Shor's Algorithm - Factorisation

Factoring - The Foundation of Modern Cryptography

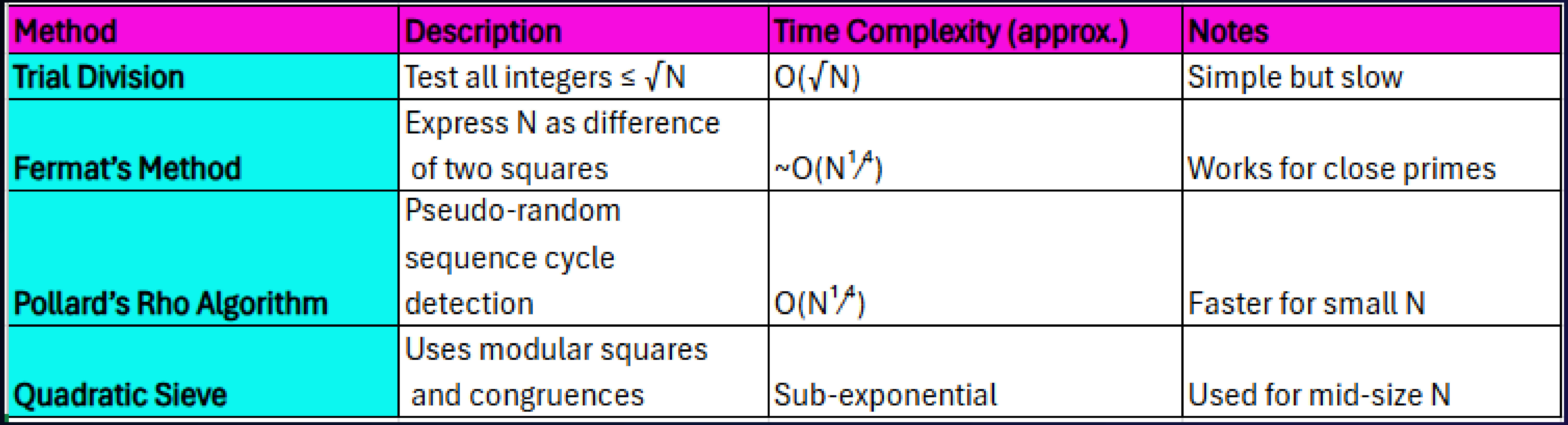
- Factorization = breaking a number into its prime components
- Seems simple for small numbers, but very hard for large numbers
- Modern security (e.g., RSA encryption) depends on this hardness.
- If we could factor large numbers efficiently, much of today’s encryption could be broken.

123018668453011775513049495838496272077
285356959533479219732245215172640050726
365751874520219978646938995647494277406
384592519255732630345373154826850791702
612214291346167042921431160222124047927
4737794080665351419597459856902143413

2000 years on 1 computer

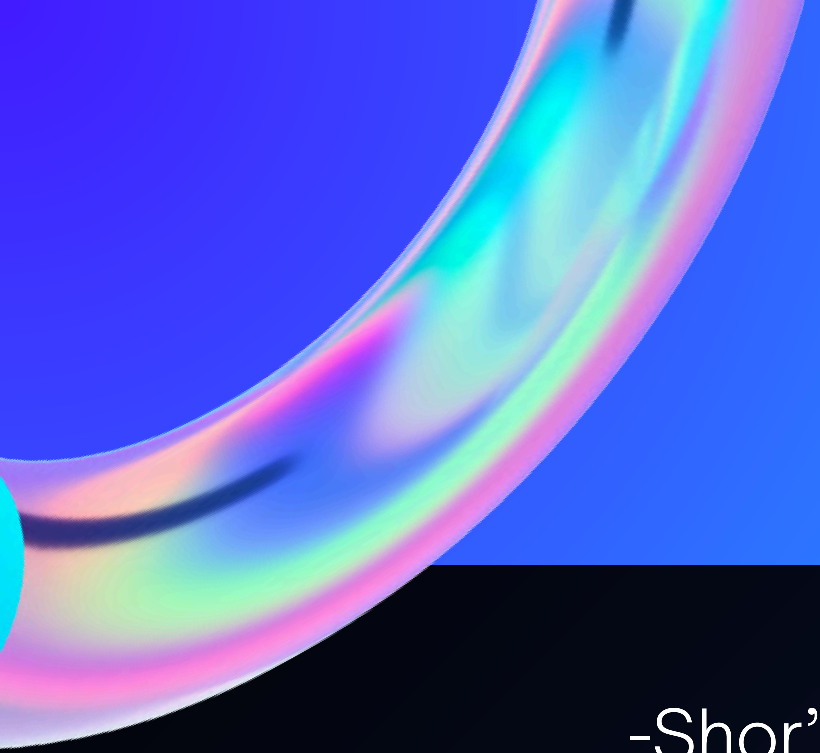
1 month on 24000 computers

Classical Factorization Algorithms



Method	Description	Time Complexity (approx.)	Notes
Trial Division	Test all integers ≤ √N	$O(\sqrt{N})$	Simple but slow
Fermat's Method	Express N as difference of two squares	$\sim O(N^{1/4})$	Works for close primes
Pollard's Rho Algorithm	Pseudo-random sequence cycle detection	$O(N^{1/4})$	Faster for small N
Quadratic Sieve	Uses modular squares and congruences	Sub-exponential	Used for mid-size N

So, factoring numbers used in RSA is practically impossible on classical machines.

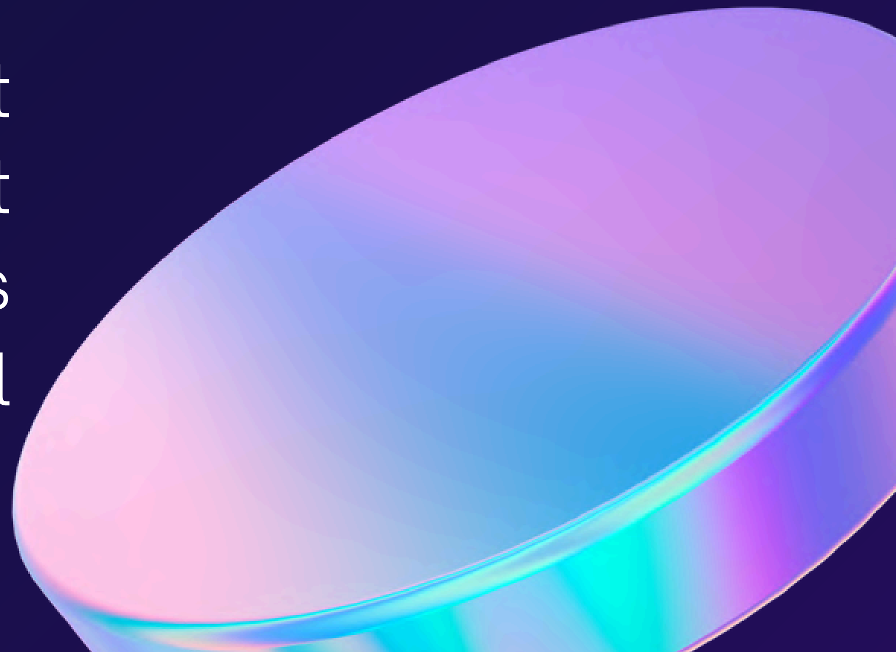


Shor’s Algorithm

-Shor’s Algorithm was proposed by Peter Shor in 1994 and is one of the most significant discoveries in quantum computing. Its primary goal is to factor large integers efficiently using the principles of quantum mechanics, particularly superposition and quantum parallelism.

