Quantum Communication and Quantum Key Distribution

Implementation of Shor's Algorithm in IBM Qiskit

Presented by: Mashal Zainab.

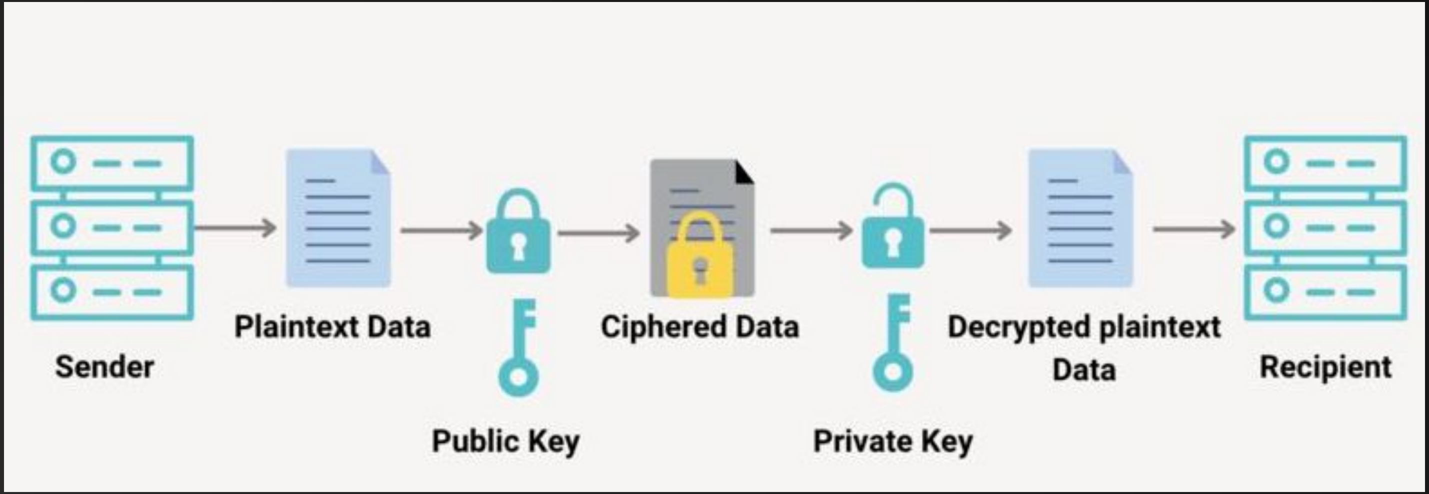
Presented on: 20th Nov, 2023.

