Analisis Verifikasi Keabsahan QR Code QRIS Menggunakan Digital Signature

Muhammad Ardi Avicenna / 18220057
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
ardiambis@gmail.com

***Abstract*— Keamanan informasi dalam era digital sangat penting, terutama dalam transmisi data dan dokumen elektronik. Untuk menjamin integritas, otentikasi, dan non-repudiasi, penggunaan tanda tangan digital pada QR Code telah menjadi solusi yang luas digunakan. QR Code adalah bentuk matriks dua dimensi yang efisien dalam menyimpan informasi dan telah digunakan dalam berbagai aplikasi, termasuk pembayaran digital. Namun, tanpa enkripsi, QR Code dapat rentan terhadap serangan dan manipulasi data. Untuk memperkuat keamanannya, konsep tanda tangan digital digunakan. Tanda tangan digital adalah metode autentikasi yang menggunakan kriptografi untuk memverifikasi keaslian dan integritas dokumen elektronik. Dengan menerapkan tanda tangan digital pada QR Code, keaslian dan integritas informasi yang disimpan di dalamnya dapat dijamin. Makalah ini membahas konsep tanda tangan digital, QR Code, dan bagaimana penerapan tanda tangan digital pada QR Code dapat meningkatkan keamanan informasi. Selain itu, juga dibahas mengenai QRIS (Quick Response Code Indonesian Standard) sebagai standar pembayaran QR Code di Indonesia, serta verifikasi menggunakan algoritma kunci-publik dan penggunaan fungsi hash SHA-3 (Keccak) dalam implementasi tanda tangan digital. Dengan implementasi yang tepat, penambahan tanda tangan digital pada QR Code QRIS dapat membantu dalam mencegah QR Code palsu dan meningkatkan keamanan transaksi digital.**

***Keywords*—Tanda tangan digital, QR Code, QRIS, RSA, SHA-3**

I. Introduction

Keamanan informasi menjadi salah satu aspek yang sangat penting dalam era serba digital kini. Transmisi data dan dokumen elektronik membutuhkan metode yang dapat menjamin integritas, otentikasi, dan non-repudiasi. Salah satu solusi yang digunakan secara luas adalah penggunaan tanda tangan digital pada QR Code.

QR Code (Quick Response Code) adalah bentuk matriks dua dimensi yang dapat menyimpan informasi secara efisien. QR Code telah menjadi populer dalam berbagai aplikasi, seperti pembayaran digital, identifikasi produk, dan penyimpanan informasi pribadi. Namun, dalam keadaan yang tidak terenkripsi, QR Code dapat rentan terhadap serangan dan manipulasi data.

Untuk memperkuat keamanan QR Code, digunakanlah konsep tanda tangan digital. Tanda tangan digital adalah metode autentikasi yang menggunakan kriptografi untuk memverifikasi keaslian dan integritas suatu dokumen elektronik. Dengan menerapkan tanda tangan digital pada QR Code, informasi yang disimpan di dalamnya dapat dijamin keasliannya dan integritasnya.

Makalah ini akan dibahas tentang konsep tanda tangan digital, QR Code, dan bagaimana penerapan tanda tangan digital pada QR Code dapat meningkatkan keamanan informasi.

II. Landasan Teori

A. QRIS

QRIS adalah singkatan dari Quick Response Code Indonesian Standard. Ini adalah standar nasional untuk pembayaran menggunakan QR code di Indonesia yang diluncurkan pada tahun 2019. Kode QRIS adalah jenis QR code yang dapat digunakan untuk melakukan pembayaran di pedagang yang mendukung sistem tersebut. Ini adalah sistem pembayaran yang terpadu yang memungkinkan pengguna melakukan pembayaran menggunakan ponsel mereka. Kode QRIS didukung oleh semua bank utama di Indonesia dan diatur oleh Bank Indonesia.

