi

Unit Genshiken ITB merupakan unit kebudayaan yang menghasilkan karya-karya kontemporer. Hasil karya dapat berupa cerita, lagu, gambar, dan lainnya. Untuk mewakili hasil karya anggota Unit Genshiken ITB, digunakan dokumen digital sembarang yaitu sebuah dokumen gambar hasil salah satu anggota Unit Genshiken ITB.



Gambar 9. Gambar Hasil Karya Anggota Genshiken

(Sumber: Hadiah dari Sugar - Ruth Ardita Damanik)

Pada awal penghidupan sistem, pengguna disambut dengan tampilan menu dan beberapa opsi. Terdapat tiga pilihan yang dapat dipilih pengguna dengan cara mengetik pilihan menggunakan *keyboard*. Pilihan pertama yaitu pilihan *sign* digunakan untuk menandatangani dokumen. Pilihan kedua yaitu pilihan *verify* digunakan untuk memastikan keaslian dokumen digital. Pilihan ketiga yaitu pilihan keluar digunakan oleh pengguna untuk keluar dari sistem.

```
Selamat datang di sisteeeemm
 ~~~DIGISIGN GENSHI DESUUUUU~~~

Masukkan Command yang anda ingingkan!!

> sign -> melakukan sign pada dokumen, nyaa~
> verify -> memastikan kebenaran dokumen, nyaa~
> keluar -> keluar dari sistem, nyaa~

>
```

Gambar 10. Menu Utama Sistem

(Sumber: Dokumentasi Penulis)

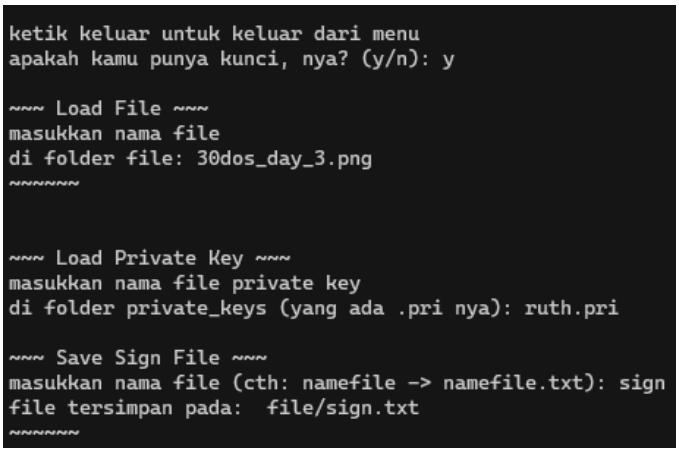
Jika pengguna memilih pilihan pertama, pengguna akan ditanya kepemilikan kunci. Apabila pengguna belum mempunyai kunci, pengguna akan diminta untuk mengisi nama pengguna dan nama berkas kunci privat milik pengguna untuk membuat kunci privat baru milik pengguna dan memasukkan nama pengguna beserta kunci publik yang dibuat bersamaan dengan kunci privat ke dalam basis data sistem. Apabila pengguna menjawab dengan sudah memiliki kunci, pengguna akan diminta nama berkas karya, nama berkas kunci privat pengguna, dan nama berkas tanda tangan digital yang akan dihasilkan. Setelah semua pertanyaan terpenuhi, berkas tanda tangan akan dihasilkan oleh sistem.