uni.lu

UNIVERSITÉ DU LUXEMBOURG

Contents of this presentation

RSA and its Applications

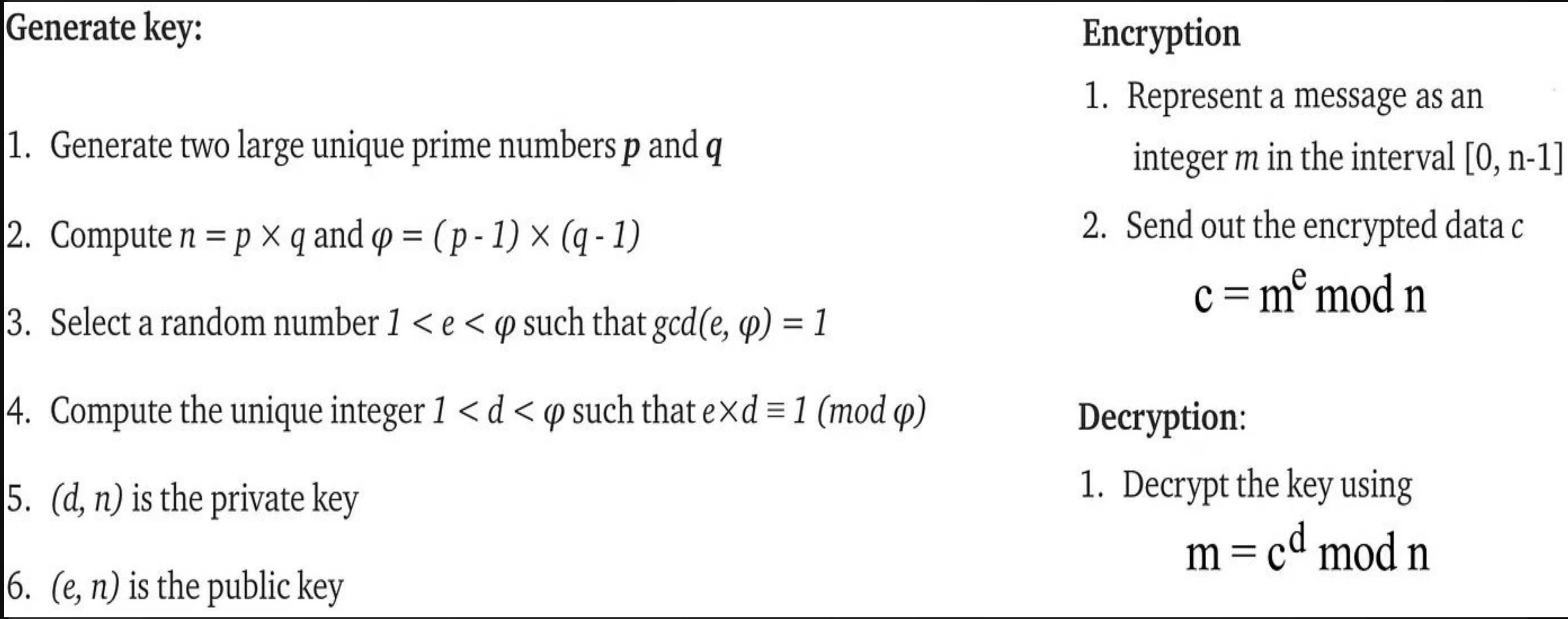


5571

https://
SSL
TLS

DIGITAL
SIGNATURE

The principle of RSA



Generate key:

1. Generate two large unique prime numbers ***p*** and ***q***
2. Compute $n = p \times q$ and $\varphi = (p - 1) \times (q - 1)$
3. Select a random number $1 < e < \varphi$ such that $gcd(e, \varphi) = 1$
4. Compute the unique integer $1 < d < \varphi$ such that $e \times d \equiv 1 \ (mod\ \varphi)$
5. (d, n) is the private key
6. (e, n) is the public key

Encryption

1. Represent a message as an integer *m* in the interval [0, n-1]
2. Send out the encrypted data *c*

$$c = m^e \bmod n$$

Decryption:

1. Decrypt the key using

$$m = c^d \bmod n$$

Prime Factor Complexity

With classical computing, the best we can solve prime factorization is:

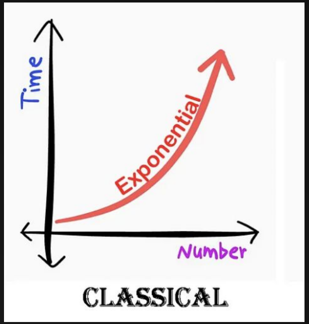
$$O\left(\exp\left(\sqrt[3]{\frac{64}{9}n(\log n)^2}\right)\right)$$

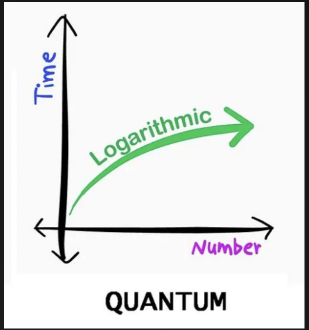
where n is the number of bits to represent the product of the prime numbers. Shor's algorithm can do it in

$$O(n^3 \log n)$$

with the number of gates about

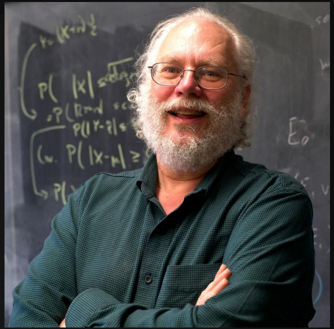
$$O(n^2 \log n \log \log n)$$





A rough time *vs* number graph comparing classical and quantum computation.

Shor’s Algorithm



- Shor’s algorithm is a quantum algorithm which can find the prime factors of a number and can ”undo” this factoring problem much more easily than a classical computer.
- It was developed by mathematician Peter Shor in 1994.
- It is designed to efficiently factor large composite numbers into their prime factors.
- How Shor’s algorithm works is by finding the “period” of a function.
- The period of the function is when you apply some operation and the results go up and back down, like a frequency wave.
- We can find the factors quickly if we have a fast way of finding the period of a known periodic function
- $f(r) = x^r \pmod{N}$

How does Shor’s Algorithm work?

