Implementasi SHA-3 dan RSA pada Tanda Tangan Digital Dalam Pakta Integritas Ujian Online

Aldi Fadlian Sunan 18220086 (*Author*)
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
aldifadlian7@gmail.com

***Abstract*—Identitas keaslian atas suatu pernyataan atau pakta integritas dapat dinyatakan dengan tanda tangan, termasuk pada pernyataan keaslian di ujian pendidikan. Perkembangan teknologi menyebabkan adanya transformasi ke dunia digital, sehingga pelaksanaannya pun perlu diadaptasi secara *online*. Adaptasi ke dunia digital pada tanda tangan dapat dilakukan dengan memanfaatkan kriptografi berupa algoritma kunci publik RSA dan fungsi Hash. RSA dan Hash digunakan untuk menghasilkan nilai keunikan pada output tanda tangan. Dengan begitu, pernyataan pada ujian *online* dapat lebih dipastikan keasliannya dengan adanya tanda tangan dari peserta.**

Keywords—Tanda Tangan; Ujian; RSA; Hash

I. Pendahuluan

Perkembangan teknologi menjadi salah satu hal yang berpengaruh ke aspek kehidupan. Terutama, saat Pandemi Covid-19 melanda, banyak kegiatan yang dialihkan ke dalam jaringan (daring) termasuk dalam ujian. Adaptasi tersebut terus dilaksanakan hingga sekarang karena ujian *online* dianggap lebih efektif dan efisien untuk dilaksanakan. Institut Teknologi Bandung menjadi salah satu tempat yang masih menerapkan konsep ujian *online*, baik melalui *Google Form* ataupun platform Edunex.

Akan tetapi, pelaksanaan integritas dari ujian *online* masih harus dipertimbangkan. Sering kali juga terjadi penyangkalan dalam pelaksanaan ujian. Jika dibandingkan dengan ujian *offline* dimana pernyataan keaslian atau pakta integritas pengerjaan peserta dapat dinyatakan langsung melalui tanda tangan tertulis, pada ujian *online* pernyataan keaslian hanya menggunakan teks dan *button.* Hal tersebut menjadi suatu permasalahan karena tidak dapat dijadikan bukti bahwa pengerjaan ujian benar-benar dilakukan oleh orang tersebut.

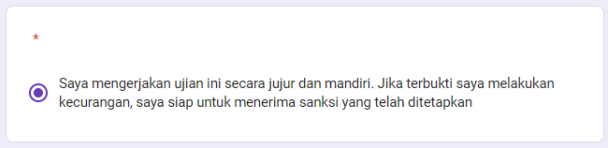
Permasalahan tersebut bisa diselesaikan dengan menggunakan tanda tangan secara digital. Tanda tangan digital ini dapat menjaga *non-repudiation* untuk menjamin keaslian dari ujian yang dikerjakan oleh peserta. Usulan tanda tangan digital akan menggunakan teknologi berbasis algoritma kunci publik RSA dan fungsi Hash SHA-3. Proses penandatanganan akan menggantikan opsi *field* keterangan pernyataan di awal ujian. Proses validasi dari tanda tangan akan dilakukan pada sistem secara langsung tanpa ke server. Harapannya dengan digunakannya tanda tangan digital ini, keaslian dari ujian yang dikerjakan akan lebih terpercaya.

II. Dasar Teori

A. *Google Form*

Google Form merupakan salah satu produk dari Google yang dapat digunakan untuk melakukan survei ataupun pengumpulan respon. Melalui Google Form, dapat pula membuka metode untuk pengumpulan jawaban pada ujian menggunakan fitur-fitur yang ada. Seperti contohnya dapat membuat berbagai variasi tipe pertanyaan mulai dari text, multiple choices, hingga check boxes. [1]

Akan tetapi, pengisian data di Google Form masih bisa disangkal karena belum tersedianya fitur *signature* di dalamnya. Terutama untuk data yang berkaitan dengan suatu pernyataan tertentu, fitur yang dapat digunakan oleh Google Form hanya sebatas *radio button* ataupun *text* yang tidak dapat dijamin keasliannya.



