Implementasi Fungsi RSA dan SHA-3 (Keccak) untuk Enkripsi dan Tanda Tangan Digital pada Dokumen Rekening Koran

Ilmagita Nariswari (18221101)
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): ilmagita@gmail.com

***Abstract*—Dalam era digital, keamanan dokumen rekening koran menjadi sangat penting untuk melindungi data pribadi dan finansial nasabah. Dokumen rekening koran umumnya hanya diamankan menggunakan tanggal lahir nasabah. Hal ini menimbulkan masalah, karena tanggal lahir merupakan suatu hal yang dapat diketahui oleh banyak orang. Oleh karena itu, makalah ini mengusulkan penggunaan algoritma RSA dan fungsi hash SHA-3 (Keccak) untuk meningkatkan keamanan dokumen rekening koran yang dikirim melalui email. Kriptografi RSA dibangkitkan menggunakan tanggal lahir, dan juga digunakan untuk memberikan tanda tangan digital. RSA digunakan untuk enkripsi, sementara SHA-3 digunakan untuk tanda tangan digital, memastikan integritas dan non-repudiation dari dokumen. Implementasi menunjukkan bahwa kombinasi RSA dan SHA-3 efektif dalam melindungi informasi sensitif dari ancaman keamanan.**

Kata kunci: RSA, SHA-3, enkripsi, tanda tangan digital, keamanan dokumen, rekening koran.

I. Pendahuluan

Dokumen rekening koran adalah dokumen yang berisikan rangkuman dari informasi mengenai debit atau kredit dari suatu rekening yang dimiliki oleh nasabah. Umumnya, dokumen rekening koran dikirimkan oleh bank kepada nasabah setiap bulannya sebagai rekapitulasi alur uang keluar-masuk rekening. Oleh karena itu, keamanan dokumen ini menjadi krusial untuk melindungi data pribadi dan finansial nasabah. Di era digital, pengiriman dokumen rekening koran melalui email telah menjadi praktik umum. Namun, keamanan dokumen digital masih rentan terhadap berbagai ancaman.

Pada zaman digital ini, rekening koran telah dikirimkan melalui surat elektronik (*e-mail*). Dalam pengiriman, dokumen dikunci dengan kata sandi yang umumnya berupa tanggal lahir nasabah. Untuk membuka dokumen tersebut, seseorang cukup menerima dokumen yang terenkripsi dan menginput suatu kata sandi. Biasanya, kata sandi yang diberikan bank adalah tanggal lahir. Setelah memasukkan kata sandi tersebut, dokumen dapat terbuka dan dibaca oleh siapa pun.

Permasalahan muncul apabila hanya kata sandi yang digunakan untuk mengenkripsi dokumen tanpa ada lapisan keamanan tambahan. Tanggal lahir nasabah dapat mudah diketahui oleh banyak orang, sehingga dokumen ini rawan dibuka oleh pihak selain nasabah. Hal ini menjadi masalah, karena dokumen berisikan informasi sensitif yang dapat disalahgunakan. Oleh karena itu, pada makalah ini, akan diusulkan sebuah kriptosistem untuk dokumen rekening koran menggunakan algoritma RSA, dengan kunci yang digunakan merupakan tanggal lahir nasabah.

II. Dasar Teori

A. Dokumen Rekening Koran

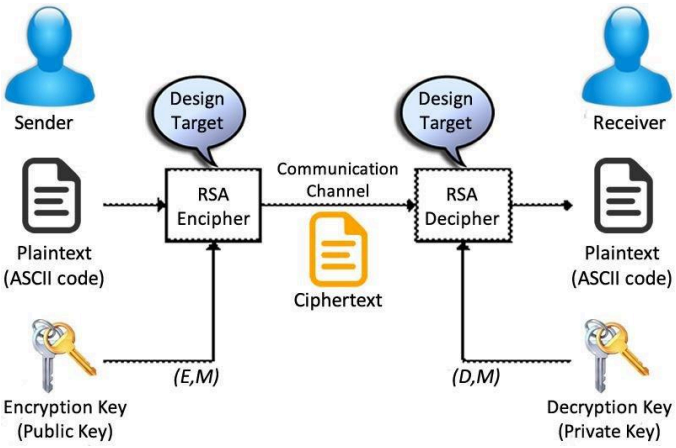
Dokumen rekening koran merupakan dokumen ringkasan yang dikirimkan pada setiap nasabah suatu bank setiap bulannya. Pada beberapa bank di Indonesia, dokumen ini disebut *consolidated statement.* Pengertian *consolidated statement* oleh Bank Mandiri yaitu ‘penggabungan seluruh informasi dana, pinjaman, dan investasi rekening koran milik nasabah yang dikirimkan melalui sarana email sehingga informasi dapat terkirim secara cepat, tepat dan aman.’ [1] Dengan adanya *consolidated statement*, nasabah dapat mengetahui seluruh dana yang ia miliki di bank, melihat saldo keseluruhan, dan melihat detail transaksi.

