Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

Perpustakaan Univeristas Multimedia Nusantara

BAB III

METODOLOGI PENELITIAN DAN PERANCANGAN APLIKASI

3.1 Metodologi Penelitian

Metodologi penelitian yang digunakan dalam penelitian ini dibagi menjadi lima tahap, yaitu tahap telaah literatur, analisis dan perancangan, implementasi, pengujian, evaluasi.

1. Telaah Literatur

Di dalam tahap telaah literatur ini dilakukan telaah literatur yang berhubungan dengan Algoritma Kuantum Shor yang berisi istilah komputasi kuantum seperti *Qubit, Superposition, Modular Multiplication, dan Inverse QFT*, dan juga terdapat literatur sistem Kriptografi RSA, metode implementasi yang ingin dicoba di penelitian ini yaitu dengan menggunakan Qiskit *IBM Q Experience* untuk menjalankan sirkuit kuantum Shor untuk meretas kriptografi RSA, dan metode simulasi untuk menjalankan komputasi kuantum dengan menggunakan *backend quantum simulator* milik Qiskit.

2. Analisis dan Perancangan

Pada tahap ini dilakukan analisis dan perancangan untuk pembuatan program implementasi algoritma *Shor* dan fungsi dekripsi RSA dan pembuatan data yang akan dipakai untuk tahap uji coba. Untuk implementasi algoritma Shor, digunakan implementasi sirkuit kuantum Shor oleh Qiskit sebagai sumber dalam pembuatan fungsi kuantum seperti *Modular Multiplication* dan *Inverse Quantum Fourier Transform (QFT)*. Implementasi RSA dirancang dengan sesuai dengan

batasan jumlah *bit* yang tersedia. RSA yang dirancang akan melakukan dekripsi untuk setiap huruf yang di *input* dan tidak menggunakan konsep *padding*. Pembuatan *private key* RSA akan menggunakan *Extended Euclidean Algorithm.*

3. Implementasi

Di dalam tahap implementasi ini dilakukan implementasi perancangan yang telah dibuat di tahap perancangan sebelumnya dengan menggunakan Bahasa *python* untuk Menyusun *web service* yang akan menjadi program untuk implementasi peretasan RSA menggunakan sirkuit kuantum Shor. Akan ditunjukan potongan kode untuk fungsi yang digunakan untuk setiap implementasi algoritma Shor dan RSA. Digunakan Kuantum Simulator dari IBM Q Experience dan *library* Qiskit untuk bagian kuantum dari program implementasi. Untuk antarmuka digunakan *Python Flask* sebagai tampilan dasar *web service python*.

4. Pengujian

Di dalam tahap pengujian ini diuji apakah implementasi sirkuit kuantum Shor untuk peretasan RSA sesuai dengan perancangan yang telah dibuat di tahap perancangan sebelumnya. Implementasi diuji apakah telah berjalan dengan benar dan lancar dengan *input* dan *ouput* yang sesuai. Pengujian dilakukan dengan menggunakan teknik *black box testing* untuk melihat hasil *output* dari sirkuit kuantum yang di implementasi. Pengujian implementasi dilakukan dengan membandingkan hasil implementasi dan hasil enkripsi dan dekripsi secara manual untuk mengetahui apakah hasil dekripsi sudah betul atau belum.

5. Evaluasi

Di dalam tahap evaluasi ini diukur dan dievaluasi hasil implementasi dari sirkuit kuantum Shor untuk peretasan RSA sesuai dengan perancangan simulasi

yang telah dirancang. Evaluasi yang diukur adalah hasil waktu *user time* dan hasil *output decrypted text*.

3.2 Perancangan Aplikasi

Perancangan aplikasi untuk melakukan simulasi peretasan RSA dengan sirkuit kuantum Shor akan dibuat dengan menggunakan bahasa pemrograman *python* yang akan berbentuk aplikasi *web* yang akan berfungsi untuk melakukan peretasan. Perancangan aplikasi akan dibagi menjadi dua tahap yaitu perancangan fungsi aplikasi dan perancangan antarmuka halaman aplikasi.

