Implementasi Tanda Tangan Digital untuk Pengamanan Sertifikat yang Didigitisasi Terkhusus untuk Kegiatan Internal Institut Teknologi Bandung

Vic Finnegan Dyell (18220019)
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: vicdyell.vd@gmail.com

***Abstract*—Pada beberapa tahun kemarin, terlebih setelah adanya masa pandemi, terjadi perubahan yang cukup besar dalam sisi teknologi. Hampir semua hal berubah menjadi digital tidak terkecuali sertifikat baik sertifikat organisasi, kepanitiaan, maupun lomba. Hal ini mengakibatkan perlu adanya administrasi yang dikelola secara terintegrasi untuk mengatasi permasalahan banyaknya pemalsuan sertifikat demi kebanggaan atau mengisi CV semata. Maka dari itu dibuatlah suatu solusi dengan melakukan verifikasi menggunakan tanda tangan digital yang dibangkitkan dengan menggunakan algoritma RSA dan fungsi SHA-3 (Keccak).**

Keywords—tanda tangan digital; sertifikat yang didigitisasi; RSA; SHA-3; Keccak

I. PENDAHULUAN

Adanya masa pandemi sejak tahun 2020, benar-benar mengubah hampir semua sektor di kehidupan ini menjadi lebih digital dengan menggunakan teknologi yang telah tersedia. Seperti contoh, kuliah yang awalnya diharuskan ke kampus tiba-tiba bisa diselenggarakan secara daring dengan menggunakan aplikasi video *conference*. Tugas-tugas yang pada tahun-tahun dulu selalu dicetak di kertas sebelum dikumpulkan juga pada zaman sekarang semuanya dikumpulkan secara digital baik menggunakan gforms ataupun platform lainnya. Begitu juga dengan penerbitan sertifikat yang pada saat ini hampir semua sertifikat organisasi, kepanitiaan, ataupun lomba berupa sertifikat yang didigitisasi (*digitized certificate*).

Pada saat sertifikat masih dicetak secara manual, keamanan yang disertakan cukup mumpuni, misalnya saja dengan menggunakan tambahan hologram, benang pengaman, atau setidaknya dari tekstur sertifikatnya sehingga sulit untuk dipalsukan sebab biasanya ketika dipalsukan setidaknya ada warna dari tinta yang berbeda antara yang palsu dan asli [1]. Namun, hal berbeda dengan sertifikat yang didigitisasi. Sertifikat yang didigitisasi sangat sulit untuk diberikan keamanan. Memang secara umum, pada setiap sertifikat memiliki nomor seri, namun hal tersebut sangat sulit untuk dipercaya sebab tidak ada juga alat untuk memverifikasinya, apalagi jika sertifikat tersebut dikirim ke pihak eksternal. Maksudnya adalah misalnya pihak ITB membuat suatu sertifikat kepada mahasiswa A atas pengabdian kepanitiaan A dalam membuat suatu acara, lalu sertifikat tersebut digunakan oleh A untuk mengisi CV dan melamar kerja di perusahaan X. Perusahaan X tidak bisa memverifikasi sertifikat tersebut termasuk memverifikasi nomor seri sertifikatnya.

Untuk mengatasi permasalahan tersebut, maka dibutuhkan suatu sistem pengamanan sertifikat yang didigitisasi yang dapat digunakan oleh para organisasi tekhusus Institut Teknologi Bandung. Sistem pengamanan yang dimaksud merupakan tanda tangan digital yang akan dibuat dalam suatu file terpisah yang disebut dengan file *signature* sehingga jika nantinya sertifikat tersebut dikirim ke instansi lain, instansi tersebut dapat melakukan verifikasi sertifikat yang digitisasi dengan menggunakan file *signature* yang tersedia dengan cara dimasukkan ke dalam suatu aplikasi untuk mengecek status keaslian dari sertifikat yang didigitisasi tersebut.

II. DASAR TEORI

A. Tanda Tangan Digital

Tanda tangan digital merupakan tanda tangan elektronik yang digunakan untuk membuktikan keaslian identitas si pengirim dari suatu pesan atau dokumen. Secara umum, fungsi dari tanda tangan digital berguna untuk mengamankan pesan atau dokumen dari pihak yang tidak berhak atau berwenang, mengamankan data sensitif, dan menguatkan kepercayaan *signer* dan mendeteksi upaya perusakan [2].

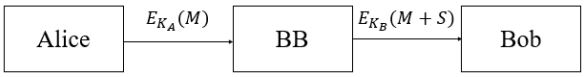
Tanda tangan digital melayani 2 keamanan yang disediakan kriptografi yaitu otentikasi dan juga anti-penyangkalan. Tanda tangan digital memiliki nilai kriptografis yang bergantung pada isi pesan dan kunci yang digunakan. Sebenarnya, tanda tangan digital memiliki karakteristik yang sama dengan tanda tangan pada umumnya. Berikut merupakan beberapa karakteristik dari tanda tangan.

1. Tanda tangan adalah bukti yang otentik
2. Tanda tangan tidak dapat dilupakan
3. Tanda tangan tidak dapat dipindahkan untuk digunakan ulang
4. Dokumen yang telah ditandatangani tidak dapat diubah
5. Tanda tangan tidak dapat disangkal

Pada implementasinya, terdapat dua skema yang dapat digunakan untuk membuat tanda tangan digital yaitu enkripsi dengan menggunakan algoritma kriptografi kunci-simetri dan enkripsi menggunakan algoritma kriptografi kunci-publik.

