Implementasi Algoritma RSA dan Hash SHA-256 untuk Tanda Tangan Digital dalam Membangkitkan Kode QR Akses Masuk Kampus

Justin Dermawan Ikhsan 18219095
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
18219095@std.stei.itb.ac.id

***Abstract*—Saat ini status pandemi COVID-19 mulai dilonggarkan. Kampus ITB sebagai salah satu bagian dari sektor pendidikan yang sebelumnya terdampak pandemi mulai menerapkan kembali pertemuan secara bauran dan luring. Untuk mendapatkan akses masuk kampus digunakan sistem yang memerlukan mahasiswa untuk memindai kode QR. Namun, sistem validasi saat ini bergantung pada jaringan internet. Sistem validasi ini sangat rentan karena jika terjadi gangguan internet maka tidak dapat dilakukan pendataan akses masuk kampus. Untuk itu pada makalah ini akan dibuat aplikasi untuk membangkitkan kode QR dan memverifikasi akses masuk kampus dengan menggunakan tanda tangan digital. Tanda tangan digital yang akan dikembangkan akan berbasis algoritma kunci publik RSA dan algoritma hash SHA-256. Implementasi algoritma yang digunakan sudah berhasil membangkitkan dan memverifikasi kode akses masuk kampus secara luring (tanpa jaringan internet) dengan kecepatan kurang dari satu detik..**

Keywords—Kode QR; Tanda tangan digital; RSA; SHA-256; Aplikasi; Performa

I. PENDAHULUAN

Saat ini pemerintah sudah mulai melonggarkan keketatan protokol kesehatan terkait COVID-19. Banyak sektor yang sudah mulai memperbolehkan kembali kegiatan secara luring. Salah satunya pada sektor pendidikan, kampus Institut Teknologi Bandung menjadi salah satu kampus yang sudah mulai menerapkan kegiatan belajar mengajar secara bauran maupun luring secara penuh.

Untuk tetap dapat menyaring akses mahasiswa ke kampus, ITB menerapkan peraturan kepada mahasiswanya untuk mengisi Aplikasi Mawas Diri (AMARI). Pengisian AMARI berguna untuk melakukan pencatatan terhadap akses masuk area kampus tertentu seperti gedung kelas atau laboratorium oleh mahasiswa.

Setelah mengisi AMARI, mahasiswa akan dikirimkan sebuah kode QR yang akan digunakan sebagai metode verifikasi untuk masuk ke area kampus. Saat ini kode QR yang dibangkitkan berisikan informasi pranala (url) yang akan diakses oleh perangkat sistem untuk memverifikasi akses mahasiswa tersebut.

Namun, hal ini sangat rentan karena proses verifikasi kode QR bergantung terhadap tersedianya jaringan internet pada perangkat sistem. Ketika terdapat ganggung internet maka kode QR yang dikirimkan menjadi tidak dapat digunakan. Maka dari itu, akan dikembangkan sebuah teknologi validasi kode QR akses kampus yang bisa dilakukan secara offline.

Teknologi yang akan digunakan adalah teknologi tanda tangan digital berbasis algoritma kunci publik RSA dan Hash SHA-256. Proses validasi kode QR dapat dilakukan langsung pada sistem tanpa harus mengakses server terlebih dahulu.

Dengan dikembangkannya teknologi ini diharapkan proses verifikasi akses menjadi lebih dapat diandalkan dan juga menjadi lebih aman. Penggunaan tanda tangan digital pada data yang disimpan dalam kode QR akan menjamin tidak ada perubahan ataupun kecurangan lainnya dalam pembangkitan kode QR.

II. DASAR TEORI

A. Kode QR

Kode QR (*Quick Response Code*) pertama kali dikembangkan oleh peneliti dari Jepang pada tahun 1994. Kode QR adalah pengembangan lebih lanjut dari kode batang (*barcode*). Kode QR memanfaatkan dua dimensi untuk menyimpan lebih banyak data [1].

Pada umumnya, kode QR terdiri dari kumpulan kotak persegi berwarna hitam yang disusun pada sebuah kisi (*grid*) berbentuk persegi dengan latar belakang putih. Namun, saat ini kode QR sudah berkembang dan dapat terdiri dari berbagai warna, berbagai bentuk (tidak harus terdiri dari kotak sempurna), dan bahkan dapat disisipkan gambar/logo.

Kode QR dapat menyimpan data dengan panjang yang bervariasi dan ketelitian yang bervariasi juga. Versi terakhir dari kode QR berukuran 177x177 dan dapat menampung data hingga lebih dari 1852 karakter. Kode QR juga memiliki tingkat ketelitian yang berbeda-beda. Pada kode QR ketelitian tinggi jika sebagian dari kode QR ada yang hilang/rusak data masih dapat dibaca [2].

B. Algoritma Kunci Publik RSA

Algoritma RSA(Rivest-Shamir-Adleman) adalah algoritma kriptografi kunci publik yang pengaplikasiannya sudah sangat luas terutama dalam mengamankan pengiriman data. Algoritma RSA diciptakan oleh tiga orang peneliti yaitu Ronald Rivest, Adi Shamir, dan Len Adleman [3].