3.2.1 Fungsi Aplikasi

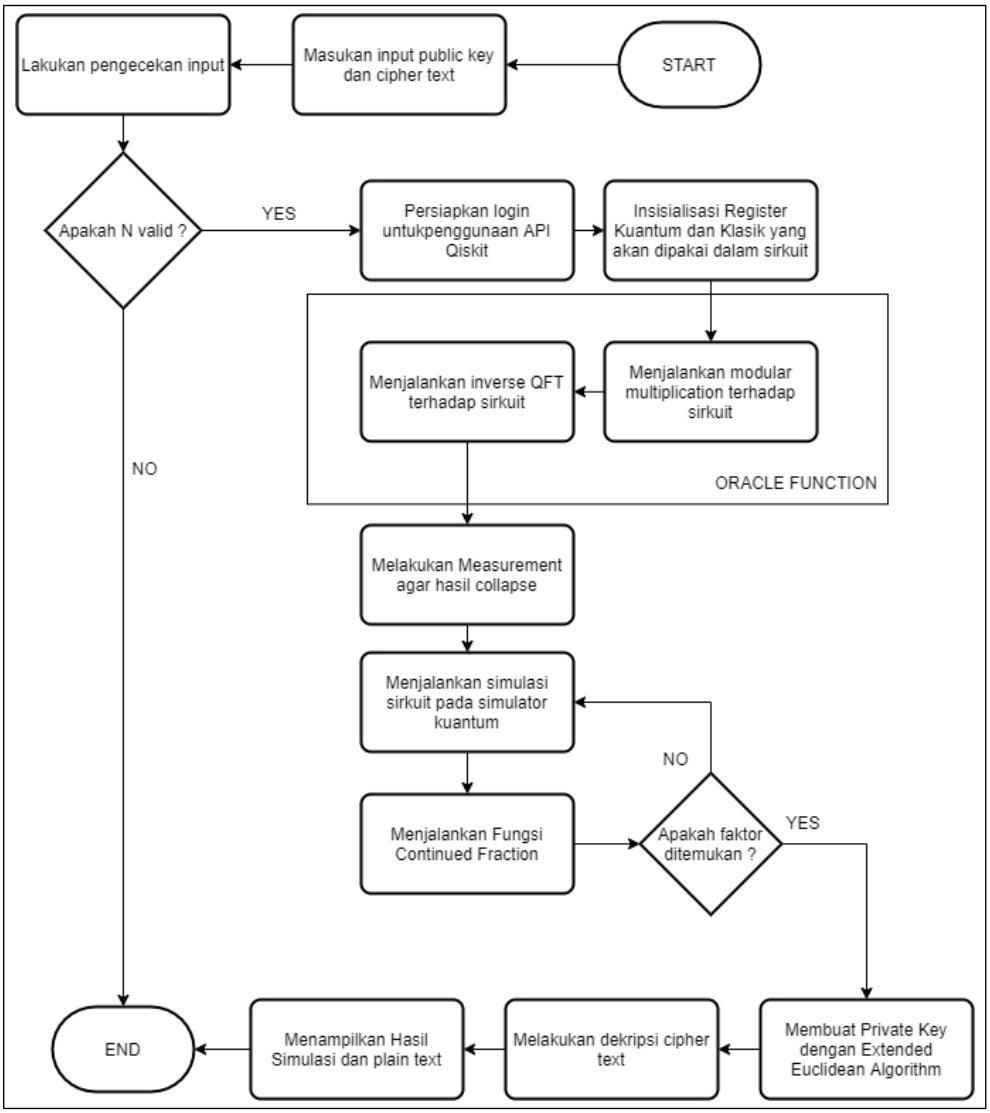
Perancangan fungsi aplikasi akan berdasar kepada algoritma kuantum Shor dan algoritma RSA. Implementasi sirkuit kuantum Shor akan menggunakan implementasi algoritma Shor yang telah dibuat oleh *Qiskit* yang dapat digunakan untuk umum. Implementasi tersebut akan menggunakan *qubit* sebanyak $4n + 2$ di mana n adalah ukuran *bit* dari angka yang akan difaktorkan. Implementasi algoritma RSA akan dibuat berdasarkan Langkah-langkah dasar RSA dengan melakukan dekripsi untuk setiap huruf yang ada di dalam *string*, hal ini dilakukan karena jumlah *qubit* yang dapat digunakan untuk dijadikan *key* berukuran kecil hanya sebatas 26 *qubit.*

Berikut adalah rancangan fungsi dari implementasi sirkuit kuantum Shor oleh.

1. Menerima *input* yang merupakan *ciphertext* dan *public key* RSA.
2. Melakukan *preprocessing* untuk mengubah *ciphertext* menjadi bentuk *ASCII*, dan mengambil variabel *N* dari *public key*.
3. Melakukan validasi kepada variabel *N* yaitu bilangan yang ingin di faktorkan.
4. Melakukan *login* kepada *API IBM Q Experience* untuk mengakses *backend* simulator kuantum yang akan dipakai.
5. Menginisialisasi register kuantum dan klasik untuk sirkuit yang akan dijalankan.
6. Menjalankan *Oracle Function* dari algoritma Shor yang terdiri dari *modular multiplication* untuk menjabarkan hasil fungsi periodik dalam *state superposition* dan *inverse QFT* untuk mendapatkan sebuah angka yang mempunyai probabilitas untuk menjadi periode dari hasil fungsi periodik tersebut.
7. Menampung hasil dari operasi sebelumnya ke dalam sirkuit klasik untuk di *measure* dan menghasilkan *r* variabel periode yang digunakan untuk mencari faktor *N* pada algoritma pemfaktoran Shor.
8. Menjalankan pengulangan simulasi sirkuit dengan menggunakan *backend quantum simulator* dari Qiskit *IBM Q Experience* dengan parameter *Shots = 1*.