```
> sign
ketik keluar untuk keluar dari menu
apakah kamu punya kunci, nya? (y/n): n
membuat key ...
masukkan nama kamu, nya: ruth
ini private key kamu:
(42930284106668084307033785646692266753033904549937381028659755280
41113735947570961505787486062450536222398774543378377145382062995 5
78458830441057013052533931839457301372058392877034052123694789113 5
585, 55140366551532710906960661666889658580069524186530140935066 53
84533546736134999765988320600248182763933710637606750645256814256 4
30115201389007350796378301718182805753294723956867944653669230385 6
6423977)

~~~ Save Private Key ~~~
masukkan nama file (cth: namefile -> namefile.pri): ruth
~~~~~~
```

Gambar 11. Membuat Kunci Baru

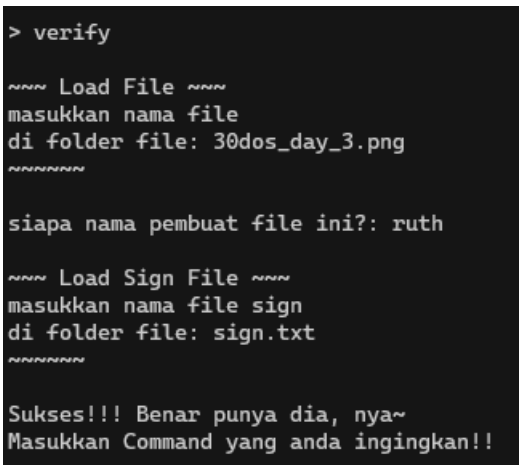
(Sumber: Dokumentasi Penulis)



Gambar 12. Melakukan tanda tangan

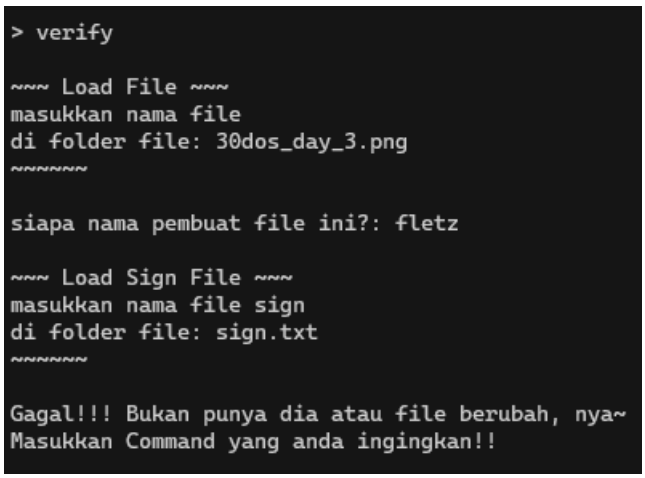
(Sumber: Dokumentasi Penulis)

Jika pengguna memilih pilihan *verify* pada menu utama, pengguna akan diminta mengisi beberapa hal, yaitu nama berkas karya yang ingin diuji keasliannya, nama pembuat berkas karya, dan nama berkas tanda tangan. Setelah pengguna menjawab semua permintaan sistem, sistem akan mencari kunci publik pembuat berkas karya dalam basis datanya menggunakan nama pembuat berkas karya. Setelah itu, karya seni di-*hash* sehingga menghasilkan sebuah *digest*. *Digest* kemudian dibandingkan dengan hasil dekripsi dan sistem akan menampilkan keluaran sesuai dengan hasil perbandingan.



Gambar 13. Verifikasi dengan Nama Anggota Pembuat Karya

(Sumber: Dokumentasi Penulis)



Gambar 14. Verifikasi dengan Nama Anggota Bukan Pembuat Karya

(Sumber: Dokumentasi Penulis)

IV. KESIMPULAN DAN SARAN

A. *Kesimpulan*

Untuk menjaga aspek keutuhan dan *authentication* dokumen, tanda tangan digital cukup baik diterapkan. Karena menggunakan kriptografi kunci publik, Pembuat karya tidak bisa menyanggah sudah membuat karya tersebut jika karya sudah ditandatangani. Tetapi, untuk mencari tahu apakah seseorang yang melakukan klaim terhadap hasil karya seni cukup sulit. Hal ini terjadi karena terdapat kemungkinan adanya perubahan pada dokumen digital sehingga hasil komparasi pasti akan berbeda.

B. *Saran*

Sebaiknya antarmuka sistem tidak hanya sekedar *command line interface* (CLI). Hal ini cukup menyulitkan pengguna karena pengguna harus mengetahui *path* berkas-berkas yang diperlukan. Selain itu, kesulitan seperti informasi yang diberikan menjadi terlalu padat akan mempersulit analisis.

VIDEO LINK AT YOUTUBE *(Heading 5)*

https://youtu.be/ll4ciuWn54c

REFERENCES

[1] R. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital SIgnatures and Public-Key Cryptosystems," *Communications of the ACM*, p. 120–126, 1978.

[2] M. Peeters, G. Bertoni, J. Daemen, R. Van Keer and G. Van Assche, "The sponge and duplex constructions," [Online]. Available: https://keccak.team/sponge_duplex.html.

[3] E. Paul, "What is Digital Signature: How it works, Benefits, Objectives, Concept," EMP Trust HR, 12 September 2017. [Online]. Available: https://www.emptrust.com/blog/benefits-of-using-digital-signatures/. [Accessed 22 May 2023].

[4] J. Chia, J.-J. Chin and S.-C. Yip, "Digital signature schemes with strong existential unforgeability," 16 November 2021. [Online]. Available: https://doi.org/10.12688/f1000research.72910.1. [Accessed 22 May

2023].

[5] M. Bellare and P. Rogaway, Introduction to Modern Cryptography, California, 2005, p. 14.

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

Muhammad Raihan Aulia
18220031

Lampiran

Hasil implementasi berupa kode program terdapat pada link github berikut.

https://github.com/FlitzyBlue332/kripto-makalah

Modul Utama Sistem (main.py)