Pada algoritma kunci publik enkripsi data dilakukan dengan menggunakan sebuah kunci publik yang dapat disebarkan ke siapapun khususnya pengirim data, sedangkan dekripsi data dilakukan dengan sebuah kunci privat yang hanya diketahui oleh pihak penerima data. Algoritma RSA memanfaatkan tingkat kerumitan dalam memfaktorkan dua bilangan prima yang sangat besar untuk menjaga kerahasiaan data [4].

Inti dari algoritma RSA merupakan penurunan dari Teorema Euler yaitu [3]:

$$a^{\phi(n)} \equiv 1(mod\ n)$$

Dengan syarat yaitu:

1. Nilai α harus relatif prima dengan n
2. *Totient Euler* atau $\phi(n)$ adalah fungsi yang menentukan jumlah bilangan yang relatif prima terhadap n.

Dalam implementasinya, algoritma RSA memiliki dua prosedur penting. Pertama terdapat prosedur pembangkitan kunci yang akan menghasilkan kunci privat dan kunci publik serta prosedur enkripsi dan dekripsi data.

Agar dapat melakukan enkripsi dan dekripsi data, maka diperlukan sepasang kunci yang akan menjadi salah satu masukan pada algoritma enkripsi dan dekripsi. Berikut merupakan tahapan prosedur pembangkitan kunci pada algoritma RSA [3]:

1) Pilih dua buah bilangan prima acak yang akan disimpan sebagai variabel p dan q. Nilai bilangan prima acak ini bersifat rahasia dan dicari nilai sebesar mungkin agar semakin sulit untuk ditebak.
2) Hitung nilai n dengan menggunakan rumus:

$$n = p * q$$

3) Lalu, hitung nilai *Totient Euler* dengan menggunakan rumus:

$$\phi(n) = (p-1) * (q-1)$$

Nilai *Totient Euler* ini merupakan properti yang bersifat rahasia.

4) Pilih sebuah nilai e yang akan menjadi kunci publik. Nilai e harus relatif prima dengan nilai *totient euler*. Relatif prima berarti nilai FPB dari *totient euler* dan e = 1.
5) Hitung nilai d yang akan menjadi kunci privat. Mencari nilai d dengan menggunakan rumus:

$$d \equiv e^{-1} mod\ \phi(n)$$

Dari algoritma tersebut, sudah berhasil dibangkitkan kunci publik dan kunci privat. Kunci publik terdiri dari (e,n) dan kunci privat terdiri dari (d,n) [3].

Prosedur selanjutnya pada algoritma RSA adalah prosedur enkripsi. Berikut merupakan tahapan prosedur enkripsi pada algoritma RSA [3]:

1. Pisahkan plainteks atau data yang akan dienkripsi menjadi blok-blok plainteks atau data.

$$m_1, m_2, m_3, \dots, m_i$$

Dengan syarat $0 \leq m_1 < n - 1$.

2. Hitung blok cipherteks c_i untuk setiap blok plainteks m_i dengan menggunakan kunci publik (e, n) dengan menggunakan rumus:

$$c_i = m_i^e\ mod\ n$$

Untuk melakukan dekripsi dari cipherteks kembali menjadi plainteks. Dapat digunakan algoritma yang sama dengan menggunakan kunci privat (d, n). Berikut merupakan tahapan prosedur dekripsi pada algoritma RSA [3]:

1. Pisahkan cipherteks yang akan didekripsi menjadi blok-blok pesan atau data.

$$c_1, c_2, c_3, \dots, c_i$$

2. Hitung blok plainteks m_i untuk setiap blok cipherteks c_i dengan menggunakan kunci privat (d, n) dengan menggunakan rumus:

$$m_i = c_i^d\ mod\ n$$

C. Algoritma Hash SHA-256

Algoritma atau fungsi hash adalah fungsi yang akan menerima pesan dengan panjang sembarang dan melakukan kompresi menjadi pesan dengan panjang yang tetap (*fixed*) [5].

Hasil dari algoritma hash disebut dengan message-digest atau hash value. Panjang dari message-digest relatif terhadap algoritma hash yang digunakan. Beberapa variasi panjang message-digest antara lain 128 bit, 224 bit, 256 bit, 512 bit, dan sebagainya.

Algoritma hash SHA-256 merupakan algoritma hash yang hasil pengembangan lanjutan dari algoritma Secure Hash Algorithm (SHA). SHA merupakan fungsi hash yang dikembangkan oleh lembaga NIST(National Institute of Standards and Technology) dan digunakan oleh lembaga DSS (Digital Signature Standard) [6].

Algoritma hash SHA-256 akan menerima masukan pesan dengan panjang maksimal yaitu 264-1 bit dan menghasilkan keluaran dengan panjang tetap yaitu 256 bit.

Berikut merupakan langkah-langkah dari algoritma hash SHA-256 [7]:

1. Penambahan *padding bits* atau bit pengganjal agar panjang pesan yang akan di-*hash* sesuai dengan standar yang digunakan oleh tiap fungsi *hash*.

2. Tambahkan *length bit* atau bit panjang pesan. *Length bit* memiliki panjang 64 bit. Inilah alasan kenapa, panjang pesan yang dapat di-*hash* dalam satu kali pemanggilan algoritma adalah 2^{64}-1.

