Implementasi Algoritme Shor pada Sirkuit Kuantum untuk *Cracking* Algoritme RSA

Fitra Hutomo[1)], Rini Wisnu Wardhani[2)], Dion Ogi[3)]

1) Rekayasa Perangkat Keras Kriptografi, Politeknik Siber dan Sandi Negara, fitrahutomo62@gmail.com

2) Department of Information Convergence Engineering, School of Computer Science and Engineering, Pusan National University, rini.wisnu@pusan.ac.kr

3) Rekayasa Perangkat Keras Kriptografi, Politeknik Siber dan Sandi Negara, dion.ogi@poltekssn.ac.id

Abstrak

Penelitian ini membahas mengenai pemanfaatan teknologi *quantum computing* berupa sirkuit kuantum modular *exponentiation* untuk mencari periode dari algoritme Shor yang digunakan untuk memfaktorkan bilangan prima N yang digunakan dalam algoritme RSA. Sirkuit modular *exponentiation* yang digunakan dalam penelitian ini terdiri dari sirkuit *adder*, *reverse adder*, *modular adder*, *reverse modular adder*, *modular multiplier* dan *reverse modular multiplier* yang didesain menggunakan pendekatan dari VBE. Penelitian ini berhasil mengimplementasikan sirkuit modular *exponentiation* 8 bit dengan MSB bernilai 1 menggunakan *tools open-source* SDK Qiskit. Dari pengujian yang dilakukan terhadap nilai N = 143, 187, 209, 221, 247 dan 253 didapatkan hasil bahwa sirkuit kuantum modular *exponentiation* yang didesain menghasilkan periode yang sesuai dengan akurasi sebesar 100 %. Untuk menguji apakah implementasi algoritme Shor pada penelitian ini berhasil dalam melakukan *cracking* algoritme RSA maka dilakukan pengujian terhadap sistem yang dibuat dan didapatkan bahwa algoritme Shor berhasil dalam melakukan *cracking* algoritme RSA dengan cara memfaktorkan bilangan N yang dimasukkan dan menghitung kunci privat dari algoritme RSA.

Kata kunci: *Qiskit, Quantum Computing, Quantum Gate, RSA, Shor*

Abstract

This study discusses quantum computing technology in building a circuit of modular exponentiation circuits to find the period of Shor's algorithm, which will later be used to factor N numbers in the RSA algorithm. The modular exponentiation circuit used in this research consists of the adder, reverse adder, modular adder, reverse modular adder, modular multiplier, and reverse modular multiplier circuits designed using the Vedral et al. approach. This research has implemented an 8-bit modular exponentiation circuit with the constraint that the MSB is 1 using the open-source SDK Qiskit tool. For N equal to 143, 187, 209, 221, 247, and 253, the modular exponentiation circuit designed produces the appropriate period with an accuracy of 100%. Testing the implementation of the Shor algorithm will be successful in cracking the RSA algorithm; a scenario test is carried out on the circuit design result. The Shor algorithm's output successfully cracked the RSA algorithm for chosen N by factoring in the N number entered and calculating the RSA algorithm's private key.

Keywords: *Qiskit, Quantum Computing, Quantum Gate, RSA, Shor*

1. PENDAHULUAN

Teknologi *quantum computing* merupakan teknologi berbasis cahaya yang sedang dikembangkan dan sudah banyak diimplementasikan dalam berbagai bidang [1][2]. Teknologi ini menggunakan teori mekanika kuantum seperti *interference*, *superposition*, dan *entanglement* untuk memproses *Quantum Bit* (*Qubit*) [3][4][5], dengan tujuan untuk menyelesaikan permasalahan kompleks yang tidak bisa dilakukan oleh komputer konvensional [6][7]. *Qubit* merupakan unit informasi dasar dalam komputasi kuantum [8]. *Qubit* beda halnya dengan bit pada komputer klasik karena *qubit* dapat mengalami keadaan superposisi yaitu keadaan ketika *qubit* 1 dan *qubit* 0 terjadi dalam satu waktu, sehingga komputasi dapat dijalankan secara paralel dengan proses yang lebih cepat [7][9][10].

Kecepatan komputasi merupakan hal yang sangat penting, salah satunya dalam proses enkripsi dan dekripsi pesan menggunakan algoritme kriptografi [2][11]. Kecepatan komputasi dapat digunakan untuk mempersingkat waktu proses dalam uji performa dan uji serangan suatu algoritme kriptografi [11][12]. Saat ini komputasi masih mengandalkan komputer konvensional sehingga membutuhkan waktu yang lama dalam memproses [13]. Seiring berkembangnya teknologi *quantum computing* pemrosesan algoritme kriptografi juga akan diimplementasikan ke dalam teknologi tersebut [14][15]. Salah satu contoh algoritme kriptografi kunci publik yang banyak digunakan adalah algoritme Rivest-Shamir-Adleman (RSA). Keamanan algoritme RSA terletak pada pemfaktoran bilangan yang besar menjadi faktor-faktor primanya [16]. Selama belum ada algoritme pemfaktoran yang efektif dan efisien untuk memfaktorkan bilangan besar menjadi faktor-faktor prima maka keamanan algoritme RSA akan tetap terjamin [10]. Menurut beberapa penelitian yang sudah dilakukan, algoritme RSA diklaim mampu dipecahkan menggunakan teknologi *quantum computing* dengan memanfaatkan algoritme Shor

untuk memfaktorkan bilangan yang besar [14][17][18][19][20].

Pada penelitian ini sebagian proses dari algoritme Shor diimplementasikan pada sirkuit kuantum untuk melakukan *cracking* algoritme RSA dengan cara mendapatkan nilai hasil pemfaktoran bilangan N dan kunci privat yang digunakan dalam algoritme RSA. Kompleksitas untuk menemukan faktor dari bilangan yang besar akan meningkat secara eksponensial tetapi hal tersebut bisa diatasi menggunakan algoritme Shor yang diterapkan pada teknologi *quantum computing* [17]. Sirkuit kuantum digunakan sebagai bagian dalam proses perhitungan untuk melakukan pencarian periode algoritme Shor. Ketika nilai periode sudah didapatkan, dilakukan perhitungan nilai *p* dan *q* yang merupakan hasil pemfaktoran bilangan *N* pada algoritme RSA.

