Memvalidasi Integritas Data dan Mengidentifikasi Pengguna melalui Kombinasi Algoritma RSA dan SHA-256

Marvel Pangondian - 13522075[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522075@itb.ac.id

***Abstract*—Kriptografi adalah cabang utama dalam ilmu keamanan komputer yang melibatkan teknik-teknik matematis dan komputasi untuk mengamankan komunikasi dan informasi. Tujuan kriptografi adalah teknik-teknik matematis dan komputasi untuk mengamankan komunikasi dan informasi. Salah satu algoritma yang digunakan untuk enkripsi dan deskripsi teks adalah algoritma RSA, tetapi hanya menggunakan algoritma tersebut dapat menyebabkan berbagai masalah. Salah satunya adalah kunci privat/publik yang tidak sesuai ketika mencoba enkripsi/dekripsi teks, keamanan enkripsi/dekripsi yang tidak terlalu bervariasi, serta integritas data yang tidak dapat dideteksi. Dalam makalah ini, akan dibahas penggunaan algoritma *hashing* yakni algoritma SHA-256 dalam meningkatkan keamanan data dan identifikasi pengguna, serta memeriksa integritas kunci publik dan privat yang digunakan.**

***Keywords*—Kriptografi, Algoritma RSA, Algoritma SHA-256**

I. PENDAHULUAN

Kriptografi ilmu yang menggunakan matematika untuk mengenkripsi dan mendekripsi data[1]. Sejarah kriptografi bermula ribuan tahun yang lalu, dan penciptaannya didorong oleh kebutuhan akan komunikasi yang aman dan perlindungan informasi yang sensitif. Salah satu metode enkripsi dan dekripsi yang terkenal pada peradaban kuno adalah Sandi Caesar, dinamai oleh penemunya Julius Caesar. Julius menciptakan Sandi Caesar karena dia tidak percaya kurir yang mengirimkan surat tersebut.

Kriptografi diciptakan karena kebutuhan manusia untuk menjaga kerahasiaan data yang mereka miliki. Salah satu algoritma yang efektif dalam melakukan hal ini adalah algoritma RSA(*Rivest-Shamir-Adleman*). Algoritma ini akan mengenkripsi dan mendekripsi teks menggunakan dua kunci berbeda yakni kunci privat dan kunci publik. Kunci publik merupakan kunci untuk melakukan enkripsi kepada sebuah teks, kunci ini bersifat publik dan dapat disebarkan. Kunci privat adalah kunci untuk dekripsi, kunci ini tidak dapat disebarkan. Kedua kunci tersebut akan mengenkripsi dan mendekripsi sebuah teks menggunakan teori bilangan yang akan dibahas lebih dalam pada bagian dasar teori

Algoritma RSA merupakan algoritma yang kuat, tetapi memiliki beberapa kelemahan. Apa yang terjadi jika pengguna mencoba mengakses *plain text* dengan kunci publik/privat yang tidak sesuai ? Atau apa yang terjadi jika enkripsi data diubah, bagaimana pengguna mengetahui hasil enkripsinya telah diubah oleh pihak ketiga ? Untuk menjaga keamanan dan integritas data, diperlukan konsep dan algoritma lain. Konsep yang akan digunakan untuk menangani adalah konsep *signature* .

Signature adalah konsep/cara untuk memverifikasi integritas data, pengguna, dan kunci. Dengan konsep *signature*, dapat dibuat sebuah program untuk mendeteksi integritas sebuah data dan mendeteksi apakah data tersebut sudah diubah sebelumnya atau tidak. *Signature* juga digunakan untuk memverifikasi identitas pengguna

Salah satu cara untuk menerapkan konsep *signature* adalah penerapan *signature* kepada sebuah *encrypted text*. Ketika sebuah teks dienkripsi, hasil enkripsi tersebut dapat diubah oleh pengguna lainnya. Untuk mengetahui integritas dari *encrypted text*, dapat digunakan konsep *signature* yang menyimpan *hash value text* sebelumnya, sehingga *text* dapat dideteksi integritasnya menggunakan *signature* yang sudah disimpan. Jika hasil *hash text* berbeda dengan *signature*, maka *text* tersebut sudah diubah sebelumnya

Salah satu algoritma yang dapat digunakan untuk konsep *signature* ini adalah algoritma SHA-256 yang dapat mengubah sebuah *string* menjadi *hash value unique* sebesar 256 bits. Dengan algoritma ini, dapat dibentuknya program untuk mengecek integritas pengguna dan data yang ingin didekripsi

II. LANDASAN TEORI

A.Bilangan Bulat

Bilangan bulat adalah angka-angka yang tidak memiliki bagian desimal atau pecahan. Bilangan ini dapat berupa angka positif, negatif, atau nol. Bilangan bulat termasuk dalam kategori bilangan cacah (non-negatif) dan bilangan negatif. Bilangan bulat dapat berupa bilangan positif (1, 2, 3, ...), bilangan negatif (-1, -2, -3, …), dan nol (0).

B.Bilangan Riil

Bilangan riil adalah bilangan yang mencakup seluruh bilangan yang mencakup seluruh bilangan rasional dan irasional. Bilangan riil terdiri atas bilangan bulat (…,-3, -2, -1, 0, 1, 2, 3,…), bilangan rasional (1/2, -5/4, dll), dan bilangan irasional (√2, π, dll).

C.Bilangan Prima

Bilangan prima adalah bilangan bulat yang lebih dari satu dan hanya dapat dibagi oleh 1 dan bilangan tersebut sendiri. Contoh bilangan bulat adalah 7 karena 7 hanya dapat habis dibagi oleh 1 dan 7 saja. Bilangan selain bilangan prima disebut sebagai bilangan komposit

D.Sifat Pembagian

Sebuah bilangan bulat *a* habis membagi sebuah bilangan bulat *b* jika dan hanya jika terdapat sebuah bilangan *c* yang merupakan bilangan bulat, sedemikian hingga *ac* = *b* . Jika tidak ada *c*, maka *a* tidak habis membagi *b*. Berikut adalah notasi sifat pembagian :

$$a \mid b \; jika \; dan \; hanya \; jika \; ac = b, c \in \mathrm{Z} \; dan \; a \neq 0$$

E.Teorema Euclidean

Teorema Eculidean menyatakan bahwa untuk setiap dua integer, *m* (pembilang) dan *n* (penyebut), di mana n tidak sama dengan nol, terdapat dua integer *q* (hasil bagi) dan *r* (hasil sisah) sedemikian hingga,

$$m = nq + r \, , 0 \leq r < n$$

F.Pembagi Bersama Terbesar (gcd)

