THUẬT TOÁN LƯỢNG TỬ PHÁ MÃ RSA

QUANTUM ALGORITHM BREAKING RSA

Dụng Văn Lữ*, Nguyễn Văn Linh, Nguyễn Thị Ly Ly, Huỳnh Phương Anh, Huỳnh Bảo Nguyên

Trường Đại học Sư phạm - Đại học Đà Nẵng, Đà Nẵng, Việt Nam[1]

*Tác giả liên hệ / Corresponding author: dvlu@ued.udn.vn

(Nhận bài / Received: 28/12/2022; Sửa bài / Revised: 03/02/2023; Chấp nhận đăng / Accepted: 07/02/2023)

Tóm tắt - Trong bài báo này, nhóm tác giả đề xuất sử dụng nền tảng phần mềm hỗ trợ làm việc với máy tính lượng tử Qiskit để nghiên cứu các thuật toán lượng tử sau khi đối sánh các đặc tính của bốn nền tảng phổ biến là Forest, ProjectQ, QDK và Qiskit. Cùng với đó, nhóm tác giả khai triển hợp số $N = 15$ thành các thừa số bằng thuật toán lượng tử Shor và chạy chúng trên máy tính lượng tử IBM thông qua cloud của nền tảng Qiskit. Kết quả cho thấy, với một bài toán gần như được xem là bất khả thi đối với thuật toán cổ điển lại có thể dễ dàng được giải bằng thuật toán lượng tử nhờ các tính chất lượng tử thông qua việc chỉ ra các tính chất và "hành xử" của vật lí lượng tử trong từng bước của thuật toán.

Từ khóa - Thuật toán lượng tử Shor; máy tính lượng tử; Qiskit; phân tích thừa số; RSA

Abstract - In this paper, the authors propose to use a software platform Qiskit that supports working with quantum computers to study quantum algorithms by comparing the characteristics of four popular platforms: Forest, ProjectQ, QDK and Qiskit. Along with that, The authors factorize $N = 15$ into factors using the quantum Shor's algorithm and run them on IBM quantum computers through the cloud of the Qiskit platform. The results show that a problem that is almost impossible for classical algorithms can easily be solved by quantum algorithms thanks to quantum properties by showing the properties and "behavior" of quantum physics in each step of the algorithm.

Key words - Quantum Shor's algorithm; quantum computer; Qiskit; factoring; RSA

1. Giới thiệu

Cho đến nay, mật mã RSA (lấy từ 3 chữ cái đầu của tên 3 tác giả Rivest, Shamir và Adleman) [1] vẫn an toàn so với các cuộc tấn công phá mã bằng thuật toán hoạt động trên máy tính cổ điển tốt nhất hiện nay. Thuật toán RSA sử dụng hai khoá: Khóa công khai (public key) và khóa bí mật (private key). Khoá công khai gồm số tự nhiên N, e và bản mã c. Khoá bí mật là d. Nếu phân tích $N = pq$ thì tính được khoá bí mật d nhờ $ed = mod(p-1)(q-1)$, khi đó, có thể tìm được bản rõ $m = c^d \ mod \ N$, (*mod*: là phép chia lấy dư).

Để phá mã RSA, vấn đề trọng tâm và khó nhất là phải phân tích một hợp số N (được công khai) thành cặp thừa số nguyên tố p và q duy nhất, nghĩa là $pq = N$. Độ khó của nó càng cao khi N càng lớn và p, q là các số nguyên tố lớn gần nhau. Chính vì vậy, RSA đang được sử dụng phổ biến trong thương mại điện tử và dùng để tạo chữ ký số cho văn bản [2].

Tuy nhiên, năm 1994 Shor đã đề xuất một thuật toán lượng tử có thể phá mã RSA, tức là có thể phân tích hợp số N thành các thừa số nguyên tố p và q chỉ trong thời gian đa thức, nghĩa là nhanh hơn nhiều so với thuật toán cổ điển [3]. Thuật toán lượng tử là thuật toán có bản chất lượng tử hoặc dựa trên một tính năng thiết yếu của tính toán lượng tử bao gồm chồng chập hoặc/và vướng víu lượng tử. Một máy tính lượng tử được dự đoán sẽ hoạt động tốt hơn máy tính cổ điển trong một số chức năng nhất định [4].

Được cho là sẽ thay đổi hoàn toàn các quy tắc của máy tính cổ điển, máy tính lượng tử được kỳ vọng sẽ mang lại khả năng tính toán gấp hàng triệu lần máy tính thông thường. Trong thời gian vừa qua, các công ty như IBM, Google, Microsoft, D-wave đều công bố các bước tiến mới đối với máy tính lượng tử, đặc biệt IBM đã bắt đầu đưa máy tính lượng tử 5 qubit lên cloud để người dùng trên thế giới có thể làm quen và thử nghiệm khả năng vượt trội của nó [5]. Bên cạnh đó, các ngôn ngữ lập trình lượng tử và bốn nền tảng phát triển phần mềm dựa trên máy tính lượng tử cũng được các công ty đặc biệt chú ý [6].