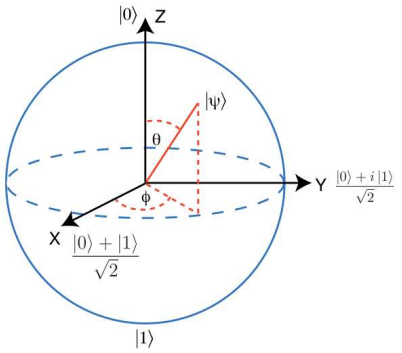
Setelah sirkuit kuantum berhasil diimplementasikan, dilakukan pengujian terkait akurasi sirkuit dalam mencari periode yang digunakan algoritme Shor untuk memfaktorkan bilangan *N*. Dalam penelitian ini implementasi sirkuit kuantum dan pengujiannya dilakukan dengan menggunakan aplikasi *open source* buatan IBM Research yaitu Quantum Information Science Kit (Qiskit). Sirkuit kuantum selanjutnya dikirimkan ke jaringan server milik IBM untuk dikompilasi dalam lingkungan riil komputer Kuantum.

2. LANDASAN TEORI

2.1. Komputasi Kuantum

Evolusi dari komputasi kuantum diprakarsai oleh Hukum Moore yang menyatakan bahwa kompleksitas sebuah mikroprosesor akan meningkat dua kali lipat setiap 1,5 tahun sekali [21]. Efek kuantum akan memainkan peran penting dalam pemodelan transistor berukuran kecil dengan memanfaatkan fenomena dasar fisika kuantum seperti *superposition* dan *entanglement* [19]. Istilah *superposition* mengacu pada prinsip ketika suatu objek berada dalam dua keadaan di waktu yang sama [5]. Istilah *entanglement* atau yang disebut juga dengan Bell State merupakan fenomena fisik yang terjadi ketika sepasang atau sekelompok partikel saling mempengaruhi walaupun dipisahkan oleh jarak yang sangat jauh [5]. Dua teorema kuantum tersebut yang mendasari pengembangan teknologi *Quantum Computing*.

Dalam komputer konvensional, informasi dikodekan dalam bit yang hanya memeiliki nilai 1 atau 0, sedangkan dalam komputasi kuantum, informasi dikodekan dalam *qubit* yang dapat memiliki nilai 0, 1 ataupun nilai 1 dan 0 dalam satu waktu yang disebut keadaan superposisi. Gambar 1 merupakan contoh dari *qubit* dalam keadaan superposisi yang divisualisasikan dalam bentuk vektor (memiliki arah dan besaran) yang terdapat di dalam sebuah bola blok (*Block Sphere*).



Gambar 1. Ilustrasi *Qubit* dalam Keadaan Superposisi [22]

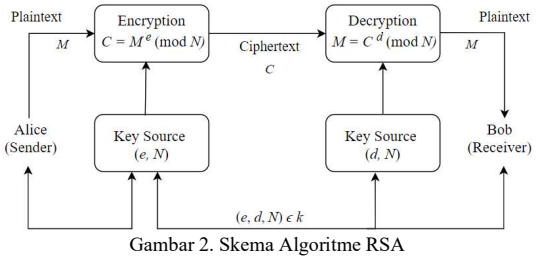
Dalam komputasi kuantum terdapat gerbang kuantum yang berfungsi untuk memproses *qubit* seperti gerbang logika pada komunikasi data digital. *Qubit* dalam komputer kuantum merupakan unit dasar informasi kuantum yang berfungsi untuk mengkodekan informasi. Komputasi kuantum memproses data *qubit* dengan menggunakan gerbang kuantum yang dinyatakan dalam bentuk matriks, yang mana setiap gerbang memiliki representasi matriks yang berbeda.

2.2. Algoritme RSA

Algoritme RSA terbagi menjadi tiga proses, yaitu pembangkitan kunci, enkripsi dan dekripsi pesan [23]. Dasar dari proses enkripsi dan dekripsi algoritme ini menggunakan konsep bilangan prima dan aritmatika modulus. Pasangan kunci publik (*e*, *N*) bersifat umum dan digunakan untuk proses enkripsi pesan, sedangkan pasangan kunci privat (*d*, *N*) bersifat rahasia dan digunakan untuk proses dekripsi pesan. Persamaan $C = M^e \bmod N$ untuk mengenkripsi pesan, sedangkan $M = C^d \bmod N$ untuk mendekripsi pesan dengan:

- C : *Ciphertext*
- M : Pesan
- e : kunci publik
- d : kunci privat
- N : nilai $p.q$

Keamanan algoritme RSA terletak pada tingkat kesulitan untuk memfaktorkan bilangan besar menjadi faktor-faktor prima dari N [10]. Gambar 2 merupakan gambaran skema dari algoritme RSA.



Gambar 2. Skema Algoritme RSA

2.3. Algoritme Shor

Algoritme Shor merupakan algoritme kuantum

yang dibuat oleh matematikawan Peter Shor pada tahun 1994 yang berfungsi untuk menyelesaikan faktorisasi bilangan dalam waktu polinomial yaitu $O((logN)^3)$ [24]. Pemfaktoran dari dua bilangan prima yang besar merupakan masalah yang sulit sampai saat ini ketika menggunakan komputer konvensional, namun masalah tersebut akan teratasi dengan cara menerapkan algoritme Shor pada teknologi *quantum computing* [17].

Berikut merupakan langkah-langkah yang dijalankan algoritme Shor untuk memfaktorkan bilangan [25].