Dokumen rekening koran dianggap sebagai sangat rahasia karena di dalamnya terdapat banyak informasi. Pertama, terdapat informasi alamat sesuai yang terdata di bank, nomor rekening, dan saldo rekening. Selain itu, terdapat juga setiap transaksi finansial yang terjadi dalam satu bulan beserta nominal. Jika terjadi kebocoran atau dokumen rekening koran dibuka oleh pihak-pihak yang tidak bertanggung jawab, seorang nasabah menjadi rentan terhadap serangan penipuan atau risiko keamanan informasi lainnya.

Kebocoran dari dokumen rekening koran dapat menimbulkan masalah untuk nasabah dan bank. Reputasi bank dapat menjadi buruk dengan adanya kebocoran informasi. Selain itu, informasi yang ada di dalam rekening koran

bersifat sensitif dan dapat digunakan untuk hal-hal yang merugikan seperti penipuan atau penyalahgunaan identitas. Oleh karena itu, perlu adanya lapisan keamanan tambahan selain menggunakan kata sandi berupa tanggal lahir untuk mengamankan dokumen ini.

B. Algoritma RSA



Gambar 1 Ilustrasi Skema Algoritma RSA. Sumber: Secure SMS Encryption Using RSA Encryption Algorithm on Android Message Application

Algoritma RSA, atau Rivest-Shamir-Aldeman, adalah algoritma yang diusulkan pertama kali oleh para penulisnya yaitu Ron Rivest, Adi Shamir and Leonard Adleman di makalahnya yang berjudul "*A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*" yang rilis pada tahun 1977. Sebelumnya, untuk mengirimkan pesan rahasia, pengirim dan penerima pesan harus menggunakan kunci yang sama agar dapat melakukan enkripsi dan dekripsi pada pesan. Hal ini menimbulkan masalah baru, karena kunci tersebut harus dirahasiakan dan sulit untuk dikirimkan secara rahasia. Idealnya, kunci harus dikirimkan melalui suatu saluran yang aman dan rahasia. Akan tetapi, pada kenyataannya, saluran pengiriman kunci ini umumnya sangat mahal dan sulit untuk diimplementasikan.

Kriptografi kunci publik hadir untuk menyelesaikan masalah tersebut. Konsep kriptografi kunci publik pertama kali diusulkan dalam makalah yang berjudul '*New Directions in Cryptography*' oleh Whitfield Diffie dan Martin E. Hellman pada tahun 1976. Kriptografi kunci publik, atau nama lainnya kriptografi asimetris, berbeda dari kriptografi simetris karena penggunaan kunci enkripsi dan kunci dekripsi yang berbeda diterapkan. Seseorang dapat mengirimkan pesan yang terenkripsi menggunakan kunci enkripsi, yang selanjutnya disebut sebagai kunci publik. Kunci publik dapat dilihat oleh siapa saja. Selanjutnya, pesan terenkripsi ini dikirimkan, dan pesan ini dibuka oleh penerima menggunakan kunci dekripsi atau kunci privat penerima, yang secara teoritis seharusnya tidak diketahui oleh siapa pun kecuali penerima[2].

Adanya kriptografi asimetris ini berbuahkan berbagai kelebihan dan kekurangan. Kebutuhan untuk mengirimkan kunci pada saluran yang aman ditiadakan karena kunci publik telah dirilis ke publik, sementara kunci yang digunakan untuk mendekripsi hanya diketahui oleh sang penerima. Jadi, hanya kunci privat yang perlu dijaga kerahasiaannya. Akan tetapi, enkripsi dan dekripsi menggunakan kriptografi asimetris umumnya memakan waktu lebih lama dan daya komputasi yang lebih besar. Hal ini dikarenakan enkripsi dan dekripsi menggunakan bilangan-bilangan yang berukuran besar.