1. Enkripsi menggunakan algoritma kriptografi kunci-simetri

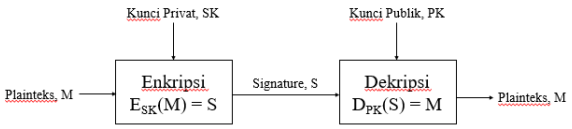
   Pembuatan tanda tangan digital dengan menggunakan algoritma ini sudah memberikan solusi untuk otentikasi pengirim karena kunci simetri hanya diketahui oleh pengirim dan penerima. Namun, algoritma ini tidak dapat menyediakan cara untuk melakukan anti-penyangkalan. Oleh karena itu, untuk mendapatkan layanan anti-penyangkalan, diharuskan pihak ketiga yang dipercaya sebagai penengah (*arbitrase*). Berikut merupakan skema dari cara kerja algoritma ini.



Gambar II.1 Tanda tangan digital kunci simetri
Sumber: Slide Kuliah II4031 Kriptografi dan Koding: Tanda Tangan Digital

2. Enkripsi menggunakan algoritma kriptografi kunci-publik

   Pembuatan tanda tangan digital dengan menggunakan algoritma kunci publik "normal" dengan cara mengenkripsi pesan menggunakan kunci publik penerima pesan dan mendekripsinya dengan kunci privat penerima pesan tidak dapat mengotentikasi si pengirim pesan karena kunci publik diketahui oleh siapapun. Untuk menyelesaikan permasalahan tersebut, maka prosesnya dibalik, yaitu pesan dienkripsi dengan menggunakan kunci privat si pengirim dan pesan didekripsi dengan kunci publik si pengirim. Berikut merupakan skema dari cara kerja algoritma ini [3].



Gambar II.2 Tanda tangan digital kunci publik
Sumber: Slide Kuliah II4031 Kriptografi dan Koding: Tanda Tangan Digital

B. Kriptografi Asimetris (RSA)

RSA merupakan algoritma kunci-publik yang paling terkenal dan paling banyak pengaplikasiannya. Algoritma ini digunakan untuk membangkitkan sepasang kunci publik dan kunci privat yang dirasa cukup aman karena sulitnya memfaktorkan bilangan bulat yang besar menjadi faktor-faktor prima. Algoritma ini ditemukan oleh 3 peneliti dari MIT yaitu Ronald Rivest, Adi Shamir, dan Leonard Adleman pada tahun 1976. Semua orang yang mendapatkan kunci publik daapt menggunakannya untuk mengenkripsi suatu pesan, sedangkan hanya satu orang saja yang memiliki rahasia tertentu dalam hal ini kunci privat untuk melakukan pembongkaran terhadap sandi yang dikirim untuknya.

Berikut beberapa komponen dari algoritma kunci publik RSA.

- p dan q → bilangan prima dan rahasia
- n = p*q → tidak rahasia
- $\phi(n) = (p-1)*(q-1)$ → rahasia
- e (kunci enkripsi) → tidak rahasia
- d (kunci dekripsi) → rahasia
- m (plaintext) → rahasia
- c (cyphertext) → tidak rahasia

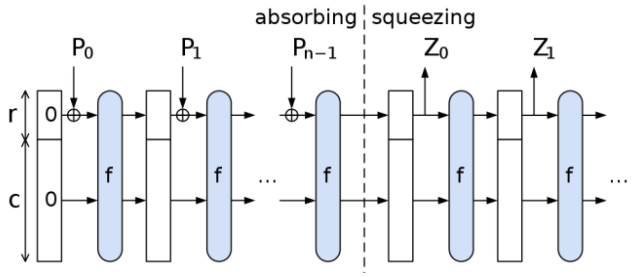
Setelah mendapatkan kunci publik dan kunci privat dari algoritma RSA ini, kunci privat akan digunakan untuk melakukan enkripsi pesan, dan kunci publik akan digunakan untuk mendekripsi pesan [4].

C. Hash SHA-3 (Keccak)

Fungsi *hash* SHA-3 diciptakan pada saat kompetisi NIST yang berakhir pada 2012. Fungsi *hash* ini berhasil mengalahkan kompetitor-kompetitor lainnya seperti BLAKE, Grøstl, JH, dan juga Skein. Fungsi ini diciptakan oleh Guido Breton, Joan Daemen, Micahel Peeters, dan Gilles Van Assche. Keccak menggunakan fungsi non-kompresi untuk menyerap dan kemudian "memeras" *digest*. Secara garis besar, terdapat 3 tahapan dalam proses pembuatan *hash* dengan menggunakan algoritma ini, yaitu fase pra-proses, fase penyerapan (*absorbing*), dan fase pemerasan (*squeezing*).

1. Fase Pra-proses
   Misalkan panjang *digest* yang diinginkan adalah d bit. Maka, pesan M akan ditambah terlebih dahulu dengan bit-bit pengganjal (*padding*) menjadi *string* P sehingga habis dibagi dengan r atau n = length(P)/r. Selanjutnya, P dipotong menjadi blok-blok P_i berukuran r-bit. Kemudian, b-bit dari peubah staus S diinisialisasi menjadi nol dan selanjutnya akan masuk ke fase spons yaitu penyerapan dan pemerasan.
2. Fase Penyerapan
   Untuk setiap blok masukan P_i berukuran r-bit, akan dilakukan XOR dengan r-bit pertama dari *state* S, lalu dimasukkan ke dalam fungsi permutasi f untuk menghasilkan *state* baru S. Bila semua blok masukan selesai diproses, konstruksi spons beralih ke fase pemerasan (*squeezing*).
3. Fase Pemerasan
   Message digest akan disimpan di dalam Z, lalu Z akan diinisialisasi dengan *string* kosong (*null string*). Selama panjang Z belum sama dengan d, r-bit pertama dari *state* S disambungkan ke Z. Jika panjang Z masih belum sama dengan d, masukkan ke dalam fungsi permutasi f menghasilkan *state* baru S.