Pembagi bersama terbesar (gcd) merupakan bilangan bulat terbesar *c* sedemikian hingga dapat habis membagi dua bilangan bulat *a* dan *b*.

$$c \mid a \; dan \; c \mid b$$

G.Algoritma Euclidean

Algoritma Euclidean adalah algoritma untuk mencari gcd dari dua bilangan bulat. Algoritma ini menyatakan bahwa untuk setiap bilangan bulat non negatif *m* dan *n* di mana *m* ≥ *n*, maka jika $m = r_0$ dan $n = r_1$ berlaku :

$$\begin{aligned} r_0 &= r_1 q_1 + r_2 \quad 0 \leq r_2 < r_1 \\ r_1 &= r_2 q_2 + r_3 \quad 0 \leq r_3 < r_2 \\ &\vdots \\ r_{n-1} &= r_n q_n + 0 \end{aligned}$$

Dengan aturan di atas, maka gcd *m* dan *n* dapat dicari menggunakan Teorema 2,

$$GCD(m,n) = GCD(r_0, r_1) = \ldots = GCD(r_n, 0) = r_n$$

*H.*Relatif Prima

Dua bilangan, *m* dan *n*, dikatakan relatif prima jika dan hanya jika,

$$GCD(m,n) = 1$$

Relatif prima akan digunakan dalam implementasi enkripsi menggunakan algoritma RSA

I.Aritmatika Modulo

Untuk bilangan bulat *m* dan *n* di mana berlaku kombinasi linier sebagai berikut :

$$m = nq + r, \qquad 0 \leq r < m$$

maka dapat ditulis ulang sebagai berikut :

$$a \bmod m = r$$

notasi di atas dapat diinterpretasikan sebagai "*a* dibagi dengan *m* menghasilkan sisa *r* "

J.Kongruen dan Sifat – Sifatnya

Untuk bilangan bulat *a* dan *b* dikatakan kongruen jika dan hanya jika terdapat modulus *c* yang membagi *a* dan *b* dan menghasilkan sisa *r* yang sama

$$a \bmod c = r \, , b \bmod c = r$$

Jika berlaku di atas, maka *a* kongruen dengan *b* dalam modulus *c*. Bilangan *a* dapat dikatakan kongruen dengan b dalam jika dan hanya jika (*a*-*b*) juga habis dibagi dengan *c*.

$$a \equiv b (mod \; c) \;\; iff \;\; c \mid (a - b)$$

melalui kongruen, didapat beberapa sifat yakni,

1. Jika $a \equiv b \pmod{c}$, dan *k* adalah sembarang bilangan bulat, maka berlaku
   i.$(a + k) \equiv (b + k) \pmod{c}$
   ii. $ak \equiv bk \pmod{c}$
   iii.$a^k \equiv b^k \pmod{c}$, *k* bilangan bulat tak-negatif

2. Jika $a \equiv b \pmod{c}$ dan $k \equiv m \pmod{c}$, maka
   i. $(a + k) \equiv (b + m) \pmod{c}$
   ii. $ak \equiv bm \pmod{c}$

K.Balikan Modulo (Modulo Inverse)

Untuk mencari inverse dari *a* (mod b), maka harus berlaku gcd(*a*, *b*) = 1, jika tidak maka *a* (mod b) tidak memiliki inverse. Untuk gcd(*a*, *b*) = 1, maka berlaku :

$$am + bn = 1$$

melalu implikasi, didapat :

$$am + bn \equiv 1 \; (mod \; b)$$

Perhatikan bahwa

$$bn \equiv 0 \; (mod \; b)$$

maka,

$$am = 1 \; (mod \; b)$$

Kekongruenan terakhir ini berarti m merupakan *inverse* dari a (mod b). Cara lain mencari inverse dari *a* (mod *b*) adalah dengan cara berikut :

$$ax \equiv 1 \; (mod \; b)$$

$$x = \frac{1 + bk}{a}$$

x yang merupakan *inverse* dari a (mod b) dapat diperoleh dengan mencari nilai k sedemikian hingga hasil x di atas merupakan bilangan bulat.

L.Kekongruenan Linier dan Sistem Kekongruenan linier

Kekongruenan linier secara umum memiliki bentuk sebagai berikut:

$$ax \equiv r_0 (mod\ m_0)$$

solusi x dapat dicari dengan mengubah bentuk kongruen di atas menjadi

$$x = \frac{(r_0 + m_0 k)}{a}$$

dengan memilih nilai k sedemikian hingga hasil x adalah bilangan bulat. Nilai k yang menghasilkan nilai x merupakan bilangan bulat.

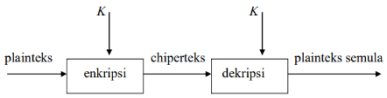
Sistem kekongruenan linier berarti sistem kongruen dengan lebih dari satu kongruen, sebagai contoh :

$$x \equiv r_0\ (mod\ m_0)$$
$$x \equiv \mathrm{r}_1 (mod\ m_1)$$
$$x \equiv \mathrm{r}_2 (mod\ m_2)$$
....

Sistem ini dikenal sebagai sistem kekongruenan linier karena setiap persamaan adalah kongruen linier. Untuk memecahkan sistem kekongruenan linier, kita perlu mencari x yang memenuhi semua kongruen pada sistem kekongruenen linier. Dalam banyak kasus, sistem kekongruenan linier dapat dipecahkan menggunakan teorema Chinese Remainder Theorem (CRT)

M.Kriptografi

Kriptografi adalah seni menyembunyikan data menggunakan ilmu matematika. Kriptografi berasal dari bahasa Yunani "kryptós" yang berarti tersembunyi/rahasia, serta "gráphein" yang berarti menulis. Ketika kedua kata tersebut digabung, didapat kata kriptografi yang berarti studi mempelajari teknik – Teknik untuk menyembunyikan data agar terlindungi dari pihak ketiga. Kriptografi memiliki tujuan yakni menjaga pesan/data agar tidak dapat dibaca oleh pihak ketiga. Kriptografi terdiri atas dua komponen yakni pesan (*plaintext*) yang berisi data/informasi yang ingin disembunyikan, serta *cipherteks* (*ciphertext*) yang berisi hasil setelah pesan disembunyikan. Terdapat dua proses utama dalam kriptografi yakni proses enkripsi dan dekripsi. Proses enkripsi adalah proses mengubah sebuah data, biasanya sebuah pesan, menjadi data yang tidak dapat dibaca oleh orang lain secara umum. Hasil proses enkripsi adalah text/data yang bersifat *unreadable/indecipherable*. Proses berikutnya dalam kriptiografi adalah proses dekripsi. Proses ini melibatkan *ciphertext* dan mengubah kembali menjadi *plain text*. Proses ini hanya dapat dilakukan oleh pengguna yang mengetahui jenis algoritma yang digunakan untuk mengenkripsi *plain text* serta mengetahui kunci – kunci yang digunakan.