Algoritma RSA adalah salah satu algoritma yang memanfaatkan konsep kunci asimetris ini. Secara garis besar, algoritma RSA memiliki empat tahapan yaitu:

1. Pembangkitan kunci
    a. Pilihlah dua bilangan prima, yaitu p dan q. p tidak boleh sama dengan q, dan p dan q dirahasiakan.
    b. Hitung $n = p * q$.
    c. Hitung fungsi totien Carmichael, yaitu $\phi(n) = (p - 1) * (q - 1)$.
    d. Pilih bilangan bulat e dengan syarat $1 < e < \phi(n)$, serta e relatif prima terhadap $\phi(n)$ (FPB dari e dan $\phi(n) = 1$). e akan menjadi bagian dari kunci publik.
    e. Hitung kunci dekripsi d , yaitu d yang memenuhi persamaan $ed \equiv 1 \pmod{\phi(n)}$.

    Hasil dari pembangkitan kunci di atas adalah kunci publik yaitu (e, n) dan kunci privat yaitu (d, n).

2. Pendistribusian kunci

    Misalkan terdapat penerima bernama Alice dan pengirim bernama Bob, dan Bob ingin mengirimkan pesan terenkripsi kepada Alice, sebelumnya Alice harus memberikan kunci publiknya kepada Bob. Kunci privat Alice sendiri tidak dapat disebarkan.

3. Enkripsi

    Pengenkripsian dilakukan pada pesan m dengan kunci publik (e, n) dengan persamaan berikut:

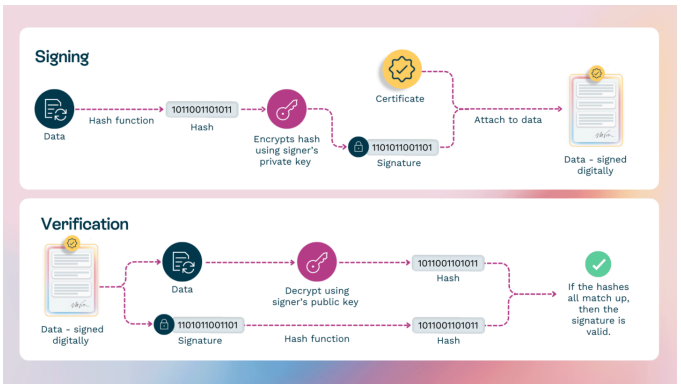
$$c = m^e \bmod n$$

4. Dekripsi

    Dekripsi dilakukan pada pesan terenkripsi dengan kunci privat (d, n) dengan persamaan berikut:

$$m = c^d \bmod n$$

C. *Tanda Tangan Digital*



Gambar 1 Skema tanda tangan digital. Sumber: https://oneflow.com/digital-signatures/

Meski RSA telah banyak diakui, ketidakrahasiaan kunci publik mengakibatkan kekhawatiran baru, karena pembuktian dan pengotentikasian pengirim menjadi sulit. Oleh karena itu, tanda tangan digital hadir sebagai salah satu elemen yang dapat membuktikan keaslian pengirim. Tanda tangan digital menggabungkan konsep kriptografi asimetri dan fungsi *hash*. Untuk membuktikan keaslian pengirim, tanda tangan bergantung pada isi pesan dan kunci. Ada dua proses utama dalam tanda tangan digital, yaitu *signing* dan *verifying*.

1. *Signing*

   Dalam proses *signing*, sebuah pesan akan dimasukkan ke dalam fungsi *hash*. Setelah itu, keluaran *hash* akan dienkripsi menggunakan kunci privat sang pengirim. Penggunaan kunci privat akan menjamin keaslian pengirim, karena kunci privat hanya diketahui oleh sang pengirim. *Hash* yang telah dihasilkan dan dienkripsi kemudian akan dikirim ke penerima bersamaan dengan pesan asli. Berikut merupakan persamaan dari tanda tangan digital (*S*) semisalkan kunci privat pengirim dinotasikan dengan (*d, n*), dan pesan yang dimasukkan ke dalam fungsi *hash* adalah *h(M)*.

$$S = (h(M))^d \bmod n$$

2. *Verifying*

   Untuk memverifikasi pengirim, penerima mendapatkan pesan asli dan pesan yang telah ter-*hash* dan terenkripsi. Penerima kemudian akan melakukan dekripsi dengan kunci publik penerima kepada pesan yang telah ter-*hash* dan terenkripsi. Berikut merupakan persamaan dari proses verifikasi tanda tangan dari penerima, semisalkan *h(M)'* adalah pesan *hash* hasil dekripsi, dan kunci publik pengirim dinotasikan dengan *(e, n)*.