-Shor’s Algorithm factorizes in polynomial time, with a time complexity of approximately ***O((log N)³)***. This represents an exponential speedup over the best known classical methods.

-Shor’s Algorithm combines quantum computation for the hard part (finding the period) with classical computation for the easy part (calculating the greatest common divisor). This hybrid approach allows it to factor numbers exponentially faster than any known classical algorithm,



Applications

🔐 **Cryptography**

- Breaking RSA:

Shor’s algorithm can factor the large numbers used in RSA public keys.

🧠 **Quantum Benchmarking**

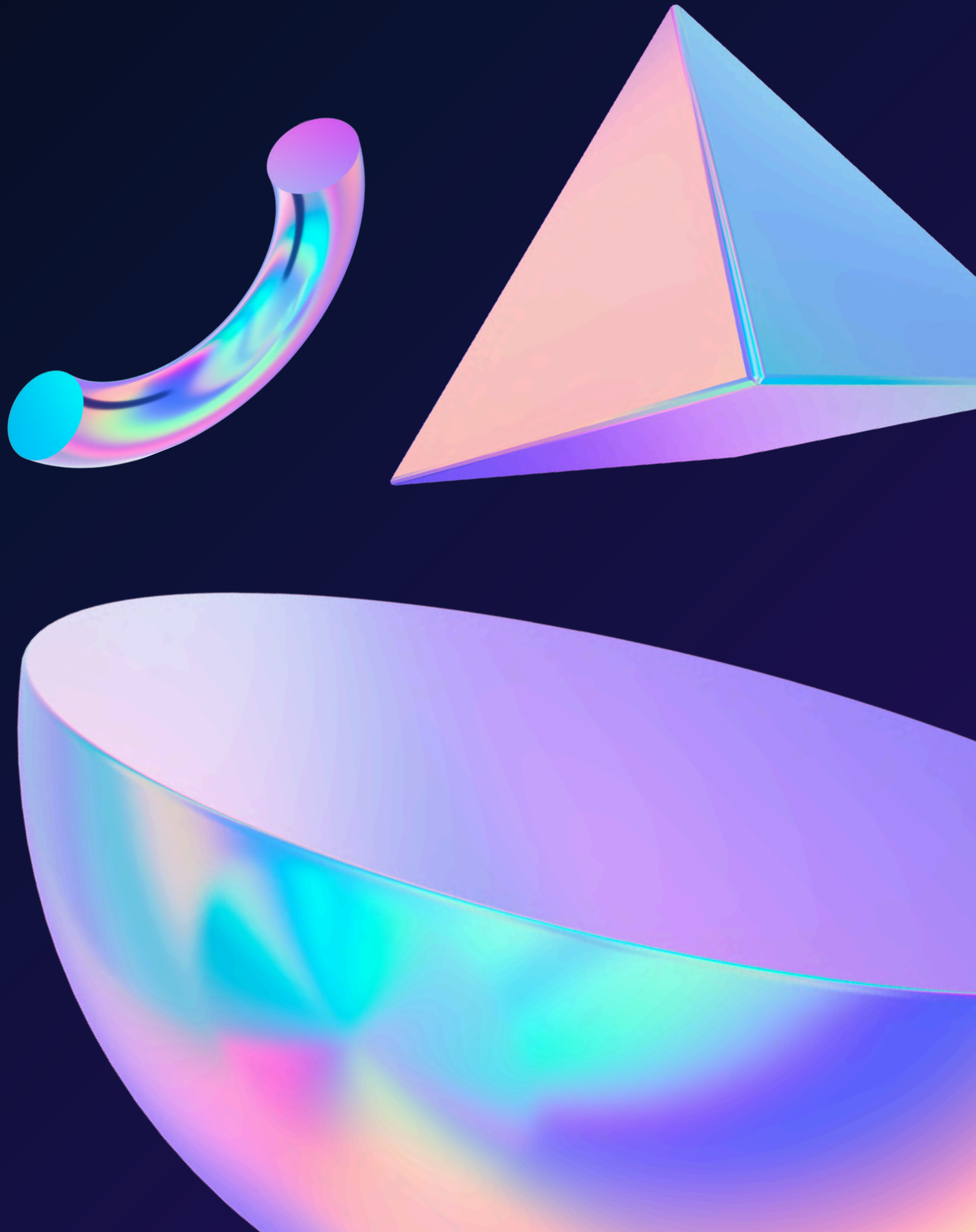
- Serves as a benchmark for quantum hardware and error correction performance.

🔒 **Post-Quantum Cryptography**

- Motivates the development of quantum-resistant algorithms
- Example: NIST’s post-quantum cryptography (PQC) standards.

🧪 **Scientific Interest**

- Demonstrates how quantum Fourier transform and quantum parallelism can outperform classical computation in number theory problems.



HOW RSA ENCRYPTION WORKS IN CREDIT CARD TRANSACTIONS

- RSA Encryption is a public-key cryptography method used to secure data transmission — e.g., when you pay with a credit card online.