Gambar 1. Ilustrasi Pernyataan pada Ujian Online (sumber: pribadi)

Hal tersebut berarti tidak terdapat cara untuk mengumpulkan data *signature* secara *default* di Google Forms, sehingga harus mencari cara lain untuk mengumpulkan *signature* yang diperlukan atas pernyataan tertentu. Pada Google Form, bisa saja mengetikkan nama atau inisial tertentu secara manual sebagai identitas. Meskipun demikian, hal tersebut tidak bisa mengkonfirmasi keaslian dari pengisian data dan dapat dengan mudah untuk disangkal. Cara lain yang mungkin dapat dilakukan ialah, dengan memanfaatkan fitur *upload files* untuk memasukkan tanda tangan digital atas pernyataan tertentu menggunakan pemanfaatan kriptografi.

B. *Kriptografi*

Kata kriptografi berasal dari bahasa Yunani, *cryptós* dan *gráphein* yang berarti "tulisan rahasia". Dengan begitu, kriptografi merupakan suatu teknik untuk menjaga kerahasiaan dari informasi suatu pesan. [2] Aman dalam kriptografi tersebut mencakup terjaga kerahasiaannya (*confidentiality*), keasliannya (*data integrity*), keyakinannya (*authentication*), dan anti penyangkalan (*non repudiation*).

Kriptografi akan menjaga hal-hal tersebut terhadap pesan yang akan dikirim dan diterima. Terdapat proses dalam pelaksanaannya berupa enkripsi, dimana pesan (berupa *plaintext*) akan diubah menjadi *ciphertext*, dan dekripsi, dimana *ciphertext* diubah kembali menjadi pesan yang dapat dibaca (*plaintext*). Proses tersebut akan menggunakan kunci yang didapatkan dari algoritma tertentu dalam implementasinya. Dengan demikian, penggunaan kriptografi akan meningkatkan tingkat keamanan dalam menjaga informasi pada pesan.

C. *Algoritma Kunci Publik RSA*

Algoritma kunci publik RSA merupakan algoritma yang ditemukan oleh Ronald Rivest, Adi Shamir, dan Len Adleman. Algoritma ini menggunakan pemfaktoran pada suatu bilangan bulat dalam prosedurnya. Pada penggunaannya, algoritma RSA memanfaatkan kunci publik dan kunci privat, dimana kunci publik dapat diketahui oleh siapa saja tetapi kunci privat hanya diketahui oleh pemilik kunci. Berikut merupakan prosedur yang dilakukan pada algoritma kunci publik RSA:

1. Pilih dua bilangan prima, p dan q, dimana nilai p tidak sama dengan nilai q. Pemilihan bilangan dapat dilakukan secara acak dengan nilai yang besar dan bersifat rahasia.
2. Kedua bilangan p dan q dikalikan untuk mendapatkan nilai n (modulus) dengan rumus
$$n = p*q$$
3. Hitung nilai $\phi(n)$ menggunakan rumus
$$\phi(n) = (p\text{-}1)(q\text{-}1)$$
4. Pilih kunci publik e yang berupa bilangan relatif prima dengan nilai $\phi(n)$
5. Hitung kunci privat d menggunakan rumus
$$ed \equiv 1 \ (mod\ \phi(n)) \text{ atau } \ d \equiv e^{-1} \ (mod\ \phi(n))$$

Prosedur dari algoritma RSA tersebut menghasilkan kunci publik berupa nilai (e,n) dan kunci privat berupa nilai (d,n) [3].