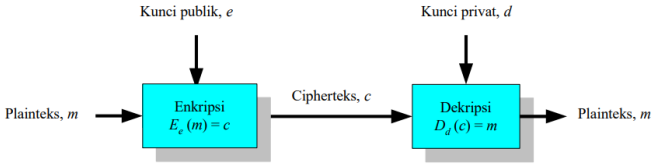


Gambar 2.1 Alur Kriptografi
Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Teori-Bilangan-2020-Bagian3.pdf

Kripografi terdiri atas dua tipe, yakni tipe pertama *Symmetric-Key Cryptography* dan tipe kedua *Public-Key Cryptography (Asymmetric Cryptography)*[2]. Untuk tipe pertama, kriptografi menggunakan satu kunci saja untuk enkripsi dan dekripsi, sedangkan tipe kedua menggunakan dua kunci yakni kunci publik untuk enkripsi, dan kunci privat untuk dekripsi. Di antara kedua tipe, tipe kedua memiliki keamanan yang lebih ketat dibandingkan tipe pertama. Salah satu algoritma implementasi kriptografi tipe *Asymmetric Cryptography* adalah algoritma RSA (*Rivest-Shamir-Adleman*).

N.Algoritma RSA (Rivest-Shamir-Adleman)

Algoritma RSA dinamakan berdasarkan penemunya yaitu Ron Rivest, Adi Shamir, dan Leonard Adleman. Algoritma ini merupakan penerapan jenis kriptografi yakni *Asymmetric Cryptography* yang menggunakan dua kunci[8]. Dua kunci tersebut adalah kunci publik untuk enkripsi dan kunci privat untuk dekrispi.



Gambar 2.2 Alur Algoritma RSA
Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Teori-Bilangan-2020-Bagian3.pdf

alur pembuatan kunci algoritma RSA adalah sebagai berikut :

1. Pertama, dipilihnya dua angka, p dan q yang merupakan angka prima yang *random* dan berbeda.
2. Menhitung $n = pq$ yang merupakan modulus algoritma RSA (tidak perlu dirahasiakan)
3. Menghitung $\phi(n) = (p - 1)(q - 1)$ (perlu dirahasiakan)
4. Mencari kunci publik (e), kunci tersebut memiliki nilai $1 < e < \phi(n)$ dan relatif prima dengan n dan $\phi(n)$. Kunci ini akan digunakan untuk enkripsi dan dapat disebarkan
5. Mencari kunci privat (d) yang memenuhi syarat $de \equiv 1$ (mod $\phi(n)$). Kunci privat (d) akan digunakan untuk mendekripsi *ciphertext*

alur enkripsi dan dekripsi dengan algoritma RSA:

1. Pertama, akan dibuat kunci publik (e) dan kunci privat (d). Pembentukan kunci biasanya dilakukan dengan memilih p dan q yang random
2. Untuk enkripsi, mengubah tiap *character* pada *plain text* menjadi unicode, setelah itu mengubah tiap *value character* menjadi *value* lain dengan rumus berkut :

$$c' = c^e\ mod\ n$$

Setelah itu hasilnya disimpan menjadi *cipher text*

3. Untuk dekripsi, pertama menerima *cipher text* dan mengubah setiap *character value* pada *cipher text* kembali menjadi *value* semula dengan rumus berikut :

$$c = c'^{d} mod\ n$$

hasilnya kemudian disimpan menjadi *plain text* yang semula

O.Fungsi Hash

Fungsi *hash* adalah algoritma matematis yang mengubah input data (pesan) menjadi nilai *hash* dengan panjang tetap digunakan untuk beragam keperluan, termasuk memastikan keutuhan data, memvalidasi kata sandi, dan menyimpan data dalam bentuk struktur data *hash*. Salah satu fungsi *hash* yang efektif adalah SHA-256 (*Secure Hash Algorithm 256-bit*)

P.Fungsi Hash SHA-256 (Secure Hash Algorithm 256-bit)

SHA-256, yang merupakan singkatan dari Secure *Hash Algorithm* 256-bit, adalah suatu algoritma *hash* kriptografis yang termasuk dalam serangkaian algoritma Secure *Hash Algorithm* (SHA) yang dirancang oleh National Institute of Standards and Technology (NIST) Amerika Serikat. Proses SHA-256 menghasilkan nilai *hash* yang memiliki panjang tetap sebesar 256 bit, setara dengan 32 byte. Algoritma SHA-256 dimulai dengan mengubah input (berupa string) menjadi binary sesuai dengan ASCII. Binary masing – masing character dikonkatenasi menjadi satu binary yang besar. Sebelum melakukan kalkulasi, hasil biner dipadding terlebih dahulu dengan satu "1" dan sisanya "0" sampai jumlah bit merupakan kelipatan 512 dikurang 64 bit. 64 bit terakhir diisi dengan representasi biner dari panjang awal biner. Hasil akhir proses ini adalah pesan input yang diubah menjadi blok – blok berukuran 512 bits. Selanjutnya, setiap blok pesan diproses secara berurutan melalui serangkaian fungsi kompresi yang melibatkan operasi logika biner, bitwise, dan fungsi matematika kriptografis. Iterasi ini berulang untuk setiap blok pesan, dan setiap iterasi menggabungkan hasil dari iterasi sebelumnya dengan blok pesan berikutnya. Setelah semua blok pesan diproses, hasil akhir dari iterasi-iterasi tersebut adalah nilai *hash* final yang memiliki panjang 256 bit (32 byte).

Q.Digital Signature

Digital signature adalah jenis tanda tangan elektronik khusus yang melibatkan penggunaan teknik kriptografi. *Digital signature* menyediakan cara yang aman dan tahan terhadap penyusup untuk mengonfirmasi asal dan otentikasi pesan atau dokumen digital. *Signature* ini bergantung pada kriptografi kunci publik, di mana kunci pribadi digunakan untuk menghasilkan tanda tangan, dan kunci publik yang sesuai digunakan untuk memverifikasinya. *Digital Signature* digunakan untuk memastikan bahwa informasi berasal dari pengguna yang benar dan memvalidasi integritas data[10].

