

Carlos Benitez - Luciano Bello

Ekoparty #17  
2 de noviembre de 2021

**The  
silent  
partners**



...o de cómo Shor se asoció a Fourier para factorizar números grandes y romper el cifrado asimétrico en computadoras cuánticas...





Carlos Benitez

Ing. y Mg. de la UTN FRBA

- Investigador en procesamiento de señales acústicas submarinas
- Primer Lab en Seguridad Informática (Si6) y primer SOC Defensa
- Asesor técnico de la Subsecretaría de Ciberdefensa
- Docente de posgrado y consultor en ciberseguridad
- Co-fundador de Platinumciber
- Formador y mentoring de equipos
- SOC, Ethical Hacking y Análisis y Gestión de Riesgos
- Quantum Computing enthusiast



Luciano Bello

Ing. de la UTN FRBA  
PhD de Chalmers (Suecia)

- Desarrollador Senior en IBM Research
- Doctor en seguridad basada en lenguajes
- Python coder
- Desarrollador de Software Libre en Debian
- Diseñador de lenguajes formales
- Infosec enthusiast
- Geocacher

- previously...
- el problema
- RSA
- la factorización
- fourier
- el algoritmo de shor
- la implementación
- resumen

- previously...

- el problema

- RSA

- la factorización

- fourier

- el algoritmo de shor

- la implementación

- resumen





- Introducción a la QC
- Propiedades del qubit
- Hardware existente
- Algoritmo de Deutsch
- Los mitos de la QC
- Las QC online disponibles



- Introducción a la QC
  - Propiedades del qubit
  - Hardware existente
  - Algoritmo de Deutsch
  - Los mitos de la QC
  - Las QC online disponibles
- Algoritmo de Shor

- previously...

- **el problema**

- RSA

- la factorización

- fourier

- el algoritmo de shor

- la implementación

- resumen



4673331833592310999883355855611155212513211028177144957985823385935679234805211772074843110997402088  
4962136809003804931724836744251351914436524922028678749922492363963303861930595117077052285035601177  
9638644050954128274109548519743273551014325753249976993808191641040774990607027085131780854431482719  
2879270515747600591825011224264939011775241470201122113881802463571203852569710311808614896188925840  
6775097681495456790744215925392808604345151310705231857280062253517330504393154504927694689628526886  
9674944342112985792233732337801754241421827174125670264416644353313890442672256181107628062641550510  
9923842039912255378570492258674504781998501869851883957199630080387179659069436984462272457690484426  
2407704045651692639000865172646299059376059542948679165463356213921674455767274649788443435352045655  
6797052450980481438931349795938877105350614496693489409255155953306872814733490045565082856578190868  
9333271410463787949726552668938875959796413163310288065921775297698341521241159133233652681667066444  
7314331097452082351397488563625371930195040660572209571807917346778421232394122570849227616267884850  
4240175619575042958633387070067162448853074807881260125089824549619209919961134580250514604063519606  
6349781811986135030515811634635556335663198966158276043886903297960119840962705373835796308746810568  
4364981806767810342409685957459438712247657182807255734439478038002420178354866749517974669619084362  
2986643959450031252516686566594252074345358101581042723707992427571828820948849065861590065859074391  
3778281730346837019251147896187873091018870042890461298730287464163869574436440285888093124299088608  
8429761386038681691058654221673907407349829719072729465621072367247273840980957893067062615  
3249685719278922751692157130896807074646252738464363045866499709336331569812027362273631245871  
5213560116148604399880851787687637847326255851154572084804325957762864987064580881350389488  
22247351807130888403164645794446319243717913259933477012109944588157536373010228501014181466326553  
71509238946006503860955997146916585180447609432284852910385003949578790594766307709643249139518714  
4359123613861061413590157894888931401420525131224866516461147016667016761431407500872246038892346552  
0552808611097373063518704213113039301625336279405825183612805551854967893065836645527291551181195205  
8888392595313861166137697724678782720586680474567342842176857469092018559835324800004598444782456084  
4859045762973388136611991702058586520990334357374055942865874795790720345913488804917784805628948577  
8804617732179296825817159750372007979156699208305582486657901255718072275107846292479424843965207746  
7237403658550061799279956704411031254567465451105499362798947781210188466981867175415780089815289984  
9247926557842034715220235736186498477432122404953393979535957780605535102962617356735540344891086853  
8926634460813390353581444858227418449682398889148543390840713875511844096578060875650223903048389134  
9117815030583034147983474364473410786162232983781676844315610390028354503758000566280031377558992067  
1170248809970337175034006733019227380745118643000374192911602711339184034355819927937019521413721001  
3559545759948525467412161869068267441364577423716904925542472866535799610399706977624396801087677647  
5830443576635739972027953438424843103674054245431824410173440025375378765370235220916664367509996157  
3987156731808048535495650986676078713033080449444052838485327694559548811642663228650685618461821860  
7927872621210107894980393234159043797743621683940604454280871426803006376885767541206049493503285861  
4372477002463478278523399030686898768798513296184049579718910789231320899515797161738788846785311885



- previously...
- el problema

- **RSA**

- la factorización
- fourier
- el algoritmo de shor
- la implementación
- resumen



# RSA

Se define un número arbitrariamente grande, producto de dos primos:

$$n = p \cdot q$$

Se calcula la función  $\varphi$  de Euler como:

$$\varphi(n) = (p - 1)(q - 1)$$

Se obtienen  $e$  (la clave para cifrar) y  $d$  (la clave para descifrar) y se generan las claves pública y privada:

$$d \cdot e \bmod \varphi(n) = 1$$
$$\text{pubkey} = (n, e) \quad \text{privkey} = (n, d)$$

Teniendo un mensaje  $m$ :

$$\text{message} = m$$

Se genera un mensaje cifrado  $c$  como:

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

Una vez enviado, se recupera el mensaje descifrado  $m$  como:

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



# RSA attack

Dado que en la siguiente ecuación se conocen:

$$d \cdot e \bmod \varphi(n) = 1$$

Entonces, para calcular  $d$  hace falta conocer  $\varphi(n)$ .

Pero  $\varphi(n)$  es:

$$\varphi(n) = (p - 1)(q - 1)$$

Para obtener  $\varphi(n)$  se necesitan  $p$  y  $q$ . Pero como:

$$n = p \cdot q$$

Entonces, factorizando  $n$  se obtienen  $p$  y  $q$  y, en consecuencia  $d$ .



- previously...

- el problema

- RSA

- **la factorización**

- fourier

- el algoritmo de shor

- la implementación

- resumen



## Trial division

```
# Python3 implementation of
# Trial Division Algorithm

# Function to check if a number is
# a prime number or not
def TrialDivision(N):

    # Initializing with the value 2
    # from where the number is checked
    i = 2

    # Computing the square root of
    # the number N
    k = int(N ** 0.5)

    # While loop till the
    # square root of N
    while(i <= k):

        # If any of the numbers between
        # [2, sqrt(N)] is a factor of N
        # Then the number is composite
        if(N % i == 0):
            return 0
        i += 1

    # If none of the numbers is a factor,
    # then it is a prime number
    return 1

# Driver code
if __name__ == "__main__":
    N = 49
    p = TrialDivision(N)

# To check if a number is a prime or not
if(p):
    print("Prime")
else:
    print("Composite")
```



## Fermat

```
# Python 3 implementation of fermat's factorization

from math import ceil, sqrt

#This function finds the value of a and b
#and returns a+b and a-b
def FermatFactors(n):

    # since fermat's factorization applicable
    # for odd positive integers only
    if(n <= 0):
        return [n]

    # check if n is a even number
    if(n & 1) == 0:
        return [n / 2, 2]

    a = ceil(sqrt(n))

    #if n is a perfect root,
    #then both its square roots are its factors
    if(a * a == n):
        return [a, a]

    while(True):
        b1 = a * a - n
        b = int(sqrt(b1))
        if(b * b == b1):
            break
        else:
            a += 1
    return [a-b, a + b]

# Driver Code
print(FermatFactors(6557))
```