$$h(M)' = S^e \bmod n$$

   Di sisi yang sama, penerima juga melakukan *hashing* terhadap pesan asli yang diterima. Pesan yang di-*hash* oleh penerima dinotasikan sebagai *h(M)*. Sehingga, apabila *h(M)'* merupakan pesan *hash* hasil dekripsi, dan

$$h(M)' = h(M)$$

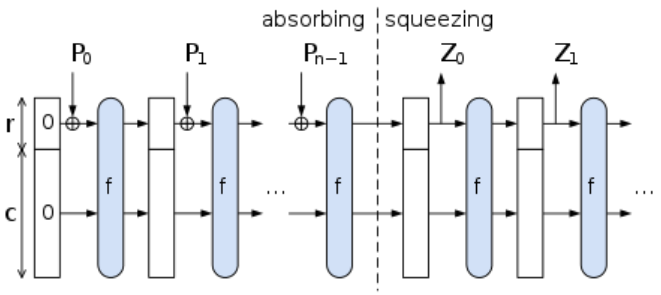
   maka, pengirim dikatakan terbukti keasliannya.

D. *Fungsi SHA-3 (Keccak)*

Pada makalah ini, fungsi hash yang dipakai adalah fungsi hash SHA-3. Fungsi *hash* adalah fungsi yang melakukan kompresi pada suatu pesan berukuran sembarang sehingga pesan tersebut menjadi *string* yang berukuran kecil dan tetap. Hasil *string* ini adalah tetap dan tidak dapat dikembalikan ke semula. Dalam bidang keamanan informasi, fungsi *hash* digunakan untuk memverifikasi keaslian pengirim pesan. Terdapat banyak fungsi *hash* yang telah ditetapkan sebagai standar oleh NIST, seperti SHA-1, SHA-2, dan SHA-3.

SHA-3 merupakan anggota terbaru dari standar *hash* Secure Hash Algorithm yang dipublikasikan National Institute of Standards and Technology. SHA-3 diusulkan dengan nama asli Keccak oleh Guido Bertini, Joan Daemen, Michael Peters, Gilles Van Assche, dan Ronny Van Keer. SHA-3 sendiri menggunakan konsep konstruksi 'spons'. Dengan konstruksi 'spons', data pertama-tama di-'serap' pada fase *absorption* lalu di-'kompres' layaknya spons pada fase *squeezed* [4]. Semisalkan pesan input *M* ingin dijadikan sepanjang *d* bit dan panjang blok masing-masing *r* bit. Pertama-tama, pesan input *M* dilakukan *padding* atau ditambahkan bit-bit pengganjal menggunakan suatu fungsi *pad*. Fungsi *pad* yang digunakan adalah 10*1, yaitu bit 1, dilanjutkan dengan bit 0 sebanyak maksimal (*r* - 1) bit, dan bit 1 lagi. Fungsi ini akan menghasilkan sebuah pesan *P* yang memiliki panjang yang dapat habis dibagi bulat dengan *r*. Setelah pesan *P* dihasilkan, *P* akan dipotong-potong menjadi blok-blok P_i berukuran *r* bit. Selanjutnya, sebuah *b*-bit dari peubah status (*state*) S diinisialisasi menjadi bit nol semua, dan fase penyerapan (*absorption*) dimulai.

Pada fase penyerapan (*absorption*), setiap blok P_i akan dilakukan XOR dengan *r*-bit pertama dari *state* S, lalu hasil dari XOR ini akan dimasukkan ke dalam fungsi permutasi *f* untuk menghasilkan *bit state* S yang baru. Setelah semua blok P_i telah dilakukan XOR dengan *state* S, fase penyerapan selesai, dan fase pemerasan dimulai. Pada fase pemerasan (*squeezing*), sebuah string kosong *Z* diinisialisasikan. Selama panjang *Z* lebih pendek dari *d*, *r* bit pertama dari *S*, fungsi permutasi *f* akan diaplikasikan pada *S*, menghasilkan *state S* yang baru. Terakhir, panjang *Z* akan dipotong menjadi sepanjang *d* bit.



Gambar 2 Ilustrasi Konstruksi Spons untuk Fungsi Hash.
Sumber: http://sponge.noekeon.org/

III. RANCANGAN

Berikut merupakan rancangan untuk program yang akan diimplementasikan.