3. Siapkan nilai *default* untuk penyangga (*buffer*). Nilai *buffer* dibangkitkan dari 32 bit pertama dari bagian pecahan akar 8 bilangan prima pertama. Terdapat 8 nilai penyangga yaitu:

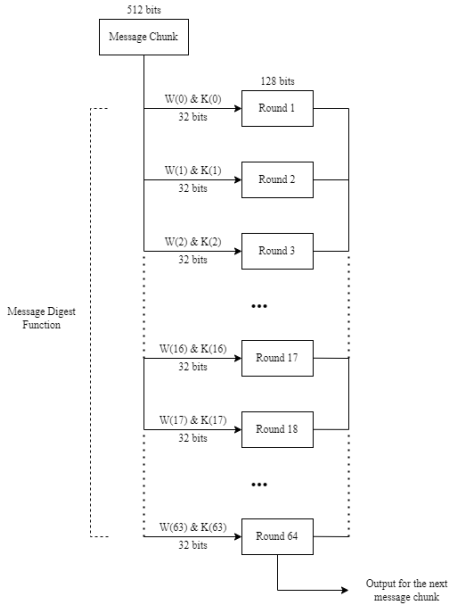
```
a = 0x6a09e667
b = 0xbb67ae85
c = 0x3c6ef372
d = 0xa54ff53a
e = 0x510e527f
f = 0x9b05688c
g = 0x1f83d9ab
h = 0x5be0cd19
```

Selain nilai penyangga, akan disiapkan juga nilai putaran konstan yang terdiri dari 32 bit pertama dari bagian pecahan dari akar pangkat tiga 64 bilangan prima pertama. Nilai putaran konstan tersebut terdiri dari:

```
k[0..63] = {0x428a2f98, 0x71374491,
0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b,
0x59f111f1, 0x923f82a4, 0xab1c5ed5,
0xd807aa98, 0x12835b01, 0x243185be,
0x550c7dc3, 0x72be5d74, 0x80deb1fe,
0x9bdc06a7, 0xc19bf174, 0xe49b69c1,
0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc,
0x76f988da, 0x983e5152, 0xa831c66d,
0xb00327c8, 0xbf597fc7, 0xc6e00bf3,
0xd5a79147, 0x06ca6351, 0x14292967,
0x27b70a85, 0x2e1b2138, 0x4d2c6dfc,
0x53380d13, 0x650a7354, 0x766a0abb,
0x81c2c92e, 0x92722c85, 0xa2bfe8a1,
0xa81a664b, 0xc24b8b70, 0xc76c51a3,
0xd192e819, 0xd6990624, 0xf40e3585,
0x106aa070, 0x19a4c116, 0x1e376c08,
0x2748774c, 0x34b0bcb5, 0x391c0cb3,
0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
0x748f82ee, 0x78a5636f, 0x84c87814,
0x8cc70208, 0x90befffa, 0xa4506ceb,
0xbef9a3f7, 0xc67178f2}
```

Nilai konstan ini akan dipanggil pada putaran yang dieksekusi pada tahap selanjutnya.

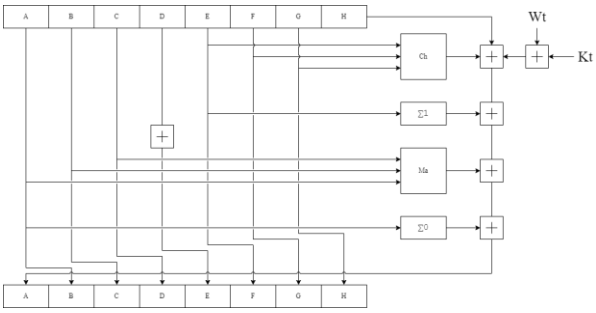
4. Selanjutnya pesan akan dibagi menjadi *message chunk* sepanjang 512 bit tiap pesan. Untuk tiap *message chunk* akan diputar sebanyak 64 putaran dan keluaran dari setiap putaran akan digunakan sebagai input putaran selanjutnya.



Gambar 1. Ilustrasi putaran untuk tiap *message chunk*.

Pada setiap putaran akan terdapat dua masukan yaitu W(i) dan K(i), untuk 16 ronde pertama pesan 512 bit akan dipisah menjadi 16 bagian dengan ukuran 32 bit. Untuk ronde selanjutnya nilai W(i) akan dihitung pada setiap langkah.

Berikut merupakan fungsi *message digest* yang dieksekusi pada tiap putaran.



Gambar 2. Ilustrasi fungsi yang dieksekusi untuk tiap ronde

Setiap putaran juga terdiri dari beberapa fungsi dasar. Fungsi dasar tersebut antara lain:

```
Ch(E, F, G) = (E AND F) XOR ((NOT E) AND G)
Ma(A, B, C) = (A AND B) XOR (A AND C) XOR (B AND C)
Σ(A)    = (A >>> 2) XOR (A >>> 13) XOR (A >>> 22)
Σ(E)    = (E >>> 6) XOR (E >>> 11) XOR (E >>> 25)
```

```
+     = addition modulo 2³²
```