1. Memilih nilai *x* acak dengan syarat $1 < x < N$, dengan gcd (*x, N*) = 1.
2. Menghitung nilai *r*, yang merupakan periode dari *x* (mod *N*), $x^r \equiv 1(mod\ N)$.
3. Mengecek nilai *r* apakah $r \equiv 0(mod\ 2)$, dan Mengecek nilai $x^{r/2} \not\equiv \pm 1\ (mod\ N)$.
4. Langkah terakhir adalah menghitung nilai p dan q dengan rumus berikut:
   $p = \gcd\ (x^{r/2} - 1, N)$
   $q = gcd\ (x^{r/2} + 1, N)$

2.4. *Quantum Information Science Kit*

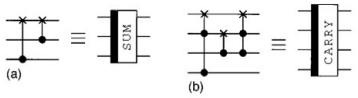
Qiskit (*Quantum Information Science Kit*) merupakan kerangka kerja *quantum computing* yang dirancang dan dikembangkan oleh IBM Quantum Laboratorium untuk digunakan dalam penelitian mengenai komputer kuantum dan aplikasinya [5][26]. Qiskit menyediakan *tools* untuk membuat, memanipulasi dan menjalankan program kuantum menggunakan komputer kuantum milik IBM. Dalam perkembangannya, saat ini Qiskit dapat digunakan untuk membentuk *quantum environment* tidak hanya di komputer kuantum milik IBM tetapi juga untuk membentuk *environment* di *local computer*. Dalam pemrograman sirkuit di *quantum computing,* untuk memudahkan membentuk *environment* ini, digunakan *software* Jupyter Notebook untuk mempermudah dalam instalasi, menambahkan *package*, *import* fungsi dan menjalankan program sirkuit kuantum.

2.5. Sirkuit Kuantum Algoritme Shor

Komputasi pada komputer kuantum membutuhkan operasi aritmatika kuantum yang tersusun dari gabungan *quantum gate* [27]. Dalam operasi aritmatika kuantum berlaku prinsip *reversible* maksudnya operasi pada sirkuit kuantum dapat dibalikkan atau dilakukan *inverse* dengan aturan pencerminan [27][28]. Penelitian yang dilakukan oleh Vedral *et.al* [27] menjelaskan secara lengkap mengenai konstruksi dari sirkuit *modular exponentiation* yang digunakan dalam algoritme Shor untuk mencari nilai periode. Dalam penelitian tersebut dijelaskan bahwa sirkuit *modular exponentiation* tersusun dari sirkuit *plain adder, adder modulo N* dan *controlled-multiplier modulo N* yang disusun berdasarkan aturan tertentu. Pada penelitian Larasati dan Kim [28], dilakukan konstruksi ulang sirkuit tersebut menjadi sirkuit *adder*, *modular adder*, *modular multiplier* dan *modular exponentiation* kemudian mengimplementasikannya pada *tools* Qiskit. Berikut ini penjelasan detil terkait sirkuit-sirkuit tersebut.

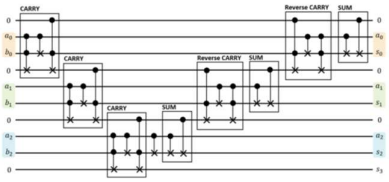
A. Sirkuit *Adder*

Sirkuit *adder* merupakan sirkuit kuantum yang memetakan $|a, b\rangle \rightarrow |a, a + b\rangle$ [28]. Sirkuit ini membutuhkan tiga buah *register* yaitu *register a* dan *b* yang merupakan input penjumlahan *a* dan *b,* dan juga *register carry* sebagai *temporary register* yang menyimpan hasil selama operasi penjumlahan. Sirkuit *adder* tersusun dari gerbang SUM dan CARRY yang terdiri dari gabungan gerbang CNOT dan Toffoli seperti ditunjukkan pada Gambar 3.



Gambar 3. Gerbang SUM dan CARRY [27]

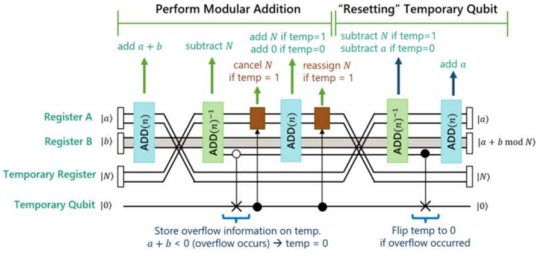
Gerbang tersebut dirangkai berdasarkan aturan yang sudah ditetapkan dan menghasilkan sirkuit *adder* seperti pada Gambar 4.



Gambar 4. Sirkuit VBE Adder [28]

B. Sirkuit *Modular Adder*

Sirkuit *modular adder* memetakan $|a, b\rangle \rightarrow |a, a + b\ mod\ N\rangle$ ketika $0 \leq a, b < N$. Sirkuit ini terdiri dari 3 sirkuit *adder* dan 2 sirkuit *reverse adder*. Selain itu sirkuit *modular adder* memiliki tambahan *register N* yang berfungsi sebagai input nilai modulo *N* dan juga *control register* sebagai *register* pengontrol operasi pada sirkuit *adder* berikutnya. Jumlah *register* yang dibutuhkan pada sirkuit ini memenuhi persamaan $4n + 2$. Gambar 5 menunjukkan sirkuit *modular adder*.



Gambar 5. Sirkuit *Modular Adder* [28]

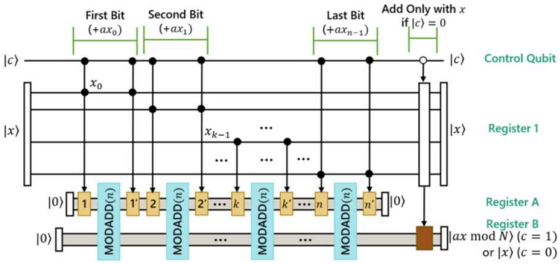
C. Sirkuit *Modular Multiplier*

Sirkuit *modular multiplier* melakukan operasi $ax\ mod\ N$ dengan nilai *a* adalah konstan [28]. Operasi ini dapat diperoleh dengan ekspansi biner dari *x* seperti yang ditunjukkan pada persamaan berikut:

$$ax = ax_{n-1}2^{n-1} + \cdots + ax_1 2^1 + ax_0 2^0 = \sum_{x_k} a2^k$$

Biner dari *x* dapat diperluas, sehingga bisa dilakukan perkalian dengan penjumlahan berulang dari $a2^k$ *mod N* untuk setiap bit dari *x* jika x_k = 1. Berdasarkan Gambar 6, blok berwarna kuning menempatkan $a2^k$ *mod N* ke *register* a yang merupakan input dari *modular adder*, kemudian dilakukan pembersihan pada *temporary register* sebelum masuk ke operasi *modular adder* berikutnya. Berikut ini pemetaan dari sirkuit *modular multiplie*r.