## Pollard rho

```
# Python 3 program to find a prime factor of composite using
# Pollard's Rho algorithm
import random
import math

# Function to calculate (base^exponent)%modulus
def modular_pow(base, exponent, modulus):

    # initialize result
    result = 1

    while (exponent > 0):

        # if y is odd, multiply base with result
        if (exponent & 1):
            result = (result * base) % modulus

        # exponent = exponent/2
        exponent = exponent >> 1

        # base = base * base
        base = (base * base) % modulus

    return result

# method to return prime divisor for n
def PollardRho( n):

    # no prime divisor for 1
    if (n == 1):
        return n

    # even number means one of the divisors is 2
    if (n % 2 == 0):
        return 2

    # we will pick from the range [2, N)
    x = (random.randint(0, 2) % (n - 2))
    y = x
```

```
# the constant in f(x).
# Algorithm can be re-run with a different c
# if it throws failure for a composite.
c = (random.randint(0, 1) % (n - 1))

# Initialize candidate divisor (or result)
d = 1

# until the prime factor isn't obtained.
# If n is prime, return n
while (d == 1):

    # Tortoise Move: x(i+1) = f(x(i))
    x = (modular_pow(x, 2, n) + c + n) % n

    # Hare Move: y(i+1) = f(f(y(i)))
    y = (modular_pow(y, 2, n) + c + n) % n
    y = (modular_pow(y, 2, n) + c + n) % n

    # check gcd of |x-y| and n
    d = math.gcd(abs(x - y), n)

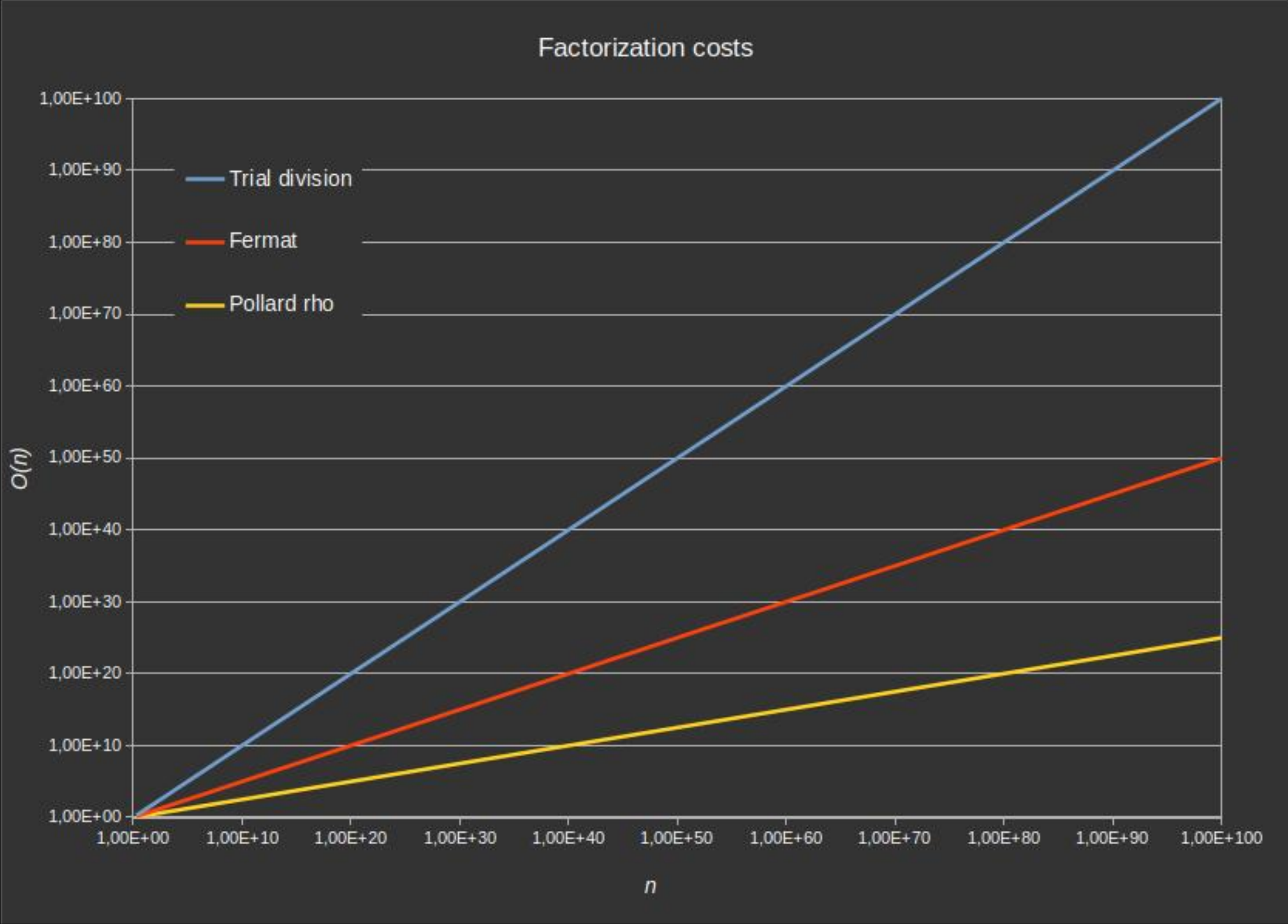
    # retry if the algorithm fails to find prime factor
    # with chosen x and c
    if (d == n):
        return PollardRho(n)

    return d

# Driver function
if __name__ == "__main__":

    n = 10967535067
    print("One of the divisors for", n , "is ",PollardRho(n))

# This code is contributed by chitranayal
```





- previously...
- el problema
- RSA
- la factorización
- **fourier**
- el algoritmo de shor
- la implementación
- resumen