Secara keseluruhan, proses dari *hashing* dengan menggunakan algoritma SHA-3 atau Keccak ini dapat dilihat pada gambar II.3 berikut [5].



Gambar II.3 *Hashing* SHA-3
Sumber: Slide Kuliah II4031 Kriptografi dan Koding: SHA-3 (Keccak)

III. Rancangan Solusi dan Implementasi

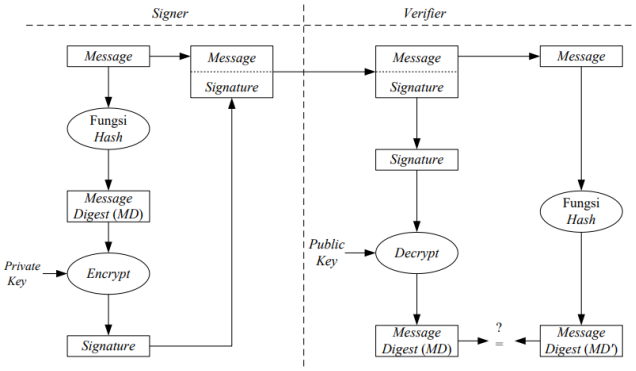
Dalam menyelesaikan masalah mengenai banyaknya pemalsuan dalam pembuatan sertifikat yang didigitisasi, digunakan sistem tanda tangan digital dengan fungsi enkripsi RSA dan *hash* SHA-3 (Keccak). Hal ini dilakukan untuk tetap menjaga prinsip-prinsip yang ada pada kriptografi khususnya adalah otentikasi dan anti penyangkalan.

Dalam dokumen ini, akan dibuat sebuah *prototype* dari aplikasi yang dapat memiliki 3 fungsi utama, yaitu membangkitkan kunci publik dan juga kunci privat dengan menggunakan algoritma RSA, menandatangani sertifikat, dan memverifikasi sertifikat. Format file *digitized certificate* yang disediakan adalah berupa JPEG ataupun PDF.

Untuk mewujudkan berjalannya konsep ini dengan baik dan lancar, maka perlu dilakukan integrasi terlebih dahulu. Maksudnya adalah diperlukan sebuah organisasi terpusat yang dipercaya untuk dapat menandatangani sertifikat. Untuk itu, pada dokumen ini penulis fokuskan untuk diimplementasikan di lingkungan ITB. Hal ini berarti setiap organisasi atau ada kegiatan yang berada di bawah ITB, haruslah meminta ITB untuk menandatangani *digitized certificate* tersebut sebagai bukti pengesahan. Prinsip ini mengadopsi dari implementasi *digital certificate* untuk mengesahkan sertifikat yang dimiliki oleh *website-website* dan ITB berperan sebagai *certification Authority* (CA). ITB menjadi satu-satunya pihak yang mengetahui kunci privat yang dibangkitkan dengan menggunakan algoritma RSA. Setiap kali ITB akan menandatangani sertifikat, ITB sebagai CA akan membangkitkan sepasang kunci dimana kunci publik akan diberikan kepada pemilik sertifikat, sedangkan kunci privat akan digunakan oleh ITB untuk menandatangi sertifikat. Maka dari itu, pada *prototype* ini, akan ada 2 buah aplikasi, yaitu satu aplikasi untuk ITB sebagai CA yang berisikan fungsi untuk membangkitkan kunci dan fungsi untuk menandatangani, sedangkan aplikasi satu lagi akan digunakan oleh pihak ketiga untuk melakukan verifikasi tanda tangan. Secara garis besar, berikut merupakan tahapan dari pelaksanaan sistem yang diajukan.

1. Pihak organisasi mengirimkan satu wakilnya (wajib merupakan mahasiswa ITB dengan NIM yang terdaftar) sebagai penanggungjawab untuk melakukan pengajuan tanda tangan digital kepada pihak ITB. Pengajuan ini tentu harus dilakukan secara sekaligus, tidak boleh hanya satu buah sertifikat untuk mengurangi resiko pemalsuan (kecil kemungkinan organisasi hanya dilakukan oleh satu orang atau pemenang lomba hanya satu orang).
2. Pihak ITB akan mengecek validitas dari pihak yang mengajukan secara manual, mulai dari identitas si penanggungjawab dan juga organisasinya.
3. Pihak ITB akan memasukkan sertifikat satu persatu ke dalam aplikasi penandatanganan. *Output* dari proses ini berupa suatu file .zip yang berisikan 3 file yaitu file sertifikat terkait, kunci publik, dan file tanda tangan.
4. Pihak ITB akan mengirimkan semua file .zip tersebut kepada perwakilan organisasi yang melakukan pengajuan tadi.
5. Pihak organisasi membagikan file .zip tersebut kepada orang yang bersangkutan.
6. Pihak eksternal atau pihak ketiga (termasuk dari pemilik sertifikat) akan melakukan ekstraksi terhadap file .zip, lalu memasukkan file sertifikat, kunci publik, dan tanda tangan ke dalam aplikasi verifikasi.

Secara detail, proses dari penandatanganan dan verifikasi tanda tangan digital dapat dilihat pada gambar III.1 berikut.



Gambar III.1 Pembuatan dan verifikasi tanda tangan digital
Sumber: Slide Kuliah II4031 Kriptografi dan Koding: Tanda Tangan Digital

A. *Hashing*

Tahap pertama pada implementasi tanda tangan digital adalah melakukan *hashing* sertifikat tersebut dengan menggunakan SHA-3. Sama seperti apa yang tercantum pada dasar teori, berikut merupakan langkah-langkah pada proses *hashing*.