```
import base64
from src.rsa import *
from src.sha3 import sha3_512 as hash
from src.utility import *
from src.database import *

def sign():
    private_key = None
    # dapatin kunci
    haskey = 'uwu'
    while(private_key == None and haskey != 'keluar'):
        print("ketik keluar untuk keluar dari menu")
        print("apakah kamu punya kunci, nya? (y/n): ", end='')
        haskey = input()

        if(haskey != 'y'):
            generatekey()

        else:
            content = openFile()
            hashed_content = hash(content)
            private_key = openPrivateKey()

            encrypted = encrypt(private_key, int(hashed_content, 16))
            saveSignFile(hex(encrypted))

def verify():
      public_key = None
      content = openFile()
      hashed_content = hash(content)

      # ambil public key
      print("siapa nama pembuat file ini?: ", end='')
      nama = input()
      data = getdatawithname(nama)

      if(len(data) != 0):
        public_key = stringtokey(data[0].public_keys)
        sign = openSignFile()
        decrypted_sign = decrypt(public_key, int(sign, 16))
        if(decrypted_sign == int(hashed_content,16)):
            print("Sukses!!! Benar punya dia, nya~")
        else:
            print("Gagal!!! Bukan punya dia atau file berubah, nya~")



def generatekey():
    # generate key
            print("membuat key ...")
            public_key, private_key = generate_key_pair(generate_prime_number(), generate_prime_number())
            print("masukkan nama kamu, nya: ", end='')
            name = input()

            # masukin fungsi masukin data baru ke database
            insertdata(name, str(public_key))
            print("ini private key kamu: ")
            print(private_key)
            savePrivateKey(private_key)

command = "ulang"
print("Selamat datang di sisteeeemm\n ~~~DIGISIGN GENSHI DESUUUUU~~~\n")
while command != 'keluar':
    print("Masukkan Command yang anda ingingkan!!\n")
    print("> sign -> melakukan sign pada dokumen, nyaa~")
    print("> verify -> memastikan kebenaran dokumen, nyaa~")
    print("> keluar -> keluar dari sistem, nyaa~")
    print("\n> ", end='')
    command = input()
    if(command == 'sign'):
        sign()
    elif(command == 'verify'):
        verify()
    elif(command == 'keluar'):
         print("Bye byee~~")
    else:
         print("masukan anda salah, nyaa~.\n masukan yang bener dong!!")
```

Modul RSA

```
import random
from math import gcd
from Crypto.Util.number import isPrime, inverse

# program ini hanya menerima dan mengeluarkan bilangan bulat
def totient(p, q):
    return (p-1)*(q-1)

def generate_prime_number():
    while True:
        p = random.randint(10**50, 10**100)
        if isPrime(p):
            return p

def generate_key_pair(p, q):
    '''
    menghasilkan public key, private key
    '''
    n = p * q
    m = totient(p, q)
```