Kunci-kunci tersebut digunakan untuk proses enkripsi dengan proses sebagai berikut:

1. Pesan yang berukuran besar dipisahkan menjadi blok plainteks $m_1, m_2, m_3 \ldots$ dengan $0 \leq m_i < \text{n-1}$
2. Cipherteks c_i dihitung menggunakan kunci publik e dengan rumus
$$c_i = m_i^{\ e} \ mod\ n$$

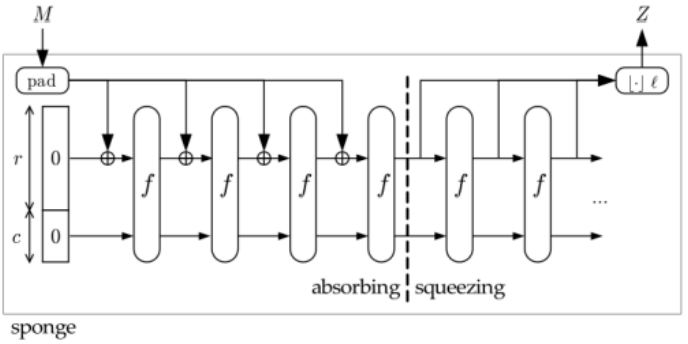
Sedangkan untuk proses dekripsi, proses yang dilakukan sebagai berikut:

1. Cipherteks dipisahkan menjadi c_1, c_2, c_3
2. Blok plainteks m_i dari blok cipherteks c_i dihitung kembali dengan kunci privat d dengan rumus
$$m_i = c_i^{\ d} \ mod\ n$$

Dari prosedur tersebut, algoritma RSA menjadi algoritma yang efektif untuk diimplementasikan. Jika semakin besar nilai p dan q, maka tingkat keamanan dari RSA akan semakin tinggi. Hal tersebut karena akan sulit untuk melakukan pemfaktoran pada dua bilangan acak prima tersebut. [4]

D. *Fungsi Hash SHA-3*

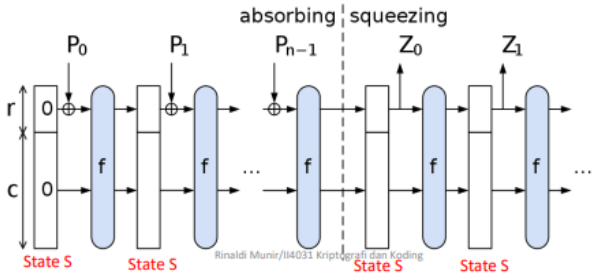
Fungsi hash SHA-3 (Keccak) merupakan salah satu fungsi *secure hash algorithm* untuk menghasilkan nilai hash yang unik dari pesan tertent. Fungsi ini menggunakan fungsi nonkompresi untuk memproses nilai *digest*.



Gambar 2. Ilustrasi SHA-3 [5]

SHA-3 memiliki fungsi konstruksi spons yang diawali dengan mengubah pesan M dengan menambahkan bit-bit *padding* menjadi suatu string P yang habis dibagi dengan r atau $n = length(P)/r$. String P tersebut dipecah menjadi blok P_i seukuran r-bit. *State S* dengan b-bit tersebut nantinya akan diinisiliasi menjadi 0 untuk memasuki tahapan *absorbing* dan *squeezing*. [5]

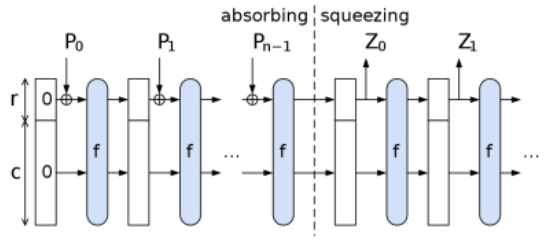
1. *Absorbing*



Gambar 3. Ilustrasi fase absorbing [5]

Pada fase *absorbing* atau penyerapan, terjadi proses XOR. Prosedur tersebut terjadi antara setiap masukan P_i berukuran r-bit dengan r-bit pertama dari state S. Hasil dari proses *absorbing* ini nantinya akan dimasukkan ke dalam fungsi permutasi f untuk state S yang baru.