Namun, terdapat beberapa kasus pemalsuan kode QRIS yang telah terjadi, misalnya pemalsuan kode QRIS pada sebuah masjid di daerah Blok M, Jakarta Selatan. Modus pemalsuan ini merupakan sebuah bentuk skema pencurian uang melalui penggantian gambar kode QRIS yang ditempel pada kotak amal menjadi kode QRIS ke kantong pribadi. Modus ini tertangkap pada kamera pengawasan Masjid sehingga bisa digagalkan secepatnya. Hal ini terjadi karena belum adanya bentuk verifikasi ekstra dari metode pembayaran menggunakan QRIS sendiri.

B. Kriptografi Verifikasi

Verifikasi dengan kriptografi adalah proses memastikan bahwa data yang diterima adalah asli dan tidak ada yang mengubahnya. Kriptografi digunakan untuk otentikasi dalam berbagai situasi, seperti saat mengakses rekening bank, masuk ke dalam komputer, atau menggunakan jaringan yang aman.

Metode kriptografi digunakan oleh protokol otentikasi untuk mengonfirmasi identitas pengguna dan memastikan bahwa mereka memiliki hak akses yang diperlukan ke sumber daya. Kriptografi juga memungkinkan otentikasi data, yang dapat digunakan untuk memverifikasi integritas data dan identitas pengirim, artinya data tidak dapat diubah atau dimanipulasi tanpa otorisasi.

Terdapat dua metode verifikasi kriptografi yang ummum digunakan : Verifikasi menggunakan algortima kunci-simetri dan verifikasi menggunakan algoritma kunci-publik.

Verifikasi menggunakan algoritma kunci-simetri biasanya dilakukan dengan bantuan arbitrase melibatkan penggunaan algoritma kriptografi yang sama dan kunci yang sama antara pihak yang berkomunikasi. Algoritma kunci-simetri, juga dikenal sebagai kriptografi kunci-sesi, menggunakan kunci tunggal yang sama untuk mengenkripsi dan mendekripsi data.Dalam proses verifikasi ini, terdapat pihak ketiga yang bertindak sebagai arbiter atau penengah. Pihak ketiga ini bertugas untuk memastikan bahwa pesan yang diterima oleh penerima adalah asli dan tidak diubah oleh pihak lain.

Verifikasi menggunakan algoritma kunci-publik (atau juga dikenal sebagai kriptografi asimetris) melibatkan penggunaan pasangan kunci yang terdiri dari kunci publik dan kunci pribadi. Kunci publik digunakan untuk enkripsi dan verifikasi tanda tangan digital, sementara kunci pribadi hanya diketahui oleh pemiliknya dan digunakan untuk dekripsi dan pembuatan tanda tangan digital.

C. Tanda Tangan Digital

Tanda tangan digital adalah skema matematis untuk memverifikasi keaslian pesan atau dokumen digital. Tanda tangan digital yang valid memberikan keyakinan yang sangat tinggi kepada penerima bahwa pesan tersebut dibuat oleh pengirim yang dikenal (keaslian), dan bahwa pesan tersebut tidak diubah dalam perjalanan (integritas).

Tanda tangan digital biasanya dilakukan dengan implementasi algoritma kunci-publik. Secara umum proses tanda tangan digital adalah sebagai berikut,

1. Pengirim membuat hash dari pesan menggunakan fungsi hash.
2. Pengirim mengenkripsi hash tersebut menggunakan kunci pribadinya untuk membuat tanda tangan digital.
3. Pengirim mengirimkan pesan beserta tanda tangan digital kepada penerima.
4. Penerima mendekripsi tanda tangan digital menggunakan kunci publik pengirim untuk mendapatkan nilai hash.
5. Penerima membuat hash dari pesan yang diterima menggunakan fungsi hash yang sama dengan pengirim.
6. Penerima membandingkan kedua nilai hash (yang diperoleh dari tanda tangan digital yang telah dideskripsi dan yang dibuat dari pesan yang diterima) untuk memverifikasi bahwa mereka cocok.