```
    while True:
        e = random.randint(1, m-1)
        if gcd(e, m) == 1:
            break
    d = inverse(e, m)
    return (e, n), (d, n)

def encrypt(public_key, plaintext):
    key, n = public_key
    cipher = pow(plaintext, key, n)
    return cipher

def decrypt(private_key, ciphertext):
    key, n = private_key
    plain = pow(ciphertext, key, n)
    return plain

def encypttext(public_key, plaintext):
    key, n = public_key
    cipher = pow(plaintext, key, n)
    return cipher
```

Modul SHA-3 (sha3.py)

```
import hashlib


def sha3_256(data):
    hash_object = hashlib.sha3_256()
    hash_object.update(data)
    return hash_object.hexdigest()

def sha3_512(data):
    hash_object = hashlib.sha3_512()
    hash_object.update(data)
    return hash_object.hexdigest()
```

Modul Database (database.py)

```
from typing import Optional
from sqlmodel import Field, Session, SQLModel,
create_engine, select


engine = create_engine("sqlite:///public_keys.db")
s = Session(engine)

class Public_keys(SQLModel, table=True):
    '''
    isi database adalah:
    nama:str
    public_keys:str
    '''
    id: Optional[int] = Field(default=None,
primary_key=True)
    nama:str
    public_keys:str

# query
def getalldata():
    '''
    ambil semua data pada tabel train
    '''
    statement = select(Public_keys)
    datas = s.exec(statement).all()
    return datas

def getdatawithname(name):
    '''
    ambil data sesuai nama
    '''
    statement =
select(Public_keys).where(Public_keys.nama == name)
    datas = s.exec(statement).all()
    return datas

def insertdata(nama, public_keys):
    '''
    insert data sesuai input
    '''
    sample = Public_keys(
        nama=nama,
        public_keys=public_keys
    )
    s.add(sample)
    s.commit()
```

Modul Utility (utility.py)

```
def savePrivateKey(private_key):
    print("\n~~~ Save Private Key ~~~")
    print("masukkan nama file (cth: namefile ->
namefile.pri): ", end='')
    file_name = input()
    f = open("private_keys/"+file_name+".pri", 'w')
    f.write(str(private_key))
    f.close()
    print("~~~~~~\n")


def openPrivateKey():
    print("\n~~~ Load Private Key ~~~")
    print("masukkan nama file private key \ndi folder
private_keys (yang ada .pri nya): ", end='')
    file_name = input()
    try:
        f = open("private_keys/"+file_name, 'r')
        content = f.read()
        f.close()
        return stringtokey(content)
    except:
        print("~~~~~~\n")
        print("ngga ada nama file itu di folder
private_keys, nyaa~")

    print("~~~~~~\n")

def saveSignFile(content):
    print("\n~~~ Save Sign File ~~~")
    print("masukkan nama file (cth: namefile ->
namefile.txt): ", end='')
    file_name = input()
    f = open("file/"+file_name+".txt", 'w')
    f.write(str(content))
    f.close()
    print("file tersimpan pada: ",
"file/"+file_name+".txt")
    print("~~~~~~\n")

def openSignFile():
    print("\n~~~ Load Sign File ~~~")
    print("masukkan nama file sign \ndi folder file: ",
end='')
    file_name = input()
    try:
        f = open("file/"+file_name, 'r')
        content = f.read()
        f.close()
        print("~~~~~~\n")
        return content
    except:
```

```
        print("ngga ada nama file itu di folder file,
nyaa~")

    print("~~~~~~\n")

def openFile():
    print("\n~~~ Load File ~~~")
    print("masukkan nama file \ndi folder file: ",
end='')
    file_name = input()
    try:
        f = open("file/"+file_name, 'rb')
        content = f.read()
        f.close()
        print("~~~~~~\n")
        return content
    except:
        print("ngga ada nama file itu di folder file,
nyaa~")

    print("~~~~~~\n")



def stringtokey(stringkey:str):
    '''
    Mengembalikan nilai key dari key yg dalam bentuk
string
    '''
    key_temp = (stringkey.replace(')', '')).split('(')[1]
    k = key_temp.split(', ')[0]
    n = key_temp.split(', ')[1].replace("'", '')
    key = int(k), int(n)
    return key
```