2. *Squeezing*



Gambar 4. Ilustrasi fase squeezing [5]

Proses dilanjutkan ke *squeezing* dimana *message digest* disimpan ke dalam Z yang diinisialisasi dengan string kosong. Lalu, selama nilai Z belum sepanjang d, *r*-bit pertama dari *state S* akan ditambahkan ke Z. Penambahan (*append*) tersebut terus dilakukan, dan jika masih belum sama dengan *d*, Z dimasukkan ke fungsi permutasi *f* untuk state baru *S.*

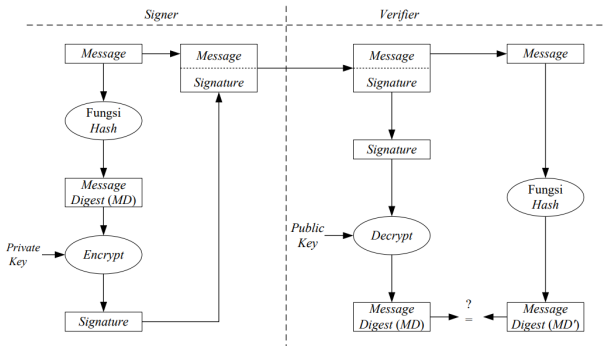
E. Tanda Tangan

Tanda tangan menjadi bentuk identitas seseorang atas suatu keperluan maupun perjanjian yang dapat menjadi bukti persetujuan. Tanda tangan memiliki atribut yang berbeda sehingga dapat dianggap unik untuk tiap individu. Tanda tangan biasanya dibuat secara tertulis dengan tinta di atas kertas dengan bentuknya yang unik dan beragam. Termasuk, pada kertas ujian untuk menyatakan pernyataan keaslian atau pakta integritas atas kejujuran peserta ujian

Gambar 4. Ilustrasi Tanda Tangan [6]

Namun, seiring berkembangnya teknologi, penggunaan tanda tangan mulai beralih ke tanda tangan digital. Keunikan pada tangan tangan digital, dilihat dari nilai kunci yang beragam untuk tiap tanda tangan di tiap dokumen. Tanda tangan digital dapat berupa nilai kriptografis yang bergantung ke kunci yang digunakan untuk tiap pesan. Dengan demikian, jika terjadi isi pesan berubah meskipun hanya pada satu karakter, tanda tangan digital yang digunakan pun akan berbeda nilainya. [6]

Prosedur tanda tangan digital terdiri dari 2 tahap, *signing*, dimana pesan ditanda-tangani, dan *verification,* dimana tanda tangan yang digunakan akan dicek nilainya. Prosedur tersebut dapat dinyatakan pada diagram berikut:



Gambar 5. Ilustrasi Prosedur Tanda Tangan Digital [7]

Prosedur tanda tangan menggunakan algoritma kriptografi kunci publik saat proses enkripsi dan dekripsinya. Dengan menggunakan algoritma kunci publik, seperti RSA, akan menghasilkan pasangan kunci privat dan kunci publik yang dapat digunakan.

Proses *signing* diawali dengan menghasilkan nilai *message digest* (MD) dari pesan melalui fungsi hash. Kemudian, dilakukan enkripsi dari MD menggunakan *private key* untuk menghasilkan tanda tangan pada *message.* Selanjutnya, untuk memverifikasi keaslian dari tanda tangan, dilakukan proses *verification* dengan memisahkan pesan dengan tanda tangan. Pesan dimasukkan ke fungsi hash untuk menghasilkan *message digest*. Sedangkan tanda tangan akan didekripsi dengan *public key* untuk menghasilkan *message digest* juga. Kemudian, kedua *message digest* akan dibandingkan. Jika keduanya senilai, maka tanda tangan telah terverifikasi dengan pesan yang tidak diubah. [7]

III. RANCANGAN

Sistem yang akan digunakan akan memanfaatkan Google Form sebagai sistem utama pelaksanaan ujian, serta sistem pembangkitan dan verifikasi tanda tangan. Ketiga sistem tersebut digunakan pada satu perangkat yang sama di peserta.