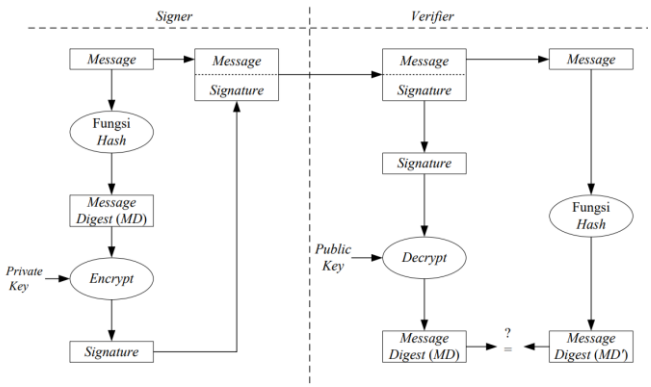
5. Keluaran dari setiap putaran akan menjadi masukan untuk proses selanjutnya. Hasil putaran terakhir akan menjadi *message digest* untuk keseluruhan pesan. Panjang dari *message digest* tersebut adalah 256 bit.

D. Tanda Tangan Digital

Tanda tangan digital adalah bentuk tanda tangan yang digunakan pada dokumen digital yang dapat memvalidasi konten dalam dokumen tersebut. Tanda tangan digital berbentuk nilai kriptografis yang bergantung dengan isi pesan atau dokumen dan kunci yang digunakan [8].

Nilai pada tanda tangan digital bergantung pada isi dari pesan atau dokumen yang ditandatangni. Jika terjadi perubahan sedikit saja pada pesan maka nilai tanda tangan digitalnya akan berbeda. Untuk menghasilkan tanda tangan digital yang unik dan aman akan digunakan kombinasi fungsi *hash* dan algoritma kriptografi kunci publik [8].

Secara garis besar proses pembangkitan tanda tangan digital dan verifikasi tanda tangan dilakukan dengan proses sebagai berikut:



Gambar 3. Ilustrasi alur pembangkitan dan verifikasi tanda tangan digital [8]

Saat menandatangani dokumen atau membangkitkan tanda tangan, pesan akan dimasukkan ke dalam suatu fungsi *hash*. *Message digest* yang dihasilkan akan dienkripsi dengan algoritma kriptografi kunci publik dan dihasilkan sebuah tanda tangan digital. Tanda tangan digital kemudian dapat disisipkan ke dalam pesan atau dokumen, atau juga dapat disimpan menjadi *file* terpisah [8].

Dalam memverifikasi isi pesan atau dokumen, untuk menjamin isinya tidak berubah pesan dan tanda tangan akan dipisahkan. Pesan akan dimasukan ke dalam fungsi *hash* sedangkan tanda tangan digital akan didekripsi dengan algoritma kriptografi kunci publik. Lalu akan dicek apakah hasil dekripsi tanda tangan dan hasil *hash* pesan memiliki nilai yang sama. Jika nilainya sama maka isi pesan tidak berubah [8].

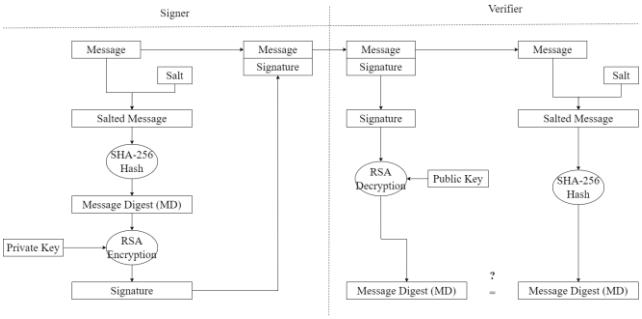
III. DESAIN DAN RANCANGAN

Sistem yang akan dikembangkan akan dapat membangkitkan dan memvalidasi kode QR dengan menggunakan tanda tangan digital. Pesan dan tanda tangan digital akan disisipkan kedalam kode QR.

A. Modifikasi Algoritma Tanda Tangan Digital

Algoritma yang akan digunakan untuk membangkitkan tanda tangan digital adalah kombinasi algoritma kriptografi kunci publik RSA dan algoritma fungsi *hash* SHA-256.

Untuk menambah tingkat keamanan dari tanda tangan digital, akan dilakukan modifikasi pada alur pembangkitan dan verifikasi tanda tangan digital. Modifikasi ini dilakukan dengan penambahan *salt* yang akan dikodekan secara manual dalam algoritma sistem. Penambahan *salt* ini akan menambah tingkat keteracakan hasil kode *hash*. Sehingga alur pembangkitan dan verifikasi tanda tangan digital menjadi sebagai berikut:



Gambar 4. Ilustrasi alur pembangkitan dan verifikasi tanda tangan digital dengan *salt*

B. Format Pesan

Data atau pesan yang akan disematkan dalam kode QR akan disusun dalam bentuk JSON (JavaScript Object Notation). JSON dipilih karena kemudahan dalam mengolah data dari yang awalnya bertipe *string* ke dalam tipe data *dictionary*. Pesan akan terdiri dari dua *key*/objek yaitu *key* data dan *key* tanda tangan digital. *Key* data akan terdiri dari dua *key* lainnya yaitu *key* NIM untuk menyimpan identitas mahasiswa, dan *key* yang akan menyimpan tanggal akses. Berikut merupakan rancangan format pesan yang akan disimpan pada kode QR.

```
{
   "data": {
        "nim": "18219095",
        "date": "2022-05-20"
   },
   "signature": "0808baeb2f95f1e69d..."
}
```

C. Pembangkitan Kode QR

Dari format pesan pada bagian sebelumnya, akan dibangkitkan sebuah kode QR yang jika dipindai oleh perangkat sistem akan dapat membaca pesan. Pesan yang akan dihasilkan akan memiliki panjang yang tetap. Total panjang

pesan adalah 144 karakter, yang terdiri dari: tanda tangan digital sepanjang 76 karakter (dihasilkan dari transformasi *byte* ke dalam *string hexadecimal*), data NIM sepanjang 8 karakter, data tanggal sepanjang 10 karakter, dan karakter tambahan pembentuk pesan sebanyak 50 karakter.