Trong bài báo này, giới thiệu các nền tảng phần mềm hỗ trợ làm việc với máy tính lượng tử, nó cung cấp các công cụ để tạo và thao tác các thuật toán lượng tử trên thiết bị lượng tử thực hay phần mềm giả lập. Bằng cách so sánh các đặc tính của bốn nền tảng phần mềm lượng tử Forest (pyQuil) từ Rigetti, ProjectQ từ ETH Zurich, QDK (Q#) của Microsoft và Qiskit từ IBM, nhóm tác giả sẽ cung cấp cái nhìn tổng quát về chúng dựa trên các khía cạnh về phần cứng, yêu cầu và cài đặt, ngôn ngữ lập trình, cú pháp cũng như trình mô phỏng lượng tử. Từ đó, tính toán thuật toán Shor phân tích số 15 để làm rõ các tính chất lượng tử và chỉ ra bản chất lượng tử và triển khai trên máy tính lượng tử IBM thông qua icloud của trình mô phỏng Qiskit. Thuật toán này được thực hiện trên quy mô lớn trong một máy tính lượng tử và trả về kết quả là hai thừa số nguyên tố với độ chính xác cao. Kết quả cho thấy, với một bài toán gần như được xem là bất khả thi đối với thuật toán cổ điển lại có thể dễ dàng bị phá giải bằng thuật toán lượng tử Shor.

2. Các nền tảng thực thi thuật toán lượng tử

Việc triển khai các thuật toán lượng tử trên máy tính lượng tử là điều khó thực hiện đối với các Quốc gia hay phòng thí nghiệm chưa có máy tính lượng tử. Do đó, việc triển khai các thuật toán lượng tử thường được thực hiện trên các thiết bị mô phỏng lượng tử, các trình giả lập hay nền tảng phần mềm tính toán lượng tử chạy qua cloud của máy tính lượng tử thực.

[1] The University of Danang - University of Science and Education, Danang, Vietnam (Dung Van Lu, Nguyen Van Linh, Nguyen Thi Ly Ly, Huynh Phuong Anh, Huynh Bao Nguyen)

Gần đây, có nhiều nền tảng phần mềm lượng tử do các công ty lớn nghiên cứu điện toán lượng tử phát triển có thể làm "bối rối" cho những người lựa chọn sử dụng. Trong đó, có bốn nền tảng phần mềm phổ biến, Forest (pyQuil) của Forest, Qiskit của IBM, ProjectQ của ETH Zurich và Bộ công cụ phát triển lượng tử (QDK, Q#) Microsoft [6] cho phép các nhà nghiên cứu sử dụng thiết bị lượng tử thực và mô phỏng. Nhóm tác giả tóm lược các yêu cầu cài đặt, ngôn ngữ lập trình lượng tử, phần cứng lượng tử, kích thước mô phỏng và đặc trưng mô phỏng lượng tử cho từng nền tảng được thể hiện trong Bảng 1.

Sau khi thử nghiệm các nền tảng và ngoài những phân tích về điểm mạnh của Qiskit trong bài báo [7], Qiskit có những ưu thế hơn, như kích thước mô phỏng lên đến 65 qubit qua IBM và tạo mã ngôn ngữ assembly lượng tử QASM mà các nền tảng khác không có. Qiskit dùng ngôn ngữ lập trình lượng tử Python gần gũi với ngôn ngữ Python cổ điển đang dùng phổ biến hiện nay, còn các nền tảng khác sử dụng ngôn ngữ lập trình lượng tử nên gây bất tiện cho người dùng, đặc biệt là người bắt đầu nghiên cứu thuật toán lượng tử. Do đó, nhóm tác giả lựa chọn sử dụng nền tảng Qiskit của IBM để triển khai thuật toán lượng tử Shor.

Bảng 1. *Các nền tảng phổ biến để thực thi thuật toán lượng tử*

Nền tảng	**Forest**	**ProjectQ**	**QDK**	**Qiskit**
Nhà phát triển	Rigetti	ETH Zurich	Microsoft	IBM
Trang chủ	https://github.com/rigetti	https://projectq.ch/	https://github.com/microsoft/Quantum	https://qiskit.org/
Bảng phát hành đầu tiên	01/2017	01/2017	01/2018	03/2017
Mã nguồn mở	X	X	X	X
Hệ điều hành hỗ trợ	Mac, Windowns, Linux	Mac, Windowns, Linux	Mac, Windowns, Linux	Mac, Windowns, Linux
Yêu cầu	Python 3.7+, Anaconda	Python 2 hoặc 3	Visual Studio Code	Python 3.7+, Jupyter, Anaconda 3, Notebooks
Ngôn ngữ lập trình lượng tử	pyQuil	ProjectQ	Q#	Python
Phần cứng lượng tử	QPU Aspen 11 (40 qubits) Aspen-M-2 (80 qubits)	Có thể được kết nối với chương trình phụ trợ của IBM	Cần tải xuống và cài đặt NET Core SDK 3.1 trở lên. Năm 2022, đã cập nhật .NET 6 thay vì .NET Core 3.1 và Visual Studio 2022 thay vì VS 2019.	IBMQX2 (5 qubits), IBMQX4 (5 qubits), IBMQX5 (16qubits), QS1_1 (20 qubits)
Kích thước mô phỏng	30 qubit với hầu hết các khóa API cho QVM	~28 qubit cục bộ	30 qubit cục bộ, 40 qubit qua đám mây Azure	~25 qubit cục bộ, 65 qubit qua IBM
Đặc trưng	Tạo mã Quil, ví dụ thuật toán trong Grove, trình biên dịch cho cấu trúc liên kết cụ thể, tính năng nhiễu trong trình mô phỏng, kênh cộng đồng Slack	Xây dựng vi mạch, kết nối với các mô-đun máy chủ IBM, tính khả dụng của một số chương trình gắn với IBM	Các thuật toán và ví dụ tích hợp sẵn	Tạo mã QASM, trình biên dịch cấu trúc liên kết cụ thể, kênh cộng đồng Slack, bảng mạch, thư viện Aqua