9. Memproses semua hasil r yang didapatkan dan mencari *Continued Fractions* dari setiap r dan mengecek apakah hasil tersebut merupakan faktor dari N. Jika berhasil kembalikan nilai faktor dari N dan jika tidak kembalikan bahwa faktor tidak ditemukan dan melanjutkan pengulangan simulasi.
10. Menggunakan hasil-hasil faktor dari N untuk membuat *private key* RSA dengan menggunakan algoritma *Extended Euclidean Algorithm*.
11. Menggunakan *private key* untuk melakukan dekripsi *cipher text* berdasarkan rumus $M = C^d Mod\ n$ untuk mendapatkan *plain text*.

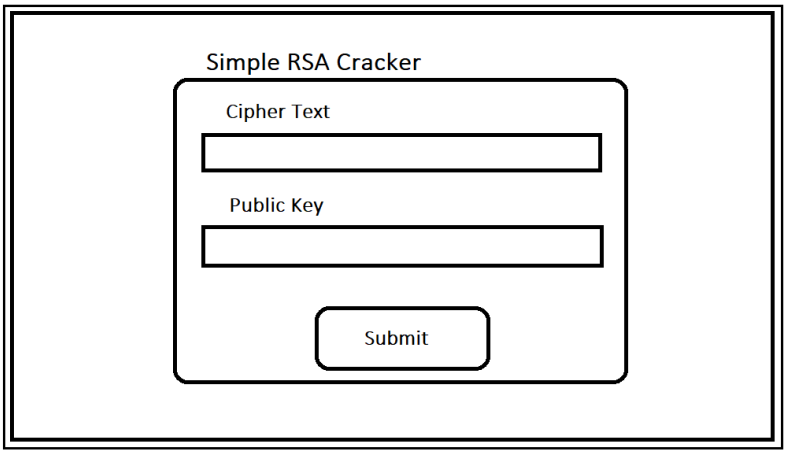
Dengan tujuan memperjelas proses dari fungsi aplikasi untuk implementasi ini dibuat *flowchart* untuk menjadi dasar perancangan aplikasi implementasi sirkuit kuantum Shor untuk meretas kriptografi RSA. *Flowchart* ini akan menjadi dasar perancangan dari pembuatan aplikasi implementasi ini namun akan terdapat penambahan fitur seperti *input* dan *output* data yang perancangannya dapat dilihat pada rancangan antarmuka halaman aplikasi *web*.



Gambar 3.1 *Flowchart* aplikasi implementasi sirkuit Shor untuk meretas kriptografi RSA

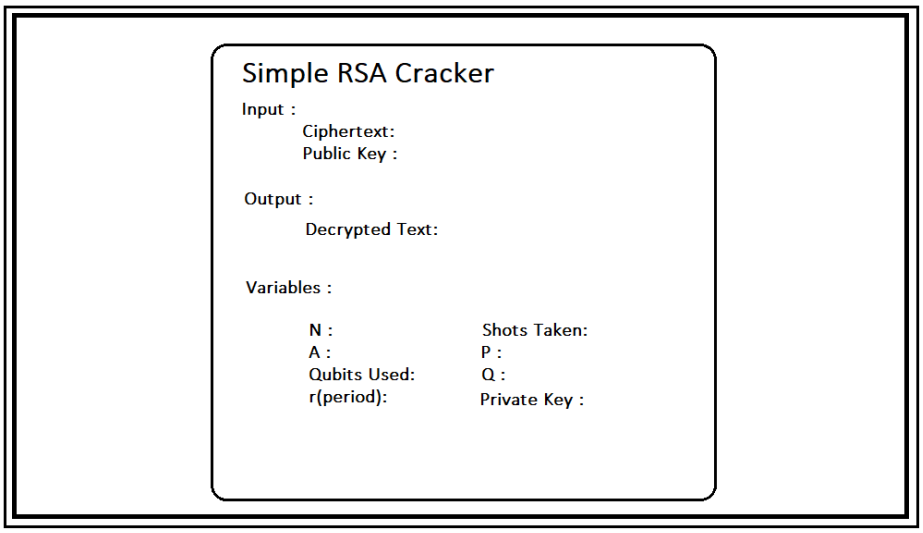
3.2.2 Antarmuka Halaman

Rancangan tampilan antarmuka untuk program dapat dilihat pada Gambar 3.2 dan 3.3. Proses *input* terdapat pada halaman *input* yang terdiri dari *form* yang akan menerima *input cipher text* dan *public key* RSA, dan tombol *submit* yang akan memindahkan ke halaman *result* yang menampilkan hasil.