Kode QR yang dibangkitkan akan berukuran 45*45 blok. Dengan tingkat *error correction* yang digunakan yaitu tingkat rendah. Tingkat *error correction* ini dipilih karena semakin tinggi tingkat *error correction* maka ukuran kode QR akan semakin besar dan semakin rumit. Kerumitan kode QR dapat menambah waktu yang dibutuhkan oleh perangkat pemindai kode QR untuk membaca pesan pada kode QR. Tingkat *error correction* rendah juga dipilih karena kode QR akan dikirimkan secara digital ke mahasiswa sehingga kerusakan pada kode QR akan jarang terjadi.

IV. Implementasi Prototipe

Pada bagian ini dilakukan proses implementasi dari desain dan rancangan yang dibuat pada bagian sebelumnya. Implementasi akan dilakukan dengan menggunakan bahasa pemrograman Python. Bahasa pemrograman Python dipilih karena kemudahan dalam pengembangan dan tersedianya banyak sumber daya *online* sebagai bahan referensi. Dalam membuat antarmuka sistem akan digunakan *library* PyQt. *Library* ini dipilih karena kemudahan pengembangan antarmuka.

Implementasi yang masuk dalam lingkup pengembangan program dan antarmuka yaitu modul pembangkitan kode QR, modul pembangkitan tanda tangan, modul verifikasi kode QR, dan modul pembangkitan kunci RSA.

A. *Modul Pembangkitan Kode QR*

Modul pertama yang akan dikembangkan adalah modul untuk membangkitkan kode QR. Berikut merupakan potongan kode program pembangkitan kode QR:

```
def generate(NIM: str, Date:str, d: int, n: int) ->
str:
        data = {}
        data['data'] = {}
        data["data"]['nim'] = NIM
        data["data"]['date'] = Date
        data["signature"] =
ds.createSignature(json.dumps(data['data']),d,n)

        qr = qrcode.QRCode(
            version=1,

error_correction=qrcode.constants.ERROR_CORRECT_L,
            box_size=10,
            border=4,
        )
        qr.add_data(json.dumps(data))
        qr.make(fit=True)

        img = qr.make_image(fill_color=(0, 90, 171),
back_color=(256, 256, 256))

        filename =
f'generated_qr_{data["data"]["nim"]}_{data["data"]["dat
e"]}.png'
        img.save(filename)

    return filename
```

Kode Program 1. Potongan kode fungsi pembangkitan kode QR

Fungsi *generate* akan menerima empat buah masukan yaitu NIM bertipe *string*, Date bertipe *string*, pasangan kunci privat yaitu nilai *d* dan *n* bertipe *integer*. Jika fungsi berhasil dieksekusi akan mengembalikan nama *file* kode QR yang berhasil dibangkitkan dalam bentuk *string*. Pembangkitan kode QR pada fungsi ini memanfaatkan *library* qrcode. Berikut merupakan contoh kode QR yang berhasil dibangkitkan dengan menggunakan fitur tersebut:

Gambar 5. Kode QR yang dibangkitkan dengan masukan (“18219095”, “5/21/2022”, 1071952793644849, 1521351019836551)

B. *Modul Pembangkitan Tanda Tangan Digital*

Modul selanjutnya yang dikembangkan adalah modul pembangkitan tanda tangan digital. Fungsi pembangkitan tanda tangan digital ini akan dipanggil oleh modul pembangkitan kode QR untuk disisipkan dalam kode QR. Berikut merupakan potongan kode fungsi pembangkitan tanda tangan digital:

```
def baseHash(message: str)->bytes:
    return sha256(message.encode('utf-8')).digest()
def createSignature(text: str, d: int, n: int) -> str:
       try:
           secret_salt = "KRIPTOPOWERFULTOOL"
           md = baseHash(text + secret_salt)
           encryptedMd = encrypt(md, d, n)
           return encryptedMd.hex()
       except Exception as E:
            return E
```

Kode Program 2. Potongan kode fungsi pembangkitan tanda tangan digital

Fungsi *createSignature* akan menerima tiga buah masukan yaitu pesan yang akan dijadikan tanda tangan serta pasangan kunci privat yaitu nilai *d* dan *n* bertipe *integer*. Proses *hashing* yang dilakukan saat pembangkitan kunci akan memanfaakan fungsi sha256 yang disesdiakan oleh *library* Hashlib. Proses enkripsi akan memanfaatkan algoritma kriptografi kunci publik RSA yang sudah pernah penulis buat sebelumnya.

Fungsi ini akan mengembalikan hasil pembangkitan tanda tangan digital yang terdiri dari *string* dari hasil perubahan *byte* hasil enkripsi menjadi kode *hex*.