3. Triển khai thuật toán lượng tử Shor

3.1. Bài toán phân tích thừa số nguyên tố

Như ta biết, nếu cho *p, q* là hai số nguyên tố thì ta dễ dàng tính được $N = pq$ với máy tính cổ điển thông thường. Ngược lại, cho một hợp số *N*, cần tìm cặp số nguyên tố *p* và *q* sao cho $N = pq$. Còn gọi là bài toán phân tích thừa số nguyên tố.

Nhóm tác giả sẽ làm rõ cách mà thuật toán lượng tử hoạt động và giải quyết dựa trên tính chất vật lí lượng tử. Đồng thời triển khai trên nền tảng Qiskit.

3.2. Giải pháp cổ điển

Để phân tích hợp số *N* thành hai số nguyên tố p và q, thuật toán cổ điển nổi tiếng nhất yêu cầu thời gian siêu đa thức bậc $\exp\left[(\log N)^{1/3}\log(\log N^{2/3})\right]$ [4].

Với số *N* lớn thì thuật toán cổ điển hầu như không thể thực hiện được nên hệ thống mật mã RSA dựa vào việc phân tích số nguyên tố cho đến nay chưa bị bẻ khoá.

3.3. Giải pháp lượng tử: Thuật toán Shor

Giải pháp lượng tử dựa trên ý tưởng chính của Shor (còn gọi là thuật toán Shor) giải bài toán trên trong thời gian đa thức $O(\log N^3)$ [4]. Thuật toán gồm hai phần: Phần thứ nhất là biến bài toán phân tích thừa số thành bài toán tìm chu kỳ của một hàm được triển khai theo kiểu cổ điển. Phần thứ hai là tìm chu kỳ bằng cách sử dụng biến đổi Fourier lượng tử và thể hiện ưu việt tính toán lượng tử.

3.3.1. Phần cổ điển

Phần cổ điển có thể được thực hiện trên máy tính cổ điển với những thuật toán cổ điển đã biết, gồm các bước:

Bước 1: Chọn một số ngẫu nhiên $a < N$;

Bước 2: Tính ước chung lớn nhất giữa *a* và *N* (kí hiệu: $\gcd(a, N)$) bằng thuật toán cổ điển Euclide;

Bước 3: Nếu $\gcd(a, N) \neq 1$ thì tìm được thừa số của *N*, $p = \gcd(a, N)$ và $q = \frac{N}{p}$;

Bước 4: Nếu không thoả mãn thì tìm chu kỳ r của hàm $f(x) = a^x mod N$, tức là $f(x + r) = f(x)$. Đối với số kí tự lớn thì thuật toán cổ điển thực hiện bước này rất khó và thời gian chạy lâu. Thuật toán lượng tử chỉ tính trong thời gian đa thức;

Bước 5: Nếu r là số lẻ hoặc $\gcd\left(a^{\frac{r}{2}}, N\right) \neq 1$, quay lại bước đầu tiên;

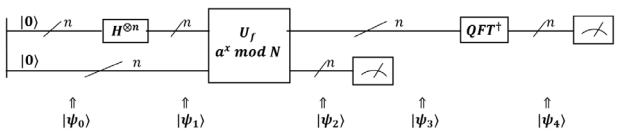
Bước 6: Nếu r là số chẵn thì: $p = \gcd\,(a^{\frac{r}{2}} + 1, N)$ và $q = \gcd\,(a^{\frac{r}{2}} - 1,\ N)$.

3.3.2. Phần lượng tử, chương trình con tìm chu kỳ

Ở phần trên, bước có độ phức tạp cao nhất là bước tìm chu kỳ r của hàm $f(x)$. Để tìm chu kỳ của một hàm số, có thể biến đổi Fourier hàm số này, khi đó kết quả biến đổi Fourier sẽ cực đại tại các giá trị s/r với s là các số nguyên. Do đó, bước này có thể được thực hiện bằng tính toán lượng tử như sau. Hàm $f(x)$ là hàm tuần hoàn có thể nhận tối đa N giá trị, và có thể được biểu diễn bằng Q trạng thái cơ sở, để tối ưu thì Q được chọn sao cho $N^2 \leq Q \leq 2N^2$ [4]. Ta có thể trình bày ngắn gọn như sau:

1. Thiết lập một hệ vật lí gồm hai thanh ghi cạnh nhau được sơ đồ hoá như Hình 1, khởi tạo hai thanh ghi đều ở trạng thái nghỉ $|0\rangle$. Số qubit n là số nguyên nhỏ nhất lớn hơn logarit cơ số 2 của N, sao cho $Q = 2^n > N^2$:

$$|\psi_0\rangle = |0\rangle^{\otimes n}|0\rangle^{\otimes n} \tag{1}$$



Hình 1. *Sơ sồ mạch thuật toán lượng tử Shor*

2. Các qubit thanh ghi thứ nhất qua cổng Hadamard (cổng H) [5] tương ứng, còn thanh ghi thứ hai vẫn ở trạng thái nghỉ $|0\rangle$. Trạng thái của hệ hai thanh ghi sau bước này:

$$|\psi_0\rangle \xrightarrow{H^{\otimes n}\otimes I^{\otimes n}} |\psi_1\rangle = Q^{-\frac{1}{2}} \sum_{x=0}^{Q-1} |x\rangle|0\rangle \tag{2}$$

Ở đây, $|x\rangle$ là trạng thái n qubit của thanh ghi thứ nhất được kí hiệu theo dạng thập phân, ví dụ: $|0\cdots0101\rangle \equiv |5\rangle$; $|0\rangle \equiv |0\cdots0\rangle$: Trạng thái nghỉ của n qubit.

3. Xây dựng hàm $f(x) = a^n mod\ N$, bằng cách áp dụng toán tử lượng tử đơn nguyên được điều khiển bởi thanh ghi thứ nhất U_f vào thanh ghi thứ hai và thu được trạng thái mới của hệ là

$$|\psi_1\rangle \overset{U_f}{\rightarrow} |\psi_2\rangle = Q^{-\frac{1}{2}} \sum\nolimits_x |x\rangle|a^x mod\ N\rangle \tag{3}$$

Nếu $f(x)$ là hàm với chu kỳ r, thì:

$$f(x_0) = f(x_0 + r) = \cdots = f(x_0 + br) = \cdots$$

với $x_0 < r$, $x_0 + br < Q$ nên giá trị lớn nhất của b là số nguyên của $\lfloor (Q - 1 - x_0)/r \rfloor$.

4. Thực hiện đo ở thanh ghi thứ hai. Trạng thái của hệ sẽ sụp đổ thành:

$$|\psi_3\rangle = \frac{1}{\sqrt{Q/r}} \sum_{b=0}^{Q/r-1} |x_0 + br\rangle|\boldsymbol{f(x_0)}\rangle \tag{4}$$

5. Áp dụng biến đổi Fourier lượng tử lên thanh ghi thứ nhất:

$$|\psi_4\rangle = \frac{1}{\sqrt{Q}} \sum_{y=0}^{Q-1} e^{\frac{2\pi i}{Q}(x_0+br)y} |y\rangle \tag{5}$$

với y là một trong r số nguyên dương mod N sao cho $\frac{y.r}{Q}$ là số nguyên s nào đấy, tức là $y = \frac{sQ}{r}$, ta viết lại trạng thái:

$$|\psi_4\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} e^{\frac{4\pi i}{r}s} \left|\frac{sQ}{r}\right\rangle \tag{6}$$

6. Thực hiện phép đo trạng thái ở thanh ghi thứ nhất để tìm r. Kết quả đo là trạng thái $|C\rangle$, nghĩa là $\frac{sQ}{r} = C \rightarrow \frac{s}{r} = \frac{C}{Q}$. Thuật toán sẽ phân tích thừa số s/r cho đến phân số tối giản, trong đó xác suất để s là bội số của 1/r là cao. Giá trị ở mẫu số chính là r.

7. Thử lại, bằng tính toán cổ điển, xem nếu $f(x) = f(x + 1/s)$ thì kết thúc. Nếu không thì thử với các giá trị là 1/s với các s nguyên khác nhau bằng tính toán cổ điển, nếu một trong các giá trị này thỏa mãn thì kết thúc.

8. Nếu không thoả mãn thì lặp lại từ bước đầu tiên.

Tiếp theo, nhóm tác giả dùng thuật toán lượng tử Shor phân tích thừa số N = 15 nhằm chỉ rõ "hành xử" của tính chất lượng tử trong các bước tính toán. Do đó, chỉ tập trung vào phần lượng tử của thuật toán để hiểu rõ cách mà thuật toán lượng tử thể hiện tính "quyền năng" của nó.

3.4. Phân tích N = 15 bằng thuật toán lượng tử

Với N = 15, giả sử số a ngẫu nhiên được chọn: $a = 2 \rightarrow \gcd(2,15) = 1$.

Bây giờ sử dụng thuật toán lượng tử Shor để tìm chu kỳ r, vì $a^0 \equiv 1\ (mod\ 15)$ nên $a^r \equiv 1\ (mod\ 15)$.

Tạo hai thanh ghi vật lí với số qubit n phải thỏa mãn điều kiện $N^2 < 2^n < 2N^2 \rightarrow n = 8, Q = 2^n = 256$.

1. Khởi tạo hai thanh ghi ở trạng thái nghỉ $|0\rangle$ như biểu thức (1), đây là trạng thái năng lượng thấp nhất trong hệ lượng tử.