III. ANALISIS MASALAH

A. Implementasi Konsep Digital Signature

Salah satu cara untuk memvalidasi integritas data adalah melalui *digital signature*. *Digital signature* digunakan untuk memastikan data valid/tidak. Implementasi *digital signature* akan menggunakan fungsi *hash* SHA-256 untuk menghash data berupa *text* menjadi nilai *hash* yang memiliki panjang tetap sebesar 256 bit, setara dengan 32 byte. Alasan penulis memilih fungsi *hash* SHA-256 adalah memastikan besar *signature* selalu konsisten serta algoritma SHA-256 telah terbukti memiliki probabilitas *collision* yang rendah[3]. *Signature* akan dimasukan pada hasil enkripsi (*cipher text*) dan akan diambil kembali ketika ingin memvalidasi integritas data. *Signature* juga akan digunakan untuk mengecek integritas *private key* dan *public key* yang digunakan pengguna untuk enkripsi atau dekripsi.

B. Validasi Integritas Data

Data akan divalidasi melalui *signature* yang ada dalam data tersebut. Pada proses enkripsi, *text* akan diubah menjadi *hash value* 256 bit menggunakan algoritma SHA-256. *Hash value* tersebut kemudian dijadikan *signature* dengan menggunakan *private key (d)*. Caranya adalah dengan mengenkripsi *hash value* tersebut menggunakan *d* dan *modulus (n)* ($signature = hash^{d} \bmod n$). *Signature* kemudian dimasukan ke dalam *cipher text*. Ketika ingin validasi integritas data, maka *signature* dalam *cipher text* perlu dipsah terlebih dahulu. *Cipher text* perlu didekripsi, setelah itu hasil dekripsi di *hash* menggunakan algoritma SHA-256. Signature awal text di ubah menjadi *value* mulanya (*value* sebelum di enkripsi menjadi signature) dengan menggunakan kunci publik (*e*) dan modulus (*n*) ($hash = signature^{e} \bmod n$). Hasil *hash* semula kemudian di bandingkan dengan hasil *hash* dekripsi. Jika sama, maka data sama dengan data orijinal, jika beda maka data telah diubah/diedit oleh pihak ketiga (*data is compromised*).

C. Validasi Pengguna

Ketika mencoba untuk mendekripsi/mengenkripsi data, perlu dipastikan integritas *private key* dan *public key* yang digunakan. Hal tersebut dapat dilakukan dengan melakukan validasi pengguna. Setiap pengguna akan memiliki *username* dan *password* pilihan mereka sendiri, kemudian program akan menggunakan *private key*, *username*, dan *password* untuk membuat *signature* unik yang akan digunakan untuk memvalidasi *public key* dan *private key* ($signature = hashvalue(username + password)^{d} \bmod n$), *signature* tersebut kemudian disimpan di *database*. Ketika pengguna ingin mengenkripsi/mendekrispsi, program akan meminta *username*, *password*, *public key*, dan *private key* pengguna, kemudian program akan memvalidasi integritas *public key*, dan *private key* berdasarkan *signature* yang sudah disimpan di database.

D. Implementasi Algoritma RSA

Algoritma enkripsi dan dekripsi yang akan digunakan adalah algoritma RSA (*Rivest-Shamir-Adleman*). Berikut alur enkripsi dalam implementasi RSA menggunakan python :

1. Menerima *message* yang akan dienkripsi
2. Mengubah tiap *character* pada *message* menjadi integer dengan menggunakan *public key* dan modulus ($c' = c^{e} \bmod n$)
3. Tiap integer diubah menjadi string, dan di padding dengan "0" untuk memastikan tiap representasi *character* memiliki panjang 6 angka
4. Setelah itu, tiap representasi *character* di konkatenasi menjadi satu string yang besar

5. String tersebut kemudian di save di database yakni pada .json file
6. Pada .json file juga disimpan *signature message* yang sudah dibuat dengan menggunakan *private key*

Berikut alur dekripsi dalam implementasi RSA menggunakan python :

1. Pertama, user akan memilih *entry/message* yang ingin didekripsi
2. Program akan mengambil data dari *database* yakni *cipher text*
3. *Cipher text* tersebut kemudian di parser menjadi *block of 6 characters*, kemudian tiap blok tersebut diubah menjadi integer
4. Tiap block merupakan representasi tiap *character* yang perlu didekrispi. Tiap blok didekripsi dengan menggunakan *private key (d)* dan *modulus (n)* dari pengguna ($c = c'^{d} mod\ n$). Setelah tiap blok didekripsi, maka perlu dikonkatenasi menjadi satu *string* yang merupkan *plain text*
5. Sebelum hasil dekripsi dikirim ke pengguna, perlu validasi integritas data terlebih dahulu. *Signature data* yang disimpan pada *database* akan digunakan dan dibandingkan dengan *signature* yang dihasilkan menggunakan hasil dekripsi. Jika kedua *signature* berbeda, maka terbukti bahwa *cipher text* telah diubah oleh pihak ketiga.

E. Database

Untuk menyimpan hasil *signature* supaya dapat memvalidasi isi data dan memvalidasi pengguna, diperlukan *database* untuk menyimpan *signature* pengguna serta *cipher text* yang berisi *signature data*. Jenis data yang penulis gunakan untuk menyimpan hal-hal tersebut adalah .json file. Implementasi ini masih sederhana, dalam dunia nyata .json file jarang digunakan sebagai tempat penyimpanan data, tetapi dalam kasus simpel ini penulis memutuskan untuk menggunakan .json file untuk memudahkan proses *debugging*. .json file akan berupa *list of dictionary*, dalam setiap *dictionary* akan terdiri atas "id", "username", "signature", "data" sebagai keys nya. *Value* "id" berisi id dari *dictionary*, *value* "username" akan berisi username pengguna yang sudah di *hash* menggunakan algoritma SHA-256 (untuk menjaga keamanan dan menghemat data, ukuran string dalam *python* secara default 48 byte (jika string kosong), jauh lebih besar dibandingkan hasil *hash* SHA-256 yakni 32 byte[9]), *value* "siganature" adalah *signature* pengguna yang akan digunakan untuk validasi pengguna, *value* "data" adalah *cipher text* berserta *signature* data yang sudah disisipkan pada *cipher text* yang akan digunakan untuk memvalidasi integritas data