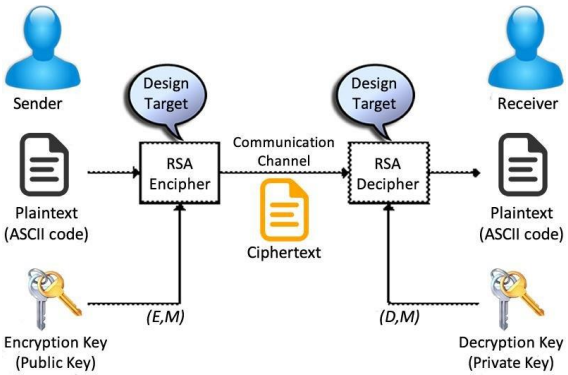
Proses ini memastikan bahwa pesan tidak diubah selama transmisi dan memang dikirim oleh pengirim. Bentuk implementasi tanda tangan digital yang umum dilakukan adalah dengan menggunakan Algoritma RSA dan Fungsi Hash SHA-3 Keccak.

D. Algoritma RSA

RSA adalah sebuah sistem kriptografi kunci-publik yang banyak digunakan untuk transmisi data yang aman. Sistem ini ditemukan oleh Ron Rivest, Adi Shamir, dan Leonard Adleman pada tahun 1977 dan dinamai berdasarkan inisial mereka. RSA didasarkan pada masalah matematika dalam memfaktorkan angka-angka besar menjadi faktor-faktor prima. Keamanan enkripsi RSA terletak pada kesulitan dalam memfaktorkan angka-angka besar menjadi faktor-faktor primanya. Berikut merupakan alur dari RSA secara umum,

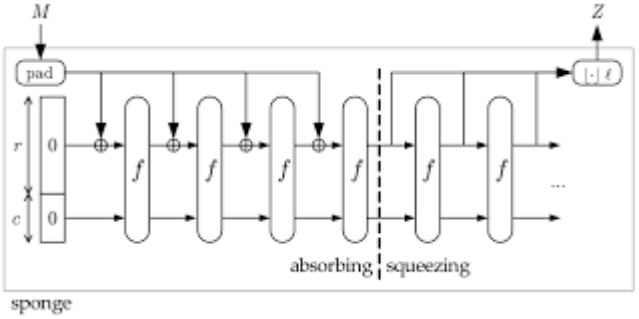
1. Pembuatan Kunci:
    a. Pilih dua bilangan prima besar, misalnya p dan q.
    b. Hitung n = p * q, di mana n adalah modulus.
    c. Hitung nilai totien $\varphi(n) = (p-1) * (q-1)$, di mana $\varphi(n)$ merupakan fungsi totien Euler yang menghitung jumlah bilangan bulat positif yang saling prima dengan n.
    d. Pilih sebuah bilangan bulat e, di mana $1 < e < \varphi(n)$ dan e saling prima dengan $\varphi(n)$. e akan menjadi kunci publik.
    e. Hitung nilai d, di mana $d * e \equiv 1 \pmod{\varphi(n)}$. d akan menjadi kunci pribadi.
2. Enkripsi:
    a. Konversi pesan menjadi bentuk numerik sesuai dengan skema tertentu, misalnya ASCII atau UTF-8.
    b. Bagi pesan numerik menjadi blok-blok yang lebih kecil, jika diperlukan.
    c. Untuk setiap blok pesan numerik, hitung $C = M^e \pmod{n}$, di mana M adalah blok pesan numerik dan C adalah ciphertext yang dihasilkan.
3. Dekripsi:
    a. Untuk setiap blok ciphertext C, hitung $M = C^d \pmod{n}$.
    b. Jika diperlukan, gabungkan blok-blok pesan numerik menjadi bentuk pesan awal.

Pada langkah-langkah di atas, proses enkripsi menggunakan kunci publik (n, e), sedangkan proses dekripsi menggunakan kunci pribadi (n, d). Kunci publik digunakan untuk mengenkripsi pesan, sementara kunci pribadi digunakan untuk mendekripsi pesan. Berikut juga dilampirkam diagram dari alur kerja RSA.