$$\text{CMODMULT}(n)\,|c,x,0,0\rangle = \begin{cases} |c,x,0,ax \bmod N\rangle & (c=1) \\ |c,x,0,x\rangle & (c=0) \end{cases}$$



Gambar 6. Sirkuit *Modular Multiplier* [28]

D. Sirkuit *Modular Exponentiation*

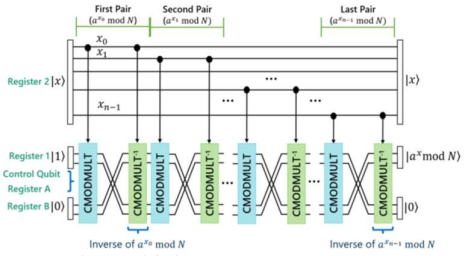
Sirkuit *Modular Exponentiation* melakukan operasi $a^x mod\ N$ dengan nilai *a* dan *N* adalah konstan. Berdasarkan ekspansi biner dapat ditulis dalam persamaan berikut.

$$a = a_{n-1}2^{n-1} + \cdots + a_1 2 + a_0$$
$$x = x_{n-1}2^{n-1} + \cdots + x_1 2 + x_0$$

Modular exponentiation dapat dianggap sebagai rangkaian perkalian dari ekspansi biner seperti ditunjukkan dalam persamaan berikut:

$$a^x = a^{x_{n-1}2^{n-1}} \times a^{x_{n-2}2^{n-2}} \times \ldots\ a^{x_1 2} \times a^{x_0}$$
$$= \prod_{x_{k=1}}^{n-1} a^{2^k}$$

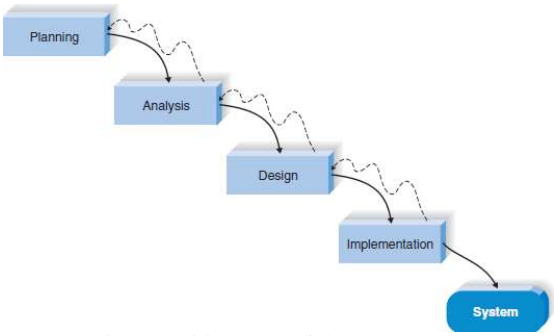
Berdasarkan Gambar 7 sirkuit *modular exponentiation* akan memetakan $|x,1,0\rangle \rightarrow |x,a^x \bmod N,0\rangle$. Sirkuit ini terdiri dari rangkaian pasangan *modular multiplier* (dalam pasangannya terdapat satu sirkuit yang dibalik/*inverse*) dengan gerbang *swap* di tengah setiap pasangan untuk mempertahankan setiap hasil operasi perkalian. Setiap *n*-bit *exponentiation* memerlukan *n* pasangan *modular multiplier* [28].



Gambar 7. Sirkuit *Modular Exponentiation* [28]

3. METODE PENELITIAN

Metodologi yang digunakan dalam penelitian ini adalah *System Development Life Cycle* (SDLC) dengan pendekatan *Waterfall Development*. Pada pendekatan *Waterfall Development*, terdapat beberapa proses yang dilakukan yaitu *planning*, *analysis*, *design* dan *implementation* yang dilakukan secara bertahap. Seperti yang ditunjukkan pada Gambar 8 berikut.



Gambar 8. Pendekatan *Waterfall Development* [29]

Adapun langkah-langkah yang dilakukan dalam penelitian ini adalah sebagai berikut:

a) *Planning* (Perencanaan)

Dalam tahapan ini ditentukan bagaimana sistem *cracking* RSA bekerja, yang dirumuskan dalam gambaran umum sistem berdasarkan referensi yang diperoleh. Metode yang digunakan dalam tahap ini adalah *feasibility study*.

b) *Analysis* (Analisis Kebutuhan)

Pada tahap ini ditentukan aspek-aspek yang diperlukan dalam membangun sistem *cracking* RSA, berupa kebutuhan fungsional dan kebutuhan nonfungsional serta kebutuhan *software* dan *hardware*. Pada tahapan ini ditentukan persyaratan dan spesifikasi sistem yang dibangun berdasarkan gambaran umum, dengan metode studi literatur.

c) *Design* (Desain)

Tahap desain merupakan tahapan untuk merumuskan pemodelan atas sistem *cracking* RSA yang dibangun. Pemodelan sistem menggunakan *flowchart*. *Flowchart* yang dibuat mempermudah dalam penyusunan program dan memberikan gambaran yang jelas mengenai alur yang dijalankan program.

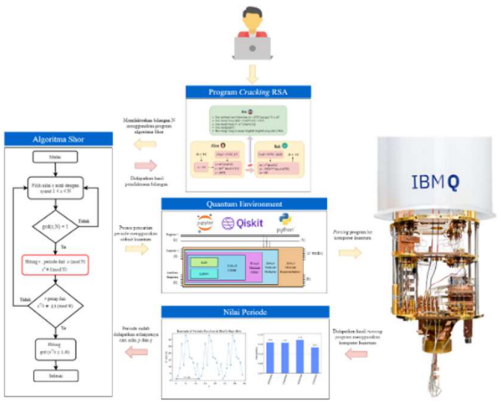
d) *Implementation* (Implementasi)

Pada tahap ini dilakukan penyusunan program berdasarkan desain yang sudah dirumuskan pada tahap sebelumnya. Dalam tahap ini dilakukan implementasi sistem *cracking* RSA, menggunakan skema algoritme Shor pada sirkuit kuantum dengan cara memprogram menggunakan Python pada aplikasi Qiskit. Integrasi sirkuit *adder*, *modular adder*, *modular multiplier* dan *modular exponentiation*. Setelah sirkuit kuantum dan program berhasil

diimplementasikan, maka dilakukan pengujian untuk mengetahui akurasi dari sirkuit dan program yang sudah dibuat.