Modular Arithmetic:

$$\mathbf{a = b \bmod N \Leftrightarrow b = a \bmod N}$$

which simply means

$$\mathbf{a = b + \mathit{k}N}$$

where k is an integer.

Consider $\mathbf{x^r = 1 \bmod N}$
where x and N are co-primes, i.e., gcd(x, N) = 1.

We can prove that finding the period r is equivalent to factoring N.

How does Shor’s Algorithm work?

$m_1 * m_2 = N \Leftrightarrow 1 \bmod N$

Consider the equations:

$y^2 = 1 \bmod N$
$y^2 - 1 = 0 \bmod N$
$(y + 1)(y - 1) = 0 \bmod N$
$(y + 1)(y - 1) = 0 \bmod m_1 * m_2$

then we have:

$y + 1 = 0 \bmod m_1$
$y - 1 = 0 \bmod m_2$

or

$y + 1 = 0 \bmod N$
$y - 1 = 0 \bmod 1$

How does Shor's Algorithm work?

We acquire a trivial solution:

$$y + 1 = 0 \mod N$$
$$y - 1 = 0 \mod 1$$

And the desired solution:

$$y + 1 = 0 \mod m_1$$
$$y - 1 = 0 \mod m_2$$

Now, if we can find r and r is even,

$$y^2 = (x^{r/2})^2 = 1 \mod N$$

Then:

$$m_1 = \gcd(x^{r/2} + 1, N)$$
$$m_2 = \gcd(x^{r/2} - 1, N)$$

provided that the solution is non-trivial. And if r is not even, we change x and try again.

Factoring reduces to period finding.

$f(r) = x^r \pmod N$

Why doesn’t Shor’s Algorithm work on Classical Computers?

Shor's algorithm does not work on classical computers because it relies on quantum phenomena that are not available to classical computation.

1. Quantum Parallelism.
2. Quantum Fourier Transform.
3. Quantum Phase Estimation.
4. Quantum Superposition.
5. Quantum Entanglement.

Classical factorization algorithms, like the General Number Field Sieve (GNFS), seek to break down composite numbers into their prime factors using classical computing methods. Despite being more efficient than earlier methods like the Quadratic Sieve and Pollard's rho algorithm, classical factorization algorithms face challenges when dealing with extremely large numbers, especially in comparison to the potential exponential speedup offered by quantum algorithms like Shor's Algorithm.

Implementational Details

Shor's algorithm can be implemented by implementing:

1. **Classical Part:**
   Reducing the factorization problem into a period finding problem with the use of some Arithmetic, Group theory, Euler's Theorem and Bézout's identity.
2. **Quantum Part:**
   The quantum part of the Shor's algorithm is the period finding subroutine.

   Briefly, the quantum subroutine relies on the evaluation of the function *f* at all points *x* simultaneously, and then transform the superposition with an inverse quantum Fourier transform (QFT†) to deduce the period *r* from a single measurement with high probability (>0.4).
   More formally, the routine applies the quantum phase estimation algorithm to find the phase of an eigenvector of the unitary modular multiplication operator *Ua.*

Quantum Fourier Transform

- The quantum Fourier transform (QFT) is the quantum implementation of the discrete Fourier transform over the amplitudes of a wave-function.
- The discrete Fourier transform acts on a vector (x0,x1,...,xN−1) and maps it to the vector (y0,y1,....,yN−1) according to the formula:

$$y_k = \sum_{j=0}^{N-1} e^{\frac{2\pi ikj}{N}} x_j.$$

- The quantum Fourier transform (QFT) transforms between two bases, the computational (Z) basis, and the Fourier basis.
- The H-gate is the single-qubit QFT, and it transforms between the Z-basis (computational) states |0⟩ and 1⟩ to the X-basis states |+⟩ and |−⟩.
  Similar to discrete Fourier transform, the quantum Fourier transform acts on a quantum state and maps it to a quantum state, according to the formula:

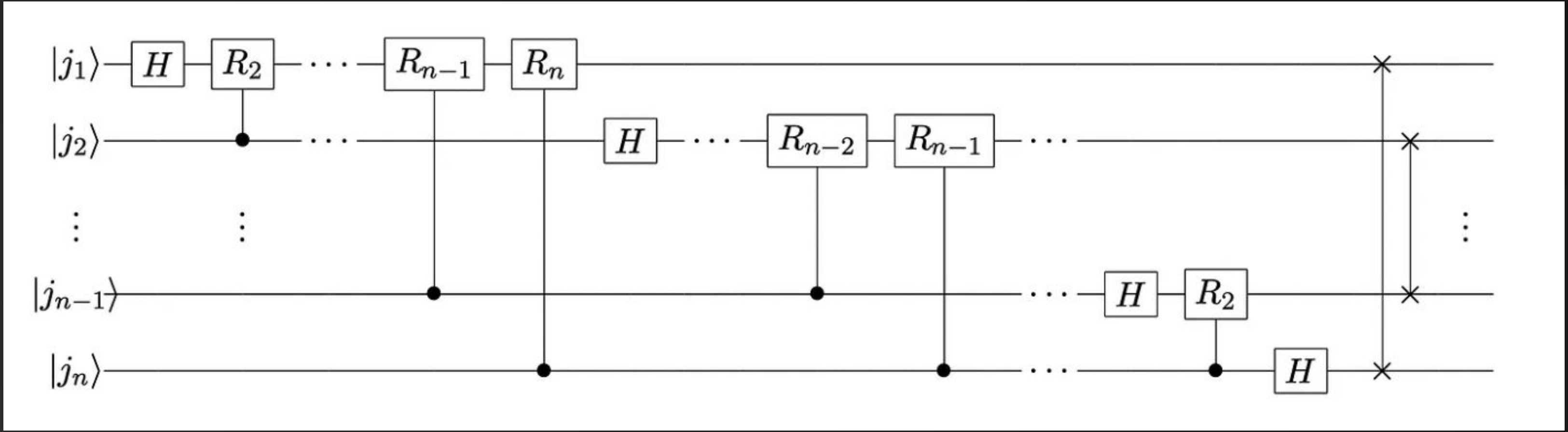
$$\sum_j \alpha_j |j\rangle \to \sum_k \tilde{\alpha}_k |k\rangle, \quad \text{where} \quad \tilde{\alpha}_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi ijk/N} \alpha_j$$

Quantum Fourier Transform contd.

The circuit that implements QFT makes use of two gates.

1. Hadamard gate (which we all are familiar with).
2. The controlled rotation gate which applies a relative phase change to |1>.

$$\hat{R}_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix}$$



Circuit Diagram for n-qubit Quantum Fourier Transform

Quantum Phase Estimation

The eigenvalue and the eigenvector of a matrix **A** satisfy the following relationship.

$$A\mathbf{v} = \lambda\mathbf{v}$$

$$\underbrace{\begin{bmatrix}1 & -3 & 3\\ 3 & -5 & 3\\ 6 & -6 & 4\end{bmatrix}}_{\mathrm{A}}\begin{bmatrix}1/2\\ 1/2\\ 1\end{bmatrix} = \overset{\text{eigenvalue}}{4}\underbrace{\begin{bmatrix}1/2\\ 1/2\\ 1\end{bmatrix}}_{\text{eigenvector}}$$

Phase estimation is about estimating the phase (eigenvalue) of an eigenvector of a unitary operator.
Since a unitary operation preserves the norm of the state vector to 1, the corresponding eigenvalue is just an exponential complex function with real value ϕ below.
ϕ is between 0 and 1.
Phase estimation simply means estimate the value of ϕ in the eigenvalue.

$$U\overset{\text{eigenvector}}{|\psi\rangle} = \underbrace{e^{2\pi i\phi}}_{\text{complex eigenvalue}}|\psi\rangle\,, 0 \le \phi < 1$$

In Shor's algorithm, we want to find the period of some modulo function.
But to do it efficiently, we establish a relationship between the period of a function and the phase value of the eigenvalue.
So solving the phase helps us to find the period.