1. Menambahkan pesan M dengan bit-bit pengganjal (*padding*) menjadi *string* P hingga habis dibagi dengan r.
2. Memotong P menjadi blok-blok P_i berukuran r-bit.
3. Menginisialisasi b-bit dari *state* S dengan nol.
4. Melakukan XOR blok masukan P_i dengan r-bit pertama dari *state* S untuk menghasilkan *state* baru S (lakukan hingga semua blok telah diproses).
5. Inisialisai Z (tempat penyimpanan *message digest*) dengan *string* kosong.
6. Selama panjang Z belum sama dengan d, r-bit pertama dari *state* S di-*append* ke Z.

7. Jika panjang Z masih belum sama dengan d, masukkan ke dalam fungsi permutasi f menghasilkan *state* baru S.

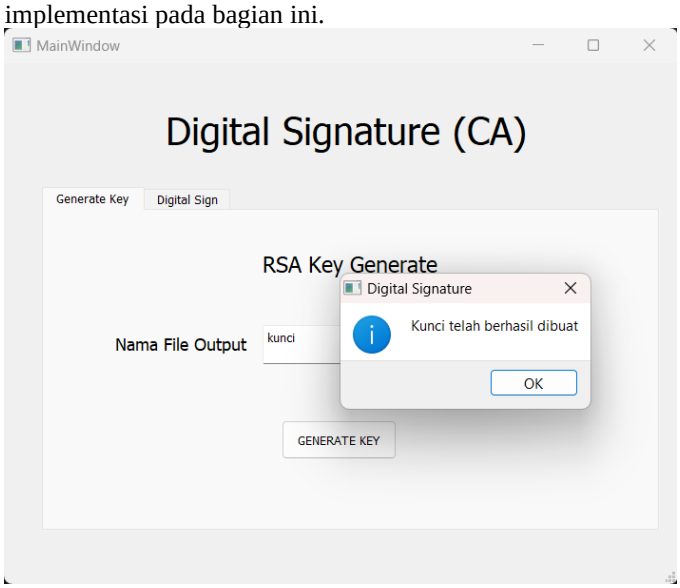
Pada dokumen kali ini, implementasi dari fungsi *hashing* dilakukan dengan memanfaatkan algoritma *hashing* SHA-3 224 yang telah tersedia pada *library hashlib*. Hasil dari implementasi bagian ini dapat dilihat pada gambar III.2 di bawah (sebagai catatan, hasil dari *hashing* tidak dikirimkan ke *user*).

```
Berikut merupakan hasil hash dari file tersebut sebelum dilakukan enkripsi:
2041e146e4370b303b7ecfd92a13727077adf91e6b103b10573d5517
```

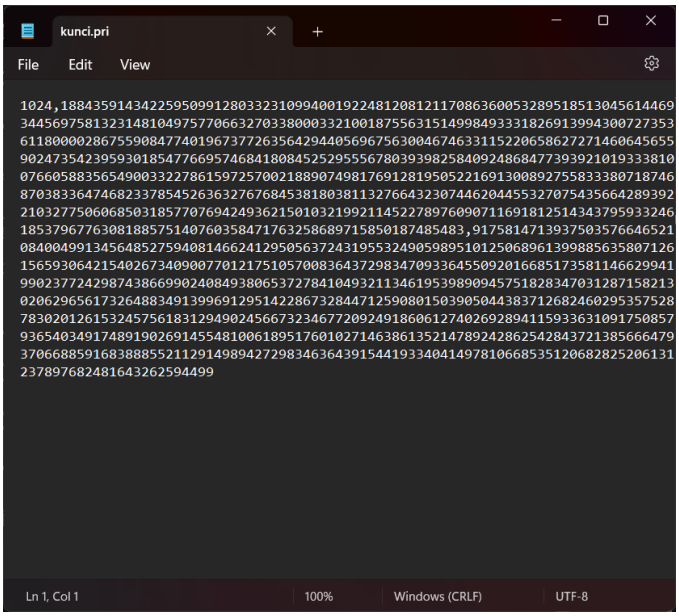
Gambar III.2 Pembuatan dan verifikasi tanda tangan digital

B. *Pembangkitan Kunci Asimetri*

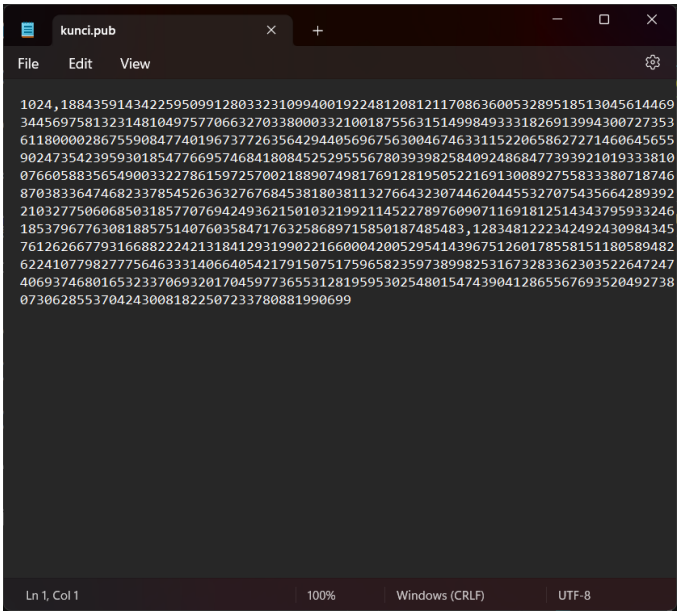
Tahapan pembangkitan kunci ini dilakukan dengan inisialisasi nilai p, q, dan e dimana p dan q merupakan bilangan prima sedangkat e merupakan nilai yang relatif prima dengan ϕ(n) atau merupakan hasil dari (p-1)*(q-1). Selanjutnya, hitungan nilai n yaitu merupakan hasil kali dari p dan q. Langkah terakhir adalah menentukan nilai dari d yang didapat dari $e^{-1}(\text{mod}(\phi(n)))$. Untuk kunci publik, akan dikirimkan ke penerima sertifikat berupa file .pub yang isinya berupa tuple yaitu (1024, e, n) dan untuk kunci privat akan dikirimkan ke pihak ITB berupa file .pri yang isinya berupa tuple yaitu (1024, d, n). Pada implementasi kali ini, panjang *key* adalah 1024 bit. Berikut merupakan hasil dari implementasi pada bagian ini.