4. HASIL PENELITIAN

Berdasarkan Gambar 9, implementasi *cracking* algoritme RSA dilakukan dengan cara memfaktorkan bilangan *N* menggunakan algoritme Shor. Dalam algoritme Shor terdapat proses pencarian periode dari *x* (mod *N*) yang dijalankan menggunakan komputer kuantum. Implementasi algoritme Shor pada komputer kuantum dilakukan dengan cara memprogram hasil desain sirkuit *adder*, *modular adder*, *modular multiplier* dan *modular exponentiation* pada *tools* Qiskit yang diprogram menggunakan *software* Jupyter Notebook dengan bahasa pemrograman Python.

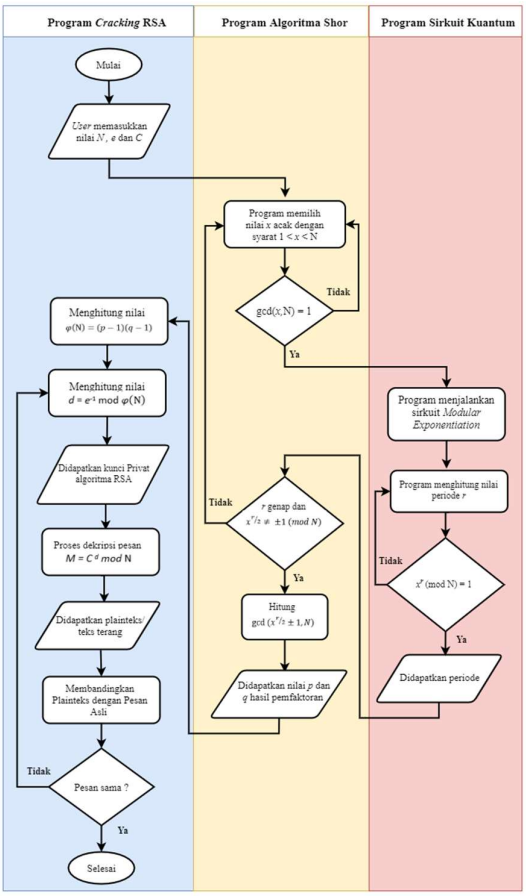


Gambar 9. Gambaran Umum Sistem

Hasil pemrograman Qiskit dapat dijalankan dalam Qiskit *quantum environment.* Setelah didapatkan periode, maka dilakukan perhitungan nilai *p* dan *q* sesuai rumus yang sudah dijelaskan sebelumnya berdasarkan nilai *r* yang sudah didapatkan. Kemudian dihitung kunci privat berdasarkan parameter yang diketahui untuk melakukan dekripsi *ciphertext* yang dimasukkan.

Gambar 10 merupakan *flowchart* dari keseluruhan sistem yang dibuat. Dalam sistem ini terdapat tiga program yang dibuat yaitu program *cracking* algoritme RSA, program algoritme Shor dan program sirkuit kuantum. Berdasarkan Gambar 10 langkah-langkah yang dijalankan *User* adalah memasukkan nilai *N*, kunci Publik (*e*) dan *Cipherteks* (*C*). Kemudian program akan memfaktorkan bilangan *N* menggunakan algoritme Shor. Dalam proses pemfaktoran bilangan *N* digunakan program Algoritme Shor dengan langkah-langkah yang dijalankan yaitu program akan memilih nilai *x* acak dengan syarat nilai *x* lebih dari satu dan kurang dari *N.*

Dalam proses pemfaktoran bilangan *N* akan menggunakan program Algoritme Shor dengan langkah-langkah yang dijalankan yaitu program akan memilih nilai *x* acak dengan syarat nilai *x* lebih dari satu dan kurang dari *N.* Dari nilai *x* acak tersebut dihitung nilai *x* yang memenuhi syarat gcd(*x*, *N*) = 1. Nilai *x* yang dipilih akan digunakan untuk mencari periode $x^r \equiv 1\ (mod\ N)$. Setelah periode didapatkan, maka nilai *r* akan dicek apakah memenuhi syarat. Ketika nilai memenuhi syarat *r* genap dan nilai $x^{r}/2 \not\equiv \pm 1\ (mod\ N)$ maka nilai *p* dan *q* dapat dihitung dengan rumus gcd $(x^{r}/2 \pm 1, N)$.



Gambar 10. *Flowchart* sistem *cracking* RSA

Setelah didapatkan hasil pemfaktoran bilangan maka program akan menghitung nilai $\varphi(\mathrm{N}) = (p-1)\,(q-1)$ dan menghitung kunci privat (*d)* menggunakan rumus $d = e^{-1}$ mod $\varphi(\mathrm{N})$. Setelah didapatkan kunci privat algoritme RSA maka program akan menghitung *M* atau teks terang menggunakan rumus $M = C^{d}\ mod$ N. Langkah terakhir adalah membandingkan teks terang yang didapatkan dari proses dekripsi pesan dengan pesan asli, jika pesan sama maka proses *cracking* algoritme RSA berhasil dilakukan.