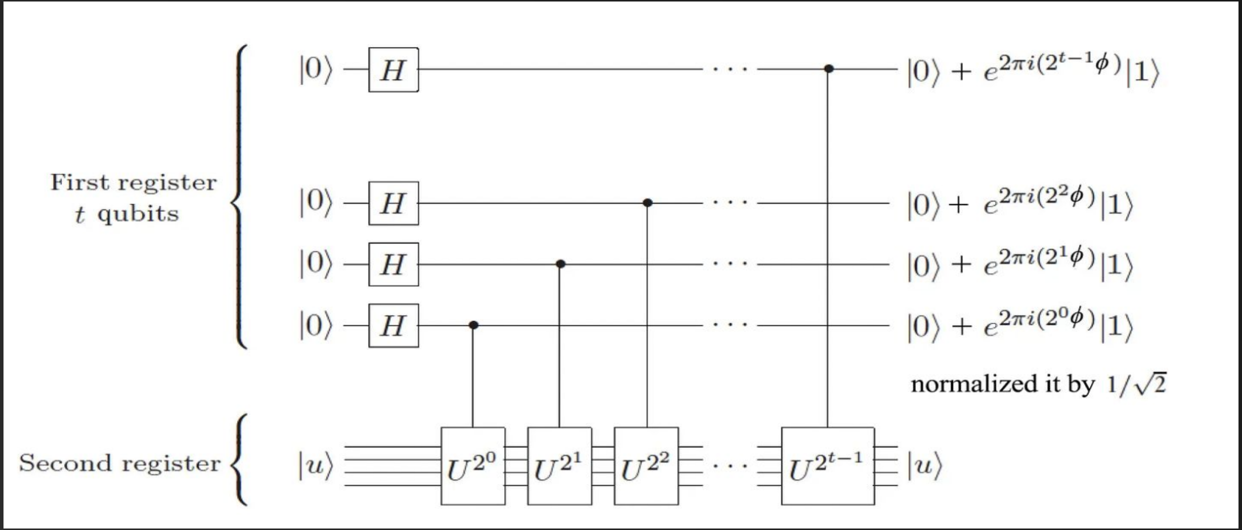
Quantum Phase Estimation contd.

If we know U and |ψ⟩ below, we can estimate ϕ.

This is applied in a circuit like below, consisting of, Hadamard gates and controlled modular multiplication gates.

$$\overset{\text{Known}}{U|\psi\rangle} = e^{\overset{\text{Estimate}}{2\pi i\phi}}|\psi\rangle$$

where U is a unitary operator and $e^{2\pi i\Phi}$ is its eigen value and | ψ> is its eigen vector.



First register t qubits
Second register
$|0\rangle$
H
$|u\rangle$
U^{2^0}
U^{2^1}
U^{2^2}
$U^{2^{t-1}}$
$|0\rangle + e^{2\pi i(2^{t-1}\phi)}|1\rangle$
$|0\rangle + e^{2\pi i(2^2\phi)}|1\rangle$
$|0\rangle + e^{2\pi i(2^1\phi)}|1\rangle$
$|0\rangle + e^{2\pi i(2^0\phi)}|1\rangle$
normalized it by $1/\sqrt{2}$
$|u\rangle$

Circuit Diagram for Quantum Phase Estimation

Quantum Phase Estimation contd.

The circuit on the previous slide just prepare the following supposition:

$$2^{-t/2}\left(|0\rangle + e^{2\pi i 0.\phi_t}|1\rangle\right) \otimes \left(|0\rangle + e^{2\pi i 0.\phi_{t\text{-}1}\phi_t}|1\rangle \otimes \cdots \otimes \left(|0\rangle + e^{2\pi i 0.\phi_1 \ldots \phi_t}|1\rangle$$

or

$$\frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} e^{2\pi i \phi j} |j\rangle$$

So, the prepared superposition is just the inverse QFT of the phase.

$$\frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} e^{2\pi i \phi j} |j\rangle \xrightarrow{\text{QFT}^{-1}} |2^t \phi\rangle$$

E.g. $|10111\rangle$

So by applying the inverse of the QFT on the prepared superposition, the phase value can be found.

Implementation of Shor’s algorithm

The output, for this algorithm is a superposition of all r eigenvectors of Ua, so the phase that was measured is $\phi=s/r$, where s is a random integer between 0 and $r - 1$. Thus, r can be deduced classically from ϕ with a simple continued fraction algorithm, except if $\phi=0$.
And after finding the period, r, we can classically find the factors of N.

Implementation of Shor’s algorithm in Qiskit

Qiskit is an open-source SDK for working with quantum computers at the level of circuits, pulses, and algorithms.
It provides tools for creating and manipulating quantum programs and running them on prototype quantum devices on IBM Quantum Platform or on simulators on a local computer.
Using the built in functionalities of Qiskit, a simplified code to factor 15, was implemented in Python.
With 8 qubits.
And the code was run on the Qiskit simulator backend, specifically **QasmSimulator**.