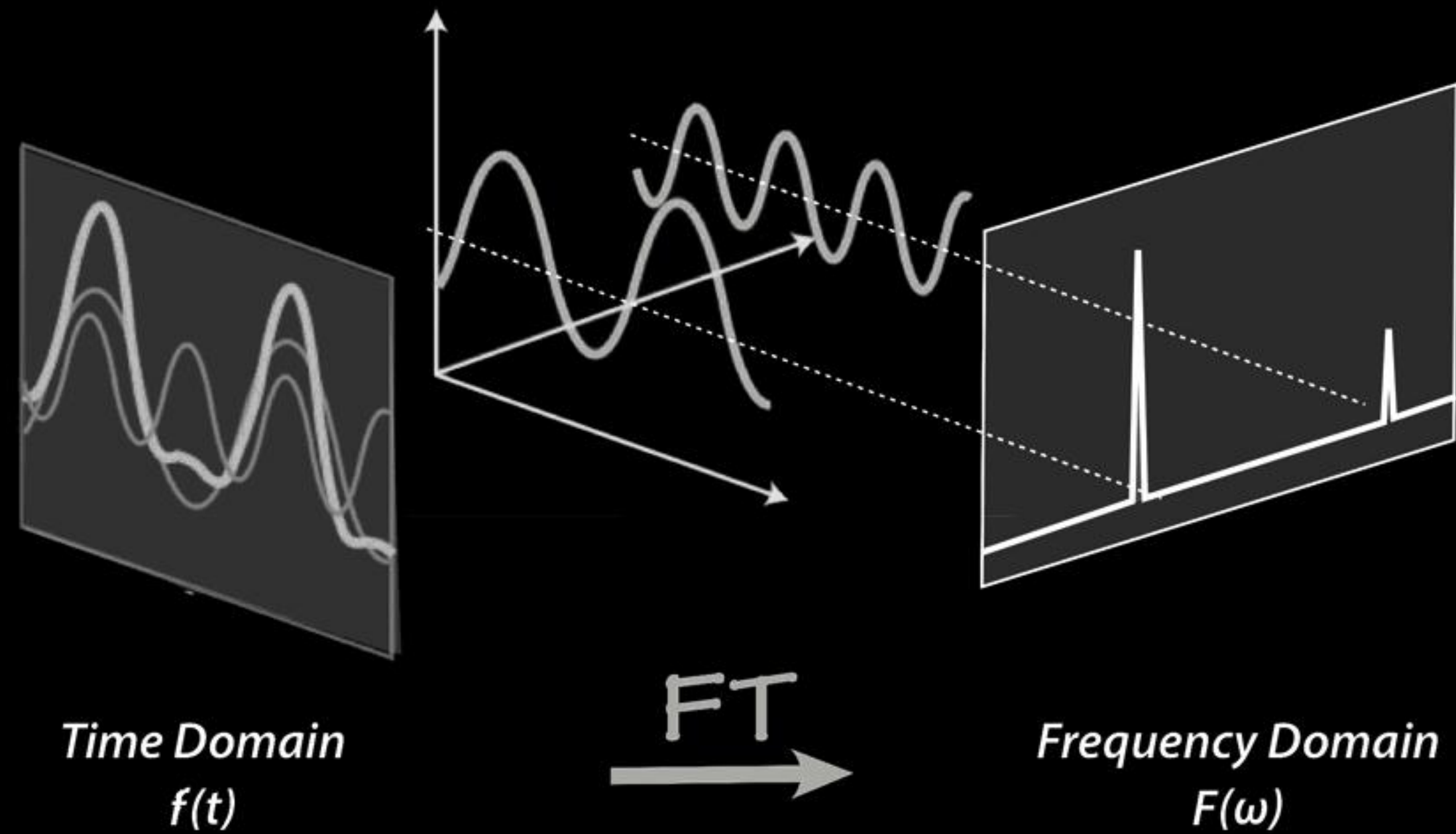


Joseph Fourier 1768-1830



# Transformada de Fourier

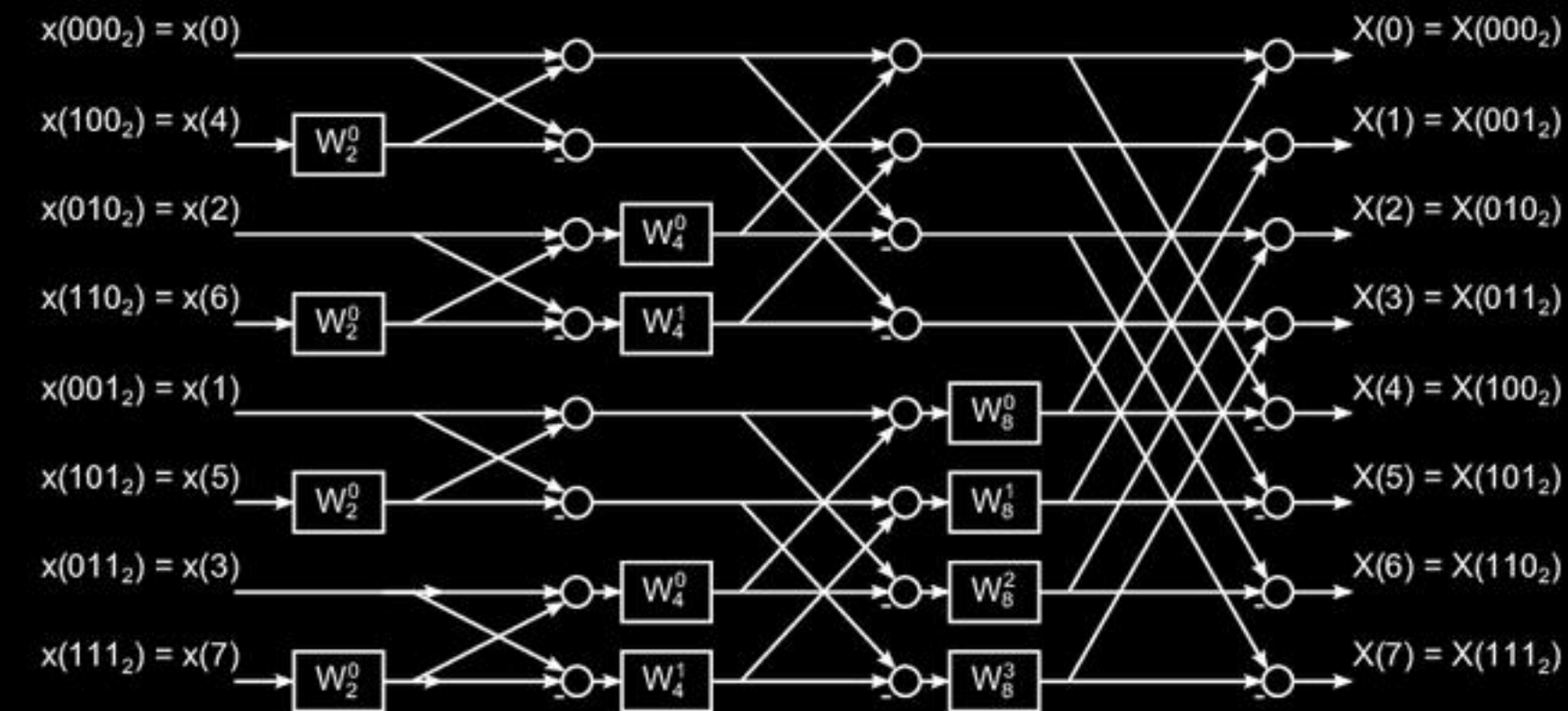
$$\mathcal{F}(\omega) = \int_{-\infty}^{\infty} f(x) \cdot e^{j\omega x} \cdot dx$$



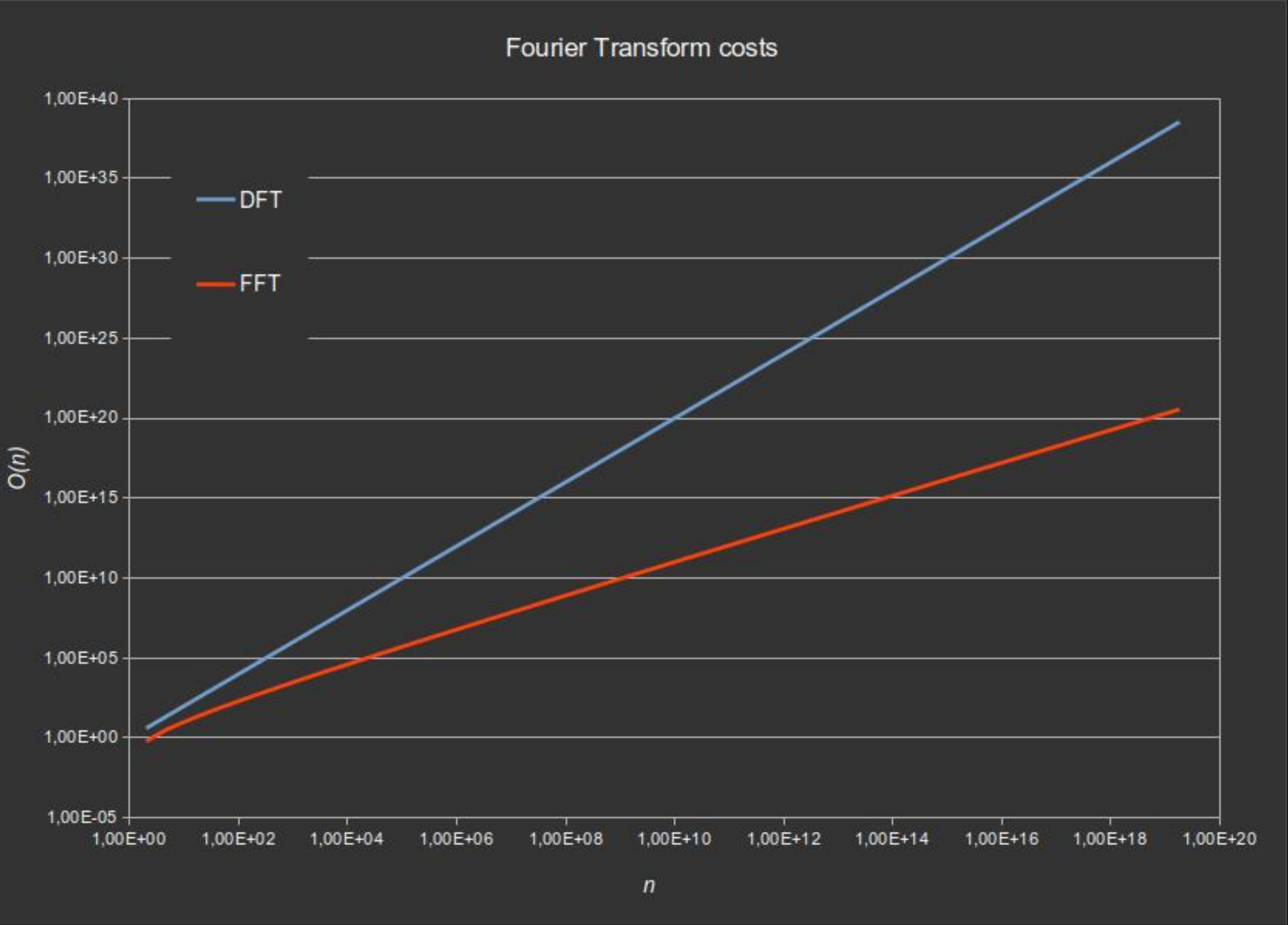
## Transformada Discreta de Fourier

$$\begin{aligned}
 X_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{j2\pi}{N} kn} \\
 &= \sum_{n=0}^{N-1} x_n \cdot \left[ \cos \left( \frac{2\pi}{N} kn \right) - j \cdot \sin \left( \frac{2\pi}{N} kn \right) \right]
 \end{aligned}$$

## Transformada Rápida de Fourier (FFT)









# FFT

SOSUS/Caesar

Garmonya







 *DSP*  
TMS320AV7100PGW  
CH 05A06HW



- previously...
- el problema
- RSA
- la factorización
- fourier
- **el algoritmo de shor**
- la implementación
- resumen



4673331833592310999883355855611155212513211028177144957985823385935679234805211772074843110997402088  
4962136809003804931724836744251351914436524922028678749922492363963303861930595117077052285035601177  
9638644050954128274109548519743273551014325753249976993808191641040774990607027085131780854431482719  
2879270515747600591825011224264939011775241470201122113881802463571203852569710311808614896188925840  
6775097681495456790744215925392808604345151310705231857280062253517330504393154504927694689628526886  
9674944342112985792233732337801754241421827174125670264416644353313890442672256181107628062641550510  
9923842039912255378570492258674504781998501869851883957199630080387179659069436984462272457690484426  
2407704045651692639000865172646299059376059542948679165463356213921674455767274649788443435352045655  
6797052450980481438931349795938877105350614496693489409255155953306872814733490045565082856578190868  
9333271410463787949726552668938875959796413163310288065921775297698341521241159133233652681667066444  
7314331097452082351397488563625371930195040660572209571807917346778421232394122570849227616267884850  
4240175619575042958633387070067162448853074807881260125089824549619209919961134580250514604063519606  
6349781811986135030515811634635556335663198966158276043886903297960119840962705373835796308746810568  
4364981806767810342409685957459438712247657182807255734439478038002420178354866749517974669619084362  
2986643959450031252516686566594252074345358101581042723707992427571828820948849065861590065859074391  
3778281730346837019251147896187873091018870042890461298730287464163869574436440285888093124299088608  
8429761386038681691058654221673907349829719072729465621072367247273840980957893067062615  
32496857192789227516921571308968073466252738464363045866499709336331569812027362273631245871  
5213560116148604399880851787687637817326255851154572084804325957762864987064580881350389488  
22247351807130888403164645794446319243717913259933477012109944588157536373010228501014181466326553  
71509238946006503860955997146916585180447609432284852910385003949578790594766307709643249139518714  
4359123613861061413590157894888931401420525131224866516461147016667016761431407500872246038892346552  
0552808611097373063518704213113039301625336279405825183612805551854967893065836645527291551181195205  
8888392595313861166137697724678782720586680474567342842176857469092018559835324800004598444782456084  
4859045762973388136611991702058586520990334357374055942865874795790720345913488804917784805628948577  
8804617732179296825817159750372007979156699208305582486657901255718072275107846292479424843965207746  
7237403658550061799279956704411031254567465451105499362798947781210188466981867175415780089815289984  
9247926557842034715220235736186498477432122404953393979535957780605535102962617356735540344891086853  
8926634460813390353581444858227418449682398889148543390840713875511844096578060875650223903048389134  
9117815030583034147983474364473410786162232983781676844315610390028354503758000566280031377558992067  
1170248809970337175034006733019227380745118643000374192911602711339184034355819927937019521413721001  
3559545759948525467412161869068267441364577423716904925542472866535799610399706977624396801087677647  
5830443576635739972027953438424843103674054245431824410173440025375378765370235220916664367509996157  
3987156731808048535495650986676078713033080449444052838485327694559548811642663228650685618461821860  
7927872621210107894980393234159043797743621683940604454280871426803006376885767541206049493503285861  
4372477002463478278523399030686898768798513296184049579718910789231320899515797161738788846785311885