Gambar III.3 Aplikasi *generate key*



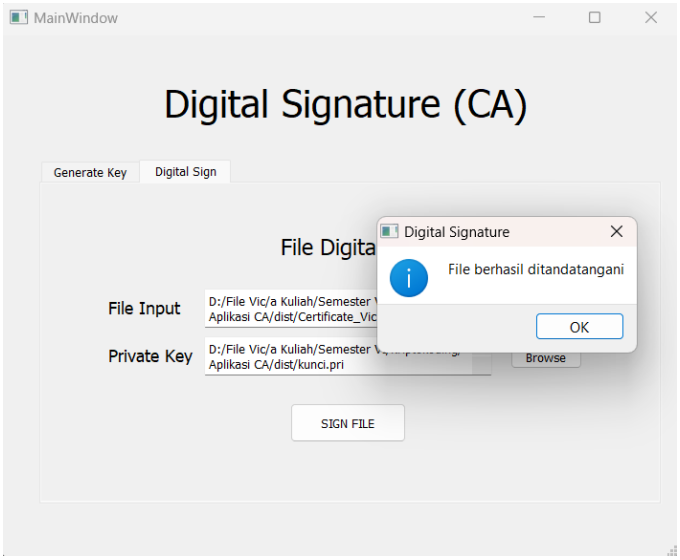
Gambar III.4 Kunci privat



Gambar III.5 Kunci publik

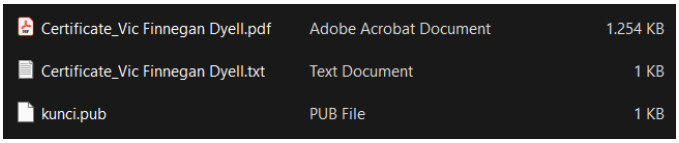
C. *Penandatanganan Dokumen*

Pada tahapan ini, akan digunakan nilai *hash* yang telah didapat dari tahap sebelumnya, lalu akan dienkripsi dengan menggunakan kunci privat. Hasil dari enkripsi ini disebut dengan tanda tangan digital atau *digital signature*. Pada implementasi kali ini, ketika suatu sertifikat ditandatangani, *output*-nya berupa 1 file .zip yang berisikan *digitized certificate*, kunci publik, dan file yang berisikan tandatangan. *Digitized certificate* diletakkan di *directory* yang sama dengan letak aplikasi CA dan tidak akan berubah sama sekali, serta tandatangannya bukan di-*embed* melainkan ditulis pada file terpisah. Berikut hasil dari implementasi pada bagian ini.