A. Asal Pembangkitan Kunci RSA

Untuk membangkitkan kunci RSA, diperlukan bilangan *p* dan *q* yang merupakan bilangan prima berukuran besar. Dalam kriptosistem ini, angka *p* dan *q* dipilih berdasarkan tanggal lahir pengguna dengan rincian sebagai berikut:

1. Pengguna memasukkan tanggal lahir dengan format DD-MM-YYYY
2. Tanggal lahir akan dijadikan dasar dari pembangkitan kunci. Misalkan pengguna lahir pada tahun 1 Januari 1970, kunci yang dimasukkan adalah 01011970 atau setelah membuang angka 0 di depan menjadi 1011970. Angka dari tanggal lahir ini disebut sebagai *A*.
3. Apabila *A* bukan merupakan bilangan prima, akan dicarikan bilangan prima selanjutnya yang paling terdekat dengan *A*, hingga *A* ← bilangan prima dari tanggal lahir atau bilangan prima terdekat selanjutnya. Dalam hal ini, bilangan prima terdekat selanjutnya dari 1011970 adalah 1011973.
4. Selanjutnya, akan diperiksa apabila kebalikan dari *A* juga bilangan prima atau bukan. Apabila kebalikan dari *A* bukan bilangan prima, akan dicarikan kembali bilangan prima terdekat selanjutnya. Dalam contoh dari langkah sebelumnya, kebalikan dari *A* adalah 3791101, yang merupakan bilangan prima.
5. *A* akan dimasukkan ke dalam bilangan *p* untuk pembangkitan kunci RSA, dan kebalikan dari *A* dimasukkan ke dalam bilangan *q* untuk pembangkitan kunci RSA.
6. Sehingga, untuk tanggal kelahiran 1 Januari 1970, *p* adalah 1011973 dan *q* adalah 3791101.

B. Modul-modul

Modul-modul dibagi menjadi tiga, yaitu modul *hash* dan algoritma RSA. Modul-modul diprogram menggunakan Python.

1) Modul hash

Modul *hash* menggunakan *library* `hashlib` dari Python. *Hashing* dilakukan secara langsung pada file biner yang dipetakan menjadi kode American Standard Code for Information Interchange (ASCII).

```
import hashlib

def hash_binary_file(file_path):
        with open(file_path, 'rb') as file:
        binary_data = file.read()

        ascii_text = binary_data.decode('ascii',
errors='ignore')

        hash_obj = hashlib.sha3_256()

hash_obj.update(ascii_text.encode('utf-8'))
        hashed_result = hash_obj.digest()

        hashed_result_int_array = [byte for byte
in hashed_result]
        return hashed_result_int_array
```

2) Modul algoritma RSA

Modul algoritma RSA digunakan untuk membangkitkan kunci RSA dan diimplementasikan sendiri.

```
import sympy
import random
from math import gcd

## HELPER FUNCTIONS
def generate_large_prime():
        while True:
        number = random.randint(1, 9999999999)
        if sympy.isprime(number):
        return number

def next_prime(n):
        """Generate the next closest prime
number if n is not a prime."""
        if n <= 1:
        return 2
        prime = n
        found = False

        if sympy.isprime(prime):
        found = True

        while not found:
        prime += 1
        if sympy.isprime(prime):
        found = True
```

```
        return prime

def reverse_number(number):
        reversed_str = str(number)[::-1]
        reversed_number = int(reversed_str)
        return reversed_number

## RSA FUNCTIONS
def generate_public_key(p, q, e):
        return (e, p * q)

def generate_private_key(p, q, e):
        totient = (p - 1) * (q - 1)
        d = sympy.mod_inverse(e, totient)
        return (d, p * q)

def is_relative_prime(a, b):
        return gcd(a, b) == 1

def generate_rsa_key(birthdate=27022003):
        birthdate = next_prime(birthdate)
        reversed_birthdate                    =
next_prime(reverse_number(birthdate))
        p = birthdate
        q = reversed_birthdate

        while p == q:
        q = generate_large_prime()

        n = p * q
        totient = (p - 1) * (q - 1)

        e = generate_large_prime()
        while not is_relative_prime(e, totient):
        e = generate_large_prime()

        pubKey = generate_public_key(p, q, e)
        privKey = generate_private_key(p, q, e)

        return {
        "pubKey":     {"e":     pubKey[0],     "n":
pubKey[1]},
        "privKey":    {"d":     privKey[0],    "n":
privKey[1]}
        }
```