IV. IMPLEMENTASI PROGRAM

Penulis berhasil membuat program yang dapat memvalidasi integritas data serta mengidentifikasi pengguna dengan menggunakan bahasa *pyhton*. Program secara garis besar akan membuat *digital signature* menggunakan *private key* dan *text* yang ingin pengguna enkripsi. *Digital signature* tersebut dibuat dengan menggunakan algoritma SHA-256. Untuk algoritma SHA-256 sendiri, penulis menggunakan *library hashlib*. Untuk algortima enkripsi serta dekripsi, penulis menggunakan algoritma RSA. Implementasi program menggunakan *library* serta fungsi sebagai berikut :

1. ***hashlib, json, random, sympy, art***
   Library yang digunakan berupa *library hashlib* untuk implementasi SHA-256, *library json* untuk implementasi *database*, *library random* yang digunakan saat ingin memilih bilangan prima secara acak serta memilih *public key*, *library sympy* untuk membantu dalam memilih bilangan prima, dan library *art* untuk menu utama

```
import hashlib
import json
import random
from sympy import *
from art import *
```

Gambar 4.1 Library yang digunakan dalam implementasi program
Sumber : Dokumentasi pribadi

2. **gcd**
   Fungsi ini menerima dua angka dan mengembalikan gcd (*greates common divisor*) dari kedua angka tersebut. Penulis menggunakan rekursi dan algoritma euclidean dalam implementasi fungsi ini.

```
def gcd(num1,num2):
    if (num1 == 0 or num2 == 0):
        return 0
    if (num1 == num2):
        return num1
    if (num1 > num2):
        return gcd(num1 - num2, num2)
    return gcd(num1,num2 - num1)
```

Gambar 4.2 Fungsi gcd untuk menemukan gcd dua angka
Sumber : Dokumentasi pribadi

3. **generate_random_prime**
   Fungsi ini akan mengembalikan sebuah bilangan prima secara acak. Fungsi ini menggunakan *library random* dalam memlih angka dan fungsi *isprime* yang merupkan fungsi dari *library sympy* untuk memastikan bilangan merupakan bilangan prima. Penulis membatasi *range* dari angka yang dipilih yakni dari 500 sampai 1000 untuk mempermudah kalkulasi serta menghindari terpilihnya bilangan prima yang terlalu kecil (< 100).

```
def generate_random_prime():
    num = random.randint(500,1000)
    while (not (isprime(num))):
        num = random.randint(500,1000)
    return num
```

Gambar 4.3 Fungsi untuk mencari bilangan prima secara acak
Sumber : Dokumentasi pribadi

4. **is_coprime**
Fungsi ini menerima dua angka dan mengembalikan *true* jika kedua angka tersebut adalah relatif prima

```
def is_coprime(num1,num2):
    return (gcd(num1,num2) == 1)
```

Gambar 4.4 Fungsi untuk memastikan dua angka relatif prima
Sumber : Dokumentasi pribadi

5. **generate_e_key**
Fungsi ini akan menghasilkan *public key* yang sesuai dengan modulus dan hasil *phi function (φ(n))*. Pemilihan *public key* dilakukan secara acak, yakni dengan menggunakan *library random.*

```
def generate_e_key(n,phi_n):
    e = random.randint(2,(phi_n)//2)
    while not (is_coprime(e,n) and is_coprime(e,phi_n) ):
        e = random.randint(2,(phi_n)//2)
    if (e > pow(e, -1, phi_n)):
        return generate_e_key(n,phi_n)
    return e
```

Gambar 4.5 Fungsi untuk membuat public key
Sumber : Dokumentasi pribadi

6. **generate_d_key**
Fungsi ini akan menghasilkan *private key* sesuai dengan *public key* dan hasil *phi function* yang sudah dibuat sebelumnya

```
def generate_d_key(e, phi_n):
    d = pow(e, -1, phi_n)
    return d
```

Gambar 4.6 Fungsi untuk membuat private key
Sumber : Dokumentasi pribadi

7. **generate_random_key**
Fungsi ini akan membuat dan mengembalikan *modulus*, *public key*, dan *private key*.

```
def generate_random_key():

    #generate random prime numbers
    p = generate_random_prime()
    q = p
    while (q  == p):
        q = generate_random_prime()

    #generate n and phi_n
    n = p*q
    phi_n = (p - 1) * (q - 1)

    # generating encryption key, this is the public key
    e = generate_e_key(n,phi_n)

    # generating the decryption key, this is the
    d = generate_d_key(e,phi_n)

    return (n,e,d)
```

Gambar 4.7 Fungsi untuk membuat modulus, private key, dan public key
Sumber : Dokumentasi pribadi

8. **generate_signature**
Fungsi in akan membuat *digital signature* dari *username*, *password*, *modulus*, dan *private key* pengguna. *Digital signature* ini akan dipakai ketika ingin identifikasi pengguna serta memvalidasi *private key* dan *public key* pengguna. Fungsi akan menggabungkan *username* dan *password* pengguna menjadi satu string yang kemudian akan di *hash* menggunakan algoritma SHA-256. Hasil *hash* akan dijadikan *digital signature* dengan mengenkripsi hasil *hash* menggunakan *private key*.

```
def generate_signature(username,password,modulus,private):
    n,d = modulus,private
    user_pass = username + password

    # hashing using sha256
    hash_user_pass = int(hashlib.sha256(user_pass.encode()).hexdigest(), 16)
    hash_user_pass = hash_user_pass % n # making sure hash is within range of n

    # signature
    signature = pow(hash_user_pass, d, n)

    return signature
```

Gambar 4.8 Fungsi untuk membuat digital signature pengguna
Sumber : Dokumentasi pribadi

9. **verify_password**
Fungsi ini akan memvalidasi *username*, *password*, *private key*, dan *public key* yang akan digunakan oleh pengguna. Cara kerja fungsi secara garis besar sama dengan yang sudah dijelaskan pada bagian *Validasi Pengguna* bab III laporan.

```
# verification functions
def verify_password(username,password, signature,modulus,private_key, public_key):

    # Hash the provided password with SHA-256
    hashed_password = int(hashlib.sha256((username + password).encode()).hexdigest(), 16)

    # Ensure the hashed password is within the range of n
    hashed_password = hashed_password % modulus

    # Verification
    verified_password1 = pow(signature, public_key, modulus)
    verified_password2 = pow(hashed_password, private_key, modulus)

    return (verified_password1 == hashed_password) and (verified_password2 == signature)
```