The process of implementation can be simplified in the following steps:

Step 1: Initializing the qubits
Step 2: Modular exponentiation
Step 3: Applying the Inverse Quantum Fourier Transform
Step 4: Implementing the circuit
Step 5: Running the program on a quantum simulator and measuring the first n qubits
Step 6: Classical Post-processing to obtain factors of the number 15

Step 1: Initializing the qubits

We will create a circuit by using two registers and initializing:

1. n qubits which are the measurement qubits.
   Here, n = 4.
   n qubits will have 2^n number of computational basis states in superposition.
   The measurement qubits will be set to |0> and a hadamard gate is applied on them to create a state of superposition with n qubits.
2. m qubits which are the target qubits.
   And m is also = 4.
   The target qubits will be set to |1> and serve as eigenstate on which a unitary operator U will be applied.
   The *x* gate is applied on the last qubit (n + m - 1) as numbering starts from 0th qubit.

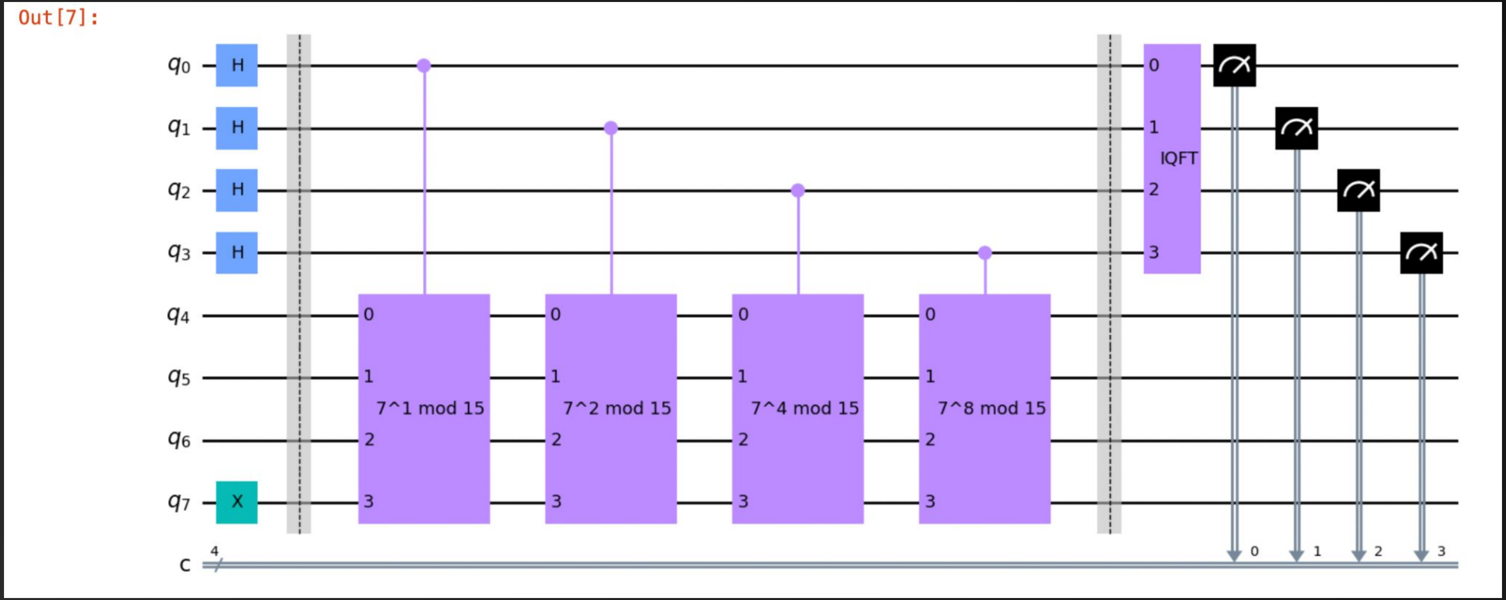
Step 2: Modular Exponentiation

Second, we map the modular exponentiation operations to the measurement qubits. This is done by applying the unitary operator U with various powers onto the target qubits while controlling it with each of the different measurement qubits. The unitary operator in this case implements modular exponentiation.

Step 3: Applying Inverse Fourier Transform

Third, we expose the period by applying an inverse quantum Fourier transform on the n measurement qubits. This works since the quantum phase estimation subroutine encoded the modular exponentiation functions with a fourier basis on the measurement qubits. Applying an inverse QFT just translates that fourier basis into the computational basis thereby revealing periodic elements.