Selain itu, terdapat juga kode untuk melakukan enkripsi dan dekripsi *file*.

```
from ciphers import rsa
import os
import base64

def write_keys_to_files(keys):
        pub_key = keys["pubKey"]
        priv_key = keys["privKey"]
        e = pub_key['e']
        n_pub = pub_key['n']

        d = priv_key['d']
        n_priv = priv_key['n']

        with open('./keys/key.pub', 'w') as
pub_file:
        pub_file.write(f"({e}, {n_pub})")

        with open('./keys/key.pri', 'w') as
priv_file:
        priv_file.write(f"({d}, {n_priv})")

def utf8_to_base64(utf8_text):
        return
base64.b64encode(utf8_text.encode("utf-8")).deco
de("utf-8")

def base64_to_utf8(base64_text):
        utf8_text =
base64.b64decode(base64_text).decode("utf-8")
        return utf8_text

def binary_data_to_int_array(binary_data):
        array_of_integers = [int(byte) for byte in
binary_data]
        return array_of_integers

def int_array_to_binary_data(array_of_integers):
        binary_data = bytes(array_of_integers)
        return binary_data

## ENCRYPTION AND DECRYPTION
def encrypt(message, d, n):
        return ''.join(hex(pow(ord(m), d, n)) for m
in message)

def decrypt(cipher, e, n):
        cipher = [int(x, 16) for x in
cipher.split('0x')[1:]]

        return ''.join(chr(pow(c, e, n)) for c in
cipher)

def encrypt_file(file_path, e, n):
        base_name, file_extension =
os.path.splitext(file_path)

        with open(file_path, 'rb') as file:
        plainfile = file.read()
        plain_array =
binary_data_to_int_array(plainfile)
        cipher = [hex(pow(num,e,n)) for num in
plain_array]
```

```
        with open(base_name + '_encrypted' +
file_extension, 'w') as encrypted_file:
        encrypted_bytes = ''
        for val in cipher:
        encrypted_bytes += str(val)

        encrypted_file.write(encrypted_bytes)

def decrypt_file(file_path, d, n):
        base_name, file_extension =
os.path.splitext(file_path)

        with open(file_path, 'r') as file:
        cipherfile = file.read()
        cipherfile = cipherfile.split('0x')
        cipherfile = cipherfile[1:]

        cipher = (int('0x'+content,16) for content
in cipherfile)
        plain = [(pow(num, d, n)) for num in
cipher]
        plain = int_array_to_binary_data(plain)

        with open(base_name + '_decrypted' +
file_extension, 'wb') as decrypted_file:
        decrypted_file.write(plain)
```

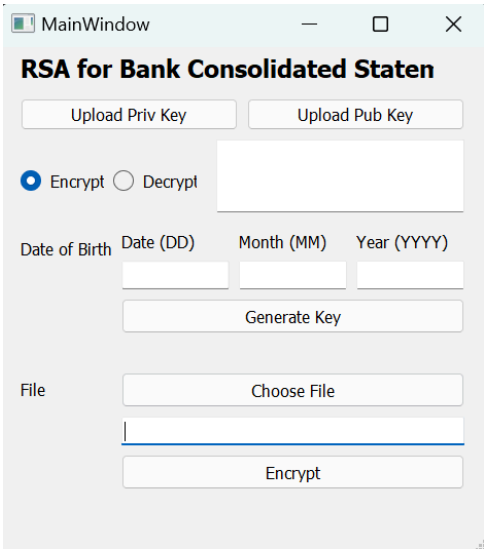
3) Modul tanda tangan

Modul tanda tangan digunakan untuk membuat tanda tangan dan melakukan verifikasi terhadap tanda tangan.

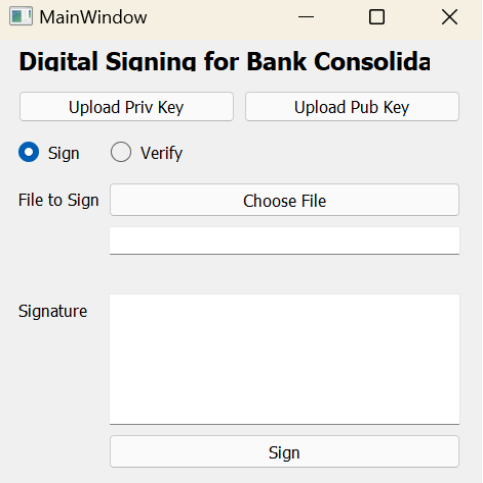
```
def sign_file(file_path, d, n):
        hashed_int_array =
hash_binary_file(file_path)

        signature_array = []

        for ch in hashed_int_array:
        signature = pow(ch, d, n)
        signature_array.append(signature)

        base_name, _ = os.path.splitext(file_path)

        with open(base_name + '_signature.txt',
'w') as signature_file:
        signature_file.write(str(signature_array))

def verify_signature(file_path, signature_path, e,
n):
        hashed_int_array =
hash_binary_file(file_path)

        with open(signature_path, 'r') as
signature_file:
        content = signature_file.read()

        content = content.replace('[', '').replace(']',
'')
        signature_array = [int(byte) for byte in
content.split(',')]

        decrypted_array = []

        for ch in signature_array:
        decrypted = pow(ch, e, n)
        decrypted_array.append(decrypted)

        return hashed_int_array ==
decrypted_array
```