C. *Modul Verifikasi Kode QR*

Modul selanjutnya yang dikembangkan adalah modul verifikasi. Berikut merupakan potongan kode fungsi yang akan memverifikasi kode QR:

```
def verify(dir:str, d: int, n: int) -> int:
    data = decode(Image.open(dir))
    stringify_data = data[0].data.decode("utf-8")
    extracted = json.loads(stringify_data)
```

```
    verif =
v.verifySignature(json.dumps(extracted["data"]),extract
ed["signature"],d,n)

    if verif:
        getDateEnter =
datetime.strptime(extracted["data"]["date"], "%Y-%m-
%d")
        getToday =
datetime(year=datetime.today().year,month=datetime.toda
y().month,day=datetime.today().day)

        if getDateEnter < getToday:
                return 1
        else:
                return 2
    else:
         return 0
```

Kode Program 3. Potongan kode fungsi verifikasi kode QR

Fungsi *verify* akan menerima tiga buah masukan yaitu direktori dari file kode QR bertipe *string* serta pasangan kunci publik yaitu nilai *e* dan *n* bertipe *integer*. Proses ekstraksi pesan dari gambar kode QR pada fungsi ini memanfaatkan *library* pyzbar.

Fungsi ini juga akan memeriksa apakah sebuah kode QR valid atau tidak, jika terjadi perbedaan antara tanda tangan dan isi pesan akan dikembalikan nilai 0. Jika tanda tangan dan isi pesan sama, selanjutnya akan diperiksa apakah kode QR tersebut sudah kadaluarsa. Jika sudah kadaluarsa akan dikembalikan nilai 1 dan jika belum (kode QR masih valid) akan dikembalikan nilai 2.

D. Modul Pembangkitan Kunci RSA

Modul terakhir yang dikembangkan adalah modul pembangkitan kunci RSA. Modul ini akan menjadi modul pelengkap untuk aplikasi agar pengguna dapat membangkitkan kunci yang berbeda tergantung dengan kebutuhan (periode, perfakultas, dan aturan lainnya). Berikut merupakan potongan kode fungsi yang akan digunakan untuk membangkitkan kunci RSA:

```
def genKeys(p : int, q : int ,e : int) -> Tuple:
        d = 0
        if (checkPrime(p) and checkPrime(q)):
            n = p*q
            toit = (p-1)*(q-1)
            if (relativePrime(toit,e)):
                d = (pow(e, -1, toit))
                return (int(d),n)
            else:
                raise Exception("E dan Toitent tidak
relatif prima. Pilih E lain!")
        else:
         raise Exception("P atau Q bukan bilangan prima.
Pilih bilangan lain!")
```

Kode Program 4. Potongan kode fungsi pembangkitan kunci RSA

Fungsi *genKeys* akan menerima tiga buah masukan yaitu nilai *p*, *q* dan *e*. Ketiga nilai ini akan ditentukan sendiri oleh pengguna atau dapat dibangkitkan secara acak. Setelah itu akan diperiksa apakah nilai *p* dan *q* merupakan bilangan prima. Jika salah satunya bukan prima, program akan mengembalikan sebuah pesan *exception* yang akan dicetak ke layar. Sedangkan jika kedua bilangan *p* dan *q* merupakan prima, akan dihitung nilai *totient euler*-nya. Nilai *totient euler* akan dibandingkan dengan nilai *e* dan diperiksa apakah relatif prima. Jika tidak relatif prima akan dikirimkan *exception* yang akan dicetak ke layar. Selanjutnya program akan menghitung nilai *d* menggunakan fungsi *pow()* yang ada pada bahasa Python. Fungsi *pow()* sudah menerapkan algoritma Fermat untuk menghitung bilangan dengan pangkat yang sangat besar sehingga perhitungan menjadi lebih cepat. Fungsi ini juga dapat digunakan untuk mencari *inverse modulo* dengan cepat.

V. PENGUJIAN DAN DISKUSI

Setelah melakukan implementasi, aplikasi yang sudah dibuat akan diuji untuk memastikan dapat digunakan sesuai fungsinya. Pengujian akan dibagi menjadi dua yaitu pengujian fungsional pada modul yang dibuat dan pengujian performa algoritma.

A. Pengujian Fungsional Aplikasi

Pengujian pertama yang akan dilakukan adalah pengujian fungsional dari modul. Pengujian akan dilakukan dengan melakukan *unit testing* pada tiap-tiap modul yang dikembangkan. Akan diberikan sebuah kasus uji lalu harapan hasil dari kasus tersebut.