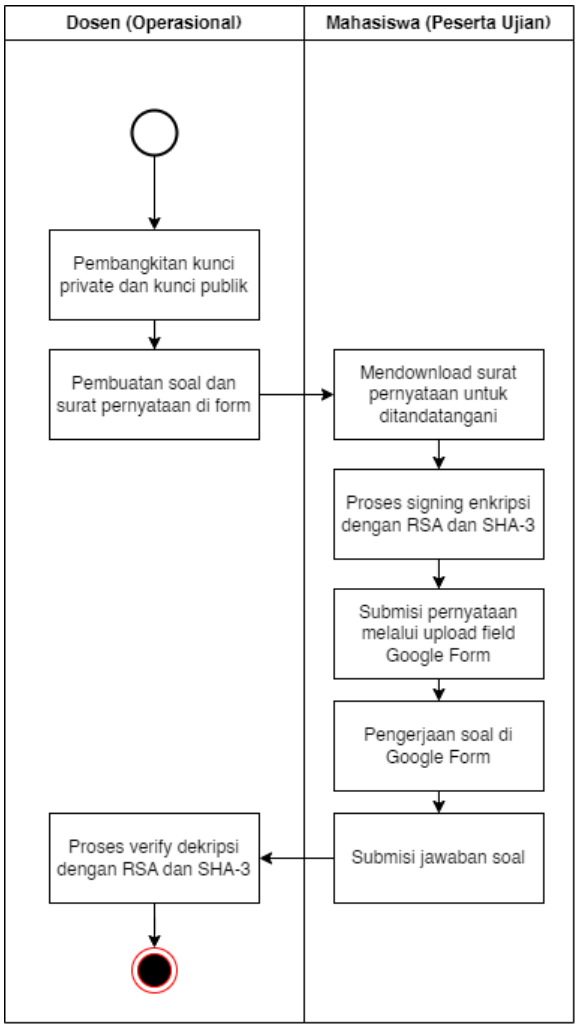
A. Format Pesan

Pesan pernyataan pakta integritas disimpan ke dalam file berbentuk PDF untuk ditandatangiani oleh peserta sebelum ujian dimulai. Berikut merupakan isi dari file pesan tersebut:

> "Saya mengerjakan ujian ini dengan jujur dan mandiri tanpa bantuan pihak manapun. Jika suatu saat saya terbukti melakukan kecurangan, saya bersedia menerima segala sanksi yang berlaku"

B. Rancangan Mekanisme Program

Mekanisme dirancang dengan melibatkan akun Google peserta ujian yang terdaftar dengan pihak operasional (dosen). Berikut merupakan diagram alir dari sistem ini:



Gambar 6. Rancangan Mekanisme Program
(sumber: pribadi)

C. *Submisi Surat Pernyataan*

Pesan pernyataan yang harus ditandatangani dapat diunduh terlebih dahulu pada form kemudian ditandatangani. Pada form yang sama nantinya akan tersedia *field* untuk mengupload *file* pernyataan yang sudah ditandatangani

Gambar 7. Ilustrasi Submisi Surat Pernyataan
(sumber: pribadi)

IV. IMPLEMENTASI

Implementasi dilakukan berdasarkan desain rancangan yang dibuat. Pemrograman dilakukan menggunakan bahasa Python karena mudah untuk digunakan serta memanfaatkan library yang ada untuk membangkitkan fungsi SHA-3. Implementasi dibagi menjadi modul utama berupa pembangkitan *SHA-3,* modul *key generator*, dan modul RSA.