Gambar 1 Diagram Algoritma RSA (Sumber : https://www.researchgate.net/publication/298298027/figure/fig2/AS:339820552441867@1458030941634/RSA-algorithm-structure.png)

Untuk memperjelas alur, dilampikan juga diagram dari fungsi hash ini,



Gambar 2 Diagram Fungsi Hash SHA-3 (Sumber : https://asecuritysite.com/hash/s3)

E. Fungsi Hash SHA-3 (Keccak)

SHA-3 adalah fungsi hash kriptografi yang merupakan bagian dari keluarga Secure Hash Algorithm. Fungsi ini dirancang oleh Guido Bertoni, Joan Daemen, Michaël Peeters, dan Gilles Van Assche. SHA-3 dipilih pada tahun 2012 setelah melalui kompetisi publik antara beberapa fungsi hash. Ini adalah anggota terbaru dari keluarga Secure Hash Algorithm dan dianggap lebih aman daripada pendahulunya. Fungsi *hash*

Berikut adalah langkah-langkah umum dalam cara kerja fungsi SHA-3:

1. Persiapan Input: Pesan yang akan di-hash, biasanya dalam bentuk blok-blok data, dipersiapkan dengan penambahan bit-padding dan penyesuaian struktur input sesuai dengan aturan SHA-3.
2. Absorbing Phase: Blok-blok pesan diabsorb ke dalam keadaan internal fungsi hash, yang disebut "sponge". Sponge memiliki keadaan internal berukuran tetap, yang biasanya disebut "kapasitas" (capacity) dan "lempeng" (rate). Blok pesan yang diabsorb di-XOR dengan keadaan internal sponge dan diaplikasikan dengan fungsi peremukan (bit-permutation). Proses ini dilakukan secara iteratif untuk semua blok pesan.
3. Pemampatan (Squeezing) Phase: Setelah semua blok pesan diabsorb, tahap pemampatan dimulai. Keadaan internal sponge diubah menjadi hasil hash yang dihasilkan. Proses ini melibatkan iterasi peremukan pada keadaan internal sponge.
4. Penyediaan Output: Hasil hash yang dihasilkan diambil sebagai output fungsi SHA-3. Output biasanya dalam bentuk blok-blok yang sesuai dengan panjang yang diinginkan, seperti SHA-3-256 menghasilkan output 256-bit.

III. RANCANGAN IMPLEMENTASI

Implementasi solusi penyelesaian masalah verifikasi dirancang pada bab ini. Solusi akan diimplementasikan menggunakan algoritma enkripsi RSA dan fungsi hash SHA-256 sebagai basis dari tangan digital berbasis kunci publik.

Rancangan solusi dan implementasi dibagi menjadi beberapa modul sebagai berikut :

1. Modul Pembangkitan Kunci

   Pada modul ini, akan digunakan *library* random untuk membangkitkan nilai p dan q acak yang prima. Alur dari pembangkitan Kunci adalah sebagai berikut,

   a. Dibangkitkan nilai p dan q acak yang bersifat prima
   b. Nilai p dan q ini kemudian dihitung nilai n dan totient-nya
   c. Menggunakan nilai n dan totient, dibangkitkan nilai *public key*
   d. Menggunakan nilai totient dan *public key* dibandkitkan nilai *private key*

2. Modul Pemberian Tanda Tangan Digital

   Pada mosul ini, alam digunakan *library* Hashlib untuk memakai modul SHA-3. Alur dari pemberian tanda tangan digital adalah sebagai berikut,

   a. Diterima pesan teks yang berupa hasil *scan* dari QRIS
   b. Pesan tersebut akan di-*hash* menggunakan modul yang telah diambil dari *library*
   c. Pesan yang telah di-*hash* akan dienkripsi menggunakan algoritma RSA

3. Modul Verifikasi Tanda Tangan Digital

a. Diterima pesan teks yang berupa hasil *scan* dari QRIS dan file tanda tangan dari QRIS

b. Pesan tersebut akan di-*hash* menggunakan modul yang telah diambil dari *library*

c. Tanda Tangan akan didekripsi menggunakan algoritma RSA

d. Tanda tangan terdekripsi akan dibandingkan dengan pesan yang telah di-*hash*

e. Jika mereka berisi sama, maka QR CODE QRIS berhasil terverifikasi

f. Jika tidak sama, maka QR CODE QRIS telah diubah atau palsu

IV. REALISASI IMPLEMENTASI

Realisasi implementasi adalah source code sebagai berikut dibagi berbagai modul :