Pengujian akan dilakukan melalui tiap antarmuka aplikasi yang sudah dibuat sehingga terdapat kasus uji yang ditangani di modul antarmuka (bukan pada fungsi pada bagian sebelumnya). Modul pertama yang akan diuji adalah modul pembangkitan kode QR

TABEL I. PENGUJIAN MODUL PEMBANGKITAN KODE QR

No.	Kasus Uji	Masukan (*Input*)	Harapan Hasil (*Output*)	Hasil Pengujian
1.	Input valid	NIM = 18219095 Date = 5/21/2022 D = 107195279364 4849 N = 152135101983 6551	Kode QR dibangkitkan dan disimpan.	Kode QR dibangkitkan dan disimpan.
2.	Panjang NIM tidak sesuai	NIM = 1234 Date = 5/21/2022 D = 107195279364 4849 N = 152135101983 6551	Kode QR gagal dibangkitkan dan muncul peringatan.	Kode QR gagal dibangkitkan dan muncul peringatan.
3.	Nilai D tidak dimasukkan	NIM = 18219095 Date = 5/21/2022 N = 152135101983 6551	Kode QR gagal dibangkitkan dan muncul peringatan.	Kode QR gagal dibangkitkan dan muncul peringatan.
4.	Nilai N tidak dimasukkan	NIM = 18219095 Date = 5/21/2022	Kode QR gagal dibangkitkan dan muncul peringatan.	Kode QR gagal dibangkitkan dan muncul

No.	Kasus Uji	Masukan (*Input*)	Harapan Hasil (*Output*)	Hasil Pengujian
		D = 107195279364 4849		peringatan.

Dari hasil pengujian pada kasus uji yang didefinisikan modul pembangkitan kode QR yang dibuat sudah berhasil memenuhi setiap kasus uji tersebut. Untuk kasus uji "parameter *date* tidak dimasukkan" tidak dimasukkan kedalam daftar pengujian karena bagian antarmuka dari aplikasi sudah secara *default* memiliki tanggal yaitu tanggal hari ini.

Modul selanjutnya yang akan diuji adalah modul pembangkitan tanda tangan digital. Karena modul ini tidak memiliki antarmuka khusus, pengujian akan dilakukan langsung pada modul.

TABEL II. PENGUJIAN MODUL PEMBANGKITAN TANDA TANGAN DIGITAL

No.	Kasus Uji	Masukan (*Input*)	Harapan Hasil (*Output*)	Hasil Pengujian
1.	Input valid	TEXT = "KRIPTO IS FUN! 2022" D = 107195279364 4849 N = 152135101983 6551	Tanda tangan dibangkitkan	Tanda tangan berhasil dibangkitkan "0d2a99e112a0aeaa 7a05b338349b2f64d 68fc1a25ac1ec4888 8d07d5a57db3fc07a ef0d97ffa89c35f64d 955dc"
2.	Tidak ada teks yang dimasukkan	D = 107195279364 4849 N = 152135101983 6551	Tanda tangan gagal dibangkitkan dan muncul peringatan.	Tanda tangan gagal dibangkitkan dan muncul peringatan " missing 1 required positional argument: 'text'"
3.	Nilai D tidak dimasukkan	TEXT = "KRIPTO IS FUN! 2022" N = 152135101983 6551	Tanda tangan gagal dibangkitkan dan muncul peringatan.	Tanda tangan gagal dibangkitkan dan muncul peringatan " missing 1 required positional argument: 'd'"
4.	Nilai N tidak dimasukkan	TEXT = "KRIPTO IS FUN! 2022" D = 107195279364 4849	Tanda tangan gagal dibangkitkan dan muncul peringatan..	Tanda tangan gagal dibangkitkan dan muncul peringatan " missing 1 required positional argument: 'n"

Dari hasil pengujian pada kasus uji yang didefinisikan modul pembangkitan tanda tangan digital yang dibuat sudah berhasil memenuhi setiap kasus uji tersebut.

Modul ketiga yang akan diuji adalah modul verifikasi kode QR. Modul ini akan menerima *inpu* berupa directory kode QR dan pasangan kunci publik.

TABEL III. PENGUJIAN MODUL VERIFIKASI KODE QR

No.	Kasus Uji	Masukan (*Input*)	Harapan Hasil (*Output*)	Hasil Pengujian
1.	Input valid	Kode QR yang dibangkitkan dengan parameter (NIM = 18219095 Date = 5/21/2022 D = 107195279364 4849 N = 152135101983 6551) Dan kunci publik dengan parameter (E = 5430209809 N = 152135101983 6551)	Kode QR valid	Kode QR valid
2.	QR Code Kadaluarsa	Kode QR yang dibangkitkan dengan parameter (NIM = 18219095 Date = 5/21/2021 D = 107195279364 4849 N = 152135101983 6551) Dan kunci publik dengan parameter (E = 5430209809 N = 152135101983 6551)	Kode QR valid tetapi kadaluwarsa.	Kode QR valid tetapi kadaluwarsa
3.	QR Code dibangkitkan dengan cara lain	Kode QR yang dibangkitkan dengan aplikasi lain dimana parameter tanggal diganti Dan kunci publik dengan parameter (E = 5430209809 N = 152135101983 6551)TEXT = "KRIPTO IS FUN! 2022"	Kode QR tidak valid.	Kode QR tidak valid.

No.	Kasus Uji	Masukan (*Input*)	Harapan Hasil (*Output*)	Hasil Pengujian
		N = 152135101983 6551		
4.	Kunci publik tidak berpasa ngan	Kode QR yang dibangkitkan dengan parameter (NIM = 18219095 Date = 5/21/2022 D = 107195279364 4849 N = 152135101983 6551) Dan kunci publik dengan parameter (E = 79 N = 3337)	Dideteksi perbedaa n kunci	Dideteksi perbedaan kunci

Dari hasil pengujian pada kasus uji yang didefinisikan modul verifikasi kode QR yang dibuat sudah berhasil memenuhi setiap kasus uji tersebut.