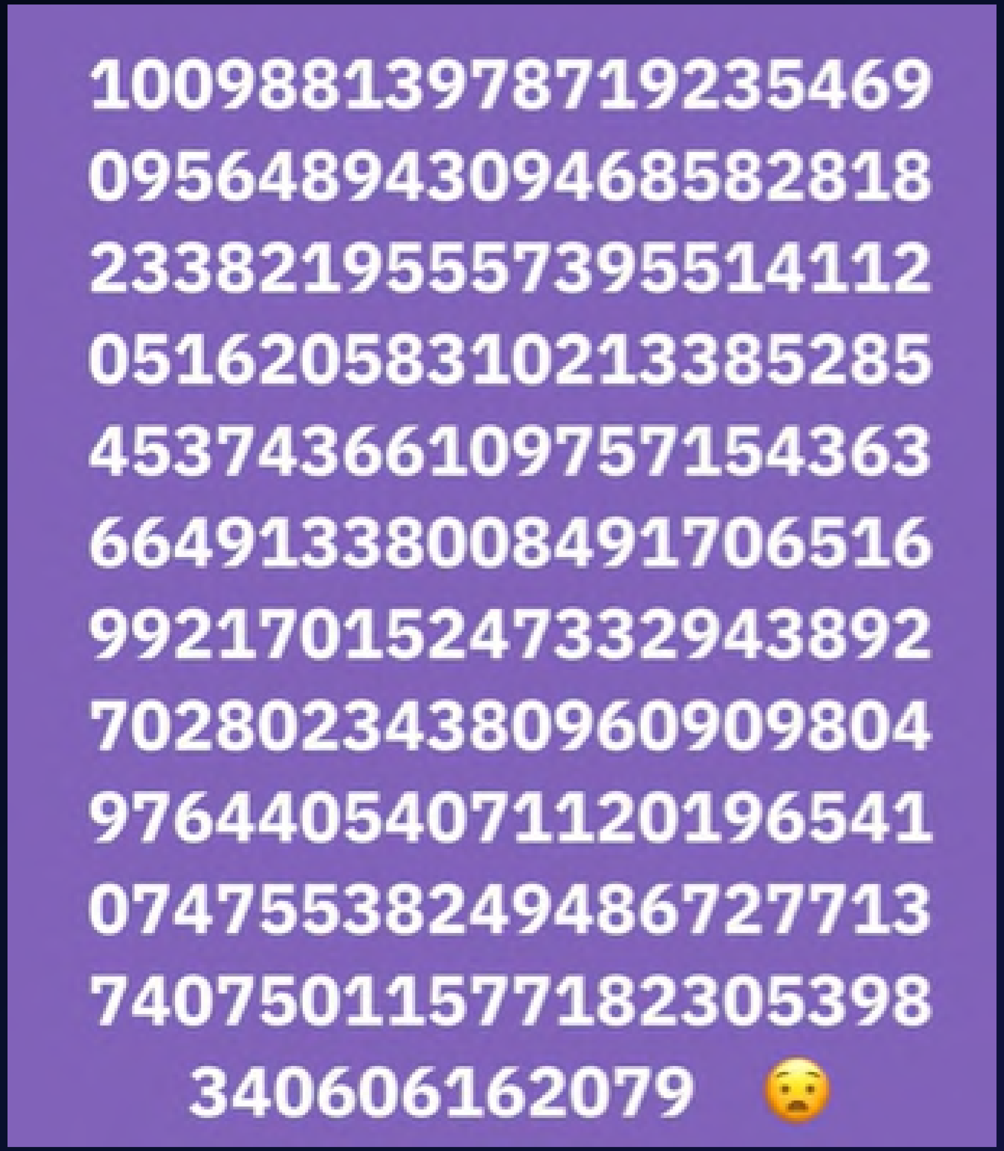
- It works using two keys:
- 🔑 **PUBLIC KEY (ENCRYPTION KEY):**
  - Shared openly.
  - It's typically a large number N = p × q, where p and q are prime numbers.
- 🔒 **PRIVATE KEY (DECRYPTION KEY):**
  - Kept secret by the receiver.
  - Derived from the prime factors (p, q) of N.

Process Flow:

1. The sender encrypts data (credit card details) using the public key (N).
2. The encrypted data can only be decrypted using the private key that contains the factors of N.
3. The security of RSA depends on how difficult it is to find those prime factors.

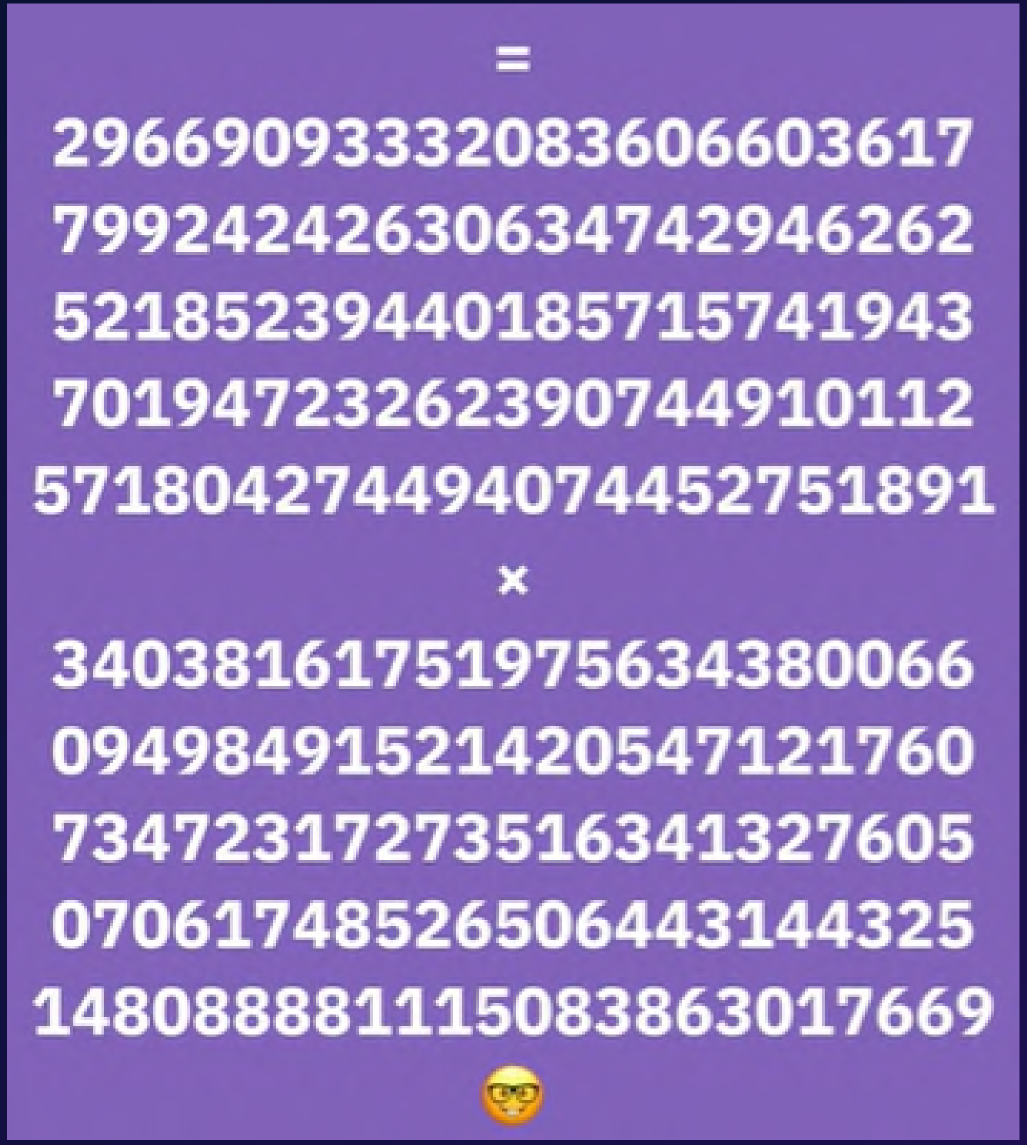
SEMI-PRIMES AND FACTORIZATION

- A semi-prime is a number that is the product of two prime numbers.
- Example: 15 = 3 × 5, or 77 = 7 × 11.
- Factoring semi-primes is extremely hard for classical computers, For example, a 232-digit semi-prime might take 1000+ years to factor on a normal computer!