2. Thanh ghi thứ nhất qua cổng Hadamard

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle|0\rangle = \frac{1}{\sqrt{256}} \sum_{x=0}^{255} |x\rangle|0\rangle \tag{7}$$

Như vậy, từ trạng thái $|0\rangle$ ban đầu cổng Hadamard tạo ra trạng thái chồng chập đều, thực chất của cổng này là phép biến đổi Fourier lượng tử từ không gian tính toán (cơ sở Z) sang không gian Fourirer (cơ sở X) trên từng qubit đơn.

3. Tác dụng biến đổi đơn nguyên U_f lên trạng thái của thanh ghi thứ hai thành hàm $f(x) = a^x\ mod\ N$:

$$\begin{aligned}
|\psi_2\rangle &= \frac{1}{\sqrt{256}} \sum_{x=0}^{255} |x\rangle|2^x\ mod\ 15\rangle = \\
&= \frac{1}{\sqrt{256}} (|0\rangle|\mathbf{1}\rangle + |1\rangle|\mathbf{2}\rangle + |2\rangle|\mathbf{4}\rangle + |3\rangle|\mathbf{8}\rangle + |4\rangle|\mathbf{1}\rangle \\
&\qquad + \cdots + |254\rangle|\mathbf{4}\rangle + |255\rangle|\mathbf{8}\rangle) = \\
&= \frac{1}{\sqrt{256}} [(|0\rangle + |4\rangle + |8\rangle + |10\rangle + \cdots + |252\rangle)|\mathbf{1}\rangle + \\
&\quad + (|1\rangle + |5\rangle + |9\rangle + |11\rangle + \cdots + |253\rangle)|\mathbf{2}\rangle + \\
&\quad + (|2\rangle + |6\rangle + |10\rangle + |12\rangle + \cdots + |254\rangle)|\mathbf{4}\rangle + \\
&\quad + (|3\rangle + |7\rangle + |11\rangle + |13\rangle + \cdots + |255\rangle)|\mathbf{8}\rangle\ ].
\end{aligned} \tag{8}$$

Rõ ràng ta thấy chu kỳ $r = 4$, tuy nhiên chúng ta vẫn tiếp tục thực hiện các phép tính để hiểu được r xuất hiện ở

đâu và làm thế nào để thu được. Nhóm tác giả thực hiện nhóm lại các trạng thái thành phần mà có cùng giá trị ở thanh ghi thứ hai. Ở đây, các giá ở thanh ghi thứ hai (giá trị $f(x)$) được in đậm lên để dễ phân biệt với giá trị thanh ghi thứ nhất. Ta nhận thấy rằng, U_f tạo ra kết nối có điều kiện giữa các qubit trong thanh ghi thứ nhất và qubit thanh ghi thứ hai tạo điều kiện cho giao thoa lượng tử và kết nối lượng tử. Ví dụ nếu ta đo thanh ghi thứ nhất cho giá trị là 6, thì ta biết chắc rằng thanh ghi trạng thái thứ hai là 4. Nếu đo thanh ghi thứ hai cho giá trị 4 thì thanh ghi thứ nhất chỉ có thể là chồng chập các trạng thái có giá trị (2+4b), trong đó b là số nguyên.

Để tính chu kỳ của hàm f, ta phải kiểm tra hàm tại tất cả các điểm đồng thời. Tuy nhiên, vật lí lượng tử không cho phép chúng ta truy cập trực tiếp tất cả thông tin này. Do đó, ta thực hiện đo trạng thái ở bước 4.

4. Thực hiện đo ở thanh ghi thứ hai. Chú ý rằng trong cơ học lượng tử, phép đo nghĩa là chiếu nó lên một trạng thái nào đó. Giả sử rằng chiếu lên phép đo trạng thái $|\mathbf{4}\rangle$. Trạng thái của hệ sẽ sụp đổ về thành:

$$|\psi_3\rangle = \frac{1}{\sqrt{256/r}} \sum_{y=0}^{256/r-1} |2 + br\rangle|\mathbf{4}\rangle = \\ = \frac{1}{\sqrt{64}}(|2\rangle + |6\rangle + |10\rangle + \cdots + |254\rangle)|\mathbf{4}\rangle. \quad (9)$$

Ở thanh ghi thứ nhất vẫn còn chồng chập trạng thái, ta thực hiện biến đổi để đưa về trạng thái có xác suất cao. Điều này có thể thực hiện nhờ phép biến đổi Fourier lượng tử.

5. Áp dụng biến đổi Fourier lượng tử (5) lên thanh ghi thứ nhất, với $y = \frac{sQ}{r}$, ta viết lại trạng thái:

$$|\psi_4\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} e^{\frac{4\pi i}{r}s} \left|\frac{256s}{r}\right\rangle \quad (10)$$

Như vậy, sau bước này thì chu kỳ r mới xuất hiện ở trạng thái, pha và cả ở hệ số chuẩn hoá.