A. *SHA-3*

Modul ini berfungsi untuk menghasilkan *message digest* ketika diimplementasikan ke pesan. SHA-3 pada sistem ini memanfaatkan *library* yang tersedia pada Python

```
import hashlib

def hash_message(message: str) ->
str:
    return
hashlib.sha3_256(message.encode()).
hexdigest()
```

Kode Program 1. SHA3.py

B. *Key Generator*

Modul ini berfungsi untuk membangkitkan kunci

```
import random
from typing import Tuple
from util import is_relative_prime

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

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
```

Kode Program 2. key_generator.py

C. *RSA*

Modul ini berfungsi untuk melakukan enkripsi dan dekripsi terhadap tanda tangan

```
from typing import Tuple, IO
from io import StringIO, BytesIO
from SHA3_256 import hash_message

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

def generate_public_key(p: int, q:
int, e: int) -> Tuple[int, int]:
    return e, p * q

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

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

    cipher = []

    for m in message:
        cipher.append(pow(ord(m),
d, n))

    return ''.join(hex(x) for x in
cipher)

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)

    message = ""
    for c in cipher:
        message += chr(pow(c, e,
n))

    return message

def create_signature(text: str,
private_key: Tuple[int, int]) ->
str:
    d, n = private_key

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

```
def sign_in_file(file: IO,
private_key: Tuple[int, int]) ->
StringIO:
    s = BytesIO(file.read())

    signature =
create_signature(s.getvalue().decod
e(), private_key)
    s.seek(0, 2)

    s.writelines([b'\n*** Begin of
digital signature ****\n',
f'{signature}\n'.encode(), b'***
End of digital signature ****'])

    return s

def sign_separate_file(file: IO,
private_key: Tuple[int, int]) ->
StringIO:
    signature =
create_signature(file.read().decode
('latin1'), private_key)

    s = StringIO()
    s.writelines(['\n*** Begin of
digital signature ****\n',
f'{signature}\n', '*** End of
digital signature ****'])
    return s

def verify_in_file(file: IO,
public_key: Tuple[int, int]) ->
bool:
    e, n = public_key

    lines = file.readlines()
    file.seek(0)
    try:
        if lines[-1] == b'*** End
of digital signature ****' and
lines[-3] == b'*** Begin of digital
signature ****\n':
            print(lines)
            signature =
lines[-2][:-1]
        else:
            raise Exception('File
doesn\'t have signature')
    except:
        raise Exception('File
doesn\'t have signature')

    decrypted =
decrypt(signature.decode('latin-1')
, e, n)
```

```
    content =
file.read().decode('latin-1')

    return
hash_message('\n'.join(content.spli
t('\n')[:-3])) == decrypted

def verify_separate_file(file: IO,
signature_file: IO, public_key:
Tuple[int, int]) -> bool:
    e, n = public_key
    s =
file.read().decode('latin-1')

    file.seek(0)

    lines =
signature_file.readlines()
    signature_file.seek(0)
    try:
        if lines[-1] == '*** End of
digital signature ****' and
lines[-3] == '*** Begin of digital
signature ****\n':
            signature =
lines[-2][:-1]
        else:
            raise Exception('File
doesn\'t have signature')
    except:
        raise Exception('File
doesn\'t have signature')