## El algoritmo de Shor (I)

$$N = g \cdot h$$

$$\hookrightarrow 2, 3, 5, 7, 11, 13, 17, \dots$$

$$35 = ? \cdot ?$$

$$\hookrightarrow 2 \rightarrow \{ \text{SHOR} \} \rightarrow 5$$

Dados dos coprimos  $g$  y  $N$  existen  $p$  y  $m$  tales que

$$g^p = m \cdot N + 1$$



## El algoritmo de Shor (II)

$$g^p = m \cdot N + 1$$

$$g^p - 1 = m \cdot N$$

$$\left(g^{\frac{p}{2}} - 1\right) \left(g^{\frac{p}{2}} + 1\right) = m \cdot N$$

$$p = ?$$

$$g^p = m \cdot N + 1$$

$$g^p \bmod N = 1$$

### El algoritmo de Shor (III)

$$g^p \bmod N = 1$$

$$g^x \bmod N = r$$

$$3^x \bmod 35 = r$$

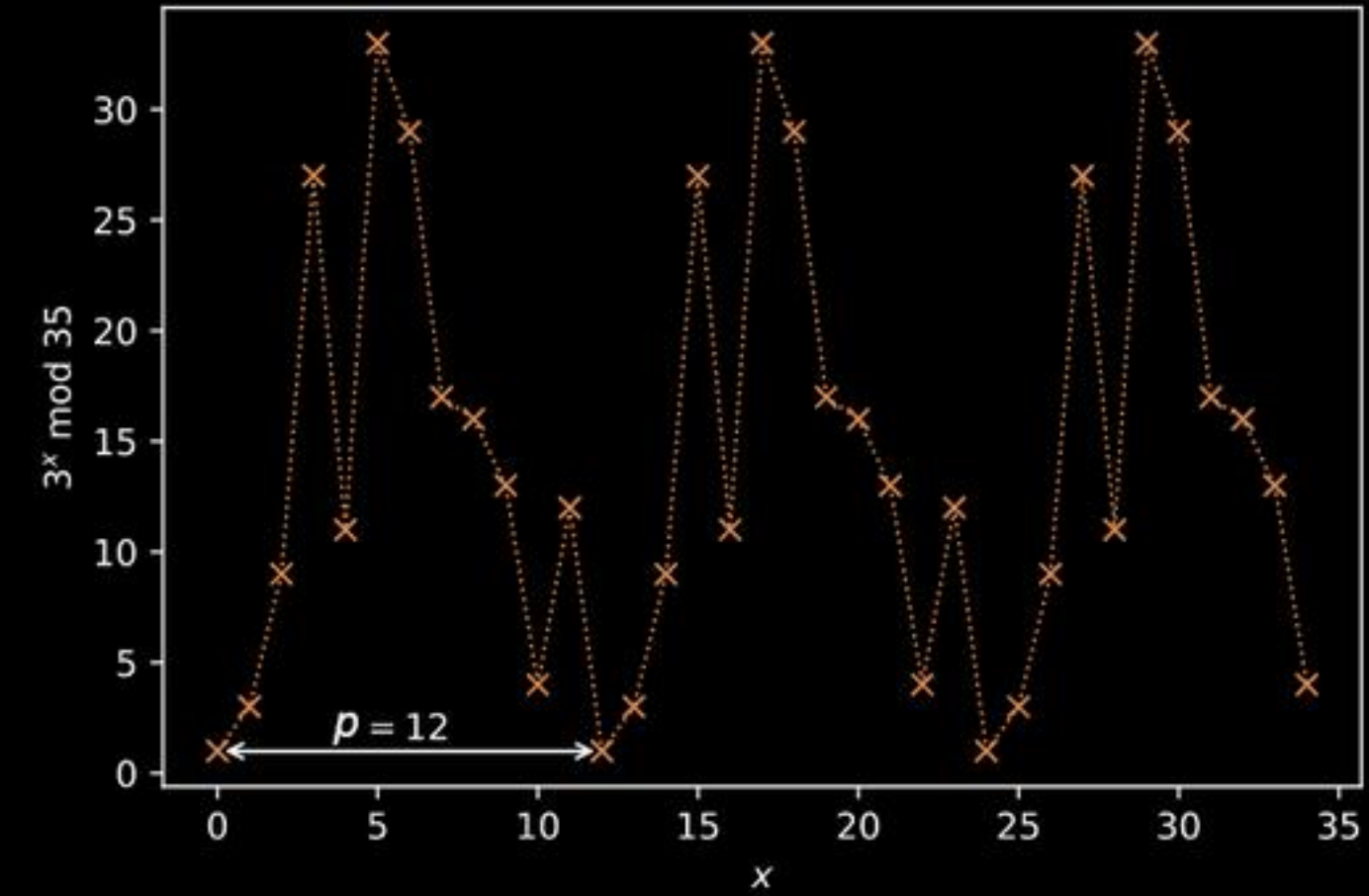
$$x_k = k.p$$

$$g^{x_k} \bmod N = r_x$$

$$g^{kp} \bmod N = r_x$$

$$3^{k \cdot 12} \bmod 35 = r_x$$

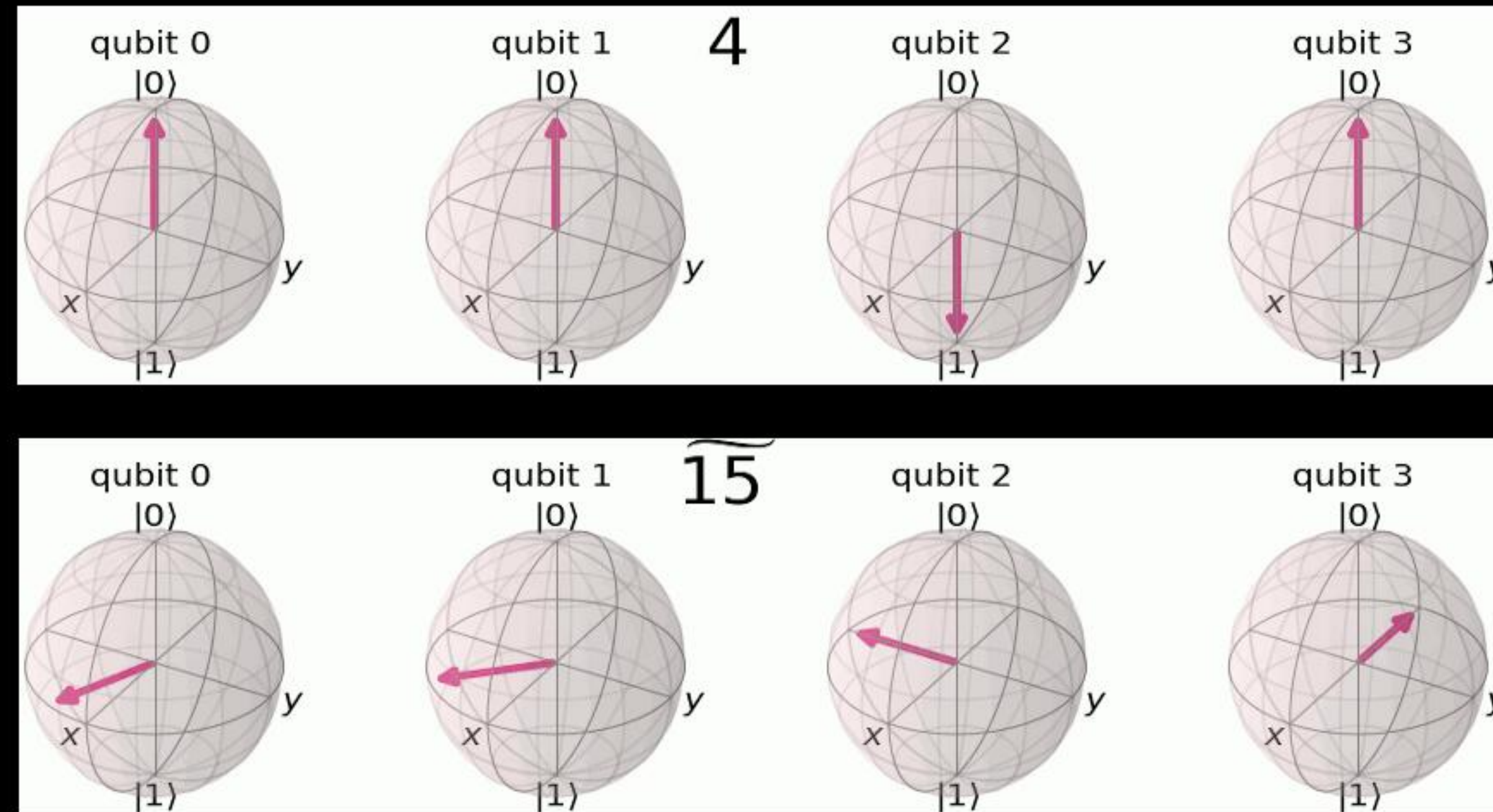
$$3^0 \bmod 35 = 3^{12} \bmod 35 = 3^{24} \bmod 35 = 1$$