6. Sau khi biến đổi Fourier, máy tính lượng tử sẽ thực hiện phép đo trạng thái ở thanh ghi thứ nhất. Như vậy phép đo chỉ thực hiện lên không gian trạng thái trong cơ sở tính toán (cơ sở Z). Kết quả đo trạng thái $|C\rangle$ chỉ có thể là một trong trạng thái với pha tương ứng như sau:

$s = 0 \rightarrow \left|\frac{sQ}{r}\right\rangle = |0\rangle$, pha: $e^{\frac{4\pi i}{r}s} = e^0 = 1$

$s = 1 \rightarrow \left|\frac{sQ}{r}\right\rangle = |64\rangle$, pha: $e^{\frac{4\pi i}{r}s} = e^{\pi i} = -1$

$s = 2 \rightarrow \left|\frac{sQ}{r}\right\rangle = |128\rangle$, pha: $e^{\frac{4\pi i}{r}s} = e^{2\pi i} = 1$

$s = 3 \rightarrow \left|\frac{sQ}{r}\right\rangle = |192\rangle$, pha: $e^{\frac{4\pi i}{r}s} = e^{3\pi i} = -1$

Xác suất đo được các trạng thái trên là như nhau, 25%:

$$|\psi_4\rangle = \frac{1}{2}(|0\rangle - |64\rangle + |128\rangle - |192\rangle) \quad (10)$$

Nếu trong một lần đo, phép đo "may mắn", nghĩa là thu được trạng thái $|C\rangle \rightarrow |64\rangle, |192\rangle$ thì ta nhận được giá trị r, bằng cách thực hiện tối giản phân số C/Q, ví dụ

$$\frac{C}{Q} = \frac{192}{256} = \frac{s}{r} = \frac{3}{4} \rightarrow r = 4$$

thỏa mãn lí thuyết. Và ta tính được: $p = \gcd(2^{4/2} + 1{,}15) = 5 \rightarrow q = 3$. Nếu phép đo cho trạng thái $|C\rangle = |128\rangle$ thì $r = 2$, ta vẫn tính được: $p = \gcd(2^{2/2} + 1{,}15) = 3 \rightarrow q = 5$.

Nếu không "may mắn", thì ta thực hiện đo lại và thuật toán đưa vào vòng lặp con.

Ta thấy thuật toán dựa vào khả năng máy tính lượng tử ở nhiều trạng thái đồng thời (sự chồng chập), trong khi một phép đo sẽ chỉ mang lại một trong tất cả các giá trị có thể và phá hủy tất cả các giá trị khác. Do đó, chúng phải được triển khai "nhanh" với một số cổng lượng tử thấp, tạo sự chồng chập của các trạng thái (nhờ cổng Hadamard), tìm trạng thái có xác suất cao (nhờ biến đổi Fourier lượng tử).

Như vậy, khi thực hiện trên máy tính lượng tử, ta mới nhận kết quả từ bước cuối cùng này, giá trị đo được thể hiện ở dạng xác suất, ta phải biện luận pha để thu được kết quả mong đợi. Điều này sẽ được làm rõ hơn ở Mục 3.5 triển khai thực tế trên máy tính lượng tử.

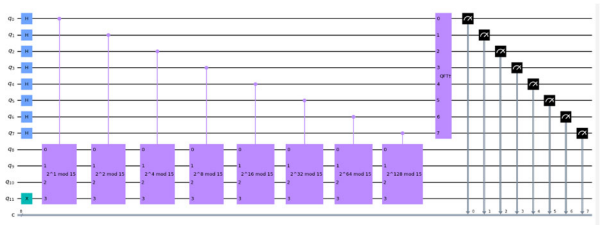
3.5. Triển khai thuận toán trên nền tảng Qiskit

Tiếp theo, nhóm tác giả sử dụng code Python [5] và một số bước cải tiến để triển khai thuật toán lượng tử này với $a = 2$ và $N = 15$. Mở đầu là các câu lệnh để xây dựng và xuất sơ đồ mạch thuật toán.

```
        import matplotlib.pyplot as plt
        import numpy as np
        from qiskit import QuantumCircuit, Aer,
transpile, assemble
        from qiskit.visualization import plot_histogram
        from math import gcd
        from numpy.random import randint
        import pandas as pd
        from fractions import Fraction
        print("Imports Successful")
        def c_amod15(a, power):
            U = QuantumCircuit(4)
            for iteration in range(power):
                    U.swap(0,1)
                    U.swap(1,2)
                    U.swap(2,3)
                    for q in range(4):
                        U.x(q)
            U = U.to_gate()
            U.name = "%i^%i mod 15" % (a, power)
            c_U = U.control()
            return c_U
        n_count = 8
        a = 2
        def qft_dagger(n):
            """n-qubit QFTdagger the first n qubits in
circ"""
            qc = QuantumCircuit(n)
            for qubit in range(n//2):
                qc.swap(qubit, n-qubit-1)
            for j in range(n):
                for m in range(j):
                    qc.cp(-np.pi/float(2**(j-m)), m, j)
                qc.h(j)
            qc.name = "QFT†"
            return qc
        qc = QuantumCircuit(n_count + 4, n_count)
        for q in range(n_count):
            qc.h(q)
```

```
        qc.x(3+n_count)
        for q in range(n_count):
            qc.append(c_amod15(a, 2**q),
                      [q]  +  [i+n_count  for  i  in
range(4)])
        qc.append(qft_dagger(n_count), range(n_count))
        qc.measure(range(n_count), range(n_count))
                        qc.draw(fold=-1)
```