Gambar 11 menunjukkan hasil implementasi dari program pembangkitan kunci algoritme RSA. Berdasarkan gambar tersebut didapatkan kunci privat adalah (67, 143) dan kunci publik adalah (43, 143) yang diperoleh dari pembangkitan kunci menggunakan bilangan prima $p = 11$ dan $q = 13$. Nilai *p* dan *q* tersebut dipilih berdasarkan syarat yang terdapat pada pembatasan masalah dalam penelitian

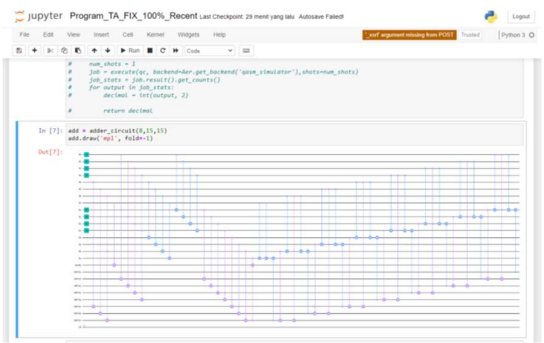
ini yaitu nilai p dan $q > 7$. Kunci privat dan kunci publik tersebut akan digunakan untuk menguji apakah program yang dibuat berhasil dalam melakukan cracking dari algoritme RSA.

Gambar 12 menunjukkan contoh sirkuit kuantum *Adder* yang diimplementasikan pada Qiskit dan dijalankan pada lingkungan komputer kuantum IBM. Pada implementasi algoritme Shor khususnya pada pemfaktoran bilangan N, diimplementasikan sirkuit kuantum *Modular Exponentiation* dengan skematik sebagaimana ditunjukkan pada Gambar 13. Sirkuit tersebut dibuat dengan membuat program pada Qiskit sehingga menghasilkan sirkuit kuantum pada Gambar 14. Sirkuit inilah yang kemudian dijalankan pada lingkungan komputer kuantum IBM Lab.

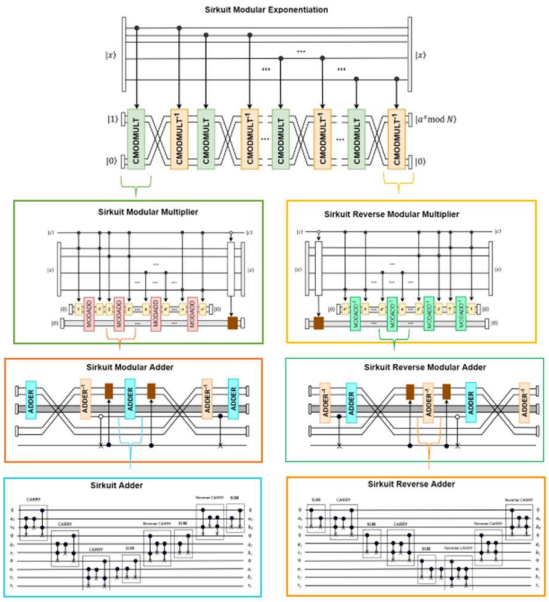
Gambar 15 menunjukkan hasil implementasi program keseluruhan berdasarkan *flowchart* yang dirumuskan sebelumnya. Ditunjukkan bahwa langkah yang dijalankan *User* yang pertama adalah memasukkan nilai $N = 143$, kunci publik = 43 dan *Ciphertext*. Program akan memilih bilangan x dengan syarat x lebih dari satu dan kurang dari N dengan nilai gcd(x, N) = 1. Nilai x yang dipilih adalah 38, digunakan untuk mencari periode dari $x^r \equiv 1\ (mod\ N)$ menggunakan sirkuit kuantum *modular exponentiation* dan didapatkan nilai periode adalah 10. Setelah dipastikan nilai periode memenuhi syarat, maka program akan menghitung nilai p dan q yang merupakan hasil pemfaktoran bilangan N dan didapatkan nilai $p = 11$ dan $q = 13$. Nilai hasil pemfaktoran tersebut digunakan untuk menghitung kunci privat dari algoritme RSA dan didapatkan kunci privat adalah (67, 143). Setelah dilakukan proses dekripsi pesan, maka didapatkan teks terang **PoltekSSN** dan hasil tersebut sama seperti pada Gambar 11.

```
<------------------------------------------>
        Program Pembangkitan Kunci
              Algoritma RSA
<------------------------------------------>
Ket: Masukkan 2 bilangan prima yang berbeda

Masukkan bilangan Prima Pertama (p) : 11
Masukkan bilangan Prima Kedua (q) : 13
<-> Nilai dari n adalah [n = pq] : 143
<-> Nilai dari r adalah [r = (p-1)(q-1)] : 120
<-> Kunci Public adalah : (43, 143)
<-> Kunci Private adalah : (67, 143)
<------------------------------------------>
       Program Enkripsi Algoritma RSA
<------------------------------------------>

Masukkan nilai e : 43
Masukkan nilai n : 143
Masukkan pesan yang ingin di enkripsi : PoltekSSN
Hasil Ciphertext : [115, 45, 69, 51, 140, 94, 73, 73, 78]
```

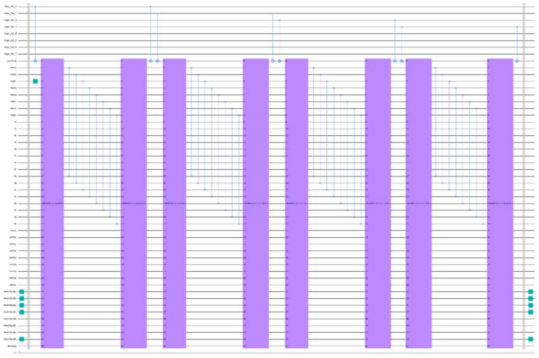
Gambar 11. Hasil Pembangkitan Kunci dan Proses Enkripsi RSA