Gambar 4.9 Fungsi untuk memvalidasi pengguna
Sumber : Dokumentasi pribadi

10. **verify_content**
Fungsi ini akan memvalidasi integritas data *cipher text* yang sudah disimpan di *database*. Cara kerja fungsi secara garis besar sama dengan yang sudah dijelaskan pada bagian *Validasi Integritas Data* bab III laporan.

```
def verify_content(text,signature):
    hashed_text = int(hashlib.sha256(text.encode()).hexdigest(), 16)
    hashed_text = hashed_text % modulus

    # verification
    verified_text = pow(signature, public_key, modulus)

    return (verified_text == hashed_text)
```

Gambar 4.10 Fungsi untuk memvalidasi integritas data
Sumber : Dokumentasi pribadi

11. **encryption_message**
Fungsi untuk mengenkripsi pesan pengguna (dengan memanggil fungsi *encryption_rsa*) serta membuat *signature* pesan tersebut.

```python
def encryption_message():
    n,d = modulus,private_key

    # encrpytion text
    print(" Encryption ".center(50,"="))
    text = input("Enter your log : ")
    text_enc = encryption_rsa(modulus,public_key,text)
    text_enc = [text_enc]

    print()
    # creating signature to ensure integrity
    hash_message = int(hashlib.sha256(text.encode()).hexdigest(), 16)
    hash_message = hash_message % n # making sure hash is within range of n
    signature_text = pow(hash_message, d, n)

    # adding signature to data
    text_enc.append(signature_text)

    ob["data"].append(text_enc)
    print("Encryption successful !")
```

Gambar 4.11 Fungsi untuk mengenkripsi dan membuat signature pesan
Sumber : Dokumentasi pribadi

12. encryption_rsa

Fungsi untuk mengenkripsi pesan menggunakan algoritma RSA. Fungsi in mengembalikan *cipher text* yang merupakan *string integer* yang merupakan representasi integer hasil enkripsi dalam bentuk *string* yang sudah dipadding dengan angka 0. Alasan dipadding adalah supaya setiap blok *string* yang merepresentasikan karakter itu konsisten panjangnya, yakni 6 *character integer.*

```python
def encryption_rsa(modulus,public,message):
    ciphertext = [pow(ord(char), public, modulus) for char in message]
    text_enc = ""
    for i in ciphertext:
        text_enc = text_enc + str('{:06}'.format(i)) # pad zero

    return text_enc
```

Gambar 4.12 Fungsi untuk mengenkripsi string menggunakan algoritma RSA
Sumber : Dokumentasi pribadi

13. decryption_message

Fungsi ini akan mendekripsi *cipher text* yang dipilih pengguna. Setiap pengguna memiliki *entry* masing – masing, yakni *cipher text* yang sudah disimpan di *database*. Setiap pengguna dapat memiliki lebih dari satu *entry* sehingga pengguna harus memilih dulu *entry* yang ingin didekripsi. Setelah itu, fungsi ini akan mencoba mendekripsi *entry* yang sudah dipilih dengan memanggil fungsi *decryption_rsa*. Hasil dekripsi tersebut akan divalidasi integritasnya dengan menggunakan fungsi *verify_content*

```python
def decryption_message():
    print(" Decryption  ".center(50,"="))
    arr_text = ob["data"]
    if (arr_text != []):
        k = len(arr_text)
        selection = int(input(f"Select entry ({k} entries) : "))
        while (selection < 1 or selection > k):
            print("Please enter the correct entry !")
            selection = int(input(f"Select entry ({k} entries) : "))


        dec_text = decryption_rsa(modulus,private_key,arr_text[selection - 1][0])

        # verify content
        if (not verify_content(dec_text,arr_text[selection - 1][-1])):
            print_with_color("WARNING, DATA INTEGRITY HAS BEEN COMPROMISED!\n",91)
            print("CORRUPTED TEXT : ")
            print(dec_text)
        else:
            print("Decrypted Text : ")
            print(dec_text)
    else:
        print("Data is empty")
```

Gambar 4.13 Fungsi untuk mendekripsi dan memvalidasi cipher text
Sumber : Dokumentasi pribadi

14. decryption_rsa

Fungsi ini akan menerima *modulus, private key*, dan *cipher text* untuk mengembalikan hasil dekripsi *cipher text. Cipher text* akan diparse mejadi *block of 6 characters*. Setelah diparse, setiap blok akan diubah kembali menjadi integer dan kemudian didekripsi menggunakan algoritma RSA

```python
def decryption_rsa(modulus,private,enc_message):
    message = enc_message
    message_parsed  = []
    message_arr = []
    i = 0
    while i < int(len(message)):
        if ((i + 6) >= int(len(message)) ):
            message_parsed.append(message[i:])
        else:
            message_parsed.append(message[i:i+6])
        i += 6
    for i in range (len(message_parsed)):
        try:
            message_arr.append(int(message_parsed[i]))
        except ValueError:
            raise ValueError("empty...")
    decoded_message = ''
    for char in message_arr:
        decrypted_char = pow(char, private, modulus)
        decoded_message += chr(decrypted_char % 128)
    return decoded_message
```

Gambar 4.14 Fungsi untuk mendekripsi cipher text
Sumber : Dokumentasi pribadi

15. load_database dan save_database

Fungsi *load_database* akan membuka dan *load* data yang ada pada *database*. Fungsi *save_database* akan menyimpan data ke *database*

```python
def load_database():
    with open("./database/data.json", 'r') as openfile:
        json_object = json.load(openfile)
    return json_object

def save_database():
    data = load_database()
    data[ob["id"]] = ob
    json_object = json.dumps(data, indent=4)
    with open("./database/data.json", "w") as outfile:
        outfile.write(json_object)
```

Gambar 4.15 Fungsi – fungsi yang mengatur database
Sumber : Dokumentasi pribadi

16. login

Fungsi ini akan meminta *username, password, private key*, dan *public key* untuk identifikasi pengguna. Login juga menggunakan *username, password, private key*, dan *public key* untuk validasi integritas *public key* dan *private key* yang digunakan pengguna.