    return decrypt(signature, e, n)
== hash_message(s)
```

Kode Program 3. RSA.py

V. Analisis Pengujian

Pengujian dilakukan untuk sistem yang sudah diimplementasikan. Pengujian dilakukan mulai dari fungsionalitas pembangkitan kunci, pengujian *signing* dan *verification,* pengujian submisi surat pakta integritas

A. Pengujian Pembangkitan Kunci

No	*Test Case*	*Input*	*Expected Output*	*Results*
1	Input Valid	p = 260189 q = 1700003491 e = 940002187	Key dibangkitkan key.priv dan key.pub	Berhasil
2	p dan q tidak prima	p = 260182 q = 1700003492 e = 940002187	Pesan error yang menunjukkan nilai p dan q harus prima	Berhasil
3	e tidak prima	p = 260189 q = 1700003491 e = 940002182	Pesan error yang menunjukkan nilai totient n dan e harus relatif prima	Berhasil

B. Pengujian Signing dan Verification

No	*Test Case*	*Input*	*Expected Output*	*Results*
1	Sign file .pdf	File = “Pakta Integritas.pdf” Key = private key X	File berhasil ditandatangani	Berhasil
2	Verification file .pdf key sesuai	File = “Pakta Integritas.pdf” Key = public key X	File berhasil diverifikasi	Berhasil
3	Verification file .pdf key tidak sesuai	File = “Pakta Integritas.pdf” Key = public key Y	Pesan peringatan file tidak berhasil diverifikasi	Berhasil

C. Pengujian Submisi Surat Pakta Integritas

No	*Test Case*	*Input*	*Expected Output*	*Results*
1	Input valid .pdf	file = “Pakta Integritas.pdf”	File terupload	Berhasil
2	Input tidak valid .png	file = “Pakta Integritas.png”	Pesan peringatan bahwa format file harus .pdf	Berhasil

VI. Kesimpulan dan Saran

Sistem tanda tangan untuk pernyataan pakta integritas pada ujian *online* berhasil dibuat menggunakan algoritma RSA dan

SHA-3. Sistem ini dapat bekerja dengan baik dengan tingkat keamanan yang lebih terjaga dari sisi *non-repudiation, integrity,* dan *authentication.* Penggunaan RSA dan fungsi hash SHA-3 juga telah terimplementasi dengan baik.

Harapannya, sistem ini dapat digunakan untuk menjaga keaslian pernyataan yang diadaptasi dari ujian *offline* ke ujian *online*. Untuk saran ke depannya, sistem ini mungkin saja bisa diintegrasikan menjadi fitur tersendiri yang ada di dalam Google Form. Dengan demikian pernyataan dalam ujian *online* akan lebih terjaga.

GITHUB

github.com/aldifadlian/Exam-Simulation-Digital-Signature

GOOGLE FORM EXAM SIMULATION

forms.gle/SFV9NWSr1r8WPhr89

ACKNOWLEDGMENT

Terima kasih untuk Tuhan Yang Maha Esa atas rahmat dan karunianya sehingga penulis dapat menyelesaikan makalah berjudul "Implementasi SHA-3 dan RSA pada Tanda Tangan Digital Dalam Pakta Integritas Ujian Online" untuk pemenuhan tugas II4031 Kriptografi dan Koding 2021/2022. Saya juga berterima kasih atas segala pihak yang telah membantu saya mempelajari materi selama perkuliahan berlangsung, khususnya untuk Bapak Dr. Ir. Rinaldi Munir, M.T. Harapannya, semoga makalah ini dapat bermanfaat dan dapat memberikan inspirasi serta solusi atas permasalahan yang ada.

REFERENCES

[1] Adelia, R. Miftahur, Nurpathonah, Z. Yoan, T. Ihsan, "The Role of Google Form As An Assessment Tool in ELT: Critical Review of The Literature", 2021.

[2] N. Sharma , Prabhjot and H. Kaur, "A Review of Information Security using Cryptography Technique," International Journal of Advanced Research in Computer Science, vol. 8, no. Special Issue, pp. 323-326, 2017.

[3] R. Munir, "Algoritma RSA," 2023

[4] S. Nisha, M. Farik, "RSA Public Key Cryptography Algorithm – A Review", 2017.

[5] R. Munir, "SHA-3 (Keccak)," 2023

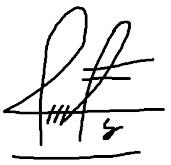
[6] D. Team, "Wet Signatures vs. Electronic Signatures — A Deep Dive", 2021.

[7] R. Munir, "Tanda Tangan Digital," 2021.

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



Aldi Fadlian Sunan 18220086