Untuk modul pembangkitan kunci RSA tidak dilakukan pengujian pada makalah ini karena modul yang digunakan adalah penyesuaian dari tugas sebelumnya dan sudah melalui proses pengujian (agar tidak redundan).

B. Pengujian Performa Aplikasi

Pengujian selanjutnya yang akan dilakukan adalah pengujian performa untuk modul yang membutuhkan proses komputasi kompleks. Untuk tiap *input* akan dilakukan pengujian sebanyak 5 kali dan waktu yang dibutuhkan akan dirata-rata.

Pengujian performa pertama yang akan dilakukan adalah pada modul pembangkitan kode QR. Performa yang akan dihitung pada modul ini termasuk dengan proses pembangkitan tanda tangan digital.

TABEL IV. PENGUJIAN PERFORMA PEMBANGKITAN KODE QR

No.	Masukan (*Input*)	Rata-rata waktu (dalam detik)
1.	NIM= 18219095 Date = 5/21/2022 D = 1071952793644849 N = 1521351019836551	0.0685244
2.	NIM= 18219095 Date = 5/21/2022 D = 1019 N = 3337	0.0632506

Dari hasil pengujian ini terlihat bahwa perbedaan panjang kunci privat (d,n) tidak terlalu berpengaruh dalam waktu yang dibutuhkan untuk membangkitkan kode QR. Hal ini kemungkinan terjadi karena algoritma yang digunakan untuk mengenkripsi menggunakan fungsi pow() dengan tiga parameter yang dapat langsung memodulo bilangan tanpa harus dipangkatkan terlebih dahulu.

Pengujian performa selanjutnya yang akan dilakukan adalah pada modul verifikasi kode QR

TABEL V. PENGUJIAN PERFORMA PEMBANGKITAN KODE QR

No.	Masukan (*Input*)	Rata-rata waktu (dalam detik)
1.	Kode QR yang dibangkitkan dengan parameter: NIM= 18219095 Date = 5/21/2022 D = 1071952793644849 N = 1521351019836551 Kunci Publik E = 5430209809 N = 1521351019836551	0.0265662
2.	Kode QR yang dibangkitkan dengan parameter: NIM= 18219095 Date = 5/21/2022 D = 1019 N = 3337 Kunci Publik E = 79 N = 3337	0.0274184

Sama seperti pengujian performa sebelumnya tidak terlihat perbedaan yang signifikan terhadap waktu yang dibutuhkan untuk melakukan verifikasi kode QR.

KESIMPULAN DAN SARAN

Telah berhasil dibuat sebuah aplikasi untuk membangkitkan dan memverifikasi kode QR yang dapat digunakan untuk akses masuk kampus ataupun area umum lainnya. Performa yang dimiliki oleh sistem yang digunakan juga sudah dapat dibilang cukp cepat.

Algoritma RSA dan fungsi algoritma *hash* SHA-256 juga berhasil diimplementasi untuk membangkitkan tanda tangan digital yang digunakan untuk memverifikasi data yang disimpan pada kode QR.

Untuk pengembangan lanjutan, aplikasi yang sudah dibuat dapat dikembangkan dengan pemanfaatan perangkat keras kamera untuk membaca kode QR agar dapat semakin mirip dengan sistem verifikasi akses kampus ITB saat ini.

VIDEO LINK AT YOUTUBE

https://youtu.be/wzdu886HR7A

SOURCE CODE

https://github.com/jjojas/SignedCheckin

ACKNOWLEDGMENT

Terima kasih kepada Tuhan Yang Maha Esa karena atas karunianya penulis dapat menyelesaikan makalah yang berjudul "Implementasi Algoritma RSA dan Hash SHA-256 untuk Tanda Tangan Digital dalam Membangkitkan Kode QR Akses Masuk Kampus". Terima kasih juga saya ucapkan kepada segala pihak

yang telah membantu saya selama kegiatan belajar mengajar pada mata kuliah II4031 Kriptografi dan Koding, khususnya kepada Bapak Dr. Ir. Rinaldi Munir, M.T. Semoga hasil makalah ini dapat menjadi inspirasi untuk pihak lainnya yang akan mengeksplorasi masalah maupun solusi serupa.

REFERENSI

[1] S.-H. Hung, C.-Y. Yao, Y.-J. Fang, P. Tan, R.-R. Lee, A. Sheffer dan H.-K. Chu, "Micrography QR Codes," *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[2] "QR Code - About 2D Code," Denso-Wave, 5 Juni 2016. [Online]. Available: https://www.qrcode.com/en/about/. [Diakses 5 Mei 2022].

[3] R. Munir, "Algoritma RSA," 2021.

[4] R. L. Rivest, A. Shamir dan L. M. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Communications of the ACM*, 1978.

[5] R. Munir, "Fungsi Hash," 2021.

[6] R. Munir, "Secure Hash Algorithm (SHA)," 2021.

[7] A. Anand, "Breaking Down : SHA-256 Algorithm," InfoSec Write-ups, 27 November 2019. [Online]. Available: https://infosecwriteups.com/breaking-down-sha-256-algorithm-2ce61d86f7a3. [Diakses 21 Mei 2022].

[8] 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, 21 Mei 2022

Justin Dermawan Ikhsan