Step 4: Implementing the circuit

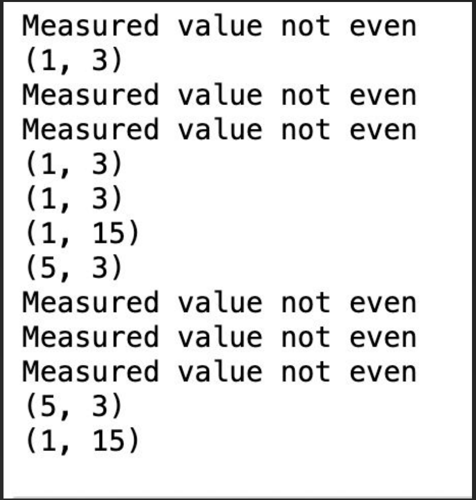


Resulting Circuit with 8 qubits

Step 5 includes running this circuit on a quantum simulator and measuring the qubits.

Step 6: Classical Post-processing

Now that we know r, we have a much higher likelihood of finding the prime factors.
if r isn't even, we have to restart the algorithm and choose a different r value.
The final result of the factorization of 15 that I got can be seen as following:



```
Measured value not even
(1, 3)
Measured value not even
Measured value not even
(1, 3)
(1, 3)
(1, 15)
(5, 3)
Measured value not even
Measured value not even
Measured value not even
(5, 3)
(1, 15)
```

Factoring 15 with Shor's algorithm

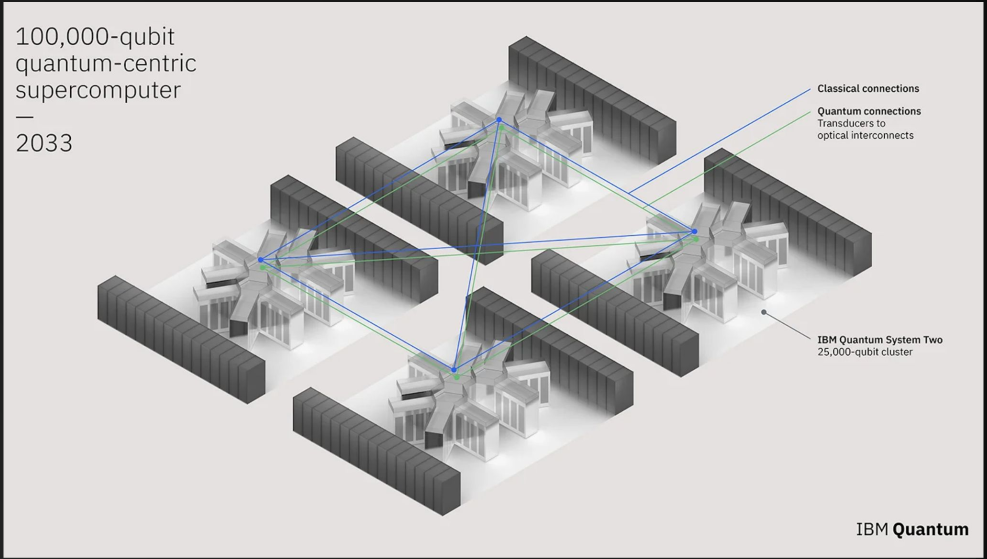
The code is available on my Github:
https://github.com/mashalbhatti/Quantum-Computing-.git

Impact of Quantum Computing on different encryption schemes

Cryptographic Algorithm	Type	Purpose	Impact of QC
AES-256	Symmetric key	Encryption	Secure
SHA-256, SHA-3	Hashing	Hash functions	Secure
RSA	Public key	Signatures, key establishment	No longer secure
ECDSA, ECDH	Public key	Signatures, key exchange	No longer secure
DSA (Finite Field Cryptography	Public key	Signatures, key exchange	No longer secure

But…

- The largest number factored by Shor’s algorithm is still 21.
- It is estimated that 2048-bit RSA could be broken using a quantum computer comprising **4,098 qubits and 100 million gates**.
- Late last year, IBM took the record for the largest quantum computing system with a processor that contained 433 qubits.
- Experts speculate that quantum computers of this size may be available within the next 20-30 years.
- Now, IBM has set its sights on a much bigger target: a 100,000-qubit machine that it aims to build within 10 years, by 2033.



A concept rendering of IBM Quantum’s 100,000-qubit quantum-centric supercomputer, expected to be deployed by 2033.
Source: IBM Research Blog. (2021). *Charting the course to 100,000 qubits*. [online] Available at: https://research.ibm.com/blog/100k-qubit-supercomputer.

References

➔ Shor, P.W. (1997). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, [online] 26(5), pp.1484–1509. doi:https://doi.org/10.1137/s0097539795293172.