.

```python
def login():
    global username
    global password
    global modulus
    global private_key
    global public_key
    global has_login
    global done
    global user_input
    global ob
    print(" login  ".center(50,"-"))

    data = load_database()
    username = input("Enter username     : ")
    username_hash = int(hashlib.sha256(username.encode()).hexdigest(), 16)
    dict_data = {}
    for ob in data :
        if (ob["username"] == username_hash):
            dict_data = ob

    if not(bool(dict_data)):
        print("That username doesn't exist")
        username = ""
        return

    password_temp =     input("Input password     : ")
    private_temp =  int(input("Input private key  : "))
    public_temp =   int(input("Input public key   : "))
    modulus_temp =  int(input("modulus           : "))

    if (verify_password(username,password_temp,dict_data["signature"],modulus_temp,private_temp,public_temp)):
        print("Success !")
        print()
        password = password_temp
        modulus = modulus_temp
        private_key = private_temp
        public_key = public_temp
        ob = dict_data
        has_login = 1

    else:
        print("verificaiton failed")
        logout()
```

Gambar 4.16 Fungsi untuk mengidentifikasi pengguna
Sumber : Dokumentasi pribadi

17. logout_and_init

Fungsi ini digunakan pada saat pengguna ingin ”logout”, yakni ketika pengguna mungkin ingin menggunakan *public key* dan *private key* yang berbeda, atau pengguna ingin *signing* baru. Fungsi ini akan me-*reset* variabel global yang ada pada program.

```python
def logout_and_init():
    global username
    global password
    global modulus
    global private_key
    global public_key
    global has_login
    global done
    global user_input
    global ob

    user_input = 0
    modulus = -999
    public_key = -999
    private_key = -999
    username = ""
    password = ""
    has_login = -1
    done = 0
    if (ob != {}):
        save_database()
    ob = {}
```

Gambar 4.17 Fungsi untuk logout
Sumber : Dokumentasi pribadi

18. signing

Fungsi ini akan dipanggil ketika pengguna ingin membuat *digital signature* baru yang akan digunakan untuk identifikasi pengguna. Fungsi ini akan meminta *username* dan *password* yang unik dari pengguna untuk digunakan dalam pembuatan *digital signature*. Kemudian, fungsi akan memberikan *private key*, *public key*, dan *modulus* yang pengguna harus simpan. Fungsi ini ibaratnya membuat “akun” baru bagi pengguna yang pengguna dapat gunakan untuk enkripsi dan dekripsi pesan.

```python
def signing():
    print(" Sign Up  ".center(50,"-"))
    # loading data
    data = load_database()
    usernames = [ob["username"] for ob in data]

    username = input("Enter username    : ")
    username_hash = int(hashlib.sha256(username.encode()).hexdigest(), 16)
    while (username_hash in usernames):
        print("That username has already been taken, please enter a new username")
        username = input("Enter username    : ")
        username_hash = int(hashlib.sha256(username.encode()).hexdigest(), 16)

    password = input("Enter password    : ")

    keys = generate_random_key()
    n,e,d = keys
    signature = generate_signature(username,password,n,d)

    print()
    print("Generating keys....")
    print("Private key (d)  :",str(d))
    print("Public key (e)   :",str(e))
    print("Modulus          :",str(n))
    print()
    print("Please remember the keys generated to encrypt and decrypt in the future")

    print("To use features, please login to account")


    # updating data
    data.extend([{"id":len(data),"username" : username_hash,"signature":signature,"data":[]}])
    json_object = json.dumps(data, indent=4)
    with open("./database/data.json", "w") as outfile:
        outfile.write(json_object)
```

Gambar 4.18 Fungsi untuk signing
Sumber : Dokumentasi pribadi

V. HASIL UJI COBA

Penulis akan mencoba program dengan pertama membuat *signing* baru dan mencoba mengenkripsi dan mendekripsi sebuah pesan. Setelah selesai, penulis akan mencoba mengubah data yang tersimpan pada *database*. Penulis melakukan hal ini dengan tujuan mengetes program dalam hal pengecekan validasi integritas data. Berikut hasil uji coba yang dilakukan penulis :

1. membuat signing baru dan mencoba identifikasi pengguna

Pertama, penulis akan mencoba untuk membuat *signing* baru dan mengetes kemampuan identifikasi pengguna pada program. Berikut hasilnya:

```
================== Sign Up  ==================
Enter username    : marvel pangondian
Enter password    : this_is_my_life

Generating keys....
Private key (d)  : 163145
Public key (e)   : 6221
Modulus          : 483101

Please remember the keys generated to encrypt and decrypt in the future
To use features, please login to account
```

Gambar 5.1 Signing pada program
Sumber : Dokumentasi pribadi

```
==================== login  ====================
Enter username     : marvel pangondian
input password     : this_is_my_life
Input private key  : 123
Input public key   : 6221
modulus            : 483101
verification failed
```

Gambar 5.2 Gagal verifikasi, private key salah
sumber : Dokumentasi pribadi

```
==================== login  ====================
Enter username      : marvel pangondian
input password      : this_is_WHAT
Input private key   : 163145
Input public key    : 6221
modulus             : 483101
verification failed
```

Gambar 5.3 Gagal verifikasi, password salah
Sumber : Dokumentasi pribadi

```
==================== login  ====================
Enter username      : marvel pangondian
input password      : this_is_my_life
Input private key   : 163145
Input public key    : 6221
modulus             : 483101
Success !
```

Gambar 5.4 Verifikasi berhasil
Sumber : Dokumentasi pribadi

2. percobaan enkripsi dan dekripsi

Setelah berhasil identifikasi pengguna, penulis mencoba berbagai tes mengenai enkripsi dan dekripsi. Pada salah satu tes, penulis mengubah data yang terdapat di dalam database.

```
================== main menu ==================
1.Login
2.Logout
3.Sign up
4.Encryption
5.Decryption
6.End program
Choice : 4

================= Encryption ==================
Enter your log : I love you mom !

Encryption successful !
```

Gambar 5.5 Enkripsi pesan pertama
Sumber : Dokumentasi pribadi

```
================== main menu ==================
1.Login
2.Logout
3.Sign up
4.Encryption
5.Decryption
6.End program
Choice : 4

================= Encryption ==================
Enter your log : this will be corrupted later :)

Encryption successful !
```

Gambar 5.6 Enkripsi pesan kedua
Sumber : Dokumentasi pribadi

```
================== main menu ==================
1.Login
2.Logout
3.Sign up
4.Encryption
5.Decryption
6.End program
Choice : 5

================= Decryption  =================
Select entry (2 entries) : 1
Decrypted Text :
I love you mom !
```

Gambar 5.7 Dekripsi pesan pertama
Sumber : Dokumentasi pribadi

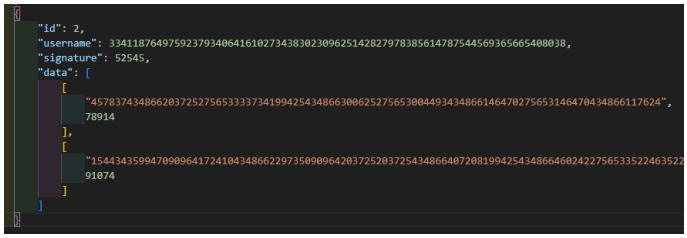
```
================== main menu ==================
1.Login
2.Logout
3.Sign up
4.Encryption
5.Decryption
6.End program
Choice : 5

================= Decryption  =================
Select entry (2 entries) : 2
Decrypted Text :
this will be corrupted later :)
```