1009881397871923546909564894309468582818233821955573955141120516205831021338528545374366109757154363664913380084917065169921701524733294389270280234380960909804976440540711201965410747553824948672771374075011577182305398340606162079 😧

=



=

296690933320836066036177999242426306347429462625218523944018571574194370194723262390744910112571804274494074452751891

×

34038161751975634380066094984915214205471217607347231727351634132760507061748526506443144325148088881115083863017669

🤓

THE THREE PARTS OF SHOR'S ALGORITHM

Step 1 – Modular Exponentiation

- Choose a random number a such that $1 < a < N$.
- Compute: $f(x) = a^x \bmod N$
- This function repeats (is periodic) after some value r (the period).
- The period r holds the key to the factors of N.

IN **CIRCUIT:**

- This is represented by a modular exponentiation circuit, built using controlled multiplications.
- Example in code: hardcoded for $a = 7$, $N = 15$.

Step 2 – Quantum Fourier Transform (QFT)

- Apply QFT on the quantum state to extract frequency information (periodicity).
- QFT gives you peaks in measurement outcomes corresponding to r.
- This is where quantum speed-up happens — QFT replaces slow classical searches.

IN CIRCUIT:

- Implemented as a QFT subcircuit.
- Involves Hadamard gates, controlled phase shifts (CU1), and swaps to reorder qubits.

Step 3 – Classical Post-Processing

- Use formulas:

$$p = \gcd(a^{r/2} - 1, N)$$

$$q = \gcd(a^{r/2} + 1, N)$$

- These yield the prime factors of N.

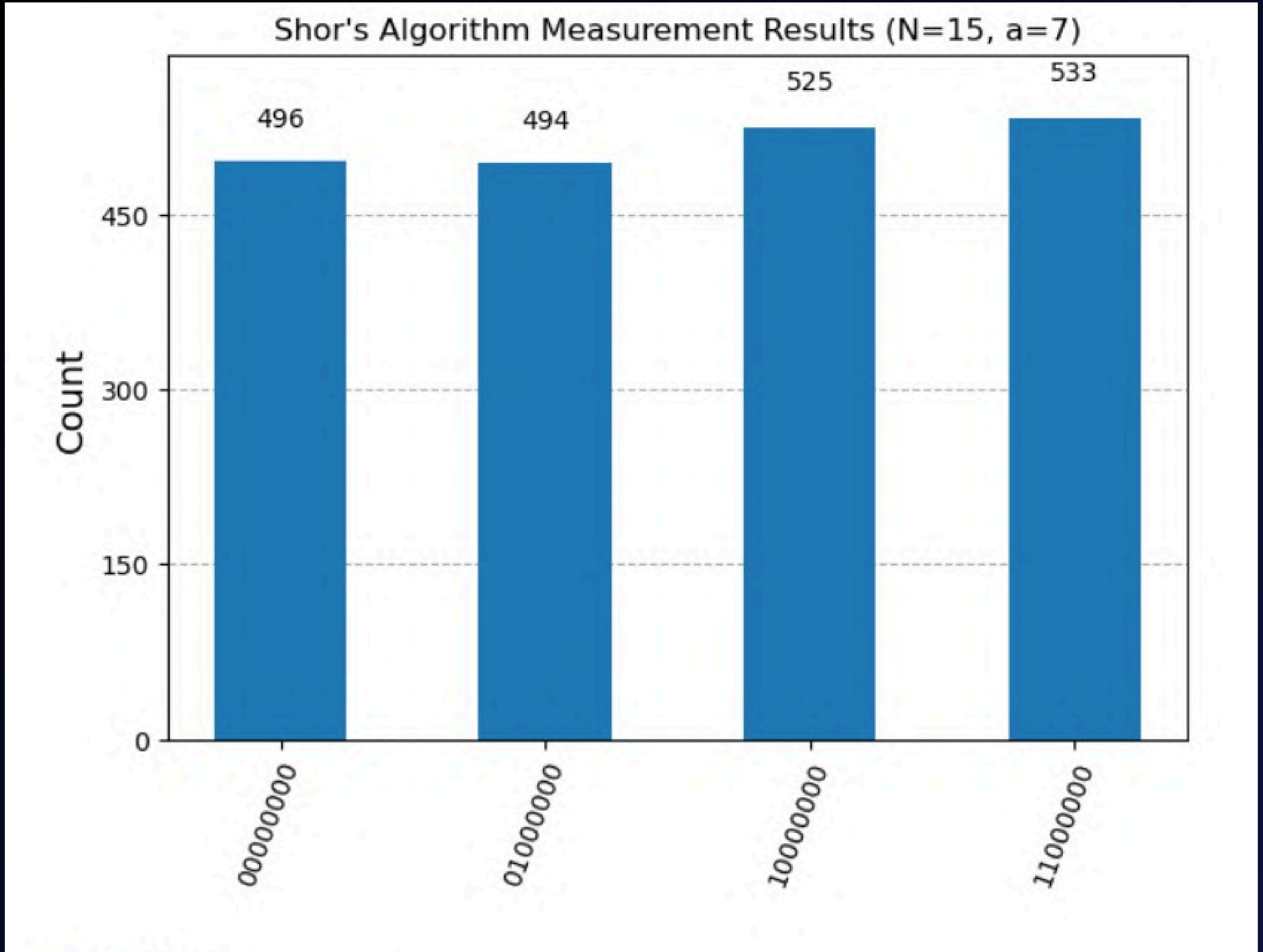
Example:

For $N = 15$, $a = 7$, and $r = 4$:

- $a^{r/2} - 1 = 7^2 - 1 = 48$
- $a^{r/2} + 1 = 7^2 + 1 = 50$

- $\gcd(48, 15) = 3$
- $\gcd(50, 15) = 5$

Factors: 3 and 5



```
Order (r) = 4
Possible factors of 15: 5, 3
```

- **Title:** Shor's Algorithm Measurement Results (N=15, a=7)
- **Y-axis:** Count — how many times each measurement result appeared (from multiple runs or "shots")
- **X-axis:** Binary values — the measurement outcomes of the **counting qubits** after the **Quantum Fourier Transform (QFT)** was applied.