# Transformada Cuántica de Fourier (QFT)

$$\text{QFT}_N |x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{xk} |k\rangle$$



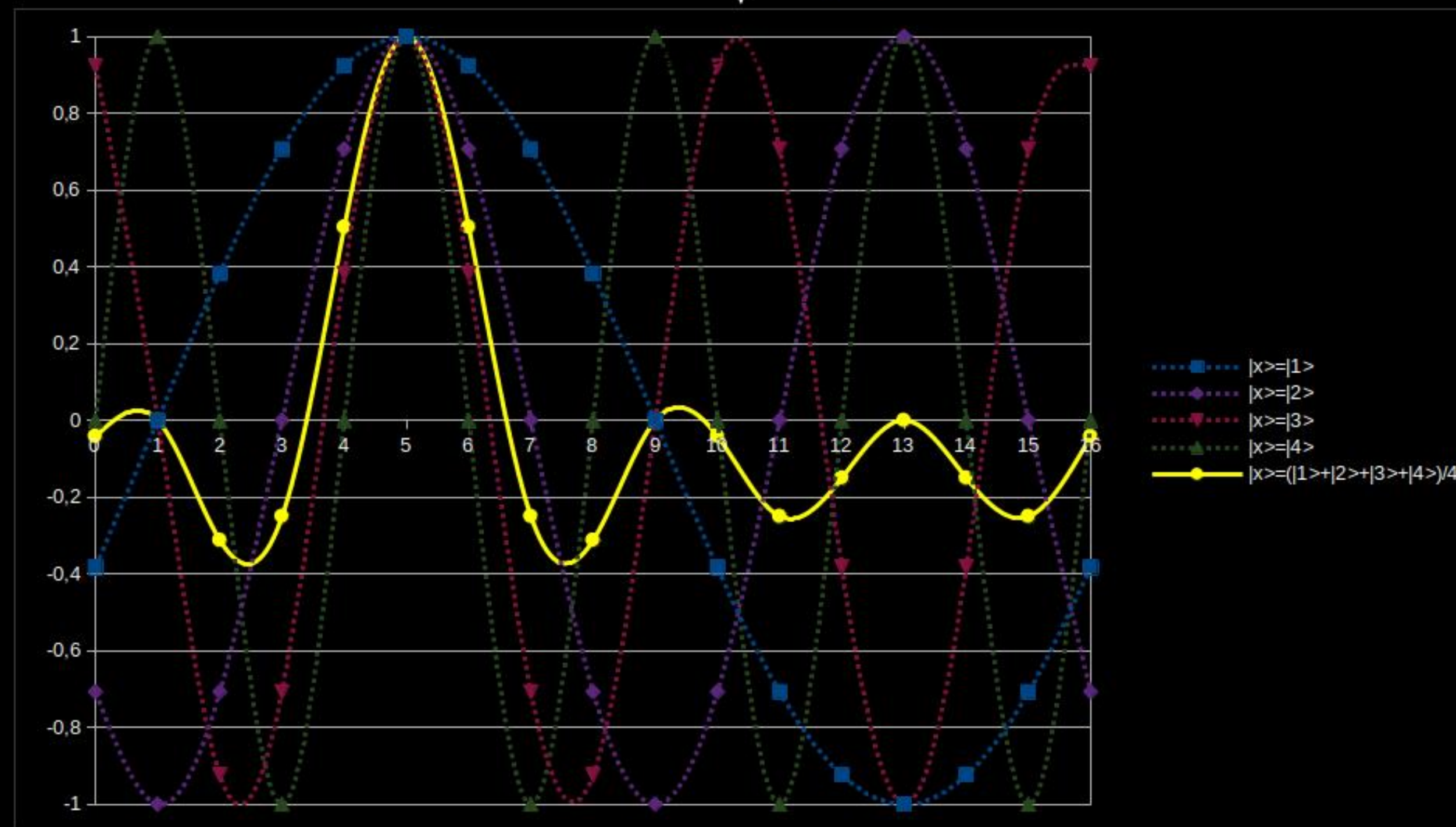
## Transformada Cuántica de Fourier (QFT)

$$|x\rangle + |x+p\rangle + |x+2p\rangle \dots + \dots |x+np\rangle \mapsto \{\text{QFT}\} \mapsto \left| \frac{1}{p} \right\rangle$$

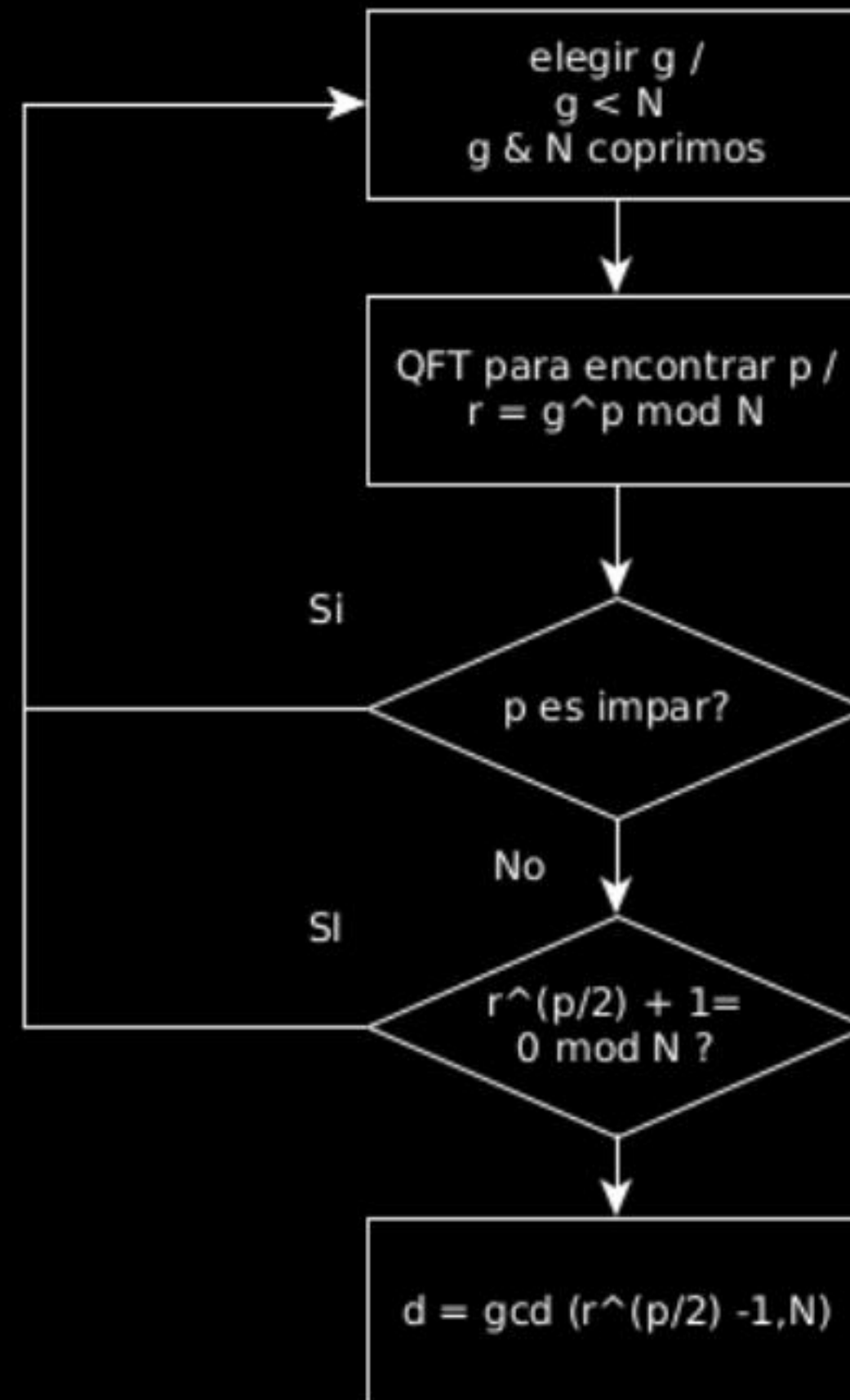


## Transformada Cuántica de Fourier (QFT)

$$\text{QFT}_N |x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{xk} |k\rangle$$