Gambar 5.8 Dekripsi pesan kedua
Sumber : Dokumentasi pribadi

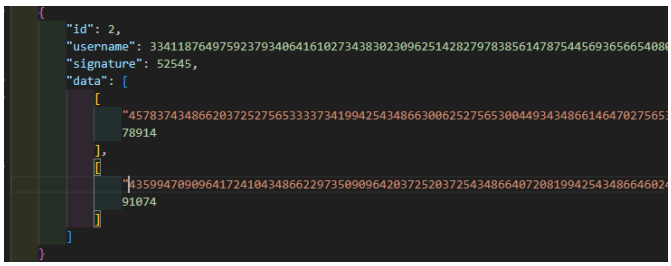
```
================== main menu ==================
1.Login
2.Logout
3.Sign up
4.Encryption
5.Decryption
6.End program
Choice : 5

================= Decryption  =================
Select entry (2 entries) : 2
WARNING, DATA INTEGRITY HAS BEEN COMPROMISED!
CORRUPTED TEXT :
u4→!C◀5,P5k{3P(]↓FO=*↓↓);♠"
```

Gambar 5.9 Dekripsi pesan kedua setelah diubah isi datanya
Sumber : Dokumentasi pribadi

Pada gambar 5.9, penulis menggantikan isi dari *cipher text* pada *database*. Penulis menghapus 5 angka pertama pada *cipher text* sehingga membuat *cipher text* terkorupsi. Berikut adalah sebelum dan sesudah data diubah pada *database* :



Gambar 5.10 Data sebelum diubah
Sumber : Dokumentasi pribadi



Gambar 5.11 Data setelah diubah
Sumber : Dokumentasi pribadi

Melalui uji coba ini, dapat terlihat bahwa metode menggunakan *digital signature* merupakan metode yang efektif dalam memvalidasi integritas data identifikasi pengguna. Implementasi *digital signature* menggunakan kombinasi algoritma SHA-256 dan RSA merupakan implementasi yang efektif karena dapat menghasilkan *signature* yang unik, rentan *collision*, serta efektif dalam memvalidasi integritas data.

VI. KESIMPULAN

Data pribadi merupakan aset yang memerlukan perlindungan optimal. Salah satu pendekatan untuk menjaga keamanannya adalah melalui kriptografi yang merupakan metode mengubah data, biasanya teks biasa, menjadi teks sandi yang tidak dapat dibaca. Meskipun demikian, risiko perubahan oleh pihak ketiga masih dapat terjadi terhadap teks sandi tersebut. Untuk menanggulangi risiko ini, penggunaan tanda tangan digital muncul sebagai solusi efektif. Penerapan tanda tangan digital, terutama dengan menggunakan kombinasi algoritma SHA-256 dan RSA, membentuk suatu metode yang mampu memvalidasi integritas data (teks sandi) yang telah disimpan. Melalui mekanisme ini, kita dapat dengan cepat mendeteksi apakah data telah mengalami perubahan oleh pihak yang tidak sah. Dengan demikian, penggunaan tanda tangan digital tidak hanya meningkatkan integritas data, tetapi juga memperkuat keamanan secara keseluruhan.

VII. UCAPAN TERIMA KASIH

Puji syukur penulis panjatkan pada Tuhan Yang Maha Esa atas rahmatnya yang berlimpah sehingga penulis dapat menyelesaikan karya tulis ilmiah yang berjudul “Memvalidasi Integritas Data dan Mengidentifikasi Pengguna melalui Kombinasi Algoritma RSA dan SHA-256”. Karya tulis ini bertujuan memenuhi salah satu tugas pada mata kuliah Matematika Diskrit semester ganjil Tahun 2023/2024 Jurusan Teknik Informatika, Institut Teknologi Bandung. Penulis juga mengucapkan terima kasih kepada dosen mata kuliah IF2120 Matematika Diskrit penulis yaitu Dr. Fariska Zakhralativa Ruskanda, S.T., M.T. yang telah membimbing penulis dalam menulis karya ilmiah ini. Penulis juga mengucapkan terima kasih kepada orang tua dan teman-teman penulis yang sudah mendukung penulis dalam menyelesaikan karya ilmiah ini.

REFERENCES

[1] Dale Liu, Caceres, M., Robichaux, T., Forte, D. v., Seagren, E. S., Ganger, D. L., Smith, B., Jayawickrama, W., Stokes, C., & Jan Kanclirz, Jr. (2009). Chapter 3 - An Introduction To Cryptography. In Next Generation SSH2 Implementation.

[2] https://www.shiksha.com/online-courses/articles/types-of-cryptography/ diakses pada 10 desember 2023

[3] H Mendel, F., Pramstaller, N., Rechberger, C., Rijmen, V. (2006). Analysis of Step-Reduced SHA-256. In: Robshaw, M. (eds) Fast Software Encryption. FSE 2006. Lecture Notes in Computer Science, vol 4047. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11799313_9

[4] https://informatika.stei.itb.ac.id/~rinaldi.munir//Matdis/2023-2024/14-Teori-Bilangan-Bagian1-2023.pdf , diakses pada 10 Desember 2023

[5] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/15-Teori-Bilangan-Bagian2-2023.pdf , diakses pada 10 Desember 2023

[6] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/16-Teori-Bilangan-Bagian3-2023.pdf , diakses pada 10 Desember 2023

[7] Ikbal, J. (2007). *An introduction to cryptography*. *Information Security Management Handbook, Sixth Edition*. https://doi.org/10.2307/2695435’

[8] https://www.encryptionconsulting.com/education-center/what-is-rsa/

[9] https://stackoverflow.com/questions/53899931/why-does-an-empty-string-in-python-sometimes-take-up-49-bytes-and-sometimes-51 , diakses pada 10 Desember 2023

[10] https://www.ibm.com/docs/en/b2badv-communication/1.0.0?topic=overview-digital-signature , diakses pada 11 Desember 2023

[11] https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm , diakses pada 11 Desember 2023

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, 11 Desember 2023

Marvel Pangondian, 13522075

LAMPIRAN

Untuk uji coba dan pengembangan lebih lanjut, berikut adalah laman *github* penulis :
https://github.com/MarvelPangondian/Simple-Data-Integrity-and-User-Identification-Program