➔ Knill, E., Laflamme, R., Barnum, H., Dalvit, D., Dziarmaga, J., Gubernatis, J., Gurvits, L., Ortiz, G., Viola, L. and Zurek, W. (n.d.). From Factoring to Phase Estimation From Factoring to Phase Estimation A discussion of Shor's algorithm. *Los Alamos Science Number*, 27, p.2002.

➔ An overview of Quantum Cryptography and Shor's Algorithm. (2020). *International Journal of Advanced Trends in Computer Science and Engineering*, 9(5), pp.7487–7495. doi:https://doi.org/10.30534/ijatcse/2020/82952020.

➔ Hayward, M. (2005). *Quantum Computing and Shor's Algorithm*.

➔ Aurelien-Pelissier (2022). *Medium/Quantum Programming/src/Shor_factoring at main · Aurelien-Pelissier/Medium*. [online] GitHub. Available at: https://github.com/Aurelien-Pelissier/Medium/tree/main/Quantum%20Programming/src/Shor_factoring [Accessed 18 Nov. 2023].

➔ IBM Research Blog. (2021). *Charting the course to 100,000 qubits*. [online] Available at: https://research.ibm.com/blog/100k-qubit-supercomputer.

➔ MIT Technology Review. (n.d.). *IBM wants to build a 100,000-qubit quantum computer*. [online] Available at: https://www.technologyreview.com/2023/05/25/1073606/ibm-wants-to-build-a-100000-qubit-quantum-computer/#:~:text=Late%20last%20year%2C%20IBM%20took.

➔ Shtetl-Optimized. (2007). *Shor, I'll do it*. [online] Available at: https://scottaaronson.blog/?p=208.

➔ www.qmunity.tech. (n.d.). *Shor's Algorithm* | *Q-munity Tutorials*. [online] Available at: https://www.qmunity.tech/tutorials/shors-algorithm [Accessed 18 Nov. 2023].

References contd.

- ➔ Churchill, S. (2020). *Applying Shor's Algorithm*. [online] Qiskit. Available at: https://medium.com/qiskit/applying-shors-algorithm-bbdfd6f05f7d [Accessed 18 Nov. 2023].
- ➔ gryta (2014). *PPT - Lecture note 8: Quantum Algorithms PowerPoint Presentation, free download - ID:5703191*. [online] SlideServe. Available at: https://www.slideserve.com/gryta/lecture-note-8-quantum-algorithms [Accessed 18 Nov. 2023].
- ➔ Hui, J. (2019a). *QC — Cracking RSA with Shor's Algorithm*. [online] Medium. Available at: https://jonathan-hui.medium.com/qc-cracking-rsa-with-shors-algorithm-bc22cb7b7767 [Accessed 18 Nov. 2023].
- ➔ Hui, J. (2019b). *QC — Period finding in Shor's Algorithm*. [online] Medium. Available at: https://jonathan-hui.medium.com/qc-period-finding-in-shors-algorithm-7eb0c22e8202 [Accessed 18 Nov. 2023].
- ➔ Hui, J. (2019c). *QC — Phase estimation in Shor's Algorithm*. [online] Medium. Available at: https://jonathan-hui.medium.com/qc-phase-estimation-in-shors-algorithm-acef265ebe50 [Accessed 18 Nov. 2023].
- ➔ Hui, J. (2019d). *QC — Quantum Fourier Transform*. [online] Medium. Available at: https://jonathan-hui.medium.com/qc-quantum-fourier-transform-45436f90a43.
- ➔ Katz, N. (2021). *Quantum Factorization*. [online] Medium. Available at: https://towardsdatascience.com/quantum-factorization-b3f44be9d738.
- ➔ Pelissier, A. (2023). *The Ultimate Hack: Crack your Credit Card with Quantum Programming*. [online] MLearning.ai. Available at: https://medium.com/mlearning-ai/the-ultimate-hack-crack-your-credit-card-with-quantum-programming-17ddd045eba6 [Accessed 18 Nov. 2023].
- ➔ Rakhade, K. (2022). *Shor's Algorithm (for Dummies)*. [online] Medium. Available at: https://kaustubhrakhade.medium.com/shors-factoring-algorithm-94a0796a13b1.
- ➔ www.amarchenkova.com. (n.d.). *5 Quantum Algorithms That Could Change The World*. [online] Available at: https://www.amarchenkova.com/posts/5-quantum-algorithms-that-could-change-the-world.

Thank you for your attention!

Any questions would be appreciated.