The four outcomes you see are:

Binary Result	Decimal Equivalent	Fraction (out of 256)
00000000	0	0 / 256
01000000	64	64 / 256
10000000	128	128 / 256
11000000	192	192 / 256

These correspond to **four evenly spaced peaks.**

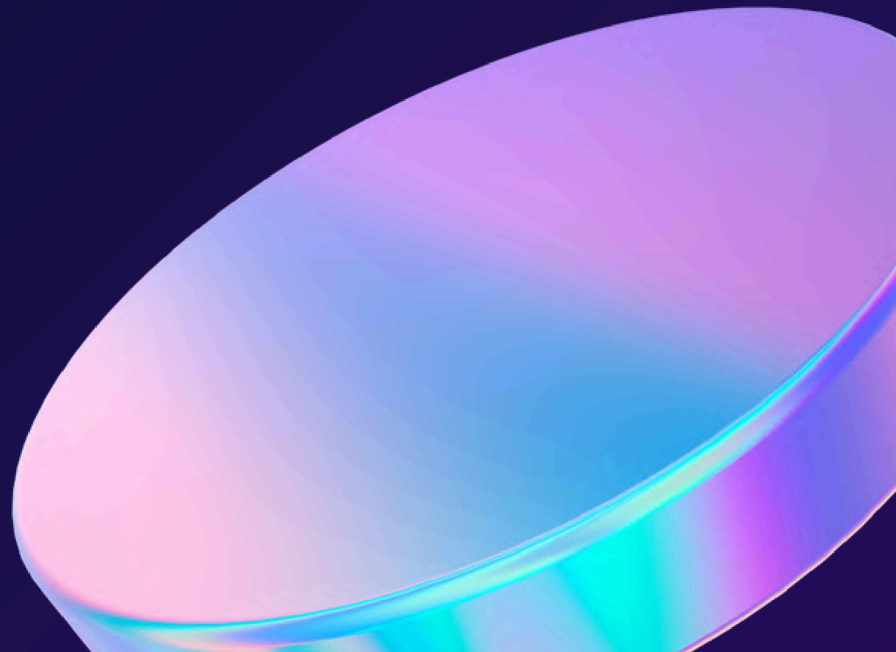
Working of Shor’s Algorithm

Goal:
Find the prime factors of a large integer N efficiently using quantum computation.

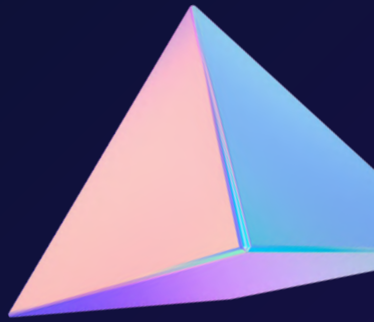
Key Idea:
Factorization is reduced to a period-finding problem, which quantum computers can solve efficiently.

Two Main Parts:

1. Quantum part → finds the period (r) using Quantum Fourier Transform (QFT).
2. Classical part → uses (r) to compute the factors of N using GCD.

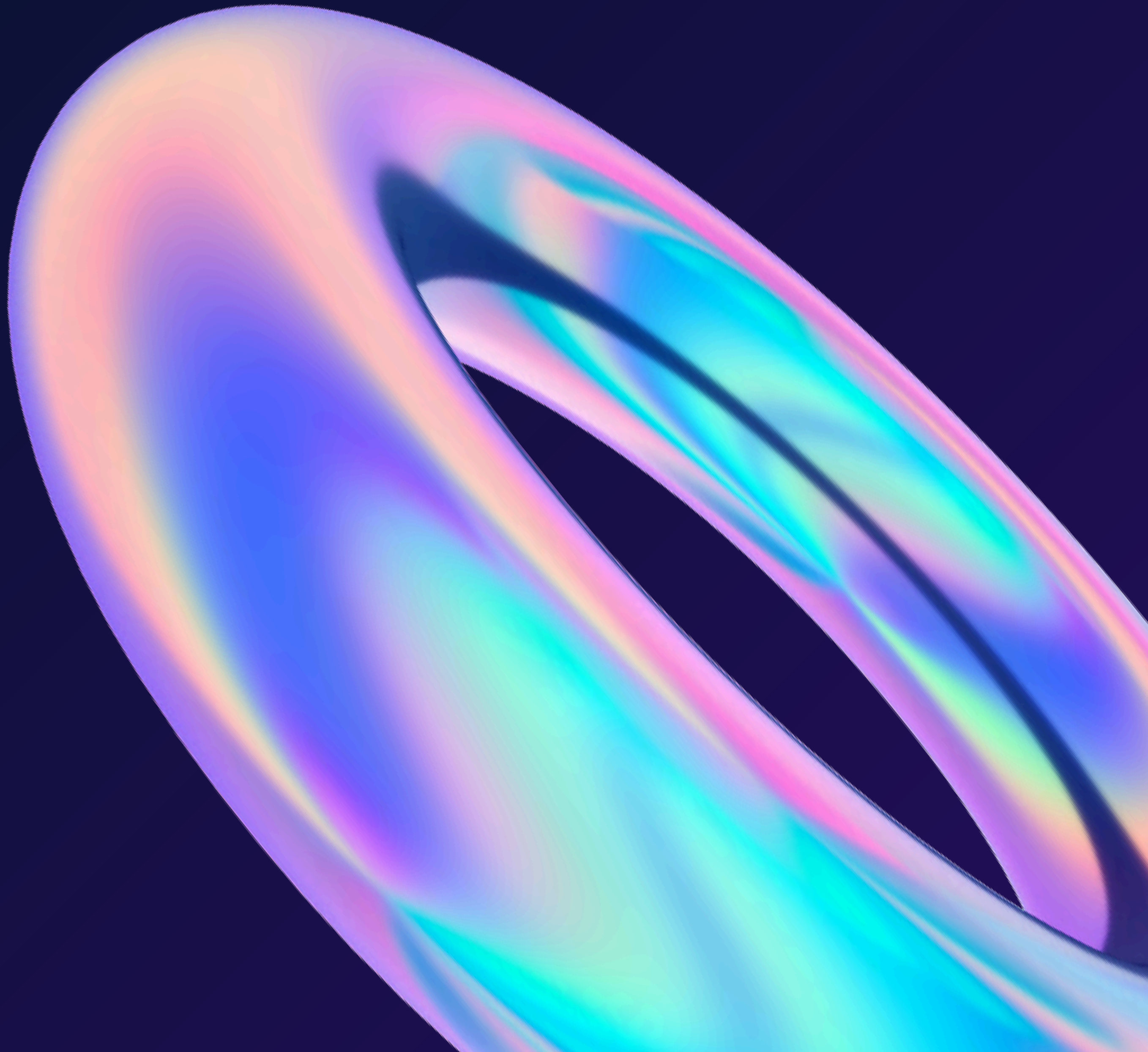


Step-by-Step Process



Goal: find r (the period).

Classically hard, quantumly fast!



Choose random number ‘a’

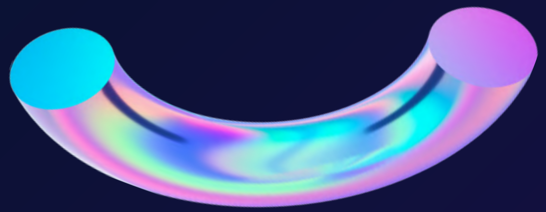
- Pick a such that $1 < a < N$ and $\gcd(a, N) = 1$.
- Example: $N = 15$, $a = 2$.