Gambar 3.2 Rancangan antarmuka halaman *Input* program

Untuk tampilan antarmuka setelah dilakukan *input* dan tombol *submit* telah ditekan dapat dilihat pada Gambar 3.3



Gambar 3.3 Rancangan antarmuka halaman *Result* program

3.3 Pembuatan Data Uji Coba

Pada tahap uji coba, dibutuhkan data yang akan dijadikan sebagai *input*. Aplikasi implementasi sirkuit kuantum Shor untuk meretas kriptografi RSA membutuhkan data uji coba berupa *cipher text* dan *public key*. Pada tahap ini akan dibuat lima bentuk data uji coba yang tediri dari lima *cipher text* dalam bentuk *text*, dan lima bentuk *public key* dalam bentuk penulisan *public key* yaitu (*e*,*N*). Pemilihan *cipher text* dipilih sebuah *plain text* yang merupakan sebuah kata Bahasa Indonesia atau Bahasa Inggris dan mengubahnya menjadi *cipher text* dengan menggunakan program RSA sederhana. Pada Gambar 3.4 terdapat isi potongan kode untuk program RSA sederhana yang dibuat menggunakan *python* untuk pembuatan data uji coba. Program RSA ini menerima *input p* dan *q* sebagai faktor angka prima yang akan dijadikan *public* dan *private key*, dan *plaint text* yang akan dijadikan *cipher text*. Program RSA ini akan mengeluarkan *output* berupa *n* angka hasil perkalian *p* dan *q*, *e public key*, *t* yang merupakan *totient* yang mempunyai rumus $\Phi(N) = (p-1)(q-1)$, *d private key*, dan *cipher text* hasil enkripsi. Pembuatan *key* menggunakan algoritma *extended eucledian algorithm* sederhana.

```python
from decimal import Decimal

def gcd(a,b):
    if b==0:
        return a
    else:
        return gcd(b,a%b)

p = int(input('Enter the value of p = '))
q = int(input('Enter the value of q = '))
input = input('Enter the value of text = ')
no = [ord(c)-64 for c in input]
print("This is no : ", no)
n = p*q
t = (p-1)*(q-1)

for e in range(2,t):
    if gcd(e,t)== 1:
        break


for i in range(1,10):
    x = 1 + i*t
    if x % e == 0:
        d = int(x/e)
        break

ctt = [pow(i,e) for i in no]
ct = [(i % n) for i in ctt]

dtt = [pow(i,d) for i in ct]
dt = [(i % n) for i in dtt]

print("")
print('n = '+str(n)+' e = '+str(e)+' t = '+str(t)+' d = '+str(d)+'\nCipher text = '+str(ct)+'\nDecrypted text = '+str(dt))
```

Gambar 3.4 Potongan kode RSA sederhana untuk pembuatan data uji coba

Pembuatan data uji coba *public key* dilakukan dengan memilih hasil perkalian angka prima yang mempunyai jumlah *bit* yang dapat dipakai dalam implementasi sirkuit kuantum Shor yang cukup dengan jumlah *qubit* yang dimiliki oleh simulator kuantum milik Qiskit *IBM Q Experience*. Implementasi sirkuit kuantum Shor akan menggunakan *qubit* sebanyak $4n + 2$ di mana n adalah jumlah *bit* dari angka yang akan difaktorkan. Tabel 3.1 berisi nilai N yaitu hasil perkalian prima yang mempunyai jumlah *bit* yang dapat digunakan pada simulator kuantum yang akan digunakan dan *public key* yang akan digunakan untuk uji coba beserta *cipher text* yang ditentukan.

Tabel 3.1 Data uji coba

Primes	N	Bits	Qubits	Cipher Text	Public Key
3x5	15	4	18	GAMIL	(15,3)
3x7	21	5	22	POBTQI	(21,5)
5x7	21	6	26	IOaX	(35,5)
3x13	39	6	26	JETRCV\C	(39,5)
3x17	51	6	26	[RA[EOie	(51,3)
3x19	57	6	26	SMcHaAEgcHuJM^	(57,5)