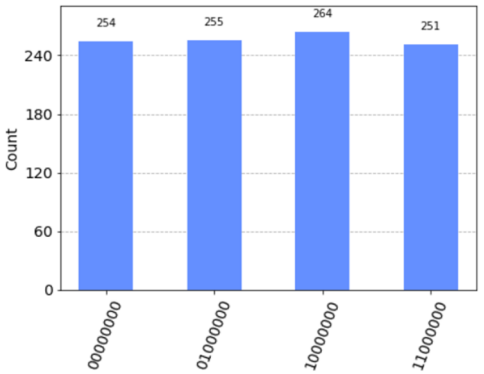
Ý nghĩa đoạn code trên để đưa vào các thông số, các cổng lượng tử và lệnh hiển thị sơ đồ mạch của thuật toán này. Kết quả chạy đoạn code trên qua nền tảng Qiskit xuất mô hình mạch như Hình 2.



Hình 2. *Mô hình mạch thuật toán Shor # ****

Tiếp theo, ta thực hiện phép đo trạng thái. Kết quả nền tảng phần mềm Qiskit cho kết quả như Hình 3, các trạng thái đo được |0⟩, |64⟩, |128⟩ và |192⟩ biểu diễn ở dạng nhị phân. Về tính toán lí thuyết thì xác suất các trạng thái là như nhau, nhưng kết quả đó có sự khác nhau là do nhiễu lượng tử. Thiết bị càng nhạy, càng vi mô thì khả năng nhiễu càng cao. Trong đó xác suất của trạng thái |10000000⟩ ≡ |128⟩ có xác suất lớn nhất.

```
aer_sim = Aer.get_backend('aer_simulator')
t_qc = transpile(qc, aer_sim)
qobj = assemble(t_qc)
results = aer_sim.run(qobj).result()
counts = results.get_counts()
plot_histogram(counts)
```



Hình 3. *Kết quả đo được khi chạy thuật toán Shor*

Thực hiện đo pha của các trạng thái, xuất ra các kết quả pha ứng với các giá trị chu kỳ *r*. Kết quả thể hiện ở Hình 4.

```
        rows, measured_phases = [], []
        for output in counts:
            decimal = int(output, 2)  # Convert (base
2) string to decimal
            phase   =   decimal/(2**n_count)   #   Find
corresponding eigenvalue
            measured_phases.append(phase)
            rows.append([f"{output}(bin)=
{decimal:>3}(dec)",
                         f"{decimal}/{2**n_count}=
{phase:.2f}"])
        headers=["Register Output", "Phase"]
        df = pd.DataFrame(rows, columns=headers)
        print(df)
        Fraction(0.666)
        Fraction(0.666).limit_denominator(15)
        rows = []
        for phase in measured_phases:
            frac=
Fraction(phase).limit_denominator(15)
            rows.append([phase,
f"{frac.numerator}/{frac.denominator}",
frac.denominator])
        headers=["Phase", "Fraction", "Guess for r"]
        df = pd.DataFrame(rows, columns=headers)
        print(df)
```

```
Imports Successful
             Register Output           Phase
0  11000000(bin) = 192(dec)  192/256 = 0.75
1  10000000(bin) = 128(dec)  128/256 = 0.50
2  00000000(bin) =   0(dec)    0/256 = 0.00
3  01000000(bin) =  64(dec)   64/256 = 0.25
   Phase Fraction  Guess for r
0   0.75      3/4            4
1   0.50      1/2            2
2   0.00      0/1            1
3   0.25      1/4            4
```

Hình 4. *Pha và chu kỳ r đo được khi chạy thuật toán Shor*

Các kết quả trên chưa đưa ra chu kỳ cũng như giá trị thừa số nguyên tố. Và chú ý rằng giá trị *r* đo được có ba giá trị (4, 2, 1) nên để xác định giá trị *p* và *q* chuẩn xác ta cần đặt điều kiện cho *p* và *q*. Sau khi thiết lập xong mạch thuật toán (vị trí được đánh dấu #***), nhóm tác giả cải tiến một số câu lệnh như sau:

```
        while True:
            aer_sim = Aer.get_backend('aer_simulator')
            t_qc = transpile(qc, aer_sim)
            qobj = assemble(t_qc)
            results = aer_sim.run(qobj).result()
            counts = results.get_counts()
            plot_histogram(counts)
            rows, measured_phases = [], []
            for output in counts:
                decimal  =  int(output,  2)   #  Convert
(base 2) string to decimal
                phase  =  decimal/(2**n_count)   #  Find
corresponding eigenvalue
                measured_phases.append(phase)
                rows.append([f"{output}(bin)=
{decimal:>3}(dec)",
                             f"{decimal}/{2**n_count}=
{phase:.2f}"])
            headers=["Register Output", "Phase"]
            df = pd.DataFrame(rows, columns=headers)

            Fraction(0.666)
            Fraction(0.666).limit_denominator(15)
            rows = []
            for phase in measured_phases:
                frac=
Fraction(phase).limit_denominator(15)
                rows.append([phase,
f"{frac.numerator}/{frac.denominator}",
frac.denominator])
            headers=["Phase",  "Fraction",  "Guess  for
r"]
            dt = pd.DataFrame(rows, columns=headers)
            frac=
Fraction(phase).limit_denominator(15)
            s, r = frac.numerator, frac.denominator
            p= gcd(a**(r//2)-1, 15)
            q= gcd(a**(r//2)+1, 15)
```

```
        if  (p!=1) and (q!=1) :
            break
    print(df)
    print(dt)
    print("r=",r)
    print("p=",gcd(a**(r//2)-1,       15)”,;"“,q=",
gcd(a**(r//2)+1, 15))
    Imports Successful
    sai
                Register Output            Phase
    0  10000000(bin) = 128(dec)  128/256 = 0.50
    1  11000000(bin) = 192(dec)  192/256 = 0.75
    2  00000000(bin) =   0(dec)    0/256 = 0.00
    3  01000000(bin) =  64(dec)   64/256 = 0.25
       Phase Fraction  Guess for r
    0   0.50      1/2            2
    1   0.75      3/4            4
    2   0.00      0/1            1
    3   0.25      1/4            4
    r= 4
    p= 3 ; q= 5
```