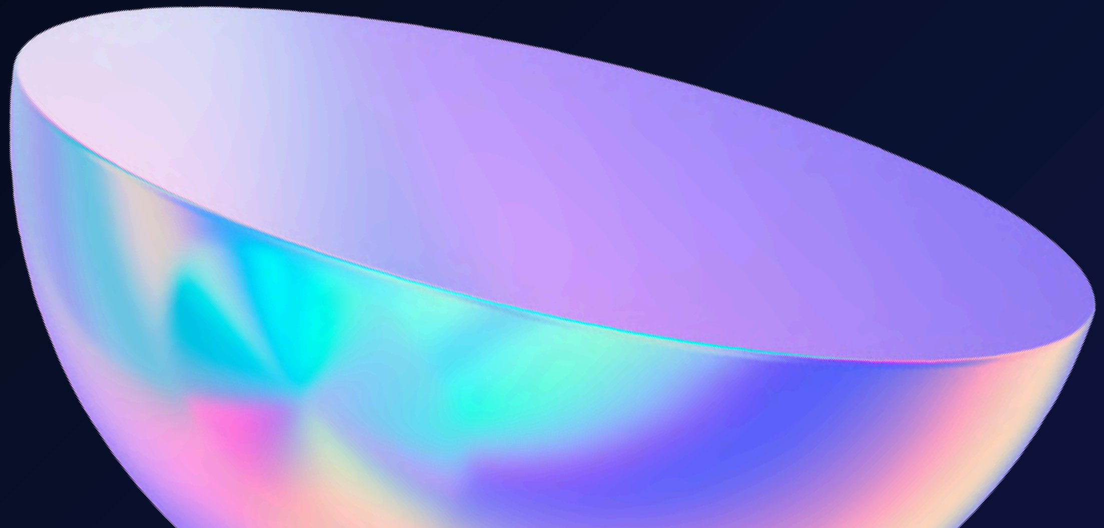
Define periodic function

$f(x)=axmodN$

→ This repeats every r steps (period r).

Example: $2^1=2$, $2^2=4$, $2^3=8$, $2^4=1 \Rightarrow r = 4$





Modular Exponentiation

Repeated Pattern: 1, 2, 4, 8, 7, 5

$\Rightarrow$ The period $r = 6$

$$2^0 \equiv 1\ mod(9)$$
$$2^1 \equiv 2\ mod(9)$$
$$2^2 \equiv 4\ mod(9)$$
$$2^3 \equiv 8\ mod(9)$$
$$2^4 \equiv 7\ mod(9)$$
$$2^5 \equiv 5\ mod(9)$$
$$2^6 \equiv 1\ mod(9)$$
$$2^7 \equiv 2\ mod(9)$$
$$2^8 \equiv 4\ mod(9)$$

$$2^9 \equiv 8\ mod(9)$$
$$2^{10} \equiv 7\ mod(9)$$
$$2^{11} \equiv 5\ mod(9)$$
$$2^{12} \equiv 1\ mod(9)$$
$$2^{13} \equiv 2\ mod(9)$$
$$2^{15} \equiv 4\ mod(9)$$
$$2^{16} \equiv 8\ mod(9)$$
$$2^{17} \equiv 7\ mod(9)$$
$$\vdots$$

$N = pq$, where p and q are prime

For an a, where $1 < a < N$ and $gcd(a, N) = 1$

Let r be the period of modular exponentiation of $a^x \; mod(N)$

With a good approximation of r, the $gcd(a^{\frac{r}{2}} - 1, N)$ and $gcd(a^{\frac{r}{2}} + 1, N)$ has a good chance of containing p and/or q

$$U_{a,N}|x\rangle = |xa\ mod(N)\rangle$$

$$U_{a,N}|x\rangle = |xa\ mod(N)\rangle$$

$$U^0_{a,N}|1\rangle = |1\ mod(N)\rangle$$

$$U^1_{a,N}|1\rangle = |a\ mod(N)\rangle$$

$$U^2_{a,N}|1\rangle = |a^2\ mod(N)\rangle$$

$$U^3_{a,N}|1\rangle = |a^3\ mod(N)\rangle$$

$$U^4_{a,N}|1\rangle = |a^4\ mod(N)\rangle$$

$$\vdots$$

$$U^r_{a,N}|1\rangle = |a^r\ mod(N)\rangle = |1\ mod(N)\rangle$$

For the rest of the lesson we will write $U_{a,N}$ as just U for clarity

$$|u_s\rangle = \frac{1}{\sqrt{r}}\Big(e^{-2\pi i s(0)/r}|a^0\ mod(N)\rangle + e^{-2\pi i s(1)/r}|a^1\ mod(N)\rangle + ...$$
$$+e^{-2\pi i s(r-2)/r}|a^{r-2}\ mod(N)\rangle + e^{-2\pi i s(r-1)/r}|a^{r-1}\ mod(N)\rangle\Big)$$

$$U|u_s\rangle = U\frac{1}{\sqrt{r}}\Big(e^{-2\pi i s(0)/r}|a^0\ mod(N)\rangle + e^{-2\pi i s(1)/r}|a^1\ mod(N)\rangle + ...$$
$$+e^{-2\pi i s(r-2)/r}|a^{r-2}\ mod(N)\rangle + e^{-2\pi i s(r-1)/r}|a^{r-1}\ mod(N)\rangle\Big)$$

$$U|u_s\rangle = \frac{1}{\sqrt{r}}\Big(e^{-2\pi i s(0)/r}|a^1\ mod(N)\rangle + e^{-2\pi i s(1)/r}|a^2\ mod(N)\rangle + ...$$
$$+e^{-2\pi i s(r-2)/r}|a^{r-1}\ mod(N)\rangle + e^{-2\pi i s(r-1)/r}|a^r\ mod(N)\rangle\Big)$$

$$U|u_s\rangle = \frac{1}{\sqrt{r}}\Big(e^{-2\pi i s(0)/r}|a^1\ mod(N)\rangle + e^{-2\pi i s(1)/r}|a^2\ mod(N)\rangle + ...$$

$$+e^{-2\pi i s(r-2)/r}|a^{r-1}\ mod(N)\rangle + e^{-2\pi i s(r-1)/r}|a^r\ mod(N)\rangle\Big)$$

$$a^0\ mod(N) = a^r\ mod(N) = 1\ mod(N)$$

$$U|u_s\rangle = e^{2\pi i s/r}\frac{1}{\sqrt{r}}\Big(e^{-2\pi i s(1)/r}|a^1\ mod(N)\rangle + e^{-2\pi i s(2)/r}|a^2\ mod(N)\rangle + ...$$

$$+e^{-2\pi i s(r-1)/r}|a^{r-1}\ mod(N)\rangle + e^{-2\pi i s(r)/r}|a^0\ mod(N)\rangle\Big)$$

$$U|u_s\rangle = e^{2\pi i s/r}\frac{1}{\sqrt{r}}\Big(e^{-2\pi i s(1)/r}|a^1\ mod(N)\rangle + e^{-2\pi i s(2)/r}|a^2\ mod(N)\rangle + ...$$

$$+e^{-2\pi i s(r-1)/r}|a^{r-1}\ mod(N)\rangle + e^{-2\pi i s(0)/r}|a^0\ mod(N)\rangle\Big)$$