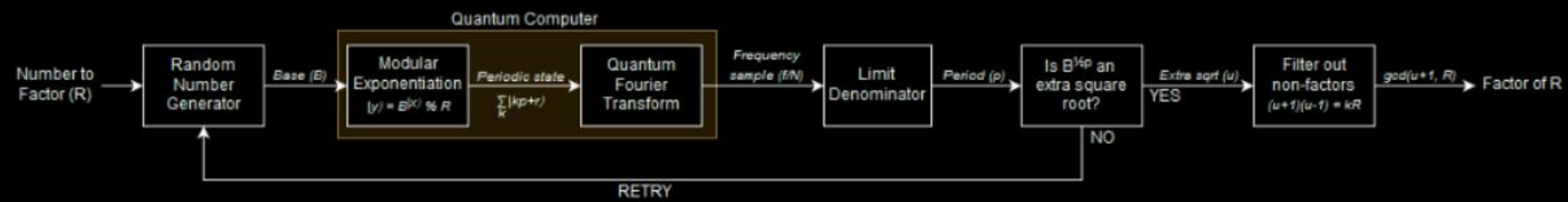
## El Algoritmo de Shor (IV)





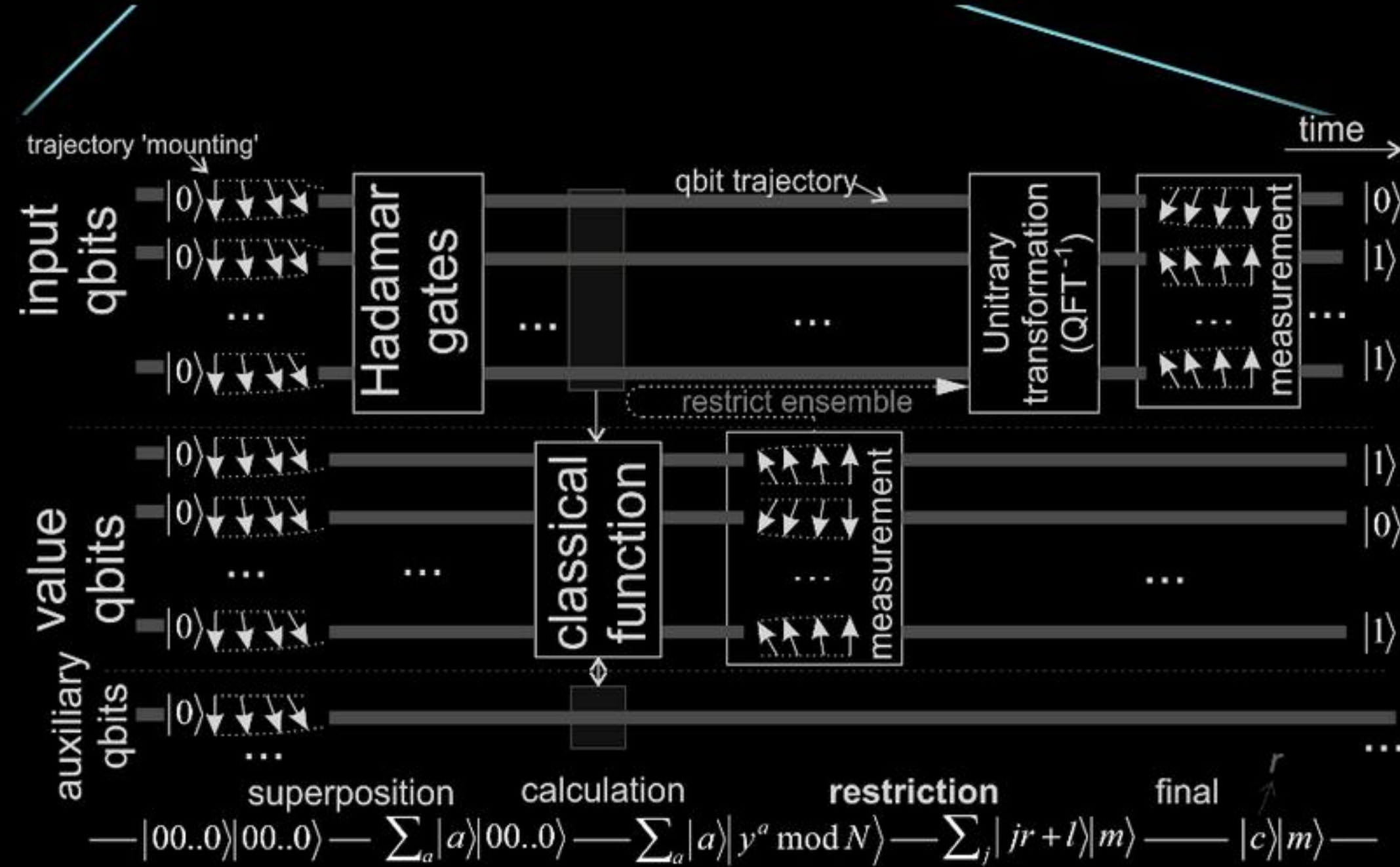
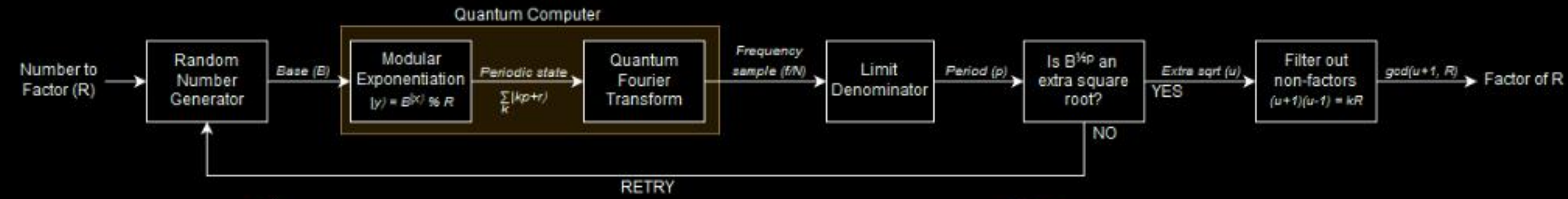
- previously...
- el problema
- RSA
- la factorización
- fourier
- el algoritmo de shor
- la implementación
- resumen

# El algoritmo de Shor





# El algoritmo de Shor



## Algoritmo de Shor: Implementación

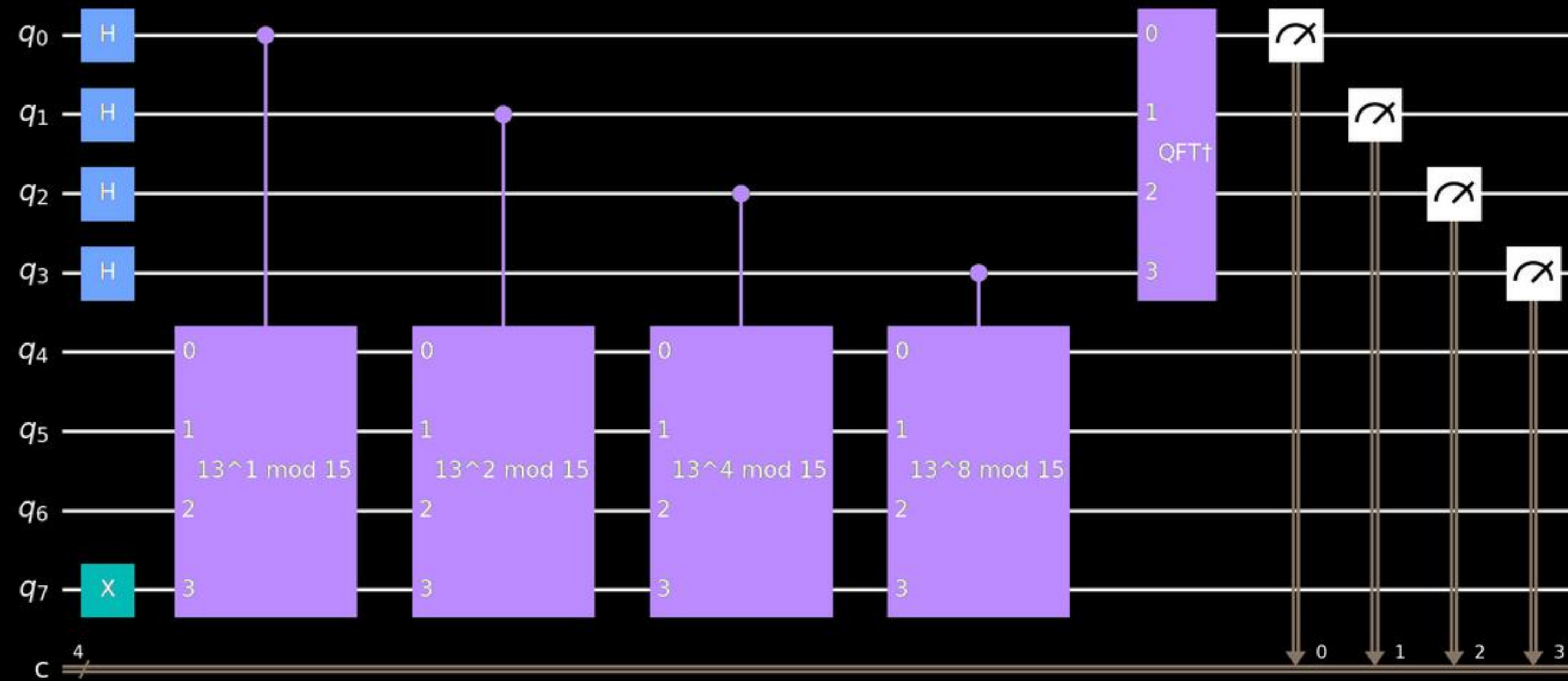
$$\text{QFT}_2 |x\rangle \mapsto \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