Hình 5. *Chu kỳ và thừa số nguyên tố đo được*

Kết quả nhóm tác giả cải tiến các vòng lệnh và thu được thể hiện ở Hình 5, máy tính có thể xuất ra giá trị chu kỳ $r = 4$ và thừa số nguyên tố là 3 và 5.

Như vậy, nền tảng Qiskit thuận lợi để nghiên cứu và phát triển các thuật toán lượng tử đối với những nhóm nước chưa có máy tính lượng tử thực. Qua việc phát triển thuật toán lượng tử trên máy tính lượng tử, nhóm tác giả nhận thấy là phức tạp và dễ bị lỗi do ồn ào (nhiễu) lượng tử. Tuy nhiên, thuật toán Shor đã thể hiện khả năng phân tích thừa số là có thể. Do đó, việc cần thiết là tăng số qubit cho máy tính lượng tử cũng như sửa lỗi để máy tính lượng tử hoạt động hiệu quả hơn.

4. Kết luận

Trong bốn nền tảng phần mềm lượng tử phổ biến Forest, ProjectQ, QDK và Qiskit hỗ trợ làm việc với máy tính lượng tử, cung cấp công cụ để tạo và thao tác các thuật toán lượng tử trên thiết bị lượng tử thực và phần mềm giả lập, thì Qiskit có ưu thế hơn đối với người dùng nghiên cứu thuật toán lượng tử. Việc thực thi thuật toán lượng tử Shor và chạy chúng trên máy tính lượng tử IBM thông qua cloud của trình mô phỏng Qiskit cho kết quả phù hợp với tính toán lí thuyết. Nếu một máy tính lượng tử có đủ số qubit hoạt động mà không bị ảnh hưởng bởi các tác nhân gây nhiễu thì thuật toán lượng tử Shor hoàn toàn có thể phá mã khóa công khai RSA. Nếu máy tính lượng tử sử dụng càng nhiều qubit thì khả năng bị nhiễu và lỗi rất cao. Do đó, song song với việc nhóm nghiên cứu phần cứng phát triển số qubit và giảm độ nhiễu cho máy tính lượng tử, có thể kết hợp giữa máy tính cổ điển và máy tính lượng tử: Máy tính cổ điển chỉ xử lí các lệnh cổ điển để giảm thiểu việc sử dụng qubit của máy tính lượng tử; Trong khi đó, máy tính lượng tử chỉ xử lí những lệnh “lượng tử” then chốt của thuật toán; Khi đó, số lượng qubit cần dùng sẽ giảm. Đồng thời, đưa ra các thuật toán để sửa lỗi lượng tử. Đây là ý tưởng mà nhóm tác giả sẽ định hướng trong các nghiên cứu tiếp theo.

Lời cảm ơn: Nghiên cứu này được tài trợ bởi Quỹ phát triển Khoa học Trường Đại học Sư phạm – Đại học Đà Nẵng trong đề tài có mã số T2022-TN-09.

TÀI LIỆU THAM KHẢO

[1] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". *Communications of the ACM,* vol. 21, no. 2, pp. 120–126, 1978.

[2] J. Chia, J. J. Chin, and S. C. Yip, “Digital signature schemes with strong existential unforgeability” [version 1; peer review: 1 approved]. *F1000Research,* 2021, pp: 10:931.

[3] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring”, *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124-134, doi: 10.1109/SFCS.1994.365700.

[4] B. Bishnoi, "Quantum Computation". *arXiv preprint* arXiv:2006.02799, 2020.

[5] A. Abbas et al., “Learn Quantum Computation Using Qiskit”, *Community.qiskit.org/textbook*, 2020, [Online]. Availabe: https://lab.quantum-computing.ibm.com, [Accessed: December 10, 2023].

[6] R. LaRose, “Overview and Comparison of Gate Level Quantum Software Platforms”, *Quantum,* vol. 3, pp. 130, 2019.

[7] D. V. Lu and L. L. Hang. “Implementating Some Quantum Basic Algorithms”. *The University of Danang - Journal of Science and Technology*, vol 20, no. 7, pp. 111-6, 2022.