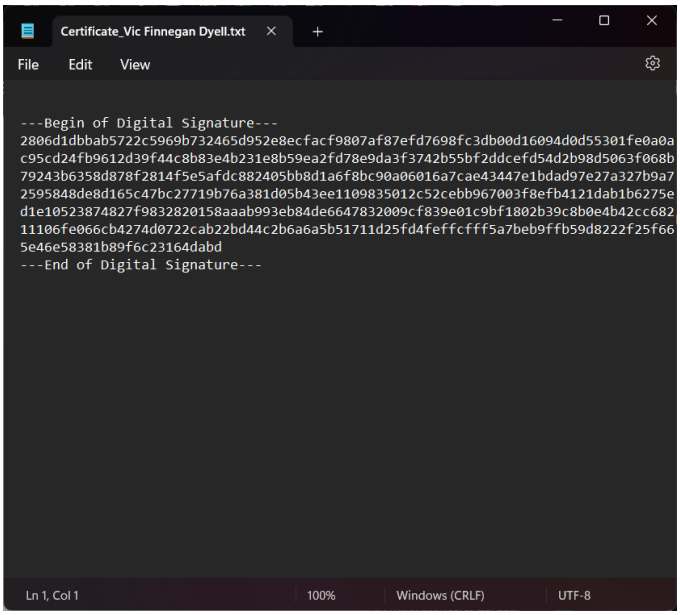


Gambar III.6 Aplikasi tanda tangan digital

Gambar III.7 File .zip hasil tanda tangan digital



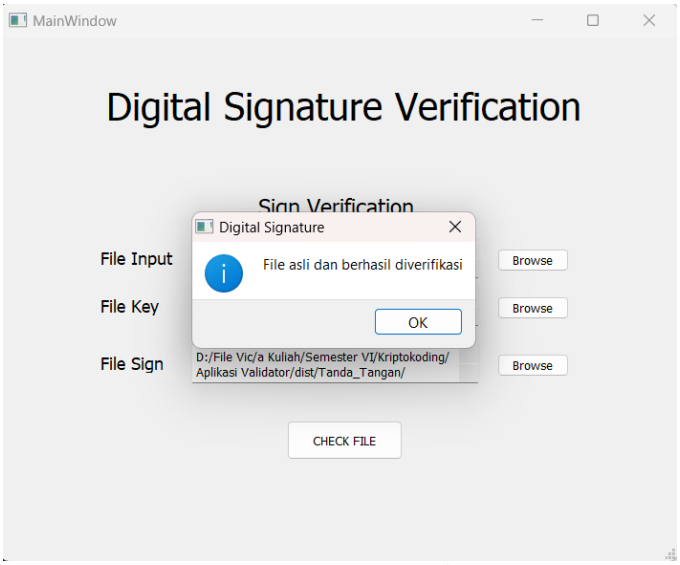
Gambar III.8 Isi file .zip



Gambar III.9 Tanda tangan digital

D. Verifikasi Dokumen

Tahapan ini menerima input berupa hasil ekstraksi dari .zip yang dihasilkan tadi. Pada aplikasi, akan diminta untuk memasukkan 3 file yaitu file sertifikat, tandatangan, dan kunci publik. *Output* dari tahap ini berupa status dari verifikasi apakah sertifikat asli atau tidak. Hasil implementasi pada bagian ini dapat dilihat pada gambar III.10 di bawah.



Gambar III.10 Aplikasi verifikasi

IV. KESIMPULAN

Berdasarkan hasil pengerjaan yang telah dilakukan, didapatkan kesimpulan bahwa penggunaan tanda tangan digital dapat dilakukan untuk melakukan verifikasi suatu sertifikat yang didigitisasi terkhususnya sertifikat seperti perlombaan, kepanitiaan, ataupun organisasi. Pengembangan pada makalah ini dicontohkan untuk suatu institusi yaitu ITB yang menjadi CA. Pengembangan perlu dilakukan lebih lanjut dikarenakan aplikasi ini masih sangat sederhana dan masih cukup terbatas, seperti letak dari file yang akan ditandatangani harus satu *directory* dengan aplikasinya serta tampilan GUI yang masih sangat sederhana dan kurang menarik.

Kedepannya solusi ini diharapkan dapat dikembangkan lebih lanjut terkhusus untuk pada bagian fleksibilitas dokumen dan GUInya supaya lebih nyaman digunakan. Selain itu, implementasi dari ide ini juga diperluas supaya tidak banyak lagi sertifikat-sertifikat palsu yang beredar hanya sekadar memenuhi CV saja.

SOURCE CODE

https://github.com/VicDyell09/Paper-Kriptokoding

VIDEO LINK AT YOUTUBE

https://youtu.be/HWb40kB6b3M

REFERENCES

[1] Pura Group. 2020. Total Security System. Retrieved May 2023, from https://www.puragroup.com/id/total-security-system-20/

[2] Diskominfo dan Statistik Kabupaten Ponorogo. 2019. Tanda Tangan Elektronik vs Tanda Tangan Digital. Retrieved May 2023, from https://kominfo.ponorogo.go.id/tanda-tangan-elektronik-vs-tanda-tangan-digital/

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

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

[5] Munir, Rinaldi. 2023. Slide Kuliah II4031 Kriptografi dan Koding: SHA-3 (Keccak)

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

Vic Finnegan Dyell (18220019)

Appendix

**Disclaimer*: *source code* diadaptasi dari *source code* yang telah dibuat pada tugas ketiga kelas Kriptografi dan Koding.

1. Baca_File.py

```
import pathlib

# fungsi read file
def readkey(filename):
    with open(filename, "r") as f:
        text = f.read().split(",")
    return(text)

# fungsi read file
def readfile(filename):
    with open(filename, "r") as f:
        text = f.read()
    return(text)

# fungsi read file in bytes
def readfilebin(filename):
    bytes=[]
    with open(filename, "rb") as f:
        while True:
            b = f.read(1)
            if not b:
                break
            bytes.append(int.from_bytes(b,
byteorder="big"))
    text =""
    for i in range(len(bytes)):
        text+=chr(bytes[i])
    return(text)

#funsi write file txt
def writefile(text,filename):
    with open('%s.txt' % (pathlib.Path(filename).stem),
'w') as f:
        f.write(text)
    f.close()

#fungsi return file extention
def fileext(filename):
    return pathlib.Path(filename).suffix

def name(filename):
    return pathlib.Path(filename).stem

def appendfile(text, filename):
    with open('%s.txt' % (pathlib.Path(filename).stem),
"a") as f:
        f.write(text)
    f.close()
```