C. Tampilan Program

GUI program menggunakan PyQt5. Terdapat dua menu utama dari program, yaitu menu RSA dan menu penandatanganan. Menu RSA digunakan untuk membangkitkan kunci, mengenkripsi, dan mendekripsi suatu file biner. Sementara itu, menu penandatanganan digunakan untuk membangkitkan dan memverifikasi tanda tangan.



Gambar 3 Menu RSA

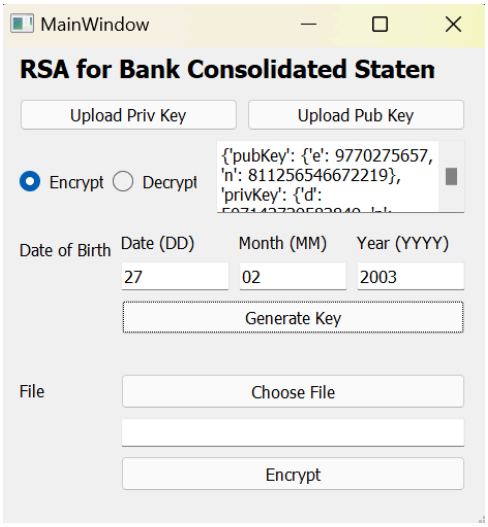


Gambar 4 Menu Tanda Tangan

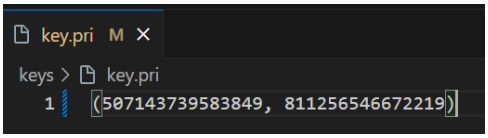
IV. Hasil Implementasi dan Pengujian

Berikut merupakan hasil pengimplementasian dan pengujian dari program.

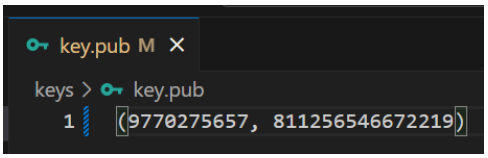
1. Pembangkitan kunci



Gambar 5 Menu RSA setelah menakan tombol 'Generate Key'



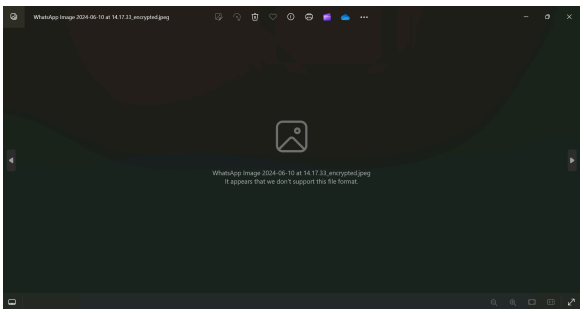
Gambar 6 Hasil Kunci Privat dalam *file* key.pri



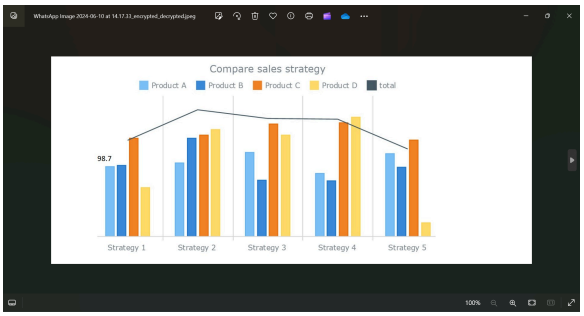
Gambar 7 Hasil Kunci Privat dalam *file* key.pub