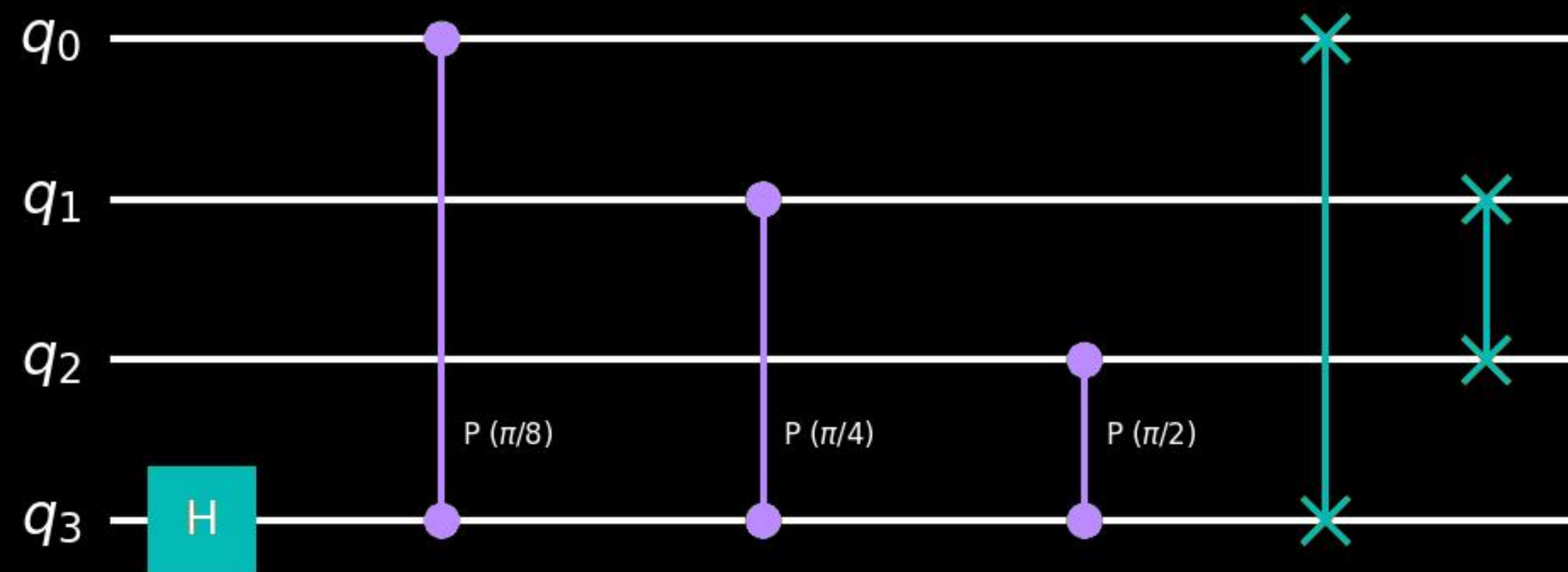
$$U \text{ ROT}_k = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 \\ 0 & e^{j \frac{2\pi}{2^k}} \end{bmatrix}$$



# El algoritmo de Shor circuito cuántico (4 qubits)

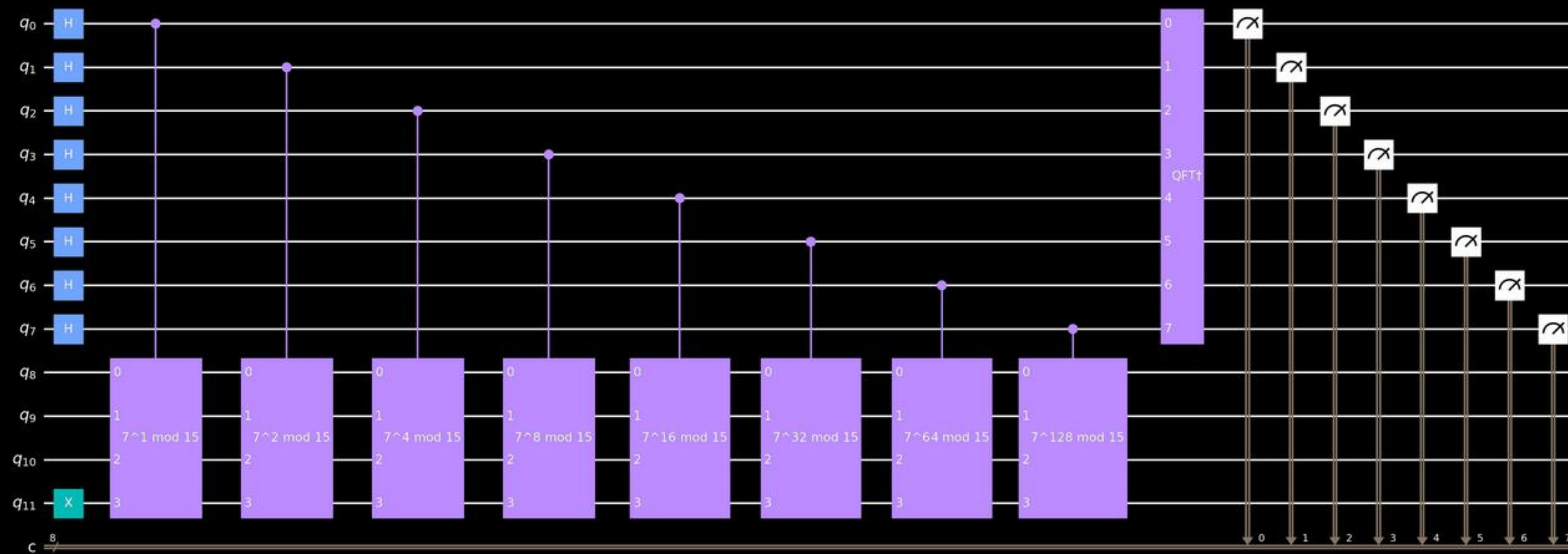


El algoritmo de Shor  
circuito cuántico de QFT (4 qubits)