1. Modul Pembangkitan Kunci

Berikut Merupakan Source Code dari modul ini

```
import random

def is_prime(num):
    if num <= 1 :
        return False

    for i in range (2, int(num**0.5)+1):
        if num % i == 0 :
            return False

    return True

def random_prime():
    num = random.randrange(1, 1000000000)
    while not(is_prime(num)):
        num = random.randrange(1, 1000000000)

    return num

def initiate(p,q):
    n = p*q
    totient = (p-1)*(q-1)

    return n, totient

def generate_public_key(n, totient):
    e = random.randrange(1, totient)
    while greatest_common_divisor(e,
totient) != 1:
        e = random.randrange(1, totient)

    return (e,n)

def generate_private_key(totient, pubkey):
    key, n = pubkey
    d_old, d_new = 0, 1
    r_old, r_new = totient, key
    while r_new != 0:
        quotient = r_old // r_new

        temp = r_old
        r_old = r_new
        r_new = temp - quotient * r_new

        temp1 = d_old
        d_old = d_new
        d_new = temp1 - quotient * d_new

    d = 0
    if r_old == 1 :
        d = d_old % totient

    return (d, n)

def greatest_common_divisor(a, b):
    while b != 0 :
        temp = a
        a = b
        b = temp % b
    return a

def check_relative_prime(a, b):
    is_relative_prime = False
    while b != 0 :
        temp = a
        a = b
        b = temp % b

    if a == 1 :
        is_relative_prime = True

    return is_relative_prime
```

2. Modul Pembangkitan Kunci

Berikut Merupakan Source Code dari modul ini,

```
BUF_SIZE = 65536  # lets read stuff
in 64kb chunks!

def hashfile(filename):
    sha3 = hashlib.sha3_256()
    with open(filename, 'rb') as f:
        while True:
            data = f.read(BUF_SIZE)
            if not data:
                break
            sha3.update(data)
    return sha3.hexdigest()

def hashstring(string):
    sha3 = hashlib.sha3_256()
    sha3.update(string.encode())
    return sha3.hexdigest()


def encrypt_digest(prikey,
plaintext):
    key, n = prikey
    ciphertext = []

    for char in plaintext:
        ciphertext.append(pow(ord(ch
ar),key,n))

    hex_sign = f''
    for char in ciphertext:
        hex_sign = hex_sign +
hex(char)

    return (hex_sign)

def sign_message(message,
private_key):
    message_hashed =
hashlib.sha3_256(message.encode()).h
exdigest()
    return
encrypt_digest(private_key,
message_hashed)

def sign_file(file_path,
private_key):
    file_hashed =
hashfile(file_path)
    return
encrypt_digest(private_key,
file_hashed)
```

3. Modul Pembangkitan Kunci

Berikut Merupakan Source Code dari modul ini,

```
BUF_SIZE = 65536  # lets read stuff
in 64kb chunks!

def hashfile(filename):
    sha3 = hashlib.sha3_256()
    with open(filename, 'rb') as f:
        while True:
            data = f.read(BUF_SIZE)
            if not data:
                break
            sha3.update(data)
    return sha3.hexdigest()

def hashstring(string):
    sha3 = hashlib.sha3_256()
    sha3.update(string.encode())
    return sha3.hexdigest()

def decrypt_digest(pubkey,
ciphertext):
    plaintext_arr = []
    key, n = pubkey
    ciphertext = ciphertext.strip()
    if ciphertext.startswith('0x'):
        parts =
ciphertext.split('0x')[1:]
        for part in parts:
            plaintext_arr.append(int
(part, 16))
```

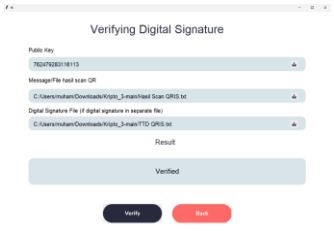
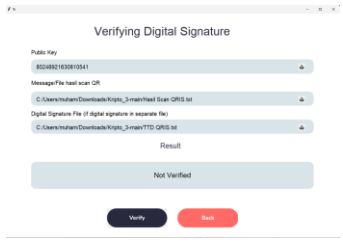
```python
from RSA import *


def verification(digest, signature, pubkey):
    decrypted_signature = decrypt_digest(pubkey, signature)
    if digest == decrypted_signature:
        return True
    else:
        return False
```