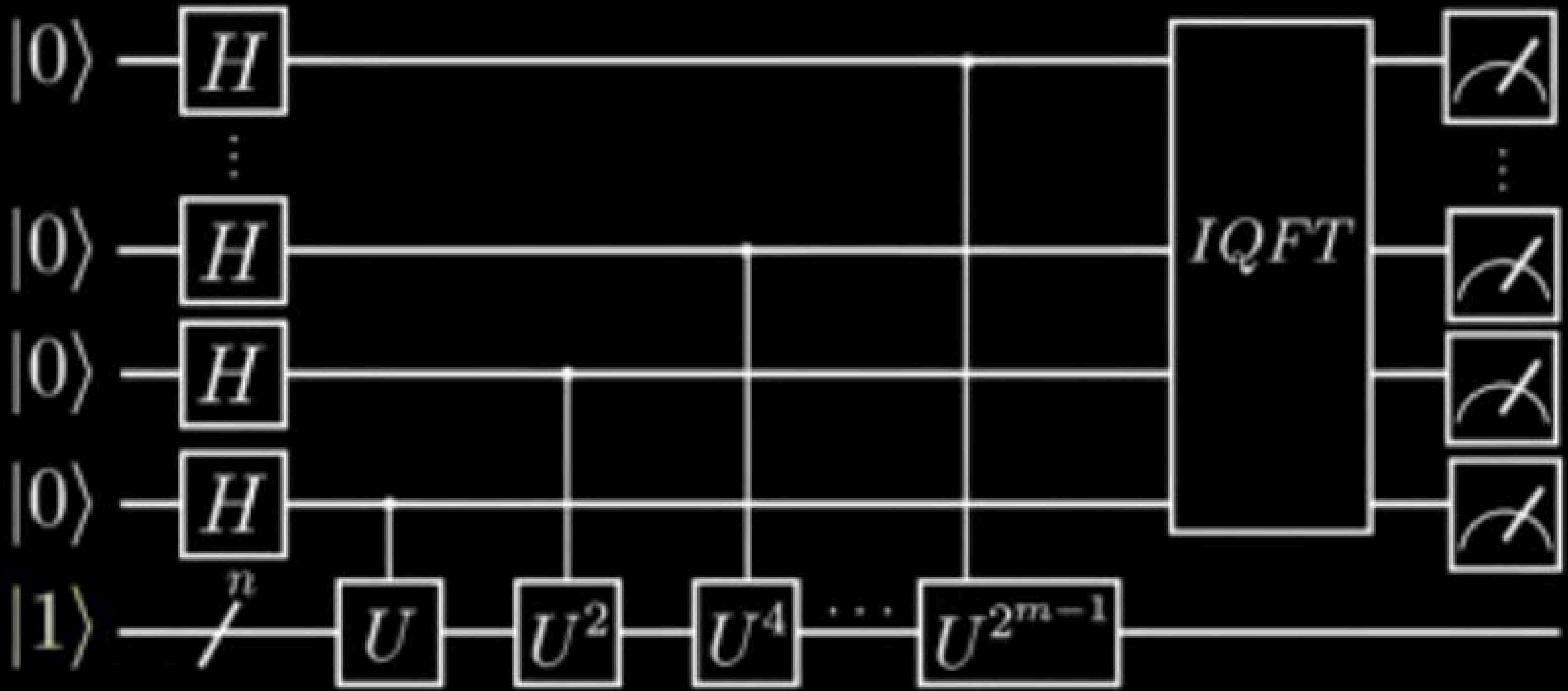
$$e^{-2\pi i s(r)/r} = e^{-2\pi i s(0)/r} = 1$$

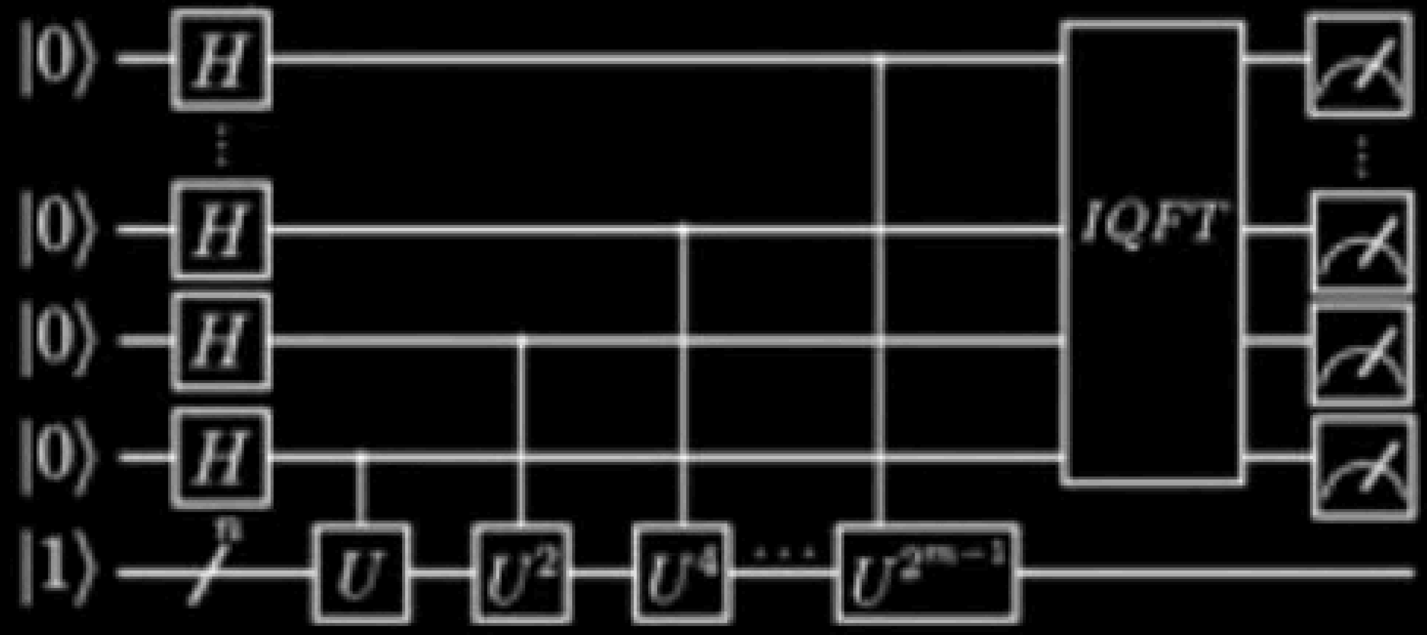
$$U|u_s\rangle = e^{2\pi i s/r}|u_s\rangle$$

$$\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|u_s\rangle$$

$$\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|u_s\rangle = |1\ mod(N)\rangle$$