2. Pembangkitan_Kunci.py
3.

```
import random
import os
import sys

def rabinMiller(num):
   s = num - 1
   t = 0

   while s % 2 == 0:
      s = s // 2
      t += 1
   for trials in range(5):
      a = random.randrange(2, num - 1)
      v = pow(a, s, num)
      if v != 1:
         i = 0
         while v != (num - 1):
            if i == t - 1:
               return False
            else:
               i = i + 1
               v = (v ** 2) % num
      return True

def isPrime(num):
   if (num < 2):
      return False
   lowPrimes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
37, 41, 43, 47, 53, 59, 61,
   67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113,
127, 131, 137, 139, 149, 151,
   157, 163, 167, 173, 179, 181, 191, 193, 197, 199,
211, 223, 227, 229, 233, 239, 241,
   251, 257, 263, 269, 271, 277, 281, 283, 293, 307,
311, 313,317, 331, 337, 347, 349,
   353, 359, 367, 373, 379, 383, 389, 397, 401, 409,
419, 421, 431, 433, 439, 443, 449,
   457, 461, 463, 467, 479, 487, 491, 499, 503, 509,
521, 523, 541, 547, 557, 563, 569,
   571, 577, 587, 593, 599, 601, 607, 613, 617, 619,
631, 641, 643, 647, 653, 659, 661,
   673, 677, 683, 691, 701, 709, 719, 727, 733, 739,
743, 751, 757, 761, 769, 773, 787,
   797, 809, 811, 821, 823, 827, 829, 839, 853, 857,
859, 863, 877, 881, 883, 887, 907,
   911, 919, 929, 937, 941, 947, 953, 967, 971, 977,
983, 991, 997]

   if num in lowPrimes:
      return True
   for prime in lowPrimes:
      if (num % prime == 0):
         return False
   return rabinMiller(num)

def generateLargePrime(keysize = 1024):
   while True:
      num = random.randrange(2**(keysize-1),
2**(keysize))
      if isPrime(num):
         return num

def generateKeys(keysize=1024):
    e = d = N = 0

    p = generateLargePrime(keysize)
    q = generateLargePrime(keysize)

    # print("p :", p)
    # print("q :", q)

    N = p*q
    phiN = (p - 1) * (q - 1)

    while True:
        e = random.randrange(2 ** (keysize - 1), 2 **
keysize - 1)
        if (isCoPrime(e,phiN)):
```

```
            break

    d = modilarInv(e,phiN)

    return e, d, N

def generateLargePrime(keysize):
    """
        return bilangan prima dengan jumlah bit sebesar
keysize
    """

    while True:
        num = random.randrange(2 ** (keysize - 1), 2 **
keysize - 1)
        if (isPrime(num)):
            return num

def isCoPrime(p, q):
    """
        return true kalau gcd(p,q) = 1
        relatif prima
    """
    return gcd(p, q) == 1

def gcd(p, q):
    """
        euclidean algoritm → mencari fpb p dan q
    """
    while q:
        p, q = q, p % q
    return p

def egcd(a,b):
    s = 0; old_s = 1
    t = 1; old_t = 0
    r = b; old_r = a

    while r != 0:
        quotient = old_r // r
        old_r, r = r, old_r - quotient * r
        old_s, s = s, old_s - quotient * s
        old_t, t = t, old_t - quotient * t

    #return gcd, x, y
    return old_r, old_s, old_t


def modilarInv(a,b):
    gcd, x ,y = egcd(a,b)

    if x <0:
        x += b

    return x

def enkripsi(e, N, text):
    cipher = ""

    for c in text:
        m = ord(c)
        cipher += str(pow(m, e, N)) + " "

    return cipher

def dekripsi(d, N, cipher):
    text = ""

    parts = cipher.split()
    for part in parts:
        if part:
            c = int(part)
            text += chr(pow(c, d, N))

    return text

def dekripsihex(e, N, text):
    cipher = ""
    m = ""

    for c in text:
        m += str(ord(c))

        cipher = hex(pow(int(m), e,
N)).replace("0x","")+" "
    # print (hex(int(m)))
    return cipher

def enkripsihex(d, N, cipher):
    text = ""
    parts = cipher.split()
    for part in parts:
        if part:
            c = int(part,16)
            text += hex(pow(c, d, N))
    # print (f"c : {c}")
    return text

def writeKey(filename):
    keySize = 1024
    e, d, N = generateKeys(keySize)
    if os.path.exists('%s.pub' % (filename)) or
os.path.exists('%s.pri' % (filename)):
        sys.exit('WARNING: The file %s.pub or %s,.pri
already exists! Use a different name or delete these
files and re-run this program.' % (filename, filename))

    fo = open('%s.pub' % (filename), 'w')
    fo.write('%s,%s,%s' % (keySize, N, e))
    fo.close()

    fo = open('%s.pri' % (filename), 'w')
    fo.write('%s,%s,%s' % (keySize, N, d))
    fo.close()
```

4. Tanda_Tangan.py

```
import hashlib
import Baca_File
import Pembangkitan_Kunci
import zipfile


def hashText(text):
    hash = hashlib.new("sha3_224", text.encode())
    return hash.hexdigest()

def generateDigitalSigned(filename, privatekey):
    if Baca_File.fileext(filename) == ".txt":
        text = Baca_File.readfile(filename)
    else:
        text = Baca_File.readfilebin(filename)
    digest = hashText(text)
    key = Baca_File.readkey(privatekey)
    d = int(key[2])
    N = int(key[1])
    signature = Pembangkitan_Kunci.dekripsihex(d, N,
digest) #enkripsi dengan private key
    new_signature = "\n---Begin of Digital Signature---
\n"+signature+"\n---End of Digital Signature---"
```

```
    if Baca_File.fileext(filename) == ".txt":
        Baca_File.appendfile(new_signature,filename)
    else:
        Baca_File.writefile(new_signature,filename)
        myzip = zipfile.ZipFile('Tanda_Tangan.zip',
'w')
        myzip.write('%s.txt' %
(Baca_File.name(filename)))
        myzip.write('%s.pub' %
(Baca_File.name(privatekey)))
        myzip.write('%s.pdf' %
(Baca_File.name(filename)))


def validateDigitalSigned(filename, publickey,
filesig=""):
    key = Baca_File.readkey(publickey)
    e = int(key[2])
    N = int(key[1])
    text = Baca_File.readfilebin(filename)
    digest = hashText(text)
    isisig = Baca_File.readfile(filesig).split("---
Begin of Digital Signature---\n")
    plaintext = isisig[1].split("\n---End of Digital
Signature---")
    signature = plaintext[0]
    signatureDigest =
Pembangkitan_Kunci.enkripsihex(e,N,signature)
    digests = ""
    for c in digest:
        digests += str(ord(c))
    hexdigest = hex(int(digests))
    if hexdigest == signatureDigest:
        return("Valid")
    else:
        return("Tidak Valid")
```