Gambar 12. Implementasi Sirkuit *Adder*



Gambar 13. Skematik Sirkuit *Modular Exponentiation*



Gambar 14. Implementasi Sirkuit *Modular Exponentiation*

Pengujian program keseluruhan dilakukan dengan cara memasukkan nilai *N,* kunci publik (*e*) dan *Ciphertext* (*C*) pada program yang sudah dibuat berdasarkan perhitungan yang sudah dilakukan sebelumnya.

```
<------------------------------------------>
       Program Cracking Algoritma RSA
<------------------------------------------>
Masukkan Nilai N : 143
Masukkan Kunci Publik (e) : 43
Masukkan Cipherteks (C) : 115, 45, 69, 51, 140, 94, 73, 73, 78
Pilih bilangan 1 < x < N dengan syarat gcd(x,N) = 1
Nilai x yang dipilih : 38
1 . Hasil dari 38 ^ 1 mod 143 = 38
2 . Hasil dari 38 ^ 2 mod 143 = 14
3 . Hasil dari 38 ^ 3 mod 143 = 103
4 . Hasil dari 38 ^ 4 mod 143 = 53
5 . Hasil dari 38 ^ 5 mod 143 = 12
6 . Hasil dari 38 ^ 6 mod 143 = 27
7 . Hasil dari 38 ^ 7 mod 143 = 25
8 . Hasil dari 38 ^ 8 mod 143 = 92
9 . Hasil dari 38 ^ 9 mod 143 = 64
10 . Hasil dari 38 ^ 10 mod 143 = 1
Periode : 10
Nilai r memenuhi syarat
p : 11
q : 13
<-> Nilai dari ϕ(N) adalah [ϕ(N) = (p-1,q-1)] :  120
<-> Kunci Public adalah : (43, 143)
<-> Kunci Private adalah : (67, 143)
Hasil Plaintext : PoltekSSN
Pesan Awal Sama Dengan Pesan Hasil Dekripsi
```

Gambar 15. Hasil Implementasi Sistem *cracking* RSA

Nilai *N* yang diuji adalah 143, 187, 209, 221, 247 dan 253. Nilai N tersebut dipilih karena penelitian ini menggunakan sirkuit *Modular Exponentiation* 8 bit, dengan nilai N berkisar antara 0 hingga 256. Namun berdasarkan pengujian program *cracking* algoritme RSA, program yang dibuat menghasilkan *output* yang sesuai dengan syarat nilai p dan $q > 7$. Ketika nilai p dan $q \leq 7$ maka pesan hasil dekripsi berbeda dengan pesan yang dimasukkan di awal program, sehingga nilai N yang memenuhi syarat adalah nilai-nilai tersebut.

Berdasarkan pengujian yang telah dilakukan terhadap program *cracking* algoritme RSA, program algoritme Shor dan program sirkuit kuantum, didapatkan hasil bahwa program algoritme Shor menghasilkan *output* pemfaktoran bilangan *N* yang sesuai jika nilai *x* dan periode *r* memenuhi syarat.

Pengujian yang dilakukan terhadap sirkuit kuantum dengan rentang nilai 0-255 untuk semua kemungkinan pada sirkuit *adder* dan *reverse adder,* rentang nilai $0 \leq a, b < N$ untuk semua kemungkinan dengan nilai *N* = 143 pada sirkuit *modular adder* dan *reverse modular adder,* dan rentang nilai $0 \leq a, x < N$ untuk semua kemungkinan dengan nilai *N* = 143 pada sirkuit *modular multiplier*, *reverse modular multiplier* dan *modular exponentiation* didapatkan hasil bahwa sirkuit yang didesain menghasilkan *output* yang sesuai.

Hasil pengujian yang dilakukan pada progam keseluruhan, dengan cara memasukkan nilai *N* = 143, 187, 209, 221, 247 dan 253 didapatkan bahwa program berhasil dalam melakukan *cracking* algoritme RSA dengan cara memfaktorkan bilangan *N* yang dimasukkan dan menghitung kunci privat algoritme RSA yang digunakan untuk mendekripsi *ciphertext* yang dimasukkan di awal program.

5. KESIMPULAN

Berdasarkan hasil pengujian dan analisis yang telah dilakukan dalam penelitian ini, maka didapatkan kesimpulan sebagai berikut:

a) Dalam penelitian ini telah berhasil dilakukan implementasi sirkuit *modular exponentiation* 8 bit dengan MSB bernilai 1 yang terdiri dari sirkuit *adder*, *reverse adder, modular adder*, *reverse modular adder*, *modular multiplier*, dan *reverse modular multiplier* menggunakan *tools* Qiskit. Dari pengujian yang dilakukan terhadap setiap sirkuit dengan rentang nilai 0-255 untuk semua kemungkinan pada sirkuit *adder* dan *reverse adder,* rentang nilai $0 \leq a, b < N$ untuk semua kemungkinan dengan nilai *N* = 143 pada sirkuit *modular adder* dan *reverse modular adder,* dan rentang nilai $0 \leq a, x < N$ untuk semua kemungkinan dengan nilai *N* = 143 pada sirkuit *modular multiplier*, *reverse modular multiplier* dan *modular exponentiation* didapatkan hasil bahwa sirkuit yang didesain menghasilkan *output* yang sesuai. Kemudian untuk pengujian terhadap sirkuit *modular exponentiation* dalam mencari periode algoritme Shor dengan cara memasukkan nilai *N* = 143, 187, 209, 221, 247 dan 253 didapatkan hasil bahwa sirkuit *modular exponentiation* berhasil mendapatkan hasil periode yang sesuai dengan akurasi sebesar 100 %.
b) Berdasarkan hasil pengujian dan analisis yang telah dilakukan, algoritme Shor yang diimplementasikan pada penelitian ini telah berhasil dalam melakukan *cracking* algoritme RSA dengan cara memfaktorkan nilai *N* = 143, 187, 209, 221, 247 dan 253 dari algoritme RSA dan menghitung kunci privat yang digunakan untuk mendekripsi *ciphertext* yang dimasukkan di awal program.

REFERENSI

[1] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, no. July, pp. 1–20, 2018, doi: 10.22331/q-2018-08-06-79.

[2] H. A. Al-Mohammed, M. S. Al-Ali, and M. Alkaeed, "Quantum Computer Architecture from Non-Conventional Physical Simulation up to Encryption Cracking, Machine Learning Application, and More," *16th Int. Comput. Eng. Conf. ICENCO 2020*, pp. 17–24, 2020, doi: 10.1109/ICENCO49778.2020.9357401.