2. Hasil pengenkripsian dan pendekripsian

Gambar 8 File sebelum dienkripsi

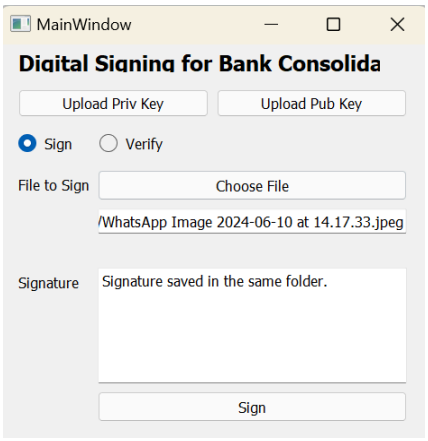


Gambar 9 File setelah dienkripsi



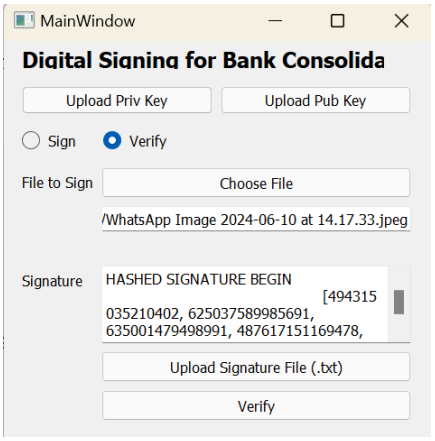
Gambar 10 File enkripsi yang didekripsi

3. Pembangkitan tanda tangan digital

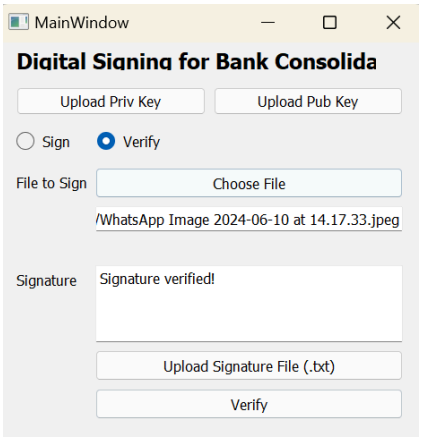


Gambar 11 Pembangkitan tanda tangan

4. Verifikasi tanda tangan digital



Gambar 12 Pembacaan tanda tangan dari *file*



Gambar 13 Hasil verifikasi tanda tangan

V. Kesimpulan dan Saran

Perlu dicatat bahwa dalam makalah ini, bilangan *p* dan *q* yang dipilih maksimal berukuran 8 digit sebagai contoh umum. Akan tetapi, Rindleman dkk. mengusulkan nilai *p* dan *q* masing-masing lebih dari 100 digit. Hasil kali *n* yaitu *p* * *q* untuk *p* dan *q* sebanyak 100 digit akan berukuran lebih dari 200 digit. Oleh karena itu, sebaiknya angka yang dipilih berdasarkan tanggal lahir dapat direplikasi sehingga angka *p* dan *q* masing-masing berukuran lebih dari 100 digit.

Berdasarkan penelitian dan percobaan yang dilakukan, penggunaan kriptografi RSA dan kunci SHA-3 akan membantu mengamankan informasi dokumen rekening koran. Adanya penggunaan kriptografi RSA dan tanda tangan digital akan memenuhi konsep *integrity* dan *non-repudiation* untuk dokumen rekening koran.

Link Kode Sumber

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

Video Link at Youtube

https://youtu.be/3zE2roFtevk

Acknowledgment

Penulis ingin mengucapkan rasa syukur kepada Allah SWT dan terima kasih kepada keluarga, Bapak Rinaldi Munir selaku dosen mata kuliah Kriptografi dan Koding, dan teman-teman sejurusan atas dukungannya selama di mata kuliah ini.

References

[1] Bank Mandiri, "Layanan E-Statement," https://www.bankmandiri.co.id/faq-layanan-e-statement , 2024. https://www.bankmandiri.co.id/faq-layanan-e-statement

[2] W. Diffie and M. Hellman, "New directions in cryptography," IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644–654, Nov. 1976, doi: https://doi.org/10.1109/tit.1976.1055638.

[3] K. Aggarwal and H. K. Verma, "Hash_RC6 — Variable length Hash algorithm using RC6," IEEE Xplore, Mar. 01, 2015. https://ieeexplore.ieee.org/document/7164747 (accessed Apr. 18, 2023).

[4] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assege, "Keccak Team," keccak.team, Jan. 14, 2011. https://keccak.team/sponge_duplex.html

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, 12 Juni 2024

Ilmagita Nariswari 18221101