5. MainCA.py (diluar GUI)
   Jhn\

```
def clicker(self):
        fname = QFileDialog.getOpenFileName(
            parent = MainWindow,
            caption = "Open File",
            directory = "",
            filter = "All Files (*)")
        if fname:
            return fname[0]

    def genkey(self):
        filename = self.plainTextEdit.toPlainText()
        writeKey(filename)
        self.popupnotifbiasa("Kunci telah berhasil
dibuat")

    def gensign(self):
        filename = self.plainTextEdit_2.toPlainText()
        privatekey = self.plainTextEdit_3.toPlainText()
        if filename == "" or privatekey == "":
            self.popupnotiferror("File dan private key
tidak boleh kosong!")
        elif Baca_File.fileext(filename)!=".pdf":
            self.popupnotiferror("File harus .pdf")
        else:
            if self.prikeychecker(privatekey)==True:
                generateDigitalSigned(filename,
privatekey)
                self.popupnotifbiasa("File berhasil
ditandatangani")
            else:
                self.popupnotiferror("Masukkan kunci
private (dengan file .pri)!")

    def browsefile(self):
        link = self.clicker()
        self.plainTextEdit_2.setPlainText(link)

    def browseprivate(self):
        link = self.clicker()
        self.plainTextEdit_3.setPlainText(link)


    def popupnotifbiasa(self, kalimat):
        msg = QMessageBox()
        msg.setWindowTitle("Digital Signature")
        msg.setText(kalimat)
        msg.setIcon(QMessageBox.Information)

        x = msg.exec_()

    def popupnotifwarning(self, kalimat):
        msg = QMessageBox()
        msg.setWindowTitle("Digital Signature")
        msg.setText(kalimat)
        msg.setIcon(QMessageBox.Warning)

        x = msg.exec_()

    def popupnotiferror(self, kalimat):
        msg = QMessageBox()
        msg.setWindowTitle("Digital Signature")
        msg.setText(kalimat)
        msg.setIcon(QMessageBox.Critical)

        x = msg.exec_()

    def pubkeychecker(self, file):
        if Baca_File.fileext(file) == ".pub":
            return True
        else:
            return False

    def prikeychecker(self, file):
        if Baca_File.fileext(file) == ".pri":
            return True
        else:
            return False
```

6. MainValidator.py (diluar GUI)
   Sd

```
def clicker(self):
        fname = QFileDialog.getOpenFileName(
            parent = MainWindow,
            caption = "Open File",
            directory = "",
            filter = "All Files (*)")
        if fname:
            return fname[0]


    def browsefileverif(self):
        link = self.clicker()
        self.plainTextEdit_5.setPlainText(link)

    def browsepubkey(self):
        link = self.clicker()
        self.plainTextEdit_4.setPlainText(link)

    def browsesign(self):
        link = self.clicker()
```

```
        self.plainTextEdit_6.setPlainText(link)

    def signverif(self):
        filename = self.plainTextEdit_5.toPlainText()
        filekey = self.plainTextEdit_4.toPlainText()
        filesign = self.plainTextEdit_6.toPlainText()

        if filename=="" or filekey="":
            self.popupnotiferror("File name dan file
key tidak boleh kosong!")

        else:
            if Baca_File.fileext(filename) == ".txt":
                if filesign != "":
                    self.popupnotiferror("Kosongkan
file sign! \n.txt tidak perlu filesign")
                else:
                    if self.pubkeychecker(filekey) ==
True:
                        hasil =
validateDigitalSigned(filename, filekey, filesign)
                        if hasil == "Valid":
                            self.popupnotifbiasa("File
asli dan berhasil diverifikasi")
                        elif hasil == "Tidak Valid":

self.popupnotifwarning("Verifikasi gagal! \n File asli
sudah berubah atau Anda salah memasukkan kunci")
                    else:
                        self.popupnotiferror("Masukkan
kunci publik (dengan file .pub)!")

            else:
                if filesign == "":
                    self.popupnotiferror("Masukkan file
sign!")
                else:
                    if self.pubkeychecker(filekey) ==
True:
                        hasil =
validateDigitalSigned(filename, filekey, filesign)
                        if hasil == "Valid":
                            self.popupnotifbiasa("File
asli dan berhasil diverifikasi")
                        elif hasil == "Tidak Valid":

self.popupnotifwarning("Verifikasi gagal! \n File asli
sudah berubah atau Anda salah memasukkan kunci")
                    else:
                        self.popupnotiferror("Masukkan
kunci publik (dengan file .pub)!")

    def popupnotifbiasa(self, kalimat):
        msg = QMessageBox()
        msg.setWindowTitle("Digital Signature")
        msg.setText(kalimat)
        msg.setIcon(QMessageBox.Information)

        x = msg.exec_()

    def popupnotifwarning(self, kalimat):
        msg = QMessageBox()
        msg.setWindowTitle("Digital Signature")
        msg.setText(kalimat)
        msg.setIcon(QMessageBox.Warning)

        x = msg.exec_()

    def popupnotiferror(self, kalimat):
        msg = QMessageBox()
        msg.setWindowTitle("Digital Signature")
        msg.setText(kalimat)
        msg.setIcon(QMessageBox.Critical)

        x = msg.exec_()

    def pubkeychecker(self, file):
        if Baca_File.fileext(file) == ".pub":
            return True
        else:
            return False

    def prikeychecker(self, file):
        if Baca_File.fileext(file) == ".pri":
            return True
        else:
            return False
```