[3] P. V. Zahorodko, S. O. Semerikov, V. N. Soloviev, A. M. Striuk, M. I. Striuk, and H. M. Shalatska, "Comparisons of performance between quantum-enhanced and classical machine learning algorithms on the IBM Quantum Experience," *J. Phys. Conf. Ser.*, vol. 1840, no. 1, 2021, doi: 10.1088/1742-6596/1840/1/012021.

[4] S. Nayak, S. Nayak, and J. P. Singh, "An Introduction to Basic Logic Gates for Quantum Computer," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, no. 10, p. 2277, 2013, [Online]. Available: www.ijarcsse.com.

[5] P. N. Singh and S. Aarthi, "Quantum circuits - An application in qiskit-python," *Proc. 3rd Int. Conf. Intell. Commun. Technol. Virtual Mob. Networks, ICICV 2021*, no. Icicv, pp. 661–667, 2021, doi: 10.1109/ICICV50876.2021.9388498.

[6] E. H. Shaik and N. Rangaswamy, "Implementation of quantum gates based logic circuits using IBM qiskit," *Proc. 2020 Int. Conf. Comput. Commun. Secur. ICCCS 2020*, pp. 1–6, 2020, doi: 10.1109/ICCCS49678.2020.9277010.

[7] C. Gidney and M. Ekerå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits," *Quantum*, vol. 5, pp. 1–31, 2021, doi: 10.22331/Q-2021-04-15-433.

[8] www.geeksforgeeks.org, "Difference between Bits and Quantum Bits," 2021. .

[9] V. N. Chernega, O. V. Man'ko, and V. I.

Man'ko, "God Plays Coins or Superposition Principle for Classical Probabilities in Quantum Suprematism Representation of Qubit States," *J. Russ. Laser Res.*, vol. 39, no. 2, pp. 128–139, 2018, doi: 10.1007/s10946-018-9699-z.

[10] D. Aggarwal and U. Maurer, "Breaking RSA Generically is Equivalent to Factoring," *IEEE Trans. Inf. Theory*, vol. 62, no. 11, pp. 6251–6259, 2016, doi: 10.1109/TIT.2016.2594197.

[11] V. Mavroeidis, K. Vishi, M. D. Zych, and A. Jøsang, "The impact of quantum computing on present cryptography," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 3, pp. 405–414, 2018, doi: 10.14569/IJACSA.2018.090354.

[12] P. Nair, "Quantum Computing in Data Security : A Critical Assessment," 2020.

[13] K. Li and Q. yu Cai, "Practical Security of RSA Against NTC-Architecture Quantum Computing Attacks," *Int. J. Theor. Phys.*, vol. 60, no. 8, pp. 2733–2744, 2021, doi: 10.1007/s10773-021-04789-x.

[14] M. Sharma, V. Choudhary, R. S. Bhatia, S. Malik, A. Raina, and H. Khandelwal, "Leveraging the power of quantum computing for breaking RSA encryption," *Cyber-Physical Syst.*, vol. 7, no. 2, pp. 73–92, 2021, doi: 10.1080/23335777.2020.1811384.

[15] L. Jian *et al.*, "A survey on quantum cryptography," *Chinese J. Electron.*, vol. 27, no. 2, pp. 223–228, 2018, doi: 10.1049/cje.2018.01.017.

[16] X. Zhou and X. Tang, "Research and implementation of RSA algorithm for encryption and decryption," *Proc. 6th Int. Forum Strateg. Technol. IFOST 2011*, vol. 2, pp. 1118–1121, 2011, doi: 10.1109/IFOST.2011.6021216.

[17] V. Bhatia and K. R. Ramkumar, "An Efficient Quantum Computing technique for cracking RSA using Shor's Algorithm," *2020 IEEE 5th Int. Conf. Comput. Commun. Autom. ICCCA 2020*, pp. 89–94, 2020, doi: 10.1109/ICCCA49541.2020.9250806.

[18] A. Veliche, "Shor ' s Algorithm and Its Impact On Present-Day Cryptography," no. Math 4020, pp. 1–19, 2018.

[19] K. K. Soni and A. Rasool, "Cryptographic attack possibilities over RSA algorithm through classical and quantum computation," *Proc. Int. Conf. Smart Syst. Inven. Technol. ICSSIT 2018*, no. Icssit, pp. 11–15, 2018, doi: 10.1109/ICSSIT.2018.8748675.

[20] V. Gheorghiu and M. Mosca, "Benchmarking the quantum cryptanalysis of symmetric, public-key and hash-based cryptographic schemes," pp. 1–19, 2019, [Online]. Available: http://arxiv.org/abs/1902.02332.

[21] S. Aralikatti, "Quantum Computing: Challenges and Opportunities," *2021 4th Int. Conf. Electr. Comput. Commun. Technol. ICECCT 2021*, pp. 13–16, 2021, doi: 10.1109/ICECCT52121.2021.9616647.

[22] Qiskit.org, "Quantum State and Qubits." https://learn.qiskit.org/course/ch-states/introduction.

[23] C. Intila, B. Gerardo, and R. Medina, "A study of public key 'e' in RSA algorithm," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 482, no. 1, pp. 0–9, 2019, doi: 10.1088/1757-899X/482/1/012016.

[24] R. LaPierre, "Shor Algorithm," pp. 177–192, 2021, doi: 10.1007/978-3-030-69318-3_13.

[25] D. Hutama, "Shor ' s Algorithm," 2018.

[26] T. Alexander *et al.*, "Qiskit pulse: Programming quantum computers through the cloud with pulses," *Quantum Sci. Technol.*, vol. 5, no. 4, 2020, doi: 10.1088/2058-9565/aba404.

[27] V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," vol. 54, no. 1, pp. 147–153, 1996.

[28] H. T. Larasati and H. Kim, "Simulation of modular exponentiation circuit for shor's algorithm in qiskit," *Proceeding 14th Int. Conf. Telecommun. Syst. Serv. Appl. TSSA 2020*, pp. 3–9, 2020, doi: 10.1109/TSSA51342.2020.9310794.

[29] A. Dennis, B. H. Wixom, and R. M. Roth, *Systems Analysis & Design*. 2012.