QUANTUM PHASE ESTIMATION CIRCUIT





This circuit will estimate the eigenvalue $e^{2\pi is/r}$, for one $0 \leq s \leq r-1$

$$\text{Since } |1\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = \frac{1}{\sqrt{r}} \left(|u_0\rangle + |u_1\rangle + ... + |u_{r-1}\rangle \right)$$

So we are calculating the eigenvector of all the $|u_s\rangle$ states for $0 \leq s \leq r-1$.
When we measure we will get the eigenvalue for one $|u_s\rangle$

From the Quantum Circuit we get $j = \frac{s}{r}$

$$0.312 = 0 + \cfrac{1}{3 + \cfrac{1}{4 + \cfrac{1}{1 + \cfrac{1}{7}}}}$$

$$0.312 = 0 + \frac{1}{3 + \frac{1}{4 + \frac{1}{1 + \frac{1}{7}}}}$$

From the Quantum Circuit we get $j = \frac{s}{r}$

We can estimate s/r by rounding the continued fraction off. Here are the first 3 estimations for 0.312:

$$0.312 \approx \frac{1}{3}$$

$$0.312 \approx \frac{1}{3 + \frac{1}{4}} = \frac{4}{13}$$

$$0.312 \approx \frac{1}{3 + \frac{1}{4 + \frac{1}{1}}} = \frac{5}{16}$$

We choose the estimate where the denominator that is less than N, since $r < N$

$$a^r \equiv 1 \; mod(N)$$

$$\implies a^r - 1 \equiv 0 \; mod(N)$$

Therefore $a^r - 1$ has N as a factor

$$a^r - 1 = (a^{r/2} - 1)(a^{r/2} + 1)$$

The $gcd(a^{r/2} - 1, N)$ and/or $gcd(a^{r/2} + 1, N)$ contain a non-trival factor of N, so with our estimate of r we have a good chance of finding this factor. If not we repeat the algorithm

For formal proof of this theorem see the problem set for this lesson for resources

Shor's Algorithm Example: $N = 15$, so $p = 3, q = 5$

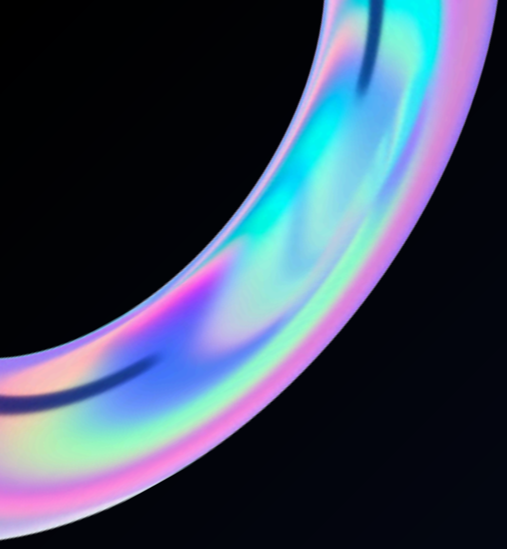
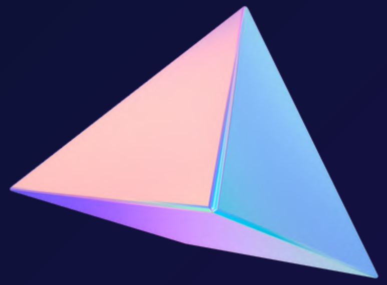
Step 1: Choose and a such that $gcd(a, 15) = 1$, for this example we choose $a = 7$

Step 2: Use a quantum computer to estimate r. We need r such that $7^r \equiv 1 \ mod(15)$. Using the algorithm we find $r = 4$

Step 3: Calculate

$$gcd(7^{4/2} - 1, 15) = gcd(48, 15) = 3$$

$$gcd(7^{4/2} + 1, 15) = gcd(50, 15) = 5$$

Limitations

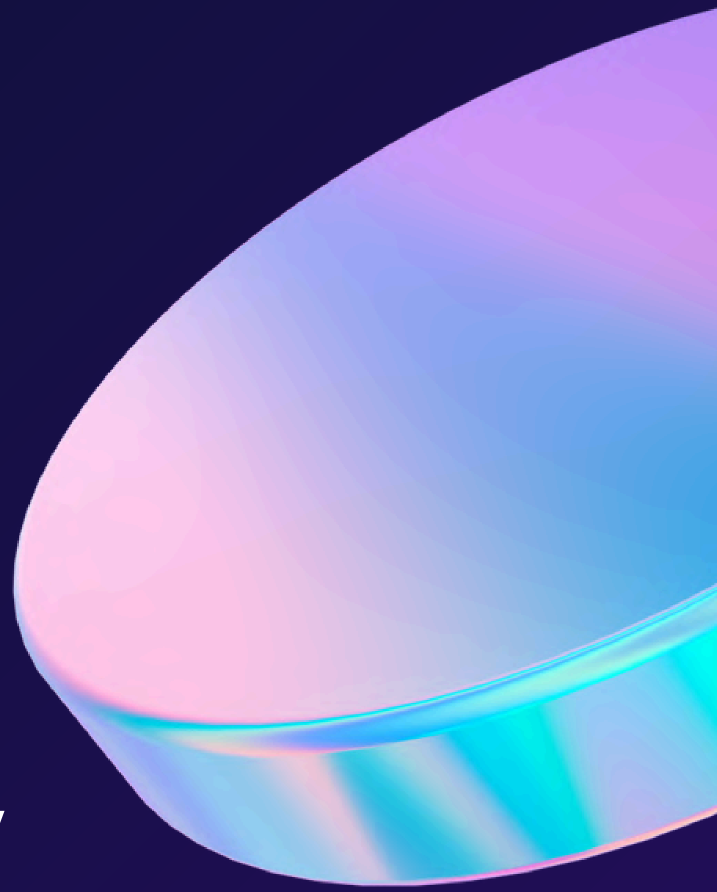
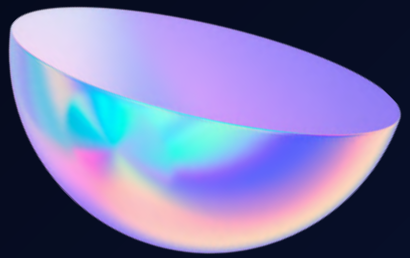
1. Requires a large-scale quantum computer

- Current quantum computers are too small and noisy to factor large numbers like 2048-bit RSA keys.
- For meaningful real-world RSA keys, we would need thousands to millions of high-quality qubits with error correction.

2. Sensitive to errors

- Shor’s algorithm requires a highly precise quantum computation; even small errors in QFT can ruin the period calculation.
- This necessitates quantum error correction, which dramatically increases qubit requirements.

3. Randomness in success

- The algorithm relies on choosing a random coprime r<Nr < Nr<N.
- Some choices of rrr produce odd periods or rp/2≡−1mod Nr^{p/2} ≡ -1 \mod Nrp/2≡−1modN, which give trivial factors.
- You may need to repeat with different r several times before successfully finding factors.