4. Pengujian

Berikut ini merupakan tabel pengujian dari implementasi ini,

TABEL I. TABEL PENGUJIAN

No.	Kasus	*Verdict*
1.	Tanda tangan QR Diverifikasi dengan Kunci Publik yang benar dan Pesan Tidak Diubah	Program berhasil Memverifikasi tanda tangan dengan hasil scan qris sehingga pada kasus ini pengujian dinyatakan **Berhasil** 
2.	Tanda tangan QR Diverifikasi dengan Kunci Publik yang benar dan Pesan Diubah	Program tidak berhasil Memverifikasi tanda tangan dengan hasil scan qris palsi sehingga pada kasus ini pengujian dinyatakan **Berhasil** 
3.	Tanda tangan QR Diverifikasi dengan Kunci Publik yang salah dan Pesan Tidak Diubah	Program tidak berhasil Memverifikasi tanda tangan dengan hasil scan qris sehingga pada kasus ini pengujian **Berhasil** 

V. KESIMPULAN DAN SARAN

Berdasarkan implementasi solusi di atas, dapat disimpulan bahwa penambahan tanda tangan pada QR Code QRIS dapat membantu pengguna untuk menghindari kasus QRIS palsu. Kesimpulan ini didapatkan dari hasil implementasi solusi yang dapat melakukan verifikasi tanda tangan tanpa menghadapi masalah yang berarti.

Saran untuk penelitian selanjutnnya dapat dilakukan implementasi dengan kemampuan langsung membandingkan gambar QR Code dengan tanda tangan asli. Implementasi ini akan membutuhkan program yang dapat menjalankan fungsi kamera dan bentuk implementasi dapat dipasang pada perangkat *mobile* seperti *handphone* android.

LINK GITHUB

https://github.com/ardiavi/Kripto-Paper

REFERENCES

[1] Menezes, A. J., Oorschot, P. C. van, & Vanstone, S. A. (2010). Handbook of applied cryptography. CRC press.

[2] "Kok Bisa Terjadi Pemalsuan QRIS? Begini Penjelasan Lengkap BI," CNBC Indonesia, 11-Apr-2023. [Online]. Available: https://www.cnbcindonesia.com/tech/20230411115931-37-428904/kok-bisa-terjadi-pemalsuan-qris-begini-penjelasan-lengkap-bi. [Accessed: 18-May-2023].

[3] A. Tomb, "Automated Verification of Real-World Cryptographic Implementations," in IEEE Security & Privacy, vol. 14, no. 6, pp. 26-33, Nov.-Dec. 2016, doi: 10.1109/MSP.2016.125.K.

[4] Sudaryono, A. Faturahman, N. P. Lestari Santoso, W. Y. Prihastiwi and B. A. Almadania Laksminingrum, "SaaS Platform for Blockchain Based E-Document Authentication applications," 2022 International Conference on Science and Technology (ICOSTECH), Batam City, Indonesia, 2022, pp. 1-7, doi: 10.1109/ICOSTECH54296.2022.9829113.

[5] M. Bansal, S. Gupta and S. Mathur, "Comparison of ECC and RSA Algorithm with DNA Encoding for IoT Security," 2021 6th International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, 2021, pp. 1340-1343, doi: 10.1109/ICICT50816.2021.9358591.

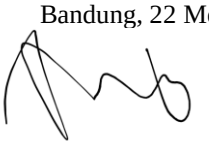
[6] V. Arribas, "Beyond the Limits: SHA-3 in Just 49 Slices," 2019 29th International Conference on Field Programmable Logic and Applications

(FPL), Barcelona, Spain, 2019, pp. 239-245, doi: 10.1109/FPL.2019.00044.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023



Muhammad Ardi Avicenna (18220057)