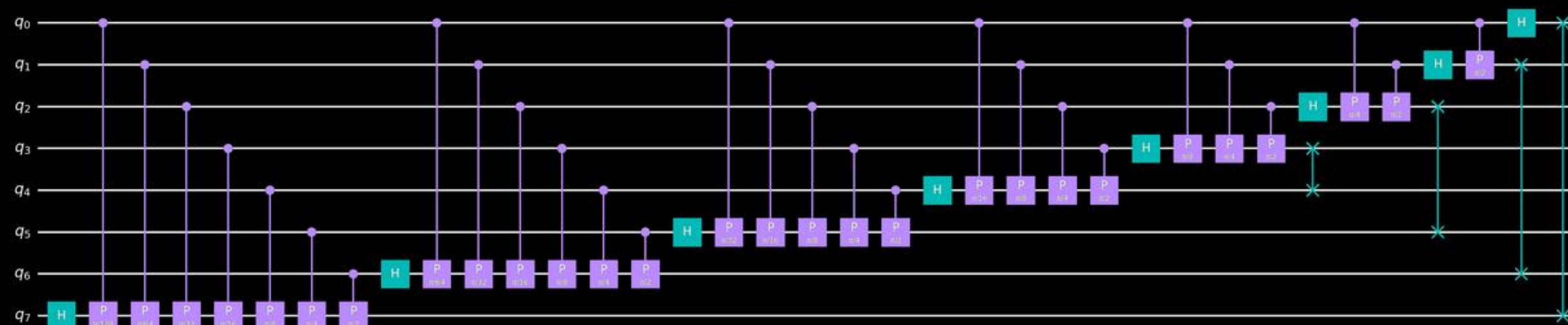




# El algoritmo de Shor circuito cuántico (8 qubits)



# El algoritmo de Shor circuito cuántico de QFT (8 qubits)



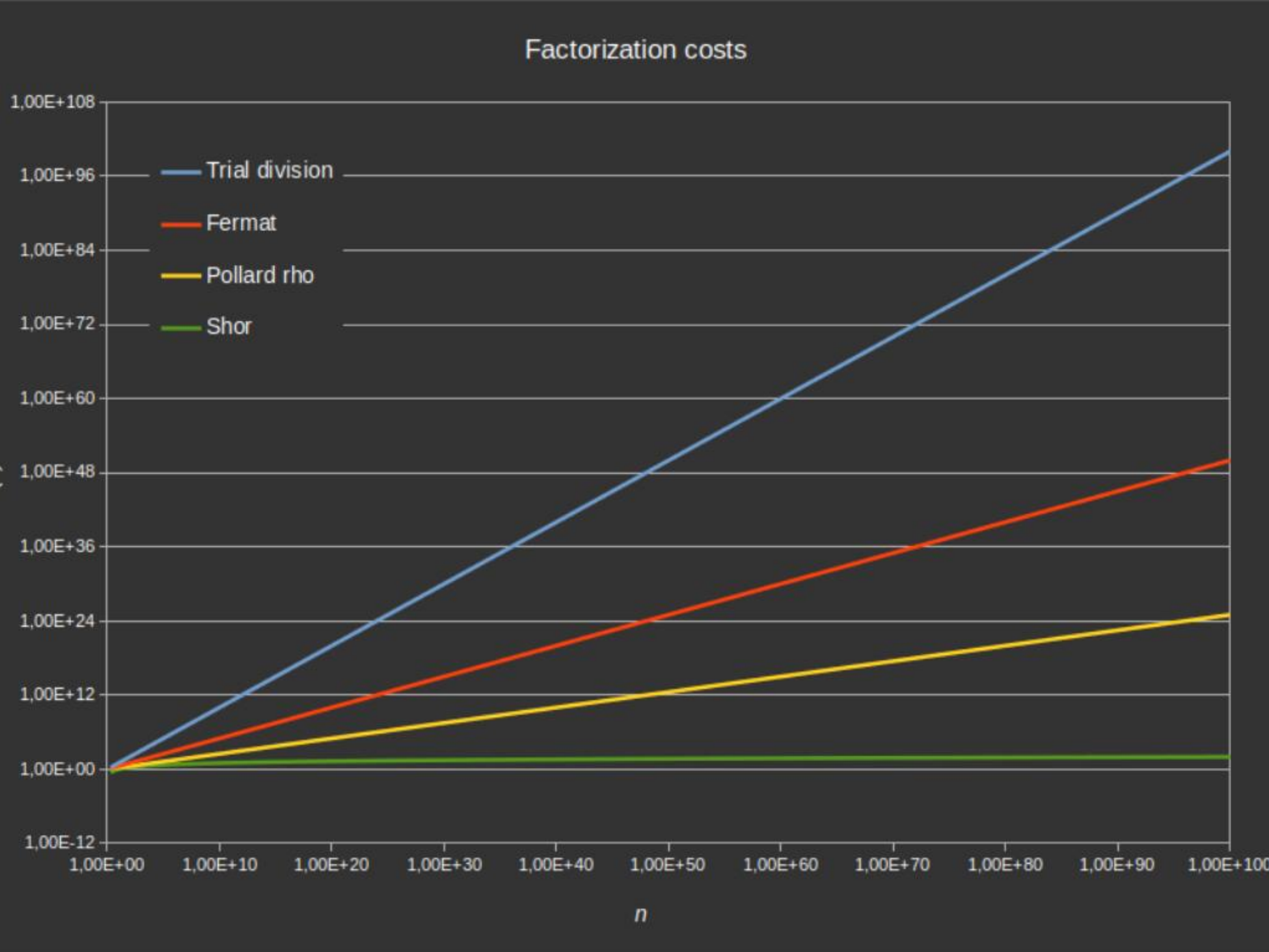


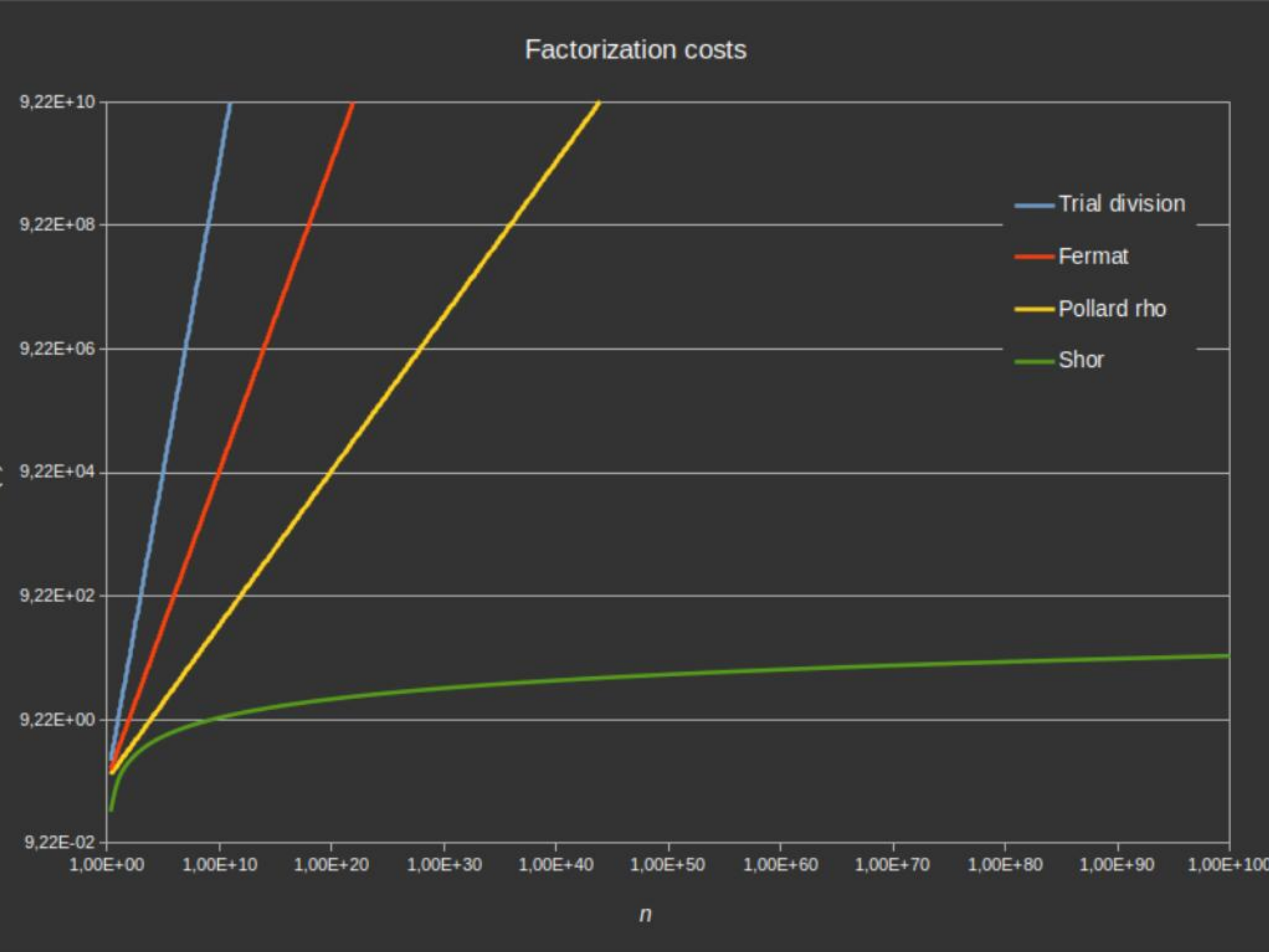
- previously...
- el problema
- RSA
- la factorización
- fourier
- el algoritmo de shor
- la implementación
- **resumen**



We are talking  
about speed...









de qué hablamos...?

- previously...
- factorización de grandes números...
- Fourier su serie y su transformada
- reescritura de la ecuación de factorización
- periodicidad en la función
- Shor 'crea' la QFT para encontrar el período
- del período sale 'p'... y de p 'q'
- implementación de Shor en una QC
- no se puede factorizar  $N > 56153 = 233 \cdot 241$
- ...por ahora...



# Referencias

How does RSA works

<https://hackernoon.com/how-does-rsa-work-f44918df914b>

Trial Division Algorithm

<https://www.geeksforgeeks.org/trial-division-algorithm-for-prime-factorization/amp/>

Fermat's Factorization Method

<https://www.geeksforgeeks.org/fermats-factorization-method/>

Pollard's Factorization Method

<https://www.geeksforgeeks.org/pollards-rho-algorithm-prime-factorization/amp/>

Fourier Transform in qiskit

<https://leftasexercise.com/2019/02/25/implementing-the-quantum-fourier-transform-with-qiskit/>

Quantum Factorization

<https://towardsdatascience.com/quantum-factorization-b3f44be9d738>

How Quantum Computers Break Encryption | Shor's Algorithm Explained

<https://www.youtube.com/watch?v=FRZQ-efABeQ>

How Shor's Algorithm Factors 314191

<https://www.youtube.com/watch?v=lvTqbM5Dq4Q>

New largest number factored on a quantum device is 56,153

<https://phys.org/news/2014-11-largest-factored-quantum-device.html>

Self

[https://github.com/ch4r1i3b/Eko2021\\_The\\_Silent\\_Partners](https://github.com/ch4r1i3b/Eko2021_The_Silent_Partners)



# ¡muchas gracias!

**Luciano Bello**  
lucian0<at>ieee.org  
@microluciano

**Carlos Benitez**  
carlos<at>platinumciber.com  
@ch4r1i3b

Ekoparty #17  
2 de noviembre de 2021

